Beyond von Neumann: weakly programmable processor arrays and their programming

Andersson, Per; Kuchcinski, Krzysztof; Janneck, Jörn; Zhang, Chenxin

Published in:
[Host publication title missing]

2011

Link to publication

Citation for published version (APA):
Andersson, P., Kuchcinski, K., Janneck, J., & Zhang, C. (2011). Beyond von Neumann: weakly programmable processor arrays and their programming. In *[Host publication title missing]* swedsoft. http://www.swedsoft.se/

Total number of authors:
4

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Beyond von Neumann: weakly programmable processor arrays and their programming

P. Andersson, K. Kuchcinski, J. Janneck
Department of computer science
Lund University
P.O. Box 118
SE-221 00 Lund
Sweden
Email: Per.Andersson@cs.lth.se,
Krzysztof.Kuchcinski@cs.lth.se,
jorn.janneck@cs.lth.se

C. Zhang
Department of Electrical and Information
Technology
Lund University
P.O. Box 118
SE-221 00 Lund
Sweden
Email: Chenxin.Zhang@eit.lth.se

ABSTRACT

The age of parallelism is here. For a sustainable software development for massively parallel architectures the von Neumann model need to be replaced by one with native support for parallelism. We suggest a data flow model for signal processing applications. This will make it possible to reuse software implementations for different targets and future platform generations. We also outline our development tool flow for compiling CAL, a data flow language, to parallel architectures. We also present our processor array, which can be configured to handle massively parallel computations. We demonstrate its power by implementing part of a software radio receiver.

1. INTRODUCTION

When looking at the embedded system market. There are emerging applications which require 2-3 magnitudes of more processing power compared to todays systems, for example communication over a 3G radio uses 10 Gops while a 4G radio will need 1000 Gops. Increased computational power also enables new applications, for example phased array antennas for car radars which needs 10 Tops. To be competitive we need the hardware capable to deliver these computations. However hardware is not enough. These applications are increasing in complexity and to manage the development cost we also need to make sure the implementations are reusable, i.e. minimize the NRE.

Specialized hardware has traditionally played an important roll for high performance embedded systems. They are however becoming less important due to several reasons. 1) high NRE, the integration and verification of a hardware component is time consuming and costly for each new platform. 2) lack of flexibility, an accelerator can only handle one specific type of computations. 3) adoptability, the hardware is often developed before standards are fixed and also products are expected to manage new standards. Hardwired accelerators are not suitable for this event though reconfigurable hardware can help.

Processors combined with well designed software have proven to be a great solutions for manny problems. Software components are much less problematic to reuse, over time or for different hardware, compared to hardware. Modern processors provide sufficient computational power for many applications. A processor can take advantage of extra transistors to achieve parallelism in two ways 1) parallel instructions, issuing multiple instructions each clock cycle will give fine grains parallelism 2) multi-core, concurrent execution of multiple threads will provide coarse grained parallelism.

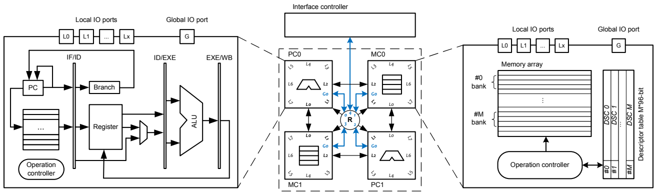

Figure 1: The WPPA architecture developed at EIT, Lund University.

Limited instruction level parallelism in the application and limited bus bandwidth in a multi core SMP architecture are challenges that must be addressed for increased processor performance.

Weakly Programmable Processors(WPPA) is an promising family of architectures which can provide massively parallel and programmable computational resources. Our WPPA developed at EIT, Lund University, proposed in [9], is constructed from a mesh of heterogeneous resource cells. The architecture not only supports data sharing between different processing elements, but also enables direct data transfers between memory cells without additional control logic. An example of the proposed cell array is shown in Figure 1.

The processing cell is a RISC core with support of SIMD alike operations. The communication I/O port registers are directly accessible in the same way as the general purpose registers. Hence, clock cycle penalties associated with memory data movements as found in traditional load-store architectures are eliminated. Each memory cell contains one memory array that may be divided into one or more memory banks. Each bank can be used as a random access memory, or as a FIFO by dynamic reconfiguration.

To demonstrate the flexibility and processing power of the proposed architecture, OFDM time synchronization have been implemented onto a 2-by-2 cell array, supporting three wireless radio standards (IEEE 802.n, 3GPP LTE, and DVB), and is capable of processing two concurrent data streams from any combination of the above 3 standards.

2. SOFTWARE

Software are becoming an increasingly important part of embedded systems. There are many reasons for this. In addition to the flexibility and possibility of update after deployment one major advantage of software is ease of reuse. Today a mobile phone contains millions of lines of code. Writing this amount of code for each new model or platform is not possible, but porting the existing code to a new target is manageable.

The enabling technology which makes it possible to port millions of lines of code to a new hardware platform is target independent high level languages. Well written software do not need to be changed for each new processor or even system architecture. The job of generating efficient executables are handled by compilers. One major reason the compilers can do a great job for different processors is that they have a common foundation, the Von Neumann model. Most processors behaves as if there is one global memory which is manipulated by a sequence of operations. This is also the view of processors for most software programmers. The Von Neumann is a common model for both hardware and software developers. It allows the software developer to describe the functionality of the application without using specific features of the target architecture. At the same time the von Neumann model gives the hardware architect freedom to chose which operations to implement, the compiler will generate code for the rest. The model also allows run-time optimizations, such as out of order execution and branch prediction to improve performance. The von Neumann model has liberated the software developer from the target hardware for single threaded applications.

The von Neumann model specifies mathematical computations, memory accesses and program control flow. Programs that uses other constructs, such as threads and graphical user interfaces, are hard and time consuming to port.

One of the problems facing todays software programmers is how to develop efficient programs for multi core processors. For this problem the von Neumann model will not help the programmer but rather makes things worse. The von Neumann model is inherently sequential and the program must be divide into several threads to generate efficient executables for multi core processors. There are some techniques for automating this, but in general the compiler can not do this efficiently. Therefore the problem has been moved to the programmer. The programer must divide the program into threads which uses synchronization mechanisms, such as semaphores or message passing, to ensure the correct functionality when the threads are depending on each other. It is known that developing threaded applications is much more time consuming and there are more bugs compared to single threaded applications.

3. SOLUTIONS FOR MANY CORES

Researches have been working on parallelizing compilers for languages derived from the von Neumann model for over 30 years. It is unlikely that this research will bring the improvements that are needed to generate efficient executables for tomorrows many core architectures. For this we need a radical change. The underlaying problem is that the target hardware has deviated to far from the von Neumann model for compilers to do a good job.

Today most WPPAs require special programs, either there is a special language or an API the programmer must use to develop efficient programs. This is an easy way to give the programmer access to the full potential of the architecture. However this will lead to problems in the future. The main obstacle will be that the code is no longer portable, it is locked to a target hardware. Furthermore it is likely not to be very efficient on the next generation of the same target platform leading to an explosion of software maintenance costs.

Adding constructs to expose hardware characteristics in a high level language leads to operations not known and analyzable by the compiler and will make optimizations impossible. For example, the EIT architecture [9] support communication between parallel task by both shared memory and point to point links. Which is best depends on the mapping of the tasks to the processing units. Not allowing the compiler to make the choice of communication implementation, i.e. shared memory or direct link will lead to inefficient compiled code.

Adding constructs and concept from an application domain to a language can simplify the work for the application developer. In contrast to extensions which expose hardware details to the programmer, these extensions can help the compiler to do a good job while preserving the portability of the implementation. Domain specific language extensions normally is a combination of constructs already present in the original language and can then be compiled to the intermediate format used by the compiler.

4. SOFTWARE DEVELOPMENT

Considering the limitations of parallelizing compilers, we believe it is necessary for parallelism to be explicit in the applications. Introducing parallelism in high level software languages is a specialization for the target, parallel hardware. These extensions can not be mapped to the intermediate model, the von Neumann model, and will cause problems for optimizing compilers. We believe all platforms in the near future will have multi or many cores. For a sustainable software development where code can be reused on different targets we believe it is necessary to replace the von Neumann model with one that allows concurrency and data synchronizations between threads. Finding a general model which is suitable for most programs is an attractive and ambitious goal. This is however beyond the goal of our current work. For the moment we are focussing on one application domain, namely DSP applications which include streaming and multimedia applications.

4.1 Data Flow Languages

Data Flow Languages have proven to be suitable for implementing signal processing, media and streaming applications. These applications are also among the ones requiring most computations in embedded systems. Programs written in data flow languages are by construct concurrent. Actors with local state communicate trough FIFO-queues. Actors fire concurrent and independent of each other. This model provides the compilers with a great flexibility to map the applications in an efficient way to a parallel processing architecture.

CAL is a data flow languages where actors have local state and communicate using unbounded FIFO-Queue [2]. The syntax allows an easy implementation of manny real world applications, for example the language supports guards and schedules for easy implementation of state machines.

CAL is standardized and is supported by for example Ericsson, the mpeg consortsium, and Xilinx. CAL is open source. The opendf.org compiler has backends for software [7] and hardware [4]. In our work we are developing a backend for WPPAs.

5. COMPILING CAL

We are developing a compiler for CAL in which we intend to use an *abstract actor machine* as intermediate representation. This representations makes it possible for the compiler to reason about static and dynamic ordering of communication, synchronization and scheduling decisions. We expect to generate code with performance equal to hand written C in more cases than current approaches for data flow compilation, i.e. non static DFG. The compiler will also analyze the code and generate suitable implementations for action scheduling, communication and thread synchronization.

5.1 Data Centric Mapping

In a WPPA the amount of local memory is limited. It is important to ensure that all data needed for a computation fit in the local memory. We intend to use data dependence analysis to guide the mapping and scheduling. Careful synchronization of producers and consumers in neighboring nodes will ensure minimal needs for data buffers and minimize communication overhead. We have published a technique to minimize the amount of temporary data in [1]. The idea is to analyze the data access patterns and schedule the producer and consumer thread in sync.

A general data flow program do not have the static behavior needed for the technique outlined above. However it is possible to detect static regions in a DFG and from this derive the constraints and parameters needed for mapping and scheduling [3].

5.2 Constraint Programming

Constraint programming is a technique, which is very powerful for optimization of complex system composed of many dependent parameters. It has proven efficient for, among others, high level synthesis, scheduling, and selection configurations in reconfigurable systems [8]. We have recently used our JaCoP solver, implemented in Java [5], to map programs into a run-time reconfigurable architecture [6]. In this work, we have used constraint programming to formalize the architecture model together with a specific application program. We have evaluated our approach using several multimedia applications from the Mediabench set. In 78% of cases, our system provides results that are proved optimal. Our current mapping problems are similar to the problems defined in the paper and we will use similar approach for mapping of actors to the WPPA.

6. SUMMARY

The age of multi-core is here and tomorrow we have many-core systems. In this paper we present our WPPA architecture. It provides the computational power and flexibility required by tomorrows systems. To demonstrate this we have implemented OFDM time synchronization for wireless radio. The implementation supports processing of two concurrent streams from any of the supported standards. Software is becoming an increasingly important part of embedded systems. In the paper we argue for the need to replace the von Neumann model with a model of computation that natively support parallelism. We suggest data flow languages and we present our approach for compiling CAL to our WPPA.

7. ACKNOWLEDGMENTS

This work is supported by VINNOVA trough the two industrial excellence centers EASE and SoS at Lund University. The work is also supported by SSF trough the HiPEC project and the ELLIIT center.

8. REFERENCES

[1] P. Andersson and K. Kuchcinski. Automatic local memory architecture generation for data reuse in custom data paths. In *International Conference on Engineering of Reconfigurable Systems and Algorithms*, June 21–24 2004.

[2] J. Eker and J. W. Janneck. Cal: Cal actor language. document edition 1, Electronics Research Laboratory, University of California at Berkeley, Tech. Rep. UCB/ERL M03/48, December 2003.

[3] R. Gu, J. W. Janneck, M. Raulet, and S. S. Bhattacharyya. Exploiting statically schedulable regions in dataflow programs. In *ICASSP*, pages 565–568. IEEE, 2009.

[4] J. W. Janneck, I. D. Miller, D. B. Parlour, G. Roquier, M. Wipliez, and M. Raulet. Synthesizing hardware from dataflow programs: An mpeg-4 simple profile decoder case study. In *Proceedings of the IEEE Workshop on Signal Processing Systems, SiPS 2008, October 8-10, 2008, Washington, D.C. Metro Area, USA*, pages 287–292, 2008.

[5] K. Kuchcinski and R. Szymanek. JaCoP Library. User's Guide. `http://www.jacop.eu`, 2009.

[6] E. Raffin, C. Wolinski, F. Charot, K. Kuchcinski, S. Guyetant, S. Chevobbe, and E. Casseau. Scheduling, binding and routing system for a run-time reconfigurable operator based multimedia architecture. In *Conference on Design and Architectures for Signal and Image Processing (DASIP)*, Edinburgh, United Kingdom, October, 26-28 2010.

[7] G. Roquier, M. Wipliez, M. Raulet, J. W. Janneck, I. D. Miller, and D. B. Parlour. Automatic software synthesis of dataflow program: An mpeg-4 simple profile decoder case study. In *Proceedings of the IEEE Workshop on Signal Processing Systems, SiPS 2008, October 8-10, 2008, Washington, D.C. Metro Area, USA*, pages 281–286, 2008.

[8] C. Wolinski, K. Kuchcinski, and E. Raffin. Automatic design of application-specific reconfigurable processor extensions with UPaK synthesis kernel. *ACM Trans. Des. Autom. Electron. Syst.*, 15(1):1–36, 2009.

[9] C. Zhang, T. Lenart, H. Svensson, and V. Öwall. Design of coarse-grained dynamically reconfigurable architecture for dsp applications. In *Proceedings of International Conference on ReConFig*, pages 338–343, Dec. 2009.