Improved hard real-time scheduling and transformations for embedded Streaming Applications

Spasic, J.

Author: Spasic, J.
Title: Improved hard real-time scheduling and transformations for embedded Streaming Applications
Issue Date: 2017-11-14

Chapter 1

Introduction

IN the modern-day world, electronics is not only a tool for survival but an integral part of almost every aspect of human lives. Everything from our home appliances, cars, tablets to our cell-phones uses electronics or electronic components in some way. Constantly improving, the electronics technology is making life faster, easier and more convenient for people. Modern electronics technology is rapidly changing the way people communicate and transmit data and information. Thus, it is possible and common today to execute work related tasks remotely. Health-care systems have also benefited a lot from electronics technology. There, electronics technology is helping doctors accurately diagnose and treat illnesses in a timely manner. For example, in Philips Healthcare, live image guided intervention has been used in treatment of structural heart diseases. Using electronics in home automation received popularity in the past decades. People have the capability to control almost everything in their "smart homes" from heating, air conditioning, and lighting, to kitchen appliances and security systems.

Even though electronics technology has been used in all of the above cases, in each case it has its dedicated purpose within a larger system it has been embedded into, hence the name "embedded electronics". Embedded electronics, that is, **embedded systems**, are tightly coupled to the environment in which they operate. They collect information about the environment through sensors and control that environment through actuators, hence embedded systems must provide real-time guarantees, that is, a correct on-time output [Mar06]. Given that embedded systems are dedicated towards a certain application, they are designed to implement well-defined set of functionalities. In addition, having that many embedded systems are battery-operated they have to be efficient in terms of energy consumption and resource usage.

An important class of embedded systems are **embedded streaming systems**. Embedded streaming systems process a long, potentially infinite, stream of input data coming from the environment. Each data item is processed for a limited time. The processing operations on different data items are self-contained and there is little control flow between the operations. The result of the processing is a long, potentially infinite, stream of output data fed into the environment. Usually, streaming applications must process a large amount of data within short periods of time. Thus, efficiency, in terms of both *throughput* and *latency*, is of primary concern in the design of embedded streaming systems. The throughput represents the rate at which output data items are produced, while the latency represents the time interval between the arrival of a data item to the application input and the production of the corresponding data item at the application output. Examples of streaming applications include audio beamforming, video encoding and decoding, image and signal processing, network protocol processing, navigation, computer vision, and others.

One of the key properties of embedded streaming systems is that their correct functionality depends not only on the correct result but also on the time at which the result is produced. Such systems, where the timing is critical to the correct functionality, are called **real-time systems**. Real-time systems can be classified into **hard** and **soft** systems. A hard real-time system is one where not meeting the timing requirements leads to a system failure, which, in life-critical systems, may have catastrophic consequences. In contrast, in soft real-time systems, not meeting the timing requirements does not lead to a failure but to degraded system performance that can be tolerated given that the timing miss rates are below a certain threshold. Classifying a system into hard or soft real-time depends usually on the overall system requirements and the environment where the system is deployed.

As examples of real-time embedded streaming systems, today we have increasing number of various autonomous mobile systems that need to interact and respond to their dynamic environment extremely fast. These include very complex systems such as self-driving cars and planes, but also modern “toys” such as drones. In recent years, drones have been used extensively as data collectors in many areas. For example, drones have been used in law enforcement for surveillance, tracking and rescue operations. They have been used for monitoring purposes in agriculture and farming, archaeological and land surveying, for delivery purposes in healthcare, crowd monitoring and control, and other cases of monitoring and control. They can carry various types of equipment including live-feed video cameras, infrared cameras, inertial, position and heat sensors. The large amount of data which should be collected and (pre-)processed, the battery-powered operation nature, and the need to react in a short time create demand for designing a high-performance energy-efficient real-time embedded streaming systems. In the next section, we discuss the current trends in designing such systems.

1.1 Trends in the Design of Embedded Streaming Systems

As introduced earlier, there is a demand in modern embedded streaming systems for high performance, in terms of high application throughput and short application latency, and a demand for real-time and energy-efficient execution. In addition, the complexity of applications running on embedded platforms increases [EJ09]. Therefore, we discuss below the trends in designing such complex embedded streaming systems to meet all the demands.

1.1.1 Platform Trend: Multi-Processor System-on-Chip (MPSoC)

Following the trend in general purpose systems, embedded streaming systems designers have relied for a long time on improvement of the computational power of uniprocessors to meet the high-performance requirements of streaming applications. The improvements of the computational power were driven by the increase in the clock frequency, advances in the semiconductor technology, that is, technology scaling, and innovations in the architecture (pipelining, out-of-order execution, branch prediction, and others.) [HP06]. However, the monotonically increasing performance curve with the successive generations of uniprocessors flattened in the early 2000s [PDG06]. The reasons for the curve flattening were increased dynamic power consumption and design complexity with the frequency increase and architecture innovations as well as increased static power and power density with the technology scaling [PDG06]. To increase system performance further, such that high-performance requirements of running applications are met, designers went for **multi-processor platforms** as the natural next evolutionary step in staying on the increasing performance curve [HP06]. By using multiple processors, the issue of increased power consumption is partially addressed by lower operating voltage and frequency, thereby decreasing the power consumption while maintaining high system performance through parallel execution. Moreover, nowadays, embedded systems designers integrate multiple processors, memories, interconnections, and peripherals into a **Multi-Processor System-on-Chip (MPSoC)** [JTW05].

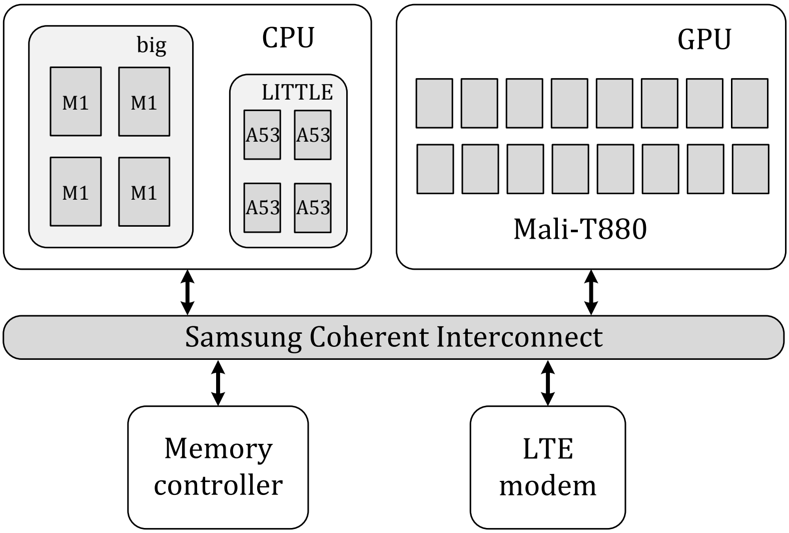

Figure 1.1: *An MPSoC platform example.*

Usually, an MPSoC contains different kinds of processors dedicated to certain functionalities: Central Processing Unit (CPU) for general purpose processing, Graphics Processing Unit (GPU) for graphical processing, a dedicated processor for wireless communication, and others. The processors communicate with each other through an on-chip communication infrastructure. To enable efficient communication, designers proposed and developed high-performance buses, such as ARM AMBA communication infrastructures [ARM], and Network-on-Chip (NoC) [BDM02] infrastructures, such as Xpipes [BB04] and Æthereal [GDR05]. Figure 1.1 gives an example of an MPSoC, the Exynos 8 Octa 8890 [Sama], which can be found in the Samsung S7 mobile phones. The Exynos 8 Octa has eight CPUs in a big.LITTLE architecture [Gre11]. That is, by integrating CPUs with different power-performance characteristics, namely, performance-efficient Exynos M1 cores (big cores) and energy-efficient Cortex A53 cores (LITTLE cores), this MPSoC provides more than 30% improvement in performance and 10% improvement in power efficiency compared to its predecessor [Sama]. The MPSoC also contains a 16-core GPU for 2D/3D graphical processing. The on-chip LTE modem is used for high-speed wireless data communication. All the processors are connected through a high-performance cache coherent interconnect. In this thesis, we consider such type of MPSoC platforms, efficiently utilizing their CPU part by mapping streaming applications on the CPUs.

1.1.2 Design Trend: Model-based Design Methodology

Driven by constant improvements in the semiconductor technology, MPSoC platforms integrate more and more processing elements on a chip. On the other hand, the complexity of embedded software also increases [EJ09]. In order to design such a complex embedded streaming system in an efficient manner in terms of system quality, design effort and time, designers had to raise the level of design abstraction from *Register-Transfer-Level (RTL)* to *system-level* [KMN$^+$00], [NSD08]. At the system-level, a hardware **platform** is modeled as a set of primitive blocks describing, at a high-level of abstraction, processing elements, memories and interconnects. An **application** is modeled as a set of tasks which can be allocated to hardware resources in many different ways, which means that there are many possible mappings of tasks to platform resources. Once it is determined how the application tasks are going to be allocated to the hardware resources such that all design *constraints* are met, that is, once we have a **mapping** specification, an Electronic System-Level (ESL) synthesis tool [GHP$^+$09] generates in an automated way the hardware description at a lower level of abstraction and the software for each processor in a platform.

In order to achieve the desired performance, the applications which are going to execute on the MPSoC platform have to be specified in a way which utilizes the *parallel* processing elements in the platform. In general, identifying parallelism in an application is a difficult step. In addition, designers should determine the mapping and execution order, that is, scheduling, of application tasks to a platform, and code should be generated for each used processor in the platform. In order to perform all these design steps in an efficient way, designers raise the level of abstraction, as introduced earlier, by building high-level models of applications. Then, the designer can use these models to analyze the performance of different applications-to-platform mappings. Such design approach is called **Model-based design** and the models used in such an approach are called **Models of Computation (MoCs)**. A MoC describes in a formal way how an application works. In this thesis, we consider only *parallel* MoCs because they are suitable for expressing parallelism in an application which is going to be executed on an MPSoC platform. In a parallel MoC, an application is decomposed into tasks which can be executed in parallel. The parallel MoC defines how tasks communicate and synchronize with each other.

Streaming applications have ample amount of parallelism which should be exploited efficiently to satisfy the performance requirements. Researchers have identified three types of parallelism:

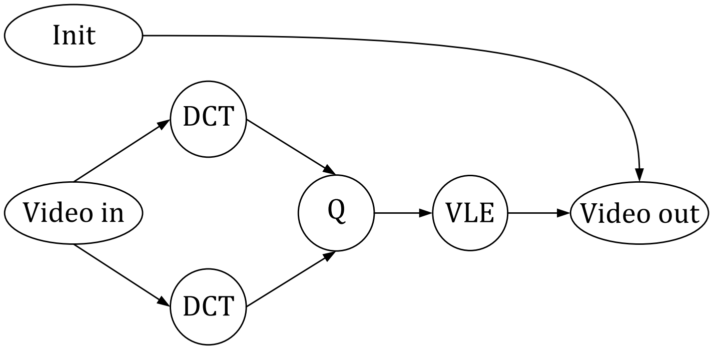

Figure 1.2: *Motion JPEG encoder application.*

1. Task-Level Parallelism (TLP): an application is split into set of tasks which can execute concurrently;
2. Data-Level Parallelism (DLP): a task of an application executes in parallel on multiple processing elements where each copy of the task processes its own data stream;
3. Pipeline-Level Parallelism (PLP): different iterations of a pair of data producer and consumer tasks execute in parallel.

Usually, an application contains more than one type of parallelism. Task-level parallelism is typically considered first when specifying an application as a set of concurrent tasks. Data-level parallelism is usually used by replicating tasks in an application in order to process more data in parallel and, hence, increase the application performance. However, if consecutive executions of a task depend on each other, data-level parallelism cannot be exploited and pipeline parallelism comes as an important form of parallelism to exploit. Figure 1.2 shows a Motion JPEG (MJPEG) encoder application represented as a set of communicating tasks which can execute concurrently. Here, we can identify examples of all three types of parallelism introduced above: 1) TLP between *Video in* and *Init*; 2) DLP between the two *DCT* tasks; and 3) PLP between different iterations of *VLE* and *Video out*.

It has been identified that *dataflow* MoCs are the most suitable parallel MoCs to express parallelism in streaming applications [TA10]. In a dataflow MoC, an application is represented as a *directed* graph, with graph nodes representing the application tasks and graph edges representing data dependences among the tasks. Thus, the parallelism is explicitly specified in the model. Dataflow MoCs differ among each other in their *expressiveness* and *decidability*. The expressiveness of a model indicates which type of applications can be modeled by the model and how compact the model is [SGTB11]. The decidability of a model represents the extent to which designers can analyze liveness and performance of an application at compile-time. In general, expressiveness and decidability of MoCs are inversely related, meaning that more expressive MoCs are less decidable, and the opposite; hence the choice of a suitable MoC depends on the problem being addressed. For example, within the Daedalus^{RT} [BZNS12] design methodology, the Cyclo-Static Dataflow MoC [BELP96] is used as an analysis model, to analyze design non-functional properties such as throughput, latency, hard real-time behavior, while the Polyhedral Process Network MoC [VNS07] is used as an implementation model. The MoCs considered in this thesis to represent streaming applications are **Synchronous Data Flow (SDF)** [LM87], **Cyclo-Static Dataflow (CSDF)** [BELP96] and **Polyhedral Process Network (PPN)** [VNS07], given in the order of increased expressiveness, hence decreased analyzability. Because of their very good analyzability, we use the SDF and CSDF MoCs to analyze the application throughput, latency, hard real-time behavior and calculate the required size of buffers used to implement inter-task communication. On the other hand, we use the PPN MoC to generate efficient code for processors in an MPSoC and build highly accurate energy model to analyze the energy consumption. A more detailed and complete comparison of different dataflow MoCs is given in [SGTB11].

1.2 Design Requirements and Basic Approaches to Meet the Requirements

In Section 1.1.2, it has been explained that model-based design methodologies have been used to design embedded streaming MPSoCs to provide the desired system performance. In this section, we introduce the requirements which are usually put on embedded streaming MPSoCs and the basic approaches proposed by research communities to meet these requirements.

1.2.1 Timing Requirements

As mentioned earlier, the performance of a streaming application running on an MPSoC is represented with two metrics: throughput and latency. Usually, embedded streaming MPSoCs execute simultaneously multiple applications and for each application throughput and latency requirements have to be met. In addition, these multiple applications should be **temporally isolated** between each other. This means that an application can be started or stopped at run-time without violating the timing requirements of other running applications. Beside performance requirements, many embedded streaming systems have to process data within a certain time interval, that is, before a *deadline*, meaning that they have hard real-time requirements.

In general, to provide timing guarantees for streaming applications, researchers proposed either analysis approaches on the dataflow MoCs, or they specified applications as periodic real-time tasks, or devised techniques which are mixture of the previous two. In the first case, techniques are devised to provide timing guarantees for streaming applications by performing analysis on a dataflow MoC, for example, techniques proposed in [GGS$^+$06], [SGB08], [MB07] and [BMKdD13]. The approach in [GGS$^+$06], [SGB08] analyzes an application modeled using the (C)SDF MoC [LM87] by performing state-space exploration of the (C)SDF graph in order to find the application throughput and latency. On the other hand, the approach in [MB07] converts an initial SDF application specification into an equivalent homogeneous SDF (HSDF) specification, and does the performance analysis on the HSDF. However, the state-space of an SDF graph is exponential in the worst case, and the conversion from an SDF into an equivalent HSDF results in an application graph which size grows exponentially in the worst case, hence the analysis approaches in [GGS$^+$06] and [MB07] have high time complexity. The approach in [BMKdD13] does application performance analysis on a CSDF graph by formulating the problem of finding performance guarantees as an Integer Linear Programming (ILP) problem. Thus, that approach has high time complexity given that ILP-based approaches suffer from severe scalability issues. All the approaches [GGS$^+$06], [MB07] and [BMKdD13], do not provide temporal isolation among applications and need complex design space exploration to find the minimum number of processors in a platform required to provide timing guarantees.

Another way of providing timing guarantees is by specifying applications as *classical real-time tasks* [DB11]. The classical real-time task model [LL73] specifies applications as *independent tasks*. The invocations of tasks are periodic, with constant execution time for each invocation and constant interval between invocations. By using the hard real-time schedulability theories [DB11], the minimum number of processors needed to schedule applications while providing timing guarantees, and temporal isolation between the applications can be determined in a fast analytical way. However, this classical real-time task model does not model data dependencies among tasks usually found in streaming applications.

Recently, several approaches, such as [BS11], [BS13] and [BTV12], have been proposed which combine advantages of the previously mentioned approaches by converting an application specified using a dataflow MoC, hence modeling data dependencies, to real-time tasks, thus enabling temporal isolation and fast calculation of the minimum number of processors to provide timing guarantees. Therefore, in this thesis, we utilize benefits of both dataflow MoCs and real-time task models to further improve the system timing guarantees and the utilization of hardware resources.

1.2.2 Energy Requirements

As indicated in Section 1.1.1, one of the main reasons for the flattening of the performance curve across different generations of uniprocessors was the increased power consumption due to the increased clock frequency to boost performance, and the technology scaling. The idea of using MPSoC platforms partially solved the power consumption issue by allowing performance boost through parallel execution while running processors at a lower frequency. Given that the technology scaling is still one ongoing process which provides more parallel processing resources but also results in larger power dissipation, it has been identified by the International Technology Roadmap for Semiconductors (ITRS) [fSI] that the power and energy consumption are the main problems in the system design. This results in a need for design techniques which target more performance and functionality at constant power density, constrained by thermal issues, and constant energy consumption, constrained by the battery capacity. The inability to manage power dissipation limits the amount of switched-on logic content in a SoC, known as the "dark silicon" issue [EBSA$^+$11].

Widely used techniques to reduce the power/energy consumption are Voltage-Frequency Scaling (VFS) and Power Management (PM). VFS reduces the power consumption by adjusting the voltage and operating frequency of processors while PM exploits idle times of processors by putting them to a very low-power sleep mode. In addition, according to ITRS reports, heterogeneous MPSoCs were identified as a promising solution in terms of energy-efficiency [Mit15]. Heterogeneous MPSoCs [Mit15] have been also considered as a promising solution to the dark silicon problem. Especially, the asymmetric multi-core architecture, also known as a single-ISA heterogeneous architecture, was recognized as a good trade-off in terms of energy-efficiency and programming effort [Mit15]. A single-ISA heterogeneous MPSoC consists of cores with different power-performance characteristics but with the same instruction-set architecture (ISA). Apart from containing cores with different power-performance characteristics, such heterogeneous MPSoCs cover large set of power-performance design points through voltage-frequency scaling of the cores [Mit15]. However, with the advent of many-core systems, per-core VFS becomes impractical due to the high hardware cost and area requirement [HM07]. Therefore, to balance the energy saving and the hardware cost, cores are grouped into clusters and cores in each cluster run at the same voltage and frequency level. In addition, it has been recognized by ITRS that the accuracy of power modeling and estimation has to be improved in order to manage the power consumption to extreme limits [Kah13].

1.3 Problem Statement

After introducing the trends and requirements in the design of embedded streaming systems in Section 1.1 and Section 1.2, in this section, we formulate the problems addressed in this thesis concerning the design of embedded streaming systems.

1.3.1 Problem 1

Meeting the timing requirements is one of the most important design objectives when designing embedded streaming MPSoCs. As explained in Section 1.2.1, there are several research approaches on how to guarantee the timing behavior of streaming applications. Among them, the most appropriate one is the research approach which combines the benefits of dataflow MoC-based analysis and hard real-time analysis. The existing works [BS11], [BS13], [BTV12], following this approach, assume that each execution of an application task takes the same amount of time. However, a common behavior in streaming applications is that different executions of the same application task differ in execution time. When such changing execution nature of an application is hidden by considering one and the same value for the execution time of an application task, the application throughput is underestimated, the application latency overestimated, while the processors in an MPSoC platform are underutilized. Thus, the first problem addressed in this thesis is:
Problem 1: Can we apply the hard real-time scheduling theory for real-time periodic tasks to streaming applications while considering different execution times among different executions of an application task to obtain tighter bounds on throughput and latency and better utilize processors?

1.3.2 Problem 2

As introduced in Section 1.1.2, streaming applications contain ample amount of parallelism and can be efficiently represented by using parallel MoCs. However, the initial parallel application specification often is not the most suitable one for a given MPSoC platform. This is because application developers mainly focus on realizing certain application behavior while the computational capacity and power consumption profile of the MPSoC platform is often not fully taken into account. That is, the initial parallel specification does not expose enough parallelism, particularly in the form of DLP, to better exploit the platform to satisfy timing and energy requirements. To better utilize the underlying MPSoC platform, the initial specification of an application, that is, the initial task graph, should be transformed to an alternative one that exposes more DLP while preserving the same application behavior. This can be achieved through an unfolding transformation where the tasks from the initial graph are replicated, in an equivalent graph, a certain number of times. Special care should be taken during the unfolding transformation to avoid all unnecessary overheads caused by data management among replicas. Moreover, having more tasks' replicas than necessary results in an inefficient system due to overheads in code and data memory, scheduling and inter-tasks communication. Thus, the right amount of DLP, depending on the underlying MPSoC platform, should be determined in a parallel application specification to achieve maximum performance and timing guarantees. Therefore, the second problem we address in this thesis consists of two sub-problems. The first sub-problem is:

Problem 2a: How to convert an initial application graph into input-output equivalent graph while avoiding unnecessary overheads caused by data management among task replicas?

The second sub-problem follows as:

Problem 2b: How many times to replicate each task in the initial application graph, such that the obtained equivalent graph exposes the right amount of parallelism that maximizes the utilization of the available processors in an MPSoC platform while meeting all timing requirements?

1.3.3 Problem 3

Apart from timing requirements, energy consumption requirements are very important requirements to be met for proper functioning of embedded systems. As introduced in Section 1.2.2, the ITRS proposed heterogeneous parallel processing and frequency islands as design innovations to address the power/energy consumption requirements. In particular, the asymmetric multi-core architecture was recognized by both academia and industry as a good platform for design of energy-efficient embedded systems. Some examples of commercial asymmetric cluster MPSoCs are Samsung Exynos 5 Octa

SoC [Samb], nVidia Tegra X1 [NVI15], which include ARM big.LITTLE [Gre11] integrating high-performance cores into big clusters and low-power cores into LITTLE clusters. As mentioned in Section 1.3.2, when developing an application, application designers often do not have the timing and energy behavior of a platform in mind. Hence, it may happen that an application consists of highly imbalanced tasks in terms of the task workload, that is, task utilization. Especially, in cluster heterogeneous MPSoCs, when several tasks are mapped onto the same cluster, the task with the heaviest utilization will determine the required voltage and frequency of the whole cluster and will significantly increase the energy consumption of the other tasks mapped on the same cluster. When task replication, that is, DLP, is applied to application tasks with heavy utilization, their utilization can be decreased while still providing the same application performance. Thus, the third problem, we address in this thesis is:
Problem 3: How to map embedded streaming applications under timing requirements by utilizing per-cluster VFS and task replication to reduce the energy consumption of a system?

1.3.4 Problem 4

It was pointed out by ITRS that the accuracy of power/energy modeling is very important for efficient power/energy management [Kah13]. Model accuracy is usually traded-off for modeling and evaluation effort. Energy models used with more analyzable functional models, that is, MoCs, are usually more abstract in order to be more efficient in terms of modeling and evaluation effort and time, hence they try to capture the worst-case energy consumption. Such a model of the energy consumption results in safe but not very accurate estimates when compared with the actual energy consumption measurements on real implementations. Hence, an energy model should be more closely related to the actual running system yet be enough efficient in terms of modeling and evaluation effort and time. More expressive MoCs, as the PPN MoC for example, can give better insight of the final implementation of application tasks on a platform, hence they are often used as implementation models. However, providing timing guarantees by doing analysis on the PPN MoC is rather difficult, if not impossible [Zha15], hence the PPN MoC is used in systems where the timing requirements are not necessarily specified but it is required that the system runs at the best of its capacity (best-effort systems). Therefore, as the forth problem, we investigate:
Problem 4: How to model as accurately as possible the energy consumption of a mapping of an application onto an MPSoC platform while such a system runs at the best of its capacity?

1.4 Research Contributions

By addressing the research problems outlined in Section 1.3, in this section, we summarize the research contributions of this thesis.

Contribution 1: Proposing a scheduling approach which converts dataflow MoCs to real-time periodic tasks while considering different execution times for different executions of an application task.
To address the first problem, namely, Problem 1 in Section 1.3.1, we propose a scheduling approach, published in [SLCS15] and [SLCS16], and presented in Chapter 3, to schedule streaming applications modeled as acyclic CSDF graphs on an MPSoC platform. The proposed approach converts each task in a CSDF graph to a set of real-time periodic tasks by deriving task parameters (periods, start times, and deadlines) while considering different execution times for different executions of each task in the CSDF graph. The conversion enables application of many hard real-time scheduling algorithms which offer fast calculation of the required number of processors for scheduling the tasks with a certain guaranteed throughput and latency. In addition, the proposed approach calculates the minimum buffer sizes of the communication channels between the tasks in the CSDF graph such that the converted tasks can be scheduled as periodic real-time tasks. As part of our scheduling approach, we propose a method to reduce the graph latency by carefully selecting the deadlines of the converted real-time periodic tasks. We show, on a set of real-life streaming applications, that our approach leads to equal or higher application throughput and shorter application latency while reducing the number of processors required to schedule a given application, compared to a related approach which does not consider different execution times for different executions of CSDF tasks. However, our approach results in increased memory requirements to implement the communication among the tasks.

Contribution 2: Proposing a graph transformation and an approach that uses the transformation to exploit the right amount of parallelism that maximizes the utilization of the available processors in an MPSoC platform while meeting all timing requirements.
We address the second problem, Problem 2a-2b stated in Section 1.3.2, by proposing an unfolding graph transformation for SDF graphs and an algorithm that adapts the exploited parallelism in an application modeled using the SDF MoC according to the resources in an MPSoC by using the unfolding transformation and the hard-real time scheduling approach of CSDF graphs devised within Contribution 1 such that the application performance is maximized and hard real-time behavior guaranteed. Our contribution has been published in [SLS16b] and explained in Chapter 4. In particular, our unfolding

graph transformation carefully distributes data among task replicas, enabling more parallel execution of tasks, and our algorithm determines simultaneously which SDF tasks and how many times to replicate them, and the allocation of tasks to processors in the MPSoC. We show, on a set of real-life streaming applications, that our unfolding graph transformation for SDF graphs results in graphs with the same application throughput, shorter application latency and smaller communication memory compared to related approaches. In addition, we show that our algorithm delivers, in 98% of the conducted experiments, a solution with a shorter latency, smaller communication memory and smaller values for task replication factors compared to a related approach while the same performance and timing requirements are satisfied.

Contribution 3: Proposing an approach that exploits the right amount of parallelism in an application and per-cluster VFS to map the application onto a cluster heterogeneous MPSoC such that the energy consumption is minimized and all timing requirements met.
In our third contribution which addresses Problem 3 in Section 1.3.3, we propose a novel algorithm [SLS16a], presented in Chapter 5, to efficiently map real-time streaming applications onto cluster heterogeneous MPSoCs, which are subject to throughput constraints, such that the energy consumption of the cluster heterogeneous MPSoC is reduced by using task replication and per-cluster VFS. By using the hard real-time scheduling approach of CSDF graphs, we devised within Contribution 1, we propose an efficcient way to determine a suitable processor type for each task in an (C)SDF graph such that the energy consumption is minimized and the throughput constraint is met. Then, by using our unfolding graph transformation, devised within Contribution 2, we propose a method to determine a replication factor for each task in an SDF graph such that the distribution of the workload on the same type of processors is balanced, which enables processors to run at a lower frequency, hence reducing the energy consumption. We show, on a set of real-life streaming applications, that our proposed energy minimization approach outperforms related approaches in terms of energy consumption while meeting the same throughput constraints.

Contribution 4: Proposing an accurate energy model for best-effort streaming applications mapped onto heterogeneous MPSoC platforms.
To address the problem of accurate power/energy modeling, namely, Problem 4 in Section 1.3.4, we devise an accurate energy model [SS13] for streaming applications modeled by the PPN MoC and mapped onto heterogeneous MPSoC platforms. The energy model is based on the well-defined properties of the PPN application model. To guarantee the accuracy of the energy model,

values of important model parameters are obtained by real measurements. In addition, our energy model can model different types of communication infrastructures: with and without contention. The accuracy of the proposed energy model is evaluated on FPGA-based MPSoC platforms running two real-life streaming applications against real measurements of the energy consumption from the FPGA. The model and its accuracy and efficiency is presented in Chapter 6.

1.5 Thesis Outline

Below we give an outline of this thesis, summarizing the contents of the following chapters.

Chapter 2 gives an overview of the dataflow MoCs considered in this thesis, and some techniques from hard-real time scheduling theories relevant for this thesis.

Chapters 3 to 6 contain the contributions of this thesis. Each chapter is organized in a self-contained way, meaning that each chapter contains more specific introduction to the problem addressed, related work, the proposed solution approach, experimental evaluation, and concluding discussion.

Chapter 3 presents our hard real-time scheduling approach for streaming applications modeled as acyclic CSDF graphs.

Chapter 4 describes our unfolding graph transformation for SDF graphs and our algorithm for finding proper replication factors for each task in an SDF graph, which uses our scheduling framework described in Chapter 3, such that the processing resources are utilized as best as possible, while providing hard real-time guarantees.

Chapter 5 presents our energy-minimization approach which uses our scheduling framework described in Chapter 3 and our unfolding transformation described in Chapter 4 to find task-to-processor type assignment and task replication factors such that the energy consumption is minimized.

Chapter 6 presents our accurate energy model for streaming applications modeled using the PPN MoC and mapped onto MPSoC platforms.

Chapter 7 ends this thesis by providing conclusions regarding the work done within this thesis and discussions on potential future work.