

Evaluating the Model Accuracy in Automated Design Space Exploration

Kalle Holma, Mikko Setälä, Erno Salminen, and Timo D. Hämäläinen
Tampere University of Technology, Institute of Digital and Computer Systems
P.O.Box 553, FI-33101 Tampere, Finland
kalle.holma@tut.fi

Abstract

Design space exploration is used to shorten the design time of System-on-Chips (SoCs). The models used in the exploration need to be both accurate and fast to simulate. This paper introduces a multi-level communication cost to improve the accuracy of the abstracted system models. During the simulation, one of three different communication costs is applied for each inter-task communication event based on the mapping of the communicating tasks. The accuracy of three system abstraction models including the presented communication cost is evaluated using a Motion-JPEG (M-JPEG) application described in Unified Modeling Language (UML). According to the results, the average error in frames per second (FPS) is 3.8% for the trace model, 4.3% for the modulo model, and 12.8% for the probabilistic model compared to FPGA execution. The results show that with the multi-level communication cost the accuracy is increased significantly, and accurate results can be achieved with arbitrary mappings.

Keywords—System-on-Chip, design space exploration, accuracy

1. Introduction

When designing a modern System-on-Chip (SoC), it is crucial to explore design space fast and accurately already in the early phase of the design process. In the design space exploration, the objective is to find optimized resource allocation, task mapping, and scheduling according to given criteria. The resources are processing elements (PEs), memories, Network-on-Chip (NoC) components, and external interfaces. Exploration models always impose a trade-off between speed and accuracy. Thus, the problem is to find a model that obtains sufficient exploration accuracy and a good simulation speed-up compared to traditional low-level cycle-accurate simulations.

For example, a good compromise between accuracy and speed-up would be less than 10% error in accuracy of the execution time with all allocations and mappings compared to real execution. At the same time, the simulation should be at least a hundred times faster than cycle-accurate low-level simulation. Furthermore, it is important to compare simulation accuracy against real execution for reliable results.

Various models have been used in literature for design space exploration and network evaluation. A summary of related work including this paper is presented in Table 1. In [1]-[3], it is possible to explore application mappings and PE allocations whereas the target domain of the other papers is in communication exploration.

In Table 1 the fourth column expresses the granularity of the simulation method, i.e. the level of smallest unit in the execution of the application. In fine grained execution, the smallest unit is e.g. basic block in assembly, whereas in medium level it can be e.g. one task or function. In coarse grained simulation the application is not separated into different tasks, a PE only executes the application or performs network transactions, which is the case in [4]. This kind of model is mainly targeted for NoC development, but it lacks the necessary granularity for analyzing the performance of the application in terms of e.g. frames per second (FPS).

Interestingly, the level of application granularity does not necessarily indicate much about the achieved speed-up or accuracy. The smallest error is achieved in [1][4][5] being less than 2%. However, they all have different granularity levels. In addition, the speed-up of [4] and [5] is at most one tenth of the speed-up in [3][6][7] even though the granularity of the simulation is fine in [3] and coarse in [4]. On the contrary, in [3][6][7] the estimation error is significantly larger than in [4] and [5], being at most 18% in average in [6]. In [1] and [2] no speed-up is reported.

Apart from [7], the number of mappings used per

Table 1
Comparison of Reported Exploration and Accuracy Properties.

Ref.	Exploration domain	Automation	Application granularity	RTOS	Speed-up against low-level simulation	Runtime estimation error	Compared to	Applications	Mappings per application
[1]	System	A	Medium	N/A	N/A	<2%	FPGA exec.	1	2
[2]	System	A	Medium	N/A	N/A	<26%	Simulation	1	4
[3]	System	M	Fine	x	~1000x	~5%	Simulation	1	2
[4]	NoC	A	Coarse	N/A	~10x	<2%	Simulation	2	1
[5]	NoC	M	Fine	N/A	2-4x	<2%	Simulation	4	1-6
[6]	NoC	M	Variable	N/A	~100x	~18%	Simulation	2	6
[7]	NoC	M	Medium	N/A	~450x	<10%	Simulation	1	112
this	System	A	Medium	x	~230x	~4-13%, <11-31%	FPGA exec.	1	9

A = Automated, M = Manual, N/A = The information was not available, ~ = Average, < = Maximum

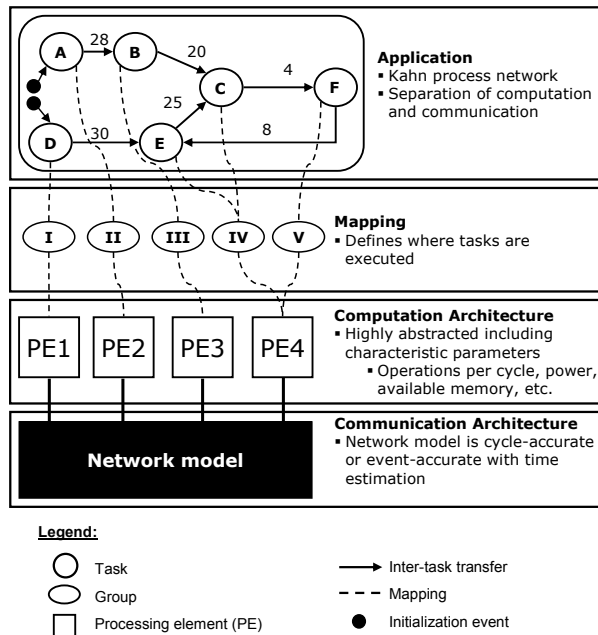


Fig. 1. Abstraction of both the application and the architecture for the exploration.

application is low, being only one in [4] and not more than six in [5] and [6]. Only in [1] the error has been evaluated against real execution whereas in [2]-[7] the comparison is between two simulations.

Further, only in [1][2][4] the exploration is automated, and only in [3] a real-time operating system (RTOS) utilized. Using RTOS and other software layers may simplify software (SW) development and reuse. However, they introduce overheads, such as increased runtime and memory footprint. Thus, they must be included in the model, although it may increase complexity.

As a solution to the accuracy-speed problem, the level of details in the abstract system model should be optimized. In this paper, we introduce a new cost fac-

tor, a multi-level communication cost that will include one previously neglected detail.

Our claim is that this is the key factor to improve the model accuracy without sacrificing speed. By separating the communication costs from the actual execution time of a task, the costs need to be measured only once and can be reused as such in the explorations of other applications. In addition to this paper, only in [1] the communication costs have been separated from the execution time of a task.

As a proof of concept, the accuracy of the model including the communication costs is evaluated with multiple mappings against FPGA execution and several metrics. The exploration is done using Koski design flow [8]. The exploration framework in Koski is implemented in Tool Command Language (TCL) and SystemC.

The rest of the paper is organized as follows. In Section 2 the used Model of Computation (MoC) in the exploration and the multi-level communication cost are described. Section 3 gives an overview of the exploration flow and Section 4 describes the measurements. Results are presented in Section 5. Finally, in Section 6 the paper is concluded.

2. System abstraction for design space exploration

Overview of the used design space exploration simulation method is depicted in Fig. 1. It is separated into four sections: application, computation architecture, communication architecture, and mapping. The mapping is presented with dashed lines between the application and the computation architecture.

2.1. Application model

The application model is based on the Kahn proc-

ess network model [9], but it is extended with many properties to support better modeling of practical systems. The task network consists of a set of tasks (vertices) connected by unidirectional channels (edges), which carry data packets (tokens) between the tasks.

If a task has multiple input channels, the dependence between them can be *OR* or *AND* type. If the dependence is type *OR*, a task starts executing as soon as one of its inputs has received data. In the case of *AND* dependence, the task cannot start execution until it has received data to all of its inputs.

In the application model there is at least one initial event, which triggers the execution of the application. Each event triggers only one task. The events can be either *one-shot* or *periodical*. The triggering by *one-shot* occurs only once at given time, whereas the *periodical* event triggers the task repeatedly with the given time interval. The amount of pipelining and parallel execution in the application can be controlled by adjusting the time intervals of *periodical* events.

Fig. 1 illustrates a simple task network consisting of six tasks (*A-F*) and two initial events, which trigger tasks *A* and *D*. The tasks are grouped into five groups (*I-V*) that are mapped to four PEs (*PE1-PE4*). The data amounts in bytes are expressed beside each edge. Assuming that the events in this application are *periodical* and their time interval is set properly, all the PEs 1-4 can execute tasks simultaneously.

The tasks are characterized by their operation counts, for instance 1000 integer operations. That translates to runtime according to the performance of the PE executing the task, for example 1.0 operations per clock cycle. Currently, the tasks do not perform any real execution. Instead, the execution is abstracted by waiting for the time specified by the operation count and the performance of the PE.

In our case, the application can be abstracted into *probabilistic*, *modulo*, or *trace* based MoC. The abstraction is based on profiling the application execution either on FPGA or in an instruction set simulator.

In the *probabilistic* and the *modulo* models, both the execution times of the tasks and the transferred data amounts have constant statistical average values. Moreover, in the *probabilistic* model, a task always outputs data to an output port with a fixed probability independent of its current execution count. On the contrary, in the *modulo* model the task outputs data to an output port only in specified execution counts calculated from the modulus of its current total execution count. For instance, a task in the *probabilistic* model sends a data packet to its certain output with a 33% probability in each execution, whereas the same task in the *modulo* model sends the packet on every third execution. Hence, the *modulo* model is especially suitable

Table 2 Differences Between the MoCs.

	Operation count	Data amount to send	Send occurrence
Probabilistic	Constant	Constant	Distinct probability for each output channel.
Modulo	Constant	Constant	Depends on the modulus of the execution count
Trace	Depends on the execution count	Depends on the execution count	Depends on the execution count

for modeling applications which behave regularly.

The *trace* model represents the real execution of the application based on the collected data in the profiling stage. Thus, the behavior of a task is unique in its each execution count. The *trace* model is very detailed, but then, because of its size the exploration using it is slower than with the other models. This is due to increased model processing and compilation times.

Table 2 summarizes the differences between the used MoCs. The rightmost column presents whether the task outputs data to its output channel or not. Thus, in the *probabilistic* model, the probability concerns only whether the data is sent or not, not the amount of the data to be sent nor the operation count.

2.2. Mapping model

The mapping of the application tasks into PEs is presented in the mapping model. First, the tasks are grouped into task groups, and then the groups are mapped into the PEs. One task group is executed as a whole on a single PE. In this case, one purpose for the task groups is to model RTOS threads.

During automated exploration, the position of a task or a group is movable or immovable. This defines whether the task can be moved to another group or the group to another PE. Moreover, the contents of a PE or a group are mutable or immutable, which defines whether the contents can be modified. The tasks mapped to one PE may be scheduled using round-robin, static priority, or FIFO scheduling.

2.3. Computation architecture model with communication costs

The purpose of the computation architecture model is to abstract the PEs with only a few parameters, which include performance (e.g. integer operations per second), area, and maximum frequency. Moreover,

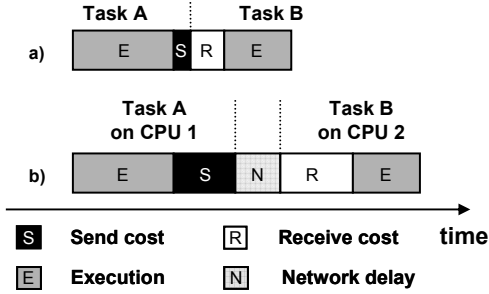


Fig. 2. Example of inter-task communication

multi-level communication costs are given for each PE. They consist of three levels: *intragroup*, *intergroup*, and *inter-PE* costs, all of which are further divided into *send* and *receive costs*. *Send cost* is applied in the source task and *receive cost* in the destination task for every communication event between the tasks.

Intragroup cost is applied when both the sender and the receiver task are grouped into the same group. When the tasks are in different groups, but the groups are mapped into the same PE, *intergroup* cost is used, whereas *inter-PE* cost is applied when the tasks are mapped into different PEs. For example in Fig. 1, communication between the tasks *C* and *E* has type *intragroup*, between *C* and *F* *intergroup* and between *C* and any other task *inter-PE*.

The costs model the operations needed for the inter-task communication, e.g. preparing and copying the data and in the case of *inter-PE* communication initializing the Direct Memory Access (DMA) transfer. The costs are expressed with a polynomial function of the amount of data to be sent, but they can also be constant values.

The communication costs depend on the selected PE, software platform (RTOS, device drivers), and memory architecture. However, they are application independent, so once measured they can be reused in the exploration of any application. By separating the communication costs from the execution of the application tasks, accurate results can be achieved with arbitrary mappings.

Fig. 2 presents two examples of inter-task communication. In a), the communication occurs between two tasks in the same PE, i.e. either *intragroup* or *intergroup*, whereas b) presents *inter-PE* communication. In both cases the execution times of the tasks are the same. The network delay is naturally not present in a), but otherwise the communication procedure is the same as in b). In addition, both send and receive costs are shorter in a), because the data does not have to be prepared for the network transaction. The network delay will be discussed next.

2.4. Communication architecture model

The computation architecture model is combined with the communication architecture model during the simulation. As presented in Fig. 2 b), the network model adds dynamic delay to the communication between the PEs. In addition, the network delay is not included in the *inter-PE* communication costs, because it cannot be determined statically before the simulation due to network contention. Therefore, the variable network delay is defined by the network model during the simulation. Further, the network topology and its abstraction level (register transfer level vs. transaction level) can be changed independently from the other models (application, mapping, computation architecture).

3. Exploration in the Koski design flow

In the Koski design flow [8], the system is first modeled in a Unified Modeling Language (UML) design environment. The functionality of the tasks can be designed from scratch or obtained from existing reference codes, implemented e.g. in C. A majority of the reference codes can be reused as such, because only the control structures need to be re-implemented.

Second, the model is automatically transformed into abstracted model, which is stored in an intermediate eXtensible Markup Language (XML) System Model (XSM) file format between the tools in Koski. Currently, the *probabilistic* and the *trace* model can be automatically generated from the UML system description, whereas the *modulo* model needs minor manual modifications. However, the *modulo* model cannot be easily generated to irregular applications.

Next, automated design space exploration optimizes various system parameters, such as resource allocation, task mapping, and NoC configuration within user-defined limits. The goal of the design space exploration is to minimize a case-dependent, user-defined cost function, e.g. $runtime \times area$, in heuristic fashion. The design space exploration utilizes a system-level simulation engine Transaction Generator [10], which combines the models presented in Fig. 1, and performs the simulation on the described system. In addition, it supports a cost for context switch. This can be used when simulating an application mapping with multiple groups.

Finally, the described application is automatically implemented with optimized architecture in FPGA. More detailed description of the exploration can be found from [8].

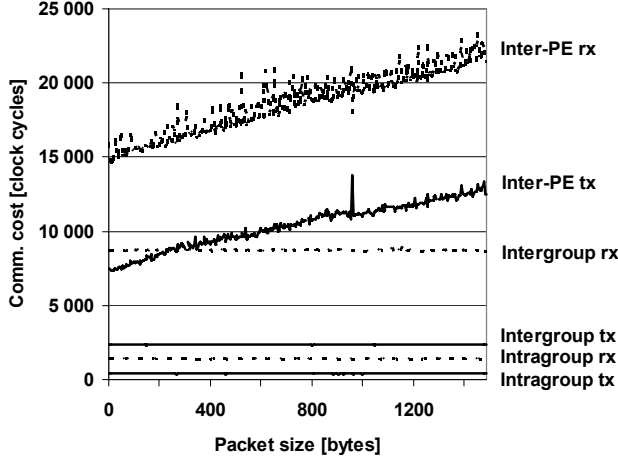


Fig. 3. Measured communication costs.

4. Measurements

This section describes the measurement setup used in evaluating the accuracy. Real FPGA execution and transaction level simulations were compared to obtain accuracy results. For speed-up, transaction level simulation was compared to cycle-accurate register transfer level (RTL) execution on Mentor Seamless CVE.

4.1. Platform

The system was implemented on Altera Stratix FPGA board, and therefore the processors had to be synthesizable. Altera Nios II processors utilizing eCos [11] RTOS connected through a shared bus were used. The clock frequency of the processors and the bus was 50 MHz. The simulations were run on a 3.0 GHz Pentium 4.

4.2. Measuring the communication costs

A simple test application was created for measuring the communication costs. It consisted of two tasks, which sent data packets with size of 4-1488 bytes to each other without performing any actual execution. The tasks were mapped to the same group, to different groups within the same PE, and to different PEs. With each configuration, clock cycle counts both for sending and receiving the data were measured. The network was otherwise unloaded.

The communication cost clock cycle counts are shown in Fig. 3 for sending (tx) with continuous lines and for receiving (rx) with dashed lines. Only the *inter-PE* communication costs are dependent from the data amount, since data is copied between local data memory and temporary memory buffer. In addition,

Table 3 Used Communication Cost Functions.

	Send cost (clock cycles)	Receive cost (clock cycles)
Intragroup	393	1400
Intergroup	2351	1400
Inter-PE	$3.47x+7796$	$4.57x+15285$

because the data has to be prepared for the network transaction, *inter-PE* costs possess a substantial constant cost. This is due to, for example, serialization of complex data structures prior sending. Since it is known that the data reception procedures for *intragroup* and *intergroup* transfers are exactly the same, their difference is modeled as a context switch cost. It was measured to be 7293 clock cycles. However, the cycle count not only includes the actual RTOS context switch, but also additional time for operations caused by the UML environment.

The formulas for the communication costs were formed from the measured values. Here, we used linear polynomial functions. They are presented in Table 3, where x is the data amount in bytes.

4.3. Test case

As a test case for measuring the accuracy of the exploration we used a Motion-JPEG (M-JPEG) codec described in UML. It consists of ten tasks: *Frame-to-MB*, *DCT*, *Quant*, *CodeIntraMB*, *VLD*, *IQuant*, *IDCT*, *MB-to-Frame*, *VGA-ctrl*, and *Camera*. The encoder consists of the first four tasks and the decoder of the next four. The last two tasks are for display controller and camera.

The task network graph of the used codec is presented in Fig. 4. The data amounts sent in bytes are presented between the tasks. The integer operation count for the most common case of each task used in the *probabilistic* and in the *modulo* models is presented inside the node. For example, the integer operation count of *Frame-to-MB* depends on the source of the data. The input from *Camera* causes execution duration of 151 operations. More commonly, the input from *CodeIntraMB* yields 5970 operations. Here, the dependences between the inputs are type OR.

The probabilities used in the *probabilistic* model are presented beside each output in Fig. 4. If a task has multiple inputs, the input which causes the evaluation of the probability is presented with a dashed line between the input and the probability. In the *VLD* output, probability $1+C$ means that the data packet is always sent once and, in addition, once more with probability C .

We evaluated the accuracy with nine mappings

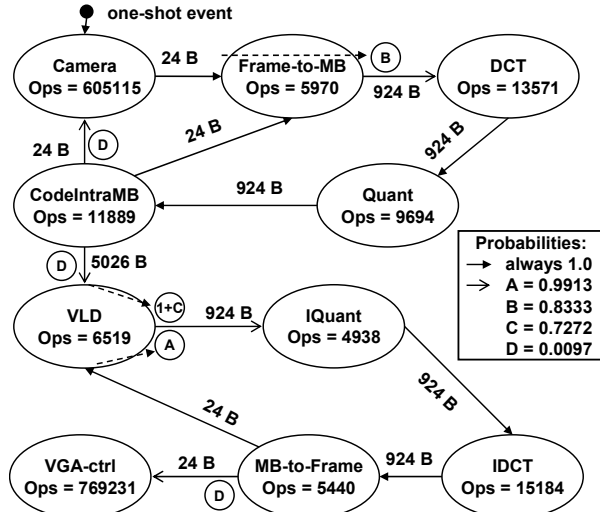


Fig. 4. Task network graph of M-JPEG application with FPGA measured values.

having 1-4 processors with *trace*, *modulo*, and *probabilistic* models. Because of the nondeterministic behavior of the *probabilistic* model, its results were averaged over ten simulations per mapping. In the first two mappings all the tasks were mapped into one processor. Mappings 3 and 4 consisted of two, mappings 5 and 6 of three, and mappings 7-9 of four processors. In mappings 2 and 9 the tasks were separated into multiple task groups (threads) within the processors. In other mappings, all the tasks mapped to one PE were grouped into a single group. The simulation time was one second starting from the first execution of the task *VLD*. The tasks mapped to the same PE were scheduled using FIFO scheduling, which was also used in FPGA execution.

5. Results

Three metrics were used to evaluate the accuracy: total number of executed tasks, PE utilization, and FPS. The error percentages for the total task counts are presented in Fig. 5 and Fig. 6 for each mapping. The errors were calculated for all tasks which were executed over ten times. Thus, *VGA-ctrl* and *Camera* were excluded. The execution counts of the other tasks were in the order of 400-1000 during the simulation.

5.1. Total number of executed tasks

As can be seen from Fig. 5, the errors for the *probabilistic* model are the biggest throughout the mappings. Mappings 2 and 9, in which the tasks are separated into different task groups, have the biggest differences along with mapping 6. Moreover, the average errors for the *modulo* and the *trace* models remain

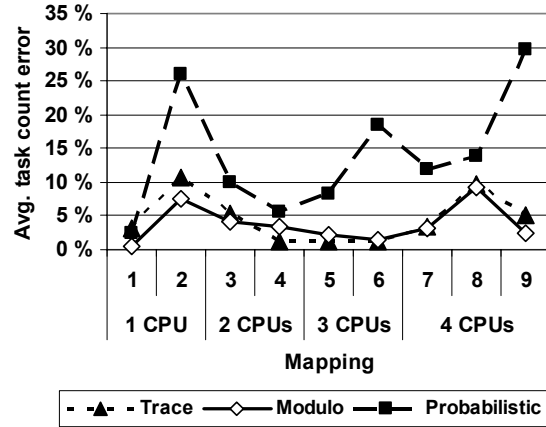


Fig. 5. Average task count errors.

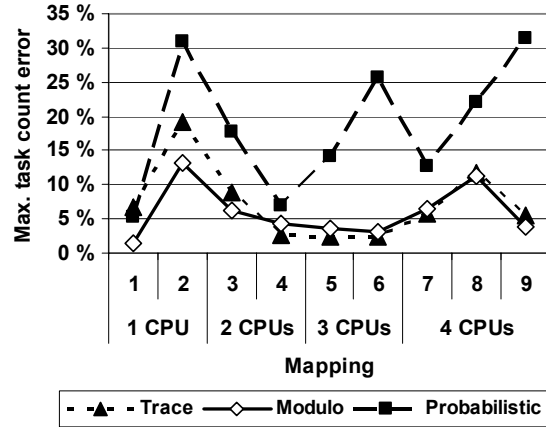


Fig. 6. Maximum task count errors.

within 10% in all mappings, whereas the average error for the *probabilistic* model is at most 30%. In the case of multiple groups in mappings 2 and 9, the experiments revealed a minor bug in the *trace* model execution. Thus, its results are slightly worse than they should be.

As illustrated in Fig. 6, the shapes of the maximum task count error curves closely follow the shapes of the average curves presented in Fig. 5. Again, the *modulo* and the *trace* models give lower error than the *probabilistic* model in all mappings. Except for mappings 2 and 8, also the maximum errors in the *modulo* and the *trace* models remain below 10%, whereas in the *probabilistic* model only the mappings 1 and 4 are able to achieve that limit.

5.2. PE utilization

The PE utilization including the RTOS execution varied from 50% to 100% depending on the mapping. Fig. 7 shows the average PE utilization errors for each

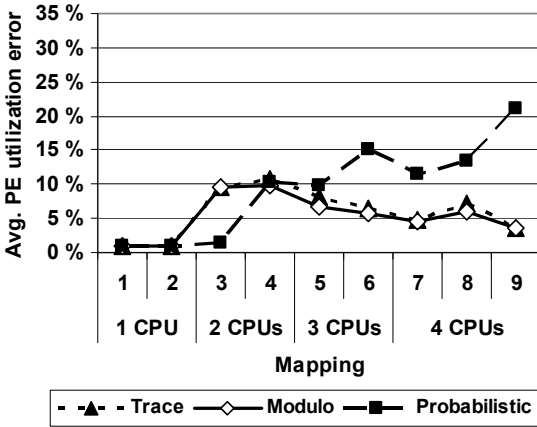


Fig. 7. Average PE utilization errors.

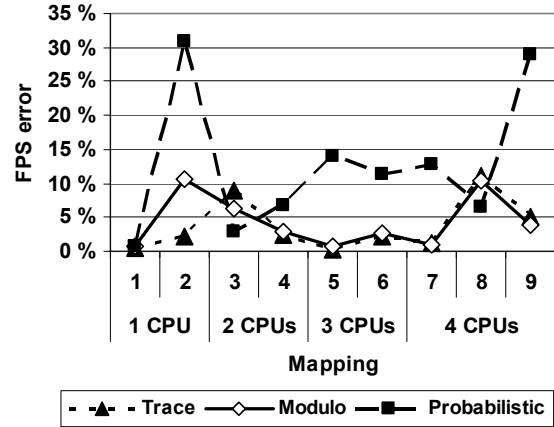


Fig. 9. Errors in frame rate (FPS) estimation.

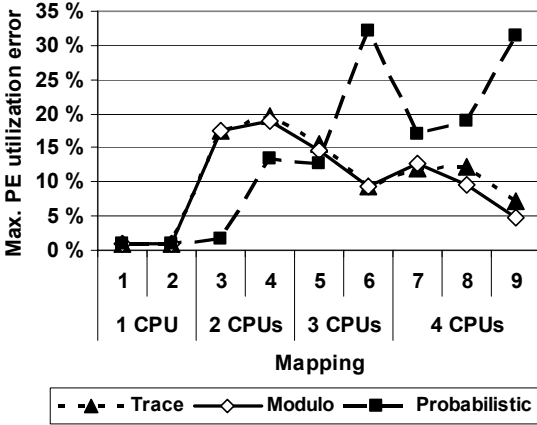


Fig. 8. Maximum PE utilization errors.

mapping. In mapping 3, the error is the smallest for the *probabilistic* model, but in all other mappings its error is the same or worse than the error of the *modulo* model. Furthermore, only in mappings 3 and 4 the *probabilistic* model has smaller error than the *trace* model. The biggest average errors for the *modulo* and the *trace* models are 9.8% and 10.7%, respectively in mapping 4. The biggest average error of the *probabilistic* model is 21.0% obtained in mapping 9.

For the maximum PE utilization errors in Fig. 8, the relations between the models do not differ significantly from the average errors. However, in addition to mapping 3, also in mappings 4 and 5 the utilization errors are the smallest for the *probabilistic* model.

5.3. Frame rate

The frame rate (FPS) varied from 4 to 10 depending on the mapping. The FPS errors for the different mappings are illustrated in Fig. 9. In mappings 3 and 8,

the error is the smallest for the *probabilistic* model. Both the *modulo* and the *probabilistic* models get their biggest errors in the mapping 2. For the *modulo* model it is 10.7% and for the *probabilistic* model 31.0%. Moreover, the biggest error for the *trace* model is 11.2% in mapping 8. All these are significantly smaller than the difference in FPS between the worst and the best mapping, which was 225%. This guarantees that the performance orders of the different mappings and allocations are clearly distinguishable despite the errors that the abstracted models possess. Hence, the exploration algorithms can decide the suitability of various mappings successfully.

5.4. Discussion

It is worth mentioning that when the tasks are spread into multiple groups within one PE, the errors in the *probabilistic* model are considerably bigger than in the other two models. The reason is that in the *probabilistic* model, the execution does not necessarily go fluently in a pipelined manner because of the distinct probabilities for each data output. This causes more context switches, thus too much execution penalties, in the *probabilistic* model.

The average and maximum values of each measured property can be seen from Table 4. The task count errors have been calculated over 72 values (9 mappings, 8 errors in each) and the PE utilization errors over 24 values according to the PE count in each mapping. The average errors for the task count and FPS are 3.5-4 times bigger in the *probabilistic* model than in the *modulo* model and 1.6-3.4 times bigger than in the *trace* model. On the other hand, the average error of the PE utilization is only 2 times bigger in the *probabilistic* model than in the other models.

For reference, the errors were evaluated also with-

Table 4 Summary of Error Values

	Task count		PE utilization		FPS
	Avg	Max	Avg	Max	Avg
Trace	4.6%	19.2%	6.2%	19.6%	3.8%
Modulo	3.8%	13.1%	5.6%	19.0%	4.3%
Prob.	14.1%	31.3%	11.8%	32.1%	12.8%

out the communication costs in mapping 9. It was chosen because it included all the three different communication scenarios. For the *modulo* model, the error in the FPS was 55.0% and in the average PE utilization 15.4% compared to 3.9% and 3.5% when the costs were included. For the *trace* model, the FPS and the average PE utilization errors were 27.0% and 60.7%, respectively. Both the errors with these two models were substantially larger than with the communication costs. For the *probabilistic* model, they were 50.9% for the FPS and 14.0% for the average PE utilization, compared to 28.9% and 21.0% respectively.

The compilation time was significantly longer with *trace* model than with other two models, 9 minutes compared to 12 seconds. However, the application model needs only to be compiled once during the first iteration of the automated exploration. In the rest of the exploration iterations the compilation time is 10 seconds for all models. Further, the communication costs did not notably affect the simulation speed, which was approximately 500 000 cycles per second for *probabilistic* and *modulo* models. Simulation with *trace* model was 10% slower due to model complexity. In earlier study, the speed-up with RTL bus model compared to low-level cycle-accurate HW/SW co-simulation was 230x. However, that was obtained with a slightly different application, but presents the magnitude in the speed-up. Since we here used a transaction level bus model instead of an RT-level model, the value is pessimistic.

6. Conclusions

In this paper, we introduced a new cost factor, a multi-level communication cost to the design space exploration model. In addition, we evaluated the accuracy of our model for design space exploration with nine mappings against FPGA execution.

It is shown that the multi-level communication cost increases the accuracy of the exploration significantly, and it cannot be ignored under any circumstances. Further, by separating the costs from the execution time of a task, accurate results can be achieved with arbitrary mappings. The average error in FPS was 3.8% for the *trace* model, 4.3% for the *modulo* model, and 12.8%

for the *probabilistic* model while the simulation speed-up was 230x. Together the accuracy and the speed-up outperform the results in the related work.

In future, the accuracy must be evaluated also with more applications, systems having more PEs, and systems including also HW accelerators. In order to take full advantage of the accuracy of the *modulo* model, its generation must be automated. Moreover, the suitability of the *modulo* model for irregular applications must be evaluated and the execution of the trace model corrected in the case of multiple groups.

References

- [1] A.D. Pimentel, "The Artemis workbench for system-level performance evaluation of embedded systems," *Int. Journal of Embedded Systems*, vol. 1, no. 7, 2005.
- [2] S. Mohanty and V. Prasanna, "Rapid system-level performance evaluation and optimization for application mapping onto SoC architectures," in *Proc. IEEE Int. ASIC/SOC Conf.*, Sept. 2002, pp. 160-167.
- [3] A. Bouchhima, I. Bacivarov, W. Youssef, M. Bonaciu, and A. Jerraya, "Using abstract CPU subsystem simulation model for high level HW/SW architecture exploration," in *Proc. ASP-DAC.*, vol. 2, Jan. 2005, pp. 969-972.
- [4] K. Lahiri, A. Raghunathan, and S. Dey, "Design space exploration for optimizing on-chip communication architectures," *IEEE TCAD*, vol. 23, no. 6, June 2004, pp. 952-961.
- [5] S. Mahadevan et al., "A network traffic generator model for fast network-on-chip simulation," in *Proc. DATE*, Mar. 2005, pp. 780-785.
- [6] A. Bobrek, J.J. Pieper, J.E. Nelson, J.M. Paul, and D.E. Thomas, "Modeling shared resource contention using a hybrid simulation/analytical approach," in *Proc. DATE*, vol. 2, Feb. 2004, pp. 1144-1149.
- [7] S. Kim, C. Im, and S. Ha, "Schedule-aware performance estimation of communication architecture for efficient design space exploration," *IEEE TVLSI Systems*, vol. 13, no.5, May 2005, pp. 539-552.
- [8] T. Kangas et al., "UML-based Multiprocessor SoC Design Framework," *ACM TECS*, vol. 5, no. 2, May 2006, pp. 281-320.
- [9] G. Kahn, "The semantics of a simple language for parallel programming," in *Proc. IFIP Congress 74*, Aug. 1974, pp. 471-475.
- [10] T. Kangas, J. Riihimäki, E. Salminen, K. Kuusilinna, and T.D. Hämäläinen, "Using a Communication Generator in SoC Architecture Exploration," in *Proc. Int. Symposium of System-on-Chip*, Nov. 2003, pp. 105-108.
- [11] eCos homepage, Oct. 2006, <http://ecos.sourceforge.org>