Scenario Based Mapping of Dynamic Applications on MPSoC: A 3D Graphics Case Study

Narasinga Rao Miniskar[1,3], Elena Hammari[2], Satyakiran Munaga[1,3], Stylianos Mamagkakis[1], Per Gunnar Kjeldsberg[2], and Francky Catthoor[1,3]

[1] IMEC,Kapeldreef 75, Leuven 3001,
miniskar, satyaki, mamagka, catthoor@imec.be
[2] NTNU, 7491 Trondheim, Norway
elena.hammari, per.gunnar.kjeldsberg @iet.ntnu.no
[3] K.U.Leuven, ESAT Dept., Leuven 3001

Abstract. Modern multimedia applications are becoming increasingly dynamic. The state-of-the-art scalable 3D graphics algorithms are able to adapt at run-time their hardware resource allocation requests according to input, resource availability and a number of quality metrics. Additionally, the resource management mechanisms are becoming more dynamic themselves and are able to cope efficiently at run-time with these varying resource requests, available hardware resources and competing requests from other applications. In this paper, we study the dynamic resource requests of the Wavelet Subdivision Surfaces (WSS) based scalable 3D graphics application. We also show how to schedule its computational resources at run-time with the use of the Task Concurrency Management (TCM) methodology and the System Scenario based approach on MPSoC platform with very heterogeneous Processing Elements (including RISC, VLIW and FPGA accelerator resources).

1 Introduction

It is common for embedded hardware platforms, nowadays, to feature multiple Processing Elements (PEs) and thus to be able to execute highly complex multimedia algorithms with big computational resource requirements. Scalable 3D graphics algorithms can demonstrate multimedia content that is created once and then scaled each time to match the characteristics of the embedded system, where it is deployed (e.g., according to different consumer device displays) [1]. Therefore, such software applications have very dynamic computational resource requests, because the timing and size of each request is not known at design-time and is only determined at run-time based on the user actions and dynamic scalability response of the algorithm itself according to the quality targeted [14].

Additionally, it is very likely that other software applications will be executing concurrently on the embedded platform, sharing the available resources dynamically, as they are loaded and unloaded at run-time (e.g., you receive an

email as you play a 3D game on your mobile phone). Therefore, scheduling the tasks of these software applications is very challenging, because it is not known at design-time: (i) the computational resource requests of each task, (ii) the number of tasks executed by each software application and (iii) the combination of software applications that will be executed concurrently. The solution given today to this problem is calculating the worst case resource request of any combination of tasks and software applications and allocating resources accordingly at design-time. This solution requires a huge amount of computational resources and is very energy inefficient as it can not adapt the scheduling according to the actual run-time software application real-time demands. Therefore the scheduler not only pre-allocates but also uses the maximum amount of processing resources.

In this paper, we characterize the run-time computational resource needs of a very demanding and dynamic scalable 3D graphics application [14], which executes on a heterogenous Multiple Processor System on Chip (MPSoC) concurrently with other applications. For this case study, we implement the Task Concurrency Management (TCM) scheduling methodology [9] to schedule its tasks both at design-time and run-time and also implement the System Scenario approach [6] to avoid using one worst-case resource requirement solution. It is the first time a hardware accelerator is included in an MPSoC platform used for run-time scheduling algorithms like TCM. The computation demanding software of Wavelet Subdivision Surfaces (WSS) for 3D scalable graphics applications as well as three more Task Graphs generated from the TGFF[3] will be scheduled on the aforementioned platform. The structure of this paper is as follows. The next section describes the related work. In section 3, we describe the Task Concurrency Management (TCM) methodology and the System Scenario approach. In section 4, we describe the Wavelet Subdivision Surfaces (WSS) case study, implement the methodology and show some experimental results. Finally, in section 5, we draw our conclusions.

2 Related work

In the context of scenarios, scenario-based design[2] has been used for some time in both hardware[11] and software design[4] of embedded systems. They use case diagrams[5] which enumerate, from a functional and timing point of view, all possible user actions and the system reactions that are required to meet a proposed system function. These scenarios are called use-case scenarios and do not concentrate on the resources required by a system to meet its constraints. In this paper, we concentrate on a different and complementary type of scenarios, which we call system scenarios [6]. These are derived from the combination of the behavior of the application and its mapping on the system platform. These scenarios are used to reduce the system cost by exploiting information about what can happen at run-time to make better design decisions at design-time, and to exploit the time-varying behavior at run-time. While use-case scenarios classify the application's behavior based on the different ways it can be used, system scenarios classify the behavior based on the multi-dimensional cost trade-

off during the implementation trajectory. In the context of this paper, whenever scenarios are mentioned they imply System Scenarios.

In the context of task scheduling, a good overview of early scheduling algorithms can be found in [12]. This paper uses the terminology task scheduling 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). In the context of this paper, we have selected the Task Concurrency Management methodology [9] to implement a combination of design-time and run-time scheduling, which can balance performance versus energy consumption trade-offs.

3 Task Concurrency Management and System Scenarios Overview

As can be seen in Fig. 1, we use the Task Concurrency Management (TCM) methodology and System Scenarios approach to do the task scheduling of WSS and any other applications that might be executing concurrently on the MPSoC platform. The two phases shown on the left are carried out at design-time and the two phases on the right side are carried out at the run-time. All four phases are part of the System Scenario approach [6] and are instantiated for the TCM scheduling [9].

- At the **Scenario identification** phase, we define a number of scenarios for each application executed on the selected MPSoC platform. These scenarios are defined based on typical application inputs and their impact on the control flow and data flow of the software application. For each one of these scenarios, we evaluate the execution time and energy consumption of each task if it would be mapped on any of the Processing Elements (PEs) of the selected MPSoC platform. We extract these energy and timing values at design-time via simulation and profiling and insert them in addition to the application's task-graph in the Grey box model for TCM scheduling [9]
- At the **Scenario exploitation** phase, we produce at design-time a set of Pareto-optimal schedules using the Grey Box model in the TCM methodology. Each one of these schedules is a specific ordering of one scenario's tasks and their assignment on specific PEs. Each schedule is represented by a Pareto point on an energy vs performance Pareto curve and each scenario is represented by the Pareto curve itself.
- At the **Scenario detection** phase, we monitor and detect at run-time the pre-calculated scenarios. Each time that a scenario is detected, the Scenario switching mechanism is initiated. Additionally, the energy and real-time constraints are monitored in order to select the optimal performance vs energy consumption trade-off.

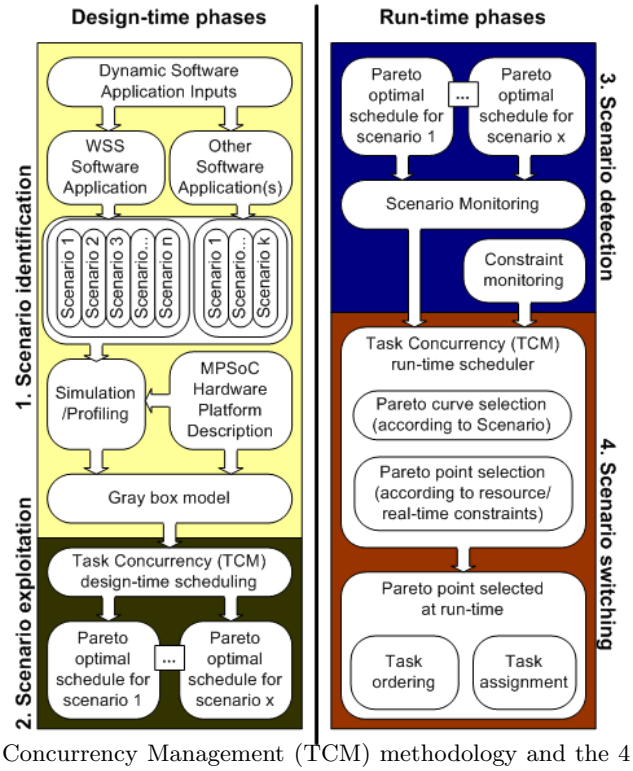

Fig. 1. Task Concurrency Management (TCM) methodology and the 4 phases of the System Scenarios approach.

– At the **Scenario switching** phase, when a scenario is detected, the TCM run-time scheduler selects and implements one of the pre-calculated Pareto-optimal schedules, thus switching from one Pareto curve to another. If the TCM run-time scheduler evaluates that a constraint (e.g., real time) will not be met, then it switches from one Pareto point to another (within the same Pareto curve) in order to meet this constraint in the expense of another resource (e.g., energy). The final result at every moment at run-time is the execution of one Pareto-optimal, pre-selected schedule.

4 WSS Characterization and task scheduling

4.1 Software Application Description

3D content made available on the Internet, such as X3D/VRML and MPEG-4 content, is transported over and consumed on a lot of different networks (wireless and wired) and terminals (mobile phones, PDA's, PCs, ...). A WSS based scalable 3D graphics framework is proposed in [14], where the object's quality dynamically adapts to the viewing conditions while respecting constraints such as platform resources. Based on the user-commands (Ex: Move forward, Move backward, rotate, pitch, yaw, etc...), the best triangle budget and the related

Level Of Detail (LOD) settings for each visible object will be decided (online) at run-time. Once the triangles are obtained for each visible object, they will passed to Renderer. The abruptness in the no.of visible objects based on the user-commands, the object-level LOD settings will be varied. In the implemented framework, the over-all scene level quality is fixed.

The software modules responsible for the processing of each 3D scene content are given below.

- Initialization Module
  - Process Scene Description: Reads the 3D scene information and all 3D objects base meshes, texture, pre-processed quality error details, and scenario based visibility and area data, and holds it in its internal data structure, shown as Mesh Database(DB). This is a one-time initialization process of 3D content.
  - WSS-Decoder: The WSS based decoding[14] is very computation heavy task ($\sim$1.8secs on TI-C64X+ and $\sim$33secs on StrongARM processors), thus this task is mapped to a hardware accelerator and all the objects are decoded to the maximum LOD in the initialization phase itself. The triangles for all LOD settings are held in storage memory.
- Frame based invocation modules: These modules will be called for each frame when the user presses a command like rotate, yaw, pitch etc....
  - Check Visibility: This modules checks the visibility of each object based on the bounding box concept explained in [14] and outputs the visibility information in the Mesh DB.
  - Build Global Pareto curve: This module generates the global Pareto-curve(trade-off curve) based on the error contributed by each visible object and its corresponding triangle budget. It uses the gradient descent algorithm to build this[14].After building the global Pareto plot, this module calls the decoder based on the changes in quality parameters.
  - Converter(Prepare for Renderer): Finally this module converts the triangles into vertices on the 3d scene and later this will be passed to renderer [14].

For each frame, the application calls the above mentioned modules as shown in the sequence above. In this application there is an object level parallelism, where we can potentially handle the objects in parallel which can enable the multiple parallelization versions of this application. However in the context of this paper, we still would like to show the benefits of TCM scheduling even without exploring parallelism (i.e., this is listed as future work).

4.2 Target Platform

The generic MPSoC platform that we decide to use in this paper is shown in Fig. 2. More specifically, we have considered a heterogeneous platform with two RISC processors (Strong ARM 1100x), two VLIWs with six FUs each (TI-C64X+) and one RISC processor(Strong ARM 1100x) with an FPGA Hardware accelerator

(Virtex-5 XC5VSX95t speed grade-3) as a co-processor. This is for handling control functions, and data and instruction level parallelism respectively. The StrongARM 1100x processors run at 1.48V, 0.1632A, with clock frequency of 133MHz and the TI-C64X+ processors run at 1.2V with clock frequency of 500MHz and the FPGA runs at 100MHz and 1.0V. We assume a cache memory (C) for each processor and a single shared global memory (SM). The Hardware accelerator is implemented with a Virtex-5 SXT FPGA from Xilinx for the purpose of accelerating the GlobalParetoBuildTime software module of the WSS task. The communication topology considered is a cross-bar switch. Any potential implications of communication or memory latency bottlenecks are not calculated in the context of this paper and are subject to future study.

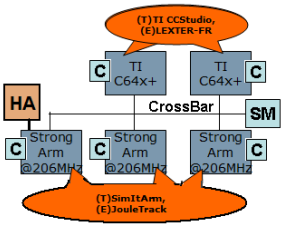

Fig. 2. Target platform.

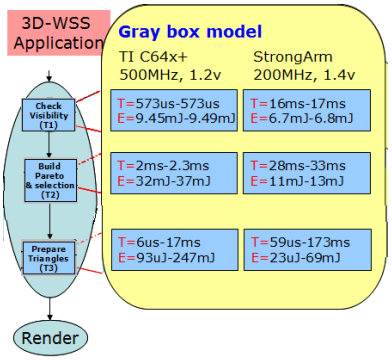

Fig. 3. Gray-box-model of WSS

In order to apply the TCM Methodology we require Energy and Timing information of each task (see Thread Frames and Thread Nodes in [10, 9]) for the Gray-box model. For these experiments we have used profiling information from the StrongARM and TI-C64X+ processors simulators (see SimItARM Simulator and TI CCStudio v3.3 evolution, respectively). For getting energy information for tasks running on StrongARM processor we have used JouleTrack [13]. For the TI-C64X+ energy profiling, we have used the functional level power analysis model of TI-C6X[8], modified for the functional units, instruction set, and memory hierarchy of the TI-C64X+.

Parts of the heterogeneous platform has been used for experiments in earlier papers[9]. It is however, the first time an FPGA-based hardware accelerator is included. We have explored performance and energy consumption of parts of the GlobalParetoBuild task, when implemented on a Virtex-5 SXT FPGA from Xilinx. Selected parts for the hardware acceleration are Div, Div+Sqrt, Div+Mul and Div+Mul+Sub operations. These parts were translated into a high-level Simulink model, and implemented in the FPGA using the newly introduced Simulink-to-FPGA design flow based on an automatic code generation tool Xilinx System Generator [15] and the widely used Xilinx ISE development

environment. System Generator includes a built-in library of generic IP blocks: adders, multipliers, registers, memories, filters, processors, etc. ready for use in Simulink. In our design, we directly mapped the selected operations to the IP blocks performing the same operations. This resulted in a very simple model with straight forward interconnection between the blocks. The conversion blocks were inserted to equalize input formats to each block and to satisfy output accuracy requirements. The blocks were configured with the appropriate data formats and the pipelined implementation of each block was chosen to maximize the operating frequency of the hardware accelerator.

4.3 Inputs to 3D-WSS and Scenarios

In the experiments below, we consider WSS in a scalable 3D graphics engine for 3D games. In the game example there is a input representing 3 rooms (R1, R2 and R3), where in each room there will be some objects (e.g., some fruits on a bowl). There will be 4 camera positions (C1, C2, C3 and C4) in each room to see the objects from different angles. As an in-game character enters from one room after another, the best camera position can be chosen by the application controller according to the in-game cinematic effects. In each room of the game we have applied sequence of User commands that define the in-game character movement and perspective. There are 13 objects in Room 1 (R1), 17 objects in Room 2 (R2) and 22 objects in Room 3 (R3). We have applied the Systematic Scenario identification methodology proposed in [6] based on the in-depth analysis of the application. We have identified few parameters that are causing variation in the software modules of this application (room no., and selected camera position by the application). Based on the room no. in which the in-game is present and the camera position, we can derive 12 scenarios(i.e., 3 rooms times, 4 camera-positions). For each scenario we will get one Gray-box-model for the TCM scheduling. For each scenario, we have considered the worst case estimation on the no.of objects, no.of pareto points in all objects, and no.of triangles, and computed the Worst case estimation time(WCET) for all three software modules of this application. There is still a dynamism in execution time in each scenario, because of the varying no.of visible objects(Ex.: 8 to 13 in Room 1), varying no.of pareto points(Ex.: 200 to 350 in Room 1), and varying no.of triangles(Ex.: 47000 to 50000 in Room 1), in each room. However we still don't have a methodology to deal the dynamism due to these kind of variables (Data variables), which has large range of values for each variable in the program execution. In this paper we are focusing only on the parameters which can be handled through Systematic Scenario Identification methodology. As we have computed the WCET using profiling, we couldn't give any hard-guarantees based on this execution-times.

4.4 Gray Box Modelling

The task graph of the WSS application is shown in Fig. 3. The Gray-box-model for the 3D-WSS is constructed based on this task graph and the profiling in-

formation (Energy, Execution Time) captured. For this we need to consider the WCET for each scenario and for each module of WSS. The WCETs of 3 modules for StrongARM are shown in Table 1. However, not all scenarios are required due to similarity in their WCET. We have clustered the WCETs according to Systematic clustering approach [6, 7] and they are represented in colors for each module in Table 1. For the GlobalParetoBuild module the clusters are (C1,C2,C4) and (C3) for each room, and hence total 6 scenarios instead of 12. Like this, the end scenarios derived after clustering for CheckVisibility module are 3 for all rooms, and for PrepareRender module are 8 for all rooms. We have repeated the clustering for TI processor WCET too. However, we have obtained the same clusters as for StrongARM processor. Each cluster is treated as one Scenario for the module. The Scenarios obtained at this point are at sub-task level. However, we need the scenarios at the complete task-graph level, using which we need to optimize pareto optimal mappings at run-time. Based on the clusters we have obtained from sub-tasks, we have selected the sub-task clusters which has maximum clusters among all. We have checked whether all other clusters fall under this, if not we can further cluster the set. In the case of WSS, the maximum clusters obtained from PrepareRender module are sufficient for the whole task-graph level too. At the end, we have obtained total 8 scenarios for the task-graph.

Ri, Ci	**CheckVisibility**	**GlobalParetoBuild**	**PrepareRender**
R3-C1	45.036	106.91	386.08
R3-C2	44.772	101.97	426.97
R3-C3	45.587	35.63	174.44
R3-C4	45.656	102.76	389.35
R2-C1	34.832	79.43	362.65
R2-C2	34.626	73.08	374.22
R2-C3	35.198	30.71	154.33
R2-C4	35.266	73.57	359.25
R1-C1	26.624	52.11	268.52
R1-C2	26.509	50.23	274.12
R1-C3	26.892	18.47	99.51
R1-C4	26.905	45.36	210.06

Table 1. Scenario clustering of WCETs(m.secs) for StrongARM

After the Scenario identification phase, the Gray-box model is constructed based on the obtained WCET profiling information for the 8 scenarios. An example of the Gray-box model generated per scenario is shown in Fig. 3. The 3 Tasks seen in the Gray-box model are CheckVisibility(T1), Build-Pareto & Selection(T2), and Prepare Triangles(T3).

4.5 TCM Design Time Scheduling

At the Scenario exploitation phase, the TCM Design-time scheduling produces 8 Pareto curves (i.e., one per scenario). In Fig. 4, we show as an example 4

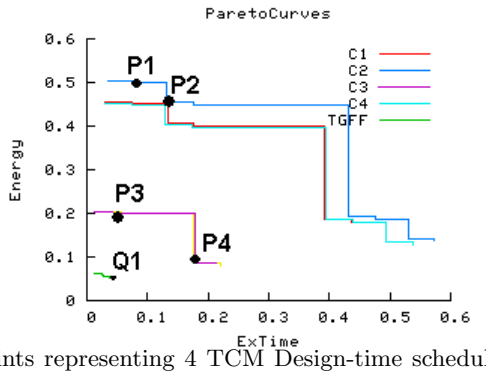

Fig. 4. 4 Pareto points representing 4 TCM Design-time schedule examples (P1-P4) of the WSS application. The 4 Pareto curves represent 4 of the scenarios (C1-C4 in Room 3). Q1 represents the schedule of an another application competing for resources (generated automatically with TGFF).

of these Pareto curves. Each Pareto curve has a number of Pareto points which represent a pareto optimal ordering and assignment of the WSS tasks to the PEs. In Fig. 4, we show 4 of these schedules (P1-P4) belonging to two Pareto curves (i.e., scenarios). Additionally, we have applied the same for TGFF [3] generated taskgraphs to demonstrate the schedule that would be produced for a random application sharing the MPSoC resources. The TGFF application schedule is represented by Q1. Each pareto curve is occupying $\tilde{1}$.3KBytes of memory and there are approximately 8 to 15 points in each task each scenario pareto curve.

4.6 TCM Run Time Scheduling

At the Scenario switching phase, the TCM Run-time scheduler implements a Pareto point selection algorithm [16], to select at run-time the optimal schedule for each application, while meeting the real time constraints and minimizing the energy consumption. The deadline for the WSS application is set initially at 0.5 secs (i.e., the frame will be updated by input user commands coming at a rate of a 0.5 secs period). The TGFF application also has the same frame rate. If the deadline is changed at 0.115 secs (e.g., because user commands come faster) and both applications are sharing resources, then the TCM runt-time scheduler switches to a faster schedule point P1. However if the deadline is increased to 0.22 secs, then the TCM Run-time scheduler chooses schedule point P2 for WSS saving 10% less energy, as in StateB, without missing any real-time constraints.

At the Scenario detection phase, if the Camera position switches from C2 to C3 then the TCM Run Time Scheduler is activated again and selects a schedule from the corresponding Pareto curve. In this particular case it would move from schedule P2 to schedule P4, thus saving an additional 55% of energy consumption, without missing any real-time constraints.

From the heterogeneity experiments, we have identified that there is 10% performance gain and energy gain just with a small Hardware accelerator for 2 kind of instructions. This will be more, if we explore it for the whole Global-ParetoBuild module. The slacked gained from the Hardware accelerator is used

by the TCM run-time scheduler and provided the 16% energy efficient solution at the end for the WSS application.

5 Conclusions and Future work

In this paper, we have characterized the computational and energy resource requests of the Wavelet Subdivision Surfaces (WSS) algorithm for scalable 3D graphics on a MPSoC platform. Moreover, this paper demonstrates the implementation of a number of experimental design-time and run-time techniques at various abstraction design levels and shows how they can be combined for a single case study. More specifically, we have demonstrated WSS at the application level, System Scenarios at the system level and Task Concurrency Management at the middleware (resource management) level. In our future work, we plan to parallelize certain tasks of the WSS algorithm, do a more thorough exploration of the available WSS scenarios and MPSoC platform options and calculate the switching overhead between the TCM schedules at run-time. We have also explored the TCM methodology for the heterogeneous platform which has Hardware Accelerator and shown more gains from the TCM methodology.

References

1. Iso/iec 14496 information technology-coding of audio-visual objects - part 10: Advanced video coding. 2004.
2. . J. M. Carroll, editor. *Scenario-based design: envisioning work and technology in system development*. 1995.
3. R. P. Dick. Tgff: task graphs for free. In *CODES/CASHE '98*, 1998.
4. B. P. Douglass. *Real Time UML: Advances in the UML for Real-Time Systems (3rd Edition)*. 2004.
5. M. Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 2003.
6. S. V. Gheorghita. System scenario based design of dynamic embedded systems. In *ACM*, 2007.
7. J. Hamers. Resource prediction for media stream decoding. In *DATE*, 2007.
8. J. Laurent. Functional level power analysis: An efficient approach for modeling the power consumption of complex processors. In *DATE '04*.
9. Z. Ma and F. Catthoor. *Systematic methodology for real-time cost effective mapping of dynamic concurrent task-based systems on heterogeneous platforms*. Springer, 2007.
10. Z. Ma, D. P. Scarpazza, and F. Catthoor. Run-time task overlapping on multiprocessor platforms. In *ESTImedia*, pages 47–52, 2007.
11. J. M. Paul. Scenario-oriented design for single-chip heterogeneous multiprocessors. *IEEE Trans. VLSI Syst.'06*.
12. K. Ramamritham. Issues in the static allocation and scheduling of complex periodic tasks. In *RTOSS '93*, 1993.
13. A. Sinha and A. Chandrakasan. Jouletrack - a web based tool for software energy profiling. In *DAC*, 2001.
14. N. Tack. Platform independent performance metrics for 3d terminal quality of service - phd thesis. 2006.
15. Xilinx Inc. *System Generator for DSP User Guide*, March 2008. Version 10.1.
16. P. Yang and F. Catthoor. Pareto-optimization-based run-time task scheduling for embedded systems. In *CODES+ISSS '03*. ACM.