Volume 2, Issue 10, October 2012 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

New Era for Design and Run-Time Management and Mapping of Dynamic Embedded Software in MPSOC (Multi-Processors System on Chip)

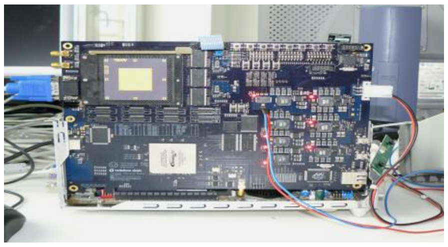
Vyas Dhaval Shashikant
Assistant Professor,
C.U.Shah College of MCA.

Abstract— **Recent technological trends have led to the introduction of Multi-Processor Systems-on-Chips (MPSoCs).Driven by the power efficiency, heterogeneous architectures incorporating functional units optimized for specific functions are commonly employed. This trend has dramatic consequences on the design technology. Techniques are required, which support run-time management of sophisticated application software on the MPSoCs**

Keywords—: API, OMAP, MPSoC

I. INTRODUCTION

The Tomahawk MPSoC exploits instruction, data and task level parallelism in order to meet stringent performance requirements with low energy consumption. Figure 3 shows a schematic of the Tomahawk. Below, we briefly discuss the components building this architecture. Two Tensilica DC212GP RISC processors build the platform for the operating system and control code execution. The signal processing block of the Tomahawk is composed of six 4-fold SIMD fixed-point vector DSPs (VDSP), two scalar floating-point DSPs (SDSP), a low density parity check code (LDPC) decoder ASIP, a deblocking filter ASIP and an entropy decoder ASIC. The Core Manager performs the scheduling of signal processing tasks issued from the control code onto the VDSPs and SDSPs. The VDSPs are meant for processing vectorized parallel algorithms such as FFTs or DCTs. Each VDSP is able to provide 20 MOPS/MHz, resulting in an accumulated processing power of 21 GOPS at 175 MHz operation frequency. Algorithms with a high dynamic range such as matrix inversions for MIMO processing can be processed by the SDSP. Furthermore, the SDSP is intended to be used for bit stream processing. For that purpose, all processor instructions can be executed conditionally. Both SDSPs jointly contribute 0.7 GOPS to the total processing power of the Tomahawk. The LDPC decoder ASIP [3] is able to decode variable block length at throughput of several hundred MBit/s. For instance, a gross data rate of 1.1 GBit/s for a (3; 6) code is achieved by our implementation. The 64 way parallel, 8 bit ¯xed point SIMD-VLIW decoder architecture delivers a performance of 12 GOPS. The peripheral part of the Tomahawk consist of the following components: an FPGA bridge enabling additional functionalities by the mapping of o®-chip components into the address space of the Tomahawk, a single lane PCI Express interface realizing communication links of 2 GBit/s to a host computer, a VGA/Streaming interface that allows interfacing an AD/DA converter or a normal VGA display, a freely programmable DMA controller, general purpose I/Os and an UART interface. All components on the chip are connected by two low latency, high bandwidth, crossbar-like master/slave networkson- chip (NoC) [14] with 32 bit bus-width. The NoC performs static priority arbitration per slave and supports burst transfers of up to 63 data words. For latency improvement, the NoCs operate on negative clock transition. A sustained throughput of 5.47 GBit/s is achieved for each master-slave connection. The crossbar-like architecture allows parallel communication channels for each master-slave pair. For extensibility, the FPGA bridge provides a full bandwidth interface to the external world. Hence, additional components can be connected to Tomahawk. Moreover, depending on the function of the component attached to it, the FPGA bridge can be configured to work as master or as slave. The VDSPs and SDSPs, the LDPC decoder and the deblocking filter ASIP implementations are all based on the synchronous transfer architecture (STA) [4]. STA processors explicitly use register ¯le bypassing. In contrast to the traditional approach, the STA functional units hold and exchange data directly with each other, instead of using a central register ¯le. This significantly reduces the I/O bandwidth, size and power consumption of the register file.



(Figure -1 MPSOC)

The goal of this thesis project is to design the middleware layer of the software stack on MPSoCs, namely **the Run-Time Library(API)**, which performs the communications management, quality management and resource management for the underlying architectures. The API needs to be lightweight and easy to be ported to different platforms, e.g. real hardware prototype (TIOMAP) or virtual platforms (at different abstraction levels).You have the opportunity to learn and use state-of-the-art embedded software technology, investigate various ways of implementing run-time support for modern MPSoCs, and get hands-on experience of building software on silicon.

II. METHODOLOGY

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 ow and data ow 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 s 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.

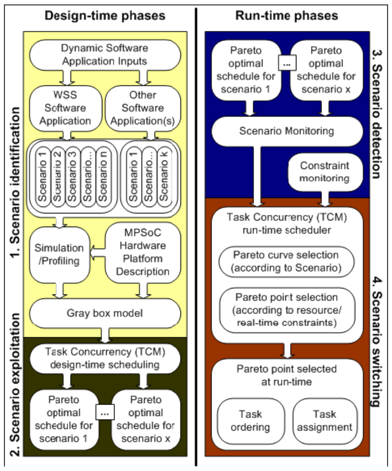

Design-time phases
Run-time phases
1. Scenario identification
Dynamic Software Application Inputs
WSS Software Application
Other Software Application(s)
Scenario 1
Scenario 2
Scenario 3
Scenario...
Scenario n
Scenario 1
Scenario...
Scenario k
Simulation /Profiling
MPSoC Hardware Platform Description
Gray box model
2. Scenario exploitation
Task Concurrency (TCM) design-time scheduling
Pareto optimal schedule for scenario 1
...
Pareto optimal schedule for scenario x
3. Scenario detection
Pareto optimal schedule for scenario 1
...
Pareto optimal schedule for scenario x
Scenario Monitoring
Constraint monitoring
4. Scenario switching
Task Concurrency (TCM) run-time scheduler
Pareto curve selection (according to Scenario)
Pareto point selection (according to resource/ real-time constraints)
Pareto point selected at run-time
Task ordering
Task assignment

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. 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 State B, 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- Pareto Build 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.

III. 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. Tg_: 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 e_cient approach for modeling the power consumption of complex processors. In DATE '04.
[9]. Z. Ma and F. Catthoor. Systematic methodology for real-time cost e_ective 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.