Journal of Universal Computer Science, vol. 6, no. 10 (2000), 906-907
submitted: 2/10/00, accepted: 10/10/00, appeared: 28/10/00

J.UCS Special Issue on Multithreaded Processors and Chip-Multiprocessors

Jörg Keller
(FernUniversität Hagen, Germany
joerg.keller@fernuni-hagen.de)

Theo Ungerer
(Universität Karlsruhe, Germany
ungerer@informatik.uni-karlsruhe.de)

Today's superscalar processors are able to issue up to six instructions per cycle from a single sequential instruction stream. VLSI technology will soon allow future microprocessors to issue and execute eight or more instructions per cycle. However, instruction level parallelism (ILP) found in a conventional instruction stream is limited. Recent studies show the limits of processor utilization even in today's superscalar microprocessors reporting instructions per cycle (IPC) values between 0.14 and 1.9. One solution to increase performance is an additional utilization of more coarse-grained parallelism either by integrating two or more complete processors on a single chip or by using a multithreaded approach.

A multithreaded processor is able to pursue multiple threads of control in parallel within the processor pipeline. The functional units are multiplexed between the thread contexts. Most approaches store the thread contexts in different register sets on the processor chip. Latencies are masked by switching to another thread. A fine-grained multithreaded processor interleaves execution of instructions of different threads on a cycle-by-cycle basis, whereas a block-multithreaded processor executes instructions of a single thread until a context-switching event, e.g. a cache miss, occurs. Moreover, a simultaneous multithreaded (SMT) processor issues instructions of several threads simultaneously. It combines a wide-issue superscalar processor with multithreading.

The importance of multithreaded execution to both the research and microprocessor industries is rapidly increasing, as can be seen by the increasing amount of research papers and by recent announcements of the computer industry, in particular, IBM's Power4 with two processors on a die, the 4-threaded SMT Alpha processor 21464 of Compaq, and the MAJC-5200 processor of Sun which features two 4-threaded processors on a single die.

Chip-multiprocessors and multithreaded processors are able to boost performance of a multithreaded program mix, i.e. programmer-visible or compiler-generated instruction sequences, operating system threads or even whole processes. Another more recent research trend targets the performance increase of single-threaded programs by dynamically utilizing speculative thread-level parallelism. Sequences of instructions are dynamically extracted from sequential binaries and speculatively executed by different processing elements or in multiple thread slots within a single processor. In case of misspeculation, the results of the speculative thread and of subsequent threads are discarded. Codrescu and Wills investigate different dynamic partitioning schemes, in particular, thread-generation by dynamically parallelizing loop iterations, procedure calls, or using fixed instruction length blocks. A new, more flexible algorithm – called MEM-slicing algorithm - is proposed that generates a thread starting from a slice instruction up to a maximum thread length. All approaches are evaluated in context of the Atlas chip-multiprocessor.

Gopinath and Narasimhan M.K. investigate the performance of switch blocking where waiting threads are disabled and signaled at completion of the wait vs. switch-spinning where waiting threads poll and execute in round-robin fashion in context of a block-multithreaded processor.

A root of multithreaded execution is the coarse-grain dataflow execution model that relies on non-blocking threads generated from single-assignment dataflow programs. Threads start execution as soon as all operands are available. Such threads may be generated with the aim to decouple memory accesses from execute instructions. Kavi, Arul and Giorgi present a decoupled scheduled dataflow architecture where a (dataflow) program is compiler-partitioned into execution and memory-access threads and executed on a decoupled dataflow machine.

Beyls and D'Hollander present a technique to generate at compile-time computation threads and data-fetch threads which ensure that the computation thread does not experience cache misses. Li and Jenq theoretically investigate the thread scheduling problem that deals with the compile-time schedule of a data dependency graph on a multithreaded architecture.

Evripidou and Kyriacou propose Networks of Workstations as basis for multithreaded program execution. The hardware implementation of a thread synchronization unit to coordinate such workstations is presented. The overall basis is a decoupled dataflow architecture, where the thread synchronization unit is able to schedule the execution threads on the different workstations as processing elements.

However, multithreading may also be applied for event handling due to its fast context switching ability. Metzner and Niehaus propose the use of multithreaded processors for real-time event handling. Several block-multithreaded MSparc processors are supervised by an external thread scheduler called EVENTS, that assigns computation-intensive real-time threads to the different MSparc processors.

The focus of the special issue thus ranges from execution and performance models of multithreaded processors and chip multiprocessors to speculative multithreading, compiler interaction and event handling by multithreading. The seven papers in this issue represent a broad spectrum of activities within the field. We hope you enjoy the selection.