Task concurrency management methodology to schedule the MPEG4 IM1 player on a highly parallel processor platform

Chun Wong
IMEC, Kapeldreef 75, Leuven, Belgium
chwong@imec.be

Paul Marchal
IMEC
marchal@imec.be

Peng Yang *
IMEC
yangp@imec.be

ABSTRACT

This paper addresses the concurrent task management of complex multi-media systems, like the MPEG4 IM1 player, with emphasis on how to derive energy-cost *vs* time-budget curves through task scheduling on a multi-processor platform. Starting from the original "standard" specification, we extract the concurrency originally hidden by implementation decisions in a "grey-box" model. Then we have applied two high-level transformations on this model to improve the task-level concurrency. Finally, by scheduling the transformed task-graph, we have derived energy-cost *vs* time-budget curves. These curves will be used to get globally optimized design decisions when combining subsystems into one complete system or to be used by a dynamic scheduler. The results on the MPEG4 IM1 player confirm the validity of our assumptions and the usefulness of our approach.

Keywords

concurrency, scheduling, MPEG-4, embedded system, cost-efficiency

1. INTRODUCTION

Today, a new heterogeneous architectural design paradigm is emerging, usually called a *platform*. It includes several programmable components, augmented with some specialized data paths or co-processors (accelerators). The programmable components run software, being slow to medium speed algorithms, while time-critical parts are executed on dedicated hardware accelerators. By this evolution, embedded processors become ubiquitous and a new role for embedded software in contemporary and future ASIC (application specific IC) systems is reserved. When looking at existing design practices for mapping software (and hardware) on such a platform, one can only conclude that these systems nowadays are designed in a very *ad hoc* manner. The design trajectory starts by identifying the global specification entities that functionally belong together, called *tasks* or *processes*. Most of the time, these tasks and processes are at least partly dynamically created and deleted. This step is followed by a manual *hardware-software partitioning*. Because of separate implementation of the different tasks and of the software and hardware, afterwards a *system integration* step is inevitable. This manual step performs the *system/software embedding*, and synthesizes the interface hardware, which closes the gap between the software and the hardware component.

The main goal of system/software embedding is to encapsulate the concurrent tasks in a control shell which takes care of the *task scheduling* (*software scheduling* in the restricted sense) and the inter-task communication. Task scheduling is an error-prone process that requires computer assistance to consider the many interactions between constraints. Unfortunately, current design practices for reactive real-time systems are *ad hoc* and not very retargetable. From this, one can conclude that a need exists for a systematic methodology at the system level for the *co-design of hardware and software* including especially the management of dynamic concurrent tasks. In this paper we will present in Section 3 the framework of a methodology that we are currently developing to solve the task concurrency problem. An important step in this methodology is the design-time mapping of different tasks in a cost-efficient way on a multiple processor platform. In Section 4.2 we will explain a heuristic that helps us to select the most important task to run first on the platform. We will illustrate this scheduling heuristic with an example extracted from a very complex realistic driver, namely the MPEG4 IM1 player. The results will be represented in Pareto curves which do not give a single working point but which allow us to globally trade-off cost (e.g. energy) *vs* constraints (e.g. time-budget).

2. TARGET APPLICATION

The target applications of our task-level system synthesis approach are advanced real-time multi-media and information processing(RMP) systems, such as consumer multi-media electronics and personal communication systems. These applications involve a combination of complex data- and control-flow where complex data types are manipulated and transferred in a dynamic way including creation and deletion of both data and tasks. Most of these applications are implemented with compact and portable devices, putting stringent constraints on the degree of integration (i.e. chip area) and on their power consumption. Secondly, these systems are extremely *heterogeneous* in nature and combine high performance data processing (e.g. data processing on transmission data input) as well as slow rate control processing (e.g. system control functions), syn-

*They are all also Ph.D. students of K.U.Leuven-ESAT

chronous as well as asynchronous parts, analog and digital, and so on. Thirdly, time-to-market has become a critical factor in the design phase. Finally, these systems are subjected to stringent real-time constraints (both hard and soft deadlines are present), complicating their implementation considerably.

The main driver for our research is the IM1 player (see Fig. 1). This player is based on the MPEG4 standard, which specifies a system for the communication of interactive audio-visual scenes composed of objects. As this player consists of many different modules (audio, video, 3D animation, etc., coded in over 80,000 lines of high-level C++ specification), we believe it is representative for other multi-media applications.

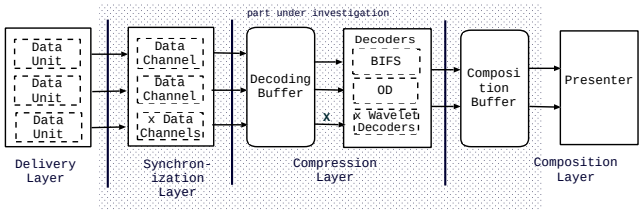

Figure 1: System level modules of the MPEG4 IM1 player

Each frame in a MPEG4 compatible input stream consists of several parts, video object planes (VOPs), that can be decoded independently. In this paper we will assume that up to 5 different VOPs are simultaneously active, which is realistic. For each VOP two interacting controllers need to be scheduled (see Fig. 3). Several other decoders may be needed to interpret additional information required by the VOPs (such as texture-data, audio, video). In this paper we focus on the mapping of the controllers on the programmable components of the platform. The other more static signal processing tasks are assigned to dedicated accelerators.

3. FRAMEWORK OF THE TASK CONCURRENCY MANAGEMENT METHODOLOGY AT THE GREY-BOX ABSTRACTION LEVEL

The design of concurrent real-time embedded systems, especially embedded software, is a difficult problem to be performed manually due to the complex consumer-producer relationships, the presence of various timing constraints, the non-determinism in the specification and the sometimes tight interaction with the underlying hardware. Here we present a new cost-oriented approach to the problem of task scheduling on multiple processors. It fits in a global task concurrency management approach outlined in [2]. The approach uses as much as possible pre-ordering of the concurrent behavior under real-time constraints and minimizes the run-time overhead. At the same time, the scheduler tries to minimize cost such as the energy consumption.

The framework of our methodology is depicted in Fig. 2. An embedded system can be specified at a grey-box abstraction level in a combined MTG-CDFG model [1, 33, 36][1]. This specification is functional in representing the concepts of concurrency, timing constraints and interaction at either an abstract or a more detailed level, depending on what is required to perform good exploration decisions afterwards. As in [2], the task concurrency management can

[1] MTG is the acronym for multi thread graph

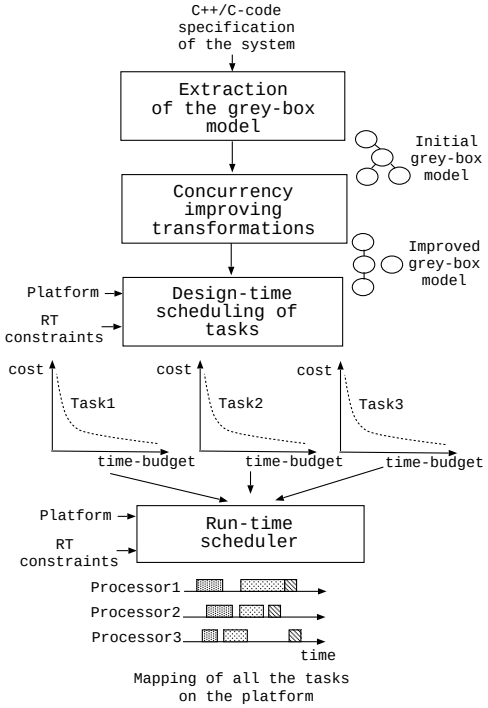

Figure 2: Framework of the task concurrency management methodology

be implemented in four major steps. Firstly, the concurrency extraction is performed. Transformations [3] on the specified MTG-CDFG are then applied to increase the opportunities for concurrency exploration and cost minimization. After the extraction, we obtain a set of *tasks*, each of which consists of many *thread nodes*. *Tasks* can be viewed as a more or less independent part of the whole application. Then static scheduling will be applied inside each *task* at design time, including processor assignment in the context of multiple processing elements (PEs). Finally, a dynamic scheduler will schedule these *tasks* at run time on the given platform [5].

The purpose of task concurrency management is to determine a cost-optimal constraint-driven scheduling, allocation and assignment of various *thread nodes* to a set of processors. Each *thread node* will be executed at a different speed on a different processor with a different cost, i.e. energy consumption. Given a *task*, static scheduling is done at design time to explore all the possibilities of valid assignment and scheduling. Each possibility means a different choice in time budget and energy consumption trade-off (i.e. another working point) and together they compose the global Pareto curve. Since the static scheduling is done at design time, computation efforts can be paid as much as necessary, provided that it can give a better scheduling result and reduce the computation efforts of dynamic scheduling.

4. SCHEDULING HEURISTIC FOR A LARGE NUMBER OF THREAD NODES

4.1 Related work and motivation

Abundant work has been done on scheduling. In literature, the granularity levels which different scheduling algorithms work at are quite different. However, the entity which scheduling algorithms deal with is always called *task*, *job* or *node*. To avoid confusion,

it is better to know that the meaning of these terminologies in our work can be different from other existing work.

When a set of concurrent tasks, i.e., tasks that can overlap in time, has to be executed on one or more processors, *scheduling algorithm*, must be applied to decide the order in which those tasks are executed. For a multiprocessor system, another procedure, assignment is also needed to determine on which processor one task will be executed.

In the real-time community, researchers use a *black-box* view of the task behavior. Comprehensive overviews of scheduling algorithms for real-time systems are given in [1, 6, 7, 8]. In contrast, in the embedded system community, many papers focus on *white-box* task descriptions [9, 10], which are typically not available at the early design stage. A *grey-box* model is used in our approach. As an example, we show in Fig. 3 the grey-box model we are going to use in the following scheduling experiment. Formal definition of this model is given in [1]. The most important semantics of this model is explained in the appendix.

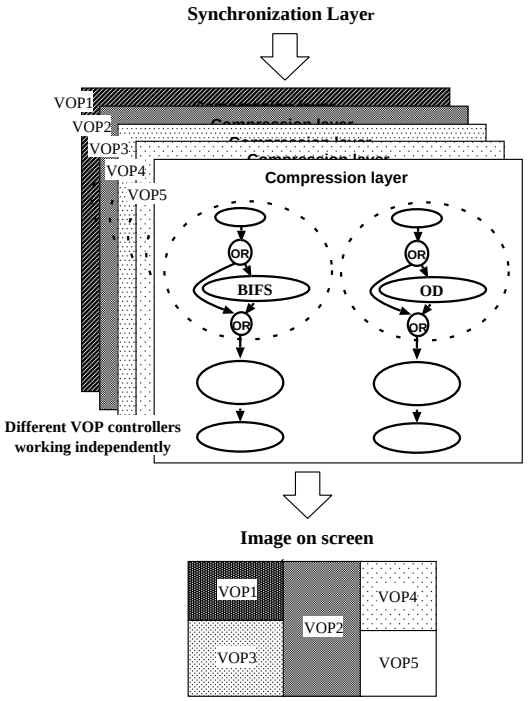

Figure 3: Grey-box model of the compression layer of the IM1 player

Existing scheduling algorithms can be roughly divided into dynamic and static scheduling. In a multiprocessor context, when the application has a large amount of non-deterministic behavior, dynamic scheduling has the flexibility to balance the computation load of processors at run-time. However, the run-time overhead for code size and energy consumption may be excessive. In addition, most multimedia applications have limited non-deterministic behavior. Moreover, it is usually impossible to make globally optimal scheduling decisions at run-time. Consequently, scheduling decisions should be made as much as possible at design-time. We selected a combination of the static and dynamic scheduling to take advantage of both of them.

A large body of scheduling algorithms stem from Liu and Layland's classical paper[11], in which they gave five basical assumptions and studied the optimizability and schedulability of Rate Monotonic (RM) and Earliest Deadline First (EDF) algorithms in a fixed and dynamic priority environment respectively. Most of the later work can be classified as one of the two above, namely fixed priority or dynamic priority, and aims at relaxing the assumptions given in Liu's paper and dealing with shared resources, aperiodic tasks, and tasks with different importance levels. Leung and Whitehead [12] suggested Deadline Monotonic to relax the deadline assumption. Later Chetto *et al* [13] introduced a transformation to the original task set to include the precedence constraints in EDF. To provide services for aperiodic or sporadic tasks, several bandwidth preserving algorithms have been proposed [14, 15, 16, 17, 18]. The problem of resource sharing is tackled with Priority Inheritance Protocol in [6, 19]. They provide a solid basis of performance oriented scheduling theory.

Since more and more embedded systems are targeted at multiple processor architectures, multiple processor scheduling plays a more and more important role. El-Rewini *et al* [20] give a clear introduction to the task scheduling in multiprocessing systems. Hou *et al* [21] alleviate the *saturation effect* caused by excessive interprocessor communication in distributed embedded systems. Hoang *et al* [22] try to maximize the throughput by balancing the computation load of the distributed processors. All the partition and scheduling decisions are made at design time. This approach is limited to pure data flow applications. Teich *et al* [23] try to handle non-determinism in the heterogeneous embedded system. They have avoided the explicit enumeration of execution paths. However, since their work is based on the *white-box* viewpoint, it can not scale to large systems. On the contrary, our *grey-box* approach aims at handling such large systems. Yen *et al* [24, 25] try to combine the processor allocation and process scheduling into a gradient-search cosynthesis algorithm. It's a heuristic method and can only handle periodic tasks statically.

However, in most of the above work, performance is the only concern. In other words, they only consider how to meet real-time constraints. For embedded systems, cost factors, like energy, must be taken into account.

Only a few papers involve the cost, like power, apart from performance issues. Gruian *et al* [26] used constraint programming to minimize the energy consumption at system level but their method is purely static and no dynamic tactic is applied to exploit more energy reduction. Since the energy consumption of CMOS digital circuits is approximately proportional to the square of the supply voltage, decreasing the supply voltage is lucrative to low power design, though it will also slow down the clock speed. Traditionally, the CPU works at a fixed supply voltage, even though at a light workload. In fact, under such situations, the fast speed of the CPU is unnecessary and can be traded for a lower energy/power consumption by reducing the supply voltage. Chandrakasan[27] compared several energy saving tactics concerned Dynamic Voltage Scheduling. Based on this investigation, several real-time scheduling algorithms are provided, e.g. by Hong[28] and Okuma[29]. However, the platforms for such algorithms are restricted to a single processor with dynamically variable supply voltage. Jha *et al* [30, 31, 32] go pretty far in the power-consicious design. Their early work [30, 31] focus more on how to meet hard real-time constraints of aperiodic tasks and how to reduce hardware cost. Based on techniques in this early work, power in multiple processor context is treated by

evenly distributing the workload [32]. However, their work doesn't tackle power consumption directly. No manifest power and performance relation is used to steer the design-space exploration. In addition, they assume a continuously scaled voltage, which simplifies the processor assignment problem.

4.2 A new scheduling heuristic for large number of tasks

Our approach can be applied to the multiprocessor platform without the need of changing the processor voltage dynamically. For a given platform, i.e., the number of high-speed processors and low-speed processors, the heuristic will derive an energy-cost *vs* time-budget curve. Compared with other scheduling algorithms, like MILP (mixed integer linear programming), the heuristic does not treat real-time constraints directly. Hence, the computation complexity is reduced. The heuristic derives a set of working points on the energy-cost *vs* time-budget plane instead of only one point. These working points range from the most optimal performance point within that given platform to the point barely meeting timing constraints. Existence of these different working points is mostly due to the different *thread node* to processor assignments (some of it is due to idle time in the schedule). As a result we also have to deal with a crucial assignment problem. We will show in Section 5 how a trade-off between energy-cost and time-budget can be made by a proper assignment decision. These points are crucial when combining subsystems into a complete system or during the dynamic scheduling stage. The reason is that in both cases we need to select working points of subsystems and combine them into a globally optimal working point.

To schedule the *thread nodes* on the above multiple processor platform, we have developed a static scheduling heuristic. The heuristic uses two criteria to decide which *thread node* to run first. The first criterion is the *self-weight* of a *thread node*. It is defined as the execution time of the *thread node* on the low-speed processor. The larger the self-weight of a *thread node*, the higher its priority to be mapped on a high-speed processor.

The second criterion is the *load* of a *thread node*. If some *thread nodes* are depending on a *thread node*, the sum of the self-weights of all the dependent *thread nodes* is defined as the load of the *thread node*. It is worth noting that "dependent" means control dependence. The more load a *thread node* has, the earlier it should start to execute.

From the above criteria, when a processor is available, the following strategy takes care of selecting a candidate *thread node*.

1. When a *thread node* is dominant both in the self-weight and load over the other candidate *thread nodes*, it will be scheduled first.
2. When one *thread node* has a dominant self-weight and another *thread node* has a dominant load, either of them can be scheduled first. By alternating their order different points on the energy-cost *vs* time-budget plane can be generated.

It is better to realize that the heuristic implicitly includes energy considerations. Because for a given processor, the energy consumption is directly related to the execution time. The *self-weight* and *load* in the heuristic are merely two interpretations of execution time from differen perspectives. Applying this heuristic to the IM1 player will be discussed in the following section. Even though it seems relatively simple, it turns out to be very effective for scheduling the tasks in the MPEG-4 IM1 player.

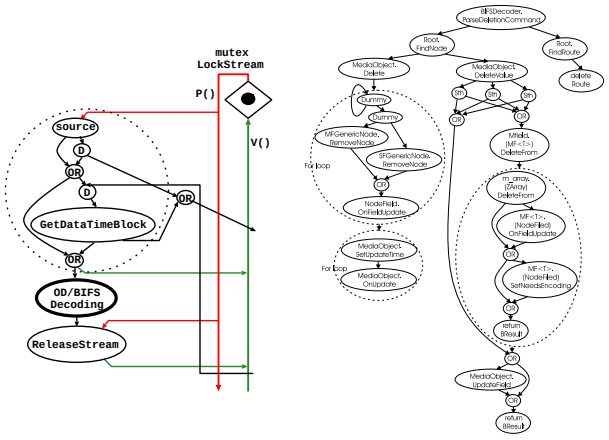

Figure 4: Grey-box model of BIFS decoding and decoder setup (left) and an example of the unfolding of the BIFS decoding task (right)

.

5. TASK SCHEDULING EXPERIMENT

To get smooth scenes, a frame rate of 30ms is required. Currently, the IM1 player is implemented on a Pentium or Pentium-compatible platform. Such an implementation is far from meeting the 30ms timing constraint. This real-time constraint is only reachable when an extremely parallel and powerful instruction-set processor is used. In our experiment, the IM1 player is mapped to a multiple processor platform including custom accelerators next to the instruction-set processors. We have extracted the task graph from the IM1 implementation code. By analyzing the profiling information, we know how many VOPs one frame contains, how many BIFS nodes one VOP has and how many decoder setup are done in one VOP.

We pick up an example frame of 5 VOPs. The number of BIFS nodes and the number of decoder setup are listed in Table 1. For every BIFS node decoding and decoder setup, we get the execution time from the profiling data. The grey-box model of a typical BIFS decoding and decoder setup is shown in Fig. 4. The left part of Fig. 4 is the context of the BIFS decoding or decoder setup. The right part is the unfolding of an example BIFS decoding. In reality, instances of the BIFS decoding and decoder setup take place inside a VOP with variations in the underlying graphs and execution times.

VOP	1	2	3	4	5
Number of BIFS nodes	4	1	4	4	3
Number of decoder setup	12	3	3	5	9

Table 1: A typical frame structure

To meet the 30ms timing constraint for the protocol subsystem, we use a number of StrongARM processors [34] and custom accelerators. The accelerator can be for example, the Ozone custom processor design of wavelet functionality [35]. The processors run at either 100MHz or 233MHz. For a *thread node*, the execution time and energy cost are different on the two processors. We use t_L and t_H to denote the execution time on the low-speed and high-speed

processor respectively. Δt is the difference between these two execution times. Similarly, we use E_L and E_H to denote its energy cost on these two processors. ΔE is the energy difference. Our calculations are based on the following formulae.

$$\Delta t = t_L - t_H = (1 - \frac{1}{2.33})t_L \quad (1)$$

$$\Delta E = E_H - E_L = (2.33^2 - 1)E_L \quad (2)$$

We have tried different processor combinations. A given processor combination means a number of fixed high-speed and low-speed processors. For each processor combination, we make different processor assignment and put *thread nodes* into different orders by the static scheduling heuristic described in Section 4.2. Hence, an energy cost *vs* time-budget curve is derived. Altogether, there are four processor combinations which satisfy the timing constraints (Table 2).

Processor combination	1	2	3	4
Number of high-speed processors	6	5	4	3
Number of low-speed processors	0	3	5	7
Total Number of processors	6	8	9	10

Table 2: Different processor combinations

For Combination 1 and 4, only one solution exists which meets the timing constraints. Consequently, there is no trade-off between energy cost and time-budget. For Combination 2 and 3, we have derived a Pareto-optimal energy *vs* time-budget curve for each of them.

Applying this scheduling heuristic to the sample problem, we have derived two curves for processor combination 2 and 3, which are shown in Fig. 5. From the two Pareto curves, we know that points on the curve of Combination 3 provide globally optimal working points until point 6 is reached. A lower time-budget, i.e., a more tight timing constraint cannot be satisfied by the platform of Combination 3. This is because the existing concurrency in the task graph has been fully utilized by the processors in Combination 3. To further decrease the time-budget, we have to use extra high-speed processors to reduce the critical path. That is, either to replace low-speed (low-power) processors with high-speed (high-power) processors or to introduce additional high-speed processors. In other words, two ways exist to decrease the time-budget. One is to utilize the concurrency, the other is to reduce the critical path. Therefore, to get a lower time-budget, we have to resort to the platform of Combination 2 and pay a large energy penalty. This explains the big jump of the global Pareto curve from point 6 of Combination 3 to point 6 of Combination 2.

As no related work exists that fully matches our problem statement, we have made a comparison with an exact MILP formulation. We have applied a well-known public domain MILP solver [4] to this scheduling problem. Since the amount of MILP description for scheduling the task graph on a multiple processor platform will explode, we have to apply the MILP solver to a two processor platform. Shown in Fig. 6 are three curves. One is got from the heuristic, the rest two are the outputs from the MILP solver for the first time and after it runs for five days, respectivley. It is clearly seen that the MILP solver has made large improvement after five days. However, to get all the optimal points over the time-budget axis, which should be lower than the points given by the heuristic, it will

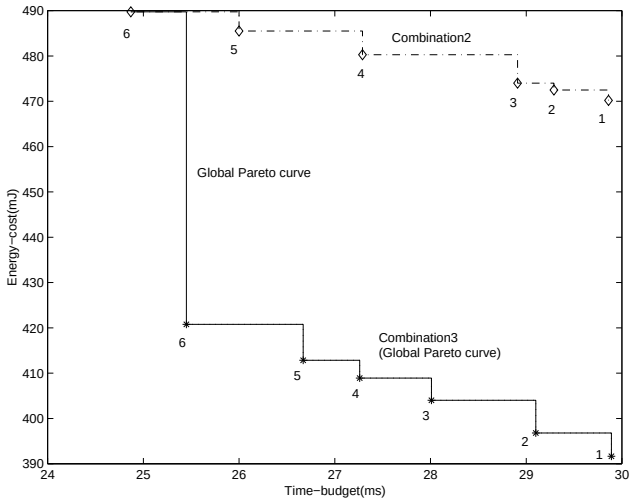

Figure 5: Pareto curves for processor combination 2 and 3

take even longer time. The fact motivates the need for a heuristic. It also substantiates our claim that our heuristic provides a good quality and run-time complexity. We have only one major demonstrator at present, but we believe that the complexity and heterogeneity of the IM1 player is representative of a whole class of dynamic and concurrent applications. Therefore the result in this section should be conclusive enough.

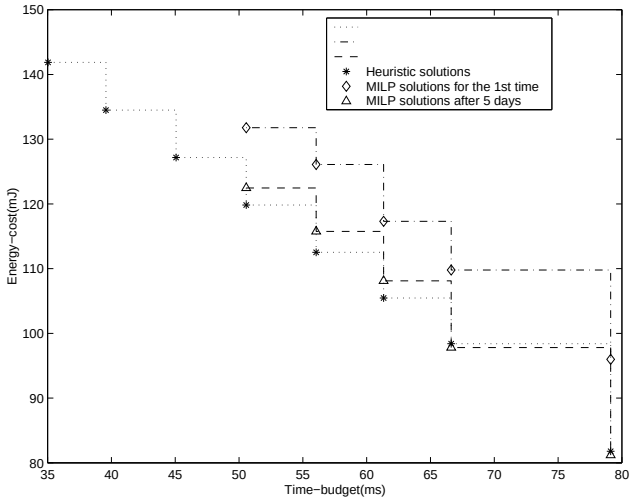

Figure 6: Comparison of energy *vs* time-budget curves derived by the heuristic and MILP

6. CONCLUSIONS

We have demonstrated in this paper that a grey-box task graph model should be used to design advanced real-time dynamic multimedia applications. This task-level model allows us to systematically explore the design at a high abstraction level in order to obtain more and better task schedules. Transformations on the task graph level have been applied to improve this further. On a multiprocessor targeted platform including custom accelerators for the video coders, the different schedules obtained with a heuristic scheduling approach are represented in Pareto curves trading-off time-budget *vs* energy-cost. This information is crucial to decide on run-time

trade-offs involving other subsystems present in the entire application.

7. ADDITIONAL AUTHORS

Aggeliki Prayati (Department of Electrical and Computer Engineering, University of Patras, Greece), Francky Catthoor (IMEC, also professor of K.U.Leuven-ESAT), Rudy Lauwereins (Katholieke Universiteit Leuven-ESAT, Leuven, Belgium), Diederik Verkest (IMEC) and Hugo De Man (IMEC, also professor of K.U.Leuven-ESAT)

8. REFERENCES

[1] F. Thoen and F. Catthoor, "Modeling, Verification and Exploration of Task-level Concurrency in Real-Time Embedded Systems", ISBN 0-7923-7737-0, Kluwer Aca. Publ., Boston, 1999.

[2] A. Prayati, C. Wong, P. Marchal *et al*, "Task Concurrency Management Experiment for Power-efficient Speed-up of Embedded MPEG4 IM1 Player", Proceedings of International Workshops on Parallel Processing 2000, Toronto, August 2000, pp. 453-460.

[3] P. Marchal, C. Wong, A. Prayati *et al*, "Impact of task-level concurrency transformations on the MPEG4 IM1 player for weakly parallel processor platforms", International Conference on Parallel Architectures and Compilation Techniques 2000, Philadelphia, October 2000.

[4] ftp://ftp.ics.ele.tue.nl/pub/lp_ solve

[5] P. Yang, D. Desmet, F. Catthoor, and D. Verkest, "Dynamic Scheduling of Concurrent Tasks with Cost Performance Trade-off", International conference on compilers, architecture, and synthesis for embedded systems, San Jose, November 2001.

[6] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization" *IEEE Transactions on Computers*, 39(9), Sept. 1990, pp. 1175-1185.

[7] K. Ramamritham and J. A. Stankovic, "Scheduling Algorithms and Operation Systems Support for Real-Time Systems" *Proceedings of the IEEE*, 82(1), Jan. 1994, pp. 55-67.

[8] N. Audsley, A. Burns, R. Davis, K. tindell, and A. Wellings, "Fixed Priority Pre-emptive Scheduling: an Historical Perspective", *Real-Time Systems*, 8(2), 1995, pp. 173-198.

[9] T. Benner and R. Ernst, "An Approach to Mixed Systems Co-Synthesis", In *Proceedings of the International Workshop on Hardware/Software Codesign(CODES)*, 1997, pp. 9-14.

[10] F. Balarin *et al* , *Hardware-Software Co-Design of Embedded Systems: the POLIS Approach*. Kluwer Academic Publishers, 1997.

[11] C. L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment" *Journal of the Association for Computing Machinery*, 20(1), 1973, pp. 46-61.

[12] J. T. Leung and J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks" *Performance Evaluation*, 2, 1982, pp. 237-250.

[13] H. Chetto, M. Silly, and T. Bouchentouf, "Dynamic Scheduling of Real-Time Tasks Under Precedence Constraints", *Real-Time Systems*, 2, 1990.

[14] B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic Task Scheduling for Hard-Real-Time Systems", *Real-Time Systems*, 1, 1989, pp. 27-60.

[15] J. P. Lehoczky, L. Sha, and J. K. Strosnider, "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments", In *Proceedings of the IEEE Real-Time System Symposium*, 1987, pp. 261-270.

[16] S. Ramos-Thuel and J. P. Lehoczky, "On-Line Scheduling of Hard Deadline Aperiodic Tasks in Fixed-Priority Systems", In *Proceedings of the IEEE Real-Time System Symposium*, 1993, pp. 160–171.

[17] M. Spuri and G. C. Buttazzo, "Efficient Aperiodic Service under Earliest Deadline Scheduling", In *Proceedings of the IEEE Real-Time System Symposium*, 1994, pp. 2-11.

[18] M. Spuri and G. C. Buttazzo, "Scheduling Aperiodic Tasks in Dynamic Priority Systems", *Real-Time Systems*, 10(2), 1996.

[19] T. P. Baker, "Stack-Based Scheduling of Realtime Processes", *Real-Time Systems*, 3(1), 1991, pp. 67-99.

[20] H. El-Rewini, H. H. Ali, and T. Lewis, "Task scheduling in multiprocessing systems", *IEEE Computer*, 28(12), Dec. 1995, pp. 27-37.

[21] C. J. Hou and K. G. Shin, "Allocation of Periodic Task Modules with Precedence and Deadline Constraints in Distributed Real-Time Systems", *IEEE Transactions on Computers*, 46(12), Dec. 1997, pp. 1338-56.

[22] P. D. Hoang and J. M. Rabaey, "Scheduling of DSP Programs onto Multiprocessors for Maximum Throughput", *IEEE Transactions on Signal Processing*, 41(6), June 1993, pp. 2225-2235.

[23] K.Strehl, L.Thiele, D.Ziegenbein, R.Ernst, and J.Teich, "Scheduling hardware/software systems using symbolic techniques", In *Proceedings of the Seventh International Workshop on Hardware/Software Codesign*, 1999, pp. 173-177.

[24] T. Y. Yen and W. Wolf, "Sensitivity-Driven Co-Synthesis of Distributed Embedded Systems", In *Proceedings of International Symposium on System Synthesis*, Sept. 1995, pp. 4-9.

[25] T. Y. Yen and W. Wolf, "Communication Synthesis for Distributed Embedded Systems", In *IEEE/ACM International Conference on Computer-Aided Design*, 1995, pp. 288-294.

[26] F. Gruian and K. Kuchcinski, "Low-Energy Directed Architecture Selection and Task Scheduling", In *EUROMICRO'99*, 1999, pp. 296-302.

[27] A. Chandrakasan, V. Gutnik, and T. Xanthopoulos, "Data Driven Signal Processing: An Approach for Energy Efficient Computing", In *Proceedings of International Symposium on Low Power Electronic Device*, 1996, pp. 347-352.

[28] I. Hong, M. Potkonjak, and M. B. Srivastava, "On-Line Scheduling of Hard Real-Time Tasks on Variable Voltage Processor", In *IEEE/ACM International Conference on Computer-Aided Design*, San Jose, CA, 1998, pp. 653-656.

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

[30] B.Dave, G. Lakshminarayana, and N.Jha, "COSYN: Hardware-software co-synthesis of embedded systems", *Proc. 34th Design Automation Conference*, New York, USA, 1997, pp. 703-708

[31] B.Dave and N.Jha, "CASPER: concurrent hardware-software co-synthesis of hard real-time aperiodic and periodic specifications of embedded system architectures", *Proc. 1st ACM/IEEE Design and Test in Europe Conf.*, Paris, France, Feb. 1998, pp. 118-124.

[32] J.Luo and N.Jha, "Power-conscious Joint Scheduling of Periodic Task Graphs and Aperiodic Tasks in Distributed Real-time Embedded Systems", *Proc. International Conference on Computer Aided Design 2000*, San Jose, USA, Nov. 2000, pp. 357-364.

[33] N. Cossement, R. Lauwereins, and F. Catthoor, "DF*: An extension of synchronous dataflow with data dependency and non-determinism", accepted for *Forum on Design Languages (FDL)*, Tuebingen, Germany, Sep. 2000.

[34] http://developer.intel.com/design/strong/datashts/

[35] L. Nachtergaele, B. Vanhoof, M. Peon, G. Lafruit, J. Bormans, and I.Bolsens, "Implementation of a scalable MPEG-4 wavelet-based visual texture compression system", *Proc. 36th ACM/IEEE Design Automation Conf.*, New Orleans LA, June 1999, pp. 333-336.

[36] C. Wong, F. Thoen, F. Catthoor, and D. Verkest, "A slack-based static task scheduling heuristic for embedded systems", *accepted by Journal of Systems Architectures*