GENERATION OF MULTI-CORE SYSTEMS FROM MULTITHREADED SOFTWARE

Alexander Wold, Jim Torresen

Department of Informatics
University of Oslo, Norway
email: {alexawo,jimtoer}@ifi.uio.no

Andreas Agne

Computer Engineering Department
University of Paderborn, Germany
email: agne@upb.de

ABSTRACT

A heterogeneous system with soft CPU tailored to the individual threads of the application, while still software based, offers the potential for improved performance and resource utilization over a homogeneous system. In this paper we present a method to automatically create a heterogeneous multi-core system from a multithreaded software application. The resulting system consists of processing elements based on customized MIPS soft CPUs coupled with their respective programs. Using instruction set architecture (ISA) subsetting, we adapt the individual soft CPUs to the specific computations they have to perform. We have carried out a case study with a constraint solver application for which we find a performance increase of 1.54 accompanied with an area reduction of 22.5% compared to a homogeneous multi-core system. We also present an automated toolchain that generates synthesizable IP-cores from software threads with little additional development overhead.

1. INTRODUCTION

FPGAs allow not only the implementation of custom multiprocessors systems, but also implementation of complete application specific systems. This can be further exploited to implement a heterogeneous multi-core system [1]. These system may consist of multiple soft CPUs with a custom interconnect [2, 3]. Compared to a single feature rich CPU, multiple lightweight CPUs have the potential to improve performance of multithreaded applications implemented on FPGAs. This takes into account that even parallel applications typically contain significant parts of sequential code. In this paper, we combine multi-core processing and lightweight application specific soft CPUs, to allow concurrent execution of multithreaded applications.

We partition the application into a set of threads which are concurrently executed. Our goal is to exploit concurrent execution, increased memory throughput (each soft CPU has dedicated local memory), and data locality. In addition, we optimize the soft CPUs individually to each software thread at compile time. The result is an optimized, lightweight soft CPU that typically allows an increase in clock frequency in addition to reduced resource requirements.

A soft CPU can be adapted to a particular program. One way to accomplish this is with instruction set extensions (ISEs) [4]. Typically, soft CPU customization is undertaken with the target of accelerating the most frequently used functions. This has resulted in performance improvements as reported in previous publications [5, 6]. ISEs usually target applications where the CPU time is dominated by smaller code sections (i.e. compute kernels). Compute kernels are prevalent for example in audio and video applications. In applications without compute kernels, the effect of ISEs is limited, and may also be detrimental to performance/area. In this case, a CPU *without* ISEs may offer higher performance per area than a CPU with ISEs. We customize the soft CPUs by implementing only instructions that are actually used by the software. This is known as ISA subsetting [6]. We accomplish this by parsing the compiled code and then remove unused instructions from the soft CPU.

Our work targets the ReconOS [7] framework. The ReconOS framework is an operating system layer that implements the multithreaded programming model for software as well as for hardware threads. ReconOS features a POSIX interface for inter-thread synchronisation and communication in software as well as in hardware. This is implemented using FIFO based channels that connect the hardware threads to the main CPU that executes a multithreading host operating system. ReconOS supports message passing and shared memory, enabling seamless communication between hardware and software threads. Since ReconOS hardware threads are typically specified in a HDL, a wide range of hardware can be integrated as hardware threads. In our approach, we implement MIPS based multi-cores as ReconOS hardware threads. We generate ReconOS hardware threads as IP-cores compatible with EDK/ISE and other electronic design automation (EDA) tools. The IP-cores consist of a ROM containing the instructions of the pre-compiled software threads, the custom soft CPUs, and other generated HDL code and project files. In addition, for simulation, testbenches are generated. This allows unit testing of the IP-core. Communication between ReconOS and the hardware threads is implemented by memory mapping the hardware thread's memory and operating system interfaces to the data bus of the soft CPU system. This allows the generated IP-cores to be implemented as ReconOS hardware threads. The customization and generation of the soft CPUs is fully automated in the tool flow.

The remainder of the paper is organized as follows: background and related work are presented in the following section. In Section 3 we describe our system architecture that is based on ReconOS. In Section 4, we present our toolchain. Results are presented in Section 5. This is followed by a

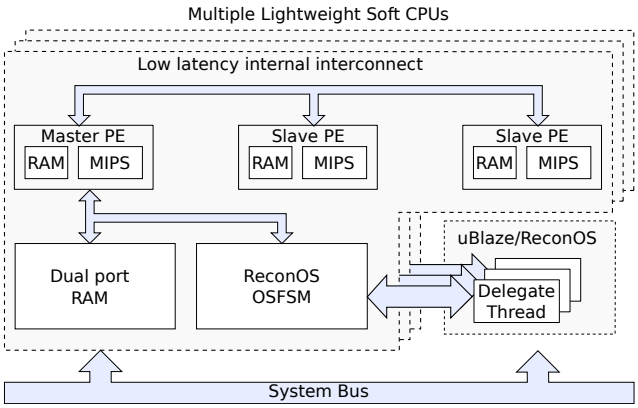

Fig. 1. A ReconOS hardware thread capable of executing multithreaded software concurrently.

conclusion in Section 6.

2. BACKGROUND AND RELATED WORK

Within the FPGA and network on chip (NoC) community, several related publications focus on multicore architectures on FPGAs. The majority of publications on multicore systems consider homogeneous processing elements. In [2], Kinsy et al. present a MIPS based multicore system. The system features a fully parameterizable NoC style interconnect. In addition, the memory system is parameterizable for each soft CPU. In [3], Motruk et al. describes the use of such a system in a safety application. In [1], Tumeo et al. present a heterogeneous multiprocessor FPGA platform. Tumeo et al. use both hard core PowerPC CPUs and MicroBlaze soft CPUs. The main difference between these publications, and the work presented in this paper, is that we present a toolchain to generate systems consisting of soft CPUs which are optimized for particular threads.

A significant amount of related work on application specific customization of processors has been undertaken. A survey on ISEs is presented in [4]. To implement ISEs, first it has to be determined which functions of an application have potential benefit of acceleration. Functions can be identified manually or with profiling. Customizations targeting the ISA are usually tightly coupled to the CPU giving access to registers. The ISEs then have to be implemented and integrated into the CPU. The tool chain has to be extended with support for the ISEs. Furthermore, the application has to be adapted to use the ISEs. This can be accomplished with inline assembly in the source code. The process can be manually undertaken or with support of automated frameworks. In [8], [9], work to automate the process is presented.

In addition to ISEs, unused instructions can be removed from the soft CPU. Eliminating unused instructions improves performance per area. The design is simplified and typically, as a result, runs at a higher frequency. The term ISA subsetting is used for removing unused parts of the instruction set. ISA subsetting is undertaken by parsing the compiled code, and removing instructions not used from the soft CPU. The customized soft CPU executes unchanged applications. In [6], Yiannacouras et al. report area reductions of up to 60% and clock frequency increases from 4% to 20%. In the results presented in [6], 25% to 50% of the MIPS-I instruction set is unused. ISA subsetting is typically more useful for compute kernels. Compute kernels tend to use a smaller portion of the instruction set. Larger applications typically use a greater part of the instruction set, and the impact of ISA subsetting decreases, as fewer instructions can be removed. For example, we found that the embedded operating system uCos compiled with GCC use 48 (out of 75) different MIPS instructions (depending on compiler options). Thus reduces of the amount of instructions which can be removed. Thus, the benefit of ISA subsetting for larger applications is limited. Both ISEs and ISA subsetting can potentially improve performance per area. In this work, we have implemented ISA subsetting in order to show the benefit of a heterogeneous multi-core system.

The number of different instructions an application use can be reduced with compiler options. Compilers can be configured to not use specific instructions such as floating-point coprocessor instructions and multiplication/division instructions. Depending on how often the omitted instructions are used, this can result in decreased application throughput. However, the resource footprint of the soft CPU will be reduced at the same time. Further, it is possible to emulate instructions in software or to use run-time reconfiguration of the CPU instruction set. This allows the use of binaries with instructions not implemented in the soft CPU.

3. LIGHTWEIGHT PROCESSING ELEMENTS IN RECONOS

ReconOS is an operating system layer that implements the multithreaded programming model for software and hardware threads. A ReconOS system usually consists of a reconfigurable system on chip (SoC) with a main CPU and multiple threads implemented either in software or in hardware. It provides a common interface to both software and hardware threads. Up to 14 hardware threads are supported in ReconOS. The hardware threads are typically implemented in HDL (VHDL or Verilog).

In a ReconOS system, a host operating system with multithreading support runs on the main CPU. For each hardware thread, there exists one software *delegate* thread that transparently communicates with its associated hardware thread and executes operating system functions on behalf of the hardware thread. The ReconOS delegate thread uses the host operating system's POSIX interface to handle communication, while a VHDL library offers a functionally equivalent interface to the hardware thread. POSIX features such as message passing, semaphores, and shared memory are supported. A shared memory address space allows for communication between the threads without invoking the main CPU.

In our work, the hardware threads are lightweight processing elements (PEs). A PE contains a soft CPU, local memory and the program. The lightweight soft CPU is a parameterizable MIPS-derived CPU. The effect of ISA subsetting

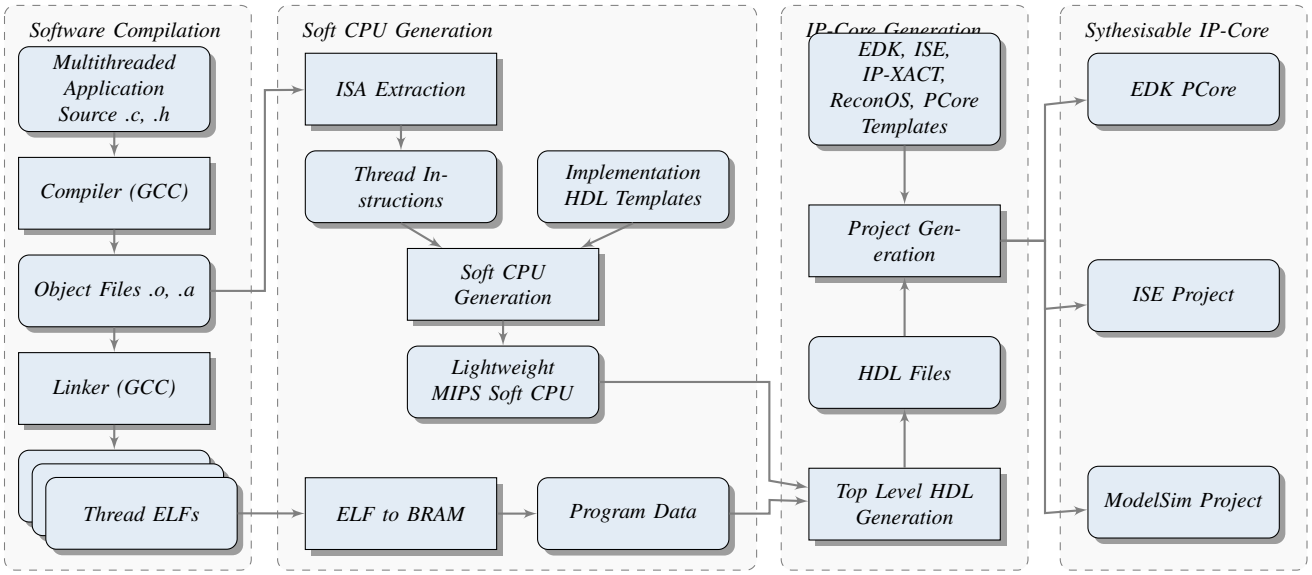

Fig. 2. The tool flow depicts the process to generate thread specific soft CPUs

on performance and area depends on the number of different instructions used. Each program uses only a limited set of instructions. Thus, we instantiate soft CPUs with limited functionality, potentially allowing removal of a large part of the ISA.

A ReconOS hardware thread can be implemented with multiple PEs. This is depicted in Fig. 1. In contrast to a thread which posesses heavy-weight feature-rich interfaces to the operating system and the system memory, PEs within a thread may exchange data using a comparatively inexpensive and fast communication infrastructure. This allows for low latency communication between the PEs of a ReconOS hardware thread where message passing, for example, can be used without involving the main CPU. The soft CPUs can be still be heterogeneous when implemented in a single ReconOS thread. In this configuration, the soft CPUs are partitioned into one master and a pool of slaves. Only the master communicates with the delegate thread. This configuration targets applications where the master schedules jobs on the slaves.

Internal communication within hardware thread between the soft CPUs is implemented with FIFO based channels. Generics are used in the implementation to adapt the FIFO channels to the application. For example, both the channel width and the FIFO depth can be configured (manually). A software library provides functions for asynchronous and blocking message passing.

4. TOOLCHAIN

The toolchain allows implementation of a multithreaded application into one or more of the IP-cores introduced in the previous section. This is accomplished with scripts and a template library. The template library contains project and HDL files. IP-cores are generated by adapting the templates to the application. We have automated this with scripts. The complete tool flow is depicted in Fig. 2. In the following sections we describe the tool flow in detail. Our flow consists of the following steps:

The first step is to compile the application. Compilation is undertaken with standard GNU tools. The compilation is organized by a makefile. This builds a separate ELF file for each soft CPU. However, it is up to the developer to partition the application into appropriate threads.

The lightweight MIPS soft CPU is generated from a VHDL template. Each compiled thread is parsed to identify used instructions. Only instructions present in the compiled thread are implemented in the generated soft CPU. This keeps the generated soft core lightweight. The resulting thread specific soft CPU implements a subset of the MIPS ISA.

After a soft CPUs have been generated for each thread, the interconnection infrastructure is generated. The interconnection infrastructure is adapted to the application. This consists of generating a top level VHDL based on a template. The top level file instantiates and connects the soft CPUs to the internal interconnection infrastructure and the ReconOS interfaces.

The result of the IP-core generation is an IP-core ready to be integrated into ReconOS. The system can then be implemented and simulated with FPGA vendor tools. Implementation can be done with Xilinx EDK. The IP-core also contains simulation scripts written in TCL and ModelSim project files.

5. EXPERIMENTAL RESULTS

We have implemented different variants of a concurrent embedded constraint solver to evaluate our tool flow. The con-

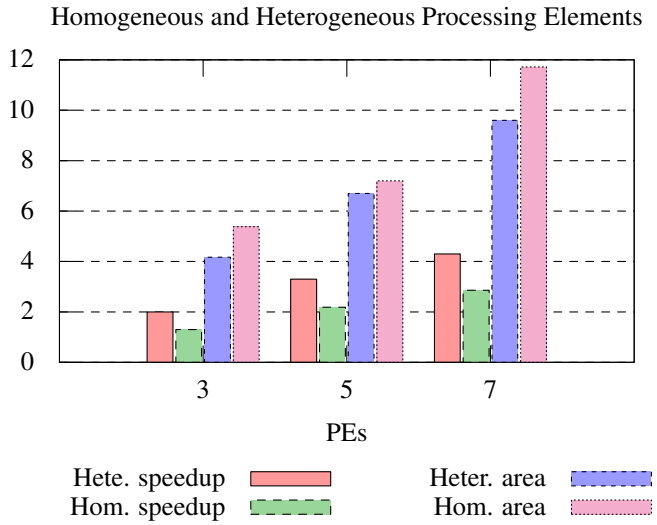

Fig. 3. Area and speedup of the constraint solver using homogeneous and heterogeneous PEs.

current constraint solver is using one, three, five and seven PEs. We have measured speedup and resource increase using both homogeneous PEs and heterogeneous PEs. In Fig. 3, the speedup relative to an single PE implementation is depicted for heterogeneous as well as homogeneous systems, with varying number of PEs. Area is given normalized to the resource usage of a single PE. The results show a performance improvement of 1.54 for the heterogeneous system for any number of PEs, and at the same time, an area reduction of up to 22.5% in the case of a 3 PE system. We observe that the performance gain, and area reductions are in large parts due to removing instructions that implement a barrel shifter from the PEs. This confirms the findings of [6], that removing the shifter leads to large area savings.

6. CONCLUSION

In this paper, we presented a heterogeneous multi-core system for reconfigurable hardware consisting of application specific processing elements contained in ReconOS hardware threads. The processing elements are comprised of MIPS based soft core CPUs combined with their respective program. The soft CPUs have been optimized for their specific programs by removing unused instructions from the implemented instruction set (ISA subsetting). The resulting tight coupling of program code with custom processing hardware yields improved performance by increased clock frequencies as well as significant savings in hardware resources. We presented an automated toolflow that generates an application specific multi-core system from a multi-threaded software implementation. In order to adapt an application to our system, only minimal changes need to be made to the conventional software design flow, enabling the developer to benefit from a customized heterogeneous multi-core system for almost no additional overhead in development. We demonstrated the capabilities of our system by conducting a case study for a multi-threaded constraint solver. We evaluated the system's performance and resource usage for several points in design space, using up to 7 processing elements. Our findings show that for the constraint solver application we gain speedups of up to 1.54 accompanied by an area reduction of 22.5% to a non-customized homogeneous multi-core system.

Acknowledgment

This work is funded by the Research Council of Norway (grant 191156V30) and by the European Union (grant 257906).

7. REFERENCES

[1] A. Tumeo, M. Branca, L. Camerini, M. Ceriani, M. Monchiero, G. Palermo, F. Ferrandi, and D. Sciuto, "Prototyping pipelined applications on a heterogeneous FPGA multiprocessor virtual platform," in *2009 Asia and South Pacific Design Automation Conference*. IEEE, Jan. 2009.

[2] M. a. Kinsy, M. Pellauer, and S. Devadas, "Heracles: Fully Synthesizable Parameterized MIPS-Based Multicore System," in *2011 21st International Conference on Field Programmable Logic and Applications*. IEEE, Sep. 2011.

[3] B. Motruk, J. Diemer, R. Buchty, R. Ernst, and M. Berekovic, "IDAMC: A Many-Core Platform with Run-Time Monitoring for Mixed-Criticality," in *2012 IEEE 14th International Symposium on High-Assurance Systems Engineering*. IEEE, Oct. 2012.

[4] C. Galuzzi and K. Bertels, "The Instruction-Set Extension Problem," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 4, no. 2, May 2011.

[5] C. Banz, C. Dolar, F. Cholewa, and H. Blume, "Instruction set extension for high throughput disparity estimation in stereo image processing," in *ASAP 2011 - 22nd IEEE International Conference on Application-specific Systems, Architectures and Processors*. IEEE, Sep. 2011.

[6] P. Yiannacouras, J. G. Steffan, and J. Rose, "Exploration and Customization of FPGA-Based Soft Processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, Feb. 2007.

[7] E. Lubbers and M. Platzner, "ReconOS: An RTOS Supporting Hard-and Software Threads," in *2007 International Conference on Field Programmable Logic and Applications*. IEEE, Aug. 2007.

[8] D. Goodwin and D. Petkov, "Automatic generation of application specific processors," in *Proceedings of the international conference on Compilers, architectures and synthesis for embedded systems - CASES '03*. New York, New York, USA: ACM Press, 2003.

[9] A. Peymandoust, L. Pozzi, P. Ienne, and G. De Micheli, "Automatic instruction set extension and utilization for embedded processors," in *Proceedings IEEE International Conference on Application-Specific Systems, Architectures, and Processors. ASAP 2003.* IEEE Comput. Soc, 2003.