UNIVERSITY OF JOENSUU
DEPARTMENT OF COMPUTER SCIENCE
Report Series A

Memory Module Structures for Shared Memory Simulation

Martti Forsell and Ville Leppänen

Report A-2000-5

ACM B.3.2
UDK 681.3. 02
ISSN 0789-7316
ISBN 951-708-950-3

Memory Module Structures for Shared Memory Simulation

Martti Forsell

Department of Computer Science, University of Joensuu, PB 111, FI-80101 Joensuu, Finland, Phone +358 - 13 - 251 3110, Fax +358 - 13 - 251 3290
Email: Martti.Forsell@cs.joensuu.fi

Ville Leppänen

Department of Computer Science, University of Turku, Lemminkäisenkatu 14-18, Datacity, FI-20520 Turku, Finland, Phone +358 - 2 - 333 8676, Fax +358 - 2 - 333 8600
Email: Ville.Leppanen@cs.utu.fi

November 13, 2000

Abstract

The shared memory programming model on top of a physically distributed memory machine (SDMM) is a promising candidate for easy-to-program general purpose parallel computation. There are, however, certain open technical problems which should be sufficiently solved before SDMM can meet the expectations. Among them is low-level structure of memory system, because most academic studies of the subject assume unrealisticly ideal memory properties, ignoring completely, e.g., the speed difference between processors and memories.

In this paper we propose three memory module structures based on low-level interleaving and caching for solving this speed difference problem. We evaluate these structures along with three reference solutions by determining the overall cost factor of memory references in respect to ideal SDMM using real parallel programs. According to the evaluation the cost less than two is achieved with an interleaved solution, if proper amount of parallelism is available. Moreover, caching increases the speed of the memory, if caches are placed in modules rather than next to processors, but it provides lower throughput than interleaving.

Keywords: Parallel computing, shared memory simulation, memories, interleaving, caches

1. Introduction

The shared memory programming model on top of a physically distributed memory machine (SDMM) is a promising candidate for easy-to-program general purpose parallel computation [Schwartz80, Gottlieb83, Gajski83, Hillis85, Pfister85, Ranade87, Abolhas91, Leppänen94, Kautonen96, Forsell97b].

There are, however, certain open technical problems which should be sufficiently solved before SDMM can meet the expectations. Among them is low-level structure of memory system.

In most investigations [Leppänen94, Leppänen95, Forsell96, Kautonen96] it is assumed that memory modules are as fast as or even faster than processors. A close look at the development of semiconductor products, however, suggest a whole contrary situation: The speed gap between processors and main memories is increasing constantly.

Since the mid 80's the performance of processors has doubled in every 12 to 18

months, while the performance of the building blocks of main memories, Dynamic Random Access Memory (DRAM) chips, has only improved by about 5% per year [Hennessy90] (see Table 1).

Year	Size	Cycle
1980	64 kbit	250 ns
1983	256 kbit	220 ns
1986	1 Mbit	190 ns
1989	4 Mbit	165 ns
1992	16 Mbit	140 ns
1995	64 Mbit	120 ns
2000	256Mbit	65 ns

Table 1. The size and speed development of DRAM chips.

This phenomenon can lead to remarkable loss of performance, because processors are now as much as 70 to 100 times faster than DRAMs: The cycle times of DRAM chips are currently 65 to 90 ns [Samsung00a, Samsung00b] and the cycle times of fastest processors are less than 1 ns [Intel00, AMD00]. Future parallel processors may be even faster than current sequential ones due to more regular structure of operational elements simplifying silicon lay-out and extensive use of clock cycle minimization techniques like superpipelining [Forsell97a].

Sequential computer designers have solved this problem traditionally by placing one or more levels of cache memories between the processor and memory [Hennessy90]. In addition to that, memory techniques, like SDRAM [Samsung00a, Samsung00b] and RDRAM [Rambus00, Samsung00c], supporting memory cell matrix row caching, simple banking, interleaving, and burst transfers, are being used to improve the throughput of the memory system.

Caches in SDMM memory systems are, however, not be as useful as in sequential computers due to very difficult cache coherency problems [Hennessy90, Tomasevic93]. Caches also lose their ability to exploit spatial locality due to semi-random hashing of memory locations around the modules of the computer unless cache rows are hashed instead of single words. Similarly, SDRAM and RDRAM techniques relying on sequential bursts, row caching, and regular access patterns are not able to provide sustained low cycles time required in SDMM memory systems heavily applying random or irregular access patterns.

In vector computers this speed difference problem is traditionally solved by using independent interleaved memory banks and even fast SRAM along with a multistage connection network allowing for pipelining of memory references. This solution has, however, its limitations when compared to the SDMM model: Efficient vector computing is limited to execution of vectorizable algorithms only and its deterministic fixed interleaving causes heavy contention of messages with certain access patterns causing dramatic slowdown [Russell78, Hennessy90].

In this paper we propose three memory module structures based on low-level interleaving and caching for solving this speed difference problem in a SDMM. We evaluate these structures along with three reference solutions by determining the overall cost factor of memory references in respect to ideal SDMM using real parallel programs. According to the evaluation the cost less than two is achieved with an asynchronous interleaved solution, if proper amount of parallelism is available. Moreover, caching increases the speed of the memory, if caches are placed in modules rather than next to processors, but it provides lower throughput than interleaving.

In Section 2 we describe briefly the SDMM model. The structure of an ideal memory module is described in Section 3. In Section 4 we describe two non-caching memory module solutions for reference purposes and propose two interleaved solutions. The basic terms of caching are explained as well as a proposal for a memory module solution using caching are given in Section 5. In Section 6 we consider implementation issues of the proposed low-level solutions. The solutions are evaluated and compared to ideal SDMM in Section 7. Finally, in Section 8 we give our conclusions.

2. Machine model

The standard SDMM [Forsell97b] consists of P multithreaded processors connected to M shared memory modules through a high capacity communication network (see Figure 1). Memory modules may have internal structure, i.e., they may contain multiple DRAM banks, cache memories, message queues, and necessary interconnection circuitry or they may consist of just plain DRAM memory. Individual memory locations are accessed by sending memory access messages to modules through the communication network.

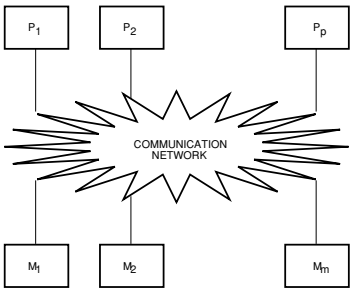

Figure 1. A SDMM, where processors $P_1,...,P_p$ are connected to memory modules $M_1,...,M_m$ through a communication network.

The latency of the network is hidden by using parallel slackness of programs, i.e., letting a processor execute other threads while one is accessing memory. Memory locations are distributed around the modules by a randomly chosen polynomial hashing function. According to investigations [Leppänen95] this kind of hashing can be used to prevent contention of messages, such that underlying DMM can time-processor optimally simulate ideal shared memory with very high probability.

3. Ideal memory module

A popular memory solution of a SDMM in the literature of shared memory simulations is an *ideal memory module* (ID) [Leppänen94, Leppänen95, Forsell96, Kautonen96] (see Figure 2).

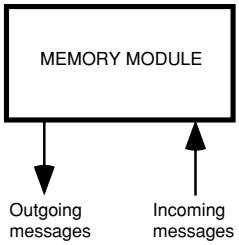

Figure 2. A block diagram of an ideal memory module.

It has properties that make it suitable for simulations: The access time of an ideal memory module is lower than the cycle time of processors and the cycle time of the module equals the cycle time of processors. It is possible to manufacture small ideal memory modules by using fast static RAM instead of conventional dynamic RAM, but they are far too expensive for commercial purposes.

In next two sections, we describe various memory module solutions using realistic latency memory chips. Therefore we need to introduce some symbols, which bind certain properties of the machine into physical world:

Consider a SDMM consisting of P processors each running T threads. Processors are connected to M memory modules through a communication network. In this paper we assume that $M \geq P$ due to technical reasons discussed in Section 1. During one processor cycle each processor executes single instruction. Assume that the cycle time of processors is C_p, the cycle time of memory modules, i.e., the time from one random memory cell access to another, is C_m, and the latency of the communication network is L. Let us denote the time taken by a cache hit with C_c, and the number of memory banks in a module with B. Assume that the length of access queues connected to banks is Q, the average time between memory references in processor cycles is M_r, the size of a memory module is S_m, the size of a cache memory is S_c, and the probability of a cache hit is P_c. In addition, let us denote C_p/C_m with C and S_m/S_c with S.

4. Non-caching memory modules

4.1 Trivial solution

A *trivial memory module* (TR) solution is to replace ideal memory modules just with modules using realistic latency chips and a Q-slot queue to hold messages while memory is accessed by another message (see Figure 3).

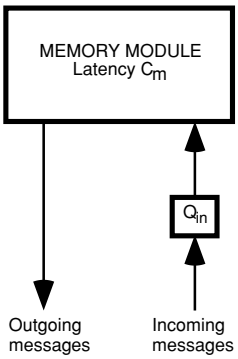

Figure 3. A block diagram of a trivial memory module.

4.2 Replication of memory modules

Replication of memory modules (RE) is a technique to eliminate speed gap between processor and memory: Assuming P processors of the machine are C times faster than P memory modules, one can use ACP memory modules to compensate the speed difference, where A is a constant depending on M_r, L, and required cost factor.

The structure of a module remains the same as in the TR solution (see Figure 3). Only the number of modules is increased by a factor of AC.

Note that this solution may require expensive changes to communication network, because the number of memory modules, communication lines, and routers increases substantially.

4.3 Banked module

Our proposal for solving the speed difference problem is to retain the high level structure of the memory system and change only the internal structure of memory modules by applying banking and interleaving to it. As a result we get yet another memory module solution called *banked module*, which can be divided into two basic variations—asynchronous and interleaved.

In a *banked asynchronous memory module* (BA) solution, a module is divided into B banks, which are accompanied with Q-slot queues of incoming messages and a common Q-slot queue for outgoing messages (see Figure 4).

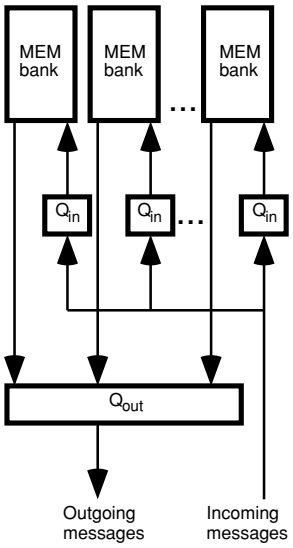

Figure 4. A block diagram of a banked asynchronous memory module.

The queue for outgoing messages can be eliminated by accessing the banks in interleaved manner. This solution is called *banked interleaved memory module* (BI) solution. Alternatively the BI solution can be seen as a set of banks which are connected to single B-phase Q-slot incoming message queue, which feeds memory banks in interleaved way (see Figure 5).

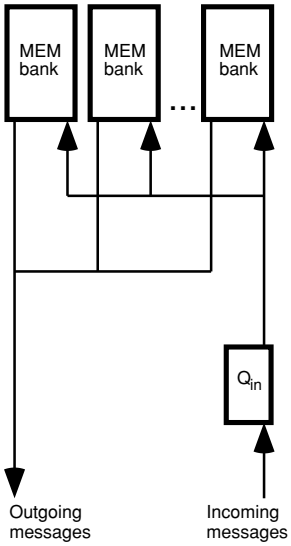

Figure 5. A block diagram of a banked interleaved memory module.

5. Cached memory module

Cache memories are small and fast associative memories, which are used to compensate the speed gap between processors and main memories [Kilburn62, Hennessy90]. The speed gain of caching is based on the behaviour of memory references in average programs: If a memory location *X* is referenced once, it is likely that the same location *X* or a location close to *X* is referenced later during the execution of the program. In the former case we say that the programs have *temporal locality* and in the latter case we say that the programs have *spatial locality*.

Caches can not be used in a SDMM in traditional way placing them next to processors, because this leads to cache coherency problems causing even quadratic costs with respect to the number of processors [Hennessy90, Tomasevic93, Forsell94].

Our proposal to exploit caches in SDMM is to place them into memory modules eliminating the cache coherency problem. This kind of modification of the internal structure of modules can be used also for decreasing the slowdown caused by the speed difference problem. Thus, we call the solution, in which the modules are equipped with caches, *cached memory module* (CA) (see Figure 6).

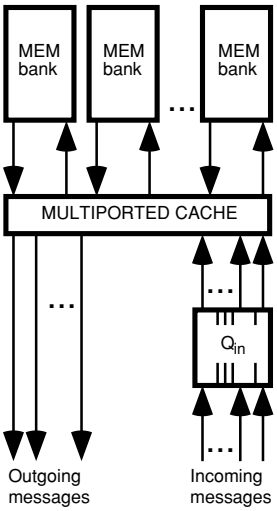

Figure 6. A block diagram of a cached memory module.

Spatial locality can be exploited despite of random hashing by dividing module into banks, attaching a *Q*-slot queue to each word of a cache line, and distributing cache lines instead of single memory words over the modules. In such a case, however, multiport caches and changes to network are required to guarantee unrestricted memory access increasing both complexity and the cost of the memory system.

Alternatively, single port caches may be used, but then it is not possible to reference to the words of a cache line simultaneously. This may not, however, affect performance very much, because currently the memory cycle times are much longer than the processor cycle times.

6. Implementation issues

The silicon implementation of the proposed models require certain special techniques due to very high speed operation like in RDRAMs.

6.1 Spreading data to banks

Spreading data to banks in memory modules may require pipelining due to very high speed operation and the large number of banks driven by single input.

From the analysis point of the view, this kind of pipelining translates into an additional delay in memory cycle time. This delay is logarithmic in respect to the number of banks, because tree-like structures can be used. The base of the logarithm is at most FAN-OUT of applied gate technology.

6.2 The number of banks

For the optimum performance, the number of banks should be at least as high as the speed difference C between processors and memory. It may not, however, be possible to implement a module as single silicon chip, if the size of the module is large. In that case, current multichip module technologies or solutions used in RDRAMs may provide high speed solutions for modules.

6.3 Queues

Queues incorporated in the proposed solutions except the one in BI solution are conventional FIFO queues. The length of queues should be at least the estimated maximum number of messages targeted to single module or bank in single step with high probability.

6.4 Cache memories

In this investigation, fully associative caches are used. Due to high performance requirements, large number of parallel comparators is required to determine if cache hit occurs. The number of comparators can be limited by using n-way set associative caches instead of fully associative caches. The performance advantage of caches will then be partially lost.

7. Evaluation

We evaluated the performance of proposed memory systems by running real parallel programs with various machine settings in parallel machine simulators and memory system simulators.

7.1 Evaluated systems

The six memory system solutions described and proposed in previous sections were used as basic models for evaluation (see Table 2).

ID	- IDeal memory module
CA	- Cached fully Associative memory module
TR	- TRivial memory module
RE	- REplicated memory modules
BA	- Banked Asynchronous module
BI	- Banked Interleaved module

Table 2. The basic models for evaluation.

We assumed that actual memory banks consist of plain DRAM. We did not use SDRAMs or RDRAMs with their cache-like active rows, because we wanted to isolate the effects of caching into the CA model only.

We evaluated the performance of the solutions by measuring the execution time, memory system statistics and cost of shared memory simulation in respect to perfect SDMM (i.e. machine that uses ideal multiport memory and consumes no time for communication) of five Fork95 [Hagerup92, Kessler95] programs and a random message pattern (see Table 3) in a SDMM having 1, 8, 64 and 1024 threads of execution. Due to inefficiency of the machine code produced by the Fork95 compiler, we measured the cost of cycles referencing memory only, i.e., $M_r = 1$. This assumption may, however, not be a big limitation, because our previous investigation revealed that M_r was almost 1 in many highly optimized toy benchmarks designed for evaluation of an efficient multithreaded architecture [Forsell97a].

block	A program that moves a block of words to the other location in memory.
fft	A program that calculates a fast fourier transform of a array of complex numbers.
mandel	A programs that calculates a set of Mandelbrot.
random	Actually not a program, but a set of messages sent to random target modules.
sieve	A program that calculates prime numbers using the sieve of Eratosthenes.
sort	A program that sorts a short table of words.

Table 3. Tested programs.

Communication system latencies were determined by assuming three dimensional mesh (or coated mesh) communication topology and a single cycle router delay. We assumed also fixed latency for all messages. This will not be exactly the case in real life, but it gives a good approximation of communication latencies [Leppänen96]. The ranges of other test parameters are shown in Table 4.

Parameter	Value range
PxT	64x1, 8x8, 1x64, 1024x1, 16x64, 1x1024
C	90
S_c (kB)	16

Table 4. Tested parameters.

The evaluation was made in two parts: First, programs were executed in *SB-PRAM simulator* [Pramsim98] and the memory references of each run were written into trace files. In the second part our memory system simulator *MemSim* [Forsell00] was run with given parameters for each trace file.

Caches were organized as single port, fully associative, four words per line caches using write back and least frequently used (LRU) data replacement policy. Caches were flushed before each run of the benchmark programs. Other factors like periodical refreshing times required for DRAM chips and possible DMA driven memory activity were not taken into account.

Due to large number of tests and long simulation times (even tens of hours in 333 MHz Apple BowerBook G3 computer), some simulation runs were truncated after several hours when it seemed that the cost of simulation had already converged into certain fixed value.

7.2 Results

The results of committed simulations are shown in Table 5.

The costs of real parallel programs are lower than the costs of random access tests. This is caused by uneven distribution of thread-level parallelism in benchmark programs in general and low locality of references generated by the random access test in the case of the CA solution.

To be able to hide the latency of memory references the number of threads in SDMM must be larger than the total latency of single memory reference in cycles. This condition is fulfilled only in the case of the 1 processor x 1024 threads test:

- The ID solution is the fastest one. The cost of simulation is 1 for every benchmark.
- The TR solution performs poorly. As expected the cost is even the cycle time of memory chips divided by the cycle time of processors—the processor/memory speed factor 90.
- The RE solution turns out to be efficient. The cost of simulations is below 1.9.
- The CA solution turns out to be inefficient, although we assume expensive features like full associativity. The maximum cost is 15 excluding the random access test. Including the random access test cost raises to 90.

Parameter	Value range					
PxT	64x1	8x8	1x64	1024x1	16x64	1x1024
C	90	90	90	90	90	90
L	12	6	0	32	8	0
ID Model						
block	19.50	2.19	1.02	49.51	1.20	1.00
fft	39.89	5.62	1.00	35.66	7.76	1.00
mandel	27.00	2.31	1.00	67.00	1.00	1.00
random	29.59	3.26	1.05	71.22	1.52	1.00
sieve	15.33	2.27	1.00	35.15	1.00	1.00
sort	21.07	1.72	1.00	52.22	1.00	1.00
CA Model						
block	45.00	16.10	11.92	76.33	8.35	15.38
fft	21.32	2.57	1.00	55.45	1.00	1.00
mandel	25.07	2.07	1.00	85.00	1.00	1.00
random	407.45	164.33	116.55	653.63	143.20	89.70
sieve	16.86	2.17	25.89	35.01	1.00	10.48
sort	16.95	1.74	1.00	51.23	1.00	1.00
TR Model						
block	103.04	90.88	90.02	123.70	90.15	90.00
fft	104.42	13.08	2.23	124.56	2.07	1.00
mandel	104.00	12.69	1.95	126.00	1.56	1.00
random	370,24	138.64	90.05	532.07	110.65	90.00
sieve	132.77	114.84	84.50	137.85	53.79	39.63
sort	104.00	36.15	14.69	124.00	11.83	4.14
RE Model						
block	147.00	14.75	1.64	*	1.95	1.07
fft	151.91	15.33	2.16	*	2.49	1.00
mandel	174.00	15.31	2.15	*	2.03	1.00
random	193.17	26.77	5.45	*	7.64	1.89
sieve	148.00	16.18	4.52	*	3.14	1.56
sort	150.58	15.02	1.84	*	2.02	1.00
BA Model						
block	104.04	13.38	2.41	124.70	2.58	1.06
fft	104.03	12.26	1.94	124.00	1.57	1.00
mandel	104.00	12.69	1.93	126.00	1.56	1.00
random	185.95	23.05	4.73	258.60	6.54	1.55
sieve	105.48	13.29	2.38	124.72	2.03	1.00
sort	104.16	12.43	1.63	124.05	1.62	1.00
BI Model						
block	120.00	16.88	4.29	*	4.53	3.04
fft	173.76	20.73	2.98	*	3.12	1.00
mandel	170.01	21.25	2.86	*	2.34	1.00
random	244.98	38.02	7.16	*	9.75	2.15
sieve	228.77	28.31	4.14	*	3.96	1.00
sort	204.60	23.50	2.98	*	3.05	1.00

Table 5. The results of simulations. (* not measured due to high memory requirements)

- The BA solution performs almost as well as the ID solution. The cost of simulation is clearly below 1.6.

- Surprisingly the BI solution does not perform as well as the BA solution: The cost maximum cost is 3.0.

If the number of threads is lower than the total latency of single memory reference, the ID solution gives the best performance, the BA solution gives the second best performance, and the CA solution gives the worst performance. If the random access test is not taken into account, the CA solution performs, however, better than other realistic latency models. This is caused by the lack of parallel slackness and the fact that in sequential computing caches can be used to reduce memory reference time if access patterns are regular enough.

8. Conclusions

We have proposed three low-level memory module structures for solving the speed difference problem between processors and memories based on low-level interleaving and caching. We evaluated these structures along with three reference solutions by determining the overall cost factor of memory references in respect to ideal SDMM using real parallel programs.

According to the evaluation the cost less than two is achieved with an asynchronous interleaved solution, if proper amount of parallelism is available. The potentially increased chip count of this solution can be eliminated by integrating banking inside a memory chip. Caching seems to increase the speed of the memory, if caches are placed in modules rather than next to processors, but it does not provide as high performance as interleaving.

An interesting topic of further research would be the possibility to add module level caches to banked asynchronous model and commit this kind of simulations in a more realistic network environment with more complex benchmark programs.

References

[Abolhas91]
F. Abolhassan, J. Keller and W. J. Paul, On the Cost-effectiveness and Realization of the Theoretical PRAM Model, FB 14 Informatik, *SFB-Report 09/1991*, Universität des Saarlandes, Saarbrucken, 1991.

[Alverson90]
R. Alverson, D. Callahan, D. Cummings, B. Kolblenz, A. Porterfield and B. Smith, The Tera Computer System, *Proceedings of the International Conference on Supercomputing*, June 11-15, 1990, Amsterdam, The Netherlands, 1-6.

[AMD00]
1000 MHz Athlon, AMD, http://www.amd.com/, 2000.

[Forsell94]
M. Forsell, Are multiport memories physically feasible?, *Computer Architecture News* **22**, 4 (September 1994), 47-54.

[Forsell96]
M. Forsell, V. Leppänen and M. Penttonen, Efficient Two-Level Mesh based Simulation of PRAMs, *Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks*, June 12-14, 1996, Beijing, China, 29-35.

[Forsell97a]
M. Forsell, MTAC—A Multithreaded VLIW Architecture for PRAM Simulation, *Journal of Universal Computer Science* **3**, 9 (1997), 1037-1055.

[Forsell97b]
M.Forsell, Implementation of Instruction-Level and Thread-Level Parallelism in Computers, *Dissertations 2*, Department of Computer Science, University of Joensuu, Joensuu, 1997.

[Forsell00]
M. Forsell, MemSim—A memory system simulator for SDMMs, *Report B-2000-1*, Department of Computer Science, University of Joensuu, Joensuu, 2000.

[Hagerup92]
T. Hagerup, A. Schmitt, H. Seidl, FORK: A High--Level Language for PRAMs,

Future Generation Computer Systems **8**, 379-393, 1992.

[IBM97]
IBM04364ARLAA - 128K x 36 SRAM, *Preliminary Data Sheet*, Micro Electronics Division, IBM, 1997.

[Intel00]
1000 MHz Pentium III, Intel, http://www.intel.com/, 2000.

[Kautonen96]
A. Kautonen, V. Leppänen and M. Penttonen, Simulation of PRAM on Complete Optical Networks, *Lecture Notes in Computer Science* **1124** (EuroPar '96, France), (August 1996), 307 - 310.

[Kessler95]
C. Kessler and H. Seidl, Fork95 Language and Compiler for the SB-PRAM, *Proceedings of 5th International Workshop on Compilers for Parallel Computers*, Malaga, June 28-30, 1995.

[Leighton91]
T. F. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann, San Mateo, 1992.

[Kilburn62]
T. Kilburn, D. Edwards, M. Kanigan and F. Sumner, One-level storage system, *IRE Transactions on Electronic Computers* EC-11 (April), 1962, 135-148.

[Leppänen94]
V. Leppänen and M. Penttonen, Simulation of PRAM Models on Meshes, *Lecture Notes in Computer Science* **817** (Proccedings of Parallel Architectures and Languages Europe94, Springer Verlag), (1994), 146-158.

[Leppänen95]
V. Leppänen and M. Penttonen, Work-Optimal Simulation of PRAM Models on Meshes, *Nordic Journal of Computing* **2**, 1 (1995), 51-69.

[Leppänen96]
V. Leppänen, Studies on the Realization of PRAM, *TUCS Dissertations 3*, Turku Centre for Computer Science, University of Turku, Turku, 1996.

[McColl92]
W. F. McColl, General Purpose Parallel Computing, *Lectures on Parallel Computation Proceedings 1991 ALCOM Spring School on Parallel Computation (Editors: A. M. Gibbons and P. Spirakis)*, Cambridge University Press, Cambridge, 1992, pp. 333-387.

[McKee98]
S. McKee, R. Klenke, K. Wright, W. Wulf, M. Salinas, J. Aylor and A. Batson, Smarter Memory: Improving Bandwidth fro streamed References, *Computer* , 7 (1998), 54-63.

[Pramsim98]
Pramsim, *simulator program*, Universität des Saarlandes, Saarbrucken, 1998 (available at address ftp://ftp-wjp.cs.uni-sb.de/pub/pram/pramsim/pramsim-current.tar.gz).

[Rambus00]
RDRAM, Rambus, http://www.rambus.com, 2000.

[Ranade87]
A. G. Ranade, S. N. Bhatt, S. L. Johnson, The Fluent Abstract Machine, *Technical Report Series BA87-3*, Thinking Machines Corporation, 1987.

[Russell78]
R. Russell, The Cray-1 computer system, *Communications of the ACM* **21**, 1 (1978), 63-72.

[Samsung00a]
M464S3254BT1 SDRAM SODIMM, *Data Sheet*, Samsung, 2000.

[Samsung00b]
M368L6423BT1 512MB 184pin Unbuffered DDR SDRAM MODULE, *Data Sheet*, Samsung, 2000.

[Samsung00c]
MR18R1624(6/8/C/G)MN0 4/6/8/12/16d RIMM™ Module with 288Mb RDRAMs , *Data Sheet*, Samsung, 2000.

[Tomasevic93]
M. Tomasevic and V. Milutinovic, A Survey of Maintenance of Cache Coherence in Shared Memory Multiprocessors, *Proceedings of the Hawaii International Conference on System Sciences*, Koloa, January 5-8, 1993.