EVALUATING A MULTITHREADED SUPERSCALAR MICROPROCESSOR VERSUS A MULTIPROCESSOR CHIP

T. UNGERER

Dept. of Computer Design and Fault Tolerance, D-76128 Karlsruhe, Germany
Phone: +49 721 608–6048 Fax +49 721 370455
ungerer@Informatik.Uni-Karlsruhe.de

U. SIGMUND

VIONA Development GmbH, Karlstr. 27, D-76133 Karlsruhe, Germany
Phone: +49 721 913440 Fax: +49 721 9134499
uli@viona.de

This paper examines implementation techniques for future generations of microprocessors. While the wide superscalar approach, which issues 8 and more instructions per cycle from a single thread, fails to yield a satisfying performance, its combination with techniques that utilize more coarse-grained parallelism is very promising. These techniques are multithreading and multiprocessing. Multithreaded superscalar permits several threads to issue instructions to the execution units of a wide superscalar processor in a single cycle. Multiprocessing integrates two or more superscalar processors on a single chip. Our results show that the 8-threaded 8-issue superscalar processor reaches a performance of 4.19 executed instructions per cycle. Using the same number of threads, the multiprocessor chip reaches a higher throughput than the multithreaded superscalar approach. However, if chip costs are taken into consideration, a 4-threaded 4-issue superscalar processor outperforms a multiprocessor chip built from single-threaded processors by a factor of 1.8 in performance/cost relation.

1 Introduction

Current microprocessors utilize instruction-level parallelism by a deep processor pipeline and by the superscalar instruction issue technique. DEC Alpha 21164, PowerPC 604 and 620, MIPS R10000, Sun UltraSparc and HP PA-8000 issue 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. Possible techniques are a wide superscalar approach (IBM power2 processor is a 6-issue superscalar processor), the VLIW-approach, the SIMD approach within a processor as in the

HP PA-7100LC, and the CISC-approach, where a single instruction is dynamically split into its RISC particles, as in the AMD K5 or the Intel PentiumPro. However, the instruction-level parallelism found in a conventional instruction stream is limited. Recent studies show the limits of processor utilization even of today's superscalar microprocessors. Using the SPEC92 benchmark suite, the PowerPC 620 shows an execution of 0.96 to 1.77 instructions per cycle [1], and even an 8-issue Alpha processor will fail to sustain 1.5 instructions per cycle [2].

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. For example, the Texas Instruments TMS320C80 Multimedia Video Processor [3] integrates four digital signal processors and a scalar RISC processor on a single 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. Depending on the specific multithreaded processor design, only a single instruction pipeline is used, or a single dispatch unit issues instructions from different instruction buffers simultaneously. Because of the multiple register sets, context switching is very fast. 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.

While the multiprocessor chip is easier to implement, use of multithreading in addition to a wide issue bandwidth is a promising approach. Several approaches of multithreaded processors exist in commercial and in research machines:

- The cycle-by-cycle interleaving approach, exemplified by the Denelcor HEP [4] and the Tera processor [5] switches contexts each cycle. Because only a single instruction per context is allowed in the pipeline, the single thread performance is extremely poor.
- The block-interleaving approach, exemplified by the MIT Sparcle processor [6], executes a single thread until it reaches a long-latency operation, such as a remote cache miss or a failed synchronization, at which point it switches to another context. The Rhamma processor [7] switches contexts whenever a load, store or synchronization operation is discovered.

- The simultaneous multithreading approach [2] combines a wide issue superscalar instruction dispatch with the multiple context approach by providing several register sets on the microprocessor and issuing instructions from several instruction queues simultaneously. Therefore the issue slots of a wide issue processor can be filled by operations of several threads. Latencies occurring in the execution of single threads are bridged by issuing operations of the remaining threads loaded on the processor. In principle the full issue bandwidth can be utilized.

While the simultaneous multithreading approach [2] surveys enhancements of the Alpha 21164 processor, our multithreaded superscalar approach is based on the PowerPC 604 [8]. Both approaches, however, are similar in their instruction issuing policy. We simulate the full instruction pipeline of the PowerPC 604, and extend it to employ multithreading. However, we simplify the instruction set (using an extended DLX [9] instruction set instead), use static instead of dynamic branch prediction, and renounce the floating point unit. We install the same base processor in our multiprocessor chip simulations to guarantee a fair comparison with the multithreaded superscalar approach.

2 Evaluation Methodology

2.1 The Superscalar Base Processor

Our superscalar base processor (see Fig. 1) implements the six-stage instruction pipeline of the PowerPC 604 processor (fetch, decode, dispatch, execute, complete, and write-back).

The processor uses various kinds of modern microarchitecture techniques as e.g. separate code and data caches, branch target address cache, static branch prediction, in-order dispatch, independent execution units with reservation stations, rename registers, out-of-order execution, and in-order completion.

The fetch and decode units always work on a continuous block of instructions. These blocks may not always be the maximum size, as there are limitations by the cache size (instruction fetch cannot overlap cache lines) and by branches that are predicted to be taken. As block size we choose the number of instructions that could be issued simultaneously.

The dispatch unit is restricted by the maximum issue bandwidth - the maximum number of instructions that can be issued simultaneously to the execution

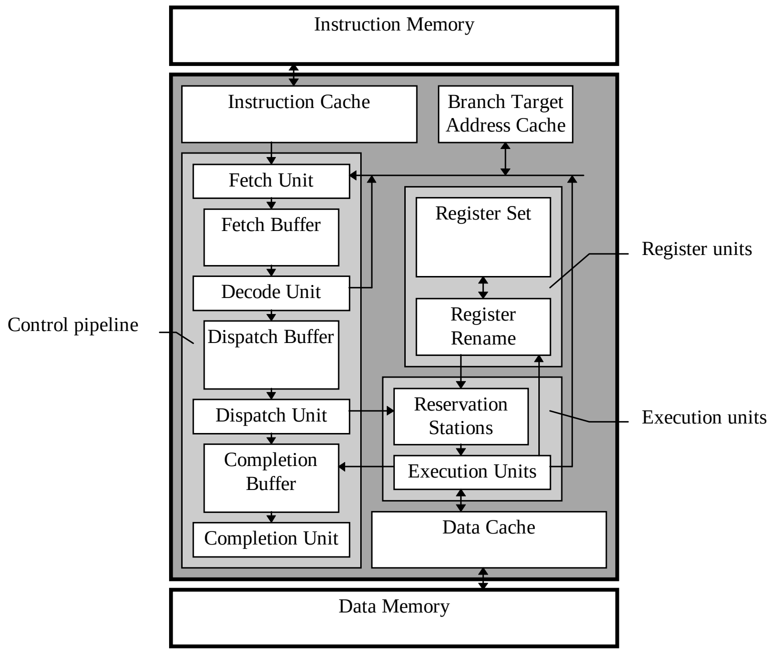

Figure 1: Superscalar architecture

units. The maximum issue bandwidth of a processor is mainly limited by the restricted number of busses from the rename registers to the execution units, and not by the instruction selection matrix. Due to the issuing policy of the PowerPC 604, executions are always issued in program order to the reservation stations of the execution units. The instructions are executed out-of-order by the execution units. The use of separate reservation stations for the execution units simplifies the dispatch, because only the lack of instructions, the lack of resources and the maximum bandwidth limit the simultaneous instruction issue rate. Data dependencies are taken care of by rename registers and reservation station. Instructions can be dispatched to the reservation stations without checking for control or data dependencies. An instruction will not be executed until all source operands are available.

All standard integer operations are executed in a single cycle by the simple integer units. Only the multiply and the divide instructions are executed in a complex integer unit. The execution of multiply instructions is fully pipelined, and consumes a specified number of cycles. The integer divide is not pipelined, its latency may also be specified in the simulator.

The branch unit executes one branch instruction per cycle. Branch prediction starts in the fetch unit using a branch target address cache. A simple static branch prediction technique is applied in the decode unit. Each forward branch is predicted as "not taken", each backward branch as "taken". A static branch prediction simplifies processor design, but reduces the prediction accuracy.

Completion is controlled by the completion unit, which retires instructions in program order with the same maximum rate as the maximum issue bandwidth. When an instruction is retired, its result is copied from the rename register to its register in the register set. The rename register and the slot in the completion buffer are released.

The memory interface is defined as a standard DRAM interface with configurable burst sizes and delays, to simulate advanced RAM types like SDRAM or EDO. All caches are highly configurable to test penalties due to cache thrashing caused by multiple threads.

The processor is designed scalable: the number of execution units and the size of buffers are not limited by any architectural specification. This allows experimentation to find an optimized configuration for an actual processor depending on chip size and expected application load.

2.2 Multithreaded Superscalar Base Processor

While the multiprocessor chip simply comprises two or more base processors of a specific issue bandwidth, the multithreaded superscalar approach is more complicated. In the multithreaded superscalar processor (see Fig. 2), the buffers of the control pipeline (fetch buffer, dispatch buffer, and completion buffer) and the register set are duplicated according to the number of hosted threads. Each thread has its own set of buffers and registers, thus running logically independent of the other threads. The processor is designed scalably with respect to the number of hosted threads. To the user of a multithreaded superscalar processor, the machine behaves like a multiprocessor. Each thread executes in its own context and is not affected by other threads.

Since the fetch and decode unit only work on a single thread per cycle, they may also be duplicated to gain a higher throughput. Instructions are still fetched and decoded in blocks of contiguous instructions. There is only a

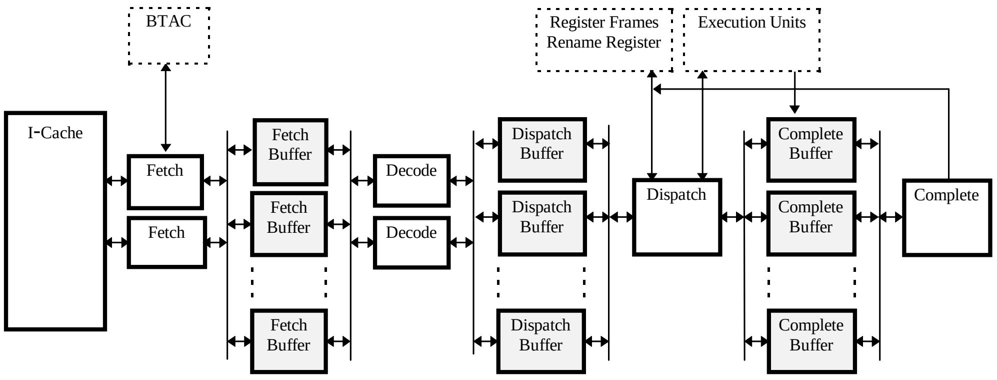

Figure 2: Multithreaded control pipeline

single dispatch and a single completion unit. The dispatch unit simultaneously selects instructions from all independent dispatch buffers up to its maximum issue bandwidth. The completion unit simultaneously retires any number of instructions of any thread (up to a total maximum of retired instructions per cycle).

The dispatch unit is not restricted with respect to the number of instruction issues according to each thread. There is no fixed allocation between threads and execution units in the multithreaded superscalar processor - in contrast to the multiprocessor chip.

The rename registers, the branch target address cache, the data and the instruction cache are shared by all active threads. There is no fixed allocation of any of these resources to specific threads. This allows for maximum performance with any number of active threads.

Each thread executes in a separate register set. The contents of the registers of a register set describe the state of a thread and form a so-called activation frame. An activation frame is called active if the thread is currently executed by the processor, i.e., the activation frame is represented in a register set. In addition to the active activation frames in the processor, we provide an activation frame cache, which holds activation frames that are not currently scheduled for execution. The activation frames in the activation frame cache and in the register sets can be interchanged without a significant penalty. By adding a cache of not currently running threads, the machine can be virtualized to an arbitrary number of threads. When a thread performs an instruction with long latency, like an external synchronization, the thread can be preempted by a "ready" thread in the cache.

The activation frame cache is also used to implement a derivation of a register window technique. A new activation frame is created for every subprogram activation, thereby significantly reducing the number of memory accesses to the data cache.

The execution units are expanded by a thread control unit that is responsible for creation and deletion of threads, for synchronization and communication between threads. Except for the load-/store and the thread control unit, each kind of execution unit may be arbitrarily duplicated.

Within a multithreaded processor latencies are almost all covered by other threads, so the penalty created by a static branch prediction should not affect average executions per cycle.

2.3 Simulator and Application Workload

Starting with the superscalar base processor we conducted a software simulation [10], evaluating various configurations of the multithreaded superscalar approach and of the multiprocessor chip models. All functional units were simulated with correct cycle behaviour. We employed an instruction-driven simulator, not a code tracer, so all execution related effects are simulated. The simulator is either script-driven or interactive. All functional units can be observed during the interactive execution in separate windows, which yields the ability to investigate all run-time effects in detail. In script-driven mode, versatile scripts can be defined to build up more complex test runs. Various simulation results are collected for all units of the processor(s) and for each thread. These reports can be automatically evaluated to create combined results for several tests.

The simulation workload is generated by a configurable workload generator, which creates random high level programs, and compiles them to our machine language. The distribution of the machine instructions is similar to that generated from high level programs. This approach allows us to create workloads for different types of programs without the need for a complete compiler.

3 Performance Results

For the performance results, presented in this section, we choose a multithreaded simulation work load that represents general purpose programs without floating-point instructions for a typical register window architecture.

Instruction type	Average use
Integer	63.8%
Complex-integer	1.1%
Load	13.2%
Store	7.0%
Branch	10.8%
Threadcontrol	4.1%

With a single load/store unit (used in our approach) the chosen work load has a theoretical maximum throughput of about five instructions per cycle (the frequency of load and store instructions sums up to 20.2 %).

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 buffer, up to 4 simple integer units, single complex integer, load/store, branch, and thread control units, each execution unit with a 4-entry reservation station. For the multithreaded approach we used two fetch units and two decode units. 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.

The simulation results in Fig. 3 show that the single-threaded 8-issue superscalar processor throughput 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.

We also see in Fig. 3 that 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. 3 shows a marginal gain in

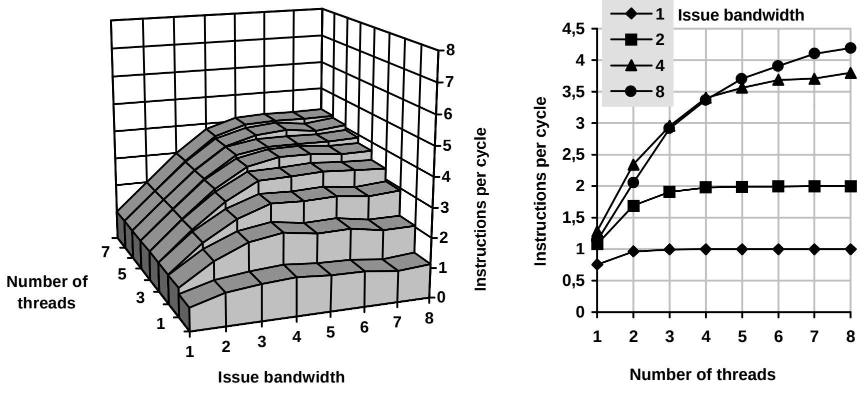

Figure 3: Average instruction throughput per processor

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.

Further simulations, shown in [11], identify the single load/store unit as a principal bottleneck. The difference between the expected five instructions per cycle and the simulation result of 4.2 is due to bubbles in the load/store pipeline caused by data cache misses. Our multithreaded superscalar approach reaches the maximum throughput that is possible with a single load/store unit.

To compare the multithreaded approach with a multiprocessor solution, we show in Fig. 4 the results normalized in relation to the number of threads (the number of instructions per cycle divided by the number of threads). It allows the comparison of parallel systems built up from basic multithreaded or single-threaded processors. Fig. 4 also shows that a multiprocessor built of single-threaded superscalar processors delivers the highest average instruction throughput per thread (note: the throughput per processor is shown in Fig. 3). Each step to more parallel threads delivers less average throughput per thread. It looks as if the multithreaded approach falls behind the multiprocessor chip approach. However, the costs for the different processor configurations were not taken into account. The multiprocessor approach duplicates complete processors, whereas the multithreaded design only duplicates parts of the processor.

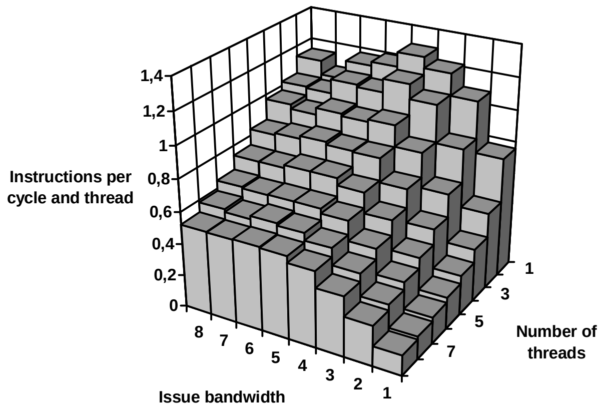

Figure 4: Average instruction throughput per thread

To verify our results, we compare them to Tullsen's simulation in [2]. We see that our simulator produces different results, especially viewing the 8-threaded 8-issue superscalar approach 1*(8,8) in the table below.

Number(Threads,Issue)	Tullsen's simulation [2]	Our simulation
1*(8,8)	6.64	4.19
8*(1,1)	5.13	6.07
2*(4,4)	6.80	6.80
1*(4,8)	4.15	3.37
4*(1,2)	3.44	4.32
2*(1,4)	1.94	2.56

The reason for the deviating results of Tullsen's simulation follows from the high number of execution units in Tullsen's approach and from the limited exploitation of instruction-level parallelism in the Alpha processor used as the base of Tullsen's simulation. Compared to our simulations, Tullsen's approach favours the multithreaded superscalar approach over the multiprocessor chip approach. For example, up to eight load/store units are used in Tullsen's simulation, ignoring hardware costs and design problems (we do not believe that it is cost-effective to implement 8 simultaneously working load/store units within a multithreaded superscalar processor).

It is obvious that different processor configurations can only be compared if a measurement for their costs (e.g. in chip space) is used. Otherwise, unrealistic processors are compared with each other, simply stating that more units result in more performance.

To get a rough measurement of the costs for a processor configuration, we propose a formula based on the Power PC 604 floor plan. The formula expresses hardware costs based on chip space usage per unit. Estimated costs per unit:

Unit type	Estimated cost
Integer unit	2
Load/Store unit	3
Fetch and decode unit	3
Branch unit	3
Caches	6
Registers	2∗Number of Threads
Completion unit	2∗Issue Bandwidth
Dispatch unit	1∗Number of Threads∗Issue Bandwidth

The formula is only a rule of thumb. The formula for the dispatch unit is based on the required interconnections between the dispatch unit, the register sets, and the execution units.

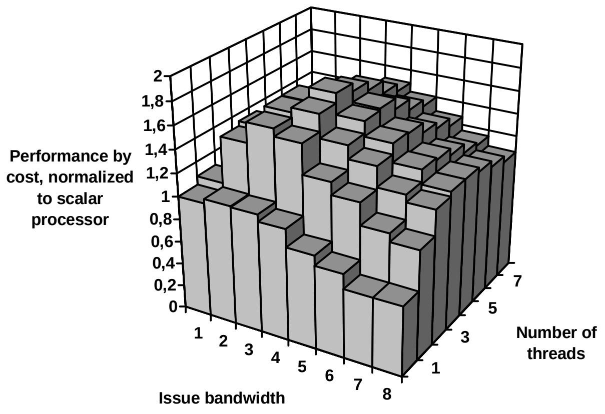

Figure 5: Average instruction throughput in relation to chip costs

As each thread's register set and dispatch queue has to be connected with all execution units, the required chip space is proportional to the product of the number of hosted threads and the issue bandwidth of the processor.

Fig. 5 displays the instructions per cycle in relation to the hardware costs of a specific processor configuration. The solution with four threads and issue bandwidth four shows the best performance/cost relation. However, this observation is application and design specific. The advantage of multithreading is highly dependent on the ratio of load/store instructions to other instructions in the workload. Also the chip costs change with different architectural decisions.

4 Conclusion

This paper examined the multithreaded superscalar processor in comparison to the multiprocessor chip approach, taking performance and hardware costs into consideration. For our research study we used a simulator for multithreaded processors based on the PowerPC 604.

While the single-threaded 8-issue superscalar processor only reaches a throughput of about 1.14, the 8-threaded 8-issue superscalar processor executes 4.19 instructions per cycle (the load/store frequency in the work load sets the theoretical maximum to 5 instruction per cycle). Increasing the issue bandwidth from 4 to 8 yields only a marginal gain in instruction throughput - a result that is nearly independent of the number of threads in the multithreaded processor.

The multiprocessor chip approach with 8 single-threaded scalar processors reaches 6.07 instructions per cycle. Using the same number of threads, the multiprocessor chip reaches a higher throughput than the multithreaded superscalar approach (refer to the previous paragraph). However, if we take the chip costs into consideration, a 4-threaded 4-issue superscalar processor outperforms a multiprocessor chip built from single-threaded processors by a factor of 1.8 in performance/cost relation.

5 References

1. T.A. Diep, C. Nelson, J.P. Shen: Performance Evaluation of the PowerPC 620 Microprocessor. The 22nd Annual International Symposium on Computer Architecture, Santa Margherita Ligure, June, 22-24, 1995, 163 - 174.

2. D. E. Tullsen, S. J. Eggers, H. M. Levy: Simultaneous Multithreading: Maximizing On-Chip Parallelism. The 22nd Annual International Symposium

on Computer Architecture, Santa Margherita Ligure, June, 22-24, 1995, 392 - 403.

3. Texas Instruments: TMS320C80 Technical Brief. Multimedia Video Processor (MVP). Texas Instruments 1994.

4. B. J. Smith: The Architecture of HEP. In: J. S. Kowalik (Ed.): Parallel MIMD Computation: The HEP Supercomputer and Its Applications. The MIT Pr ess, Cambridge 1985.

5. R. Alverson et al.: The Tera Computer System. 4th International Conference on Supercomputing, Amsterdam, June 11-15, 1990, 1- 6.

6. A. Agarwal et al.: The MIT Alewife Machine: Architecture and Performance. The 22nd Annual International Symposium on Computer Architecture, Santa Margherita Ligure, June, 22-24, 1995, 2 - 13.

7. W. Gruenewald, Th. Ungerer: Towards Extremely Fast Context Switching in a Block-multithreaded Processor. 22nd Euromicro Conference, Prague, Sept., 2-5, 1996.

8. S. P. Song, M. Denman, J. Chang: The PowerPC 604 RISC Microprocessor. IEEE Micro, Vol. 14, No. 5, Oct. 1994, 8 - 17.

9. J. L. Hennessy, D. A. Patterson: Computer Architecture a Quantitative Approach, San Mateo 1996.

10. U. Sigmund: Design of a Multithreaded Superscalar Processor. Master Thesis, University of Karlsruhe 1995 (in German).

11. U. Sigmund, Th. Ungerer: Identifying Bottlenecks in Multithreaded Superscalar Microprocessors. To be published.