A Design Framework for Asynchronous Multithreaded Architectures

D. K. Arvind, J. Hossell, A. Koppe, T. Lewis, R. Rangaswami, J. Schneiders, A. Stanley and S. Zhong
Institute for Computing Systems Architecture, University of Edinburgh,
Mayfield Road, Edinburgh EH9 3JZ, Scotland.
Email: dka@dcs.ed.ac.uk

Abstract

This paper outlines a framework for the design of integration platforms for high-performance, low-power programmable systems for embedded applications. We seek high performance by exploiting concurrency in the cooperating threads in the application, and low power consumption in the asynchronous implementation of the multithreaded platforms. Results are presented for the estimation of power consumption and speed of simple C benchmark programs executing on a micronet-based multithreaded architecture which exercises the design flow in the framework.

1 The Design Framework

This framework (see Figure 1) addresses the challenge of designing programmable, high-performance and low-power embedded systems. The system is organised as a set of cooperating threads of execution which receive, analyse and act on information. It is assumed that the system is programmed in a combination of Java and C programming languages.

Java's universal executable format is less useful for embedded systems in which the Instruction Set Architecture is known. Rather, execution by interpretation is more likely to impose a penalty which is unacceptable for time-critical applications. The approach in this paper advocates compiling the Java program to architectures which execute threads concurrently [AHKR01]. Such an architecture should also support both loop- and instruction-level parallelism, and inter-thread communication and synchronisation.

The programs are first converted to an intermediate representation called the Static Single Assignment (SSA) form. The dependences in the code are explicitly represented, which is suitable for analysis and optimisation for multithreading. The *Thread Analyser* examines program structures such as loops, conditionals and method calls for sources of concurrent execution. The resulting code has annotations at points in the program where threads are likely to be sparked. This is optimised in an iterative fashion using a *Cost Analyser* for both speed of execution and power consumption based on target architectural models. The final stage is the generation of assembly code for both the Java and C programs which is executed on an instruction set simulator of the multi-threaded target.

The architecture is based on the micronet model [Arv94], in which a system is viewed recursively as networks of entities which compute concurrently and communicate asynchronously. The architecture is modelled at different levels of abtraction: at the level of thread processing units with gross values for computation and communication latencies and power costs, down to the handshaking protocols of the micronet architecture which reflect accurate costs for both speed and power. The framework is able to map C or Java application programs, through the intermediate representations and assembly code, to the handshaking signals which are excited when executed on the micronet architecture. Quantifiable changes in the speed and power consumption can be affected by code optimisations at the higher levels in the design framework.

A feature of the framework is the clean separation between target-independent and target-specific issues in multithreaded compilation, which enables the comparative study of different multi-threaded models of computation.

2 Micronet-based Multithreaded Architecture

The multithreaded architecture consists of a network of Thread Processing Units (TPUs), Data and Instruction memories and a First-level Scheduler, as shown in Figure 2. Each TPU in turn is a network of registers, functional units such as ALU and Memory Unit, Instruction Buffer, Communication and Thread Issue Units. The First-level Scheduler distributes chunks of instructions to each TPU which are then held in the Instruction Buffer for issue by the Thread Issue Unit. The Communication Unit handles

the interaction between the TPU and the First-level Scheduler, and with the other TPUs.
The standard MIPS instruction set is augmented with six additional instructions to support multi-threading. These include the following:

Fork - This executes a new thread on a different TPU whose identifier is passed as an argument, along with the label at which the code has to begin execution.

Stop - This terminates a thread and resets the TPU.

PassSignal - This is used to direct a particular word to another specified TPU.

Wait - This is used in conjunction with the previous instruction and specifies the destination register for the word.

InterruptSignal - This interrupts the thread running at the target TPU, and saves state to resume the interrupted thread on completion of the interrupt sequence.

MoveOldProgramCounter - This resumes the execution of the thread interrupted by the previous instruction

3 Results

The models of the micronet architecture capture the behaviour, timing and interaction of the constituent parts. The duration, nature and pattern of these activities give an approximation of the power consumption in the architecture. Events such as Register, Memory and Bus Read/Write, and ALU instructions are given a normalised power cost and the power consumption for a program running on the architecture is estimated. The overall power consumption for a 4X4 matrix multiplication program written in C and running on the multithreaded target is shown in Figure 3. For the same program, the power consumption in the individual Thread Processing Units are shown in Figures 4-7. TPU0 initialises three matrices and then spawns three new threads in TPU1-3. This is refected in the activity in TPU0 in the first half (and by the absence of any in the rest of TPUs).

4 Conclusion

This paper has given a brief outline of a framework for the design of high-performance, low-power programmable integration platforms. Results have been demonstrated for simple benchmark programs which exercise the different stages in the framework. Future work will investigate an apt level for integrating the C and Java strands which currently run in parallel, although still within the same framework. Multithreaded asynchronous architectures which combine soft-programmable (realised as Instruction Set Architectures), and hard-programmable (realised as FPGAs) elements are currently being investigated. The results in the paper are based on a version of the simulation platform which runs on a single Linux machine. A distributed simulation platform based on the Area Virtual Time protocol [AS01] is being implemented on a 16-node Beowulf cluster dedicated to architectural simulation. This will provide a sandbox for an integrated approach to the design of programmable asynchronous systems which considers the interplay between the compiler and the architecture for optimising the performance of practical embedded systems.

Acknowlegement

All the authors wish to acknowledge the Scottish Higher Educational Funding Council grant No. 85 entitled *Design Clusters for System Level Integration* for equipment grant. Some of the authors wish to acknowledge partial funding from research grants from the following organisations: ARM Ltd., Cambridge, Hitachi America Ltd., San Jose, Hitachi MicroSystems Europe, Maidenhead, Sharp Laboratories Europe, Oxford, Triscend Corp., Mountain View, and Xilinx Inc., San Jose.

References

[Arv94] Arvind, D. K. and Rebello, V. E. F. 1994. “Instruction-level Parallelism in Asynchronous Processor Architectures”, In Proc. of the Third International Workshop of Algorithms and Parallel VLSI Architectures, Elsevier, Leuven, Belgium, August 1994.

[AHKR01] Arvind, D. K., Hossell, J., Koppe, A., and Rangaswami, R. 2001. “Java Compilation for Multithreaded Architectures”, In Proc. 9th Workshop on Compilers for Parallel Computers, Edinburgh, Scotland, June 2001.

[AS01] Arvind, D. K. and Schneiders, J. 2001. “Area Virtual Time”, In Proc. 5th IEEE Int. Workshop on Distributed Simulation and Real-Time Applications, IEEE Press, Cincinnati, Ohio, August 2001.

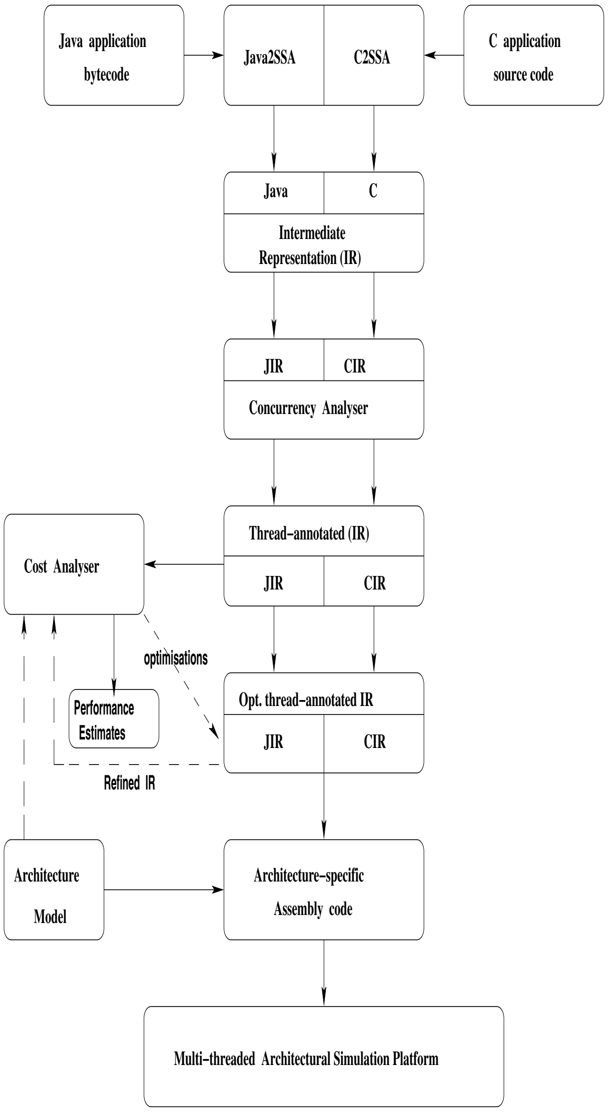

Figure 1. The Design Framework

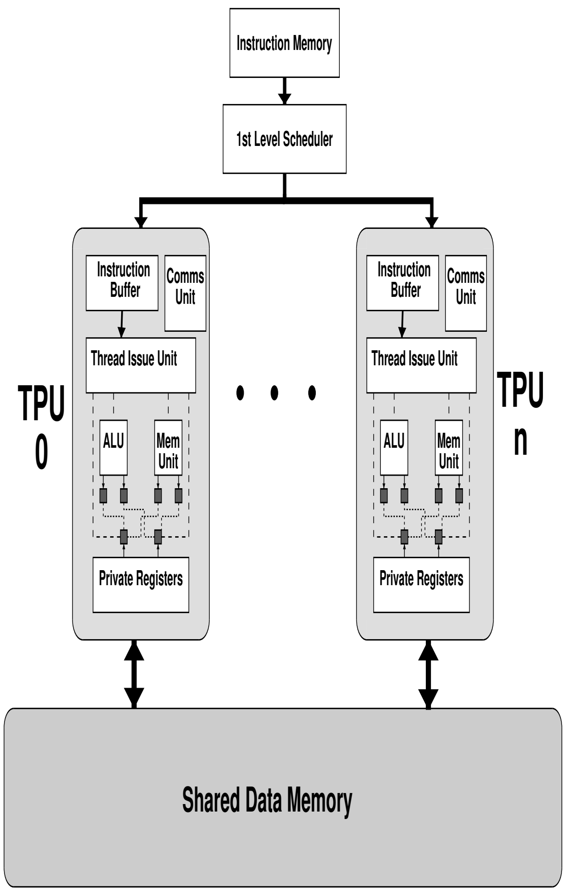

Figure 2. Micronet-based Multi-threaded Architecture

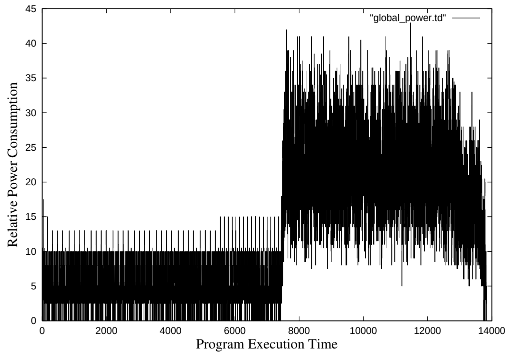

Figure 3. Overall Power Consumption and Execution Time for Matrix Multiplication in the Multithreaded Architecture

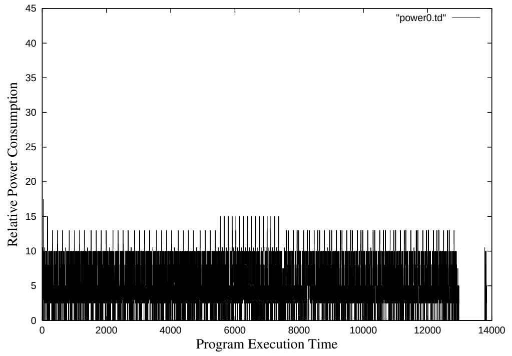

Figure 4. Power Consumption and Execution Time in TPU0

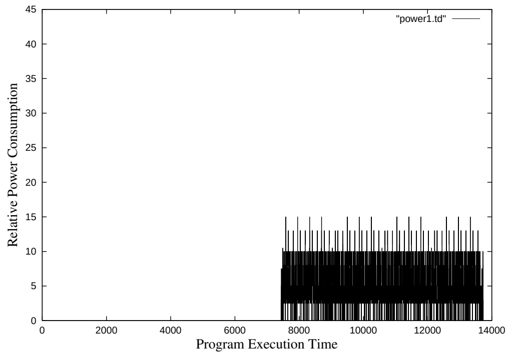

Figure 5. Power Consumption and Execution Time in TPU1

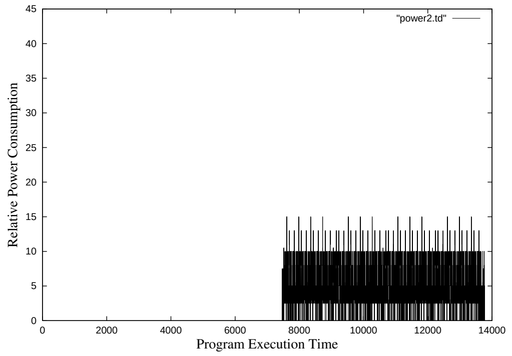

Figure 6. Power Consumption and Execution Time in TPU2

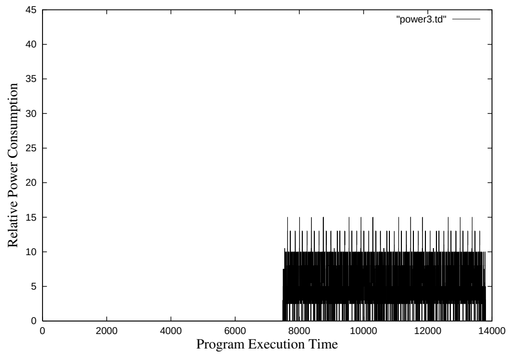

Figure 7. Power Consumption and Execution Time in TPU3