EXPLOITING LOOP-LEVEL PARALLELISM ON MULTI-CORE ARCHITECTURES FOR THE WiMAX PHYSICAL LAYER

Ying Yi, Wei Han, Adam Major, Ahmet T. Erdogan and Tughrul Arslan

University of Edinburgh, The King's Buildings, Mayfield Road, Edinburgh, EH9 3JL, UK

ABSTRACT

Multiprocessor system-on-chip (MPSoC) architectures are increasingly being adopted in the design of emerging complex embedded systems. This paper introduces a new application mapping technique for MPSoC architectures using a case study. The technique incorporates profiling-driven task partitioning, task transformations, loop level partitioning and memory architecture aware data mapping to reduce system execution time. Experiments are conducted to evaluate the technique by implementing the WiMAX physical layer on several multi-core architectures based on newly emerging dynamically reconfigurable processor cores. The results demonstrate that the proposed technique is able to generate high-quality mappings of realistic applications on the target architecture, achieving up to 4.92x speedup by employing only five dynamically reconfigurable processor cores.

I. INTRODUCTION

MPSoC designs offer high performance and flexibility and at the same time promise low-cost and power-efficient implementations. However, the semiconductor industry is still facing several technological challenges with MPSoC systems. Important issues in MPSoC design are the communication infrastructure, memory architecture, and task mapping. In our newly proposed embedded MPSoC platform which has several dynamically reconfigurable processors [1], the shared memory heavily affects the execution time and power consumption. Hence, it is essential to introduce new shared memory elements in order to reduce the energy consumption and the execution time.

In MPSoC architectures all parallel tasks in an application have the potential to be executed simultaneously. However the number of such tasks may exceed the number of available processors. Therefore task mapping is required to assign the parallel tasks to the available processors. In the past, task merging and task replication have been proposed with the goal of re-allocating tasks when performance bottlenecks are met. Since task merging requires more local memory and task replication needs more processors to implement the same task [1], an MPSoC architecture which does not feature sufficient memory and processors will severely limit the available mapping options using the current methodology.

This work introduces the deployment of a shared multi-bank register file for MPSoC architectures in order to reduce the overall memory access time. A new mapping approach, which divides the tasks at the instruction level instead of at the function level, is developed. The proposed memory architectures and novel mapping support enable high-quality mappings of the applications onto the MPSoC architectures. The efficiency of the technique is demonstrated by the WiMAX implementation.

II. RELATED WORK

The access time of a register file depends on both the size of register file and the number of ports. The authors of [2] have proposed the use of distributed schemes as opposed to a central implementation. Similarly, [3] used techniques to split the global micro-architecture into distributed clusters with subsets of the register file and functional units. Multi-level register file organizations have also been introduced to reduce the register file's size [4]. All the works mentioned above focus on reducing the number of registers, however, other techniques exist which split the register file into interleaved banks; reducing the total number of ports in each bank, but retaining the idea of a centralized architecture [5]. In contrast to previously proposed techniques, we propose a shared register file which is split into independent banks with a reduced number of ports per bank.

Application development on multi-core processors requires the designer, or automated tool, to divide tasks between available processors and to determine data mappings for the required memory elements. A SystemC-based simulation framework for mapping an application to a platform and evaluating its performance has been presented in [6]. The authors in [7] have introduced scheduling and mapping parallel applications onto an MPSoC platform. Mapping solution for NoC-based MPSoC has been described in [8]. Some automated system-level mapping techniques for application development on network processors have also been proposed [9]. We propose a mapping technique which incorporates profiling-driven task partitioning, task transformation, loop level partitioning

978-1-4244-2596-9/08/$25.00 ©2008 IEEE

and memory architecture aware data mapping in order to reduce the overall system execution time.

III. PROPOSED ARCHITECTURAL EXTENSIONS

Some applications demand a closer interconnection between the participating processors to achieve the required performance. Such a communication can be realised using distributed shared register files. The baseline architecture [1] consists of a selectable number of dynamic reconfigurable (DR) processors, which communicate with a shared memory through a full crossbar network. This architecture has been extended and modified by incorporating the shared register file into the system memory architecture in order to support the loop level parallelism proposed in this paper.

In order to reduce access time, we propose to split the register file into independent banks with a reduced number of ports per bank. The proposed approach employs a multi-bank dual-port register file where each bank consists of a 32x32-bit single-write-port and single-read-port data register file. A number of custom register file interface cells, which provide access to these banks arbitrated by the individual arbiters, are introduced to support distributed shared register files. The number of register file banks can be parameterised at synthesis stage. The MPSoC architecture proposed in this paper is based on a recently introduced DR processor architecture [10]. It contains an array of instruction-set functional units connected by a programmable interconnection. All instruction cells in the DR processor can be connected to all of the register file interface cells, but each register file interface cell is only able to connect to one fixed register bank. This scheme allows all possible connections between the instruction cells and the shared register banks, but with a reduced MPSoC interconnection complexity.

The existing DR processor tool-flow provides full support for the custom cells through the simulator libraries. A fully automated system generator compiles the standard simulator libraries with the custom cell objects and generates all the required support files alongside the custom simulator binary, replacing the standard simulator used in the DR processor tool flow. The shared register file has been synthesised with Cadence Ambit BuildGates using the UMC 0.18um standard CMOS cell library. The power consumption of the synthesized register file has been obtained using Synopsys PrimePower. The generated timing, area and power information is used in the DR processor tool flow for scheduling and simulation purposes.

IV. PROPOSED MAPPING FLOW

The mapping and scheduling of tasks involved in the implementation of wireless applications on a multiprocessor architecture are complex optimisation problems, which need to be solved simultaneously to maximise the throughput. In addition, data transmission time between different processors must also be taken into account during task mapping and scheduling.

The proposed mapping flow (Fig. 1) allows the designer to explore applications on the target architecture. A detailed description of an initial mapping methodology and the multi-core processor simulator has been given in [1]. This paper mainly focuses on the automatic mapping techniques and improving the memory architecture. An integer linear programming (ILP) based approach is proposed for task mapping, loop level parallelism and pipelined scheduling, taking the data transmission time into account for embedded applications targeted on the multiprocessor platform.

During the process of task partitioning and task mapping, execution-time estimates for the different tasks, as well as the time estimates for transferring the data between processors, are required. It is also necessary to schedule the execution order of these pipelined tasks to improve the system performance. The execution time of a task performed by a processor can be obtained by profiling using the custom DR simulator. The mapping flow starts from the description of an application in standard sequential C code which is then optimised and profiled for a single DR processor implementation. The proposed mapping approach is based on the control data flow graph and static profiling generated by the single DR processor tool flow. The static profiling information contains the timing characteristics for each task and the access frequency for the various data items. This information is used for mapping tasks to available processors as well as mapping various data items to memory locations. The control data flow graph provides the instruction level parallelism information for loop level partitioning.

The mapping problem can be described as follows: Given a characterised application and a target architecture, the objective is to obtain an application to architecture mapping such that the worst case throughput is maximised. Our solution consists of dividing the problem into two stages and solving each consecutively. The first stage assigns functions or parallel instruction packages to processors, assuming an idealistic memory mapping, where all data items are mapped to the fastest possible level of memory, ignoring memory capacities. The ILP formulation of this stage includes task merging, task replication, and loop level partitioning. The task merging combines several tasks into a single task that sequentially performs tasks in their original order. This technique reduces the number of required processors, but needs more local instruction and data memories. Task replication assigns the same task to several processors such that all instances of the

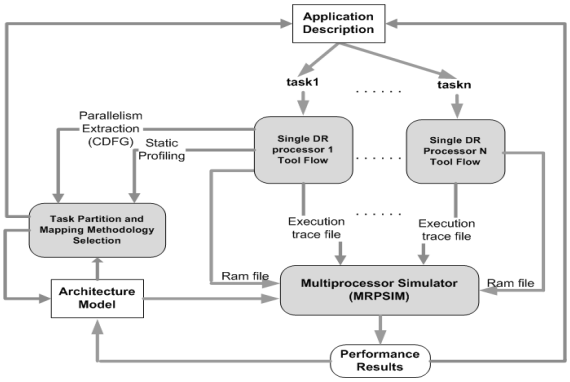

Figure 1: Mapping methodology

task are executed in parallel. Therefore, task replication needs more processors to implement an application and also more global memory to save the shared data. A new mapping approach is needed when the workload among the processors is imbalanced and task replication cannot be used due to the limited number of available processors. The new mapping approach divides the tasks at the instruction level instead of at the function level in order to explore the instruction level parallelism. This approach will be illustrated in section V, using an implementation of the WiMAX physical layer.

The second stage performs a mapping of data items to memory locations and explores the memory architecture for minimising memory access latencies. Each DR processor can access three types of memory: (a) the shared multi-bank register file, (b) local memory, and (c) global memory. The local memories are private to a processor and cannot be accessed by any other processors. Thus, shared data items, accessed by tasks assigned to different processors, cannot be mapped to these memories. Instead, they are mapped either to the shared register file or to global memory, which can be accessed by the different processors.

V. CASE STUDY: A WiMAX PHYSICAL LAYER

Let us consider the execution of the WiMAX application on a single DR processor. The WiMAX application consists of five tasks [1], which are channel coding, IFFT, FFT, demodulation, and decoding with execution times of 60μs, 108μs, 109μs, 28μs and 1206μs, respectively. To demonstrate the proposed mapping methodology, the WiMAX application is mapped to several different multi-core architectures.

The application is mapped onto a multi-core processor architecture described in the scenario 1 (Table 1). The first stage of mapping merges the channel coding, IFFT, FFT, and demodulation tasks into a single task. The merged tasks are assigned to one processor and the remaining decoding task is assigned to the other. This mapping leads to a highly imbalanced workload since the throughput is still determined by the most time-consuming task (i.e., decoding), which takes 80% of the total execution time. Task replication cannot be applied to the decoding task because there are not enough DR processors.

To further improve the throughput, a loop partitioning method is proposed to balance the workload between the two processor cores. The decoding task contains the most time-consuming loop body, and has a loop-carried [1] dependence, which prevents further efficient partitioning at the task level. The body of the main loop of the decoding task consists of both independent and dependent data paths. It is possible to run the independent data paths on different processor cores to increase the instruction level parallelism using loop splitting. After the loop partitioning, the merged task and part of the first 670 iterations are assigned to one processor core, the remaining part of the 670 iterations and the whole of the last 200 iterations are assigned to the second processor core (Fig. 2).

The second stage of mapping performs the mapping of data items to memory locations. If there are sufficient local memories, all local data items of each processor are mapped to the local memory and shared data items between the different processors are mapped to either the shared registers or the global memory. Otherwise, some local data items have to be moved from the local memory to the global memory. Shared memory can become a performance bottleneck for the loop partitioning method due to the frequent data communication between the DR processors. Therefore, frequently accessed shared data should be mapped to the shared register file in order to reduce memory access time and memory access conflicts.

A more efficient mapping can be implemented for a multi-core processor architecture which has five available processor cores. The decoding task is replicated across the additional three DR processor cores; assuming sufficient instruction and data memory. These different mappings based on several multi-core architectures are described in Table I. To make fair performance comparisons, all scenarios are executed with 100 frames. The experimental results are based on the following assumptions: the DR processors operate at 500MHz, the shared memory access delay is 6ns, the local private memory access delay is 2ns, and the shared multi-bank register file access delay is 1ns. The data mapping results together with the number of memory access operations for each scenario are given in Table II. Table II provides the average number of memory access operations per frame. Table III shows the total execution time, the speedup, parallel efficiency, and average idle ratio [1] of the different scenarios. As the results show, the scenarios with sufficient

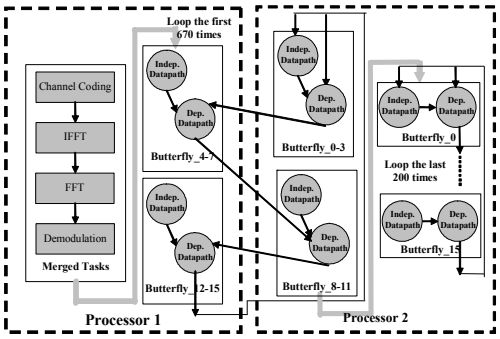

Figure 2: Loop Partition Method

processors employing task merging and replication achieve both the highest speedup and the highest parallel efficiency, compared to other task mapping methods. In addition, all processor cores in the scenarios are much more efficient with very low idle ratios. This is mainly because task replication employs more processor cores and dispatches a more balanced workload to the DR processor cores. The scenarios with sufficient local memory have better throughput compared to their counterparts which have limited local data memory. Scenario 5 with sufficient local memory and processor cores achieve 4.92x speedup and a parallel efficiency of 0.98 with five target DR processor cores. Based on the multi-core architecture with limited processor cores, scenario 3, which uses task merging and loop level partitioning, has a much lower total idle ratio and better performance compared to scenarios using task merging only. This is due to the improved instruction level parallelism and the efficient multi-core architecture with a shared register file between different processor cores. The simulation results show that the proposed mapping methodology offers a better solution to previous mapping methodologies based on task merging and/or replication.

VI. CONCLUSIONS

This paper has proposed a new application mapping technique for MPSoC architectures. The technique utilizes profiling-driven task partitioning and transformation. These are intelligently fused with a novel loop level partitioning and a memory aware data mapping in order to reduce system execution time. Several mapping solutions based on different MPSoC architectures have been generated for the WiMAX physical layer application. Simulation results demonstrate the effectiveness of the proposed mapping strategies showing up to 4.92x speedup and 0.98 parallel efficiency for an MPSoC architecture with five DR processor cores.

Table 1: Several Architectures and Mapping Methodologies

	No. Procs.	Mapping Methods	Memory Capacity		
			Shared	Local	Reg. File
Sce.1	2	Merging	256KB	2 KB	Nil
Sce.2	2	Merging	256KB	64 KB	Nil
Sce.3	2	Merging + Loop Parti.	256KB	2 KB	8*32*32 b
Sce.4	5	Merging+ Replication	256KB	2 KB	Nil
Sce.5	5	Merging + Replication	256KB	64 KB	Nil

Table 2: Data Mapping

	Mapping (Memory Usage)			Memory Accesses		
	Shared	Local	Reg.	Shared	Local	Reg. File
Sce.1	7 KB	1.6 KB	Nil	159100	1879	Nil
Sce.2	0.6 KB	8 KB	Nil	3481	157498	Nil
Sce.3	7.1 KB	1.6 KB	14 B	161865	1897	28188
Sce.4	8.7 KB	1.9 KB	Nil	199139	1871	Nil
Sce.5	2.3 KB	8.1 KB	Nil	9484	157526	Nil

Table 3: Implementation Results

App.	Time (ns)	Speedup	P. E.	I.R.
Sce.1	132721581	1.15	0.58	36.9%
Sce.2	119755277	1.28	0.64	38.1%
Sce.3	99053964	1.55	0.77	12.0%
Sce.4	38347883	3.99	0.80	7.9%
Sce.5	31155968	4.92	0.98	5.2%

REFERENCES

1. W. Han, Y. Yi, M. Muir, I. Nousias, T. Arslan, and A. T. Erdogan, "Multi-core Architectures with Dynamically Reconfigurable Array Processors for the WIMAX Physical Layer," in 6th IEEE Symposium on Application Specific Processors, 2008.
2. V.V. Zyuban, P.M. Kogge, “The energy complexity of register files”, International Symposium on Low Power Electronics and Design, pp.305-310, 1998.
3. V. V. Zyuban and P. M. Kogge, “Inherently lower-power highperformance superscalar architectures”, IEEE Transactions on Computers, 50(3):268–285, March 2001.
4. J.L. Cruz, A. Gonzalez, etc, “Multiple-banked register file architecture”, the 27th international Symposium on Computer Architecture, pp. 316 - 325, 2000.
5. I. Park, M.D. Powell, T.N. Vijaykumar, “Reducing register ports for higher speed and lower energy”, in Proceedings. 35th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 171-182, 2002.
6. T. Kempf, M. Doerper, R. Leupers, G. Ascheid, H. Meyr, T. Kogel, and B. Vanthournout, "A modular simulation framework for spatial and temporal task mapping onto multi-processor soc platforms," in Proceedings of the conference on Design, Automation and Test in Europe (DATE), pp. 876-881, 2005,.
7. P. G. Paulin, "Automatic mapping of parallel applications onto multiprocessor platforms: a multimedia application," in Digital System Design, Euromicro Symposium, pp. 2-4, 2004.
8. C. Marcon, A. Borin, A. Susin, L. Carro, F. Wagner, “Time and Energy Efficient Mapping of Embedded Applications onto NoCs”, Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 33- 38, Vol. 1, 2005.
9. C. Ostler, K.S. Chatha, “An ILP Formulation for System-Level Application Mapping on Network Processor Architectures”, Design, Automation & Test in Europe Conference & Exhibition (DATE), pp.1-6, 2007
10. S. Khawam, I. Nousias, M. Milward, Y. Yi, M. Muir, and T. Arslan, "The Reconfigurable Instruction Cell Array," Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, vol. 16, pp. 75-85, 2008.