Hierarchical DSE for multi-ASIP platforms

Laura Micconi*, Rosilde Corvino†, Deepak Gangadharan*, Jan Madsen*, Paul Pop* and Lech Jóźwiak†
Technical University of Denmark*, Technische Universiteit Eindhoven, The Netherlands†
E-mail: lmic@dtu.dk, r.corvino@tue.nl, dega@dtu.dk, jama@dtu.dk, paupo@dtu.dk, l.jozwiak@tue.nl

Abstract—**This work proposes a hierarchical Design Space Exploration (DSE) for the design of multi-processor platforms targeted to specific applications with strict timing and area constraints. In particular, it considers platforms integrating multiple Application Specific Instruction Set Processors (ASIPs) and each ASIP is automatically synthesized and tuned for a specific set of tasks. The definition of the platform (number of processors and their interconnection) and of the micro-architecture of each single ASIP are tightly coupled. Tasks can be allocated to the different ASIPs only knowing their performance and therefore the ASIP micro-architecture. At the same time an ASIP can be derived only knowing the functionality that it has to implement, i.e. the tasks that are assigned. We break this circular dependency with an iterative hierarchical DSE, applied at platform and micro-architecture level. We evaluate different platforms and micro-architecture alternatives to find a multi-ASIP platform targeted to the input application and able to meet the design constraints. We evaluate our design flow using a MJPEG encoder application.**

Multi-ASIP; platform synthesis; hierarchical DSE

I. INTRODUCTION

State-of-the-art embedded systems rely on heterogeneous multi-processor platforms to satisfy the requirements of highly demanding applications. In particular, they integrate Application Specific Instruction Set Processors (ASIPs), which due to their high degree of customization, offer a good trade-off between performance and cost. Designing heterogeneous multi-ASIP platforms is a complex and time-consuming task, involving inter-dependent decisions on hardware and software architectures at platform and processor levels, e.g. selection of the number of ASIPs, their interconnections and of their micro-architectures. The main issue to face in the multi-ASIP platform design is the existence of a circular dependency between the platform design and the processor design. It is impossible to define the optimized ASIP micro-architectures without knowing which tasks they execute. Conversely, it is not feasible to choose a meaningful task partitioning and mapping onto ASIPs without knowing the ASIP micro-architectures and their performances. We propose to break this circular dependency through a hierarchical design flow based on an iterative two-level DSE, i.e. a platform level DSE and a micro-architecture level DSE. In Fig. 1 a schematic of the flow is presented. The inputs are the application code, its task graph representation and the design constraints, as the maximum execution latency (also called application deadline) and the maximum system area. The output of the framework is the description of a multi-ASIP platform customized for the input application under the design constraints. Numerous platform synthesis approaches [1—4] exist, which bypass the circular dependency problem assuming that the details of processor micro-architectures and performances are known. These methods are not appropriated for system design with customizable ASIPs. Indeed, if they consider ASIPs [3, 4], they assume that the ASIPs are already synthesized, and their performances are known. Instead in [5], a small set of microarchitectural configurations is considered. These approaches severely limit the design space, discarding very good platform design solutions, because they would require a different set of ASIP microarchitectures. To the best of our knowledge, this is the first work on platform synthesis coupling the platform and ASIPs DSE for solving this circular dependency problem.

II. METHOD DESCRIPTION

A. Multi-ASIP platform

Heterogeneous bus-based multi-processor platforms may contain multiple processors (P) as general-purpose processors (GPPs), application specific integrated circuits (ASICs), digital signal processors (DSPs) or ASIPs. In this work, we focus on the case of platforms containing multiple ASIPs, which number and micro-architecture are unknown and will be defined by our exploration method. *A platform instance* contains a number of processors and a bus system. The *k-th* processor is denoted P_k and has a frequency f_{P_k}. A bus is characterized by a frequency and a bandwidth. We denote $b_w^{f_b}$, a bus with a bandwidth w and a frequency f_b. During the platform synthesis, our method explores different clustering solutions. A solution is composed of a set of clusters, i.e. group of tasks executed on the same processor. For each clustering solution evaluated, it allocates an appropriate number of ASIPs. At platform level, our exploration also accounts for different types of buses. *An ASIP* is a very long instruction set processor (VLIW), which micro-architecture depends on the clustered tasks. It can contain scalar or vector instructions and one or more issue slots (IS). Each IS is a predefined set of functional units (FU) available in a library. The ISs are connected with local memories (LM), register file(s) (RFs) and input/output FIFO ports (examples of library components are described in [6]). Depending on the functionalities required by a given ASIP, the micro-architecture DSE evaluates how the ISs, LMs and RFs should be selected and combined inside each processor. We use a non-preemptive static scheduling policy for both the execution of tasks on the platform and the execution of the tasks on a single ASIP.

The work on this paper has been performed in the scope of the ASAM project of the European ARTEMIS Research Program and has been partly supported by the ARTEMIS Joint Undertaking under grant no. 100265.

B. Application model

Similarly to our DSE method, also our application model is hierarchical and it describes the application at two different granularity levels: a higher level that specifies inter-tasks dependencies and a lower level that specifies intra-task (data-) dependencies. We assume one or more data-intensive applications A_i as input. *At the higher level*, each application is modeled as a task graph $A_i(V_i, E_i)$, where each vertex in V_i represents a task τ_j (i.e. a part of the application code) and each edge in E_i represents a data dependency. The data dependencies are modeled by *messages* $m_g \in E_i$. With the static scheduling, a task can start only after all its input messages have arrived. The task graph captures the task-level parallelism in the application. Each application has a deadline d_i and a period T_i. *At the lower level*, each data-intensive task is represented as a repetition of an elementary task, which consumes and produces multidimensional arrays piecewise. For each task, a polyhedral model is given, which describes the task iteration domain and the array access functions as presented in [7]. The higher level of the application model is used to explore the application partitioning possibilities and to address the platform design issues. The lower level is used to analyze and parallelize the parts of the application code mapped on a given processor and to address the processor design issues. The code parallelization exploits techniques of loop transformations, which are based on the polyhedral model. In our design flow, all the information needed to generate the hierarchical application model is inferred from an input C code of the application analyzed through Compaan[1]. Compaan generates a Kahn Process Network (KPN) model of the application, which includes the information needed to construct our input application task graph and the polyhedral model of each task.

C. Hierarchical design flow

The proposed flow for multi-ASIP platform design and synthesis involves two phases (cf. Fig. 1): a platform DSE and a micro-architecture DSE.

The platform DSE explores different task clustering solutions resulting in alternative platform implementations with different sets of ASIPs and different bus system instances. This phase is in charge of taking system level design decisions. It is divided in two sub-phases, a probabilistic and a deterministic DSE, which work with different levels of information. Given the initial task graph, the probabilistic DSE makes a first educated guess on the task clustering, which only considers the execution time of different tasks as provided by an application analysis. The probabilistic DSE provides the micro-architecture DSE with a first set of clustering solutions. The micro-architecture DSE analyzes a cluster of tasks at a time. For each cluster of tasks, it explores different application optimizations, as node (or loop) fusion and vectorization. For each optimized version of the application, the micro-architecture DSE allocates a corresponding ASIP micro-architecture, including issue slots, local memories and register files, and it estimates its performance and cost. A Pareto front is selected and the performance and cost indicators of these Pareto solutions are given to the deterministic DSE for a further analysis. The deterministic DSE collects the Pareto solutions for all the clusters. It evaluates and selects the best combination of the Pareto fronts, selecting a single (final) Pareto solution for each cluster. Finally, the micro-architecture DSE generates the ASIP architecture descriptions and the optimized C codes for these (final) Pareto solutions.

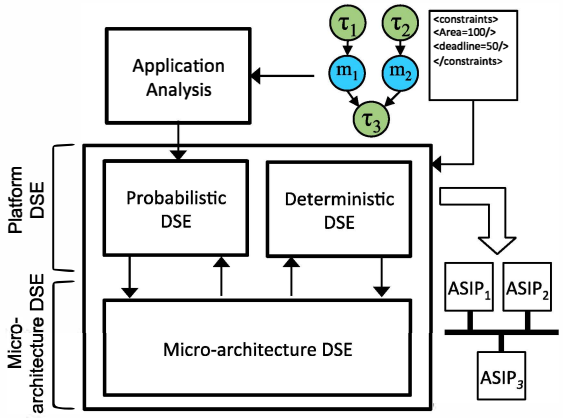

Figure 1. Schematic of the hierarchical design flow

In the rest of this section we describe the different phases of our flow.

Application Analysis. The application analysis calculates the lower and upper bounds of the WCET of all the application tasks. The lower WCET bound is the expected execution time when the application task is executed on a virtual processor according to an as soon as possible (ASAP) scheduling without architectural constraints. The upper WCET bound is the expected execution time when the application task is executed on a scalar processor. The application analysis is realized as a pass of the LLVM compiler and it computes the WCET compiling and running a binary code with real profile data or estimated profile data.

Probabilistic DSE. The probabilistic DSE is executed in the early stages of the flow when there is no information about the platform and its composition. The only inputs available are the task graphs of one or more applications and their design constraints (both in a XML format). The purpose of this DSE stage is to identify which tasks should be clustered on the same ASIP without a precise knowledge of the underlying HW (ASIPs and their interconnections). Therefore, the outputs obtained from this DSE stage represents a set of clustering solutions that have a high chance to meet the design constraints when continuing through the different successive phases of the ASAM flow (micro-DSE, communication-DSE).

As the platform composition and the detailed micro-architecture of the ASIPs is not known, we do not have accurate information about the worst-case execution time (WCET) of each task. Therefore, it is not possible to perform a precise static schedulability analysis of each application to verify if the deadline is met. In order to override this problem, we model the WCET of each task as a stochastic variable that captures a range of performances corresponding to a number of micro-architectural configurations. More precisely, the

[1] "Compaan compiler, http://www.compaandesign.com/."

WCET bounds provided by the *Application Analysis* stage for each task are used to construct a Cumulative Distribution Function (CDF) for WCET as described in [8]. As the WCET of each task could assume any value in the performance range modeled using the CDF, the performance of any clustering solution is quantified as the probability of the application meeting the deadline. In order to obtain the clustering solutions that give the best chance of meeting application deadlines, we use an Evolutionary Algorithm (EA) along with Monte Carlo Simulation (MCS). The EA searches for the various clusters. For each of these clusters, MCS is used to find its performance for a range of micro-architectural configurations modeled using the WCET CDF for each task. The outputs of the probabilistic DSE are: set of clustering solutions (one or more) with a high probability of meeting the deadline. A certain number of clustering solutions with a high chance of meeting the deadline are then provided to the micro-architecture DSE stage for ASIP optimization. The initial platform is a multi-ASIP bus-based system (the communication optimization is part of the future work and will consider also NoC).

Micro-architecture DSE. The micro-architecture DSE performs a data-oriented software and hardware co-design to decide the ASIP parallel processing, communication and storage architectures. It takes as input the C code of the nodes clustered by the platform DSE. It explores and selects possible parallel software structures and allocates the corresponding hardware architectures. The micro-architecture DSE is built on an internal data-oriented polyhedral representation of the application [7], which is used to rapidly evaluate the proposed software and the corresponding hardware architectures. In order to select Pareto parallel versions of the loop-based code and infer from this the corresponding ASIP architectures, the micro-architecture DSE evaluates the improvements of the code execution due to selected loop transformations. This evaluation is a result of a static analysis performed on the internal data-oriented representation. The evaluation also accounts for the performances of the allocated ASIP architecture through an analytical model of its area and execution time. Subsequently, using several established allocation and mapping rules, the micro-architecture DSE infers the ASIP architecture from the internal data-oriented representation. In particular, it decides for memory hierarchy and data parallelism through vectorization or usage of multiple issue slots.

To better understand the allocation and mapping rules, let us consider some examples. A transformation such as loop unrolling can be used to identify possibility for vectorization. Indeed, the unrolling of independent loop iterations identifies identical kernels (i.e. independent iterations of a same loop body), which can be vectorized, provided that the necessary vector instructions are allocated. As a consequence, the architecture design should allocate standard or custom issue slots containing vector instructions to realize the foreseen vectorization. Another example of allocation rules can be given by loop fusion. Loop fusion merges two or more loops in a same iteration space, reducing the loop control to a single thread. As a consequence, it is possible to process the tasks of two or more merged loops in parallel on the same ASIP. To make this possible, the architecture design should allocate corresponding parallel issue slots, register files and local memories, to execute in parallel the loop bodies of the merged loops. In order to rapidly evaluate and select the solutions of this exploration, the micro-architecture DSE uses an analytical model. This technique avoids the error-prone and time-consuming process that actually constructs all the considered architectures. In particular, the analytical model estimates the execution time of the proposed software structure onto the proposed architecture; it estimates the area of the proposed solution; and, in a future version, it will estimate the power consumption. An exploration process based on the Opt4J [9] genetic algorithm framework creates and evaluates a large number of possibilities. Few solutions with the best area and execution time trade-off are selected to be communicated to the platform DSE for an exploration of the Pareto front combinations per cluster. For each final Pareto solution selected by the platform DSE, the micro-architecture DSE generates two outputs: 1) The optimized C code of the application part. 2) An XML file used to generate the ASIP architecture from the library components.

Deterministic DSE. Using the execution trace and area values for the Pareto solutions identified by the micro-architecture DSE, the deterministic DSE identifies the best combination of micro-architecture configurations that allows the system to meet the design constraints on total area and performance. If this is not possible, then the Deterministic DSE has to identify which specific cluster needs to be changed, i.e. which task should be re-mapped to a different ASIP. Each change implies a new iteration with the micro DSE, which could consume a substantial amount of time. Therefore, Deterministic DSE has to take care that the number of changes required is minimized. However, the technique used to minimize the number of iterations of micro-architecture DSE is out of this paper scope. The deterministic DSE uses a multi-objective EA in order to obtain a Pareto-front of ASIP configurations that optimize for multiple objectives of application deadline and power. Once an acceptable platform implementation has been defined and the input constraints are guaranteed, it is possible to synthesize the entire multi-ASIP platform.

III. EXPERIMENTAL EVALUATION

To demonstrate our hierarchical design method we use a motion JPEG (MJPEG) encoder application provided with the Compaan tool. Compaan tool analyses the C code of the application and generates the corresponding KPN in which the code is partitioned into multiple tasks (or nodes) as shown in Fig. 2a. From the KPN we can easily generate a task graph (Fig. 2b) as the one required in input by our method. The constraints provided as input are the application deadline, $d = 5000\,ms$ and the maximum area of the platform, $a_p = 2 * 10\ \mu m^2$. Moreover we consider two different types of buses, b_{32}^{10} and b_{16}^{10} (32 and 16 bits width with a frequency of 10 MHz) during the hierarchical DSE. The C code of each

single task τ_j is elaborated by the application analysis phase that generates the upper and lower WCETs values, respectively indicated as $WCET_j^u$ and $WCET_j^l$. TABLE I. presents the values obtained for each task and normalized with respect to the $WCET_{\tau_1}^l$ (the smallest value). For the ASIPs, we assume a working frequency of 100 MHz that is compatible with the available components of the micro-architecture library.

The probabilistic DSE is executed and, from the set of clustering solutions with the highest probability of meeting the deadline d, we select the clustering solution with the minimum number of clusters (i.e. two) and the smaller bus (b_{16}^{10}). The selected clustering solution is described in TABLE II. Given this clustering solution, the micro-architecture DSE generates two ASIPs customized to the specific task clusters. For each ASIP, depending on the available optimizations, multiple Pareto micro-architecture solutions are generated: in particular, 1 solution for P_1 (that allows a limited optimization as it contains only one task) and 5 solutions for P_2. The start and end points of the tasks execution time for the Pareto solutions are estimated together with the areas values. The error of the start and end times estimation is still work in progress, but in this paper we wanted to give a first proof of concept. Then, the deterministic DSE selects between the available Pareto solutions evaluating the area and performances of the entire system. The results obtained are shown in Fig. 3. For the given input constraints (red lines in Fig. 3), there are two acceptable solutions. For the final implementation, we selected the one with the smaller WCET. With this case study we demonstrate that our design flow can generate a heterogeneous bus-based multi-ASIP system targeted to a specific application, which meet the design constraints.

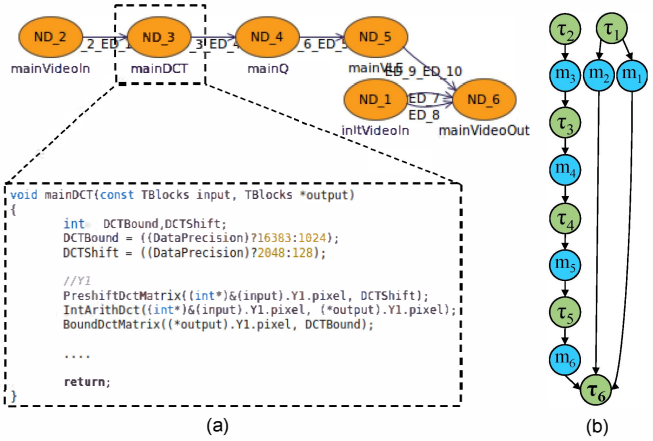

Figure 2. KPN of the MJPEG encoder (a) and corresponding task graph (b)

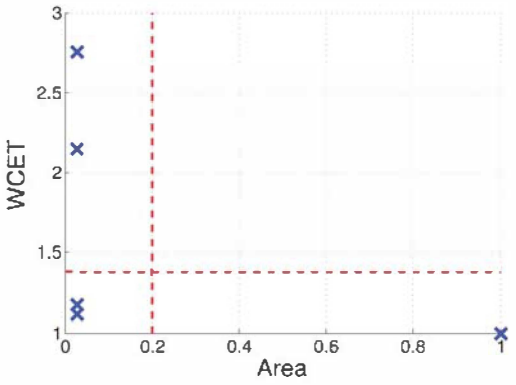

Figure 3. Normalized output of the deterministic DSE

TABLE I. NORMALIZED OUTPUT OF THE APPLICATION ANALYSIS

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6
$WCET_j^u$	1.4	83.5	1541	1746	790.1	26.2
$WCET_j^l$	1	45	1151	83.3	540.5	22.1

TABLE II. SELECTED CLUSTERING SOLUTION

P_1	P_2
τ_2	$\tau_1, \tau_3, \tau_4, \tau_5, \tau_6$

IV. CONCLUSION

This paper proposes a hierarchical Design Space Exploration (DSE) method to address the circular dependency problem in ASIP-based system design and to produce efficient designs of heterogeneous multi-ASIP platforms given an application task graph and design constrains as input specification. The method is demonstrated with the MJPEG case study.

REFERENCES

[1] T. Kangas *et al.*, "UML-based multiprocessor SoC design framework," *ACM Transanctions on Embedded Computing Systems*, vol. 5, pp. 281–320, 2006.

[2] H. Nikolov, T. Stefanov, and E. Deprettere, "Systematic and automated multiprocessor system design, programming, and implementation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 3, pp. 542–555, 2008.

[3] O. Muller, A. Baghdadi, and M. Jézéquel, "From parallelism levels to a multi-ASIP architecture for turbo decoding," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 17, no. 1, pp. 92–102, 2009.

[4] C. Brehm, T. Ilnseher, and N. Wehn, "A scalable multi-ASIP architecture for standard compliant trellis decoding," in *Proc. International SoC Design Conference*, 2011, pp. 349–352.

[5] S. L. Shee and S. Parameswaran, "Design methodology for pipelined heterogeneous multiprocessor system," in *Proc. 44th Design Automation Conference*, 2007, pp. 811–816.

[6] Tim 2 language, http://www.es.ele.tue.nl/~heco/courses/platformdesign 2008/tim2_guide.pdf.

[7] C. Glitia *et al.*, "Repetitive model refactoring strategy for the design space exploration of intensive signal processing applications," *Journal of Systems Architecture*, vol. 57, pp. 815 – 829, Oct. 2011.

[8] L. Micconi, D. Gangadharan, P. Pop, and J. Madsen, "Multi-ASIP platform synthesis for real-time applications," in *Proc. Symposium on Industrial Embedded Systems*, June 2013.

[9] M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich, "Opt4J - A Modular Framework for Meta-heuristic Optimization," in *Proc. of the Genetic and Evolutionary Computing Conference*, 2011