Universität
Rostock
Traditio et Innovatio

System-Level Design of Energy-Efficient Sensor-Based Human Activity Recognition Systems

A Model-Based Approach

Dissertation
to obtain the academic degree
Doktor-Ingenieur (Dr.-Ing.)
of the Faculty of Computer Science and Electrical Engineering
at the University of Rostock

submitted by
Florian Grützmacher
born on 13.01.1990 in Crivitz, Germany
from Rostock

Rostock, March 29th, 2021

https://doi.org/10.18453/rosdok_id00003373

Dieses Werk ist lizenziert unter einer
Creative Commons Namensnennung 4.0 International Lizenz.

Reviewers:

- **Prof. Dr.-Ing. habil. Christian Haubelt**
  Embedded Systems Chair
  Institute of Applied Microelectronics and Computer Engineering
  University of Rostock, Germany

- **Prof. Dr.-Ing. Thomas Kirste**
  Mobile Multimedia Information Systems Chair
  Institute for Visual and Analytic Computing
  University of Rostock, Germany

- **Prof. Dr. Andy D. Pimentel**
  Parallel Computing Systems Chair
  Informatics Institute
  University of Amsterdam, Netherlands

Date of submission: 29.03.2021
Date of defense: 12.10.2021

Acknowledgments

First and foremost, I would like to express my sincere gratitude to Prof. Dr. Christian Haubelt for the confidence he has placed in me and for the superb supervision that I received from him during my time at the University of Rostock. Without him, this thesis would have not been possible. I would also like to thank Prof. Dr. Thomas Kirste and Prof. Dr. Andy Pimentel, who accepted to review this thesis.

Moreover, I would like to express my thanks to Dr. Rainer Dorsch and Bosch Sensortec GmbH for their support and provision of various hardware prototypes, Dr. Sander Stuijk and Dr. Bart D. Theelen for the many helpful discussions about the theory of data flow graphs and their analyses, Prof. Dr. Kristof Van Laerhoven and Jochen Kempfle for the cooperation on piecewise linear approximation algorithms, and all other project partners and collaborators during the last years.

Many thanks go to my colleagues from the Institute of Applied Microelectronics and Computer Engineering of the University of Rostock, for the really pleasant working atmosphere and the many scientific and non-scientific, but always interesting and helpful discussions. Especially, I would like to thank Benjamin Beichler, Johann-Peter Wolff, Kai Neubauer, and Dr. Michael Rethfeldt for their support and availability for many fruitful discussions.

Finally, I would like to thank my friends and especially my family for motivating and supporting me in many ways. Special thanks go to my beloved Pily for her support and her patience with me and my frequent mental absence while writing this thesis.

The support of all of you has been instrumental to the success of this work.

Abstract

Today, we are surrounded by a plethora of different processor systems in our everyday life and almost every situation. Mobile and wearable devices are equipped with embedded processors to provide advanced functionality and assistance in everyday situations. Furthermore, embedded systems can be found in our daily vicinity, be it at home, in offices, public places or in transport. Besides pervasive integration, embedded systems have seen a transformation from single processor systems to powerful multi-processor systems on chip, in order to overcome technological limitations in form of heat dissipation. Furthermore, the diversity of processor architectures has increased to heterogeneous multi-processor systems that offer trade-offs between energy demands, computational power, and flexibility for a wide range of application-specific demands. With technological advances, assistive systems have seen a further development by industry and research as well. As a fundamental requirement, in order to provide us with relevant services, our current *activity* needs to be known by the assistive system. Consequently, assistive systems must be able to infer our activities automatically by observations, without additional, possibly obtrusive interaction.

Inertial sensors have gained an important role for such observations, as advances in micro-electro-mechanical systems technology allow them to be manufactured as small and unobtrusive, but also energy-efficient and cheap components. Since then, sensor-based human activity recognition systems have increasingly been subject to research and development towards high recognition accuracy, but also many conceptual and algorithmic optimizations with respect to low latency and energy efficiency have been developed. The diversity and availability of computational resources as well as algorithmic solutions leads to promising possibilities for system designs such that software components can be executed on suitable processing units, in order to meet application-specific requirements on latency, data throughput, and energy consumption.

However, the huge number of possibilities comes at the price of a vast design space, that becomes increasingly hard to explore. In order to reduce design time, methods to quantify and thus substantiate design decisions early in the design process become crucial. The thesis at hand proposes model-based design and analysis techniques as a possible solution. To this end, dataflow models of computation are evaluated towards their ability to capture abstracted behavior of human activity recognition systems. These models can further capture design decisions with respect to mappings, schedules, functional parameters, and existing conceptual optimizations of human activity recognition system, and allow a formal analysis of extra-functional properties.

To this end, the thesis at hand contributes an evaluation of state-of-the-art dataflow models of computation regarding their suitability for a model-based design and analysis of human activity recognition systems, in terms of expressiveness and analyzability, as well as model accuracy. Different aspects of state-of-the-art human activity recognition systems have been modeled and analyzed. Based on existing methods, novel analysis approaches have been developed to acquire extra-functional properties like processor utilization and data communication rates, which directly influence energy consumption of the system. Furthermore, energy consumption models are introduced with which hardware elements can be annotated, in order to estimate the impact of design decisions on the energy consumption at design time.

Kurzfassung

In unserem heutigen Alltag sind wir von einer Vielzahl verschiedener Prozessorsysteme umgeben. Mobile und tragbare Geräte sind mit unterschiedlichsten Prozessoren ausgestattet, um erweiterte Funktionalitäten und Unterstützung bereitzustellen. Darüber hinaus finden wir eingebettete Systeme in unserer täglichen Umgebung, sei es zu Hause, in Büros, an öffentlichen Orten oder im Transportwesen. Neben allgegenwärtiger Integration haben sich eingebettete Systeme von Einzel-, zu leistungsstarken Multiprozessorsystemen entwickelt, um technologische Einschränkungen durch unzureichende Wärmeableitung zu überwinden. Darüber hinaus hat die Vielfalt an heterogenen Multiprozessorsystemen zugenommen, die Abwägungen zwischen Energiebedarf, Rechenleistung und Flexibilität für anwendungsspezifische Anforderungen ermöglicht. Mit dem technologischen Fortschritt wurden auch Assistenzsysteme durch Industrie und Forschung weiterentwickelt. Als Grundvoraussetzung um uns relevante Dienste bereitstellen zu können, muss unsere aktuelle *Aktivität* dem Assistenzsystem bekannt sein. Folglich müssen Assistenzsysteme in der Lage sein, unsere Aktivitäten eigenständig durch Beobachtungen abzuleiten.

Inertialsensorik hat für solche Beobachtungen eine wichtige Rolle eingenommen, da sie dank der Fortschritte in der mikroelektromechanischen Systemtechnologie als kleine und unauffällige, aber auch energieeffiziente und kostengünstige Komponenten hergestellt werden können. Seitdem wurden sensorgestützte Aktivitätserkennungssysteme zunehmend erforscht und entwickelt, um eine hohe Erkennungsgenauigkeit zu erreichen. Es wurden jedoch auch konzeptionelle und algorithmische Optimierungen hinsichtlich Latenz und Energieeffizienz entwickelt. Die Vielfalt und Verfügbarkeit von Rechenressourcen sowie Algorithmen bietet vielversprechende Möglichkeiten für den Systementwurf, sodass Softwarekomponenten auf geeigneten Prozessoreinheiten ausgeführt werden können, um anwendungsspezifische Anforderungen zu erfüllen.

Der Entwurfsraum wird mit zunehmender Größe jedoch schwieriger zu explorieren. Um die Entwurfszeit zu verkürzen, werden Methoden zur Quantifizierung und dadurch zur Untermauerung von Entwurfsentscheidungen früh im Entwurfsprozesses von entscheidender Bedeutung. Die vorliegende Arbeit schlägt modellbasierte Entwurfs- und Analysetechniken als mögliche Lösung vor. Zu diesem Zweck werden Datenflussberechnungsmodelle dahingehend bewertet, ob sie das abstrahierte Verhalten von Aktivitätserkennungssystemen erfassen können. Diese Modelle können Entwurfsentscheidungen in Bezug auf Bindung, Ablaufplanung, Funktionsparameter und vorhandene konzeptionelle Optimierungen von Aktivitätserkennungssystemen erfassen und eine extrafunktionale Eigenschaftsprüfung ermöglichen.

Zu diesem Zweck liefert die vorliegende Arbeit eine Bewertung bestehender Datenflussberechnungsmodelle hinsichtlich ihrer Eignung für den modellbasierten Entwurf und Analyse von Aktivitätserkennungssystemen basierend auf ihrer Ausdrucksmächtigkeit und Analysierbarkeit sowie Modellgenauigkeit. Verschiedene Aspekte modernster Aktivitätserkennungssysteme wurden dazu modelliert und analysiert. Basierend auf vorhandenen Methoden wurden neuartige Analyseansätze entwickelt, um extrafunktionale Eigenschaften wie Prozessorauslastung und Datenkommunikationsraten zu berechnen, die den Energieverbrauch des Systems direkt beeinflussen. Darüber hinaus werden Energieverbrauchsmodelle eingeführt, mit denen Hardware-Elemente annotiert werden können, um die Auswirkungen von Entwurfsentscheidungen auf den Energieverbrauch zur Entwurfszeit abzuschätzen.

Contents

List of Abbreviations **IX**

1 Introduction **1**
1.1 Problem Statement and Approach 4
1.2 Contributions 6
1.3 Author's Publications 7

2 Fundamentals **9**
2.1 Dataflow Models of Computation 9
2.1.1 Synchronous Dataflow Graphs 9
2.1.2 Cyclo-Static Dataflow Graphs 14
2.1.3 Scenario-Aware Dataflow Graphs 17
2.2 System Modeling 22

3 Energy-Efficient Sensor Networks **25**
3.1 Related Work 26
3.2 Case Study 29
3.3 Modeling 31
3.3.1 Model Refinement 32
3.3.2 Application Model 35
3.3.3 Analysis Model 36
3.4 Analysis 40
3.4.1 Real-Time Behavior 40
3.4.2 Energy Trade-Off 43
3.5 Experiments 46
3.5.1 Configurations 46
3.5.2 Implementation 48
3.5.3 Model Evaluation 50
3.5.4 Experimental Setup 51
3.5.5 Experimental Results 51
3.6 Discussion 54

4 Thread-Level Parallelism **55**
4.1 Related Work 56
4.2 Dynamic Time Warping 58
4.3 Case Study 59
4.4 Parallelization Approaches 60
4.4.1 Intra-Segment Parallelization 61
4.4.2 Inter-Segment Parallelization 63
4.4.3 Hybrid Parallelization 66
4.4.4 Parallelization Configurations 68
4.5 Experiments 69
4.5.1 Hand Gesture Recognition Application 69

4.5.2 Hardware Architecture . . . 73
4.5.3 Analysis Model . . . 73
4.5.4 Model Refinement . . . 77
4.6 Evaluation . . . 84
4.6.1 Performance assessment . . . 84
4.6.2 Evaluation Results . . . 85
4.7 Discussion . . . 91

5 Context Awareness and Dynamic Behavior 93
5.1 Scenario-Based Dynamic Behavior . . . 94
5.1.1 Related Work . . . 95
5.1.2 Modeling Scenario-Based Dynamic Behavior . . . 96
5.1.3 Analysis Models . . . 101
5.1.4 Model Analysis . . . 103
5.1.5 Experiments . . . 105
5.1.6 Evaluation . . . 105
5.1.7 Discussion . . . 114
5.2 Data-Dependent Dynamic Behavior . . . 117
5.2.1 Related Work . . . 118
5.2.2 Simple Linear Regression . . . 120
5.2.3 Online Piecewise Linear Approximation . . . 122
5.2.4 Evaluation . . . 130
5.2.5 Discussion . . . 139

6 Conclusions 141
6.1 Summary . . . 141
6.2 Discussion . . . 141
6.3 Limitations . . . 143
6.4 Future Work . . . 144

Bibliography 145

List of Publications 157

A Implementation Details on the Hand Gesture Recognition System 159

B Supplementary Experimental Data 163

List of Abbreviations

ANN	Artificial Neural Network
ANOVA	Analysis Of Variance
AP	Application Processor
ARC	Activity Recognition Chain
ASIC	Application-Specific Integrated Circuit
ASIP	Application-Specific Instruction-Set Processor
BDF	Boolean Dataflow
BLE	Bluetooth Low Energy
CCW	Counter Clockwise
CFG	Control Flow Graph
CPLR	Connected Piecewise Linear Regression
CSDF	Cyclo-Static Dataflow
CV	Coefficient of Variation
CW	Clockwise
DDF	Dynamic Dataflow
DIT	Decimation In Time
DMA	Direct Memory Access
DSP	Digital Signal Processor
DT	Decision Tree
DTW	Dynamic Time Warping
DVFS	Dynamic Voltage and Frequency Scaling
ECG	Electrocardiogram
EMA	Exponential Moving Average
FFT	Fast Fourier Transform
FIFO	First-In-First-Out
FPU	Floating-Point Unit
FSM	Finite State Machine
FSM-SADF	Finite State Machine Scenario-Aware Dataflow
GCC	GNU Compiler Collection
GPP	General-Purpose Processor
GPU	Graphics Processing Unit
HAR	Human Activity Recognition
HMM	Hidden Markov Model
HSDF	Homogeneous Synchronous Dataflow
I^{2}C	Inter-Integrated Circuit
ICR	Inverse Compression Ratio
IFL	Inter-Firing Latency

IMU	Inertial Measurement Unit
k-NN	k-Nearest Neighbors
LCG	Linear Constraint Graph
LDA	Linear Discriminant Analysis
MAC	Multiply and Accumulate
MCM	Maximum Cycle Mean
MCR	Maximum Cycle Ratio
MEMS	Micro-Electro-Mechanical System
MoC	Model of Computation
MPSoC	Multi-Processor System-on-Chip
MSM	Multicore Shared Memory
MSMC	Multicore Shared Memory Controller
NoC	Network-on-Chip
OLS	Ordinary Least Squares
PCA	Principle Component Analysis
PFSM-SADF	Parametrized Finite State Machine Scenario-Aware Dataflow
PLA	Piecewise Linear Approximation
SADF	Scenario-Aware Dataflow
SCC	Strongly Connected Component
SDF	Synchronous Dataflow
SF	Swing Filter
SIMD	Single Instruction, Multiple Data
SiP	Systems-in-Package
SRAM	Static Random-Access Memory
SSR	Sum of Squared Residuals
SVM	Support Vector Machines
SW	Sliding Window
SWAB	Sliding Window and Bottom Up
TI	Texas Instruments
ToF	Time of Flight
VLSI	Very Large Scale Integration
VPDF	Variable-Rate Phased Dataflow
WCET	Worst-Case Execution Time
WSN	Wireless Sensor Network

1 Introduction

With the advances in microelectronics and computer architectures, *Human Activity Recognition* (HAR) as a research field has increasingly gained attention in the past 25 years. The demand of automatic detection and classification of human behavior by computer systems has been increasing in various social, medical, safety, and security areas. While approaches based on computer vision have been dominating the first attempts, the technological advances in inertial sensor systems have shifted the focus to sensor-based human activity recognition systems. Inertial sensors are composed of accelerometers and gyroscopes that, when attached to an object, allow the measurement of acceleration and angular speed of that object into the direction and around the rotation axis of the sensor, respectively. By integrating three of these sensors perpendicular to each other, the 3D movement and rotation can be captured. Often inertial sensors are combined with magnetometers, in order to measure the magnetic field in each axis, which allows to transform the measured signals into the earth coordinate system and further calculate object orientation from it. Especially the introduction of Micro-Electro-Mechanical System (MEMS) technology into the development, has lead to inertial sensor systems, that are small and unobtrusive enough to be worn on the human body. This can be in form of dedicated sensing devices or integrated in mobile and wearable devices like smartphones, smart watches, fitness trackers, glasses, or ear plugs. Figure 1.1 depicts the package configuration of an BMI260 IMU sensor from Bosch Sensortec GmbH, the dimensions of which are $3.0x2.5x0.83mm^3$ [1].

Application-Specific Instruction-Set Processors (ASIPs) can be integrated with MEMS sensors of different modalities as so-called Systems-in-Package (SiP). These ASIPs provide the interface to the host system that the sensor sub-system is embedded into, provide sample timestamps, execute signal correction algorithms for offset or drift compensation, and perform signal fusion to derive further information, e.g., sensor orientation. Modern sensor sub-systems also perform basic gesture or activity recognition, which is executed on their ASIP [2]. The signals acquired from IMU sensors are mostly equidistantly sampled timeseries of multiple dimensions whose sampling frequencies typically span from a few hertz up to one or two kilohertz, which however varies depending on the application domain. In human gesture and activity recognition systems, typical sampling frequencies range from $25-200Hz$. In Figure 1.2, the signal from an accelerometer attached to the foot of a walking person is depicted.

In the thesis at hand, sensor-based human activity recognition refers to the classification of human gestures or activities from signals that are derived from IMU sensors, which are attached to the human body, e.g., in form of wearable devices.

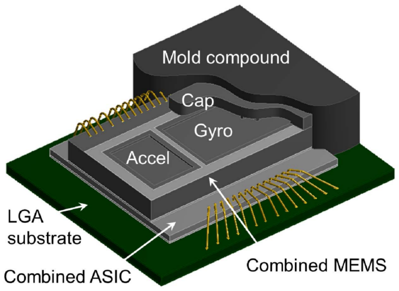


Figure 1.1: BMI260 IMU sensor from Bosch Sensortec GmbH. From [1].

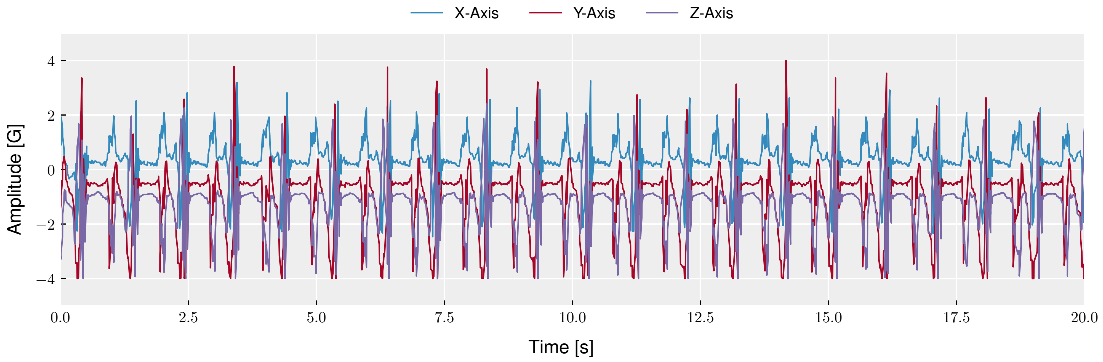


Figure 1.2: Sensor signal acquired from an accelerometer attached to the foot of a walking person.

The general principle of sensor-based human activity recognition was summarized by Bulling et al. in [3], in terms of an Activity Recognition Chain (ARC). An ARC consists of several processing stages from raw sensor signals to the final classification results, that can be typically found in state-of-the-art HAR systems in the literature. Generally, the stages involve *data acquisition* from inertial sensors in the first place, and a following *pre-processing*, e.g., the application of smoothing, low-pass, or high-pass filters, or general signal enhancement approaches. The pre-processed, potentially infinitely long signals are then segmented into finite lengths, e.g., by a sliding window, in the *segmentation* stage. For each segment, descriptive features are calculated in the *feature extraction* stage. The resulting feature vectors of each segment are then classified by an appropriate classification algorithm in the *classification* stage. However, due to the possibly high number of dimensions of the feature vector, which can cause a poor discriminative performance of the classification algorithm, a *dimensionality reduction* stage is often implemented after feature extraction, in order to transform the high dimensional feature vector into a lower dimensionality in which decisive characteristics are still retained. Principle Component Analysis (PCA) or Linear Discriminant Analysis (LDA) are commonly applied methods for dimensionality reduction.

Although many HAR systems follow this general pattern, the variety of methods that are utilized for each stage is huge. In the data acquisition stage, sensor data can be sampled from accelerometers, gyroscopes, magnetometers, or a combination of these, which have different energy consumption demands. Pre-processing can range from no pre-processing, to simple moving average filters to computationally intensive sensor fusion algorithms, e.g., Kalman filters. Segmentation methods are typically sliding window approaches. However, window sizes and overlaps have a huge impact on the run time of the feature extraction stage, and on the frequency with which subsequent stages are executed. Other approaches involve segmentation algorithms which lead to dynamically sized windows. Feature extraction can be composed of statistical moments like mean and variance, which can be efficiently computed, but also computationally intensive frequency domain features which involve a Fast Fourier Transform (FFT) to be computed. Therefore, size and type of features have a huge impact on the computational effort, energy consumption, and computation time. Dimensionality reduction is typically performed by PCA or LDA, which have a typically small computational overhead, but again depend on the dimensionality of the input and the output of the stage. Finally, for classification, a huge amount of algorithms exist, ranging from computationally lightweight classifiers like, decision trees, to more computationally intensive algorithms like Support Vector Machines (SVM). Furthermore, computation time and memory demands of classification algorithms are usually a trade-off between training time and run time. For example, the training time of k-Nearest Neighbors (k-NN), is usually zero[1.1], as the complete set of training features is provided at run time, to find the *k* nearest feature vectors in the feature space and classify the sample under test by a majority voting of the *k* nearest feature vector classes. Execution time for classification and memory consumption therefore depend on the size of the training dataset. On the contrary, other classification algorithms, e.g., SVMs, might need an increased execution time for training to adjust discrimination functions adequately, but a shorter execution time for classification at run time. This variety of parameters and algorithms in each stage leads, therefore, to disparate HAR systems w.r.t. computational effort, memory consumption, and latency, finally depending on the anticipated application, activities to be recognized, and the available sensor sources and their quality. Note that Artificial Neural Networks (ANN) are considered as possible classifiers in the classification stage of the ARC. Yet, due to the focus on systems that follow the ARC structure, approaches that are processing all the steps contained in an ARC in a single ANN with many layers, i.e., deep learning approaches, are not considered in this work in general. However, this limitation could be relaxed for ANNs, which can be structured as several stages that could be processed in a pipelined or distributed manner, e.g., based on a certain degree of sparsity in the weights. This approach however, requires further research and is not covered in this thesis.

Despite the variety of possible ARC setups, many HAR systems share a similar set of requirements, which are first and foremost functional properties regarding recognition accuracy. However, when a suitable ARC setup for the anticipated application has been developed and tested regarding its recognition accuracy, the transformation into a final system must satisfy requirements on extra-functional properties. In the thesis at hand, the extra-functional properties of interest are real-time performance, latency, and energy consumption. Real-time performance denotes the ability of the HAR system to process the stream of continuously sampled sensor data during run time of the system, without the loss or congestion of sensor data at any ARC stage. As a result, the system throughput is a direct indicator of its real-time performance, which has to be higher than the highest input data rate that can be expected during run time of the system. Latency is defined as the time from sampling a particular sensor sample, until the classification result of the corresponding segment containing that sample is computed by the HAR system. As multiple samples are usually contained in a segment, and segment size can vary, the worst-case latency is decisive. Finally energy consumption of the system, or of components of the underlying hardware architecture is of importance as, e.g., for wearable devices, it directly influences the time until the batteries are drained and thus impacts the usability of the system.

Typically, HAR systems premise all of the aforementioned requirements, but however, these strongly depend on the anticipated application. As an example, the latency requirements of an assistive system for kitchen tasks may be rather relaxed in the range of seconds. An industrial worker assistive system in contrast, may have a requirement of a few milliseconds or even microseconds on latency for the detection of an emergency situation, in order to stop machinery like, e.g., industrial robot arms, to prevent any harm to the worker. Likewise, requirements on throughput and energy consumption are depending on the activities to be recognized, anticipated sensor sampling rates that are required for the recognition task at hand, and on deployed hardware and batteries.

Apart from the aforementioned, there are many more requirements on HAR systems, e.g., memory utilization, robustness, or security, which however, are out of scope in the thesis at hand.

[1.1] Here, a standard k-NN approach is assumed, without optimizations as, e.g., space partitioning data structures like K-D-Trees [4], the construction of which would include additional pre-processing.

1.1 Problem Statement and Approach

Although many technological, algorithmic, and conceptual approaches exist to accelerate sensor-based human activity recognition systems and to reduce energy consumption of utilized hardware components, the impact of proposed optimizations highly depends on the particular software and hardware that is deployed. Generalized statements about their improvement factor are thus not possible and have to be evaluated per application. Furthermore, the software complexity of human activity recognition systems is steadily increasing by means of individual processing stages from the acquisition of sensor data until the final inference of the currently performed gesture or activity. Likewise, the hardware architecture that is used for human activity recognition systems is getting increasingly complex as well. Sensor subsystems with integrated on-chip microcontrollers for sensor fusion and pre-processing, are combined with low-power wireless controllers on wireless sensor nodes. Possible processor architectures for data aggregation from multiple wireless sensors, i.e., of smartphones, tablets, smart watches, or dedicated on-body devices, are composed of heterogeneous processing units ranging from of ultra low power microcontrollers for data aggregation, heterogeneous general purpose processors (GPPs), to powerful digital signal processors (DSPs), graphics processing units (GPUs), and ASIPs. Further processing units are located in the vicinity of protagonists like home servers, processor architectures in cars, or assistive systems in public buildings or offices. Due to their static location and constant power supply, these can be deployed with powerful Multi-Processor Systems-on-Chip (MPSoCs). The communication between hardware components is composed of equally diverse standards, that are low-power wireless standards like Bluetooth Low Energy (BLE), Narrowband-IoT, or LoRa for unobtrusive and small devices with respectively small batteries, to high throughput standards like WiFi and 5G connecting devices that are less battery-constrained like static servers or wearable devices with increased battery technology and capacities. The resulting possibilities of a suitable hardware platform for human activity recognition systems are huge and constitute a likewise complexity that has to be managed by the system designer. However, the distribution of software to hardware has a huge impact on important extra-functional properties like latency, throughput, and energy consumption. Substantiation of design decisions based on prototype implementations and measurements of said extra-functional properties is unfeasible considering the huge design space spanned by the number of possible software to hardware mappings. In order to approach this problem, design-time analysis methods are crucial to reduce time to market and still provide system designs that meet necessary requirements.

A possible solution are model-based design and analysis approaches. In a model-based design flow, software and hardware are specified as formal models, with which hardware selection, software mapping, and scheduling decisions can be formally specified as a design point. Furthermore, design points can be formally evaluated with respective analysis and optimization methods that have been developed for the modeling formalism that is used. Moreover, analysis results of different design points can be compared, in order to substantiate design decisions at design time.

The thesis at hand proposes a model-based design and analysis of energy-efficient sensor-based online human activity recognition systems to address the aforementioned open problems. The model-based approach is based on Models of Computation (MoC) that have become a quasi-standard in the design of signal processing and streaming applications, i.e., dataflow graph models. To this end, the thesis at hand deals with the following research questions:

RQ1 **Which dataflow MoC is suitable to capture state-of-the-art human activity recognition systems?**

There exists a variety of dataflow MoCs that differ in analyzability and expressiveness. As a general trend, the analyzability of available dataflow MoCs decreases with expressiveness. To this end, suitable MoCs have to be identified, that are expressive enough to capture state-of-the-art human activity recognition systems, but still allow for the analysis of important extra-functional properties like latency, throughput, and energy consumption at design time.

RQ2 **How can important extra-functional properties be accurately analyzed from dataflow graph models?**

While there exist a plethora of formal analysis techniques for dataflow graphs regarding timing behavior, comparably little research has been undertaken towards the estimation of energy consumption from dataflow graph models in the literature. Furthermore, the accuracy of energy analysis approaches is of interest.

RQ3 **Can existing conceptual optimizations regarding latency, throughput, and energy consumption of sensor-based human activity recognition systems be represented in dataflow graph models?**

As indicated earlier, many optimization techniques exist in the literature, that reduce energy consumption and improve timing behavior of human activity recognition systems. However, the impact thereof highly depends on the application structure and hardware selection. Since generalized assumptions on the improvement factor are not possible, a model-based approach should be able to represent such optimizations and their impact should be quantifiable at design time.

RQ4 **Can dynamic behavior of human activity recognition systems be captured by dataflow graph models and corresponding analysis methods?**

Many human activity recognition systems are subject to changes at run time, which are based on data-dependent, context-aware, or hardware-related changes in order to improve recognition accuracy, decrease latency, or conserve energy of deployed battery-powered devices. Dynamic behavior is a challenging aspect in the model-based representation of applications, as it often requires a certain degree of expressiveness that contrasts its analyzability. While there exist dataflow MoCs that allow to model a certain degree of dynamic behavior, the proposed model-based analysis techniques w.r.t. energy consumption indicators should be extendable to such MoCs.

In the following section, the contributions of the thesis at hand towards the aforementioned research questions are summarized.

1.2 Contributions

Model-based design and analysis approach of sensor-based human activity recognition systems (Chapter 3) The key contribution of the thesis at hand, is a novel design approach for sensor-based human activity recognition systems based on well-known models of computation, i.e., dataflow graphs. General system structures of HAR systems in the literature have been identified and translated into a model-based representation, allowing further analysis methods to be applied. Based on existing analysis methods, novel approaches have been developed to analyze key properties w.r.t. energy efficiency from dataflow models at design time, that are processor utilization and data communication rate on hardware components. Due to the integration of the input data process into models, introduced analysis methods are enabled to evaluate worst-case behavior at design time.

Energy consumption estimation of hardware components (Chapter 3) Based on the aforementioned analysis methods regarding processor utilization and data transmission rates, energy consumption models have been introduced that capture the relationship to said indicators, allowing to quantify effects of decisions at design time. Introduced energy consumption models differ from existing approaches in the sense, that a long-run average estimate on energy consumption rate can be acquired, in contrast to an absolute energy consumption w.r.t. an execution of a particular sequence of instructions. The latter neglects the rate which the sequence of instructions is executed with. Energy consumption models are estimated from measurements in the thesis at hand which, however, is orthogonal to existing approaches, i.e., can be acquired formally, as proposed in existing literature. Finally, energy consumption models are separated from the dataflow representation of the software and annotated to components of the hardware model, allowing to abstract and encapsulate technological aspects into separate models.

System-level parallelization on multi-processor architectures (Chapter 4) Introduced model-based design and analysis approaches are evaluated w.r.t. their ability to represent state-of-the-art parallelization approaches of computationally intensive processing stages within human activity recognition systems on multi-processor architectures. The representation of system-level parallelization into system models allows to evaluate corresponding effects of system throughput, latency, and the distribution of processor utilization at design time. Furthermore, their trade-offs can be considered to substantiate design decisions and possibilities for further reduction of energy consumption, i.e., clock-gating or dynamic voltage and frequency scaling, can be assessed.

Scenario-based dynamic behavior (Section 5.1) The proposed modeling and analysis approaches have been extended to existing models of computation, that allow the representation of a certain degree of dynamic behavior. That is, dynamic changes between a finite set of static scenarios. This allows, to represent human activity recognition systems, that change their system configuration at run time based on the environment, i.e., context awareness, system properties like battery-level-dependent dynamic sensor or feature selection of wireless sensor nodes, or data-dependent data reduction algorithms like piecewise linear approximation of sensor signals, that confine to a small finite set of static behaviors. Aforementioned approaches have been introduced in the literature to increase recognition accuracy and reduce energy consumption of human activity recognition systems. In order to substantiate design decisions regarding their deployment for the anticipated application, their impact must be quantifiable at design time. To this end, the model-based design approach in the thesis at hand is extended to models of computation that are expressive enough to allow the representation of dynamic changes between static scenarios, but are still analyzable w.r.t. key properties. Proposed analysis methods w.r.t, processor utilization have been extended for the selected model of computation.

Piecewise linear approximation of sensor signals (Section 5.2) The proposed model-based design and analysis approaches in the thesis at hand, foster awareness of predictability and analyzability of algorithms developed for energy-efficient online human activity recognition systems. A key aspect in this regard is the data-independent timing behavior of algorithms. Apart from model-based design and analysis, the thesis at hand contributes to the algorithmic optimization of existing data- and thus energy consumption reducing techniques based on piecewise linear approximation of sensor signals. Two novel piecewise linear approximation algorithms are proposed in the thesis at hand, that advance existing methods, towards a small and constant per-sample execution time and memory complexity w.r.t. their compression ability. As a result, their execution time per sensor sample and their memory utilization is small enough to be implemented on resource constrained sensor sub-systems without compromising any functional properties regarding their approximation quality. More importantly, their execution time is predictable at design time, allowing them to be modeled and analyzed with the approaches proposed in the thesis at hand.

1.3 Author’s Publications

Most of the work presented in this thesis has been previously published by the author. The thesis at hand refines and unifies published concepts and results and provides a consistent presentation of such. In the following, a summary is presented, that relates the key aspects of each chapter to the corresponding publications of the author in which they were initially introduced. Furthermore, the author’s contributions to the corresponding publications are highlighted.

Energy-Efficient Sensor Networks (Chapter 3) In Chapter 3, the modeling approach for wireless on-body sensor networks in the context of human activity recognition systems is presented. The motivation of trading-off energy consumption of different hardware components based on their workload stems from initial studies on the topic which was presented in [G1]. The author of this thesis has developed the main concept, performed implementation, and conducted the experiments in [G1]. Furthermore, the analysis of processor utilization as well as data communication rate from dataflow graph models of wireless sensor networks in human activity recognition systems was introduced in [G2] and [G3] and the concept of corresponding energy consumption models was published in [G4]. The author of the thesis at hand developed the presented concepts and implemented, performed, and evaluated corresponding experiments.

Thread-Level Parallelism (Chapter 4) In Chapter 4, the dataflow-based modeling of parallelization strategies for online gesture and activity recognition systems is presented and evaluated on experimental implementations in different configurations. Model-based analysis results are evaluated with measurements from the experiments. The experimental implementation and evaluation of the presented parallelization approaches have been published in [G5] and [G6]. In [G6], a first dataflow graph model has been introduced to capture respective parallelization approaches, which has been later refined for the thesis at hand, based on the modeling approach in [G2]. The author of the thesis at hand, developed the respective concepts and implemented, performed, and evaluated corresponding experiments in the aforementioned publications.

Scenario-Based Dynamic Behavior (Section 5.1) In Section 5.1, the modeling and analysis approaches are extended to scenario-aware dataflow graphs to support the representation of dynamic changes between static scenarios in the system models. Analysis methods w.r.t. processor utilization have been adapted to the selected MoC. The basic concept of using scenario-aware dataflow graphs as a MoC and a corresponding analysis technique towards processor utilization, was initially published in [G7]. The modeling approach and especially the analysis of processor utilization is thoroughly refined in the thesis at hand, including its limitations to the application domain. The author of this thesis developed the concept and implemented, performed, and evaluated the experiments in [G7]. The analysis of scenario occurrence probabilities from scenario-aware dataflow graphs was conceived and implemented by the co-author Bart D. Theelen.

Data-Dependent Dynamic Behavior (Section 5.2) In Section 5.2, two novel piecewise linear approximation algorithms for sensor signals are introduced, i.e., CPLR and fastSW. The concept, implementation, and evaluation of CPLR was previously published in [G8]. The concept and implementation of fastSW has not been published at the time of submission of the thesis at hand. The presented evaluation of fastSW in Section 5.2 is based on and integrated into the experimental evaluation of CPLR presented in [G8]. The author of this thesis developed the concept and implemented, conducted, and evaluated the experiments in [G8] and conceived and implemented fastSW.

2 Fundamentals

In the thesis at hand, model-based system-level design and analysis approaches for sensor-based human activity recognition systems are introduced, with which extra-functional properties can be acquired at design time in order to evaluate different system configurations early in the design process. The introduced approaches are based on formal models, i.e., dataflow-based MoCs, and their respective analysis methods. The fundamentals regarding dataflow MoCs on which this thesis builds on are discussed in Section 2.1. In Section 2.2, the system modeling approach that is used throughout the thesis is introduced.

2.1 Dataflow Models of Computation

In the literature exist several dataflow MoCs with a considerable range in expressiveness, analyzability, and implementation efficiency [5]. As a general trend, analyzability is inversely proportional to the expressiveness. However, analyzability is generally desirable, e.g., schedules that can be derived at design time, can be implemented statically or quasi-statically and therefore avoid overheads at run time. In contrast, expressiveness is important as well, as with more expressive models, the behavior of the system under design can be captured more accurately. As a result, the selection of appropriate MoCs is a trade-off between analyzability and expressiveness

The MoCs that have been used for modeling HAR systems in the thesis at hand are introduced in the following together with a short reasoning about the respective choices. For a review and classification of a broader range of dataflow MoCs, the interested reader is referred to [5].

2.1.1 Synchronous Dataflow Graphs

Synchronous Dataflow (SDF) graphs are weighted directed graphs and have been introduced by Lee and Messerschmitt in [6]. A notion of timing is associated with SDF graphs, in order to allow analysis of schedules or throughput at design time [7]. In the following, timed SDF graphs are assumed when referred to SDF graphs. Let $\mathbb{N} = \{1,2,3,\dots\}$ denote the set of natural numbers. Furthermore, the set of non-negative integers is denoted by $\mathbb{Z}_{0+} = \{0,1,2,3\dots\}$ and analogously $\mathbb{R}_{0+}$ denotes the set of non-negative real numbers including zero.

Definition 1 (SDF graph) *An* SDF graph $G = (V, E, prod, cons, D_0, \delta)$, *consists of a set of vertices* V, *directed edges* $E : V \times V$, *production rates* $prod : E \to \mathbb{N}$, *consumption rates* $cons : E \to \mathbb{N}$, *an initial token distribution* $D_0 : E \to \mathbb{Z}_{0+}$, *and an execution time function* $\delta : V \to \mathbb{T}$.

In Figure 2.1, an example of an SDF graph is depicted. Vertices of SDF graphs, so-called *actors* V (A, B, and C in Figure 2.1), are representing function executions and communicate data over unidirectional *channels* with First-In-First-Out (FIFO) semantics, represented by edges $e = (v, v') \in E$ (c_0,c_1,c_2, and c_3 in Figure 2.1). Actors v and v' are said to be the source and destination of $e = (v, v')$, denoted by $src(e)$ and $dst(e)$, respectively. Presence and number of (units of) data on channels is represented by *tokens*. A number of *initial tokens* $D_0(e)$ is associated with each channel $e \in E$, which is depicted with bullets and can be found on channels c_0 and c_2 in Figure 2.1. In general, each bullet represents one initial token. Alternatively, for the sake of readability, a number next to a bullet may represent the corresponding number of initial tokens. The number of tokens a source actor v produces on an outgoing channel $e = (v, v') \in E$ is represented by the production rate $prod(e)$ of e. Likewise, the number of

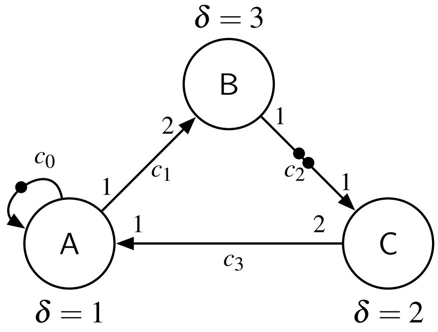


Figure 2.1: Example of a synchronous dataflow graph.

tokens a destination actor $v' \in V$ consumes from an incoming channel $e = (v, v') \in E$ is denoted by the consumption rate $cons(e)$ of channel e. Production and consumption rates of channels are expressed as numbers next to edges, e.g. $prod(c_1) = 1$ and $cons(c_1) = 2$ in Figure 2.1. For the sake of readability, edges whose production or consumption rates equal one, may be visualized without the corresponding annotation. For example, edge c_0 has a production and consumption rate of $prod(c_0) = cons(c_0) = 1$ in Figure 2.1. Actor delays $\delta(v)$ specify an execution time of represented functions and can either be continuous ($\mathbb{T} = \mathbb{R}_{0+}$) or discrete ($\mathbb{T} = \mathbb{Z}_{0+}$). Typically, the latter is the case, which is depicted in Figure 2.1 with annotations next to actors, e.g., $\delta(\mathsf{A}) = 1$. Actors without delay annotations are assumed to have zero delay, if not otherwise indicated.

The *channel state* of an SDF graph G is captured by a function $D : E \to \mathbb{Z}^{0+}$, that assigns a number of tokens to each channel of G. The channel state of an SDF graph dictates, whether actors are *enabled* to *fire* or not.

An actor is enabled to fire, if a minimum number of tokens, denoted by the consumption rate, is present on each input edge, i.e., an actor v is enabled to fire if $\forall e = (v', v) \in E : D(e) \geq cons(e)$ holds. When an actor $v \in V$ fires, $cons(e)$ tokens are removed (consumed) from each incoming edge $e = (v', v) \in E$ and $prod(e')$ tokens are added (produced) on all outgoing edges $e' = (v, v'') \in E$ after $\delta(v)$ time units. In Figure 2.1, only actor C is enabled to fire in the depicted initial channel state.

If the channel state allows, SDF graphs explicitly permit the simultaneous firing of multiple instances of the same actor. This behavior is referred to as *auto-concurrency* and can be constrained by a self-edge ($(v, v) \in E$) on an actor v with unit production and consumption rate and as many initial tokens as maximal simultaneous firings are desired. In Figure 2.1, the auto-concurrency of actor A is eliminated by self-edge c_0 with unit production and consumption rate and a single initial token. As a result, only a single instance of actor A can be fired at a time, forcing a sequential execution. Self-edges can thus be used to model stateful functions. In contrast, actors B and C are allowed for simultaneous firings. In particular, two instances of actor C are enabled to fire in the initial channel state depicted in Figure 2.1.

Definition 2 (Consistency) *Let $\widetilde{\gamma}_G : V \to \mathbb{N}$, denote a vector, that for each actor $v \in V$ assigns a number of firings. If a vector $\widetilde{\gamma}_G$ exists that solves the balance equations $\forall e = (v, v') \in E : \widetilde{\gamma}_G(v) \cdot prod(e) = \widetilde{\gamma}_G(v') \cdot cons(e)$, the SDF graph is said to be* consistent*, i.e., the execution of all actors $v \in V$ firing exactly $\widetilde{\gamma}_G(v)$ times, has no net effect on the token distribution of the SDF graph G.*

The same channel state of a consistent SDF graph is revisited after $\widetilde{\gamma}_G(v)$ respective firings of all $v \in V$. This furthermore implies, that for a consistent SDF graph, infinitely many vectors $\widetilde{\gamma}_G$ exist, i.e., all integer multiples of $\widetilde{\gamma}_G$ solve the balance equations as well. The smallest of such vectors $\gamma_G = min\ \widetilde{\gamma}_G$ with $\gamma_G(v) \in \mathbb{N}$ for all $v \in V$ is referred to as *repetition vector*. As a result, the existence of a repetition vector indicates consistency. The example SDF graph in Figure 2.1 is consistent and has a repetition vector of $\gamma = [2,1,1]^T$ with corresponding order $[\mathsf{A},\mathsf{B},\mathsf{C}]^T$. Hence, A has to be fired twice and both B and C have to be fired once to reach the initial channel state again.

Definition 3 (Iteration) *If $G = (V,E,prod,cons,D_0,\delta)$ is a consistent SDF graph, with repetition vector γ_G, a set of executions in which each actor $v \in V$ fires exactly $\gamma_G(v)$ times, is called an* iteration.

Consistency guarantees that a channel state of an SDF graph is revisited after a single execution of an iteration. Thereafter, an infinite sequence of iterations can be executed with finite buffer sizes.

Definition 4 (State) *Let $G = (V,E,prod,cons,D_0,\delta)$ be a timed SDF graph. Its* state *can be captured by a tuple $S = (D,\tau)$, with D denoting the channel state of G, and $\tau : V \rightarrow \mathbb{T}^*$ a time structure*[2.1]*, that assigns remaining execution times to all instances of an actor that concurrently execute [8]. The initial state of G is thus, (D_0,τ_0), with $\forall v \in V : \tau_0(v) = \{\}$.*

The timing of SDF graphs allows the analysis of timing behavior or schedules. Since execution times may vary due to different hardware aspects like caches, branches, or interrupts, a fully static schedule is often not practical. As a solution, actors are annotated with worst-case execution times of the functions that they represent. This permits so-called *self-timed* schedules [9]. In a self-timed schedule, each actor fires as soon as it is ready to fire. This allows an ordering on actor executions to be determined at compile time when annotated with worst-case execution times, which can be deployed without timing information in a final implementation, thus avoiding run-time scheduling overheads. Furthermore, by assuming self-timed execution, formal analysis of timed SDF graphs considers the best possible timing behavior. In Figure 2.2, the self-timed execution of the SDF graph from Figure 2.1 is depicted.

Due to timing of actors, the initial state $s_0 = (D_0, \mathrm{T}_0) \in S$ may not be reached again in the self-timed execution. However, in [8], Ghamarian et al. prove, that consistent and strongly connected timed SDF graphs reach a steady state in the self-timed execution after a so-called transient phase, i.e., a recurring state $s' \in S$ is revisited periodically. Ghamarian et al. show furthermore that, for strongly connected and consistent SDF graphs, the steady state in the self-timed execution is an integer multiple (including zero) of an iteration. Note that a zero multiple of an iteration indicates a deadlock. However, efficient methods to check for deadlock freedom in SDF graphs exist as well [6, 8]. In the thesis at hand, only consistent and deadlock-free SDF graphs are considered.

In Figure 2.2, the SDF graph is in its initial state $s_0 = ((1,0,2,0),\{\{\},\{\},\{\}\})$ at time 0 and starts with an immediate state transition to $((1,0,0,0),\{\{\},\{\},\{2,2\}\})$ by starting to fire two instances of C at time 0. The steady-state of the self-timed execution begins at time 4 in state $((0,0,0,1),\{\{1\},\{3\},\{\}\})$. Note that multiple state transitions without time progress are represented by a combined transition and thus actor A and B started their firing in the final state at time 4 already. The same state is revisited at time 11 after two graph iterations. As a result, the steady-state period takes 7 time units to execute. It is important to note that iterations can overlap in time, as it can be seen in Figure 2.2.

[2.1] Here, $\mathbb{T}^*$ denotes the Kleene star operator * on the set $\mathbb{T}$, representing the set of all possible sequences of elements from the set $\mathbb{T}$, including the empty set $\{\}$.

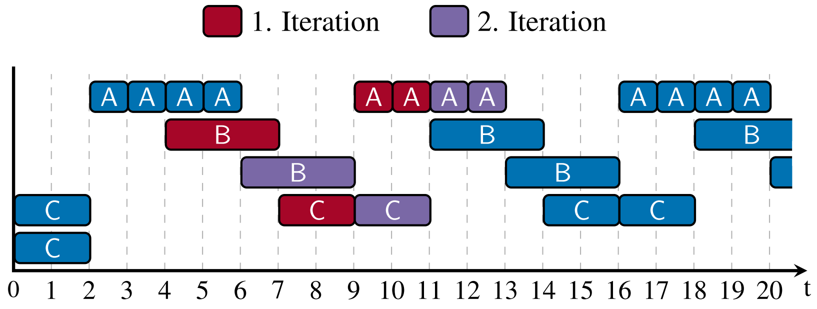


Figure 2.2: Self-timed schedule of the example SDF graph in Figure 2.1.

In the thesis at hand, strong connectedness is not assumed generally. As a result, the self-timed execution of an arbitrary consistent and deadlock-free SDF graph can lead to unbounded channels [10]. Hence, a self-timed schedule is not guaranteed to form a periodical phase in the execution state space. However, Ghamarian et al. introduce in [10] the property of so-called *self-timed boundedness* and a corresponding analysis method. Self-timed boundedness guarantees, that an arbitrary consistent and deadlock-free SDF graph can execute infinitely often in a self-timed schedule with finite buffer sizes on its channels. The maximum achievable average rate, at which an iteration of a consistent SDF graph G can be executed with finite buffer sizes of all its channels $e \in E$, is referred to as the *throughput* $TH(G)$ of graph G. Furthermore, the corresponding maximal average rate at which an actor $v \in V$ of a consistent SDF graph G can be executed with finite buffer sizes of all its channels $e \in E$, is referred to as *actor throughput* $TH(v)$. Due to its consistency, the relationship between $TH(G)$ of SDF graph G and $TH(v)$ and $TH(v')$ of any actors $v, v' \in V$ is:

$$\forall v, v' \in V : TH(G) = \frac{TH(v)}{\gamma_G(v)} = \frac{TH(v')}{\gamma_G(v')}. \tag{1}$$

In [10], it is shown, that the throughput of a consistent and deadlock-free SDF graph G, corresponds to the self-timed execution of G if G is self-timed bounded. Furthermore, Ghamarian et al. show in [10], how the throughput $TH(G)$ of an arbitrary consistent and deadlock-free SDF graph G can be acquired by subsequently calculating the throughput of all its Strongly Connected Components (SCCs).

In the thesis at hand, the average time between two consecutive iteration executions in a schedule of a consistent SDF graph G that achieves the throughput $TH(G)$ is referred to as the *average iteration period* T_G, and is calculated by:

$$T_G = \frac{1}{TH(G)}. \tag{2}$$

The average iteration period of the example SDF graph in Figure 2.1, is 3.5 and its corresponding throughput is $\frac{1}{3.5}$.

There exist many different techniques to analyze the throughput of SDF graphs in the literature. A straight forward method is the transformation of an SDF graph in its corresponding Homogeneous Synchronous Dataflow (HSDF) graph, which is the equivalent of the SDF graph w.r.t. actor firings, however with unit production and consumption rates on edges, and a maximum of one initial token on any edge. Furthermore, in contrast to SDF, HSDF graphs are multigraphs, i.e., multiple equally directed edges can exist between two actors. The transformation of an SDF graph into its equivalent

HSDF representation is described in [7]. From this HSDF representation, the throughput of the corresponding SDF graph can be calculated by a Maximum Cycle Mean (MCM) or Maximum Cycle Ratio (MCR) analysis. For a detailed description of SDF to HSDF transformation and a review and comparison of MCM and MCR analysis methods, the interested reader is referred to [7] and [11], respectively. Although originally limited to strongly connected SDF graphs, throughput analysis based on HSDF conversion can make use of the techniques presented in [10], in order to check for self-timed boundedness or to analyze the maximum achievable throughput for bounded executions. Since conversion from SDF to HSDF graphs may result in an exponential growth of the graph, a more efficient method for acquiring the throughput of SDF graphs from their execution state space is explained in [8]. However, this approach is restricted to strongly connected SDF graphs. De Groote et al. introduced another improvement in [12], that converts an SDF graph into a more reduced representation than HSDF graphs, i.e., Linear Constraint Graphs (LCGs), that allow the throughput computation in a shorter time and a more compact representation than HSDF graphs in many cases. Again, that approach can be applied to arbitrary consistent and deadlock-free SDF graphs with the techniques from [10].

Throughput of consistent and deadlock-free SDF graphs is an essential property on which the proposed methods in the thesis at hand are based on. A tool, that has been extensively used for the evaluations in the thesis at hand is the SDF^3 framework of Stuijk et al. [13]. The SDF^3 framework implements a parser for SDF graphs represented in XML format and offers a wide range of analysis techniques, including consistency, deadlock freedom, repetition vector, and throughput.

Their static consumption and production rates as well as execution times allow for a variety of analyses of SDF graphs at design time. However, their expressiveness is limited and often demands a high abstraction and overestimation when modeling and analyzing state-of-the-art dataflow-oriented applications, leading to pessimistic results. Therefore, a more expressive MoC, that still allows for design-time analysis of properties, that the approaches in the thesis at hand build on, is explained in the following section.

2.1.2 Cyclo-Static Dataflow Graphs

The Cyclo-Static Dataflow (*CSDF*) MoC was introduced by Bilsen et al. in [14] as an extension of a special form of SDF graphs, namely SDF graphs without the ability for auto-concurrency. Such SDF graphs are extended by the concept of cyclic changing behavior. A formalization of the CSDF concept as a proper extension of SDF graphs, i.e., allowing auto-concurrency, was introduced by Stuijk et al. in [15]. Furthermore, Stuijk et al. introduced a formalization of execution times to CSDF actors, similar to timed SDF. In the thesis at hand, the definition of CSDF graphs is based on this formalization. Let $i \bmod_1 n$ be shorthand notation for $(i-1) \bmod n + 1$, with mod defined as the modulo operator, i.e., $a \bmod b = a - b \left\lfloor \frac{a}{b} \right\rfloor$.

Definition 5 (CSDF graph) *A* CSDF graph *G is defined by $G = (V, E, p, c, D_0, d)$ with vertices V, edges $E : V \times V$, production rates $p : E \rightarrow \mathbb{Z}^n_{0+}$, consumption rates $c : E \rightarrow \mathbb{Z}^n_{0+}$, an initial token distribution $D_0(e) : E \rightarrow \mathbb{Z}_{0+}$, and a delay function $d : V \rightarrow \mathbb{T}^n$.*

In Figure 2.3, an example CSDF graph is depicted. Similar as timed SDF graphs, CSDF graphs are composed of actors $v \in V$ represented by vertices, e.g., A, B, and C in Figure 2.3. Actors $v \in V$ communicate tokens over unbounded FIFO channels, referred to as edges $e = (v, v') \in E$ (c_0, c_1, c_2, and c_3 in Figure 2.3). Presence and number of tokens on channels, i.e., the channel state is represented by $D(e)$, with initial channel state $D_0(e)$, that is depicted by bullets in term of number and presence on edges, e.g., on channels c_0 and c_2 in Figure 2.3. In CSDF, actors $v \in V$ are associated with a number of *phases* $\phi(v) \in \mathbb{N}$. The phases represent different behaviors of actors that change in a cyclic manner. Accordingly, the production rates $p(e) = [p_1, p_2, \ldots, p_{\phi(v)}]$ and consumption rates $c(e') = [c_1, c_2, \ldots, c_{\phi(v)}]$ of edges $e = (v, v') \in E$ and $e' = (v'', v) \in E$ that an actor $v \in V$ is connected to as source or destination actor, respectively, are sequences of length $\phi(v)$. Furthermore, associated delays $d(v) = [d_1, d_2, \ldots, d_{\phi(v)}]$, representing execution times of an actor $v \in V$, are sequences of size $\phi(v)$. Execution time sequences $d(v)$, production rate sequences $p(e), e = (v, v') \in E$, and consumption rate sequences $c(e'), e' = (v'', v) \in E$, represent the behavior of actor $v \in V$ in its different phases that repeat with a period of $\phi(v)$, with $\forall v \in V, \forall e = (v, v') \in E, \forall e' = (v'', v) \in E : |p(e)| = |c(e')| = |d(v)| = \phi(v)$. Edges without annotations, are assumed to have unit production and consumption rates in each phase of its source and destination actor, respectively. In Figure 2.3, actor A has three phases, and produces 2 and 1 tokens on channel c_1 and c_0, respectively, consumes two and one tokens from channels c_3 and c_0, respectively, and has an execution time of two, in its first firing. Furthermore, CSDF actors $v \in V$ with $\phi(v) = 1$ resemble SDF actors. In Figure 2.3, actors B and C each have one phase and thus resemble SDF actors. The number of tokens produced on channel $e = (v, v') \in E$ by the i-th firing of actor $v \in V$ is represented by function $prod(e, i) = p_{i \bmod_1 \phi(v)}, i \in \mathbb{N}$. Likewise $cons(e, i) = c_{i \bmod_1 \phi(v)}, i \in \mathbb{N}$ denotes the number of tokens that actor v consumes from channel e in its i-th firing. Finally, $\delta(v, i) = d_{i \bmod_1 \phi(v)}, i \in \mathbb{N}$ denotes the execution time of the i-th firing of actor $v \in V$.

A CSDF actor v in its current phase $j \in [1, \ldots, \phi(v)]$ is enabled to fire, if sufficient tokens on all of its input edges $(v', v) \in V$ are available, i.e., if $\forall (v', v) \in E : D(e) \geq cons(e, j)$ holds. When an actor $v \in V$ fires in its j-th phase, it consumes $cons(e, j)$ tokens from its input channels $e = (v', v) \in E$, produces $prod(e', j)$ tokens after $\delta(v, j)$ time units on all of its output edges $e' = (v, v'') \in E$, and changes its phase to $(j+1) \bmod_1 \phi(v)$. In Figure 2.3, two instances of actor C are enabled in the initial channel state depicted in Figure 2.3. While auto-concurrency of CSDF graphs is assumed throughout the thesis

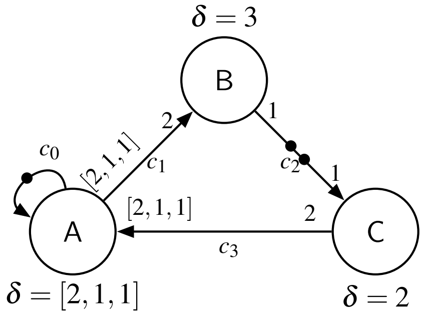


Figure 2.3: Example of a cyclo-static dataflow graph.

at hand, the presented approaches are applicable for CSDF graphs without auto-concurrent behavior to most extend. This is further detailed at the end of this section.

Note that in contrast to the above definition, the number of phases of all actors $v \in V$ needs to be equal when using SDF^3 for analysis. The presented methods in [15] however, as well as design and analysis approaches introduced in the thesis at hand, are applicable to CSDF graphs with different lengths of phase sequences among its actors. Furthermore, as stated in [15], the sequence of actor phases can be concatenated for each actor in a way, such that the length of the resulting sequence is equal to the least common multiple of all actors individual lengths of phase sequences before concatenation, to acquire an CSDF graph that can be analyzed with SDF^3.

The repetition vector $\gamma_G \in \mathbb{N}^{|V|}$ of a CSDF graph G is composed of $\gamma_G(v) = \phi(v) \cdot r(v)$ for all actors $v \in V$, with $r \in \mathbb{N}^{|V|}$ being the smallest vector that solves the following balance equations:

$$\forall e = (v, v') \in E : r(v) \cdot \sum_{i=1}^{\phi(v)} prod(e,i) = r(v') \cdot \sum_{j=1}^{\phi(v')} cons(e,j).$$

Similar to SDF graphs, the existence of a vector r and thus a repetition vector γ_G of CSDF graph G guarantees consistency and γ_G describes an iteration of G. Furthermore, after executing an iteration of CSDF graph G, the same channel state as before the execution is reached again. The depicted CSDF graph in Figure 2.3 is consistent and has a repetition vector of $\gamma = [3,2,2]^T$ with corresponding order $[\mathsf{A}, \mathsf{B}, \mathsf{C}]^T$.

In [15], Stuijk et al. generalize from [8]. Hence, the state space of CSDF graphs with bounded storage space consists of a transient phase followed by a steady-state. Consequently, considering the results from [10], the steady-state represents the self-timed execution in case the CSDF graph is self-timed bounded, otherwise it represents the fastest possible schedule in terms of throughput, which can be executed infinitely often with finite buffer size usage on all channels. The self-timed schedule of the example CSDF graph in Figure 2.3, is depicted in Figure 2.4. The CSDF graph is consistent, deadlock free, and strongly connected (and therefore self-timed bounded [10]) and its steady-state begins at time 4 which is revisited at time 11. Its steady-state consists of a single iteration, which is highlighted in Figure 2.4.

Similar to SDF graphs, the throughput $TH(G)$ of a consistent CSDF graph G is defined as the maximum achievable average rate of iterations, that can be executed infinitely often with finite buffer size usage

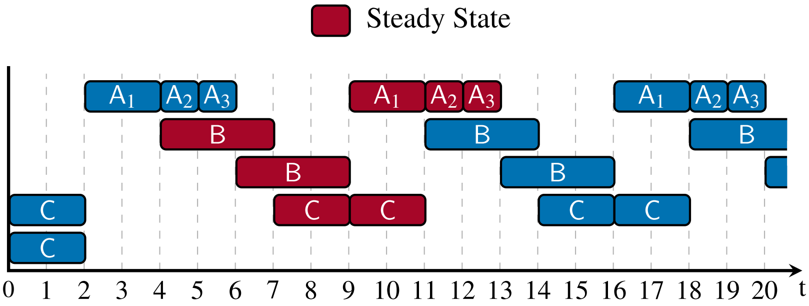


Figure 2.4: Self-timed schedule of the example CSDF graph in Figure 2.3.

of all channels $e \in E$. Accordingly, the actor throughput $TH(v)$ of any actor $v \in V$ can be derived from the graph throughput $TH(G)$ and the repetition vector entry $\gamma_G(v)$ by Equation (1). In the thesis at hand, the reciprocal of the throughput $TH(G)$ of a consistent CSDF graph G is denoted as average iteration period T_G, calculated by Equation (2). The throughput of the example CSDF graph from Figure 2.3, is $TH(G) = \frac{1}{7}$ and its corresponding average iteration period is thus $T_G = 7$.

Throughput of CSDF graphs can be calculated by a transformation into equivalent HSDF graphs and subsequent MCR analysis [14]. For arbitrary consistent CSDF graphs, the subsequent throughput calculation on all SCCs of the HSDF graph can derive the maximum achievable throughput of all infinite schedules with bounded channels [10]. Multiple optimizations have been introduced to transform a CSDF graph into a reduced presentation of SDF graphs [16] or HSDF graphs [12, 17] from which throughput can be calculated by using the techniques of [10].

Since the definition of CSDF graphs in [15] as well the implementation of [13] allow auto-concurrency of CSDF actors, the order of tokens produced on a channel by subsequent firings of an actor can change due to different execution times and thus break so-called *functional determinacy* [17, 18]. That is, in general, the order of actor firings in SDF graphs and thus, the execution times of actors do not influence the result. However, if a concurrent phase of an actor can overtake another, the result might indeed be changed, i.e., the graph is not functionally determinate anymore. However, in the modeling and analysis approach presented in the thesis at hand, analysis models are eliminating auto-concurrency of CSDF actors. Additional actors with enabled auto-concurrency for model refinement may be added to analysis models, as will be explained in Section 4.5.4. However, these additional actors are restricted to SDF actors, which preserves functional determinacy [16].

Due to the cyclic behavior of actors, the CSDF MoC is more expressive than SDF. Although analysis of CSDF graphs may require a higher computational effort, due to subsequent transformations to SDF or directly HSDF graphs and a corresponding increase in graph size [16], design-time analysis of important properties like consistency and throughput is still possible. Thus CSDF graphs allow a tighter modeling of dataflow-oriented applications and systems, that follow a periodic behavior.

However, dynamic behavior, e.g., data-dependent execution, cannot be modeled efficiently with CSDF graphs. There exist a variety of dataflow MoCs allowing for data-dependent behavior, such as Boolean Dataflow (BDF), Dynamic Dataflow (DDF), or Kahn Process Network (KPN), which in general however, are not analyzable w.r.t. throughput, consistency, or deadlock freedom at design time [5]. As a remedy, the scenario-aware dataflow (SADF) MoC allows to represent dynamic behavior that is restricted to a finite number of scenarios for which the behavior is static, but scenario changes can occur. The SADF MoC is analyzable w.r.t. throughput, consistency, deadlock freedom, and a variety of long-run average metrics, at design time. In the following section, the SADF MoC and the most important properties that can be analyzed from it are explained in more detail.

2.1.3 Scenario-Aware Dataflow Graphs

The Scenario-Aware Dataflow (SADF) MoC has been introduced by Theelen et al. in [19]. Scenario-aware dataflow graphs express dynamic behavior in form of finite sets of operational modes of actors with corresponding probabilistic information on possible execution orders. SADF graphs are therefore more expressive than CSDF graphs, but still analyzable at design time.

SADF graphs extend SDF and CSDF graphs by the concepts of scenarios. To this end, two kinds of actors exist for SADF graphs, namely *kernels* and *detectors*. Kernels are similar to actors in SDF and CSDF, but can operate in different scenarios. Detectors are special actors, that control the scenarios of kernels. The scenario of a kernel can be controlled by a detector, via a so-called *control channel* that transports *control tokens*, which select the corresponding scenario of the kernel. Dynamic behavior is modeled by scenario transitions. In the SADF MoC that is used in the thesis at hand, scenario transitions are modeled by probabilistic choices, i.e., as a discrete-time Markov chain. Besides the aforementioned, there exists another variant of SADF, namely Finite State Machine (FSM) SADF, that models choices by a finite state machine [5]. In FSM-SADF, however, zero production or consumption rates, like in CSDF, are not allowed, while Markov chain SADF does allow for zero production and consumption rates. In turn, Markov chain SADF does not support auto-concurrency as described in [19], which is possible in CSDF and FSM-SADF. However, as discussed in Section 2.1.2, this does not affect the modeling and analysis approaches introduced in the thesis at hand, as auto-concurrency is removed from final analysis models. Hence, the thesis at hand focuses on Markov chain SADF, which will be referred to as SADF in the following.

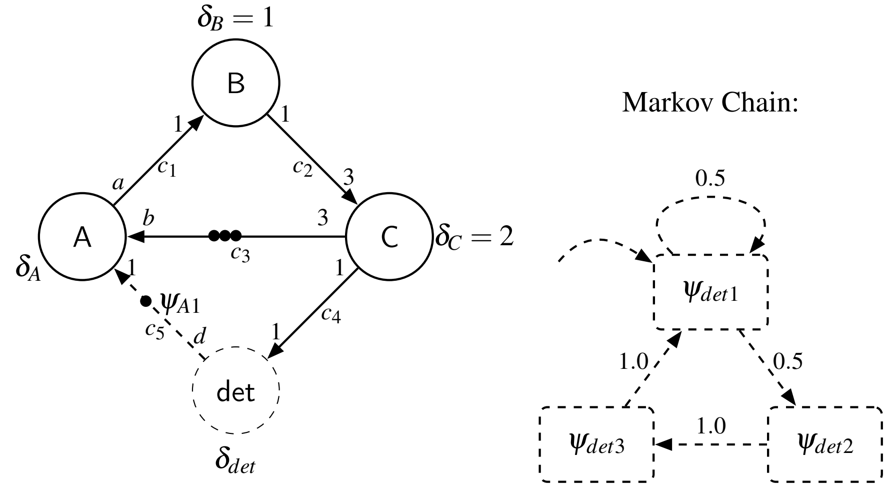
Let Ψ denote a finite set of scenarios. The set of all possible non-empty sequences of scenarios $\psi \in \Psi$, is denoted by Ψ^+[2.2]. A sequence of scenarios is indicated by $\widetilde{\psi} \in \Psi^+$ and the length of a scenario sequence is denoted by $|\widetilde{\psi}|$. Furthermore, a sequence of n equal scenarios $\psi \in \Psi$ is denoted by ψ^n.

Definition 6 (SADF graph) *An* SADF graph *is defined as* $G = (V, E, p, c, D_0, d)$ *with vertices* $V = V_k \cup V_d$*, edges* $E = E_d \cup E_c$*, with* $E_d : V \times V$ *and* $E_c : V_d \times V_k$*, production rates* $p : E \to \mathbb{Z}^n_{0+} \cup {\Psi^+}^n$*, consumption rates* $c : E \to \mathbb{Z}^n_{0+}$*, an initial token distribution* $D_0 = (D_{d_0}, D_{c_0})$*, with* $D_{d_0} : E_d \to \mathbb{Z}_{0+}$ *and* $D_{c_0} : E_c \to \Psi^+$*, and a delay function* $d : V \to \mathbb{T}^n$*.*

An example SADF graph is depicted in Figure 2.5. Similar to CSDF actor phases, SADF actors $v \in V$ are associated with a set of *scenarios* Ψ_v. However, in SADF graphs, scenarios are not necessarily changing in a cyclic manner, but are rather controlled by special actors. To this end, two kinds of actors, i.e., *kernels* V_k and *detectors* V_d, build the set of actors $V = V_k \cup V_d$, with $|V_k| \geq 1$. Actors are depicted as vertices, with solid lines for kernels (A,B, and C in Figure 2.5(a)) and dashed lines for detectors (det in Figure 2.5(a)). According to the original definition in [19], an SADF graph can consist of multiple detectors. However, for the sake of brevity, the number of detectors in V_d is assumed to equal one in the thesis at hand, i.e., $|V_d| = 1$. Furthermore, an actor $v \in V$ can either be

---

[2.2] Here, Ψ^+ denotes the Kleene plus operator $^+$ on the set Ψ, representing the set of all possible sequences of elements from the set Ψ, which are non-empty.



(a) (b)

Figure 2.5: Example of a scenario-aware dataflow graph.

kernel or detector, with $V_k \cap V_d = \varnothing$. Kernels and detectors communicate tokens over unbounded FIFO channels E represented by edges. The set of channels E is composed of *data channels* $E_d \subseteq V \times V$ (similar to channels of SDF and CSDF graphs) which are depicted as solid edges and so-called *control channels* $E_c \subseteq V_d \times V_k$ depicted as dashed edges, with $E = E_d \cup E_c$ and $E_c \cap E_d = \varnothing$. Moreover, the number of control channels between each pair of detector $v_d \in V_d$ and kernels $v_k \in V_k$ is at most one, i.e., $\forall (v_d, v_k) \in V_d \times V_k : |\{(v_d, v_k) \in E_c\}| \leq 1$. While data channels e_d communicate abstract units of data between actors without values (like in SDF and CSDF), each control channel $e_c = (v_d, v_k)$ is associated with a finite set of control token values, that corresponds to the set of scenarios ψ_{v_k} of the controlled kernel v_k.[2.3] To this end, for each control channel $e_c = (v', v) \in E_c$, Ψ_v also denotes the finite set of possible token values that can be transferred over the control channel, which is referred to as *channel alphabet*. The production rate of an actor $v \in V$ in its scenario $\psi_v \in \Psi_v$ on channel $e = (v, v') \in E$ is denoted by $p(e, \psi_v)$. Function $p(e, \psi_v)$ returns a number $n \in \mathbb{Z}_{0+}$ in case $e \in E_d$ is a data channel. Otherwise, i.e., if $e \in E_c$ is a control channel, $p(e, \psi_v)$ returns a non-empty sequence $\widetilde{\psi} = \psi^n, \psi \in \Psi_{v'}, n \in \mathbb{N}$ of scenarios of its destination kernel $v' \in V_k$. The consumption rate on channel $e' = (v'', v)$ is denoted by $c(e', \psi_v)$ and returns a number $n \in \mathbb{Z}_{0+}$ in case $e' \in E_d$ and $c(e', \psi_v) = 1$, in case $e' \in E_c$. That is, the consumption rate on each control channel equals one. Furthermore, the consumption rate on all channels $(v, v_d) \in E_d$ is equal in all scenarios of detector $\psi_{v_d} \in \Psi_{v_d}$. Production rates and consumption rates that are changing among scenarios are depicted as parameters on edges, e.g., a, b, and d in Figure 2.5(a). Their corresponding values for each scenario can be found in Table 2.1.

While in [19], an execution time distribution with a finite sample space is associated with each actor $v \in V$ for each of its scenarios $\psi_v \in \Psi_v$, in the thesis at hand, execution times are assumed to be fixed per scenario, which are denoted by $d(v, \psi_v)$. Furthermore, execution times are assumed to be non-negative integers, i.e., $\mathbb{T} = \mathbb{Z}_{0+}$. Execution times are depicted next to actors by either their corresponding value

---

[2.3] This definition and the following definitions are adapted to the case of a single detector, which is assumed in the thesis at hand. For a definition with multiple detectors, the interested reader is referred to [19].

in all scenarios if applicable (SDF actors δ_B and δ_C in Figure 2.5(a)), or by parameters (δ_A and δ_{det} in Figure 2.5(a)) in case of varying execution times among scenarios. The corresponding execution times of the example SADF graph in Figure 2.5(a) can be found in Table 2.1 for each scenario.

In the thesis at hand, the *channel state* D of an SADF graph G is captured by a tuple (D_d, D_c). It is composed of the data channel state $D_d(e)$, assigning a number of tokens to each data channel $e \in E_d$, and the control channel state $D_c(e)$, assigning a control token sequence $\widetilde{\psi} \in \Psi^*_{v'}$ to each control channel $e = (v_d, v') \in E_c$ that kernel v' is connected to as destination actor. The initial channel state is denoted by $D_0 = (D_{d_0}, D_{c_0})$, depicted as bullets on data edges, e.g., on c_3 in Figure 2.5(a), or by bullets and their associated value on control channels, e.g., on channel c_5 in Figure 2.5(a). Note that according to its original definition in [19], in the initial channel state, the number of control tokens on each control channel $e_c \in E_c$ is zero. However, in the thesis at hand, this restriction is relaxed, which is in line with the SADF implementation in [20].

Kernels $v_k \in V_k$ without a control channel (B and C in Figure 2.5(a)), are supposed to represent static behavior, i.e., SDF behavior. and their number of scenarios equals $|\Psi_{v_k}| = 1$. Kernels with a control channel are referred to as *controlled kernels* (A in Figure 2.5(a)). Furthermore, while kernel scenarios are controlled by consuming control tokens values from their control channels, the detector scenarios $\psi_{v_d} \in \Psi_{v_d}$ are changed by an associated Markov chain [21] with detector v_d. A Markov chain is a triple $(\mathbb{S}, \iota, \mathbb{P})$, with $\mathbb{S}$ denoting a finite set of states, ι denoting an initial state with probability 1, and $\mathbb{P}$ denoting an $|\mathbb{S}| \times |\mathbb{S}|$ matrix. Each entry $\mathbb{P}(S,T) \in [0,1]$ denotes a transition probability from state $S \in \mathbb{S}$ to $T \in \mathbb{S}$ with $\sum_{T \in \mathbb{S}} \mathbb{P}(S,T) = 1$, for all $S \in \mathbb{S}$. The set of states $\mathbb{S}$ of its Markov chain, represents the scenarios Ψ_{v_d} of detector $v_d \in V_d$. Hence, the scenarios of detectors are controlled by its Markov chain states and change with their associated transitions and corresponding transition probabilities. The Markov chain of detector det in Figure 2.5(a) is depicted in Figure 2.5(b), with states ψ_{det1}, ψ_{det2}, and ψ_{det3} and scenario transitions as dashed edges annotated with their associated transition probabilities. Note that edges representing transitions with zero probability are omitted. The initial state ψ_{det1} is indicated by a dashed edge without a source node. An important property of the Markov chain for the computation of long-run average metrics from the SADF graph, is ergodicity. That is, the Markov chain has a single strongly connected component of recurrent states and possibly multiple components of transient states. A recurrent state is a state that is reachable from all other states with probability 1. Otherwise it is called transient [19].

Similar as in (C)SDF, an SADF actor can fire, when sufficient tokens are available on all its input channels. However, each actor firing is preceded by a transition phase, in which the actor performs a scenario transition, based on a consumed control token for controlled kernels $v_k \in V_k$ when it is

Table 2.1: Scenario parameters of the example SADF graph in Figure 2.5.

Scenario	d	δ_{det}	a	b	δ_A
ψ_{det1}	ψ_{A1}	3	/	/	/
ψ_{det2}	$\psi_{A2}, \psi_{A2}, \psi_{A2}$	1	/	/	/
ψ_{det3}	ψ_{A1}	2	/	/	/
ψ_{A1}	/	/	3	3	2
ψ_{A2}	/	/	1	1	1

available on its control channel $(v_d, v_k) \in E_c$, or based on a Markov chain state transition for detector $v_d \in V_d$. Kernels without a control channel simply remain in their single scenario. The transition phase can happen earliest when an actor finished its previous firing and additionally for controlled kernels when a control token is present on its control channel. Note that this explicitly excludes auto-concurrency for SADF graphs. After the transition phase, each actor $v \in V$ waits until sufficient data tokens are available on all edges $(v', v) \in E_d$ that v is connected to as destination actor, i.e., until $D_d(e_d) \geq c(e_d, \psi_v)$. After sufficient data tokens are available, actor $v \in V$ can start firing by consuming $c(e_d, \psi_v)$ tokens from all channels $e_d = (v', v) \in E_d$ and producing $p(e', \psi_v)$ tokens on all channels $e' = (v, v') \in E$ after $d(v, \psi_v)$ time units.

Let $prod : E_d \times \widetilde{\psi} \rightarrow \mathbb{N}$ denote a function, that returns the accumulated number of tokens that actor $v \in V$ produces on data channel $e = (v, v') \in E_d$ after an execution of a particular sequence $\widetilde{\psi} \in \Psi_v^+$ of its scenarios Ψ_v, with $prod(e, \widetilde{\psi}) = \sum_{\psi \in \widetilde{\psi}} p(e, \psi)$. Accordingly, let $cons : E_d \times \widetilde{\psi} \rightarrow \mathbb{N}$ denote a function, that returns the accumulated number of tokens that actor $v' \in V$ consumes from data channel $e = (v, v') \in E_d$ after an execution of a particular sequence $\widetilde{\psi} \in \Psi_{v'}^+$ of its scenarios $\Psi_{v'}$, with $cons(e, \widetilde{\psi}) = \sum_{\psi \in \widetilde{\psi}} c(e, \psi)$. Function $\delta : V \times \widetilde{\psi} \rightarrow \mathbb{Z}_{0+}$ returns the accumulated execution times of an actor $v \in V$ corresponding to an execution of a particular sequence $\widetilde{\psi} \in \Psi_v^+$ of its scenarios $\psi_v \in \Psi_v$, with $\delta(v, \widetilde{\psi}) = \sum_{\psi \in \widetilde{\psi}} d(v, \psi)$. Let further $subs(v, \psi_{v_d})$ denote a function, that assigns to each actor $v \in V$a so-called *sub-scenario sequence* $\widetilde{\psi} = \psi_v^n$ of one of its scenarios $\psi_v \in \Psi_v$ with $n \in \mathbb{N}$, for each scenario $\psi_{v_d} \in \Psi_{v_d}$ of detector v_d, with:

$$subs(v, \psi_{v_d}) : \begin{cases} \psi_{v_d} & \text{if } v = v_d, \\ \psi_v & \text{if } \nexists (v_d, v) \in E_c, \\ p((v_d, v), \psi_{v_d}) & \text{otherwise,} \end{cases}$$

with $\nexists$ denoting non-existence. That is, in case $v \in V$ is the detector v_d itself, each sub-scenario sequence of v_d for each of its scenarios $\psi_{v_d} \in \Psi_{v_d}$ is simply the corresponding scenario ψ_{v_d} itself. In case $v \in V$ is a kernel without a control channel, each of its sub-scenario sequences $subs(v, \psi_{v_d})$ is simply its single static scenario $\psi_v \in \Psi_v$ itself. Otherwise $v \in V$ reflects a controlled kernel, for which each of its sub-scenario sequences $subs(v, \psi_{v_d})$ reflects the scenario sequence represented by the control token production rate $p((v_d, v), \psi_{v_d})$ of detector $v_d \in V_d$ in scenario $\psi_{v_d} \in \Psi_{v_d}$ on its control channel $(v_d, v) \in E_c$. With this, the repetition vector of an SADF graph G can be conveniently constructed.

In the thesis at hand, a repetition vector γ_G of an SADF graph G is a function $\gamma_G : V \times \Psi \rightarrow \mathbb{N}$ that assigns for each detector scenario $\psi_{v_d} \in \Psi_{v_d}$ a number $n \in \mathbb{N}$ to each actor $v \in V$, such that the following two balance equations are solved:

$$\begin{aligned} &\forall \psi_{v_d} \in \Psi_{v_d}, \forall e = (v, v') \in E_d : \\ &\gamma_G(v, \psi_{v_d}) \cdot prod(e, subs(v, \psi_{v_d})) = \gamma_G(v', \psi_{v_d}) \cdot cons(e, subs(v' \psi_{v_d})), \end{aligned} \tag{3}$$

$$\begin{aligned} &\forall \psi_{v_d} \in \Psi_{v_d}, \forall e = (v_d, v') \in E_c : \\ &\gamma_G(v_d, \psi_{v_d}) \cdot |p(e, \psi_{v_d})| = \gamma_G(v', \psi_{v_d}). \end{aligned} \tag{4}$$

Equation (3) states, that the number of produced data tokens on each data channel of G equals the number of consumed tokens, in each detector scenario. Equation (4) states, that the number of produced control tokens equals the number of consumed control tokens on each control channel of G in each individual detector scenario. Equation (4) directly follows from the unit consumption rate of control channels.

The existence of a repetition vector is a necessary condition for bounded infinite executions and the absence of deadlocks for an SADF graph G. However, iterations do not necessarily correspond to single scenarios, but can span over multiple detector executions with corresponding scenario transitions. To this end, the property of *strong consistency* was introduced in [19].

Definition 7 (Strong consistency) *An SADF graph G is said to be strongly consistent if and only if it has a repetition vector γ_G, whose entries for the detector equal one in all of its scenarios $\psi_{v_d} \in \Psi_{v_d}$, i.e., $\forall \psi_{v_d} \in \Psi_{v_d} : \gamma_G(v_d, \psi_{v_d}) = 1$. The smallest repetition vector γ_G of a strongly consistent SADF graph is referred to as* the *repetition vector of G.*

The example SADF graph in Figure 2.5(a) is strongly consistent, and its repetition vector for scenarios ψ_{det1}, ψ_{det2}, and ψ_{det3} returns $[1,3,1,1]^T$, $[3,3,1,1]^T$, and $[1,3,1,1]^T$, respectively, with order $[\mathsf{A},\mathsf{B},\mathsf{C},\mathsf{det}]^T$.

Due to the concept of choice of scenarios, an iteration of an SADF graph G is not as straight forward to define as for (C)SDF graphs. As a result, an iteration of an SADF graph always corresponds to a particular scenario $\psi_{v_d} \in \Psi_{v_d}$ of the detector, in the thesis at hand. In all scenarios $\psi_{v_d} \in \Psi_{v_d}$, each actor $v \in V$ fires exactly $\gamma_G(v, \psi_{v_d})$ times. For the same reason, only the actor throughput $TH(v)$ is defined for strongly consistent SADF graphs as an expected long term average rate of firing completions of $v \in V$. By considering all possible scenario transitions and their respective transition probabilities, the actor throughput $TH(v)$ of a strongly consistent SADF graph G is the maximum achievable rate with which an actor $v \in V$ can fire per time unit on average, among all possible execution schedules that correspond to the transition probabilities of the associated Markov chain and which can be executed infinitely often with finite buffer sizes of all channels $e \in E$. Note that due to considering the transition probabilities of the detector Markov chain, the actor throughput does not reflect a worst-case achievable actor throughput for an associated application with worst-case execution time annotations, as it does not correspond to the worst-case execution path among all possible scenario transitions, as, e.g., in [22]. However, it indeed corresponds to a worst-case actor throughput of an application modeled as a corresponding SDF or CSDF graph, if the Markov chain would consist of a single state, or transitions would enforce a strictly cyclo-static transition scheme, respectively.

Furthermore, the throughput analysis of SADF graphs additionally relies on so-called *strong dependency*, which relates to strong connectedness for SDF and CSDF graphs, but across scenario transitions. For a detailed explanation, the interested reader is referred to [19]. While for SDF and CSDF graphs the throughput analysis methods implicitly return the throughput of the fastest schedule that is infinitely often executable with finite buffer sizes in case the graph is not strongly connected, the throughput analysis methods for SADF graphs require strong dependency. However, some of the analysis approaches in the thesis at hand rely on modeling only such channels that actually communicate data in the anticipated system. As a result, feedback channels between processors or devices might not exist in the analysis models, which would break strong connectedness or strong dependency, respectively. As a solution, a maximal buffer size with a corresponding value is specified for such channels in SADF models. Specifying a maximum buffer size of a channel is equally treated by SADF analysis methods in [20] as modeling a corresponding feedback channel in the model with the respective number of initial tokens.

The execution of an SADF graph can be captured by a Markov state space, which is, for the sake of brevity, not explained in the thesis at hand. For a detailed description of the execution state space of

SADF graphs and the corresponding derivation of analysis methods, the interested reader is referred to [19] and [23]. Furthermore, in [20], existing analysis techniques categorized into extrema, reachability, and long-run average metrics are summarized and implemented as a module that extends SDF^3 [13].

One of the worst-case metrics of an SADF graph that is important for the approaches in the thesis at hand, is the *maximum inter firing latency* (IFL) of an actor $v \in V$. This is the maximum time between two successive completed firings of an actor $v \in V$ corresponding to the self-timed schedule considering all possible execution paths in the Markov state space of SADF graph G. Similarly, the average inter firing latency $\overline{IFL}(v)$ denotes the long-run average time between successive firings of an actor $v \in V$, which is reciprocal to its throughput, i.e., $\overline{IFL}(v) = \frac{1}{TH(v)}$.

Another important worst-case metric for the approaches in the thesis at hand, is the *maximum response time* of an actor $v \in V$, that is the latest time at which an actor $v \in V$ completes its first firing in the self-timed schedule considering all possible paths in the Markov state spaces of an SADF graph G.

Finally, modeling and analysis approaches presented in the thesis at hand, rely on the *scenario occurrence probability* of actor scenarios in SADF graphs, which was introduced in [G7]. The scenario occurrence probability $\pi(v, \psi_v)$ is a long-run expected occurrence probability for each scenario $\psi_v \in \Psi_v$ of actor $v \in V$ of an SADF graph G, with $\sum_{\psi_v \in \Psi_v} \pi(v, \psi_v) = 1$, for all actors $v \in V$. It is calculated from the equilibrium distribution of the detector Markov chain and the scenario value of the associated sub-scenario sequence of actor v.

2.2 System Modeling

In the thesis at hand, a basic system-modeling approach has been chosen that is based on the well-known Y-chart methodology [24], specifying individual models for the application and the architecture. Both models are then combined by a software to hardware mapping and a scheduling. Many modeling approaches are utilizing this concept of separation of concerns [25–28] (sometimes also referred to as orthogonalization of concerns [29]) since it allows the quantification of design choices regarding software, hardware, mapping, or scheduling. Furthermore, separation of concerns fosters application model and hardware model reuse.

In the thesis at hand, the application model is represented by dataflow-based MoCs, as presented in the previous sections. An SDF application model of an example software is shown in Figure 2.6(a). Actor S1 models a sensor sampling process with a period of $20ms$. A self-edge c_{SE-0} ensures to eliminate auto-concurrent behavior. Each sample consists of two units of data, e.g., an accelerometer and gyroscope sample. Actor P1 consumes each token and processes it individually, i.e., consumption rate of 1 on channel c_1. This actor represents a stateless function that can be executed in parallel for each sample. Finally, actor P2 represents a second function, consuming both processed samples from P1 together. In order to execute this application, a model of a hardware platform needs to be specified.

Let $pre(v)$ of actor $v \in V$ of a (C)SDF or SADF graph G denote a predecessor $v' \in V$ that is connected to v via a data channel $(v', v) \in E_d$ with $v \neq v'$.
A *hardware model* $H = (P, S, T)$ consists of heterogeneous processing elements P, data source elements, i.e., sensors S, and a set of directed communication channels $T \subseteq S \times P \cup P \times P$ from sensors $s \in S$ and to and between processors $p \in P$. Processors $p \in P$ can execute actors that have predecessor actors. Sensors $s \in S$ are representing data sources and thus, execute actors without predecessors. Hardware communication channels $t \in T$ communicate data between sensors $s \in S$ and processors

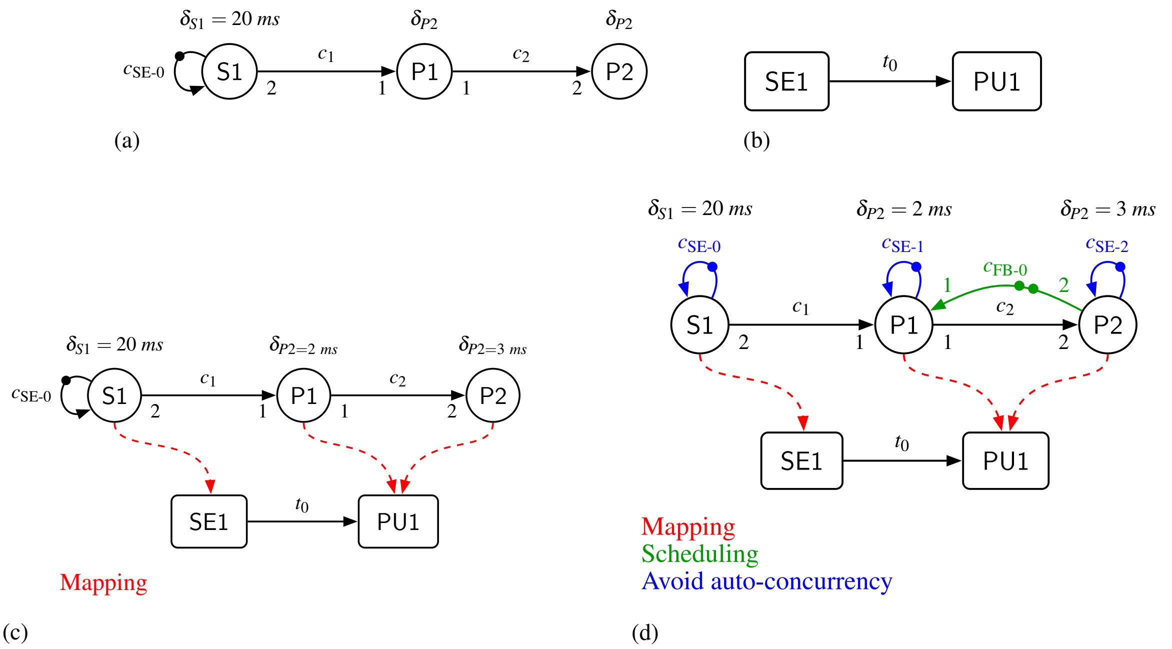


Figure 2.6: Depiction of the design flow from an application model (a), a hardware model (b) including mapping (c), and scheduling (d) to an analysis model.

$p \in P$. In the thesis at hand, at most one hardware channel per direction is assumed to connect two processing elements or sensors, i.e., $\forall (p, p') \in S \times P \cup P \times P : |\{(p, p') \in T\}| \leq 1$. An example hardware model is shown in Figure 2.6(b), consisting of a sensor SE1, a processing unit PU1, and a hardware communication channel t_0 from SE1 to PU1.

In order to select, which actor is executed on which hardware element, a mapping needs to be specified. Given a dataflow graph G and a hardware model H, a *mapping* $M = (M_V, M_E)$ consists of an *actor mapping* $M_V : V \rightarrow S \cup P$ that maps each actor $v \in V$ of G to a processing element $p \in P$ or sensor $s \in S$, and a *channel mapping* $M_E \subset E_d \times T$ that assigns data channels $e \in E_d$ of G to hardware channels $t \in T$ of H, such that each channel $e \in E$ of G is assigned at most once to one of the hardware channels $t \in T$ of hardware model H, i.e., $\forall e \in E : |\{(e', t) \in M_E \mid e' = e\}| \leq 1$. Note that this explicitly excludes the mapping of control channels of SADF graphs. Furthermore, a mapping is only valid if the pair of actors connected by an edge $(v, v') \in E$ are either mapped to the same processor or to sensors and processors which are directly connected by an equally directed hardware channel $t \in T$ of H, i.e., $\forall e = (v, v') \in E,\ \exists (v, p), (v', p') \in M_V : \exists (p, p') \in T \vee p = p'$. Furthermore, the number of actors mapped to a sensor $s \in S$ is at most 1, i.e., $\forall s \in S : |\{(v, s') \in M_V \mid s' = s\}| \leq 1$ and no predecessor $pre(v)$ exists for actors $v \in V$ that are mapped to one of the sensors $s \in S$, i.e., $\forall (v, s) \in M_V : \nexists\ pre(v)$. Note that due to the restriction of a single hardware channel per direction between processing elements or sensors, channel mapping M_E can be implicitly derived from M_V and is not further indicated in figures in the thesis at hand. Note that actor execution times typically depend on the underlying processor hardware. As a result, actor execution times in the application model are assigned based on a particular mapping. In Figure 2.6(c), an example mapping $M = (\{(\text{S1}, \text{SE1}), (\text{P1}, \text{PU1}), (\text{P2}, \text{PU1})\}, \{(c_1, t_0), (c_2, \varnothing)\})$

between the application graph in Figure 2.6(a) and the hardware model in Figure 2.6(b), is depicted by red dashed mapping edges.

To finalize the specification of the system, concurrent execution of actors mapped to the same processor need to be eliminated. This is achieved, by introducing self-edges with unit production and consumption rates and a single token on them to all actors, in order to avoid auto-concurrency. Secondly, scheduling edges need to be added in a way, such that all executions of actors mapped to the same processor unit are executed sequentially. Techniques to derive viable schedules are out of scope in the thesis at hand. However, several works exist on scheduling SDF, CSDF and dataflow graphs in general [30–33], that only use elements of the respective MoC. This allows analysis of the resulting model with existing dataflow-based techniques. The resulting mapped and scheduled model, is referred to as *analysis model*. The final analysis model of the mapped and scheduled example application with scheduling edges in green and self-edges in blue, is depicted in Figure 2.6(d).

Note that although the dataflow graph of the analysis model (see Figure 2.7) implicitly contains decisions regarding a particular mapping and schedule, the explicit information of the hardware platform and mapping cannot be discarded, as these and additional annotation of hardware components allow estimation of further extra-functional properties, which will be subject in Section 3.4 of the following chapter.

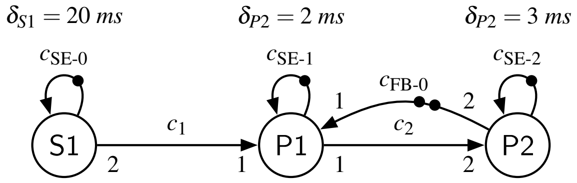


Figure 2.7: Dataflow graph of the example analysis model.

3 Energy-Efficient Sensor Networks

The approaches and results described in the following have partially been published previously by the author in the following publications:

- Towards Energy Efficient Sensor Nodes for Online Activity Recognition [G1]
- Energy Efficient On-Sensor Processing for Online Activity Recognition [G4]
- Model-Based Design of Energy-Efficient Human Activity Recognition Systems with Wearable Sensors [G2]
- Model-Based Real Time Analysis of Distributed Human Activity Recognition Stages in Wireless Sensor Networks [G3]

The energy consumption of wireless sensor nodes in human activity and gesture recognition systems plays a key role in the applicability and usability of the entire system. Lower energy consumption prolongs the lifetime of batteries or allows to deploy batteries with smaller capacities, which can be smaller in size and thus more unobtrusive for the user. However, existing optimization techniques for reduced energy consumption of HAR systems are either application specific or do not allow a quantitative estimate on the energy consumption saving between possible configurations. Furthermore, estimating energy consumption at design time is crucial in order to substantiate design decisions early in the design process. Orthogonal to existing optimization techniques for energy-efficient activity recognition systems, the thesis at hand focuses on the energy-efficient distribution of recognition software onto the hardware of the system, i.e., *mapping*. Different mappings result in differently utilized processing units and communication channels. Their respective energy consumption is thus affected and needs to be traded off for optimal energy consumption of devices. In order to compare different mappings at design time, design and analysis approaches are necessary that capture important extra-functional properties of the system. These have to provide a sufficient accuracy w.r.t. the final implementation of the system. As a necessary prerequisite for the aforementioned methodology, this chapter focuses on the question how human activity recognition systems can be efficiently modeled at a system-level, that is abstract enough w.r.t. analysis speed, but still allows for accurate enough estimations of device energy consumption. To this end, presented approaches are evaluated on state-of-the-art systems and compared to their respective implementations in experiments.

In the thesis at hand, modeling and analysis methods are based on dataflow graphs that capture important software and hardware aspects. Based on existing analysis techniques, novel methods for estimating device energy consumption at design time are introduced. As a resulting methodology, activity recognition software that has been designed and optimized towards recognition accuracy offline is modeled and analyzed at design time to estimate energy consumption savings between possible configurations. To this end, the available hardware for the target architecture of the system is modeled, and software to hardware mappings as well as scheduling is represented in the dataflow graph models. State-of-the-art methods to analyze extra-functional properties that indicate real-time performance, i.e., system throughput, are applied to substantiate design decisions w.r.t. timing behavior. Furthermore, novel methods are presented and applied that extend existing timing analysis towards extra-functional properties that are indicators for energy consumption of the wireless sensor devices, i.e., processor utilization and communication data rates, and finally energy consumption itself.

In the course of this chapter, the developed techniques are applied and tested on examples which cover the recognition software up to the feature extraction on the software side (cf. Figure 3.1 on page 30) and the wireless network of sensor nodes and a data aggregating device, i.e., a smartphone, on the hardware side. The reduced scope has two reasons: a) the proposed techniques can be shown in an appropriate level of detail and b) the real-time performance of the wireless sensor network is mainly affected by processing stages up to the feature extraction, as in most scenarios it is the last processing step that can be performed on individual sensor signals or signal dimensions. For classification or a possible dimensionality reduction, feature information from multiple sensors need to be aggregated, potentially on a device that offers substantial computational resources. Therefore, the latter is excluded from the scope of this chapter. Nevertheless, possible hardware architectures with sophisticated computational resources for classification will be covered in Chapter 4 and Chapter 5.

The chapter is structured as follows. In Section 3.1, related work is presented and its relation to the proposed methods in the thesis at hand is described. Thereafter, a case study will be introduced in Section 3.2 on which the proposed methods are explained during the course of this chapter. In Section 3.3, the modeling approach of human activity and gesture recognition is explained. Corresponding analysis methods proposed by the thesis at hand are introduced in Section 3.4 together with a discussion of their limitations. In Section 3.5, experiments and the acquired results are presented, with which the proposed model-based energy consumption estimation is evaluated. Finally, the experimental results and conclusions are discussed in Section 3.6.

3.1 Related Work

The proposed methods for system-level design and analysis of energy-efficient wireless sensor nodes in activity recognition systems are related to existing work from the areas of dataflow graphs and human activity recognition. Their respective relation to the presented approaches in the thesis at hand is described in the following.

Dataflow graphs Dataflow graphs are a quasi-standard for the model-based design and analysis of streaming applications on multi-processor architectures. In particular, SDF Graphs, CSDF graphs, Finite State Machine (FSM) SADF, and Markov chain SADF graphs play a particular role, due to their trade-off in expressiveness and analyzability. Many research has been undertaken towards formal analysis of their timing behavior, e.g., throughput analysis [8, 12, 22, 34, 35], buffer-sizing [15, 36–38], and latency analysis and optimization [39–44]. However, design-time estimation of energy consumption from system-level dataflow graph models has not attained much attention yet in the literature.

Regarding analysis of energy consumption from system-level models based on synchronous dataflow graphs, the approach in [45] and [46] is related to the methods proposed in the thesis at hand. Das et al. study energy-aware task-mapping and scheduling optimization techniques for heterogeneous MPSoCs in [45] and [46]. Although they focus on the joint optimization of throughput, computation energy, and communication energy for reactive fault tolerance of MPSoCs, their computation and communication energy estimation approach from SDF graphs is highly related to the methods presented in the thesis at hand. In [45] and [46], absolute dynamic energy consumption per graph iteration is calculated, however, neglecting the iteration period in comparisons between configurations, and thus the frequency which an iteration is executed with, i.e., throughput. Furthermore, overlapping of iteration periods is not considered as well, which can vary for different mappings and schedules. In contrast, in the thesis at hand, an average energy consumption *rate* is estimated, rather than an absolute energy consumption. This takes the throughput of the graph into account. Furthermore, energy consumption rate estimates are decoupled from graph analysis by an intermediate step, calculating the average processor utilization. Average processor utilization as acquired from the graph representation is defined as the ratio of active time of the processor core to active time and idle time together (iteration period). As a second step, the acquired processor utilization is then used to estimate an average energy consumption rate from a corresponding energy consumption model, that captures the respective relationship. As a result, technological details about the processor core, e.g., clock-gating or deep sleep modes, are abstracted into the energy consumption models.

In addition to the technological abstraction, energy consumption models can be acquired by various approaches, e.g., mathematically, as presented in [45] and [46], by measurements, as carried out in the thesis at hand, or by other potential modeling approaches [47, 48]. The aforementioned distinction from methods described in [45] and [46] applies for the energy consumption rate estimation of communication channels accordingly. Finally, the methods for estimating energy consumption in [45] and [46] are applied to SDF graphs, while in the thesis at hand, the approach is extended and applied to the more expressive CSDF and SADF MoCs.

Energy-efficient human gesture and activity recognition The energy-efficient design of gesture and activity recognition systems is crucial, as energy consumption directly affects battery lifetimes and therefore, the applicability of wireless sensor-based recognition systems. As a result, energy-efficient solutions are prominently represented in the state-of-the-art literature and are ongoing research.

A variety of energy-efficient approaches for sensor-based online activity and gesture recognition systems can be found in the literature. In [49], [50], and [51], sensors are dynamically turned off in situations for which certain sensor modalities do not contribute to the overall recognition accuracy. Due to the dynamic deactivation of negligible sensors, a considerable amount of energy can be saved, prolonging the battery usage times. Saving energy at a conceptual level can be achieved by feature selection strategies, to only spend computation time and thus, energy on the calculation of features that contribute to the overall recognition accuracy [52, 53]. Additional to appropriate classifier selection, Anguita et al. have studied the energy savings from adapted classification algorithms using fixed-point arithmetic [54].

While the aforementioned approaches focus on different hardware and software aspects that can be optimized w.r.t. energy consumption, other approaches concentrate on the mapping of software to the available hardware [55–59]. By performing the feature extraction stage on wireless sensor nodes, feature values are transmitted to a data aggregating device that performs the final classification or stored in flash memory for offline evaluation. On the one hand, feature extraction adds computational load to the processing unit of the sensor node and thus, increases its energy consumption. On the other hand, the amount of data communicated via wireless transceivers or stored in flash memory is reduced and thus, decreases their energy consumption. As a result, the total device energy consumption is a trade-off between processor utilization and data reduction, and can possibly be reduced by on-sensor feature extraction.

In [60], a classification is introduced that structures different energy efficiency techniques. A comparison between possible architectures indicates as well that the mapping of software onto the available hardware is a crucial aspect, as trade-offs between energy consumption of processing units and communication channels are a result. However, although aforementioned literature reports possible energy savings by particular software to hardware mappings, generalized design decisions cannot be concluded, due to different energy consumption characteristics of the available hardware components and the high diversity in software designs for different application scenarios.

In the field of Wireless Sensor Networks (WSN), some studies that are concerned with device energy consumption are related to the methods in the thesis at hand. In [61] and [62], design space exploration approaches for energy-efficient task mapping and scheduling for latency-constrained applications, and latency minimal task mapping and scheduling for energy-constrained applications in WSN are introduced. Their task-mapping and scheduling is based on directed acyclic graphs, which correspond to acyclic HSDF graph representations. Their real-time criterion is based on latency. In contrast, in the thesis at hand, HAR systems are modeled with more expressive dataflow MoCs, i.e., CSDF and SADF. Furthermore, the models in the thesis at hand have cyclic dependencies, which is crucial for the presented real-time criterion based on graph throughput. Furthermore, the presented methods to acquire device energy consumption from dataflow graphs are relying on the throughput and express energy consumption in terms of a rate. In [61] and [62], throughput is not considered in their approach, and total energy consumption for a single execution of the models is calculated, without considering an execution rate. However, in [61] and [62], energy consumption of WSN nodes is mathematically acquired, similarly as in [45], but by considering static power dissipation as well. As a result, the approaches described in [61] and [62] and the presented approaches in the thesis at hand, can be considered as complementary. The energy consumption models presented in the this thesis could be well substituted by more accurate analytical methods as described in [61] and [62]. In turn, the dataflow-based modeling and analysis approaches in the thesis at hand could extend the methods from [61] and [62] by relying on more expressive MoCs and their corresponding analysis approaches that allow for cyclic dependencies, throughput analysis, and thus energy consumption rate estimations.

In [63] and [64], a design methodology for WSN applications based on Kahn Process Networks (KPN) is introduced. Kahn process networks are MoCs that can model dynamic behavior and are more expressive than SDF, CSDF, and SADF [5]. While the authors of [63] and [64] mention, that their KPN-based approach is deterministic and ensures deadlock freedom, very little information on the actual analysis is provided. Furthermore, their approach is focused in synthesizing communication protocols within the WSN, instead of analyzing extra-functional properties at design time. The modeling and analysis approaches described in the thesis at hands are not generally applicable to WSN systems, as the latter are usually control-flow-oriented systems, whose structure can change during run time. Furthermore, sensor sampling is not generally performed at constant rates in WSN and can occur sporadic. The software of HAR systems is, in contrast, mainly dataflow-oriented with little control flow, and sensor sampling is typically performed with predefined sampling frequencies.

In order to substantiate design decisions for sensor-based HAR systems early in the design process, system-level modeling and analysis methods are proposed in the thesis at hand, from which timing behavior in terms of system throughput and latency as well es energy consumption can be estimated at design time. The proposed approach is based on well studied models of computation and their respective timing analysis techniques, which have become a quasi-standard for the design of signal processing and streaming applications on homogeneous and heterogeneous multi-processor on chip architectures. Existing timing analysis techniques have been utilized to acquire energy consumption estimations at design time from model-based representations of the software and hardware at a system level.

3.2 Case Study

The presented modeling and analysis approach will be explained alongside a case study. In [65], an activity recognition and action reconstruction system is designed as an experimental study for the applicability of computational state space models for inference in realistic state spaces. The system was designed to infer the sequential states of a meal preparation routine in a kitchen environment from sensor data. To this end, protagonists were instrumented with wearable IMU sensors. Although the study in [65] focuses on the classifier part of the system, sensor configurations and feature extraction setup represents a viable candidate configuration for activity recognition systems that can be implemented with wireless sensor nodes. Moreover, the chosen study is a representative of today's and future recognition systems that involve many different processing stages in order to successfully infer activities in real life scenarios. However, the focus in [65] does not lie on designing an online recognition system, but rather on the applicability of a particular classification method.

The modeling and analysis approach presented in this chapter is applied to such recognition systems, which have yet been evaluated functionally, i.e., sensor modalities, sampling frequencies, segmentation methods, features, dimensionality reduction techniques, and others, that offer best performance in terms of recognition accuracy. These systems are possibly implemented in application-specific programming languages as offline systems, in order to optimize them for recognition accuracy. In order to develop an applicable online system, a target hardware has to be chosen, the recognition algorithms have to be mapped to the hardware and finally implemented on it. However, implementing different mappings for comparisons regarding extra-functional properties is time consuming and possibly has to be performed in multiple iterations until requirements are met, if at all.

A model-based design approach for software, hardware, corresponding possible mappings, and scheduling representations enables formal analysis of extra-functional properties from these models to substantiate decisions early in the design process. As this chapter focuses on real-time performance and energy consumption in sensor networks as an essential part of online activity and gesture recognition systems, the part of the recognition software from [65] that can potentially be executed within the wireless sensor network, is modeled and analyzed with the presented methods as a case study.

Target software The functional parameters of the recognition system under design are depicted in Figure 3.1 in form of an ARC. The first ARC stage represents the sensor data acquisition process, in which 3D sensor data is sampled from each an accelerometer and a gyroscope at $100 Hz$ of in total 5 sensor devices attached to the body of a human protagonist. Note that in [65] the sensor data is actually sampled at $120 Hz$. However, the available sensor hardware that has been chosen to evaluate the presented approaches in experiments only offers sampling frequencies that are power of two multiples of $12.5 Hz$. Therefore, the nearest sampling frequency of $100 Hz$ has been chosen. For the pre-processing stage of the sensor data, no specific techniques have been reported in [65]. Here, the proprietary correction algorithms, which are already implemented in the target hardware sensors, are used. In the segmentation stage, a sliding window with a size of 128 samples and an overlap of $75\,\%$ is used to segment the continuous stream of sensor data into windows, with a resulting window rate of $3.125 Hz$. In the feature extraction stage, for each window (on all 6 dimensions separately), the mean, variance, skewness, and kurtosis, as well as its dominant frequency (referred to as *peak* in [65]) and corresponding magnitude (referred to as *energy* in [65]) are calculated as features for each sensor device. The result of each window is thus a 36-dimensional feature vector per sensor device, which in total results in a stream of 180-dimensional feature vectors at $3.125 Hz$ which are subject

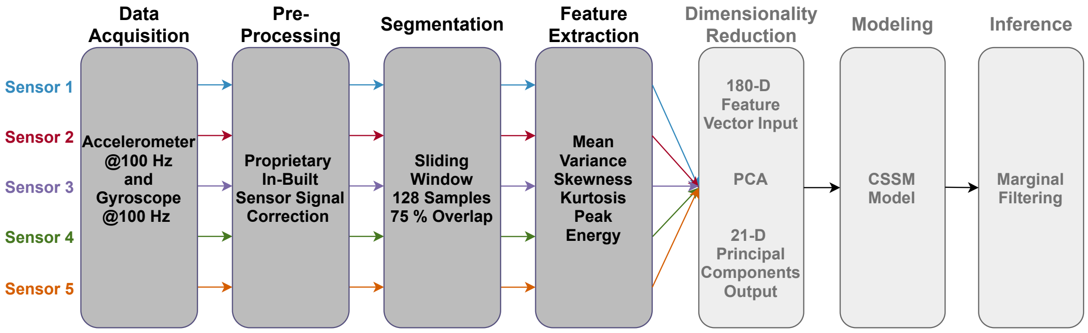

Data Acquisition
Pre-Processing
Segmentation
Feature Extraction
Dimensionality Reduction
Modeling
Inference
Sensor 1
Sensor 2
Sensor 3
Sensor 4
Sensor 5
Accelerometer @100 Hz and Gyroscope @100 Hz
Proprietary In-Built Sensor Signal Correction
Sliding Window 128 Samples 75 % Overlap
Mean Variance Skewness Kurtosis Peak Energy
180-D Feature Vector Input
PCA
21-D Principal Components Output
CSSM Model
Marginal Filtering

Figure 3.1: Example activity recognition chain based on [65].

to a dimensionality reduction in form of a principal component analysis. In [65] different number of principal components are used in the experimental part. However, since the principal components calculation is performed on entire 180-dimensional feature vectors, the feature extraction stage is the last stage in the ARC that can possibly be executed on the hardware of the sensor devices within the sensor network, as until there, every signal dimension can be processed individually. The computation of the principal components has to be performed on a device that has access to the feature vectors of all 5 sensor devices together for its computation. Therefore, it is considered to be performed on a data aggregating device in the study at hand. The dimensionality reduction as well as following ARC stages, which are concerned with the actual classification task do therefore not affect the sensor network and are thus excluded from the modeling and analysis part in this chapter. For sake of completeness, these stages are depicted in Figure 3.1 in gray.

Target hardware As a target hardware, an on-body network of wireless sensor nodes from Bosch Sensortec GmbH has been set up. Each sensor node, referred to as *DIANA board* is equipped with a BHI160 ultra low power sensor-hub [2] housing a BMI160 Inertial Measurement Unit (IMU) with a triaxial accelerometer and gyroscope as well as support for external sensors that can be connected via an external sensor Inter-Integrated Circuit (I^2C) interface. Additionally, the BHI160 integrates a 32-bit floating-point domain-specific microcontroller, referred to as *Fuser Core*. This is used to offer additional sensor correction and fusion support, i.e., the computation of orientation quaternions or step detection algorithms. Sensor fusion on the Fuser Core also integrates sensors connected via the external sensor interface. A triaxial magnetometer BMM150 and an environmental sensor BME280 sensing relative humidity, barometric pressure, and ambient temperature, are connected via the external sensor interface but are not considered to be used in the system under design. The BHI160 is connected via I^2C to a DIALOG DA14583 microcontroller with an integrated BLE stack and radio for data transmission. The DIALOG controller is used to send sensor sample packets received from the BHI160 to a data aggregating device. A Samsung Galaxy S5 smartphone has been chosen, which collects data from all 5 wireless sensors and either pre-processes it on board or sends it to a possibly computationally more powerful computing architecture. An overview of the main components of the wireless sensor nodes is given in Figure 3.2.

In the study at hand, only the BMI160 is considered as data acquisition device to sample accelerometer and gyroscope data, and the Fuser Core for early sensor signal processing. The DA14583 BLE controller is entirely reserved for establishing the BLE connection and transmission of sensor samples to the smartphone.

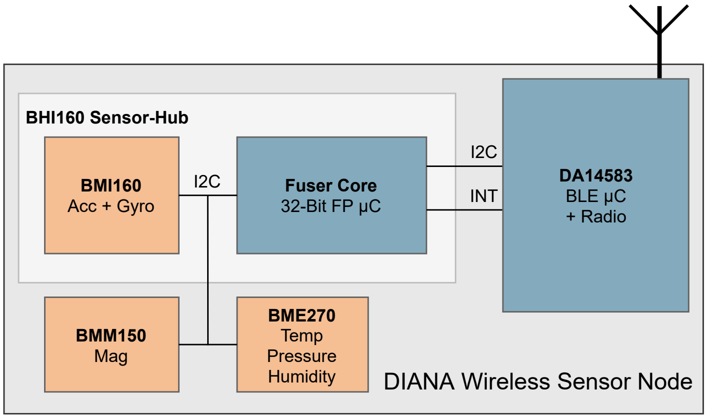


Figure 3.2: Wireless sensor node *DIANA board*.

3.3 Modeling

In this section, it will be shown how online activity recognition systems can be modeled with (C)SDF semantics at different levels of abstraction. For the sake of comprehensibility, a simple example ARC setup is modeled with SDF graphs first. The final model for the selected case study will then be presented at the end of this section.

Due to its dataflow-oriented individual stages, the ARC can be directly represented as an SDF graph. In Figure 3.3, an example SDF graph is shown, that represents a simple ARC setup, that is explained in the following. In the graph, each token represents a 3D floating-point sample (either raw, pre-processed, or as feature vector entry). To this end, the data acquisition stage (actor DA) produces two tokens in each firing, representing a 3D accelerometer and a 3D gyroscope sample. The execution time of DA is $5,000\,\mu s$, corresponding to a $200\,Hz$ sampling frequency. A self-edge c_0 with a single initial token restricts auto-concurrency of DA, i.e., DA can only fire again as soon as its previous firing is finished. The pre-processing stage, calculating the 3D sensor orientation from the accelerometer and gyroscope sample, is modeled by actor PP, consuming the two sensor sample tokens from c_1 and producing a single token on c_2, representing the 3D orientation sensor sample. In the segmentation stage, a sliding window of length 4 without window overlap is implemented, represented by actor SW. As soon as four samples have been aggregated on c_2, actor SW can fire, producing four tokens at once onto c_3 for the feature extraction. The feature extraction stage calculates the mean, variance, and the dominant frequency based on an FFT for each sensor axis, producing in total three 3D floating-point features per window. Actor FE represents the feature extraction, consuming four tokens from c_3, performing the calculations, and producing the three tokens onto channel c_4.

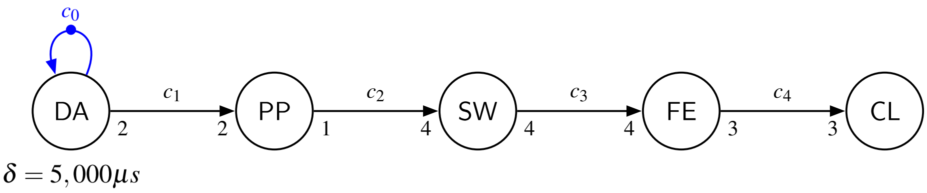


Figure 3.3: SDF graph of an example ARC.

Finally, for each produced feature vector, i.e., three tokens, actor CL consumes all of them and performs the classification routine, e.g., a k-NN algorithm. The output of CL is not explicitly modeled. The repetition vector of the SDF graph shown in Figure 3.3 is $\gamma = [4,4,1,1,1]^T$.

Although the example ARC translates intuitively into SDF semantics, in few systems, the segmentation stage will actually be implemented explicitly, but rather in combination with the feature extraction stage as a function combining both stages by filling a buffer for the sliding window and a counter. Whenever the counter indicates a complete sliding window, the feature extraction can be directly performed on the buffer. Furthermore, not all features need the entire data of a sliding window, but can rather be updated incrementally with each new sensor sample in the buffer. Lastly, a window overlap is often desired when deploying a sliding window segmentation in activity recognition systems, in order to capture important gesture or activity transitions, start, or end points. In order to model the aforementioned concepts in a level of abstraction that accounts for the desired system behavior, represents exact data communication, and avoids explicitly modeling implied schedules in order to reduce model complexity, a model refinement is shown in the following section.

3.3.1 Model Refinement

Combined segmentation and feature extraction In order to combine the sliding window segmentation and feature extraction, actor SW can simply be removed from the model in Figure 3.3. The snippet of the resulting model from the pre-processing stage PP to the classification stage CL is shown in Figure 3.4(a). The actor FE is fired every four samples (after a new window is filled) and extracts three features that are produced on its outgoing edge. Hence, the buffering of samples within a sliding window takes place on the edge between PP and FE implicitly. However, this step does not allow modeling the buffering of each new sample with its execution time, which will be addressed at the end of this section.

To this end, the following example execution times are assumed. The execution time of the feature extraction from a single window is assumed to be in total $108\,\mu s$. This represents the accumulated example execution times of an FFT of $100\,\mu s$, and the calculation of mean and variance with incremental steps of together $1\,\mu s$ per sample and a final calculation of $4\,\mu s$ for mean and variance together. The reasoning behind this apportionment will get obvious during the refinement steps in the following.

Sliding window overlap If a sliding window with an overlap is desired, there are actually multiple sliding windows in parallel with a relative displacement defined by the overlap $O_{SW} \in [0,1)$. Note that the overlap is defined as a ratio w.r.t. the window length. To model overlapping sliding windows, as many actors as parallel processed windows ($\lceil \frac{1}{1-O_{SW}} \rceil$) are included, with additional initial tokens to interleave the sliding windows with a displacement of $size_{SW} \cdot O_{SW}$ samples. See Figure 3.4(b) for an example of the sliding window with $size_{SW} = 4$ and $O_{SW} = 0.5$. However, modeling the overlapping sliding windows in SDF semantics requires initial tokens on the output edges and production rates of the FE actors twice as high, in order to synchronize the firing of actor CL. This results in a token production of the actors FE and a token consumption of actor CL that is actually higher than it would be implemented in the final system. To overcome this, CSDF semantics can be used to cycle the input rates of CL, matching the actual token consumption rate and reducing the token production rate of the actors FE to its actual amount of calculated features. The principle is shown in Figure 3.4(c).

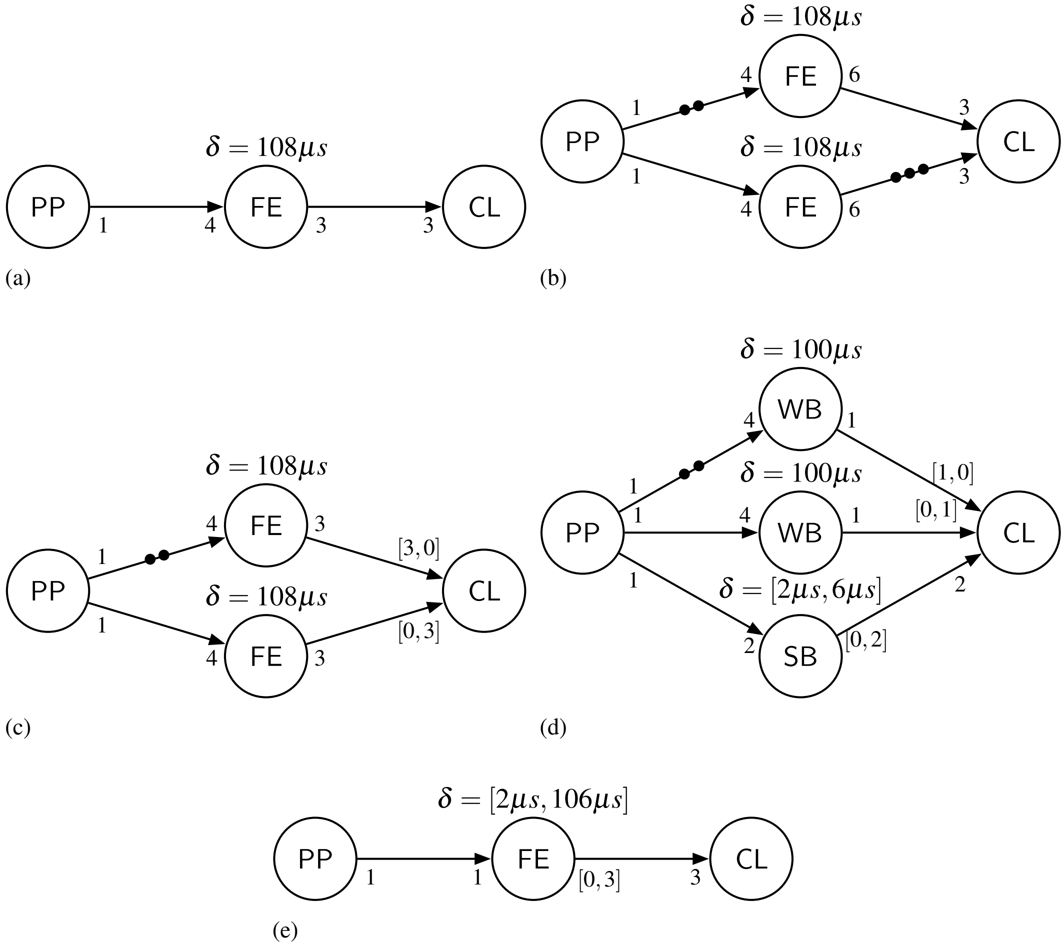


Figure 3.4: Modeling of the sliding window-based feature extraction at different levels of abstraction.

Fine grained feature calculation Up to now, the calculation of three features of a whole, completely filled window has been modeled. In the example, the feature extraction is composed of a mean value of all samples in the window, its variance, and the most dominant signal frequency calculated by an FFT. All three features are calculated whenever four samples have been gathered into a new window. However, in order to reduce the computation delay after the fourth gathered sample, some features can be calculated on partial windows, i.e., incrementally updated with each new sample in the window. Examples are one-pass computations of statistical moments of arbitrary order [66, 67] such as mean, variance, skewness, and kurtosis, which are often-used features in activity recognition systems [65, 68, 69]. As an example, for computing the variance, a mean and a mean of squares is updated with each new sample. After updating with the last sample of the window, the actual variance is calculated from the squared mean and the mean of squares in an $O(1)$ step. In such sample-based calculation, the feature extraction is calculating the fraction of an entire window corresponding to each new sensor sample but only produces an output with the last sample filling the window. However, this principle cannot be applied to the FFT calculation, which is performed on the whole window after it is filled. This behavior can also be captured with CSDF actors as shown in Figure 3.4(d). If the updating of mean and mean of squares takes $1\,\mu s$ for each new sample, and the finalizing, after updating with the last sample, takes $4\,\mu s$, the delay function of such an actor would be $[1\,\mu s, 1\,\mu s, 1\,\mu s, 5\,\mu s]$ and the corresponding production rate is $[0,0,0,2]$ for a sliding window of $size_{SW} = 4$. Calculating two windows in parallel because of a window overlap of 50%, the actual delays and production sequences are superpositioned with a displacement of two samples, resulting in a periodical firing of two cycles with a delay of $[2\,\mu s, 6\,\mu s]$ and a production rate of $[0,2]$. This sample-based feature calculation of mean and variance is included in Figure 3.4(d) in actor SB together with the window-based FFT calculation WB (previously FE) producing a single token. Note again, execution times of the aforementioned explanations are examples, annotated for the sake of comprehensibility.

To analyze real-time performance of processing units and communication channels, as well as estimating energy consumption from the CSDF models, it is necessary that the actual processing times of actors and the correct token consumption rates match the implementation in order to extract the processor load and the transmission overhead on hardware communication channels, which will be addressed in Section 3.4. Although refinement of developed analysis methods is possible by considering token sizes specified in the model, which is a feature in the SDF^3 tools [13], tokens should represent unified data sizes in the model, e.g., a 3D single-precision floating-point vector, in order to apply analysis methods as presented in Section 3.4.

Reduction of model complexity On wireless sensor nodes with integrated microcontrollers, feature extraction functions can be implemented in a sample-based fashion. These perform both the update of the sample-based feature calculations with each new sample, and the buffering of samples for the last invocation for window-based calculations, e.g., an FFT. These functions are invoked with each new sample and imply a static order schedule of the different feature calculations. Such implementation can be represented by a single CSDF actor, summarizing the feature extraction calculations in its cycling invocations, which is shown in Figure 3.4(e). Note that a fixed sized segmentation is assumed, to actually result in a cyclo-static behavior in the long run of the system. The delay of actor FE summarizes the added processing times of all firing actors in each cycle, and their token production rate as well. This, on the one hand, simplifies the CSDF graph, but prevents the mapping of different feature calculations to different processing units and implies a certain schedule of actor firings. However, this level of abstraction is continued in the following sections, as it matches with the implementation scheme of the firmware development kit of the target architecture, i.e., the BHI160 sensor sub-system.

The full CSDF application model of the aforementioned example feature extraction, together with example delays for actor PP and actor CL, is shown in Figure 3.5. Note that execution times in Figure 3.5 are examples, annotated for the sake of comprehensibility. In the design flow, execution times will be actually annotated to the analysis model, i.e., after mapping the application to the particular hardware model, which will be described in Section 3.3.3.

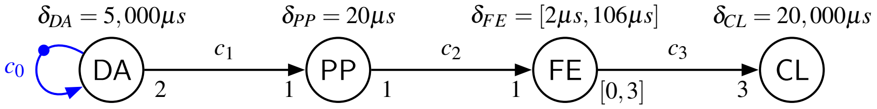

$\delta_{DA} = 5{,}000\mu s$
$\delta_{PP} = 20\mu s$
$\delta_{FE} = [2\mu s, 106\mu s]$
$\delta_{CL} = 20{,}000\mu s$
c_0
DA
c_1
2
1
PP
c_2
1
1
FE
c_3
[0,3]
3
CL

Figure 3.5: Example CSDF application graph.

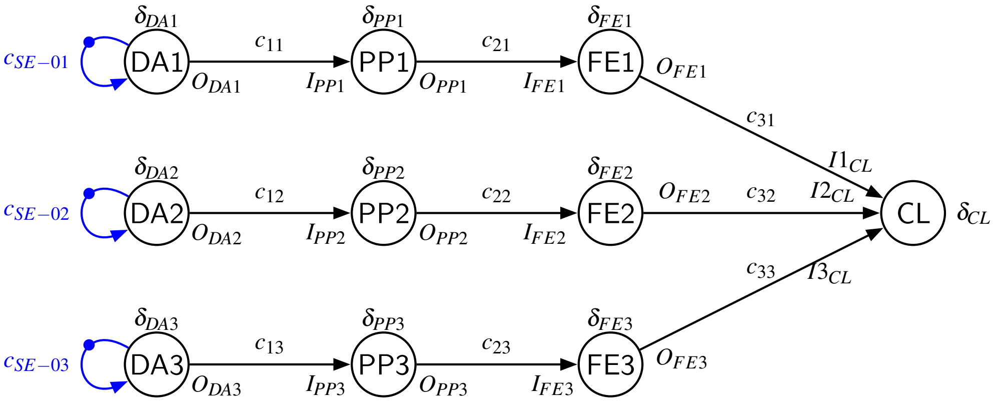


Figure 3.6: Application model of the case study activity recognition chain.

3.3.2 Application Model

Based on the modeling approach in the previous section, the CSDF model of the case study is introduced in the following. Although the system under design is constructed for five sensor nodes, the corresponding models in this chapter will only include three out of these five sensors for the sake of readability. The extension towards five sensors is straightforward. The analyzed model of the on-body sensor network of the experiments as well as the experimental implementation does indeed include all five sensors.

The application model of the case study is shown in Figure 3.6. For the sake of readability, production and consumption rates are captured in parameters, which can be found in Table 3.1. In Table 3.1, the dimensionalities of parameters correspond to the number of phases of CSDF actors. The dimensionality is shown next to the parameter name, with $\mathbb{Z}^n_{0+}$ denoting an n-dimensional non-negative integer parameter vector. Without a dimensionality indication, the number of phases is one, i.e., SDF behavior. Furthermore, if only a single parameter value is shown on the right hand side of Table 3.1, although the parameter's dimensionality is greater than one, the parameter is considered to be equal across all the phases of the particular actor. Finally, a repetition of a parameter value x for subsequent phases is indicated by $\{x\}_{\times n}$, with n denoting the number of repetitions of x, e.g., $[\{0\}_{\times 3}, 1]$ is a shorthand notation for $[0,0,0,1]$.

Table 3.1: Application model graph parameters in all configurations of the case study.

Graph Parameters	Parameter Values
δ_{DAx}	$9\,900\,\mu s$
O_{DAx}	2
I_{PPx}	2
O_{PPx}	2
$I_{FEx} \in \mathbb{Z}^{32}_{0+}$	2
$O_{FEx} \in \mathbb{Z}^{32}_{0+}$	$[\{0\}_{\times 31}, 12]$
Ix_{CL}	12

Corresponding to the ARC of the case study in Figure 3.1, the sensor sampling actors DA1-DA3 have a self-edge with one initial token to avoid auto-concurrency. Although the target sampling frequency is $100\,Hz$, the sampling frequency can deviate by $\pm 1\,\%$ according to the specification, which has to be considered for the input triggered system when worst-case performance is analyzed w.r.t. real-time requirements. As a result, the maximum sensor sampling frequency of $101\,Hz$ is chosen in order to evaluate worst-case behavior. This corresponds to a sampling period of $9,900\,\mu s$, which is annotated as execution time δ_{DAx}. The production rate of actors DAx on $c_{11} - c_{13}$ is two, representing a 3D accelerometer and a 3D gyroscope sample. The pre-processing is captured by actors PP1-PP3, consuming each two tokens and producing two tokens on edges $c_{21} - c_{23}$. The feature extraction is captured in CSDF actors FE1-FE3, each firing in a cycle of 32 phases (75 % of the sliding window size of 128 samples), consuming two pre-processed tokens from $c_{21} - c_{23}$ in each invocation, but producing 12 tokens (six 3D features of the accelerometer signal and six 3D features of the gyroscope signal) every 32 firings. Finally, the 12 features of each sensor are consumed by actor CL, abstracting the last ARC stages of the case study into placeholder actor CL for classification.

Note that execution times are abstracted into parameters as well, as these depend on the particular hardware they are executed on. However, the execution time annotation of sensor sampling processes DA1-DA3 is already given in Table 3.1, as it does not represent a software execution, and does not change for different mappings, unless the system specification is changed to other sampling rates. The execution time parameters of software actors are assigned according to a particular mapping, which will be addressed in the following section.

3.3.3 Analysis Model

In this section, the application model of the case study will be transformed into analysis models, which represent different mapping configurations. As a first step, an abstract model of the target hardware is constructed. This consists of sensors, communication channels, and processor units, that are allocated for executing functions of the recognition software. For the presented target hardware in Section 3.2, the Fuser Core of each sensor node, as well as the smartphone processor are selected as possible candidate processing units for feature extraction. Additionally, the BMI160 sensors are modeled in order to map the sensor sampling actors accordingly. The hardware model of the target architecture is shown in Figure 3.7. The BMI160 sensors are represented by hardware units SE1-SE3, the Fuser Core of each sensor sub-system is represented by units FU1-FU3, and the smartphone Application Processor (AP) is represented by unit AP. Hardware channels $t_{01} - t_{03}$ represent the I^{2}C channel between the

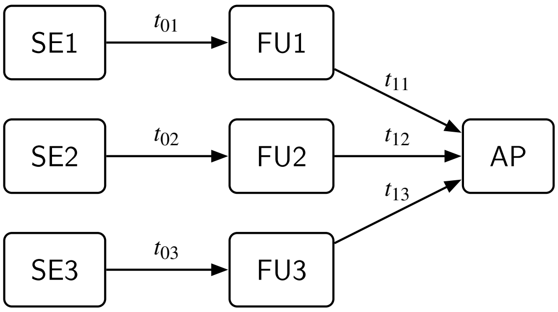


Figure 3.7: Hardware model of the selected case study with three out of five wireless sensor nodes.

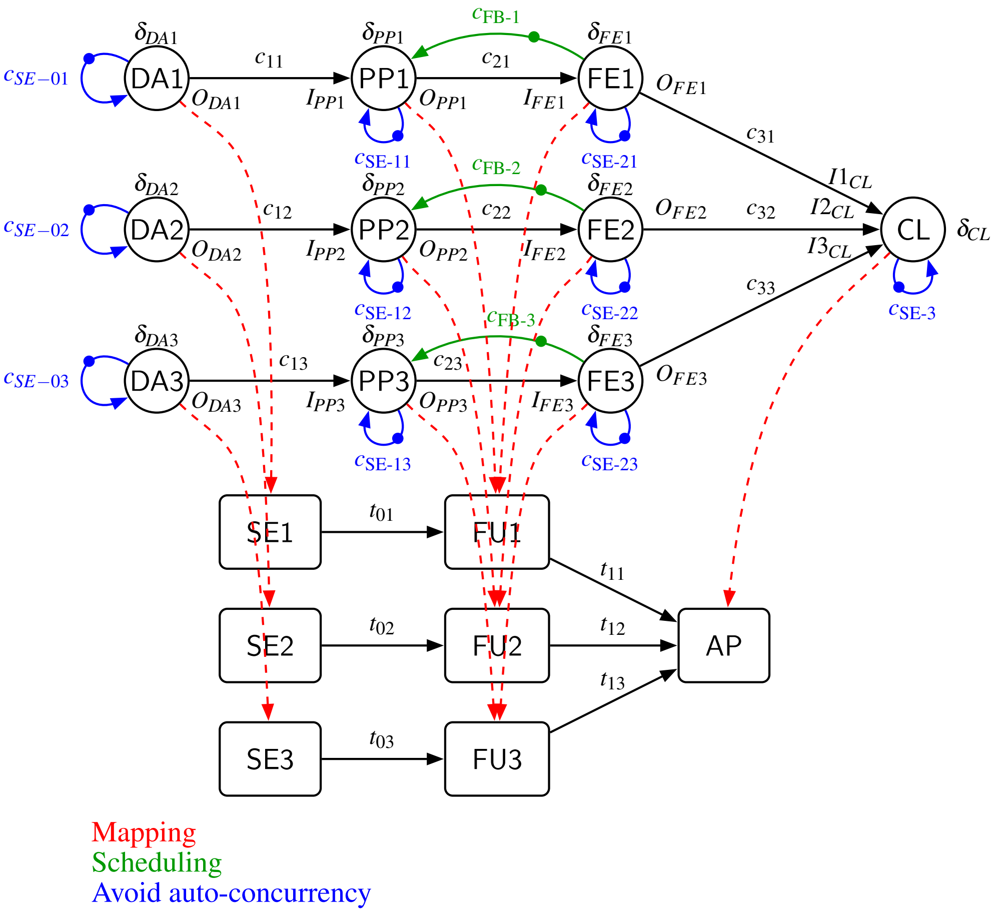


Figure 3.8: CSDF graph in mapping A, where the feature extraction is computed on the Fuser Core.

BMI160 and the Fuser Core, and hardware channels $t_{11} - t_{13}$ represent the BLE connection between BHI160 and Smartphone AP. The BLE controllers are abstracted to communication channels, as their sample-based processing into BLE packets and transmission can be considered constant per communicated sample and scale with the throughput.

In order to avoid multiple instances of a single actor to be executed concurrently on a processor unit (auto-concurrency), to each software actor, a self-edge with a single initial token is attached, before mapping. In the study at hand, two possible mappings are evaluated to asses the affect of on-sensor vs. off-sensor feature extraction w.r.t. real-time performance of the on-body sensor network and the energy consumption of the sensor nodes. The two mappings A and B, representing on-sensor feature extraction and off-sensor feature extraction, are depicted in Figure 3.8 and Figure 3.9, respectively. The corresponding production rates, consumption rates, and initial tokens are listed in Table 3.2.

In mapping A, feature extraction actors FE1-FE3 together with the pre-processing actors PP1-PP3 are mapped to the Fuser Core of each sensor node FU1-FU3. Sensor sampling actors DA1-DA3 are mapped to sensors SE1-SE3. Mappings are indicated by red dashed edges in Figure 3.8. In order to sequence the executions of actors that are mapped to the same processor unit, scheduling edges are added in a way, such that none of these actors can fire concurrently. Initial tokens have to be

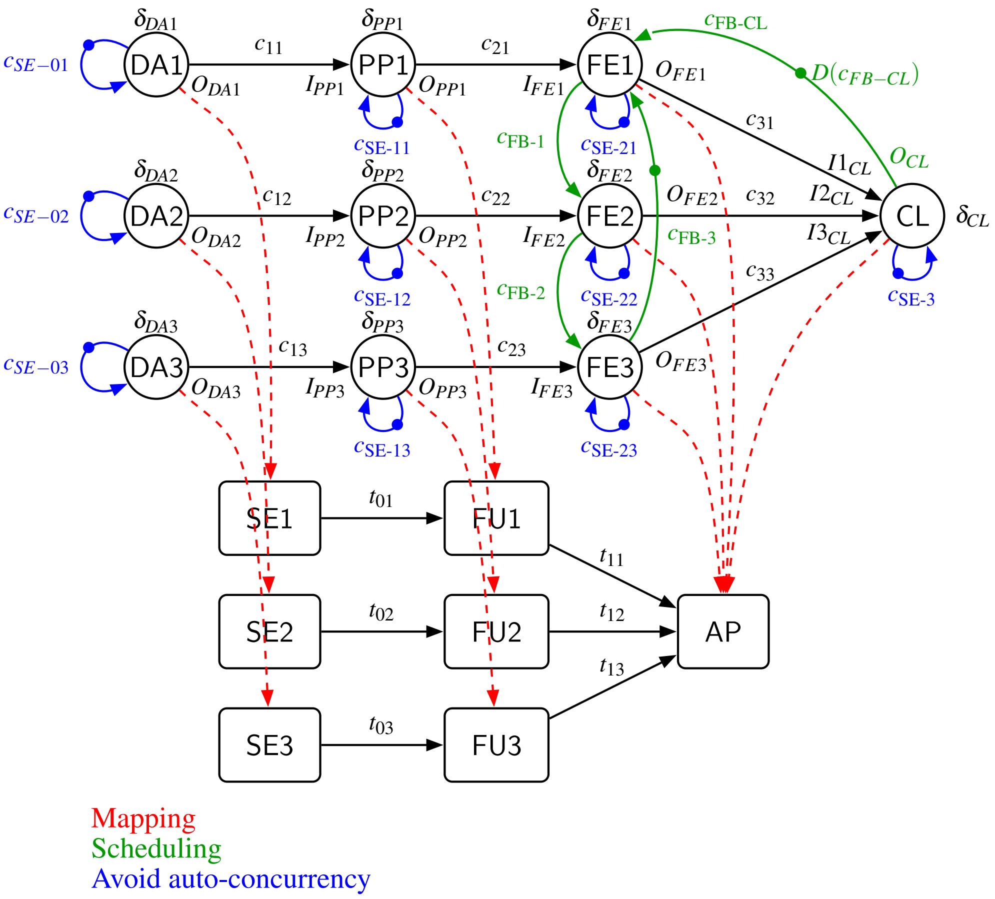


Figure 3.9: CSDF graph in mapping B, where the feature extraction is computed on the smartphone application processor.

Table 3.2: Graph parameters of the analysis models in all configurations of the case study.

Graph Parameters	Parameter Values
$O_{DAx}, I_{PPx}, O_{PPx}, I_{FEx}$	2
$O_{FEx} \in \mathbb{N}^{32}$	$[\{0\}_{\times 31}, 12]$
$Ix_{CL} \in \mathbb{N}^{32}$	$[\{0\}_{\times 31}, 12]$
$O_{CL} \in \mathbb{N}^{32}$	$[\{0\}_{\times 31}, 32]$
$D(c_{FB-CL})$	32
δ_{DAx}	$9\,900\,\mu s$
δ_{PPx}	$1\,250\,\mu s$
$\delta_{FEx} \in \mathbb{N}^{32}$	$[407\,\mu s, \{275\,\mu s\}_{\times 30}, 23\,500\,\mu s]$
$\delta_{CL} \in \mathbb{N}^{32}$	$[\{0\,\mu s\}_{\times 31}, 1\,000\,\mu s]$

annotated in order to provide the correct start condition. The schedule of mapping A was modeled, such that actors PP1-PP3 execute first on each sensor, before feature extraction (FE1-FE3) takes place. Scheduling edges are shown in green in Figure 3.8.

Methods to model different scheduling schemes is out of scope for the thesis at hand. However, for the sake of completeness, the interested reader is referred to [30–33, 70, 71] for relevant literature on the scheduling of SDF, CSDF, and dataflow graphs in general.

In mapping B, the feature extraction actors FE1-FE3 together with the classification placeholder CL are mapped to AP, representing the smartphone application processor. Pre-processing actors PP1-PP3 are mapped to the Fuser Core of each sensor node, as they are part of the Fuser Core firmware. In order to sequence the executions of FE1-FE3 and CL on the smartphone processor AP, scheduling edges have been added with corresponding initial tokens, depicted in green in Figure 3.9.

For each mapping option, execution times of software actors have been annotated with their Worst-Case Execution Times (WCETs) on the corresponding processing unit. Note that in general WCETs are acquired by analysis, e.g., on assembler code, and represent a guaranteed worst-case bound on a specific hardware. However, in the study at hand, WCETs of actors have been measured on the Fuser Core, which merely results in an *observed* WCET, and not necessarily represents a guaranteed worst-case bound. However, these observed WCETs have been annotated to the model and will be referred to as WCETs in the following. Furthermore, the performance of the wireless on-body network has only been analyzed w.r.t. the wireless sensor nodes. The different mappings represent the difference of BLE transmission throughput and processor utilization of the sensor nodes between on-sensor and off-sensor computation of the feature extraction stage. In the latter case, the features are neither calculated on the Fuser Core nor on the smartphone, as the smartphone AP computations does not influence the sensor nodes. For the sake of simplicity, the same execution times on the Fuser Core have been annotated to the actors mapped to the smartphone.

In terms of real-time performance of the sensor network, both mappings represent different throughputs on the BLE transceivers of the sensor nodes, as well as the processor utilization of the Fuser Core, which are both constrained. Furthermore, both mappings represent a trade-off between processor utilization of the Fuser Core and BLE transmission throughput, which both affect the total energy consumption of the sensor nodes. Based on the two mappings, corresponding analysis methods will be introduced in the following section, which will also be evaluated in experimental studies.

In general, the deployed model is rather complex w.r.t. the number of mapping choices that are evaluated, which might raise the question of its applicability, especially given that actor WCETs have to be analyzed for each hardware unit individually. By having N actors and M different hardware units, in total $N \cdot M$ measurements or WCET analyses, respectively, have to be performed and annotated. However, with these annotations, M^N different mapping options can be analyzed formally. As a result, the number of WCET analyses increases linearly with the number of actors or additional hardware units, while the number of possible mappings that can be analyzed with it increases exponentially with the number of actors N and polynomial with the number of processing units M. Given the increasing complexity of software and hardware of today's and future activity recognition systems, and the increased number of possible mappings that can be analyzed at design time, the number of WCET analyses to be performed is worth the comparably low effort. Note that the additional number of possible configurations that can be analyzed, based on graph parameter changes, e.g., sensor sampling rates, window overlaps, or schedules, are not contained in the aforementioned approximation yet.

In the course of this chapter, a particular mapping and scheduling of an application model to the target hardware will be referred to as *system configuration*. In contrast, a particular set of ARC parameters (e.g., sliding window size, overlap, or number and type of extracted features) will be referred to as *application configuration*. Both system configuration and application configuration are together referred to as *configuration* during the course of this chapter. Each configuration affects the utilization of processor units as well as the communication data rates on hardware communication channels. This further influences the energy consumption of single components or a group of components together, e.g., all hardware units that belong to the wireless sensor nodes. However, a configuration can be fully captured in the presented analysis models. It can then be analyzed and compared to different configurations, according to whether a change in system configuration, application configuration or both is desirable. The developed analysis methods that can be applied to the presented analysis models will be introduced in the following section.

3.4 Analysis

The analysis methods presented in the thesis at hand are capable of estimating worst-case properties, i.e., processor utilization, communication throughput, and finally energy consumption. However, the presented methods only produce meaningful worst-case estimates for configurations that meet a certain real-time constraint, i.e., system throughput, which will be explained in the following section.

In the design of activity recognition systems, other extra-functional properties such as latency, can be considered as real-time indicators as well. Although analysis methods for latency in synchronous dataflow graphs exist [39], it is not considered as a real-time requirement in this chapter, but will be subject in Chapter 4 and Chapter 5. The following section describes necessary real-time analysis methods, before energy consumption indicators are explained in Section 3.4.2.

3.4.1 Real-Time Behavior

In order to guarantee real-time performance of gesture and activity recognition systems, the system throughput is of major concern. In the thesis at hand, system throughput refers to the amount of data that the system can process per time unit. Thus, the throughput states the data rate limit, which the system can process without causing data congestion or loosing data from the sensors at the inputs.

In terms of dataflow semantics, the throughput is defined as an event rate, that corresponds to the average number of graph iterations per time duration in the steady state of the graph execution [8]. However, both definitions correspond to each other, as in (C)SDF graphs, the number of tokens produced by the sensors (actors DAx in Figure 3.8 and Figure 3.9) within each graph iteration is constant. Therefore, graph throughput can be considered as a measure of system throughput.

In the thesis at hand, the sensor data input and its data throughput is explicitly modeled into the application and analysis models. The real-time behavior of the system is thus verified, if the processing of data is not slower than the input data process. In terms of dataflow semantics, the throughput of the input data process $TH(\mathsf{DA})$, which has a constant behavior, is exactly the reciprocal of its annotated execution time (sampling period). In case the processing of data performs at a smaller throughput than necessary, i.e., the analysis model is not self-timed bounded [10], a corresponding decreased throughput of the input data process $TH(\mathsf{DA})$ indicates that the analysis model represents a configuration that is not performing in real time. In other words, the corresponding model cannot be implemented with a self-timed schedule, without the congestion of data at the input.

Note that throughput constraints of communication channels can limit the system throughput in a likewise manner. However, this presumes, that throughput constraints are implicitly modeled into the communication channels of the analysis models. This however, has not been intended in the current chapter, in favor of an explicit analysis. The latter allows to infer energy consumption indicators, which will be explained with the introduction of communication data rate on page 42. However, the implicit modeling of constrained communication throughput will be subject in Chapter 4 and Chapter 5.1.

Processor performance Due to the mapping and scheduling, the processor core performance is implicitly modeled into the analysis models. By checking the actor throughput of the input data process against its execution time annotation, the real-time performance of the processor cores can be verified.

In addition to implicit processor performance, the utilization of each processor core can be analyzed explicitly, with certain restrictions. In the thesis at hand, the processor utilization is defined as the fraction of active time of a processor within a given period, i.e., software is actively executed, within an observation period, divided by that period. In terms of CSDF semantics, an intuitive observation period is the average iteration period. As a result, the processor utilization $\widehat{U}(p)$ is an accumulation of all execution times of mapped actors $v \in V'$ w.r.t. each executed phase multiplied by their repetition vector entries and the graph throughput $TH(G)$. The set of actors that are mapped to the processor under analysis $p \in P$, is denoted by V', i.e., $V' = \{v \in V \mid \exists (v,p) \in M_V\}$. In the scope of the analyses, processor utilization denotes the average processor utilization across an average iteration period, which equals its long-run average processor utilization for SDF and CSDF graphs. Formally, the processor utilization can be calculated by:

$$\widehat{U}(p) = TH(G) \cdot \sum_{v \in V'} \frac{\gamma_G(v)}{\phi(v)} \sum_{i=1}^{\phi(v)} \delta(v,i), \tag{5}$$

with $TH(G)$ being the graph throughput, V' the set of mapped actors to processor core $p \in P$, $\phi(v)$ the number of CSDF phases of each actor $v \in V'$, $\gamma_G(v)$ its repetition vector entry, and $\delta(v,i)$ the execution time of its i-th phase. The notation $\widehat{U}$ indicates a model-based estimate.

Let $m_P(v,p)$ indicate if an actor $v \in V$ is mapped to a particular processor $p \in P$, with:

$$m_P(v,p) = \begin{cases} 1 & \text{if } (v,p) \in M_V, \\ 0 & \text{otherwise.} \end{cases}$$

Due to consistency, and the relationship between graph throughput $TH(G)$, actor throughput $TH(v)$, and repetition vector $\gamma_G(v)$, (cf. Equation (1)), the average processor utilization $\widehat{U}(v,p)$ induced by a particular actor $v \in V$ on processor $p \in P$, can be calculated by:

$$\widehat{U}(v,p) = m_P(v,p) \cdot \frac{TH(v)}{\phi(v)} \sum_{i=1}^{\phi(v)} \delta(v,i). \tag{6}$$

Note that processor utilization calculated by Equation (5) and (6) only represents a useful estimate, when none of the actors representing software executions lie on the critical path regarding graph throughput. In other words, the critical path regarding throughput must lie on the actor or actors representing a constant input data rate, which thus has to be modeled as well. Otherwise, the iteration period of the graph does only represent the worst case and can be shorter in a real application. This leads to a variability depending on execution times within the nominator as well as the denominator of the calculated processor utilization, which cannot be interpreted in a meaningful way. Therefore, the aforementioned real-time analysis w.r.t. the actor throughput $TH(\mathsf{DA})$ is a prerequisite for estimating the processor utilization. Furthermore, in order to receive worst-case estimates regarding processor utilization and its induced energy consumption, the modeled input data process has to be annotated with execution times, corresponding to the highest input data throughput that can be possibly expected in the final application. In turn, the software actors have to be annotated with their WCET, in order to get worst-case estimates on the processor utilization.

As Equation (5) iterates over all phases of all mapped actors, its computation time scales in the worst case with the accumulative number of phases $\phi(v)$ of all actors $v \in V$, as possibly all actors can be mapped to a processor core p. Its computational complexity is thus $O(|V| * \overline{\phi})$, with $\overline{\phi}$, denoting the average number of phases of all actors $v \in V$.

Note that the computational complexity applies to the calculation of processor utilization of a single processor p, but also for the calculation of all processor core utilizations in the system, as each actor can only be mapped once.

Communication data rate In order to analyze the data rate on communication channels, an explicit analysis method is presented, as it forms the basis for analyzing energy consumption indicators as introduced by the thesis at hand. However, implicit modeling and analysis is possible as well, which will be applied in Chapter 4.

In the thesis at hand, data rate $\widehat{r}(t)$ on a hardware communication channel $t \in T$ is defined as the amount of data that is communicated per time unit. In terms of dataflow semantics, this corresponds to the total number of tokens that is consumed on all edges $e \in E'$ per iteration. Here, E' denotes the set of edges that are mapped to the hardware channel $t \in T$, that is, $E' = \{e \in E \mid \exists (e,t) \in M_E\}$.

Formally, the average data rate $\widehat{r}(t)$ on hardware channel t is calculated for mapped and scheduled CSDF graphs by:

$$\widehat{r}(t) = TH(G) \sum_{(v,v') \in E'} \frac{\gamma_G(v')}{\phi(v')} \sum_{i=1}^{\phi(v')} cons(e,i). \tag{7}$$

Let $m_T(e,t)$ indicate if a channel $e \in E$ is mapped to a particular hardware channel $t \in T$, with:

$$m_T(e,t) = \begin{cases} 1 & \text{if } (e,t) \in M_E, \\ 0 & \text{otherwise.} \end{cases}$$

Similar to processor utilization, the average data rate $\widehat{r}(e,t)$ induced by a particular edge $e = (v,v') \in E$ on a hardware communication channel $t \in T$ can be calculated from the destination actor v, by:

$$\widehat{r}(e,t) = m_T(e,t) \cdot \frac{TH(v')}{\phi(v')} \sum_{i=1}^{\phi(v')} cons(e,i), e = (v,v'). \tag{8}$$

Similar to processor utilization, the data rate is calculated over an average iteration period, which represents the long-run expectation for SDF and CSDF graphs. Furthermore, similar to worst-case average processor utilization, the average data rate calculated by Equation (7) and (8) only represents the worst case, when none of the actors representing software executions lie on the critical path regarding graph throughput. Therefore, it is only meaningful, when the real-time criterion w.r.t. actor throughput of the input data process is met.

The computation of Equation (7) scales in the worst case with the number of edges $|E|$ and with the total number of phases $\phi(v)$ of all actors $v \in V$. Its computational complexity is thus $O(|E| * \overline{\phi})$, with $\overline{\phi}$, denoting the average number of phases of all actors $v \in V$. However, there are optimization possibilities, as the iteration over the phases of actors that are destinations of multiple relevant edges, only has to be done once per actor and intermediate results can be reused. Therefore, Equation (7) can be calculated in $O(|E| + |V| * \overline{\phi})$ time, which can be faster, as the number of edges $|E|$ may potentially be higher than the number of actors $|V|$. However, this comes at the price of a space complexity of $O(|V|)$, as for each actor the intermediate result needs to be stored. Note that the computational complexity is not affected by scheduling edges, as these are only added between actors mapped to the same processor core. Further note that the aforementioned computational complexity applies to the calculation of the data rate of a single hardware channel t, to which potentially all edges $e \in E$ are mapped, but it also applies to the computation of data rate of all hardware channels together, as edges can only be mapped once. The calculated data rate $\widehat{r}(t)$ can then be checked with the maximum data rate of the communication channel of the target architecture, in order to explicitly asses if real-time performance of the configuration under test is possible.

In on-body sensor networks, wireless low energy standards, e.g., BLE, are often used to communicate data. For shared communication mediums, generally message scheduling has to be considered which can affect timing behavior like throughput and latency. However, for wireless communication mediums, which are additionally prone to collision and data loss, scheduling can be difficult to predetermine. Additionally, the clocks of wireless sensors can drift and, without additional synchronization mechanisms, clocks can vary among wireless sensors within the same network. This poses additional non-determinism for an exact construction of a schedule. Worst-case approximations instead of explicit schedules are a possible solution, e.g., in form of upper and lower bound service curves as deployed by real-time calculus [72], which has been shown for the performance analysis of marked graphs in [73] and more recently been applied to model and analyze worst-case latency bounds on shared resources for systems modeled with FSM-SADF graphs [41].

3.4.2 Energy Trade-Off

Across different configurations, the major characteristics that influence total energy consumption of the wireless sensors are sampling frequency, processor utilization, and data rate on the wireless communication channels. Furthermore, the total device energy consumption is often a trade-off between increased processor utilization and reduced wireless transmissions, or vice versa. In any case, the energy consumption needs to be evaluated per configuration, which is possible if, for each individual hardware component the relationship between said characteristics and its energy consumption is known.

Energy consumption rate of the aforementioned components can be either acquired from technical specifications, calculated formally [45, 46] (given that necessary properties of the hardware are known) or fitted in small experiments. By analysis of processor utilization and data rate on communication

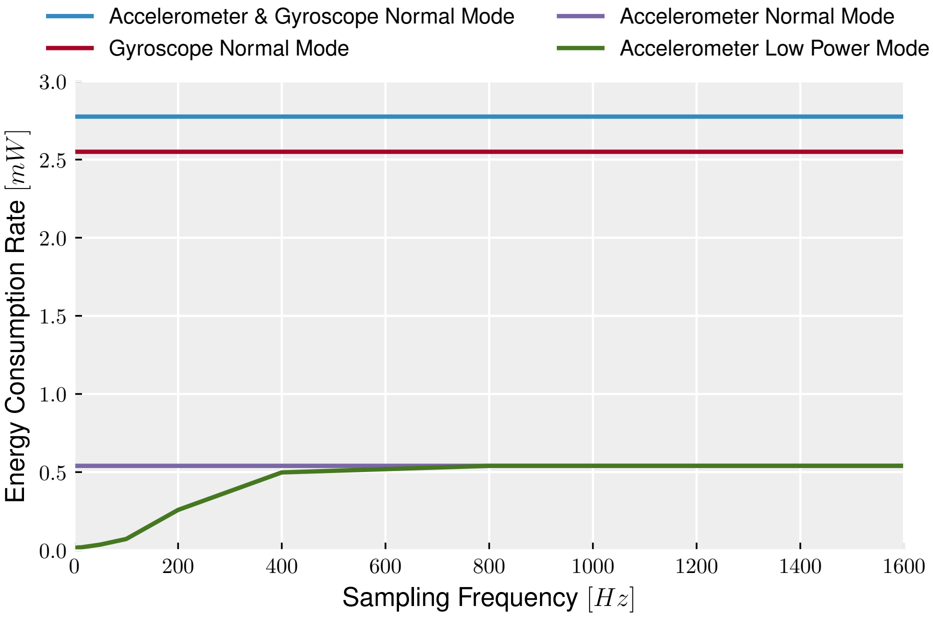


Figure 3.10: BMI160 energy consumption rate in relation to the sensor modality and the sampling frequency. Based on [74].

channels, differences in total device energy consumption rate can be assessed between individual configurations. Since energy consumption is the power consumption integrated over a time interval, analyses based on graph iterations naturally describe time intervals in which changes in power consumption repeat periodically. Thus, energy consumption or its differences between configurations can be calculated from average power consumption over a graph iteration, multiplied by its iteration period. However, for meaningful comparisons, graph iteration periods have to be equal in such case. Alternatively, average power consumption during the graph iteration periods can be compared directly. The latter was chosen in the thesis at hand and is in the following referred to as *energy consumption rate*.

Relationships between energy consumption rate can be linear or non-linear w.r.t. the aforementioned component characteristics. The energy consumption rate in relation to the sampling frequency of the BHI160 sensor sub-system used in the conducted experiments depends on the sensor type, and for the accelerometer is also non-linearly dependent on the sampling frequency, as Figure 3.10 depicts. For each sensor modality, a model-based estimate $\widehat{P}(f_{\mathsf{SE}})$ denotes the expected energy consumption rate at a certain sensor sampling frequency of the sensor SE in the hardware model of the DIANA boards.

The energy consumption rate of processor cores, depends on their architecture. Architectures without cache, branch predictions, pipelining and in general data-independent behavior, like microcontrollers, can be expected to have a rather linear relationship between processor utilization and energy consumption, since during idle time the controller has a smaller energy consumption rate compared to active time. This is shown in a small experiment with five sensors and five measurements per sensor. For each measurement, the processor load is changed by varying the amount of calculations within a loop, for each processed sensor sample. The calculations of the loop body are based on repeated Multiply and Accumulate (MAC) operations, which are representing the majority of instructions in the application domain. The corresponding processor utilization is calculated from measurements of execution time

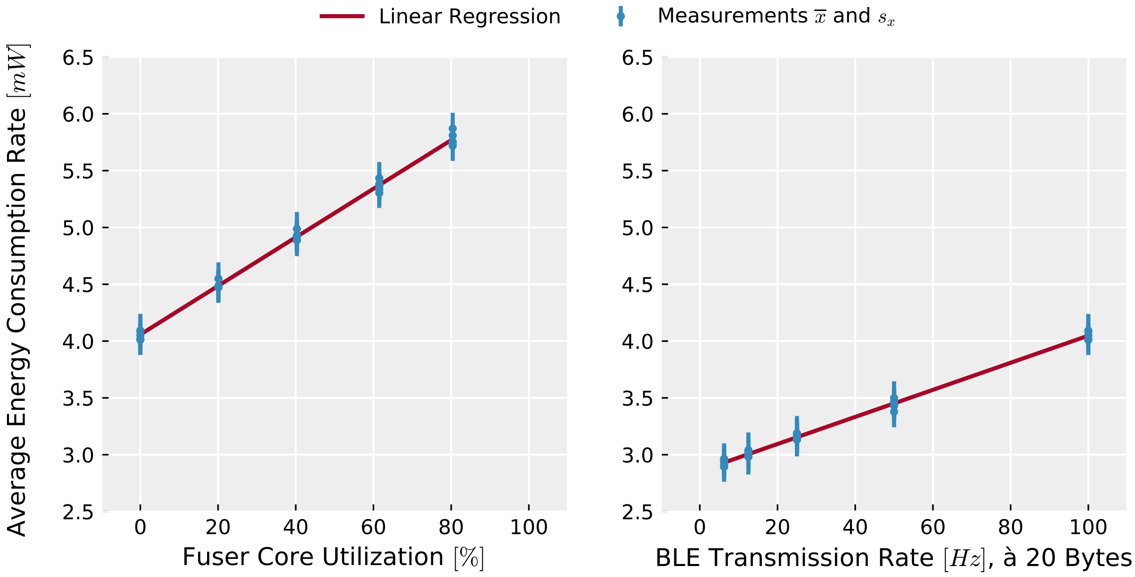


Figure 3.11: Energy consumption rate in relation to the Fuser Core utilization (left) and the BLE transmission frequency (right).

of the predefined processing load, divided by the period with which the code is executed, i.e., sensor sampling period. The detailed setup and execution of the experiments is explained in Section 3.5.4, as the same setup and method has been used there to evaluate the model accuracy. The relationship between processor utilization and energy consumption rate of the BHI160 Fuser Core can be seen on the left hand side in Figure 3.11.

Likewise, the relationship between energy consumption rate and data rate of the wireless communication over BLE has been derived from five of the sensors in experiments. Here, the energy consumption rate can be expected to increase linearly with the data rate. The data rate has been changed for different measurements, by communicating only each second, fourth, eights, etc., sample, without altering the sampling frequency. Note that the measured relationship, actually represents the change in energy consumption rate of the I^2C channel between sensor and BLE controller, the processor utilization of said BLE controller and the BLE transmissions together, in relation to the data rate. However, this can be considered as appropriate, as these components together represent the abstract communication channel within the dataflow model. The relationship between energy consumption rate and communication data rate is depicted on the right hand side in Figure 3.11.

From the conducted experiments, it can be seen that device energy consumption rate linearly depends on the processor load of the microcontroller and the BLE communication channel. The slopes of the fitted measurements represent model-based estimates of energy consumption rate $\widehat{P}$ in relation to the Fuser Core processor utilization $\widehat{P}(U(\mathsf{FU}))$ and the BLE transmition rate $\widehat{P}(r(t_1))$ on channel t_1 of the hardware model. The energy consumption rate models $\widehat{P}(U(\mathsf{FU}))$, $\widehat{P}(r(t_1))$, and $\widehat{P}(f_{\mathsf{SE}})$, can be used to annotate the results acquired from the analysis model, and summed up for each analyzed configuration, to allow for comparison across configurations. With an additional baseline configuration, for which the total energy consumption rate of the sensor node is known or measured, total worst-case device energy consumption rate for other configurations can be estimated. This will be shown in the experimental evaluation, assessing model accuracy. The measurements in Figure 3.11 have been fitted by linear regression, approximating $\widehat{P}(r(t_1)) = 0.0119\,mW/Hz$ BLE packet rate à $20\,bytes$ and $\widehat{P}(U(\mathsf{FU})) = 0.0214\,mW/\%$ processor utilization.

3.5 Experiments

An experimental study has been conducted to evaluate the accuracy of predicted energy consumption rates from analysis models. The worst-case energy consumption rate acquired from the models is compared to the measured energy consumption rate of five wireless sensor nodes with corresponding implementations on the Fuser Core. In order to provide additional test scenarios, five additional configurations to that of the case study have been designed for an extended evaluation, which will be described in the following.

3.5.1 Configurations

A set of five different on-sensor feature extraction configurations (mapping A) and two off-sensor feature extraction configurations have been selected, which are listed in Table 3.3. The first off-sensor feature extraction configuration is used as a reference for which the total energy consumption rate of the wireless sensor node is measured. By model-based analysis, differences in energy consumption rate to all other configurations w.r.t. the reference are analyzed. This allows to acquire an estimate of the total energy consumption rate of all other configurations, in order to compare with experimental implementations.

In the first column of Table 3.3, the configurations are named from C1 to C6. The second column specifies the sensor modality, which is either accelerometer (ACC), gyroscope (GYR), or both. Sampling frequencies of the sensors are listed in the third column, ranging from $50\,Hz$ to $200\,Hz$. The column *Window Length* specifies the size in samples of the sliding window segmentation. Note that *ANY* is used to indicate, that in configurations with mapping B, no feature extraction is actually implemented. The fifth column specifies the window overlap of the sliding window segmentation in % of the sliding window length. In the second to last column, the feature set is specified. The feature set description is a combination of abbreviations of mean (M), variance (V), skewness (S), kurtosis (K), peak (P), energy (E), and a sum ($\sum$), that are calculated over each axis of sensor data within the sliding window.

Table 3.3: Sensor settings, sampling frequencies f, sliding window parameters, feature sets, and mappings of all configurations.

Cfg.	Sensor	$f(\mathsf{SE})$ $[Hz]$	Window Length	Overlap	Features	Mapping
Ref.	ACC	100	*ANY*	*ANY*	*ANY*	B
C1	ACC, GYRO	100	128	75 %	MVSKPE	A
C2	ACC, GYRO	50	128	75 %	MVSKPE	A
C3	ACC	100	128	75 %	MVSKPE	A
C4	GYRO	200	50	0 %	$\sum$	A
C5	ACC	100	128	25 %	MV	A
C6	GYRO	200	*ANY*	*ANY*	*ANY*	B

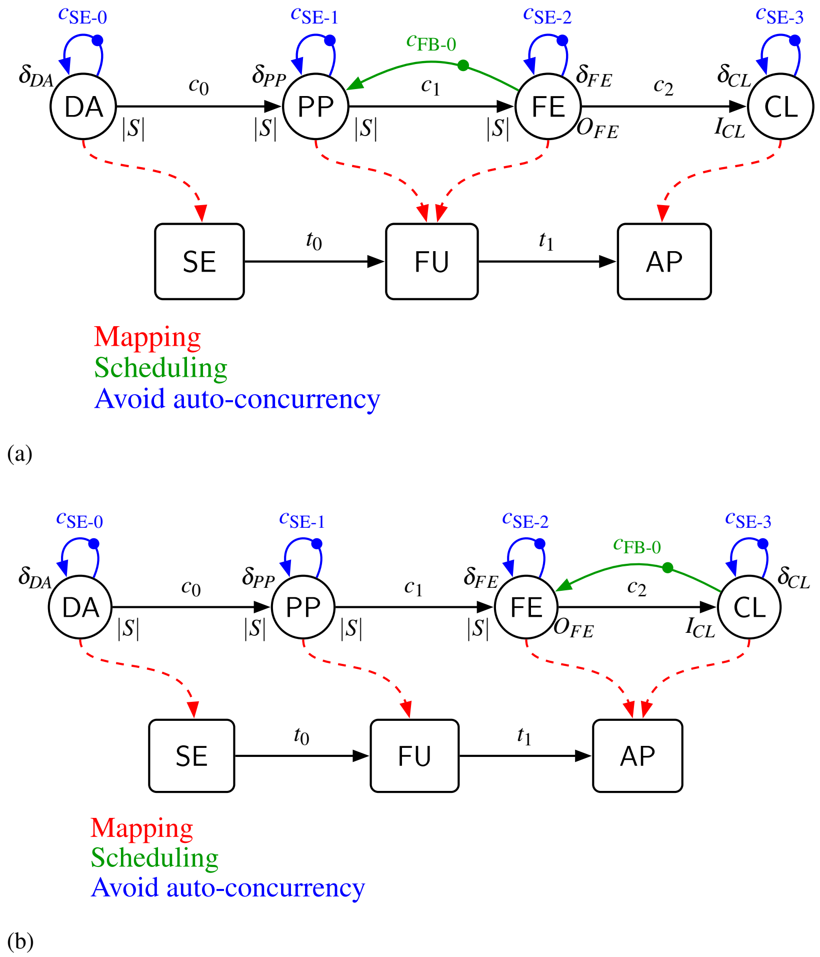


Figure 3.12: Two different mappings, calculating the feature extraction on (a) the Fuser Core (mapping A) or (b) the smartphone (mapping B).

Configuration C1 from Table 3.3 represents the feature set of the case study calculated on the wireless sensor node, i.e., Fuser Core. Furthermore, the reference configuration in the first row of Table 3.3 corresponds to C3, but in a different mapping, as well as C4 and C6 correspond to each other in mapping A and mapping B, respectively.

For the sake of simplicity, the analysis models have been reduced to represent only a single wireless sensor node of the full on-body-network, as extra-functional properties of all sensors are equal in same configurations in the dataflow representation. The analysis models are shown in Figure 3.12. Model annotations that change across configurations are abstracted and listed in Table 3.4. In each configuration, the number of sensor modalities (accelerometer and/or gyroscope) equals the production rates of DA on c_0 and PP on c_1 and the corresponding consumption rates of PP from c_0 and FE from c_1. These are denoted as $|S|$ in Figure 3.12 and listed in Table 3.4 for each configuration.

In order to acquire worst-case estimates of total device energy consumption rate of the wireless sensor nodes, the difference in energy consumption rates of the configuration under analysis and the reference must be worst-case estimates. To this end, worst-case execution times and sensor sampling frequencies (highest) have been annotated to the analysis models of configurations C1-C6, but best-case execution times and sensor sampling frequencies (slowest) have been annotated to the analysis model of the reference configuration.

For each analysis model, execution times of software actors have been annotated with their corresponding worst-case or best-case execution times on the corresponding processing unit. In the study at hand, software execution times captured by actors have been measured on the Fuser Core. Furthermore, the performance of the wireless on-body network has only been analyzed until the BLE receiver of the smartphone. Thus, for mapping B, the features are neither calculated on the Fuser Core nor on the smartphone, as the smartphone AP computations do not affect the sensor nodes. Hence, the same execution times on the Fuser Core have been annotated to the actors mapped to the smartphone, for the sake of simplicity. For the same reason, actor CL is annotated with a placeholder execution time of $\delta_{CL} = 1,000\,\mu s$.

The corresponding implementation of each configuration is presented in the following section.

3.5.2 Implementation

For the experimental study, five of the wireless sensor nodes have been used to implement the configurations explained in Section 3.5.1. In mapping B, the wireless sensor nodes send pre-processed sensor samples to the smartphone. Each 3D single-precision floating-point sensor sample is packed together with a one-byte ID value specifying the sample type (raw sample or feature vector entry and in the latter case, its feature ID), into a 20-byte BLE packet (remaining bytes are zeros). The same packing scheme is used across all different configurations in the experiments.

For mapping A, the window-based feature extraction has been implemented on the Fuser Core as a virtual sensor routine. In each invocation, the new sensor samples of accelerometer and/or gyroscope are stored in a ring buffer for the sliding window segmentation. Furthermore, the online one-pass computation of mean, variance, skewness, and kurtosis [66] has been implemented for each of the overlapping sliding windows. The implementations are based on an existing implementation from J. D. Cook [75] for configurations C1, C2, C3, and mean and variance only in C5. For C4, the sum of sensor sample values $\sum$ is also implemented as an online updating of its current value with each new sample. In the last invocation (32-th for C1,C2, and C3, 50-th for C4, and 96-th for C5), the final values for

Table 3.4: Analysis model parameters of all configurations.

Cfg.	δ_{DA} $[\mu s]$	$\lvert S\rvert$	$\phi(FE)$	O_{FE}	δ_{PP} $[\mu s]$	δ_{FE} $[\mu s]$	I_{CL}	Mapping
Ref.	10101	1	32	$[\{0\}_{\times 31}, 6]$	375	$[\{93\}_{\times 31}, 11343]$	6	B
C1	9900	2	32	$[\{0\}_{\times 31}, 12]$	1375	$[407, \{375\}_{\times 30}, 23500]$	12	A
C2	19801	2	32	$[\{0\}_{\times 31}, 12]$	1375	$[407, \{375\}_{\times 30}, 23500]$	12	A
C3	9900	1	32	$[\{0\}_{\times 31}, 6]$	563	$[219, \{204\}_{\times 30}, 11766]$	6	A
C4	4950	1	50	$[\{0\}_{\times 49}, 1]$	1094	$[\{32\}_{\times 49}, 63]$	1	A
C5	9900	1	96	$[\{0\}_{\times 95}, 2]$	563	$[110, \{94\}_{\times 94}, 188]$	2	A
C6	4950	1	50	$[\{0\}_{\times 49}, 1]$	1094	$[\{32\}_{\times 49}, 63]$	1	B

mean, variance, skewness, kurtosis, and $\sum$ are calculated for the entire window. Furthermore, for C1, C2, and C3, an FFT is calculated in the last invocation over the 128 samples that have been buffered in the window. The FFT is a 128-point implementation composed of seven Radix-2 butterfly stages as described by Cooley and Tukey in [76], which are computed over an in-place buffer with Decimation In Time (DIT). To this end, the in-place array of 128 complex single-precision floating-point values is filled and re-ordered at initialization with the 128 input samples of the window with bit reversed indexing. For faster computation, twiddle factors have been calculated at compile time and stored in an array, which is indexed at run time.

However, as the FFT calculation in C1, C2, and C3 for all three axes of a sensor signal together requires more time than the sensor sampling period of C1 and C3, i.e., $10\,ms$, a general implementation scheme has been selected for C1, C2, and C3, in which the FFT computation for each individual axis of a sensor signal is distributed to the following virtual sensor routine invocations. Although not represented by the analysis models, this leads to the same throughput and average processor utilization across 32 sampling periods. In the analysis models, the last CSDF phase of the feature extraction actor is annotated with the execution time of computing the FFT for all sensor axes in a single invocation. Although this exceeds the sampling period, it does not exceed together with the execution time of each invocation of the pre-processing actor the iteration period of the analysis models. This is possible due to the FIFO buffer characteristics of graph edges, which are unbounded in the analysis models. However, in the implementation on the Fuser Core, such buffering of samples is not possible. As a result, in the implementation, the FFT calculation of each axis needs to be sequenced over the following virtual sensor routine invocations. Nevertheless, the inability to buffer samples can be integrated in the analysis models by feedback channels with the corresponding number of initial tokens. This results in a throughput constraint of the graph, which is lower than the modeled sensor sampling frequency when the FFT for all sensor axes in performed in each 32-th invocation. Thus it indicates a performance which does not meet real-time requirements, similar to a corresponding implementation. However, unbounding the channels in the analysis model, does not reflect the buffering constraint on the target hardware but leads to the same graph results w.r.t. graph throughput, transmission throughput, and processor utilization, as if buffers were bounded, and sequenced FFT calculations were modeled. Although, the analysis models in its presented form (not including buffer constraints and FFT sequencing) do not reflect the same time behavior as the actual implementation, e.g., w.r.t. delay, they reflect the important extra-functional properties, that are of concern in this chapter, i.e., BLE transmission throughput, system throughput, and processor utilization. This has been substantiated by an additional analysis model, that reflects the aforementioned implementation details, which however, is not shown for the sake of brevity.

In order to allow an FFT calculation of each axis sequenced over the following invocations of the virtual sensor routine, the ring-buffer containing samples of all overlapping sliding windows for C1, C2, and C3 is implemented with a size of 230 samples [3.1], instead of 224 samples.

Also different to the analysis models, the virtual sensor routine does not sent all calculated feature vector entries at the same time after the 32-th (C1, C2, C3) or 96-th (C5) sample has been processed as part of one of the overlapping windows. The feature vector entry sending is rather sequenced

---

[3.1] The 230 samples in the ring buffer account for a full sliding window (128), the additional samples of the three following overlapping sliding windows ($128 \cdot (1 - 0.75) \cdot 3$), and the oldest six (maximum number of axes among C1, C2, and C3, for which the FFT calculation is sequenced) samples that need to be stored for the following six invocations due to the sequenced FFT calculation.

over the following virtual sensor routine invocations as well, as the firmware of the BHI160 does not allow a burst communication of several 3D sensor samples within a single routine invocation into its FIFO buffer. However, this restriction might not be known at design time of the system and is therefore not considered in the analysis models, leading to a different delay, but the same throughput on the BLE channels between wireless sensor nodes and smartphone. Nevertheless, if such hardware-specific knowledge is known at design time, it can be integrated in the analysis models by adapting the production rates of corresponding CSDF actor phases. In fact, the additional analysis model mentioned for verification of the sequenced FFT calculation includes this communication behavior of the implementation as well and does indeed produce equal results w.r.t. graph throughput and data throughput on the BLE channels as the presented analysis models in Figure 3.12.

As for each configuration, data transmission rates are known by application parameters, a corresponding BLE connection interval has been chosen for each configuration that is as large as possible to just allow the necessary transmission throughput including a margin of $+1\%$. The margin accounts for the uncertainty of the sensor sampling frequency, as stated in the data sheet of the BHI160 [2]. For each configuration a corresponding connection interval length has been chosen, based on the maximum number of BLE packets per connection interval of the Samsung Galaxy S5. The number of BLE packets per connection interval of the Samsung Galaxy S5 has been empirically determined to be four. A non-exhaustive list of number of packets per connection interval for common smartphones can be found in [77]. The BLE latency, i.e., the number of BLE connection intervals that can be skipped by the BLE controller of the sensor nodes before sending a particular BLE packet, was set to zero. In other words, each BLE packet that is ready to be send, must be sent at the next possible point in time within the connection interval progression between BLE sender and receiver.

3.5.3 Model Evaluation

For each configuration, repetition vector and throughput have been analyzed from the analysis models by using the SDF^3 tools [13]. From throughput and repetition vector, processor utilization $U(\mathsf{FU})$ of the Fuser Core and the BLE packet transmission rate $r(t_1)$ have been calculated by using Equation (5) and (7), respectively, as described in Section 3.4. Based on processor utilization, BLE transmission throughput, and sensor sampling frequency, together with sensor energy consumption models, differences of device energy consumption rates between all configurations and the reference have been calculated. Furthermore, they have been added to the measured energy consumption rate of the reference configuration, in order to acquire worst-case estimates of total device energy consumption rates $\widehat{P}(D)$ for each configuration. The acquired results are summarized in Table 3.5. Note that according to the analysis models, actor CL should only fire once per iteration, and thus should have a repetition vector entry of $\gamma(CL) = 1$. However, although CSDF in general allows different number of phases for different actors, the SDF^3 implementation is restricted to a uniform number of CSDF phases across all CSDF actors. This can either be achieved, by modeling CL and all other actors with as many phases as the least common multiple of number of CSDF phases across all actors with equal behavior of CL in each of its firings. However, since actor CL is firing once every 32, 50, or 96 samples, this would lead to a graph iteration that is a hyper period of 32, 50, or 96 samples and the corresponding phases of CL, increasing the analysis time. Instead, actor CL, was modeled with 32, 50, or 96 phases, of which the first 31, 49, or 95, respectively, have been annotated with a production rate, consumption rate, and execution time of zero, in order to avoid hyper period iterations but without additional timing affects on the graph results.

3.5.4 Experimental Setup

In this section, the experimental setup is described, with which device energy consumption rate of the wireless sensor nodes is acquired for each configuration. Due to its ease, the approach from [78] has been followed to measure device energy consumption rate. The wireless sensor nodes are supplied by a constant $3V$ power supply with a 100Ω shunt resistor R_S in series. The voltage drop over R_S has been measured by a DSO-X 3034A oscilloscope from Agilent Technologies. For each configuration, the voltage drop has been measured and averaged in $10ms$ windows with a resulting resolution of $10MSamples/s$ by the oscilloscope over a time of at least $100s$. The temperature dependency of the measured result was considered to be neglectable, although environmental temperature at the begin and end of all experiments as well as air humidity have been put on record in the experiment protocol. Due to the abstract nature of energy consumption rate estimation from analysis models, a certain degree of error is expected w.r.t. final implementations. Since these errors are expected to predominate acquired results, an estimation and propagation of measurement error has not been carried out.

The energy consumption rate of each sensor node has been measured individually for each configuration. To this end, each sensor node has been placed in a distance of $50cm$ from the smartphone. After establishing the BLE connection between sensor node and smartphone, the arrival of sensor data at the smartphone has been monitored. After successful transmission of sensor data over a period of approximately $10s$, the measuring process on the oscilloscope has been started, which ran at least $100s$ before results were stored on an attached flash drive.

3.5.5 Experimental Results

The energy consumption rate P of each sensor node measured in all configurations is depicted on the left hand side of Figure 3.13. The corresponding mean energy consumption rate $\overline{P}$ across all sensor nodes in each configuration and standard deviation s_P is depicted on the right hand side of Figure 3.13. The underlying measurements P , their mean $\overline{P}$, and Coefficient of Variation (CV) CV_P, are summarized in Table 3.6 and corresponding relative deviations of measurements from the model-based results are summarized in Table 3.7. Note that in the thesis at hand, a relative deviation Δ_r $[\%]$ is calculated by subtracting the measured results from the model-based results (absolute deviation Δ) and divided by the measured results. Thus, a relative deviation of Δ_r reads as: "the model-based results deviate $\Delta_r\%$ from the measurements".

Table 3.5: Model-based analysis results.

Cfg.	$TH(G)$ $[\mu s^{-1}]$	γ_G	$\widehat{U}(\mathsf{FU})$	$\widehat{r}(t_1)$ $[s^{-1}]$	$\widehat{P}(D)$ $[mW]$
Ref.	$3.09375E-06$	$[32,32,32,32]^T$	3.7%	99.00	4.051
C1	$3.15657E-06$	$[32,32,32,32]^T$	25.0%	37.88	6.031
C2	$1.5782E-06$	$[32,32,32,32]^T$	12.5%	18.94	5.802
C3	$3.15657E-06$	$[32,32,32,32]^T$	11.4%	18.94	3.099
C4	$4.0404E-06$	$[50,50,50,50]^T$	22.8%	4.04	5.402
C5	$1.05E-06$	$[96,96,96,96]^T$	6.6%	2.10	2.897
C6	$4.04040E-06$	$[50,50,50,50]^T$	22.1%	202.02	7.761

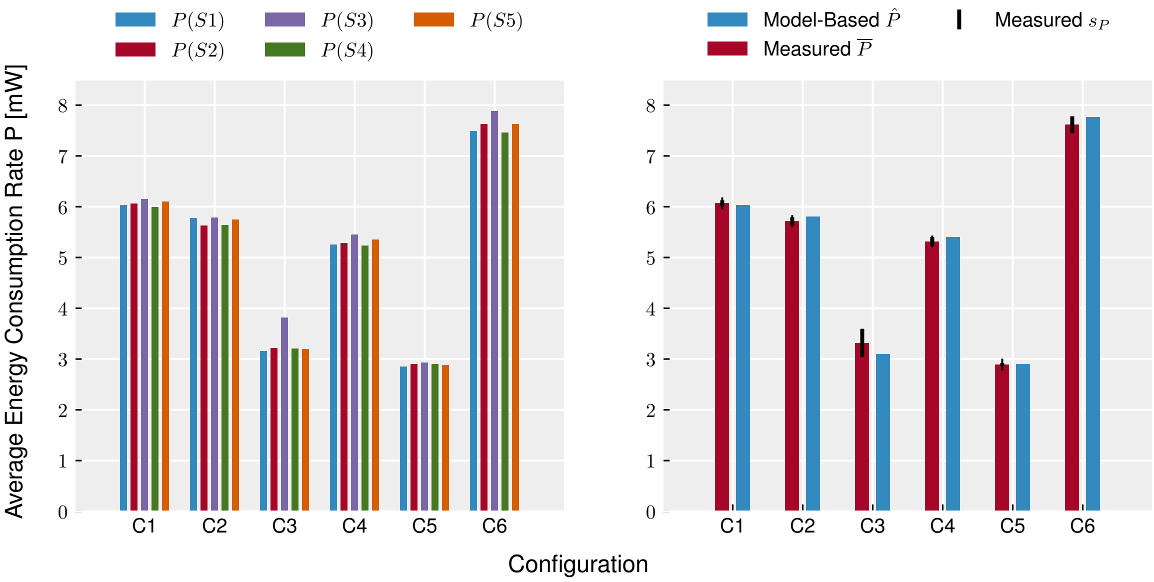


Figure 3.13: Measured (left) and comparison with model-based results (right) of the average energy consumption rate in all configurations.

The mean energy consumption rate $\overline{P}$ between different configurations ranges from $2.89\,mW$ to $7.62\,mW$. This shows the high impact of application configurations as well as system configurations (compare C5 and C6) on the device energy consumption rate. The coefficient of variation shows a small deviation across different sensor nodes in the same configuration in the range of single-valued percentages. However, the coefficient of variation of C3 is by far the highest, which is caused by a substantial deviation of sensor S3 in configuration C3, which can be seen on the left hand side of Figure 3.13. A Grubbs test over all relative differences between model-based and measured results from Table 3.7 showed sufficient confidence (at a significance threshold of $\alpha = 0.05$) that the measurement from S3 in C3 is an outlier, with $G(N = 30) = 4.57,\ p < 0.001$. Furthermore, a Dixon test over the relative differences of only C3 from Table 3.7 could confirm with sufficient evidence, that S3 in C3 is an outlier, with $Q(N = 5) = 0.90,\ p < 0.001$. Note that a Dixon test has been used over the measurements from C3, since the sample size of $N = 5$ is too small for a meaningful Grubbs test. This outlier can be caused by a damaged sensor or power supply of the sensor node.

Table 3.6: Measured average energy consumption rate of the DIANA boards in all configurations.

Cfg.	$P(S1)$ $[mW]$	$P(S2)$ $[mW]$	$P(S3)$ $[mW]$	$P(S4)$ $[mW]$	$P(S5)$ $[mW]$	$\overline{P}$ $[mW]$	CV_P
C1	6.03	6.06	6.15	5.99	6.10	6.07	1.01 %
C2	5.78	5.63	5.79	5.64	5.74	5.72	1.34 %
C3	3.15	3.21	3.81	3.21	3.20	3.32	8.41 %
C4	5.25	5.28	5.45	5.24	5.35	5.31	1.67 %
C5	2.85	2.90	2.93	2.90	2.88	2.89	1.02 %
C6	7.49	7.63	7.88	7.46	7.63	7.62	2.16 %

Table 3.7: Relative deviations of the model-based estimates from the measured energy consumption rates.

Cfg.	$\Delta_r\,P(S1)$	$\Delta_r\,P(S2)$	$\Delta_r\,P(S3)$	$\Delta_r\,P(S4)$	$\Delta_r\,P(S5)$	$\Delta_r\,\overline{P}$
C1	$-0.02\,\%$	$-0.44\,\%$	$-1.95\,\%$	$0.65\,\%$	$-1.12\,\%$	$-0.58\,\%$
C2	$0.40\,\%$	$3.14\,\%$	$0.26\,\%$	$2.84\,\%$	$1.01\,\%$	$1.51\,\%$
C3	$-1.76\,\%$	$-3.51\,\%$	$-18.75\,\%$	$-3.33\,\%$	$-3.04\,\%$	$-6.56\,\%$
C4	$2.90\,\%$	$2.21\,\%$	$-0.91\,\%$	$3.18\,\%$	$0.91\,\%$	$1.64\,\%$
C5	$1.72\,\%$	$-0.17\,\%$	$-0.99\,\%$	$-0.03\,\%$	$0.72\,\%$	$0.24\,\%$
C6	$3.62\,\%$	$1.77\,\%$	$-1.49\,\%$	$4.01\,\%$	$1.77\,\%$	$1.90\,\%$

A comparison to the results acquired from the models is depicted on the right hand side of Figure 3.13. Apart from sensor node S3 in configuration C3 as previously discussed, the highest deviation from the model-based results is approximately $-3.51\,\%$ (S2 in C3). Although model-based results are supposed to represent worst-case estimates, not all measurements have been found below model-based estimates. However, a trend can be seen by comparing the average measurements with model-based results. Furthermore, except comparably small deviations otherwise, energy consumption rate of sensor S3 can be observed highest among all sensor nodes in each configuration, which might be an indicator for a faulty soldered joint in the power supply of the sensor node. Moreover, for all sensor nodes, configuration C3 seems to be generally under estimated by the model. A two-factorial Analysis of Variance (ANOVA) of the relative differences between model-based and measured results excluding C3 from S3, with factors *sensor* and *configuration*, showed a significant effect for *configuration* on the relative prediction error, with $F(5,19) = 16.05,\ p < 0.001$, and a significant effect for *sensor*, with $F(4,19) = 8.05,\ p < 0.001$. As a result, it can be concluded that the dataflow graph model does not account for all effects on the device energy consumption. This can be explained by the level of abstraction of the simplistic energy models. As these merely relate the average energy consumption of, e.g., the processing units, to its processor utilization by means of execution time, the type of code that is actually executed (e.g., fixed-point operations, floating-point operations, or memory operations) is not accounted for in the models. Furthermore, the difference between sensors can be explained by small deviations due to manufacturing of the sensor nodes, as well as in the soldering of cables for voltage supply for the conducted experiments. The impacts by means of group mean differences between configurations and between sensors can be seen from the ANOVA coefficients summarized in Table 3.8. It can be observed, that the configurations tend to have a higher impact on the relative differences between model-based and measured results than the sensors.

Table 3.8: ANOVA coefficients of the relative deviations between model predictions and measurements.

(Intercept)	S2	S3	S4	S5	C2	C3	C4	C5	C6
$0.34\,\%$	$-0.64\,\%$	$-2.89\,\%$	$0.08\,\%$	$-1.10\,\%$	$2.10\,\%$	$-2.83\,\%$	$2.23\,\%$	$0.82\,\%$	$2.51\,\%$

However, a t-test of all relative differences between model and measurements does not show a significant difference from zero, with $t(28) = 1.10,\ p = 0.28$, w.r.t. the significance level of $\alpha = 0.05$.

The aforementioned statistical tests have been performed under the assumption of normally distributed relative differences, which however, is not generally evident from the acquired results. Therefore, the performed tests have been complemented by additional non-parametric tests (Friedmann rank sum test [79] and Wilcoxon signed rank test [80], respectively), which could substantiate the aforementioned conclusions. As a result, the acquired model-based analysis results show a sufficient accuracy w.r.t. to their corresponding implementations in order to substantiate design decisions regarding system configuration as well as application configuration.

3.6 Discussion

In this chapter, state-of-the-art activity recognition systems have been modeled with cyclo-static dataflow graphs up to the dimensionality reduction stage in an activity recognition chain. Furthermore, an on-body sensor network of wireless sensor nodes with integrated processing capabilities has been modeled, to which dataflow actors have been mapped and scheduled. From the model-based mapping and scheduling representations, formal analysis of throughput and repetition vector has been conducted with available frameworks. From these, a new approach for estimating worst-case energy consumption rate of the wireless sensor nodes has been applied and evaluated by experiment with corresponding implementations on a selected target hardware. In the conducted experiments an average estimation accuracy of over 92 % could be achieved.

The proposed model-based estimation approach for device energy consumption rate showed a sufficient accuracy to substantiate design decisions in early design stages of energy-efficient human activity recognition systems. However, the proposed estimation approach for device energy consumption rate has its limitations in applicability. Firstly, the data source has to be explicitly modeled into the system model, in order to acquire system throughput estimates that are expected during run time of the system. Based on these, worst-case processor utilization and transmission throughput can be further analyzed as a foundation for the estimation of device energy consumption rate. Secondly, for accurate estimation, the highest input data rate during run time of the system should be known at design time and modeled into the system model, in order to represent worst-case behavior. As a requirement, input data rates have to be deterministic across iteration periods, or even constant. This requirement cannot generally be assumed. However, sensor-based human activity recognition systems tend to fulfill the aforementioned criteria, as predetermined sampling rates are typical in the application domain.

The results in this chapter are focused on dataflow-oriented ARC designs with very little to no control flow. The presented techniques are expected to have a significantly lower estimation accuracy, when the system is dynamically changing, e.g., in context-aware applications, or in dynamic approaches for feature and sensor selection. Here, an estimate of the worst-case behavior is possible, but likely to be too pessimistic. Although, dynamic changes have not been addressed in the chapter at hand, applicable dataflow models of computation exist, to successfully capture dynamic behavior and estimate extra-functional properties that indicate energy consumption rates. Corresponding design and analysis approaches are presented in Section 5.1.

4 Thread-Level Parallelism

Contents of this chapter have been previously published in the following publications:

- Exploiting Thread-Level Parallelism in Template-Based Gesture Recognition with Dynamic Time Warping [G5]
- Sensor-Based Online Hand Gesture Recognition on Multi-Core DSPs [G6]

This chapter investigates different parallelization strategies for multi-processor architectures based on task- and data-level parallelism in computationally intensive ARC stages. The goal is to identify the impact of parallelization on different extra-functional properties like throughput, latency, but also on load balancing and thus indirectly on the energy efficiency of the overall system. More importantly, it is evaluated how different parallelization approaches can be represented by dataflow models. As the mapping and scheduling of the recognition software onto the available hardware directly influences said extra-functional properties, design-time estimations are of major importance to avoid costly design cycles. To this end, analysis results of latency, throughput, and processor utilization from system-level models are compared with experimental implementations to evaluate model accuracy.

Finding the best mapping and scheduling of a recognition software onto an available hardware w.r.t. to some extra-functional property at design time is one of the major motivations of model-based analysis. However, a crucial requirement of that process is the evaluation of how accurate model-based analysis results can represent a real implementation. To this end, state-of-the-art parallelization approaches are represented into analysis models and compared with corresponding implementations. More specifically, the accuracy of extra-functional properties analyzed from such models is evaluated w.r.t. corresponding implementations.

The extra-functional properties that play a major role for activity and gesture recognition systems are throughput, latency, and energy efficiency. The energy consumption induced by communication channels, as presented in Chapter 3, is not considered in this chapter, as the majority of data communication is performed via shared-memory, which is abstracted in dataflow actors in the deployed models. In general, energy efficiency is not directly evaluated as the total energy consumption of the system in this chapter, but indirectly by means of processor utilization and the corresponding implications for energy reduction techniques. Furthermore, processor utilization is affected by other application parameters like sensor sampling frequency, window lengths, or window overlap. As a result, processor utilization is evaluated as an indirect indicator for energy efficiency. The evaluations in this chapter are focused on these extra-functional properties.

The chapter is structured as follows. In Section 4.1, related work is discussed and distinguished from methods applied in this chapter. In Section 4.3, a hand gesture recognition application is introduced to which different parallelization approaches have been applied as a case study. Fundamentals on dynamic time warping, on which the hand gesture recognition system is based, are provided in Section 4.2. This is followed by the description of the selected parallelization approaches and their representation in dataflow-based system-level models in Section 4.4. In Section 4.5 the experimental setup of the hand gesture recognition system in different parallelization configurations is described as well as the derivation of the corresponding analysis models from the application model. Details about the implementation of the hand gesture recognition system can be found in Appendix A. The evaluation of the model accuracy as well as recognition accuracy of the experimental implementation is conducted in Section 4.6. Lastly, conclusions are drawn in Section 4.7.

4.1 Related Work

Dynamic time warping For the evaluation part of this chapter, a hand gesture recognition system has been developed. Hand movements are captured with a glove that is fitted with accelerometer sensors. In the literature, different classification algorithms can be found for the recognition of hand gestures performed with a sensor glove. A k-NN approach has been studied in [81] and compared to a Dynamic Time Warping (DTW) approach. The latter showed slightly better recognition accuracy, but at higher computational costs. Other approaches make use of Hidden Markov Models (HMM) [82] or Artificial Neural Networks (ANN) [83]. A comparison of recognition algorithms for hand gestures performed with a Wii controller showed a superior performance of DTW compared to HMMs, SVMs, k-NN, and ANNs. Based on the aforementioned results, the hand gesture recognition system developed for evaluation purposes in Section 4.6 is based on DTW.

Some studies show, that a DTW-based recognition on mobile devices is not optimal, due to the high computational complexity and its quadratic memory consumption [84]. To this end, a DTW implementation with linear space complexity has been used in the evaluation part. A high computational complexity is considered as a suitable choice, as the purpose of exploiting parallelism in the classification of gestures and activities is to reduce latency and increase throughput of computationally demanding algorithms. To this end, DTW is considered as a suitable candidate. Furthermore, the selected platform for evaluations is a multi-processor DSP platform. DSP architectures have shown a good suitability for DTW-based recognition systems [85].

Some studies exist, exploiting instruction-level parallelism to accelerate DTW [86]. As application-specific architectures are required, this has not been adopted in the thesis at hand. Furthermore, inter-core communication could be identified as a major impact on the overall performance. As a result shared-memory architectures are recommended in [87]. Therefore, a shared-memory architecture has been selected for implementations described in this chapter. In contrast to instruction-level parallelism, thread-level parallelism could be successfully exploited for DTW implementations in [88]. Among others, this parallelization approach has been adopted in Section 4.4 as well.

Parallelized classification algorithms In the last two decades, there have been attempts to accelerate classification algorithms by exploiting parallelism of different kinds. A parallel implementation of an SVM classifier on GPUs was introduced in [89], increasing the throughput for offline classification in large datasets. In [90], parallel implementations of a variety of classification algorithms are introduced, i.e., k-NN , Naive Bayes, and Decision Trees (DT). The parallel implementations are based on MapReduce [91] to batch process large datasets on computing clusters. Run-time environments like MapReduce [91] and Dryrad [92] are exploiting thread-level parallelism in batch processing tasks of large datasets and offer an efficient distribution and scheduling of the software onto multi-core CPUs of a single computer up to large clusters of computing systems. The parallelization scheme is evaluated on dataflow representations of the software. However, while the underlying parallelization approaches derived from dataflow representations can be adopted for online activity and gesture recognition, Dryrad and MapReduce are designed for batch processing tasks, and are suited for offline classification tasks in datasets, rather than online recognition with embedded systems.

Zaki et al., introduced a parallel implementation of DTs on a shared-memory multi-processor architecture [93]. Specifically, they propose an approach for the construction of parallel decision trees during the training phase. Their approach resembles a candidate for inter-segment parallelization, which will be introduced in Section 4.4.1.

A parallel implementation of HMMs as a low-level classifier on multiple sensor nodes has been proposed by Zappi et al. in [51]. Furthermore, a dataflow representation of the system software is used to activate data acquisition and processing from different sensors at run time. The online sensor selection approach allows prolonging system lifetime in case some sensors become unavailable due to, e.g., drained batteries, while preserving a certain recognition accuracy. Dataflow representations of the system software are used for decisions on the sensor selection, at run time. The approach proposed by Zappi et al. exploits system-level parallelism by executing low level classifier on different sensor nodes and uses dataflow representation for optimal mapping and scheduling at run time. However, no design-time estimation is performed, although it could complement and extend the run-time control by selecting configurations that allow a certain recognition accuracy and a maximum latency, minimum throughput, or energy consumption of the system. The approaches discussed in this chapter are complementary to their work.

Parallelism in dataflow representations Several approaches of exploiting parallelism from dataflow representations exist in the literature. One of these approaches is implemented in the run-time environment Sprout [94]. As Sprout is introduced to speedup the execution of streaming applications, it has been applied to video-based gesture and activity recognition systems [95]. Although, no design-time analysis is performed to substantiate design decisions, the proposed parallelization approaches have been adopted in Section 4.4.

Other approaches automatically exploit task-, instruction-, and data-level parallelism from software written in programming languages that are based on dataflow representations like StreamIt [96]. These dataflow representations are SDF alike and allow automated exploitation of the aforementioned parallelism by integrated compiler analysis [50, 97–99]. Some of these are estimating execution time at compile time for optimal mapping and scheduling on the underlying multi-processor hardware [50]. Activity and gesture recognition systems qualify as streaming applications as defined by [96]. That is, they follow typical characteristics, e.g., large streams of data, stable communication patterns, enumerable occasional modifications of the stream structure, and high performance expectations [96]. However, the aforementioned approaches assume that software is written from scratch. In human activity recognition, often implementations of software stages already exist as a result of an offline evaluation w.r.t. recognition accuracy. These are typically implemented in various programming languages (e.g. Python, C, R, Tcl/Tk, Java). Furthermore, activity recognition systems are often composed of multiple heterogeneous devices and processor architectures, that come with their individual compilers and programming languages, e.g., Java and Kotlin for Android, C/C++ for multi-processor DSPs like the TI TMS320C6678, or embedded sensor controllers in modern sensor sub-systems. Thus, a rewriting of available software stages in a specific language causes increased development time and compilers for various devices in such language might not be available. Therefore, these languages have a rather restricted application for the design of activity and gesture recognition systems.

Moreover, approaches to automatically exploit parallelism are integrated in compilers of specific processor architectures and devices [50, 97–99]. Thus, they can only operate on a partition of the distributed system software that is executed on a device supporting these approaches.

The model-based system-level approaches introduced in the thesis at hand, can be considered as the system-level counterparts of the aforementioned platform-based approaches. In contrast to these, the focus of the thesis at hand is on mapping and static order scheduling at design time together with a design-time analysis of extra-functional properties, e.g., throughput, latency, processor utilization.

Furthermore, a domain-specific language for system-level design of embedded gesture and activity recognition systems does not exist. To this end, task- and data-level parallelism is exploited by hand directly on abstract system-level models based on synchronous and cyclo-static dataflow graphs.

4.2 Dynamic Time Warping

In this chapter, the studied parallelization approaches are applied to a hand gesture recognition system as a case study. This system is based on dynamic time warping, in order to detect gestures within the sensor signals that capture the hand movements of a person. Consequently, DTW will be briefly introduced in the following.

Dynamic time warping is an algorithm to determine the similarity of two time-discrete signals. This is done by calculating a distance measure[4.1] between the two signals while the signals are non-linearly aligned along their time axes. This allows for comparison of signals of difference length. The application of the DTW algorithm ranges from speaker or voice recognition [85, 100], activity recognition [101], hand written word recognition [102, 103], to gesture recognition [104, 105].

DTW aligns, or *warps*, the time axes of two signals in a non-linear way to achieve maximal similarity between the signals. The global distance of the two signals is the accumulation of the local distances between the corresponding signal values of the best non-linear time alignment. Figure 4.1 shows the warping of two time-discrete signals. Every dashed line represents the correspondence of two signal values for the best alignment. In Figure 4.1, color coding is used to represent local distances of corresponding signal values. In order to find the best signal alignment and the corresponding global distance, a dynamic programming approach is applied on a cost matrix D. The cost matrix is an $N \times M$ matrix, with N and M corresponding to the lengths of two time-discrete signals $X = (x[0], x[1], ..., x[N-1])$ and $Y = (y[0], y[1], ..., y[M-1])$. Each matrix element $D[i, j]$ with $0 \leq i \leq N-1$ and $0 \leq j \leq M-1$ corresponds to the pair $(x[i], y[j])$ and captures the distance of the path with the currently smallest accumulated local distances d from the origin of the matrix, i.e., $D[0,0]$ to the corresponding element $D[i, j]$. The local distance $d(x[i], y[j])$ between the signal values $x[i]$ and $y[j]$ can be any distance metric. Commonly, the euclidean distance is used for d. The path is restricted to begin at the origin of the matrix and follow only neighboring elements that either correspond to an incremental step in one or in both signals along their time axes. Hence, no signal value is skipped. The cost matrix is filled either row or column wise beginning in $D[0,0]$ by calculating each element value $D[i, j]$ with the following equation:

$$D[i,j] = min \begin{pmatrix} D[i-1,j] \\ D[i-1,j-1] \\ D[i,j-1] \end{pmatrix} + d(x[i], y[j]). \tag{9}$$

The global distance of the two non-linear aligned signals X and Y is represented by $D[N-1, M-1]$. The path with the smallest accumulated distance from the origin $D[0,0]$ to the last element $D[N-1, M-1]$ is called the *warping path* and represents the best alignment between the two signals w.r.t. to the accumulated local distances along the path. In order to acquire the warping path, a backtracking from $D[N-1, M-1]$ to $D[0,0]$ has to be performed. However, for applications that only require the DTW

[4.1] Note that although the calculated DTW distance between two signals might be based on local distance metrics, the overall resulting DTW distance is not a distance in the sense of a metric. However, the term *DTW distance* is established in the literature and is used as such in the thesis at hand.

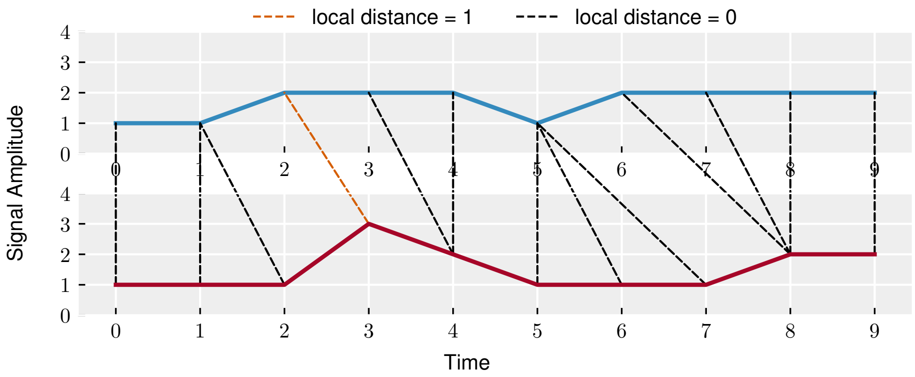


Figure 4.1: Visualization of warping two signals using dynamic time warping.

distance, e.g., to compare a new signal to a set of reference signals in terms of global distances and select the most similar one, the warping path and thus backtracking is not necessary.

The computational complexity of DTW is $O(N \cdot M)$. However, some optimizations exist that either constrain the warping path [106, 107] or approaches that successively calculate the cost matrix from a coarse to a more fine grained resolution, reducing the computation time [108]. However, these approaches speed up the DTW calculation by a constant factor but still have a computational complexity of $O(N \cdot M)$ [108]. The space complexity of the DTW algorithm is $O(N \cdot M)$ when a backtracking of the warping path is necessary. Otherwise, the row or column wise filling of the cost matrix requires only memory space for the current and the previous row or column, respectively. Such an implementation has a memory complexity of $O(n)$, with n being the length of the signal along which the cost matrix is filled.

In the experimental implementation, which is introduced in Section 4.5, the DTW algorithm without aforementioned computation time optimizations but with linear memory complexity has been implemented. The computation time optimizations have not been applied, as the focus of the implementation lies in the evaluation of different parallelization approaches of recognition systems on multi-processor architectures. However, as the computation time optimizations of DTW are orthogonal to the parallelization approaches, no mutual dependencies regarding performance effects are expected.

4.3 Case Study

In order to apply and evaluate different parallelization approaches and their effect on throughput, latency, and other extra-functional properties, a hand gesture recognition system has been developed as a case study. This is due to the idea, that latency tends to play an even more important role in gesture recognition than activity recognition.

The system should detect hand gestures which are performed by a person. In order to capture hand movements, a sensor glove is worn by the user. Sensor signals are acquired by accelerometers attached to the fingertips of the glove. Multiple templates of gesture classes are recorded during training of the system. At run time, a sliding window of sensor data is compared to all templates based on DTW distance calculations. The gesture class of that template yielding the lowest distance to the sensor signal contained in the current window is selected as the classification result.

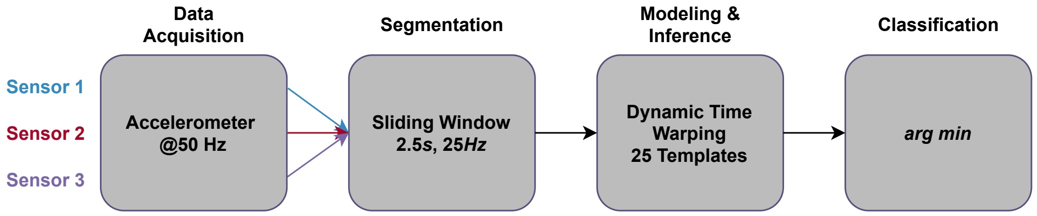


Figure 4.2: Activity recognition chain of the developed hand gesture recognition system.

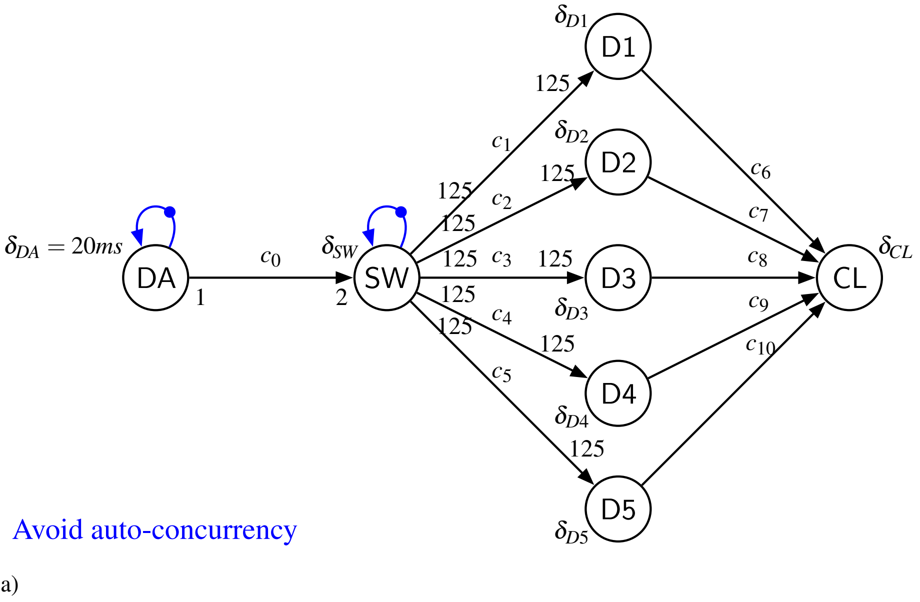
In Section 4.5.1, the recognition system will be described in more detail. However, the ARC structure is necessary to build a first application model, on which different parallelization strategies are explained. The corresponding ARC of the recognition system is depicted in Figure 4.2. Sensor data is sampled from three sensors with a sampling rate of $50Hz$. A sliding window lengths of 125 samples has been selected, which is shifted by two samples, corresponding to a window rate of $25Hz$. The DTW distances between sliding window and 25 templates will be calculated. The sliding window will be labeled with the class label of the template that has the smallest DTW distance. For the sake of comprehensibility, only five out of 25 templates will be assumed when parallelization approaches will be discussed in the following. The final application model however, will be presented in Section 4.5.3.

In Figure 4.3(a), the simplified application model is depicted as an SDF graph. The data acquisition of accelerometer sensors sampled at $50Hz$ is modeled by actor DA, with a self-edge and an execution delay of $20ms$. The sliding window segmentation is modeled by actor SW with a consumption rate of two, corresponding to the sample skip of the sliding window. The size of the windows of 125 samples equals the production rate on all outgoing channels of SW. To each window, the DTW distance to all templates will be calculated, which is modeled by actors D1-D5 representing the five DTW executions, each corresponding to a particular template. Note that these actors are modeled as parallel actors, as these can be subject to different parallelization approaches. Finally, actor CL receives the distances from each DTW execution and performs the *arg min* classification for the corresponding window.

This model is a simplification of the final application, but it includes all necessary properties to show the principle of different parallelization approaches.

4.4 Parallelization Approaches

In order to reduce latency or increase throughput of the DTW distance calculations compared to a sequential single-processor execution, multiple processor cores can be utilized in different parallelization approaches. In this chapter, possible parallelization strategies will be discussed and applied to the DTW distance calculation of the template-based hand gesture recognition system of the introduced case study. For the sake of comprehensibility, the preceding and succeeding ARC stages, namely segmentation and *arg min* evaluation, respectively, are performed on individual processor cores. To this end, an initiator-worker-evaluator structure is applied to the available processor cores. That is, one of the processor cores takes the role as *initiator*, one takes the role as *evaluator*, and the remaining processor cores, referred to as *worker cores*, perform the DTW distance calculations to all templates with different parallelization strategies. Hence, the segmentation is performed on the initiator core, the DTW distance calculations will be performed on the worker cores, and the distance evaluation and win-



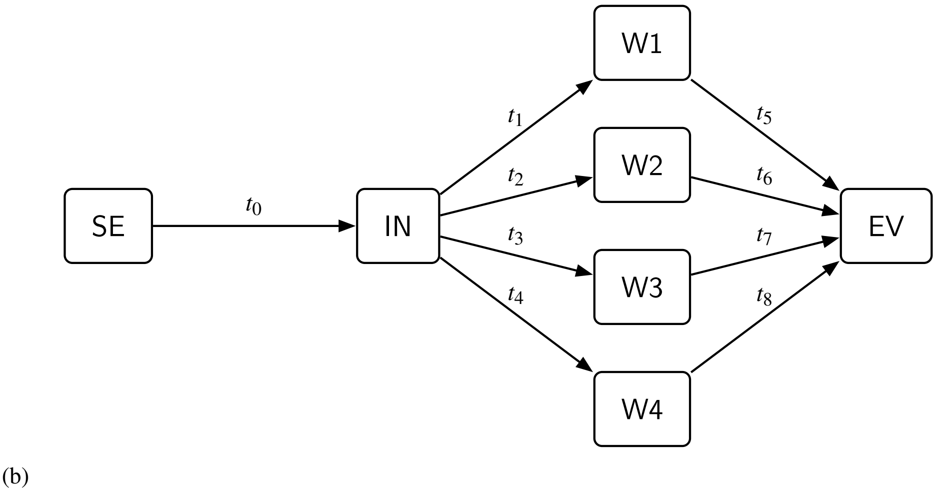


Figure 4.3: Reduced application model (a) and hardware model (b) of the hand gesture recognition system.

dow labeling will be performed on the evaluator core. In Figure 4.3(b), the initiator-worker-evaluator structure is depicted, with IN representing the initiator, W1-W4 the worker cores, and EV the evaluator.

4.4.1 Intra-Segment Parallelization

The first parallelization approach distributes the DTW calculations for a single segment of the sensor signal stream to multiple processors cores, in order to reduce execution time. Thus, the templates are distributed over all worker cores, calculating their DTW distances to the sliding window in parallel. This parallelization approach, that can be considered as task-level parallelization, has already been introduced by Yoder and Siegel [88] as "Serial-Parallel" approach for DTW in particular.

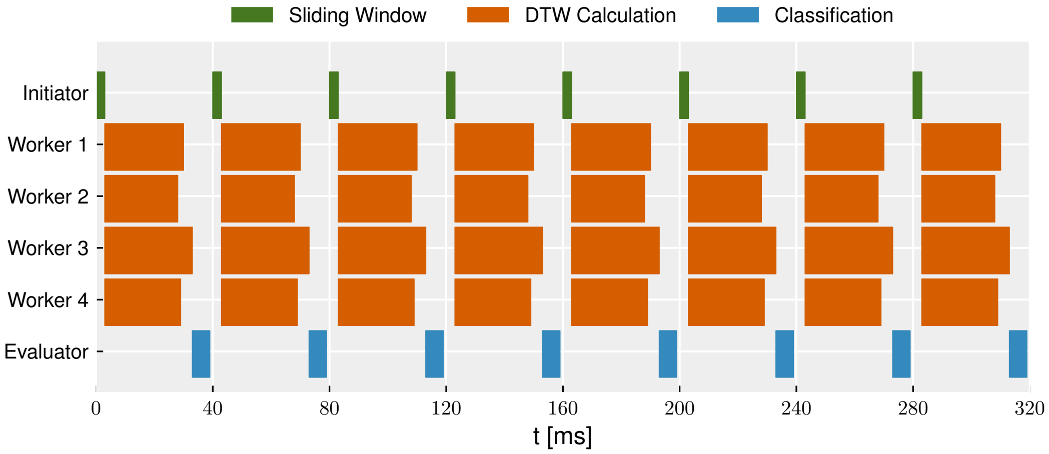


Figure 4.4: Visualization of the intra-segment parallelization approach.

Furthermore, Yoder and Siegel compare this approach to other instruction-level parallelization approaches on Single Instruction, Multiple Data (SIMD) and application-specific Very Large Scale Integration (VLSI) processors for DTW-based algorithms. However, the focus of this chapter is on parallelization approaches exploiting task- and data-level parallelism at a system-level, as this can be exploited by algorithm distribution on multi-processor hardware, without the need for ASICs. Therefore, only the Serial-Parallel approach from [88] is considered for investigation in the thesis at hand. In [94], the same approach has been introduced to reduce latency in video processing applications and applied to video-based activity recognition in [95]. In their work, it is referred to as "intra-frame parallelization" [95]. Due to the broader sense of segments, this approach is referred to as *intra-segment parallelization* in this thesis. In general, the same approach can be applied to a variety of sensor-based activity recognition algorithms. The instances to be distributed can either be class models, i.e., HMMs [51], a database of feature vectors, e.g., for k-NN classifiers, parallel decision tree implementations [93], or templates in template-based gesture recognition. The latter, realized by a DTW-based approach, was selected in this thesis, in order to apply model-based design, parallelization, and analysis approaches and evaluate the model accuracy. For these evaluations recognition accuracy is of minor concern.

For the selected case study of DTW-based gesture classification, the templates for all gestures are distributed statically among the set of worker cores. That way all templates are partitioned into a number of blocks matching the number of worker cores. The initiator sends the acceleration signals of the current sliding window to all worker cores. Each worker core calculates the DTW distances between the templates which have been assigned to it and the signal contained in the window. After each worker core has finished its DTW calculations, the distances are sent to the evaluator for comparisons. The evaluator then selects the class label from the template that has the smallest distance to the sliding window among all templates. This comparison is started as soon as the distance information of all worker cores have been received by the evaluator. In Figure 4.4 this process is depicted as a Gantt chart.

Since calculating the DTW distances of all templates to the current sliding window is performed in parallel, the response time is expected to be reduced compared to a serial computation. However, the DTW computation time for each template to the sliding window scales with $N_{SW} \cdot N_T$, where N_{SW} is the number of samples of the sliding window and N_T is the number of samples of the template. Although N_{SW} is constant for each calculation, N_T can differ for each template. Therefore, the execution time depends linearly on the template size. Since the distribution of templates among a number of cores not necessarily leads to an even distribution in accumulated template size, the response time to complete the classification for a single sliding window depends on the core with the highest accumulated size of assigned template gestures. This is depicted in Figure 4.4 as well. Consequently, the speedup of classifying a single sliding window with this parallelization approach will be less than the number of cores W. Additionally, as pointed out by Yoder and Siegel in [88], the aforementioned parallelization approach can only be effective with $W \leq M$ worker cores, with W being the number of worker cores and M being the number of templates. With more than M worker cores, $W - M$ worker cores would remain idle.

Modeling From a modeling perspective, this approach is a straightforward mapping of the DTW instances (templates) to the available worker cores. As depicted in Figure 4.3, the example application consists of five templates and thus five DTW instances that can be executed in parallel. Furthermore, there are four worker cores to which individual DTW instances can be assigned to. In Figure 4.5 an example mapping including a static order schedule is shown. Note that processor cores W1 to W4 are color coded w.r.t. the sliding window that they are processing. When applying intra-segment parallelization, all processors are coded in the same color, as they are processing data from the same sliding window concurrently. The idea behind color coding will get more obvious with the following parallelization approaches.

Since in this example, there are more DTW instances (templates) than available worker cores, at least one worker core has to be assigned with more than one DTW instance. In Figure 4.5, this is worker core W4 to which D1 and D2 are assigned. For the scheduling of processor cores to which only a single SDF actor is assigned, a self-edge is sufficient to avoid overlapping executions of different actor firings. For multiple actors mapped to the same processor core, i.e., actors D1 and D2 on W4, a static order schedule is achieved by creating a dependency through a scheduling edge between them and a feedback edge with one initial token.

In Figure 4.5 it can be seen that actor CL can only fire, after all DTW actors have finished their executions on their assigned processor cores. Therefore, the efficient distribution of actors to worker cores is of major importance w.r.t. the latency.

4.4.2 Inter-Segment Parallelization

A second approach is based on a parallel computation of subsequent segments, i.e., windows, on different worker cores. This parallelization approach, that can be considered as data-level parallelization, has been introduced by Pillai et al. [94] as "inter-frame parallelization" for video processing applications including video-based activity recognition. As it is applied to segments in a broader sense, it is referred to as *inter-segment parallelization* in the thesis at hand.

For this approach, each worker core performs a sequential processing of an entire sliding window. Instead of distributing the DTW distance calculations of a single sliding window onto multiple cores, consecutive sliding windows are processed in a cyclo-static manner on different cores. This approach is useful in systems where the classification of a single sliding window can not be efficiently distributed to multiple cores but the execution on a single core does not meet the necessary throughput for the desired

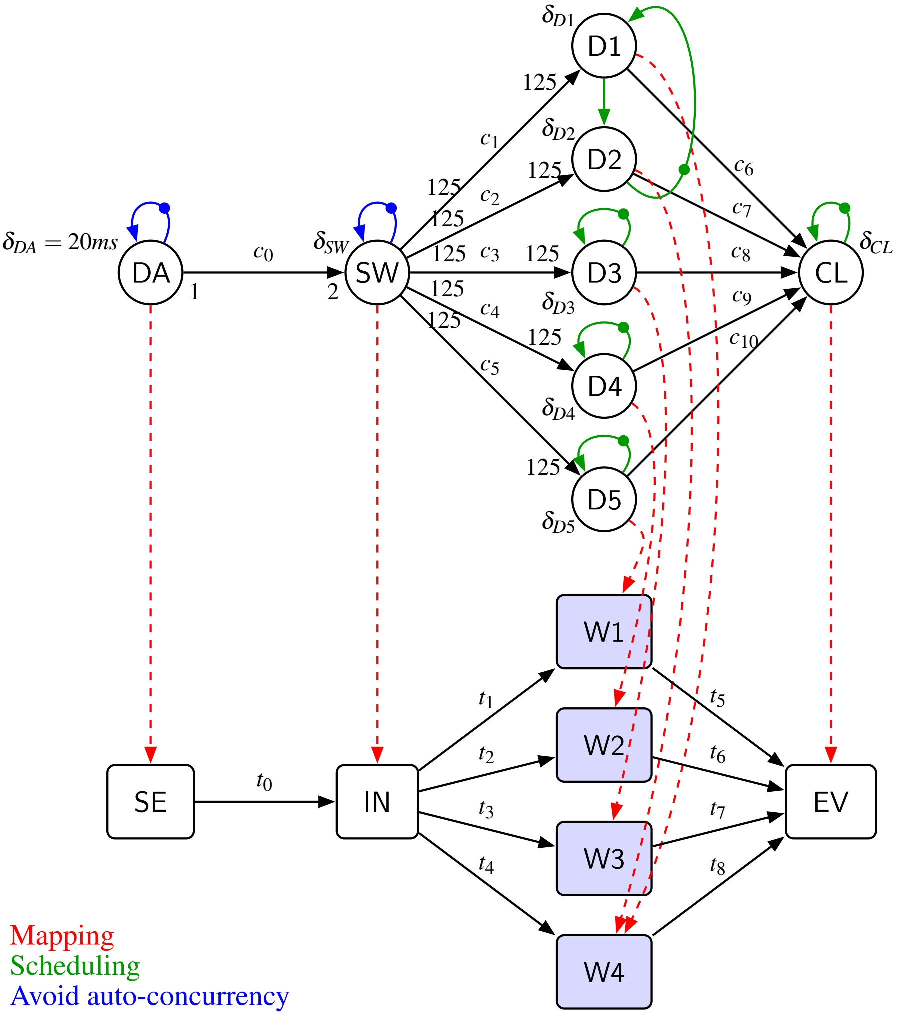


Figure 4.5: Mapped and scheduled application graph with intra-segment parallelization.

application, i.e., the execution time of an entire sliding window processed on a single core exceeds the period between consecutive sliding windows. While the current sliding window is processed by one of the worker cores the succeeding sliding window is processed by another worker core. Figure 4.6 depicts this concept.

For the selected case study, each worker core contains all templates and calculates their DTW distances to the sensor signal in the sliding window that it receives. When a new sliding window arrives at the initiator it will be sent to the next free worker core. This will lead to a cyclo-static behavior of window processing if enough worker cores are used for real-time performance. In contrast to intra-segment parallelization, this approach can also be effective with $W > M$ worker cores, with W being the number of worker cores and M being the number of templates. Although the classification latency is not reduced by this approach compared to a single-core implementation, the throughput of the system can be increased to meet real-time performance.

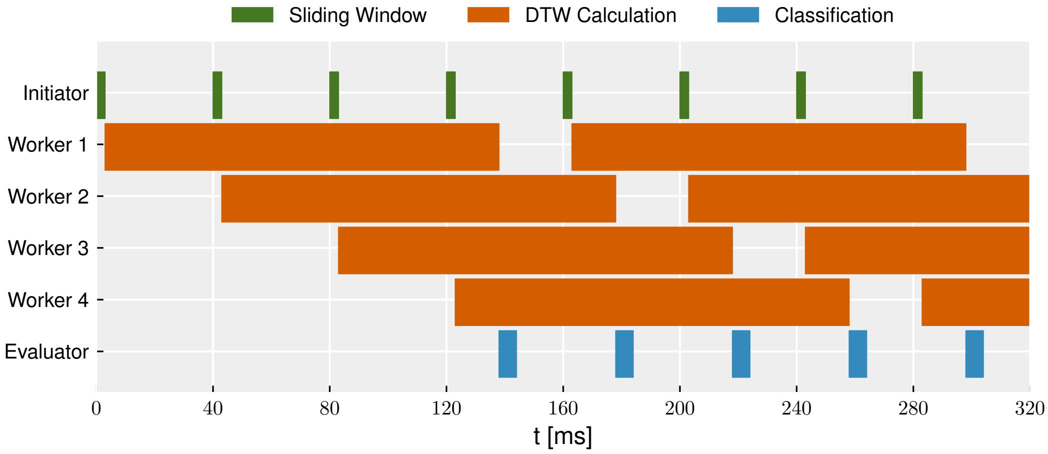


Figure 4.6: Visualization of the inter-segment parallelization approach.

Modeling In contrast to intra-segment parallelization, all DTW actors of the dataflow graph are mapped to a single worker core. For the sake of readability in the following steps, all actors D1-D5 are mapped to the same worker core and are abstracted into a single actor D with accumulated execution times. The resulting model is presented in Figure 4.7.

In order to model the cyclo-static processing of sliding windows, a technique called partial expansion can be used which is described in [109] by Tran et al. Partial expansion is used to expose task and data parallelism in SDF graphs. To fully expose parallelism, an SDF graph can be expanded into its equivalent HSDF graph. However, the transformation can lead to an exponential growth in the number of actors and edges of the original SDF graph. Furthermore, a full expansion does not take into account the number of available processing elements. In order to control the degree of parallelism and the actors to be expanded, partial expansion of selected actors can be performed by the designer or a design tool [109]. In contrast to other approaches [99, 110], the partial expansion technique described by Tran et al. comes without the need for additional split/join actors or buffer managers.

The application graph in Figure 4.7 shall be partially expanded towards a parallel processing of consecutive sliding windows. Therefore, actor D can be expanded to a degree which corresponds to the number of available processor cores. Since Tran et al. indicated a poor trade off between buffer sizes and performance gains for expansion rates higher than the number of available processor cores, the expansion rate for the example at hand is chosen to match the number of available processor cores, i.e., four. As a result, actor D is expanded to four instances, i.e., D_a, D_b, D_c, and D_d. Actor SW is converted

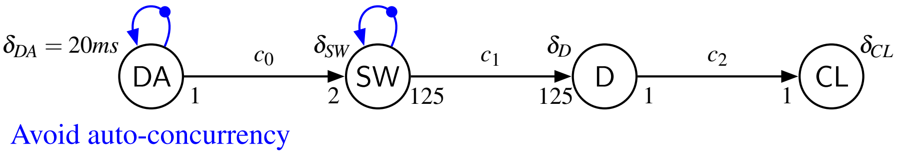


Figure 4.7: Application graph of the hand gesture recognition with actor D representing a sequential processing of an entire sliding window by all DTW instances.

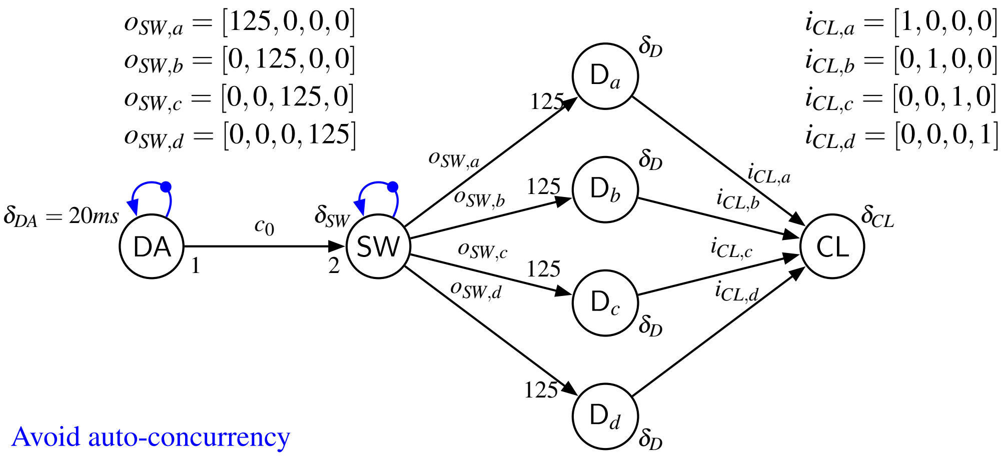


Figure 4.8: Partially expanded application graph of the hand gesture recognition with inter-segment parallelization.

into a CSDF actor, cyclo-statically producing its 125 tokens of a sliding window to one of the actors $\mathsf{D}_a,\ldots,\mathsf{D}_d$. Likewise, actor CL is converted into a CSDF actor, consuming cyclo-statically from one of its incoming channels from the actors $\mathsf{D}_a,\ldots,\mathsf{D}_d$. This expansion corresponds to the combination of cases 2 and 4 in [109] as actor D is consumer and producer with equal production and consumption rates on each of its incoming and outgoing channels. The resulting graph is shown in Figure 4.8.

Due to the partial expansion, the graph iteration has been unfolded by four w.r.t. the repetition vector entries for DA, SW, and CL, whereas in each fold one of the actors D_a to D_d fires (in cyclo-static order) with its original repetition vector entry. After partial expansion of actor D, each expanded instance can be mapped to a different worker core. The resulting mapped and scheduled CSDF graph is shown in Figure 4.9. Note that the color coding of the processor cores indicates, that each processor core is processing a different sliding window. The sliding window x will be processed on processor core W4, while $x+1$ is processed on W3, $x+2$ on W2, $x+3$ on W1, and $x+4$ on W4 again.

4.4.3 Hybrid Parallelization

Finally, as indicated by Chen et al. [94], a hybrid of the two aforementioned parallelization approaches is possible. This can be beneficial, if the classification of a single sliding window cannot be parallelized to the same degree as number of available processor cores. In [G6], this approach has been introduced by utilizing the concept of tiles. To this end, the available processor cores are divided into groups of cores referred to as tiles. The size of a tile should be a number of processor cores to which a single sliding window classification can be efficiently distributed. If this distributed processing reduces execution time that is still higher than the time between two sliding windows, the succeeding window can be processed on another tile. This way, intra-segment parallelization is applied within a tile to reduce latency and inter-segment parallelization is applied between tiles to increase throughput of the system.

In general, the processor cores within a tile not necessarily need to be homogeneous. Furthermore, the number and type of processor cores not necessarily needs to be equal among different tiles. However,

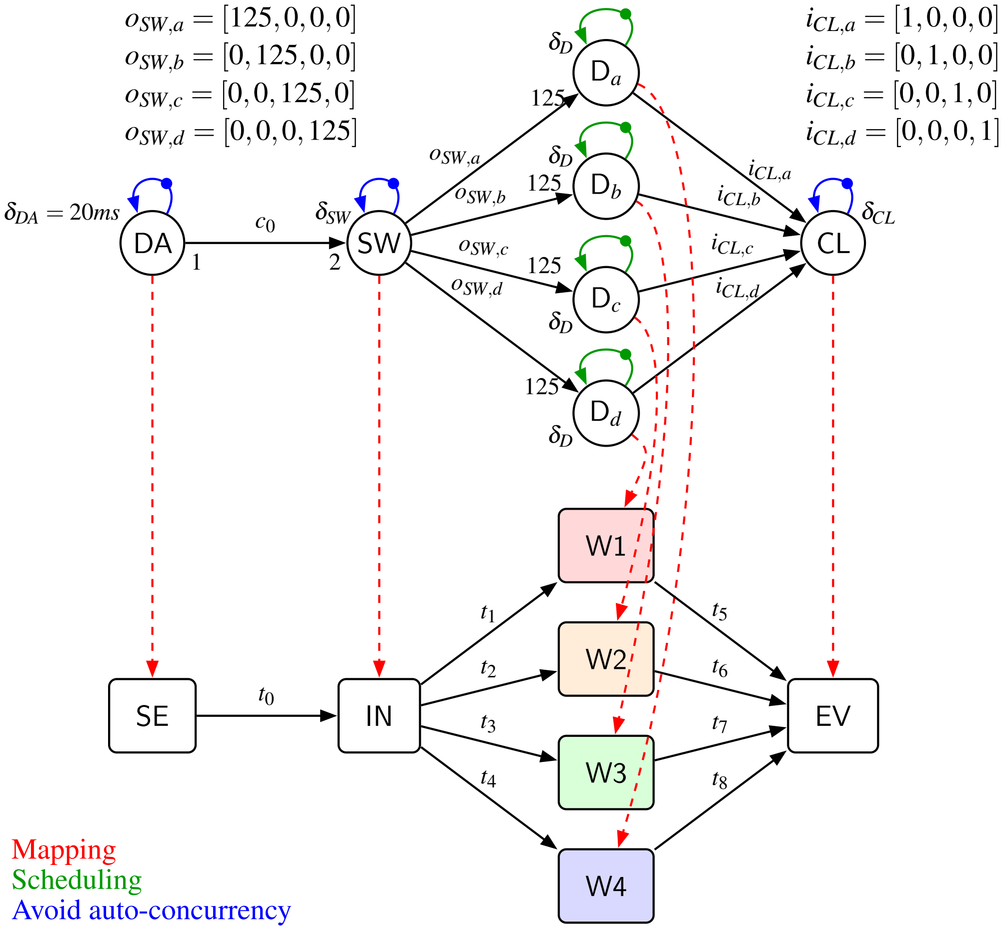


Figure 4.9: Mapped and scheduled application graph with inter-segment parallelization.

for the sake of comprehensibility only equally composed tiles of homogeneous groups of processor cores are considered in the remainder of this chapter. Note that this implies equal communication channels between processor cores within a tile and among different tiles. This way, the classification of a single sliding window can be expected to have the same execution time on any of the available tiles, given an equal software distribution within all tiles.

In order to apply the hybrid parallelization approach for the selected case study, the available worker cores are grouped into K equal tiles. A tile is responsible for the parallel processing of a sliding window. To this end, the templates are distributed onto the worker cores of a tile. While one tile processes one sliding window at a time, the succeeding sliding window will be dispatched to the next available tile in the system. Figure 4.10 depicts this concept.

Modeling From a modeling perspective, the initial synchronous dataflow graph in Figure 4.3 is mapped to the hardware architecture, corresponding to intra-segment parallelization. However, instead of mapping the actors D1 - D5 to all available worker cores, they are only mapped to the processor cores of a single tile. In Figure 4.11, an example mapping to the tile composed of processors W3 and W4 is depicted. Actors D1 and D2 are mapped to W4 and actors D3, D4, and D5 are mapped to W3.

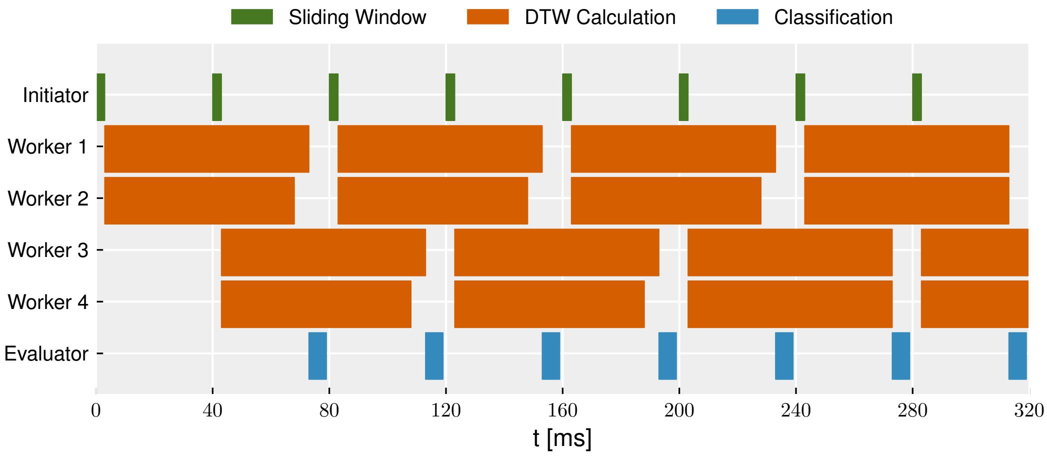


Figure 4.10: Visualization of the hybrid parallelization approach with 2 tiles.

An example schedule is shown in green. The resulting graph represents intra-segment parallelization for a single tile. For the sake of comprehensibility, in the following steps actors D1 and D2 are merged into actor D1' which is mapped to W4 and actors D3, D4, and D5 are merged into actor D2' which is mapped to W3. Similar to the modeling of inter-segment parallelization, actors D1' and D2' will be expanded by an expansion rate, equal to the number of tiles. In this example, the worker cores W1-W4 are grouped into two tiles. Thus, the SDF graph will be expanded by an expansion rate of two. The second fold of the expanded part is then mapped to the remaining tile, processing every other sliding window. The resulting CSDF graph is depicted in Figure 4.12.

4.4.4 Parallelization Configurations

Without loss of generality, all of the aforementioned parallelization approaches are deployed by tiling the available processor cores. Furthermore, only homogeneous symmetrical tiles are considered, such that the number of processor cores divided by the number of tiles results in an integer number of cores per tile. This is not generally necessary, but eases comprehensibility and deployment of different parallelization approaches in the experiments. In order to distinguish different parallelization approaches, each is identified by the number of deployed tiles N_K, which will also be referred to as the parallelization *configuration* CN_K, e.g., configuration C2 is deployed with two tiles: $N_K = 2$. To this end, the number of processor cores per tile N_{W_K} is determined by the number of available processor cores N_W divided by the number of tiles N_K, i.e.:

$$N_{W_K} = \left\lfloor \frac{N_W}{N_K} \right\rfloor .$$

Generally, the result is rounded down in case N_W is not a multiple of N_K or N_{W_K}. However, in the experiments of the thesis at hand, for each evaluated configuration $N_{W_K} \cdot N_K = N_W$ holds. Each configuration is considered as an realization of the hybrid parallelization approach with different number of tiles. Thereby, the configuration $N_W = N_{W_K}$ and thus $N_K = 1$ inherently represents intra-segment parallelization, and $N_W = N_K$ and thus $N_{W_K} = 1$ represents inter-segment parallelization, as those are the extremes of the hybrid approach.

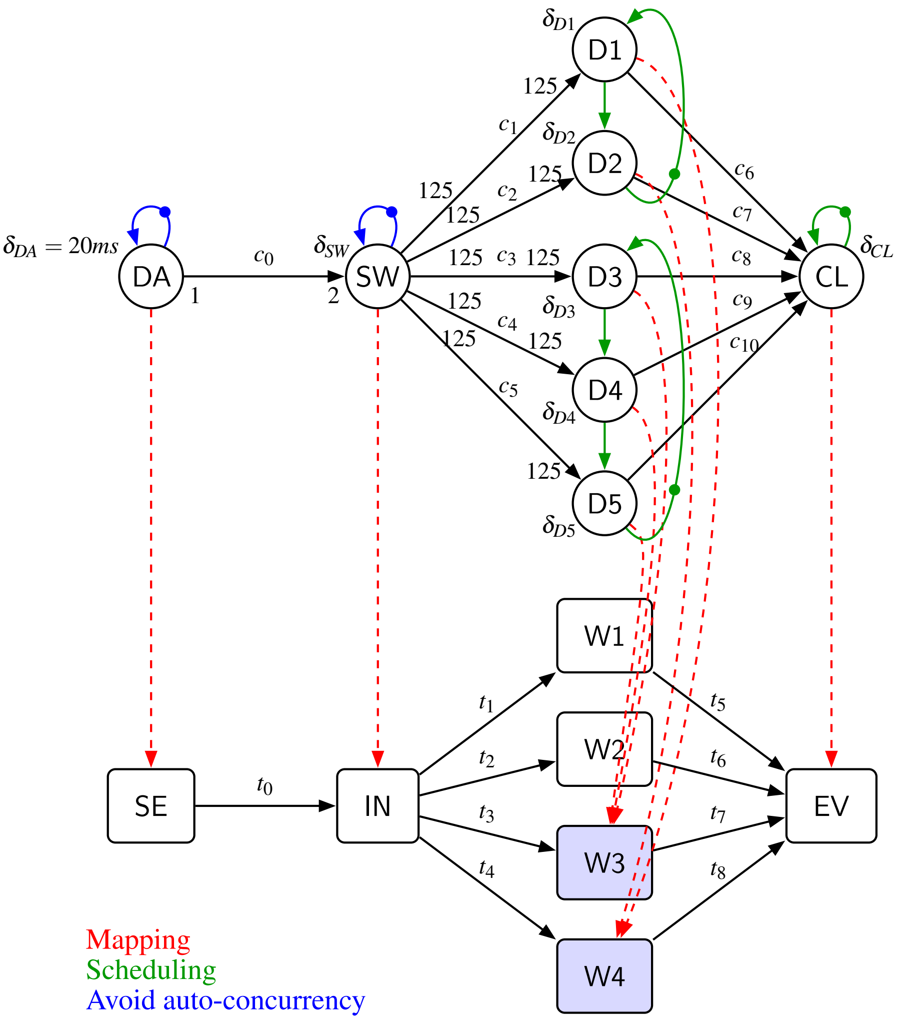


Figure 4.11: Mapped and scheduled application graph of the hand gesture recognition application to a multi-core tile.

4.5 Experiments

In order to evaluate the accuracy of the dataflow models, the hand gesture recognition system has been implemented in the presented parallelization approaches. In the following section, the hand gesture recognition system will be explained in more detail.

4.5.1 Hand Gesture Recognition Application

The system was designed to recognize five different gestures performed with a sensor glove. To this end, a sensor glove fitted with BNO055 [111] sensors at the fingertips has been used to capture 3D acceleration of the fingers. Figure 4.13 shows the sensor glove.

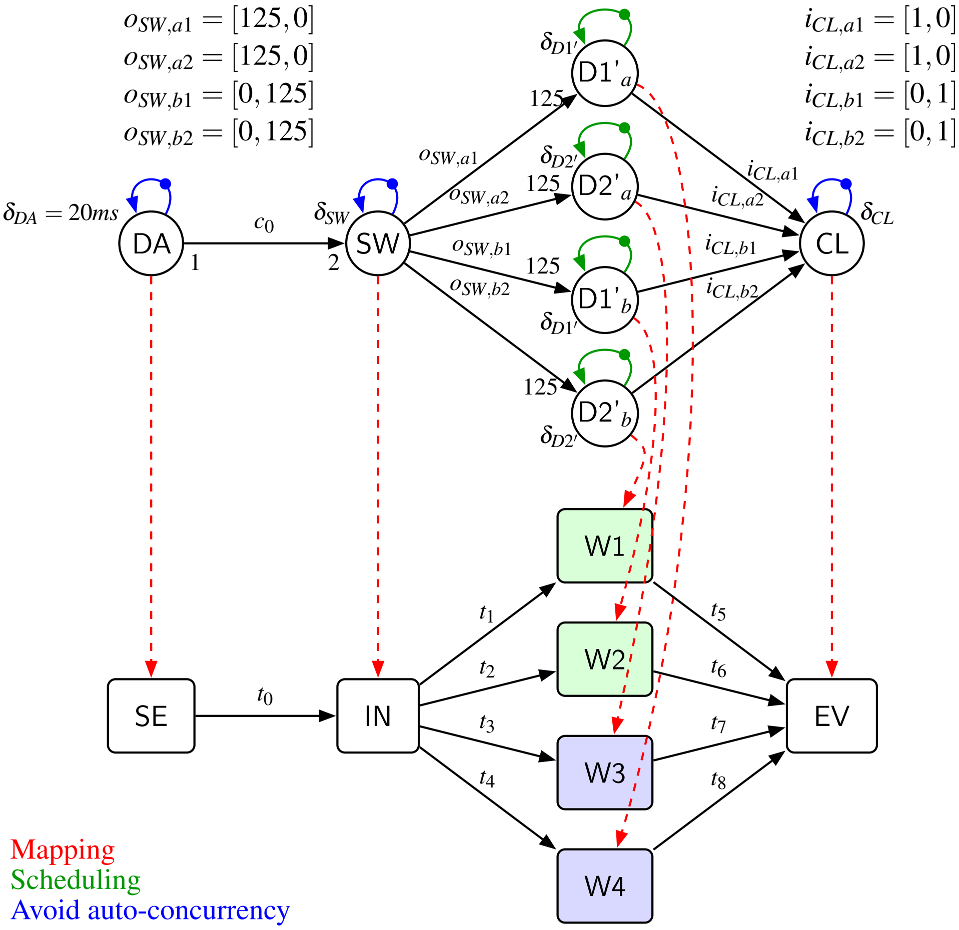


Figure 4.12: Mapped and scheduled application graph with hybrid parallelization.

Five gestures have been defined that should be recognized from the captured acceleration signals. The gestures to be detected have been defined as *swipe left* and *swipe right* with two fingers, drawing a circle *clockwise* (CW) and *counter clockwise* (CCW) with the index finger and a *grasp* motion. In order to reduce dimensionality of the data stream, only the sensors attached to the thumb, index finger, and middle finger are used as they are expected to have the least redundant information among all fingers. Each accelerometer samples sensor data at $50\,Hz$ in a range of $\pm 4\,G$ with a fixed-point resolution of $14\,bits$. However, each sensor sample is converted into a single-precision floating-point value ($4\,bytes$) per axis. This leads to a 3D sensor sample size of $12\,bytes$ per finger and $36\,bytes$ in total.

For the sake of reproducibility, training and test sequences have been recorded offline. For each defined gesture, five templates have been recorded, resulting in 25 templates ranging from 21 to 123 samples ($0.42s$ to $2.46\,s$, respectively) in total. A *test dataset* for evaluating recognition accuracy and model accuracy has been recorded, which consists of 19 test sequences with a total of $28,876$ samples and an average of 7.5 performed gestures per sequence. In total 144 test gestures have been recorded. The distribution of gesture classes in the test dataset can be found in Table 4.1. Furthermore, a *training dataset* of gestures has been recorded in order to train parameters for gesture segmentation, that will be introduced later in this section. The training set has also been used to acquire model annotations

Table 4.1: Training and test dataset composition.

Class	Training Dataset		Test Dataset	
	Occurrences	Ratio	Occurrences	Ratio
Grasp	12	22 %	34	24 %
Swipe Left	11	20 %	28	19 %
Swipe Right	11	20 %	34	24 %
Circle CCW	11	20 %	21	15 %
Circle CW	10	18 %	27	19 %

by measuring particular code sections, which will be discussed in Section 4.5.3. The training dataset consists of seven training sequences. In total 55 performed gestures have been recorded in the training dataset. The distribution of gesture classes in the training dataset can be found in Table 4.1 as well.

The classification of gestures is based on calculated DTW distances. To this end, test sequences are segmented into sliding windows of 125 samples in length ($2.5\,s$), in order to fit the largest among the templates. The sliding window is shifted by two samples. All windowed sensor signals are compared to all 25 templates. Therefore, a DTW procedure for 3D signals with a linear memory utilization w.r.t. window length has been implemented to reduce memory footprint. Note that this is sufficient as only the final distance between window and template is necessary instead of the actual warping path information. The final distances are normalized with the template lengths, in order to reduce their influence on the DTW distances. Although a scaling with the length of the actual warping path might have been a more accurate scaling method, the implemented method was selected in favor of a linear memory complexity. To this end, the DTW distance between a sliding window and a template is calculated for each finger individually and accumulated over all three fingers into a single

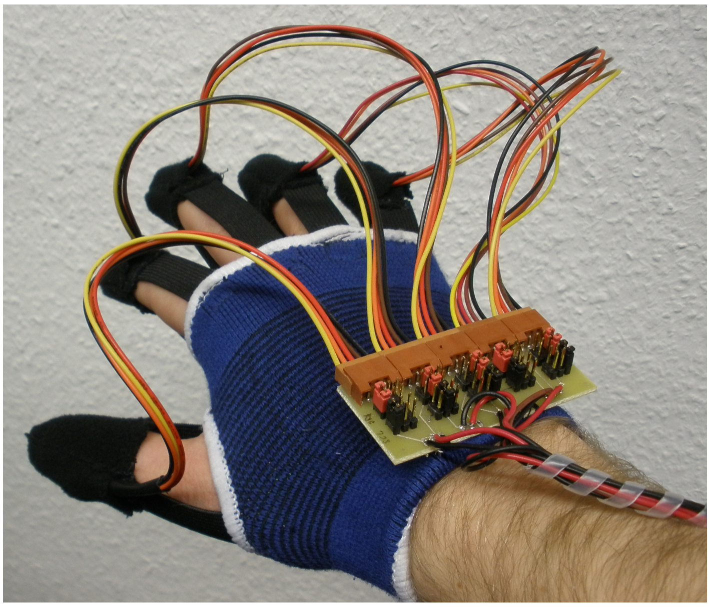


Figure 4.13: Sensor glove with 5 BNO055 sensors mounted at the fingertips.

Table 4.2: Empirically determined class-dependent DTW distance thresholds.

Class	DTW distance threshold
Grasp	1.20
Swipe Left	2.50
Swipe Right	2.00
Circle CCW	2.20
Circle CW	1.75

distance value. The calculated distances to all templates are recorded for the particular window. In order to account for a possible *null* class, class-dependent thresholds on the DTW distance have been empirically determined by experiments. The distance of each template to the window is compared to its class-dependent threshold and discarded in case of exceedance. An *arg min* evaluation selects the class label among the remaining distance values of a window. The selected class-dependent distance thresholds are listed in Table 4.2.

Since a gesture can span over multiple sliding windows, the begin and end of a performed gesture needs to be detected. Therefore, the fluctuation of sensor data within each window is calculated as proposed by [112]. To this end, each sensor sample is considered as a 9-dimensional vector $\vec{v}[t]$ at time t. The squared euclidean norm of the derivative signal, that is an element-wise subtraction of subsequent sensor samples, is calculated by: $\eta[t] = ||\vec{v}[t] - \vec{v}[t-1]||_2^2$. Here, $||\vec{x}||_2$ denotes the euclidean norm of a vector $\vec{x}$. This squared euclidean norm $\eta[t]$ is then averaged over a window by an Exponential Moving Average (EMA) filter $EMA[t] = (1-\alpha) \cdot EMA[t-1] + \alpha \cdot \eta[t]$ with a smoothing factor of $\alpha = 0.2$ as recommended by [112]. Although [112] used this approach without sliding windows, but on the continuous stream of sensor data to detect the begin and end of an gesture by thresholds, this can also be implemented on a window-based segmentation. Using fixed sized sliding windows, the processing time of calculating DTW for a particular sliding window is independent of the length of the performed gesture. However, in contrast to [112], the EMA in a window-based processing detects the begin and end of a gesture among multiple windows. Furthermore, since the EMA emphasizes fluctuations in more recent sensor samples, a higher correlation between actual and detected gesture start and end points is expected compared to an evely weighted average.

The detection of gesture start and end is finally performed by an *activation threshold* and a *release threshold* on the EMA value of an entire window. The activation threshold, which is greater than the release threshold, has to be exceeded by the EMA to detect the start of a gesture. The end of a gesture is detected when the EMA falls below the release threshold. The resulting hysteresis effect is intended to avoid false positives and false negatives at the transitions. The values for activation and release threshold have been empirically determined by experiments as 20.0 and 2.0, respectively.

After calculating the DTW distances to all templates and the EMA of a window, the class assignment is performed in the following way: If the EMA value indicates no gesture or none of the DTW distances falls below its class-dependent distance threshold, the window is labeled as null. Otherwise, the *arg min* evaluation selects the class label among the remaining distance values that fall below their class-dependent distance threshold.

The aforementioned recognition system can be optimized in many ways regarding computation time and recognition performance. Especially the latter is necessary for a practical application of the system. This however is beyond the scope of the implementation, as it solely serves as an experimental implementation for the evaluation of model accuracy in different parallelization approaches. Nevertheless, one optimization that gets immediately obvious, is the disabling of DTW calculations for windows without signal fluctuations (and thus, no gestures to be classified), as detected by the EMA. This however, imposes data-dependent behavior which cannot be captured by CSDF graphs. The modeling and analysis of HAR systems with dynamic behavior is discussed in Chapter 5.

4.5.2 Hardware Architecture

In order to facilitate parallel processing, the described DTW-based recognition system has been implemented on a multi-processor system in different parallelization configurations. As a hardware platform, the Texas Instruments (TI) TMS320CC678 8-core fixed- and floating-point digital signal processor [113] has been chosen. The TMS320C6678 features 8 homogeneous DSP cores with clock rates of $1.0\,GHz$, $1.25\,GHz$, or $1.4\,GHz$.

Each DSP core has $32\,KB$ of first level Static Random-Access Memory (SRAM) for program (L1P) and data (L1D), each, which can be partially or fully configured as cache. The SRAM that is not configured as cache is used as scratchpad memory. Supported L1P and L1D cache sizes are $0\,KB$, $4\,KB$, $8\,KB$, $16\,KB$, and $32\,KB$. When configured as such, the L1P cache is direct mapped, while L1D is two-way set associative. Furthermore, each processor core features $256\,KB$ second level SRAM for data (L2D), that can be fully or partially configured as four-way set associative cache in sizes of $0\,KB$, $32\,KB$, $64\,KB$, $128\,KB$, $256\,KB$, or $512\,KB$.

Additionally, all processor cores share $4\,MB$ of so-called Multicore Shared Memory (MSM) SRAM. The MSM can either be configured as L2 shared SRAM which can only be cached by L1 caches or configured as L3 shared SRAM which can also be cached by L2 caches. As development platform the TMDSEVM6678L evaluation board has been used, which additionally provides $512\,MB$ shared L3 DDR3 RAM which is, together with the MSM, managed by the Multicore Shared Memory Controller (MSMC).

The so-called Multicore Navigator offers 8192 multipurpose hardware queues and a queue manager as well as a Direct Memory Access (DMA) unit for zero-overhead data transfers. The communication between processor cores, MSMC, Multicore Navigator, and other peripherals is realized by a Network-On-Chip (NoC) based on a non-blocking switch fabric, called TeraNet.

For an overview see Figure 4.14. Implementation details on the hand gesture recognition system on the TMS320C6678 architecture can be found in Appendix A.

4.5.3 Analysis Model

The Application model of the hand gesture recognition system for the experiments is shown in Figure 4.15(a), together with the hardware model in Figure 4.15(b). In order to analyze different mappings and schedules the application model is annotated with mapping edges. Accordingly, execution times depending on the underlying hardware component are attached. For the experiments, all actor execution times have been measured from an actual implementation on the target hardware. Furthermore a schedule has to be added to the application model. The resulting model represents the analysis model.

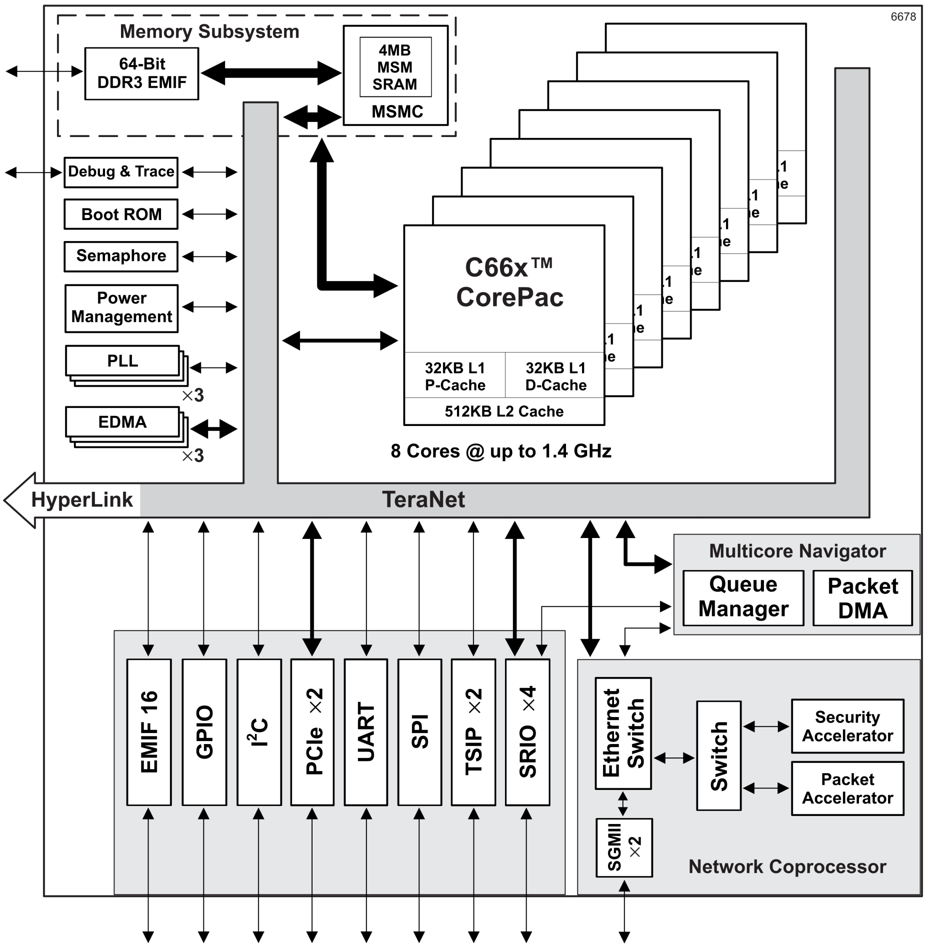


Figure 4.14: Block diagram of the TMS320C6678 multi-core DSP architecture by Texas Instruments. From [113].

Note that mapping edges are only used as an intermediate step in the construction of the analysis model, and are not part of the resulting dataflow graph to be analyzed.

Four representative generic mappings and schedules have been chosen for the experiments. In each configuration, actor DA is mapped to the sensor SE, the sliding window segmentation is mapped to the initiator core IN, and the evaluation is mapped to the evaluator core EV. The differences between the four configurations regarding the mapping are mainly the different distributions of DTW instances to the available processor cores of a single tile, while a tile consists of different processor cores in each configuration as described in Section 4.4.4.

Since the gesture recognition system is performing 25 DTW distance calculations, the application graph consists of 25 DTW actors D1-D25, which generally could be executed in parallel. Note that the actual DTW calculation is solely depending on the length of the templates and the sliding window. However, in the experiments the sliding window has a fixed size of 125 samples. Therefore, the actual DTW execution time in the experiments is directly depending on the template sizes. The

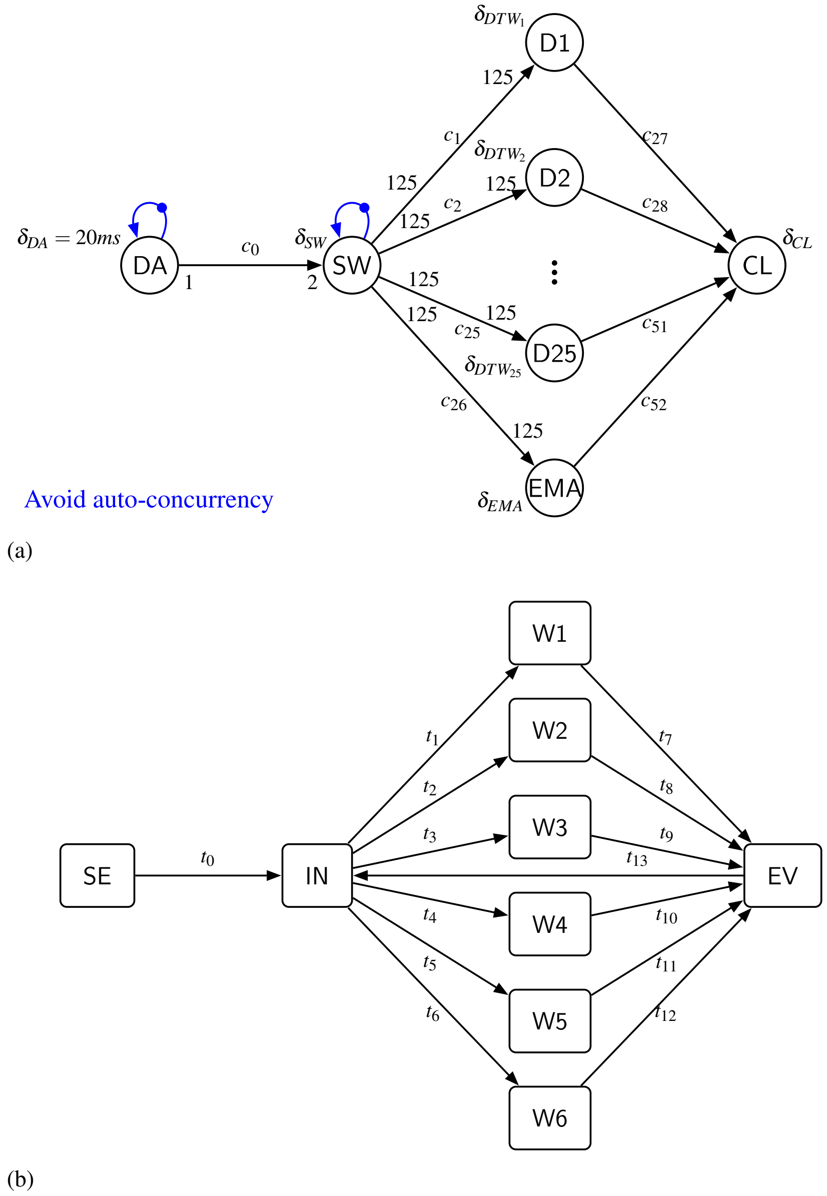


Figure 4.15: Application model (a) and hardware model (b) of the hand gesture recognition system.

DTW calculation of a particular template is referred to as a DTW instance. As an example, the DTW calculation for template x will be referred to as DTW_x, and likewise the corresponding execution time will be referred to as δ_{DTW_x}. Additionally, the EMA calculation can be assigned in parallel to the DTW instances as well. Thus, in theory all actors D1-D25 and EMA can be processed in parallel in the application model. However, during the mapping, actors are assigned to available worker cores of the system, i.e., W1-W6, and a sequential static order schedule is constructed. Since finding the optimal mapping involves a selection among $6^{26} \approx 1.7 \cdot 10^{20}$ different mapping options, and a design space exploration is out of scope for the thesis at hand, a simple heuristic has been used to select a mapping for each configuration.

Table 4.3: Actor mapping of the application model to the hardware architecture in all four configurations.

Actor	Configuration				Template Size $[Samples]$
	C1	C2	C3	C6	
DTW14	W1	W1	W1	W1	189
DTW23	W1	W1	W1	W1	1,107
DTW10	W1	W1	W1	W1	279
DTW22	W1	W1	W1	W1	918
DTW13	W2	W1	W1	W1	306
DTW21	W2	W1	W1	W1	864
DTW15	W2	W1	W1	W1	315
DTW17	W2	W1	W1	W1	810
DTW11	W3	W2	W1	W1	315
DTW20	W3	W2	W1	W1	783
DTW12	W3	W2	W1	W1	342
DTW2	W3	W2	W1	W1	720
DTW1	W3	W2	W1	W1	360
DTW18	W4	W2	W2	W1	702
DTW9	W4	W2	W2	W1	387
DTW25	W4	W2	W2	W1	693
DTW6	W4	W2	W2	W1	459
DTW16	W5	W3	W2	W1	675
DTW4	W5	W3	W2	W1	549
DTW19	W5	W3	W2	W1	612
DTW7	W5	W3	W2	W1	477
DTW24	W6	W3	W2	W1	585
DTW8	W6	W3	W2	W1	513
DTW3	W6	W3	W2	W1	459
DTW5	W6	W3	W2	W1	522
EMA	W6	W3	W2	W1	/
DA	SE	SE	SE	SE	
SW	IN	IN	IN	IN	
CL	EV	EV	EV	EV	

To this end, the templates, and thus the DTW instances are first sorted is descending order w.r.t. their number of samples and then reordered by subsequently selecting instances from the top and bottom of the sorted list in an interleaved manner, starting with the longest template. The EMA is appended to the reordered list. The reordered list can be found in the first column of Table 4.3 along with the corresponding template sizes in the last column. The mapping of each configuration is then defined by a generic mapping function taking the number of cores per tile N_{W_K} of a particular configuration as a parameter. Furthermore, the function takes the index $j \in [0, N_{W_K} - 1]$ of a worker core within a tile and returns a closed interval $[s,t]$ which specifies the indices (starting at zero) within the reordered list of DTW instances that are mapped to a worker core j, with:

$$[s,t] = \left[\left\lfloor j \cdot \frac{N_D}{N_{W_K}} + 0.5 \right\rfloor, \left\lfloor (j+1) \cdot \frac{N_D}{N_{W_K}} + 0.5 \right\rfloor - 1 \right], \tag{10}$$

where N_D is the number of DTW instances (templates) and N_{W_K} is the number of cores per tile of that configuration. The last worker core $W_{N_{W_K}-1}$ of a tile additionally gets EMA appended to its mapping list. For the sake of readability, the mapping edges have been omitted in Figure 4.15. Instead, the mapping for each configuration is listed in Table 4.3.

For each particular mapping, a generic schedule has been selected. Since in the experiments all of DTW and EMA actors could start at the same point in time, i.e., when a new window has arrived, and the results are only sent to EV when the last DTW or EMA instance has finished, the processing order on a particular worker core has no impact on the total latency of a sliding window. Therefore, a simple static order schedule from s to t in ascending order has been implemented, with $[s,t]$ calculated from Equation (10). Additionally, EMA is executed at last on the last worker cores $W_{N_{W_K}-1}$ of each tile.

4.5.4 Model Refinement

To account for timing on the selected target hardware, a model refinement has been performed for the analysis models, which will be explained in the following. To reduce model complexity, all actors mapped to the same worker core are merged into a single actor. Each chosen configuration for the experiments is utilizing six worker cores, either as a single tile (configuration C1) or as multiple tiles as discussed in Section 4.4.4. Therefore, each analysis model has exactly six actors D1-D6 representing DTW and EMA calculations on each worker core. By merging all actors on a worker core, the channels of the dataflow model are altered. This is in line with the implementation, as will be explained in the following.

Instead of sending entire windows over the network-on-chip, messages are used to communicate pointers to the corresponding addresses in shared memory containing the sliding window data. Thus, the communication of sliding window data is performed as a combination of message passing and shared memory communication. To account for the message sending and receiving as the primary direct communication and synchronization between processor cores, token production and consumption rates between SW and DTW actors have been changed from 125 to 1, as only a single message is sent. Furthermore, merging all DTW actors mapped to the same worker core into a single actor, refines the message sending as well, as only a single message is sent to a processor core in the implementation. This is shown in Figure 4.16 for the analysis model in configuration C2. It can be seen that, in addition to self-edges of stateful actors of the application model as well as scheduling edges during mapping, a feedback channel from CL to SW with six initial tokens has been added. The feedback channel is synchronizing the execution of SW and CL in order to indicate that a particular tile is available for processing a new sliding window. The production and consumption rate on the feedback channel correspond to the number of messages, i.e., the number of worker cores per tile, which have to be sent back to SW in order to dispatch the next window to the worker cores. In a configuration with two tiles and thus, three worker cores per tile (C2), the production and consumption rate on said feedback channel is 3 correspondingly. This is shown in Figure 4.16 as well. However, note that six initial tokens are on the feedback channel, as in every configuration 6 worker cores are processing in parallel.

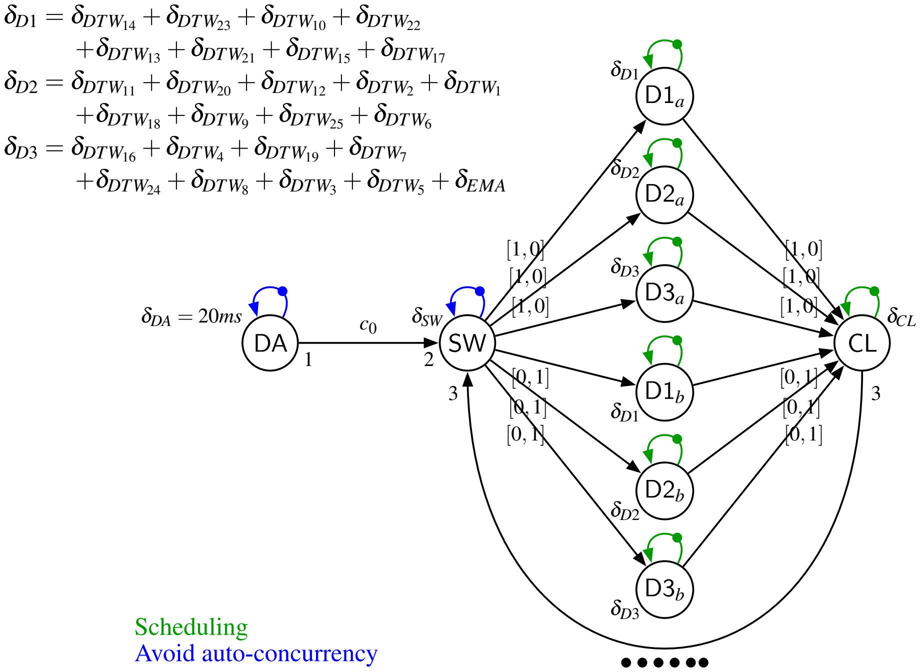

$\delta_{D1} = \delta_{DTW_{14}} + \delta_{DTW_{23}} + \delta_{DTW_{10}} + \delta_{DTW_{22}} + \delta_{DTW_{13}} + \delta_{DTW_{21}} + \delta_{DTW_{15}} + \delta_{DTW_{17}}$
$\delta_{D2} = \delta_{DTW_{11}} + \delta_{DTW_{20}} + \delta_{DTW_{12}} + \delta_{DTW_{2}} + \delta_{DTW_{1}} + \delta_{DTW_{18}} + \delta_{DTW_{9}} + \delta_{DTW_{25}} + \delta_{DTW_{6}}$
$\delta_{D3} = \delta_{DTW_{16}} + \delta_{DTW_{4}} + \delta_{DTW_{19}} + \delta_{DTW_{7}} + \delta_{DTW_{24}} + \delta_{DTW_{8}} + \delta_{DTW_{3}} + \delta_{DTW_{5}} + \delta_{EMA}$
$\delta_{DA} = 20ms$
DA
1
c_0
δ_{SW}
SW
2
3
δ_{D1}
$D1_a$
δ_{D2}
$D2_a$
δ_{D3}
$D3_a$
$D1_b$
δ_{D1}
$D2_b$
δ_{D2}
$D3_b$
δ_{D3}
[1,0]
[1,0]
[1,0]
[0,1]
[0,1]
[0,1]
[1,0]
[1,0]
[1,0]
[0,1]
[0,1]
[0,1]
CL
δ_{CL}
3
Scheduling
Avoid auto-concurrency
••••••

Figure 4.16: Analysis model in configuration C2 with merged actors on each worker core.

To account for the actual timing of message passing and shared memory communication, the refinement is continued in the following.

As mentioned in Section 3.4.1, the real-time performance of the communication channels can be modeled implicitly. This is straight forward for the selected hardware platform, as messages are communicated over non-shared hardware queues. In order to implicitly model throughput constraints into the analysis model, communication channels between different processor cores have to be modeled by additional actors.

In Figure 4.17, an example SDF graph is shown. Actors A and B are mapped to different processor cores P1 and P2. A communication channel c between A and B is implicitly mapped to the hardware communication link t. When A produces n tokens on c, these tokens immediately become available for B in the example dataflow graph. However, in an actual implementation, communication of data is limited by a certain throughput constraint of the channel, i.e., due to the transmission time t_T of a message. Additionally, communication is subject to a delay, e.g., the Time of Flight (ToF) which is the time that passes during propagation of each bit over the hardware link from the sender to the receiver. Furthermore, each communication latency is subject to a sending and receiving overhead, which in the context of this thesis is defined as the active time of the processor that is spend when sending or receiving data, respectively.

To refine the analysis model towards the transmission delay of data, a mapped dataflow channel can be represented by additional dataflow actors representing the aforementioned timing aspects, i.e., ToF, transmission time, sending overhead, and receiving overhead. In very short communication links,

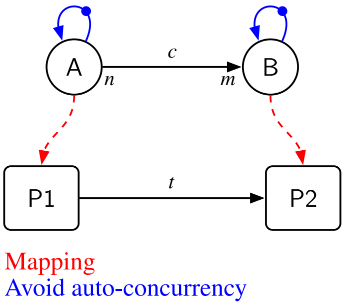


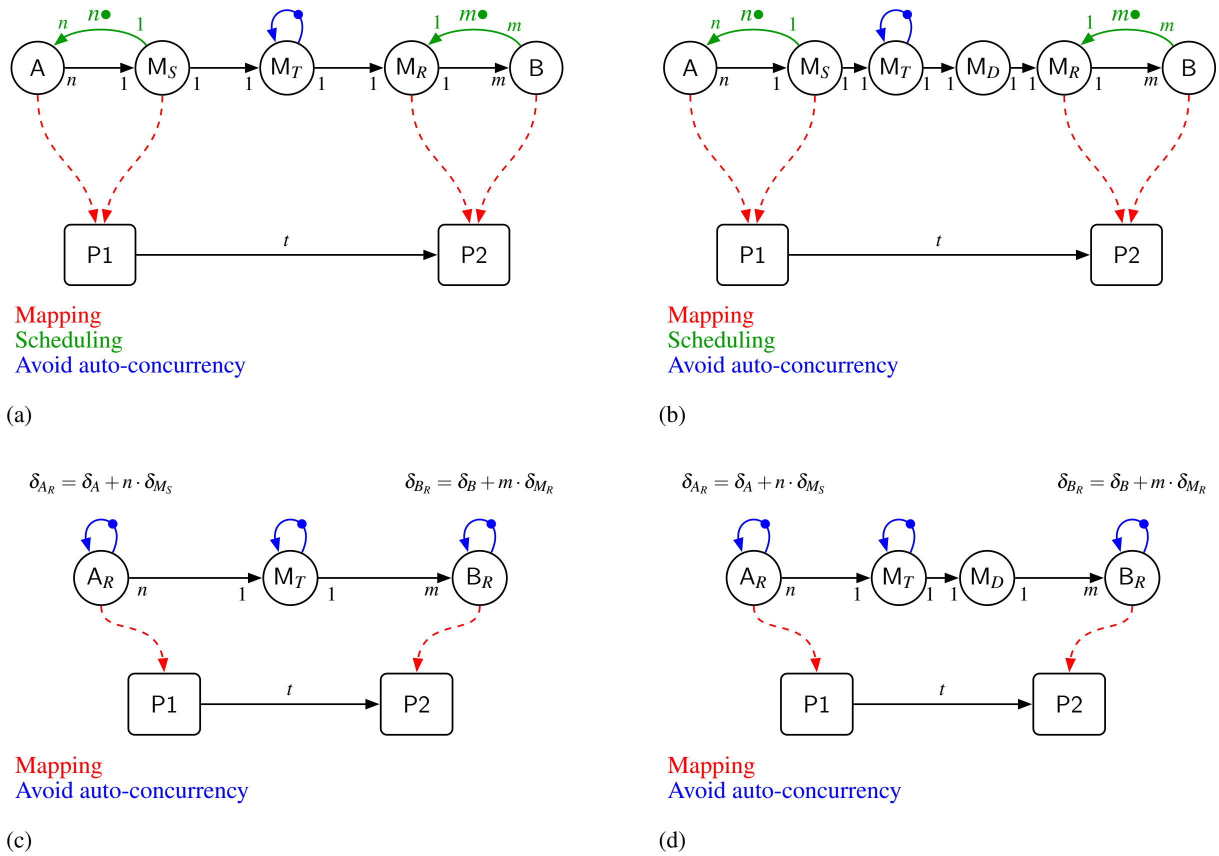
Figure 4.17: Mapped synchronous dataflow graph model with inter-processor communication between the two actors A and B.

e.g., NoCs, ToF might be in the scale of picoseconds to nanoseconds which may allow to neglect this aspect in the dataflow model, depending on the level of abstraction. In the experiments section of this chapter, ToF has been neglected and was not included in the model. However, for the sake of completeness, both versions, with and without ToF will be discussed in the following. The refined model of Figure 4.17 is shown in Figure 4.18(a) and Figure 4.18(b), without and with ToF, respectively.

The sending overhead is modeled by actor M_S. As it represents active time on the processor core, this actor must be mapped to the same processor core as the actor whose data sending it represents. For the same reason, this actor is subject to scheduling. As one token represents a defined unit of data for the application model, the production rate of actor A represents n units of data. To this end, the execution time δ_{M_S} of the sending overhead M_S is the time overhead for a single data unit, which has to be fired n times and thus has a repetition vector entry of $\gamma(M_S) = \gamma(A) \cdot n$. Finally, n initial tokens are required as initial start condition of the graph on the scheduling channel between M_S and A or on the channel from A to M_D, depending on the initial start condition. Alternatively, M_S could represent the communication of n units at once, which would result in a consumption from and production to A of n tokens instead of one and a correspondingly scaled execution time by n. Likewise, the receiving overhead on actor B is modeled with actor M_R, which is subject to mapping and scheduling accordingly.

The transmission time of data over the hardware link is modeled by actor M_T (see Figure 4.18(a) and Figure 4.18(b)) or more specifically, by its execution time δ_{M_T}. As there can only be one data transmission on a hardware channel at a time, this actor receives a self-edge, to avoid auto-concurrency. This actor and its self-edge model the maximum throughput of the hardware channel, which is $\frac{1}{\delta_{M_S}}$, resulting from its transmission time of one data unit over the hardware link. To this end, the production and consumption rate of M_T equals one. This is also necessary, as production and consumption rates of A and B, respectively, may be co-prime. Using a production and consumption rate of one for M_T allows actor B to be fired as soon as m tokens are received, i.e. m data units have arrived at B, instead of waiting for an integer multiple x of n units of data to arrive at B with $x \cdot n \geq m$.

Finally, as mentioned above, the ToF or other additional delays, which are not affecting the throughput but each unit of data equally, can be modeled as an additional actor, e.g., actor M_D in Figure 4.18(b). This actor has no self-edge, as it represents a delay that does not affect the throughput. Hence, each unit of data will be affected by a constant delay, which allows auto-concurrency for the corresponding actor in the model. Again, the production and consumption rate equals one, as the delay applies to each unit of data individually and immediately upon propagation. Note that additional actors without self-edges


A
M_S
M_T
M_R
B
M_D
n•
m•
n
m
1
P1
P2
t
Mapping
Scheduling
Avoid auto-concurrency
$\delta_{A_R} = \delta_A + n \cdot \delta_{M_S}$
$\delta_{B_R} = \delta_B + m \cdot \delta_{M_R}$
A_R
B_R

(a) (b) (c) (d)

Figure 4.18: Refinement of inter-processor communication (a) without and (b) with constant propagation delay of the hardware link and abstraction of sending and receiving overheads (c) without and (d) with constant propagation delay of the hardware link.

are restricted to SDF actors in case of CSDF analysis models, as otherwise functional determinacy might be compromised [16]. However, in the experiments of this chapter, ToF is neglected due to the comparably short structures of the NoC and thus, is not part of the analysis models.

The above refinement has been performed for the analysis models of each configuration in the experiments. In order to account for the shared memory communication, the worst-case timing of copying the current window from shared memory to the scratchpad memory of the worker core is contained in its receiving overhead. This memory copying is referred to as *MCP* in the following. Likewise, writing back the DTW and EMA results into the message is contained in the sending overhead of each worker core. The memory write back is referred to as *MWB* in the following. The time of reading the results of all messages together is abstracted into the execution time of CL. This abstraction is based on the worst case, i.e., configuration C1, with six messages arriving to CL. As a result, the annotated execution time of CL does not need to be changed for different mappings of D1-D6 and is thus, independent of the parallelization configuration in the models. Furthermore, actor SW does not contain shared memory writes of sliding window data in its sending overhead, as in the experiments, data for the whole test sequence is already present in shared memory. Therefore,

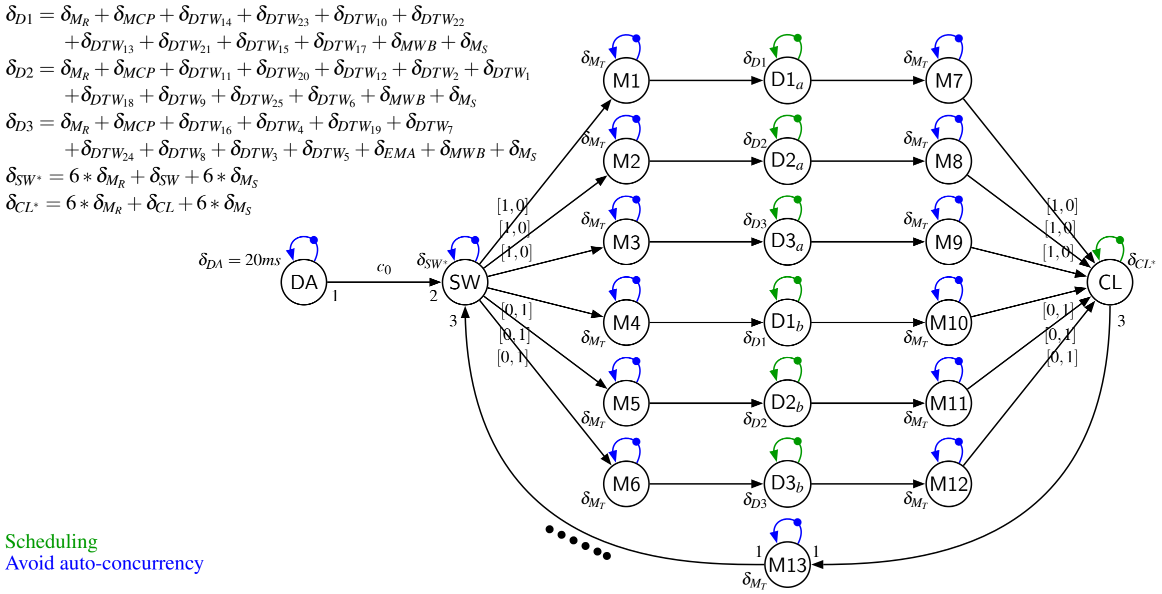

$\delta_{D1} = \delta_{M_R} + \delta_{MCP} + \delta_{DTW_{14}} + \delta_{DTW_{23}} + \delta_{DTW_{10}} + \delta_{DTW_{22}} + \delta_{DTW_{13}} + \delta_{DTW_{21}} + \delta_{DTW_{15}} + \delta_{DTW_{17}} + \delta_{MWB} + \delta_{M_S}$
$\delta_{D2} = \delta_{M_R} + \delta_{MCP} + \delta_{DTW_{11}} + \delta_{DTW_{20}} + \delta_{DTW_{12}} + \delta_{DTW_2} + \delta_{DTW_1} + \delta_{DTW_{18}} + \delta_{DTW_9} + \delta_{DTW_{25}} + \delta_{DTW_6} + \delta_{MWB} + \delta_{M_S}$
$\delta_{D3} = \delta_{M_R} + \delta_{MCP} + \delta_{DTW_{16}} + \delta_{DTW_4} + \delta_{DTW_{19}} + \delta_{DTW_7} + \delta_{DTW_{24}} + \delta_{DTW_8} + \delta_{DTW_3} + \delta_{DTW_5} + \delta_{EMA} + \delta_{MWB} + \delta_{M_S}$
$\delta_{SW^*} = 6 * \delta_{M_R} + \delta_{SW} + 6 * \delta_{M_S}$
$\delta_{CL^*} = 6 * \delta_{M_R} + \delta_{CL} + 6 * \delta_{M_S}$
$\delta_{DA} = 20ms$
DA
c_0
δ_{SW^*}
SW
δ_{M_T}
M1
M2
M3
M4
M5
M6
δ_{D1}
δ_{D2}
δ_{D3}
D1$_a$
D2$_a$
D3$_a$
D1$_b$
D2$_b$
D3$_b$
M7
M8
M9
M10
M11
M12
M13
CL
δ_{CL^*}
[1, 0]
[0, 1]
1
2
3
Scheduling
Avoid auto-concurrency

Figure 4.19: Refined analysis model of the hand gesture recognition system in configuration C2.

the timing analysis can be considered from the time as soon as sensor data samples are received and written to shared memory of the initiator core.

During model refinement, the model complexity is increasing due to added refinement actors. Therefore, the refined analysis model is abstracted again, although with a different focus. This is performed for the analysis models in the experiments, in order to reduce scheduling complexity of message sending and receiving overheads. To this end, the schedule between an actor and its receiving overheads, e.g., actors A and M$_S$ in Figure 4.18(a), is abstracted to its worst-case behavior. An actor A and its receiving overhead are merged into a single actor and their execution times are accumulated by a weighted sum w.r.t. their repetition vector entry ratio. As an example, the repetition vector entry of actor M$_S$ equals $\gamma(M_S) \cdot n = \gamma(A)$. Thus, the resulting execution time of the merged actor A$_R$ equals $\delta_{A_R} = \delta_A + n \cdot \delta_{M_S}$.

The resulting merged actor A$_R$ is depicted in Figure 4.18(c). The aforementioned merging, abstracts the sending time of each unit of data to the sending time of the last unit of data that is propagated over the hardware channel. Furthermore, this step can be applied to the sending overheads of multiple messages from one actor to different channels, abstracting the schedule of message sending to its worst-case behavior. Likewise, the same abstraction can be applied to the receiving overhead of the receiver actor, i.e., actors B and M$_R$, which results in actor B$_R$ in Figure 4.18(c). For the sake of completeness, the corresponding graph with additional ToF modeling is depicted in Figure 4.18(d).

The aforementioned abstraction has been performed for the analysis models of all four configurations in order to reduce model complexity. All sending overheads and receiving overheads of SW, D1-D6, and CL have been merged and included into the execution times of said actors. The resulting analysis model of configuration C2 is depicted in Figure 4.19. For the sake of brevity, only the analysis model of configuration C2 is shown, as its contains the modeling aspects within, as well as between different tiles.

Model annotation All execution time annotations for the refined models have been measured on the target platform. Therefore, timestamp functions have been used before and after the individual code section of each actor. Measurements for model annotation have been performed on the training dataset. Especially, the worst-case execution times δ_{MCP}, $\delta_{DTW_1} - \delta_{DTW_{25}}$, δ_{EMA}, δ_{MWB}, and δ_{CL} have been measured for each sliding window and their maximum values have been used for actor annotations. The execution times δ_{M_S} and δ_{M_R} have been measured in loops at initialization time of the software as well as the round trip time t_{RTT} of sending a message from initiator to different processor cores which immediately sent it back. From the sets of measured sending and receiving overheads t_{M_S} and t_{M_R} and the round trip time $t_{M_{RTT}}$, the worst-case transmission time δ_{M_T} has been calculated by:

$$\delta_{M_T} = \frac{max\ t_{M_{RTT}}}{2} - (min\ t_{M_S} + min\ t_{M_R}).$$

The resulting WCET of message transmission times resulted in $\delta_{M_T} = 11,894 ns$. The underlying minimum (*min t*), maximum (*max t*), and average ($\bar{t}$) of the measurements together with their corresponding standard deviation s_t, coefficient of variation CV_t, and the number of all measurements # are summarized in Table 4.4.

The execution time δ_{SW^*} basically consists of sending and receiving overheads, since the sliding window data is already in shared memory. Therefore, δ_{SW} is assumed to be close to zero and thus δ_{SW^*} solely consists of six sending and receiving overheads. Note that δ_{SW^*} as well as δ_{CL^*} include six sending and receiving overheads regardless of the configuration. This abstracts its timing behavior to the WCET among all configurations and reduces design complexity by keeping δ_{SW^*} and δ_{CL^*} independent from the configuration. As a result, only execution time annotations of actors the mapping relations of which are actually affected by different configurations have to be adapted.

From the coefficients of variation in Table 4.4 it can be seen that most of the code sections are subject to very little timing variations. This is a) due to the dataflow-oriented nature of most of the recognition software stages with little control flow and b) the rather high determinacy of the hardware architecture, especially considering the implemented memory configuration with scratchpad memories instead of caches. However, for *CL* and *MWB*, the measured values are subject to high variations in execution time with coefficients of variation of up to almost 60 %. The reason behind the high variation in execution time of *MWB* is its linear dependency w.r.t. to the number of DTW distances that have to be written to memory and thus the number of executed DTW instances on a worker core. Therefore, the execution time of *MWB* is directly dependent on the configuration. This could be substantiated by analyzing its statistical timing parameters per configuration, leading to a coefficient of variation less than 0.4 % for each configuration individually.

Lastly, the high coefficient of variation of 28.77 % in execution time of *CL* is caused by two factors. The timing of *CL* depends on the number of messages to evaluate for a single sliding window and thus on the configuration. However, its high level of control flow in the management of asynchronously arriving messages, the selection of DTW values that fall below their class-dependent thresholds, and the *arg min* evaluation of remaining DTW values is causing high variations in its execution time as well. While all other actor WCET annotations are rather close to their mean values, the high level of control flow of actor *CL* suggests a proper WCET analysis for model annotation. This however, has not been performed in the experiments, and its maximum observed execution time has been annotated in the analysis models.

Table 4.4: Execution time t_{ex} measurements for CSDF analysis model annotation.

Name	#	$\bar{t}_{ex}\ [ns]$	$s_{t_{ex}}\ [ns]$	$CV_{t_{ex}}$	$min\ t_{ex}\ [ns]$	$max\ t_{ex}\ [ns]$
$DTW1$	22,053	5,848,250	4,051	0.07 %	5,837,706	5,862,066
$DTW2$	22,053	11,666,500	13,433	0.12 %	11,629,310	11,709,908
$DTW3$	22,053	8,901,620	6,139	0.07 %	8,882,718	8,922,812
$DTW4$	22,053	7,453,820	5,307	0.07 %	7,439,192	7,475,085
$DTW5$	22,053	8,459,820	7,094	0.08 %	8,438,800	8,489,155
$DTW6$	22,053	7,449,020	4,216	0.06 %	7,437,838	7,466,050
$DTW7$	22,053	7,763,200	4,715	0.06 %	7,750,156	7,783,218
$DTW8$	22,053	8,304,760	5,410	0.07 %	8,289,191	8,327,348
$DTW9$	22,053	6,301,300	3,355	0.05 %	6,292,621	6,315,007
$DTW10$	22,053	4,555,650	2,122	0.05 %	4,549,821	4,564,800
$DTW11$	22,053	5,118,320	2,368	0.05 %	5,112,592	5,127,810
$DTW12$	22,053	5,549,230	3,004	0.05 %	5,540,701	5,560,983
$DTW13$	22,053	4,980,820	2,629	0.05 %	4,975,045	4,991,360
$DTW14$	22,053	3,100,000	1,413	0.05 %	3,096,055	3,105,033
$DTW15$	22,053	5,120,380	2,501	0.05 %	5,113,399	5,129,857
$DTW16$	22,053	11,002,000	8,701	0.08 %	10,972,647	11,036,159
$DTW17$	22,053	13,148,000	11,813	0.09 %	13,109,106	13,188,169
$DTW18$	22,053	11,395,400	9,767	0.09 %	11,365,694	11,431,977
$DTW19$	22,053	9,924,200	6,219	0.06 %	9,905,805	9,947,128
$DTW20$	22,053	12,734,400	10,715	0.08 %	12,702,107	12,774,217
$DTW21$	22,053	14,062,100	15,056	0.11 %	14,020,475	14,112,473
$DTW22$	22,053	14,870,600	21,123	0.14 %	14,821,051	14,945,991
$DTW23$	22,053	17,927,400	23,368	0.13 %	17,872,229	17,997,618
$DTW24$	22,053	9,528,740	7,143	0.07 %	9,509,424	9,554,205
$DTW25$	22,053	11,266,300	10,789	0.10 %	11,231,304	11,299,466
EMA	22,053	25,960	0	0.00 %	25,960	25,960
CL	22,053	584	168	28.77 %	350	1,038
MCP	56,529	15,046	39	0.26 %	15,014	15,252
MWB	56,529	709	425	59.94 %	316	1,766
M_S	336,000	4,425	12	0.27 %	4,422	5,282
M_R	336,000	104	1	0.96 %	104	742
M_{RTT}	1,008,000	31,215	108	0.35 %	31,143	32,839

4.6 Evaluation

In this section, the accuracy of model-based results is evaluated experimentally by comparison to corresponding implementations. As a result, the performance metrics to be evaluated need to be measured from the implementation. The conducted performance assessments are described in the following section and the acquired results are be presented in Section 4.6.2.

4.6.1 Performance assessment

As each parallelization configuration imposes different effects on the throughput of the application, the *real-time behavior* of the application is monitored. Real-time performance w.r.t. throughput is achieved when each released sliding window is successfully dispatched to one of the tiles of the system. Each tile, when a sliding window is dispatched to it, remains unavailable from the moment the initiator core sent its messages to the individual worker cores of that tile until the initiator receives the messages back from the evaluator which evaluated the DTW distance results and the EMA value of that particular window. Therefore, the time of sending a window to the worker cores of a tile, processing the window, sending the results to the evaluator, evaluating the DTW distances and EMA result, and sending the messages back to the initiator needs to be shorter than $N_K \cdot T_{SW}$ with N_K being the number of tiles and T_{SW} being the period between two consecutive sliding windows, i.e., $40ms$. If this is satisfied, at least one tile is available upon dispatching of a new sliding window. Otherwise, windows have to be skipped due to insufficient throughput. Thus, a simple monitoring of successfully dispatched windows to one of the tiles is sufficient to assess real-time performance in the experiments. The results are recorded along with sliding window IDs and written to a file after all windows of a test sequence have been processed. The recording is performed at the end of the initiator loop, in order to not interfere with the the sending of sliding windows and thus influencing the timing behavior.

Another performance metric that is of importance is the *latency* of the recognition system, as it directly influences its responsiveness and thus, its usability. In this context, latency is described as the time that elapses from the release of a new window on the initiator, until the classification of the window at the evaluator core, based on its calculated EMA and DTW distance values. In order to measure latency, timestamp modules provided by the TI SYS/BIOS kernel have been used. However, since the DSP processor cores have individual clock sources which are not synchronized w.r.t. each other, taking a start timestamp at the initiator and an end timestamp on the evaluator is not meaningful. To this end, a *timestamp* task with a higher priority than the classification task has been implemented on the evaluator. The timestamp task is implemented as a loop with a blocking message read at its entry. Note that special *timing* messages and corresponding message queues have been implemented, in order to distinguish these from messages with DTW distance information. The timestamp task implements a timestamp read upon successful return of the timing message read operation and saves the timestamp in the timing message before sending back to its origin, i.e., the initiator. The initiator uses this functionality and sends a timing message to the evaluator and waits for its return before sending a new sliding window to the worker cores. The returned timing message result will then be written and carried along within the messages that communicate sliding window pointers to the worker cores and DTW distances to the evaluator. At the evaluator side, a second timestamp is taken after labeling an evaluated window. The differences are recorded along with sliding window IDs and written to a file after the test sequence of sensor data has been entirely processed. However, the overhead, resulting from the additional transfer of the timing message from evaluator to initiator, has been removed from the latency measurements. Due to the independent hardware queues provided by the Multicore

Table 4.5: Measured round trip time of time messages between initiator and evaluator core.

#	$\bar{t}_{RTT,t}$ $[ns]$	$s_{t_{RTT,t}}$ $[ns]$	$CV_{t_{RTT,t}}$	$min\ t_{RTT,t}$ $[ns]$	$max\ t_{RTT,t}$ $[ns]$
228,000	21,969	41	0.19 %	21,919	22,763

Navigator and the fixed message sizes, the transmission time of a timing message can be expected to suffer from only minor variations, which could be substantiated by further evaluation. In order to assess the imposed latency overhead due to the additional timing message transmission time, the round trip time $t_{RTT,t}$ of that timing message path is measured $3,000$ times at initialization. Note that the t subscript is used to distinguish the round trip time for timing messages from the round trip time of messages that contain sliding window and DTW distance information. The $3,000$ measurements have been evaluated regarding mean $\bar{t}_{RTT,t}$, standard deviation $s_{t_{RTT,t}}$, coefficient of variation $CV_{t_{RTT,t}}$, minimum $min\ t_{RTT,t}$, and maximum $max\ t_{RTT,t}$ among all 19 test sequences in all four configurations, i.e., $228,000$ measurements in total. The results are listed in Table 4.5, which show that the variation in transmission time is comparably small. These measured values have been used to compensate the measuring overhead. From all timing measurements of the recognition process, half of the measured RTT of the time messages is subtracted in order to compensate the induced overhead. Note that for mean latency, half of the mean RTT has been subtracted but for minimum and maximum latency, half of the maximum and minimum RTTs have been subtracted, respectively, in order to calculate consistent worst- and best-case results.

The third performance indicator that has been measured is the *processor utilization*. In the thesis at hand, processor utilization U_p is defined as the average ratio of active time t_a in which the processor is actively executing the application code and the observation time t_o. Hence, the difference $t_o - t_a$ denotes the idle time of the processor core or time that the processor is executing code that is not part of the actual application, e.g., performance assessment related code. In order to measure processor utilization, TI System Analyzer instrumentation libraries have been integrated into the application code, in order to monitor performance indicators like processor load, execution traces, and events by logging and uploading to a host computer at run time. The execution analysis feature has been utilized, which allows for building execution graphs, analyzing concurrency, and profiling tasks. To this end, log buffers have been configured for the worker cores and the System Analyzer module has been compiled into the application for run-time task profiling. From the acquired logs, which include start and end times of threads, processor utilization has been analyzed.

4.6.2 Evaluation Results

In this section, model-based analysis results and their corresponding measurements on the experimental implementation are presented and compared with each other.

Model results The analysis models have been analyzed with CSDF analysis tools of SDF^3 [13], w.r.t. repetition vector and throughput. However, since the CSDF analysis tools do not include a maximum response time analysis, all analysis models have additionally been converted to their corresponding SADF equivalent with `sdf3convert-csdf-sadf`, since the SADF tools include maximum response time analysis. The repetition vector of all analysis models is shown in Table 4.6.

Table 4.6: Repetition vector γ_G entries of the CSDF analysis models.

Cfg.	DA	SW	M1	M2	M3	M4	M5	M6	M7	M8	M9
C1	2	1	1	1	1	1	1	1	1	1	1
C2	4	2	1	1	1	1	1	1	1	1	1
C3	6	3	1	1	1	1	1	1	1	1	1
C6	12	6	1	1	1	1	1	1	1	1	1

Cfg.	M10	M11	M12	M13	D1	D2	D3	D4	D5	D6	CL
C1	1	1	1	6	1	1	1	1	1	1	1
C2	1	1	1	6	1	1	1	1	1	1	2
C3	1	1	1	6	1	1	1	1	1	1	3
C6	1	1	1	6	1	1	1	1	1	1	6

The maximum response time $max\,\widehat{t_{rsp}}$ acquired from the analysis models is representing the latency of the recognition pipeline. That is the time between a new sliding window is released by the initiator until the window is classified by the evaluator. The maximum response time results of each configuration acquired from the analysis models are listed in Table 4.7. It can be seen that configurations with less tiles have a shorter latency. As an example, in configuration C6, a sliding window is processed in sequence on a worker core. However, the processing of a window cannot generally be parallelized to an arbitrary degree. This can also be observed from the maximum response times of all configurations. Ideally, with N processor cores, configuration C1 would be performed in $1/N$ time w.r.t. to its single core computation. However, as Amdahl states in [114], most of real world computing tasks are not fully parallelizable. Therefore, the maximum response times are not linearly decreasing w.r.t. the number of processor cores of a tile. In fact, although latency itself cannot be used as a measure of real-time behavior in terms of the system throughput, it can be seen that the maximum response time of configuration C1 exceeds its sliding window period of $40\,ms$. The maximum response times of configurations C2, C3, and C6 fall below their tile-specific sliding window periods, i.e., $80\,ms$, $120\,ms$, and $240\,ms$, respectively.

The real-time indicator at hand however, is the actor throughput $TH(\mathsf{DA})$ which directly informs about the ability of processing sliding windows at the desired rate with which sensor data is sampled. From Table 4.7 it can be seen, that this is not the case for configuration C1, while all other configurations

Table 4.7: Response time $max\,\widehat{t_{rsp}}$, graph throughput $TH(G)$, and throughput $TH(DA)$ of actor DA analyzed from the CSDF analysis models.

Cfg.	$max\,\widehat{t_{rsp}}$ $[ns]$	$TH(G)$ $[s^{-1}]$	$TH(DA)$ $[s^{-1}]$	Real Time
C1	$41,155,100$	24.26	48.51	No
C2	$78,155,500$	12.50	50.00	Yes
C3	$119,190,000$	8.33	50.00	Yes
C6	$227,264,000$	4.17	50.00	Yes

Table 4.8: Processor utilization acquired from the CSDF analysis models.

Cfg.	$\widehat{U}(\mathsf{W1})$	$\widehat{U}(\mathsf{W2})$	$\widehat{U}(\mathsf{W3})$	$\widehat{U}(\mathsf{W4})$	$\widehat{U}(\mathsf{W5})$	$\widehat{U}(\mathsf{W6})$	$\overline{\widehat{U}}$
C2	97.57 %	96.96 %	89.48 %	97.57 %	96.96 %	89.48 %	94.67 %
C3	99.24 %	90.08 %	99.24 %	90.08 %	99.24 %	90.08 %	94.66 %
C6	94.65 %	94.65 %	94.65 %	94.65 %	94.65 %	94.65 %	94.65 %

provide the necessary throughput $TH(\mathsf{DA}) = 50\,s^{-1}$. Hence, the mapping of DTW instances on 6 processor cores (configuration C1) is either not possible in an efficient way, or is at least not efficient enough in the particular chosen mapping. Consequently, configuration C1 is not further analyzed w.r.t. processor utilization. For configurations C2, C3, and C6, the model-based results suggest real-time performance.

The processor utilization $\widehat{U}(p)$ of the worker cores is analyzed from the analysis models, as introduced in Section 3.4.1. With Equation (5) all worker core processor utilizations have been calculated from the analysis models of configurations C2, C3, and C6. The results are shown in Table 4.8. Inspecting the average processor utilization among all worker cores $\overline{\widehat{U}}$, it can be seen that the configuration does not have a substantial impact on the total processor utilization among all worker cores. Although the discrete nature of distributing differently sized software instances on a set of processor cores is noticeable in the processor utilization of each particular core, it does not impact the overall processor utilization of the MPSoC significantly. However, the number of worker cores per tile does influence the total processor utilization slightly. In a configuration with N_{W_K} worker cores per tile, the sliding window data has to be loaded from shared memory N_{W_K} times, which also increases the active time of each worker core to some degree. In Table 4.8, a slight decrease in average processor utilization for configurations with more tiles and thus, less worker cores per tile, can be observed.

Measured results In order to compare the aforementioned model-based results, timing of the corresponding implementations has been measured in experiments. Therefore, the recognition software has been executed on all 19 test datasets. For each processed sliding window, the latency (response time t_{rsp}) has been measured as described in Section 4.6.1, and the drop of windows due to insufficient throughput has been monitored. The worst and best case response times $min\ t_{rsp}$ and $max\ t_{rsp}$, respectively, as well as the mean response time $\bar{t}_{rsp}$, its standard deviation $s_{t_{rsp}}$, and its coefficient of variation $CV_{t_{rsp}}$ are listed in Table 4.9.

Table 4.9: Response times t_{rsp} measured from the implementation.

Cfg.	#	$\bar{t}_{rsp}\ [ns]$	$s_{t_{rsp}}\ [ns]$	$CV_{t_{rsp}}$	$min\ t_{rsp}\ [ns]$	$max\ t_{rsp}\ [ns]$	Real Time
C1	2,807	40,237,123	20,757	0.05 %	40,169,856	40,311,056	No
C2	5,955	76,303,161	48,464	0.06 %	76,162,442	76,468,304	Yes
C3	6,108	116,404,797	63,106	0.05 %	116,208,746	116,646,730	Yes
C6	6,538	222,041,748	99,997	0.05 %	221,731,938	222,476,012	Yes

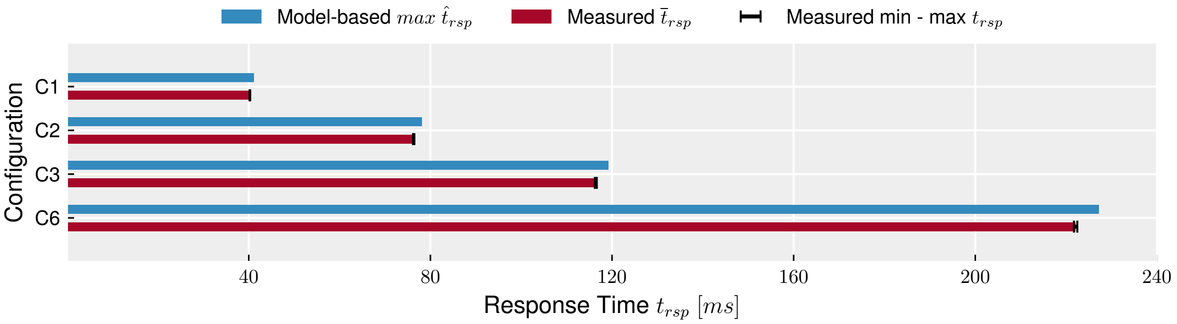


Figure 4.20: Measured and model-based maximum response time results.

It can be seen, that similar to the model-based results, the mean, max., and min. response times of configuration C1 are higher than the sliding window period of $40\,ms$. Response times of all other configurations are below their tile-based sliding window periods, i.e., $80\,ms$, $120\,ms$, and $240\,ms$ for configuration C2, C3, and C6, respectively. However, in order to verify real-time behavior, it has been logged whether all windows could be dispatched successfully or some had to be discarded due to unavailability of worker cores. While for configuration C2, C3, and C6 no window was discarded, in configuration C1 every other window had to be discarded due to unavailable tiles upon window release. Hence, configuration C1 is not performing in real time, while configurations C2, C3, and C6 do. This could be correctly indicated by the throughput results of the analysis models.

Furthermore, the small coefficients of variation of the response times in Table 4.9 indicate a highly dataflow-oriented performance of the recognition system with little control flow. This substantiates the motivation to model and analyze human activity recognition systems with dataflow graphs.

In order to evaluate the accuracy of model-based results, a comparison between measured maximum response times and model-based analysis results is shown in Figure 4.20. Additionally, the underlying values are listed in Table 4.10, including the absolute and relative difference between model-based and measured maximum response times. It can be seen that the model-based results indeed show a worst-case estimation in the conducted experiments, as the worst-case measured response times fall below the model-based worst-case estimations. Furthermore, the relative difference is rather small with 2.09 % to 2.21 %, which offers an accuracy that is sufficient to substantiate design decisions at design time.

Table 4.10: Comparison of measured ($max\ t_{rsp}$) and model-based analysis ($max\ \widehat{t_{rsp}}$) response time results.

Cfg.	$max\ \widehat{t_{rsp}}$ $[ns]$	$max\ t_{rsp}$ $[ns]$	$\Delta\ max\ t_{rsp}$ $[ns]$	$\Delta_r\ max\ t_{rsp}$
C1	41,155,100	40,311,056	844,045	2.09 %
C2	78,155,500	76,468,304	1,687,197	2.21 %
C3	119,190,000	116,646,730	2,543,271	2.18 %
C6	227,264,000	222,476,012	4,787,989	2.15 %

Table 4.11: Measured processor utilization from implementation.

Cfg.	U(W1)	U(W2)	U(W3)	U(W4)	U(W5)	U(W6)	$\overline{U}$
C2	96.34 %	95.76 %	88.18 %	96.34 %	95.76 %	88.34 %	93.46 %
C3	98.08 %	88.79 %	98.08 %	89.03 %	98.08 %	89.03 %	93.52 %
C6	93.29 %	93.81 %	93.81 %	93.81 %	93.81 %	93.81 %	93.72 %

In a second experiment, the recognition software has been performed on all 19 test datasets and analyzed by the TI System Analyzer tool to log task profiling indicators, i.e., begin and end timestamps of task executions on the worker cores. From these task profiling logs, processor utilization has been analyzed. Table 4.11 summarizes the results.

As the model-based results already indicated, it can be seen that the processor utilization of all worker cores in configurations C6 is balanced best, which is due to the same amount of processing effort on each worker core in this configuration. A comparison of model-based and measured processor utilization is depicted in Figure 4.21. The calculated relative differences Δ_r are additionally listed in Table 4.12. The model-based worst-case estimates of all configurations are higher than the results measured on the implementation. Furthermore, the relative differences between model-based and measured results are less than 1.4 %.

It can be seen, that similar to the model-based results, the average processor utilization among all processor cores is mostly independent of the parallelization configuration. However, slight differences can be observed, which are contrasting the model-based results, i.e., the average processor utilization is slightly increasing with the number of tiles. Since the increase in processor utilization is comparably small and within the worst-case abstraction of model-based results, a further investigation of the cause has been abstained from in this chapter. However, a possible cause is the instrumentation of processor cores itself, which is investigated in more detail in Section 5.1.

Although the average processor utilization is not considerably affected by the chosen parallelization configuration, the processor utilization of each individual worker core depends on the distribution of templates. Such results allow to consider possible approaches reducing dynamic energy consumption, e.g., reduction of clock rate and operating voltage, for individual processor cores, if applicable.

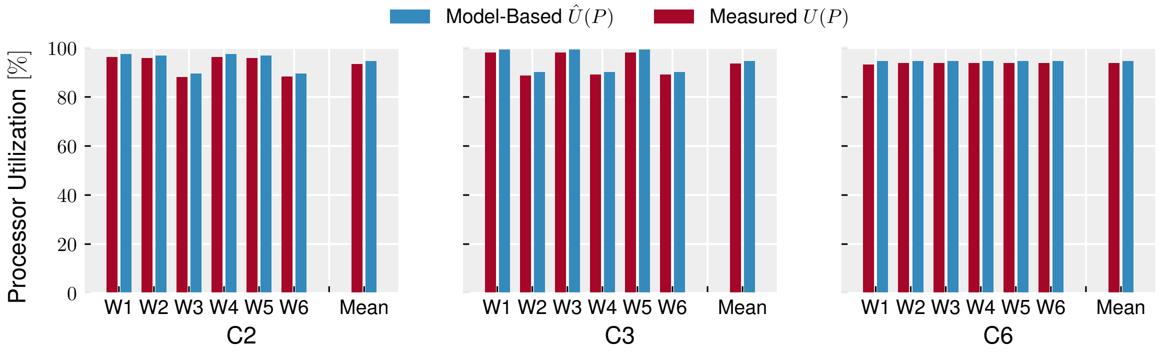


Figure 4.21: Measured and model-based processor utilization.

Table 4.12: Relative differences between model-based and measured processor utilization analysis results.

Cfg.	$\Delta_r\, U(\mathsf{W1})$	$\Delta_r\, U(\mathsf{W2})$	$\Delta_r\, U(\mathsf{W3})$	$\Delta_r\, U(\mathsf{W4})$	$\Delta_r\, U(\mathsf{W5})$	$\Delta_r\, U(\mathsf{W6})$	$\Delta_r\, \overline{U}$
C2	1.27 %	1.25 %	1.48 %	1.28 %	1.25 %	1.29 %	1.30 %
C3	1.18 %	1.46 %	1.18 %	1.18 %	1.18 %	1.18 %	1.22 %
C6	1.47 %	0.90 %	0.90 %	0.90 %	0.90 %	0.90 %	0.99 %

To this extend, considerations regarding energy efficiency of the processor cores in a particular parallelization approach and their task mapping can be made at design time from model-based analysis results.

In general, the experimental results indicate a model accuracy which is sufficient to substantiate design decisions at early design stages. The model-based analysis allows to evaluate real-time behavior in terms if throughput and latency, as well as unused processor resources in terms of processor utilization to make use of possible energy reduction methods, e.g., voltage an frequency scaling or clock-gating.

Recognition accuracy Finally, the recognition accuracy has been evaluated in all configurations. As the ground truth has been recorded in terms of gesture sequences instead of labeled timestamps, a string-based evaluation has been used. To this end, the Levenshtein distance between the ground truth gesture sequence and the sequence of detected gestures has been calculated. This, however, does not allow for exact reasoning about recognition accuracy in terms of timing. Anyhow, since recognition performance is of minor importance for the conducted performance evaluations, a subsequent recording of new gesture datasets including time-based ground truth annotations has been abstained from.

From the string-based Levenshtein distance, the precision, recall, and F1-score has been calculated. The results are listed in Table 4.13. It can be seen, that the overall recognition accuracy of the system is rather mediocre. However, the experimental implementation does not have the aspiration to be a ready to use gesture recognition system, and was thus not optimized in this regard. One main reason for the rather mediocre recognition performance is the relatively small number of templates w.r.t. the high dimensionality of the signals, i.e., nine dimensions. Furthermore, some of the sensor signal axes might not hold important information for discriminating between the selected gestures, which can be interpreted as noise that accumulates in the DTW distances. However, optimization regarding the recognition accuracy is beyond the focus of the thesis at hand.

Table 4.13: Recognition performance of the experimental hand gesture recognition system.

Configuration	Precision	Recall	F1-Score
C2	59.31 %	52.76 %	55.84 %
C3	59.31 %	52.76 %	55.84 %
C6	59.31 %	52.76 %	55.84 %

What can be seen from the results in Table 4.13, is that the parallelization configurations do not have an impact on the recognition accuracy. This is in line with the concept, as none of the decisive parameters like sampling rate, window length, or window overlap is changed among different configurations. Indeed, such parameters have not been optimized for the experimental implementation and are likely off from the optimal parameters w.r.t. recognition accuracy. However, functional parameters like sampling rate, window size, and window overlap are in fact influencing extra-functional properties like latency, throughput, and processor utilization. Thus, it is crucial to analyze extra-functional properties as early in the system design as possible to substantiate that selected parameters during training allow a real-time and energy-efficient system design on a chosen hardware. A design-time analysis of extra-functional properties takes functional parameters like sampling frequency or window length and overlap into account. This allows for fast optimization cycles on the functional parameters providing possibilities for trade-offs with extra-functional properties like throughput, latency, or energy efficiency.

4.7 Discussion

In this chapter, state-of-the-art parallelization approaches for HAR systems on multi-processor architectures have been evaluated w.r.t. system throughput, latency, and processor utilization. Furthermore, their representation in a MoC, i.e., CSDF, has been presented in order to model and analyze their impact on said extra-functional properties at design time. The model-based estimation of extra-functional properties has been evaluated with corresponding implementations on a state-of-the-art multi-processor DSP system-on-chip, which could achieve an average accuracy above 97 %. The acquired results show, that CSDF graphs are a well suited MoC to capture dataflow-oriented processing stages of sensor-based human activity and gesture recognition systems and respective parallelization strategies. The accuracy of model-based results in the experiments is accurate enough to substantiate decisions on parallelization approaches w.r.t. their impact on latency, throughput, and processor utilization early in the design process.

The parallelization approaches that have been applied in this chapter represent a trade-off between throughput and latency. While inter-segment parallelization is suitable to reduce latency of classification tasks which can be partitioned into sub-tasks that can be performed individually, intra-segment parallelization can still be used to preserve a high throughput when the classification of a single segment can not be partitioned. The hybrid of both is suitable to find a trade-off, when the number of possible partitions is still to small to utilize all available resources efficiently or when partitioning would result in highly unbalanced partitions. However, these approaches are limited to cases, where the processing of subsequent segments does not impose any dependencies. Furthermore, only equally sized segments have been considered in this section. When constraints on the maximum segment lengths can be guaranteed, presented approaches are applicable to worst-case behavior. However, without a restriction on segment length or for segments with a high variability in lengths, the proposed methods in this section are unsuitable and modeling and analysis approaches for dynamic behavior are necessary. Therefore, the following section will focus on dynamic behavior.

5 Context Awareness and Dynamic Behavior

The approaches and results described in this chapter have partially been published previously by the author in the following publications:

- Dataflow-Based Modeling and Performance Analysis for Online Gesture Recognition [G7]
- Time and Memory Efficient Online Piecewise Linear Approximation of Sensor Signals [G8]

This section investigates data-dependent behavior of the activity recognition chain. Data-dependent behavior complicates early design-time analysis by adding another level of complexity to the system. Furthermore, appropriate models of computation are studied that capture dynamic system behavior, which are still analyzable w.r.t. decisive extra-functional properties like throughput, latency, and processor utilization.

Possible cases of data-dependent behavior are feature or sensor selection strategies combined with context awareness. Here, different feature sets or sensors are evaluated based on different user situations, which are referred to as context. For some contexts, the possibilities of activities and gestures of the user to perform may be unlikely, which can be exploited by the recognition software by tailoring templates or feature sets and sensors to each specific context. Hence, specific stages or the entire ARC can be different for each context, which can be dynamically selected with context changes at run time.

Here, contexts can be of different nature like location, daytime, or the overall battery level of sensors [51]. Such systems, i.e., systems that dynamically change a predefined finite set of behaviors at run time, can be represented by multiple ARCs that can differ in some or even all stages. However, real-time capabilities and energy efficiency of configurations, mappings, and schedules must suffice in all possible contexts. To this end, different contexts need to be integrated into a model-based design. Corresponding analysis methods are necessary to estimate a real-time and energy-efficient execution in and across all possible contexts at design time. In the thesis at hand, this kind of dynamic behavior, which is characterized by a finite number of different scenarios that can change during run time, is referred to as scenario-based dynamic behavior. For each of the scenarios, a static ARC exists. The behavior of which can be modeled statically or cyclo-statically as described in the previous chapter. This type of scenario-based dynamic behavior is investigated in the first part of this chapter beginning with Section 5.1.

In the second part of this chapter (Section 5.2), examples of ARC stages with data-dependent execution times are discussed. Although SADF is expressive enough to capture discrete execution time distributions and scenario dependencies within a Markov chain, algorithms with a non-constant computational complexity due to data-dependent factors can only be modeled to some extend, i.e., worst-case abstractions. Finally, fully data-dependent behaviors can be modeled by dynamic dataflow models, which however, do not allow design-time analysis of important properties like latency and throughput in general [5]. Furthermore, algorithms with a non-constant computational complexity are not only difficult to analyze at design time, they impose a difficulty for real-time implementations on embedded devices with harshly constrained resources, e.g., embedded sensor sub-systems. For such implementations, the execution time needs to be constrained in order to guarantee real-time performance by possibly reducing their functionality. A prominent example in the gesture and activity recognition domain is the Piecewise Linear Approximation (PLA) of sensor signals. PLA techniques are used to reduce the amount of data that needs to be stored on the device or sent over wireless networks. However, state-of-the-art PLA algorithms impose linear (or worse) computational or memory complexities w.r.t. the lengths of the linear segments. Hence, they have a data-dependent execution time and memory demand. Rather than providing a model-based design and analysis approach for such dynamic behavior, an algorithmic solution is described in the second part of this chapter, leading to two novel PLA algorithms.

5.1 Scenario-Based Dynamic Behavior

In this section, scenario-based dynamic behavior in activity recognition systems is investigated with a possible modeling and design-time analysis of extra-functional properties like throughput, latency, and processor utilization. The notion of dynamic behavior that is considered in this section can be defined as a finite collection of quasi-static behaviors of an application. As an example, a feature or sensor selection approach for a context-aware activity recognition might be used for optimized trade-offs between recognition accuracy and energy consumption. For each context that the recognition system is able to derive, an optimized ARC can be designed, which is as computationally efficient as possible but still achieves a reasonable recognition accuracy for its application by using sensor or feature selection approaches. The corresponding ARC configurations are then selected at run time when context switches are detected by the system. In this example, a context can be considered as a scenario with a static behavior while context switches change the ARC to predefined configurations at run time. As a basic case study, the gesture recognition system in Chapter 4 can be considered. There, an EMA filter is applied to the derivative signal of squared vector lengths of the raw nine-dimensional sensor samples within a sliding window. This EMA filter is used to quantify fluctuations of the sensor signals and thus detect movements of the sensors. Thresholds on the filter value are used to detect the begin and end of a movement and thus exclude non-movement regions from the recognition process. However, the system in Chapter 4 has a static ARC and for non-movement segments a gesture recognition is still performed although the results are discarded and the corresponding windows are labeled as a null class. A possible optimization w.r.t. the utilization of computational resources and thus energy efficiency can be achieved by avoiding the execution of costly recognition algorithms within regions of non-movement in the sensor signals. As a result, the system can perform in two scenarios, i.e., *Full Power (FP)* and *Low Power (LP)*, on regions of the sensor signals with and without signal fluctuations, respectively.

The ARC of both scenarios is shown in Figure 5.1. The ARC depicted in Figure 5.1(a) is identical to the system described and implemented in Chapter 4. It is used as the ARC of scenario FP, in which the gesture recognition is performed on sliding windows, due to detected signal fluctuations. In situations without signal fluctuations, the scenario LP is performed. The ARC of scenario LP is depicted in Figure 5.1(b). In contrast to scenario FP, the gesture recognition is omitted in scenario LP, but the EMA calculation is still performed for the sliding windows. Furthermore, no *arg min* classification is performed, but the sliding windows are constantly labeled as null while executing in scenario LP.

Note that DTW and EMA calculations are placed in the same ARC stage in Figure 5.1. Although it would be more intuitive to separate both in two ARC stages with a corresponding order of EMA followed by the DTW calculations, they have been placed in the same ARC stage, in order to follow the mapping from Chapter 4. This allows to partially reuse the models and experimental implementation from Chapter 4 but has some effects on the accuracy and timing behavior as will be discussed in Section 5.1.6.

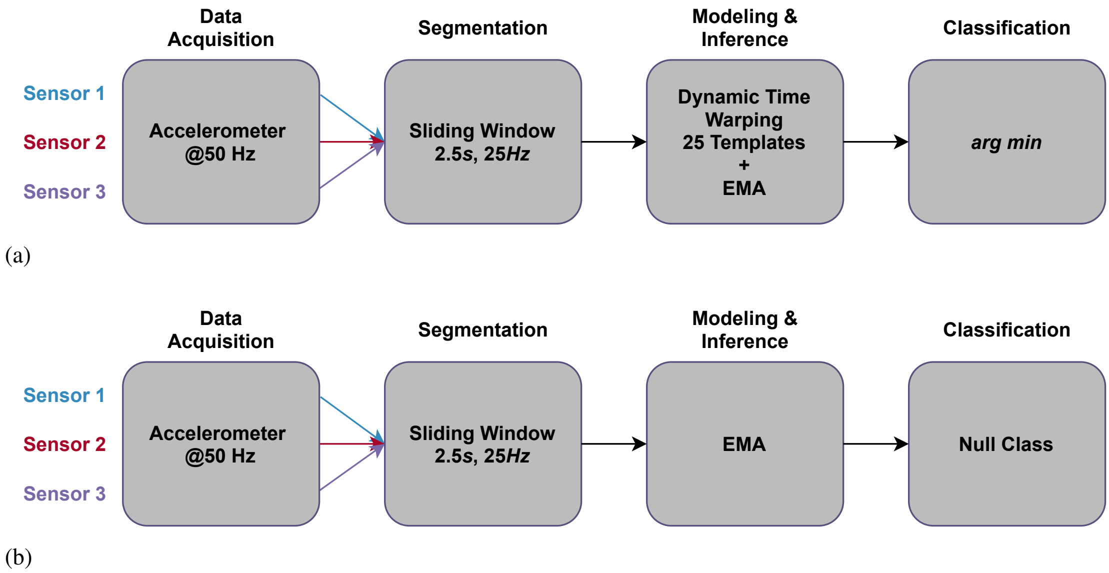


Figure 5.1: Activity recognition chain of the hand gesture recognition system in two scenarios, (a) FP with movements in sensor signals and (b) LP with non-movement in sensor signals.

In order to model the scenario-based dynamic behavior, different dataflow variants are available. Among others, Variable-Rate Phased Dataflow (VPDF) [115], SADF[19], Finite State Machine Scenario-Aware Dataflow (FSM-SADF) [5], and Parametrized FSM-SADF (PFSM-SADF) [116], are possible examples. A comparison between different MoCs, expressiveness, succinctness, implementation efficiency, and finally their analyzability can be found in [5]. Considering the requirement for analysis of extra-functional properties like throughput, latency, and processor utilization and the capability to capture CSDF behavior, SADF has been selected to model the ARCs in Figure 5.1.

5.1.1 Related Work

This chapter extends the CSDF models of different parallelization configurations of Chapter 4. As a result, most of the related work was already discussed regarding SDF and CSDF. Additionally to that, there exists some SADF-specific work which is related to the following section. In order to tightly capture a certain degree of control flow within dataflow-oriented streaming applications, SADF has been applied for MPEG-3 and MPEG-4 decoders in [23] and [20], respectively. Furthermore, SADF has also been used for selecting proper DVFS modes for streaming applications in [117]. Resource utilization with SADF has been addressed in [118] and [119]. In [118], a run-time configuration based on trade-offs between resource utilization, memory consumption, and energy consumption, that are assumed to be provided at design time, has been introduced. However, the evaluation of resource utilization at design time is not described. In the thesis at hand, processor utilization is analyzed from mapped and scheduled dataflow models at design time.

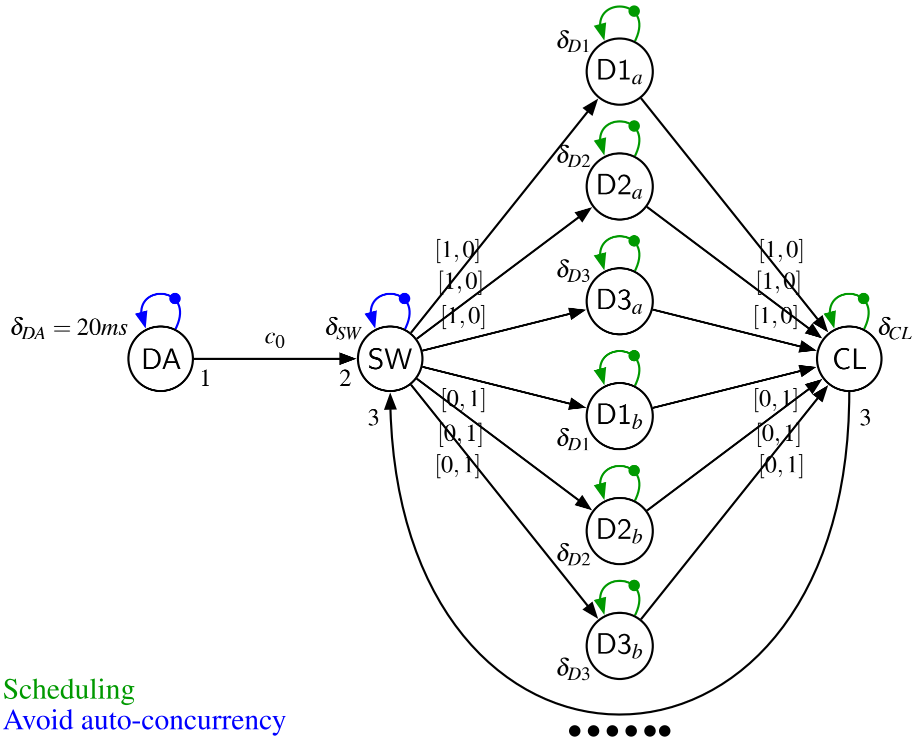


Figure 5.2: CSDF Analysis model in configuration C2 in scenario FP.

5.1.2 Modeling Scenario-Based Dynamic Behavior

Before modeling the dynamic changes between scenarios FP and LP, the adoption of the CSDF analysis model from Chapter 4 and how it is correspondingly modeled in SADF will be described first. In Figure 5.2, the CSDF analysis model of configuration C2, with two tiles, and three cores per tile is shown. It represents the behavior of scenario FP, and has to be converted to an SADF model, in order to model scenarios changes. For the sake of simplicity, the analysis model without message refinement is used to explain the adoption.

Cyclo-static behavior with SADF In order to model the CSDF behavior of sliding window dispatching to the tiles in a cyclo-static manner, actor SW and actor CL will execute in two different scenarios depending on their mode, i.e., a and b, sending a window to tile a or b, respectively. In scenario a, actor SW has a production rate of one on each channel to actors D1$_a$-D3$_a$, and a production rate of zero on each channel to actors D1$_b$-D3$_b$. Correspondingly, actor CL has a consumption rate of 1 on each channel from actors D1$_a$-D3$_a$ and a consumption rate of zero on each channel from actors D1$_b$-D3$_b$ in scenario a. In scenario b, the consumption rates of one and zero are swapped on channels to and from actors D1$_a$-D3$_a$ and D1$_b$-D3$_b$ for actors SW and CL, correspondingly. In order to change both scenarios in a cyclo-static manner, a detector is added to the graph. The detector det and its control channels which change the two scenarios are depicted in Figure 5.3. Note that the shown tokens on the control channels from det to SW and CL are intended to depict a controlled single

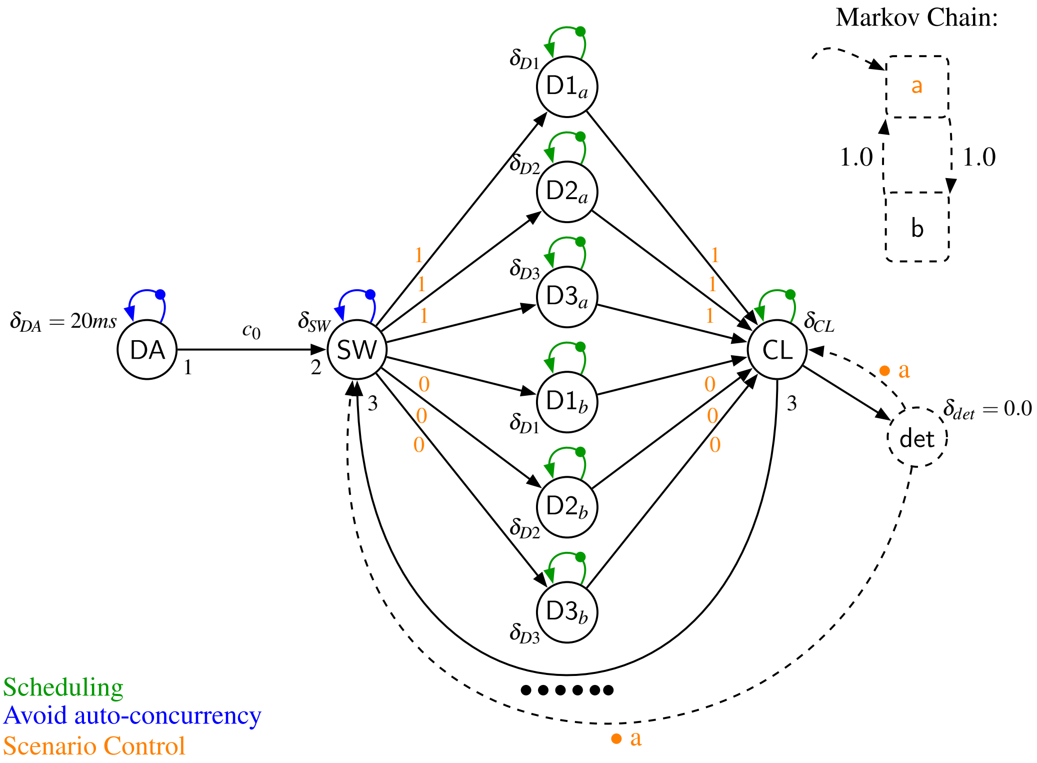
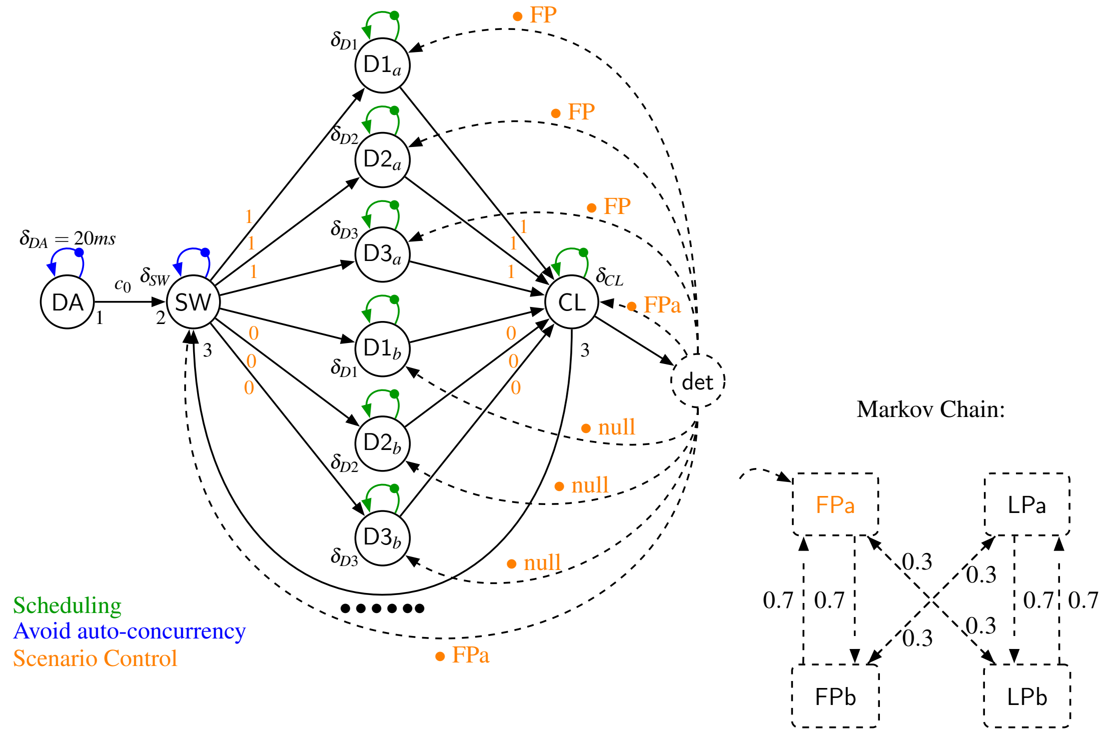


Figure 5.3: SADF analysis model in configuration C2 in scenario FP.

execution of SW and Cl in scenario a, instead of representing initial tokens. The actual number of initial tokens should match the number of tiles, in order to allow the tiles to execute interleaved in parallel. Furthermore, their control values should exhibit the corresponding cycling pattern of scenarios a and b. The corresponding Markov chain of detector det is shown in the top right corner in Figure 5.3. The transition probabilities between the states are 1.0, and the transition probabilities on the self-edges are 0.0 and therefore omitted. The initial state is marked as a. With the given parameters, the Markov chain actually resembles a finite state machine cycling between a and b, with a as the initial state. Note that the detector Markov chain states represent its sub-scenarios. The actual token values send over the control channels control the actual scenario of each controlled actor. For the CSDF behavior replication, Markov chain sub-scenarios and corresponding control token values (actor scenarios) are equal. As a result, the cyclo-static behavior of the original CSDF graph is achieved, controlled by the additional detector det.

Alternatively to introducing a separate detector, actors SW and CL could also be modeled as detectors themselves, each having the same Markov chain as depicted in Fig 5.3 controlling their scenarios locally. However, their behavior needs to be adapted further when integrating scenario changes between FP and LP into the model, which are conceptually controlled in the evaluation stage of the software. Therefore, detector det is modeled separately. By annotating detector det with an execution time of zero, the timing behavior of the evaluator core is not affected by the added detector and scheduling edges do not need to be adapted.



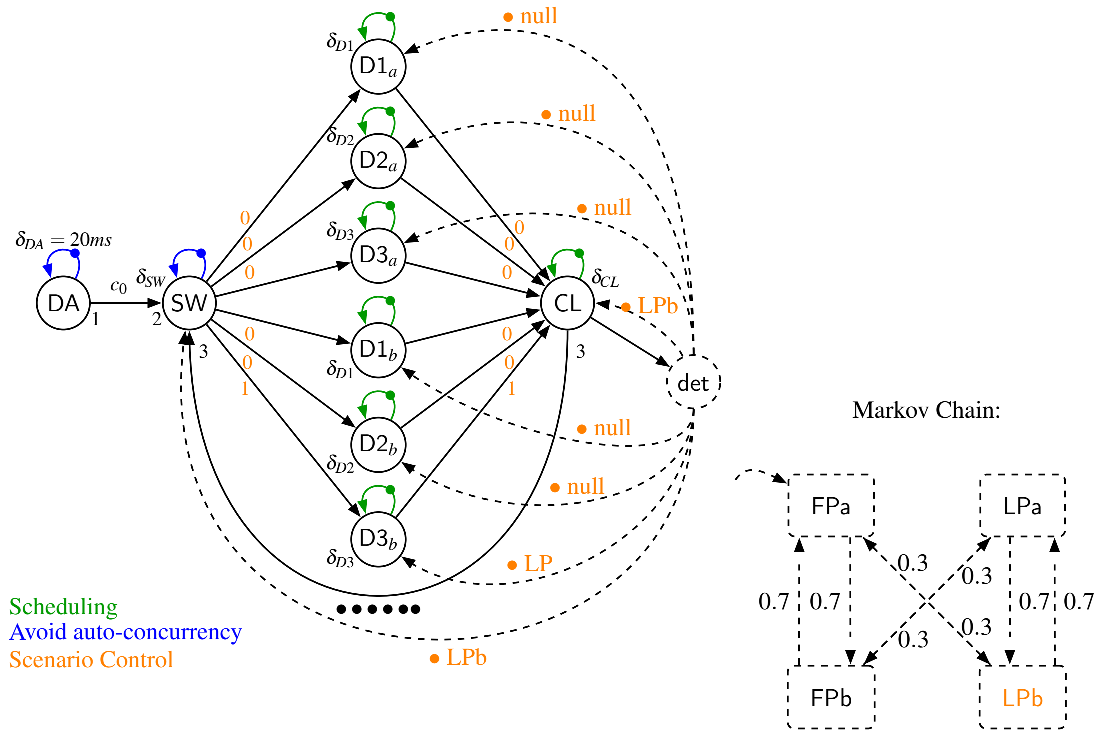
(a)

(b)

Figure 5.4: SADF analysis model in configuration C2 in scenario FP1 (a) and corresponding Markov chain (b).

By the aforementioned changes, the SADF model imposes an identical timing and communication behavior as the corresponding CSDF model in Figure 5.2, except for the additional control tokens, which have to be excluded from possible data rate analysis techniques on the hardware channels. In the next step, the integration of scenarios FP and LP into the SADF model will be described.

Scenario modeling with SADF In order to model the scenario changes between LP and FP, actors D1$_a$ - D3$_b$ can execute in 2 different scenarios, namely FP and LP (plus a third null scenario, which will be introduced shortly). The scenario transition probabilities are example probabilities, with $p = 0.7$ for staying in FP or LP, respectively, and $p = 0.3$ for changing from FP to LP or vice versa. In order to acquire annotations for the Markov chain transition probabilities for activity recognition systems, a priori knowledge, expectations, or case studies about the system behavior need to be available, either from specifications of the system or in form of user studies that represent expected or corner cases of the system. These can be analyzed w.r.t. the frequency or ratio of staying in the specified scenarios and transitioning from such into other scenarios. This procedure has been conducted on test sequences for offline evaluations of recognition accuracy in the thesis at hand and will be shown for the analysis models in Section 5.1.3. The resulting Markov chain is a product of the Markov chain defining the cyclo-static behavior from Figure 5.3 and the aforementioned scenarios LP and FP with their corresponding transition probabilities. The resulting Markov chain is depicted in Figure 5.4(b). The structure allows a change from FP to LP at any state with the aforementioned transition probabilities,



(a)

(b)

Figure 5.5: SADF analysis model in configuration C2 in scenario LPb (a) and corresponding Markov chain (b).

but only allows progressions that exhibit the cyclo-static pattern between *a* and *b*. The corresponding control tokens for a single execution of all controlled actors of the SADF graph in sub-scenario FPa are shown in Figure 5.4(a). Note that while actors $\mathsf{D1}_a$-$\mathsf{D3}_a$ receive the corresponding FP control tokens, actors $\mathsf{D1}_b$-$\mathsf{D3}_b$ receive a null control token. This is an extra scenario for possibly all controlled actors, which specifies a production and consumption rate of zero on all incoming and outgoing data channels, as well as an execution time of zero. The scenario null is implemented for the sake of consistency w.r.t. the repetition vector entry of the corresponding actor, allowing it to fire without influencing the timing or communication behavior of the model.

Furthermore, note that the scenarios of actors $\mathsf{D1}_a$-$\mathsf{D3}_b$ only specify the scenario LP or FP (and null) for the corresponding sub-scenario of det, while their cyclo-static execution is controlled by placing the right control tokens on the control channels from det. However, for actors SW and CL, the scenarios actually match all sub-scenarios of det, namely the product of FP and LP and *a* and *b*, as not only the cyclo-static behavior needs to be controlled, but also the production rates on channels to a specific tile depending on its scenario (FP or LP).

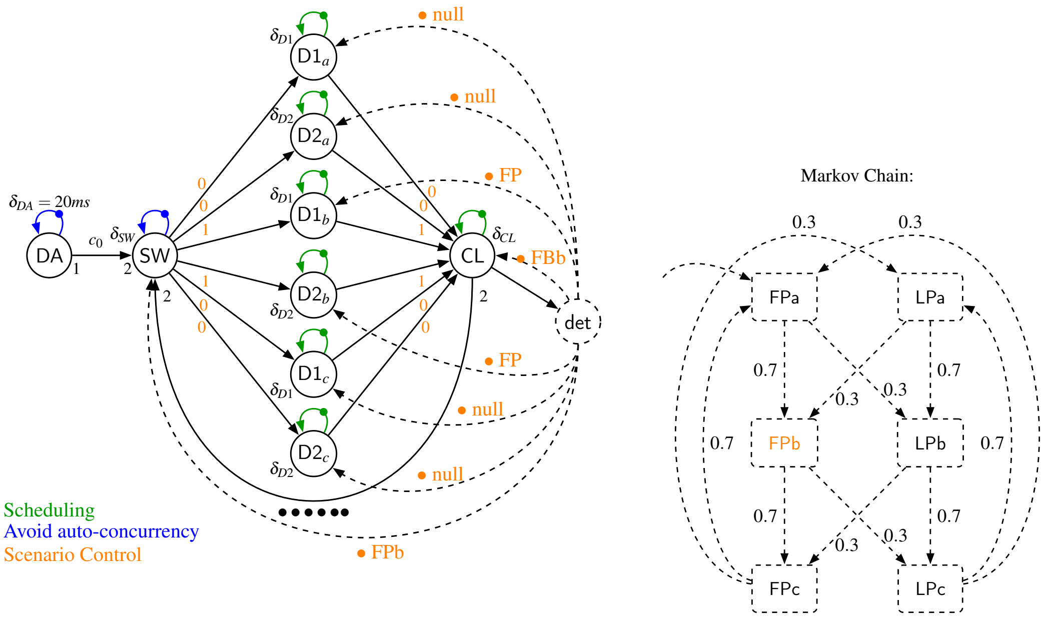
In Figure 5.4(a), the production rate on each channel to actors belonging to tile 1 is simply 1, as in FP all cores of that tile are performing DTW executions. However, in scenario LP only the last worker core of a tile is actively executing the EMA calculation and therefore the production rate on the channel to that actor is 1 while all others are 0. The corresponding control tokens on the actors $\mathsf{D1}_x$-$\mathsf{D3}_x$ are

Table 5.1: Scenario control tokens for each Markov chain sub-scenario in Figure 5.5(b).

Sub-Scenario	SW	CL	D1$_a$	D2$_a$	D3$_a$	D1$_b$	D2$_b$	D3$_b$
FPa	FPa	FPa	FP	FP	FP	null	null	null
FPb	FPb	FPb	null	null	null	FP	FP	FP
LPa	LPa	LPa	null	null	LP	null	null	null
LPb	LPb	LPb	null	null	null	null	null	LP

LP for the last core and null for the remaining cores of each tile, respectively. This is depicted in Figure 5.5 representing a single execution of all controlled actors in sub-scenario LPb. A summary of all control tokens sent to controlled actors for each sub-scenario encoded in the Markov chain states is shown in Table 5.1.

For the different configurations of the model, the CSDF behavior exhibits different number of modes that are cyclo-statically changing, i.e., the number of modes of SW and CL equals the number of tiles. Hence, the Markov chain needs to be constructed for each configuration individually, by building the product of a cyclic Markov chain with as many states as tiles and the two scenarios FP and LP. For the sake of comprehensibility, the SADF analysis model and corresponding Markov chain of the gesture recognition system in configuration C3 is shown in Figure 5.6(a), again with highlighted control tokens for a single execution of all controlled actors.



(a) (b)

Figure 5.6: SADF analysis model in configuration C3 in scenario FPb (a) and corresponding Markov chain (b).

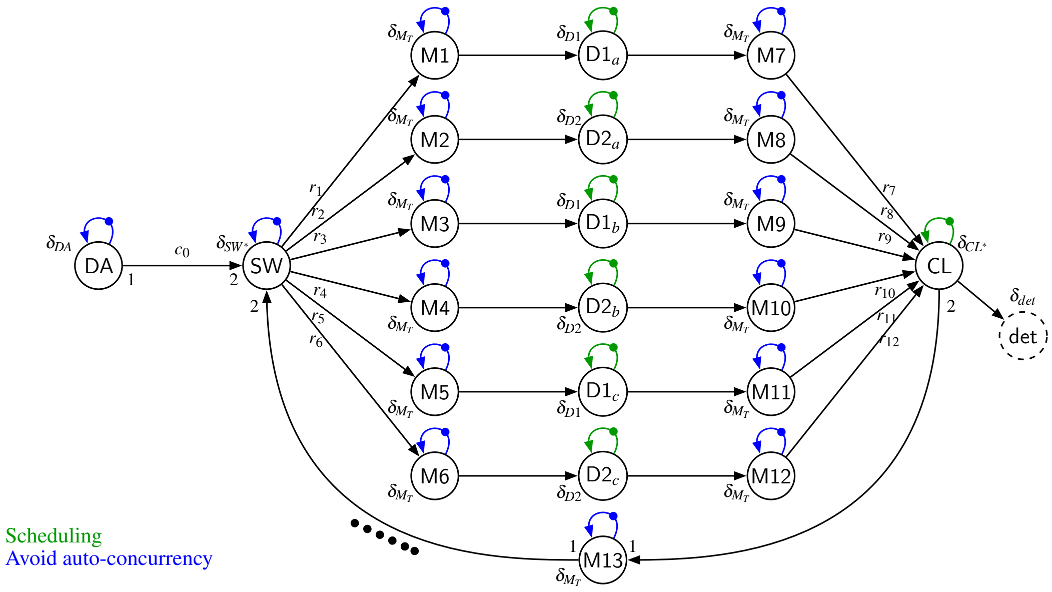


Figure 5.7: Refined SADF analysis model in configuration C3.

5.1.3 Analysis Models

The previous section showed the basic principle on how to integrate scenario changes as well as cyclo-static behavior into the analysis models. However, the analysis model in the previous section did not integrate the refinement regarding message communication between different worker cores. As a result, the analysis models need to be extended for that purpose, in order to get comparable results with the models from Chapter 4. To this end, the additional step from the SADF model of configuration C3 from the previous section towards a corresponding analysis model with communication refinement will be explained next.

The basic changes that need to be made are the integration of actors M1-M13 on the channels between actors executing on different worker cores. These are basically the channels from and to actors D1$_a$-D2$_c$ and on the feedback channel between actor CL and SW, with their corresponding executing time capturing the communication throughput. The additional execution times of sending and receiving messages on each core are integrated, as explained in Chapter 4, accordingly.

However, with introducing scenarios for actors SW, D1$_a$-D2$_c$, and CL, additional scenarios for M1-M12 have to be introduced as well. As the sizes of the messages are not changing in different scenarios and the throughput of hardware communication channels is not changing either, the only scenario that needs to be integrated is an additional null scenario. Any time, one of the actors D1$_a$-D2$_c$ is executing in a null scenario and the corresponding production and consumption rates on its channels from SW and to CL are zero, the corresponding actor Mx is receiving a null control token as well. Otherwise, the actors M1-M12 are executing their *default* scenario, regardless of the scenario of the corresponding D$_x$ actor. The final SADF analysis model of configuration C3 is shown in Figure 5.7. For the sake of readability, control edges from detector det to all controlled actors are omitted in Figure 5.7 and rather summarized in Table 5.2, together with the corresponding control tokens sent in each sub-scenario and initial token distribution on all control channels. Note that the initial scenario for each analysis model

Table 5.2: Scenario control tokens of the SADF analysis model in configuration C3.

Actor	FPa	FPb	FPc	LPa	LPb	LPc	Initial Tokens
SW	FPa	FPb	FPc	LPa	LPb	LPc	FPa,FPb,FPc
M1	def	null	null	null	null	null	def,null,null
M2	def	null	null	def	null	null	def,null,null
M3	null	def	null	null	null	null	null,def,null
M4	null	def	null	null	def	null	null,def,null
M5	null	null	def	null	null	null	null,null,def
M6	null	null	def	null	null	def	null,null,def
$D1_a$	FP	null	null	null	null	null	FP,null,null
$D2_a$	FP	null	null	LP	null	null	FP,null,null
$D1_b$	null	FP	null	null	null	null	null,FP,null
$D2_b$	null	FP	null	null	LP	null	null,FP,null
$D1_c$	null	null	FP	null	null	null	null,null,FP
$D2_c$	null	null	FP	null	null	LP	null,null,FP
M7	def	null	null	null	null	null	def,null,null
M8	def	null	null	def	null	null	def,null,null
M9	null	def	null	null	null	null	null,def,null
M10	null	def	null	null	def	null	null,def,null
M11	null	null	def	null	null	null	null,null,def
M12	null	null	def	null	null	def	null,null,def
CL	FPa	FPb	FPc	LPa	LPb	LPc	FPa,FPb,FPc

is defined as FP, which will be triggered for the first N iterations, with N denoting the configuration number or number of tiles, respectively.

The SADF analysis models of all other configurations are adapted accordingly. The execution times of all actors are composed in the same way as in Chapter 4 except for actors in scenario LP, in which the DTW execution times are excluded, and except for scenario null, in which all execution times are zero. The measurements for annotating the scenario LP are shown in Table 5.3. Note that only functions, the execution time of which is affected by the scenario LP, have been measured. For all remaining functions, the results from Table 4.4 have been reused.

Table 5.3: Execution time t_{ex} measurements for SADF analysis model annotation in scenario LP.

Name	#	$\bar{t}_{ex}$ $[ns]$	$s_{t_{ex}}$ $[ns]$	$CV_{t_{ex}}$	$min\ t_{ex}$ $[ns]$	$max\ t_{ex}$ $[ns]$
EMA	25,237	25,960	0	0.00 %	25,960	25,960
CL	25,237	106	13	12.26 %	92	148
MCP	25,237	15,039	38	0.25 %	15,014	15,234
MWB	25,237	15	0	0.00 %	15	15

Table 5.4: Markov chain transition probabilities acquired from a test sequence of gestures.

(LP,LP)	(LP,FP)	(FP,LP)	(FP,FP)
99.05 %	0.95 %	3.17 %	96.83 %

In order to analyze long-run average worst-case processor utilization, scenario transition probabilities have been acquired from a selected test sequence of gestures, that has been recorded for the purpose of recognition accuracy evaluation. The test sequence has been analyzed offline with the implemented EMA filter, and evaluated w.r.t. the number of times that the system will stay in each of the LP and FP scenarios and the number of times that the system will switch from LP to FP and from FP to LP, for that particular input sequence. The transition probability annotation of a particular scenario transition (ψ_1, ψ_2) has been calculated as the ratio of the number of analyzed scenario transitions (ψ_1, ψ_2) and the total number of transitions from ψ_1 into any scenario. The resulting transition probabilities are shown in Table 5.4.

5.1.4 Model Analysis

In order to verify real-time performance for SADF-based analysis models, a slightly different approach is used than in the previous chapters. While for SDF and CSDF graphs, the actor throughput $TH(\mathsf{DA})$ represents the maximum throughput at which it can be scheduled such that the entire system execution is bounded in terms of finite buffer size usage on all channels, the actor throughput in SADF represents a long-run average metric considering scenario transition probabilities. For a corresponding real-time verification, the worst-case actor throughput, e.g., as defined by [22] for FSM-based SADF graphs, is of importance, that is however not implemented for SADF graphs in the module [20] extending SDF^3 [13]. As a solution, the maximum inter firing latency ($max\, IFL(\mathsf{DA})$) of actor DA is utilized to analyze real-time performance.

If $max\, IFL(\mathsf{DA})$ of an SADF graph is greater than its annotated execution time, there exists a path in the execution space, for which the system throughput is smaller than the data input throughput. As a result, the SADF graph does not represent a real-time enabled system. However, since the IFL of actor DA cannot be smaller than its execution time annotation (which is constant in all scenarios), due to the eliminated auto-concurrency, a maximum IFL that matches its execution time annotation indicates, that (at least one) critical path w.r.t. its actor throughput and thus graph throughput lies on DA itself. In other words, the IFL of actor DA is constant across all possible execution paths. This allows to infer real-time behavior of the system modeled with SADF graphs by observing the maximum IFL of its input data process, i.e., $max\, IFL(\mathsf{DA})$. Furthermore, it follows, that the actor throughput $TH(\mathsf{DA})$ is constant across all possible scenario transitions in case $max\, IFL(\mathsf{DA})$ matches its execution time annotation.

Note that this is not valid for SADF graphs in general, and limited to real-time-enabled models in which the input data process is modeled explicitly with a constant execution time in all scenarios with eliminated auto-concurrency. This can either represent a constant input data process or its worst-case behavior, i.e., representing the highest input data throughput that can be expected in the anticipated application.

The long-run average processor utilization $\widehat{U}(p)$ of processor core $p \in P$ is its average active time (executing actors) along the execution state space considering scenario transition probabilities, divided by the total elapsed time. The input actor of analysis models that meet the real-time criterion of the thesis at hand, is known to have a constant inter firing latency. This can be utilized as a reasonable time basis for the total elapsed time during a graph iteration for each particular detector scenario $\psi_{v_d} \in \Psi_{v_d}$. Let V' denote the set of actors that are mapped to processor p, i.e., $V' = \{v \in V \mid m_p(v, p) = 1\}$. For each detector scenario $\psi_{v_d} \in \Psi_{v_d}$, the active time of processor $p \in P$ induced by a particular actor $v \in V'$ is then the accumulated execution time of each firing of $v \in V$ during its sub-scenario sequence in ψ_{v_d}, i.e., $\delta(v, subs(v, \psi_{v_d}))$, corresponding to the iteration period. The processor utilization of p induced by v in scenario ψ_{v_d} is then the active time divided by the iteration period, which is the accumulated execution time of actor DA during its sub-scenario sequence in ψ_{v_d}, i.e., $\delta(\mathsf{DA}, subs(\mathsf{DA}, \psi_{v_d}))$, or alternatively its inter-firing latency multiplied by the number of repetitions in ψ_{v_d}, that is, $\frac{1}{TH(\mathsf{DA})} \cdot \gamma_G(\mathsf{DA}, \psi_{v_d})$. The average processor utilization $\widehat{U}(v, p)$ of processor $p \in P$ induced by an actor $v \in V'$, w.r.t. the scenario occurrence probabilities $\pi(v_d, \psi_{v_d})$ of detector scenarios $\psi_{v_d} \in \Psi_{v_d}$, is calculated by:

$$\widehat{U}(v, p) = \frac{\sum_{\psi_{v_d} \in \Psi_{v_d}} \pi(v_d, \psi_{v_d}) \cdot \delta(v, subs(v, \psi_{v_d}))}{\sum_{\psi_{v_d} \in \Psi_{v_d}} \pi(v_d, \psi_{v_d}) \cdot \delta(\mathsf{DA}, subs(\mathsf{DA}, \psi_{v_d}))}. \tag{11}$$

Equation (11) is a straight forward derivation from a combined metric of sample average (average active time per iteration) and an event rate (average number of firings per time unit, i.e., actor throughput) from [23]. The derivation of Equation (11), which is based on detector scenarios, is limited to strongly consistent and strongly dependent SADF analysis models that meet the real-time criterion as defined in the thesis at hand, i.e., $max\, IFL(\mathsf{DA}) = \delta(\mathsf{DA}, \psi_{\mathsf{DA}})$ with $|\{\psi_{DA} \in \Psi_{DA}\}| = 1$.

Note that for real-time configurations $max\ IFL(\mathsf{DA}) = \overline{IFL}(\mathsf{DA})$, and thus, $\delta(\mathsf{DA}, subs(\mathsf{DA}, \psi_{v_d}))$ equals $\frac{1}{TH(\mathsf{DA})} \cdot \gamma_G(\mathsf{DA}, \psi_{v_d})$. Furthermore, $\frac{1}{TH(\mathsf{DA})}$ is constant across all scenarios and corresponding transitions. As a result, Equation (11) can be rewritten to:

$$\widehat{U}(v, p) = TH(\mathsf{DA}) \cdot \frac{\sum_{\psi_{v_d} \in \Psi_{v_d}} \pi(v_d, \psi_{v_d}) \cdot \delta(v, subs(v, \psi_{v_d}))}{\sum_{\psi_{v_d} \in \Psi_{v_d}} \pi(v_d, \psi_{v_d}) \cdot \gamma_G(\mathsf{DA}, \psi_{v_d})}. \tag{12}$$

Finally, to acquire the total average processor utilization $\widehat{U}(p)$ of each processor $p \in P$, the average processor utilizations $\widehat{U}(v, p)$ of p induced by actors $v \in V'$ need to be accumulated for all $v \in V'$:

$$\widehat{U}(p) = TH(\mathsf{DA}) \cdot \sum_{v \in V'} \frac{\sum_{\psi_{v_d} \in \Psi_{v_d}} \pi(v_d, \psi_{v_d}) \cdot \delta(v, subs(v, \psi_{v_d}))}{\sum_{\psi_{v_d} \in \Psi_{v_d}} \pi(v_d, \psi_{v_d}) \cdot \gamma_G(\mathsf{DA}, \psi_{v_d})}, \tag{13}$$

with V' denoting the set of actors $v \in V$ that are mapped to processor $p \in P$, i.e., $V' = \{v \in V \mid (v, p) \in M_V\}$. The throughput $TH(\mathsf{DA})$ and scenario occurrence probabilities $\pi(v_d, \psi_{v_d})$ of all detector scenarios $\psi_{v_d} \in \Psi_{v_d}$ can be acquired from the SADF module [20] and the extension described in [G7], respectively. The repetition vector $\gamma_G(\mathsf{DA}, \psi_{v_d})$ can be acquired from transformation of each single scenario ψ_{v_d} into a corresponding SDF graph with `sdf3transform-sadf` to fix the detector scenario and `sdf3convert-sadf-sdf` for the transformation into a corresponding SDF graph [20]. The corresponding SDF graphs can then be analyzed w.r.t. their repetition vector with SDF analysis tools [13]. Execution times and sub-scenario sequences are directly acquired from the specified analysis model.

5.1.5 Experiments

For evaluating the model-based results, the same implementation as introduced in Section 4.5 has been used. In order to integrate the scenarios LP and FP, the sliding window is only sent to the last core of a tile in scenario LP for EMA calculation. Scenario FP corresponds to the original implementation as in Section 4.5. The selection of the scenario is based on the EMA result of the last evaluated window on the evaluator core. This is due to the original application graph structure in Section 4.3, in which EMA and DTW instances for a sliding window can be calculated in parallel without interdependencies. A more straightforward application design would calculate the EMA before any DTW calculation is performed and possibly on the initiator core. However, as the integration of a low power scenario was not initially intended for the recognition system in Chapter 4, and for the sake of reusing experimental implementations, the application structure is kept in line with that introduced in Section 4.3. As a result, the EMA of the currently evaluated sliding window is used for the scenario choice of the next sliding window that will be dispatched. Note that this implies a latency between detecting an end point of a gesture and the actual time of ending the DTW calculations for subsequent, but yet already dispatched sliding windows. This will be explained in more depth in the following evaluation.

5.1.6 Evaluation

The evaluations are based on the same test sequences which have been used in Section 4.5. However, for the average processor utilization, only one test sequence has been used, i.e., for which the scenario occurrence probabilities in Table 5.4 have been analyzed.

Real-time behavior and latency The model-based analysis results indicate a real-time behavior in all configurations of the system, when the execution is fixed to scenario LP. The results are listed in Table 5.5. However, with scenario transitions, configuration C1 is not performing in real time, as scenario FP is not real-time enabled itself (cf. Section 4.6.2). The model-based real-time analysis results with scenario transitions are listed in Table 5.6. In the experiments it could be verified that in configuration C1 every other window is skipped, when the system operates in scenario FP. Therefore, this configuration is not real-time enabled. In all other configurations, no windows are skipped due to sufficient throughput of the system.

Furthermore, the maximum response times acquired from the analysis models in scenario LP and in executions with scenario transitions between LP and FP are listed as well in Table 5.5 and Table 5.6, respectively. Note that the latter correspond to the results of scenario FP from Chapter 4.

Table 5.5: Maximum response time $max\,\widehat{t_{rsp}}$, actor throughput $TH(DA)$, and real-time performance analyzed from the SADF analysis models in scenario LP.

Configuration	$max\,\widehat{t_{rsp}}$ $[ns]$	$TH(DA)$ $[s^{-1}]$	Real Time
C1	83,217	50.00	Yes
C2	83,217	50.00	Yes
C3	83,217	50.00	Yes
C6	83,217	50.00	Yes

Table 5.6: Maximum response time $max\,\widehat{t}_{rsp}$, actor throughput $TH(DA)$, and real-time performance analyzed from the SADF analysis models with scenario transitions.

Configuration	$max\,\widehat{t}_{rsp}$ $[ns]$	$TH(DA)$ $[s^{-1}]$	Real Time
C1	41,155,100	49.83	No
C2	78,155,500	50.00	Yes
C3	119,190,000	50.00	Yes
C6	227,264,000	50.00	Yes

The measured response times in scenario LP are appended in Table B.1. A comparison is depicted in Figure 5.8. The corresponding absolute and relative differences are listed in Table 5.7. In comparison, the model-based maximum response times $max\,\widehat{t}_{rsp}$ are up to 1.19 % smaller than the measured results $max\ t_{rsp}$ in scenario LP.

Processor utilization in scenario LP The processor utilization in scenario LP acquired from analysis models is depicted in Figure 5.9. The underlying values are appended in Table B.2 as supplementary information. The average processor utilization among all worker cores in scenario LP, acquired from the analysis models, is approximately 0.02 %. The measured processor utilization in scenario LP is depicted in Figure 5.9 as well. The underlying values are appended in Table B.3. It can be seen, that the implemented low power scenario can reduce the processor utilization to approximately 0.022 % in situations where no gestures are performed. This is a substantial saving compared to scenario FP from Table 4.11 in Chapter 4. As a result, the long-run average worst-case processor utilization highly depends on the scenario transitions and thus on the usage of the sensor glove.

However, a comparison of the model-based results reveals a much higher difference to the measured processor utilization, when compared to the model accuracy in Chapter 4. Furthermore, the model-based estimates on processor utilization in scenario LP underestimate the implementation, although they are supposed to provide upper bounds. The corresponding relative differences are listed in Table 5.8. It can be seen that the model-based results underestimate the measured results by approximately 11.4 % to 11.8 %. This rather high deviation is partially induced by additional context change overheads between threads that are executed on the worker cores. These overheads increase the active times of the worker cores, which has not been taken into account into the annotation of actors in the analysis models.

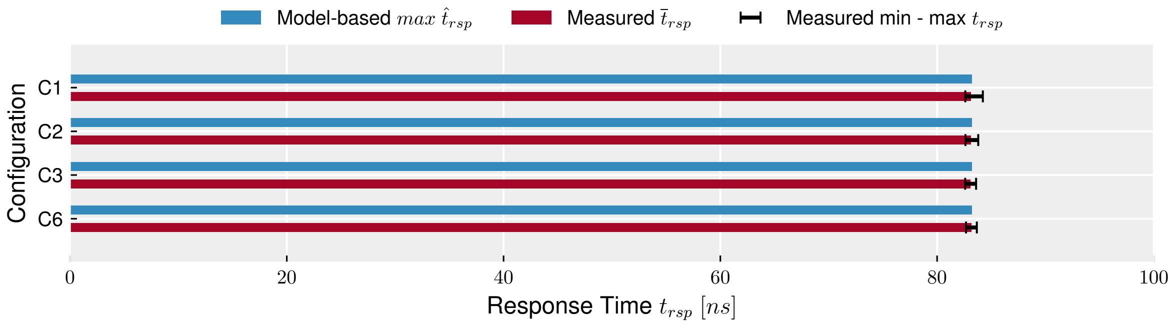


Figure 5.8: Measured and model-based maximum response time results in scenario LP.

Table 5.7: Comparison of measured ($max\ t_{rsp}$) and model-based ($max\ \widehat{t_{rsp}}$) maximum response time results in scenario LP.

Cfg.	$max\ \widehat{t_{rsp}}\ [ns]$	$max\ t_{rsp}\ [ns]$	$\Delta\ max\ t_{rsp}\ [ns]$	$\Delta_r\ max\ t_{rsp}$
C1	83,217	84,221	−1,004	−1.19 %
C2	83,217	83,795	− 578	−0.69 %
C3	83,217	83,597	− 380	−0.45 %
C6	83,217	83,665	− 448	−0.53 %

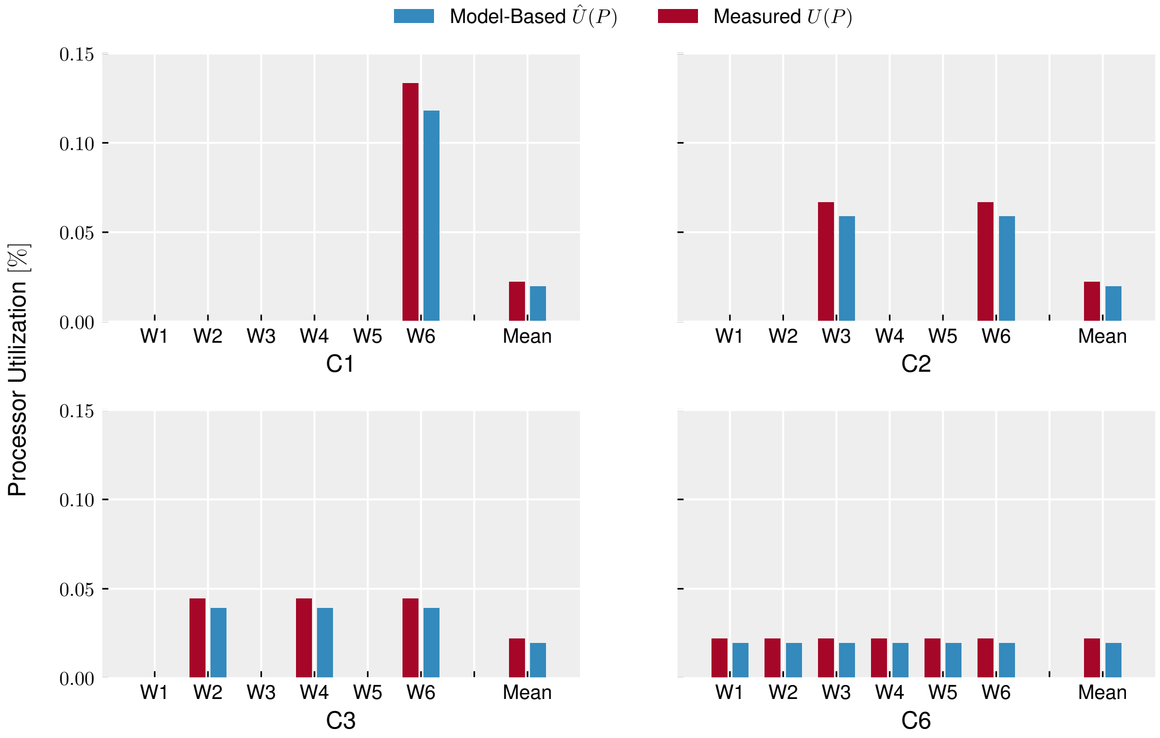


Figure 5.9: Measured and model-based processor utilization in scenario LP.

Table 5.8: Relative difference of processor utilization between model-based and measured results in scenario LP.

Cfg.	$\Delta_r\ U(\text{W1})$	$\Delta_r\ U(\text{W2})$	$\Delta_r\ U(\text{W3})$	$\Delta_r\ U(\text{W4})$	$\Delta_r\ U(\text{W5})$	$\Delta_r\ U(\text{W6})$	$\Delta_r\ \overline{U}$
C1	/	/	/	/	/	−11.482 %	−11.482 %
C2	/	/	−11.615 %	/	/	−11.615 %	−11.615 %
C3	/	−11.549 %	/	−11.549 %	/	−11.549 %	−11.549 %
C6	−11.747 %	−11.747 %	−11.747 %	−11.747 %	−11.747 %	−11.747 %	−11.747 %

Since the LP scenario results in a substantially smaller processor utilization, due to its comparably small active time of approximately $50\,\mu s$, the context change overhead with $4.2\,\mu s$ on average has a significantly higher influence than on the active time in scenario FP (approximately $70\,ms$ to $225\,ms$). An additional analysis of the task profiling logs reveals, that without the context change overhead the model-based results would deviate by $3.9\,\%$ to $4.4\,\%$ from the measurements. The resulting relative differences between model-based and measured processor utilizations without context change overheads are listed in Table 5.9. After removing the influence of context change overheads, a minor increase in the relative differences between model-based and measured results with an increasing number of deployed tiles can still be observed in the last column of Table 5.9. A linear regression of all active times of the worker cores that are computing in scenario LP w.r.t. the number of deployed tiles (and thus cores) shows a significant linear dependency w.r.t. a significance threshold of $\alpha = 0.05$, with $\beta_1 = 46.67\,ns/\#cores,\ t(10) = 61.97,\ p < 0.001$. Furthermore, the number of deployed cores explains a significant proportion of the variance in the active time in scenario LP on the worker cores, with $R^2 = 0.997,\ F(1,10) = 3840,\ p < 0.001$. These results suggest a dependency between active time of processor cores in scenario LP and the number of deployed tiles.

All worker cores receive the exact same binary. The execution time of the LP code on different worker cores does not have any obvious dependency on other worker cores or on the number of worker cores that are computing, since, although different sliding windows are dispatched to different cores, e.g., configuration C6, the processing of a window (approximately $50\,\mu s$) is already finished before a new window is dispatched after $40\,ms$. As a result, the involved worker cores do neither compute, perform memory operations, nor communicate concurrently, which could be substantiated by evaluating the task profiling information. As a consequence, the increase in active time is assumed to be caused by the task profiling routines itself. In that case, the intercept of the above mentioned linear regression represents an estimate of active time without that influence. This would leave a relative difference between model-based and measured results of $\Delta_r = -3.8\,\%$ for each configuration.

In general, the aforementioned influences are explained by the measuring system itself and the context changes between threads that should have been taken into account into model annotations given the small range of execution times. The reason why this has not been performed is consistency with the models for the static FP scenario in Chapter 4. The introduction of scenario LP has been performed after the modeling, implementation, and analysis of the models and experiments in Chapter 4 and consistently embedded into the SADF models. However, the acquired results suggest that for annotations of comparably low execution times w.r.t. the performance and domain-specific architectural optimizations of the target hardware, overheads in execution time can have a significant influence,

Table 5.9: Relative difference of processor utilization in scenario LP between model-based and measured results without context change overheads.

Cfg.	$\Delta_r\, U(\mathsf{W1})$	$\Delta_r\, U(\mathsf{W2})$	$\Delta_r\, U(\mathsf{W3})$	$\Delta_r\, U(\mathsf{W4})$	$\Delta_r\, U(\mathsf{W5})$	$\Delta_r\, U(\mathsf{W6})$	$\Delta_r\, \overline{U}$
C1	/	/	/	/	/	$-3.920\,\%$	$-3.920\,\%$
C2	/	/	$-3.998\,\%$	/	/	$-3.998\,\%$	$-3.998\,\%$
C3	/	$-3.998\,\%$	/	$-3.998\,\%$	/	$-3.998\,\%$	$-3.998\,\%$
C6	$-4.464\,\%$	$-4.464\,\%$	$-4.464\,\%$	$-4.464\,\%$	$-4.464\,\%$	$-4.464\,\%$	$-4.464\,\%$

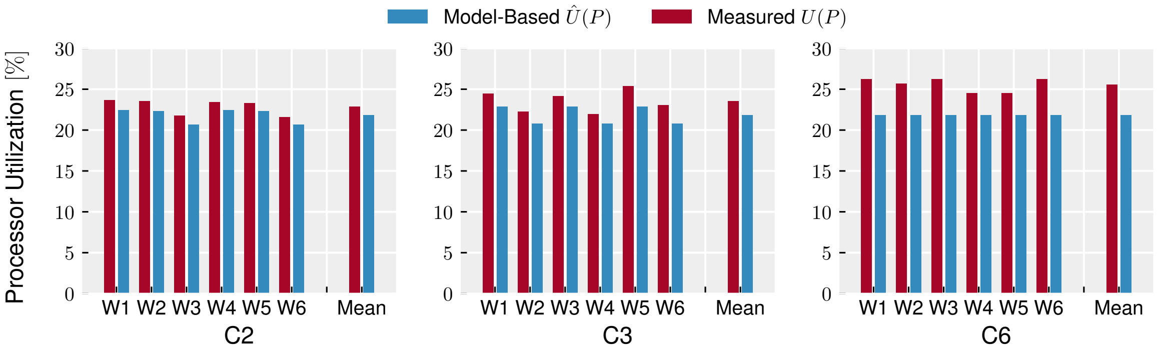


Figure 5.10: Measured and model-based processor utilization with scenario transitions.

which have to be taken into account into the modeling by refined models or by additional worst-case abstractions. Additionally, as stated in Section 3.3.3, typically WCETs of actor code are acquired by analysis, e.g., on the assembler code, and provide worst-case guarantees. However, in the thesis at hand, actor annotations have been measured on the target hardware, and reflect worst-case observed execution times of the individual code segments. When implemented into the final binaries without timestamp functions to measure individual execution times, the source code was re-compiled. Since the compiler of the TMS320C6678 architecture performs optimizations which are rather unpredictable based on the surrounding source code, the acquired annotations can indeed be smaller than the corresponding code that appears in the final implementation. This has to be taken into account, and if possible, actor code should be kept pre-compiled and only linked into the final application in order to avoid differences due to compiler optimizations.

The remaining underestimation of approximately $-3.8\,\%$ indicates that the measured WCETs for model annotation were not worst-case representatives, caused by the aforementioned compiler optimizations when actor annotations have been measured. Furthermore, additional run-time overheads due to context changes have not been taken into account into the annotations of the model, which results in rather high relative differences between model-based and measured results for scenario LP of up to $11.7\,\%$ in the conducted experiments.

Processor utilization with scenario transitions The model-based results of long-run average worst-case processor utilizations with scenario transitions are depicted in Figure 5.10. The underlying values are appended in Table B.4. The effect of the reduced processing in situations without movement of the sensor glove, can be observed in the long-run average worst-case processor utilization of all worker cores. The model-based estimations indicate a reduction by approximately $76.9\,\%$ compared to scenario FP in Table 4.8 from Chapter 4. This reduction in expected worst-case processor utilization is induced by the reduced processing for windows that do not contain gestures. The corresponding processor utilizations measured from the experimental implementation are depicted in Figure 5.10 as well. The underlying measurements are appended in Table B.5 as supplementary material. The relative differences between model-based and measured results are listed in Table 5.10. The model-based results including scenario transitions show an even higher difference from the measurements of the corresponding implementation, ranging from $-4.59\,\%$ to $-14.65\,\%$. By looking at the mean processor utilizations in Figure 5.10, it can be seen that the measured results are higher than the model-based results, increasing with the number of tiles, e.g., the measured mean processor utilization among

Table 5.10: Relative difference of processor utilization between model-based and measured results with scenario transitions.

Cfg.	$\Delta_r\, U(\mathrm{W1})$	$\Delta_r\, U(\mathrm{W2})$	$\Delta_r\, U(\mathrm{W3})$	$\Delta_r\, U(\mathrm{W4})$	$\Delta_r\, U(\mathrm{W5})$	$\Delta_r\, U(\mathrm{W6})$	$\Delta\, \overline{U}$
C2	−4.959 %	−5.031 %	−5.004 %	−4.147 %	−4.218 %	−4.193 %	−4.592 %
C3	−6.556 %	−6.598 %	−5.370 %	−5.416 %	−9.938 %	−9.969 %	−7.308 %
C6	−16.944 %	−15.054 %	−16.944 %	−11.014 %	−11.016 %	−16.947 %	−14.653 %

all cores in configuration C6 is higher than in configuration C3 or C2. The reason behind this is a discrepancy in behavior between analysis models and corresponding implementations which leads to different scenario occurrence distributions and thus, average processor utilization.

In the analysis models, scenario control tokens are consumed in order. This means, in a configuration with N tiles, the scenario selection for the next sliding window is based on the N-th to last processed window, due to the possibly parallel processing of N windows in a pipelined manner, depending on their actual processing time. That is, windows are cyclo-statically dispatched to different tiles, but due to the processing time that is smaller than the time between subsequent sliding windows in scenario LP, windows are in effect processed sequentially in that scenario. In scenario FP however, the processing time of a window spans multiple sliding window periods, such that N windows are processed in parallel in a configuration with N tiles. Due to consistency between scenario transitions, the selection of scenarios is consistent among different scenarios. Thus, the scenario selection solely depends on the N-th to last processed window in a configuration with N tiles, regardless of the scenario.

However, in the implementation, the most recent scenario selection of the evaluator is considered for the next window. This scheme leads to different scenario selection delays for the two scenarios. In scenario LP, scenario selection for a sliding window is based on the EMA result of the previous window, since in LP the processing of a window is finished before a new window is dispatched. In scenario FP however, the scenario selection of a sliding window is based on the N-th to last processed window in a configuration with N tiles. As a result, scenario selection delay in scenario FP matches between analysis models and implementation while it differs in scenario LP.

The decision on the scenario selection scheme, that reduces scenario selection delay when possible, was made during implementation, in order to optimize the detection latency w.r.t. the beginning of gestures among subsequent windows. However, as the experimental results show, this discrepancy does have an impact on the processor utilization.

While in the model, scenario selection delay is constant among scenarios transitions, in the implementation, scenario selection delays match with the model behavior in scenario transitions from FP to LP, but are much shorter from LP to FP depending on the implemented configuration. Hence, sliding window processing in FP starts in effect earlier in the implementation compared to the model, but stops similarly. As a result, the actually observed scenario occurrence ratio of scenario FP in the implementation is higher than the scenario occurrence probability analyzed from the model. This can be observed by comparing scenario occurrence probabilities analyzed from the SADF models with *a posteriori* observed scenario occurrence distributions in the experiments. The a posteriori observed scenario distribution between FP and LP in the experiments is listed in Table 5.11 together with the scenario occurrence probabilities from the SADF models. It can be seen that with an increasing number

Table 5.11: A posteriori observed scenario occurrences in the experiments and model-based scenario occurrence probabilities.

Scenario	Core	Measured			Model
		C2	C3	C6	All
LP	1	75.42 %	75.00 %	71.88 %	76.98 %
	2	75.42 %	75.00 %	72.50 %	76.98 %
	3	75.42 %	75.31 %	71.88 %	76.98 %
	4	75.63 %	75.31 %	73.75 %	76.98 %
	5	75.63 %	74.06 %	73.75 %	76.98 %
	6	75.63 %	74.06 %	71.88 %	76.98 %
FP	1	24.58 %	25.00 %	28.13 %	23.02 %
	2	24.58 %	25.00 %	27.50 %	23.02 %
	3	24.58 %	24.69 %	28.13 %	23.02 %
	4	24.38 %	24.69 %	26.25 %	23.02 %
	5	24.38 %	25.94 %	26.25 %	23.02 %
	6	24.38 %	25.94 %	28.13 %	23.02 %

of tiles (and thus, with an increase in scenario selection delay) the FP scenario in the implementation has a higher ratio of occurrences than predicted from the SADF models. This difference is caused by the aforementioned latency optimization in the implementation. Other possible influences regarding timing behavior could be excluded by additional investigations which are described in the following.

In order to exclude that conducted experiments might have been too short to observe the expected long run scenario distribution (scenario occurrence probabilities), the mixing time of the SADF Markov chain, i.e., the number of transitions after which the probabilities of being in each of the Markov chain states is close to its equilibrium distribution, has been analyzed. In the thesis at hand, the mixing time is defined as the number of transitions, until the probabilities $Pr(\psi_t)$ of all Markov chain states $\psi \in \Psi_{v_d}$ after t transitions differ less than 0.1 % from their corresponding equilibrium distribution entries, i.e., scenario occurrence probabilities $\pi(v_d, \psi)$. According to [120], the total variation distance δ_{tv}, which represents that difference taking all Markov chain states into account, can be calculated by:

$$\delta_{tv} = \frac{1}{2} \sum_{\psi \in \Psi_{v_d}} |Pr(\psi_t) - \pi(\psi)|. \tag{14}$$

By analyzing the total variation distance w.r.t. the number of successive Markov chain transitions in configuration C1, it could be observed that after 159 transitions (graph iterations), the total variation distance is below 0.1 %. The plotted total variation distance over time (number of graph iterations) t in configuration C1 is provided in Figure B.1 as supplementary material. The actual experiments measuring the average processor utilization were conducted over 960 iterations (sliding windows). As a result, the experiments can be considered as sufficiently long to reach the mixing time and thus observe the corresponding long-run scenario distribution of FP and LP. However, a posteriori observed and model-based estimated scenario distributions differ in the conducted experiments.

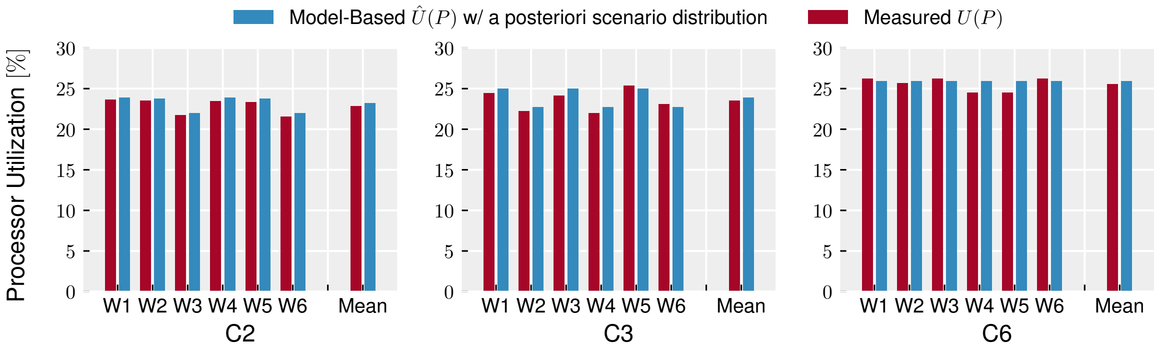


Figure 5.11: Measured and model-based processor utilization with a posteriori observed scenario occurrence probabilities.

These differences lead to the comparatively high discrepancy between model-based estimates and measured average processor utilization. This could be substantiated by a model-based estimation of processor utilization, by taking into account the a posteriori observed distribution of scenario occurrences among the whole experimental run (not per core as listed in Table 5.11) instead of scenario occurrence probabilities acquired from the analysis models. The results are depicted in Figure 5.11. The relative differences between model-based results and measurements are listed in Table B.6 as supplementary information.

Indeed, when knowledge about the actual distribution of scenario occurrences is available, the relative differences between model-based estimates and measurements are in the range of 1.45 % to 1.55 % regarding the mean of all worker cores, which is a reasonable accuracy. These results reflect the model accuracy, when timing behavior between analysis models and implementation matches. However, when looking at configuration C6 in Figure 5.11, a difference between measurements and a posteriori adjusted model-based results can still be observed. The latter are equal among different tiles (cores) in configuration C6, as the workloads in scenario FP are equal among cores, and the scenario occurrences for each individual tile (and thus, worker core in configuration C6) are equally reflected in the scenario occurrence probabilities acquired from the analysis models (cf. Table5.11). However, in the experiments, the actual scenario occurrences for each tile can differ, as sliding windows are dispatched in a cyclo-static manner. That is because the actual time at which a scenario transition occurs is decisive for the scenario occurrence distribution of each individual tile. Therefore, the scenario occurrences observed for individual tiles in the experiments are not equally distributed among each other as predicted by the model. This effect can also be observed from measured processor utilizations (in red) of the last tile in configuration C3 (W5 and W6) compared to the second tile (W3 and W4) in Figure 5.11.

These differences observed between tiles is indeed explained by the lengths of experiments. Due the cyclic window dispatching to tiles (and thus between cyclo-static Markov chain states that correspond to either LP or FP), the ratios between LP and FP for individual tiles are expected to equal each other in the long run. The Markov chain mixing time however is not an indicator for an appropriate lengths in this regard and the decisive factor is the number of actual scenario transitions within the test dataset, which is rather small in the conducted experiments. As a conclusion, these differences are a result of comparing model-based long-run estimates with finite experiments, and are expected to decrease with longer runs.

However, although the model-based long-run average results could not be observed within the conducted experiments as estimated by the analysis models, the work load ratios within a tile, as well as the total or mean of all processor utilizations is contained in the models with an accuracy of up to 98 %, when the behavior of analysis models and implementation and thus, their scenario occurrence distribution match. The discrepancy in token consumption order between analysis models and implementation however, leads to a relative difference of up to 14.65 % between model-based and measured processor utilization. As a result, a mismatch between analyzed models and the corresponding implementation can cause long-run average metrics to be skewed and a consistent implementation w.r.t. analysis models should be prioritized in order to preserve design time estimates. The expected results thereof are represented by the model-based estimates with a posteriori observed scenario occurrences, which show an accuracy of total processor utilization among all cores of up to 98 %.

In general, the long-run average processor utilization depends on the equilibrium distribution of the SADF Markov chain under the real-time constrain that has been introduced in the thesis at hand. However, in order to get reliable estimates, annotations have to be rather accurate, as deviations can have an substantial impact on the resulting scenario distribution and thus the long-run average metric under analysis.

Recognition accuracy Finally, the recognition accuracy has been evaluated with integrated scenario transitions. The introduced scenario transitions do not change the results of the gesture detection for scenario FP in general, as in contrast to the scenario selection, the gesture results are always evaluated for the current sliding window. However, a transition from LP to FP still imposes a delay on the begin of a gesture by a single sliding window (for the implemented mapping in the experiments), as a gesture cannot be evaluated in scenario LP, although the EMA value indicates a start of a gesture. The gesture detection will be activated for the next sliding window. As the end of a gesture is detected for the current sliding window, a gesture that does only appear in a single sliding window will not be detected by the system. This needs to be considered when the sliding window overlap is decreased. However, this behavior is induced by the ARC design, as EMA and DTW for a sliding window are calculated in the same ARC stage, which could be solved when processed in sequence.

A scenario transition from scenario FP to LP imposes a scenario selection delay w.r.t. the age of the sliding window. However, it does not influence the detection of the end of a gesture in general. As due to scenario selection delay, DTW calculation may be performed on subsequent sliding windows although the end has already been detected in a previous window, it does not delay the detection of the corresponding end point of that gesture. Therefore, a single scenario transition from FP to LP does not influence the recognition delay and accuracy. However, in cases of a sliding window progression

Table 5.12: Recognition performance of the experimental hand gesture recognition system with scenario transitions.

Configuration	Precision	Recall	F1-Score
C2	58.45 %	50.92 %	54.43 %
C3	58.45 %	50.92 %	54.43 %
C6	56.46 %	50.92 %	53.55 %

between two gestures, with the number of non gesture sliding windows $N_{SW_{NG}}$ of $1 \leq N_{SW_{NG}} \leq N_k - 1$ in a configuration with N_k tiles, the FP to LP scenario selection delay causes a false negative recognition in the second window after the detection of the beginning of the second gesture. Although this can be prevented by specifying and implementing a minimal time that has to pass between two successive gestures, it has not been implemented for the experiments at hand.

Both situations influence the recognition accuracy compared to the static FP implementation in Chapter 4. However, as both situations occur comparably rare within the test dataset, the recognition accuracy dropped only slightly to a still rather mediocre recognition performance. The precision, recall, and F1-scores for all configurations are summarized in Table 5.12. It can be seen that the F1-score decreased by roughly 1-2 % compared to the static FP implementation (cf. Table 4.13 on page 90).

5.1.7 Discussion

In this sub-chapter, modeling approaches of parallelization strategies for (cyclo-) static ARC setups have been extended to MoCs capturing dynamic changes between static scenarios, i.e., SADF. Proposed analysis techniques w.r.t. processor utilization have been extended from CSDF to SADF in the context of sensor-based online human activity and gesture recognition systems.

While the applied parallelization approaches still consider equally sized segments, an approach has been shown, that can be applied to dynamic changes between static scenarios. In cases of segments with a high variability in size, the range of possible segment sizes can be divided into a number of finite scenarios, each representing the worst case of the sub-range of segment sizes that it covers. This approach has already been shown for an MPEG-4 simple profile decode in [19].

The integration of scenarios allows the modeling of context-aware systems that change their ARC parameters or entire ARC stages at run time. This allows to integrate low power modes or scenario changes into the model and further allows the design-time estimation of worst-case and long-run expectations of extra-functional properties like real-time capability, latency, and processor utilization. While design time estimates are accurate enough to substantiate design decisions early in the design process, the results of this chapter show, that a consistent implementation of the modeled behavior is crucial to preserve the acquired extra-functional properties in the final system.

When considering that scenario transition probability annotations should be acquired by user studies, in order to analyze the system for an expected user behavior (e.g., expected number and lengths of gestures per day), the acquired model-based estimations can be far off at run time of the system, in which the user might behave differently. Moreover, scenario occurrence distributions in the first place, can be easier to specify and analyze in user studies than scenario transition probabilities. As a consequence, instead of parameterizing a Markov chain for design-time analysis, an expected scenario occurrence distribution from user studies or specifications is sufficient for design-time estimation of long-run average metrics. However, this requires the SADF models to satisfy the real-time criterion introduced in the thesis at hand. As a result, an FSM could substitute the Markov chain, which still permits analysis of worst-case throughput and thus real-time behavior, as well as the introduced analysis approaches for extra-functional properties that indicate energy consumption of the system at design time. Furthermore, an FSM-based approach could allow for more efficient timing analysis methods, e.g., as for FSM-SADF graphs [5].

As a further interesting consequence, system specification could provide probability distributions of scenario occurrence distributions, e.g., in form of Dirichlet distributions. Since long-run average metrics of real-time capable SADF graphs are linear combinations of single scenario metrics weighted by their occurrence distribution, Dirichlet distributions of user behavior or other factors that model context, could allow to estimate extra-functional properties at design time that give a tighter expectation and standard deviation or variance of all possible user behaviors or context changes. This approach however, has not been studied in the thesis at hand and may be subject to future work.

In conclusion, the results from this sub-chapter could show, that the SADF MoC is expressive enough for modeling context-aware human activity recognition systems. Furthermore, SADF is analyzable enough for the estimation of extra-functional properties like real-time ability, latency, throughput, and processor utilization, in order to substantiate design decisions on, e.g., mapping, scheduling, parallelization, or platform choices as well as on functional ARC parameters. This chapter extends the state of the art by introducing novel analysis approaches of long-run average processor utilization, which can act as indicators for energy consumption of processing units.

5.2 Data-Dependent Dynamic Behavior

Apart from context-aware or scenario-based dynamic behavior, that is, dynamic changes within a finite number of static or cyclo-static behaviors, there exist algorithms the execution time of which is, e.g, linearly, dependent on some factor that dynamically changes at run time. A prominent example is piecewise linear approximation. Piecewise linear approximation is an approximation technique that, in the context of activity recognition, is applied to sensor signals. It approximates the sensor signals with linear segments (cf. Figure 5.12 on page 123), of which only the beginning and end points of segments need to be stored or transmitted for further processing. However, many state-of-the-art PLA algorithms impose a linear (or worse) execution time w.r.t. the lengths of the linear segments. As a result, design-time analysis with the presented approaches in the thesis at hand only allow for worst-case bounds, which might be very pessimistic on the on hand and constrain the functional properties of such algorithms, e.g., by constraining the maximum lengths of linear segments, and thus data compression, on the other hand. Although MoCs exist to capture higher levels of dynamic behavior (see BDF and DDF [121], or KPNs [122]), no analysis techniques exist in general for such MoCs regarding the extra-functional properties that are crucial to be estimated at design time for gesture and activity recognition systems.

In the thesis at hand, state-of-the-art PLA techniques that produce connected linear segments are evaluated. From their PLA representation, the samples can be reconstructed (with an implied loss of information) by interpolation on the receiver side or segments can be used as is, depending on the application. Furthermore, PLA algorithms take a maximum bound on the segment error as a user-defined parameter that ensures a certain approximation quality of the PLA signal. Most commonly, the Sum of Squared Residuals (SSR) error is used in the literature. Gesture and activity recognition approached like dense motif discovery [123] or continuous string and sequence matching [101, 124] are based on transforming the raw sensor signals into PLA signals. Segments are either converted into symbol sequences for string matching or into numerical representations, such as segment angles, along with sequence matching algorithms.

While providing a concise representation of the signal shape for gesture and activity recognition, another main motivation of using PLA is the reduction of data that either needs to be transmitted from wireless sensors, stored on flash memory, or processed by embedded microprocessors. Reducing the sensor data, reduces energy consumption for transmission, storage, or processing. Hence, PLA is a known approach for energy savings of resource constrained hardware like wearable wireless sensors.

In online PLA algorithms, an update function is called for each new sensor sample, that tries to include that sample into the currently growing segment. Based on the maximum segment error the current segment is extended to include the new sensor sample or a segment point is created until the previous sample and a new segment is started including that new sample. As a result, for each invocation of the update routine, either a new segment point will be created or not. This output behavior could be well modeled with SADF including two scenarios with an output rate of one or zero and scenario transition probabilities representing the data characteristics. However, state-of-the-art SSR-based PLA algorithms have a computational complexity for processing a single sensor sample, that depends on the segment length. As an example, the well-known Sliding Window (SW) method [125] has a linear computational complexity w.r.t. the segment length. The reason for the execution time dependency lies in the recalculation of the SSR error of that segment, with each newly added sensor samples. However, from a modeling and design-time estimation perspective, this data-dependent execution time prevents to derive a WCET without limiting the segment length and thus the data compression ability of SW.

Furthermore, the increased execution time with each new sample within a segment increases processor utilization and thus energy consumption of the processor unit and offsets the anticipated energy savings from the reduced amount of data transmissions.

A possible solution of preventing the data-dependent SSR error calculation is the definition of other residual limits. In [126], a PLA algorithm referred to as *Swing Filter* is introduced, which creates segments based on a least squares approximation of segments. However, the SSR is not explicitly computed and not used as a segment error bound. Instead, an upper and lower bound of the segment angle is updated with each new sample. While this reduces the execution time to a constant time complexity of processing a single sensor sample, the error bound of the resulting segments is conceptually different to that of SSR-based PLA algorithms.

As an alternative solution, two new PLA algorithms are introduced in the thesis at hand, referred to as *fastSW* and *Connected Piecewise Linear Regression (CPLR)* which are based on SW and Swing Filter. Both algorithms offer a comparable approximation quality to state-of-the-art PLA algorithms, while maintaining a constant computational complexity with a small deterministic worst-case execution time and a bound on the SSR error for each segment. The alternative SSR calculation introduces a, in practice neglectable, decrease of numerical precision but a constant time complexity, allowing for WCET estimations at design time. This makes it possible to model the PLA calculation with SADF, similarly to the discussed modeling approach in section 5.1. Furthermore, the new PLA algorithms come with a constant memory requirement which is small enough to be implemented on embedded processors with harsh memory constraints. Lastly, in contrast to all state-of-the-art SSR-based PLA algorithms, the absence of a buffer for sensor data, not only reduces memory complexity, but also removes the constraint and the parameter of a maximum segment length and allows for maximal data reduction.

5.2.1 Related Work

In the literature, PLA algorithms are sometimes also referred to as segmentation algorithms. Keogh et al. introduced in [125] the prominent PLA algorithm *SWAB* (Sliding Window and Bottom Up), which is a combination of two formerly known segmentation algorithms *Bottom-Up* and SW, the origins of which are unapparent from the literature. The combination of the offline Bottom-Up algorithm and the online SW algorithm is motivated by Keogh et al. by the superior approximation quality of Bottom-Up and the online character of Sliding Window. However, SWAB being an online PLA algorithm, SW is such as well, and the approximation quality of it could not be verified as inferior in the experiments of this thesis. Furthermore, the computational complexity of SW is $O(n)$ for processing a single sensor sample w.r.t. the segment length, in contrast to the $O(n^2)$ computational complexity of SWAB, which makes SW an equally entitled candidate for online PLA on its own.

Van Laerhoven et al. introduced a conceptual optimization of SWAB in [101] called mSWAB, which was further optimized for resource constrained wearable sensor nodes by Berlin et al. in [127], called *emSWAB*. The computational complexity of mSWAB and emSWAB processing a single sensor sample is still $O(n^2)$ w.r.t. the segment or buffer length, but the processing time is reduced by a constant factor. Furthermore, the error bound of emSWAB has been adapted from an SSR error calculation to a sum of absolute distances in [127] for a faster error calculation. Although this optimization is neglectable compared to the $O(n^2)$ time complexity w.r.t. the buffer size, emSWAB is the most optimized version and is therefore, included in the evaluations of this sub-chapter.

Due to the increase in the execution time of the aforementioned PLA algorithms with the size of segments, the processing effort of a single sensor sample is dependent on the maximum achievable data reduction. Thus, the more transmission energy is saved due to data reduction, the more processing power is consumed by the algorithms, resulting in a trade-off between transmission and processor energy. The resulting total energy consumption depends on the final hardware architecture. However, due to the increasing energy consumption of the processor, the potential energy savings due to decreased data transmission cannot be fully utilized by the aforementioned PLA algorithms.

Furthermore, all aforementioned PLA algorithms need to store the raw sensor samples of a segment in a buffer, until that segment is finalized and outputted. This introduces another constraint to the data reduction ability when implemented on architectures with harsh memory constraints, e.g. sensor sub-systems. Although memory consumption and execution time can be reduced by constraining the maximum segment length and thus the buffer, it limits the ability to reduce and represent sensor data efficiently in terms of data compression.

The PLA algorithms introduced in the thesis at hand eliminate the aforementioned limitations, providing a small and (quasi-) constant execution time per sensor sample and a small and constant memory consumption. Furthermore, the approximation quality in terms of total SSR error of the entire sensor signal is comparable with state-of-the-art PLA algorithms. Lastly, despite a possible limit due to numerical precision, the PLA algorithms introduced in the thesis at hand allow for theoretically unconstrained segment lengths due to the absence of a buffer and the computational complexity of $O(1)$ per sample, w.r.t. the segment lengths.

In the fields of data mining, gesture, and activity recognition, the aforementioned PLA algorithms are commonly chosen for the sake of data reduction and signal representation. However, other PLA algorithms can be found in the literature as well. Liu et al. introduced the PLAMLiS PLA algorithm in [128] to reduce collected data in WSN. Pham et al. [129] introduced an optimization of that PLA algorithm. Again, both algorithms are buffer based. While no computational complexity of processing a single sensor value is given, the computational complexity of approximating a signal of n samples is given with $O(n^2)$ for the original PLAMLiS algorithm [128] and for its improved version [129]. As a result, the processing time of a single sensor sample is not constant and increases with the buffer size, and thus with the possible compression factor.

Fuchs et al. introduced a segmentation algorithm based on Polynomial Least-Squares Approximation with polynomials of arbitrary order in [130], referred to as SwiftSeg. Since their approximation algorithm also includes first order polynomials, i.e., linear segments, its first order variant resembles a PLA algorithm. Furthermore, their algorithm has a constant computational complexity and depending on the windowing method a linear (sliding window) or constant (growing window) memory complexity, w.r.t. the segment length. When the sum of squared residual error is used as a segmentation condition, their growing window method is comparable in computational and memory complexity to CPLR. This is due to the fact that similar updating techniques are used for their polynomial segment representation. However, the main difference of their first order variant to CPLR is, that their approximating polynomial includes the intercept term. As a result, their approximation leads to optimal in the least square sense, but not connected segments. In contrast, CPLR is based on linear regression without an intercept term, in order to produce connected segments that are optimal in a least squares sense. Hence it is named Connected Piecewise Linear Regression.

Elmeleegy et al. introduced a PLA algorithm in [126] named Swing Filter (SF). Their algorithm is closest related to CPLR as it produces connected linear segments based on linear regression and creates segments with slopes which, in the best case, are optimal in the least squares sense. However, their algorithm is not bounding the sum of squares error of a segment, but rather bounding the trajectory of the created segment by constantly constraining the slope of the segment with updating lower and upper bounds depending on each new sample. While this segment constraint allows a constant computational and memory complexity for processing each sample, it is conceptually different to an SSR bound of segments. The fast SSR error calculation of the PLA algorithms introduced by the thesis at hand offer a constant computational and memory complexity w.r.t. the segment lengths and bound the segments to a user-defined maximum SSR error. Furthermore, with CPLR created segments are guaranteed to be optimal in a least squares sense, in the context of connected linear segments.

Luo et al. introduced a PLA algorithm with constant update time in [131]. However, their PLA algorithm is buffer-based with a worst-case space complexity of $O(n)$ and uses around 1 KB memory in their experiments which does not allow it to be executed on embedded microprocessors with harsh resource constraints. Furthermore, their approach is a mixture between connected and non-connected piecewise segments.

To the author's best knowledge, CPLR and fastSW are the only SSR-based online PLA algorithms that produce connected linear segments with a computational and memory complexity of $O(1)$ w.r.t. segment lengths, at the time of writing the thesis at hand. Both algorithms will be explained in more detail, after a short recap of basics regarding simple linear regression in the following section.

5.2.2 Simple Linear Regression

This section summarizes the mathematical representation of simple linear regression as a foundation of the derived SSR calculation for the introduced PLA algorithms. In simple linear regression, the linear relation of a dependent variable is described as a function of an independent variable in terms of the minimal SSR error. The linear model, referred to as regression line can be described by the function $y = \alpha + \beta x$, with x and y being the independent and dependent variables, respectively, α the intercept with the y coordinate, and β the slope of the regression line.

As the introduced PLA algorithms should lead to an approximation with connected linear segments, the regression line should pass the origin, i.e., $y = 0$ at $x = 0$. The resulting regression line without an y offset has the form:

$$y = \beta x. \tag{15}$$

In case of sensor signal approximation, a linear model will not perfectly describe sensor samples in general. Instead, sensor samples will deviate from the regression line, which introduces an error. The errors can be described as residual errors e_i and included into the linear model that describes n sensor samples. The resulting linear model including residual errors can be described by:

$$y_i = \beta_n x_i + e_i, \tag{16}$$

with $1 \leq i \leq n$ and β_n describing the slope that models the n sensor samples. For a given estimator $\widehat{\beta}_n$ of the slope that fits the n sensor samples best, the SSR error SSR_n of that regression line is calculated by:

$$SSR_n = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - \widehat{\beta}_n x_i)^2. \tag{17}$$

The Ordinary Least Squares (OLS) estimator $\widehat{\beta}_n$ that represents the data with the minimal SSR error is calculated by:

$$\widehat{\beta}_n = \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2} = \frac{\overline{xy}_n}{\overline{x^2_n}}, \tag{18}$$

with $\overline{xy}_n = \frac{\sum_{i=1}^{n} x_i y_i}{n}$ and $\overline{x^2}_n = \frac{\sum_{i=1}^{n} x_i^2}{n}$ denoting the mean values of all $x_i y_i$ products and x_i^2 squares up to the n-th value, respectively. This can be derived by either finding $\widehat{\beta}_n$ that minimizes SSR_n by derivation of Equation (17), or from the well-known OLS estimator of multiple linear regression with multiple independent variables:

$$\widehat{\beta}_n = (X^T X)^{-1} X^T Y, \tag{19}$$

where $Y \in \mathbb{R}^{nx1}$ is the vector of responses y_i for all $i = 1, \dots, n$ observations and $(X^T X)^{-1} X^T$ is the Moore-Penrose pseudoinverse of X. Here, $X \in \mathbb{R}^{nxm}$ is the design matrix with each row being the observation vector $x_i \in \mathbb{R}^{1xm}$ for each observation $i = 1, \dots, n$ and with columns $j = 1, \dots, m$ for each independent variable (regressor). In that case, $\widehat{\beta}_n \in \mathbb{R}^{mx1}$ is the vector of parameters for each of the m regressors. The constant term (intercept) is implicitly included by the first column in X filled with ones, i.e., $x_{i1} = 1$. Thus, the first parameter of $\widehat{\beta}_n$ gives the intercept term, which is therefore contained in the number of regressors m.

Reducing to only one regressor and eliminating the intercept, i.e., $m = 1$, the design matrix becomes $X \in \mathbb{R}^{nx1}$. Furthermore, $\widehat{\beta}_n$ becomes a scalar and the matrix multiplications with X^T on the left sides in Equation (19) resolve to scalar products of vectors X and Y, resembling Equation (18):

$$\widehat{\beta}_n = \left(\sum_{i=1}^{n} x_i x_i \right)^{-1} \sum_{i=1}^{n} x_i y_i. \tag{20}$$

Updating the means $\overline{xy}_n$ and $\overline{x^2}_n$ along with each new value can be done in $O(1)$ time and allows to recalculate $\widehat{\beta}_n$ for each new sensor sample as well in constant time. In order to calculate the SSR_n from the updated variables and the OLS estimator $\widehat{\beta}_n$ for each new data point, Equation (17) needs to be resolved with the binomial formula to:

$$SSR_n = \sum_{i=1}^{n} y_i^2 - 2\widehat{\beta}_n \sum_{i=1}^{n} x_i y_i + \widehat{\beta}_n^2 \sum_{i=1}^{n} x_i^2. \tag{21}$$

Each sum can then be rewritten using the corresponding mean values multiplied by n:

$$SSR_n = \overline{y^2}_n n - 2\widehat{\beta}_n \overline{xy}_n n + \widehat{\beta}_n^2 \overline{x^2}_n n. \tag{22}$$

Resolving $\widehat{\beta}_n^2 \overline{x^2}_n$ to $\widehat{\beta}_n \overline{xy}_n$ from Equation (18) gives:

$$SSR_n = n(\overline{y^2}_n - \widehat{\beta}_n \overline{xy}_n). \tag{23}$$

Note that the last step is only possible, if $\widehat{\beta}_n$ was actually calculated by Equation (18). This will be the case for CPLR. However, for fastSW, the slope will be calculated differently, preventing the last simplification of Equation (22) to Equation (23).

Incremental updating In order to recalculate the slope $\widehat{\beta}_n$ and the error SSR_n with each new sensor sample, the means $\overline{x^2}_n$, $\overline{xy}_n$, and $\overline{y^2}_n$ have to be updated incrementally with each new sensor sample as well. In general, a mean $\overline{z}_n$ of samples z_i can be calculated by:

$$\overline{z}_n = \frac{\sum_{i=1}^{n} z_i}{n} = \frac{\sum_{i=1}^{n-1} z_i + z_n}{n}. \tag{24}$$

Substituting $\sum_{i=1}^{n-1} z_i$ with $\overline{z}_{n-1} \cdot (n-1)$ gives the equation updating the mean $\overline{z}_n$ with the new sample z_n from [66]:

$$\overline{z}_n = \frac{\overline{z}_{n-1}(n-1)}{n} + \frac{z_n}{n}. \tag{25}$$

For sake of less multiplication and division instructions on a processor, Equation (25) can be restructured to:

$$\overline{z}_n = \overline{z}_{n-1} + \frac{z_n - \overline{z}_{n-1}}{n}, \tag{26}$$

to reduce the processing time. Equation (26) can be used for updating all means $\overline{x^2}_n$, $\overline{xy}_n$, and $\overline{y^2}_n$ with each new sensor sample to recalculate $\widehat{\beta}_n$ and SSR_n with Equations (18) and (23), respectively, in constant time.

5.2.3 Online Piecewise Linear Approximation

In the following, the problem of online piecewise linear approximation of sensor signals will be defined. A sensor signal can be sampled at different timestamps, which not necessarily have to be equidistant in time. Sampling a sensor signal, leads to a series S of sensor samples $s[m] = (v[m], \tau_v[m]) \in S^{\mathbb{N}}$, with $S = \mathbb{R} \times \mathbb{T}$ and $m \in [1, 2, \ldots, M]$, M possibly going to infinity. Each sample (v, τ_v) consists of the signal value $v \in \mathbb{R}$ and the corresponding timestamp $\tau_v \in \mathbb{T}$. Note that the above definition considers one-dimensional signals. The relation to signals with an arbitrary number of dimensions and the application of developed algorithms to these is explained at the end of this section. Furthermore, without loss of generality, timestamps are assumed to be discrete, i.e., $\mathbb{T} = \mathbb{Z}_{0+}$, in the thesis at hand.

A piecewise linear approximation of a sensor signal is a series $\widetilde{S}$ of segment points $\widetilde{s}[k] = (\widetilde{v}[k], \tau_{\widetilde{v}}[k]) \in \widetilde{S}^{\mathbb{N}}$, with $\widetilde{S} = \mathbb{R} \times \mathbb{T}$, and $k \in [1, 2, \ldots, K]$, K possibly going to infinity as well. Each segment point $(\widetilde{v}, \tau_{\widetilde{v}})$ consists of a signal value $\widetilde{v} \in \mathbb{R}$ and its corresponding timestamp $\tau_{\widetilde{v}} \in \mathbb{T}$ and represents the end point of the previous and the starting point of the next segment. A segment is thus represented by a pair of consecutive segment points $((\widetilde{v}[k-1], \tau_{\widetilde{v}}[k-1]), (\widetilde{v}[k], \tau_{\widetilde{v}}[k]))$. Furthermore, each segment point has a timestamp that equals a timestamp of one of the original sensor signal samples, i.e., $\forall (\widetilde{v}, \tau_{\widetilde{v}}) \in \widetilde{S}\ \exists (v, \tau_v) \in S : \tau_v = \tau_{\widetilde{v}}$ holds.

An example of a piecewise linear approximated Electrocardiography (ECG) signal can be seen in Figure 5.12.

For some PLA algorithms, each segment point is a sensor sample including both signal value and timestamp from the original sensor signal, i.e., $\forall (\widetilde{v}, \tau_{\widetilde{v}}) \in \widetilde{S} : (\widetilde{v}, \tau_{\widetilde{v}}) \in S$ holds. Algorithms of which the last statement holds include SWAB, mSWAB, emSWAB, and SW. However, other PLA algorithms, e.g., the Swing Filter, do not represent that special case. The thesis at hand introduces for each case a novel PLA algorithm with constant computational and memory complexity, that both produce segments that are bounded by a user-defined threshold on the maximum segment SSR error.

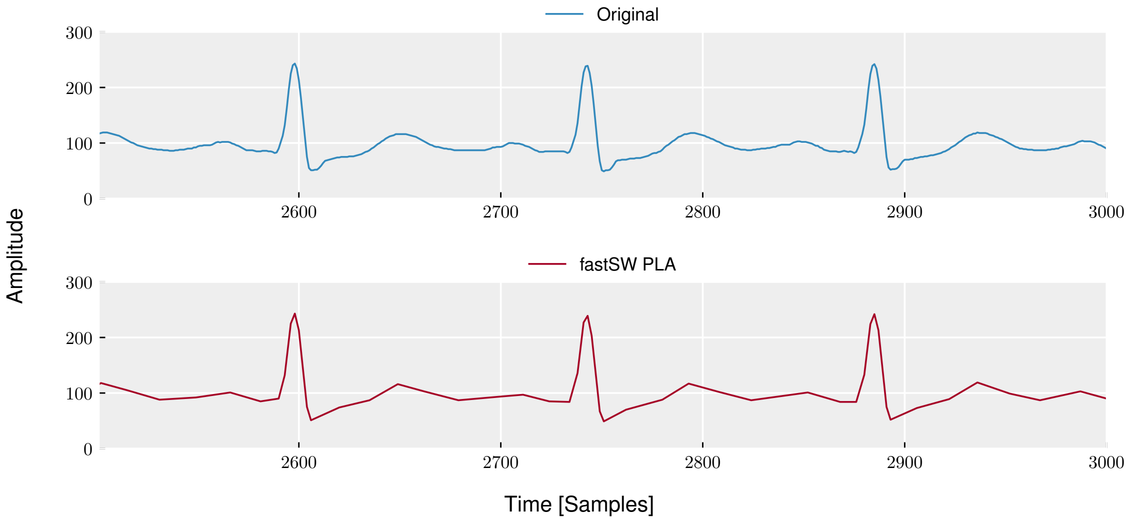


Figure 5.12: Excerpt of the ECG signal from [132] approximated with fastSW.

In order to achieve a piecewise linear approximation $\widetilde{S}$ of a sensor signal S online, the sensor signal needs to be continuously represented, i.e., the segment corresponding to the last sensor sample needs to be outputted as soon as its end point is determined due to an exceeded threshold. Therefore, online PLA algorithms perform a sample-based processing, which means that the algorithm is invoked with each new sensor sample. The processing time for each invocation of the algorithm is referred to as its execution time.

In the process of online approximation, the average execution time needs to be smaller than the sampling period, in order to allow a real-time performance. To avoid additional latencies and additional memory consumption due to input buffering, even the worst-case execution time should be smaller than the sampling period. Moreover, a data-independent WCET is preferable, as this guarantees a predictable maximum sampling period at which the algorithm is able to approximate the signal in real time without compromising the compression abilities.

The PLA algorithms fastSW and CPLR introduced in the thesis at hand, are based on two state-of-the-art PLA algorithms from the literature, namely SW and Swing Filter, respectively.

FastSW approximation The fastSW algorithm follows a mathematically equal, but algorithmically different SSR error calculation as SW. The SW algorithm adds a new sample to the current segment in its buffer, each time the algorithm is invoked. It creates a segment from the previous segment end point to the newly added sample and calculates the residual error of this segment by linear interpolation. This error calculation involves n steps for a segment length of n and has to be recalculated each time a new sample is added to the buffer. If the calculated segment error is below the SSR error bound, which is a user-defined threshold TH, the routine returns without outputting that segment and will be invoked with the next incoming sensor sample. If the segment error is above TH, the segment from the previous invocation is recreated, which still satisfied the maximum error guarantee and is outputted. A new segment is started from that point to the new sensor sample. Due to the error calculation, the execution time of the SW algorithm depends on the current segment length and increases with each invocation in which the segment error has not reached TH. Furthermore, the memory consumption for buffering the last n sensor samples increases likewise. Thus, with longer segments, both the execution time and the memory consumption of each invocation increases. In order to guarantee a maximum execution time and memory consumption, the maximum buffer size needs to be constrained to a possibly small size, which in turn limits the compression ability of the algorithm.

As a solution, a new PLA algorithm referred to as fastSW is introduced in the thesis at hand. The principle of sensor signal approximation is equal to that of SW, except the calculation of the segment SSR error. Instead of buffering all sensor samples of a corresponding segment and recalculating the SSR error iterating over all these samples in each invocation, the SSR is recalculated by applying Equation (22). The running variables $\overline{x^2}_n$, $\overline{xy}_n$, and $\overline{y^2}_n$ are updated with Equation (26), correspondingly, with x_n representing the difference of the current sample timestamp $\tau_v[m]$ and the segment starting point timestamp $\tau_{\widetilde{v}}[k-1]$, k being the index of current segment end point, and m being the index of the newest sensor sample. Likewise y_n represents the difference of the newest sample value $v[m]$ and the value of the segment start point $\widetilde{v}[k-1]$. Thus, x_n and y_n represent the coordinates of the new sensor sample value $v[m]$ and timestamp $\tau_v[m]$, respectively, projected onto the coordinate system, the origin of which is located at the last segment end point or the current segment start point $(\widetilde{v}[k-1], \tau_{\widetilde{v}}[k-1])$, respectively.

In contrast to linear regression, the segments of SW and also fastSW are connecting two original sensor samples, i.e., the last segmentation point and possibly the newest sensor sample, depending on the corresponding SSR. To this end, the slope β_n is calculated as the fraction $\beta_n = y_n/x_n$. This additionally implies, that the new sensor sample coordinates (x_n, y_n) are directly located on the segment, with a residual error $e_n = 0.0$. As a result, the SSR_n of the segment in the current invocation includes all samples represented by that segment so far, except the n-th sample. Thus, SSR_n is calculated by using the new segment slope $\beta_n = y_n/x_n$ and the running variables from the last invocation with:

$$SSR_n = (\overline{y^2}_{n-1} - 2\beta_n \overline{xy}_{n-1} + \beta_n^2 \overline{x^2}_{n-1})(n-1). \tag{27}$$

In case SSR_n is below TH, the running variables $\overline{x^2}_n$, $\overline{xy}_n$, and $\overline{y^2}_n$ will be updated with y_n and x_n for the next invocation.

The pseudo code of the function processing a new sensor sample with fastSW is shown in Algorithm 1. The threshold value TH is set upon initialization and needs to be stored globally. When starting the online approximation of a sensor signal, the very first sample $(v[1], \tau_v[1])$ will be used as the initial segment point $(\widetilde{v}[1], \tau_{\widetilde{v}}[1]) = (v[1], \tau_v[1])$. The routine *PROCESS_SAMPLE* is called for each new sensor sample $s = (v, \tau_v)$, which is the first parameter given to the function along with the array $\widetilde{S}$ for storing segments and the index k for specifying at which position in $\widetilde{S}$ the new segment will be stored. Note that the array of segments $\widetilde{S}$ does not necessarily need to store all created segments, e.g., when immediately transmitting each new segment point to another device. However, $\widetilde{S}$ needs a size of at least two: for the previous segment end point and the current segment endpoint, which will be written to it when created. The variables n, $\overline{x^2}_{n-1}$, $\overline{xy}_{n-1}$, $\overline{y^2}_{n-1}$, x_{n-1}, and y_{n-1} need to be stored globally and must be initialized with zero before starting the online approximation of a sensor signal. All other variables can be implemented as local variables.

At the beginning, the size n of the current segment is incremented by the new sample in line 2. In line 3 and 4, the coordinates y_n and x_n of $s = (v, \tau_v)$ within the coordinate system of the last segment end point $(\widetilde{v}[k-1], \tau_{\widetilde{v}}[k-1])$ are calculated, respectively. The function *value()* returns the value part and *timestamp()* returns the timestamp part of the segment point $(\widetilde{v}[k-1], \tau_{\widetilde{v}}[k-1])$ and the sensor

Algorithm 1 FastSW.

```
1:  procedure PROCESS_SAMPLE(sample $s$, segment array $\widetilde{S}[]$, index $k$)
2:      $n = n + 1$
3:      $y_n = value(s) - value(\widetilde{S}[k-1])$
4:      $x_n = timestamp(s) - timestamp(\widetilde{S}[k-1])$
5:      $\beta_n = y_n / x_n$
6:      $SSR_n = (\overline{y^2}_{n-1} - 2\beta_n \overline{xy}_{n-1} + \beta_n^2 \overline{x^2}_{n-1}) \cdot (n-1)$
7:      if $SSR_n <= TH$ then
8:          $\overline{x^2}_{n-1} = \overline{x^2}_{n-1} + ((x_n \cdot x_n) - \overline{x^2}_{n-1})/n$
9:          $\overline{xy}_{n-1} = \overline{xy}_{n-1} + ((x_n \cdot y_n) - \overline{xy}_{n-1})/n$
10:         $\overline{y^2}_{n-1} = \overline{y^2}_{n-1} + ((y_n \cdot y_n) - \overline{y^2}_{n-1})/n$
11:         $x_{n-1} = x_n$
12:         $y_{n-1} = y_n$
13:         return 0
14:     $\tau_{\tilde{v}} = timestamp(\widetilde{S}[k-1]) + x_{n-1}$
15:     $\tilde{v} = value(\widetilde{S}[k-1]) + y_{n-1}$
16:     $n = 1$
17:     $y_{n-1} = value(s) - \tilde{v}$
18:     $x_{n-1} = timestamp(s) - \tau_{\tilde{v}}$
19:     $\overline{x^2}_{n-1} = x_{n-1} \cdot x_{n-1}$
20:     $\overline{xy}_{n-1} = x_{n-1} \cdot y_{n-1}$
21:     $\overline{y^2}_{n-1} = y_{n-1} \cdot y_{n-1}$
22:     $\widetilde{S}[k] = (\tilde{v}, \tau_{\tilde{v}})$
23:     return 1
```

sample s, respectively. From lines 5 and 6, the new segment slope β_n and the new segment error SSR_n are calculated. If the new SSR_n is below TH (line 7), the means $\overline{x^2}_n$, $\overline{xy}_n$, and $\overline{y^2}_n$ are updated with the coordinates of the new sample x_n and y_n (line 8 to 10), but are stored in the global variables $\overline{x^2}_{n-1}$, $\overline{xy}_{n-1}$, and $\overline{y^2}_{n-1}$ for the next invocation. Afterwards, x_n and y_n will be stored for the next invocation as x_{n-1} and y_{n-1} in lines 11 and 12, respectively. The routine returns without creating a new segment point, indicated by a return value of 0. If SSR_n exceeds TH instead (line 7), a new segment point $(\tilde{v}, \tau_{\tilde{v}})$ is created at the previous sample timestamp by adding its coordinates x_{n-1} and y_{n-1} to the last segment point timestamp and value in lines 14 and 15. A new segment is started from there, whose size is set to 1 in line 16. The coordinates of s in the new coordinate system of $(\tilde{v}, \tau_{\tilde{v}})$ are calculated in line 17 and 18 and the running variables $\overline{x^2}_n$, $\overline{xy}_n$, and $\overline{y^2}_n$ are initialized with them in lines 19 to 21. The newly created segment point is finally stored in the segment array at position k in line 22 and the function returns 1 to indicate the creation of a new segment point. The segment points created with fastSW in relation to the SSR are illustrated in Figure 5.13.

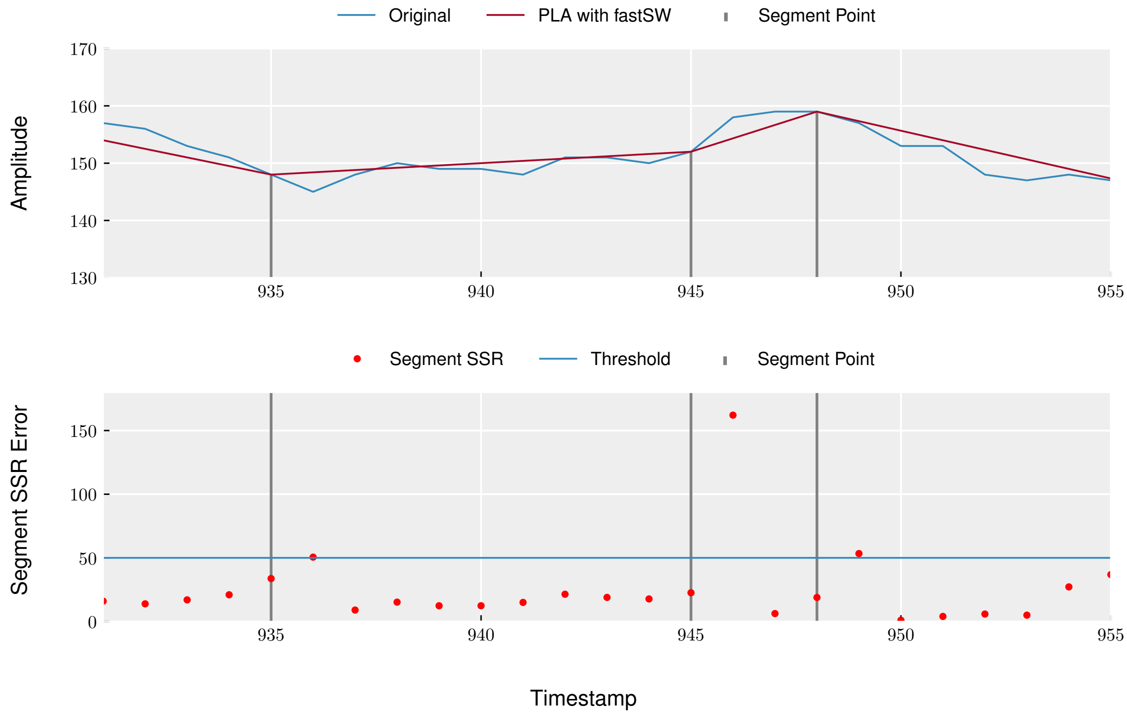


Figure 5.13: Piecewise linear approximation with fastSW including sum of squared residuals segment error per invocation.

CPLR approximation The CPLR algorithm introduced in the thesis at hand, is based on the Swing Filter [126]. The Swing Filter does not implement a maximum segment SSR error constraint, but rather a threshold $TH_{e,max}$ on the absolute sample deviation e_i from the segment. The maximum deviation is ensured, by keeping and updating a minimum slope β_L and a maximum slope β_H with each sample. To this end, the coordinate system for the approximation is set to the last segmentation point $\widetilde{S}[k-1]$, as with fastSW. Each new sample $s=(v[m],\tau_v[m])$ will then be projected into that coordination system to their corresponding coordinates x_n and y_n. Initially, β_H and β_L will be set to intersect the coordinate $TH_{e,max}$ units above and below the first sensor sample, respectively, with:

$$\beta_H = \frac{(y_n + TH_{e,max})}{x_n} \tag{28}$$

and

$$\beta_L = \frac{(y_n - TH_{e,max})}{x_n}. \tag{29}$$

In subsequent invocations, each new sample will be checked to be above β_L and below β_H. In case the newest sensor sample falls outside of these slopes, a segmentation point at the last samples timestamp will be created. The segmentation point however, will not necessarily lie on the original samples coordinate, but will be extrapolated. The slope for extrapolation is acquired by the following steps. For each sensor sample, two running variables $\overline{x^2}_n$ and $\overline{xy}_n$ will be updated and from them, the best, in a least-squares sense, fitting slope $\widehat{\beta}_n = \overline{xy}_n / \overline{x^2}_n$ is calculated. However, the slope for extrapolation is the result of $min(max(\widehat{\beta}_n, \beta_L), \beta_H)$. Thereby, the slope is forced to be within the range of $[\beta_L, \beta_H]$.

If the new sensor sample s falls above β_L and below β_H, the slopes will be updated to intersect the coordinate $TH_{e,max}$ units below and above s, respectively, in case s is more than $TH_{e,max}$ units below and above β_H and β_L, respectively. As a result, the slopes approach each other with each new sensor sample, until eventually a new sensor sample falls outside the area between them. As each update step of running variables $\overline{x^2}_n$ and $\overline{xy}_n$, as well as β_L and β_H can be calculated in constant time, the computational complexity of the Swing Filter is $O(1)$. Furthermore, its memory complexity is $O(1)$ as no sensor data has to be buffered. Additional to $TH_{e,max}$, a maximum segment length can be specified for the Swing Filter, to create a new segment point after a maximum number of samples.

However, the Swing Filter does not allow to specify a maximum segment error in terms of SSR. Furthermore, the segments are not guaranteed to represent the best fit in a least-squares sense, as the slope is constrained by β_L and β_H.

As a complement to SSR-based PLA algorithms, CPLR is introduced in the thesis at hand, which shares the linear regression nature of the Swing Filter by recalculating $\widehat{\beta}_n$ from running variables $\overline{x^2}_n$ and $\overline{xy}_n$. However, instead of keeping and updating a lower and upper bound on the slope, the running variable $\overline{y^2}_n$ is updated additionally, which allows a calculation of SSR_n for the current segment in constant time, with each new sensor sample. In contrast to fastSW, the segment does not necessarily intersect the newest sensor sample (x_n, y_n) and thus SSR_n is calculated from the running variables $\overline{x^2}_n$ and $\overline{xy}_n$, and $\overline{y^2}_n$, up to the n-th sensor sample. Furthermore, $\widehat{\beta}_n$ is calculated by $\widehat{\beta}_n = \overline{xy}/\overline{x^2_n}$, which allows the application of Equation (23) to calculate SSR_n. Similar as the Swing Filter, a new segmentation point needs to be extrapolated from the regression line at the previous sample timestamp by $\widetilde{v}[k] = \widetilde{v}[k-1] + (\widehat{\beta}_n \cdot x_{n-1})$.

The pseudo code of the function processing a new sensor sample with CPLR is shown in Algorithm 2. The parameters given to CPLR equal those of fastSW and the first signal sample $(v[1], \tau_v[1])$ will be used as the initial segment point $(\widetilde{v}[1], \tau_{\widetilde{v}}[1]) = (v[1], \tau_v[1])$. The variables n, $\overline{x^2}_{n-1}$, $\overline{xy}_{n-1}$, $\overline{y^2}_{n-1}$, x_{n-1}, and $\widehat{\beta}_{n-1}$ need to be stored globally and must be initialized with zero before starting the online approximation of a sensor signal. All other variables can be implemented as local variables. In contrast to fastSW, CPLR extrapolates $\widetilde{v}[k]$ from $\widehat{\beta}_{n-1}$ and thus needs to store $\widehat{\beta}_{n-1}$ instead of y_{n-1} globally. The index of the end point of the currently developing segment is k.

At the beginning, the size n of the current segment is incremented by the new sample in line 2. In line 3 and 4, the coordinates y_n and x_n of $s = (v, \tau_v)$ within the coordinate system originating in the last segment point $(\widetilde{v}[k-1], \tau_{\widetilde{v}}[k-1])$ are calculated, respectively. The running variables $\overline{x^2}_n$, $\overline{xy}_n$, and $\overline{y^2}_n$ need to be updated with (x_n, y_n) before the calculation of SSR_n, which is performed in lines 5 to 7. The slope $\widehat{\beta}_n$ and the segment error SSR_n are then recalculated in lines 8 and 9. If the new SSR_n is below TH (line 10), the running variables $\overline{x^2}_n$, $\overline{xy}_n$, and $\overline{y^2}_n$, the slope $\widehat{\beta}_n$, and the x_n coordinate of τ_v are stored for the next invocation as $\overline{x^2}_{n-1}$, $\overline{xy}_{n-1}$, and $\overline{y^2}_{n-1}$, $\widehat{\beta}_{n-1}$, and x_{n-1} in lines 11 to 15, respectively. In line 16, the routine returns without creating a new segment point, indicated by a return value of 0.

If SSR_n exceeds TH instead (line 10), a new segment point $(\widetilde{v}, \tau_{\widetilde{v}})$ is created at the timestamp of the previous sample by adding its coordinate x_{n-1} to the last segment point timestamp $\tau_{\widetilde{v}}[k-1]$ in line 17. In line 18, the segment point value $\widetilde{v}$ at the last sample timestamp is extrapolated from $\widehat{\beta}_{n-1}$.

A new segment is started from this point, whose size is set to 1 in line 19. The coordinates of s in the new coordinate system of $(\widetilde{v}, \tau_{\widetilde{v}})$ are calculated in lines 20 and 21 and the running variables $\overline{x^2}_{n-1}$, $\overline{xy}_{n-1}$, and $\overline{y^2}_{n-1}$ are initialized with these in lines 22 to 24. The corresponding slope $\widehat{\beta}_{n-1}$ is

Algorithm 2 Connected Piecewise Linear Regression.

```
1:  procedure PROCESS_SAMPLE(sample value $s$, segment array $\widetilde{S}[]$, index $k$)
2:      $n = n + 1$
3:      $y_n = value(s) - value(\widetilde{S}[k-1])$
4:      $x_n = timestamp(s) - timestamp(\widetilde{S}[k-1])$
5:      $\overline{x^2}_n = \overline{x^2}_{n-1} + ((x_n \cdot x_n) - \overline{x^2}_{n-1})/n$
6:      $\overline{xy}_n = \overline{xy}_{n-1} + ((x_n \cdot y_n) - \overline{xy}_{n-1})/n$
7:      $\overline{y^2}_n = \overline{y^2}_{n-1} + ((y_n \cdot y_n) - \overline{y^2}_{n-1})/n$
8:      $\widehat{\beta}_n = \overline{xy}_n / \overline{x^2}_n$
9:      $SSR_n = (\overline{y^2}_n - \widehat{\beta}_n \overline{xy}_n) \cdot n$
10:     if $SSR_n <= TH$ then
11:         $\overline{x^2}_{n-1} = \overline{x^2}_n$
12:         $\overline{xy}_{n-1} = \overline{xy}_n$
13:         $\overline{y^2}_{n-1} = \overline{y^2}_n$
14:         $\widehat{\beta}_{n-1} = \widehat{\beta}_n$
15:         $x_{n-1} = x_n$
16:         return 0
17:     $\tau_{\widetilde{v}} = timestamp(\widetilde{S}[k-1]) + x_{n-1}$
18:     $\widetilde{v} = value(\widetilde{S}[k-1]) + (\widehat{\beta}_{n-1} \cdot x_{n-1})$
19:     $n = 1$
20:     $y_{n-1} = value(s) - \widetilde{v}$
21:     $x_{n-1} = timestamp(s) - \tau_{\widetilde{v}}$
22:     $\overline{x^2}_{n-1} = x_{n-1} \cdot x_{n-1}$
23:     $\overline{xy}_{n-1} = x_{n-1} \cdot y_{n-1}$
24:     $\overline{y^2}_{n-1} = y_{n-1} \cdot y_{n-1}$
25:     $\widehat{\beta}_{n-1} = \overline{xy}_{n-1} / \overline{x^2}_{n-1}$
26:     $\widetilde{S}[k] = (\widetilde{v}, \tau_{\widetilde{v}})$
27:     return 1
```

recalculated for the next invocation in line 25 and the newly created segment point is stored in the segment array at position k in line 26. Finally, in line 27 the function returns 1 to indicate the creation of a new segment point. The segment points created with CPLR in relation to the SSR are illustrated in Figure 5.14.

Multi-dimensional signals Sensor-based activity and gesture recognition is mainly performed on multi-dimensional inertial sensor signals, e.g., sampled from 3D accelerometers, gyroscopes, or magnetometers. Furthermore, state-of-the-art sensors offer sensor fusion capabilities fusing multiple sensor modalities in order to compensate noise and drift or to deliver additional modalities, e.g.,

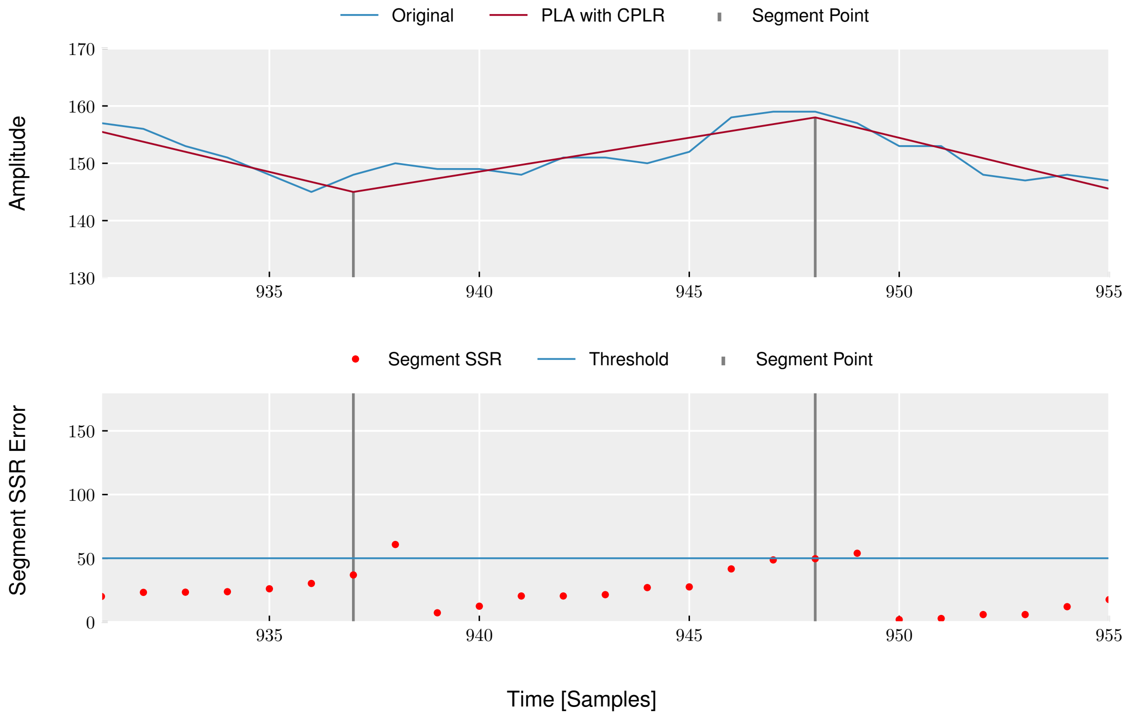


Figure 5.14: Piecewise linear approximation with CPLR including sum of squared residuals segment error per invocation.

orientation in a specified coordinate system. Orientation is often described in quaternions, leading to 4D signals.

As a result, state-of-the-art PLA algorithms support the approximation of multi-dimensional signals. In line with the definition at the beginning of Section 5.2.3, a multi-dimensional sensor signal is described as a series S of sensor samples $s[m] = (\vec{v}[m], \tau_v[m]) \in S^{\mathbb{N}}$ with $S = \mathbb{R}^D \times \mathbb{T}$, $m \in [1, 2, \dots, M]$, M possibly going to infinity, and $D \in \mathbb{N}$ denoting the number of dimensions of the sensor signal. Each sample $(\vec{v}, \tau_v)$ consists of the signal value vector $\vec{v} \in \mathbb{R}^D$ and the corresponding timestamp $\tau_v \in \mathbb{T}$.

As a result, the piecewise linear approximation of that signal is a series $\widetilde{S}$ of segment points $\widetilde{s}[k] = (\vec{\widetilde{v}}[k], \tau_{\widetilde{v}}[k]) \in \widetilde{S}^{\mathbb{N}}$, with $\widetilde{S} = \mathbb{R}^D \times \mathbb{T}$, $k \in [1, 2, \dots, K]$, K possibly going to infinity as well. Each segment point $(\vec{\widetilde{v}}, \tau_{\widetilde{v}})$ consists of a signal value vector $\vec{\widetilde{v}} \in \mathbb{R}^D$ and its corresponding timestamp $\tau_{\widetilde{v}} \in \mathbb{T}$.

At the beginning of this section, the PLA algorithms have been described with $D = 1$. However, both fastSW and CPLR can be applied to multi-dimensional signals, like most existing state-of-the-art PLA algorithms. As a brief description of such, the corresponding variables $\overline{xy}_n$, $\overline{y^2}_n$, $\widehat{\beta}_n$, SSR_n, and y_n are implemented as D-dimensional arrays and calculated individually for each dimension. This is possible, as a single residual error e_i can be calculated as the euclidean distance:

$$e_i = \sqrt{\sum_{d=1}^{D} (e_i^d)^2}. \tag{30}$$

Thus, the squared residual errors e_i^2 are the sums over all squared residuals $(e_i^d)^2$ of each individual dimension $d \in [1, 2, \ldots, D]$, and the error SSR_n of an entire segment of length n is calculated by:

$$SSR_n = \sum_{i=1}^{n} \sum_{d=1}^{D} (y_i^d - \beta_n^d \cdot x_i)^2. \tag{31}$$

The commutative property allows to calculate the segment error SSR_n as the sum of all SSR_n^d of each individual dimension d. Thus, the SSR error metric allows an updating and calculation of all running variables, slopes, and SSR of a segment for each dimension individually. The accumulation of all SSRs of each dimension represents the total SSR of that segment.

5.2.4 Evaluation

In order to evaluate the computational performance as well as the approximation quality of fastSW and CPLR, a comparative evaluation with existing state-of-the-art PLA algorithms has been performed. To this end, SW as described in [125] and the Swing Filter [126] have been chosen, as fastSW and CPLR are based on or related to them, respectively. Additionally, the emSWAB implementation from [127] has been selected as it was specifically developed for resource constrained architectures.

The selected datasets for evaluation are a combination of datasets from evaluations of SW from [125] and [133], mSWAB from [101], and datasets recorded in cooperative projects conducted during the work for the thesis at hand. The compression ratio and approximation error depends on the actual dataset. To this end different datasets have been used for evaluation.

The first dataset (*Kitchen*) was recorded in a project including the recognition of gestures and activities in a kitchen assessment scenario. In this, a sensor was attached to the wrist of a user, recording accelerometer and gyroscope data. The 3D sensor signal vector lengths have been recorded while the user performed kitchen tasks like cutting carrots with a knife, stirring a bowl of ingredients with a wooden spoon, blending ingredients in a bowl with a hand held blender, and using a hand mixer. An example of the accelerometer signal is partially shown in Figure 5.15.

The second dataset (*Walking*) was recorded by using the same setup and sensor modalities as in the kitchen scenario, recording the data of a walking person. The sensor was attached to the shoe of the user. The walking dataset includes straight paths, turns, walking upstairs and downstairs, and also different walking speeds. An example of the accelerometer signal is partially shown in Figure 5.15 as well.

As in the literature PLA algorithms are not only used for activity recognition applications, the evaluation also includes available datasets from literature of other application domains. The "timeseries" dataset (*Timeseries*) that came with the implementation of mSWAB from [101] as well as the datasets from [133] including Electrocardiogram signals (*ECG*), valve time series of a Marotta space shuttle (*Shuttle*), and the time series of a patient's respiration measured by thorax extension (*Respiration*), which are freely available at [132]. Representative extracts of these datasets are shown in Figure 5.15 as well.

As the emSWAB implementation from [127] is one-dimensional and was implemented for and evaluated on 8-bit data, all other PLA implementations process 8-bit data as well. However, as experiments to asses the approximation quality are carried out on a x86_64 architecture, all implementations benefit from internal floating-point operations likewise. The same holds for experiments assessing the algorithm execution time on a microcontroller architecture with floating-point unit, which will

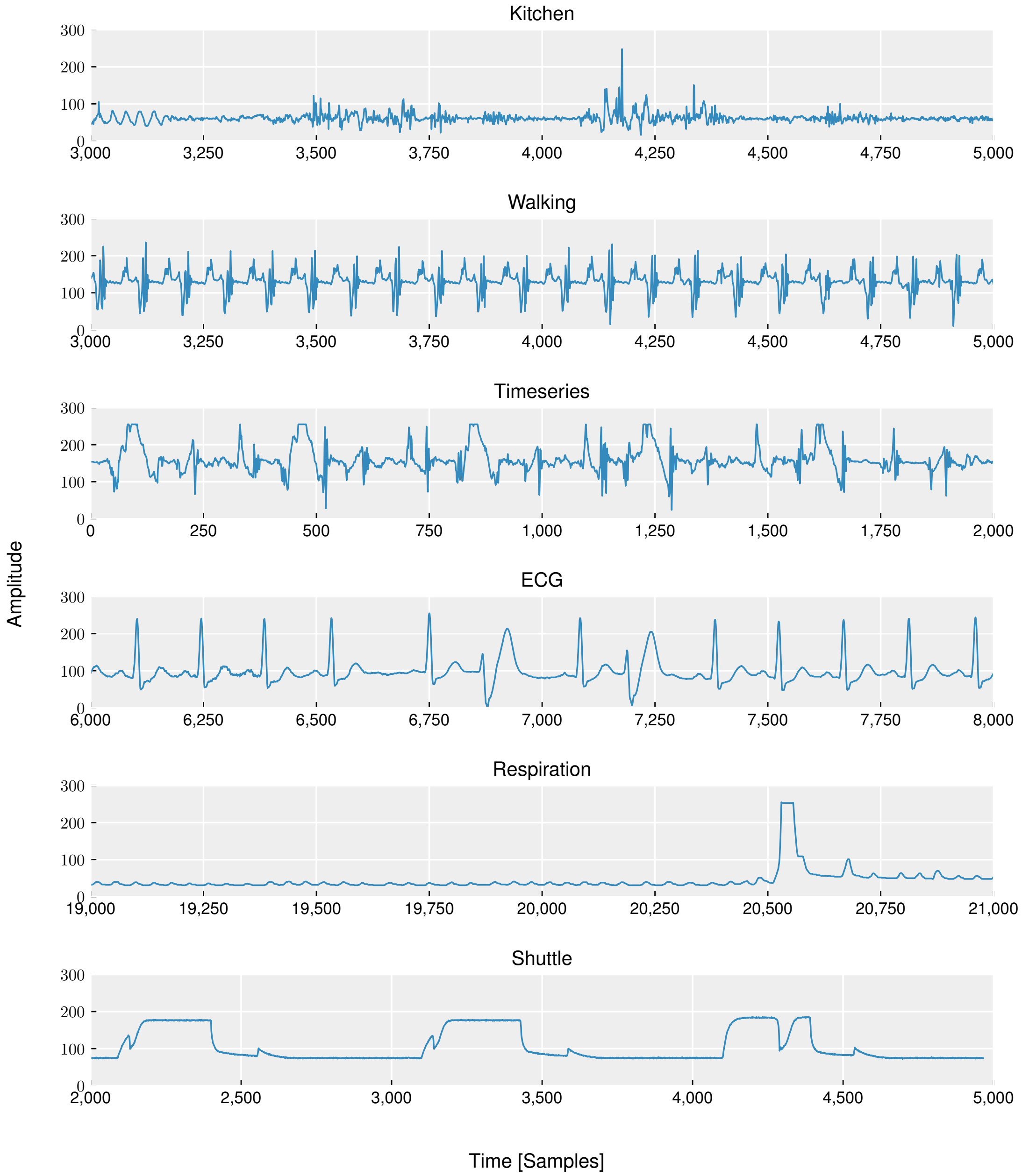


Figure 5.15: Representative excerpts of the datasets for evaluation.

be discussed in the corresponding section in more detail. Although fastSW and CPLR are designed to allow non-equidistantly sampled sensor signals to be approximated, the selected datasets were all sampled with equidistant sampling periods. Thus, the sample number of each dataset has been used as the timestamp for each sample. Furthermore, emSWAB and SW have been set to a maximum buffer size of 100 in all experiments. The Swing filter, as well as fastSW, and CPLR have been implemented without a restriction on the segment length for maximum compression ability, as their design allows.

The two decisive criteria for the approximation quality are the compression ratio and the approximation error. In the thesis at hand, the approximation error is measured in terms of an average sum of squares residual error of the PLA signal to the original signal. Note that the approximation error has to be distinguished from the segment error. The latter is part of the PLA algorithms to limit the absolute sum of squared residuals of a single segment. The approximation error instead, is the average squared residual error of an entire PLA signal w.r.t. its original signal. It is calculated for evaluation purposes. When approximating a dataset with a particular PLA algorithm and a particular threshold TH, the resulting PLA signal is interpolated at the timestamps of the original signal and its SSR is calculated and divided by the number of sensor samples.

Changing the threshold TH influences the resulting approximation error. Although monotonicity cannot be assumed in general, as a trend, the approximation error increases with the threshold. However, likewise the compression ratio is influenced for different threshold values. For the sake of visualization, the Inverse Compression Ratio (ICR) is calculated for an approximated dataset by dividing the number of segmentation points of the resulting PLA signal by the number of original samples. Thus, in the worst case, the ICR is 1.0 when the PLA signal results in the same number of segmentation points as samples in the original signal. Although monotonicity cannot be assumed here neither, the general trend is a decrease in segmentation points, and thus in the ICR, with an increased threshold value.

At a certain threshold, a particular PLA algorithm produces an approximation with a certain approximation error and a certain ICR for a particular dataset. This pair of ICR and approximation error is referred to as operating point in the thesis at hand. While most PLA algorithms in the evaluation are limiting the segment error in terms of a maximum absolute SSR (i.e., SW, fastSW, and CPLR), emSWAB uses a sum of absolute distances and the Swing Filter uses a maximum residual error e_i of each sample to the segment. This causes a different behavior in terms of approximation error and ICR w.r.t. the threshold TH. However, as the threshold TH is not a quality indicator but a control parameter, it is not sufficient to compare different PLA algorithms in terms of approximation error or ICR, respectively, at equal thresholds, as its done in some of the related literature. While SW can reach a certain approximation error at a particular threshold, emSWAB or the Swing Filter could reach the same approximation error at different thresholds. The same applies to the ICR analogously.

As a result, in the thesis at hand, the resulting approximation error of different PLA algorithms is rather compared at similar ICRs. Thus, the resulting operating points of ICR and approximation error together are compared for different PLA algorithms on each individual dataset. However, the approximation of a dataset is a discrete problem w.r.t. the segmentation points and it is generally not always possible to find a threshold, for which different PLA algorithms lead to the very same ICR. To this end, for the evaluation, each dataset is approximated with all PLA algorithms multiple times covering a high range of different threshold values and the resulting approximation errors are plotted over the resulting ICRs. This leads to plots, sketching the dependency between both quality indicators of the different PLA algorithms. The nearer the plotted curve is to the origin of the plot, the better is the approximation quality of a PLA algorithm for that particular dataset.

In the conducted experiments, 29 datasets with in total 536,175 samples have been approximated 100,001 times with fastSW, SW, and CPLR with thresholds from 0 to 100,000, 10,001 times with emSWAB with threshold from 0 to 10,000, and 12,751 times with Swing Filter with threshold from 0 to 12,750. Note that the threshold of the Swing Filter is defined in units of the sensor signal range. For 8-bit data, only 256 threshold values will provide different operating points, which are not enough for a sufficient coverage of possible operating points in comparison to all other evaluated PLA algorithms.

To this end, the actual threshold value of Swing Filter is implemented as a floating-point value scaled by a factor of $1/50$ in the conducted experiments, allowing for a higher range of different threshold values. As a consequence of their different segment error concepts, fastSW, SW, and CPLR have been tested on a higher range of thresholds as emSWAB and Swing Filter, in order to cover a similar range of operating points.

As depicted in Figure 5.16, the trend is a higher average residual error for lower IRCs. However, for the datasets Kitchen, Walking, Timeseries, and ECG, non of the PLA algorithms shows a superior or inferior approximation quality among all others.

Differences in approximation quality between the first four plots are mainly based on signal characteristics, i.e., the ECG dataset allows a higher compression ratio as Kitchen, Walking, and Timeseries, at comparable approximation errors. This results out of a slightly higher amount of linear sections of the signal. Note however, that non of the datasets contains comparably long linear or static signal sections as for example a sensor that lies flat on the ground for several seconds or minutes.

The Respiration dataset allows for a higher data compression w.r.t. the aforementioned datasets, as can be seen in Figure 5.16. This results out of a strong quantization of the signal into an 8-bit format, due to a high signal range. As a result, the signal is composed of fairly linear signal sections, which in itself appear as a piecewise linear signal (see Figure 5.17). From the approximation quality plot of the Respiration dataset in Figure 5.16, it can also be seen that the maximum achievable compression ratio of SW is 0.01 when implemented with a maximum buffer size of 100. The fastSW algorithm instead allows higher compression ratios, as the segment length is unconstrained. With ICRs below approximately 0.025 the approximation quality between SW and fastSW slightly differs, as a result of the segment length constraint of SW. Above ICRs of approximately 0.025 for the Respiration dataset the maximum segment length of SW does not appear to influence the approximation quality of SW, and both algorithms produce nearly identical results.

Furthermore, it can be seen that SF is showing more ambiguity in the approximation quality. This might be explained by the segment error bound of SF. However, in order to substantiate, further investigation is necessary, which is out of scope in the thesis at hand.

In summary, a clear superiority or inferiority of a particular PLA algorithm cannot be deduced from the conducted experiments. In most cases, the approximation quality is comparable among all implemented PLA algorithms, with obvious but not comparatively clear differences for datasets with specific signal characteristics, e.g., higher amounts of linear signal sections in the Shuttle and Respiration dataset. However, the experiments show, that the PLA algorithms introduced by the thesis at hand, i.e., CPLR and fastSW, can cope with existing state-of-the-art PLA algorithms, in terms of approximation quality. While fastSW offers superior execution time benefits compared to SW (which will be evaluated in the next paragraph), it delivers equal results in approximation quality and additionally extends these by eliminating constraints on the segment lengths and thus compression ratio. Furthermore, for each evaluated dataset, one of the evaluated PLA algorithms is best in terms of approximation quality, i.e., is closest to the coordinate system origin, in specific regions. The entirety of all best candidates in their specific regions in the plots (in the field of multi-objective optimization referred to as *pareto front*) is thus representing the state of the art. For the conducted experiments, it can be seen, that CPLR directly contributes to the state of the art by producing results which extend it further towards the origin in individual regions of the plots.

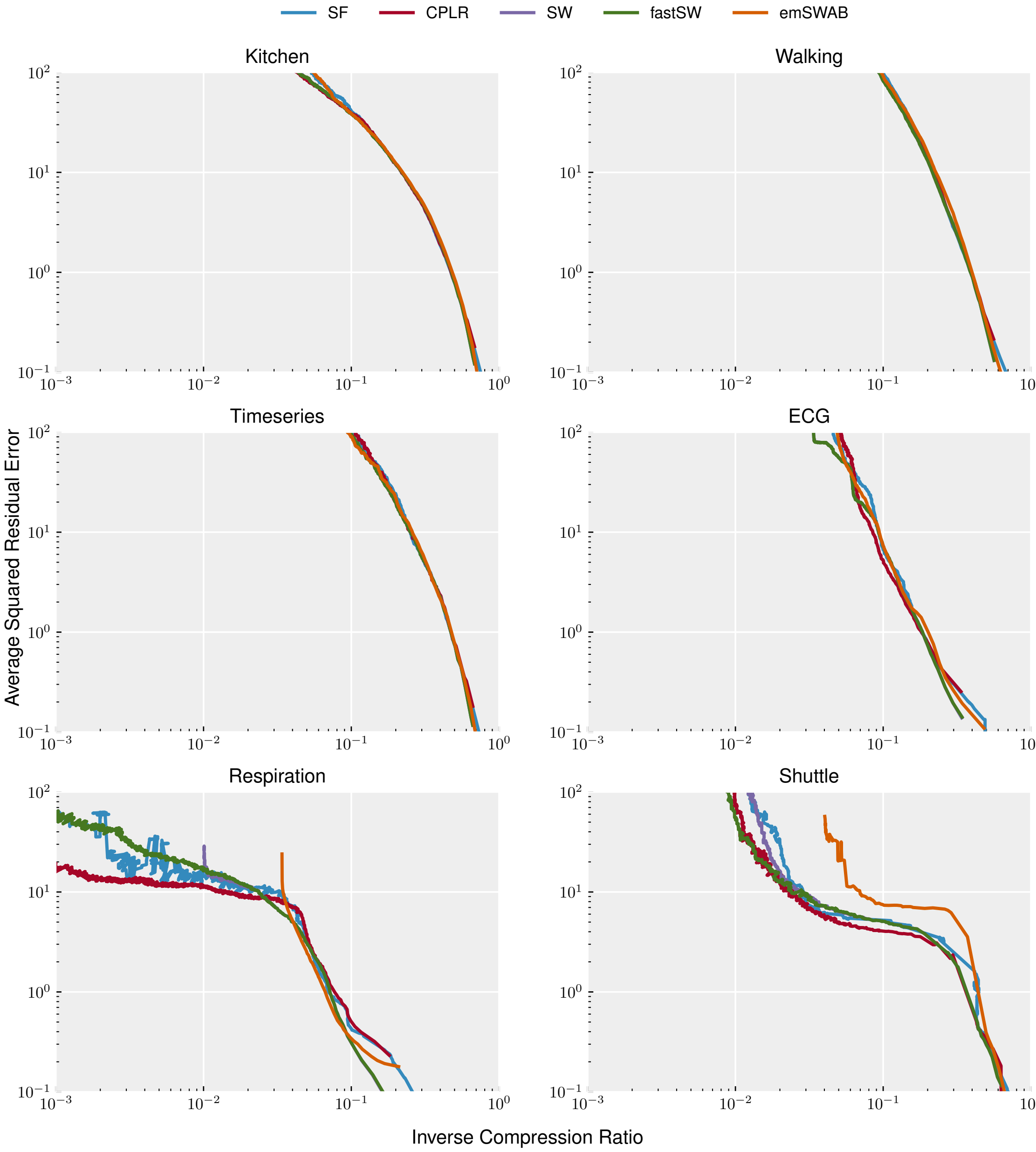


Figure 5.16: Approximation quality of the evaluated PLA algorithms on representative datasets.

Execution time To compare the execution times of CPLR and SW to state-of-the-art PLA algorithms, experiments on a x86_64 architecture have been conducted. In order to eliminate non-deterministic timing effects induced by cache, scheduling, branch miss prediction penalties, and others, an emulative approach based on the Valgrind framework [134] has been chosen. To this end, the binaries compiled with GNU Compiler Collection (GCC) C compiler in version 10.1.0 [135], have been applied to a chosen dataset. The execution time is evaluated in terms of instruction count per invocation of each PLA algorithm. The instruction count has been evaluated by using the tool Callgrind [136].

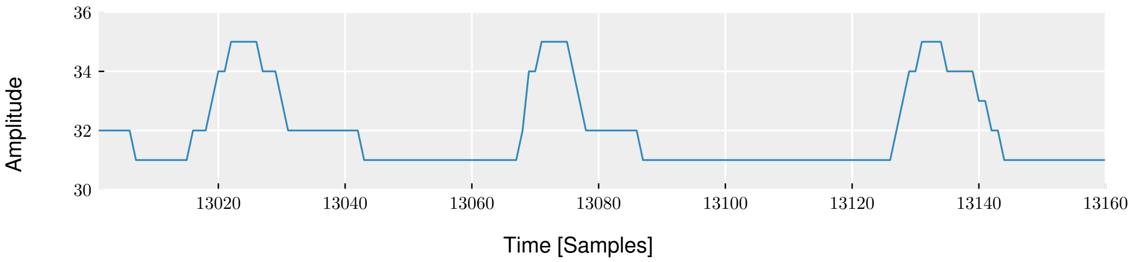


Figure 5.17: Strong quantization of the Respiration signal.

Each PLA algorithm has been used to approximate all 29 datasets and the instruction counts of each invocation have been recorded. The threshold for all PLA algorithms has been chosen as 100 and the maximum buffer size for emSWAB and SW have been set to 100 as well. The minimum, maximum, and average instruction counts, as well as its standard deviation are summarized in Table 5.13.

Although, fastSW, CPLR, and SF have the smallest maximal and average execution times, as well as the smallest standard deviations compared to emSWAB and SW, the data-dependent nature does not get obvious from the results in Table 5.13. Therefore, Figure 5.18, 5.19, 5.21, 5.20, and 5.22 show the instruction count per invocation for an excerpt of the Timeseries signal from Figure 5.15.

For CPLR, fastSW, and SF, the maximum execution time has a constant limit, which can be seen in Figure 5.18, 5.19, and 5.20. For SW, the linearly growing execution time with increasing segment sizes can be observed in Figure 5.21. In Figure 5.22, the instruction count of emSWAB can be seen to be data dependent and unpredictable, as approximation is performed at points in time that are invoked heuristically, based on the signal trajectory. The maximum instruction count is two to almost four magnitudes higher compared to all other PLA algorithms evaluated.

Since the execution times of emSWAB and SW depend on the data-dependent segment length, and thus the ICR, a fair comparison can only be made at similar ICRs. To this end, a threshold has been chosen for each algorithm, that leads to approximately the same ICR. However, as a result, the execution time analysis has been only performed on a single dataset, which has been chosen to be ECG from Figure 5.15. The thresholds have been chosen to produce a PLA that result in a ICR of at most 0.2, i.e., a compression ratio of at least 5:1. For each PLA algorithm, the minimum, maximum, and average

Algorithm	*min IC*	*max IC*	$\overline{IC}$	s_{IC}	#
CPLR	49	75	51.56	7.75	551,145
fastSW	55	56	55.89	0.31	551,145
SF	57	86	66.35	9.88	551,145
SW	22	1,327	259.30	281.95	551,145
emSWAB	38	319,575	1,472.48	10,501.33	551,175

Table 5.13: Instruction counts of CPLR, fastSW, SF, SW, and emSWAB at $TH = 100$ and a maximum buffer size of 100 for SW and emSWAB on an x86-64 architecture.

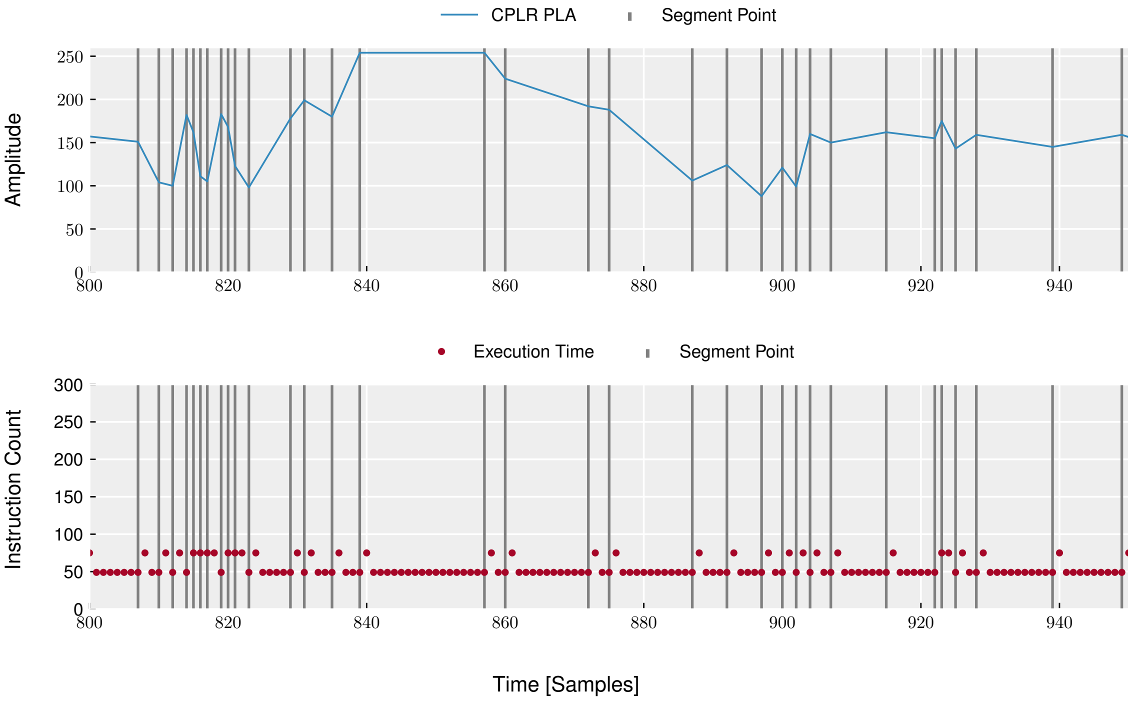


Figure 5.18: Execution time of CPLR with $TH = 100$.

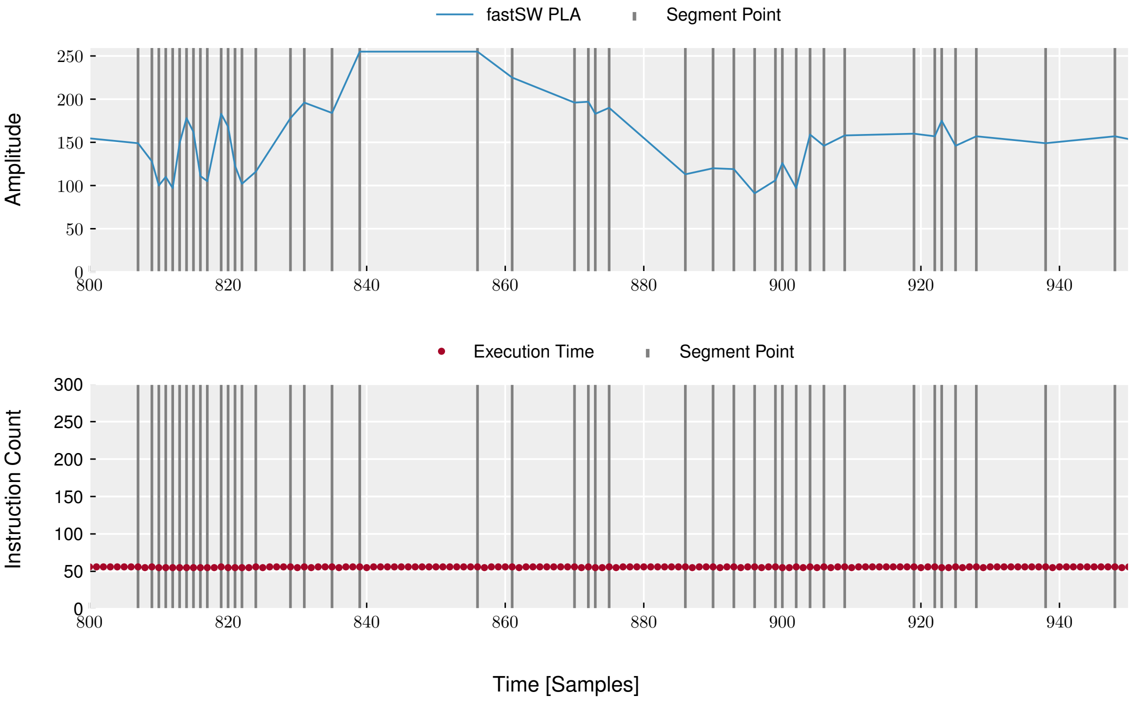


Figure 5.19: Execution time of fastSW with $TH = 100$.

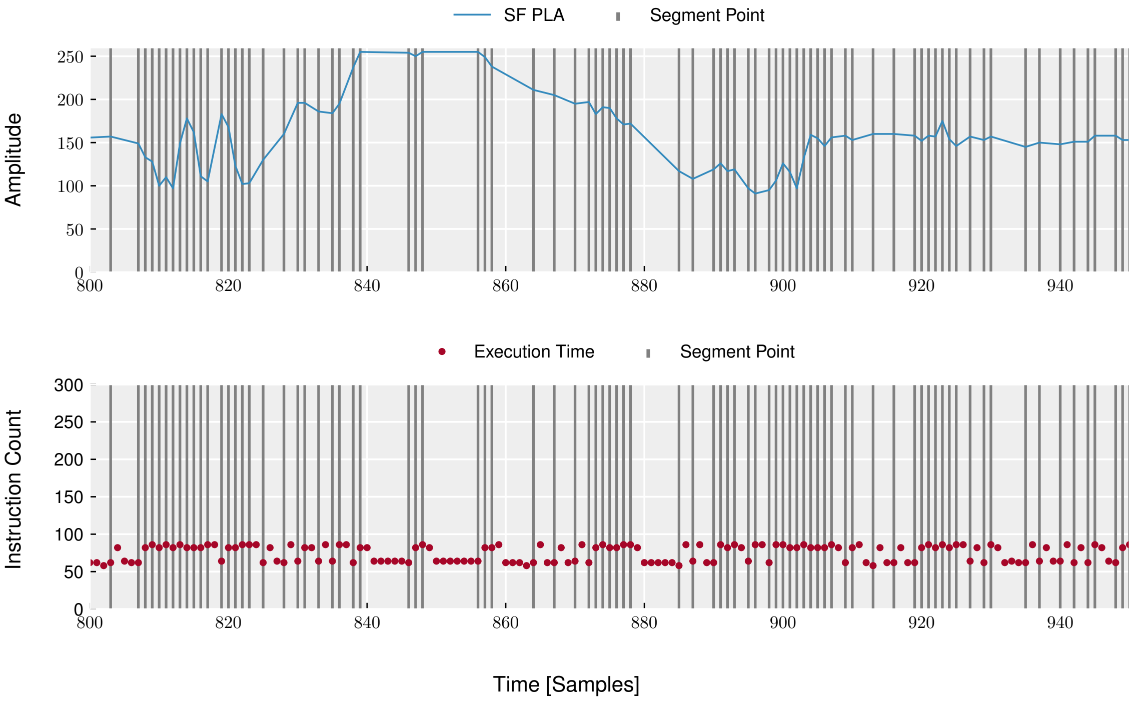


Figure 5.20: Execution time of SF with $TH = 100$.

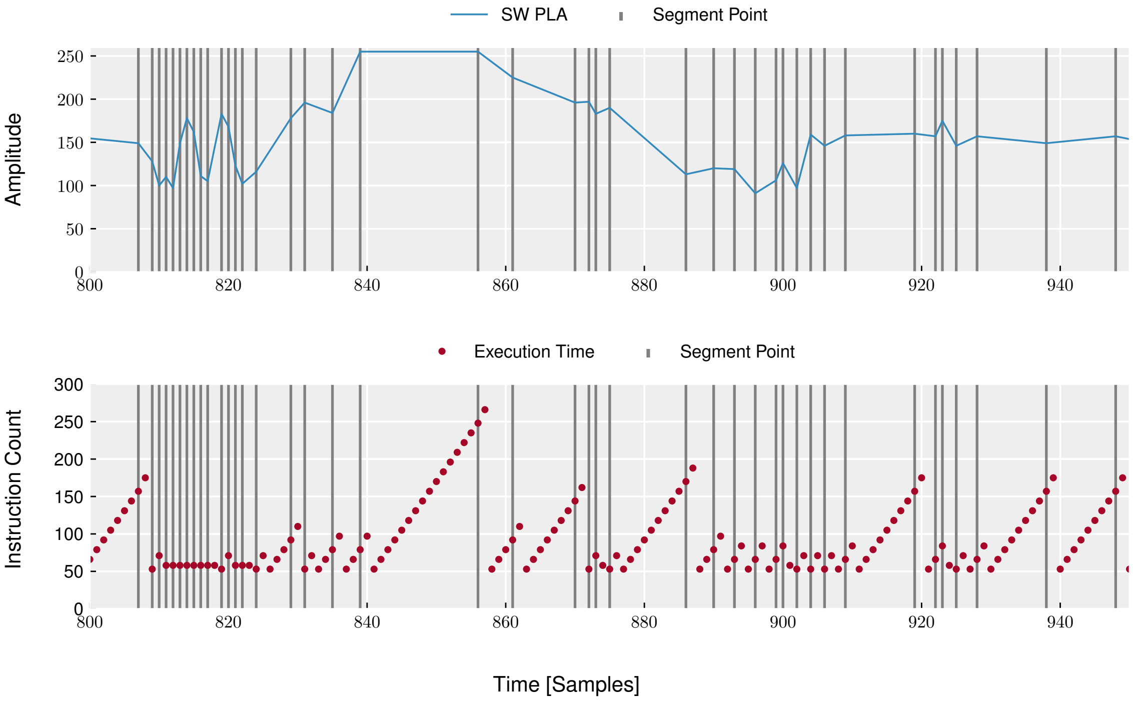


Figure 5.21: Execution time of SW with $TH = 100$.

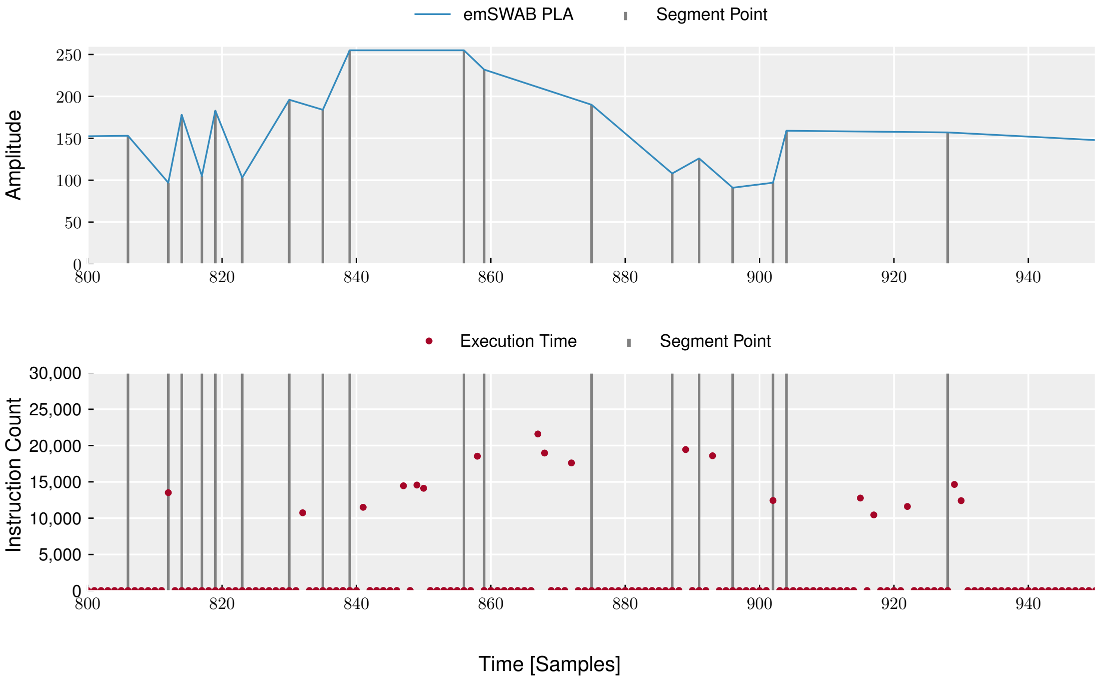


Figure 5.22: Execution time of emSWAB with $TH = 100$.

instruction count per algorithm invocation as well as its standard deviation have been recorded, which are summarized together with the corresponding parameters and resulting ICRs in Table 5.14.

Compared to an invocation with a threshold of 100 (Table 5.13), SW and emSWAB show a smaller maximum instruction count, due to the reduced compression with a smaller threshold, but especially the maximum instruction count of emSWAB is still two to three magnitudes higher than those of CPLR, fastSW, and SF.

Lastly, a WCET analysis based on Control Flow Graphs (CFG) has been conducted for a more representative architecture, i.e., an Arm Cortex-M4 microcontroller. The WCET analysis is based on a static code analysis on the corresponding assembler code, which was compiled with the Arm Embedded

Algorithm	TH	ICR	$min\ IC$	$max\ IC$	$\overline{IC}$	s_{IC}	#
CPLR	8	0.1943	49	75	54.05	10.29	14,999
fastSW	10	0.1972	55	56	55.80	0.40	14,999
SF	110	0.1997	57	86	66.74	8.77	14,999
SW	10	0.1970	36	318	103.34	45.89	14,999
emSWAB	8	0.1907	38	74,325	2,999.09	7,084.31	15,000

Table 5.14: Instruction counts of CPLR, fastSW, SF, SW, and emSWAB on an x86-64 architecture at similar ICRs with a maximum buffer size of 100 for SW and emSWAB, evaluated on the ECG dataset from Figure 5.15.

Algorithm	*min IC*	*max IC*	Computational Complexity
CPLR	46	77	$O(1)$
fastSW	51	59	$O(1)$
SF	53	96	$O(1)$
SW	20	$56+n\cdot 12$	$O(n)$
emSWAB	33	/	$O(n^2)$

Table 5.15: Instruction counts of CPLR, fastSW, SF, SW, and emSWAB on an ARM Cortex-M4 microcontroller.

GCC in version 10.1.0 of the GNU Arm Embedded Toolchain [135]. The Cortex-M4 has been set up as the target platform (command line option `-mcpu=cortex-m4`), the optimization level was set highest w.r.t. execution time (command line option `-O3`), and floating-point-specific instructions were used with Floating-Point Unit (FPU) specific calling conventions (command line option `-mfloat-abi=hard`). From the generated assembler code, control flow graphs with corresponding instruction counts of each basis block have been extracted. For SW, fastSW, SF, and CPLR, the CFGs were simple enough for a manual analysis of the shortest and longest paths, as well as data dependencies of SW. However, the control flow of emSWAB includes nested loops with both, inner and outer loops, iterating over the entire buffer in the worst case. Since previous experiments have substantiated high execution times resulting out of the $O(n^2)$ complexity, an exact WCET analysis from its rather complex CFGs has been abstained from. The minimum and maximum execution times (dependencies) in terms of instruction count as well as the time complexities are summarized in Table 5.15. It can be seen that CPLR, fastSW, and SF, have a data-independent worst-case execution time of 77, 59, and 96 instructions, respectively. The worst-case execution time of SW is at least 56 instructions and increases depending on the segment length by 12 instructions with each new sensor sample. This not only means a constraint of the compression abilities, when the execution time has to be limited for a particular application on a resource constrained hardware architecture, but also results in a increased energy consumption as the average execution time and thus processor utilization increases with longer segments and thus higher data reductions.

5.2.5 Discussion

In this sub-chapter, state-of-the-art PLA algorithms have been optimized towards a constant, deterministic worst-case execution time per sensor sample, without compromising functional properties, i.e., approximation quality. Both, CPLR and fastSW, are based on a user-specified maximum SSR segment error, while an existing alternative, i.e., the swing filter [126], is based on a maximum residual error of each sample. However, an analysis of the effects of different segment error metrics has yet to be performed, which is out of scope for the thesis at hand.

The contributions of this sub-chapter are two novel SSR-based PLA algorithms with constant memory and computational complexity w.r.t. the segment lengths. While CPLR extrapolates segment points from a regression line, segment points of fastSW are a subset of the original signal samples.

From a modeling and analysis perspective, the result show, that for CPLR, fastSW, and SF, a WCET can be determined at design time without constraining their functional properties. This allows for design-time modeling and analysis approaches as presented in the thesis at hand. The data-dependent changes of scenarios confine themselves to the output of a segment or not. Each scenario of the proposed PLA algorithms has itself a deterministic WCET and can thus be modeled with the SADF MoC as presented in Section 5.1.

6 Conclusions

This final chapter summarizes the contributions of this thesis and presents a discussion of the acquired results with respect to the initial research questions and its limitations, as well as recommendations for future work.

6.1 Summary

The thesis at hand introduced model-based design and analysis approaches of human activity recognition (HAR) systems, based on dataflow models of computations. To this end, suitable MoCs have been identified. Furthermore, based on existing analysis approaches for selected MoCs, novel methods have been developed to acquire energy consumption indicators at design time. Energy consumption models have been introduced to capture the relationship to the aforementioned indicators in separate models that are annotated to hardware elements. From these, the impact of design decisions on device energy consumption can be quantified at design time, which extends the state of the art. The introduced approaches have been applied to case studies, and their analysis results have been compared to corresponding experimental implementations. System-level parallelization approaches have been integrated into models and their influence regarding latency, throughput, and processor utilization has been evaluated on the formal models as well as in experimental implementations for the sake of comparison. Furthermore, dynamic behavior of HAR systems has been studied and the model-based design and analysis methods have been extended to SADF models. As a result, dynamic changes of the HAR system can be modeled and model-based analysis of energy consumption indicators can take scenario changes and their occurrence distribution into account. The developed methods contribute to the state of the art. The model accuracy has been evaluated in experimental implementations. Finally, two novel piecewise linear approximation (PLA) algorithms for sensor signals, i.e., CPLR and fastSW, have been introduced, that can be applied to reduce the amount of wirelessly transmitted sensor samples, and thus the energy consumption of wireless transceivers on sensor nodes. Both, CPLR and fastSW, guarantee a bound on the segment error in terms of the sum of squared residual errors. Furthermore, they provide a constant computational as well as memory complexity per sensor sample, without compromising their functional properties, i.e., achievable data reduction. Furthermore, their execution time and memory utilization is small enough to be executed on processing units with harsh resource constrains. Both algorithms have been evaluated against, and contribute to, the state of the art.

6.2 Discussion

In the following, the main results of the thesis at hand are discussed w.r.t. the initial research questions on page 5:

RQ1 **Which dataflow MoC is suitable to capture state-of-the-art human activity recognition systems?**

RQ2 **How can important extra-functional properties be accurately analyzed from dataflow graph models?**

RQ3 **Can existing conceptual optimizations regarding latency, throughput, and energy consumption of sensor-based human activity recognition systems be represented in dataflow graph models?**

RQ4 **Can dynamic behavior of human activity recognition systems be captured by dataflow graph models and corresponding analysis methods?**

The modeling and analysis based on CSDF graphs has been presented in Chapter 3. The acquired accuracy in experimental evaluations substantiates the suitability of CSDF graphs and the chosen level of abstraction for the selected case study without dynamic, data-dependent changes. These results answer research question RQ1, on how to model human activity recognition systems with dataflow MoCs. Furthermore, existing optimizations w.r.t. latency, throughput, and energy consumption, that are based on the reduction of computational effort for sensor signals and features with minor contributions to the recognition accuracy, i.e., feature-selection, sensor-selection, and reduced sensor sampling frequency approaches, can be directly represented in the dataflow model, by means of production and consumption rates as well es execution time annotations. This partially answers research question RQ3 of this thesis.

Furthermore, the introduced analysis methods of processor utilization and communication data rate are indicators for energy consumption. Introduced energy consumption models have been used to annotated hardware components, in order to estimate impacts of design decisions on the total device energy consumption rate of a wireless sensor node. This is a direct contribution to research question RQ2 regarding possible analysis methods for extra-functional properties. The addressed extra-functional properties concern energy consumption.

The system modeling approach, especially the mapping, is representing a common optimization approach w.r.t. energy consumption from the literature. Researchers have proposed to calculate feature extraction on wireless sensor nodes in order to reduce total energy consumption. However, their results are neither generalizable nor quantified w.r.t. the variability in application-specific ARC configurations. The introduced modeling approach however, directly represents the conceptual idea in the mapping of dataflow actors onto a hardware model. The introduced analysis methods, allow to quantify (at least estimate) the impact of mapping decisions w.r.t. device energy consumption at design time. This is a partial answer to research question RQ3 on how existing conceptual optimizations can be represented by dataflow-based design and analysis approaches.

The representation of existing parallelization approaches for the classification stage of HAR systems on MPSoCs and its analysis w.r.t. throughput, latency, and processor utilization is another conceptual optimization from the literature that could be successfully represented in the proposed model-based design and analysis approaches, which partially answers research question RQ3. Furthermore, the analysis of latency from the dataflow graph models has been shown, which partially answers research question RQ2 of this thesis. The acquired accuracy of model-based results compared to an experimental implementation substantiates the chosen MoCs and level of abstraction which the selected case studies have been modeled with.

The proposed model-based design approach has been extended to SADF graphs, that can capture dynamic changes of static scenarios in the model-based representation. The analysis of processor utilization has been extended to SADF graph models. The accuracy of model-based analysis results with measurements from an experimental evaluation show again a sufficient accuracy. However, with inconsistent implementations of the modeled behavior, long-run average results can become rather inaccurate. In the conducted experiments, relative deviations from the estimated average processor utilization have been up to 14.7 %. These deviations are a result of differences between modeled and implemented behavior regarding the order of the decision processor between scenarios.

As a result, SADF provides a possible solution to capture dynamic behavior but either accurate modeling of the desired behavior or consistent implementation of the modeled behavior is crucial in order to preserve model-based analysis results of extra-functional properties in the final system. These results are answering research question RQ4 of this thesis. Furthermore, the concept of scenarios can directly represent existing optimization approaches that are based on dynamic feature-selection or sensor-selection approaches based on environmental changes or remaining battery levels of wireless sensor nodes. Their impact can thus be estimated from SADF-based models. This partially answers research question RQ3 of this thesis.

Finally, the introduction of two novel piecewise linear approximation algorithms contributes to the optimization of HAR systems with respect to energy efficiency, and to the model-based design and analysis of such, as their execution time is deterministic without constraining their functional properties. This contribution goes beyond the initial research questions of the thesis at hand.

6.3 Limitations

The thesis at hand is by no means exhaustive towards the initial research questions it deals with. Consequently, it underlies certain limitations which will be discussed in the following.

While the thesis at hand offers answers to research question RQ1 regarding suitable MoCs, it does not offer a comparison to other existing MoCs that could be possibly utilized for a model-based design and analysis of energy-efficient online human activity recognition systems. Further, the thesis at hand does not attempt to offer applicable solutions for every possible sensor-based HAR system. Especially, it constraints its applicability to HAR systems with a small level of control flow and a small number of dynamically changing scenarios. Furthermore, HAR systems that do not follow the common structure of an activity recognition chain are not considered in the thesis at hand. This consequently excludes deep learning, although a possible approach has been suggested in Chapter 1.

Furthermore, the model-based analysis of memory requirements has entirely not been considered in the thesis at hand. Although respective analysis approaches for signal processing and streaming applications in general exists in the literature and are implemented in SDF^3 [13], these have not been studied in the thesis at hand and rather been abstracted into WCET annotations and abstract energy models. As a result, the influence of memory utilization on the energy consumption has not been directly evaluated as well. Other requirements like safety, security, or robustness of HAR systems are out of scope for this thesis as well.

The introduced analysis methods regarding processor utilization and communication data rate have yet not been analyzed w.r.t. their analysis time, e.g., as an implementation in existing dataflow analysis frameworks like SDF^3 [13, 20].

The model-based design and analysis approach claims to be applicable at design time. However, corresponding execution time annotation to actors, as well as introduced energy consumption models have been acquired on actual implementations, which contrasts the anticipated design flow. In general, annotations based on measurements cannot be regarded as worst-case execution times, as the worst case might not have been observed in the measurements. However, a comparison to analysis results based on annotations that have been acquired at design time has not been conducted and has yet to be evaluated.

In each chapter, certain analysis methods have been evaluated with different parts of the system, e.g., latency analysis for parallelized classification algorithms on MPSoCs or energy consumption estimation of wireless sensor nodes. However, latency has not been analyzed in wireless sensor networks, as well as energy consumption for MPSoCs. As a result, the general applicability of presented methods has been shown, but not exhaustively evaluated in this regard.

Finally, two new piecewise linear approximation algorithms for sensor signals have been introduced, i.e., CPLR and fastSW, which to the best of the author's knowledge, are the only SSR-based PLA algorithms for connected segments, that have a constant computational, as well as memory complexity per sensor sample. However, the swing filter offers constant computational and memory complexity per sample as well, but is rather based on a maximum absolute residual error metric, than a sum of squared residuals. Although the latter is commonly used in the literature, differences in their impact on approximation quality have not extensively been evaluated in the thesis at hand.

6.4 Future Work

The results of the thesis at hand show, that a model-based design and analysis of sensor-based human activity recognition systems is a promising approach to substantiate design decisions based on quantified analysis results of important extra-functional properties like throughput, latency, processor utilization, and energy consumption at design time. A further extension with annotations towards recognition accuracy based on ARC parameters could be an interesting extension, to trade-off functional and extra-functional properties at design time. Such annotation could be acquired from offline evaluations of different HAR software configurations on test datasets, which is common practice for the optimization of ARC configurations w.r.t. recognition accuracy.

The results of this thesis show furthermore, that a precise knowledge about scenario transition probabilities has to be available in order to acquire long-run average metrics from SADF-based system-level models. However, due to the introduced real-time criterion w.r.t. system throughput, scenario occurrence distributions are sufficient to estimate long-run averages. Furthermore, the derivation and specification of scenario occurrence distributions is more straight forward than scenario transition probabilities. Additionally, a specific scenario occurrence distribution or the set of transition probabilities can only model a specific use case of the system. An interesting future work could be the probabilistic modeling of uses cases in form of a Dirichlet distribution on all possible scenario occurrence distributions. This could allow to estimate long-run average metrics for the expected use case of the system but also standard deviation or statistical moments, which could refine design-time estimates on the extra-functional properties.

Finally, the model-based design and analysis of sensor-based human activity recognition systems allows to quantify the impact of design decisions on extra-functional properties like real-time performance, latency, and energy efficiency. However, the huge number of possible design points can easily become infeasible to analyze by the system designer. As the proposed modeling and analysis approaches are based on formal methods, a logical consequence is the combination with state-of-the-art design space exploration approaches. The model-based design and analysis approaches introduced in the thesis at hand can form the basis of automatic design space exploration and multi objective optimization of energy-efficient sensor-based human activity recognition systems.

Bibliography

[1] Johannes Classen, Florian Kult, Dušan Radović, Thomas Zebrowski, Amin Jemili, Andrea Visconti, Chinwuba Ezekwe, Alexander Buhmann, Manuel Dietrich, Axel Grosse, et al. “Evolution of bosch inertial measurement Uunits for consumer electronics”. In: *Proceedings of 2020 IEEE Sensors*. IEEE. 2020, pp. 1–4.

[2] Bosch Sensortec. *BHI160/BHI160B: Ultra low-power sensor hub incl. integrated IMU*. BST-BHI160B-DS000-03, `https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bhi160b-ds000.pdf` [last access: March 2021]. 2020.

[3] Andreas Bulling, Ulf Blanke, and Bernt Schiele. “A tutorial on human activity recognition using body-worn inertial sensors”. In: *ACM Computing Surveys (CSUR)* 46.3, 2014, pp. 1–33.

[4] Jon L. Bentley. “Multidimensional binary search trees used for associative searching”. In: *Communications of the ACM* 18.9, 1975, pp. 509–517.

[5] Sander Stuijk, Marc C. W. Geilen, Bart D. Theelen, and Twan Basten. “Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications”. In: *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*. IEEE. 2011, pp. 404–411.

[6] Edward A. Lee and David G. Messerschmitt. “Synchronous data flow”. In: *Proceedings of the IEEE* 75.9, 1987, pp. 1235–1245.

[7] Sundararajan Sriram and Shuvra S. Bhattacharyya. *Embedded multiprocessors: Scheduling and synchronization*. 2nd ed. CRC Press, 2009.

[8] Amir H. Ghamarian, Marc C. W. Geilen, Sander Stuijk, Twan Basten, Bart D. Theelen, Mohammad R. Mousavi, Arno J. M. Moonen, and Marco J. G. Bekooij. “Throughput analysis of synchronous data flow graphs”. In: *Proceedings of the Sixth International Conference on Application of Concurrency to System Design*. IEEE. 2006, pp. 25–36.

[9] Edward A. Lee and Soonhoi Ha. “Scheduling strategies for multiprocessor real-time DSP”. In: *Proceedings of the Global Telecommunications Conference and Exhibition "Communications Technology for the 1990s and Beyond"*. IEEE. 1989, pp. 1279–1283.

[10] Amir H. Ghamarian, Marc C. W. Geilen, Twan Basten, Bart D. Theelen, Mohammad R. Mousavi, and Sander Stuijk. “Liveness and boundedness of synchronous data flow graphs”. In: *Proceedings of the Conference on Formal Methods in Computer Aided Design*. IEEE. 2006, pp. 68–75.

[11] Ali Dasdan. “Experimental analysis of the fastest optimum cycle ratio and mean algorithms”. In: *ACM Transactions on Design Automation of Electronic Systems* 9.4, 2004, pp. 385–418.

[12] Robert de Groote, Jan Kuper, Hajo Broersma, and Gerard J. M. Smit. “Max-plus algebraic throughput analysis of synchronous dataflow graphs”. In: *Proceedings of the 38th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE. 2012, pp. 29–38.

[13] Sander Stuijk, Marc C. W. Geilen, and Twan Basten. "SDF3: SDF for free". In: *Proceedings of the Sixth International Conference on Application of Concurrency to System Design*. IEEE. 2006, pp. 276–278.

[14] Greet Bilsen, Marc Engels, Rudy Lauwereins, and Jean Peperstraete. "Cycle-static dataflow". In: *IEEE Transactions on Signal Processing* 44.2, 1996, pp. 397–408.

[15] Sander Stuijk, Marc C. W. Geilen, and Twan Basten. "Throughput-buffering trade-off exploration for cyclo-static and synchronous dataflow graphs". In: *IEEE Transactions on Computers* 57.10, 2008, pp. 1331–1345.

[16] Robert de Groote, Philip K. F. Hölzenspies, Jan Kuper, and Gerard J. M. Smit. "Multi-rate equivalents of cyclo-static synchronous dataflow graphs". In: *Proceedings of the 14th International Conference on Application of Concurrency to System Design*. IEEE. 2014, pp. 62–71.

[17] Robert de Groote. "Throughput analysis of dataflow graphs". In: *Handbook of Signal Processing Systems*. Springer, 2019, pp. 751–786.

[18] Edward A. Lee and Thomas M. Parks. "Dataflow process networks". In: *Proceedings of the IEEE* 83.5, 1995, pp. 773–801.

[19] Bart D. Theelen, Marc C. W. Geilen, Twan Basten, Jeroen P. M. Voeten, Stefan V. Gheorghita, and Sander Stuijk. "A scenario-aware data flow model for combined long-run average and worst-case performance analysis". In: *Proceedings of the Fourth ACM and IEEE International Conference on Formal Methods and Models for Co-Design*. IEEE. 2006, pp. 185–194.

[20] Bart D. Theelen. "A performance analysis tool for scenario-aware streaming applications". In: *Proceedings of the Fourth International Conference on the Quantitative Evaluation of Systems*. IEEE. 2007, pp. 269–270.

[21] Kai L. Chung. "Markov chains". In: *Springer-Verlag, New York*, 1967.

[22] Marc C. W. Geilen and Sander Stuijk. "Worst-case performance analysis of synchronous dataflow scenarios". In: *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. ACM. 2010, pp. 125–134.

[23] Bart D. Theelen, Marc C. W. Geilen, and Jeroen P. M. Voeten. "Performance model checking scenario-aware dataflow". In: *Formal Modeling and Analysis of Timed Systems*. Springer, 2011, pp. 43–59.

[24] Bart Kienhuis, Ed Deprettere, Kees Vissers, and Pieter Van Der Wolf. "An approach for quantitative analysis of application-specific dataflow architectures". In: *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors*. IEEE. 1997, pp. 338–349.

[25] Ernesto Wandeler, Lothar Thiele, Marcel Verhoef, and Paul Lieverse. "System architecture evaluation using modular performance analysis: A case study". In: *International Journal on Software Tools for Technology Transfer* 8.6, 2006, pp. 649–667.

[26] Bart D. Theelen. “Performance model generation for MPSoC design-space exploration”. In: *Proceedings of the Fifth International Conference on Quantitative Evaluation of Systems*. IEEE. 2008, pp. 39–40.

[27] Sander Stuijk, Marc C. W. Geilen, and Twan Basten. “A predictable multiprocessor design flow for streaming applications with dynamic behaviour”. In: *Proceedings of the 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*. IEEE. 2010, pp. 548–555.

[28] Kai Neubauer, Philipp Wanko, Torsten Schaub, and Christian Haubelt. “Exact multi-objective design space exploration using ASPmT”. In: *Proceedings of the Conference on Design, Automation and Test in Europe*. IEEE. 2018, pp. 257–260.

[29] Kurt Keutzer, A. Richard Newton, Jan M. Rabaey, and Alberto Sangiovanni-Vincentelli. “System-level design: Orthogonalization of concerns and platform-based design”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 19.12, 2000, pp. 1523–1543.

[30] Morteza Damavandpeyma, Sander Stuijk, Twan Basten, Marc C. W. Geilen, and Henk Corporaal. “Modeling static-order schedules in synchronous dataflow graphs”. In: *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium. 2012, pp. 775–780.

[31] Morteza Damavandpeyma, Sander Stuijk, Twan Basten, Marc C. W. Geilen, and Henk Corporaal. “Schedule-extended synchronous dataflow graphs”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32.10, 2013, pp. 1495–1508.

[32] Alok Lele, Orlando Moreira, and Pieter J. L. Cuijpers. “A new data flow analysis model for TDM”. In: *Proceedings of the tenth International Conference on Embedded Software*. ACM. 2012, pp. 237–246.

[33] Christian Zebelein, Christian Haubelt, Joachim Falk, Tobias Schwarzer, and Jürgen Teich. “Representing mapping and scheduling decisions within dataflow graphs”. In: *Proceedings of the Forum on Specification & Design Languages*. IEEE. 2013, pp. 1–8.

[34] Amir H. Ghamarian, Marc C. W. Geilen, Twan Basten, and Sander Stuijk. “Parametric throughput analysis of synchronous data flow graphs”. In: *Proceedings of the Conference on Design, Automation and Test in Europe*. IEEE. 2008, pp. 116–121.

[35] Arno J. M. Moonen, Marco J. G. Bekooij, Rene van den Berg, and Jef van Meerbergen. “Practical and accurate throughput analysis with the cyclo static dataflow model”. In: *Proceedings of the 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. IEEE. 2007, pp. 238–245.

[36] Mark A. Hanson, Harry C. Powell Jr., Adam T. Barth, Kyle Ringgenberg, Benton H. Calhoun, James H. Aylor, and John Lach. “Body area sensor networks: Challenges and opportunities”. In: *Computer* 42.1, 2009, pp. 58–65.

[37] Maarten H. Wiggers, Marco J. G. Bekooij, and Gerard J. M. Smit. “Efficient computation of buffer capacities for cyclo-static dataflow graphs”. In: *Proceedings of the 44th ACM/IEEE Design Automation Conference*. IEEE. 2007, pp. 658–663.

[38] Mohamed Benazouz, Olivier Marchetti, Alix Munier-Kordon, and Thierry Michel. “A new method for minimizing buffer sizes for cyclo-static dataflow graphs”. In: *Proceedings of the 8th Workshop on Embedded Systems for Real-Time Multimedia*. IEEE. 2010, pp. 11–20.

[39] Amir H. Ghamarian, Sander Stuijk, Twan Basten, Marc C. W. Geilen, and Bart D. Theelen. “Latency minimization for synchronous data flow graphs”. In: *Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools*. IEEE. 2007, pp. 189–196.

[40] Firew Siyoum, Marc C. W. Geilen, and Henk Corporaal. “Symbolic analysis of dataflow applications mapped onto shared heterogeneous resources”. In: *Proceedings of the 51st ACM/EDAC/IEEE Design Automation Conference*. IEEE. 2014, pp. 1–6.

[41] Firew Siyoum, Marc C. W. Geilen, and Henk Corporaal. “End-to-end latency analysis of dataflow scenarios mapped onto shared heterogeneous resources”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35.4, 2016, pp. 535–548.

[42] Jad Khatib, Alix Munier-Kordon, Enagnon C. Klikpo, and Kods Trabelsi-Colibet. “Computing latency of a real-time system modeled by synchronous dataflow graph”. In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. ACM. 2016, pp. 87–96.

[43] Guus Kuiper and Marco J. G. Bekooij. “Latency analysis of homogeneous synchronous dataflow graphs using timed automata”. In: *Proceedings of the Conference on Design, Automation and Test in Europe*. IEEE. 2017, pp. 902–905.

[44] Mohamed A. Bamakhrama and Todor Stefanov. “Managing latency in embedded streaming applications under hard-real-time scheduling”. In: *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Dodesign and System Synthesis*. ACM. 2012, pp. 83–92.

[45] Anup Das, Akash Kumar, and Bharadwaj Veeravalli. “Energy-aware task mapping and scheduling for reliable embedded computing systems”. In: *ACM Transactions on Embedded Computing Systems (TECS)* 13.2s, 2014, pp. 1–27.

[46] Anup Das, Akash Kumar, and Bharadwaj Veeravalli. “Reliability and energy-aware mapping and scheduling of multimedia applications on multiprocessor systems”. In: *IEEE Transactions on Parallel and Distributed Systems* 27.3, 2015, pp. 869–884.

[47] Vasilios Konstantakos, Alexander Chatzigeorgiou, Spiridon Nikolaidis, and Theodore Laopoulos. “Energy consumption estimation in embedded systems”. In: *IEEE Transactions on Instrumentation and Measurement* 57.4, 2008, pp. 797–804.

[48] Mostafa Bazzaz, Mohammad Salehi, and Alireza Ejlali. "An accurate instruction-level energy estimation model and tool for embedded systems". In: *IEEE Transactions on Instrumentation and Measurement* 62.7, 2013, pp. 1927–1934.

[49] Yi Wang, Jialiu Lin, Murali Annavaram, Quinn A. Jacobson, Jason Hong, Bhaskar Krishnamachari, and Norman Sadeh. "A framework of energy efficient mobile sensing for automatic user state recognition". In: *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services*. ACM. 2009, pp. 179–192.

[50] Michael I. Gordon, William Thies, and Saman Amarasinghe. "Exploiting coarse-grained task, data, and pipeline parallelism in stream programs". In: *ACM SIGPLAN Notices* 41.11, 2006, pp. 151–162.

[51] Piero Zappi, Clemens Lombriser, Thomas Stiefmeier, Elisabetta Farella, Daniel Roggen, Luca Benini, and Gerhard Tröster. "Activity recognition from on-body sensors: Accuracy-power trade-off by dynamic sensor selection". In: *Wireless Sensor Networks*. Springer, 2008, pp. 17–33.

[52] Zhixian Yan, Vigneshwaran Subbaraju, Dipanjan Chakraborty, Archan Misra, and Karl Aberer. "Energy-efficient continuous activity recognition on mobile phones: An activity-adaptive approach". In: *Proceedings of the 16th International Symposium on Wearable Computers*. IEEE. 2012, pp. 17–24.

[53] Yunji Liang, Xingshe Zhou, Zhiwen Yu, and Bin Guo. "Energy-efficient motion related activity recognition on mobile devices for pervasive healthcare". In: *Mobile Networks and Applications* 19.3, 2014, pp. 303–317.

[54] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge L. Reyes-Ortiz. "Energy efficient smartphone-based activity recognition using fixed-point arithmetic." In: *Journal of Universal Computer Science* 19.9, 2013, pp. 1295–1314.

[55] Kristof Van Laerhoven and Andre Kvist Aronsen. "Memorizing what you did last week: Towards detailed actigraphy with a wearable sensor". In: *Proceedings of the 27th International Conference on Distributed Computing Systems Workshops*. IEEE. 2007, pp. 47–47.

[56] Kristof Van Laerhoven, Hans-Werner Gellersen, and Yanni G. Malliaris. "Long term activity monitoring with a wearable sensor node". In: *Proceedings of the International Workshop on Wearable and Implantable Body Sensor Networks*. IEEE. 2006, pp. 1–4.

[57] Konrad Lorincz, Bor-rong Chen, Geoffrey W. Challen, Atanu R. Chowdhury, Shyamal Patel, Paolo Bonato, Matt Welsh, et al. "Mercury: A wearable sensor network platform for high-fidelity motion analysis." In: *Proceedings of the Conference on Embedded Networked Sensor Systems*. Vol. 9. ACM. 2009, pp. 183–196.

[58] Atis Elsts, Ryan McConville, Xenofon Fafoutis, Niall Twomey, Robert J. Piechocki, Raul Santos-Rodriguez, and Ian Craddock. "On-board feature extraction from acceleration data for

activity recognition." In: *Proceedings of the International Conference on Embedded Wireless Systems and Networks*. ACM. 2018, pp. 163–168.

[59] Xenofon Fafoutis, Letizia Marchegiani, Atis Elsts, James Pope, Robert Piechocki, and Ian Craddock. "Extending the battery lifetime of wearable sensors with embedded machine learning". In: *Proceedings of the 4th World Forum on Internet of Things*. IEEE. 2018, pp. 269–274.

[60] Tifenn Rault, Abdelmadjid Bouabdallah, Yacine Challal, and Frédéric Marin. "A survey of energy-efficient context recognition systems using wearable sensors for healthcare applications". In: *Pervasive and Mobile Computing* 37, 2017, pp. 23–44.

[61] Yuan Tian, Jarupan Boangoat, Eylem Ekici, and Fusun Ozguner. "Real-time task mapping and scheduling for collaborative in-network processing in DVS-enabled wireless sensor networks". In: *Proceedings of the 20th International Parallel & Distributed Processing Symposium*. IEEE. 2006, 10–pp.

[62] Yuan Tian, Eylem Ekici, and Fusun Ozguner. "Energy-constrained task mapping and scheduling in wireless sensor networks". In: *Proceedings of the International Conference on Mobile Adhoc and Sensor Systems*. IEEE. 2005, 8–pp.

[63] Alvise Bonivento and Alberto L. Sangiovanni-Vincentelli. "Platform based design for wireless sensor networks". In: *Proceedings of the 2nd International Workshop on Networking with Ultra Wide Band and Workshop on Ultra Wide Band for Sensor Networks. Networking with UWB*. IEEE. 2005, pp. 9–19.

[64] Alvise Bonivento, Luca P. Carloni, and Alberto L. Sangiovanni-Vincentelli. "Rialto: A bridge between description and implementation of control algorithms for wireless sensor networks". In: *Proceedings of the 5th International Conference on Embedded Software*. ACM. 2005, pp. 183–186.

[65] Frank Krüger, Martin Nyolt, Kristina Yordanova, Albert Hein, and Thomas Kirste. "Computational state space models for activity and intention recognition. A feasibility study". In: *PloS one* 9.11, 2014, p. e109381.

[66] B. P. Welford. "Note on a method for calculating corrected sums of squares and products". In: *Technometrics* 4.3, 1962, pp. 419–420.

[67] Philippe P. Pebay. *Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments.* Tech. rep. Sandia National Laboratories, 2008.

[68] Robertas Damaševičius, Mindaugas Vasiljevas, Justas Šalkevičius, and Marcin Woźniak. "Human activity recognition in AAL environments using random projections". In: *Computational and Mathematical Methods in Medicine* 2016, 2016.

[69] Tâm Huynh and Bernt Schiele. "Analyzing features for activity recognition". In: *Proceedings of the 2005 Joint Conference on Smart Objects and Ambient Intelligence: Innovative Context-Aware Services: Usages and Technologies*. ACM. 2005, pp. 159–163.

[70] Bruno Bodin, Alix Munier-Kordon, and Benoît D. de Dinechin. “Periodic schedules for cyclo-static dataflow”. In: *Proceedings of the 11th Symposium on Embedded Systems for Real-time Multimedia*. IEEE. 2013, pp. 105–114.

[71] Christian Zebelein, Christian Haubelt, Joachim Falk, and Jürgen Teich. “Model-based representation of schedules for dataflow graphs.” In: *Proceedings of the Workshop “Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen”*. Citeseer. 2013, pp. 105–115.

[72] Samarjit Chakraborty, Simon Kunzli, and Lothar Thiele. “A general framework for analysing system properties in platform-based embedded system designs”. In: *Proceedings of the Conference on Design, Automation and Test in Europe*. IEEE, 2003, p. 10190.

[73] Lothar Thiele and Nikolay Stoimenov. “Modular performance analysis of cyclic dataflow graphs”. In: *Proceedings of the Seventh International Conference on Embedded software*. ACM. 2009, pp. 127–136.

[74] Bosch Sensortec. *BMI160 - Small, low power inertial measurement unit*. BST-BMI160-DS0001-08, `https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmi160-ds000.pdf` [last access: March 2021]. 2018.

[75] John D. Cook. *Computing skewness and kurtosis in one pass*. `https://www.johndcook.com/blog/skewness_kurtosis/` [last access: March 2021]. 2014.

[76] James W. Cooley and John W. Tukey. “An algorithm for the machine calculation of complex Fourier series”. In: *Mathematics of Computation* 19.90, 1965, pp. 297–301.

[77] NORDIC Semiconductor. *BLE on Android v1.0.1*. Revision 1.0.1, `https://devzone.nordicsemi.com/cfs-file/__key/communityserver-discussions-components-files/4/BLE_5F00_on_5F00_Android_5F00_v1.0.1.pdf` [last access: March 2021]. 2016.

[78] Jeffry T. Russell and Margarida F. Jacome. “Software power estimation and optimization for high performance, 32-bit embedded processors”. In: *Proceedings of the International Conference on Computer Design. VLSI in Computers and Processors*. IEEE. 1998, pp. 328–333.

[79] Milton Friedman. “The use of ranks to avoid the assumption of normality implicit in the analysis of variance”. In: *Journal of the American Statistical Association* 32.200, 1937, pp. 675–701.

[80] Frank Wilcoxon. “Individual comparisons by ranking methods”. In: *Breakthroughs in Statistics*. Springer, 1992, pp. 196–202.

[81] Khaled Assaleh, Tamer Shanableh, and Mohammed Zourob. “Low complexity classification system for glove-based Arabic sign language recognition”. In: *Proceedings of the International Conference on Neural Information Processing*. Springer. 2012, pp. 262–268.

[82] Rachel C. King, Louis Atallah, Ara Darzi, and Guang-Zhong Yang. “An HMM framework for optimal sensor selection with applications to BSN sensor glove design”. In: *Proceedings of the 4th Workshop on Embedded Networked Sensors*. ACM. 2007, pp. 58–62.

[83] John Weissmann and Ralf Salomon. “Gesture recognition for virtual reality applications using data gloves and neural networks”. In: *Proceedings of the International Joint Conference on Neural Networks*. Vol. 3. IEEE. 1999, pp. 2043–2046.

[84] Davide Figo, Pedro C. Diniz, Diogo R. Ferreira, and João M. P. Cardoso. “Preprocessing techniques for context recognition from accelerometer data”. In: *Personal and Ubiquitous Computing* 14.7, 2010, pp. 645–662.

[85] Fariha Muzaffar, Bushra Mohsin, Farah Naz, and Farooq Jawed. “DSP implementation of voice recognition using dynamic time warping algorithm”. In: *Proceedings of the Student Conference on Engineering Sciences and Technology*. IEEE. 2005, pp. 1–7.

[86] David J. Burr, Bryan Ackland, and Neil Weste. “A high speed array computer for dynamic time warping”. In: *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*. Vol. 6. IEEE. 1981, pp. 471–474.

[87] Y. J. Bae and Michael C. Fairhurst. “Parallelism in dynamic time warping for automatic signature verification”. In: *Proceedings of 3rd International Conference on Document Analysis and Recognition*. Vol. 1. IEEE. 1995, pp. 426–429.

[88] Mark A. Yoder and Leah J. Siegel. “Dynamic time warping algorithms for SIMD machines and VLSI processor arrays”. In: *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*. Vol. 7. IEEE. 1982, pp. 1274–1277.

[89] Sergio Herrero-Lopez, John R. Williams, and Abel Sanchez. “Parallel multiclass classification using SVMs on GPUs”. In: *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. ACM. 2010, pp. 2–11.

[90] Qing He, Fuzhen Zhuang, Jincheng Li, and Zhongzhi Shi. “Parallel implementation of classification algorithms based on MapReduce”. In: *Proceedings of the International Conference on Rough Sets and Knowledge Technology*. Springer. 2010, pp. 655–662.

[91] Jeffrey Dean and Sanjay Ghemawat. “MapReduce: Simplified data processing on large clusters”. In: *Communications of the ACM* 51.1, 2008, pp. 107–113.

[92] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. “Dryad: Distributed data-parallel programs from sequential building blocks”. In: *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*. ACM. 2007, pp. 59–72.

[93] Mohammed J. Zaki, Ching-Tien Ho, and Rakesh Agrawal. “Parallel classification for data mining on shared-memory multiprocessors”. In: *Proceedings of the 15th International Conference on Data Engineering*. IEEE. 1999, pp. 198–205.

[94] Padmanabhan S. Pillai, Lily B. Mummert, Steven W. Schlosser, Rahul Sukthankar, and Casey J. Helfrich. “Slipstream: Scalable low-latency interactive perception on streaming data”. In:

Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video. ACM. 2009, pp. 43–48.

[95] Ming-Yu Chen, Lily B. Mummert, Padmanabhan S. Pillai, Alexander Hauptmann, and Rahul Sukthankar. “Exploiting multi-level parallelism for low-latency activity recognition in streaming video”. In: *Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems*. ACM. 2010, pp. 1–12.

[96] William Thies, Michal Karczmarek, and Saman Amarasinghe. “StreamIt: A language for streaming applications”. In: *Proceedings of the International Conference on Compiler Construction*. Springer. 2002, pp. 179–196.

[97] Weijia Che and Karam S. Chatha. “Unrolling and retiming of stream applications onto embedded multicore processors”. In: *Proceedings of the 49th Annual Design Automation Conference*. ACM. 2012, pp. 1272–1277.

[98] Jayanth Gummaraju, Joel Coburn, Yoshio Turner, and Mendel Rosenblum. “Streamware: Programming general-purpose multicore processors using streams”. In: *ACM SIGARCH Computer Architecture News* 36.1, 2008, pp. 297–307.

[99] Manjunath Kudlur and Scott Mahlke. “Orchestrating the execution of stream programs on multicore platforms”. In: *ACM SIGPLAN Notices* 43.6, 2008, pp. 114–124.

[100] Kin Yu, John Mason, and John Oglesby. “Speaker recognition using hidden Markov models, dynamic time warping and vector quantisation”. In: *IEE Proceedings-Vision, Image and Signal Processing* 142.5, 1995, pp. 313–318.

[101] Kristof Van Laerhoven, Eugen Berlin, and Bernt Schiele. “Enabling efficient time series analysis for wearable activity data”. In: *Proceedings of the International Conference on Machine Learning and Applications*. IEEE. 2009, pp. 392–397.

[102] Yu-Liang Hsu, Cheng-Ling Chu, Yi-Ju Tsai, and Jeen-Shing Wang. “An inertial pen with dynamic time warping recognizer for handwriting and gesture recognition”. In: *IEEE Sensors Journal* 15.1, 2014, pp. 154–163.

[103] Toni M. Rath and Raghavan Manmatha. “Word image matching using dynamic time warping”. In: *Proceedings of the Conference on Computer Vision and Pattern Recognition, 2003*. Vol. 2. IEEE. 2003, pp. II–II.

[104] Hong Cheng, Jun Luo, and Xuewen Chen. “A windowed dynamic time warping approach for 3D continuous hand gesture recognition”. In: *Proceedings of the International Conference on Multimedia and Expo*. IEEE. 2014, pp. 1–6.

[105] Ahmad Akl and Shahrokh Valaee. “Accelerometer-based gesture recognition via dynamic-time warping, affinity propagation, & compressive sensing”. In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2010, pp. 2270–2273.

[106] Hiroaki Sakoe and Seibi Chiba. “Dynamic programming algorithm optimization for spoken word recognition”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26.1, 1978, pp. 43–49.

[107] Fumitada Itakura. “Minimum prediction residual principle applied to speech recognition”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 23.1, 1975, pp. 67–72.

[108] Stan Salvador and Philip Chan. “Toward accurate dynamic time warping in linear time and space”. In: *Intelligent Data Analysis* 11.5, 2007, pp. 561–580.

[109] Hai N. Tran, Shuvra S. Bhattacharyya, Jean-Pierre Talpin, and Thierry Gautier. “Toward efficient many-core acheduling of partial expansion graphs”. In: *Proceedings of the 21st International Workshop on Software and Compilers for Embedded Systems*. ACM. 2018, pp. 100–103.

[110] George F. Zaki, William Plishker, Shuvra S. Bhattacharyya, and Frank Fruth. “Partial expansion graphs: Exposing parallelism and dynamic scheduling opportunities for DSP applications”. In: *Proceedings of the 23rd International Conference on Application-Specific Systems, Architectures and Processors*. IEEE. 2012, pp. 86–93.

[111] Bosch Sensortec. *BNO055: Intelligent 9-axis absolute orientation sensor*. BST-BNO055-DS000-17, `https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bno055-ds000.pdf` [last access: March 2021]. 2020.

[112] Zoltán Prekopcsák. *Accelerometer based real-time gesture recognition*. Tech. rep. Budapest University of Technology and Economics, 2008.

[113] Texas Instruments. *TMS320C6678 multicore fixed and floating-point digital signal processor*. SPRS691E, `https://www.ti.com/lit/ds/symlink/tms320c6678.pdf` [last access: March 2021]. 2014.

[114] Gene M. Amdahl. “Validity of the single processor approach to achieving large scale computing capabilities”. In: *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*. ACM. 1967, pp. 483–485.

[115] Maarten H. Wiggers, Marco J. G. Bekooij, and Gerard J. M. Smit. “Buffer capacity computation for throughput-constrained modal task graphs”. In: *ACM Transactions on Embedded Computing Systems* 10.2, 2011, pp. 1–59.

[116] Mladen Skelin, Marc C. W. Geilen, Francky Catthoor, and Sverre Hendseth. “Parameterized dataflow scenarios”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36.4, 2016, pp. 669–682.

[117] Morteza Damavandpeyma, Sander Stuijk, Twan Basten, Marc C. W. Geilen, and Henk Corporaal. “Throughput-constrained DVFS for scenario-aware dataflow graphs”. In: *Proceedings of the 19th Real-Time and Embedded Technology and Applications Symposium*. IEEE. 2013, pp. 175–184.

[118] Marc C. W. Geilen, Sander Stuijk, and Twan Basten. “Predictable dynamic embedded data processing”. In: *Proceedings of the International Conference on Embedded Computer Systems*. IEEE. 2012, pp. 320–327.

[119] Yang Yang, Marc C. W. Geilen, Twan Basten, Sander Stuijk, and Henk Corporaal. “Playing games with scenario-and resource-aware SDF graphs through policy iteration”. In: *Proceedings of the Conference on Design, Automation and Test in Europe*. IEEE. 2012, pp. 194–199.

[120] David A. Levin and Yuval Peres. *Markov chains and mixing times*. Vol. 107. American Mathematical Society, 2017.

[121] Joseph T. Buck and Edward A. Lee. “Scheduling dynamic dataflow graphs with bounded memory using the token flow model”. In: *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*. Vol. 1. IEEE. 1993, pp. 429–432.

[122] Gilles Kahn. “The semantics of a simple language for parallel programming”. In: *Proceedings of the IFIP Congress 74*. IFIP. 1974, pp. 471–475.

[123] Eugen Berlin and Kristof Van Laerhoven. “Detecting leisure activities with dense motif discovery”. In: *Proceedings of the Conference on Ubiquitous Computing*. ACM. 2012, pp. 250–259.

[124] Thomas Stiefmeier, Daniel Roggen, and Gerhard Tröster. “Gestures are strings: Efficient online gesture spotting and classification using string matching”. In: *Proceedings of the 2nd International Conference on Body Area Networks*. ACM. 2007, pp. 1–8.

[125] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. “An online algorithm for segmenting time series”. In: *Proceedings of the International Conference on Data Mining*. IEEE. 2001, pp. 289–296.

[126] Hazem Elmeleegy, Ahmed Elmagarmid, Emmanuel Cecchet, Walid G. Aref, and Willy Zwaenepoel. “Online piece-wise linear approximation of numerical streams with precision guarantees”. In: *Proceedings of the International Conference on Very Large Data Bases*. ACM. 2009.

[127] Eugen Berlin and Kristof Van Laerhoven. “An on-line piecewise linear approximation technique for wireless sensor networks”. In: *Proceedings of the Local Computer Network Conference*. IEEE. 2010, pp. 905–912.

[128] Chong Liu, Kui Wu, and Jian Pei. “An energy-efficient data collection framework for wireless sensor networks by exploiting spatiotemporal correlation”. In: *IEEE Transactions on Parallel and Distributed Systems* 18.7, 2007, pp. 1010–1023.

[129] Ngoc D. Pham, Trong D. Le, and Hyunseung Choo. “Enhance exploring temporal correlation for data collection in WSNs”. In: *Proceedings of the International Conference on Research, Innovation and Vision for the Future in Computing and Communication Technologies*. IEEE. 2008, pp. 204–208.

[130] Erich Fuchs, Thiemo Gruber, Jiri Nitschke, and Bernhard Sick. “Online segmentation of time series based on polynomial least-squares approximations”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.12, 2010, pp. 2232–2245.

[131] Ge Luo, Ke Yi, Siu-Wing Cheng, Zhenguo Li, Wei Fan, Cheng He, and Yadong Mu. “Piecewise linear approximation of streaming time series data with max-error guarantees”. In: *Proceedings of the 31st International Conference on Data Engineering*. IEEE. 2015, pp. 173–184.

[132] Eamonn Keogh, Jessica Lin, and Ada Fu. `https://www.cs.ucr.edu/~eamonn/discords/`. [last access: March 2021]. 2005.

[133] Eamonn Keogh, Jessica Lin, and Ada Fu. “HOT SAX: Finding the most unusual time series subsequence: Algorithms and applications”. In: *Proceedings of the Internatial Conference on Data Mining*. Citeseer. 2004, pp. 440–449.

[134] Nicholas Nethercote and Julian Seward. “Valgrind: A framework for heavyweight dynamic binary instrumentation”. In: *ACM Sigplan notices* 42.6, 2007, pp. 89–100.

[135] The Free Software Foundation. *GCC, the GNU Compiler Collection,* `http://gcc.gnu.org`. Version 10.1.0.

[136] Valgrind Developers. *Callgrind: A call-graph generating cache and branch prediction profiler*. 2010.

List of Publications

[G1] Florian Grützmacher, Johann-Peter Wolff, Albert Hein, Polichronis Lepidis, Rainer Dorsch, Thomas Kirste, and Christian Haubelt. “Towards energy efficient sensor nodes for online activity recognition”. In: *Proceedings of the 43rd Annual Conference of the IEEE Industrial Electronics Society*. IEEE. 2017, pp. 8291–8296.

[G2] Florian Grützmacher, Albert Hein, Thomas Kirste, and Christian Haubelt. “Model-based design of energy-efficient human activity recognition systems with wearable sensors”. In: *Technologies* 6.4, 2018, p. 89.

[G3] Florian Grützmacher, Benjamin Beichler, and Christian Haubelt. “Model-based real time analysis of distributed human activity recognition stages in wireless sensor networks”. In: *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and the 2019 International Symposium on Wearable Computers*. ACM. 2019, pp. 1–4.

[G4] Florian Grützmacher, Albert Hein, Benjamin Beichler, Polichronis Lepidis, Rainer Dorsch, Thomas Kirste, and Christian Haubelt. “Energy efficient on-sensor processing for online activity recognition”. In: *Proceedings of the 8th International Joint Conference on Pervasive and Embedded Computing and Communication Systems*. SciTePress, 2018, pp. 85–92.

[G5] Florian Grützmacher, Johann-Peter Wolff, and Christian Haubelt. “Exploiting thread-level parallelism in template-based gesture recognition with dynamic time warping”. In: *Proceedings of the 2nd International Workshop on Sensor-Based Activity Recognition and Interaction*. ACM. 2015, pp. 1–6.

[G6] Florian Grützmacher, Johann-Peter Wolff, and Christian Haubelt. “Sensor-based online hand gesture recognition on multi-core DSPs”. In: *Proceedings of the Global Conference on Signal and Information Processing*. IEEE. 2015, pp. 898–902.

[G7] Florian Grützmacher, Benjamin Beichler, Christian Haubelt, and Bart D. Theelen. “Dataflow-based modeling and performance analysis for online gesture recognition”. In: *Proceedings of the 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS*. IEEE. 2016, pp. 1–8.

[G8] Florian Grützmacher, Benjamin Beichler, Albert Hein, Thomas Kirste, and Christian Haubelt. “Time and memory efficient online piecewise linear approximation of sensor signals”. In: *Sensors* 18.6, 2018, p. 1672.

A Implementation Details on the Hand Gesture Recognition System

In the following, implementation details about the hand gesture recognition system on the TMS320C6678 8-core DSP architecture are given, that has been implemented for the experiments in Chapter 4. In order to realize different parallelization approaches of the gesture recognition system, the initiator-worker-evaluator structure as described in Section 4.4 is applied to the eight DSP cores of the TMS320C6678 architecture. Thus, one DSP core is acting as initiator, six as worker cores and one as evaluator.

In order to implement the gesture recognition system, the SYS/BIOS real-time kernel from Texas Instruments has been used as operating system. The SYS/BIOS is an preemptive multi-threading real-time kernel offering scheduling, synchronization, and instrumentation methods for embedded applications. The implementation of the gesture recognition system is described along the ARC in the following.

Segmentation The sliding window segmentation has been implemented as a single thread on the initiator core. At initialization, the test gesture sequence is loaded into a shared region of the DDR3 SRAM of the board. The shared region can be accessed by all processor cores, and acts as the communication channel of sliding window data. Furthermore, in order to coordinate synchronization between the processor cores during initialization, the initiator core loads the templates into a local scratchpad memory, which will be distributed to the worker cores upon initialization, depending on the parallelization configuration.

The sliding window segmentation is performed in a while loop with a blocking semaphore pend (acquiring a semaphore) operation at the loop entry. This semaphore is released by a second, higher priority task which is triggered by a clock module every $40ms$. By doing so, the online execution of the segmentation stage is simulated and each $40ms$ a new window will be released.

As the main synchronization method between different DSP cores *messages* are used. Messages have to be allocated to be sent over *message queues* which are provided by TI's `messageQ` module. A message queue is a communication channel that is established between two particular threads in the system, as the sender and receiver has to register the queue and keep handles to it in its local memory. Therefore, six message queues from the initiator to the worker cores are registered as well as six message queues from the worker cores to the evaluator, respectively. Likewise, a message queue from the evaluator back to the initiator is deployed. Each thread keeps one or multiple message queues to messages it sends and receives. Message queue reads are performed blocking, in order to provide synchronization between cores upon data transfers. After sending a message, the local pointer of the message is set to null in order to keep track of unavailable worker cores. A message queue read operation sets the corresponding message pointer after successful return. By defining messages for particular message queue paths and sending the corresponding information within the message header, the initiator and evaluator keep track from which worker core a message came and belongs to.

After release of the semaphore, the initiator checks its local message queues by non-blocking reads. Each successful read evaluates to which worker core ID the received message belongs to and assigns its address to the corresponding local pointer. A successful read further triggers another non-blocking read, until no further read function returns successfully, i.e., no new message has arrived. Afterwards, the local messages will be checked for availability. If the messages from every worker core belonging to a tile are present at the initiator, the tile is marked as available and represents a possible candidate for the current sliding window to be sent to. After checking all the local messages for availability, the current sliding window will be sent as a corresponding pointer in the payload in the messages to all worker cores which belong to the selected tile. The actual data transfer of sliding window data will be performed on the worker core by memory reads from the corresponding location in DDR3 SRAM. Additional to sending the window pointer, the initiator informs the corresponding worker core if an EMA calculation shall be performed. As the EMA only needs to be calculated once per window, the initiator is responsible for informing each worker core through a corresponding entry in the message if EMA calculation should be performed, depending on the parallelization configuration. In general, the EMA calculation is assigned to the last core, i.e. the worker core with the highest core ID within a tile. This information is sent along with the window pointer to each worker core of an available tile.

Depending on a successful sending of messages to the worker cores of a tile, the corresponding message pointers in the local array will be set to null to indicate unavailability of the tile. However, in case of no available tiles, due to exceeding execution time of all worker cores or evaluator w.r.t. the sliding window period, the current window will be skipped and the initiator does not sent a message to any worker core. Independent of a successful window dispatching, the loop ends by shifting the sliding window pointer by two samples, i.e., $36 bytes$ and re-loops to the blocking semaphore pend at the loop entry again. The exit condition of the loop is that the number of remaining test sequence samples is smaller than the window size, i.e., after $(N_{TS} - N_{WS})/2$ sliding windows, with N_{TS} being the number of test sequence samples, N_{WS} being the window size in samples, i.e., 125, and 2 represents the offset from two consecutive windows in number of samples.

DTW distance calculation On the worker cores, the actual DTW calculation between the sliding window sent to a worker core and all its templates is performed. To this end, depending on the executed parallelization configuration, each worker core receives its assigned templates upon initialization. These are stored in the L2 scratchpad memories of the worker cores, as they do not fit into L1P SRAM. The L1P and L1D SRAMs have been fully configured as cache, which however, has been disabled to avoid unpredictable timing behavior. The L2 SRAMs have been fully utilized as scratchpad memories. Furthermore, each worker core allocates the memory for the DTW cost matrix, whose pointer will be handed over to the DTW process as a function parameter. Since the DTW calculation is implemented in a row wise fashion, and the sliding window has a constant window size, the signal contained in the window is placed along a row, while each calculated row corresponds to a new sample in the particular template. Therefore, the DTW cost matrix is fixed to two rows which corresponds to twice as much as the sliding window size, i.e. 250 samples. This way, the memory consumption of the DTW calculation is kept equal among all worker cores and independent from the parallelization configuration and only the execution time of each DTW invocation is directly dependent on the particular template length.

Similar to the segmentation on the initiator core, the DTW distance and EMA calculation is performed in a loop. Here, a blocking message queue read is performed at the beginning, which a) serves as synchronization mechanism between initiator and worker cores, and b) acts as communication channel. After the blocking message read returns due to a dispatched sliding window, the sliding window pointer as well as the EMA flag are read and stored in local variables. However, the message is kept at this point, as it is also used for communication of results to the evaluator. The sliding window content of 125 samples is copied from the window pointer destination in DDR3 SRAM into an allocated scratchpad memory of the worker core. After copying the sliding window content into scratchpad memory, the worker core successively calculates the DTW distance of the signal contained in the window to each of its assigned templates and saves the resulting distance to each template in a corresponding location of an array which is also contained in the messages. Moreover, if the EMA flag was set by the initiator within the message, the worker core calculates the EMA value for the window and stores the result in a corresponding variable in the message. Finally, the message containing all DTW distances to the processed templates and the EMA, if calculated, is sent to the evaluator and the loop starts another iteration with a blocking message queue read at its entry.

DTW distance evaluation The evaluation of DTW distances for each sliding window is performed on the evaluator core. Again a loop with a blocking message queue read at its entry is used as synchronization and communication method. When the message queue read returns, the message pointer will be assigned to the corresponding local message pointer array of the evaluator. As each message can have its origin on a different tile, the evaluator checks its local message pointer array for non-null entries of each tile. If one or more entries are set to null, processing on one or more worker cores of a tile is still ongoing. If that is the case, the evaluator loop re-iterates and waits for the next message.

In the case that messages from each core of one of the tiles are present at the evaluator, the EMA value of the last worker core of that tile is compared to the activation and release thresholds as described in Section 4.5. As the EMA-based gesture detection is a state-based method, the Boolean state is recorded at the evaluator core and updated according to the calculated EMA values received from the worker cores for each window. Depending on the updated state, i.e., a gesture is currently performed or not, each DTW distance is compared to the class-dependent threshold and all other DTW distances from that tile, in order to find the template with the smallest DTW distance to the signal contained in the window. If the current gesture state is true and at least one of the template distances is below its class-dependent threshold, the class label of that template is assigned to the sliding window. Otherwise, the null class label is assigned to the window.

The actual class labels are not further sent or processed by other instances in the system, but recorded in a file, which will be saved after processing an entire test sequence of gestures. After evaluating a window, the evaluator sends all messages belonging to the processing tile back to the initiator core and assigns null to the corresponding message pointer entries of its local array, before re-iterating the loop with the next blocking message queue read operation. The sent back messages are synchronization events for the initiator, to indicate that one of the tiles is now available again for processing another window.

B Supplementary Experimental Data

In the following, additional information about the evaluation of model accuracy in Section 5.1 is provided.

Table B.1 lists average ($\bar{t}_{rsp}$), standard deviation ($s_{t_{rsp}}$), coefficient of variation ($CV_{t_{rsp}}$), minimum ($min\ t_{rsp}$), and maximum ($max\ t_{rsp}$) response times measured from the implementation of scenario LP in configurations C1, C2, C3, and C6, as well as their real-time capabilities.

Table B.1: Response time t_{rsp} measured from the implementation in scenario LP.

Cfg.	#	$\bar{t}_{rsp}$ $[ns]$	$s_{t_{rsp}}$ $[ns]$	$CV_{t_{rsp}}$	$min\ t_{rsp}$ $[ns]$	$max\ t_{rsp}$ $[ns]$	Real Time
C1	6,546	83,086	118	0.14 %	82,601	84,221	Yes
C2	7,041	83,103	98	0.12 %	82,613	83,795	Yes
C3	6,913	83,078	86	0.10 %	82,589	83,597	Yes
C6	6,517	83,167	84	0.10 %	82,667	83,665	Yes

Table B.2 lists the processor utilization $\widehat{U}$ of worker cores W1-W6 analyzed from SADF models of scenario LP in configurations C1, C2, C3, and C6. $\overline{\widehat{U}}$ denotes the average processor utilization among the worker cores.

Table B.2: Processor utilization acquired from the SADF analysis models in scenario LP.

Cfg.	$\widehat{U}$(W1)	$\widehat{U}$(W2)	$\widehat{U}$(W3)	$\widehat{U}$(W4)	$\widehat{U}$(W5)	$\widehat{U}$(W6)	$\overline{\widehat{U}}$
C1	0,0000 %	0,0000 %	0,0000 %	0,0000 %	0,0000 %	0,1181 %	0,0197 %
C2	0,0000 %	0,0000 %	0,0590 %	0,0000 %	0,0000 %	0,0590 %	0,0197 %
C3	0,0000 %	0,0394 %	0,0000 %	0,0394 %	0,0000 %	0,0394 %	0,0197 %
C6	0,0197 %	0,0197 %	0,0197 %	0,0197 %	0,0197 %	0,0197 %	0,0197 %

Table B.3 lists the measured processor utilization U of worker cores W1-W6 from the implementation of scenario LP in configurations C1, C2, C3, and C6. $\overline{U}$ denotes the average processor utilization among the worker cores.

Table B.3: Measured processor utilization from the implementation in scenario LP.

Cfg.	U(W1)	U(W2)	U(W3)	U(W4)	U(W5)	U(W6)	$\overline{U}$
C1	0.000 %	0.000 %	0.000 %	0.000 %	0.000 %	0.133 %	0.022 %
C2	0.000 %	0.000 %	0.067 %	0.000 %	0.000 %	0.067 %	0.022 %
C3	0.000 %	0.045 %	0.000 %	0.045 %	0.000 %	0.045 %	0.022 %
C6	0.022 %	0.022 %	0.022 %	0.022 %	0.022 %	0.022 %	0.022 %

Table B.4 lists the processor utilization $\widehat{U}$ of worker cores W1-W6 analyzed from SADF models including transitions between scenarios LP and FP in configurations C1, C2, C3, and C6. $\overline{\widehat{U}}$ denotes the average processor utilization among the worker cores.

Table B.4: Processor utilization acquired from the SADF analysis models with scenario transitions.

Cfg.	$\widehat{U}$(W1)	$\widehat{U}$(W2)	$\widehat{U}$(W3)	$\widehat{U}$(W4)	$\widehat{U}$(W5)	$\widehat{U}$(W6)	$\overline{\widehat{U}}$
C2	22.46 %	22.32 %	20.64 %	22.46 %	22.32 %	20.64 %	21.81 %
C3	22.85 %	20.77 %	22.85 %	20.77 %	22.85 %	20.77 %	21.81 %
C6	21.80 %	21.80 %	21.80 %	21.80 %	21.80 %	21.80 %	21.80 %

Table B.3 lists the measured processor utilization U of worker cores W1-W6 from the implementation including transitions between scenarios LP and FP in configurations C1, C2, C3, and C6. $\overline{U}$ denotes the average processor utilization among the worker cores.

Table B.5: Measured processor utilization from the implementation with scenario transitions.

Cfg.	U(W1)	U(W2)	U(W3)	U(W4)	U(W5)	U(W6)	$\overline{U}$
C2	23.83 %	23.70 %	19.63 %	23.63 %	23.50 %	19.47 %	22.30 %
C3	24.76 %	20.18 %	24.45 %	19.93 %	25.67 %	20.93 %	22.65 %
C6	24.12 %	23.60 %	24.12 %	22.55 %	22.55 %	24.13 %	23.51 %

Table B.6 lists the relative differences Δ_r between model-based results and measured processor utilization U of worker cores W1-W6 and their average $\overline{U}$, based on the a posteriori observed scenario occurrence distribution in the experiments.

Table B.6: Relative difference of processor utilization between model-based and measured results with a posteriori observed scenario occurrence probabilities.

Cfg.	$\Delta_r\, U$(W1)	$\Delta_r\, U$(W2)	$\Delta_r\, U$(W3)	$\Delta_r\, U$(W4)	$\Delta_r\, U$(W5)	$\Delta_r\, U$(W6)	$\Delta_r\, \overline{U}$
C2	1.064 %	0.987 %	0.998 %	1.927 %	1.851 %	1.860 %	1.448 %
C3	2.325 %	2.262 %	3.624 %	3.556 %	−1.378 %	−1.428 %	1.493 %
C6	−1.172 %	1.077 %	−1.172 %	5.883 %	5.882 %	−1.175 %	1.554 %

In Figure B.1, the total variation distance δ_{tv} of the Markov chain of the SADF analysis model in configuration C1 is plotted over the time (number of graph iterations) t. The Markov chain mixing time at $\delta_{tv} \leq 0.1\,\%$ is reached after 159 iterations.

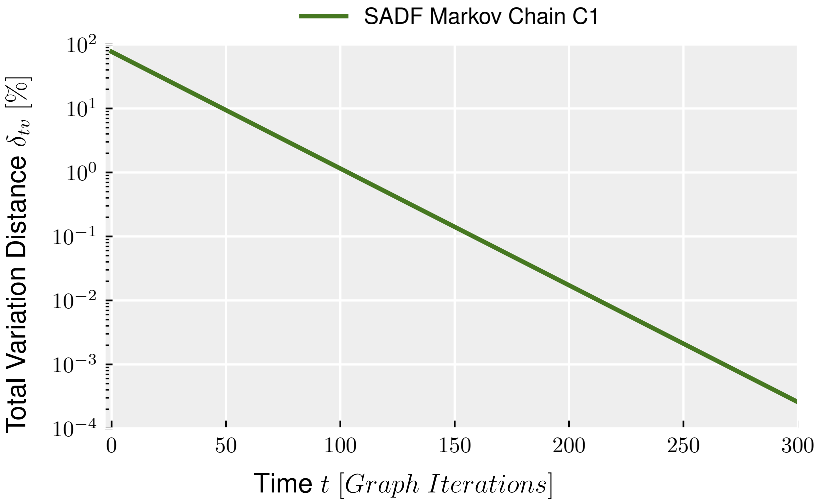


Figure B.1: Total variation distance δ_{tv} between the Markov chain state occurrence probabilities at time t and its corresponding equilibrium distribution.

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Dissertation selbstständig verfasst und ausschließlich die angegebenen Quellen und Hilfsmittel in Anspruch genommen habe, sowie alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht habe. Die vorliegende Arbeit wurde in gleicher oder ähnlicher Fassung bisher nicht als Prüfungsarbeit zur Begutachtung vorgelegt.

Rostock, 29. März 2021

Florian Grützmacher