Performance Estimation of Streaming Applications for Hierarchical MPSoCs

Martin Flasskamp[†], Gregor Sievers[†], Johannes Ax[†], Christian Klarhorst[†], Thorsten Jungeblut[†], Wayne Kelly[*], Michael Thies[†], and Mario Porrmann[†]

[†]Cognitronics and Sensor Systems Group, CITEC, Bielefeld University, Bielefeld, Germany
mflasskamp@cit-ec.uni-bielefeld.de

[*]Science and Engineering Faculty Queensland University of Technology Brisbane, Australia
w.kelly@qut.edu.au

ABSTRACT

Parallel programming and effective partitioning of applications for embedded many-core architectures requires optimization algorithms. However, these algorithms have to quickly evaluate thousands of different partitions. We present a fast performance estimator embedded in a parallelizing compiler for streaming applications. The estimator combines a single execution-based simulation and an analytic approach. Experimental results demonstrate that the estimator has a mean error of 2.6% and computes its estimation 2848 times faster compared to a cycle accurate simulator.

1. INTRODUCTION

Multiprocessor System-on-Chips (MPSoC) comprised of dozens or hundreds of CPU cores are prevalently used in current embedded systems due to their increased energy efficiency compared to single core architectures. The efficient mapping of tasks to MPSoCs that integrate many processing elements has a high computational complexity. Some frameworks [6, 15, 13] use optimization algorithms like simulated annealing or evolutionary algorithms to explore a large solution space. A typical optimization goal is to maximize the throughput of an application.

An important and time-consuming step of these algorithms is the accurate evaluation of hundreds to ten thousands of different partitions of the application. The effectiveness of this optimization is based on two conflicting goals: Firstly, the *faster* the evaluation of a single partition can be done the more different partitions can be analyzed. Accordingly, the chance is higher to derive a mapping with optimal performance from the search space of possible partitions. Secondly, the more *accurate* a partition is evaluated the better is the compiler's chance to correctly select between solutions. Thus, the challenge is to balance *computation time* against *accuracy* and find that trade-off that leads to the best overall result independent of the optimization algorithm.

Performance evaluation can be performed on various levels of abstraction starting from analytical models, over abstract ISA-simulation, up to complex but very precise RTL-simulation or real FPGA-prototypes. In Fig. 1 the relation between speed and accuracy is presented from very fast but abstract simulations using native x86 execution to most accurate micro-architectures (RTL-simulators, FPGA/ASIC-prototypes).

The main contribution of this work is a fast and accurate simulation-based estimation (SBE) framework. In addition, a more abstract analytical modeling for performance estimation is discussed. This analytical model can be used as an optimistic upper bound for the achievable performance of an application.

For the evaluation of the speed and accuracy of our estimation we use our hierarchical CoreVA-MPSoC-architecture [12] as a target hardware platform described in Section 3.1. The architecture consists of VLIW CPUs that are tightly coupled within CPU clusters via a bus-based interconnect. Several of these clusters are connected using a Network-on-chip (NoC) to allow for many-cores with hundreds of CPUs. To map applications to a huge number of CPUs, we utilize a compiler framework for streaming applications [6], c.f. Section 3.2. The compiler implements a partitioning algorithm based on simulated annealing to derive a good mapping of the application to the MPSoC. This mapping is static because streaming applications process a continuous data stream by repetitive computation.

The estimation framework is designed to be applicable to arbitrary MPSoC architectures.

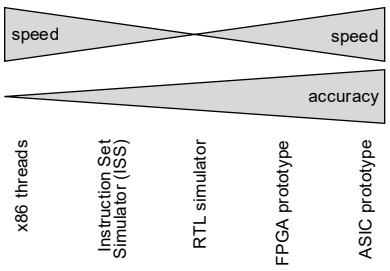

Figure 1: Estimation and simulation methods: Trade-off between accuracy and speed

RAPIDO '16 January 18 2016, Prague, Czech Republic

ISBN 978-1-4503-4072-4/16/01...$15.00

DOI: http://dx.doi.org/10.1145/2852339.2852342

2. RELATED WORK

There are plenty of methodologies to profile software on embedded systems. A comprehensive overview from Patel and Rajawat with a focus on performance estimation can be found in [9]. According to their taxonomy our SBE framework can be classified as a combination of an analytical methodology and a simulation-based approach. At the end of Section 5 we compare accuracy and simulation speed of our approaches with related work.

There are a number of works regarding the early system level design space exploration for embedded systems. The sesame framework [10] is an example for a high level hardware/software estimation environment. An analytical model is used to select candidate architectures. These architectures are than simulated on a system level and refined. Our work focus on the performance estimation for a given MPSoC architecture.

Cheung *et al.* [4] present a performance simulation framework for MPSoC. They generate structural performance models with GCC in less than 0.1 s. An estimation error of less than 1% compared to ISS is achieved at a speed close to untimed behavioral simulation. The approach was tested on four different configurations of the Xtensa LX2 processor architecture.

An analytic approach based on neural networks is proposed by Oyamada *et al.* [8]. During the MPSoC design process they aim at a rapid selection of a suitable processor to run a given application. The error of the analytic estimation is up to 17% compared to the cycle accurate simulation. The estimation time of 600 s for a cycle accurate simulation can be reduced to 17 s.

Huang *et al.* [5] utilize GNU gcov to profile execution statistics of given C code during native simulation on x86 CPUs. Afterwards, the gathered information is used to annotate a transaction accurate simulation. For an example application the estimation time can be reduced to 15 s compared to 2656 s required by a cycle accurate virtual prototype.

Benoit *et al.* [3] are targeting performance prediction of data-parallel applications for automatic parallelization. They achieve an estimation error of 1.59% compared to measured results. Their approach of estimating a task's execution costs by an execution without any partition is comparable to the strategy we utilize to estimate a task in our analytic modeling and SBE model (cf. Section 4).

3. THE COREVA-MPSOC PROJECT

As a target platform for our performance estimation environment presented in this work, we use a self-developed multiprocessor. The CoreVA-MPSoC is used for embedded application domains like software defined radio, vision processing, and robotics.

3.1 Hardware Architecture

The hardware architecture of our CoreVA-MPSoC is presented in Fig. 2. Basic building block of the MPSoC is the VLIW CPU CoreVA that is designed to provide high resource efficiency [7]. According to the application domain, the number of VLIW slots, ALUs, multiplication, and division units can be configured at design time. The CPU features software managed data and instruction scratchpad memories and six pipeline stages.

A CPU cluster is used to tightly couple up to 32 of those CPU cores using a bus-based interconnect. The interconnect can be configured at design time and supports either AXI or Wishbone standard, each featuring a crossbar or a shared bus. Within a cluster, a CPU can access the local data memory of every other CPU in a Non-Uniform Memory Access (NUMA) fashion. A FIFO is attached to each CPU to decouple write operations to the bus so the CPU is not stalled on bus congestions.

To allow for MPSoCs with several hundred CPU cores, a Network on Chip (NoC) can be used as an additional level of communication hierarchy to connect several CPU clusters. The NoC is built up of routers, implementing a packet-switched wormhole routing. Packets are segmented into flits with 64 bit payload data. The NoC allows for the usage of different network topologies. In this work we use a 2D-mesh topology. One port of each router is connected to a CPU cluster via a network interface (NI) [2]. The NI acts like a DMA controller by sending and receiving packets concurrently with CPU processing. Packet data is directly stored to and read from each CPU's local data memory. The NI supports a synchronized communication via a certain number of independent uni-directional channels.

A single issue VLIW CPU occupies an area of 0.126 mm^2 using a 28 nm FD-SOI standard cell technology including 16 kB instruction and 16 kB data memory (post place&route) [12]. A hierarchical MPSoC with 4 CPU clusters, 4 CPUs in each cluster, and 512 kB total memory requires 3.02 mm^2. The maximum clock frequency of the MPSoC is 830 MHz. A more detailed description of the hardware architecture of our MPSoC can be found in [12].

To program the MPSoC, the inter-CPU communication is encapsulated by our communication library, transparently for both cluster and NoC communication. The CPUs communicate via a block-based synchronization model and uni-directional communication channels [6]. The channels can use multi buffering to hide communication latencies.

3.2 Compiler Infrastructure

We have created a C compiler tool chain based on the LLVM compiler infrastructure. Our custom backend supports VLIW and SIMD vectorization. Nevertheless, efficiently programming a complete MPSoC with many CPUs is a challenge for the programmer. Therefore, we developed a compiler for streaming applications [6] to aid the programmer in effective usage of the MPSoC's resources. These applications need to be written in the StreamIt language developed at MIT [14]. An application is represented by a structured data flow graph, which describes the inherent parallelism of its tasks. Our compiler for streaming applications searches for a valid partition with maximum throughput. It utilizes an

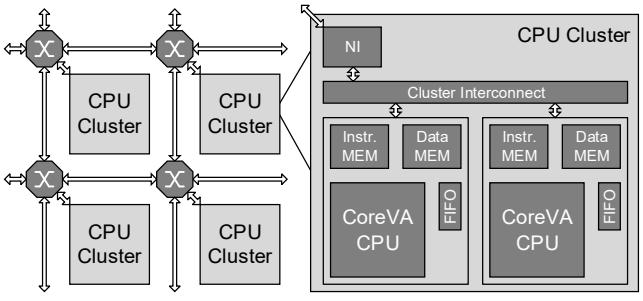

Figure 2: Hierarchical CoreVA-MPSoC architecture

approach based on simulated annealing to map the tasks of a program onto the individual CPUs of the MPSoC. During the partitioning process the compiler exploits three degrees of freedom to alter an application's data flow graph. Firstly, the compiler decides on which processor a task is placed. Secondly, a task can be cloned to exploit data parallelism. Thirdly, the granularity of work done in each iteration can be increased to reduce the overhead of communication. These changes are called mutations and lead to a huge search space for the partitioning algorithm.

Every partition is judged regarding the achieved performance but is also checked if any hardware limits are exceeded. These are upper limits for, e.g., memory consumption per CPU or the packet size in the NoC. Partitions that violate these restrictions are marked as invalid. Finally, C code is generated for all CPUs according to the final partition. For each task the computational part and the synchronization with other tasks are separated.

4. ESTIMATION AND SIMULATION METHODOLOGIES

An MPSoC can be simulated at different levels of abstraction at different levels of accuracy and speed. The most abstract level is reached if the MPSoC is reduced to a number of CPUs without considering the communication infrastructure (cf. Figure 3a). As a result, moving data from one CPU to another does not have any costs at all. This kind of model is used in our analytic modeling.

By adding topology details to the model, the communication channel between two arbitrary CPUs can be estimated by heuristics (cf. Figure 3b). Hereby, the different communication channels (e.g., within a CPU cluster or in between CPU clusters) can be distinguished regarding their latency behavior or capacity. This level of abstraction is utilized by our simulation based estimation (SBE) model presented in Subsection 4.1. The model is based on an XML schema defined as a formal description of the MPSoC architecture. For example, a CPU is described by the number and configuration of functional units and its local memory. The hierarchical interconnect is specified by, e.g., its topology, communication bandwidth, and latency.

The computational costs to execute a task are the same on each processor of the same type independent from the partition or MPSoC topology. Thus, we predetermine the runtime of all tasks of an application by measuring their execution time on a single CPU core with a cycle accurate simulator. This has to be done only once and the results are used by both analytic modeling and SBE model.

A higher estimation accuracy can be achieved if the simulation considers the hardware architecture of the MPSoC (cf. Figure 3c). We present a cycle accurate simulation (CAS) by an instruction set simulator (ISS) and RTL simulator as well as FPGA and ASIC prototypes in Subsection 4.2. The usage of simulation results can lead to a higher estimation accuracy but requires the application to be executed. However, the execution on a platform other than the target MPSoC (e.g., an x86 PC) can be done at an early design stage. Therefore, a simulation environment based on x86 threads can be utilized to simulate the functional behavior of an application during its development. The timing behavior can be quite inaccurate depending on the architectural difference to the target MPSoC.

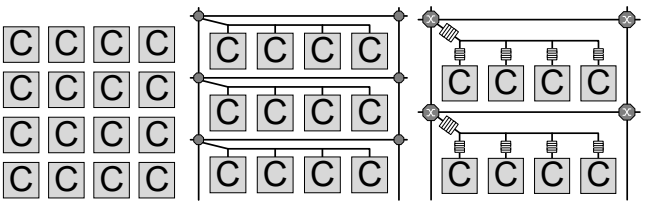

Figure 3: Level of abstraction for analytic modeling (AM), simulation based estimation (SBE), and cycle accurate simulation (CAS) of a hierarchical MPSoC.

	speed	accuracy	availability
AM	+++	−−	+++
x86 threads	++	−	++
SBE	+++	+	+
ISS (CAS)	−	++	−
RTL sim.	−−	++	−−
FPGA	++	++	−−
ASIC	+++	+++	−−−

Table 1: Comparison of speed, accuracy, and availability for x86 threads, analytic modeling (AM), simulation based estimation (SBE), ISS, RTL simulator, FPGA, and ASIC

In Table 1 a comparison of speed, accuracy, and availability of the mentioned performance evaluation methods is presented. Although, the most accurate and fastest execution-based performance ascertainment can be derived from an existing chip prototype (ASIC), it is typically not available due to an ongoing design process or high production costs. Besides, it lacks the flexibility of evaluating different hardware configurations. Our analytic modeling and SBE model are both fast as depicted in Table 1. Furthermore, the analytic modeling has the best availability but a lower accuracy, whereas the SBE model achieves a reasonable accuracy and slightly worse availability. A further discussion is provided in Section 5. Before describing the implementation details of our SBE in Subsection 4.1, we will first present our analytic modeling approach.

The amount of parallelism that an application offers limits the scaling on multiprocessor architectures. Amdahl's law divides an application in a parallel portion $f \in [0, 1]$ and a sequential portion $1 - f$. The parallel portion can be accelerated by a factor of m if it is executed on m CPUs. The sequential part of the application limits the scaling. Amdahl's law does not consider any overhead of splitting an application's tasks to several CPUs. Because of this, it is an optimistic limit for the achievable speedup for a given number of CPUs. Parallel applications contain different types of parallelism. A task features data parallelism if one call of a task is independent from the next call. Thus, each call of a task can be executed on another CPU. This type of parallelism is also called scalable parallelism. Functional parallelism (task and pipeline parallelism) describes a finite number of atomic tasks that can be executed in parallel. Amdahl's law assumes scalable parallelism, because tasks have to be splittable at arbitrary ratios. If an application

features data parallelism only ($f = 1$), a linear scaling is possible and the speedup S is equal to the number of CPUs m if communication is not considered.

For our analytic modeling we assume an application with sequential runtime w that has $f' \in [0, 1]$ data parallelism and $1 - f'$ functional parallelism. In case of streaming applications, w is the runtime of a single repetition of the periodic computation. We use a partitioning algorithm (e.g., brute force or greedy bin packing [6]) to partition all atomic tasks of the functional-parallel part of the application to the available m CPUs. There are then one or more CPUs that limit the application's speedup because the execution of their tasks result in the longest runtime w_{max}. All other CPUs do not operate at full utilization. The sum of all of these spare cycles over all CPUs is $w_{available}$. If the runtime of the data-parallel part of the application $w \cdot f'$ is at least as large as $w_{available}$, the data-parallel part can be partitioned to all CPUs so that all CPUs have the same workload. This results in linear speedup. Combining these two cases, the resulting speedup S_{AM} for our analytic modeling is:

$$S_{AM} = \begin{cases} \frac{w}{w_{max}} & \text{if } w \cdot f' < w_{available} \\ m & \text{if } w \cdot f' \geqslant w_{available} \end{cases} \quad (1)$$

It is important to note that this analytic modeling does not consider communication. Therefore, two partitions of an application for a particular MPSoC configuration cannot be distinguished. From this follows that analytic modeling cannot be used by optimization algorithms like our partitioning algorithm presented in Section 3.2. Our more accurate SBE model addresses this limitation and is presented in the next subsection. However, analytic modeling can provide information about the application's inherent parallelism at a very early design stage based on the pure computational costs of the application. In addition, analytic modeling can be used by developers as an optimistic estimation for the theoretical performance of an application.

4.1 Simulation Based Estimation (SBE)

Finding an optimal partition of an application for a particular MPSoC configuration is an optimization problem. Our SBE approach can be integrated into partitioning algorithms by providing accurate performance estimations. The traversed partitions don't have to be simulated individually because their estimation is based on preceding simulations of the individual tasks. MPSoC configurations with the same number of CPUs can differ in many ways like topology, communication infrastructure and memory layout. All these characteristics must be modeled for accurate estimations. While processing, an optimization algorithm traverses a huge search space of different partitions (cf. Section 3). Every partition has to be judged against the optimization goal of maximum throughput. The performance of the application is limited by a bottleneck, which can be the CPU or the communication link with the highest load. A CPU's work consists of the computational work and synchronization costs of the executed tasks as well as additional cycles if the communication infrastructure is overloaded.

The computational work can be calculated for each CPU by summing up the predetermined runtime of all tasks partitioned onto this CPU. We made the assumption that a task's runtime is independent of the CPU's position in the MPSoC. This requires that the CPUs have the same properties like, e.g., frequency and number of VLIW slots (cf. Section 3.1). Accordingly, the runtime of all tasks can be determined in advance for a specific type of CPU by measuring the execution on a single CPU.

Besides the placement of a task the communication between two tasks can influence the performance as well. Depending on the used communication infrastructure, there are additional software costs (in terms of CPU cycles) for synchronization and handling of communication channels in software. These costs differ if a channel connects two tasks on the same CPU or across a CPU cluster or the NoC. This overhead is computed based on heuristics from the XML description of the target MPSoC, including the topology, communication bandwidth, and latency of the MPSoC as well as software costs for CPU-to-CPU synchronization. The software costs can automatically be determined in advance. Our compiler infrastructure (cf. Section 3.2) generates micro benchmarks for typical communication patterns. The measurement is performed by executing the micro benchmarks on our cycle accurate simulator.

Multiple tasks communicating concurrently on a communication link (e.g., a NoC link or a bus interconnect) may lead to a load that is exceeding the link's bandwidth. This may cause a stall of the sending or receiving CPU and therefore results in additional cycles. Our SBE models the communication infrastructure of the MPSoC to determine the load of all network links for a certain partition. The load of one network link is the sum of all communication channels that communicate via this link.

The available data memory is a critical resource in embedded multiprocessor systems since its size is typically limited due to cost constraints. However, increasing the block size improves the application's throughput by reducing the impact of synchronization costs. Our SBE model observes the data memory size and enables the partitioning algorithm to reject partitions exceeding that constraint.

The load of all CPUs and communication links is estimated individually as described above. The combination enables our compiler for streaming applications to determine a fast but still accurate performance estimation of an application's partitions. Changing the placement of tasks or the MPSoC configuration requires only a simple reevaluation of the SBE model without execution/simulation of the application. Our partitioning algorithm utilizes this estimation for balancing the load due to computation and communication across the MPSoC. Implementation details about SBE and our partitioning algorithm can be found in the technical report [1].

4.2 Cycle Accurate Simulation (CAS)

An accurate simulator of embedded multiprocessor platforms is required to aid both software and hardware development process. The instruction set simulator (ISS) considered in this work is written in C. Every level of hierarchy (CPU, Cluster, NoC) in the simulator can be configured to easily perform a holistic design-space exploration of different topologies.

The accuracy of our ISS needs to be close to the actual hardware of the MPSoC, which is described in a Hardware Description Language (HDL). For development and verification of the HDL description an RTL simulation environment is used. In addition, we perform gatelevel simulations (post synthesis and post place&route) to determine the energy consumption of the MPSoC. The accuracy of our RTL and gatelevel simulation flow has been verified by measurements

of two ASIC prototypes [7]. Our ISS shows a simulation error of less than 1% compared to RTL simulator. In addition to the just mentioned simulations, our self-built FPGA-based rapid prototyping environment RAPTOR [11] is available for functional verification. This prototyping environment is also used as a testbed for ASIC prototypes of the CoreVA-MPSoC.

5. RESULTS

In this section we evaluate our performance estimation approach using the CoreVA-MPSoC architecture and its compiler infrastructure. Our benchmark suite contains 10 streaming applications derived from the StreamIt benchmark suite [14]. We consider 10 MPSoC configurations with 2 to 16 CPUs (e.g., 2x2x4 is a 2x2 Network on Chip with 4 CPUs within each of the 4 CPU clusters).

The speed of the different estimation and simulation variants is compared by measuring the time to produce 100 output items using the application FilterBank (cf. Figure 4). Our evaluation system is running Ubuntu Linux 12.04 on an Intel Xeon E5-1650 processor at 3.5 GHz and has 128 GB RAM. The final partition produced by the partitioning algorithm is executed for 100 iterations by the simulation based on x86 threads and ISS as well as 30 times by the RTL simulator. For FPGA and ASIC prototypes the runtime is calculated based on the cycle accurate RTL simulation and target frequencies of 50 MHz for the FPGA prototype and 830 MHz for the ASIC prototype. The RTL simulation shows the longest runtime with 164 s for one CPU to 812 s for an MPSoC with 128 CPUs. Compared to RTL simulation, the ISS is three order of magnitudes faster with 0.35 s to 1.00 s. Increasing the number of CPU cores results in a reduction of simulated MPSoC cycles which can be seen for the FPGA and ASIC prototypes. The ISS execution time is mainly affected by the number of simulated CPUs and the total number of simulated MPSoC cycles. Our SBE model shows the fastest execution time, because it only requires solving the MPSoC model to derive communication costs. The speedup of SBE compared to the ISS simulation ranges from 1224 for an MPSoC configuration with 16 CPUs to 2848 for an MPSoC with 128 CPUs. For x86 threads the runtime increases rapidly if there are more simulation threads (i.e., simulated MPSoC cores) than available physical CPU cores of the simulation host. Compared to our SBE model, x86 threads have a higher runtime and are thus used for functional verification and debugging in early design stages only. The analytic modeling is not shown in Figure 4 because it consists of a simple partitioning step of functional-parallel tasks of the application followed by solving Equation 1. The average estimation error of all applications and MPSoC configurations is 26.4% for the analytic modeling compared to the cycle accurate ISS. However, as described in Section 4, the analytic modeling is not suitable for partitioning algorithms and is not considered further.

Figure 5 shows the absolute error of our SBE model compared to the cycle accurate ISS. For this comparison all applications of our benchmark suite are used with a 2x2x4 MPSoC configuration. FFT, FilterBank, and MovingAverage show the lowest error with less than 0.1%. The applications BatcherSort and BitonicSort show a higher error of 9.8% and 6.7% due to inaccuracies of the estimation of the communication primitives. The average error considering all 10 applications is 2.6%.

For the evaluation of different MPSoC configurations, Figure 6 shows the absolute estimation error for SBE model, the application DES and 10 different MPSoC configurations with 2 to 16 CPUs. We consider different topologies from a single CPU cluster to multiple clusters (e.g., 2x1x8). There are 4 MPSoC configurations with 16 CPUs (1x1x16, 2x1x8, 2x2x4, 4x4x1) and 4 configurations with 8 CPUs (1x1x8, 2x1x4, 2x2x2, 4x2x1). For the DES application the average estimation error is 2.3%. The configurations with two CPU clusters and one NoC link show the highest error of 3.8% (2x1x4) and 4.8% (2x1x8). Nevertheless, considering all applications and MPSoC configurations, the estimation error is not depending on the MPSoC topology or CPU count. The SBE model is aware of the MPSoC's topology and is therefore suitable for accurate estimations. Due to different experimental setups and measurement methodologies, a fair comparison of our SBE model with other related work is not easy. Cheung *et*

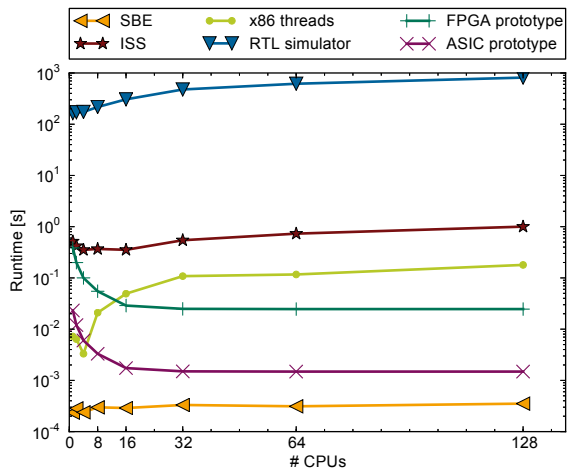

Figure 4: Execution time to produce 100 output items of FilterBank application for MPSoC configurations with 1 to 128 CPUs

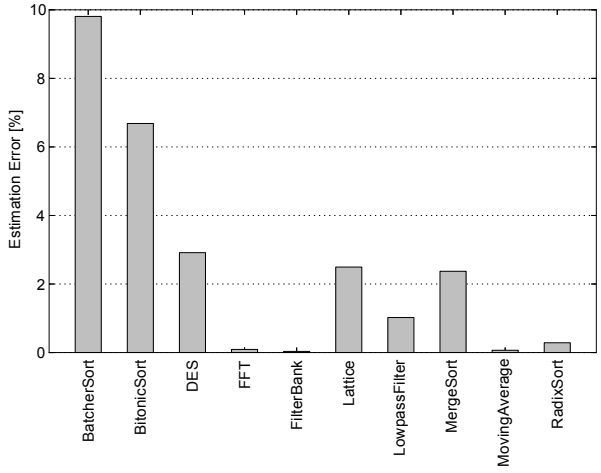

Figure 5: Estimation error of SBE model and different applications for 2x2x4 MPSoC configuration compared to ISS

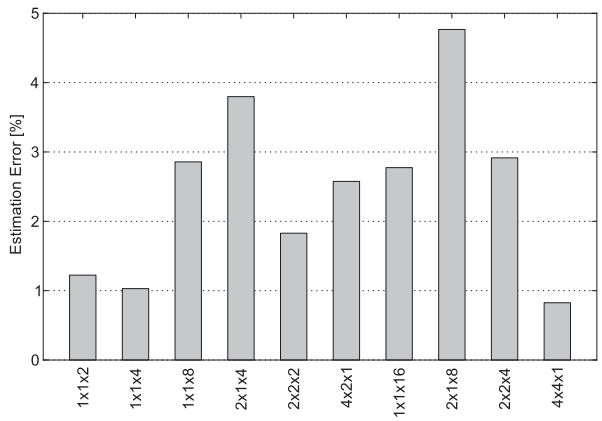

Figure 6: Estimation error of SBE model and DES application for different MPSoC configurations (*NoC dimensions* x *#CPU per cluster*) compared to ISS

al. [4] and Benoit *et al.* [3] show an estimation error of less than 2%. Oyamada *et al.* [8] present a much higher error of 17% and show a speedup of 35 compared to cycle accurate simulation. Huang *et al.* [5] show a speedup of 177.

Considering all applications and MPSoC configurations results in 100 different combinations. For these the average estimation error for our SBE model is 2.6%. The SBE model achieves a speedup in execution time of up to 2848 compared to the ISS simulation for an MPSoC containing 128 CPUs.

The high estimation accuracy and fast runtime of SBE allows our partitioning algorithm to explore a huge design space of streaming applications. Our approach does not require any simulation when integrated in an optimization algorithm since all simulation steps are independent from the MPSoC configuration and the placement of tasks.

6. CONCLUSION

In this work, we have presented a performance estimation methodology for embedded hierarchical MPSoCs. 10 applications have been considered with 10 different MPSoC configurations resulting in 100 combinations. The partitioning of applications to these MPSoCs requires fast, but accurate performance estimation. Analytical modeling can provide information about the application's inherent parallelism at a very early design stage based on the pure computational costs of the application. In this work we present a simulation based estimation (SBE) model that combines a single execution-based simulation (by measuring the runtime of individual tasks once, independent of MPSoC configuration and task placement) and an analytic approach (by modeling the communication infrastructure and communication software overheads). Compared to related work, SBE shows a high accuracy and low execution runtime. The SBE model shows a speedup of 2848 and an average estimation error of 2.6% compared to cycle accurate ISS simulation. Therefore, SBE is suitable to be utilized in our parallelizing compiler for hierarchical MPSoCs. In addition, the SBE model can be applied for the design-space exploration to find an optimal MPSoC hardware configuration for a certain application. In the future we will extend the model by estimating latency, required instruction memory, and energy consumption.

Acknowledgments

This work was funded as part of the DFG Cluster of Excellence Cognitive Interaction Technology 'CITEC' (EXC 277), Bielefeld University and the BMBF Leading-Edge Cluster "Intelligent Technical Systems OstWestfalenLippe" (it's OWL), managed by the Project Management Agency Karlsruhe (PTKA).

7. REFERENCES

[1] J. Ax et al. An Abstract Model for Performance Estimation of the Embedded Multiprocessor CoreVA-MPSoC (v1.0). Technical report, Bielefeld University, 2015. DOI: 10.13140/RG.2.1.1090.2483.

[2] J. Ax et al. System-level analysis of network interfaces for hierarchical mpsocs. In *NoCArc*. ACM, 2015. In press.

[3] N. Benoit and S. Louise. A First Step to Performance Prediction for Heterogeneous Processing on Manycores. *Procedia Computer Science*, 51:2952–2956, 2015.

[4] E. Cheung et al. Fast and accurate performance simulation of embedded software for MPSoC. In *Asia and South Pacific Design Automation Conf. (ASP-DAC)*, pages 552–557. IEEE, 2009.

[5] K. Huang et al. Profiling and Annotation Combined Method for Multimedia Application Specific MPSoC Performance Estimation. *Frontiers of Information Technology & Electronic Engineering*, 16:135–151, 2015.

[6] W. Kelly et al. A Communication Model and Partitioning Algorithm for Streaming Applications for an Embedded MPSoC. In *Int. Symp. on System on Chip (SoC)*. IEEE, 2014.

[7] S. Lütkemeier et al. A 65 nm 32 b Subthreshold Processor With 9T Multi-Vt SRAM and Adaptive Supply Voltage Control. *IEEE J. Solid-State Circuits*, 48(1):8–19, 2013.

[8] M. Oyamada et al. Software Performance Estimation in MPSoC Design. In *Asia and S. Pacific Design Autom. Conf. (ASP-DAC)*, pages 38–43. IEEE, 2007.

[9] R. Patel and A. Rajawat. Recent Trends in Embedded System Software Performance Estimation. *Design Automation for Emb. Systems*, 17(1):193–213, 2014.

[10] A. Pimentel et al. A Systematic Approach to Exploring Embedded System Architectures at Multiple Abstraction Levels. *IEEE Trans. on Computers*, 55(2):99–112, 2006.

[11] M. Porrmann et al. RAPTOR–A Scalable Platform for Rapid Prototyping and FPGA-based Cluster Computing. In *PARCO'09*, pages 592–599, 2009.

[12] G. Sievers et al. Evaluation of Interconnect Fabrics for an Embedded MPSoC in 28nm FD-SOI. In *ISCAS*. IEEE, 2015.

[13] L. Thiele et al. Mapping Applications to Tiled Multiprocessor Embedded Systems. In *Int. Conf. on Application of Concurrency to System Design (ACSD)*, pages 29–40. IEEE, 2007.

[14] W. Thies et al. StreamIt: A Language for Streaming Applications. In *Int. Conf. on Compiler Construction (CC)*, pages 179–196. Springer, 2002.

[15] Z. Wang et al. Partitioning Streaming Parallelism for Multi-Cores. In *Int. Conf. on Parallel Architectures and Compilation Techniques (PACT)*, page 307. ACM, 2010.