Managing Dynamic Concurrent Tasks in Embedded Real-Time Multimedia Systems

Peng Yang(*) Paul Marchal(*) Chun Wong(*) Stefaan Himpe (***)
Francky Catthoor(**) Patrick David Johan Vounckx Rudy Lauwereins(**) [*]

ABSTRACT

This paper addresses the problem of mapping an application, which is highly dynamic in the future, onto a heterogeneous multiprocessor platform in an energy efficient way. A two-phase scheduling method is used for that purpose. By exploring the Pareto curves and scenarios generated at design time, the run-time scheduler can easily find a good scheduling at a very low overhead, satisfying the system constraints and minimizing the energy consumption. A real-life example from a 3D quality of service kernel is used to show the effectiveness of our method.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Organization and Design—*real-time systems and embedded systems*; D.4.1 [**Operating Systems**]: Processing Management; I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis

General Terms

Design, Algorithms, Performance

Keywords

multiprocessor, embedded system, low-power, scheduling

1. INTRODUCTION

The merging of computers, consumer and communication disciplines gives rise to very fast growing markets for personal communication, multimedia and broadband networks. Technology advances lead to platforms with enormous processing capacity that are however not matched with the required increase in system design productivity.

One of the most critical bottlenecks is the very dynamic concurrent behavior of many of these new applications. They are fully specified in software oriented languages (like Java, UML, SDL, C++) and still need to be executed in real-time cost/energy-sensitive way on the heterogeneous SoC platforms. A way of mapping this SW/HW specification onto an embedded multi-processor platform is required. The main issue is that fully design-time based solutions as proposed earlier in the compiler and system synthesis communities cannot solve the problem, and run-time solutions as present in nowadays operating systems are too inefficient in terms of cost optimization (especially energy consumption) and are also not adapted for the real-time constraints.

This dynamic nature is especially emerging because of the quality-of-service (QoS) aspects of these multi-media and networking applications. Prominent examples of this can be found in the recent MPEG4/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. This paper will focus on the new requirements that result in system-level synthesis. In particular a "task concurrency management" (TCM) problem formulation will be proposed, with special emphasis on the results that can be obtained in terms of where power consumption reduction. The concept of Pareto curve based exploration is crucial in the solution for this problem.

The promising results that can be obtained with such a methodology will be illustrated with an MPEG21 based demonstrator mapped on a multi-processor simulation platform with a hierarchical share memory organization.

[*]Kapeldreef 75, Leuven, Belgium, B3001; also Ph.D. student of K.U.Leuven-ESAT(*), also professor of K.U.Leuven-ESAT(**), Ph.D. student at K.U.Leuven-ESAT (***).

2. PLATFORM BASED DESIGN

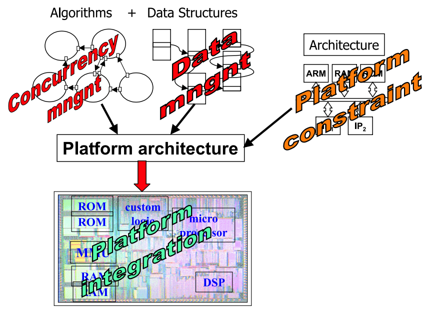

Figure 1: The platform integration.

The future of embedded multimedia applications lies in low-power heterogeneous multiprocessor platforms[10]. In the near future, the silicon market will be driven by low-cost, portable consumer devices which integrate multi-media and wireless technology. Most of these applications will be implemented with compact and portable devices, putting stringent constraints on the degree of integration (i.e. chip area) and on their power consumption (0.1-2W). The applications will require a computational performance of the order (1-40$GOPS$). Current PCs offer this performance. However, their power consumption is too high (10-100W). We must keep this performance and reduce the power consumption at least two to three orders of magnitude. Embedded systems are also subject to stringent real-time constraints, complicating their implementation considerably. Finally, the challenge to embed these applications on portable devices is increased even further because of user interaction: at any moment the user will be able to trigger new services, change the configuration of the running services, stop existing services. Hence, the execution behavior changes dynamically at run-time.

Therefore, the future embedded system should be extremely *fast*, have a extremely *low energy consumption*, and be *flexible* enough to cope with the dynamic behavior of future multi-media applications. In spite of modern voltage scaling techniques, existing single processor systems (e.g. found in PDA's) cannot deliver enough computational performance at a sufficiently low energy budget. On the other hand, full custom hardware solutions can achieve a good performance/energy consumption ratio, but are not flexible enough to cope with the dynamic behavior and have a relatively long time-to-market delay. Also the technology scaling trend with the explosion of masks' costs and of the physical design (especially due to timing closure and test) implies that custom ASICs are only feasible for high volume designs. A combination of the bests of both worlds is required for the majority of the new applications (see [7]). This explains the growing interest in platform based design [6].

The benefits of heterogeneous platforms can however only be fully exploited when applications are efficiently mapped on them. Unfortunately, current design technologies fall behind these advances in computer architecture and processing technology. When looking at contemporary design practices for systems implemented with these heterogeneous multiprocessor platforms, one can only conclude that these systems are nowadays designed in a very *ad hoc* manner. A systematic methodology and its corresponding tool set are definitely needed.

In this paper, we present a systematic design methodology and tool support to embed multi-media applications on platforms. With our design methodology, we target especially the management of dynamic and concurrent tasks and their data on heterogeneous multiprocessor platforms. This requires, given an application which has its own specific control and data structure, and a template for the platform architecture, finding a way to decide the instance platform architecture and how to map the application efficiently onto such an architecture (top part of Fig. 1). We will only focus on the concurrency management issues in this paper, even though the data management part [22] is as crucial.

Platform based design also encompasses a physical implementation problem that can be called platform integration (bottom part Fig. 1), but that is not addressed here.

3. RELATED WORK

Task scheduling in a task concurrency management context has been investigated overwhelmingly in the last decades. When a set of concurrent tasks - that is, tasks that can overlap in time - have to be executed on one or more processors, a predefined method, called *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. A good overview of scheduling algorithms can be found in [27]. In this paper, the terminology *task scheduling* is used for both the ordering and the assignment.

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 and make use of the extra slack time coming from the variation from worst case execution time (WCET). However, the run-time overhead may be excessive and a global optimal scheduling is difficult to find due to the difficulty of the problem. We have selected a combination of a design-time and run-time scheduling here to take advantage of both of them.

Since more and more embedded systems are targeted at multiprocessor architectures, multiple processor scheduling plays an increasingly important role. El-Rewini *et al* [11] give a clear introduction to the task scheduling in multiprocessing systems. Hoang *et al* [14] try to maximize the throughput by balancing the computation load of the distributed processors. All the partition and scheduling decisions are made at compile time. This approach is limited to pure data flow applications. Yen *et al* [35, 34] try to combine the processor allocation and process scheduling into a gradient-search cosynthesis algorithm. It is a heuristic method and can only handle periodic tasks statically.

In 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 as well. Gruian *et al* [13] have used constraint programming to minimize the energy consumption at system level but their method is purely static and no dynamic policy is applied to exploit more energy reduction.

Recently, DVS (dynamic voltage scaling) is getting more consideration. Since the energy consumption of CMOS digital circuits is approximately proportional to the square of the supply voltage, decreasing the supply voltage is advantageous to low power design, though it will also slow down the cycle speed. [1] Traditionally, the CPU works at a fixed supply voltage, even 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 et al. [5] have compared several energy saving techniques concerned DVS. Based on these investigations, several real-time scheduling algorithms are provided, e.g. by Hong [16] and Okuma [23]. Recently Transmeta has released a commercial processor that allows the dynamic voltage scheduling technique [8]. Normally the scheduling techniques developed in the real-time commu-

[1]On the long term, the technologically available range for this scaling will decrease as already mentioned

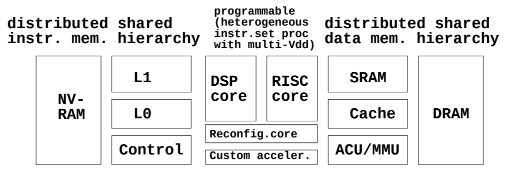
distributed shared instr. mem. hierarchy
programmable (heterogeneous instr.set proc with multi-Vdd)
distributed shared data mem. hierarchy
NV-RAM
L1
L0
Control
DSP core
RISC core
Reconfig.core
Custom acceler.
SRAM
Cache
ACU/MMU
DRAM

Figure 2: A typical heterogeneous processor platform

nity, e.g., fixed priority scheduling, EDF (earliest deadline first), slack stealing *etc.*, are still applicable in DVS, only an extra voltage decision step is needed. A good survey of the recently proposed DVS algorithms can be found in [18] and [26]. Most related work, e.g., [4, 15, 26], concentrates on saving energy of independent tasks or on a single processor, while the tasks in real-world applications usually have control or data dependencies and mapped to multiple processors, heterogeneous or homogeneous.

Power consumption in a multiple processor context is treated in [20] by evenly distributing the workload. However, no manifest power and performance relation is used to steer the design-space exploration. In addition, they also assume a continuously scalable working voltage. In [21], for a multiprocessor and multiple link platform with a given task and communication mapping, a two phase scheduling method is proposed. The static scheduling is based on the slack time list scheduling, and a critical path analysis and task execution order refinement method is used to find the off-line voltage scheduling for a set of periodic real-time tasks. The run-time scheduling is similar to resource reclaiming and slack stealing, which can make use of the variation from the WCET and provide the best-effort service to aperiodic tasks. In [36], a EDF based multiprocessor scheduling and assignment heuristic is given, which is shown better than the normal EDF. After the scheduling, an ILP model is used to find the voltage scaling accurately or approximately by simply rounding the result from a LP solver and the result is claimed within 97% accuracy. The method can be used for both continuous and discrete voltage.

Our scheduling methodology is different from the above ones in several ways. Firstly, we consider only discrete voltages, which is a more reasonable assumption in terms of future process technology and circuit design possibilities, compared to a continuous range. Actually, as illustrated in [36], it is even more difficult to solve because it corresponds to an ILP problem, not LP. Secondly, we also use a two-phase off-line and on-line scheduling. However, the design-time off-line step is more a design space exploration than a simple scheduling, because it gives a series of different scheduling trade-off points, rather than only one given by a conventional scheduler. Thirdly, we avoid having to use the WCET estimation, which is inaccurate and pessimistic due to the dynamic features of the applications. Fourthly, we apply voltage scaling at the intra-task level and make use of the run-time application information. The intra-task voltage scaling is considered only recently by other researchers[3].

4. TARGET PLATFORM ARCHITECTURE AND MODELS

In an up-to-date platform like Fig. 2 (where we focus on the digital core only), one or more (re-configurable) programmable components, either general-purpose or DSP processor cores or ASIPs, the analog front end, on-chip memory, I/O and other ASICs are all integrated into the same chip (System-on-Chip) or in the same package (System-in-Package). Furthermore, platforms typically contain also some software blocks (API, Operating System and some other middleware solutions), a methodology and a tool set to support rapid architectural exploration. Examples of such platforms are TI's OMAP[17], the Philips Nexperia[24], the ARM PrimeXsys[2] and the Xilinx Virtex-II Pro[32].

Platforms obtain a high performance at a low energy cost through the exploitation of different types of parallelism [18, 1]. In the context of this paper we will mainly focus on the task-level concurrency but also the instruction and data-level parallelism should be exploited in the mapping environment. In this way we can keep the power consumption lower by exploiting Vdd and frequency scaling in the processing cores. We will assume that only a limited set of Vdd's can be supported with a not too large range because future process technologies (see ITRS roadmap) will demand that. Large memories even do not allow any Vdd scaling but in that case the power consumption can be controlled by an efficient mapping on a distributed memory architecture [12].

A multiprocessor platform simulator was used to test the effectivity of our methodology and to answer "what-if" questions to explore the platform architecture design space. For the experiments in this paper, we will focus on the processing modules. In order to demonstrate the impact of our MATADOR-TCM approach, we will assume that either several StrongARM cores are available each with a different Vdd or a single StrongARM core with a few discrete Vdds. The ranges of the Vdd's will vary from 1.2 to 2.4V, which is motivated by the data sheet information. The power models are based on these data sheets too and are based on instruction counts obtained from profiling. The cycle counts of the thread nodes are also obtained by profiling, based on an ARMulator environment. A clock frequency of 266 MHz is assumed at 2.4V to obtain the execution times.

5. TASK CONCURRENCY MANAGEMENT

This approach addresses the dynamic and concurrent task scheduling problem on a multiprocessor platform for real-time embedded systems, where energy consumption is a major concern. Here we propose a two-phase scheduling method, which is a part of our overall design methodology and can provide the required flexibility at a low run-time overhead. The global methodology is briefly introduced in the next section, and then the scheduling method is explained. More information can be found in [25] and especially [33]. Finally we demonstrate how the platform simulator is used to support and verify this methodology.

5.1 Global TCM Methodology

The TCM methodology comprises of three stages (see Fig.3). The first is concurrency extraction. In this stage, we extract and explicitly model the potential parallelism and dynamic behavior of the application. An embedded system can be specified at a gray box abstraction level in a combined MTG-CDFG model [28, 30], where MTG is the acronym for multi-task graph. With MTG, the application can be represented as a set of concurrent thread frames [2]

[2]Please refer [33] for terminologies.

(TF) that exhibit a single thread of control. Each of these TFs consists of many thread nodes (TN) that can be looked at as a more or less independent section of the code. In the second stage, we apply concurrency improving transformations on the gray-box model. The third stage mainly consists of a two-phase scheduling approach. First the design-time scheduling step is applied to each of the identified TFs in the system. Different from traditional design-time scheduling, it does not generate a single solution but a set of possible solutions, each of which represents a different possible cost-performance trade-off point. Finally, we integrate an application-specific run-time scheduler in the RTOS of the application. The run-time scheduler dynamically selects one of these trade-off points for each running TF to find a global energy efficient solution.

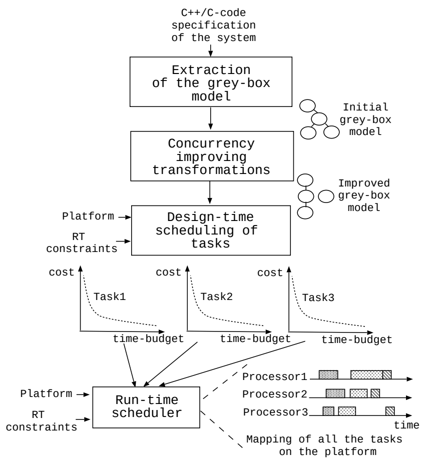
C++/C-code specification of the system
Extraction of the grey-box model
Initial grey-box model
Concurrency improving transformations
Improved grey-box model
Platform
RT constraints
Design-time scheduling of tasks
cost
Task1
Task2
Task3
time-budget
Platform
RT constraints
Run-time scheduler
Processor1
Processor2
Processor3
time
Mapping of all the tasks on the platform

Figure 3: Task Concurrency Management

5.2 Two-phase scheduling stage

The design of concurrent real-time embedded systems, and embedded software in particular, is a difficult problem, which is hard to perform 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 concurrent task scheduling on multiple processors.

The design-time scheduling is applied on the thread nodes inside each thread frame at compile time, including a processor assignment decision of the TNs in the case of multiple processing elements. On different types of processors on the heterogeneous platform, the same TN will be executed at different speeds and with different costs, i.e., energy consumption in this paper. These differences provide the possibility of exploring a cost-performance tradeoff at the system level. The idea of our two phase scheduling is illustrated in Fig. 4. Given a thread frame, our design-time scheduler will try to explore different assignment and ordering possibility, and generate a Pareto-optimal set, where every point is better than any other one in at least one way, i.e., either it consumes less energy or it executes faster. The Pareto-optimal set is usually represented by a continuous Pareto curve. Since the design-time scheduling is done at compile time, computation efforts can be paid as much as necessary,

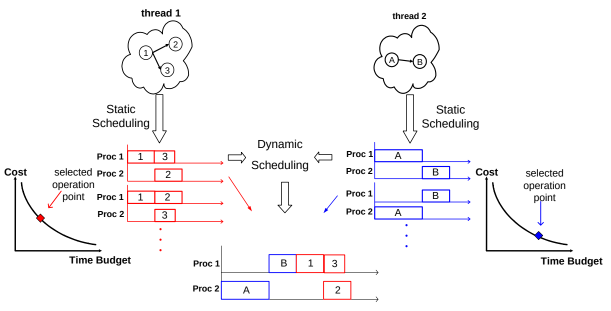

Figure 4: A two phase scheduling method.

provided that it can give a better scheduling result and reduce the computation efforts of run-time scheduling in the later stage.

At run time, the run-time scheduler will then work at the granularity of thread frames. Whenever new TFs are initiated, the run-time scheduler will try to schedule them to satisfy their time constraints and minimize the system energy consumption as well. The details inside a thread frame, like the execution time or data dependency of each thread node, can remain invisible to the run-time scheduler and this reduces its complexity significantly. Only some essential features of the points on the Pareto curve will be passed to the run-time scheduler by the design-time scheduling results, and be used to find a reasonable cycle budget distribution for all the running thread frames.

In summary, we separate the task scheduling into two separate phases, namely design-time and run-time scheduling, for three reasons. First, it lends more run time flexibility to the whole system. We can indeed accommodate more unforeseen demands for more execution time by any TF, by “stealing” time from other TFs, based on their available Pareto set. Secondly, we can minimize energy for a given timing constraint that usually spans several TFs by selecting the right combination of points. This will be illustrated in the realistic application of section7. Finally, it minimizes the run time computation complexity. The design-time scheduler works at the gray box level but still sees quite a lot of information from the global specification. The end result hides all the unnecessary details and the run-time scheduler only operates on the granularity of TFs, not single TNs.

5.3 Simulation environment

We have integrated this run-time manager on the platform with the help of an existing RTOS[31]. More in particular, the run-time manager uses the services of this RTOS to distribute the tasks/TFs in the application across the processors and to activate them in the correct order.

A simulation environment is used in this methodology to collect the necessary energy and performance profiling information for the design-time scheduling phase. The tasks/TFs are precharacterized with the execution times and their corresponding energy consumption for different configurations of the platform similar to [19] and [9]. In this context, the TF execution times and their energy consumption are measured on different single processor configurations. The interaction with other processors can be safely ignored in this case, which has been substantiated by measurements on the

ARMulator.

Secondly, the simulation environment is used to verify the functional correctness of the TCM approach. It also helps to quantify the effective energy reduction and performance gains which can be obtained with the TCM approach, including the overhead of the run-time scheduling phase.

We have applied our TCM approach on a part of a real-life application, the QoS kernel of the 3D rendering algorithm that is developed in the context of a MPEG21 project. We have simulated it on our platform. The application and experimental result are discussed in the following sections.

6. 3D RENDERING QOS APPLICATION

To test the effectiveness of our approach, a real-life application, the QoS (Quality of Service) control part of a 3D rendering algorithm, is used.

Fig. 5 shows how 3D decoding/rendering is typically performed: a 2D texture and a 3D mesh are first decoded and then mapped together to give the illusion of a scene with 3D objects. This kind of 3D rendering requires that each frame of the rendering process is recalculated completely. The required computation power depends significantly on its

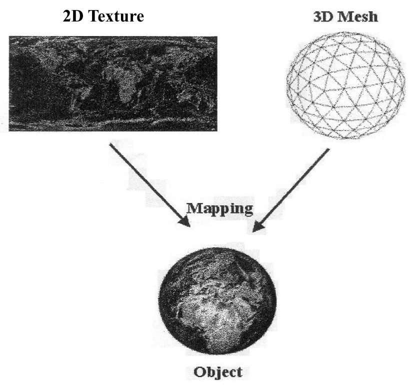

Figure 5: 3D rendering consists of 2D texture and 3D mesh decoding.

number of triangles. When the available resources are not enough to render the object, instead of letting the system break down (totally stop the decoding and rendering during a period of time), the corresponding mesh of the object can be gracefully degraded to decrease resouse consumption, while maintaining the maximal possible quality.

The number of triangles that are used to describe a mesh can be scaled up or down. This can be achieved by performing *edge collapses* and *vertex splits* respectively, as shown in Fig. 6.

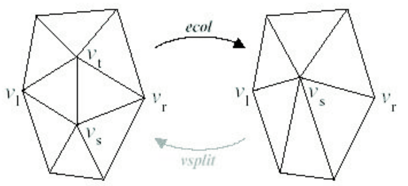

Figure 6: Edge collapse and vertex split.

To perform an edge collapse, and thus remove the edge (V_s, V_t), we first remove the triangles which V_s and V_t have in common, and we replace V_t with V_s in the triangles adjacent to V_t. We then recenter V_s, to keep the appearance of the new set of triangles as close as possible to the former one. The new set of triangles represents the same object with less detail but also with less triangles. The same principle but in a reversed way is used to perform a vertex split. The edge collapse and vertex split approaches can be used repeatedly till a desired number of triangles are achieved.

For a 3D object, the more triangles that are used to represent a mesh, the more precise the description of the object. This increases the perceived quality. However, it slows down the geometry and rasterizing stages because more computation power is needed there. Consequently it decreases the number of frames that can be generated each second(FPS, frame per second), while most videos or 3D games applications desire a fixed FPS. Another thing that we have to consider here is that the same application can be run at different platforms, e.g., a desktop PC or a PDA, which provides completely different computation ability and power consumption feature. Hence different qualities of the same service have to be supplied to achieve a similar FPS. For a given computation platform and a desired FPS, the number of triangles it can handle in one frame is almost fixed. Based on the number of objects in the current frame and what these objects are, the QoS controller will assign the triangles to each object so that the user can get the best-of-effort visual quality at a fixed frame rate.

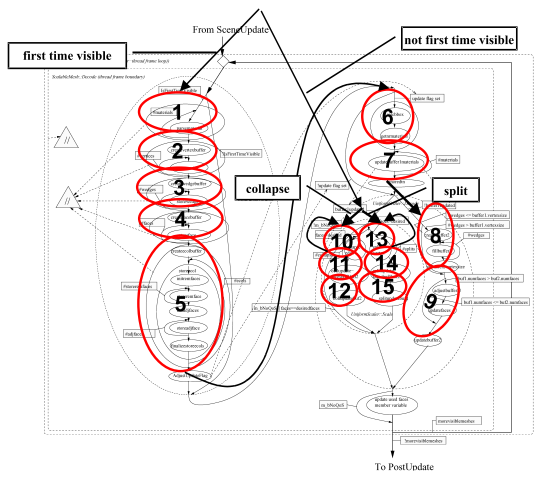

Figure 7: The gray-box model of quality adjust thread frame.

7. EXPERIMENT RESULTS

In the QoS kernel of the considered 3D application, for each visible object on the scene, a separate thread frame will be triggered, in which the number of triangles is adjusted to the number specified by the QoS algorithm. The gray-box model of that thread frame is shown in Fig. 7, where all the internal TNs are numbered as well. Table 1 gives the profiled execution time and energy consumption of each TN on a 2.4V StrongARM processor.

From the gray-box model, we can see that based on whether each object is the first time visible, which can be true only once during the whole stream, a branch will be taken. If

Thread Node	Ex. Time (us)	En. Cons. (mJ)
1	47	1.9667
2	1387.9	54.5519
3	3740.4	157.401
4	5726.4	306.2996
5	9305.3	497.5459
6	1655.2	96.0004
7	4.5	0.1682
8	778.5	36.6076
9	815.7	39.9994
10	82.4	4.1479
11	441.1	21.5862
12	0.2	0.0062
13	89	4.4328
14	718.8	35.3593
15	0.2	0.0056

Table 1: Execution time and energy consumption of TNs of the quality adjust TF.

it is the first time visible, the mesh and texture have to be parsed, generated and bounded (TNs 1 to 9). If it is not, the current number of faces will be compared to the desired number of faces to decide whether to collapse edges (TNs 10, 11 and 12) or to split more vertices (TNs 13, 14 and 15).

The edge collapse and vertex split are done in a progressive and iterative way to avoid abrupt changes of the object shape with a while loop over TN 10 or 13. The iteration number of this loop depends on the difference between the current and desired number of faces of that object, and it varies from 2 to 1000 based on the profiling data. Only one Pareto curve is not enough to represent these highly dynamic features. We have to distinguish first-time-visible or not-first-time-visible and the while loop iteration numbers. For the latter, if we would not make a difference in implementation of the while loop body, we would have to consider the implemetation for the worst case, which is 1000 iterations and much bigger than the average case. To avoid that, we have introduced the concept of "scenario selection" where different Pareto curves are assigned (in an analysis step at design time) to run-time cases that motivate a set of different implementations. Based on this analysis, we have decided to use 9 different Pareto curves in the QoS application to represent the run-time behavior of one object: the first one is when it is first time visible; the others are when it is not and has to be collapsed or split. For "collapse" and "split", each are assigned four curves with different implementations, corresponding to different iteration sub-ranges. For example, the first curve of "collapse" will be selected if the actual iteration number falls between 2 and 12. Therefore, we only have to consider the worst case of that sub-range, which is 12 in this example, not the worst case of the whole range, which is 1000. Extra code has been inserted to enable this. We have selected these ranges based on the profiling data from the application and they are illustrated in Table 2.

	first time visible	collapse or split	range
scenario 0	yes		
scenario 1	no	collapse	2-12
scenario 2	no	collapse	13-30
scenario 3	no	collapse	31-180
scenario 4	no	collapse	181-1000
scenario 5	no	split	2-4
scenario 6	no	split	5-12
scenario 7	no	split	13-60
scenario 8	no	split	61-1000

Table 2: Scenario selection.

In the QoS kernel, which is typical for future object-based multi-media applications, we know at the beginning of each frame the characteristics of its content. In this case we know how many objects we have to render and also (in our approach) the best matched scenario of each object. Each scenario is represented by a Pareto curve computed at design time. From these, the run-time scheduler uses a heuristic algorithm to select an operating point from each curve and activates it with the help of the RTOS. We have run the application for 1000 scenes and collected the data as a representative experiment. For comparison reasons, we have also generated a reference case, REF2, to show how well a state-of-the-art DVS scheduler can do. We assume it has full access to the available application parameters at run-time too, but it does not exploit the scenario selection concept so only one common implementation (schedule + assignment on the multi-processor) is available for any execution of the while loop body. For REF2, we assume it knows the number of objects it is going to schedule in that frame, but it does not exploit the scenarios. Therefore, it has to take an implementation that matches the worst case (TN 13 loops for 1000 times and the execution time is 89ms) for each object. However when one object finishes, the slack time (the difference between the real execution time and the worst case) will be reclaimed and reused by the scheduler for the subsequent tasks (i.e. slack stealing [29] is exploited). Whenever the estimated remaining execution time is smaller than the desired deadline, a continuous DVS method is used to save energy. Another reference case, REF1, is also generated, where all code is executed on the highest voltage processor. This is also the outcome if no information of the application is passed to the run-time manager, i.e. neither the number nor the kinds of the objects are known. Since we have a very dynamic application (the number of objects varies from 2 to 20), to handle the worst case and still meet the stringent deadline, the code has to be run completely on the high voltage processor. The majority of earlier techniques (see e.g. [18] for an overview) that use pre-characterized task data in terms of WCET times and corresponding energy, would lead also to the REF1 result for this type of applications with a very dynamic behaviour. Only a few would come close to REF2, as currently none of them combines all of the ingredients that were used to compose REF2.

	REF1 (1 CPU, 1 V_{dd})	REF2 (1 CPU, 2 V_{dd})	TCM (1 CPU, 2 V_{dd})	TCM (4 CPU, 4 V_{dd})
fps=5	627	577	257	162
fps=10	627	621	364	183

Table 3: Energy consumption

	REF1 (1 CPU, 1 V_{dd})	REF2 (1 CPU, 2 V_{dd})	TCM (1 CPU, 2 V_{dd})	TCM (4 CPU, 4 V_{dd})
fps=5	5	5	5	0
fps=10	26	26	26	1

Table 4: Deadline miss

The energy consumptions for all cases are shown in Tab. 3 and the number of deadline misses are shown in Tab. 4. All results are collected after the application has been run for 1000 sequential scenes and for different frame per second (FPS) requirements. The voltages we have used here are V_{dd}=1.2 and 2.4V for the two voltage case and V_{dd}=1.2, 1.6, 2.0 and 2.4V for the four processor case. Normally, with a higher FPS, to satisfy a more stringent time constraint, parts of the code that are executed at lower voltages have to be moved to a higher voltage, resulting in the increase of the energy consumption. Also, the chance that a deadline is

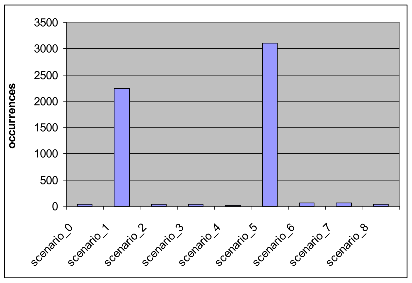

Figure 8: The distribution of the scenarios activation

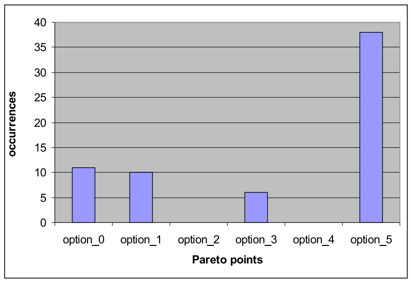

Figure 9: The distribution of the Pareto points activation in scenario 6

missed increases correspondently. Compared to REF1, for the single processor situation, the TCM approach consumes much less energy (saving of 58% when fps is 5 and 40% when fps is 10), while the deadline miss ratio remains the same. The latter is easy to understand because when the time constraint is really stringent, the TCM method will automatically schedule all thread nodes to the highest possible voltage processor, which is just what REF1 does. When the time constraint is less tight, a much cheaper solution will be found by the TCM method. Comparing REF2 and REF1 you will find that for this type of dynamic multi-media applications, state-of-the-art DVS cannot gain much because it does not exploit different combinations of TF realisations. This is especially so for a heterogeneous multi-processor context where all the prestored implementations of TN schedules and processor assignments cannot be computed at run-time any more without too much run-time overhead. Only the Vdd selection can be performed at run-time in some approaches but then not per processor. From the result we can also see that by increasing the number of processors, we can reduce the energy consumption even more while now meeting nearly all the deadlines.

The distribution of the scenario selection is given in Fig. 8, while Fig. 9 gives the distribution of the selected Pareto points in scenario 6, both when fps is 5. From the figures we can see that the TCM scheduler activates different scenarios dynamically and selects the optimal Pareto point from the activated scenario depending on the run-time situations, i.e. the resource available and the number of competitors. Most of the time, scenario 1 and 5, which are the least time consuming ones, will be selected. Therefore, we avoid the worst case estimation and have more opportunities to scale down the voltage, compared to REF2. In scenario 6, the least energy consuming solution, Pareto point 5, is selected as long as it is possible; otherwise a more expensive one is chosen to meet the time constraint. The combination of scenario and Pareto point selection gives us the advantage of heavily exploring the design space at design time and finding the most energy efficient solution exactly for that situation at run time.

8. CONCLUSION

In this paper we presented a unique approach to manage concurrent tasks of dynamic real-time applications. We explained an methodology to map the applications in a cost (especially power) efficient way onto a heterogeneous embedded multiprocessor platform. This approach is based on a design time exploration, which result in a set of schedules and assignments for each task, represented by Pareto curves. At run time, a low complexity scheduler selects an optimal combination of working points, exploiting the dynamic and non-deterministic behavior of the system.

This approach leads to significant power saving compared to state of the art voltage scaling techniques because of two major contributions. First, we effectively combine an intra-task detailed design-time exploration, giving high scheduling quality, and a low overhead run-time scheduler. Second, by considering the run-time application provided information, we use a scenario approach to avoid the worst case execution time estimation. In the future, we will extend this work to provide tool support for code synthesis, concurrency transformation and RTOS integration.

9. REFERENCES

[1] S. Adve et al. The Interaction of Architecture and Compilation Technology for High-performance Processor Design. *IEEE Computer Magazine*, 30(12):51–58, Dec. 1997.

[2] ARM. www.arm.com.

[3] A. Azevedo et al. Profile-based dynamic voltage scheduling using program checkpoints. In *Proceedings of the Design Automation and Test in Europe*, pages 168–75, 2002.

[4] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen. A Dynamic Voltage Scaled Microprocessor System. *IEEE J. Solid-State Circuits*, 35(11):1571–80, Nov. 2000.

[5] 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*, pages 347–52, 1996.

[6] J.-M. Chang and M. Pedram. Codex-dp: Co-design of Communicating Systems Using Dynamic programming. In *Proceedings of the Design Automation and Test in Europe*, pages 568–73, Mar. 1999.

[7] T. Claasen. High Speed: Not the Only Way to Exploit the Intrinsic Computational Power of Silicon. In *Proc. Int. Solid-State Circuits Conf.*, pages 22–25, San Fransisco, CA, Feb. 1999.

[8] http://www.crusoe.com.

[9] B. P. Dave, G. Lakshminarayana, and N. K. Jha. COSYN: Hardware-Software Co-Synthesis of heterogeneous Distributed Embedded Systems. *IEEE Transactions on Very Large Scale Integration(VLSI) Systems*, 7(1):92–104, Mar. 1999.

[10] H. De Man. System Design Challenges in the Post-pc Era. In *Proceedings of the 37th Design Automation Conference*, Los Angels, 2000.

[11] H. El-Rewini, H. H. Ali, and T. Lewis. Task Scheduling in Multiprocessing Systems. *IEEE Computer*, 28(12):27–37, Dec. 1995.

[12] F.Catthoor, K.Danckaert, C.Kulkarni, E.Brockmeyer, P.G.Kjeldsberg, T. Achteren, and T.Omnes. *Data Access and Storage Management for Embedded Programmable Processors*. Kluwer Academic Publishers, Boston, 2002.

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

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

[15] 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.

[16] 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*, pages 653–656, San Jose, CA, 1998.

[17] T. Instruments. www.ti.com.

[18] N. K. Jha. Low Power System Scheduling and Synthesis. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 259–63, 2001.

[19] D. Kirkovski and M. Potkonjak. System-level Synthesis of low-power Hard Real-time Systems. In *Proceedings of the 34th Design Automation Conference*, pages 697–702, Jun. 1997.

[20] J. Luo and N. Jha. Power-consious Joint Scheduling of Periodic Task Graphs and Aperiodic Tasks in Distributed Real-time Embedded Systems. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 357–364, San Jose, USA, Nov. 2000.

[21] J. Luo and N. Jha. Static and Dynamic Variable Voltage Scheduling Algorithms for Real-Time Heterogeneous Distributed Embedded Systems. In *7th ASPDAC and 15th Int'l Conf. on VLSI Design*, pages 719–26, Jan. 2002.

[22] P. Marchal, C. Wong, A. Prayati, N. Cossement, F. Catthoor, R. Lauwereins, D. Verkest, and H. De Man. Dynamic Memory Oriented Transformations in the MPEG4 IM1-player on a Low Power Platform. In *Proc. Intnl. Wsh. on Power Aware Computing Systems(PACS)*, Cambridge MA, Nov. 2000.

[23] 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.

[24] Philips. www.semiconductors.philips.com/platforms/nexperia.

[25] A. Prayati, C. Wong, P. Marchal, et al. Task Concurrency Management Experiment for Power-efficient Speed-up of Embedded MPEG4 IM1 Player. In *International Conference on Parallel Processing*, 2000.

[26] 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.

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

[28] S.Himpe, G.Deconinck, F.Catthoor, and J.Meerbergen. MTG* and Grey-Box: modeling dynamic multimedia applications with concurrency and non-determinism. In *Proc. Forum on Design Languages(FDL)*, Marseille, France, Sept. 2002.

[29] Y. Shin, K. Choi, and T. Sakurai. Power Optimization of Real-Time Embedded Systems on Variable Speed Processors. In *Proceedings of the 37th Design Automation Conference*, pages 365–8, 1999.

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

[31] Windriver. www.windriver.com.

[32] Xilinx. www.xilinx.com.

[33] 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.

[34] T.-Y. Yen and W. Wolf. Communication Synthesis for Distributed Embedded Systems. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 288–94, 1995.

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

[36] Y. Zhang, X. S. Hu, and D. Z. Chen. Task Scheduling and Voltage Selection for Energy Minimization. In *Proceedings of the 39th Design Automation Conference*, 2002.