Valutazione di efficienza ed eseguibilitá dell'allocazione e scheduling in un Multi-Processor Systems-on-Chip

Measuring Efficiency and Executability of Allocation and Scheduling in Multi-Processor Systems-on-Chip

Luca Benini, Davide Bertozzi, Alessio Guerri, Michela Milano, Francesco Poletti

SOMMARIO/*ABSTRACT*

I Multi-Processor Systems-on-Chips (MPSoCs) stanno diventando sempre piú complessi, e l'allocazione e lo scheduling di applicazioni multi-task sulle unitá computazionali é di cruciale importanza per rispettare i vincoli di esecuzione e di consumo di potenza. La definizione di modelli astratti delle componenti del sistema e lo sviluppo di tecniche avanzate per l'ottimizzazione possono aiutare nel velocizzare la ricerca di soluzioni ottime. In [3] é stato proposto, per risolvere il problema in analisi, un approccio ibrido basato sull'integrazione di Programmazione Intera (IP) e Programmazione a Vincoli (CP), dove IP realizza l'allocazione e CP lo scheduling. La soluzione ibrida decompone il problema e sceglie, per ogni sottoproblema, l'algoritmo risolutivo piú idoneo. É stata dimostrata l'efficienza dell'approccio ibrido rispetto agli approcci puri CP e IP. In questo lavoro utilizziamo una piattaforma virtuale MPSoC per trovare le differenze tra la formulazione del problema e il mondo reale al fine di stimare il loro impatto sui risultati. Abbiamo quindi valutato l'accuratezza e l'eseguibilitá delle soluzioni usando un simulatore della piattaforma MPSoC.

Multi-Processor Systems-on-Chips (MPSoCs) are becoming increasingly complex, and mapping and scheduling of multi-task applications on computational units is key to meeting performance constraints and power budgets. Abstract models of system components and deployment of advanced algorithmic techniques for the optimization problem can provide for fast design space exploration and for optimal solutions. In [3] we have proposed an efficient hybrid approach for solving the problem based on an integrated Constraint Programming (CP) and Integer Programming (IP) solution where IP performs the allocation and CP the scheduling. The hybrid solution exploits problem decomposition and chooses for each subproblem the best algorithm for solving it given its structure. We have proved the efficiency of the hybrid approach w.r.t. both stand alone CP and IP approaches. In this paper, we go a step further and exploit an accurate MPSoC virtual platform for capturing mismatches between problem formulation and real-life systems, and for assessing their impact on expected performance. We therefore evaluate the efficiency and the executability of the solution found by the algorithm using a MPSoCs platform simulator.

Keywords: Constraint Programming, Integer Programming, Allocation and Scheduling, Benders Decomposition, Multi-Processor Systems-on-Chips

1 Introduction

Constraint Satisfaction Problems - CSPs and Constraint Optimization Problems - COPs are difficult to solve and efficiency is an issue. Therefore, the experimentation process on algorithms solving CSPs and COPs is in general aimed at measuring the computational time. Indeed, this is not the only important parameter.

Problem models might contain simplifying hypothesis leading the algorithm to produce a solution (called off-line solution) which could be different from the one that can be executed in the real world (called on-line solution). In the extreme case, the off-line solution is even not executable in the real world, i.e., it violates problem constraints. This means that it is possible that the assumptions made while modelling the problems are so strong that the solution does not correspond at all to the reality. To establish a solid optimization methodology, differences between the on-line and off-line solutions should be measured and evaluated. Therefore, if the model is simplified, as it happens in all difficult cases, the experimentation, besides

efficiency, should measure the solution quality.

Moving from these considerations, in this paper we present an application of allocation and scheduling techniques to Multi Processor Systems on Chip - MPSoCs [9] where an entire system is integrated onto the same silicon die, consisting of multiple processor cores (the computational units), a memory hierarchy and channels for interprocessor communication. The main purpose of this paper is to define a robust experimental setting for validating the design-time allocation and scheduling on an accurate simulation platform.

Mapping and scheduling require a pre-characterization of the application, abstracted as a task graph, i.e. a collection of nodes and edges. The nodes represent functional program abstractions and arcs in the graph correspond to precedence and communication constraints. The application considered in this paper is a pipeline of tasks, representing the structure of a typical throughput-optimized streaming multi-media application.

Our theoretical and experimental frameworks are aimed at measuring the efficiency of the proposed allocation and scheduling algorithms and the accuracy of the model by checking if the off-line solution is executable and, in that case, how far the on-line solution is from the off-line one. For this purpose we used an MPSoC virtual platform, called MP-Arm [2, 8].

We show the benefits of validating the model accuracy on a virtual platform for the purpose of defining and refining an effective allocation and scheduling methodology on MPSoCs.

2 Problem description

Recent advances in very large scale integration (VLSI) of digital electronic circuits have made it possible to integrate more than a billion of elementary devices onto a single chip, thereby enabling the development of low-power, low-cost, high-performance single-chip multi-processors. These devices, called multi-processor systems-on-chip (MPSoCs), are finding widespread application in embedded systems (such as cellular phones, automotive control engines, etc.) where they are employed as special-purpose computing engines. In other words, once deployed in field, they always run the same application, in a well-characterized context. It is therefore possible to spend a large amount of time for finding an optimal allocation and scheduling off-line and then deploy it on the field. For this reason, many researchers in digital design automation have explored complete approaches for allocating and scheduling pre-characterized workloads on MPSoCs [9], instead of using on-line, dynamic (sub-optimal) schedulers [7, 6].

The multi-processor system we consider consists of a pre-defined number of distributed computation nodes,

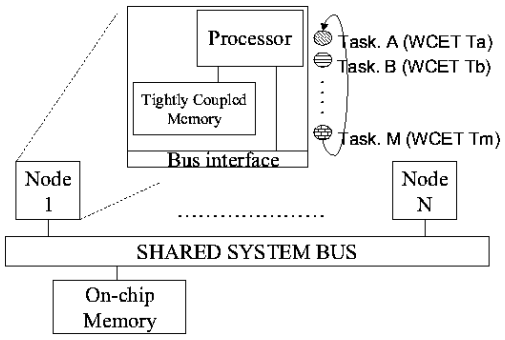

Figure 1: Single chip multi-processor architecture.

as depicted in Figure 1. All nodes are assumed to be homogeneous and made by a processing core and by a tightly coupled local memory. This latter is a low access cost scratchpad memory, which is commonly used both as hardware extension to support message passing and as a storage means for computation data and processor instructions which are frequently accessed.

Unfortunately, the scratchpad memory is of limited size, therefore data in excess must be stored externally in a remote on-chip memory, accessible via the bus. The bus for state-of-the-art MPSoCs is a shared communication channel, and serialization of bus access requests of the processors (the bus masters) must be carried out by a centralized arbitration mechanism. The bus is re-arbitrated on a transaction basis, based on several policies (fixed priority, round-robin, latency-driven, etc.).

Each task also has three kinds of memory requirements. **Program Data**: storage locations are required for computation data and for processor instructions. They can be allocated either on the local scratchpad memory or on the remote on-chip memory. **Internal State**: when needed, an internal state of the task can be stored either locally or remotely. **Communication queues**: the task needs queues to transmit and receive messages to/from other tasks, eventually mapped on different processors. In the platform we are considering, such queues should be allocated only on local memories, in order to implement an efficient inter-processor communication mechanism. Finally, the communication requirements of each task are automatically determined depending on the size of communication data and on the physical location of computation data in scratchpad or remote memory.

A task graph representation of the target application is input to our methodology. For each node/task, the average case execution time (ACET) is extracted by means of functional simulation of a significant number of runs and plays a critical role whenever application real time (RT) constraints (expressed here in terms of minimum required throughput) are to be met. In fact, tasks are scheduled on each processor based

on a time-wheel. The sum of the ACETs of the tasks for one iteration of the time wheel must not exceed time period RT, which is the same for each processor since the minimum throughput is an application requirement. We consider the ACET since it is likely to better match the task durations measured on the validation virtual platform; when our stable methodology will be used for design purposes, the worst case execution time (WCET) should be used instead, so as to avoid temporal constraints violation. The methodology proposed in this paper has been applied to a synthetic signal processing pipeline, a widely used multimedia application. Task parameters have been derived from a real video graphics pipeline processing pixels of a digital image, but it can be easily extended to any task graph. In the open literature, approaches to this kind of problems usually make very strong simplifying assumptions, like infinite number of processing units, zero time communication or unlimited memory capacity [4, 5, 1]. In addition, they often do not consider the real implementation of the solution they produce. We make simplifying assumptions to derive a problem model as well, but we also include a validation stage in our framework, in order to assess the impact of such approximations and verify the mismatch between off-line and on-line solution.

After the optimization algorithm produces an off-line solution, we used an MPSoC virtual platform, called MP-Arm [2, 8], to check if it is indeed executable. The target platform introduces restrictions on the solutions that can be executed, i.e., it introduces constraints that are hereafter denoted as *execution constraints*. If the model does not take them into account, the model represents a relaxation of the real problem and thus provides a super-optimal solution. The execution constraint we are considering for scheduling the application under test states that the solution should be representable as a list of tasks or a list of task priorities.

3 Model definition

The two main approaches followed by the system design community when facing a COP are: (1) either modelling and solving the problem as an Integer Program whatever the problem structure is or (2) using a special purpose heuristic algorithm requiring sophisticated debugging and tuning.

The intuition behind our approach is to decompose the problem and exploit the structure of each component to chose the best algorithm to solve it. The problem we are facing can be split into 2 problems: (1) the allocation of tasks to processors and memory requirement to storage devices minimizing the communication cost and (2) the scheduling sub-problem minimizing its makespan. The structure of these problems suggests to model and solve the first through Integer Linear Programming (IP), while the second is better faced through Constraint Programming (CP) techniques. The two solvers interact through the generation of no-goods. This technique is described in detail in [3]. In section 5 we will validate the strength of this approach comparing it with pure CP and IP approaches.

3.1 Allocation problem model

The allocation problem is the problem of allocating n tasks to m processors and memory requirements to storage devices. The objective function is the minimization of the amount of data transferred on the bus. Since in this problem we do not have temporal constraints, we can model the problem as an IP model.

We consider four decision variables: T_{ij}, taking value 1 iff task i executes on processor j; Y_{ij}, taking value 1 iff task i allocates the program data on the scratchpad memory of processor j; Z_{ij}, taking value 1 iff task i allocates the internal state on the scratchpad memory of processor j; X_{ij}, taking value 1 iff tasks i and $i+1$ execute on different processors, one of them being processor j.

The linear constraints introduced in the model are:

$$\sum_{j=1}^{m} T_{ij} = 1, \forall i \in 1 \dots n \quad (1)$$

$$T_{ij} + T_{i+1j} + X_{ij} - 2K_{ij} = 0 \ , \forall i \in 1 \dots n \ , \forall j \quad (2)$$

Constraints (1) state that each process can execute only on a processor, while constraints (2) state that X_{ij} can be equal to 1 iff $T_{ij} \neq T_{i+1j}$, that is, iff task i and task $i+1$ execute on different processors. K_{ij} are integer binary variables forcing the sum $T_{ij} + T_{i+1j} + X_{ij}$ to be either equal to 0 or 2 (in fact, X_{ij} is the `exor` of T_{ij} and T_{i+1j}).

We also add to the model the constraints stating that $T_{ij} = 0 \Rightarrow Y_{ij} = 0, Z_{ij} = 0$, and, for each group of consecutive tasks whose execution times sum exceeds the RT requirement, we introduce in the model a constraint preventing the solver to allocate all the tasks in the group to the same processor. To generate these constraints, we find out all groups of consecutive tasks whose execution times sum exceeds RT. Constraints are the following:

$$\sum_{i \in S} Dur_i > RT \Rightarrow \sum_{i \in S} T_{ij} \leq |S| - 1 \ \forall j \quad (3)$$

This is a relaxation of the subproblem, added to the master problem to prevent the generation of trivially infeasible solutions for the overall problem (solutions allocating most communicating tasks on few processors to minimize communication cost, but at the same time violating real-time constraints).

The objective function is the minimization of the communication cost, i.e., the total amount of data transferred on the bus for each pipeline iteration. This amount consists of three contributions: when a task allocates its program data in the remote memory, it reads them throughout the execution time; when a task allocates its internal state in the remote memory, it reads it before its execution and writes it immediately after the execution; if two consecutive tasks execute on different processors, their communication messages must be transferred through the bus from the communication queue of one processor to the other. Using the decision variables described above, we have a contribution respectively when: $T_{ij} = 1, Y_{ij} = 0$; $T_{ij} = 1, Z_{ij} = 0$; $X_{ij} = 1$. Therefore, the objective function is to minimize:

$$\sum_{j=1}^{m}\sum_{i=1}^{n} \Big(Mem_i(T_{ij} - Y_{ij}) + 2 \times State_i(T_{ij} - Z_{ij}) + \\ +(Data_i X_{ij})/2)\Big) \quad (4)$$

where Mem_i, $State_i$ and $Data_i$ are the amount of data used by task i to store respectively the program data, the internal state and the communication queue.

3.2 Scheduling problem model

Once tasks have been allocated to the processors, we need to schedule process execution. Since we are considering a pipeline of tasks, we need to analyze the system behavior at working rate, that is when all processes are running or ready to run. To do that, we consider several instantiations of the same process; to achieve a working rate configuration, the number of repetitions of each task must be at least equal to the number of tasks n; in fact, after n iterations, the pipeline is at working rate. So, to solve the scheduling problem, we must consider at least n^2 tasks (n iterations for each process), see Figure 2. In the scheduling

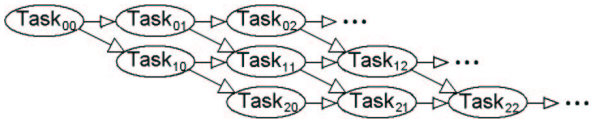

Figure 2: Precedence constraints among the activities

problem model, for each task $Task_{ij}$ we introduce a variable A_{ij}, representing the computation activity of the task. A_{ij} is the j-th iteration of the i-th process. Once the allocation problem is solved, we statically know if a task needs to use the bus to communicate with another task, or to read/write computation data and internal state in the remote memory. In particular, each activity A_{ij} must read the communication queue from the activity A_{i-1j}, or from the pipeline input if $i = 0$. To schedule these phases, we introduce in the model the activities In_{ij}. If a process requires an internal state, the state must be read before the execution and written after the execution: we therefore introduce in the model the activities RS_{ij} and WS_{ij} for each task i requiring an internal state. The duration of these activities depends on whether the data are stored in the local or the remote memory (data transfer through the bus needs more time than the transfer of the same amount of data using the local memory) but, after the allocation, these times can be statically estimated.

Figure 2 depicts the precedence constraints among the tasks. Each task $Task_{ij}$ represents the activity A_{ij} possibly preceded by the internal state reading activity RS_{ij}, and input data reading activity In_{ij}, and possibly followed by the internal state writing activity WS_{ij}. The precedence constraints among the activities are:

$$A_{i,j-1} \prec In_{ij} \ , \ \forall \ i,j \quad (5)$$
$$In_{ij} \prec A_{ij} \ , \ \forall \ i,j \quad (6)$$
$$A_{i-1,j} \prec In_{ij} \ , \ \forall \ i,j \quad (7)$$
$$RS_{ij} \preceq A_{ij} \ , \ \forall \ i,j \quad (8)$$
$$A_{ij} \preceq WS_{ij} \ , \ \forall \ i,j \quad (9)$$
$$In_{i+1,j-1} \prec A_{ij} \ , \ \forall \ i,j \quad (10)$$
$$A_{i,j-1} \prec A_{ij} \ , \ \forall \ i,j \quad (11)$$

where the symbol $\prec$ means that the activity on the right should follow the activity on the left, and the symbol $\preceq$ means that the activity on the right must start as soon as the execution of the activity on the left ends: i.e., $A \prec B$ means $Start_A + Dur_A \leq Start_B$, and $A \preceq B$ means $Start_A + Dur_A = Start_B$.

Constraints (5) state that each task iteration can start reading the communication queue only after the end of its previous iteration. Constraints (6) state that each task can start only when it has read the input data, while constraints (7) state that each task can read the input data only when the previous task has generated them. Constraints (8) and (9) state that each task must read the internal state just before the execution and write it just afterwards. Constraints (10) state that each task can execute only if the previous iteration of the following task has read the input data; in other words, it can start only when the memory allocated to the process to store the communication queue has been freed. Constraints (11) state that the iterations of each task must execute in order.

We also introduced the real time requirement constraints $Start(A_{ij}) - Start(A_{i,j-1}) \leq RT \ , \ \forall \ i,j \ ,$ whose relaxation is used in the allocation problem model. The time elapsing between two consecutive executions of the same task can be at most RT.

Each processor is modelled as a unary resource, that is a resource with capacity one. To model the bus, we

made a simplifying assumption. In the real hardware platform, the bus is shared among the processors and can be considered as a unary resource, i.e., it is accessed by a single processor at a time by means of an arbitration mechanism, which serializes bus access requests. However, the arbiter grants the bus at the fine granularity of individual bus transactions (single or burst reads and writes). On the contrary, communication requirements annotated on the task graph of the application are expressed at a much coarser granularity, i.e. the average bandwidth requirements for each couple of communicating tasks. We bridge the abstraction gap considering the bus as a shared resource which can be virtually accessed by many processes at a time. Each process consumes a fraction of the maximum available bandwidth, which limits the number of processes concurrently using the bus.

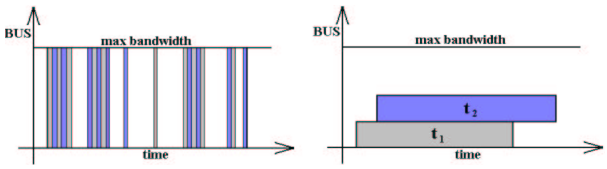

Figure 3: Bus allocation in a real processor (left) and in our model (right)

Figure 3 depicts the assumption done. The leftmost figure represents the bus allocation in a real processor, where the bus is assigned to different tasks at different times. Each task, when owning the bus, uses its entire bandwidth. The rightmost figure, instead, represents how we model the bus. Clearly the areas of the resource required by each task in the two figures are the same. The bus arbitration mechanism will then transform the bus allocation into the interleaving of fine granularity bus transactions on the real platform.

According to this assumption, to model the reading of computation data stored in the remote memory, we evaluate the real bus requirement B for each task (whose duration is d) and we simply stretch this requirement for its entire duration, i.e., the task will require B/d bus bandwidth for its entire duration. Besides, each task starts by reading its input data and its internal state, and completes by updating its internal state to memory. Here, we assume that such reads can rely on $1/N$-th of the total bus bandwidth, where N is the number of processors. We therefore derive an approximated model of the bus activity.

4 Simplifying assumptions

In this section we show which simplifying assumption have been done while characterizing the application and modelling the problem. In order to be able to generate a schedule off-line, we must estimate the activity durations as well as consider some simplifying assumptions to make the problem solvable using the models described so far. The problem that arises when considering these assumptions is that the durations might be inaccurate and the model might not exactly describe the considered problem, so we must check a posteriori if the schedule is feasible, executable and the mismatch between actual system behaviour and theoretical results.

4.1 Activity durations

Each execution run of the same processing activity usually has a different duration. We need to find a mean value for each activity. Here we consider different ways to deduce these values depending on the kind of the activities introduced in Section 3.2, on the amount of data to be transferred and on where they are stored. Considering some parameters typical of the platform we take into account (e.g. the cache miss percentage, the number of clock cycle to read/write a single data from the local/remote memory), we are able to define some formulae to compute a mean value for the duration of the activities.

4.2 Modelling the Bus

As introduced in Section 3, modelling bus allocation at the fine granularity of individual bus transaction interleaving would make the problem overly complex. Therefore a more abstract bus model was devised, thus also bridging the abstraction gap with our high level task models.

Whenever predictable performance is needed for applications, it is important to avoid high levels of congestion on the bus, since this makes completion time of bus transactions much less predictable. Moreover, under a low congestion regime, performance of state-of-the-art shared busses scales almost in the same way as that of advanced busses with topology and communication protocol enhancements. Finally, bus modelling turns out to be simpler under these working conditions. Communication cost is therefore critical for determining overall system performance, and will be minimized in our task allocation framework.

Assuming that the bus is a shared resource, we do not take in account the time spent by the bus arbitration mechanism to schedule bus accesses; in fact this time is hard to estimate since it strongly depends on the number of processes requiring the bus.

4.3 Pipeline iterations

In this problem we want to schedule a sequence of tasks represented in a task graph. If the task graph contains a finite and known number of nodes (tasks) it is simple to transform an off-line schedule in an on-line

one. We simply have to substitute the on-line scheduler with the optimal off-line scheduler. In the application considered here, we have to schedule a pipeline of tasks, and the number of repetitions is not known in advance. Therefore, either the off-line solution is expressible with priorities (and therefore the on-line solution becomes optimal) or the off-line solution is produced for a single pipeline cycle (at working rate) and repeated several times.

5 Experiments performed

In this section we describe the experiments performed for evaluating both the efficiency of the solving algorithm and the accuracy of the model. In particular, for each assumption described in section 4, we check how much it influences the results and their applicability.

We generate a large variety of problems, varying both the number of tasks and processors. All the results presented are the mean over a set of 10 problems instances for a given task or processor number. All problems have a solution. We performed the experiments on a 2GHz P4 with 512 Mb RAM, using ILOG CPLEX 8.1 and ILOG Solver 5.3 as solving tools.

5.1 Measuring efficiency

To validate the choice of our solving algorithm we have to compare its efficiency with alternative approaches. To motivate the decomposition of the problem we compare the results obtained using our hybrid model (Hybrid in the following) with results obtained using only CP and IP alone to solve the overall problem. Actually, since the first experiments showed that both CP and IP were not able to find a solution, except for the easiest instances, within 15 minutes, we simplified these models removing some variables and constraints; even with these simplifications, IP was still not able to find a solution. We found that CP is not comparable (except when the number of tasks and processors is low) with Hybrid being the search time several order of magnitude greater than the Hybrid one. Furthermore, for problems with 6 tasks or 3 processors and more, CP can find the solution only in the 50% or less of the cases. It is therefore profitable to use a decomposition technique to solve the problem.

We have now to motivate the use of both IP and CP to solve the sub-problems. We solved the two sub-problems using the same technique (CP and IP) and in Figure 4 we compare these results with Hybrid.

Times are expressed in seconds and the y-axis has a logarithmic scale. (The comparison for different number of processors has the same behaviours). We can see that these algorithms are not comparable with Hybrid, except when the number of tasks and processors is low. As soon as the number of tasks and/or processors grows, IP and CP performance worsens and their search times become order of magnitudes higher w.r.t. Hybrid. Also in these experiments, IP approach was able to find a solution only in a fraction of the considered instances.

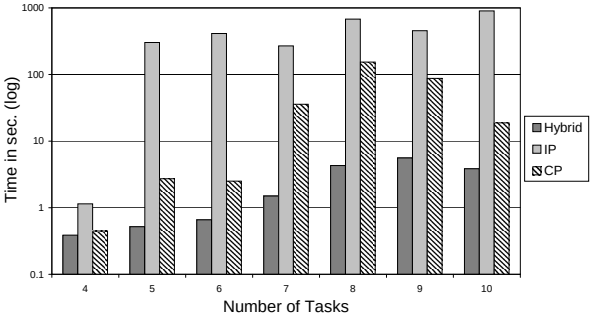

Figure 4: Comparison between algorithms search times for different task number

5.2 Measuring accuracy on activity duration

To validate the accuracy of the pre-characterization and its impact on the computed schedule we compared the activity durations proposed by the scheduler and the simulator. To simulate the activity duration we used parameters from the simulation and we compute the average duration on 100 runs.

Table 1 shows the percentage of accuracy (ratio of the durations) for each kind of activity and for the throughput. As we can see, activities accuracy is very high and this leads to an high throughput accuracy, that is the most important parameter to be taken into consideration, since we are working in scenario with RT constraints. Clearly, if the accuracy were low, the should be a feed back of the pre-characterization phase, in order to compute more realistic activity durations.

Activity	Accuracy
Processing	99.5%
Data read	99.5%
State read/write	96%
Throughput	95%

Table 1: Activity duration accuracy

5.3 Verifying executability

Once we have verified the correspondence between the scheduler and the simulator activity durations, we can focus our analysis on the executability of the optimal schedule, checking if the off-line schedule found by the scheduler can be really executed by the MPSoC platform. A schedule of tasks to be repetitively performed on a data stream of unknown length is executable only if it is periodic. In particular, if the sequence contains only one iteration for each activity (the period is

length one), providing the simulator with a task priority list derived from the off-line sequence ensures that all constraints and RT requirements will be satisfied also by the on-line schedule, given that the accuracy on execution time estimation is very high, as shown in subsection 5.2. We found that, in over 90% of the cases out of a set of 100 problems, after the initial set-up stage, the first off-line schedule found by our tool is periodic thus executable. If it is not the case, it is still possible that more than one optimal solution exists. Our efforts are currently focusing toward verifying that, for each instance, at least one periodic optimal solution exists, as well as trying to prove it theoretically. Concerning this point, we solved again the instances for which a periodic solution was not found inserting executability constraints in the model, in order to find an executable schedule. We measured the difference between the throughputs of the two schedules. We found that, even if for some instances the executable schedule is also optimal, in general it is not the case. Figure 5 depicts the probability (y-axis)

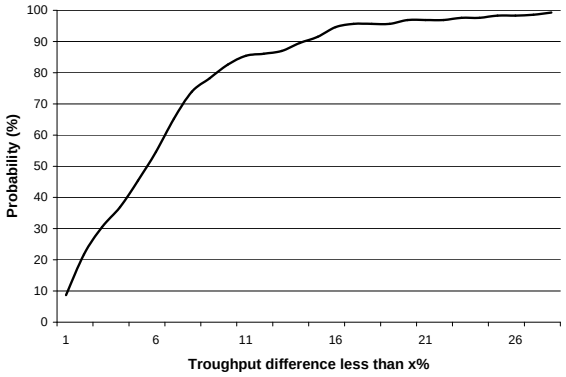

Figure 5: Probability of throughput difference

for the difference between the throughputs of the optimal (but not periodic) solution and the periodic one to be equal or less than the corresponding value in the x-axis (in %). As an example, the probability for the difference to be less than 15% is 90%. We can see that, for most of the cases, the difference is within 10%.

6 Conclusions

We have analyzed the problem of allocating and scheduling a pipelined task graph on a MPSoC. Besides the traditional assessment of efficiency of the algorithm applied to an NP-hard problem, other aspects should be taken into account for this domain. First, in the pre-characterization phase of the application, the duration of tasks is estimated. We should indeed assess the accuracy of such estimations to come up with a meaningful solution. In the performed experiments, the pre-characterization phase has been accurate enough since the estimated throughput differs from the real one of the 5%, acceptable in real applications. The assumptions made on the bus are very strong, but do not affect the quality of the result, because the bus utilization is kept low during the whole application. Concerning executability, even though often the first schedule found is periodic, some non periodic exceptions have been found. Future work will focus on proving that exists, for each instance, at least one optimal periodic schedule.

REFERENCES

[1] B.M. Al-Hashimi, A. Andrei, P. Eles, M.T. Schmitz, and Z. Peng. Quasi-static voltage scaling for energy minimization with time constraints. In *Proceedings of the IEEE Design and Test in Europe Conference*, pages 230–235, Munich, 2005.

[2] F. Angiolini, L. Benini, D. Bertozzi, M. Loghi, and R. Zafalon. Analyzing on-chip communication in a MPSoC environment. In *Proceedings of the IEEE Design and Test in Europe Conference (DATE)*, pages 752–757, Paris, February 2004.

[3] L. Benini, D. Bertozzi, A. Guerri, and M. Milano. Allocation and scheduling for MPSoCs via decomposition and no-good generation. In *Procs. of the 11th Intern. Conference on Principles and Practice of Constraint Programming (CP 2005), to appear*, Sitges, Spain, Oct. 2005.

[4] S. Chakraborty, S. Kunzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. pages 190–195, Munich, March 2003.

[5] K.S. Chatha and K. Srinivasan. An ILP formulation for system level throughput and power optimization in multiprocessor SoC architectures. In *17th International Conference on VLSI Design*, pages 255–260, Arizona, 2004.

[6] K. Compton and S. Hauck. Reconfigurable computing: A survey of systems and software. *ACM Computing Surveys*, 34(2):171–210, 1999.

[7] D. A. Culler and J. P. Singh. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, 1999.

[8] P. Marchal, A. Poggiali, and F. Poletti. Flexible hardware/software support for message passing on a distributed shared memory architecture. In *Procs. of the IEEE Design and Test in Europe Conference*, pages 736–741, Munich, 2005.

[9] W. Wolf. The future of multiprocessor systems-on-chips. In *In Procs. of the 41st Design and Automation Conference - DAC 2004*, pages 681–685, San Diego, CA, USA, June 2004.