Journal of Universal Computer Science, vol. 7, no. 9 (2001), 848-868
submitted: 16/10/01, accepted: 2/5/01, appeared: 28/9/01 © Springer Pub. Co.

On Speculation Control in Simultaneous Multithreaded Processors

Ulrich Sigmund
(VIONA Development GmbH, Karlsruhe,Germany
uli@viona.de)

Theo Ungerer
(University of Augsburg, Augsburg, Germany
ungerer@informatik.uni-augsburg.de)

Abstract: A simultaneous multithreaded (SMT) processor is able to issue and execute instructions from several threads simultaneously. A SMT processor reaches its highest performance, when all issue slots are utilized by non-speculative instructions provided that the workload is sufficient. SMT processors are able to utilize their resources by executing instructions of multiple threads simultaneously, whereas single-threaded processors fill their resources with highly speculative instructions that must frequently be discarded due to misspeculations. Consequently, we explore speculation control in SMT processor models with the target to increase performance by restricting the number of in-flight speculative instructions. We vary the sizes of internal buffers, the instruction fetch bandwidth, the instruction selection strategies and branch prediction models with the target to increase performance of the simulated SMT processor models.
Our results show (1) that retirement buffer sizes of 16 or 32 entries increase performance compared to smaller and to larger buffer sizes, (2) that the instruction fetch bandwidth can be decreased to two times four instructions per cycle without performance loss even for eight threaded eight issue processor models, (3) that an instruction selection strategy that discriminates speculative instructions in the fetch, decode, issue, and dispatch stages may increase overall performance, and (4) that a highly multithreaded processor with a sufficient workload may do without a branch prediction.

Key Words: Simultaneous multithreading, SMT, multimedia extension, MPEG-2 video decompression, speculation control

ACM CR Subject Classification: C.1

1 Introduction

Today’s superscalar processors are able to issue up to six instructions per cycle from a single sequential instruction stream [Silc et al. (99)]. VLSI technology will soon allow future microprocessors to issue eight or more instructions per cycle. However, ILP (instruction level parallelism) found in a conventional instruction stream is limited. Studies ([Bhandarkar and Ding 97], [Keeton et al. 98], [Barroso et al. 98]) show the limits of processor utilization even of today's superscalar microprocessors reporting IPC (instructions per cycle) 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. Switching to another thread masks latencies. A simultaneous multithreaded (SMT) processor, in particular, issues instructions from several threads simultaneously. It combines a wide-issue superscalar processor with multithreading. SMT approaches (see e.g. [Tullsen et al. 95, 96], [Sigmund and Ungerer 96], [Gulati, Bagherzadeh 96], or [Lo et al. 98]) are simulated and evaluated with multi-program workloads consisting of distinct programs (typically up to eight different SPEC benchmark programs executed together). Most of the simulations showed that an 8-threaded SMT is able to reach a two- to threefold IPC increase over single-threaded superscalar processors due to SMT's latency tolerance. In consequence, recent announcements by industry concern a 4-threaded SMT Alpha processor of DEC/Compaq [Emer (99)] and the MAJC-5200 processor of Sun, which features two 4-threaded processors on a single die [Gwennap 99].

While SMT is well known for boosting performance of a multi-programming workload, much fewer investigations cover multithreaded workloads with threads that are created from single programs. Such investigations concern either automatically parallelizing a single-threaded program by the processor hardware (see e.g. [Dubey 95], [Akkary 98], or [Tubella 98]) or hand parallelizing a single program into threads before executing it on an SMT processor simulator (see e.g. [Ortega 99] for SPECint95 benchmarks and [Pontius 99] for multimedia and signal processing applications).

Our approach falls in the second category. Our goal is to evaluate SMT processor performance for a single program that has been made multithreaded for the SMT processor. We choose the MPEG-2 video decompression as example for multimedia workloads. We reported on optimizations for a maximum processor model with an abundance of resources in [Oehring et al. 99a] and on a more realistic processor model in [Oehring et al. 99b]. Other approaches ([Pontius 99], [Wittenburg et al. 99]) looked at the application of the SMT technique for signal processors, however without applying multimedia instructions. Still compilers do not handle multimedia instructions efficiently. Hand coding is the state-of-the-art of commercially successful video processing algorithms. Our group did such a hand coding when transferring a commercial MPEG-2 video decompression algorithm to our SMT multimedia processor model and programming the algorithm in multithreaded fashion.

In the following we present a study of speculation control for the SMT processor models with a multithreaded MPEG-2 decompression algorithm as workload. Studies on instruction fetch and issue strategies and on branch prediction impact (see [Tullsen et al. 96], [Hily and Seznec 96a, 96b, 99]) on SMT processors have been previously performed for a workload of unrelated programs of the SPEC92 or SPEC95 benchmark suites assigning a distinct program to each thread slot in the SMT processor. Impact of different branch prediction methods has also been investigated in [Hily and Seznec 96a, 96b] for parallel programs of the SPLASH-2 benchmark suite. Several branch predictors were compared with the result that a gshare predictor [McFarling 93] performs best for the single- as well as for the multithreaded processor models. All studies show that less speculative instructions are in-flight in a SMT pipeline than in a comparable single-threaded processor pipeline. SMT is more tolerant to branch mispredictions than single-threaded processor models due to its ability to exploit inter-thread parallelism [Tullsen et al. 96]. Our study performs a deeper investigation of the potential techniques to increase performance by speculation control. We study in particular the impact of buffer sizes, instruction fetch bandwidth, instruction selection strategies from the various buffers, and branch prediction.

In our processor model, speculation is only introduced by branch instructions, which are processed assuming static or dynamic branch prediction techniques. We do not elaborate our investigations to the use of value speculation.

Section 2 introduces our workload, a MPEG-2 video decompression algorithm made multithreaded, and section 3 the basic processor model and simulator. Section 4 explains the simulation results varying the retirement buffer and reservation station sizes (section 4.1), the impact of the fetch bandwidth (section 4.2), the instruction selection strategies for the fetch, decode, issue units, the dispatch from reservation stations, and the retirement units (section 4.3), and the branch prediction method (section 4.4).

2 The Multithreaded MPEG-2 Video Decompression Algorithm

The MPEG-2 video decompression can be partitioned into the following six steps: header decode, Huffman and run-length decode, inverse quantization, IDCT (inverse discrete cosine transform), motion compensation, and display. The steps 1 and 2 must execute sequentially. The steps 3 to 6 can be executed in parallel for all macro blocks (or even blocks in the case of quantization and IDCT) of a single image. The MPEG-2 decompression is made multithreaded by splitting the task into a single parser thread, eight threads for macro block decoding, and an additional display thread. The parser thread executes the first two steps of the decompression and activates up to eight macro block decoding threads that perform steps three to five. The display thread transfers the decompressed images for display into the frame buffer (step 6). The average usage of the instructions assuming a local RAM storage for table look-up is given in Figure 1.

The parallelization overhead consists of about 1% thread control instructions. The maximum performance of the decompression is bound by the sequential parser thread. Our studies show that the sequential part covers approximately 14% of the executed instructions (depending on the bit rate and size of the encoded material). This results in a theoretical speedup of at most seven compared to the sequential algorithm. This speedup is further restricted by the roughly 20% local load-/store instructions to approximately five, if a single local load-/store unit is assumed.

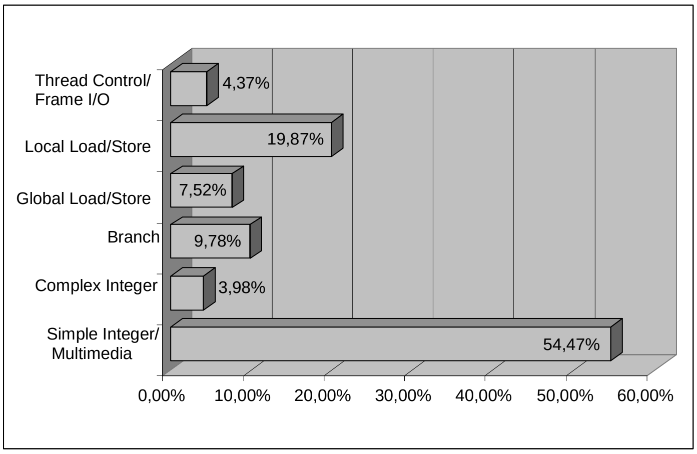


Figure 1: The average usage of instructions in our benchmark

3 The Multimedia-enhanced SMT Processor Model

In our approach to combine simultaneous multithreading and multimedia processing we start with a wide-issue superscalar general-purpose processor model based on the 6-stage pipeline of the PowerPC 604 [Song et al. 94], enhance it by simultaneous multithreading, further enhance it by combined integer/multimedia units and by on-chip RAM memory.

The SMT multimedia processor model (see Figure 2) features single or multiple instruction fetch (IF) and instruction decode (ID) units (one per thread), a single rename/issue (RI) unit, multiple, decoupled reservation stations, up to 10 execution units (four integer/multimedia units, a complex integer/multimedia unit, a branch unit, a thread unit, a global and one or two local load/store units), a single retirement (RT) and write back (WB) unit, rename registers, a branch target address cache (BTAC), separate I- and D-caches that are shared by all active threads. Type and number of execution units are carefully chosen based on further simulations reported in [Sigmund 00].

The D-cache is a non-blocking 4-way set-associative write-back cache with write-allocation. Loads and stores of the same thread are performed out of order unless an address conflict arises. The four integer/multimedia units share a single reservation station, which is able to dispatch four instructions per cycle; all other reservation stations are separate per execution unit. We employ thread-specific instruction buffers (between IF and ID), issue buffers (between ID and RI), and retirement buffers (in front of RT). Each thread executes in a separate architectural register set. However, there is no fixed allocation between threads and (execution) units. The pipeline performs an in-order instruction fetch, decode, rename/issue to reservation stations, out-of-order dispatch from the reservation stations to the execution units, out-of-order execution, and an in-order retirement and write-back.

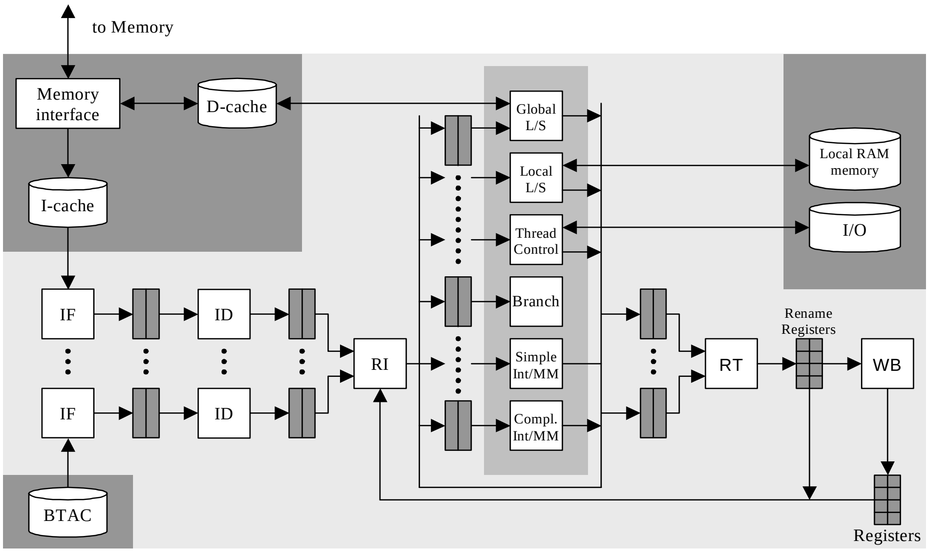


Figure 2: The SMT multimedia processor model

The rename/issue stage simultaneously selects instructions from all issue buffers up to its maximum issue bandwidth (SMT feature). The integer units are enhanced by MMX-style multimedia processing capabilities (multimedia unit feature). We employ a thread control unit for thread start, stop, synchronization, and for I/O operations. We also employ a 32 KB local on-chip RAM memory (enough for constants and variables of the simulation workload) accessed by one or two local load/store units. Simulations without the on-chip RAM showed an IPC decrease of about 0.6 for all configurations (see [Sigmund 00]).

We use the DLX instruction set enhanced by thread control and by multimedia instructions. In contrast to the simulations in [Oehring et al. 99a, 99b] which are based on static branch prediction, the simulations described below apply a dynamic branch predictor, in fact a two-level adaptive prediction with 2048 two-bit up-down counters and an 8 bit history using the gshare scheme [McFarling 93]. The misprediction penalty is 5 cycles. Up to 64 branch speculations per thread are possible simultaneously.

Moreover we choose to fix the following parameters for all simulations:

- 32-bit general-purpose registers (per thread),
- a 1024-entry branch target address cache (BTAC),
- 1024 rename registers,
- a common reservation station for the integer units,
- a 32-entry issue buffer per thread,
- 4 MB main memory (enough to store the whole simulation workload), and
- a 64-bit system bus.

The Karlsruhe SMT simulator KSMS [Sigmund 00] is an execution-based simulator that models all internal structures of the microprocessor model. Two different videos are used as data stream workload for the MPEG-2 algorithm. The simulator decompresses one to two seconds of video per simulator run. The produced picture frames can be visually and digitally analyzed to evaluate the correct working of the workload routine and the simulator. Recently the KSMS has been extended by a complexity estimation tool that allows to estimate the transistor count and chip-space of simulated microprocessors based on the simulator's configuration file [Steinhaus et al. 01a and 01b].

4 Performance Results

Our evaluations proceeded in two steps. First, we developed and optimized maximum processor models that included as many resources as a potential processor in years 2006 to 2009 on the basis of the SIA National Roadmap for Semiconductors prognosis [sematech 00]. Our transistor count of these models is a maximum of nearly 300 M transistors per processor chip for the eight threaded eight issue model including the large I- and D-caches. Second, we optimized the configurations towards more realistic processor models with a resource capacity of up to 12.5 M transistors, which is less than contemporary processors utilize. The full experimental results are reported in [Sigmund 00]. Memory hierarchy issues are described in [Sigmund and Ungerer 00], the chip space and transistor count estimations in [Sigmund et al. 00].

In this paper we focus on speculation control issues applying different "maximum" and "realistic" processor models. In the following sections our investigations vary:

- the retirement buffer and reservation station sizes to control the number of speculative in-flight instructions (see section 4.1),
- the impact of the fetch bandwidth (see section 4.2),
- the instruction selection strategies for the fetch, decode, issue units, the dispatch from reservation stations, and the retirement units (see section 4.3),
- the branch prediction method (see section 4.4).

Each simulation setup consists of simulating one to eight threads and an issue bandwidth of one to eight (1, 2, 4, 6, and 8 in the figures). A x-threaded y-issue model is dubbed as (x, y)-model.

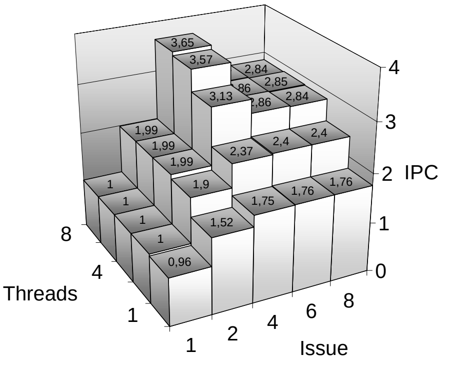
4.1 Impact of Buffer Sizes

Our initial maximum processor configuration features an abundance of resources, in detail,

- 4 MB I-cache,
- 4 MB D-cache,

- a single local load/store unit,
- 256-entry reservation stations,
- an own result bus per execution unit,
- 256-entry retirement buffers,
- The instruction fetch unit is assumed as being able to fetch as many instructions as the configuration needed, e.g. up to eight by eight instructions per cycle in the 8-threaded 8-issue case.

Figure 3 shows that a single-threaded processor reaches a maximum IPC of 1,76 instructions per cycle (IPC) with nearly no IPC increase for the 6- and 8-issue models. We notice an IPC decrease for the 4-, 6- and 8-threaded models in the 6- and 8-issue cases. The (8, 8)-processor model reaches a disappointing IPC of 2.84 which is only 78% of the IPC of the (8, 4)-model. One possible explanation for this IPC decrease of the high threaded high issue models is that single threads clog large parts of the common resources and prohibit the proceeding of instructions of other threads.

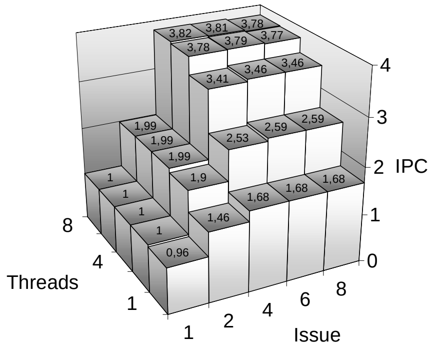


Threads	1, 2, 4, 6, 8
Issue bandwidth	1, 2, 4, 6, 8
Fetch and decode bandwidth	8 x 8
Reservation stations	
• Integer units	256
• Load-/store units	256 each
Retirement buffers	256 each
Integer units	4
Local Load-/store units	1
D-cache fill burst rate	6:2:2:2
D-cache	4 MB
I-cache	4 MB
Branch prediction method	dynamic

Figure 3: Initial maximum processor models

A look at the buffer fill levels for the configurations of Figure 3 showed an instruction overload of the processor by too many in-flight instruction in particular in the execution engine—the out-of-order sections of the pipeline. To restrict the number of in-flight instructions in the out-of order section of the pipelines we simulate models with retirement buffers of 32 entries (instead of 256) per thread next.

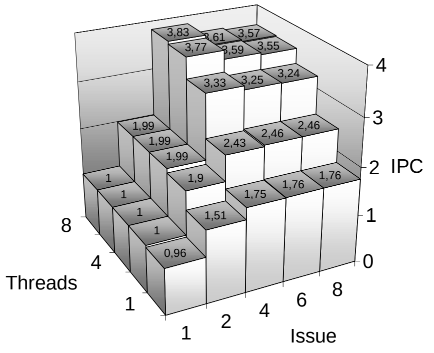
Figure 4 shows that all multithreaded models profit from smaller retirement buffers (compared to Figure 3). The performance gap of the 6- and 8-issue models to the 4-issue model is much smaller and disappears for the 4-threaded cases.



Threads	1, 2, 4, 6, 8
Issue bandwidth	1, 2, 4, 6, 8
Fetch and decode bandwidth	8 x 8
Reservation stations	
• Integer units	256
• Load-/store units	256 each
Retirement buffers	32 each
Integer units	4
Local Load-/store units	1
D-cache fill burst rate	6:2:2:2
D-cache	4 MB
I-cache	4 MB
Branch prediction method	dynamic

Figure 4: Initial maximum processor models with smaller retirement buffers

A further possibility to prevent single threads to occupy too many processor resources is to restrict the number of instructions with long latencies. A reason for a negative effect of large reservation stations for the load/store and thread control units may be the fact, that those instructions have a long latency, and typically two to three integer instructions as consumers. The effect is that the consuming integer instructions occupy space in the integer reservation station, thus blocking instructions from other threads from entering it.



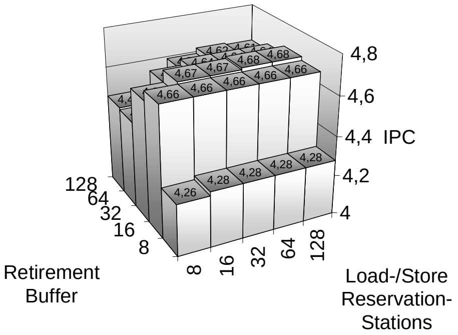
Threads	1, 2, 4, 6, 8
Issue bandwidth	1, 2, 4, 6, 8
Fetch and decode bandwidth	8 x 8
Reservation stations	
• Integer units	256
• Load-/store units	32 each
Retirement buffer	256 each
Integer units	4
Local Load-/store units	1
D-cache fill burst rate	6:2:2:2
D-cache	4 MB
I-cache	4 MB
Branch prediction method	dynamic

Figure 5: Initial maximum processor models with reduced reservation station sizes of the load-/store unit and the thread control unit

This effect of buffer clogging is made even worse by store and thread control instructions that cannot be executed speculatively and remain in the respective reservation stations until branch resolution. We therefore restrict the reservation stations of the global and local load-/store units and of the thread control unit to 32 entries each.

Figure 5 shows that our assumption is right. However, only a slight performance increase is reached, which is smaller than in the previous case.

To find out how the combination of retirement buffer and load-/store reservation station sizes impact performance we simulated various combinations.



Threads	8
Issue bandwidth	8
Fetch and decode bandwidth	8 x 8
Reservation stations	
• Integer units	256
• Load-/store units	8-128 each
Retirement buffer	8-128 each
Integer units	4
Local Load-/store units	1
D-cache fill burst rate	6:2:2:2
D-cache	4 MB
I-cache	4 MB
Branch prediction method	dynamic

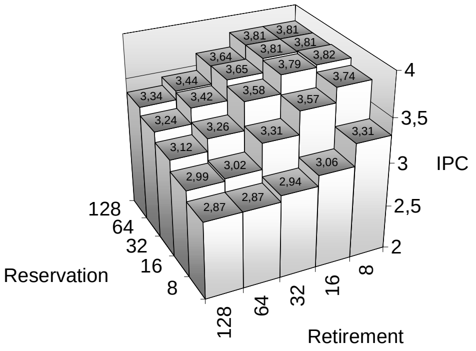
Figure 6: Retirement buffer versus load-/store reservation station sizes

Figure 6 shows that a retirement buffer length of only 8 entries is too small, but 16 to 32 entries per retirement buffer are enough to reach the maximum performance. The length of the load-/store reservation station is irrelevant for small retirement buffer sizes, but its importance grows when retirement buffer sizes are increased. 16 to 32 entries for both buffer types yield the highest performance. The reason is again the reduced number of instructions that a single thread may send in the out-of-order parts of the pipelines. Thus non-speculative instructions are preferably executed.

The following series of experiments focuses on processor models with more “realistic” memory hierarchy assumptions, in particular I- and D-caches of 64 KB each (instead of 4 MB) and a D-cache fill burst rate of more realistic 32:4:4:4 (instead of 6:2:2:2) processor cycles.

The first experiment investigates again the impact of retirement buffer sizes and reservation station sizes (varying the sizes of the reservation stations in front of the integer and the local load/store units).

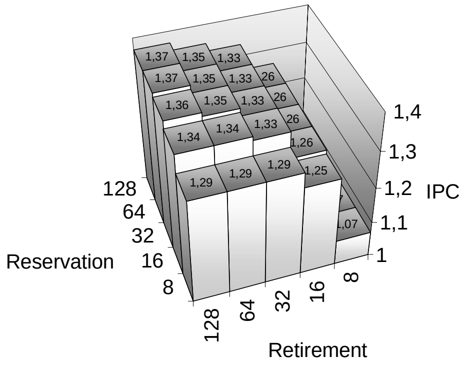
Figure 7 shows (in accordance to Figure 6) that large retirement buffers decrease performance. Smaller buffer sizes are superior for the 8-threaded model which is in contrast to the single threaded case shown in Figure 8.



Threads	8
Issue bandwidth	8
Fetch and decode bandwidth	8 x 8
Reservation stations	
• Integer units	8 - 128
• Load-/store units	8 – 128 each
Retirement buffer	8 – 128 each
Integer units	4
Local Load-/store units	2
D-cache fill burst rate	32:4:4:4
D-cache	64 KB
I-cache	64 KB
Branch prediction method	dynamic

Figure 7: Retirement buffer versus load-/store reservation station sizes of the (8, 8)-model

As expected the performance in the single-threaded model increases with the retirement buffer size because threads cannot block each other. A larger number of in-flight instructions allows to fill more issue slots in the single-threaded model, in contrast to the multithreaded models, that find enough instructions to issue and where more in-flight instructions of each thread may lead to blocking of useful instructions.



Threads	1
Issue bandwidth	8
Fetch and decode bandwidth	8 x 8
Reservation stations	
• Integer units	8 - 128
• Load-/store units	8 – 128 each
Retirement buffers	8 – 128 each
Integer units	4
Local Load-/store units	2
D-cache fill burst rate	32:4:4:4
D-cache	64 KB
I-cache	64 KB
Branch prediction method	dynamic

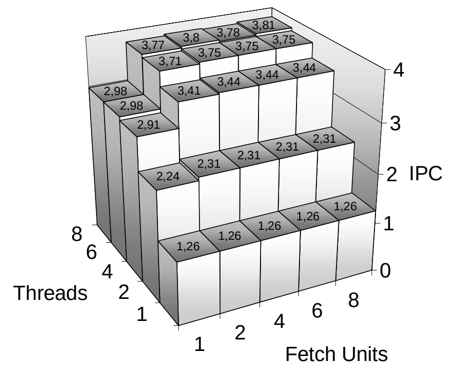
Figure 8: Retirement buffer versus load-/store reservation station sizes of the (1, 8)-model

For the following simulations we choose multithreading-friendly models with 32 entry reservation stations and 16 entry retirement buffers, which slightly penalizes the single-threaded case (for additional simulations of the single-threaded friendly models with 16 entry reservation stations and 32 entry retirement buffers see [Sigmund 00]).

4.2 Impact of Fetch Bandwidth

Another attempt to lessen the overload of the processor is to restrict the instruction fetch bandwidth assuming that the overload is caused by the scaling of the fetch bandwidth with the issue and thread bandwidth (8*8=64 instructions are fetched in the (8, 8) model). In models with 2*8 or even 2*4 fetched instructions per cycle, less instructions are fetched in the buffers, which may lessen the blocking of instructions of single threads.

The following experiments look at the realistic processor models assuming the multithreading friendly variant of 16 entry retirement buffers and 32 entry reservation stations is shown in Figure 9.



Threads	1, 2, 4, 6, 8
Issue bandwidth	8
Fetch and decode bandwidth	1 - 8 x 4
Reservation stations	
• Integer units	32
• Load-/store units	32 each
Retirement buffers	16 each
Integer units	4
Local Load-/store units	2
D-cache fill burst rate	32:4:4:4
D-cache	64 KB
I-cache	64 KB
Branch prediction method	dynamic

Figure 9: Varying the number of fetch units for the multithreading-friendly models

Figure 9 shows the number of fetch units in relation to the number of threads. It shows that two to eight fetch units with 4 instructions each yield roughly the same performance. The fetch bandwidth has only a negligible performance impact. This may be because the number of fetched instructions is still higher than the number of executed instructions, keeping the buffers always full. Only the model with a single fetch unit shows a performance decrease caused by an underutilization of the pipeline units (except for the single threaded case).

Figure 10 shows the ratio of fetched to executed instructions (left) and the ratio of decoded to executed instruction (right). The figure shows that the number of unnecessarily fetched instructions increases with a higher number of fetch units, but this increase disappears already before the decode unit. That is why the number of fetch units has no impact on the IPC.

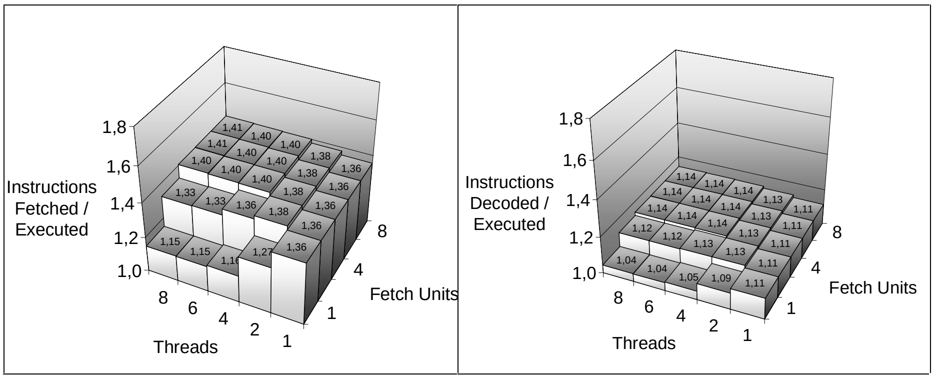


Figure 10: Varying the number of fetch units for the 8-issue models with a fetch bandwidth of four each

Additional simulations based on the "initial maximum processor models" of Figure 3 with 2*8 fetched instructions per cycle lead to the same observation except for a slight performance increase of 0.1 IPC in the high threaded models and a performance decrease for the single-threaded models. The latter case is because of the overload of the single-threaded pipeline by more speculative instruction, which are fetched by the two-fold fetch bandwidth compared to the previous simulations. Additional simulations that compare 2*8 and 2*4 fetches show that the number of fetched instructions decreases with 2*4 fetches, but the number of decoded instruction remains nearly the same. The number of instructions fetched per cycle and per unit can be reduced to four without performance impact if the number of fetch units is at least two.

4.3 Use of Buffer Selection Strategies

Our next investigations regard different thread selection strategies. [Tullsen et al. 96] examined instruction fetch policies that give highest fetch priority to the threads that are least likely to be on a wrong path (BRCOUNT), have the fewest outstanding data cache misses (MISSCOUNT), have the fewest instructions in the decode, rename and queue pipeline stages (ICOUNT), and a policy that penalizes threads that have the oldest instructions in integer and floating-point queues (IQPOSN). The combining strategies ICOUNT and IQPOSN perform better than the strategies that evaluate only a single cause (BRCOUNT and MISSCOUNT), and all are better than the basic round-robin policy. Issue selection policies like branch-first or speculative-instructions-last showed no significant improvement over the basic oldest-instruction-first policy.

We modified Tullsen et al.'s experiments by additionally controlling the decode, the issue, the dispatch from reservation stations, and the retirement due to the three different parameters thread priority, speculation depth, and saturation of the (per thread) issue and retirement buffers. Up to now we applied a simple round robin strategy, whereby instructions are selected in round robin fashion per thread. We devised three more strategies: highest-priority-thread-first, non-speculative-instructions-first (similar to BRCOUNT), and non-saturated-first (similar to ICOUNT). Highest priority is given to the parser thread.[1] All three strategies were subsequently applied to the fetch, the decode, the issue, the dispatch from reservation stations, and the retirement stages.

0	Round Robin	Instructions are selected for retirement in round robin fashion per thread.
1	Priority	Instructions of threads with high priority are preferably retired. In our workload we give highest priority to the single parser thread.
2	Saturated	Threads with a high number of instructions in their retirement buffers are preferred.

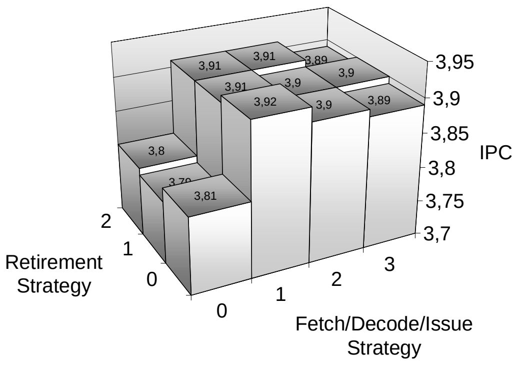
Table 1: Strategies of the retirement unit

0	Round Robin	Instructions are selected in round robin fashion per thread.
1	Priority	Instructions of threads with high priority are preferred. In our workload we give highest priority to the single parser thread.
2	Speculative	Non-speculative instructions are preferred.
3	Saturated	Threads with a high number of instructions in their retirement buffers are discriminated.

Table 2: Strategies of the fetch units, decode units, and dispatch out of the reservation stations

Figure 11 shows the (8, 8)-configuration with configuration parameters of a realistic processor, however assuming eight fetch units with a four instructions per cycle fetch each such that instruction fetch is not a bottleneck. Inspecting the selection strategies for this configuration shows a performance increase for the priority strategy, if applied to the fetch, decode and issue stages; however for retirement the simple round-robin strategy performs best.

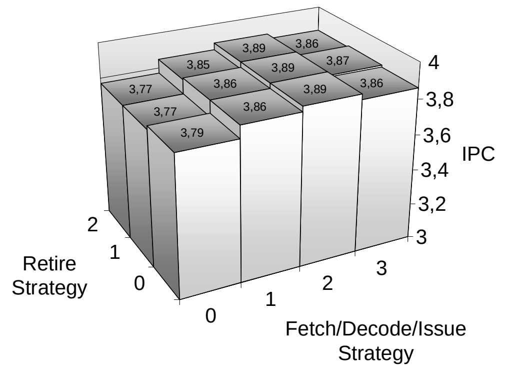
[1] Tulsen et al’s workload consisted of up to 8 different SPEC92 programs that are simultaneously active in 8 thread slots. The priority scheme makes only sense when the dependences between the threads can be taken into account which is only possible with threads that originate from a single program.



Threads	8
Issue bandwidth	8
Fetch and decode bandwidth	8 x 4
Reservation stations	
• Integer units	32
• Load-/store units	32 each
Retirement buffers	16 each
Integer units	4
Local Load-/store units	2
D-cache fill burst rate	32:4:4:4
D-cache	64 KB
I-cache	64 KB
Branch prediction method	dynamic

Figure 11: Selection strategies for the (8, 8)-model

To validate this observation we investigate the impact on the realistic processor model with only two instruction fetch and decode units in Figure 12.



Threads	8
Issue bandwidth	8
Fetch and decode bandwidth	2 x 4
Reservation stations	
• Integer units	32
• Load-/store units	32 each
Retirement buffers	16 each
Integer units	4
Local Load-/store units	2
D-cache fill burst rate	32:4:4:4
D-cache	64 KB
I-cache	64 KB
Branch prediction method	dynamic

Figure 12: Selection strategies for the (8, 8) models with two fetch and decode units

Because of the restricted fetch bandwidth the models show an increased performance of the speculation-restricted strategy for fetch, decode and issue. Further simulations (see [Sigmund 00]) justify the impression that for realistic processor models the speculation-restricting strategy (non-speculative-instructions-first) is the best choice, whatever combination of retirement buffer and reservation station sized are used.

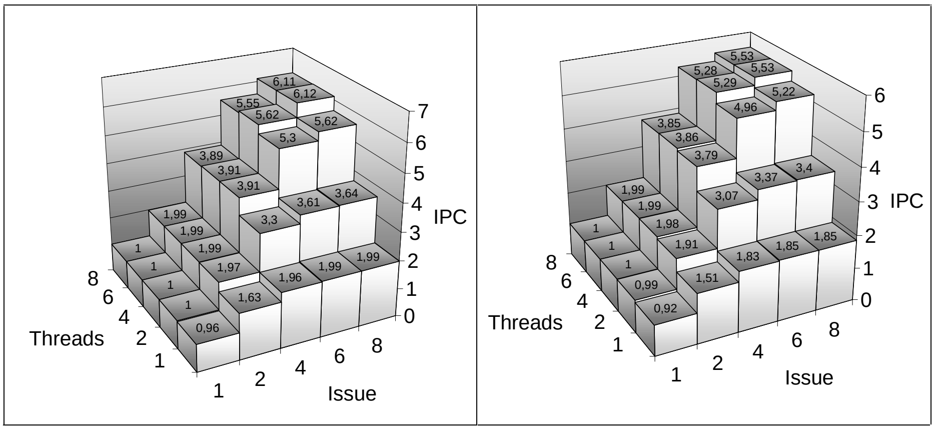
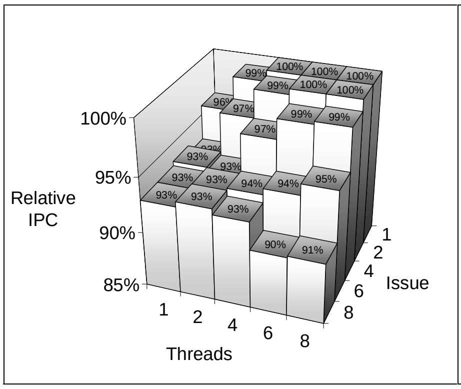


Figure 13: Maximum processor models with dynamic branch prediction (left) and static branch prediction (right), simulation parameters see Figure 14

4.4 Impact of Branch Prediction

In our MPEG-2 decompression algorithm we observed two main patterns of branch behavior:

- loops with a fixed number of interactions,
- branches with data dependent branch conditions which are hard to predict.



Threads	1, 2, 4, 6, 8
Issue bandwidth	1, 2, 4, 6, 8
Fetch and decode bandwidth	8 x 8
Reservation stations	
• Integer units	256
• Load-/store units	256 each
Retirement buffer	256 each
Integer units	4
Local Load-/store units	2
D-cache fill burst rate	6:2:2:2
D-cache	4 MB
I-cache	4 MB
Branch prediction method	dynamic/static

Figure 14: Maximum processor models with static branch prediction in relation to dynamic branch prediction

Our next set of experiments is based on processor configurations, which reached the highest throughput in our simulations. Figure 13 shows the performance of processor models with the two-level adaptive branch prediction and with a simple static prediction (backward branches predict taken, forward branches predict not taken). Figure 14 shows the IPC ratio of static versus dynamic branch prediction in percentage and the simulation parameters.

Multithreaded processors with low issue bandwidth suffer less or not at all, but the performance decreases get more dramatic with higher issue bandwidths. In particular the (6,8)- and (8,8)-models suffer most which demonstrates that multithreaded processors with a high number of resources may profit from a good branch prediction. More "realistic" models are investigated below.

The following simulations are based on a realistic processor model with a long memory access bandwidth of 32:4:4:4:4:4:4:4, which has been shown to improve memory bandwidth and thus performance of a processor with a realistic memory hierarchy.

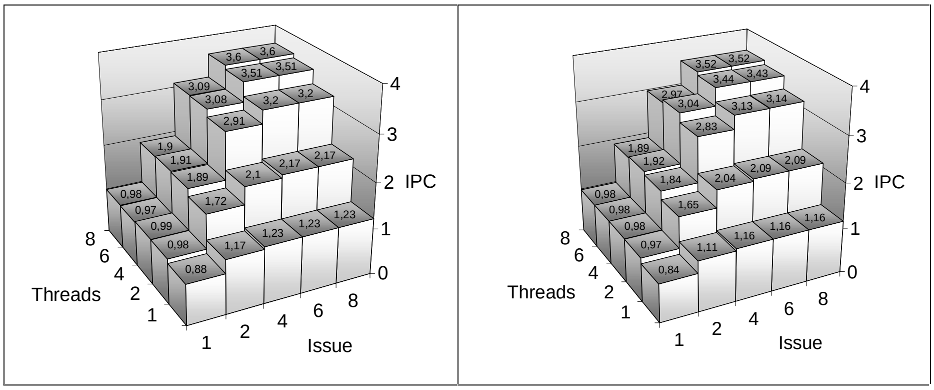
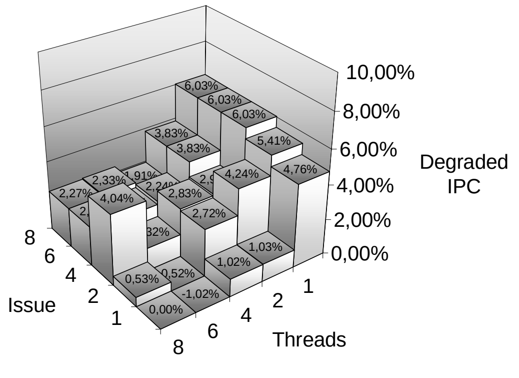


Figure 15: Realistic processor models with dynamic (left) versus static (right) branch prediction, simulation parameters see Figure 16

The IPC values in Figure 15 and the relative IPC decrease given in Figure 16 show a performance decrease for all configurations with static branch prediction. Figure 16 shows—as seen before for the maximum models—that the multithreaded models suffer less by a worse branch prediction. However, the performance decrease is so significant that a renunciation of the dynamic branch prediction is not advisable. This complies with the observations of [Hily and Seznec 96a, 96b].



Threads	1, 2, 4, 6, 8
Issue bandwidth	1, 2, 4, 6, 8
Fetch and decode bandwidth	2 x 4
Reservation stations	
• Integer units	32
• Load-/store units	32 each
Retirement buffers	16 each
Integer units	4
Local Load-/store units	1
D-cache fill burst rate	32:4:4:4:4:4:4:4
D-cache	64 KB
I-cache	64 KB
Branch prediction method	dyn./static

Figure 16: Relative IPC decrease by use of static branch prediction, parameters

An interesting remaining question is the potential performance impact by a total abandonment of branch prediction. If instruction fetching for a thread is blocked after decoding of a branch instruction until branch resolution, we yield the IPC and the relative IPC decrease given in Figure 17.

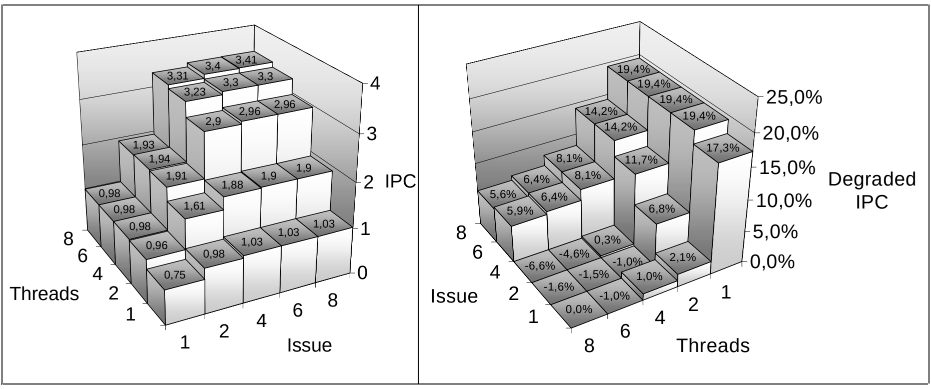


Figure 17: Realistic processor models without branch prediction (left) and relative IPC decrease by abduction of branch prediction

A significant performance decrease is observed for the single-threaded models. Comparing the relative IPC decreases shows that the single-threaded configurations suffer about 20% performance decrease. The multithreaded configurations gain up to 6% by renouncement of a branch prediction. This is a significant clue that speculation may be harmful to the performance of a multithreaded processor, because resources which are occupied by speculative instructions cannot be utilized for non-speculative instructions of other threads.

This is also in particular interesting concerning not IPC but power consumption. Because executed instructions of a wrong branch path consume unnecessarily power, multithreading without speculative execution after branches could guarantee high performance with lower power consumption, thus reaching a better ratio of executed instructions to power consumption (see also [Seng et al. 00]). Such a microarchitecture technique is technology independent and can be combined with most other kinds of power reduction mechanisms.

5 Conclusions

We simulated various SMT multimedia processor models using a hand-coded multithreaded MPEG-2 video decompression algorithm as workload. We see that multithreading is very effective if the issue bandwidth is at least four. In particular the two and four-threaded models lead to a high performance increase in the multiple-issue models. There is little improvement for the single-threaded (the contemporary superscalar case) and the two-threaded models when issue bandwidth is increased from 4 to 8. Our architectural optimizations targeted speculation control by fetch bandwidth and buffer size restrictions, by instruction selection strategies, and by renouncing branch speculation. The simulation results concerning speculation control are summarized as follows:

- The retirement buffers of a SMT processor should be small—about 16 to 32 entries per thread—to prevent single threads from blocking of resources. However, a larger retirement buffer improves performance of the single-threaded processor models, and of the SMT processor if the workload is insufficient (e.g. only a single thread) to utilize all resources. The consequence could be a dynamic solution that restricts the number of retirement buffer entries if a SMT processor is fully loaded with threads and increases the number in case of a low workload in the SMT processor, yielding a higher performance in the “single-threaded” mode.
- An instruction selection strategy that discriminates speculative instructions in the fetch, decode, issue, and dispatch from reservation station stages increases overall performance.
- Dynamic branch prediction is effective also for SMT processors. The multithreaded processor is able to bridge latencies caused by branch mispredictions, however, the throughput of the whole workload decreases if wrong-path instructions are executed. A worse branch prediction increases the number of wrong-path instructions in all processor models.
- On the other hand a highly multithreaded processor with a sufficient workload may do without a branch prediction at all. In few configurations this may even reach a better performance than with branch speculation. But if a branch speculation is used an excellent prediction method is preferable.
- If low power consumption is the main target of a processor design, then multithreading in conjunction with speculation control or even total abandonment of branch prediction could be a design paradigm. Because executed instructions of a wrong branch path consume power, multithreading without speculative execution after branches may yield a better ratio of

executed instructions to power consumption. If the workload is sufficiently multithreaded and the overall performance is more important than single-thread performance, then this combination will be cost-effective in relation to power-consumption and reach the same or an only slightly smaller performance than the more speculative version.

Our investigations showed that aggressive speculation can harm the performance of an SMT processor. The reason is that resources that are filled by speculative instructions in a single-threaded processor are utilized more effectively by executing instructions of other threads in an SMT processor.

Multithreaded processors will hit the market in a couple of years. It is therefore important to investigate how single applications can be made multithreaded to profit from the new microarchitectural feature. We showed that a MPEG-2 video decompression algorithm can be made multithreaded to provide an excellent workload for an up to 8-threaded 8-issue SMT processor that expensively uses up to 4 multimedia units. Our processor models are specifically tailored to the characteristics of our multithreaded MPEG-2 algorithm. We assume that other multimedia applications feature similar characteristics. So further investigations could be done in the area of audio and video compression and decompression as well as 3D image processing. For our investigations about speculation control we would welcome more research with different workloads. Other workloads might favor a different configuration, in particular, if the workload consists of unrelated threads of control that do not share any data.

References

[Akkary 98] Akkary, H., Driscoll, M. A.: “A Dynamic Multithreading Processor”; Proc. MICRO-31, Dallas (1998), 226-236.

[Barroso et al. 98] Barroso, L.A., Gharachorloo, K., Bugnion, E.: “Memory System Characterization of Commercial Workloads”; Proc. 25th Ann. Int. Symp. on Computer Architecture. Barcelona, Spain (1998), 3-14.

[Bhandarkar and Ding 97] Bhandarkar, D., Ding, J.: “Performance Characterization of the Pentium Pro Processor”; Proc. 3rd Int. Symp. on High-Performance Computer Architecture HPCA-3, San Antonio (1997).

[Dubey 95] Dubey, P. K., O’Brien, K., O’Brien, K. M., Barton, Ch.: “Single-Program Speculative Multithreading (SPMD) Architecture: Compiler-assisted Fine-grained Multithreading”; IBM Research Report RC 19928 (1995)

[Emer 99] Emer, J.: “Simultaneous Multithreading: Multiplying Alpha’s Performance”; Proc. Microprocessor Forum 1999, San Jose, Ca., (1999).

[Gulati, Bagherzadeh 96] Gulati, M., Bagherzadeh, N.: “Performance Study of a Multithreaded Superscalar Microprocessor”; Proc. 2nd Int. Symp. on High-Performance Computer Architectures (1996), 291-301.

[Gwennap 99] Gwennap, L.: “MAJC Gives VLIW a New Twist”; Microprocessor Report, Sept. 13, 1999.

[Hily and Seznec 96a] Hily, S., Seznec, A.: “Branch Prediction and Simultaneous Multithreading”; 1996 Proc. Int. Conf. on Parallel Architectures and Compilation Techniques PACT '96, Boston (1996), 169-173.

[Hily and Seznec 96b] Hily, S., Seznec, A.: “Branch Prediction and Simultaneous Multithreading”; IRISA Publ. No 997, Rennes, France (1996)

[Hily and Seznec 99] Hily, S., Seznec, A.: “Out-of-order Execution May Not Be Cost-effective on Processors Featuring Simultaneous Multithreading”; Proc. HPCA-5, Orlando (1999), 64-67.

[Keeton et al. 98] Keeton, K., Patterson, D.A., He, Y.Q., Raphael, R.C., Baker, W.E.: “Performance Characterization of a Quad Pentium Pro SMP Using OLTP Workloads”; Proc. 25th Ann. Int. Symp. on Computer Architecture. Barcelona, Spain (1998), 15-26.

[Lo et al. 98] Lo, J.L., Barroso, L.A., Eggers, S.J., Gharachorloo, K., Levy, H.M., and Parekt, S.S.: “An Analysis of Database Workload Performance on Simultaneous Multithreaded Processors”; Proc. 25th Ann. Int. Symp. on Computer Architecture. Barcelona, Spain (1998), 39-50.

[McFarling 93] McFarling, S.: “Combining Branch Predictors”; DEC WRL Technical Note TN-36, DEC Western Research Laboratory (1993)

[Oehring et al. 99a] Oehring, H., Sigmund, U., Ungerer, Th.: “Simultaneous Multithreading and Multimedia”; Proc. Workshop on Multithreaded Execution, Architecture and Compilation, MTEAC99, in connection with the HPCA-5 Conf., Orlando (1999).

[Oehring et al. 99b] Oehring, H., Sigmund, U., Ungerer, Th.: “MPEG-2 Video Decompression on Simultaneous Multithreaded Multimedia Processors”; Proc. 1999 Int. Conf. on Parallel Architectures and Compilation Techniques PACT '99, Newport Beach, Ca. (1999), 11-16.

[Ortega 99] Ortega, D., Martel, I., Krishnan, V., Ayguage, E., Valero, M.: “Quantifying the Benefits of SPECint Distant Parallelism in Simultaneous Multithreadeing Architectures”; Proc. 1999 Int. Conf. on Parallel Architectures and Compilation Techniques PACT '99, Newport Beach, Ca. (1999), 17-124.

[Pontius 99] Pontius, M., Bagherzadeh, N.: “Multithreaded Extensions Enhance Multimedia Performance”; Proc. Workshop on Multithreaded Execution, Architecture and Compilation, MTEAC99, in connection with the HPCA-5 Conf., Orlando (1999).

[sematech 00] http://www.sematech.org/public/home.htm

[Seng et al.00] Seng, J.S., Tullsen, D.M., Cai, G.Z.N.: “Power-Sensitive Multithreaded Architecture”; Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers and Processors, ICCD 2000, Austin, Texas, (2000).

[Sigmund 00] Sigmund, U.: “Entwurf und Evaluierung mehrfädig superskalarer Prozessortechniken im Hinblick auf Multimedia”; Doctoral Thesis, Technische Universität Karlsruhe (2000)

[Sigmund et al. 00] Sigmund, U., Steinhaus, M., Ungerer, Th.: “Performance, Transistor Count and Chip Space Assessment of Multimedia-enhanced Simultaneous Multithreaded Processors”; Proc. Workshop on Multi-Threaded Execution, Architecture and Compilation (MTEAC-4), Monterrey, Ca. (2000).

[Sigmund and Ungerer 96] Sigmund, U., Ungerer, Th.: “Evaluating A Multithreaded Superscalar Microprocessor Versus a Multiprocessor Chip”; Proc. 4th PASA Workshop–Parallel Systems and Algorithms. Forschungszentrum Jülich, Germany, World Scientific Publishing (1996), 147-159.

[Sigmund and Ungerer 00] Sigmund, U., Ungerer, Th.: “Memory Hierarchy Studies of Simultaneous Multithreaded Processors for MPEG-2 Decompression”; Proc. Workshop on Multithreaded Execution, Architecture and Compilation, MTEAC2000, in connection with the HPCA-6 Conf., Toulouse, France (2000).

[Silc et al. (99)] Silc, J., Robic, B., Ungerer, Th.: “Processor Architecture - From Dataflow to Superscalar and Beyond”; Springer, Berlin / Heidelberg / New York (1999)

[Song et al. 94] Song, S.P., Denman, M., Chang, J.: “The PowerPC 604 RISC Microprocessor”; IEEE Micro, Oktober (1994), 8-17.

[Steinhaus et al. 01a] M. Steinhaus, R. Kolla, J. L. Larriba-Pey, Th. Ungerer, M. Valero: “Transistor Count and Chip-Space Estimation of Simulated Microprocessors”; Research Report UPC-DAC-2001-16, UPC, Barcelona, Spain (2001)

[Steinhaus et al. 01b] M. Steinhaus, R. Kolla, J. L. Larriba-Pey, Th. Ungerer, M. Valero: “Transistor Count and Chip-Space Estimation of SimpleScalar-based Microprocessor Models”; Workshop on Complexity-Effective Design, , in connection with the 28th Int. Symp. on Computer Architecture, Göteborg, Sweden (2001).

[Tubella 98] Tubella, J., Gonzales, A.: “Control Speculation in Multithreaded Processors Through Dynamic Loop Detection”; Proc. Fourth International Symposium on High-Performance Computer Architecture HPCA-4, Las Vegas, (1998), 14-23.

[Tullsen et al. 95] Tullsen, D. M., Eggers, S. J., and Levy, H. M.: “Simultaneous Multithreading: Maximizing On-Chip Parallelism”; Proc. 22nd Ann. Int. Symp. on Computer Architecture. Santa Margherita Ligure, Italy (1995), 392-403.

[Tullsen et al. 96] Tullsen, D. M., Eggers, S. J., Levy, H. M., Jo, J. L., and Stamm, R. L.: “Exploiting Choice: Instruction Fetch And Issue on an Implementable Simultaneous Multithreading Processor”; Proc. 23rd Ann. Int. Symp. on Computer Architecture. Philadelphia (1996), 191-202.

[Wittenburg et al. 99] Wittenburg, J.P., Meyer, G., Pirsch, P.: “Adapting and Extending Simultaneous Multithreading for High Performance Video Signal Processing Applications”; Proc. Workshop on Multithreaded Execution, Architecture and Compilation, MTEAC99, in connection with the HPCA-5 Conf., Orlando (1999).