Public with confidential appendices

Grant agreement no. 100265
Artemis Project

ASAM

Automatic Architecture Synthesis and Application Mapping

D2.3 Method of and report on system analysis

Due Date of Deliverable	October 31, 2011
Completion Date of Deliverable	October 31, 2011
Start Date of Project	May 1, 2010 – Duration 36 months
Lead partner for Deliverable	DTU
Author(s)	L. Micconi, J. Madsen, M.R. Boesen

Revision: 1.0

Project co-funded by the Artemis Joint Undertaking Call 2009		
Dissemination Level		
PU	Public	
PU/COA	Public with confidential appendices	x
PP	Restricted to other program participants (including Commission Services)	
RE	Restricted to a group specified by the consortium (including Commission Services)	
CO	Confidential, only for members of the consortium (including Commission Services)	

Contents

1 Introduction

The aim of this report is to present how the application and platform models described respectively in deliverable D2.2 [15] and D2.1 [14] can be combined together through the mapping. The evaluation of the cost of a mapping solution in terms of performances and energy consumption will be presented and we will show how this information can be used for defining the composition of the multi-ASIP platform. Additionally the deliverable will describe the exchange of information between the macro and micro-architecture that is used to produce efficient mapping.
The report is organized as follows: in the next chapter the mapping problem and its challenges are described; chapter 3 will focus on the tools and methods available in the literature that deal with mapping of application on multi-core architecture. Appendix A is then dedicated to the presentation of our approach for solving the mapping problem. Finally Appendix B describes the interaction between the macro and micro level and the iterative process for the refinement of the final mapping and platform.

2 Challenges of mapping applications on a multi-ASIP platform

The target ASAM macro-level platform for the mapping is a Multi-ASIP architecture composed by multiple Application-Specific Instruction-set Processors (ASIP) interconnected through a bus or a Network on Chip (NoC). Moreover the platform can include HW Accelerators and shared memories. This type of architecture can satisfy both the requirements of performance and flexibility of embedded systems: ASIPs support the designers with the possibility of choosing among multiple micro-architectures and allow the extension of the instruction set according to the characteristics of the mapped applications [3]. More precisely ASIP development itself must be driven by the application and this implies an initial benchmarking of the application requirements [13]. Another advantage in using ASIPs is the speed-up of the design process and the reduction of the time-to-market for the final product. Furthermore the impact of interconnections between the ASIP cores (and additional components) is very relevant and their cost must be included in the design phase in order to guarantee the desired performances and meet the specification.

The mapping of applications on multi-ASIP architecture can be very challenging: to the complexity of properly associating tasks to the different processors (mapping and scheduling for multi-processor systems are NP-complete problems) is added the uncertainty of the configuration of the ASIPs that compose the macro-architecture. The objective of the mapping process is to find a good match between the tasks that compose the application and the processing elements, but as previously mentioned, the platform itself should be modeled/tuned according to the application requirements. These two elements are in contrast: to perform an adequate mapping the macro-architecture should be known, but this cannot be defined without knowing the characteristics of the tasks that are mapped on it (as the tuning of the processor depends on the tasks themselves). In order to solve this problem, an iterative process and exchange of information between the macro and micro level will be performed: the macro-level design phase will get estimations from the micro-level design and will use this information to build an efficient mapping and platform. This information can be gradually refined and more accurate information can be exchanged between the two levels converging to an optimal solution.

Additionally the mapping procedure is used to set the macro-architecture of the system; an estimation of the performance of a given mapping can also provide to the designer suggestions about the configuration of the HW platform in terms of number/type of cores and interconnection types. Our purpose is to include, inside the mapping, also the design space exploration of the macro-architecture. Additionally, in the macro-level design space exploration (DSE), we will to consider the entire design space that can be explored at micro-level in the tuning of the single ASIP. It is important to take into account that all the mapping decision at macro-level have repercussion on the improvements that can then be achieved at micro-level, so our main purpose is to avoid setting limitations at the design space that can exclude configurations that can bring to optimal solutions.

3 Methods and tools for automatic mapping of applications on multi-core platform

There are many approaches in the literature that face the problem of performance estimation and mapping of applications on multi-core SoC at macro-level (or system level). Following some of them are briefly presented.

3.1 Co-exploration of processors and communication

In [19] the authors implement a system-level design technique for the evaluation of a heterogeneous multi-processor architecture that takes into account the influence of processing elements and on-chip communication. In fact the impact of the communication cost can be very high (due to access to shared resources as buses and memories) and should be considered from the earlier stages of the design. The implemented framework is based on the LISA Processor Design Platform and on a SystemC Transaction Level Models; it performs an iterative and gradual refinement of the design using a co-exploration of processors and communication elements. In figure 3.1, the framework as represented by the authors in [19] is shown.

In particular the schematic shows as LISA is used for the system-level description of retargetable processing units (ASIP) and the SystemC Transaction Level Modeling (TLM) is employed for the communication elements (buses) and peripheral devices. These two interact in the definition of the

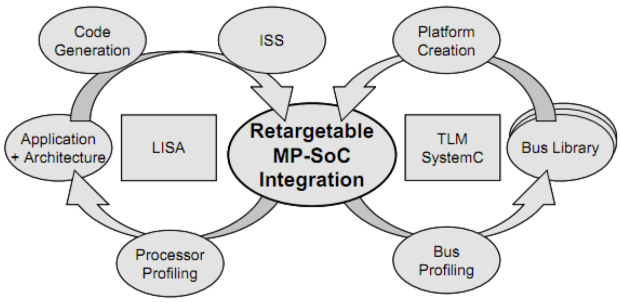

Figure 3.1: Schematic for system level co-simulation of processors and communication units

macro-architecture and the results are gradually refined and provide a cycle-accurate estimation of the system performances. In order to allow the interaction between the two models, a bus interface class library is defined. This permits to simulate the access to shared memories: the LISA memory API functions are mapped to memory accesses onto the TLM bus API. A flow composed by six phases is used for the design and gradual refinement of the macro-architecture

- Generation of a standalone processor simulator with instruction-level accuracy: the architecture of the processor is defined and internal memory resources and local buses can be instantiated and parameterized
- The previous processor model is integrated inside a SoC model (SystemC TLM). This allows taking into account the access to shared resources and their latencies and also verifying the functional behavior of the processor inside the system (instruction-level accuracy).
- The processor model is updated including the pipeline and provide cycle accurate performance estimation. In this phase however the model for concurrent access to memory resources is not refined.
- In this phase also the model for inter-processors communication and the access to shared resources is refined and becomes cycle-accurate.
- RTL model is built for the TLM bus
- RTL model is built for the ASIP

As case study, a JPEG decoder application has been used. The application has been partitioned in two tasks that have been arbitrarily mapped on two different processing units (a IDCTcore co-processor and a MIPS32) that communicate through a shared memory. Two different SoC topologies (AMBA AHB buses with different interconnections) have been explored and the gradual refinement of the performance estimations is shown. Moreover the case study enlighten how models at different levels of accuracy can be combined together in the total system evaluation.

Meanwhile in this work there is no explicit reference at the mapping problem, it explores the impact of communication and proposes a technique for exploring different SoC topologies through a gradual refinement of the evaluation of the system performances that are also key aspects in the ASAM project.

3.2 Modular Simulation Framework

In [8] the authors build a SystemC-based simulation framework for the mapping of applications on a multi-processor and multi-threaded (HW-MT) platform. This approach requires the definition of the application SW and algorithm (Functional phase) and the SW partitioning of the application so to extract the task level parallelism. Then the system architecture specification should be defined (MP-SoC platform phase) and the architecture is built as composition of high level IP blocks.
Following an approach very similar to the Y-chart model [9] (Figure 3.3), the application and architectures model are joint together through a spatial (allocation to a processing element) and

temporal (allocation of time budget) mappings. The mapping information are contained in a XML configuration file that specify the number of execution cycles per task, the number of processors and threads, the association of tasks to processors, the configuration of communication nodes and the instantiation and address mapping of the memory. The use of XML files speeds up the mapping exploration and allows the re-use of models. Moreover the concept of Virtual Processor Unit (VPU) is introduced and used to model the behavior of multiple tasks mapped on the same processing element (a VPU is defined for each processing unit present in the platform). VPU models the behavior of the tasks (multiple threads) and calculates their incoming and generated events; as a result, it computes the scheduled execution of tasks with multi-threading, task swapping and preemption. An IPv4 forwarding application with Quality of Service (QoS) has been used as case study. The target architecture for the mapping (temporal and spatial) is an Intel IXP2400 Network Processing Unit (NPU) composed by 8 RISC-like processing elements, each one supporting up to 8 hardware threads.

3.3 Design Methodology for Pipelined Heterogeneous Multiprocessor System

In [17], that authors propose a design methodology for multi-media systems composed by pipelined heterogeneous ASIPs. As their target applications are the streaming one that can exploit pipeline parallelism (at task level), the technique proposed has the main purpose of configuring the system pipeline in a balanced way, looking for a trade-off between system performances and area. The input application is partitioned in pipelined tasks according to its characteristics and for each possible partitioning a different task graph is generated. Each task in the task graph represents a stage of the pipeline and it is associated to a different ASIP. The different processors are interconnected through FIFO queues. Additionally, each ASIP in the pipelined system can be tuned/configured to improve the performance of the assigned pipeline stage.

The system design flow is represented in figure 3.2. The system receives in input the application code, a library of pre-configured processors and a set of cache and processor configurations (XPRESS, Xtensa PRocessor Extension Synthesis). After a compilation and profiling, a data flow graph of the application is built and then it is manually (or automatically) partitioned into multiple modules. Based on the partitioned application and the library and configurations provided as input, the designer generates a set of possible pipelined architectures, with different number of pipeline stages and different parallel pipeline flows. A heuristic algorithms is then used to explore the design space and find the best architectural configuration (in terms of performance and area), including the pipelined multiprocessor macro-architecture and the core configurations (i.e. cache and XPRES configurations). The design space explored is limited by a library of possible core configurations and by the assumption that all the stage of the pipeline should have similar latency (as the total execution time of a pipelined system is determined by the critical pipeline stage).

For the case studies, a JPEG and MP3 encoding algorithm have been used and mapped on Tensilica Xtensa LX platform, the authors demonstrate that compared to a single processor system, with this technique, it is possible to obtain a speed-up of respectively of 4.11 and 3.36 times.

The main differences of this approach compared to ASAM project is that we aim to develop a more general design methodology that will support the design for different application families

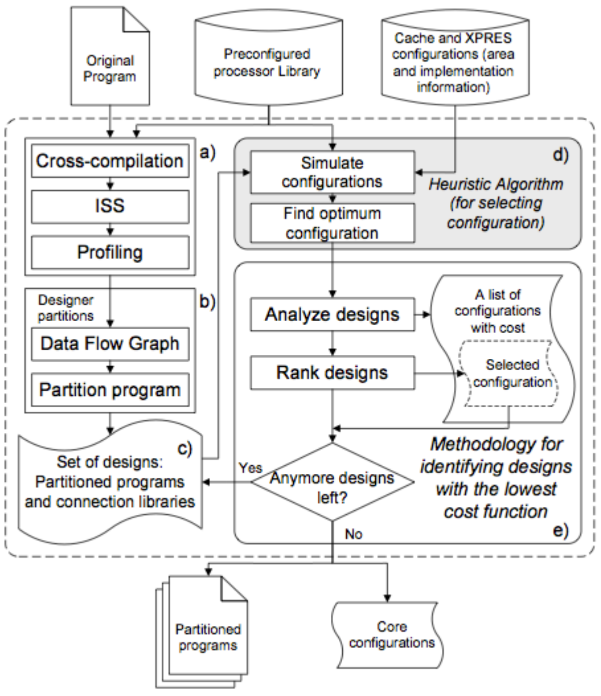
Original Program
Preconfigured processor Library
Cache and XPRES configurations (area and implementation information)
Cross-compilation
ISS
Profiling
a)
Designer partitions
Data Flow Graph
Partition program
b)
Set of designs: Partitioned programs and connection libraries
c)
Simulate configurations
Find optimum configuration
Heuristic Algorithm (for selecting configuration)
d)
Analyze designs
Rank designs
A list of configurations with cost
Selected configuration
Anymore designs left?
Yes
No
Methodology for identifying designs with the lowest cost function
e)
Partitioned programs
Core configurations

Figure 3.2: Design flow for Pipelined Heterogeneous Multiprocessor Systems (as presented in [17])

(multimedia and medical); in addition, we want to take into account all the possible ASIP configurations for a given task. Furthermore in [17], FIFOs are used as communication network between cores and no other interconnection types (as buses or NoC) are evaluated.

3.4 Metropolis and Metro II

Metropolis is a framework based on a platform-based design (PBD) methodology: its main purpose is to explore different mapping solutions and to evaluate the performance of the resulting system. It can be used in many application domains; following a specific case study is described [2]. The authors present the mapping of a JPEG encoder on MXP5800 Intel platform, through the use of Metropolis Design Framework. In the paper, particular attention has been given to the selection of the appropriate model of computation (MoC) for the application and for the platform; two different models are used in order to satisfy the different representation requirements. For the specific case of the JPEG encoder, an extension of the cycle-static data-flow is chosen: for each process the number of tokens produced and consumed during a firing can change and also can vary the number of tokens between different channels. Moreover each process consumes and produces tokens according to multiple firing rules, that are executed in a fixed order, and one writer and multiple readers are allowed per channel. Another important characteristic is the fact that a process, before writing on a channel, has to specify the number of tokens. These properties guarantee that scheduling, mapping and buffer sizing can be established (and deadlock avoided) and are also the reason for preferring a cycle-static data-flow as MoC instead of a Kahn Process Network (KPN) that can seem a more natural model for a concurrent data-streaming architecture. In fact a generic KPN can bring to artificial deadlock (i.e. process blocked on full channels) and buffer sizing and scheduling are undecidable problems. The architecture is modeled using Metropolis Meta-model. Metropolis can build a model as a composition of four different types of elements: process, media, quantity managers and netlist objects. Processes are active objects that can generate events during their execution and that can execute in parallel. The events can be associated to different physical quantities (e.g. time, power) by the quantity managers; these also manage the access to shared resources. Media are instead passive objects that provides services to processes and other media. Finally netlists allow connecting and instantiating other objects. The target architecture selected for the mapping is the Intel MXP5800, a digital media programmable processor. In this case tasks are modeled as processes, and the processing and communication (local SRAM or global registers) elements are represented by media. Metropolis framework is used to explore the mapping design space given the application and platform models. Each mapping solution is described by two different types of information: the synchronization between the events that constrains the access to the services and for each processing element, the execution order that should be used for the tasks mapped on it. The authors demonstrate the effectiveness of their method with different mapping scenarios and showing that the evaluation of the performances of Metropolis is quite accurate (1% error) compared to measurements on the real system. Metropolis implementation focuses on the modeling, mapping and performance estimation, but it does not consider the evaluation of different architectures or variation in the application modeling or implementation (that are instead a main target for the ASAM project).

Metropolis tool has been enhanced and a Metro II framework has been developed [1]. The main improvements introduced in Metro II are:

- support for pre-designed IP and for different semantic so that IP providers can use their domain specific tool and languages. From a practical point of view this means the user is supported in the construction of interfaces between the IP and the framework. Moreover Metro II allows defining custom interconnections between the different components.
- separation of the behavior of the system (functionality) from the cost of a specific solution (performance metrics), this is indicated as behavioral-performance orthogonalization. Moreover the quantity associated to the events has been divided in metric of interest and quantities for synchronization (these lasts are used by schedulers to manage shared resource access).
- exploration of the mapping design space in a structured way: the different mappings should be explored without modifying the application and platform models; moreover the two models need to be combined through the synchronizations of events. Tools are still needed for the automation of the mapping and correct-by-construction solutions should be guaranteed.

The Metro II design flow is organized in three different phases:

1. In the first phase all processes (e.g. tasks) should at least generate one event (also multiple events can be generated simultaneously) and then they block.
2. In the second phase all the events are characterized by multiple quantity of interest.
3. In the last phase a scheduling is performed and a subset of events is enabled for execution. Concurrent events are allowed, but just one per process.

Metro II proposes some solutions for the issues encountered with the application of Metropolis, however in [1] there is still no implementation for the design space exploration and automatic mapping.

3.5 UML based techniques: Koski and VTT_ABSOLUTE2.2%

Following two different techniques that exploit the potential of UML 2.0 for system-level modeling and performance estimation are presented. In [7] a system-level design flow (Koski design flow) for multi-processor SoC based on UML 2.0 is described. The flow includes design, architecture exploration, prototyping on FPGA and functional verification. As in our case the design of the architecture is tuned on the requirements of the application: the application is modeled as a set of tasks (and their internal behavior) and the main purpose of the design flow is finding a mapping of the functionality of the tasks on the architecture and defining the architecture itself. The application is represented through a functional model and the behavior of each task is captured through finite state machines. Starting from this representation, C code can be automatically generated. The system-level platform model is built starting from a library of components (Koski platform library): more in detail the platform is a combination of libraries that cover different entities as the hardware (processing elements and communication network), software (algorithm implementation of the applications and interfaces) and also the support for design automation (application distribution on multiprocessor at run-time, profiling and monitoring for performance evaluation). An interesting aspect of this work is the presence of a completely automated design flow that include the mapping of the tasks on the processing unit and use this information to perform the exploration of the architecture. The application, the architecture and the design constraints are described in UML 2.0 format, in particular the application and the architecture models can be developed independently (this differs from ASAM project, as the platform model is built considering the application model as reference). The design constraints are mainly defined through a cost function, of which the designer can define parameter weights and operations. In this way it is possible to constrain the optimization variables and also the parameters of the final platform. Application and platform model are combined during the mapping phase. Multiple tasks can be grouped together before being mapped, moreover the designer can suggest an initial mapping in order to guide the exploration and reduce the mapping possibilities. The exploration of the architecture, that is guided by mapping and scheduling, is divided in two phases:

- a static architecture exploration based on the analysis of the application and on a coarse model of the interconnection network (that takes into account power, area and bandwidth).This exploration phase is composed by an iterative process composed by allocation (to set the number of used cores), mapping and scheduling and it is guided through meta-heuristic methods (simulated annealing and group migration). The static analysis is performed building a Kahn Process Network (KPN) of the application UML model, that is then internally modified into a directed acyclic graph to allow static analysis.
- a dynamic method that explores the architecture. The communication network model used in this phase is cycle accurate or at transaction level with timing. The exploration is based again on an iterative process that evaluates minor changes in the mapping of the application

(moving tasks from one processing element to another) and estimates the platform performances through simulation. It is worth to notice that the mapping and the optimization of the communication network are separate phases, and that the interconnection parameters are optimized only after mapping and allocation are fixed.

The optimization processes are guided by a cost function and the problem is limited to a single objective optimization, meanwhile in ASAM project we focus on multi-objective design space exploration, so evolutionary or genetic algorithm are required. This makes the definition of the cost function as a crucial element, that can largely influence the design space exploration results.

As previously mentioned the design flow also covers verification and prototyping, as these two elements are out of the scope of this deliverable, they are not here described. The results obtained from the prototyping can be used to tune the application and architecture modeling and also the exploration phase in case the results obtained are not compliant with the designer's requirements.

The work presented in [10] describes a model-based technique that supports system-level design of multi-processor real-time systems through fast performance estimations (VTT_ABSOLUTE2.2%). The application is modeled in UML 2.0 or SystemC, more precisely the model captures the application workload (expressed as load primitives). The platform is modeled in SystemC and is a cycle-accurate representation of the behavior of the platform (services offered): this allows having different implementations for each service. For both application and platform hierarchical models are used; they are combined through mapping. Two interfaces have been defined to enable the interaction between the application and the platform: a higher level interface for the request of services available on the platform and required by the workload and a lower level interface for the specific load primitives (read, write, execute). The scheduling can be hard coded into the workload or can be supported by a model of the Operating System that control the access to the services. The performance of the entire system are estimated through simulation. This method mainly covers the performance estimation issue, but it should be possible to extend it to perform also design space exploration for finding a good mapping and platform definition. As a case study a mobile video player has been modeled and simulated.

3.6 Spade and Artemis

Spade framework ([12]) proposes a system level methodology for the architecture exploration of heterogeneous multi-processor signal processing systems. As the techniques previously described, it uses the separation of concerns:

- separation of computation from communication
- application and architecture can be modeled independently and then combined through the mapping for the evaluation of performance of the entire system (according to the approach described by the Y-chart ([9]).

Spade focus on a specific family of applications (signal processing) and allows the architecture exploration for a target of specific applications provided as input. The applications are modeled through a Kahn Process Network (KPN) and communication and computation workloads of each

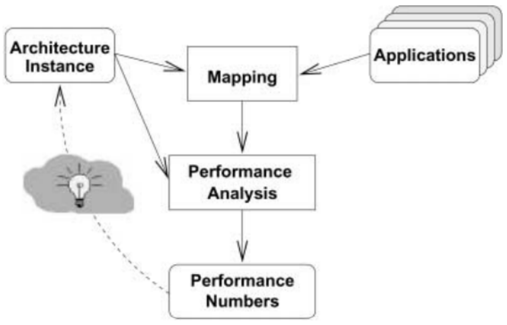

Figure 3.3: Y-chart design scheme (as presented in [9])

Kahn process are represented by traces. Moreover the application model captures the functional behavior of the system. The architecture model instead has no information about the behavior of the system and it is a composition of generic building blocks that are available in a library (processing, communication and memory resources). The building blocks do not express the functionality of the system, but they provide instead timing and synchronization information. The architecture is composed by different processing resources interconnected through a shared bus, direct link or shared memory. The mapping is then performed assigning the application traces to the different processing resources and the evaluation of the system performances is done through simulation; in particular traces are generated at run-time, during the simulation and a thread is allocated for each Kahn process. Different information according to the resources are collected and used in the estimation of the performances: processing and waiting times on I/O ports for the processing resources and the amount of data, utilization and waiting times for the communication resources. A platform and mapping solution has been defined for a MPEG-2 decoder. The architecture model of an existing commercial processor has been used as a starting point for the platform exploration and different latencies, frame rate and bus loads have been tested for the architecture design space exploration. This last characteristic differentiate ASAM design space exploration from Sesame, in fact in our case we are not forced to use some pre-existing knowledge to limit the design space exploration, but this is limited by the requirements of the application itself.

Another framework dedicated to the system design for multimedia application is the Artemis system-level workbench ([16]). It is based on the Spade framework previously described and it implements Sesame, a prototyping environment for modeling and simulation of heterogeneous multi-processor Systems-on-Chip. As in the case of Spade, Artemis exploits the separation of concerns: separation of computation from communication and of application from platform; moreover, it allows a gradual refinement of the performance estimation and the calibration of the application and platform models. The Artemis workbench follows the steps described in 3.4. A sequential application specification and an architecture template for a domain-specific (multimedia) platform should be provided as input. A KPN model is used to represents the applications: the loop parallelism inside the application (that should be specified as a parametrized static nested loop program) can be extracted by hands or through the use of a tool as Compaan that generates a KPN as output. KPN model has been chosen for its determinism (given data in input, the same application output is generated) and for capturing well the characteristics of multimedia applications. A language called YML (Y-chart Modeling Language) is used to express the application

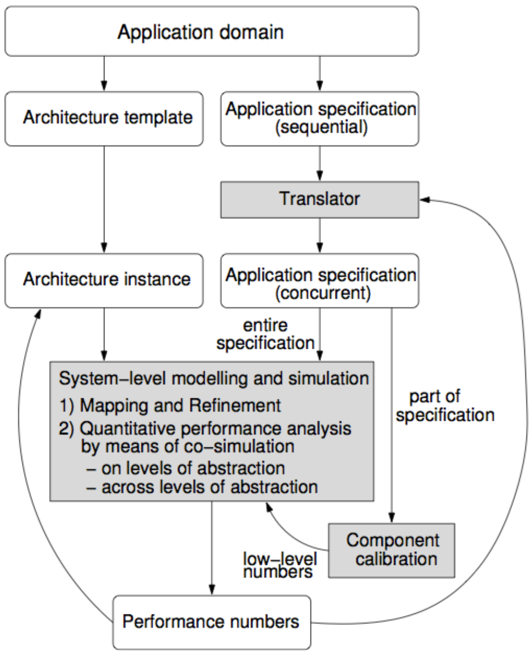
Application domain
Architecture template
Application specification (sequential)
Translator
Architecture instance
Application specification (concurrent)
entire specification
System–level modelling and simulation
1) Mapping and Refinement
2) Quantitative performance analysis by means of co–simulation
– on levels of abstraction
– across levels of abstraction
part of specification
Component calibration
low–level numbers
Performance numbers

Figure 3.4: Infrastructure of Artemis workbench (as presented in [16])

model. The architecture model is expressed as composition of generic building blocks available in a library and it is described in YML. For each component (processing and communication units and memories), a performance model is used. Starting from the architecture model, a transaction level simulator is built using either Pearl or SystemC; it simulates the behavior of the computation and communication events generated by the application. The mapping is performed through the automatic generation of a mapping layer that is composed by virtual processor components and FIFOs. Each Kahn process is associated to a virtual processor and each Kahn channel to a FIFO. According to the application model, virtual processors generate the computation and communication events that are then executed by the architecture simulator and guarantee deadlock-free scheduling of the application events. Moreover Sesame provides an initial support for performing the design space exploration of the mapping considering performance, power consumption and cost optimization. Another important aspect of Sesame is the gradual refinement of performance estimation through the transformation of the application events into architecture events that can generate more accurate simulation results.

4 Mapping estimation in ASAM Project

This section is presented in the confidential Appendix A and B.

Bibliography

[1] Abhijit Davare, Douglas Densmore, Trevor Meyerowitz, Alessandro Pinto, Alberto Sangiovanni-Vincentelli, Guang Yang, Haibo Zeng, and Qi Zhu. A next-generation design framework for platform-based design. In *DVCon 2007*, February 2007.

[2] Abhijit Davare, Qi Zhu, John Moondanos, and Alberto Sangiovanni-Vincentelli. Jpeg encoding on the intel mxp5800: A platform-based design case study. In *ESTIMedia 2005: 3rd Workshop on Embedded Systems for Real-time Multimedia*, September 2005.

[3] Nikil Dutt and Kiyoung Choi. Configurable processors for embedded computing. *Computer*, 36:120–123, January 2003.

[4] Technical University of Eindhoven (TUE) Electronic Systems. Mamps project, partitioned jpeg decoder algorithm. `http://www.es.ele.tue.nl/mamps/example.php`.

[5] J.A. Fisher, P. Faraboschi, and C. Young. *Embedded computing: a VLIW approach to architecture, compilers and tools*. Morgan Kaufmann, 2005.

[6] Joseph A. Fisher, Paolo Faraboschi, and Cliff Young. Vex, a vliw example. `http://www.hpl.hp.com/downloads/vex/`.

[7] Tero Kangas, Petri Kukkala, Heikki Orsila, Erno Salminen, Marko Hännikäinen, Timo D. Hämäläinen, Jouni Riihimäki, and Kimmo Kuusilinna. Uml-based multiprocessor soc design framework. *ACM Trans. Embed. Comput. Syst.*, 5:281–320, May 2006.

[8] Torsten Kempf, Malte Doerper, R. Leupers, G. Ascheid, and H. Meyr. A modular simulation framework for spatial and temporal task mapping onto multi-processor soc platforms. In *In Proceedings of the Design, Automation and Test in Europe (DATE) Conference*, pages 876–881. IEEE Computer Society, 2005.

[9] Bart Kienhuis, Ed F. Deprettere, Pieter van der Wolf, and Kees A. Vissers. A methodology to design programmable embedded systems - the y-chart approach. In *Embedded Processor Design Challenges*, pages 18–37, 2002.

[10] Jari Kreku, Mika Hoppari, Tuomo Kestilä, Yang Qu, Juha-Pekka Soininen, Per Andersson, and Kari Tiensyrjä. Combining uml2 application and systemc platform modelling for performance evaluation of real-time embedded systems. *EURASIP J. Embedded Syst.*, 2008:6:1–6:18, January 2008.

[11] Y. A. Li and J. K. Antonio. Estimating the execution time distribution for a task graph in a heterogeneous computing system. In *Proceedings of the 6th Heterogeneous Computing*

Workshop (HCW '97), HCW '97, pages 172–, Washington, DC, USA, 1997. IEEE Computer Society.

[12] Paul Lieverse, Pieter Van Der Wolf, Ed Deprettere, and Kees Vissers. A methodology for architecture exploration of heterogeneous signal processing systems. In *JOURNAL OF VLSI SIGNAL PROCESSING*, pages 181–190, 2001.

[13] Heinrich Meyr, Oliver Schliebusch, Andreas Wieferink, David Kammler, Ernst Witte, Olaf Lthje, Manuel Hohenauer, Gunnar Braun, and Anupam Chattopadhyay. Designing and modeling mpsoc processors and communication architectures. In Matthias Gries and Kurt Keutzer, editors, *Building ASIPS: The Mescal Methodology*, pages 229–280. Springer US, 2005.

[14] L. Micconi, G. Iordache, J. Madsen, P. Pop (DTU), P. Meloni (UNICA), M. Cocco (SH) M. Lindwer, G. Notarangelo, E. Guidetti (STM), and S.A.A. Shan (TUBS). D2.1: Initial version of generic platform model. Technical report, ASAM Project, February 2011.

[15] L. Micconi, J. Madsen, P. Pop (DTU), B. Keinhus (COMPAAN), R. Corvino, and L. Jozwiak (TUE). D2.2: Report on initial version of the hierarchical application model. Technical report, ASAM Project, May 2011.

[16] Andy D. Pimentel. The artemis workbench for system-level performance evaluation of embedded systems. *IJES*, 3(3):181–196, 2008.

[17] Seng Lin Shee and Sri Parameswaran. Design methodology for pipelined heterogeneous multiprocessor system. In *Proceedings of the 44th annual Design Automation Conference*, DAC '07, pages 811–816, New York, NY, USA, 2007. ACM.

[18] R. Corvino (TUE). Application-specific instruction identification and selection for asip; deliverable 3.2. Technical report, ASAM Project, April 2011.

[19] Andreas Wieferink, Tim Kogel, Rainer Leupers, Gerd Ascheid, and Heinrich Meyr. A system level processor/communication co-exploration methodology for multi-processor system-on-chip platforms. In *In Proc. Int. Conf. on Design, Automation and Test in Europe(DATE), February*, pages 3–11. IEEE, 2004.