Optimisation problems for dynamic concurrent task-based systems

D.Verkest, P.Yang, C.Wong, P.Marchal

IMEC, Kapeldreef 75, Leuven, Belgium

1 Introduction

One of the most critical bottlenecks in many novel multi-media applications is their very dynamic concurrent behaviour. This is especially true because of the quality-of-service (QoS) aspects of these applications. Prominent examples of this can be found in the recent MPEG4 and JPEG2000 standards and especially the new MPEG21 standard. In order to deal with these dynamic issues where tasks and complex data types are created and deleted at run-time based on non-deterministic events, a novel system design paradigm is required. Because of the high computation and communicating requirement from these kind of applications, multi-processor SoC (system on chip) is more and more accepted as a solution (eg., TriMedia), which is also promised by the advance of the processing technology. It is different from traditional general-purpose computers because it is application specific and it is an embedded system, which means costs like energy consumption are of major concern. The task scheduling on such a multiprocessor, embedded multi-media systems forms a real challenge.

This part of the tutorial will focus on the new requirements in system-level synthesis. In particular a ”task concurrency management” problem formulation will be proposed, with special focus on the formal definition of the most crucial optimization problems. The concept of Pareto curve based exploration is crucial in these formulations.

2 Novel Task Scheduling Methods

The current real-time operating systems (RTOS) actually provide limited support for task scheduling. Only some simple scheduling algorithms, such as round robin, time slice and fixed priority preemptive scheduling are widely implemented, and at most only the performance is considered. In other words, they only consider how to meet real-time constraints, not the energy consumption, QoS or some other features emerged in the novel, dynamic applications.

When the concern is the system energy or power consumption, decreasing the supply voltage is lucrative to a low power design for a CMOS digital circuit, because the energy consumption of CMOS digital circuits is approximately proportional to the square of the supply voltage (for power it is the cube), though it will slow down the cycle speed as well. Observations on the dynamic multimedia applications show that there is a big variation of the system computation load. The average system load can be quite low compared to the more less frequently highest demand. To meet the highest computation demand, the system must be fast and more energy greedy, but it can be slow down whenever the demand is at a less level to save energy, which provides the good reason for dynamic voltage scheduling (DVS). More and more researchers have been attracted to this area and many papers and algorithms have been published. A good overview can be found in [6]. Off-line scheduling algorithms for non-preemptive hard real-time tasks are discussed in [1, 3]. In [2, 3, 4], more general variable voltage processor models are used assuming that processor voltage can not change instantaneously or continuously. The more practical processor models make the problem much harder to solve. Several techniques have been developed recently to handle also QoS issues [5]. These techniques however do not yet consider any dependencies between the tasks.

3 Task concurrency management approach and optimisation problems

The purpose of the task-concurrency management approach at IMEC [7, 8] to determine a cost-optimal, constraint-driven scheduling, allocation, and assignment of a given set of tasks to a set of processors. It also includes system-level code transformations to improve the initial starting point from the concurrency point of view (see figure 1 for the complete flow). Different processors execute the same thread node at different speeds and different costs (energy consumption). These differences also make it possible to explore a cost-performance trade-off at the system level. Hence the techniques should not produce just one design point but the entire Pareto-optimal trade-off curve.

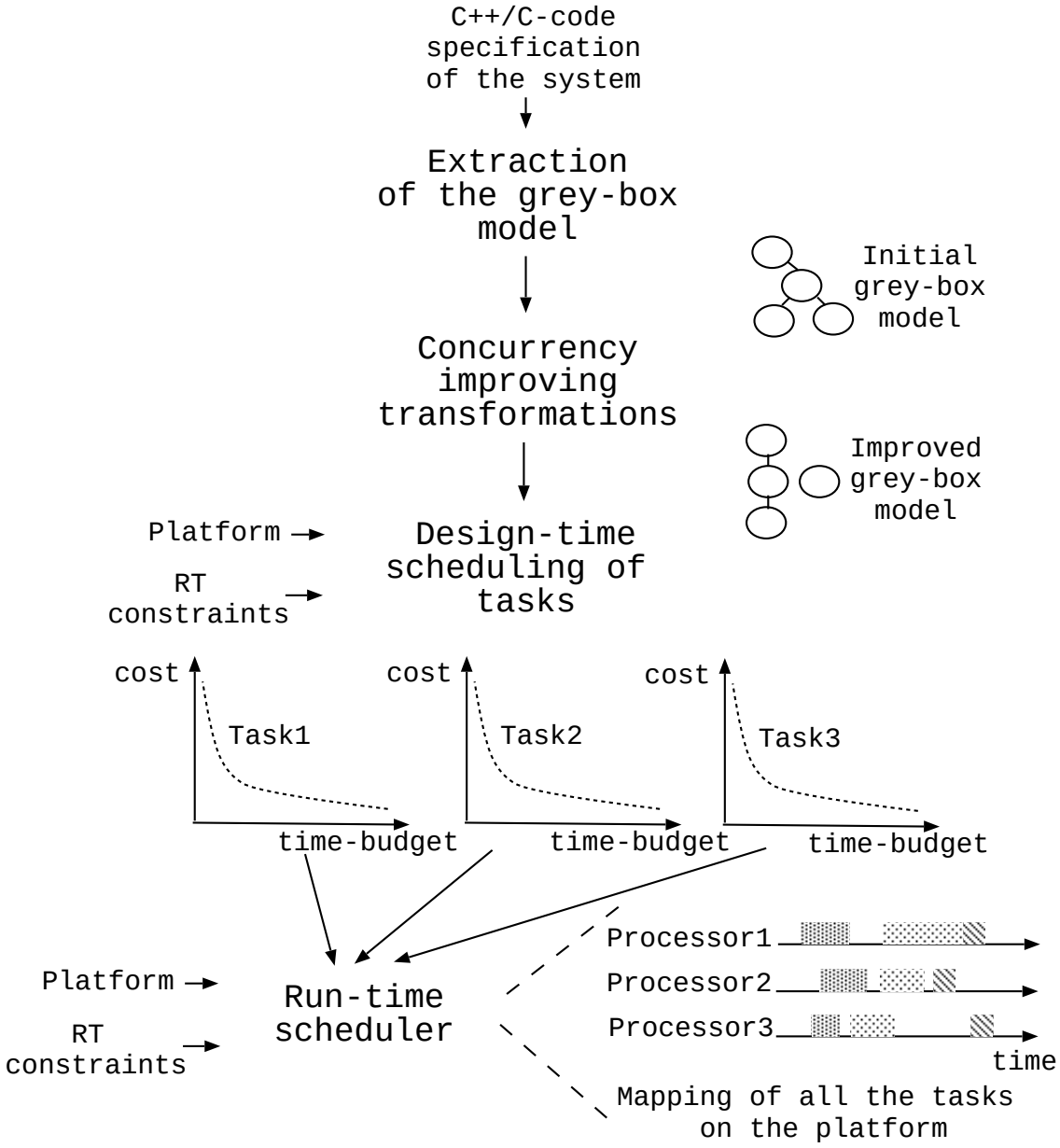

Figure 1: The task concurrency management flow.

The task scheduling problem is attacked hierarchically with two phase in the Task Concurrency Management (TCM) project[7]. First, a design-time task scheduling is performed to some small pieces of a complete task (bounded by deterministic boundaries)[9]. It includes the assignment of the voltage

of the processors on which a small piece of code is running, so in that respect it belongs to the DVS class, but with a discrete set of voltages. Different from a normal ”static” task scheduler, this technique generate not only one but all possible working points (Pareto curve) that dominate the other feasible solutions in the performance-cost trade-off space. A complete problem formulation that can be used as basis for optimisation techniques is provided in [8]. That paper also contains one possible heuristic to solve the problem under certain assumptions.

Next, a run-time scheduling step [10] selects an optimal design-time scheduling option (working point) and combine them to get the complete task scheduling. Given the Pareto curves, this decision can be dynamically updated in a global way, depending on the overall run-time behaviour of the system. So both the current system status and the available resources are continuously incorporated to obtain a optimized system scheduling. A problem formulation that can again be used as basis for optimisation techniques is provided in [11]. That paper also contains experiments demonstrating the possible impact of the global approach.

We separate task scheduling into two phases for three reasons. First, this scheme better optimizes the embedded software design. Second, it gives the entire system more runtime flexibility. Third, it reduces runtime computation complexity.

This can be explained with the following example. First of all, we would like to explain the terminology we use in this tutorial.

non-determinism a state in which behaviors (such as latency or execution time) can vary even with the same system input. Interrupts and events can cause nondeterminism

Pareto curve a set of Pareto-optimal points. Each point represents an optimal solution in at least one trade-off direction when all other directions are fixed.

thread a group of thread frames; an independent piece of code that performs a specific function.

thread frame a group of thread nodes. By definition, nondeterministic behaviors can occur only at the boundary of thread frames. The design-time scheduler works inside each thread frame, whereas the runtime scheduler treats a thread frame as an atomic scheduling unit.

thread node the atomic scheduling unit of our design-time scheduler; consists of control-dataflow graph (CDFG) nodes and arcs.

Fig. 2 illustrates the two-phase scheduling scheme. In this figure, thread frame one consists of three

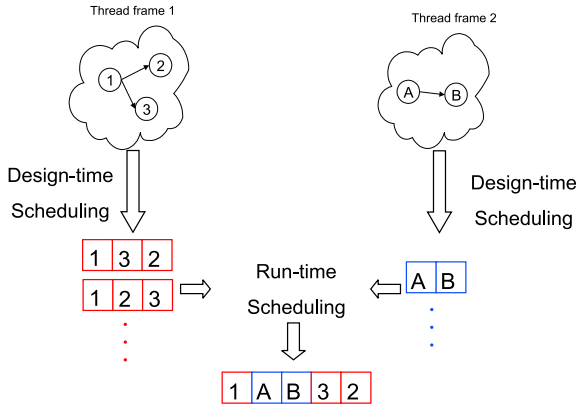

Figure 2: A two phase scheduling method.

thread nodes, 1, 2 and 3. At design time, different scheduling and assignment combinations will be tried. For example, in the first solution, thread node 3 is scheduled before thread node 2, while after node 2 in the second solution. Given a thread frame, the design-time scheduler explores these combinations and generates a Pareto curve for the performance-cost trade-off. Every point in the curve is better than any other solution in at least one way. That is, it consumes the lest energy under a given time constraint or it finishes earliest under a given energy consumption constraint. Design-time scheduling takes place at

compile time, so the design-time scheduler can exert as much computation effort as necessary – provided that it produces a better result, thus reducing the computation effort of runtime scheduling later. The same scheduling and assignment variations can also be explored for thread frame 2. At run time, the runtime scheduler will choose one solution from each thread frame (solution 1 for thread frame one) and combine them to derive the system scheduling.

The runtime scheduler works at thread frame granularity. When new thread frames come into being, the runtime scheduler tries to satisfy their time constraints and minimize system energy consumption as well. The details inside a thread frame, such as execution time or each thread node's data dependency, remain invisible to the runtime scheduler, reducing the thread frame's complexity substantially. The design-time scheduler passes only a few useful Pareto curve features to the runtime scheduler, which uses them to find a reasonable cycle budget distribution for all the running thread frames. Thus, the runtime scheduler is not a traditional dynamic scheduler because it must choose from available options in addition to scheduling them.

References

[1] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava. Power Optimization of Variable Voltage Core-Based Systems. In *Proceedings of the 35th Design Automation Conference*, pages 176–81, San Francisco, CA, 1998.

[2] I. Hong, G. Qu, M. Potkonjak, and M. B. Srivastava. Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors. In *Proceedings of the IEEE Real-Time System Symposium*, 1998.

[3] T. Ishihara and H. Yasuura. Voltage Scheduling Problem for Dynamically Variable Voltage Processors. In *Proceedings of International Symposium on Low Power Electronic Device*, pages 197–202, 1998.

[4] T. Okuma, T. Ishihara, and H. Yasuura. Real-Time Task Scheduling for a Variable Voltage Processor. In *Proceedings of International Symposium on System Synthesis*, pages 24–29, 1999.

[5] Q.Gu, M.Potkonjak, "Achieving utility arbitrarily close to the optimal with limited energy", *Proc. IEEE Intnl. Symp. on Low Power Design*, Rapallo, Italy, pp.125-130, Aug. 2000.

[6] G. Quan and X. Hu. Energy Efficient Fixed-Priority Scheduling for Real-Time Systems on Variable Voltage Processors. In *Proceedings of the 38th Design Automation Conference*, 2001.

[7] F. Thoen and F. Catthoor. *Modeling, Verification and Exploration of Task-level Concurrency in Real-Time Embedded Systems*. Kluwer Academic Publishers, 1999.

[8] C. Wong, P. Marchal, et al. Task Concurrency Management Methodology to Schedule the MPEG4 IM1 Player on a Highly Parallel Processor Platform. In *Proceedings of the International Workshop on Hardware/Software Codesign(CODES)*, 2001.

[9] C. Wong, F. Thoen, F. Catthoor, and D. Verkest. Static Task Scheduling of Embedded Systems. In *International Workshop on System Design Automation*, 2000.

[10] P. Yang, D. Desmet, F. Catthoor, and D. Verkest. Dynamic Scheduling of Concurrent Tasks with Cost Performance Trade-off. In *Int. Conf. Compilers, Architectures, and Synthesis for Embedded Systems*, San Jose, CA, 2000.

[11] P. Yang, C. Wong, P. Marchal, F. Catthoor, D. Desmet, D. Verkest, and R. Lauwereins. Energy-aware Runtime Scheduling for Embedded Multiprocessor SoCs. *IEEE Design & Test of Computers*, 19(3), Sept. 2001.