Allocation and Scheduling for MPSoCs via decomposition and no-good generation

Luca Benini, Davide Bertozzi, Alessio Guerri, Michela Milano
DEIS, University of Bologna
V.le Risorgimento 2, 40136, Bologna, Italy
{lbenini, dbertozzi, aguerri, mmilano}@deis.unibo.it

This paper proposes a decomposition approach to the allocation and scheduling of a multi-task application on a multiprocessor system-on-chip (MPSoCs) [Wolf, 2004]. This is currently one of the most critical problems in electronic design automation for Very-Large Scale Integrated (VLSI) circuits. With the limits of chip integration reaching beyond one billion of elementary devices, current advanced integrated hardware platforms for high-end consumer application (e.g. multimedia-enabled phones) contain multiple processors and memories, as well as complex on-chip interconnects. The hardware resources in these MPSoCs need to be optimally allocated and scheduled under tight throughput constraints when executing a target software workload (e.g. a video decoder).

The multiprocessor system consists of a predefined number of distributed computation nodes, as depicted in Figure 1. Each node is made by a processing core and by a tightly coupled local memory. 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 resource, and serialization of bus access requests of the processors (the bus masters) is carried out by a centralized arbitration mechanism.

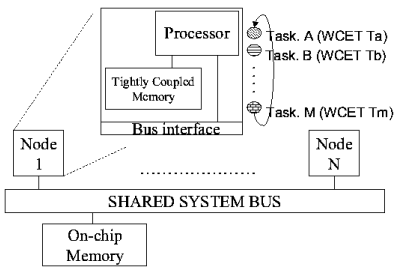

Figure 1: Single chip multi-processor architecture.

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 is simpler under these working conditions (e.g., additive models). Communication cost is therefore critical for determining overall system performance, and will be minimized in our task allocation framework.

Based on our methodology, the target application running on top of the hardware platform is pre-characterized and abstracted as a task graph, with specification of computation, storage and communication requirements. More in detail, the worst case execution time (WCET) is specified for each task and plays a critical role whenever application real time 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 WCETs of the tasks for one iteration of the time wheel must not exceed time period RT (i.e., the minimum task scheduling period ensuring that throughput constraints are met), which is the same for each processor since the minimum throughput is an application (not single processor) requirement.

The problem we are facing is a scheduling problem with alternative resources. Each task should be allocated to one of the processors (Node i in Figure 1). Each task also needs 3 memory areas for executing, that should be allocated to a memory device: **internal task state** and **program data** can be allocated either on the local scratchpad memory or on the remote on-chip memory, while **communication queue** (the memory area used by the tasks to communicate one other) must be allocated on the local scratchpad. Tasks duration depends on where memory slots are allocated; tasks need time to access the bus and use remote memory. Clearly, tasks should be scheduled in time subject to real time constraints, precedence constraints, and capacity constraints on all unary and cumulative resources. However, on a different perspective, the problem decomposes into two problems: the allocation of tasks to processors and the memory slots required by each task to the proper memory device; a scheduling problem with static resource allocation.

The objective function of the overall problem is the minimization of communication cost. This function involves only variables of the first problem. In particular, we have a communication cost each time two communicating tasks are allocated on different processors, and each time a memory slot is allocated on a remote memory device. Once the communication cost has been minimized, among feasible schedules we prefer those having minimum makespan.

The allocation problem is difficult to solve with Constraint Programming (CP). CP has a naive method for solving optimization problems: each time a solution is found, an additional constraint is added stating that each successive solution

should be better than the best one found so far. If the objective function is strongly linked to decision variables, CP can be effective, otherwise it is hopeless to use CP to find the optimal solution. In case the objective function is related to a single variable, like for makespan in scheduling problems, CP works quite well. However, if the objective function is a sum of cost variables, CP is able to prune only few values, deep in the search tree since the connection between the objective function and the problem decision variables is weak. If the objective function relates to pairs of assignments the situation is even worse. This is the case of our application where the objective function depends on pairs of assignments. In fact, a contribution to the objective function is added when two communication tasks are allocated to different processors.

On the contrary, Integer Programming (IP) is extremely good to cope with these problems, while is weak in coping with time. Scheduling problems require to assign tasks to time slots, and each slot should be represented by an integer variable, and the number of variables increases enormously. CP, instead, is very effective to cope with time constraints.

Therefore, the first problem could be solved with IP effectively, while for the second CP is the technique of choice. The question is now: how do these 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. Being the allocation problem solver an IP solver, the no-good has the form of a linear constraint.

A similar method is known in Operations Research as Benders Decomposition [Benders, 1962], where the overall problem can be decomposed in two parts connected by some variables. Some related approaches are [Hooker, 2004] and [Grossmann and Jain, 2001].

We show that this method is extremely effective if compared to the approaches considering the problem as a whole.

The methodology proposed in this paper has been applied to a video signal processing pipeline, wherein each task processes output data of the preceding task in the pipeline. Functional pipelining is widely used in the domain of multimedia applications. Task parameters have been derived from a real video graphics pipeline processing pixels of a digital image. The proposed allocation and scheduling techniques can be easily extended to all applications using pipelining as workload allocation policy, and aim at providing system designers with an automated methodology to come up with effective solutions and cut down on design time. To do that, we schedule several repetitions of the pipeline processes in order to achieve a working rate configuration.

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 were 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; 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 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 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 modelling and solving tools.

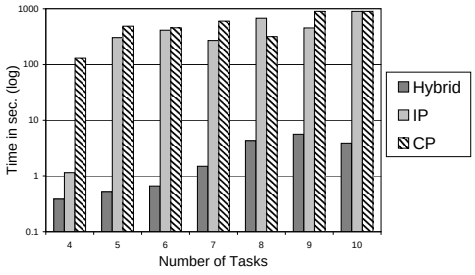

Figure 2: Comparison between algorithms search times for different process number.

In figure 2 we compare the algorithms search time for problems with a different number of tasks. Times are expressed in seconds and the y-axis has a logarithmic scale. For space reasons we omit the search time figure varying the number of processors, but it has very similar behaviours to figure 2.

Although CP and IP deal with a simpler problem model, we can see that these algorithms are in general not comparable with Hybrid and, as the number of tasks 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 more, IP and CP can find the solution only in the 50% or less of the cases.

References

[Benders, 1962] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[Grossmann and Jain, 2001] I. E. Grossmann and V. Jain. Algorithms for hybrid milp/cp models for a class of optimization problems. *INFORMS Journal on Computing*, 13:258–276, 2001.

[Hooker, 2004] J. N. Hooker. A hybrid method for planning and scheduling. In *Procs. of the 10th Intern. Conference on Principles and Practice of Constraint Programming - CP 2004*, pages 305–316, Toronto, Canada, Sept. 2004. Springer.

[Wolf, 2004] 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, June 2004. ACM.