Scheduling Driven Partitioning of Heterogeneous Embedded Systems

Paul Pop, Petru Eles, Zebo Peng
Dept. of Computer and Information Science, Linköping University, Sweden
{paupo, petel, zebpe}@ida.liu.se

Extended Abstract

In this paper we present an algorithm for system level hardware/software partitioning of heterogeneous embedded systems. The system is represented as an abstract graph which captures both data-flow and the flow of control. Given an architecture consisting of several processors, ASICs and shared busses, our partitioning algorithm finds the partitioning with the smallest hardware cost and is able to predict and guarantee the performance of the system in terms of worst case delay.

1 Introduction

A great deal of research has been done on hardware/software partitioning [1]. Several research groups consider hardware/software architectures consisting of a single programmable processor and an ASIC. In this case, the behaviour is partitioned into one software partition and one hardware partition. However, for complex systems, such a restricted architecture doesn't allow an efficient design space exploration, and therefore we will concentrate on more general architectures. As the predictability of the systems in terms of performance is becoming increasingly important, timing constraints have also to be considered.[2]

2 Problem Formulation and the Process Graph

We consider embedded systems specified as a set of interacting processes. Our goal is to develop a partitioning algorithm that, given an implementation architecture consisting of several programmable processors, ASICs and interconnection busses, together with an end-to-end deadline[1] on the execution time of the system, will generate a partitioning of processes and communication tasks that implies the smallest possible system cost. The resulted partitioning will specify for each process if it will be implemented in software and on which programmable processor or if it has to be synthesized to hardware on an ASIC. Communication between processes mapped to different processors or ASICs is modelled by so called *communication*

1. multiple deadlines and release times can also be handled

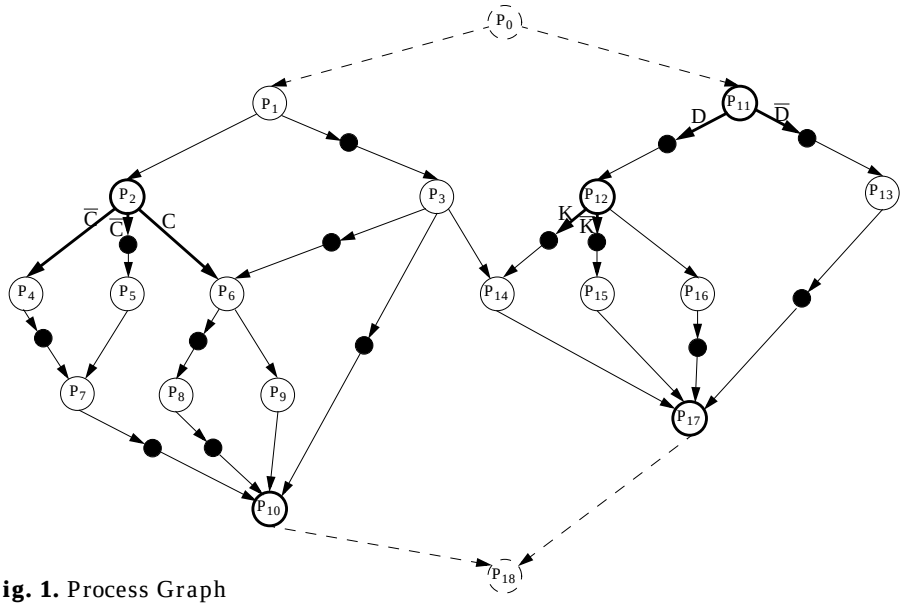

Fig. 1. Process Graph

tasks. The final partitioning will indicate to which bus each communication task is mapped. The end-to-end deadline will be guaranteed by a worst case delay and the execution of the final system will proceed according to a statically generated schedule [4].

As an abstract model for system representation we use a directed, acyclic, polar graph with *conditional edges* (Fig.1). Each node in this graph represents one process. An edge from process P_i to P_j indicates that the output of P_i is the input of P_j. Unlike a simple edge, a conditional edge (depicted with thicker lines in Fig.1) has an associated condition. Transmission on a conditional edge will take place only if the associated condition is satisfied and not, like on simple edges, for each activation of the input process P_i.

The architecture is given as a set of programmable processors, ASICs and memories interconnected by busses. An end-to-end deadline on the execution time of the system is imposed and an upper bound on the total hardware cost of ASICs is given. For each process we have an estimation of the execution time if implemented on any of the programmable processors (programmable processors can be of different types) as well as the execution time of the process if implemented on an ASIC. For a given communication task (depicted with black dots in Fig.1) connecting processes P_i and P_j we have an estimation of the corresponding communication time, based on the amount of information exchanged by the two processes.

The estimation of the hardware cost if a process is implemented on an ASIC is also given.

The designer, based on his previous experience, is allowed to express several constraints on the placement of a certain process or group of processes in the final partitioning.

3 Branch and Bound (BB) Partitioning

Finding the optimal partitioning is a problem of exponential complexity. However, for most of the real life problems, the imposed design constraints and the restrictions imposed by the designer will reduce the number of feasible partitions, thus a search algorithm like branch and bound can be used.[3]

In order to apply a BB strategy, the state space corresponding to the problem is organized as a state tree (Fig.2). The main characteristics of a BB algorithm are the branching rule, the selection rule, and the bounding rule. Their definition has a decisive influence on the number of visited states and, thus, on the performance of the algorithm.

3.1 The Branching Rule

The branching rule defines the steps which are performed for generation of new states starting from a given parent state. The generation of new states is realized by moving one task from the list of yet unmapped tasks into the partitions corresponding to each of the programmable processors or the hardware. For example, new states to be generated from the node $<P_1,\phi,P_2>$ correspond respectively to nodes $<P_1P_3,\phi,P_2>$, $<P_1,P_3,P_2>$ and $<P_1,\phi,P_2P_3>$. This generation is restricted by design constraints and user imposed restrictions.

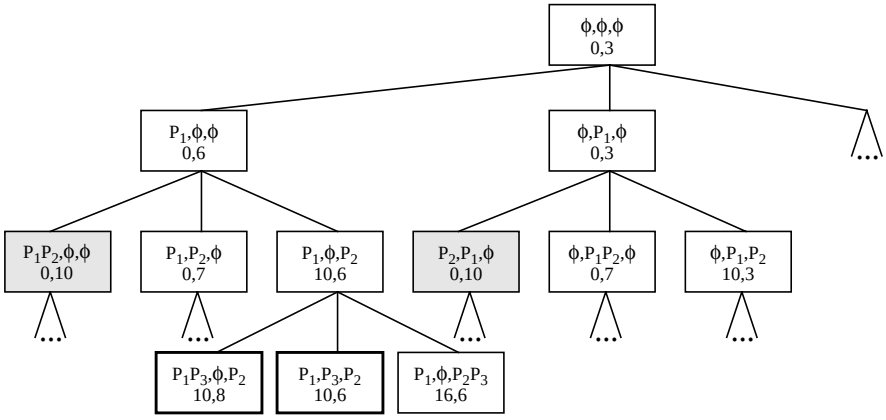

Fig. 2. Partial solution tree with three partitions: two programmable processors and one ASIC. A partitioning corresponds to a node in the state tree and it is depicted by a triplet. For example, the triplet $<P_1, P_3, P_2>$ means that P_1 is mapped on the first programmable processor, P_3 on the second, and P_2 will be an ASIC. For each node we also know its corresponding hardware cost and worst case execution time, depicted as a pair. Also, for this design the end-to-end deadline is 9 and the upper bound on total hardware cost is 11.

3.2 The Selection Rule

From the children generated by a certain node one should be selected in order to continue the branching operation. Our approach is to select the node which has the smallest hardware cost while meeting the deadline.

3.3 The Bounding Rule

The efficiency of a BB algorithm depends on how large part of the state tree is effectively generated. Before branching from a node a decision is taken if exploration has still to continue on the subtree or the subtree can be cut.

Our approach is to compare the performance estimation of a given design alternative with the deadline. If the deadline is not met, the subtree originating in the current node is cut (shaded nodes in Fig.2). Otherwise, the cost information is used for further bounding. If the estimated cost is bigger than the upper bound on the total hardware cost, the subtree is also cut (node $<P_1,\phi,P_2P_3>$).

Our estimation strategy is based on an incremental approach considering small changes performed at each decision. After a set of mapping decisions have been taken a new estimation basis is generated using the scheduling algorithm presented in [4].

4 Conclusions

We have presented an approach to system level hardware/software partitioning of embedded systems aiming at fulfilment of timing constraints. Timing constraints are given as an end-to-end deadline imposed on the execution time of the system. A worst case delay for execution time is estimated using scheduling techniques tailored for our problem. We are currently working on the implementation of this algorithm, and real-life examples will be used to test the final implementation.

References

1. G. De Micheli: Hardware/Software codesign: Application domains and design technologies, Hardware/Software Co-Design, NATO ASI (1995), G. De Micheli and M. G. Sami (eds.), Kluwer Academic Publishers, Boston (1996)
2. K. Kuchcinski: Embedded system synthesis by timing constraints solving, Proceedings of 10th International Symposium on System Synthesis (1997)
3. W.H. Kohler and K Steiglitz: Characterization and Theoretical Comparison of Branch-and-Bound Algorithms for Permutation Problems. Journal of the ACM, V. 21, N. 1 (1974) 140-156.
4. P. Eles, K. Kuchcinski, Z. Peng, A. Doboli, and P. Pop: Scheduling of Conditional Process Graphs for the Synthesis of Embedded Systems. Proceedings of the Design Automation and Test in Europe Conference, Paris, February (1998)