Thread-Level Speculation and Chip-Multiprocessors: An Overview

Francisco J. Villa, Manuel E. Acacio, and José M. García

Universidad de Murcia, Departamento de Ingeniería y Tecnología de Computadores
30071 Murcia (Spain)
{fj.villa,meacacio,jmgarcia}@ditec.um.es

Abstract. In a near future, the access time to main memory will be of several hundreds of processor cycles. This trend will be especially problematic for multiprocessors, which have been traditionally used to solve scientific problems, as the distance between processor and main memory is typically larger than in the monoprocessor case. Maybe, the most promissing technique to alleviate the increasing distance to memory is having processors with thousands of instructions in flight. One alternative to achieve this is by increasing the size of the *reorder buffer* structure and the number of *physical registers*. However, it is virtually impossible to increase the size of these hardware resources without affecting the clock cycle with current fabrication and integration techniques. Another approach is to aggresively exploit the thread-level parallelism by means of the thread-level speculation paradigm. In this paper, we describe this paradigm and review the state-of-the-art of several implementations based on chip-multiprocessors that use it, covering both research prototypes and commercial products.

1 Introduction

Complex scientific problems have been traditionally solved making use of mid- and large-range multiprocessors. However, in a near future the performance of multiprocessors will be significantly impacted by the increasing distance to memory. This high latency will be mainly motivated by the duration of the processor cycle, which is reduced continuously, and the response time of the logic used to implement the memory [1]. In this scenario, it is essential the development of techniques capable of parallelizing the accesses to main memory, in order to hide memory latency.

Superscalar processors are a common approach when high performance computing is required. These processors try to exploit the *Instruction Level Parallelism* (ILP) by issuing several instructions each cycle. Currently, most multiprocessors use superscalar processors as their processing elements. However, the complexity of the issue logic and the data dependencies will limit the performance of these processors in a near future.

One approach we can consider in order to overcome the limitations of superscalar processors is incrementing the number of entries of the *reorder buffer*

and the size of the associated resources, such as the physical register file and the instruction queues, by a factor of 1000. In this way, we would get one or two thousand instructions in flight, and consequently, the increased memory latencies could be tolerated, as we will see later. However, this technique has a fundamental drawback: increasing the size of the resources in this way would increment the duration of the clock cycle drastically, as the complexity of the logic needed grows with the number of entries of the *reorder buffer* and physical registers implemented.

Very Long Instruction Word (VLIW) processors are another architecture appeared in the last decade. This type of processors uses long instructions made up of several shorter instructions that execute in parallel. Both superscalar and VLIW processors try to extract instruction-level parallelism (ILP) in order to reduce the execution time of the applications; however, typical integer scientific programs have very low ILP, so other forms of parallelization are needed to achieve the performace required by these problems. Therefore, we must consider a couple of alternative high-performance architectures with lower cost and complexity that are suitable for solving medium- and low-sized scientific problems and, at the same time, increase the memory-level parallelism as a way to tolerate the latency of future computers.

Simultaneous MultiThreading (SMT) is a novel technique aimed at maximizing on-chip parallelism. A SMT core allows to issue instructions belonging to several independent threads to multiple functional units each cycle. Using this mechanism, it is possible to outperform the instruction throughput of a single-threaded wide superscalar, increasing the utilization of the execution units. As an example, this technique has been recently adopted in the POWER5 processor design [2].

Finally, Thread-Level Speculation (TLS) allows to extract speculative parallelism from sequential applications. TLS has been successfully used in the parallelization of some benchmarks such as SPEC95 and SPEC2000 suites, what allows their execution on a single-chip multiprocessor [3]. A single-chip multiprocessor [4,5], also called Chip-Multiprocessor or CMP, is a multiprocessor architecture in which several processor cores are integrated in a single-chip. CMPs are a promising approach in order to execute sequential (making use of a parallelization technique as TLS) and parallel scientific applications, providing the benefits of traditional multiprocessors at a lower cost, and incrementing in a natural way the number of instructions executed simultaneously. Thus, the use of TLS and CMPs for sequential applications and CMPs alone for parallel applications appears as a promising approach to solve medium-sized scientific problems and, at the same time, tolerate the high main memory latencies of future computers.

The rest of the paper is organized as follows. Section 2 analyzes the problem of the memory latency and presents how it affects the performance of future multiprocessors, for which two well-known scientific applications are employed. Then, Section 3 describes the state-of-the-art in thread-level speculation and

chip-multiprocessors. Finally, Section 4 presents the conclusions and outlines our ongoing work on this issue.

2 The problem of main memory latency

As we described in Section 1, one of the problems of future computers, especially in the case of multiprocessors, will be the access time to main memory. In order to analyze the importance of this problem, in this section we characterize the execution time of two scientific applications running on a *cache-coherent NonUniform Memory Access Multiprocessor* (also known as cc-NUMA multiprocessor) under two different scenarios: a multiprocessor with a memory latency of 80 cycles and a multiprocessor with a latency of 500 cycles, which tries to approximate the situation that will be found in a near future.

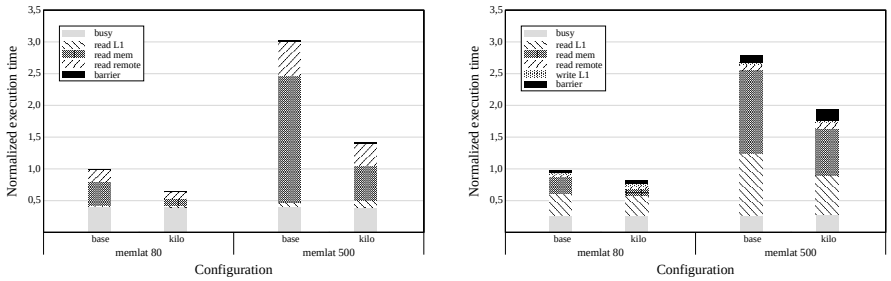
The applications we have chosen are FFT and Radix from the SPLASH-2 benchmark suite [6]. The FFT kernel is a complex 1-D version of the radix-$\sqrt{n}$ six-step algorithm. The data sets consists of the n complex data points to be transformed, and another n complex data points referred to as the *roots of unity*. Both sets of data as organized as $\sqrt{n}$ x $\sqrt{n}$ matrices partitioned so that every processor is assigned a contiguous set of rows which are allocated in its local memory. Radix is a sort iterative kernel, performing one iteration for each radix r digit of the keys. In each iteration, a processor passes over its assigned keys and generates a local histogram. The local histograms are then accumulated into a global histogram. Finally, each processor uses the global histogram to permute its keys into a new array for the next iteration.

To perform the simulations, we have used the Rice Simulator for ILP Multiprocessors (RSIM), a detailed execution-driven simulator [7]. RSIM models an out-of-order superscalar processor, a two-level cache hierarchy, a split-transaction bus on each processor node, and an aggressive memory and multiprocessor interconnection network. We have simulated four different architectures: two of them have a memory access time of 80 cycles, while for the other two accessing main memory takes 500 cycles. Each architecture consists of four identical nodes (each one containing a MIPS R10000-like processor) interconnected by a mesh network. In each group, we have studied a base case in which all the processors are configured with a number of resources typical in superscalar processors (*base*), and a configuration with an unrealistic processor, which has 2048 entries for the ROB and 1024 entries for the load/store queue (*kilo*). These four architectures are aimed at analyzing the influence that the increasing distance to memory will have on a typical multiprocessor in a near future.

Figure 1 shows the execution times that are obtained for each one of the configurations for both FFT and Radix. Execution times have been normalized with respect to those that are obtained for the base case when the 80-cycle main memory is used, which represents a contemporary multiprocessor configuration. As it can be observed, execution times have been split in several components according to the time that the CPU has been busy, the time that the program

has spent reading at the L1 cache, and so on. We only show the components that have a significant contribution.

As we can see in Figure 1(a), the execution time grows by a factor of 3 approximately for FFT when the memory latency is 500 cycles. As observed, now most of the time is spent reading from memory. This degradation is reduced when the *kilo* configuration is used. Having a thousand instructions in flight in FFT can hide most of the degradation introduced by the slower main memory. Similarly, Figure 1(b) presents the results that are obtained for Radix. As it can be derived from it, although the two configurations help to tolerate the increased distance to memory, significant degradation is still observed.



(a) FFT (b) Radix

Fig. 1. Normalized execution times for FFT and Radix.

This simple experiment demonstrates that increased memory latency that will be observed in future multiprocessors will have an important impact on applications' performance and part of the degradation can be hidden by having a large number of instructions in flight. In the next section, two techniques are reviewed that can help us to alleviate this problem by incrementing the number of instructions executing simultaneously.

3 Thread-Level Speculation and Chip-Multiprocessors

Recently, several proposals have been made to take advantage of current integration scale and fabrication techniques. We will focus on those proposals aimed at increasing the performance of the microprocessor and, in some extent, reducing the penalty incurred when accessing main memory.

The first approach consists in exploiting the ILP showed by certain sequential and parallel applications. While some authors propose to increase the number of instructions issued each cycle [8], other authors present out-of-order commit processors as an alternative design of a processor with several thousands of instructions in-flight [9].

Other proposals exploit the *Thread-Level Parallelism* (TLP) concept. This type of parallelism is based on executing several control threads simultaneously, achieving similar results to those obtained with a processor with a very large instruction window, but having several small instruction windows (one per thread) working together in the same application.

Although TLP is a good choice to parallelize sequential applications, data and control dependences (usually found in integer scientific applications) hurt the performance that can achieved with this technique. One way to increase the multithread parallelism performance provided by TLP is by using the *Thread-Level Speculation* (TLS) paradigm, which attempts to split the program into several threads in a speculative way starting from a spawning point (which is usually a data operation or a subroutine call).

Simultaneous MultiThreading (SMT) architecture [10] allows a more efficient use of the issue width of superscalar processors by executing instructions of multiple control threads concurrently. The selection of these instructions is performed dynamically, so processor's resources are generally used better. At the same time, the instruction latency incurred in a cache miss or motivated by a true data dependence is hidden through the execution of instructions available at another thread. An SMT processor behaves like a traditional superscalar processor when the application does not contain several threads.

Finally, the chip-multiprocessing approach proposes to integrate several simple processor cores onto a single chip. A CMP can be used to execute multiprogrammed scientific workloads, as the independent tasks execute in parallel in several processors. On the other hand, if the execution time of a sequential program is to be reduced, it is necessary to apply a parallelization technique, which will be typically TLS. CMPs are simpler and easier to design and implement than SMT processors (which are more flexible), so they constitute a very promissing architectural option of making use of the several billions of transistors that will be packed into a single chip in a near future.

One of the first chip-multiprocessor prototypes suggested in the literature is the *Hydra* single-chip multiprocessor [4]. This CMP integrates four cores similar to the Alpha 21064, a shared L2 cache of 256 KB and a crossbar to access the cache. Its performace is compared that achieved by a superscalar processor similar to the MIPS R10000 in [11]. With benchmarks belonging to the SPECint92, SPECint95 and SPECfp95 suites, the speed-up obtained with the CMP ranges from 50% to 100%. Other papers of the same authors focus on control and data speculation required to maximize the performance in this type of architecture [12,13]. In [13], Hammond *et al.* describe the whole implementation of the thread-level speculation support included in the Hydra CMP (basically a set of handlers for the control speculation and some changes in the L2 cache).

The *Piranha* system [5] is a research prototype developed at Compaq that exploits chip multiprocessing by integrating eight simple Alpha processor cores along with a two-level cache hierarchy onto a single chip. Piranha also integrates on-chip functionality to allow for scalable processor configurations to be built in a modular fashion. This CMP has been specifically designed to execute commer-

cial workloads such as On-Line Transaction Processing (OLTP) or Decision Support Systems (DSS). The simulation results that the authors present show that while each Piranha processor core is slower than a next-generation processor, the integration of eight cores onto a single chip allows the CMP to outperform the complex next-generation processor by up to 2.9 times on workloads such as OLTP.

The *STAMPede project* [14] investigates the architectural, compiler, and OS support necessary to effectively exploit single-chip multiprocessors. In [15], the authors propose the *Thread-Level Data Speculation* (TLDS) technique, in which the memory access operations (usually the loads) are executed speculatively but in different threads. The authors simulate applications of the SPEC92, SPEC95 and NAS Parallel suites over a single-chip multiprocessor composed of four cores. Their technique is extended in [3,16] to make it feasible for medium and large-scale machines, and in [17,18] to improve the value communication between threads. In these papers, the authors simulate different benchmarks from the SPEC2000 suite over a multi-node architecture in which every node is a CMP with one, two, four or eight processor cores.

Multiscalar processors [19] use an implementation paradigm designed to extract ILP from programs written in high-level programming languages. This was one of the first proposals related to thread-level parallelism, and the authors exposed here the need to introduce speculation in order to improve the performace achieved with TLP. In general terms, a multiscalar processor is composed of several processing units with a sequencer which assigns tasks (a set of instructions) to the processing units. The process of parallelizing the applications relies mainly on the hardware, in contrast to the STAMPede project, where part of the responsibility is assigned to the compiler.

More recently, Zilles and Sohi presented the *Master/Slave Speculative Parallelization* (MSSP) as a paradigm which improves the execution rate of sequential programs by parallelizing them speculatively [20]. With this paradigm, a *master* processor executes an approximate version of the program speculatively, in order to calculate a couple of values that the whole program will use. This values are checked by the *slave* processors, that execute the original program. Each slave processor uses the values calculated by the master in order to validate the values of the next task, so the whole execution is validated inductively.

The *MIT Multi-ALU* processor (MAP) [28] is designed to extract thread-level parallelism, with a number of instructions between 50 and 1000 belonging to each thread. This system provides three processors integrated onto the chip and support for fast communication and synchronization between them. A thread executing in a processor can write a register belonging to the other two processors. The threads synchronize themselves by blocking in the destination register of a remote write or by executing a fast barrier instruction. The MAP chip is divided into three clusters of execution, a unified cache split into three banks, a memory interface and a communication subsystem with two network interfaces and a router. It is possible to connect several MAP chips directly or using a bi-dimensional mesh in order to construct the M-Machine multicomputer.

Another approach was proposed in [21]. It is based on a processor able to execute several threads speculatively in parallel from a sequential application over a *clustered architecture*. In a clustered architecture some critical components are organized in smaller processing units named *clusters* [22]. The proposed architecture consists of a number of thread units, being each unit very similar to a superscalar processor, with an additional set of registers used to communicate values between independent thread units. The thread-level parallelism is extracted at execution time, so there is no need to modify the compiler nor the ISA of the architecture. In [23], the same authors propose a mechanism based on *profiling* to detect what are the best instructions to create new threads and where the execution of the new thread has to start. This mechanism outperforms by approximately 20% other similar proposals aimed at parallelizing speculatively a sequential program.

The *Superthreaded architecture* [26,27] exploits the task-level parallelism with multiple control threads. In its general form, a superthreaded processor consists of several thread processing units connected together through a unidirectional ring. The execution of each thread is split in several stages, each of which performs a specific function. This type of architecture relies on the compiler to extract thread-level parallelism.

Finally, other authors propose to extract thread-level parallelism from the binary code of sequential applications. This is interesting in those cases in which the source code of the application is not available, as a chip-multiprocessor architecture is only effective if the program is compiled in order to allow the speculative execution of parallel threads. In [24,25], the authors simulate applications from the SPEC92 and SPEC95 benchmark suites over a CMP with 4 MIPS R10000 processors, achieving an IPC 1.8 times greater than with a superscalar processor with an issue-width of 12 instructions. The *Atlas* single-chip multiprocessor [29] is another project that focuses on aggressive speculation techniques designed to allow the dynamic parallelization of the binary code of sequential applications with irregular data structures. The chip integrates eight cores sharing a global L2 cache and a control and value predictor. The processors communicate with the global structures through two shared buses: one is used to access the L2 cache and the other to communicate control and data predictions. Regarding the processors, they are communicated using a pipelined bidirectional ring. The processor architecture is similar to the Alpha 21164 one.

While all the previous proposals are academical prototypes, there are several commercial implementations and future releases of single-chip multiprocessors. The IBM POWER4 chip [30] was the first commercial implementation of a CMP. The chip has two processors on board, including the various execution units and the split first level instruction and data caches. The two processors share a unified second level cache (implemented as three separate cache controllers) through a Core Interface Unit (CIU), a crossbar switch between the L2 and the two processors. Four POWER4 chips can be packaged on a single module to form an 8-way Symmetrical MultiProcessor (SMP). Four such modules can be

interconnected to form a 32-way SMP. This can be accomplished via five primary interfaces integrated onto the chip.

Recently, HP has released the PA-8800 processor, that integrates two PA-8700 processor cores. Each core has private first level instruction and data caches with a size of 768 KB each one. Regarding the L2 cache, it is a 32 MB off-chip shared cache. The cache is 4-way set associative and with a line size of 128 bytes.

The Sun UltraSparc IV system [31] is a two-cored version of the UltraSparc III. The size of the off-chip L2 cache is 16 MB, although each core is able to access only 8 MB. The cache line size is 128 bytes, 4 times smaller than the UltraSparc III one. In a second generation (expected in 2005), the on-chip L2 cache will have 4 MB, and will be shared by the two processors.

There are also two implementations that will be released in a near future. The first of them is the IBM POWER5 [2], a natural extension to the POWER4. The main differences with respect to the POWER4 will be the increase of the shared L2 cache, with a total size of 1.92 MB, and the off-chip L3 cache, with a size of 36 MB. This processor will be able to manage a physical memory of 1 TByte. It has been also implemented the *Simultaneous MultiThreading* (SMT) technique in each core, so it is possible to have two *virtual* processors for each *real* processor.

Finally, the Fujitsu SPARC64 VI processor will be based on the SPARC64 V core, but integrating two cores onto the chip. The processor will have first level instruction and data caches of 128 KB for each core, while the unified second level cache of 6 MB will be shared and integrated onto the chip. The clock frequency will be in a first version of 2.4 Ghz, and the announced release date is the beginning of 2006.

4 Conclusions

In a near future, the access time to main memory will be of several hundreds of cycles. This trend will be especially problematic for multiprocessors, which have been traditionally used to solve complex scientific problems, as the distance between processor and main memory is typically larger than in the monoprocessor case.

One way of tolerating part of the distance to memory problem is by increasing the number of instructions that are simultaneously accessing memory. In the context of a multiprocessor made up of several superscalar microprocessors, this could be achieved by scaling up the number of resources in critical processor's structures such as the ROB or the physical register file. However, this would have negative effects in the clock cycle duration.

Thread-level speculation appears as a promising approach in order to parallelize sequential scientific applications, and at the same time, solve in some extent the problem of the memory latency. In this paper, we have reviewed several of the proposals made in this field, corroborating that this technique is very suitable when these sequential applications are to be executed over a single-chip multiprocessor. The importance of the CMP architecture has grown in the last

years, and the number of current and future commercial implementations indicate that chip-multiprocessors will be one of the most important architectures in the future.

At this time, we are developing a CMP simulator based on RSIM. We plan to study memory behavior and contention at some resources that is motivated by the use of several processors in a single die. In many of the proposals we have reviewed in this paper, the processors share a monolithic L2 cache through a bus which connects the private L1 caches to this single L2 cache. However, the integration into a single chip of eight or more processor cores could colapse a traditional bus or cache structure, and clever memory organizations could be required.

References

1. Agarwal, V., Hrishikesh, M. S., Keckler, S. W., Burger, D.: "Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures". In: 27th International Symposium on Computer Architecture, Vancouver, Canada (2000) 248–259
2. Kalla, R., Sinharoy, B., Tendler, J. M.: "Simultaneous Multithreading Implementation in POWER5". In: 2003 Hotchips, Standford CA (2003)
3. Steffan, J. G., Colohan, C. B., Zhai, A., Mowry, T. C.: "A Scalable Approach to Thread-Level Speculation". In: 27th International Symposium on Computer Architecture, British Columbia (2000) 1–12
4. Hammond, L., Hubbert, B. A., Siu, M., Prabhu, M. K., Chen, M., Olukotun, K.: "The Stanford Hydra CMP". IEEE Micro **20** (2000) 71–84
5. Barroso, L. A., Gharachorloo, K., McNamara, R., Nowatzyk, A., Qadeer, S., Sano, B., Smith, S., Stets, R., Verghese, B.: "Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing". In: 27th International Symposium on Computer Architecture, Vancouver, Canada (2000) 282–293
6. Woo, S. C., Ohara, M., Torrie, E., Singh, J. P., Gupta, A.: "The SPLASH-2 programs: Characterization and Methodological Considerations". In: 22nd International Symposium on Computer Architecture, S. Margherita Ligure (1995) 24–36
7. Hughes, C. J., Pai, V. S., Ranganathan, P., Adve, S. V.: "RSIM: Simulating Shared-Memory Multiprocessors with ILP Processors". IEEE Computer **35** (2002) 40–49
8. Patt, Y., Patel, S., Evers, M., Friendly, D., Stark, J.: "One Billion Transistors, One Uniprocessor, One Chip". Volume 30. (1997) 51–57
9. Cristal, A., Ortega, D., Llosa, J., Valero, M.: "Out-of-Order Commit Processors". In: 10th International Symposium on High-Performance Computer Architecture, Madrid, Spain (2004) 48–59
10. Tullsen, D. M., Eggers, S. J., Levy, H. M.: "Simultaneous Multithreading: Maximizing On-Chip Parallelism". In: 22nd International Symposium on Computer Architecture, S. Margherita Ligure (1995) 392–403
11. Olukotun, K., Nayfeh, B. A., Hammond, L., Wilson, K., Chang, K.: "The Case for a Single-Chip Multiprocessor". In: 7th International Symposium on Architectural Support for Parallel Languages and Operating Systems, Cambridge (1996) 2–11
12. Oplinger, J., Heine, D., Liao, S. W., Nayfeh, B. A., Lam, M. S., Olukotun, K.: "Software and Hardware for Exploiting Speculative Parallelism with a Multiprocessor". Technical report, Stanford University Computer Systems Lab (1997)

13. Hammond, L., Willey, M., Olukotun, K.: "Data Speculation Support for a Chip Multiprocessor". In: 8th International Symposium on Architectural Support for Parallel Languages and Operating Systems, San Jose, CA (1998) 58–69
14. The STAMPede Project: `http://www-2.cs.cmu.edu/~stampede`
15. Steffan, J. G., Mowry, T. C.: "The Potential for Using Thread-Level Data Speculation to Facilitate Automatic Parallelization". In: 4th International Symposium on High-Performance Computer Architecture, Las Vegas, Nevada (1998) 2–13
16. Steffan, J. G., Colohan, C. B., Mowry, T. C.: "Extending Cache Coherence to Support Thread-Level Data Speculation on a Single Chip and Beyond". Technical report, School of Computer Science, Carnegie Mellon University (1998)
17. Steffan, J. G., Colohan, C. B., Zhai, A., Mowry, T. C.: "Improving Value Communication for Thread-Level Speculation". In: 8th International Symposium on High-Performance Computer Architecture, Cambridge, MA (2002) 65–75
18. Zhai, A., Colohan, C. B., Mowry, T. C., Steffan, J. G.: "Compiler Optimization of Scalar Value Communication Between Speculative Threads". In: 10th International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA (2002) 171–183
19. Sohi, G. S., Breach, S. E., Vijaykumar, T. N.: "Multiscalar Processors". In: 22nd International Symposium on Computer Architecture, S. Margherita Ligure, Italy (1995) 414–425
20. Zilles, C., Sohi, G. S.: "Master/slave Speculative Parallelization". In: 35th International Symposium on Microarchitecture, Istambul, Turkey (2002) 85–96
21. Marcuello, P., González, A.: "Clustered Speculative Multithreaded Processors". In: 13th ACM International Conference on Supercomputing, Rhodes, Greece (1999) 365–372
22. Farkas, K. I., Chow, P., Jouppi, N. P., Vranesic, Z.: "The Multicluster Architecture: Reducing Cycle Time Through Partitioning". In: 30th International Symposium on Microarchitecture, Raleigh, North Carolina (1997) 149–159
23. Marcuello, P., González, A.: "Thread-Spawning Schemes for Speculative Multithreading". In: 8th International Conference on High-Performance Computer Architectures, Boston, Massachussets (2002) 55–64
24. Krishnan, V., Torrellas, J.: "Hardware and Software Support for Speculative Execution of Sequential Binaries on a Chip-Multiprocessor". In: 12th International Conference on Supercomputing, Melbourne, Australia (1998) 85–92
25. Krishnan, V., Torrellas, J.: "A Chip-Multiprocessor Architecture with Speculative Multithreading". IEEE Transactions On Computers **48** (1999) 866–880
26. Tsai, J. Y., Yew, P. C.: "The Superthreaded Architecture: Thread Pipelining with Run-Time Data Dependence Checking and Control Speculation". In: 1996 International Conference on Parallel Architectures and Compilation Techniques, Boston, MA (1996) 35–46
27. Tsai, J. Y., Huang, J., Amlo, C., Lilja, D. J., Yew, P. C.: "The Superthreaded Processor Architecture". In: IEEE Transactions On Computers. Volume 48. (1999) 881–902
28. Keckler, S. W., Dally, W. J., Maskit, D., Carter, N. P., Chang, A., Lee, W. S.: "Exploiting Fine-Grain Thread Level Parallelism on the MIT Multi-ALU Processor". In: 25th International Symposium on Computer Architecture, Barcelona, Spain (1998) 306–317
29. Codrescu, L., Wills, S., Meindl, J.: "Architecture of the Atlas Chip-Multiprocessor: Dynamically Parallelizing Irregular Applications". IEEE Transactions On Computers **50** (2001) 67–82

30. Tendler, J. M., Dodson, S., Field, S., Sinharoy, H. L. B.: "POWER4 System Architecture". Technical report, IBM Server Group (2001)
31. Sun Microsystems: "UltraSPARC4 Processor. Architecture Overview". Technical Whitepaper (2004)