Daedalus: Toward Composable Multimedia MP-SoC Design

H. Nikolov[†], M. Thompson[‡], T. Stefanov[†], A. Pimentel[‡],
S. Polstra[‡], R. Bose[⋆], C. Zissulescu[⋆], E. Deprettere[†]

[†]Leiden Embedded Research Center
Leiden University
The Netherlands
{nikolov,stefanov}@liacs.nl

[‡]Department of Computer Science
University of Amsterdam
The Netherlands
{mthompsn,andy}@science.uva.nl

[⋆]Chess B.V.
Image Processing Solutions
The Netherlands
{raj.bose,czi}@chess.nl

ABSTRACT

Daedalus is a system-level design flow for the design of multiprocessor system-on-chip (MP-SoC) based embedded multimedia systems. It offers a fully integrated tool-flow in which design space exploration (DSE), system-level synthesis, application mapping, and system prototyping of MP-SoCs are highly automated. In this paper, we describe our first industrial deployment experiences with the Daedalus framework. Daedalus is currently being deployed in the early stages of the design of an image compression system for very high resolution cameras targeting medical appliances. In this context, we performed a DSE study with a JPEG encoder application, which exploits both task and data parallelism. This application was mapped onto a range of different MP-SoC architectures. We achieved a performance speed-up of up to 20x compared to a single processor system. In addition, the results show that the Daedalus high-level MP-SoC models accurately predict the overall system performance, i.e., the performance error is around 5%.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-based Systems**]: Real-time and embedded systems; C.4 [**Performance of Systems**]: Modeling techniques; J.6 [**Computer-aided Engineering**]: Computer-aided design (CAD)

General Terms

Design, Performance

Keywords

System-level design and synthesis, Design space exploration

1. INTRODUCTION

The complexity of modern embedded systems, which are increasingly based on MultiProcessor-SoC (MP-SoC) architectures, has led to the emergence of system-level design. To cope with this design complexity, system-level design aims at raising the abstraction level of the design process. Key enablers to this end are, for example, the use of architectural platforms to facilitate re-use of IP components and the notion of high-level system modeling and simulation [6]. System-level design for MP-SoC-based embedded systems however still involves a substantial number of challenging design tasks. For example, applications need to be decomposed into parallel specifications so that they can be mapped onto an MP-SoC architecture [9]. Subsequently, applications need to be partitioned into HW and SW parts since MP-SoC architectures often are heterogeneous in nature. To this end, MP-SoC platform architectures need to be modeled and simulated to study system behavior and to evaluate a variety of different design options. Once a good candidate architecture has been found, it needs to be synthesized, which involves the synthesis of its architectural components as well as the mapping of applications onto the architecture. To accomplish all of these tasks, a range of different tools and tool-flows is often needed, potentially leaving designers with all kinds of interoperability problems. Moreover, there typically remains a large gap between the deployed system-level specifications (or models) and actual implementations of the system under study, known as the *implementation gap* [10]. Currently, there exist no mature methodologies, techniques, and tools to effectively and efficiently convert system-level MP-SoC specifications to RTL specifications.

Recently, we presented our Daedalus system-level design framework which addresses the above design challenges [17, 1]. The entire Daedalus framework has been developed as high-quality software distributed under Open Source licenses and can be downloaded from [1]. Daedalus main objective is to bridge the aforementioned implementation gap for the design of multimedia MP-SoCs. It does so by providing an integrated and highly-automated environment for system-level architectural exploration, system-level synthesis, programming, and prototyping. The Daedalus design flow, which leads the designer from a sequential application to an MP-SoC system implementation on an FPGA with a parallelized application mapped onto it, can be traversed in only a matter of hours. Evidently, this offers great potentials for quickly experimenting with different MP-SoCs and exploring design options during the early stages of design. In this paper, we report on our first deployment experiences with the Daedalus framework. Daedalus is currently being deployed in a project together with the Dutch SME Chess B.V., which involves the design of an image compression system for very high resolution (in the order of Gigapixels) cameras targeting medical appliances. In this project, the Daedalus framework is used for design space exploration (DSE), both at the level of simulations and prototypes, in order to rapidly gain detailed insight on the system performance. To this end, we present initial results from a DSE study we performed with a JPEG encoder application, which exploits both task and data parallelism and which is mapped onto a range of different MP-SoC architectures.

1.1 Related Work

Systematic and automated application-to-architecture mapping has been widely studied in the research community. The closest to

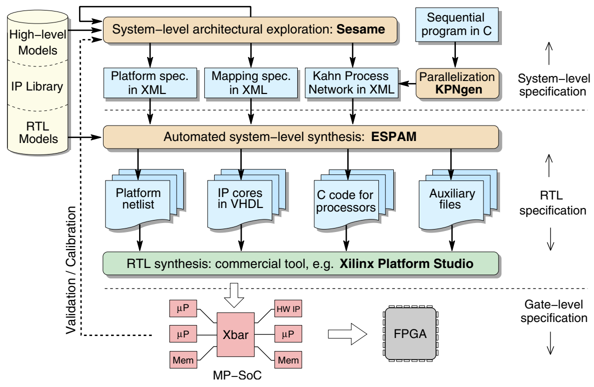

Figure 1: The Daedalus system-level design framework.

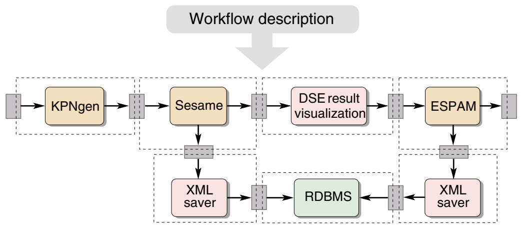

Figure 2: Daedalus' customizable work flow.

our work is the Koski MP-SoC design flow [15] and the SystemC-based design methodology presented in [5]. Koski provides a single infrastructure for modeling of applications, automatic architectural design space exploration, and automatic system-level synthesis, programming, and prototyping of selected MP-SoCs. The methodology in [5] supports automated design space exploration, performance evaluation, and automatic platform based system generation. But unlike Daedalus, [15] and [5] do not allow for automated parallelization of applications, nor design space exploration at application level. Both [15] and [5] require applications to be specified by hand in UML and SystemC, respectively.

Other examples of related work can be found in [16, 8, 2, 4]. However, these efforts are limited to processor-coprocesor architectures [16], only provide a limited degree of automation [8, 2], or do not provide an automated step towards RTL [4].

Companies such as Xilinx and Altera provide design tool chains attempting to generate efficient implementations starting from descriptions higher than (but still related to) the register transfer level of abstraction. The required input specifications are still so detailed that designing a single processor system is still error-prone and time consuming, let alone designing alternative multiprocessor systems. In contrast, Daedalus raises the design to an even higher level of abstraction allowing the exploration, design, and programming of multiprocessor systems in a short amount of time.

The next section provides a birds-eye overview of the Daedalus design flow with its three core tools. Section 3 describes Daedalus supporting infrastructure that improves the user-friendliness, and therefore also the deployability, of the framework. In Section 4, we present the results from a DSE study with the aforementioned JPEG encoder application mapped to a range of different MP-SoCs. Section 5 concludes the paper.

2. THE DAEDALUS DESIGN FLOW

In Figure 1, the conceptual design flow of the Daedalus framework is depicted. As mentioned before, Daedalus provides a single environment for rapid system-level architectural exploration, high-level synthesis, programming, and prototyping of multimedia MP-SoC architectures. Here, a key assumption is that the MP-SoCs are constructed from a library of pre-defined and pre-verified IP components. These components include a variety of programmable and dedicated processors, memories and interconnects, thereby allowing the implementation of a wide range of MP-SoC platforms. So, this means that Daedalus aims at *composable MP-SoC design*, in which MP-SoCs are strictly composed of IP library components.

Starting from a sequential multimedia application specification in C, the KPNgen tool [18] allows for automatically converting the sequential application into a parallel Kahn Process Network (KPN) [7] specification. Here, the sequential input specifications are restricted to so-called static affine nested loop programs, which is an important class of programs in, e.g., the scientific and multimedia application domains.

The generated or handcrafted KPNs (the latter in the case that, e.g., the input specification did not entirely meet the requirements of the KPNgen tool) are subsequently used by our Sesame modeling and simulation environment [13, 3] to perform system-level architectural design space exploration. To this end, Sesame uses (high-level) architecture model components from the IP component library (see the left part of Figure 1). Sesame allows for quickly evaluating the performance of different application to architecture mappings, HW/SW partitionings, and target platform architectures. Such exploration should result in a number of promising candidate system designs, of which their specifications (system-level platform description, application-architecture mapping description, and application description) act as input to the ESPAM tool [12, 11]. This tool uses these system-level input specifications, together with RTL versions of the components from the IP library, to automatically generate synthesizable VHDL that implements the candidate MP-SoC platform architecture. In addition, it also generates the C code for those application processes that are mapped onto programmable cores. Using commercial synthesis tools and compilers, this implementation can be readily mapped onto an FPGA for prototyping. Such prototyping also allows for calibrating and validating Sesame's system-level models, and as a consequence, improving the trustworthiness of these models.

3. THE DAEDALUS INFRASTRUCTURE

As discussed in the previous section, the heart of Daedalus consist of the three core tools KPNgen, Sesame and ESPAM. In addition, Daedalus also features several supporting tools to improve the user-friendliness and deployability of the framework. This section provides a brief overview of the supporting infrastructure.

In Daedalus, most design information (e.g., structural descriptions of the application, architecture, and the mapping of the former onto the latter) as well as experimental results are described using XML-based descriptions. Daedalus therefore contains the Oracle Berkeley DB XML relational database management system (RDBMS) to store all information (models, parameters and results) related to designs and experiments. This RDBMS, together with its GUI, provide the designer with a powerful tool to e.g., explore and visualize the large amounts of data generated by Daedalus design space exploration. Moreover, it guarantees the reproducibility of experiments at all times.

The vision behind the Daedalus software infrastructure is that it should be open for integration of new tools as well as that it should allow for customization of the design flow. Therefore, the design flow (or tool flow) in Daedalus is composable and constructed from 'design-flow blocks'. These design-flow blocks, which are illustrated as the dashed boxes in Figure 2, are the tools that take part in the design flow together with their input- and output descriptions. The latter descriptions, illustrated by the grey boxes in Figure 2, provide information about what input/output data a tool consumes/produces and from/to where it reads/writes this data. This allows us to describe a design flow as a simple composition of the design-flow blocks, specified in the *workflow description*. For example, Figure 2 shows a design flow which includes a visualization block to visualize Sesame's DSE results and which stores both the DSE and ESPAM's prototyping results in the RDBMS (using the so-called 'XML saver' tool). Evidently, this composability of the design flow allows for easily adding new design steps to a design flow, or to customize design flows for specific design domains.

We have also developed control and monitoring software utilities to facilitate the process of setting up and executing experiments on the FPGA-based prototypes of MP-SoCs generated by Daedalus. Such utilities are necessary and very useful for: (i) conducting an effective and efficient design space exploration at implementation level on a narrow design space defined by Sesame; (ii) measuring real performance and cost numbers used for calibration of the Daedalus' high-level architecture models [14]; (iii) preparing real HW/SW demonstrators. The control and monitoring utilities include a configuration manager, an execution control panel, and an on-line monitoring console, all supported by a GUI which allows users unfamiliar with the FPGA prototyping board to perform experiments with the MP-SoCs.

4. PUTTING DAEDALUS TO WORK

We have initiated a project together with a Dutch SME called Chess B.V. (www.chess.nl), which involves the design of a still image compression system for very high resolution images. Chess B.V. is a company that provides image processing solutions for customers that build industrial process monitoring and medical appliances. With respect to this, the still image compression systems for different customers have to meet different performance and cost requirements. Chess needs tool support for very fast exploration and implementation of alternative systems (e.g., MP-SoCs realizing JPEG or JPEG2000 encoders) whereby trade-offs can be made between cost, design time, space, performance, etc. in order to offer its customers several solutions at different prices and let the customers select the most suitable ones. The Daedalus framework provides such tool support for MP-SoC design. Therefore, it is used in a project with Chess for design space exploration (DSE) at a high-level of abstraction by running system simulations and at implementation level by evaluating real system prototypes. In this section, we report on the project's initial findings and results obtained by deploying the Daedalus framework in the early design stage of JPEG-based image compression MP-SoCs.

Before describing our DSE experiments, we first would like to point out that the design space targeted by our implementation-level DSE is currently constrained by:

1) *The amount of the available memory.* In order to achieve high performance, in our MP-SoCs we use on-chip memory for processors' program and data segments, including buffers for inter-processor data communication. We do not consider using external (off-chip) memory because of its large latency compared to the on-chip memory. Moreover, usually there is a limited number of available external memory banks which requires the external memory to be shared between several processors. This fact significantly limits the overall MP-SoC performance. We use external memories only for communication with the environment (source of data and destination of the generated results). An average size FPGA nowadays has around 200–300KB of on-chip memory distributed on several blocks. In our experiments, we use a Xilinx VirtexII-6000 FPGA,

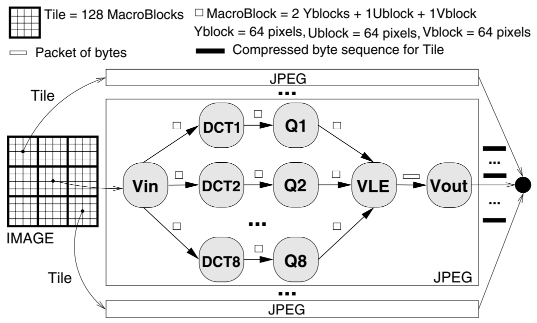
Tile = 128 MacroBlocks
MacroBlock = 2 Yblocks + 1Ublock + 1Vblock
Yblock = 64 pixels, Ublock = 64 pixels, Vblock = 64 pixels
Packet of bytes
Compressed byte sequence for Tile
Tile
JPEG
IMAGE
Vin
DCT1
DCT2
DCT8
Q1
Q2
Q8
VLE
Vout
Tile
JPEG
JPEG

Figure 3: The JPEG application KPN.

and therefore, we constrain the total MP-SoC memory to be up to 288KB, being the amount of on-chip memory of this FPGA.

2) *The type of the processing components.* The MP-SoCs are built of components from our library. Our library is under development and currently contains two programmable processors: *PowerPC* 405 (IBM) and *MicroBlaze* (Xilinx). In addition, the library contains several dedicated HW IP cores. However, for the JPEG encoder we can use only one, i.e., the Discrete Cosine Transform (*DCT*) IP. For the considered FPGA, *PowerPC* processors can not be used. Therefore, the processing components of the MP-SoCs are limited to *MicroBlaze* processors and *DCT* HW IP cores only.

4.1 Simulation-level DSE

In our experiments, we assume that the image that needs to be compressed is tiled, and that multiple JPEG encoders can process these tiles in parallel. This is illustrated in Figure 3, which also shows the applied KPN application specification for a JPEG encoder. The JPEG KPN for a single tile can again exploit task-level parallelism by pipelining tasks as well as data-parallelism by performing multiple *DCT*s and *Quantizations* in parallel. By deploying Sesame's efficient system-level simulations, we explored a substantial number of different implementations of a single JPEG encoder in the system, as represented by the KPN in Figure 3. To this end, we mapped the KPN to a variety of MP-SoC architectures, ranging from a 1-processor system (all KPN processes mapped to a single processor) to a 19-processor system (every process in a KPN with 8 data-parallel streams mapped to a different processor). Moreover, in our Sesame-based exploration we also varied the type of processors in these MP-SoC platforms: KPN tasks can be executed on a *MicroBlaze*, while for the *DCT*, *Q*(uantization) and *VLE* tasks we also assessed dedicated HW IP implementations. Evidently, our simulation-level DSE also explores *'non-implementable' design instances*. That is, design instances that cannot be further explored at implementation level since it uses HW IP components

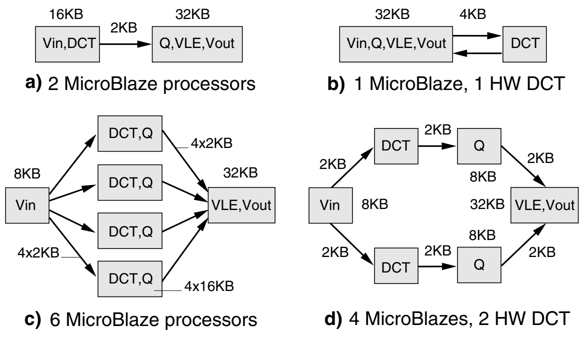

Figure 4: Alternative design instances to process one tile.

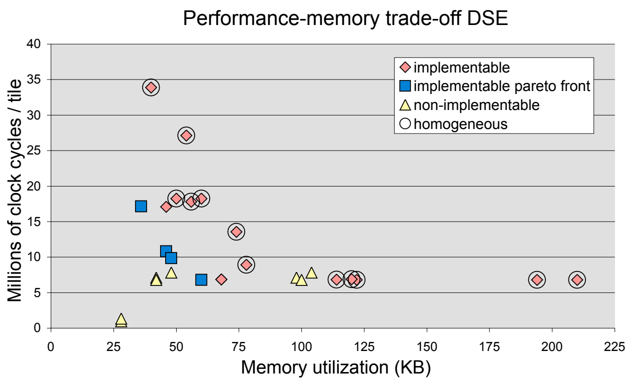

Figure 5: DSE for performance/memory utilization trade-offs.

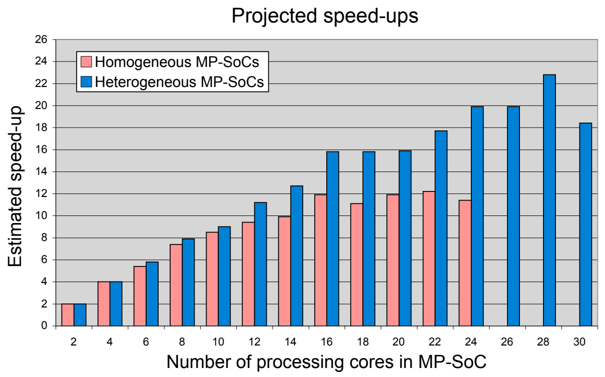

Figure 6: Estimated speed-ups.

that are not (yet) available in our library of RTL-level IP components. In Figure 4, four (implementable) example design instances are depicted.

Figure 5 shows a scatter plot with the performance results of the explored design instances plotted against the expected memory utilization of each design instance once implemented on the targeted FPGA. The memory utilization of the design instances was estimated using a simple accumulative model that has been calibrated with numbers from implementation-level experiments (see the next section). Since the memory utilization of all design points in Figure 5 is below 288KB, they will all fit on the targeted FPGA memory-wise. But, as will be shown further on, the real system will consist of a combination of multiple of these (single JPEG encoder) design instances working in parallel, which, of course, may not necessarily fit on the FPGA. The points in Figure 5 can be classified as three types of design instances: 1) design instances that are 'implementable' (i.e., do not use the non-implemented HW IP components for the Q and VLE tasks) but are not part of the Pareto front, 2) implementable design instances that are part of the Pareto front, and 3) design instances that are non-implementable (i.e., contain HW IP components for the Q or VLE tasks). Moreover, the homogeneous design instances (i.e., the platforms only using *MicroBlaze* processors) are tagged with circles.

A number of observations can be made from Figure 5. The (implementable) Pareto optimal solutions are all heterogeneous designs, containing one or two DCT HW IP components. Two of these Pareto optimal solutions are shown in Figures 4(b) and 4(d). Clearly, the (non-implementable) design instance in which the DCT, Q and VLE tasks are all implemented by a HW IP core is the fastest and most memory efficient. When considering the homogeneous design points in Figure 5, another observation can be made: The design points with a memory utilization less than 75KB are the designs that exploit task-level parallelism only. The speed-up due to task-level parallelism levels off at a performance of around 18 Mcycles/tile. But, when data-parallelism is also exploited, the speed-up levels off at around 7 Mcycles/tile at the cost of increased memory utilization. Here, we found that increasing data-parallelism beyond 4 parallel DCT-Q streams (see Figure 3) does not improve performance anymore as the VLE becomes the bottleneck.

As mentioned and as will be illustrated in the next section, the design points in Figure 5 are the building blocks for the entire system, in which multiple of these instances, possibly in a hybrid constellation, are encoding image tiles in parallel. For example, the most optimal, but (currently) non-implementable, system would consist of multiple JPEG encoders with HW IPs for the DCT, Q and VLE tasks. The projected performance of this system, considering the targeted FPGA, equals to an execution time of about 6 Giga cycles to encode an image with a 1 Gigapixel resolution. For implementable solutions, the Pareto optimal design instances from Figure 5 are obvious candidate building blocks for the MP-SoC.

Figure 6 shows the estimated maximum performance – in terms of speed-up over a single JPEG encoder executed on one *MicroBlaze* – for different JPEG compression MP-SoCs realized with a combination of implementable design instances from Figure 5. The x-axis indicates the number of processing cores (either *MicroBlaze* or HW IP) in the MP-SoC, and the y-axis shows the estimated speed-up for the optimal combination of design instances for a specific number of cores in the MP-SoC which still adheres to the memory constraints of the targeted FPGA. Furthermore, a distinction is made between homogeneous systems (i.e., only *MicroBlazes*) and heterogeneous systems (i.e., containing DCT HW IP components). For example, the optimal homogeneous 4-core system is a combination of four sequential JPEG design instances, i.e. a system containing four *MicroBlazes* that all perform a full JPEG on different image tiles in parallel. In the next section, more examples of, sometimes hybrid, combinations of design instances will be discussed.

Essentially, Figure 6 provides a projection of the feasible system performance, given the constraints of the targeted FPGA. For homogeneous solutions, our simulations predict that a speed-up of around 10 to 12 is attainable. Our memory utilization model indicates that scaling the homogeneous system beyond 24 cores is not possible because of the memory constraints. For heterogeneous systems, on the other hand, our memory model indicates that the system can be scaled to 30 cores since the HW IP components only use a fraction of the memory used by a *MicroBlaze*. Here, our predictions show that a speed-up of around 20 to 22 is feasible. The results from our simulation-level DSE, as displayed in Figures 5 and 6, are used in the next section for steering the implementation-level DSE. These implementation-level experiments will also provide a validation of our simulation-based predictions.

4.2 Implementation-level DSE

Performing DSE at a high-level of abstraction by simulation can not deliver 100% accurate performance/cost numbers but it can rapidly narrow down the design space to a few promising design points. Thus, we perform 100% accurate exploration in the narrowed design space by generating real MP-SoC prototypes and we measure the actual performance/cost in order to select the optimal MP-SoC designs given a set of physical implementation constraints. Below, we present our initial implementation-level DSE results for MP-SoCs implemented on Xilinx FPGAs.

Due to the aforementioned implementation-level constraints, some of the best design points found by the simulation-level DSE (see

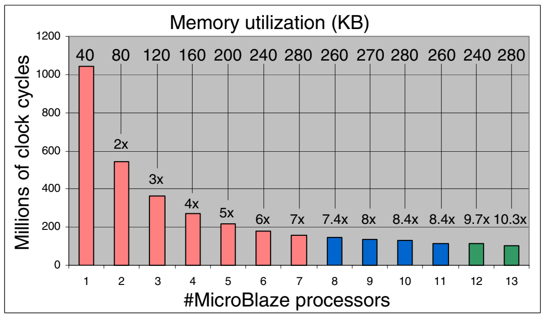

Figure 7: Performance results: homogeneous MP-SoCs.

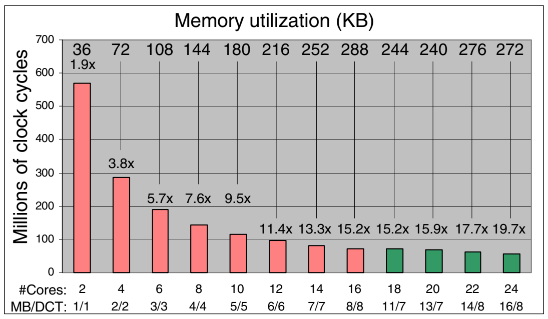

Figure 8: Performance results: heterogeneous MP-SoCs.

Figure 5) could not be implemented, e.g., all application tasks to be realized as HW IPs. Therefore, we considered the implementable design instances depicted in Figure 6. From them, we selected only the instances that have *efficiency* above 0.8, where

$$efficiency = \frac{speed-up}{number\ of\ cores}.$$

This selection resulted in implementations of homogeneous MP-SoCs with up to 13 *MicroBlaze* processors and heterogeneous MP-SoCs with up to 24 cores. Evidently, better performance is delivered by the heterogeneous systems, however, the homogeneous systems add more flexibility in choosing the right solution, e.g., performance/cost, for a particular customer.

The implementation results for the homogeneous MP-SoCs are depicted in Figure 7. The x-axis represents the number of *MicroBlaze* processors in an MP-SoC and the y-axis depicts the number of clock cycles (in millions) to compress one image consisting of 32 tiles. Above each bar, we indicated the achieved speed-up of the particular MP-SoC compared to a single *MicroBlaze* system (the leftmost bar). At the top of the figure, we present the amount of memory utilized by each MP-SoC.

As mentioned before, our JPEG encoding MP-SoCs process the input image in tiles. We started with a single *MicroBlaze* system (processing all the tiles) and then we increased the number of processors by selecting the best points found by the simulation-level DSE. These points exploit data-level parallelism, i.e., several *MicroBlazes* process different tiles. This is the most efficient way to increase performance because if we increase the number of processors that process independent tiles, then the speed-up increases linearly with the number of processors. To execute the JPEG application, a single *MicroBlaze* processor system requires 40KB of memory. Therefore, we were able to implement systems with up to 7 processors on the considered FPGA (7x40=280KB), achieving speed-ups (see the first 7 bars in the Figure 7) up to 7x.

By exploiting only data-level parallelism, with 7 *MicroBlaze*s processing 7 tiles in parallel, we reached the limit of the available memory in our FPGA. Then, the question is whether there are design points that give even better performance (with more processors) and still match the resource constraints. We were able to increase the number of processors to more than 7 by selecting points that exploit both data-level parallelism between tiles and also task-level parallelism within the tiles. For this purpose, we used the 2-*MicroBlaze* architecture depicted in Figure 4(a), where the *Vin* and all *DCT* processes (see Figure 3) are executed on the first processor and the remaining processes on the second one. By exploiting task-level parallelism, reaching linear speed-up is not possible due to data dependencies between the tasks. However, the total memory requirement of the system is reduced because the application tasks are distributed, and each processor executes a portion of the initial application. As a result, larger systems can be built, and consequently, larger overall speed-up can be achieved. For instance, a single-processor system needs 40K to execute the JPEG encoder, while a two-processor system – exploiting task-level parallelism – needs a total amount of 50KB for the same application, on average 25KB per processor. Thus, by exploiting the reduced memory requirement, we were able to increase the number of processors and to implement systems with up to 11 *MicroBlaze* processors. The selected points are actually combinations of a 1-*MicroBlaze* system per tile and a 2-*MicroBlaze* system per tile. The MP-SoCs with 8 to 11 *MicroBlazes* process 6 tiles in parallel. The achieved speed-ups are not linear, e.g., 7.4x for an 8-processor MP-SoC and 8.4x for an 11-processor MP-SoC, but they are higher than the speed-up of the 7-processor system.

In order to implement even larger systems, we exploited data-level parallelism between the tiles and data- and task-level parallelism within the tiles. We selected and implemented points representing 12 and 13 processor systems with total memory requirements that match our physical constraints. The 12-processor system processes 2 tiles in parallel where each tile is processed by a 6-*MicroBlaze* architecture depicted in Figure 4(c). This architecture requires 120KB of memory. The 13-processor MP-SoC utilizes an additional *MicroBlaze* processor (additional 40KB), therefore, increasing the number of tiles processed in parallel to 3. The results are shown at the right part (the two rightmost bars) of Figure 7. The achieved speed-up of 12- and 13-*MicroBlaze* systems is 9.7x and 10.3x respectively, compared to a 1-*MicroBlaze* system.

The implementation results for the heterogeneous MP-SoCs are depicted in Figure 8. The notation is the same as in Figure 7 with the only difference that the x-axis of Figure 8 indicates how many of the used cores are *MicroBlaze* processors and how many *DCT* HW IPs. By exploiting data- and task-level parallelism, we implemented heterogeneous MP-SoCs consisting of up to 24 cores. As a reference number to estimate the speed-up of each MP-SoC, we again used the number of clock cycles of the 1-*MicroBlaze* system (see the leftmost bar in Figure 7). We started with a 2-core system consisting of 1 *MicroBlaze* and 1 *DCT* IP. Its architecture is depicted in Figure 4(b). It exploits task-level parallelism within a tile, which affects the achieved speed-up. Although the *DCT* IP core is very efficient and fast in terms of performance, the overall speed-up is only 1.9x (see the leftmost bar in Figure 8), which actually is in line with Amdahl's law. Similarly to the experiments with the homogeneous systems, we continued with points that exploit data-level parallelism between the tiles, increasing the number of tiles processed in parallel. The 2-core system requires 36KB of memory, i.e., the *DCT* IP core reduces the *MicroBlaze* memory requirement to 32KB but with an additional 4KB used for communication buffers, see Figure 4(b). Therefore, with 288KB of memory, we were able to implement systems with up to 8 *MicroBlaze*s and 8

DCT IPs (16 cores, processing 8 tiles in parallel). The achieved speed-up linearly scales from 1.9x for 2 cores to 15.2x for 16 cores as illustrated in Figure 8.

Like in the previous experiment, with the given constraints larger MP-SoCs can be implemented (and higher speed-ups can be achieved respectively) by exploiting data-level parallelism between the tiles and data- and task-level parallelism within the tiles. The most efficient heterogeneous MP-SoC architecture found by the simulation-level DSE to exploit data- and task-level parallelism within a tile is depicted in Figure 4(d). It consists of 4 *MicroBlaze* processors and 2 *DCT* IP cores. The total memory requirement of this system is 68KB.We selected and implemented the 18-, 20-, 22-, and 24-core systems in Figure 6 which actually are combinations of 2-cores per tile (2-*CPT*) and 6-cores per tile (6-*CPT*) architectures (see Figure 4(b) and Figure 4(d) respectively). The 18-core system consists of 11 *MicroBlazes* and 7 *DCT* IPs. It processes 5 tiles in parallel: 3 tiles are processed by 3 2-*CPT* architectures and 2 tiles are processed by 2 6-*CPT* architectures. The speed-up of this MP-SoC is 15.2x. The 20-core system processes 4 tiles in parallel: 1 tile is processed by 1 2-*CPT* architecture and 3 tiles are processed by 3 6-*CPT* architectures. In total, 13 *MicroBlazes* and 7 *DCT* IPs achieve a speed-up of 15.9x. The speed-up of the 22-core system is 17.7x. This MP-SoC consists of 14 *MicroBlazes* and 8 *DCT* IPs that process 5 tiles in parallel: 2 tiles are processed by 2 2-*CPT* architectures and 3 tiles are processed by 3 6-*CPT* architectures. The 24-core MP-SoC, consisting of 16 *Microblazes* and 8 *DCT* IPs, processes 4 tiles in parallel utilizing 4 6-*CPT* architectures. The achieved speed-up by this system is 19.7x compared to a 1-*MicroBlaze* system.

5. CONCLUSIONS

We presented the Daedalus system-level design framework and our first industrial experiences in deploying Daedalus in the early design stage of JPEG-based image compression MP-SoCs. All presented DSE experiments and the real implementation of 25 MP-SoCs on FPGA were performed in a short amount of time, 5 days in total, due to the highly automated Daedalus design flow. Around 70% of this time was taken by the low-level commercial synthesis and place-and-route FPGA tools. The obtained results show that the Daedalus high-level MP-SoC models are capable of accurately predicting the overall system performance, i.e., the performance error is around 5%. By exploiting the data- and task-level parallelism in the JPEG application, Daedalus can deliver scalable MP-SoC solutions in terms of performance and cost. We were able to achieve a performance speed-up of up to 20x compared to a single processor system. The MP-SoC system performance was limited by the available on-chip FPGA memory resources and the available IP cores in Daedalus RTL library. To achieve higher performance speed-up, the RTL library has to be extended with more dedicated HW IP cores. The next step in our joint project with Chess B.V. is to use Daedalus for the exploration and design of JPEG2000-based image compression MP-SoCs.

6. ACKNOWLEDGMENTS

This work was supported by the Dutch Technology Foundation (PROGRESS/STW) under the Artemisia project (LES.6389).

7. REFERENCES

[1] Daedalus system-level design, http://daedalus.liacs.nl/.

[2] D. Lyonnard et al. Automatic Generation of Application-Specific Architectures for Heterogeneous Multiprocessor System-on-Chip. In *Proc. of the Design Automation Conference (DAC'2001)*, June 18-22 2001.

[3] C. Erbas, A. D. Pimentel, M. Thompson, and S. Polstra. A framework for system-level modeling and simulation of embedded systems architectures. *EURASIP Journal on Embedded Systems*, vol. 2007, Article ID 82123, 2007.

[4] A. Gerstlauer and D. Gajski. System-level abstraction semantics. In *Proc. 15th Int. Symposium on System Synthesis (ISSS'02)*, pages 231–236, Oct. 2-4 2002.

[5] C. Haubelt, J. Falk, J. Keinert, T. Schlichter, M. Streubuhr, A. Deyhle, A. Hadert, and J. Teich. A SystemC-Based Design Methodology for Digital Signal Processing Systems. *EURASIP Journal on Embedded Systems*, 2007:Article ID 47580, 22 pages, 2007. doi:10.1155/2007/47580.

[6] K. Keutzer et al. System level design: Orthogonalization of concerns and platform-based design. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 19(12), Dec. 2000.

[7] G. Kahn. The semantics of a simple language for parallel programming. In *Proc. of the IFIP Congress 74*, 1974.

[8] M. J. Rutten et al. A Heterogeneous Multiprocessor Architecture for Flexible Media Processing. *IEEE Design & Test of Computers*, 19(4), 2002.

[9] G. Martin. Overview of the MPSoC Design Challenge. In *Proc. Design Automation Conference (DAC)*, San Francisco, USA, July 24-28 2006.

[10] A. Mihal and K. Keutzer. Mapping concurrent applications onto architectural platforms. In A. Jantsch and H. Tenhunen, editors, *Networks on Chips*, pages 39–59. Kluwer Academic Publishers, 2003.

[11] H. Nikolov, T. Stefanov, and E. F. Deprettere. Multi-processor system design with ESPAM. In *Proc. of the Int. Conf. on HW/SW Codesign and System Synthesis (CODES+ISSS '06)*, pages 211–216, Oct. 2006.

[12] H. Nikolov, T. Stefanov, and E. F. Deprettere. Systematic and automated multi-processor system design, programming, and implementation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 27(3):542–555, March 2008.

[13] A. D. Pimentel, C. Erbas, and S. Polstra. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *IEEE Transactions on Computers*, 55(2):99–112, 2006.

[14] A. D. Pimentel, M. Thompson, S. Polstra, and C. Erbas. Calibration of abstract performance models for system-level design space exploration. *Journal of Signal Processing Systems for Signal, Image, and Video Technology*, 50(2), 2008.

[15] T. Kangas et al. UML-based multi-processor SoC design framework. *ACM Trans. on Embedded Computing Systems*, 5(2):281–320, May 2006.

[16] T. Stefanov et al. System design using Kahn process networks: The Compaan/Laura approach. In *Proc. of the Int. Conference on Design, Automation and Test in Europe (DATE)*, pages 340–345, Feb. 2004.

[17] M. Thompson, T. Stefanov, H. Nikolov, A. D. Pimentel, C. Erbas, S. Polstra, and E. F. Deprettere. A framework for rapid system-level exploration, synthesis, and programming of multimedia MP-SoCs. In *Proc. of the Int. Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS '07)*, pages 9–14, 2007.

[18] S. Verdoolaege, H. Nikolov, and T. Stefanov. PN: a tool for improved derivation of process networks. *EURASIP Journal on Embedded Systems*, vol. 2007, Article ID 75947, 2007.