Adaptive Runtime Management of Heterogeneous MPSoCs: Analysis, Acceleration and Silicon Prototype

Oliver Arnold and Gerhard Fettweis
Vodafone Chair Mobile Communications Systems
Technische Universität Dresden
Dresden, Germany
{oliver.arnold, fettweis}@tu-dresden.de

Abstract—In this paper, a dedicated runtime management unit, called CoreManager, is presented. It controls a heterogeneous Multiprocessor System-on-Chip (MPSoC). Therefore, it dynamically schedules tasks on the available processing elements. Furthermore, it is responsible for memory as well as power management. The instruction set architecture of the CoreManager is extended to improve performance for dynamic data dependency checking, task scheduling, processing element (PE) allocation and data transfer management. A significant performance improvement can be shown for all components. Performance results are presented, analyzed and compared with RISC and ASIC based approaches. The integration of the CoreManager in the Tomahawk2 MPSoC silicon prototype is shown. Furthermore, area, timing and power consumption results are provided.

Keywords—Adaptive task scheduling; heterogeneous MPSoCs; CoreManager; runtime management;

I. INTRODUCTION

Simultaneously improving both, computational power and energy efficiency, represents one of the major challenges in the development of future hardware systems. In this regard, the systems’ manufacturing process as well as the hardware architecture must be examined. Concerning architecture design, the combination of different types of processing elements (PEs) in a single chip, is a recent trend. A smart management of these PEs allows an increase in computational power as well as energy efficiency. Static task schedulers are used especially for embedded systems, where power consumption and performance are crucial. In this case, the set of applications running on the hardware is limited and the overall application requirements/tasks are known. The characterization is the main challenge of this approach [1]. In particular, looking to mobile handsets, where more and more different applications/standards have to be supported and a full characterization of applications is infeasible, especially if several applications run in parallel. In such scenarios, dynamic task scheduling is better suited since complete characterization is not required. A dedicated core is in charge of scheduling the tasks to the available PEs. The schedule is built at runtime after a task data dependency analysis.

The resource-management problem is overcome by a dynamic scheduling unit, called the CoreManager, which controls the platform’s data plane [2] [3]. This unit is responsible for PE allocation, task scheduling, data and transfer management and power management. It features an application-specific instruction set [4] which enables increasing scheduling performance and system scalability while maintaining flexibility. In [14] an implementation for many-core system (>1000 PEs) is presented. The CoreManager can administrate signal processing and general-purpose applications concurrently [15]. In the signal processing case the CoreManager adheres to the applications’ deadlines, while implementing best-effort scheduling in the general-purpose scenario. Priorities on the application and task level are additionally supported. With the energy demands predicted by Moore’s law growing significantly faster than the devices’ battery capacity, energy efficiency has become an issue of outstanding importance in MPSoC design. The CoreManager’s energy-aware mode overcomes this problem by independently power-gating PEs and dynamically adjusting their frequency and voltage levels.

In [5] a software-based runtime engine for many-core system is introduced. Nevertheless, a dedicated task scheduling unit is not used. Furthermore, performance metrics for the applied scheduling implementation are missing. In [6] a task scheduling unit with an application-specific instruction set is introduced. The introduced overhead reduces the scalability of the implementation.

The remainder of the paper is organized as follows: In section II, the hardware system, the programming model and the PE integration framework are presented. In the following section the CoreManager and its instruction set architecture extensions are introduced. In Section IV the Tomahawk2 MPSoC silicon prototype is shown. The last section presents benchmarks and experimental results.

II. SYSTEM OVERVIEW

A. Hardware

The hardware framework of our work consists of an application processor, the CoreManager as well as several PEs [7]. All components of the MPSoC are connected by a Network-on-Chip (NoC). An operating system typically runs on the application processor while the PEs are responsible to execute computational intensive workload. The PEs are integrated by using a generic framework, including local instruction and data memories. The application processor sends dynamically generated task descriptions to the CoreManager by applying the taskC programming model. The PEs as well as their on- and off-chip data movements are dynamically controlled by the CoreManager.
B. TaskC Programming Model

The TaskC programming model [7] aims to i) exploit the underlying heterogeneous hardware platform, ii) enhance programmer productivity and ease code portability and iii) provide the flexibility required for adopting application and architecture specific features. From the programmer’s perspective, TaskC introduces a high-level parallel-programming interface where specific language elements enable spawning of functions from the host processor to the processing elements. Task dispatching, dependency analysis, resource allocation, task/data scheduling and synchronization, as well as memory coherency are hidden from the programmer. Furthermore, a concurrent software and hardware development is made available.

An example of the TaskC programming model is depicted in Fig. 1. A task is defined by a task name and its input and output data. The latter is defined at runtime. One and two dimensional transfers are supported. Control code dependencies are solved by the application processor. All data dependencies are analyzed at runtime by the CoreManager. One and two dimensional transfers are supported.

For an integration of several types of PEs a generic PE integration framework is available. A PE is attached to local instruction and data memories. It is controlled by a small finite-state machine (PE control in Fig. 2). The finite-state machine is made available. A data transfer engine managed by the CoreManager by task and transfer descriptions machine (PE control in Fig. 2). The finite-state machine is and data memories. It is controlled by a small finite-state framework is available. A PE is attached to local instruction

C. PE Integration Framework

For an integration of several types of PEs a generic framework is available. A PE is attached to local instruction and data memories. It is controlled by a small finite-state machine (PE control in Fig. 2). The finite-state machine is managed by the CoreManager by task and transfer descriptions which are stored in local buffers. A data transfer engine (DMA) is responsible to prefetch necessary data before task execution.

In this work a dedicated scheduling unit, called CoreManager, is proposed to control heterogeneous MPSoCs. This unit is responsible for dynamically distributing atomic tasks on different PEs as well as managing the inherent data transfers. For this purpose, a runtime analysis of the data dependencies is firstly performed. Based on this analysis, a schedule is created to allocate the PEs and explicitly reserve and administrate the local memories. The results of the dynamic data dependency analysis are additionally reused in order to increase data locality. In particular, required data are kept in the local memories, thus reducing the number of transfers from the global memories.

It features an application-specific instruction set [4] which enables increasing scheduling performance and system scalability while maintaining flexibility [8]. The CoreManager adheres to the applications’ deadlines, while implementing best-effort scheduling in the general-purpose scenario. Priorities on the application and task level are additionally supported.

The CoreManager has been profiled in order to expose the most time consuming components. Dynamic data dependency check was found to be the limiting factor regarding system scalability. An extension of the instruction set architecture of the CoreManager has been developed and integrated to solve this issue. This extension allows speeding-up the data dependency check process by nearly two orders of magnitude. Similar performance improvements are shown for the (task-) scheduling (earliest deadline first, priority based etc.), PE allocation and local memory management. E. g., the scheduling time (each task has its own deadline and valid flag) was dramatically reduced by implementing the approach depicted in Fig. 3. 16 tasks are concurrently evaluated in a single clock cycle.

III. COREMANAGER

The CoreManager explicitly manages power. Therefore, the current hardware system status as well as the application load is regarded to allocate a suitable PE. Furthermore, the CoreManager’s energy-aware mode independently power-gates PEs and dynamically adjusts their frequency and voltage levels.
Furthermore, a failure-aware dynamic task scheduling approach for unreliable heterogeneous MPSoCs was integrated in the CoreManager [9]. It enables a detection and isolation of erroneous PEs, connections and memories. An error-free execution of the application is assured.

The applied tool flow for instruction set extension development is shown in Fig. 4. In a first step, the CoreManager runtime behavior is analyzed. Hence, the most time consuming parts of the firmware are unveiled (hotspots). Based on these results a scheduling-specific instruction set is developed to improve performance. Afterwards, a processor generator is used to generate a cycle-accurate processor model and a corresponding compiler [13]. If the performance requirements are met and the core is verified, the HDL processor generator is used to generate an RTL description of the processor. It is synthesized to obtain area, power and timing figures of the core.

![Figure 4. Tool flow for instruction extension development and analysis](image)

IV. SILICON PROTOTYPE: TOMAHAWK2 MPSOC

The developed CoreManager and PE integration framework have been integrated in a heterogeneous MPSoC, which was fabricated by TSMC in a 65 nm low power CMOS process [11]. The CoreManager is responsible for the control of the data plane. In this case the data plane consists of eight vector DSPs, eight scalar general purpose cores, a flexible forward error correction block as well as a multi-antenna detection engine. The Tomahawk2 die photo is depicted in Fig. 5. Additionally to the processing modules, an FPGA as well as two DDR2 interfaces are available. The demonstrator is depicted in Fig. 6. On top of the power module the Tomahawk2 module is plugged in. It contains the Tomahawk2 package and two DDR2 memories.

The Tomahawk2 features a combined DVFS and AVS power management architecture [11]. Each PE can be dynamically switched to one of the four performance levels (PL). The core frequency is generated by a local ADPLL with an open loop output clock generator that allows instantaneous frequency changes. This enables PL-change times below 20 ns. The chip components are connected by a packet-switched NoC. Point-to-point connections with long physical distances in the mm range are realized as high-speed serial links (dashed lines in Fig. 5). Each link provides a bandwidth of 80 Gbit/s.

![Figure 5. Tomahawk2 die photo, including the CoreManager (CM core)](image)

Figure 5. Tomahawk2 die photo, including the CoreManager (CM core)

![Figure 6. Tomahawk2 demonstrator](image)

V. RESULTS

A. Performance

In Fig. 7 the processing time of the task scheduling is shown. The number of concurrently processed tasks is varied between 1 and 32. In the case of the CM-EIS implementation with the application specific instruction set only six cycles are required to process 32 tasks - instead of over 400 cycles in the case of the RISC cores. The processing time of the PE allocation is depicted in Fig. 8. Two allocation possibilities are compared: the allocation of possible PEs and the concurrent evaluation of preferred and possible PEs. In the case of 32 PEs the required cycles are reduced from 125 (CM-LX4) to six (CM-EIS). Further performance results are available in [4], [8] and [7].

![Figure 7. Task scheduling processing time](image)
B. Area and Power Consumption

The CoreManager occupies 0.49 mm² (w/o memories) and achieves a maximum frequency of 500 MHz. Table I illustrates the measured power consumption (w/ memories, DMACs, FIFO buffers and interfaces) on the Tomahawk2. At 200 MHz between 14.1 mW and 15.6 mW and at 445 MHz between 66.7 mW and 74.6 mW are consumed, respectively.

<table>
<thead>
<tr>
<th>Frequency [MHz]</th>
<th>Power Consumption [mW]</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 @ 0.9 V</td>
<td>15.6 14.1 14.4</td>
</tr>
<tr>
<td>286 @ 1.0 V</td>
<td>28.4 25.1 25.8</td>
</tr>
<tr>
<td>333 @ 1.1 V</td>
<td>38.5 34.7 36.1</td>
</tr>
<tr>
<td>445 @ 1.2 V</td>
<td>74.6 66.7 68.0</td>
</tr>
<tr>
<td>500 @ 1.3 V</td>
<td>91.7 80.0 81.5</td>
</tr>
</tbody>
</table>

C. Comparison

Table II illustrates a comparison of three different implementations styles: a hardware block (ASIC), a general purpose RISC core as well as the application-specific instruction set processor (ASIP) in this work [11]. In contrast to the ASIC, the ASIP approach has a flexible scheduling approach.

<table>
<thead>
<tr>
<th></th>
<th>ASIC [12]</th>
<th>RISC</th>
<th>ASIP (this work)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheduling Configurability</td>
<td>Fixed</td>
<td>Flexible</td>
<td>Flexible</td>
</tr>
<tr>
<td>Task Queue size</td>
<td>16</td>
<td>16-256</td>
<td>16-256</td>
</tr>
<tr>
<td>Frequency [MHz]</td>
<td>175</td>
<td>445</td>
<td>445</td>
</tr>
<tr>
<td>Task scheduling [us]</td>
<td>0.4</td>
<td>16.9</td>
<td>0.9</td>
</tr>
<tr>
<td>Technology [nm]</td>
<td>130</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>Supply Voltage [V]</td>
<td>1.3</td>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>Power [mW] @ fmax</td>
<td>282</td>
<td>68*</td>
<td>74.6</td>
</tr>
<tr>
<td>Energy per Task [nJ]</td>
<td>113 (27')</td>
<td>1149</td>
<td>67</td>
</tr>
<tr>
<td>Area (logic) [mm²]</td>
<td>4.51 (1.13')</td>
<td>0.34'</td>
<td>0.49</td>
</tr>
<tr>
<td>ATE product [mm²<em>us</em>nJ]</td>
<td>204 (12')</td>
<td>6602</td>
<td>29</td>
</tr>
</tbody>
</table>

VI. CONCLUSION

In this paper, a dedicated runtime management unit, called CoreManager is presented. It dynamically schedules tasks on a heterogeneous MPSoC. Furthermore, it is responsible for PE and memory allocation, data transfer management as well as power management. By applying a scheduling-specific instruction set, the performance and energy efficiency of the CoreManager was improved by several orders of magnitude. Furthermore, the CoreManager was integrated in the Tomahawk2 silicon prototype. Future works aims the integration of further processing elements. Especially, high-performance hardware accelerators will be regarded.

ACKNOWLEDGMENT

The major part of this research work has been funded by the DFG through the cluster of excellence ‘Center for Advancing Electronics Dresden’ and the European Union and the state of Saxony through the IMDATA and EuroServer projects. A minor part was funded by the BMBF within the scope of the CoolBaseStations project. Furthermore, we would like to thank Synopsys and Tensilica for software and IP.

REFERENCES