Communication-Aware Allocation and Scheduling Framework for Stream-Oriented Multi-Processor Systems-on-Chip

Martino Ruggiero[†], Alessio Guerri[†], Davide Bertozzi[‡], Francesco Poletti[†] and Michela Milano[†]

[†] University of Bologna, DEIS, Viale Risorgimento, 2 - Bologna 40136 (Italy)

[‡] University of Ferrara, via Saragat, 1 - Ferrara 40132 (Italy)

Abstract

This paper proposes a complete allocation and scheduling framework, where an MPSoC virtual platform is used to accurately derive input parameters, validate abstract models of system components and assess constraint satisfaction and objective function optimization. The optimizer implements an efficient and exact approach to allocation and scheduling based on problem decomposition. The allocation subproblem is solved through Integer Programming while the scheduling one through Constraint Programming. The two solvers can interact by means of no-good generation, thus building an iterative procedure which has been proven to converge to the optimal solution. Experimental results show significant speedups w.r.t. pure IP and CP exact solution strategies as well as high accuracy with respect to cycle accurate functional simulation. A case study further demostrates the practical viability of our framework for real-life systems and applications.

1. Introduction

Forthcoming multi-processor System-on-Chip (MPSoC) platforms with dozens of embedded processors will require a new tool flow for efficient software development and validation. In particular, the integration of very powerful optimization tools is expected to overcome the limitations of ad-hoc approaches to the traditional task-to-architecture mapping problem, and to provide efficient solutions in reasonable time.

The synthesis of system architectures has been extensively studied in the past. Mapping and scheduling problems on multi-processor systems have been traditionally modelled as integer linear programming problems. In general, even though ILP is used as a convenient modelling formalism, there is consensus on the fact that pure ILP formulations are suitable only for small problem instances, i.e. task graphs with a reduced number of nodes, because of their high computational cost. For this reason, heuristic approaches are widely used, such as genetic algorithms, simulated annealing and Tabu search[11].

Complete approaches, which compute the optimal solution at the cost of an increasing computational cost, can be attractive for statically scheduled systems, where the solution is computed once and applied throughout the entire lifetime of the system.

This is the case of signal processing and multimedia application. Pipelining is one common workload allocation policy for increasing throughput of such applications, and this explains why research efforts have been devoted to extending mapping and scheduling techniques to pipelined task graphs[6].

Moving from these considerations, in this paper we present a novel framework for allocation and scheduling of pipelined task graphs on MPSoCs. We target a general template for distributed memory embedded systems, where each processor has a local scratch-pad memory for fast and energy-efficient access to program data. In such systems, the communication architecture is becoming a critical component for abstract platform modelling. Interaction of multiple traffic patterns on the system bus causes congestion and hence unpredictable communication latencies. Neglecting this behaviour in high level optimization tools for allocation and scheduling might lead to unacceptable deviations of real performance metrics with respect to predicted ones and to the violation of real-time constraints.

Our framework targets state-of-the-art shared busses for low-end MPSoCs and is communication-aware in many senses. First, it discriminates among allocation and scheduling solutions based on the communication cost. Second, a high level model of the bus is derived and working conditions guaranteeing a predictable behaviour of the system interconnect are identified through accurate functional simulation. The allocation and scheduling solutions provided by our framework force the system to work within these conditions, by using them as constraints for the combinatorial optimization problem.

Our allocation and scheduling framework is based on problem decomposition and deploys techniques mutuated from the Artificial Intelligence and the Operations Research community: the allocation subproblem is solved through Integer Programming while the scheduling one through Constraint Programming. More interestingly, the two solvers can interact with each other by means of no-good generation, thus building an iterative procedure which has been proven to converge producing the optimal solution. Experimental results show significant speedups w.r.t. pure IP and CP solution strategies.

Finally, we deploy an MPSoC virtual platform to validate the results of the optimization steps and to more accurately assess constraint satisfaction and objective function optimization. In multi-processor systems, we believe this validation phase is critical in order to check modelling assumptions and make sure that second-order effects and/or modelling approximations do not impair optimizer-predicted performance (e.g., a required throughput).A GSM-based demonstrator shows the practical viability of our framework for real-life systems and applications.

2. Previous work

Mapping and scheduling problems on multi-processor systems have been traditionally modelled as integer linear programming problems, and addressed by means of IP solvers. An early example is represented by the SOS system, which used mixed integer linear programming technique (MILP) [2]. Partitioning with respect to timing constraints has been addressed in [3]. A MILP model that allows to determine a mapping optimizing a trade-off function between execution time, processor and communication cost is reported in [4]. An hardware/software co-synthesis algorithm of distributed real-time systems that optimizes the memory hierarchy (caches) along with the rest of the architecture is reported in [5].

Pipelining is a well known workload allocation policy in the signal processing domain. An overview of algorithms for scheduling pipelined task graphs is presented in [6]. ILP formulations as well as heuristic algorithms are traditionally employed. In [7] a retiminig heuristic is used to implement pipelined scheduling, while simulated annealing is used in [8]. The work in [9] is based on Constraint Logic Programming to represent system synthesis problem,

and leverages a set of finite domain variables and constraints imposed on these variables. Pipelined execution of a set of periodic activities is also addressed in [10], for the case where tasks have deadlines larger than their periods.

The complexity of pure ILP formulations for general task graphs has led to the deployment of heuristic approaches. A comparative study of well-known heuristic search techniques (genetic algorithms, simulated annealing and tabu search) is reported in [11]. Unfortunately, busses are implicit in the architecture, unlike in [12]. A scalability analysis of these algorithms for large real-time systems is introduced in [13]. Many heuristic scheduling algorithms are variants and extensions of list scheduling [14]. In general, scheduling tables list all schedules for different condition combinations in the task graph, and are therefore not suitable for control-intensive applications.

Constraint Logic Programming is an alternative approach to Integer Programming for solving combinatorial optimization problems [15]. Both techniques can claim individual successes but practical experience indicates that neither approach dominates the other in terms of computational performance. The development of a hybrid CP-IP solver that captures the best features of both would appear to offer scope for improved overall performance. However, the issue of communication between different modelling paradigms arises. One method is inherited from the Operations Research and is known as Benders Decomposition [22]: it has been proven to converge producing the optimal solution. There are a number of papers using Benders Decomposition in a CP setting[16][17][18][19].

In the context of MPSoCs, our approach leverages a decomposition of the mapping problem into two related sub-problems: (i) mapping of tasks to processors and of memory slots to storage devices and (ii) scheduling of tasks in time on their execution units. We tackle the mapping sub-problem with IP and the scheduling one with CP. The interaction is regulated by no-good generation and the process has been proven to converge to the optimal solution[24]. Our problem formulation will be compared with the most widely used traditional approaches, namely CP and IP modelling the entire mapping and scheduling problem as a whole, and the significant cut down on search time is showed. Moreover, in contrast to most previous work, the results of the optimization framework and its modelling assumptions are validated by means of functional simulation.

3. Target Architecture

The target architecture for our mapping strategy is a general template for a message-oriented distributed memory architecture. The specific implementation of this paradigm only changes the annotated values in the application task graph (cost for communication and execution times), which is an input to our framework. However, the allocation and scheduling methodology is not affected by specific design choices. The only requirements we pose on the architecture are the support for message exchange between the computation tiles, the availability of local memory devices and of remote (i.e., non-local to the tiles) storage devices for those program data that cannot be stored in local memories. This latter device can be a unified memory with partitions associated with each processor or a separate private memory for each processor core. We deployed the simulation model of an instance of this architectural template in order to prove the viability of our approach (see Fig. 1). The computation tiles are supposed to be homogeneous and consist of ARM7 cores (including instruction and data caches) and of tightly coupled software-controlled scratch-pad memories for fast access to program operands and for storing input data. We used an AMBA AHB bus as system interconnect. In our implementation, hardware and software support for efficient messaging are provided.

Messages can be directly moved between scratch-pad memories.

Finally, each processor core has a private on-chip memory, which can be accessed only by gaining bus ownership.

The software support is provided by a real-time operating system called RTEMS [25] and by a set of high-level APIs to support message passing on the scracth-pad based distributed memory architecture[20].

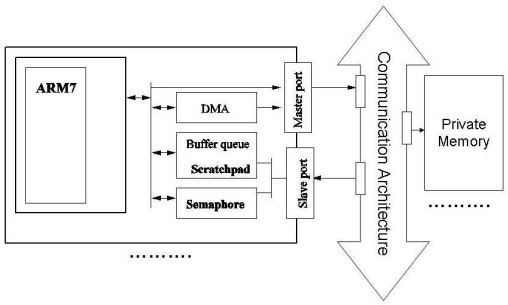


Figure 1. Message-oriented distributed memory architecture.

4. Problem Model

4.1. Task modelling

Our methodology requires the target multi-task application to be mapped and executed on top of the hardware platform as a task graph with precedence constraints. In particular, we target pipelined task graphs, representative of stream-oriented processing (video graphics pipelines, audio and video streaming, signal processing pipelines). A real-time requirement should also be specified, consisting for instance of a minimum required throughput for the pipeline of tasks. Tasks are the nodes of the graph and edges connecting any two node indicate task dependencies. Computation, storage and communication requirements should be annotated onto the graph as follows.

The execution time for the tasks may be expressed either in terms of worst-case or average-case execution time. In this paper, we opt for average-case analysis, which implies the objective to meet application soft-real time requirements. In the experimental results, we will also investigate how trustworthy our optimizer is in predicting deadline hits or misses with task allocation and schedules generating only very tight time slacks.

Since we are considering a pipeline of tasks, we will analyze the system behavior at working rate, that is when all processes are running or ready to run. To do that, we will schedule 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.

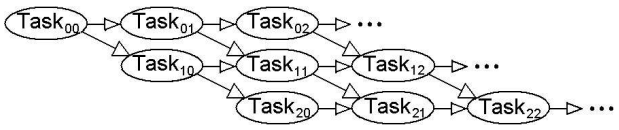


Figure 2. Precedence constraints among the activities

Figure 2 depicts the precedence constraints among the tasks. $Task_{ij}$ represents the j-th repetition of the i-th task of the pipeline. The horizontal arrows are precedence constraints, while the diagonal arrows are communication constraints.

Each task also has 3 kinds of associated memory requirements:
- **Program Data**: storage locations are required for computation data and for processor instructions. They can be allocated by the optimizer either on the local scratchpad memory or on the remote private memory.
- **Internal State**: when needed, an internal state of the task can be stored either locally or remotely.
- **Communication queues**: the task needs communication queues to store outgoing as well as incoming messages to/from other tasks. The communicating task might run on the same processor (thus incurring a negligible communication cost) or on a remote processor, which would imply a costly message exchange procedure. In order to have efficient messaging, we pose the constraint that such com-

munication queues should be stored in local scratch-pad memory only.

We assume that application tasks start with checking availability of input data and of room for writing computation results (i.e., if the output queue has been freed by the downstream task), in an SDF-like (synchronous dataflow) semantics. Actual input data transfer, task execution and generation of output data occur only when these conditions are met. These assumptions simply make the communication and computation phase of each task atomic, thus avoiding to schedule communication as a separate task.

4.2. Bus modelling

Whenever predictable performance is needed for time-critical 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[21]. Finally, when the bus is required a bandwidth from concurrent tasks that does not exceed a certain threshold, its behaviour can be abstracted by means of a very simple additive model. In other words, the bus delivers an overall bandwidth that is approximatively equal to the sum of the bandwidth requirements of the concurrent tasks that are competing for its use.

This model, provided the working conditions under which it holds are carefully determined, has some relevant advantages with respect to the scheduling problem model. First, it allows to deploy a larger granularity time unit to solve the problem. In fact, real busses rely on the serialization of bus access requests by re-arbitrating on a transaction basis. Modelling bus allocation at such a fine granularity would make the scheduling problem overly complex since it should be modelled as unary resource (i.e., a resource with capacity one). In this case, task execution should be modelled using the clock cycle as the unit of time and the resulting scheduling model would contain a huge number of variables. The additive model instead considers the bus as an additive resource, in the sense that more activities can share bus utilization using a different fraction of the total bus bandwidth. Figure 3 illustrates this assumption. Note that the maximum bandwidth is not the real maximum bandwidth that the bus is physically able to deliver, but the upper threshold beyond which the additive model fails to predict the interconnect behaviour because of the impact of contention. We will derive it in the experimental section by means of extensive simulation runs.

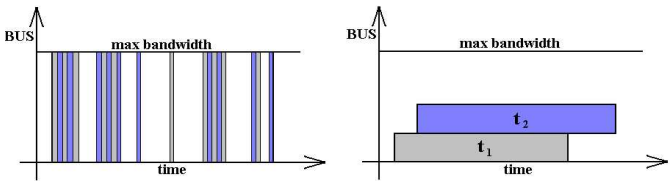


Figure 3. Bus allocation in a real system (left) and in our model (right)

In order to define the fraction of the bus bandwidth absorbed by each task, we consider the amount of data they have to access from their private memories and we spread it over its execution time. In this way we assume that the task is uniformly consuming a fraction of the bus bandwidth throughout its execution time. This assumption will be validated in presence of different traffic patterns in the experimental section.

Another important effect of the bus additive model is that task execution times will not be stretched as an effect of busy waiting on bus transaction completion. Once the execution time of a task is characterized in a congestion free regime, it will be only marginally affected by the presence of competing bus access patterns, in the domain where the additive model holds.

4.3. Problem Decomposition and No-Good Generation

The problem of allocating and scheduling task graphs is a scheduling problem with alternative resource (i.e., processors and storage devices). Tasks should be scheduled in time subject to real time constraints, precedence constraints, and capacity constraints on all unary and cumulative resources. As a whole, the problem is extremely difficult to solve. For this reason, the leading edge techniques for solving combinatorial optimization problems, namely Integer Programming (IP) and Constraint Programming (CP), fail to solve even small problem instances.

From a different perspective, the problem decomposes into two (interdependent) problems: the allocation of tasks to processors and of the memory slots required by each task to the proper memory device, and the scheduling problem with static resource allocation. The first problem is better solved with IP, while for the second CP is the technique of choice.

The critical issue is now how to make the two problems interact. We solve them separately, the allocation problem first (called master problem), and the scheduling problem (called subproblem) later. The master is solved to optimality and its solution passed to the subproblem solver. If the solution is feasible, then the overall problem is solved to optimality. If, instead, the master solution cannot be completed by the subproblem solver, a no-good is generated and added to the model of the master problem, roughly stating that the solution passed should not be recomputed again (it becomes infeasible), and a new optimal solution is found for the master problem respecting the (set of) no-good(s) generated so far.

4.4. Allocation Problem Model

Given an input pipelined task graph, the allocation problem consists of allocating n tasks to m processors, such that the total amount of memory allocated to the tasks, for each processor, does not exceed the size of the local scratch-pad.

The only simplifying assumption that we make is that the remote private memories are of unlimited size. In the worst case assumption of extremely large program data, this might require to consider the increased access cost to off-chip memories and the efficiency of a memory controller. The problem objective function is the minimization of the amount of data transferred on the bus. We model the problem as an integer program and we consider four decision variables in the model: T_{ij}, taking value 1 if task i executes on processor j, 0 otherwise; Y_{ij}, taking value 1 if task i allocates the program data on the scratchpad memory of processor j, 0 otherwise; Z_{ij}, taking value 1 if task i allocates the internal state on the scratchpad memory of processor j, 0 otherwise; X_{ij}, taking value 1 if task i executes on processor j and task $i+1$ does not, 0 otherwise.

The constraints we introduce in the model are:

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

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

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

Constraints (1) state that each process can execute only on one 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 that enforce the sum $T_{ij} + T_{i+1j} + X_{ij}$ to be equal either to 0 or 2. Constraints (3) prevent the solver to allocate a set of consecutive tasks whose execution times sum exceeds the real time requirement (RT) to the same processor. (3) are relaxations of the sub-problem added to the master problem to avoid the generation of trivially infeasible assignments where all tasks are packed into the minimal number of processors in so far as memory constraints allow. The use of a relaxation in the master problem is well known and widely used in practice and helps in producing better solutions.

We add to the problem the constraints stating that $T_{ij} = 0 \Rightarrow Y_{ij} = 0, Z_{ij} = 0$ meaning that if a processor j is not assigned to a

task i neither its program data nor the internal state can be stored in the local memory of processor j.

The objective function is the minimization of the total amount of data transferred on the bus. 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 is to minimize:

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

where mem_i, $state_i$ and $data_i$ are coefficients representing the amount of data used by task i to store respectively the program data, the internal state and the communication queue.

4.5. Scheduling problem model

Once tasks have been allocated to the processors, we need to schedule process execution. As introduced in section 4.1, we schedule at least n^2 tasks. In the CP model, we split each task $Task_{ij}$ in Figure 2 in different activities: the computation activity A_{ij}, preceded by the input data reading activity In_{ij}, and possibly preceded by the internal state reading activity RS_{ij} and followed by the internal state writing activity WS_{ij}.

The precedence constraints among the activities introduced in the model are:

$$A_{i,j-1} \prec In_{ij} \ , \ \forall\, i,j \tag{5}$$
$$In_{ij} \preceq A_{ij} \ , \ \forall\, i,j \tag{6}$$
$$In_{ij} \preceq RS_{ij} \ , \ \forall\, i,j \tag{7}$$
$$A_{i-1,j} \prec In_{ij} \ , \ \forall\, i,j \tag{8}$$
$$RS_{ij} \preceq A_{ij} \ , \ \forall\, i,j \tag{9}$$
$$A_{ij} \preceq WS_{ij} \ , \ \forall\, i,j \tag{10}$$
$$A_{i+1,j-1} \prec In_{ij} \ , \ \forall\, i,j \tag{11}$$
$$A_{i,j-1} \prec A_{ij} \ , \ \forall\, i,j \tag{12}$$

where the symbol $\prec$ means that the activity on the left should precede the activity on the right, 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., $In_{ij} \prec A_{ij}$ means $Start\_In_{ij} + Dur\_In_{ij} \leq Start\_A_{ij}$, and $RS_{ij} \preceq A_{ij}$ means $Start\_RS_{ij} + Dur\_RS_{ij} = Start\_A_{ij}$.

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 must read the communication queue just before the execution, or, if exists, just before the internal state reading (7); for each task, only the appropriate one among constraints (6) and (7) will be introduced in the model. Constraints (8) state that each task can read the data in the communication queue only when the previous task has generated them. Constraints (9) and (10) state that each task must read the internal state just before the execution and write it just after. Constraints (11) state that each task can read the communication queue only if the previous iteration of the following task has ended; in other words, a task can read the communication queue only if the following task is ready to read, in its turn, its communication queue. Constraints (12) state that the iterations of each task must execute in order.

Furthermore, we introduced the real time requirement constraints (13), whose relaxation is used in the allocation problem model. Each task must execute at least once for each time period RT.

$$Start(A_{ij}) - Start(A_{i,j-1}) \leq RT \ , \ \forall\, i,j \tag{13}$$

As explained above, the bus is modelled as an additive resource, therefore the constraint is posed that the sum of the bus bandwidth requirements of concurrently executing tasks on different processors does not exceed the upper threshold. This upper bound was experimentally characterized and set to 50% of the maximum bandwidth the AMBA bus can physically deliver.

Finally, when finding a schedule for a given allocation turns out to be infeasible, a no-good is generated in the form of a linear constraint and passed to the IP solver. The constraint prevents the current solution to be re-computed again.

4.6. Computational efficiency

To validate the strength of our approach, we now compare the results obtained using this model (**Hybrid** in the following) with results obtained using only a CP or IP model to solve the overall problem. Actually, since the first experiments showed that both CP and IP approaches are not able to find a solution, except for the easiest instances, within 15 minutes, we simplified these models removing some variables and constraints. In CP, we fixed the activities execution time not considering the execution time variability due to remote memory accesses, therefore we do not consider the In_{ij}, RS_{ij} and WS_{ij} activities, including them statically in the activities A_{ij}. In IP, we do not consider all the variables and constraints involving the bus: we do not model the bus resource and we therefore suppose that each activity can access data whenever it is necessary.

We generated 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 for each task or processor number. All problems considered have a solution. Experiments were performed on a 2GHz Pentium 4 with 512 Mb RAM. We used ILOG CPLEX 8.1 and ILOG Solver 5.3 as solving tools.

In figures 4 we compare the algorithms search time for problems with a different number of tasks and processors respectively. Times are expressed in seconds and the y-axis has a logarithmic scale.

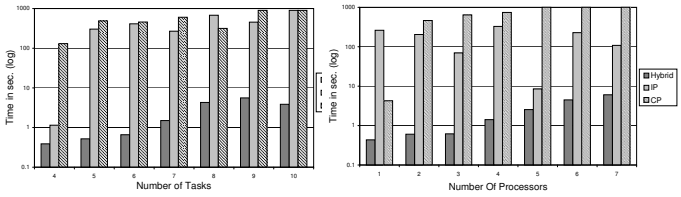


Figure 4. Comparison between algorithms search times for different task number (left) and for different processor number (right)

Although CP and IP deal with a simpler problem model, we can see that these algorithms are not comparable with Hybrid, except when the number of tasks and processors is low; this is due to the fact that the problem instance is very easy to be solved, and Hybrid looses time creating and solving two models. As soon as the number of tasks and/or processors grows, IP and CP performances worsen and their search times become orders of magnitude higher w.r.t. Hybrid. Furthermore, we considered in the figures only instances where the algorithms are able to find the optimal solution within 15 minutes, and, for problems with 6 tasks or 3 processors and more, IP and CP can find the solution only in the 50% or less of the cases, while Hybrid can solve 100% of the instances. In addition, We can see that Hybrid search time scales up linearly in the logarithmic scale.

We also measured the number of times the solver iterates between the master and the sub-problem. We found that, due to the limited size of the local memories and to the relaxation of the sub-problem added to the master, the solver iterates 1 or 2 times. Removing the relaxation, it iterates up to 15 times. This result gives evidence that, in a Benders decomposition based approach, it is very important to introduce a relaxation of the sub-problem in the master, and that the relaxation we use is very effective even if very simple.

5. Methodology

Our approach consists of using a virtual platform to pre-characterize the input task set, to simulate the allocation and scheduling solutions provided by the optimizer and to detect deviations of measured performance metrics with respect to predicted ones.

For each task in the input graph we need to extract the following information: bus bandwidth requirement for reading input data in case the producer runs on a different processor, time for reading input data if the producer runs on the same processor, task execution

time with program data in scratch-pad memory, task execution overhead due to cache misses when program data resides in remote private memory. For each pipelined task graph, this information can be collected with $2+N$ simulation runs on the MPARM simulator[23], where N is the number of tasks. Recall that this is done once for all. We model task communication and computation separately to better account for their requirement on bus utilization, although from a practical viewpoint they are part of the same atomic task. The initial communication phase consumes a bus bandwidth which is determined by the hardware support for data transfer (DMA engines or not) and by the bus protocol efficiency (latency for a single read transaction). The computation part of the task instead consumes an average bandwidth defined by the ratio of program data size (in case of remote mapping) and execution time. We use the virtual platform also to calibrate the bus additive model, specifying the range where such model holds. For an AMBA bus, we found that tasks should not concurrently ask for more than 50% of the theoretical bandwidth the bus can provide (400 MByte/sec with 1 wait state memories), otherwise congestion causes a bandwidth delivery which does not keep up with the requirements.

The input task parameters are then fed to the optimization framework, which provides optimal allocation of tasks and memory locations to processor and storage devices respectively, and a feasible schedule for the tasks meeting the real-time requirements of the application. Two options are feasible at this point. First, the optimizer uses the conservative maximum bus bandwidth indicated by the virtual platform, and derives solutions that will be accurately reproduced on the virtual platform, if modelling assumptions are correct. Second, the optimizer uses a higher bandwidth than specified, in order to improve bus utilization, and the virtual platform is used to assess the accuracy of the optimization step (e.g., constraint satisfaction, validation of execution and data transfer times). If the accuracy is not satisfactory, a new iteration of the procedure will allow to progressively decrease the maximum bandwidth until the desired level of accuracy is reached with the simulator.

6. Experimental Results

We have performed three kinds of experiments, namely (i) validation and calibration of the bus additive model, (ii) measurement of deviations of simulated throughput from theoretically derived one for a large number of problem instances, (iii) showing the viability of the proposed approach by means of a GSM-based demonstrator.

6.1. Validation of the bus additive model

The intuitive meaning of the bus additive model is illustrated by the experiment of Fig.5(A). An increasing number of AMBA-compliant uniform traffic generators, consuming each 10% of the maximum theoretical bandwidth (400 MByte/sec), have been connected to the bus, and the resulting real bandwidth provided by the bus measured in the virtual platform. It can be clearly observed that the delivered bandwidth keeps up with the requested one until the sum of the requirements amounts to 60% of the maximum theoretical bandwidth. If the communication requirements exceed the threshold, as a side effect we observe an increase of the execution times of running tasks with respect to those measured without bus contention, as depicted in Fig.5(B). For this experiment, synthetic tasks running on each processor have been employed. The 60% bandwidth threshold value corresponds to an execution time variation of about 2% due to longer bus transactions.

However, the threshold value also depends on the ratio of bandwidth requirements of the tasks concurrently trying to access the bus. Contrarily to Fig.5(A), where each processor consumes the same fraction of bus bandwidth, Fig.6(A) shows the deviations of offered versus required bandwidth for competing tasks with different bus bandwidth requirements. Configurations with different number of processors are explored, and numbers on the x-axys show the percentage of maximum theoretical bandwidth required by each task. It can be observed that the most significant deviations arise when one task starts draining most of the bandwidth, thus creating a strong interference with all other access patterns. The presence of such communication hotspots suggests that the maximum cumulative bandwidth requirement which still stimulates an additive behaviour of

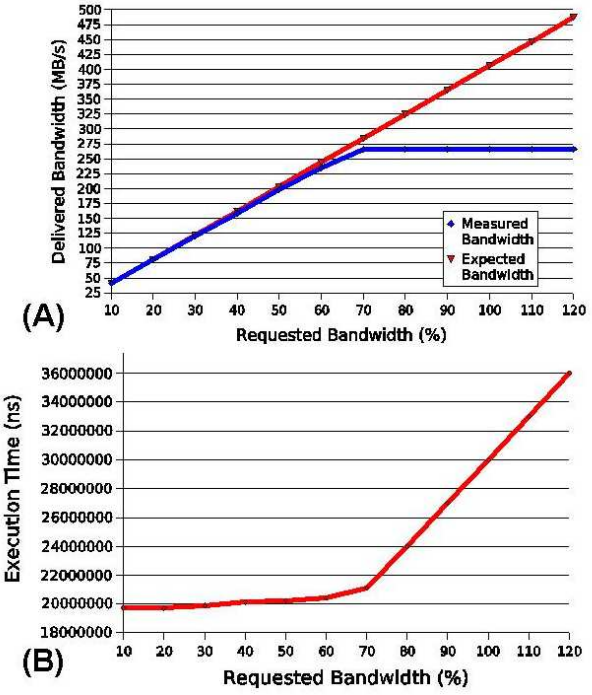


Figure 5. (A) Implications of the bus additive model; (B) Execution time variation.

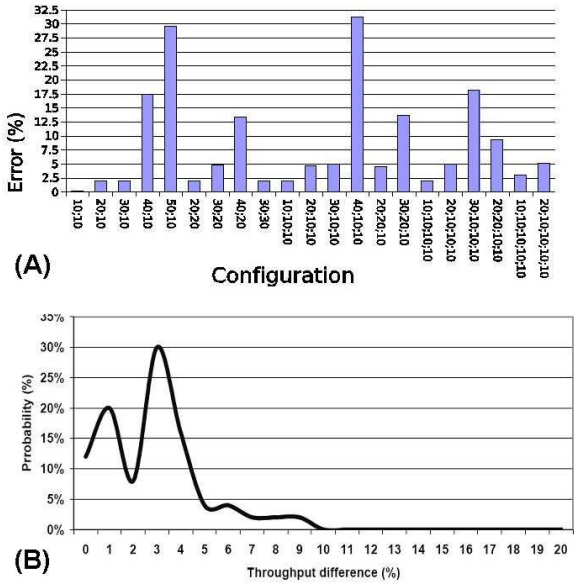
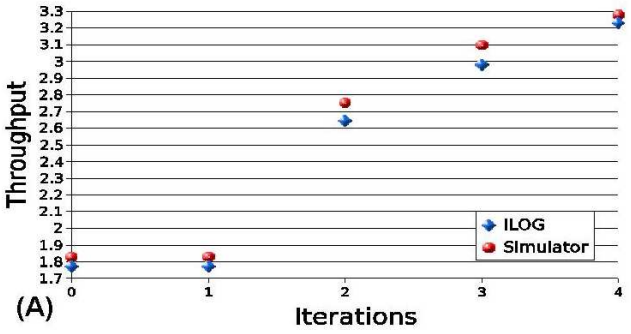


Figure 6. (A) Bus additive model for different ratios of bandwidth requirements among competing tasks for bus access; (B) Probability of throughput differences.

the bus is lower than the one computed before, and amounts to about 50% of the theoretical maximum bandwidth. We have also tried to reproduce Fig.6(A) with different burstiness of the generated traffic. Results are not reported here since the measured upper thresholds for the additive models are more conservative than those obtained with single transfers.

6.2. Validation of allocation and scheduling solutions

We have deployed the virtual platform to implement the allocations and schedules generated by the optimizer, and we have measured deviations of the simulated throughput from the predicted one for 50 problem instances.A synthetic benchmark has been used for this experiment, allowing to change system and application param-



(A)

	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6
Computation Time (ns)	281639	437038	317032	308899	306213	306470
Remote Data Overhead (ns)	3978	1620	1099	2243	1916	1707
Local Communication (ns)		4754	6675	5810	6020	5810
Remote Communication (ns)		8621	12266	10773	10609	10576
Program Data (Byte)	420	420	560	560	560	560
Communication Data In – Out (Byte)	0 - 340	340 – 444	444 - 444	444 - 444	444 - 444	444 – 0
Processor	1	1	2	2	3	3
Data Location	Local	Local	Remote	Remote	Remote	Local
4 Processors with 2048 Byte ScratchPad Memory						

(B)

Figure 7. (A) Conservative performance predictions of the optimizer; (B) GSM case study.

eters (local memory size, execution times, data size, etc.). We want to make sure that modelling approximations are not such to significantly impact the accuracy of optimizer results with respect to real-life systems. The results of the validation phase are reported in Fig.6(B), which shows the probability for throughput differences. The average difference between measured and predicted values is 4.7%, with 0.08 standard deviation. This confirms the high level of accuracy achieved by the developed optimization framework.

Fig.7(A) shows that our optimizer is not only accurate within acceptable limits, but also conservative in predicting system performance, and this is very important for meeting real-time requirements. For a given problem instance, the plot compares the throughput provided by the optimizer with the simulated one for the same allocations and schedules. The range of throughputs has been spanned by progressively making the real-time constraint of the solver tighter. This latter provides an allocation and a schedule that are able to guarantee an entire range of throughput constraints. If a lower throughput is required, than the configuration found by the solver changes. Moving from one configuration to another corresponds to increasing steps on the x-axys. At each new point, the simulated throughput is reported as well, and it is showed to provide a conservative throughput with respect to the predicted one, within the accuracy limits found above.

6.3. Application to GSM

The GSM application has been used to prove the viability of our approach. The source code has been parallelized into 6 pipeline stages, and each task has been pre-characterized by the virtual platform to provide parameters of task models to the optimizer. Such information, together with the results of the optimization run, are reported in Fig.7(B). Note that the optimizer makes use of 3 out of the 4 available processors, since it tries to minimize the cost of communication while meeting hardware and software constraints. The required throughput in this case was 1 frame/10ms, largely within the minimun GSM requirements. The obtained throughput was 1.35 frames/ms, even more conservative. As already seen, the simulation gave a better throughput than the predicted one, with a difference of 4.1%. The table also shows that program data has been allocated in scratch-pad memory for Tasks 1,2 and 6 since they have smaller communication queues. Schedules for this problem instance are trivial. The time taken by the optimizer to come to a solution was 0.1 seconds.

7. Conclusions

We target allocation and scheduling of pipelined stream-oriented applications on top of distributed memory architectures with support for messaging. We tackle the complexity of the problem by means of decomposition and no-good generation, and prove the increased computational efficiency of this approach with respect to traditional ones. Moreover, we deploy a virtual platform for validating the optimization process and to check modelling assumptions, showing a very high level of accuracy. Finally, we show the viability of our approach by means of a GSM-based demonstrator.

References

[1] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini, and R. Zafalon, "Analyzing on-chip communication in a MPSoC environment", Proc. DATE 2004.

[2] S. Prakash and A. Parker, "SOS: Synthesis of Application-Specific Heterogeneous Multiprocessor Systems", Journal of Parallel and Distributed Computing, pp. 338-351, 1992.

[3] C. Lee, M. Potkonjak and W. Wolf, "System-Level Synthesis of Application-Specific Systems Using A* Search and Generalized Force-Directed Heuristics", Procs. of the 9th Intern. Symposium on System Synthesis - ISSS '96, pp. 2-7, Nov. 1996.

[4] A. Bender, "MILP based Task Mapping for Heterogeneous Multiprocessor Systems", EURO-DAC '96/EURO-VHDL '96: Procs. of the conference on European design automation, pp. 190-197, Sept. 1996.

[5] Y. Li and W. H. Wolf, "Hardware/Software Co-Synthesis with Memory Hierarchies", 1999 IEEE Trans.on Computer-Aided Design of Integrated Circuits and Systems, pp. 1405-1417, 1999.

[6] G.De.Micheli, "Synthesis and optimization of digital circuits", McGraw Hill, 1994.

[7] K.S.Chatha and R.Vemuri, "Hardware-Software Partitioning and Pipelined Scheduling of Transformative Applications", 2002 IEEE Trans. on Very Large Scale Integration Systems, pp. 193-208, 2002.

[8] P.Palazzari and L.Baldini and M.Coli, "Synthesis of Pipelined Systems for the Contemporaneous Execution of Periodic and Aperiodic Tasks with Hard Real-Time Constraints", 18th International Parallel and Distributed Processing Symposium - IPDPS'04, pp. 121-128, Apr. 2004.

[9] K.Kuchcinski and R.Szymanek, "A constructive algorithm for memory-aware task assignment and scheduling", Procs of the Ninth International Symposium on Hardware/Software Codesign - CODES 2001, pp. 147-152, Apr. 2001.

[10] G.Fohler and K.Ramamritham, "Static Scheduling of Pipelined Periodic Tasks in Distributed Real-Time Systems", Procs. of the 9th EUROMICRO Workshop on Real-Time Systems - EUROMICRO-RTS '97, pp. 128-135, June 1997.

[11] J.Axelsson, "Architecture Synthesis and Partitioning of Real-Time Synthesis: a Comparison of 3 Heuristic Search Strategies", Procs. of the 5th Intern. Workshop on Hardware/Software Codesign (CODES/CASHE97), pp. 161-166, Mar. 1997.

[12] P.Eles, Z.Peng, K.Kuchcinski and A.Doboli, "System Level Hardware/Software Partitioning Based on Simulated Annealing and Tabu search", Design Automation for Embedded Systems, pp. 5-32, 1997.

[13] S.Kodase, S.Wang, Z.Gu and K.Shin, "Improving Scalability of Task Allocation and Scheduling in Large Distributed Real-Time Systems Using Shared Buffers", Procs. of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2003), pp. 181-188, May 2003.

[14] P.Eles, Z.Peng, K.Kuchcinski, A.Doboli and P.Pop, "Scheduling of Conditional Process Graphs for the Synthesis of Embedded Systems", Procs. of the conference on Design, automation and test in Europe, pp. 132-139, 1998.

[15] K.Kuchcinski, "Embedded System Synthesis by Timing Constraint Solving", IEEE Transactions on CAD, pp. 537-551, 1994.

[16] E.S.Thorsteinsson, "A hybrid framework integrating mixed integer programming and constraint programming", Procs. of the 7th International Conference on Principles and Practice of Constraint Programming - CP 2001, pp. 16-30, 2001.

[17] A.Eremin and M.Wallace, "Hybrid Benders Decomposition Algorithms in Constraint Logic Programming", Procs. of the 7th Intern. Conference on Principles and Practice of Constraint Programming - CP 2001, pp. 1-15, Nov. 2001.

[18] I.E.Grossmann and V.Jain, "Algorithms for Hybrid MILP/CP Models for a Class of Optimization Problems", INFORMS Journal on Computing, pp. 258-276, 2001.

[19] J.N.Hooker, "A Hybrid Method for Planning and Scheduling", Procs. of the 10th Intern. Conference on Principles and Practice of Constraint Programming - CP 2004, pp. 305-316, Sept. 2004.

[20] F.Poletti, A.Poggiali and P.Marchal, "Flexible Hardware/Software Support for Message Passing on a Distributed Shared Memory Architecture", Design And Test Europe Conference 2005 Proceedings, pp. 736-741, 2005.

[21] M.Ruggiero, F.Angiolini, F.Poletti, D.Bertozzi, L.Benini, R.Zafalon, "Scalability Analysis of Evolving SoC Interconnect Protocols", Int. Symposium on System-on-Chip, 2004.

[22] J.F.Benders, "Partitioning procedures for solving mixed-variables programming problems", Numerische Mathematik, pp. 238-252, 1962.

[23] 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), pp. 752-757, Febr. 2004.

[24] J.N.Hooker and G.Ottosson, "Logic-based Benders decomposition", Mathematical Programming, pp. 33-60, 2003.

[25] RTEMS Home Page, http://www.rtems.com