A Multiobjective Optimization Model for Exploring Multiprocessor Mappings of Process Networks

Cagkan Erbas
Dept. of Computer Science
University of Amsterdam
Kruislaan 403, 1098 SJ
Amsterdam, The Netherlands
cagkan@science.uva.nl

Selin C. Erbas
Dept. of Stochastics
Aachen Univ. of Technology
Wüllnerstrasse 3, 52056
Aachen, Germany

Andy D. Pimentel
Dept. of Computer Science
University of Amsterdam
Kruislaan 403, 1098 SJ
Amsterdam, The Netherlands

ABSTRACT

In the Sesame framework, we develop a modeling and simulation environment for the efficient design space exploration of heterogeneous embedded systems. Since Sesame recognizes separate application and architecture models within a single system simulation, it needs an explicit mapping step to relate these models for co-simulation. So far in Sesame, the mapping decision has been assumed to be made by an experienced designer, intuitively. However, this assumption is increasingly becoming inappropriate for the following reasons: already the realistic systems are far too complex for making intuitive decisions at an early design stage where the design space is very large. Likely, these systems will even get more complex in the near future. Besides, there exist multiple criteria to consider, like processing times, power consumption and cost of the architecture, which make the decision problem even harder.

In this paper, the mapping decision problem is formulated as a multiobjective combinatorial optimization problem. For a solution approach, an optimization software tool, implementing an evolutionary algorithm from the literature, has been developed to achieve a set of best alternative mapping decisions under multiple criteria. In a case study, we have used our optimization tool to obtain a set of mapping decisions, some of which were further evaluated by the Sesame simulation framework.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Design studies, Modeling techniques; C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems, Signal processing systems; G.1.6 [**Optimization**]: Integer programming

General Terms

Performance, Design

Keywords

Design space exploration, Performance estimation with simulation, Evolutionary multiobjective optimization

CODES+ISSS'03, October 1–3, 2003, Newport Beach, California, USA.

1. INTRODUCTION

In the context of the Artemis project [6], we are developing the Sesame [1], [7] framework which provides modeling and simulation methods and tools for the efficient design space exploration of heterogeneous embedded multimedia systems. This framework should allow for rapid performance evaluation of different architecture designs, application to architecture mappings, and hardware/software partitionings. In addition, it should do so at multiple levels of abstraction *and* for a wide range of multimedia applications. To achieve this flexibility, the Sesame environment recognizes separate application and architecture models within a single system simulation. The application model defines the functional behavior of an application, including both computation and communication behavior. The architecture model defines architecture resources and captures their performance constraints. An explicit mapping step maps an application model onto an architecture model for co-simulation.

So far in Sesame, the mapping step is assumed to be performed by an experienced designer, intuitively. However, this assumption is increasingly becoming inappropriate for efficient design space exploration. First of all, the Sesame environment targets exploration at an early design stage where the design space is very large. At this stage, it is very hard to make critical decisions such as *mapping* without using any analytical method or a design tool, since these decisions seriously affect the rest of the design process, and in turn, the success of the final design. Besides, modern embedded systems are already quite complicated, generally having a heterogeneous combination of hardware and software parts possibly with dynamic behavior. It is also very likely that these embedded systems will become even more complex in the near future, and intuitive mapping decisions will eventually become obsolete for future designs. Moreover, coping with the design constraints of embedded systems, there exist multiple criteria to consider, like the processing times, power consumption and cost of the architecture, all of which further complicate the mapping decision.

In this paper, we develop a mathematical model to capture the trade-offs faced during the mapping stage in Sesame. In our model, these trade-offs, namely the maximum processing time, power consumption and total cost of the architecture, are formulated as the multiple conflicting objectives of the mapping decision problem. Then using our optimization software tool, which is based on a widely-known evolutionary algorithm, we solve the mapping decision problem for a multimedia application. Finally, choosing two conflicting solutions from our optimization results, we simulate their performance consequences for validation using the Sesame simulation framework.

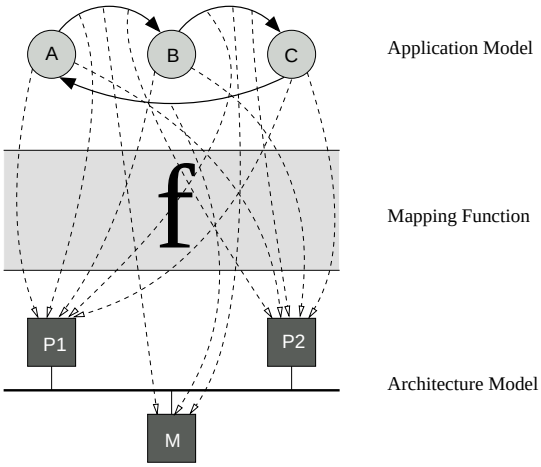

Figure 1: The mapping problem on a simple example.

The rest of the paper is organized as follows: we model the mapping decision in Sesame as a multiobjective combinatorial optimization problem in the next section. Section 3 introduces the multiobjective optimization theory and presents our solution approach. In Section 4, we study an M-JPEG encoder application and explore its design space. Section 5 discusses related work. Finally, Section 6 concludes the paper.

2. PROBLEM AND MODEL DEFINITION

In the Sesame framework, applications are modeled using the Kahn Process Network (KPN) [4] model of computation in which parallel processes – implemented in a high level language – communicate with each other via unbounded FIFO channels. By executing the application model, each process records its actions in order to generate its own trace of application events which is necessary for driving an architecture model. There are three types of application events in two groups: *execute* for computation events and *read* and *write* for communication events.

The architecture models in Sesame, simulate the performance consequences of the computation and communication events generated by an application model. They solely account for performance constraints and only model timing behavior, since the functional behavior is already captured in the application model. An architecture model is constructed from generic building blocks provided by a library which contains template models for processing cores and various types of memory.

Since Sesame makes a distinction between application and architecture models, it needs an explicit mapping step to relate these models for co-simulation. In this step, the designer decides for each application process and FIFO channel a destination architecture model component to simulate its workload. Thus, this step is one of the most important stages in the design process, since the final success of the design is highly dependent on these mapping choices. In Figure 1, we illustrate this mapping step on a very simple example. In this example, the application model consists of three Kahn processes and FIFO channels. The architecture model contains two processors and one shared memory. To decide on an optimum mapping, there exist multiple criteria to consider: maximum processing time in the system, power consumption and the total cost. This section aims at defining a mapping function, shown with **f** in Figure 1, to supply the designer with a set of best alternative mappings under the mentioned system criteria.

2.1 Application Modeling

The application models in Sesame are represented by a graph $KPN = (V_K, E_K)$ where the set V_K and E_K refer to the Kahn nodes and the directed FIFO channels between these nodes, respectively. For each node $a \in V_K$, we define $B_a \subseteq E_K$ to be the set of FIFO channels connected to node a, $B_a = \{b_{a1}, \ldots, b_{an}\}$. For each Kahn node, we define a computation requirement, shown with α_a, representing the computational workload imposed by that Kahn node onto a particular component in the architecture model. The communication requirement of a Kahn node is not defined explicitly, rather it is derived from the channels attached to it. We have chosen this type of definition for the following reason: if the Kahn node and one of its channels are mapped onto the same architecture component, the communication overhead experienced by the Kahn node due to that specific channel is simply neglected. For the communication workload imposed by the Kahn node, only those channels that are mapped onto different architecture components are taken into account. So our model neglects internal communications and only considers external communications. Formally, we denote the communication requirement of the channel b with β_b. To include memory[1] latencies into our model, we require that mapping a channel onto a specific memory asks computation tasks from the memory. To express this, we define the computational requirement of the channel b from the memory as α_b. The formulation of our model ensures that the parameters β_b and α_b are only taken into account, when the channel b is mapped onto an external memory.

2.2 Architecture Modeling

Similarly to the application model, the architecture model is also represented by a graph $ARC = (V_A, E_A)$ where the sets V_A and E_A denote the architecture components and the connections between the architecture components, respectively. In our model, the set of architecture components consists of two disjoint subsets: the set of processors (P) and the set of memories (M), $V_A = P \cup M$ and $P \cap M = \emptyset$. For each processor $p \in P$, the set $M_p = \{m_{p1}, \ldots, m_{pj}\}$ represents the memories which are reachable from the processor p. We define processing capacities for both the processors and the memories as c_p and c_m, respectively. These parameters are set such that they reflect processing capabilities for processors, and memory access latencies for memories.

One of the key considerations in the design of embedded systems is the power consumption. In our model, we consider two types of power consumption for the processors. We represent the power dissipation of the processor p during *execution* with w_{pe}, while w_{pc} represents its power dissipation during *communication* with the external memories. For the memories, we only define w_{me}, the power dissipation during *execution*. For both processors and memories, we neglect the power dissipation during idle times. In our model, we also consider the fixed costs associated with the architecture model components. Using an architecture component in the system adds a fixed amount to the total cost. We represent the fixed costs as u_p and u_m for the processors and the memories, respectively.

2.3 The Mapping Problem

We have the following decision variables in the model: $x_{ap} = 1$ if Kahn node a is mapped onto processor p, $x_{bm} = 1$ if channel b is mapped onto memory m, $x_{bp} = 1$ if channel b is mapped onto processor p, $y_p = 1$ if processor p is used in the system, $y_m = 1$ if memory m is used in the system. All the decision variables get a value of zero in all other cases. The constraints in the model are:

[1]In this paper, *memory* refers to any kind of hardware that can be used for data transfer. Buses, FIFOs, RAMs, etc. are all memories.

- Each Kahn node has to be mapped onto a single processor,

$$\sum_{p \in P} x_{ap} = 1 \qquad \text{for all } a \in V_K. \tag{1}$$

- Each channel in the application model has to be mapped onto a processor or a memory,

$$\sum_{p \in P} x_{bp} + \sum_{m \in M} x_{bm} = 1 \qquad \text{for all } b \in E_K. \tag{2}$$

- If two communicating Kahn nodes are mapped onto the same processor, then the communication channel(s) between these nodes have to be mapped onto the same processor.

$$\text{If } \exists\, b = (a_i, a_j) \in E_K \text{ with } x_{a_i p} = 1 \text{ and } x_{a_j p} = 1, \text{ then } x_{bp} = 1. \tag{3}$$

- The constraint given below ensures that when two communicating Kahn nodes are mapped onto two separate processors, the channel(s) between these nodes are to be mapped onto an external memory.

$$\text{Let } p_k \neq p_l \in P, a_i, a_j \in V_K \text{ and } b \in B_{a_i} \cap B_{a_j}. \text{ If } x_{a_i p_k} = 1 \text{ and } x_{a_j p_l} = 1 \text{ then } x_{bm} = 1 \text{ for an } m \in M_{p_k} \cap M_{p_l}. \tag{4}$$

- The following constraints are used to settle the values of y_p and y_m's in the model. We multiply the right-hand side of the first equation series by the total number of Kahn nodes and FIFO channels, since this gives an upper bound on the number of application model components that can be mapped to any processor. Similar logic is applied to the equations related with memory.

$$\sum_{a \in V_K} x_{ap} + \sum_{b \in E_K} x_{bp} \leq (\mid V_K \mid + \mid E_K \mid) y_p \qquad \text{for all } p \in P, \tag{5}$$

$$\sum_{b \in E_K} x_{bm} \leq \mid E_K \mid y_m \qquad \text{for all } m \in M. \tag{6}$$

Three conflicting objective functions exist in the optimization problem:

- The first objective function tries to minimize the maximum processing time in the system. For each processor and memory, f_p and f_m represent the total processing time in the processor and memory, respectively. We also show the total time spent by the processor for the execution events as f_p^e and for the communication events as f_p^c.

$$f_p = f_p^e + f_p^c, \tag{7}$$

$$f_p^e = \frac{1}{c_p} \sum_{a \in V_K} \alpha_a x_{ap}, \tag{8}$$

$$f_p^c = \frac{1}{c_p} \sum_{a \in V_K} x_{ap} \sum_{b \in B_a, m \in M_p} \beta_b x_{bm}, \tag{9}$$

$$f_m = \frac{1}{c_m} \sum_{b \in E_K} \alpha_b x_{bm}. \tag{10}$$

So the objective function is expressed as

$$\min \max_{i \in V_A} f_i. \tag{11}$$

- The second objective function tries to minimize the power consumption of the whole system. Similarly, g_p and g_m denote the total power consumption of processor p and memory m.

$$g_p = f_p^e w_{pe} + f_p^c w_{pc}, \tag{12}$$

$$g_m = f_m w_{me}. \tag{13}$$

$$\min \sum_{i \in V_A} g_i. \tag{14}$$

- The last objective function aims at minimizing the total cost of the architecture model.

$$\min \sum_{p \in P} u_p y_p + \sum_{m \in M} u_m y_m. \tag{15}$$

DEFINITION 1. *The Multiprocessor Mappings of KPNs Problem is the following multiobjective integer optimization problem:*

minimize $f = (\max_{i \in V_A} f_i, \sum_{i \in V_A} g_i, \sum_{p \in P} u_p y_p + \sum_{m \in M} u_m y_m)$,

subject to *(1) – (10), (12) and (13).*

3. MULTIOBJECTIVE OPTIMIZATION

DEFINITION 2. *A general multiobjective optimization problem is defined as:*

minimize $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_k(\mathbf{x}))$

subject to $\mathbf{x} \in X$

where $\mathbf{x}$ *represents a solution and* X *is a set of feasible solutions. The objective function vector* $\mathbf{f}(\mathbf{x})$ *maps a solution vector* $\mathbf{x}$ *in decision space to a point in objective space.*

In general, in a multiobjective optimization problem, it is not possible to find a single solution that minimizes all objectives, simultaneously. Therefore, one is interested to explore a set of solutions, called the *Pareto optimal* set, which are not *dominated* by any other solution in the feasible set. The corresponding objective vectors of these Pareto optimal points, named *efficient points*, form the *Pareto front* on the objective space.

DEFINITION 3. *We say* $\mathbf{x}$ *dominates* $\mathbf{x}^*$ *iff* $\forall i \in \{1, \ldots, k\}$ $f_i(\mathbf{x}) \leq f_i(\mathbf{x}^*)$ *and there exists at least one* $i \in \{1, \ldots, k\}$ *such that* $f_i(\mathbf{x}) < f_i(\mathbf{x}^*)$.

The most traditional approach to solve a multiobjective optimization problem is to aggregate the objectives into a single objective by using a *weighting mean*. However this approach has major drawbacks. It is not possible to locate the non-convex parts of the Pareto front [3] and it requires several consecutive runs of the optimization program with different weights. Recently, there has been an increasing interest in evolutionary multiobjective optimization. This is because of the fact that evolutionary algorithms (EAs) seem well-suited for this type of problems, as they deal simultaneously with a set of possible solutions called *population*. This allows us to find several members of the Pareto optimal set in a single run of the algorithm.

To solve the mapping problem given by Definition 1 in Section 2.3, we use the Strength Pareto Evolutionary Algorithm (SPEA) [11] which maintains an external set to preserve the nondominated solutions encountered so far besides the original population. It uses the concept of dominance in order to assign fitness values to individuals. Distinct fitness assignment schemes are defined for the

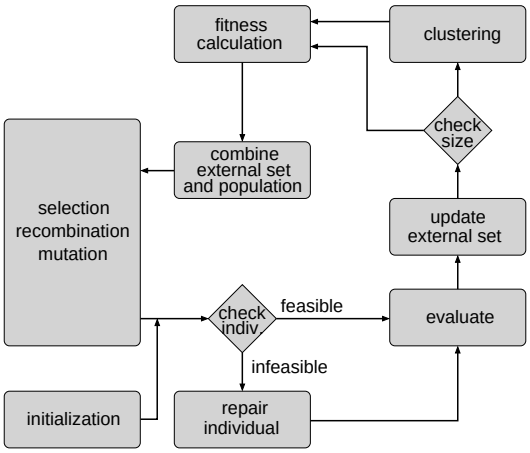

Figure 2: Overview of SPEA.

population and the external set to always ensure that better fitness values are assigned to individuals in the latter. Additionally, SPEA performs *clustering* to limit the number of individuals in the external set while also maintaining diversity among them. For selection, it uses binary tournament with replacement. Finally, both population and the external nondominated set take part in selection. In our SPEA implementation, we have introduced a repair mechanism to handle infeasibility. The repair takes place before the individuals enter evaluation to make sure that only valid individuals are evaluated. The details of our repair mechanism will be explained later. Figure 2 gives an overview of SPEA which also includes the repair mechanism. The big loop in the figure is performed for a certain number of generations.

3.1 Individual Coding

Each genotype consists of two main parts: a part for Kahn process nodes and a part for FIFO channels. Each gene in the chromosome has its own allele set which is determined by the type of the gene and the constraints of the problem. For genes representing Kahn process nodes, only the set of processors at the architecture model form the allele set, while for genes representing the FIFO channels, both the set of processors and the set of memories constitute the allele set. The constraints of the problem may include

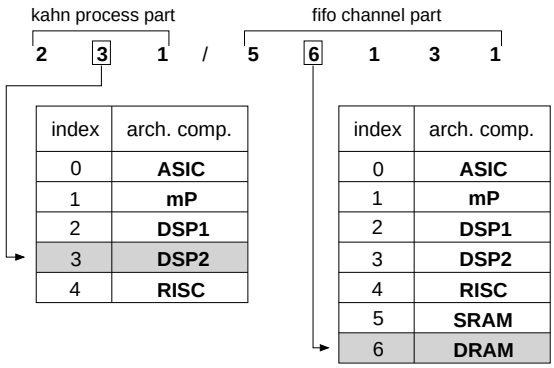

Figure 3: An example individual coding.

some limitations which should be considered in individual coding. For example, if there exists a dedicated architecture component for a specific Kahn process, then this architecture component has to be included only in the allele set of this Kahn process.

In Figure 3, an example chromosome is given. The first three genes are those for Kahn process nodes, and the rest are those for FIFO channels. For this gene, the second Kahn process is mapped onto DSP2 while the second FIFO channel is mapped onto DRAM. We also see that the allele sets for these two genes are different.

3.2 Constraint Violation

We have developed a repair mechanism to deal with the constraint violations. The encoding of the individual ensures that the constraints (1) and (2) are never violated. However, due to randomness in SPEA (in initialization, crossover and mutation steps), the constraints (3) and (4) are prone to violation. The repair mechanism given in Algorithm 1 only considers the FIFO channel part of the individuals, since constraints (3) and (4) ensure correct mapping of the FIFO channels. Our repair algorithm simply checks for each FIFO channel whether the Kahn processes it is connected to are mapped onto the same processor. If this condition holds, then it ensures that the FIFO channel is also mapped onto that processor. If the condition does not hold, which means that the Kahn processes are mapped onto different processors (say, P_1 and P_2), it finds the set of memories reachable from both P_1 and P_2 (mathematically, $M_{P_1} \cap M_{P_2}$). Then it selects a memory from this set randomly and maps the FIFO channel onto that memory. However, if $M_{P_1} \cap M_{P_2} = \emptyset$, then it maps the Kahn processes and the FIFO channel between them onto the processor P_1. We have adapted our repair algorithm in SPEA such that it is applied before the evaluation step to make sure that only valid individuals are evaluated.

Algorithm 1 Individual Repair Algorithm

input: I (individual)
output: I (individual)

```
for all FIFO channel genes do
  get Kahn nodes connected by this FIFO channel: K1 and K2.
  if K1 and K2 are mapped onto the same processor then
    repair: map FIFO channel onto the same processor.
  else if FIFO channel is mapped onto a processor then
    get processors K1 and K2 are mapped onto: P1 and P2.
    if M_P1 ∩ M_P2 ≠ ∅ then
      choose randomly a memory from M_P1 ∩ M_P2: M.
      repair: map FIFO channel on M.
    else
      repair: map FIFO channel and K2 on P1.
    end if
  end if
end for
```

4. CASE STUDY

We have chosen a modified M-JPEG encoder application, referred as M-JPEG*, to explore its design space. It differs from traditional M-JPEG encoders in three main ways: M-JPEG* only supports lossy encoding while M-JPEG supports both lossless and lossy encodings, M-JPEG* can operate on YUV and RGB video data whereas M-JPEG usually operates on YUV format, and finally M-JPEG* can change quantization and Huffman tables dynamically while M-JPEG has no such behavior. In Figure 4, we give the Kahn process network model of our M-JPEG* application. In our simulations, the frame size is 128x128 with 8 bits/pixel. The frame format is RGB. A more detailed discussion of the M-JPEG* application can be found in [7].

In Figure 5, we present a target platform architecture for the M-JPEG* application. The architecture model consists of a general purpose microprocessor (mP), two DSPs, two ASICs (VIP and

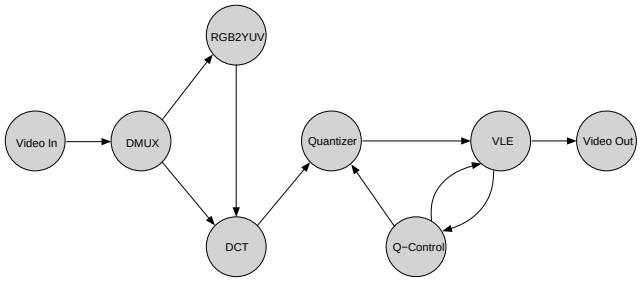

Figure 4: M-JPEG* encoder application model.

VOP), an SRAM and a DRAM. For these architecture components, realistic latency values from [7] have been used to calculate their processing capacities: c_p and c_m. Similarly, for the Kahn processes and FIFO channels in Figure 4, computational and communicational requirements (namely the parameters α_a for the nodes and the parameters α_b and β_b for the FIFO channels) have been calculated using statistics obtained from the C++ implementation code of this Kahn process network.

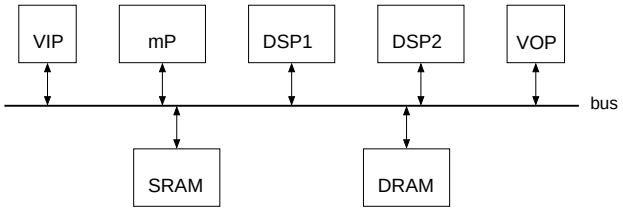

Figure 5: M-JPEG* encoder platform architecture model.

We have implemented the SPEA algorithm in C++ to solve our mapping problem given by Definition 1 in Section 2.3. Our software is based on GAlib[2] [10], a C++ library of genetic algorithm components. We have chosen a population size of 100, crossover probability of 0.5 and a mutation rate[3] of 0.056. The size of the external set is chosen as 20. In Figure 6, we show the nondominated front (approximated Pareto front) obtained by our optimization software after 300 generations in a single run. In the figure, there exist two distinct points where the maximum processing time attains its peak values. When we compare these two extreme points, we observe that with a slight increase of cost from 160 to 180, we obtain a big gain in power consumption (drops from 1003 to 919). On the other hand, one can also see that the compromise points are around the middle of the nondominated front.

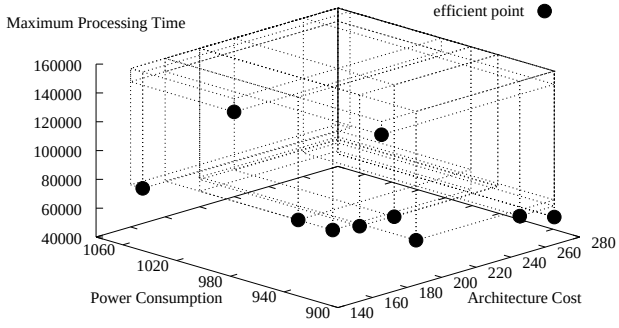

Figure 6: Nondominated front for the M-JPEG* case study.

[2]Originally, GAlib solves optimization problems with a single objective. We modified it to solve problems with multiple objectives.
[3]We calculated this with $1/(\text{chromosome length}) = 0.056$.

Table 1: Two nondominated solutions chosen for simulation.

Solution	*Max. Processing Time*	*Power Cons.*	*Cost*
$P1$	50762	981	200
$P2$	140808	1003	160

Two solutions have been selected for further evaluation by the `Sesame` framework. In Table 1, the chosen solutions $P1$ and $P2$, and their objective function values are given. $P1$ is one of the compromise solutions while $P2$ has its maximum processing time close to maxima. Our aim is to demonstrate the performance consequences of these two conflicting design decisions via simulation.

Figures 7 and 8 show the utilization details of components in the architecture models of the mappings $P1$ and $P2$, respectively. In both architecture models, only a subset of the available components are used, so actually these mappings are different instances of the platform architecture. For example, for storing data $P1$ utilizes SRAM while $P2$ uses DRAM. When we compare the architecture models of $P1$ and $P2$, we observe a dramatic difference between systems throughput. The former one has a video throughput of 154.2 frames/sec while the latter is only at 82.9 frames/sec. We can explain the reason behind this as follows: when we examine both Figures 7 and 8, we observe that the M-JPEG* application is quite sensitive to communication between the components. This is demonstrated by the large I/O times of mP and DSP1 in Figure 7 and DSP2 in Figure 8. This makes choosing a fast memory very critical for good system performance, so using an SRAM instead of a DRAM in the first architecture dramatically improves the overall performance. This fact is also noticed by the large busy times of the

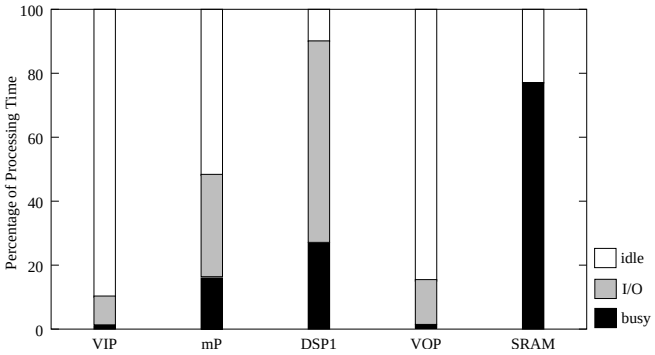

Figure 7: Simulation results showing the utilization of architecture components in $P1$. The throughput is 154.2 frames/sec.

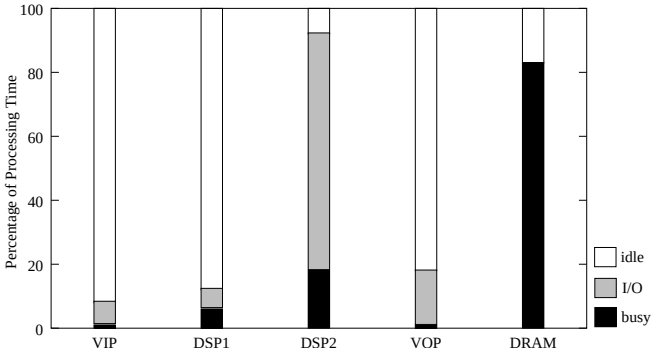

Figure 8: Simulation results showing the utilization of architecture components in $P2$. The throughput is 82.9 frames/sec.

SRAM and DRAM components in the figures. The second architecture also has an additional drawback that it is not well-balanced. When we compare the utilizations of DSP1 and DSP2 in Figure 8, we see that DSP1 is idle more than 80% of its execution time while DSP2 is only idle for less than 10%.

5. RELATED WORK

In the domain of embedded systems and hardware/software codesign, multiobjective optimization studies have been performed extensively for *system-level synthesis* [2], [8], [9]. The latter means the problem of optimally mapping a task-level specification onto a heterogeneous hardware/software architecture. Teich et al. [8] partition this problem into two steps: the selection of the architecture (allocation), and the mapping of the algorithm onto the selected architecture in space (binding) and time (scheduling). In their framework, they only consider cost and speed of the architecture, power consumption is ignored. Besides, they use penalty functions to cope with infeasibility. This technique reduces the number of infeasible individuals to an acceptable degree [8], while our repair mechanism replaces all infeasible individuals. In [9], a similar synthesis approach is applied to evaluate the design tradeoffs in packet processor architectures. But additionally, this model includes a real-time calculus to reason about packet streams and their processing.

In the MOGAC framework [2], starting from a task graph specification, the synthesis problem is solved for three objectives: cost, speed and power consumption of the target architecture. To accomplish this, an adaptive genetic algorithm which can escape local minima is utilized. However, this framework lacks the management of possible infeasibility as it treats all non-dominated solutions equally even if they violate hard constraints. The invalid individuals are only removed at the end of evolution. As a consequence of this, the MOGAC framework may confront convergence problems, i.e. convergence to a set of infeasible solutions.

In [5], configuration space of a parameterized system-on-chip (SoC) architecture is explored using an exploration algorithm which combines genetic algorithm and exhaustive search techniques. The approximated Pareto-optimal set obtained by the exploration algorithm is afterwards compared to the actual Pareto-optimal set using a distance function for fitness calculations.

In the `Sesame` framework, we do not target at the problem of system synthesis. Therefore, a schedule is not constructed at the end of the design process. Our aim is to develop a methodology which will quickly evaluate a large design space and provide us a number of approximated Pareto-optimal solutions. These solutions are then input to our simulation framework for further evaluation. After simulation, figures about system-level trade-offs (utilization of components, data throughput, communication media contention, etc.) are provided to the designer. Thus, our goal is efficient design space exploration in terms of simulation. As a final remark, we should also note that our framework differs from the mentioned frameworks in the sense that it uses process networks for algorithm specification rather than task graphs.

6. CONCLUSION

In this paper, we have developed a novel analytical method to optimally map application models, defined as Kahn process networks, onto multiprocessor target architectures. This methodology is specifically designed and developed for the `Sesame` modeling and simulation framework to be used prior to simulation. In many cases, simulation environments have to cope with large design spaces, where the simulation of all alternatives is too expensive or even impossible. Although finding the optimal mapping is an intractable problem, this study tackles the mapping problem by providing the designer a set of approximated Pareto-optimal solutions to be further evaluated in terms of simulation. To illustrate the practical aspects of our theoretical results, we have taken a modified M-JPEG encoder application and explored its design space in a case study. We could quickly evaluate two conflicting design choices for this multimedia application using `Sesame`'s efficient simulation methodology.

The current research in `Sesame` is to extend this work in two distinct tracks. One track is to formulate a new evolutionary algorithm and compare it against SPEA and probably also against other candidates from the literature. Another track is to refine the model introduced in this study. Some of the assumptions in this study can be further refined for getting better optimization results, but this should be done with special care as one should avoid adding details that may hamper the efficiency in terms of optimization time.

7. ACKNOWLEDGEMENTS

We thank Marco Laumanns and Eckart Zitzler for their help in plotting the nondominated front.

8. REFERENCES

[1] J. E. Coffland and A. D. Pimentel. A software framework for efficient system-level performance evaluation of embedded systems. In *Proc. of the ACM Symposium on Applied Computing*, Mar. 2003.

[2] R. P. Dick and N. K. Jha. MOGAC: A multiobjective genetic algorithm for hardware-software co-synthesis of distributed embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Oct. 1998.

[3] M. Ehrgott. *Multicriteria Optimization*. Lecture Notes in Economics and Mathematical Systems. Springer-Verlag, 2000.

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

[5] M. Palesi and T. Givargis. Multi-objective design space exploration using genetic algorithms. In *Proc. of the 10th Int. Symposium on Hardware/Software Codesign*, May 2002.

[6] A. D. Pimentel, P. Lieverse, P. van der Wolf, L. O. Hertzberger, and E. F. Deprettere. Exploring embedded-systems architectures with Artemis. *IEEE Computer*, Nov. 2001.

[7] A. D. Pimentel, S. Polstra, F. Terpstra, A. W. van Halderen, J. E. Coffland, and L. O. Hertzberger. Towards efficient design space exploration of heterogeneous embedded media systems. In *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation*. Springer, 2002.

[8] J. Teich, T. Blickle, and L. Thiele. An evolutionary approach to system-level synthesis. In *Proc. of the 5th Int. Symposium on Hardware/Software Codesign*, Mar. 1997.

[9] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli. A framework for evaluating design tradeoffs in packet processing architectures. In *Proc. of the ACM/IEEE Design Automation Conference*, June 2002.

[10] M. Wall. *GAlib: A C++ Library of Genetic Algorithm Components*. Massachusetts Institute of Technology, 1996.

[11] E. Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. PhD thesis, Swiss Federal Institute of Technology Zurich, 1999.