Identifying Bottlenecks in a Multithreaded Superscalar Microprocessor

Ulrich Sigmund[1] and Theo Ungerer[2]

[1] VIONA Development GmbH, Karlstr. 27, D-76133 Karlsruhe, Germany
[2] University of Karlsruhe, Dept. of Computer Design and Fault Tolerance, D-76128 Karlsruhe, Germany

Abstract. This paper presents a multithreaded superscalar processor that permits several threads to issue instructions to the execution units of a wide superscalar processor in a single cycle. Instructions can simultaneously be issued from up to 8 threads with a total issue bandwidth of 8 instructions per cycle. Our results show that the 8-threaded 8-issue processor reaches a throughput of 4.2 instructions per cycle.

1 Introduction

Current microprocessors utilize instruction-level parallelism by a deep processor pipeline and by the superscalar technique that issues up to four instructions per cycle from a single thread. VLSI-technology will allow future generations of microprocessors to exploit instruction-level parallelism up to 8 instructions per cycle, or more. However, the instruction-level parallelism found in a conventional instruction stream is limited.

The solution is the additional utilization of more coarse-grained parallelism. The main approaches are the multiprocessor chip and the multithreaded processor. The multiprocessor chip integrates two or more complete processors on a single chip. Therefore every unit of a processor is duplicated and used independently of its copies on the chip. In contrast, the multithreaded processor stores multiple contexts in different register sets on the chip. The functional units are multiplexed between the threads that are loaded in the register sets. Multithreaded processors tolerate memory latencies by overlapping the long-latency operations of one thread with the execution of other threads - in contrast to the multiprocessor chip approach. Simultaneous multithreading [1] combines a wide issue superscalar instruction dispatch with multithreading. Instructions are simultaneously issued from several instruction queues. Therefore the issue slots of a wide issue processor can be filled by operations of several threads.

While the simultaneous multithreading approach surveys enhancements of the Alpha 21164 processor, our multithreaded superscalar approach is based on a simplified PowerPC 604 processor [2]. Both approaches, however, are similar in their instruction issuing policy. This paper focuses on the identification and avoidance of bottlenecks in the multithreaded superscalar processor. Further simulations, shown in [3], install the same base processor in a multiprocessor chip and compare it with the multithreaded superscalar approach.

2 The Multithreaded Superscalar Processor Model

Our multithreaded superscalar processor uses various kinds of modern microarchitecture techniques as e.g. branch prediction, in-order dispatch, independent execution units with reservation stations, rename registers, out-of-order execution, and in-order completion. We apply the full instruction pipeline of the PowerPC 604, and extend it to employ multithreading. However, we simplify the instruction set (using an extended DLX [4] instruction set instead), use static instead of dynamic branch prediction, and renounce the floating point unit. The processor model is designed scalable: the number of parallel threads, the sizes of internal buffers, register sets and caches, the number and type of execution units are not limited by any architectural specification.

We conducted a software simulation evaluating various configurations of the multithreaded superscalar processor. For the simulation results presented below we used separate 8 KByte 4-way set-associative data and instruction caches with 32 Byte cache lines, a cache fill burst rate of 4-2-2-2, a fully-associative 32-entry branch target address cache, 32 general purpose registers per thread, 64 rename registers, a 12-entry completion queue, 4 simple integer units, single complex integer, load/store, branch, and thread control units, each execution unit with a 4-entry reservation station. The number of simultaneously fetched instructions and the sizes of fetch and dispatch buffers are adjusted to the issue bandwidth. We vary the issue bandwidth and the number of hosted threads in each case from 1 up to 8, according to the total number of execution units.

We choose a multithreaded simulation work load that represents general purpose programs without floating-point instructions for a typical register window architecture. We assume 63.8% integer, 1.1% complex integer, 13.2% load, 7.0% store, 10.8% branch, and 4.1% thread control instructions.

3 Performance Results

The simulation results in Fig. 1 (left) show that the single-threaded 8-issue superscalar processor throughput (measured in average instructions per cycle) only reaches a performance of 1.14 executed instructions per cycle. The four-issue approach is slightly better with 1.28, due to instruction cache thrashing in the 8-issue case.

Increasing the number of threads from which instructions are simultaneously issued to the 8 issue slots also increases performance. The throughput reaches a plateau at about 3.2 instructions per cycle, where neither increasing the number of threads nor the issue bandwidth significantly raises the number of executed instructions. When issue bandwidth is kept small (1 to 4 instructions per cycle), and four to eight threads are regarded, we expect a full exploitation of the issue bandwidth. As seen in Fig. 1(left), however, the issue bandwidth is only utilized by about 75%. Even a highly multithreaded processor seems unable to fully exploit the issue bandwidth. A further analysis reveales the single fetch and decode units as bottlenecks, leading to starvation of the dispatch unit.

Therefore we apply two independent fetch and two decode units, the simulation results are shown in Fig. 1(right). The gradients of the graphs representing the multithreaded approaches are much steeper, indicating an increased throughput. The processor throughput in an 8-threaded 8-issue processor is about four times higher than in the single threaded 8-issue case. However, the range of linear gain, where the number of executed instructions equals the issue bandwidth, ends at an issue bandwidth of about four instructions per cycle. The throughput reaches a plateau at about 4.2 instructions per cycle, where neither increasing the number of threads nor the issue bandwidth significantly raises the number of executed instructions. Moreover, the diagram on the right side in Fig. 1 shows a marginal gain in instruction throughput, when we advance from a 4- to an 8-issue bandwidth. This marginal gain is nearly independent of the number of threads in the multithreaded processor.

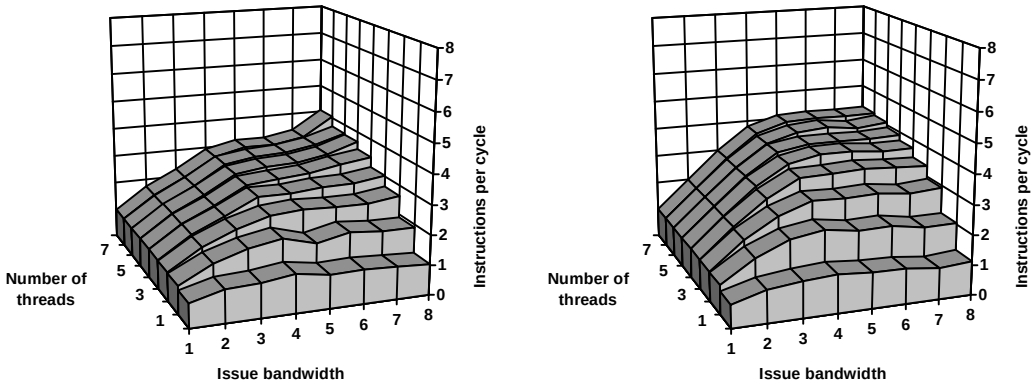

Fig. 1. Average instruction throughput per processor with one fetch and one decode unit and with with two fetch and two decode units

Further simulations (see Fig. 2) showed that a performance increase is not yielded when the number of slots in the completion queue is increased over the 16 slots assumed in the simulations above. Instruction execution is limited by true data dependencies and by control dependencies that cannot be removed. Also four write-back ports and 16 rename registers seem appropriate.

The load/store unit and the memory subsystem remain as the main bottleneck that may potentially be removed by a different configuration. The load/store unit is limited to the execution of a single instruction per cycle. Duplication of the load/store unit definitely increases performance. However, two or more load/store units that access a single data cache are difficult to implement because of consistency and thrashing problems. Using an instruction mix with 20.2% load and store instructions potentially allows a processor throughput of five instead

of the measured 4.2 instructions per cycle with a single load/store unit.

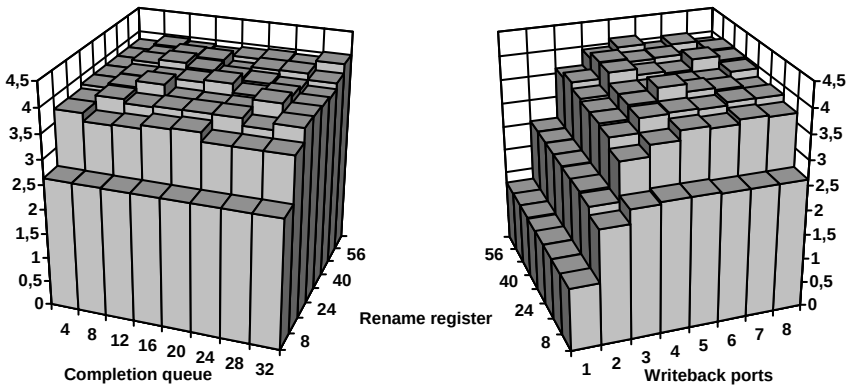

Fig. 2. Sources of unused issue slots

To evaluate if a different memory configuration might increase the throughput, we simulated different cache sizes, cache line sizes (from 8 to 128 bytes), cache schemes (direct mapped, set associative), workloads and numbers of active threads. The simulations show that our simulated processor is able to completely hide all latencies caused by cache refills (4-2-2-2) by its multithreaded execution model. The multithreaded superscalar processor reaches the maximum throughput that is possible with a single load/store unit. Penalties caused by data cache misses are responsible for the difference to the theoretical maximum throughput of five instructions per cycle.

4 Conclusion

This paper surveyed bottlenecks in a multithreaded superscalar processor, based on the PowerPC 604 microarchitecture, taking various configurations into consideration. Using an instruction mix with 20.2% load and store instructions the performance results show for an 8-issue processor with four to eight threads that two instruction fetch and two decode units, four integer units, 16 rename registers, four register ports, and a completion queue with 12 slots are sufficient. The single load/store unit proves as the principal bottleneck because it cannot easily be duplicated. The multithreaded superscalar processor (8-threaded 8-issue) is able to completely hide latencies caused by 4-2-2-2 burst cache refills. It reaches the maximum throughput of 4.2 instructions per cycle that is possible with a single load/store unit.

References

1. Tullsen, D. E. , Eggers, S. J., Levy, H. M.: Simultaneous Multithreading: Maximizing On-Chip Parallelism. The 22nd Ann. Int. Symp. on Comp. Arch. (1995) 392–403
2. Song, S. P. , Denman, M.,Chang, J.: The PowerPC 604 RISC Microprocessor. IEEE Micro, Vol. 14, No. 5 (1994) 8–17
3. Sigmund, U., Ungerer, Th.: Evaluating a Multithreaded Superscalar Microprocessor vs. a Multiprocessor Chip. 4th PASA Workshop, Juelich, World Sc. Publ. (1996)
4. Hennessy, J. L., Patterson, D. A.: Computer Architecture a Quantitative Approach, San Mateo (1996)

This article was processed using the LaTeX macro package with LLNCS style