MPSoC Programming using the MAPS Compiler

Rainer Leupers, Jeronimo Castrillon
Institute for Integrated Signal Processing Systems
RWTH Aachen University, Germany
maps@iss.rwth-aachen.de

Abstract— The problem of efficiently programming complex embedded heterogeneous Multi-Processor Systems-On-Chip (MPSoCs) continues to be one of the biggest hurdles in the IT community. Extracting parallelism from sequential applications, dealing with different programming models, and handling real time constraints in the presence of multiple concurrent applications are some of the challenges that make MPSoC programming so difficult.
In this paper we describe the MAPS tool suite, which tries to tackle these aspects of MPSoC programming in an integrated development environment built upon the Eclipse framework. We give an overview of the MAPS framework, highlighting its differences to the previous work in [7], and report on experiences using the tool.

I. Introduction

In the last years, several research groups both in academia and in industry have focused their efforts on solving the challenges of MPSoC programming. It is a fact that current SW productivity is not keeping the pace of the growing SW requirements of embedded devices. Requirements are doubled every ten months while SW productivity is doubled only every 2 years [10]. This so-called *productivity* gap is a consequence of the complexity of MPSoC programming.

MAPS (MPSoC Application Programming Studio) aims at reducing the gap by providing solutions to several problems an embedded designer is faced with. The complete tool suite is embedded in an integrated development environment that provides an easy-to-use Graphical User Interface (GUI) based on the Eclipse framework [11]. MAPS's main features are:

- Legacy code: Manual partitioning of sequential legacy code is known to be difficult and error prone. MAPS provides a set of algorithms for semi-automatic program partitioning, as reported in [7].
- Parallel programming: Dataflow programming models [27, 30] are well suited for representing signal processing applications. These programming models expose parallelism explicitly and are therefore well suited for MPSoC programming. MAPS accepts applications represented as *Kahn Process Networks* (KPNs) [17].
- Abstraction and retargetability: Given a partitioned application, moving a task to a processor of a different type requires considerable effort. Low level SW APIs (e.g. for scheduling, communication and synchronization) need to be adapted. MAPS abstracts the complexities of the underlying HW platform in a platform model, which makes MAPS retargetable.
- Mapping and scheduling: Computing schedules and mappings for a given application by hand is a tedious work. MAPS provides means for achieving this automatically.
- Functional validation: At early stages, a designer might want to evaluate a partition, or simply check if the functionality is correct without going through complex processor-specific tool chains and without having to setup and wait for cycle accurate simulators. For fast functional validation, MAPS is fully integrated with the HVP simulator [8].
- Application classes: In an embedded device, applications usually have different characteristics and constraints. Some applications will have real time constraints while others may be bound at design time to a given Application Specific Instruction Set Processor (ASIP) [16] or HW accelerator. MAPS provides GUI and code constructs to specify such constraints, thereby making clear distinctions among application classes.
- Multiple applications: Embedded devices are no longer built to execute single applications but need to be verified in the presence of multiple applications. MAPS includes means for describing multiple applications inside a project and defining how these applications interact with each other. MAPS is also equipped with algorithms to analyze multi-application scenarios.

The rest of this paper is organized as follows. Section II revisits the MAPS framework as presented in [7]. Section III discusses the related work. Details on the extensions of the MAPS framework are given in Section IV and results in Section V. Finally this paper finishes with conclusions in Section VI.

II. Background

In our previous publication [7], we presented an approach for semi-automatic parallelism extraction. A new granularity, namely the *Coupled Block* (CB), was introduced that provides more flexibility than function-level

978-1-4244-5766-3/10/$26.00 ©2010 IEEE

or basic-block level granularity. Our partitioning algorithm was benchmarked with two applications (JPEG and ADPCM), for which code was generated and simulated in the TCT MPSoC [36]. A key technology introduced in [7] was an analysis phase, in which the application code is instrumented to obtain run-time traces. These traces are used to provide dynamic information which is annotated to the control and data flow edges to steer the partitioning process.

This sequential partitioning flow remains one of the core components of the MAPS framework. It can be used even when dealing with already parallelized applications. The tracing mechanism is also used to profile KPN applications as will be discussed in Section IV.B.

III. Related Work

Several works are related to MAPS in the way they support semi-automatic parallelization of sequential applications. MPA [23][2] starts from C code and uses a parallelization specification to guide the parallelism extraction. Authors in [35] use a similar dynamic data flow analysis approach to identify parallel code regions in desktop computing. Authors in [9] proposed several application transformations that can be controlled by the designer. However, none of these approaches provides support for multiple application scenarios nor for explicitly parallel programming models.

Several frameworks support parallel specifications for single applications. HOPES [20] receives a parallel specification of the application in the form of a task graph. Data level parallelism is specified by OpenMP pragmas [26] within tasks. HOPES is retargetable, i.e. it generates code with an abstract API that can be implemented for different platforms. The DOL framework [34] uses analytical performance estimation for KPN applications and employs evolutionary algorithms to compute the mapping. Daedalus [25] is a framework for Polyhedral Process Networks (PPN). PPNs feature better properties but are more restricted than KPNs. This framework accepts mappings generated by the *Sesame* tool [28] which uses evolutionary algorithms as well. SystemCoDesigner [15] is similar to Daedalus, but includes also performance evaluation. This framework allows to implement different Models of Computation (MoCs) with a library-based approach upon SystemC called SystemMoC.

Few frameworks allow compiler analysis inside already parallelized tasks in the way we do. The work in [22] uses a so-called *gray task model* in which tasks (or *thread frames*) are subdivided into smaller blocks (or *thread nodes*), which are similar to our coupled blocks.

Finally, several works on multiple applications relate to our framework [4, 24, 32, 18, 19, 37]. Most of this research has focused on Synchronous Dataflow (SDF) graphs [21], while our framework supports a more general MoC, the KPN. We tackle the analyzability problem of KPNs via profiling and tracing as we did for the case of sequential applications.

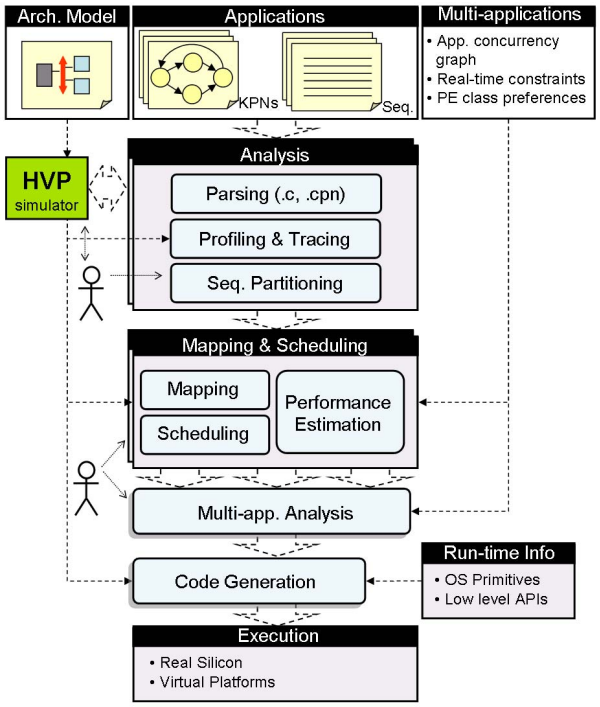

Fig. 1. MAPS flow overview

IV. Overview of the MAPS Tool Suite

Figure 1 shows a general overview of the MAPS flow. MAPS accepts applications written in a sequential and in a parallel way. We believe in an evolutionary approach and therefore both input specifications share the C syntax (see Section B). After all, around 85% of the embedded programmers still use C/C++ in their designs[13]. Apart from the applications, the user needs to provide a model of the architecture and describe multiple applications constraints and scenarios (see Section C). All this is accomplished within the MAPS GUI, which is composed of a set of plug-ins and plug-in extensions (e.g. CDT[12]) for the Eclipse framework. Based on the premise that a user interface must not add complexity to MPSoC programming, we designed an intuitive GUI as shown in Figure 2.

Independent of the type of application, the MAPS flow consists of an analysis phase, a mapping and scheduling phase, multi-application analysis and code generation. Of course, individual steps (e.g. tracing) for sequential application differ from those for parallel applications. In the analysis phase, the applications are profiled and parsed into an Intermediate Representation (IR). The partitioning phase works on this IR as discussed in [7] and provides means for user interaction. The user can also, at anytime, test the behavior (e.g. functionality, parallelism degree) with the HVP simulator. In the mapping and scheduling phase, each application is analyzed independently, taking into account the constraints provided by the user. The user can interact with this phase by varying the optimization objectives, e.g. the minimum platform utilization. This phase produces several scheduling configurations for each application, which are then used in the multi-application analysis phase. Here, dif-

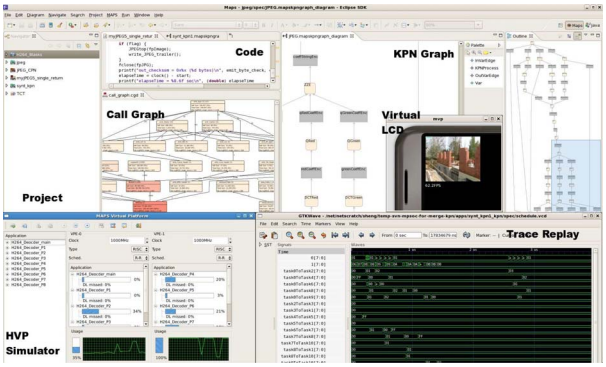

Fig. 2. MAPS GUI. Sample of sequential and parallel inputs, call graph and coupling with HVP simulator.

ferent multi-application scenarios are analyzed according to the so-called Application Concurrency Graph (ACG). Once the user is satisfied with a configuration for a given multi-application scenario, he can proceed to the code generation phase to test the results either on a virtual platform or on the real hardware. For this phase, the information about the OS primitives and low level APIs, if any, has to be available to MAPS.

In the remaining of this section, we provide details on the new components of the MAPS framework.

A. Sequential Partitioning: Revisited

In [7] we presented a hierarchical clustering approach in which the *Weighted Statement Control Data Flow Graph* (WSCDF) of a function was successively partitioned. In this approach, later partitions have a coarser granularity than former ones, and it was left to the user to decide which iteration of the algorithm to select for every function by inspecting the granularity of the graphs.

Since then, the partitioning phase of MAPS have become more powerful thanks to a more global view. Several algorithms have been introduced to the framework that analyze the application as a whole, going beyond function borders. Two of these algorithms are SCC improvement and load balancing, as described in the following.

A.1 SCC Improvement

Consider the sample partition shown in Figure 3a and its execution profile on three different Processing Elements (PEs) in Figure 3b. From this example, it can be easily seen that blocks B and C could be merged without affecting the execution time[1] (see Figure 3c). This is a typical example of loop-carried dependencies, which give birth to *Strongly Connected Components* (SCC) in the data flow graph. These SCCs are easily identified by using *Tarjan's* algorithm.

In an application, a SCC may contain nodes that belong to different loop nesting levels and different functions. This is taken into account by the algorithm, which assigns every SCC a level according to the most outer

[1]Note that block C cannot be merged with A without having to merge B due to the constraints imposed on the CBs.

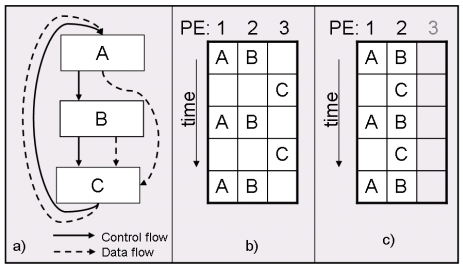

Fig. 3. Example of the SCC algorithm. (a) Initial partition. (b) Execution profile without SCC improvement. (c) Execution profile after SCC improvement.

loop to contain one of its nodes. The algorithm then tries to optimize every SCC from the most inner SCCs to the outer ones.

For the SCC optimization we have developed a simple heuristic shown in Figure 4. For a given *SCC* and application graph *G*, the algorithm first linearizes all nodes in the loop body by flattening control branches and removing backward edges. Then, the linear list is walked and nodes are clustered together as long as the function `CanMege` returns true. This function takes into account the current critical path of the SCC and analyzes if it is affected by merging the node i to the current cluster cl. In the example in Figure 3a, merging B and C does not change the critical path length, while merging A and B does. The `Merge` function performs the actual merging, which merges also other CBs so that the new block has a common dominator and post-dominator.

Algorithm: SCC_Improvement
Input: Application graph $G = (V, E, W)$, $SCC \subset V$
Output: clusters cls

$list\_of\_nodes = \texttt{LinearizeGraph}(G, SCC)$;
$cl = \{1\}$; //current cluster
$cl\_first = 1$; //first node in current cluster
$cls = \{\}$; //resulting cluster list
foreach i *in* $list\_of\_nodes$ **do**
  if not $(\texttt{CanMerge}(i, cl, G))$ **then**
    //Make a new cluster
    $cls = cls \cup \{cl\}$;
    $cl = \{i\}$;
    $cl\_first = i$;
  else
    //Add current node to existing cluster
    $\texttt{Merge}(i, cl, G)$;

if $cl \neq \{\}$ **then**
  $cls = cls \cup \{cl\}$;
return cls ;

Fig. 4. SCC Improvement Algorithm

A.2 Load Balancing

Load balancing is a very well known concept in the Operating Systems community. Figure 5 shows an example that illustrates the idea behind load balancing in the context of MAPS. In this example, the first two nodes are merged whereas the last node is split. For the splitting procedure, the partition of the previous iteration of the clustering algorithm is analyzed.

The example in Figure 5 refers to local balancing, i.e.

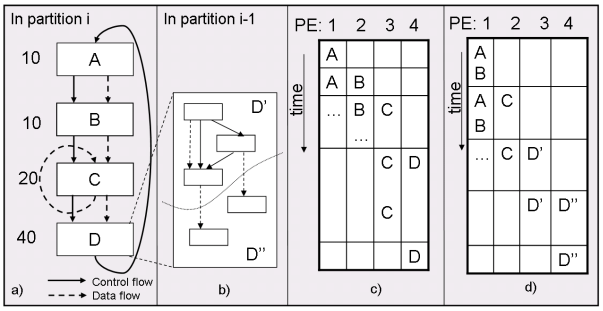

Fig. 5. Example of the load balancing algorithm. (a) Initial partition with annotated execution time per CB. (b) Block D in the partition of the previous iteration. (c),(d) Execution profile with and without load balancing respectively.

within a function. In MAPS we also address the issue of global balancing, of which the idea is to balance the granularity of partitions across different functions. Note that to be able to compare partition granularities, the context from which the function is called matters. A function which appears down in the Call Graph (or better, in a loop nesting) need not be of comparable granularity to a function called within the main function of the application. The load balancing algorithm first assigns every function under analysis a *level*. The level represents the nesting in which the function was called. A function may have different call sites, and therefore different levels. For these cases, the highest level (deepest nesting) is taken into consideration.

After assigning functions to levels, the algorithm walks the levels from the highest to the lowest (`main`). At every level, a reference granularity is determined depending on the individual granularities of the different functions in the level. Thereafter, the partitions are equalized to the reference granularity first by splitting big blocks and then by merging small blocks. The splitting step uses the partition of the previous iteration of the clustering algorithm. The merging step follows a first fit bin packing heuristic, i.e. tries to merge blocks until the granularity is similar to the reference granularity. After modifying the partitions, the time information of the functions in the next level is updated, before applying the same procedure on the functions at that level.

B. Parallel Input: KPN

Dataflow programming models have gained acceptance in the last years because of them being well suited to represent typical digital signal processing applications. At the same time, these programming models ease the task of the compiler, since the data flow between tasks (*processes* or *actors*) is explicit. There are different flavors of dataflow models, ranging from simple Homogeneous SDFs (HSDF) to complex hybrid models (see CFDF [29]). Usually, different models represent a tradeoff between expressiveness and analyzability. For the MAPS framework we have selected a KPN-based programming model [17, 27, 14, 3].

In a KPN application, the functionality is implemented by *processes* that communicate with each other through *channels*. Channels contain FIFO buffers which in the praxis are bound in size. A process execution will block when reading from an empty channel or writing to a full channel. In this section we present some extensions to the C syntax that allow to describe KPN applications. We also provide some insights on how parallel applications are analyzed within the MAPS flow. For further details, the reader is referred to [5].

B.1 KPN Representation

We have defined a set of extensions to the C language that allow to represent KPN applications with the familiar C syntax. Apart from this textual input, suitable for complex applications, the MAPS GUI comes with a plug-in for visualizing and modifying KPN applications.

Listing 1 shows an example of a process with one input channel and one output channel that decodes a Run-Length Encoded (RLE) sequence of integers. Channels are declared with the `__fifo` keyword and can be of any type, including user-defined structures. The actual declaration of the process (in Line 2) is preceded by `#pragma maps process` and followed by the declaration of input and output channels and several additional qualifiers. In this example the clause `prefer(risc)` was used that instructs the MAPS framework to try to map this process to a PE of type RISC. Accesses to channel variables (`A`, `B`) represent *push*/*pop* operations on the fifo channels. For this reason, a temporal variable `val` is used to store the value in channel `A` and copy it repeatedly to channel `B` (Lines 6, 8). Given an input stream $\{2,1,3,2,\dots\}$, this process will output $\{1,1,2,2,2\dots\}$.

```
1 __fifo int A, B;
2 #pragma maps process rle_dec, in(A), out(B), prefer(risc)
3 {int cnt, val, i;}     // Local variables to the process
4 {  // Process body: Repeated for ever
5   cnt = A;              // Reads first token: count
6   val = A;              // Reads second token: value
7   for (i = 0; i < cnt; ++i) {
8    B = val;             // Outputs count times val
9 }}
```

Listing 1 Sample code for RLE decoding

B.2 Tracing and Profiling

The tracing and profiling step of the MAPS framework for parallel applications reuses the one for sequential applications. Due to the fact that the history of tokens on the channels of a KPN does not depend on the scheduling [17], the KPN is first translated into a *pthread* [31] application and is run in the host in order to record all the tokens produced in every channel. Thereafter, each process is executed in isolation with their read accesses to channels replaced by reads to the file containing its tokens. It is on this isolated implementation of the process where the traditional sequential instrumentation and profiling step of MAPS is executed. After doing this, the parallel traces are reconstructed. The result of this process is a trace for every process in which the exact path in the control flow graph between channel accesses

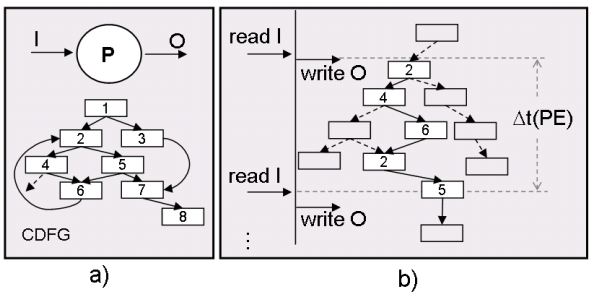

Fig. 6. Sample KPN trace. (a) Process and its internal Control-Data Flow Graph (CDFG). (b) Sample of a portion of the trace between a write and a read access.

is known. The time elapsed between channel accesses is obtained for every PE type in the platform. This is shown graphically in Figure 6.

B.3 Mapping and Scheduling

With the information provided in the traces, the mapping and scheduling phase computes different schedules and mappings with different platform utilizations. Several heuristics were reported in [5]. Improvement to those heuristics and research on new heuristics is being currently conducted.

C. Dealing with Multiple Applications

As already mentioned, MAPS is able to analyze multiple applications. One of the mechanisms that enables this is the Application Concurrency Graph (ACG), which can be created within the GUI by dragging and dropping applications belonging to the MAPS project. The ACG is a weighted graph $ACG = (V, E, W)$ in which nodes represent applications, and an edge $e = (a_1, a_2)$ indicates that applications a_1 and a_2 may run simultaneously. The weights on the edges (W) are used to assign a probability of the two applications running simultaneously and are reserved for future use.

Every subgraph of the ACG represents a potential use case, i.e. a design situation for which a scheduling configuration needs to be found and verified. Instead of trying to compute schedules for several applications at the same time, we use *composition*. In this approach, a function is used to judge if previously independently computed schedules can run simultaneously without violating the constraints. The composition function may be as simple as adding the average utilization values per processor and checking for a given threshold. More elaborate composition functions can also be used [5] and are matter of research.

D. Code Generation

Apart from code generation for the TCT framework reported in [7], we have developed backends for pthread, TI OMAP 3530 [33], HVP and an OSIP-based platform [6]. While implementing this different back-ends, several common functionalities were factored together within the code generation flow, so that writing a backend for a new platform is greatly simplified.

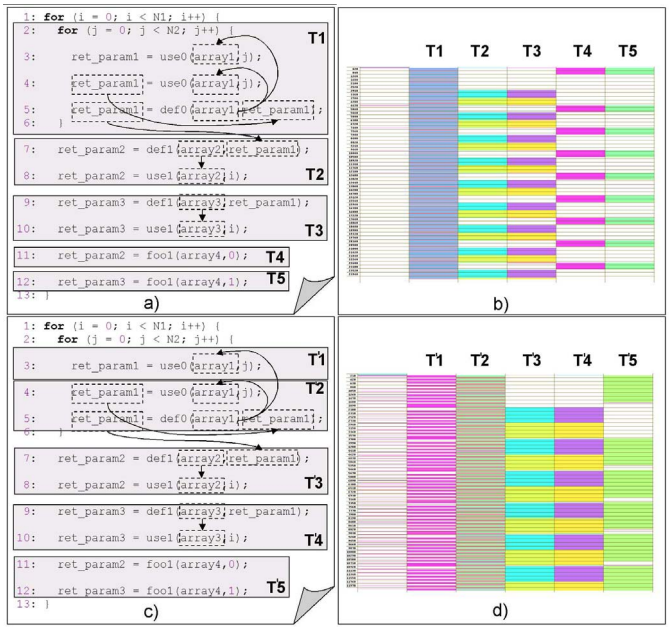

Fig. 7. Sequential partitioning on test program. (a) Initial partition of clustering algorithm. (b) Execution trace on the TCT simulator of partition in (a). (c) Partition with SCC improvement and load balancing. (d) Execution trace on the TCT simulator of partition in (c). Arrows in (a), (c) coarsely show data dependency.

V. Results

The MAPS framework has been tested on different applications of different nature in the last years. It has become more user friendly, and if desired, requires less user intervention for the sequential partitioning.

The new algorithms for sequential partitioning have been benchmarked extensively with test programs. An example is shown in Figure 7. The initial partition, shown in Figure 7a, is the result of the clustering algorithm after two iterations. It can be seen how regions connected by data dependencies were clustered together, since *def-use* pairs on the same array are kept in the same block. The load balancing algorithm first tries to split block T1, which displays the highest execution time (see Figure 7b). The previous partition of this block is composed of two blocks (see blocks T'1 and T'2 in Figure 7c) that result of applying the SCC improvement due to the SCC in lines 4 and 5. Finally, even if the calls to `foo1()` are independent, blocks T4 and T5 are merged into T'5 in order to match the granularity of the other blocks. The resulting partition and its execution trace can be seen in Figure 7c,d. This partition has a speedup of 38% with respect to the initial one, while keeping the same number of PEs. The algorithms were also tested in the JPEG application used in [7]. The initial partition had a speedup of 4.1X compared to the 3.61 reported in [7], without requiring the user to specify the iteration nor picking the functions to analyze. Moreover, after few modifications similar to those in [7], the speedup and the parallel efficiency reached 9.5X and 47% respectively.

Several KPN applications have been rewritten using the C extensions introduced in Section B. It took a programmer less than 8 hours to parallelize a JPEG applica-

tion from its standard sequential implementation using the partition scheme followed in [5]. A similar amount of time was needed to port the MPEG-2 application from the benchmarks in [1]. Both applications were analyzed, and schedules were generated for a test platform, in about 10 minutes on a Dual Core AMD Opteron running at 2.6 GHz. Further details are given in [5].

The various back-ends of the MAPS framework have been tested thoroughly. The pthread back-end, apart from enabling parallel tracing, helps in debugging the parallel code before going for target compilation. The HVP back-end has served to obtain a better grasp of performance issues and multiple applications effects taking into consideration custom schedules and mappings, still with a high level of abstraction. Finally, apart from the TCT backend, the flow has been also tested successfully on an OSIP-based platform and on the OMAP platform. The OSIP configuration is generated automatically, which greatly eases the usability, especially if the user is not familiar with OSIP's APIs. Same happens on the OMAP platform, where different implementations of the FIFOs and synchronization primitives are selected automatically depending on the mapping.

VI. Conclusions

In this paper we have described some of the new features of the MAPS framework for MPSoC programming. New algorithms to improve partitions of sequential applications written in C were presented and tested. The flow for handling parallel applications and the C extensions that allow to describe them were briefly introduced. Finally, a way of describing interactions among multiple parallel applications through the Applications Concurrency Graph was discussed, which is used during the composability analysis.

There are still many issues to be tackled in MPSoC programming to close the productivity gap, but we believe that the MAPS framework is working in the right direction and will contribute to make adding functionality to upcoming embedded devices easier.

Acknowledgments

This work has been supported by the UMIC Research Centre, RWTH Aachen University.

References

[1] Artist. *www.artist-embedded.org/artist/Benchmarks.html*.
[2] R. Baert, E. Brockmeyer, S. Wuytack, and T. J. Ashby. Exploring Parallelizations of Applications for MPSoC Platforms Using MPA. In *DATE '09: Proceedings of the conference on Design, automation and test in Europe*, 2009.
[3] T. Basten and J. Hoogerbrugge. *Communicating Process Architectures 2001*, chapter Efficient Execution of Process Networks, pages 1–14. IOS Press.
[4] M. Bekooij, R. Hoes, O. Moreira, P. Poplavko, M. Pastrnak, B. Mesman, J. D. Mol, S. Stuijk, V. Gheorghita, and J. van Meerbergen. *Dataflow Analysis for Real-Time Embedded Multiprocessor System Design*, chapter Dynamic and Robust Streaming in and between Connected Consumer-Electronic Devices, pages 81–108. Springer, 2005.
[5] J. Castrillon, R. Velasquez, A. Stulova, W. Sheng, J. Ceng, R. Leupers, and G. Ascheid. Trace-based KPN Composability Analysis for Mapping Simultaneous Applications to MPSoC Platforms. In *DATE '10: Proceedings of the conference on Design, automation and test in Europe (to appear)*.
[6] J. Castrillon, D. Zhang, B. Kempf, T. Vanthournout, R. Leupers, and G. Ascheid. Task Management in MPSoCs: An ASIP Approach. In *ICCAD '09: Proceedings of the 2009 IEEE/ACM International Conference on Computer-aided Design*, 2009.
[7] J. Ceng, J. Castrillon, W. Sheng, H. Scharwächter, R. Leupers, G. Ascheid, H. Meyr, T. Isshiki, and H. Kunieda. MAPS: An Integrated Framework for MPSoC Application Parallelization. In *DAC '08: Proceedings of the 45th annual conference on Design automation*, pages 754–759. ACM, 2008.
[8] J. Ceng, W. Sheng, J. Castrillon, A. Stulova, R. Leupers, G. Ascheid, and H. Meyr. A High-level Virtual Platform for Early MPSoC Software Development. In *CODES+ISSS '09: Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 11–20, New York, NY, USA, 2009. ACM.
[9] P. Chandraiah and R. Domer. Code and Data Structure Partitioning for Parallel and Flexible MPSoC Specification Using Designer-Controlled Recoding. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(6):1078–1090, June 2008.
[10] W. Ecker, W. Mueller, and R. Doemer. *Hardware-dependent Software - Principles and Practice*, chapter Hardware-dependent Software - Introduction and Overview. Springer, 2008.
[11] Eclipse - an open development platform. *www.eclipse.org*.
[12] Eclipse C/C++ Development Tooling. *http://www.eclipse.org/cdt*.
[13] Embedded software stuck at C. *http://www.eetimes.com/news/design/showArticle.jhtml?articleID=202102427*.
[14] M. Geilen and T. Basten. Requirements on the Execution of Kahn Process Networks. In *ESOP 2003*, pages 319–334. Springer Verlag, 2003.
[15] C. Haubelt, J. Falk, J. Keinert, T. Schlichter, M. Streubühr, A. Deyhle, A. Hadert, and J. Teich. A systemc-based design methodology for digital signal processing systems. *EURASIP J. Embedded Syst.*, (1):15–15, 2007.
[16] P. Ienne and R. Leupers. *Customizable Embedded Processors: Design Technologies and Applications (Systems on Silicon)*. Morgan Kaufmann Publishers Inc., 2006.
[17] G. Kahn. The Semantics of a Simple Language for Parallel Programming. In J. L. Rosenfeld, editor, *Information Processing '74: Proceedings of the IFIP Congress*, pages 471–475. North-Holland, New York, NY, 1974.
[18] A. Kumar, S. Fernando, Y. Ha, B. Mesman, and H. Corporaal. Multiprocessor systems synthesis for multiple use-cases of multiple applications on FPGA. *ACM Trans. Des. Autom. Electron. Syst.*, 13(3):1–27, 2008.
[19] A. Kumar, B. Mesman, B. Theelen, H. Corporaal, and Y. Ha. Analyzing composability of applications on MPSoC platforms. *J. Syst. Archit.*, 54(3-4):369–383, 2008.
[20] S. Kwon, Y. Kim, W.-C. Jeun, S. Ha, and Y. Paek. A Retargetable Parallel-Programming Framework for MPSoC. *ACM Trans. Des. Autom. Electron. Syst.*, 13(3):1–18, 2008.
[21] E. Lee and D. Messerschmitt. Synchronous Data Flow. *Proceedings of the IEEE*, 75(9):1235–1245, Sept. 1987.
[22] Z. Ma, P. Marchal, D. P. Scarpazza, P. Yang, C. Wong, J. I. Gmez, S. Himpe, C. Ykman-Couvreur, and F. Catthoor. *Systematic Methodology for Real-Time Cost-Effective Mapping of Dynamic Concurrent Task-Based Systems on Heterogenous Platforms*. Springer Publishing Company, Incorporated, 2007.
[23] J.-Y. Mignolet, R. Baert, T. J. Ashby, P. Avasare, H.-O. Jang, and J. C. Son. MPA: Parallelizing an Application onto a Multicore Platform Made Easy. *IEEE Micro*, 29(3):31–39, 2009.
[24] O. Moreira, F. Valente, and M. Bekooij. Scheduling Multiple Independent Hard-Real-Time Jobs on a Heterogeneous Multiprocessor. In *EMSOFT '07: Proceedings of the 7th ACM & IEEE international conference on Embedded software*, pages 57–66, New York, NY, USA, 2007. ACM.
[25] H. Nikolov. *System-Level Design Methodology for Streaming Multi-processor Embedded Systems*. PhD thesis, Universiteit Leiden, 2009.
[26] The OpenMP specification for parallel programming. *http://www.openmp.org*.
[27] T. M. Parks. *Bounded scheduling of process networks*. PhD thesis, Berkeley, CA, USA, 1995.
[28] A. Pimentel, C. Erbas, and S. Polstra. A Systematic Approach to Exploring Embedded System Architectures at Multiple Abstraction Levels. *IEEE Trans. Comput.*, 55(2):99–112, 2006.
[29] W. Plishker, N. Sane, and S. S. Bhattacharyya. A Generalized Scheduling Approach for Dynamic Dataflow Applications. In *DATE'09*.
[30] S. Sriram and S. S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. Marcel Dekker, Inc., New York, NY, USA, 2000.
[31] Standard for Information Technology - Portable Operating System Interface (POSIX). Shell and Utilities. IEEE Std 1003.1-2004. Issue 6, section 2.9. *IEEE and The Open Group*.
[32] S. Stuijk, T. Basten, M. C. W. Geilen, and H. Corporaal. Multiprocessor Resource Allocation for Throughput-Constrained Synchronous Dataflow Graphs. In *DAC '07: Proceedings of the 44th annual Design Automation Conference*, pages 777–782, New York, NY, USA, 2007. ACM.
[33] Texas Instruments, OMAP 3530. http://focus.ti.com/docs/prod/folders/print/omap3530.html.
[34] L. Thiele, I. Bacivarov, W. Haid, and K. Huang. Mapping Applications to Tiled Multiprocessor Embedded Systems. In *ACSD '07: Proceedings of the Seventh International Conference on Application of Concurrency to System Design*, pages 29–40, Washington, DC, USA, 2007. IEEE Computer Society.
[35] G. Tournavitis, Z. Wang, B. Franke, and M. O'Boyle. Towards a Holistic Approach to Auto-Parallelization – Integrating Profile-Driven Parallelism Detection and Machine-Learning Based Mapping. In *PLDI 09: Proceedings of the Programming Language Design and Implementation Conference. Dublin, Ireland, June 15 - 20*, 2009.
[36] M. Z. Urfianto, T. Isshiki, A. U. Khan, D. Li, and H. Kunieda. A multiprocessor system-on-chip architecture with enhanced compiler support and efficient interconnect. In *IP-SOC 2006, Design and Reuse, S.A.*, 2006.
[37] H. Yang and S. Ha. Pipelined Data Parallel Task Mapping/Scheduling Technique for MPSoC. In *DATE '09: Proceedings of the conference on Design, automation and test in Europe*, 2009.