Why Comparing System-level MPSoC Mapping Approaches is Difficult: a Case Study

Andres Goens, Robert Khasanov, Jeronimo Castrillon
Center for Advancing Electronics Dresden (cfaed),
Chair for Compiler Construction, TU Dresden
Dresden, Germany
Email: {first.last}@tu-dresden.de

Simon Polstra, Andy Pimentel
Informatics Institute,
University of Amsterdam
Amsterdam, Netherlands
Email: {s.polstra,a.d.pimentel}@uva.nl

***Abstract*—Software abstractions are crucial to effectively program heterogeneous Multi-Processor Systems on Chip (MPSoCs). Prime examples of such abstractions are Kahn Process Networks (KPNs) and execution traces. When modeling computation as a KPN, one of the key challenges is to obtain a good *mapping*, i.e., an assignment of logical computation and communication to physical resources. In this paper we compare two system-level frameworks for solving the mapping problem: Sesame and MAPS. These frameworks, while superficially similar, embody different approaches. Sesame, motivated by modeling and design-space exploration, uses evolutionary algorithms for mapping. MAPS, being a compiler framework, uses simple and fast heuristics instead. In this work we highlight the value of common abstractions, such as KPNs and traces, as a vehicle to enable comparisons between large independent frameworks. These types of comparisons are fundamental for advancing research in the area. At the same time, we illustrate how the lack of formalized models at the hardware level are an obstacle to achieving fair comparisons. Additionally, using a set of applications from the embedded systems domain, we observe that genetic algorithms tend to outperform heuristics by a factor between $1\times$ and $5\times$, with notable exceptions. This performance comes at the cost of a longer computation time, between 0 and 2 orders of magnitude in our experiments.**

I. Introduction

Several developments in the last decade have firmly established this as the multicore era. One of the main repercussions of this is that abstracting at the level of an Instruction-Set Architecture (ISA) is no longer sufficient to improve application performance. Instead, today's chips feature multiple different processing elements that are exposed individually to the software developers. This is true, for example, for the high-performance Tilera many-cores [26] (now commercialized by Mellanox Technologies) or the NoC-based homogeneous many-cores by Adapteva [15]. Not only the size of architectures keeps increasing, but also their heterogeneity. Especially in the embedded-systems domain, multi-processor systems-on-chip (MPSoC) like Texas Instruments' Keystone II [2] or the big.LITTLE™Architecture in Samsungs' Exynos [9], feature programmable cores of different types (e.g. ARM or DSP) and/or specialized hardware accelerators.

Within the gargantuan endeavor of efficiently programming MPSoCs, a central challenge is the *mapping* problem. The mapping problem refers to the task of deciding where in hardware to place the different parts of a software application. This includes mapping both computation to the different cores, as well as communication to the memory and interconnect subsystem. The difficulty lies in that fact that the number of possible mappings grows exponentially as applications and architectures become larger, rendering exhaustive approaches intractable in practice. This is why different solutions for finding near-optimal mappings have been proposed [22], [31], [21], [11], [19], [4], [16], [20]. However, the variety of programming models, target architectures and simulation environments makes it extremely difficult to make comparison across different frameworks. This represents an obstacle to assess the effectiveness of individual solutions which is fundamental for advancing research.

In this paper we analyze and compare two such frameworks, namely Sesame [11] and MAPS [19]. Sesame is an environment which allows researchers to model and simulate applications executing in MPSoCs at the system-level. Using this simulation core we evaluate a framework for design space exploration which uses evolutionary algorithms to find near-optimal mappings. MAPS, on the other hand, is a compiler framework that targets MPSoCs and uses similar models to those of Sesame. In MAPS, mappings are obtained using simple heuristics.

This paper first describes an environment setup for comparing Sesame and MAPS that allows to obtain meaningful results. Comparing two large frameworks is no simple task. If the results should be of any value, we have to ensure the input to the different frameworks is indeed comparable. We contribute to future efforts by describing the lessons learnt in the process of comparing. In particular, we explain why common abstractions are crucial for ensuring a fair comparison, and motivate the need for formal models of hardware at the system level.

For the comparison itself we use a set of applications from different areas in the embedded systems domain, including multimedia and signal processing. While efficient software execution can have different goals, like power consumption or resource usage, a minimal computation time remains one of the main goals of any system. We thus compare the execution times estimated for the mappings obtained by the different frameworks. Since the time spent while deriving a mapping is

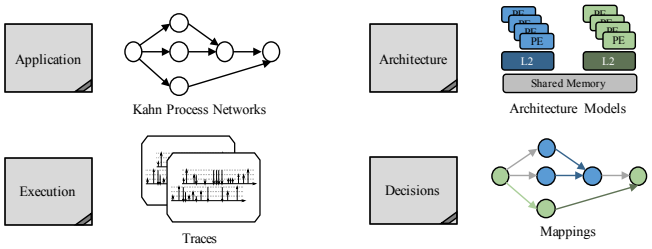

Figure 1. Abstractions in the mapping problem

important for design or compilation cycles, we also compare the execution times of the mapping algorithms themselves.

The rest of the paper is structured as follows. Section II introduces the two frameworks and their common models. The basic experimental setup for comparison is described in Section III. The comparison results are the matter of Section IV, while sections V and VI discuss related work and conclude the paper, respectively.

II. KPN Applications on Heterogeneous Platforms

The problem of programming and efficiently executing applications for heterogeneous multicores is, in general, an extremely complex problem. As is common with such general problems, abstractions are used in order to reduce the problem in a way which is general enough to produce good results, but specific enough to allow for structured approaches to solve it. An example of such an abstraction for programming heterogeneous multicore systems is to use Kahn Process Networks (KPNs) [13] to describe applications, and to consider the problem of mapping KPN applications to hardware resources. This includes abstractions on different parts of the system: at the application level, the execution model, and in describing the hardware platform.

In this section we introduce Kahn Process Networks and process traces. We then briefly state the problem of mapping KPN applications to heterogeneous hardware, and explain the different abstractions involved. Thereafter we describe the two frameworks analyzed in this paper: Sesame and MAPS.

Problem Definition

The problem of mapping KPN applications to heterogeneous hardware involves abstractions at different levels, as depicted in Figure 1. At the **application** level, programs are described using the KPN model. In this model, applications are partitioned into different *processes*, which encapsulate the different parts of the computation. These processes are not isolated; they communicate by exchanging data. In the KPN model this is abstracted by defining communication *channels* between processes that act as unbounded FIFO buffers. The Kahn *process network* is the resulting system, usually formalized as a graph with processes as nodes and channels as edges.

The structure offered by KPNs has several advantages. One of these comes at the **execution** level. From its semantics, KPN applications are deterministic [13]. That is, the process network will always produce the same results given the same input. In particular, the results are independent of the execution order of the processes and individual timings, provided no artificial deadlocks are introduced when restricting the sizes of the FIFO buffers. This fact can be utilized to create program traces of a KPN application that are independent of the execution, capturing the behavior of the execution at a high abstraction level.

Abstractions are also required for describing the hardware **architecture**. To this end, an architecture is described as a set of *processing elements* (PEs) and *communication resources*. The latter is an abstraction for any way data can be shared between processes residing on one or several PEs. These range from simply shared memories, and local scratchpads for single PEs, to specialized resources like hardware-FIFOs. Actual hardware architectures, and the libraries used (or operating system, if applicable), are much more complex than this. However, there is no simple abstraction that allows to capture all these details in a straightforward manner, which is why different frameworks use different such abstractions.

Using these models, **decisions** are made for deploying the applications onto the hardware. At this level, one also uses abstractions. As is canonical from the application description, processes are mapped to PEs, and the communication channels between them to hardware resources. It is also common to distinguish between the mapping of processes to PEs and the scheduling at runtime. In this paper, we limit ourselves to scheduling within a single PE when several processes have been mapped to it. There are additional decisions that have to be taken in this context, like *buffer sizing*, where the sizes of the FIFO buffers have to be chosen. This is crucial for an efficient and deadlock-free execution, but is outside the scope of the comparison in this paper. We limit ourselves to comparing strategies for mapping.

The problem of mapping KPN applications to heterogenous hardware, as studied in this paper, is that of finding a mapping of processes to PEs and of channels to communication resources. This mapping should be optimal or at least near-optimal in some sense for a particular execution, which is captured in the form of a trace. Optimality can be defined in different ways. In this paper, we limit ourselves to execution time.

A. The Sesame Framework

Sesame is a system-level modeling and simulation environment that facilitates automated performance analysis of MP-SoC systems, according to the Y-chart design approach [24], [11]. The various components of Sesame's modeling and simulation framework are shown in Figure 2. Sesame maps application models onto architecture models for co-simulation by means of a trace-driven simulation, while using an intermediate mapping layer for scheduling and event-refinement purposes.

For application modeling, Sesame uses the aforementioned KPN model of computation, where the processes are written in C++. To allow for rapid creation and modification of models, the structure of the KPN in the application is

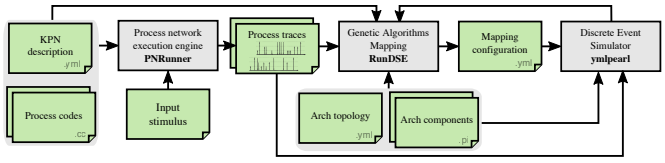

Figure 2. The Sesame flow

not hard-coded in the C++ implementation of the processes. Instead, it is described in a language called YML (Y-chart Modeling Language), which is an XML-based language. This also facilitates the creation of libraries of parameterized YML component descriptions that can be instantiated with the appropriate parameters, thereby fostering re-use of (application) component descriptions. To simplify the use of YML even further, a YML editor has been developed to compose model descriptions using a GUI.

Sesame features a process network execution engine supporting Kahn semantics, called PNRunner. It executes KPN application models, thereby generating the application events that represent the workload imposed on the architecture. This execution engine runs the KPN processes as separate threads using the Pthreads API. By executing the KPN model, annotations from manual instrumentation cause the KPN processes to generate traces of application events which subsequently drive the underlying architecture model. There are three types of application events: the communication events *read* and *write*, and the (symbolic) computational event *execute*. These application events typically are coarse grained, such as *execute(DCT)* or *read(16x16_pixel_block, channel_id)*.

The architecture models in Sesame, which typically operate at the so-called transaction level, simulate the performance consequences of the computation and communication events generated by an application model. To this end, the architecture model components are parameterized with the latencies associated with specific application events (e.g., executing a particular function, reading from memory, etc.). An architecture model is constructed from generic building blocks provided by a library, which contains template performance models for processing components (like processors and IP cores), communication components (e.g., busses and crossbar switches) and various types of memory. The structure of an architecture model is also described in YML within Sesame. It specifies which building blocks are used from the library and how they are connected. Sesame's architecture models are implemented using the in-house language Pearl, which is a small but powerful discrete-event simulation language that provides easy construction of the models and fast simulation.

To bind application tasks to resources in the architecture model, Sesame provides an intermediate mapping layer. It controls the mapping of KPN processes (i.e. their event traces) onto architecture model components by dispatching application events to the correct architecture model component. The mapping also includes the mapping of KPN channels onto communication resources in the architecture model. The mapping layer has two additional purposes. First, the event dispatch mechanism in the mapping layer provides a variety of static and dynamic policies to schedule application tasks (i.e., their event traces) that are mapped onto shared architecture model components. Second, the mapping layer is also capable of dynamically transforming application events into (lower-level) architecture events in order to facilitate flexible refinement of architecture models [24].

The output of system simulations in Sesame provides the designer with performance estimates of the system(s) under study together with statistical information such as utilization of architectural components (idle/busy times), the contention in a system (e.g., network contention), profiling information (time spent in different executions), critical path analysis, and average bandwidth between architecture components. Such results allow for early evaluation of different design choices, identifying trends in the systems' behavior, and can help in revealing performance bottlenecks early in the design cycle. Here, the process of design space exploration is also facilitated by the fact that system configurations (bindings, scheduling and arbitration policies, performance parameters, and so on) are specified using YML descriptions. Hence, different system configurations can be rapidly simulated without remodeling and/or recompilation. To actually explore the design space to find good system implementation candidates, Sesame typically deploys a Genetic Algorithm (GA). For example, to explore different mappings of applications onto the underlying platform architecture, the mapping of application tasks and inter-task communications can be encoded in a chromosome. This chromosome is subsequently manipulated by the genetic operators of the GA [10]. Such GA-based design space exploration has been demonstrated to yield good results [25].

B. The MAPS Framework

The MPSoC Application Programming Studio (MAPS) is a collection of tools for parallel programming [19], [5]. It includes a parallelizing compiler that uses profiling information to dynamically track data dependencies and thereby extract hidden parallelism from sequential C programs [8]. Besides the sequential input, MAPS accepts a parallel specification written in the so-called "C for Process Networks" language [27], an extension to the C language that allows to specify KPN applications. Finally, MAPS includes a source-to-source compiler based on Clang [17], called *cpn-cc*, that generates platform-specific C code for heterogeneous MPSoCs. As such, even if using the KPN application model, MAPS is a framework for programming rather than for modeling.

MAPS parallel programming flow for KPN applications is shown in Figure 3 (see also [6]). Similar to Sesame, the tool flow uses event traces in order to derive the mapping. These traces are not generated with user annotations but directly from the C code that defines the behavior of the processes. To this end, *cpn-cc* is used to generate an instrumented Pthreads-implementation of the application. When running this implementation on the host, a functional trace is generated which contains information about the control paths followed by every processes and the channel access events (read and

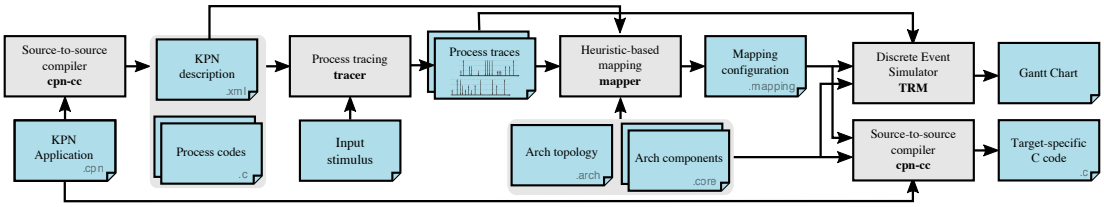

Figure 3. The MAPS flow

write). Different sequential performance estimation techniques are then used to approximate the execution time elapsed between events for each one of the processor types in the target multicore (e.g., [12]).

Similar to Sesame, MAPS includes an abstract model of the architecture. The model includes descriptors for processors, processor types, memories, interconnects, runtime parameters (e.g., scheduling policies and context switch costs) and a list of communication primitives. This list specifies, in a generic way, which APIs can be used to move data between two processors in the platform. The cost of moving data is modeled as a piecewise function that retrieves the number of cycles needed depending on the number of transferred bytes. To account for DMA-based and similar communication, the cost is split into producer and consumer side. These cost functions are obtained from the platform's data sheets and by running benchmark applications. They can be relatively complex for real architectures, as shown in [23]. On the processor side, MAPS uses a list of functional units with supported operations, number of registers and other architectural parameters. These models are used for the sequential performance estimation mentioned above. In contrast to Sesame, the architecture model was not originally meant to be used for MPSoC design but rather to model existing platforms.

For mapping, MAPS supports several user constraints, including mapping constraints and real time constraints. To find a mapping that meets the constraints in the most efficient way, or to find an unconstrained mapping with the best performance, simple heuristics are used. These include common approaches, like load balancing or a heuristic based on simulated annealing. These, as well as other heuristics like the "affinity" and "throughput" heuristics are described in [6]. The heuristics have as input the time-annotated execution traces and the architecture model. In [7], for example, a graph representation of the traces is used for buffer sizing and mapping. There, a more elaborate heuristic called Group-Based Mapping (GBM) iteratively assigns resources to the application by analyzing the dynamic critical path of the graph. As the heuristic progresses, critical channels and/or processes are fixed to good performant resources (primitives or processors). Towards the end, less critical application elements are then assigned to resources.

Akin to Sesame, MAPS uses a discrete event simulator to estimate the performance of the KPN application given a mapping. It is referred to as the "Trace Replay Module" (TRM). The estimate is then used by iterative heuristics and to provide a final estimate to the programmer. It includes a Gantt Chart, buffer utilization profiles and other execution statistics. The TRM uses the runtime information to simulate context switches, consumes computation time as estimated for the sequential processes and, retrieves the communication costs depending on the token sizes, known at compile time.

Today, the MAPS tool flow is now commercialized by Silexica as the "SLX Tool Suite" [28]. In this paper, we compare against the version of the tools and the algorithms as described in [5].

III. Comparing the Frameworks

When building large frameworks like MAPS and Sesame, even very similar ideas will result in differences in the implementation. An evident difference is, however, the way the frameworks make mapping decisions (c.f. Figure 1). While MAPS focuses on quick and simple heuristics, Sesame explores the design space with elaborate and time-consuming genetic algorithms. Both approaches come with trade-offs: while we expect simple heuristics to yield acceptable results in a short time, we also expect genetic algorithms to achieve better results at the cost of a longer computation time. One aim of this work is to investigate the accuracy of these expectations.

In this section we present the experimental setup for comparing both frameworks, as well as the lessons learned by doing it. We show why the abstract model of KPNs and traces is invaluable for comparison, and how it allows us to overcome design differences, like the granularity of the executions. Similarly, we explain how the architecture model represents the major hurdle for tallying the frameworks, and the limitations this brings to our comparison.

A. Experimental Setup

To achieve a fair comparison of the two frameworks, we leverage the abstractions at different levels of the flows as presented in Figure 1. For comparing the quality of mappings, the absolute values of simulation times obtained with different simulators would be meaningless. We therefore decided to compare performance as reported by the Sesame simulator. This implies that the mappings produced by one framework must be translated to the other for simulation purposes. Everything else is kept fixed during the comparison.

In a similar way we translate the result of the mapping, we have to ensure that both frameworks receive the exact same model of the application. As input model we selected CPN applications and translated the KPN representation to Sesame.

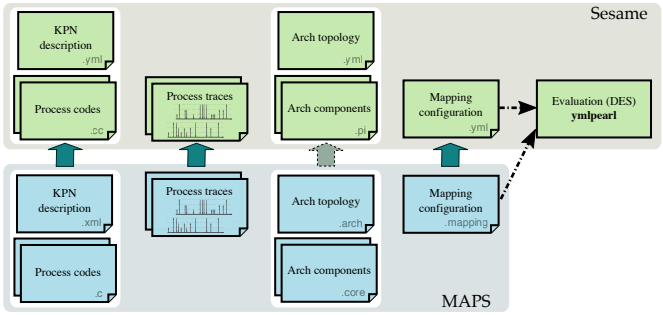

Figure 4. Comparing both flows

The reason for this choice is that MAPS integrates a flow for estimating performance on different processor types [12], while Sesame expects these estimations to be there already. Since performance estimation is outside the scope of this work, we decided to use the values produced by MAPS for both tools. This also implies that we need to translate MAPS traces into Sesame traces.

Figure 4 shows the basic flow for comparing the mapping strategies of both frameworks. In the figure we show how the intermediate abstractions from the MAPS framework, at the bottom, are translated into Sesame abstractions, at the top. We use blue arrows to represent automated translation tools implemented in the context of this work to enable the comparison. Building these tools was possible due to the formal nature of the abstractions used at the application, execution and decision levels.

A the application level, the KPN model is the same in both frameworks. There are, of course, minor implementation differences. For example, channels are described using input and output ports of processes on Sesame, in contrast to MAPS, where communication is described by writing to and reading from named channels. However, since the model itself is the same, the translation work is a straightforward task and needs only concern itself with implementation details. The same is true for translating at the level of mapping decisions.

At the execution level, process traces are also a well-defined abstraction. However, MAPS and Sesame use different philosophies for traces. MAPS annotates execution on a fine-grained level, not only documenting every write and read, but also the execution time between these, however minor it might be. Sesame, on the other hand, uses annotations at a coarser level. The processing overhead between consecutive communication events, for example, is modeled into the communication costs, and no execution annotation is made for the process. This has the advantage of reducing the trace size and thus simulation time, but incurs in accuracy loss. For comparison, however, both approaches could be used. This is where the formal model of traces shines, since it allows us to use different levels of abstraction with the same formalism. For this work we used fine-grained traces, since we obtained those from the MAPS performance estimation tool, and they allow us to better scrutinize the accuracy of both mapping decision strategies.

The dotted light-blue arrow in Figure 4 represents the translation of the architecture model. There is no formalism similar to KPN and traces for describing architectures. For this reason, there was no simple, automated approach for translation. Therefore, we manually translated the architecture model. Since there were several model differences, we calibrated both architectures iteratively, as outlined in the following section, in order to have a high fidelity in the execution times from the simulations.

B. Calibrating the Architecture Models

Calibrating the architecture models is a crucial task for ensuring a fair comparison since the mapping decisions are often based on estimations extracted from the model. In particular, we need to ensure enough fidelity between models. When choosing between two mappings, the idea is for the algorithms to choose the one with the lower execution time. However, if the two algorithms disagree on which mapping yields the lowest execution time, their performance is not being compared fairly.

Ensuring a good correlation and fidelity between the simulated execution times is difficult. Several factors play a role in this. Since there is no formal description of the architecture, details of the simulation differ. Examples of this are the modeling on when hardware is blocked during the simulation, how communication costs are modeled, the modeling of context switches, among others.

In order to calibrate the models, we ran an application as benchmark on both simulators (an mjpeg decoder, in this case). We calibrated a model of the TI Keystone II architecture, which features heterogeneous cores and communication resources [2]. We randomly generated 1000 mappings and estimated the execution time using each mapping on both frameworks. We identified points of large discordance between simulations in this fashion, and by looking at the mappings, execution statistics and Gantt charts, we identified the differences of the models that yielded different estimations. We iterated this process until we obtained a satisfactory correlation of the simulations. Figure 5 shows the results of the final calibration of the architectures. Each point in the figure represents a different randomly-generated mapping. Its coordinates represent the estimated execution times simulated on both frameworks. Thus, in the ideal case, depicted as the blue line, the values should be the same for both frameworks.

The estimations from both simulations compared in Figure 5 have a strong correlation. For these measurements, we obtained an estimate of the correlation-coefficient (Pearson product-moment) of 0.91. An important measure in design-space exploration, however, is also fidelity. To this end, we estimated both usual measures of rank correlation. The Kendal τ coefficient for the dataset yielded an estimate of $\tau = 0.69$. The Spearman rank-correlation coefficient ρ, on the other hand, resulted in a correlation of $\rho = 0.84$. This implies that while the absolute values of the simulations are strongly correlated, the fidelity (monotonicity of simulated time) is not optimal. This is a limitation of our comparison which cannot

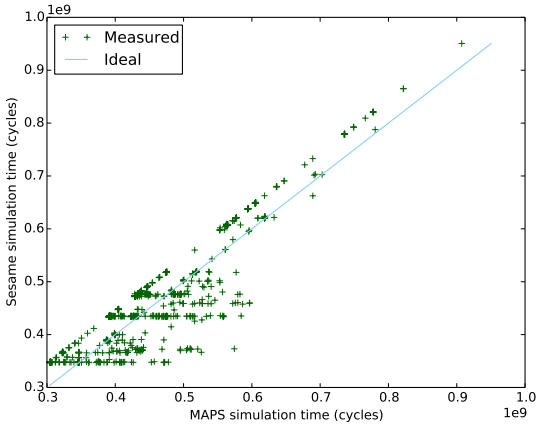

Figure 5. Calibration of the architecture and execution models

be simply circumvented, since it depends on the hardware models and implementation details of the simulators that cannot be fine-tuned only from the architecture description.

C. Lessons Learned

While working towards a fair comparison of the methods used in both frameworks, we learned important lessons along the way. We believe these lessons could provide important insights in the future, when considering a comparison of system-level design and programming frameworks, as well as when designing new ones.

From a distance, both frameworks look very similar. This is even more so apparent when comparing figures 2 and 3. Upon closer inspection, however, the small differences accumulate. Several of these differences have an obvious explanation: While Sesame is built with modeling and estimation in mind, the main focus of MAPS is on actually generating and executing code on hardware. For example, this design difference has repercussions on the models of the architectures. MAPS models the sizes of the channels and memories in bytes, while Sesame does it in tokens, independent of the data type. Similarly, MAPS models the frequencies of the processors, while Sesame deals with amounts of cycles. In the same manner, MAPS has a rigid hardware model, inspired by what was found in the examples. This includes a split-cost communication model between the producer and the consumer side or context-switching costs when multiple processes run on a single PE. Sesame, on the other hand, has a much more flexible description: it uses the PEARL language to program its functionality. While this allows to model the aspects outlined above, each of them has to be programmed additionally in PEARL for the different components.

Apart from the technical effort, translating the KPN descriptions from one framework to another is straightforward. The same is true for the traces. This is because of the formal nature of the abstract models used. Since there is a very clear definition of the model, for translating from one framework to the other we do not need to first understand what is being abstracted and how it is modeled.

In both frameworks, the hardware architecture, including the communication between component and runtime details, are modeled with ad-hoc abstractions. While this approach serves the purpose of each individual framework, comparison and integration are greatly hampered by it. This is not a deliberate choice of the frameworks: to the best of our knowledge, there is no formal model well-suited for hardware architecture abstractions in a way that is comparable to KPNs and execution traces at the application and execution levels. We believe that the system-level community as a whole would benefit from having and using such formal models at this level of abstraction.

Name	No. of pr./ch.	Short description
audio filter	$8/8$	Two-channel low-pass filter
mandelbrot	$18/32$	Mandelbrot set calculation (16 Jobs)
mjpeg	$12/15$	Motion JPEG decoder (128×128)
matmult	$5/6$	10×10 matrix multiplication
sobel	$5/15$	Sobel filter on 40×40 image

Table I
BENCHMARKS APPLICATIONS

IV. EXPERIMENTAL RESULTS

Using the basic setup described in Section III (refer to Figure 4), we evaluate MAPS and Sesame on a set of KPN applications. The benchmarks are listed in Table I, showing the number of processes and channels of each application to give an idea of the complexity for the mapping problem. The "mjpeg" and "audio filter" benchmarks from the multimedia and signal processing domains are similar to those described in [5]. Similarly, the "sobel" filter benchmark is described in [27].

For the architecture, we use the model of the TI Keystone II [2], the calibration of which we described in Section III-B. It has 8 DSP cores and 4 ARMs, each with their own local scratchpad memories. Additionally, the PEs can communicate over the L2 cache and over main memory. The model includes different communication libraries, which can have complex trade-offs depending on the token sizes and the amount of data transferred [23]. All experiments were conducted on a machine running Ubuntu Linux 15.10, with eight Intel®Core™i7-4790 CPU at 3.60 GHz and 32 GB of DDR3 memory at 1600 MHz.

We used five of the simple heuristics available in MAPS, including the more specialized GBM heuristic, as mentioned in Section II-B. Similarly, we used the GA, which uses a $\mu + \lambda$ evolutionary strategy [14] with fixed parameters of $\mu = 20 = \lambda$. With this non-tuned setup, we recorded the results of the algorithm for $5, 10, 20$ and 50 generations. Since the GAs depend on (pseudo) random number generation, we recorded the averages over five executions with five different random seeds.

Figure 6 shows the results of the different mapping strategies for each application. For comparing between benchmarks, the resulting application execution time was normalized to the one obtained with the GA with 50 generations. For the GAs, additional error bars show the unbiased estimations of the standard deviation resulting from the five executions. The figure shows that, as expected, GAs mostly outperform simple

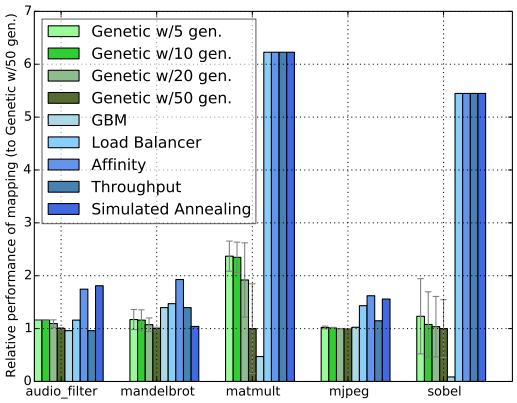

Figure 6. Performance of the mappings obtained with the different heuristics

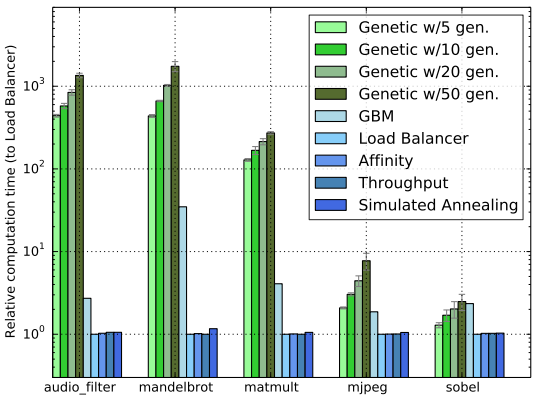

Figure 7. Computation times of the different heuristics

heuristics. Two interesting exceptions are the Sobel filter and matrix multiplication, where GBM outperforms the GAs. This is the case because in these applications, communication is fine-grained and frequent. Thus, the best mapping actually uses a single PE, because communication costs outweigh speed gains from multiple cores. This solution is difficult to find with an approach based on randomness, like GAs. This is also noticeable due to the large variability in the results of the different runs of the GA for these benchmarks, compared to the other benchmarks.

Besides the application performance results, we also evaluate the execution time of the different mapping algorithms, with a summary in Figure 7. The figure shows relative computation times normed to that of the load balancing heuristic. Again, the timings for the GA algorithms depend on the random seed, and the variation is represented by error plots showing the unbiased estimations of the standard deviation. This variability comes primarily from the fact that the GAs are implemented using a cache to avoid re-executing a simulation, and the amount of repeated mappings depends on the random seed. The figure has a logarithmic scale because of the large timing differences. We report user-time, since the GA algorithms can use several threads to evaluate mappings in parallel. Additionally, since the trace formats are different (binary and text formats), we ignore the system time required to read the files.

The trends in this graph are also very clear. In general, genetic algorithms tend to take the most time, ranging from 0 to 2 orders of magnitude longer in the examples considered. The GBM heuristic, which is more complex than the rest and has a stronger dependence on the problem size, tends to take longer than the rest of the heuristics, but less than the GAs. The difference here is within around an order of magnitude in all cases. It is important to note issues affecting these results which should be taken into account when interpreting them. The GAs are implemented using Python, a dynamic language which has to be interpreted, even though the simulations are compiled C programs. The MAPS heuristics, on the other hand, are implemented in C++ and are therefore expected to run faster than if implemented in Python. Other issues like the sizes and formats of the different input files have a large impact on execution time, since these have to be parsed.

V. Related Work

The problem of mapping on multicore systems is a growing field of research and has spawned several methods for solving it [29]. In particular, besides MAPS and Sesame, there are several other frameworks for modeling dataflow applications and mapping onto MPSoCs, most notably DOL [31], ESPAM [21], Daedalus [22] and TURNUS [4].

Similarly, genetic algorithms and other metaheuristics have been compared thoroughly in general settings [1]. At the application level, the different formal models have also been studied in-depth [18]. Even a related problem of mapping has been studied, comparing and contrasting different heuristics [3]. However, these results and approach are not apt for contemporary MPSoCs. While related, they are from the pre-multicore era and deal with very different assumptions about the computational cores and communication costs, compared to today's heterogeneous multicores, where most of the communication happens on-chip. In this work we have presented a comparison that is specific to today's problem of mapping applications to physical resources. To the best of our knowledge, there has been no direct comparison of frameworks as those described above. In particular, heuristics for solving the problem of mapping for MPSoCs as is addressed by the described approaches have never been systematically compared.

VI. Conclusion

In this paper we compared two system-level frameworks for solving the mapping problem: Sesame and MAPS. Both these frameworks are similar at a first glance: they have analogous flows and use comparable abstractions at each level. We identified that some of these abstractions are easy to compare thanks to formal models like KPNs and traces. However, there is no formalism for describing hardware architectures, which makes a fair comparison difficult. To ensure a good correlation and fidelity between the simulated execution times, we calibrated the architecture models by using numerous generated mappings. A formal hardware model would have significantly eased this phase. Altogether, we believe that the

academic community and industry would greatly benefit from using a common formal model for hardware architectures. An interesting work in this direction is the standardization effort of the Multicore Association on the Software-Hardware Interface for Multi-Many-Core (SHIM) Specification [30] (with contributions by Silexica).

We also evaluated different mapping algorithms used in Sesame and MAPS. Genetic algorithms found better mappings than heuristic-based ones, with a relative performance ranging from $1\times$ to $5\times$. At the same time, there are two extreme cases where the heuristic-based algorithm GBM outperforms genetic algorithms. Comparing the computation time, the genetic algorithms take significantly more time to produce the mapping, ranging from 0 to 2 orders of magnitude in the examples considered.

There are several directions for future work that can use the comparison setup described in this paper. In particular, we can extend the evaluation and analysis, taking particular parameters into account, like the granularity of the traces, or looking into multi-objective optimization.

ACKNOWLEDGMENTS

This work was partly supported by the German Research Foundation (DFG) within the Cluster of Excellence Center for Advancing Electronics Dresden (cfaed). The authors would also like to thank the HiPEAC Network of Excellence (www.hipeac.net) for providing the mechanisms that helped us carry out this collaborative work. Finally, we would like to thank Silexica (www.silexica.com) for making their embedded multicore software development tool available to us.

REFERENCES

[1] BALUJA, S. An empirical comparison of seven iterative and evolutionary function optimization heuristics.

[2] BISCONDI, E., FLANAGAN, T., FRUTH, F., LIN, Z., AND MOERMAN, F. Maximizing Multicore Efficiency with Navigator Runtime. White Paper, feb 2012.

[3] BRAUN, T. D., SIEGEL, H. J., BECK, N., BÖLÖNI, L. L., MAHESWARAN, M., REUTHER, A. I., ROBERTSON, J. P., THEYS, M. D., YAO, B., HENSGEN, D., ET AL. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed computing 61*, 6 (2001), 810–837.

[4] BRUNET, S. C., MATTAVELLI, M., AND JANNECK, J. W. Turnus: a design exploration framework for dataflow system design. In *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on* (2013), IEEE, pp. 654–654.

[5] CASTRILLON, J., AND LEUPERS, R. *Programming Heterogeneous MPSoCs: Tool Flows to Close the Software Productivity Gap*. Springer, 2014.

[6] CASTRILLON, J., LEUPERS, R., AND ASCHEID, G. MAPS: Mapping concurrent dataflow applications to heterogeneous MPSoCs. *IEEE Transactions on Industrial Informatics 9*, 1 (Feb. 2013), 527–545.

[7] CASTRILLON, J., TRETTER, A., LEUPERS, R., AND ASCHEID, G. Communication-aware mapping of kpn applications onto heterogeneous mpsocs. In *Proceedings of the 49th Annual Design Automation Conference* (New York, NY, USA, June 2012), DAC '12, ACM, pp. 1266–1271.

[8] CENG, J., CASTRILLON, J., SHENG, W., SCHARWÄCHTER, H., LEUPERS, R., ASCHEID, G., MEYR, H., ISSHIKI, T., AND KUNIEDA, H. Maps: an integrated framework for mpsoc application parallelization. In *Proceedings of the 45th annual Design Automation Conference* (June 2008), ACM, pp. 754–759.

[9] CHUNG, H., KANG, M., AND CHO, H.-D. Heterogeneous multiprocessing solution of exynos 5 octa with arm® big. little technology.

[10] ERBAS, C., CERAV-ERBAS, S., AND PIMENTEL, A. D. Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design. *IEEE Transactions on Evolutionary Computation 10*, 3 (2006), 358–374.

[11] ERBAS, C., PIMENTEL, A. D., THOMPSON, M., AND POLSTRA, S. A framework for system-level modeling and simulation of embedded systems architectures. *EURASIP Journal on Embedded Systems 2007*.

[12] EUSSE, J. F., WILLIAMS, C., AND LEUPERS, R. Coex: A novel profiling-based algorithm/architecture co-exploration for asip design. *ACM Transactions on Reconfigurable Technology and Systems* (may 2014).

[13] GILLES, K. The semantics of a simple language for parallel programming. In *In Information Processing'74: Proceedings of the IFIP Congress* (1974), vol. 74, pp. 471–475.

[14] GOLDBERG, D. E., ET AL. *Genetic algorithms in search optimization and machine learning*, vol. 412. Addison-wesley Reading Menlo Park, 1989.

[15] GWENNAP, L. Adapteva: More flops, less watts. *Microprocessor Report 6*, 13 (2011), 11–02.

[16] JAVAID, H., IGNJATOVIC, A., AND PARAMESWARAN, S. Rapid design space exploration of application specific heterogeneous pipelined multiprocessor systems. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 29*, 11 (2010), 1777–1789.

[17] LATTNER, C. LLVM and Clang: Next Generation Compiler Technology. The BSD Conference, Ottawa, Canada, May 2008.

[18] LEE, E. A., AND SANGIOVANNI-VINCENTELLI, A. A framework for comparing models of computation. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 17*, 12 (1998), 1217–1229.

[19] LEUPERS, R., AND CASTRILLON, J. Mpsoc programming using the maps compiler. In *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific* (Jan. 2010), pp. 897–902.

[20] NEJAD, A. B., GOOSSENS, K., WALTERS, J., AND KIENHUIS, B. Mapping kpn models of streaming applications on a network-on-chip platform. In *ProRISC 2009: Proceedings of the Workshop on Signal Processing, Integrated Systems and Circuits* (2009), p. 6.

[21] NIKOLOV, H., STEFANOV, T., AND DEPRETTERE, E. Systematic and automated multiprocessor system design, programming, and implementation. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 27*, 3 (2008), 542–555.

[22] NIKOLOV, H., THOMPSON, M., STEFANOV, T., PIMENTEL, A., POLSTRA, S., BOSE, R., ZISSULESCU, C., AND DEPRETTERE, E. Daedalus: toward composable multimedia mp-soc design. In *Proceedings of the 45th annual Design Automation Conference* (2008), ACM, pp. 574–579.

[23] ODENDAHL, M., CASTRILLON, J., VOLEVACH, V., LEUPERS, R., AND ASCHEID, G. Split-cost communication model for improved mpsoc application mapping. In *International Symposium on System on Chip (SoC), 2013* (Oct. 2013), IEEE, pp. 1–8.

[24] PIMENTEL, A. D., ERBAS, C., AND POLSTRA, S. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *IEEE Transactions on Computers 55*, 2 (2006), 99–112.

[25] QUAN, W., AND PIMENTEL, A. Towards exploring vast mpsoc mapping design spaces using a bias-elitist evolutionary approach. In *Proc. of the Euromicro Digital System Design Conference (DSD '14)* (Aug. 2014).

[26] RAMEY, C. TILE-Gx100 ManyCore Processor: Acceleration Interfaces and Architecture. Presented at HotChips 23, Aug. 2011.

[27] SHENG, W., SCHÜRMANS, S., ODENDAHL, M., BERTSCH, M., VOLEVACH, V., LEUPERS, R., AND ASCHEID, G. A compiler infrastructure for embedded heterogeneous mpsocs. *Parallel Computing 40*, 2 (2014), 51–68.

[28] SILEXICA. Slx tool suite. http://silexica.com/products. [Online; accessed 27-April-2016].

[29] SINGH, A. K., SHAFIQUE, M., KUMAR, A., AND HENKEL, J. Mapping on multi/many-core systems: survey of current and emerging trends. In *Proceedings of the 50th Annual Design Automation Conference* (2013), ACM, p. 1.

[30] THE MULTICORE ASSOCIATION, INC. *Software-Hardware Interface for Multi-Many-Core (SHIM) Specification, V1.0.* The Multicore Association, Inc, Jan. 2015.

[31] THIELE, L., BACIVAROV, I., HAID, W., AND HUANG, K. Mapping applications to tiled multiprocessor embedded systems. In *Application of Concurrency to System Design, 2007. ACSD 2007. Seventh International Conference on* (2007), IEEE, pp. 29–40.