

# Real PRAM Programming

Wolfgang J. Paul, Peter Bach\*, Michael Bosch\*, Jörg Fischer\*\*,  
Cédric Lichtenau\*\*\*, Jochen Röhrig\*\*\*

The work reported here was done while all the authors were affiliated with  
Computer Science Department, Saarland University  
Postfach 151150, 66041 Saarbrücken, Germany <sup>†</sup>  
<http://www-wjp.cs.uni-sb.de/>

**Abstract.** The SB-PRAM is a parallel architecture which uses i) multi-threading in order to hide latency, ii) a pipelined combining butterfly network in order to reduce hot spots and iii) address hashing in order to randomize network traffic and to reduce memory module congestion. Previous work suggests that such a machine will efficiently simulate shared memory with constant access time independent of the number of processors (i.e. the theoretical PRAM model) provided enough threads can be kept busy. A prototype of a 64 processor SB-PRAM has been completed. We report some technical data about this prototype as well as performance measurements. On all benchmark programs measured so far the performance of the real machine was at most 1,37 % slower than predicted by simulations which assume perfect shared memory with uniform access time.

## 1 Introduction

Successful commercial parallel machines have to use standard processors because their performance/price ratio has to grow with that of the standard processors. Standard processors use caches in order to reduce memory latency and to overcome the bandwidth bottleneck imposed by the pins of the processor chip. Due to the caches a local programming style is strongly rewarded by successful commercial machines and techniques to maintain locality are highly developed.

On the other hand asymptotically work optimal emulations of PRAMs, where locality plays no role whatsoever, are well known in the theoretical literature. For a survey see [1]. They are based on interleaved multithreading and for  $p$  processors they require networks with cost  $O(p \log p)$ . With respect to a gate level model of hardware a construction with small constants was given in [2]. Based on this construction a prototype with 64 processors was developed in the SB-PRAM project over the last decade.

---

\* currently ETAS GmbH, Stuttgart, {peter.bach,michael.bosch}@etas.de

\*\* after July 2002: Jörg Schmittler, jofis@graphics.cs.uni-sb.de

\*\*\* currently IBM Deutschland Entwicklung GmbH, Böblingen Lab,  
{lichtenau,roehrig}@de.ibm.com

<sup>†</sup> This work was supported by the German Science Foundation (DFG) under contract SFB 124, TP D4.

The foremost goal of the project was to make the PRAM programming style a reality on hardware which would scale say to hundreds of processors. The construction of a research prototype with 64 processors was completed. This paper contains technical data about the hardware as well as measured performance figures from this prototype confirming that the behaviour of the real hardware and the abstract user model of a PRAM differ by less than 1.37 % on all runs observed.

The paper is organized in the following way. In section 2 the architecture of the SB-PRAM and its relation to other architectures is sketched. In sections 3 and 4 the hardware and system software are described. Section 5 contains performance figures. We draw conclusions in section 6.

## 2 SB-PRAM architecture

Table 1 lists for various shared memory machines how the 4 basic problems of hot spots, module congestion, network traffic and latency are addressed. It can serve as a rough definition of the SB-PRAM architecture and it shows the relation of the SB-PRAM to other architectures.

1. In the SB-PRAM architecture hot spots are avoided by a combining butterfly network like in the NYU Ultracomputer [3] and the RP3 [4] machine.
2. Module congestion is made unlikely by address hashing with a randomly chosen bijective linear hash function. Nontrivial hash functions are also employed in the RP3 and TERA machines [5].
3. The randomization of network traffic and its positive effects on network delays come for free with address hashing.
4. Like HEP [6] and TERA the SB-PRAM does not use a cache at the processors hereby missing the positive effect caches have on network bandwidth. Instead the latency is hidden by multi-threading. Every processor can support a number of threads which grows with  $L(p) = c \cdot \log p$  for a constant  $c$  depending only on the relative speed of processors, network and memory. The intention is to hide almost completely the latency of the network as long as all threads can be kept busy.

For details see [2, 7] and [8].

	multi threading	address hashing	combining	caches
HEP	✓	-	-	-
TERA	✓	✓	-	-
T3E	✓	✓	at memory	✓
RP3	-	✓	✓	✓
Ultra	-	✓	✓	✓
SB-PRAM	✓	✓	✓	-

**Table 1.** Techniques used in order to avoid network congestion

Let  $L(p) \geq 3 \cdot \log p$  be the number of threads per processor on an SB-PRAM. Simulations [2] show that except in rare situations of network and memory module congestion no time at all is lost during the emulation of  $p \cdot L(p)$  threads on a machine with  $p$  processors. If  $t$  is the cycle time of the processors, then the system exhibits *almost exactly* the behavior of a PRAM with  $p \cdot L(p)$  processors and cycle time  $t \cdot L(p)$ . Note that this is a bulk synchronous computation with latency  $L(p)$  and throughput  $g = 1$  in the sense of [9, 10].

Moreover the constants for the hardware cost and cycle time are very small, at least in a gate level model of hardware [2]. Although the cost of the network grows asymptotically faster than the cost of the processors, the number of gates in the network of a 64 (resp. 128) processor machine is only 26 % (resp. 28 %) of the total cost of the machine [11] (assuming the cost of memory modules equals the cost of processors).

### 3 Hardware

The hardware of the SB-PRAM prototype is surveyed in [12]. Theses and papers documenting the design can be found at our web site:

<http://www-wjp.cs.uni-sb.de/projects/sbpram/>

The hardware effort had three major parts: chip design, board design and hardware integration.

#### 3.1 Chips and cycle time

The chips designed were

1. The processor [13]. It has a Berkeley RISC instruction set and a 32 bit floating point unit manipulating normal numbers in the IEEE 754 format. The simplest context switching scheme supporting the PRAM programming style was realized: simple interleaved context switching. The processor was designed for machines with up to  $p = 128$  processors. This required at least  $3 \cdot 7 = 21$  contexts. For machines of different sizes, the processor can be configured to store 8, 16, 24 or 32 contexts. The register files storing up to 32 copies of the processor's register set were realized by external SRAM. The processor was designed in the year 1994 in  $0.7 \mu\text{m}$  technology and fabricated by Motorola. It has about 80,000 gate equivalents, 230 signal pins and a cycle time of 16 ns [13, 14].
2. The sorting array [15]. One sorting array per processor is used. In each round it sorts the packets entering the network from the processor by destination address. This is part of the hardware support for Ranade's routing scheme [16, 17]. The sorting array also supports multiprefix operations. The sorting array was designed in the year 1995 in  $0.7 \mu\text{m}$  technology and fabricated by Motorola. It has 37,000 gate equivalents, 175 signal pins and a cycle time of 25 ns [15].

3. The network chip [18]. It also supports multiprefix operations for the operators  $\wedge$ ,  $\vee$ ,  $+$  and *max* and implements Ranade's routing scheme. In order to save pins each networkchip contains 4 (interconnected) halves of network nodes [19]. The chip was manufactured in the year 1996 in 0.8  $\mu\text{m}$  technology by Thesis. It has 67,000 gate equivalents and a worst case cycle time of 37 ns.

In the prototype the network chips work at a cycle time of 37 ns, thus the network can be clocked with 27 MHz. The processors are operated at 1/4 of the network frequency, i.e. at 6.75 MHz. The prototype is based on old low cost technology. A redesign based on 1996's technology, where the processors are clocked at 93.6 MHz is sketched in [20]. Running the processors at 1 GHz would require optical links between boards. Moreover the data rates between chips would require very advanced technology e.g. optical I/O at chip boundaries.

### 3.2 Boards

The boards designed were

1. the processor card [21, 22] with 32 MByte local memory for programs and a PCI-bus interface [23]. It also contains two SCSI-controllers to connect hard disks directly to the processor card. It has an area of 802  $\text{cm}^2$ .
2. the memory card. Each memory module [24, 25] contains 64 MByte in 4 banks of memory for each processor. Every memory card contains two memory modules. The card has an area of 743  $\text{cm}^2$ .
3. the network card [26] contains 8 network chips. It has an area of 1464  $\text{cm}^2$ .
4. various small cards for the global clock and reset distribution, for the interconnection of back-planes and for the connection to the host computer. They form a small portion of the machine.

### 3.3 Geometry of wiring

Most interconnections between boards were realized as ribbon cables. Transmission by ECL or optical fibres would have been possible [26] but not affordable. In order not to repeat painful experiences from earlier projects, theorems on the geometrical arrangement of boards, connectors and cables were proven extremely early in the project [18, 27]. These theorems bound the length of wires in order to make timing analysis possible, and they show how to remove single boards without disassembling large portions of the machine. Figure 1 shows a part of the wires arranged as prescribed by the theorems. The whole machine has 1.3 km of ribbon cables with a total of 166 km of wires.

### 3.4 Relative cost of processors and network

Figure 1 shows the entire prototype with 64 processors with the network boards, memory boards and processor boards marked separately. We show that the network boards occupy roughly 1/3 of the total printed circuit board area of the machine:

Let  $A_{PM} = 802 \text{ cm}^2 + 371.5 \text{ cm}^2$  be the area of a processor and half a memory board. Let  $A_N = 1464 \text{ cm}^2$  be the area of a network board. A network board contains 8 chips and the equivalent of 16 network nodes. A machine with  $p$  processors needs  $p \cdot \log p$  network nodes, i.e.  $(p/16) \cdot \log p$  network boards (for appropriate values of  $p$ , e.g. 64). Thus a machine with  $p$  processors occupies an area of

$$\begin{aligned} A(p) &\approx p \cdot A_{PM} + ((p/16) \cdot \log p) \cdot A_N. \\ &= 1173.5 \cdot p + 91.5 \cdot p \cdot \log p \end{aligned}$$

The relative size of the network is  $(91.5 \cdot p \cdot \log p) / A(p)$ . For  $p = 64$  this evaluates to about 32 %. This is slightly higher than the 26 % estimated by the gate model [11]. The number of processors  $p$  for which the network occupies half the total area is determined by

$$1173.5 \cdot p = 91.5 \cdot p \cdot \log p$$

which is between 4096 and 8192.

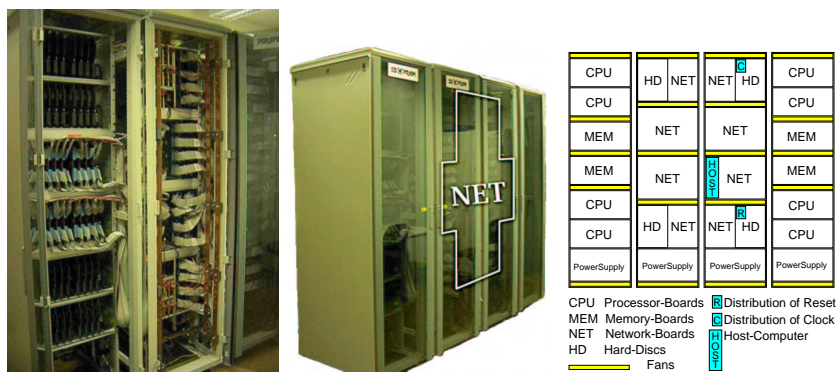


Fig. 1. left half of the 64-SB-PRAM and the complete 64-SB-PRAM

### 3.5 System integration and debugging

Debugging the 4 processor machine in 1996 took unreasonable amounts of time because not enough testability was designed into the boards. Therefore all boards were redesigned using JTAG [28] wherever possible. Test procedures were specified and theorems about the error coverage were proven [22, 25, 29]. The error model includes shorts (AND, OR and dominating errors), opens and dynamic fault models for timing violations, ground bounce and cross talk [29].

Technical difficulties in the design of the test procedures came from 3 sources of which the last two are relevant for future designs:

1. The chips designed earlier in the project have no JTAG paths.
2. The clock lines and control lines of the JTAG circuitry themselves have to be tested [25].
3. given the size of the machine the time for running the test procedures became a serious issue. A detailed self test of the whole machine takes 4 hours [29].

## 4 System Software

The system software has 4 major components: 1) A UNIX-like operating system which heavily uses the multiprefix operations of the SB-PRAM for memory management. 2) A compiler for the Language FORK[8, 30]. 3) A compiler for C (later C++) extended by the concept of shared variables. 4) A library of communication primitives for the C dialect. Roughly speaking it included the P4 macros [31] needed for the direct compilation of the SPLASH benchmarks and the communication primitives from the NYU ultracomputer project [32], most notably the parallel job queue ported to the SB-PRAM. The migration was reasonably straight forward; on the SB-PRAM the primitives have *predictable* run times (which is not surprising because they were inspired by the theoretical literature in the first place).

## 5 Measured Run Times

Before completion of the hardware numerous applications were implemented on a simulator, which simulated the *idealized user model* of an SB-PRAM on an instruction by instruction basis. Recall that for  $p$  processors with cycle time  $t$  and  $L(p)$  threads per processor the user is intended to see a priority CRCW PRAM with  $p \cdot L(p)$  processors and cycle time  $t \cdot L(p)$ , Berkeley RISC instruction set and multi-prefix operations processing one instruction per cycle. The simulations allowed to make optimistic predictions of run times and speed-ups and to compare the PRAM programming style with the common programming style which exploits locality. For a survey see [33].

With the hardware up and running the predictions can of course be checked against reality. We compare here the predicted and measured run times of 4 programs from the benchmark suites SPLASH and SPLASH 2: Radiosity, Locus Route, PTHOR and MP3D. All speed-ups reported below are about *absolute* speed-ups, i.e. the run time of a parallel program with  $p$  processors is compared with the run time of the *fastest known sequential* program running on a single processor. On the SB-PRAM sequential computation is performed by using a single thread on a single processor, but that of course makes only use of  $1/L(p) = 1/32$  of the power of a processor. In order to determine speed-up in a fair way, we measure the run time  $t_1$  of 1 thread on 1 processor, compare with time  $t_{32p}$  using 32 threads on each of  $p$  processors and scale by 32:  $speedup = (1/32) \cdot (t_1/t_{32p})$ .

### 5.1 Measured Speed-Ups

Figure 2 concerns the benchmarks Radiosity and Locus Route. It compares the speed-up of non optimized and optimized PRAM programs with that on the (cache based) DASH machine [34]. Both programs parallelize well on cache based machines. Migrating such programs in a naive way to the SB-PRAM leads to *deteriorated parallel efficiency*: with  $p$  PRAM processors one has to keep at least  $3p \cdot \log p$  threads busy, and naive migration tends not to achieve this. If one

optimizes the programs and recodes portions of the programs using multiprefix operations and the powerful communication primitives one regains most or all of the lost ground.

Figure 3 shows speed-ups for PTHOR. The only good news here is, that a nontrivial speed-up is reached at all on a real machine. The discrete event simulation PTHOR is difficult to parallelize for 3 reasons: i) access patterns to memory are non local, ii) the overhead for parallelization is enormous. The parallel program for 1 processor is about 5.4 times slower than the fastest sequential program, iii) the number of available tasks is limited. Reasons ii) and iii) are problematic for the SB-PRAM too.

Figure 3 also shows speed-ups for the particle simulation MP3D. In the simulation the particles mix in an irregular fashion. Because every particle is always processed by the same processor the handling of collisions leads to non local memory accesses. Using a cache protocol optimized for migratory sharing the MIT Alewife machine [35] handles this situation remarkably well.

This seems like the perfect benchmark for the SB-PRAM which is designed to handle non local access pattern effortlessly. Yet the parallel efficiency for 64 processors is still far from 100 %, at least with 40 000 particles. The reason is, that the large number of threads of a 64 processor SB-PRAM can only be kept busy for the central tasks of moving and colliding particles. But other portions of the program with only hundreds of parallelizable tasks begin to dominate the run time. For details see [36].

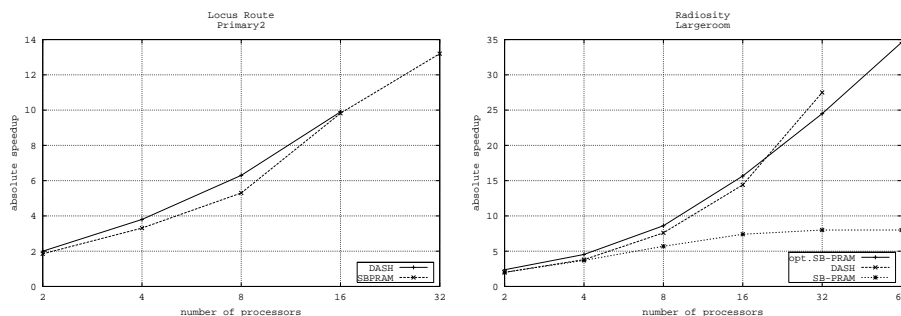
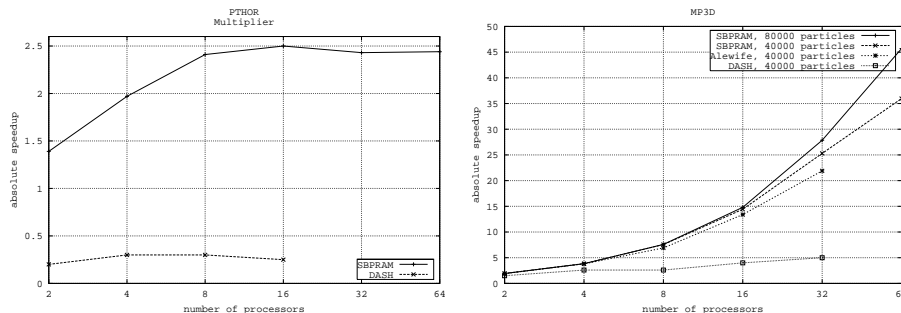


Fig. 2. Results: Locusroute and Radiosity

## 5.2 Overhead due to congestion

Consider a run of the simulator with  $r$  rounds of computation for each of  $L(p)$  user threads per processor. Then with cycle time  $t$  the prediction for the wall clock time  $w$  is  $w' = r \cdot L(p) \cdot t$ . This prediction is optimistic and assumes that latency due to congestion can be hidden completely. The quantity overhead  $ovh = (w - w')/w' = w/(r \cdot L(p) \cdot t) - 1$  measures the relative overhead incurred due to congestion in the network and at the memory modules which cannot be hidden. We report measurements of this quantity which were performed with various combinations of processor numbers and threads per processor. Runs were performed with 2 hash factors  $h$  for address hashing:



**Fig. 3.** Results: PTHOR and MP3D

1. with the trivial hash factor  $h = 1$ . In this case the maximal overhead observed was 26% for 16 processors and 223% for 64 processors. Thus switching off the address hashing leads to significant deterioration of the performance due to congestion.
2. with hash factor  $h = 0x826586F3$  generated by a random number generator. Here the result is very encouraging: in 771 runs for 16 processors the maximal overhead observed was under 1 %. In 324 runs for 64 processors the maximal overhead observed was 1.37 %. Thus deterioration of performance due to congestion was almost absent for practical purposes.

Note that the optimized application programs make heavy use of communication primitives like a fine grained parallel job queue which put a heavy load on the communication network and the memory modules.

## 6 Conclusion

The prototype of a 64 processor SB-PRAM has been completed. The cost of the construction grows asymptotically with  $p \log p$ . But in the particular (by now old) technology used, the network boards would contain less than half the area even for a machine with 4096 processors. Thus the construction is reasonably scalable.

The results of section 5.2 show that the PRAM simulation using the Ranade routing scheme is performing for 64 processors with extremely low overhead. Thus running PRAM algorithms with a run time matching the theoretical run time to within a few percent is a reality. Performance deteriorates if the address hashing is switched off.

The results of section 5.1 show that on scientific application programs the strength of the SB-PRAM in handling non local memory accesses translates at least in some situations like MP3D (and PTHOR) into a measurable gain in efficiency. The large number of threads however, which need to be kept busy, is an issue requiring careful attention.

## References

1. Valiant, L.G.: General Purpose Parallel Architectures. In van Leeuwen, J., ed.: Handbook of Theoretical Computer Science, Vol. A. Elsevier Science Publishers and MIT Press (1990) 943–971
2. Abolhassan, F., Keller, J., Paul, W.J.: On the Cost-Effectiveness of PRAMs. In: Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing. (1991) 2–9
3. Gottlieb, A., Grishman, R., Kruskal, C.P., McAuliffe, K.P., Rudolph, L., Snir, M.: The NYU Ultracomputer – Designing an MIMD Shared Memory Parallel Computer. In: IEEE Transactions on Computers, C-32(2). (Feb 1983) 175–189
4. Pfister, G.F., Brantley, W.C., George, D.A., Harvey, S.L., Kleinfelder, W.J., McAuliffe, K.P., Melton, E.A., Norton, V.A., Weiss, J.: The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture. In: International Conference on Parallel Processing, Los Alamitos, Ca., USA, IEEE Computer Society Press (1985) 764–771
5. Alverson, R., Callahan, D., Cummings, D., Koblenz, B., Porterfield, A., Smith, B.: The Tera computer system. In: Proceedings of the 1990 International Conference on Supercomputing. (1990) 1–6
6. Smith, B.J.: Architecture and applications of the HEP multiprocessor computer system. SPIE Real-Time Signal Processing IV **298** (1981) 241–248
7. Abolhassan, F., Keller, J., Paul, W.J.: On the Cost-Effectiveness of PRAMs. In: Acta Informatica 36. Springer-Verlag (1999) 463–487
8. Keller, J., Kessler, C.W., Träff, J.L.: Practical PRAM Programming. Wiley-Interscience Series on Parallel and Distributed Computing (2000)
9. Valiant, L.G.: Bulk-synchronous parallel computers. In: Parallel Processing and Artificial Intelligence. (1989) 15–22
10. Valiant, L.G.: A Bridging Model for Parallel Computation. In: Communications of the ACM. (August 1990) 103–111
11. Abolhassan, F.: Vergleich von Parallelen Maschinen mit gemeinsamen und verteilten Speichern. PhD thesis, Universität des Saarlandes, Saarbrücken (1994)
12. Abolhassan, F., Drefenstedt, R., Keller, J., Paul, W.J., Scheerer, D.: On the Physical Design of PRAMs. In: Computer Journal 1993 36(8). (1993) 756–762
13. Scheerer, D.: Der Prozessor der SB-PRAM. PhD thesis, Universität des Saarlandes, Saarbrücken (1995)
14. Keller, J., Paul, W.J., Scheerer, D.: Realization of PRAMs: Processor Design. In: Proc. WDAG '94, 8th Int. Workshop on Distributed Algorithms. Springer Lecture Notes in Computer Science No. 857 (1994) 17–27
15. Göler, T.: Der Sortierknoten der SB-PRAM. Master's thesis, Universität des Saarlandes, Saarbrücken (1996)
16. Ranade, A.G.: How to Emulate Shared Memory. In: Journal of Computer and System Sciences 42. (1991) 307–326
17. Ranade, A.G., Bhatt, S.N., Johnson, S.L.: The Fluent Abstract Machine. In: Proceedings of the 5th MIT Conference on Advanced Research in VLSI, Cambridge, MA, MIT Press (1988) 71–93
18. Walle, T.: Das Netzwerk der SB-PRAM. PhD thesis, Universität des Saarlandes, Saarbrücken (1997)
19. Cross, D., Drefenstedt, R., Keller, J.: Reduction of Network Cost and Wiring in Ranade's Butterfly Routing. In: Information Processing Letters, vol. 45 no. 2. (1993) 63–97

20. A. Formella, J. Keller, T.W.: HPP - A High Performance PRAM. Volume 2. (1996) 425–434
21. Bach, P.: Entwurf und Realisierung der Prozessorplatine der SB-PRAM. Master's thesis, Universität des Saarlandes, Saarbrücken (1996)
22. Bach, P.: Schnelle Fertigungsfehlersuche am Beispiel der Prozessorplatine CPU-LIGHT. Dissertation (paul), Universität des Saarlandes, Saarbrücken (2000)
23. Janocha, S.: Design der PCIPro-Karte. Master's thesis, Universität des Saarlandes, Saarbrücken (2000)
24. Lichtenau, C.: Entwurf und Realisierung des Speicherboards der SB-PRAM. Diplomarbeit an der universität des saarlandes fb 14 (paul), Universität des Saarlandes, Saarbrücken (1996)
25. Lichtenau, C.: Entwurf und Realisierung des Aufbaus und der Testumgebung der SB-PRAM. Dissertation, Universität des Saarlandes, Saarbrücken (2000)
26. Fischer, J.: Entwurf und Realisierung der Netzwerkplatinen der SB-PRAM. Diplomarbeit an der Universität des Saarlandes FB 14 (paul), Universität des Saarlandes, Saarbrücken (1998)
27. Keller, J.: Zur Realisierbarkeit des PRAM Modelles. PhD thesis, Universität des Saarlandes, Saarbrücken (1992)
28. 1990, I.S.: Standard test access port and boundary scan architecture. Institute of Electrical and Electronics Engineers (1993)
29. Bosch, M.: Fehlermodelle und Tests für das Netzwerk der SB-PRAM. Dissertation, Universität des Saarlandes, Saarbrücken (2000)
30. Kessler, C., Seidl, H.: Fork95 Language and Compiler for the SB-PRAM. In: Proceedings of the 5th International Workshop on Compilers for Parallel Computers. (1995) 408–420
31. Röhrig, J.: Implementierung der P4-Laufzeitbibliothek auf der SB-PRAM. Master's thesis, Universität des Saarlandes, Saarbrücken (1996)
32. Wilson, J.M.: Operating System Data Structures for Shared-Memory MIMD Machines with Fetch-and-Add. PhD thesis, Courant Institute, New York University (1998)
33. Formella, A., Grün, T., Keller, J., Paul, W., Rauber, T., Rüniger, G.: Scientific applications on the sb-pram. In: Proceedings of International Conference on Multi-Scale Phenomena and their Simulation in Parallel, World Scientific (1997)
34. Lenoski, D., Laudon, J., Joe, T., Nakahira, D., Stevens, L., Gupta, A., Hennessy, J.: The DASH prototype: Logic overhead and performance. IEEE Transactions on Parallel and Distributed Systems 4 (1993) 41–61
35. Agarwal, A., Bianchini, R., Chaiken, D., Johnson, K., Kranz, D., Kubiawicz, J., Lim, B.H., Mackenzie, K., Yeung, D.: The MIT Alewife Machine: Architecture and Performance. In: International Symposium on Computer Architecture 1995. (1995)
36. Dementiev, R., Klein, M., Paul, W.J.: Performance of MP3D on the SB-PRAM prototype. In: Proc. of the Europar'02. (2002)