# Experimental Simulations of PRAM on Complete Networks

Martti Penttonen*
Department of Computer Science, University of Joensuu,
B.O.Box 111, FIN-80101 Joensuu, FINLAND

January 3, 1997

**Abstract**

Consider the routing problem of $p$ processors sending packets to each other at random, under the OCPC restriction, where a sending fails if another processor tries to send to the same target. Finding efficient and implementable solutions to this problem would be very desirable for the implementation of PRAM (parallel random access machine). There are two principal means to speed up routing: to improve the algorithm, or to improve the architecture. We consider both. As some of the algorithms are very hard to analyse, we have implemented a routing simulator to get approximate idea of the goodness of the algorithms and architectures. We observe that with proper improvements in the architecture, routing costs close to the optimal 1 step per packet can be achieved.

## 1 Introduction

In the literature of parallel algorithms, the *parallel random access machine*, PRAM for short (see [15], e.g.), has been the dominating model of computation. There are good reasons for this situation [16], first of all the simplicity of the model — which is important as algorithm design itself, sequential or parallel, is difficult anyway. On the other hand, the PRAM model has been criticized for being overly simplistic, neglecting totally the communication that is the bottleneck in current parallel computers [12]. With current technology, distributed memory model looks much more feasible than the synchronous shared memory model (SMM) required by the PRAM. It is therefore natural to ask, if it is possible to execute PRAM programs efficiently on a distributed memory computer. For a thorough overview, see [11].

The simulations of PRAM on distributed memory model are usually based on a randomized hashing of the shared memory to the local memories of the processors [14]. The memory operation then requires routing of data between the processor needing the data, and the processor owning the data. This may look a very expensive operation, as the memory read/write operation between a processor and its memory in a sequential computer, rather problematic question itself, is replaced by routing plus memory read/write. Hence, routing must not be very costly.

It is true that routing causes some extra *latency*, i.e. time lapse $l$ between memory request and data at the target. However, latency is not necessarily wasted time, if the time can be used for something useful. Using parallel *slackness* principle [15], we assume that each physical processor works for $s$ virtual processors. Now, if $l$ is smaller than $s$, or not much greater than $s$, the computation is efficient. The real problem is, whether $s$ packets can be pumped into the routing machinery without increasing $l$ too much at the same time. This depends on the *capacity* of the routing machinery, i.e. how much data in a unit of time it is capable of transporting.

In this work we experimentally investigate the efficiency of routing in a complete network under so called OCPC criterion [2], also called 1-collision criterion. OCPC (Optically Connected Parallel Computer) criterion requires that a packet is successfully received only if no other packet

*E-mail Martti.Penttonen@cs.joensuu.fi

is targeted at the same processor at the same time. As each collision of packets means wasted time, we should avoid collisions. There are two ways to decrease the likelihood of collisions: by hardware or by software. A hardware solution is to increase the capacity of the network by adding more receivers to the processor. By software, the probability of collisions can be decreased by properly regulating the rate of transmissions — if the traffic is thinner, collisions are less likely. In the following, we try both hardware and software techniques to the routing problem.

# 2 Machine models

We assume that the programs are written for the so-called EREW PRAM model, where processors $P_0, \ldots, P_{p-1}$ share a memory $M[0, \ldots, m-1]$, and no concurrent reads from a memory location or writes to a memory location are allowed.

We consider the simulation of the EREW PRAM on a complete network of processors with local memories. We generalize this basic model of distributed memory machine in two ways. First, we may assume that, while all processors are connected by the complete network to each others, they are grouped in modules of $b$ processors that share a memory. Thus, the whole computer consists of small PRAMs, and in addition, there is a complete network connecting all processors. Second, we extend the *complete* network to a *multigraph* having $r$ lines between each pair of processors — we assume that the network is implemented by $r$ optical receivers in each processor. Both of these features are combined in Figure 1.

Assuming that these enhancements can be implemented in architecture, they are useful in simulating the PRAM efficiently.

If there are multiple receivers in processors, there are alternatives, whether all packets passing the OCPC criterion should be considered as received. We assume so, even though that implies some kind of buffering of packets, because several receivers can receive a packet successfully at the same time.

# 3 Routing algorithm

The obvious routing algorithm is just to choose a packet at random at each processor and try to send it. This is called the *greedy* algorithm. By trying to be fast, the greedy algorithm easily leads to a deadlock, i.e. a situation after which some of the packets will never arrive successfully at the target — if two processors each have only one packet targeted to the same processor, they collide until eternity. (Actually *livelock* would be a better name for this situation.) This can be eliminated by decreasing the sending probability of the packets that have collided. This idea is already used in the Ethernet protocol [8], and also in [13] and [10]. Following [10] we call this algorithm the *penalty* algorithm:

**Penalty algorithm** for routing
    **while** processor $P_j$ has packets **do**
        choose packet $\xi_j$ at random from the group of unsent packets
        **if** $1/f$(number of failures of $\xi_j$) $\geq$ RandomNumber$[0 \ldots 1)$ **then**
            attempt to send $\xi_j$

If $f$ is the constant function 1, all nonempty processors always try to send a packet, i.e. the algorithm is greedy. However, in order to avoid deadlock, it may be wise to have some penalty. Indeed, we have

**Lemma 3.1** *If the penalty function $f(x)$ is a monotonically growing function, penalty algorithm is deadlock free.*

*Proof.* When the number $x$ of collisions grows, $f(x)$ grows too and the expectation of packets being tried towards a target decreases below 1. Hence, a successful transmission will eventually be possible. ∎

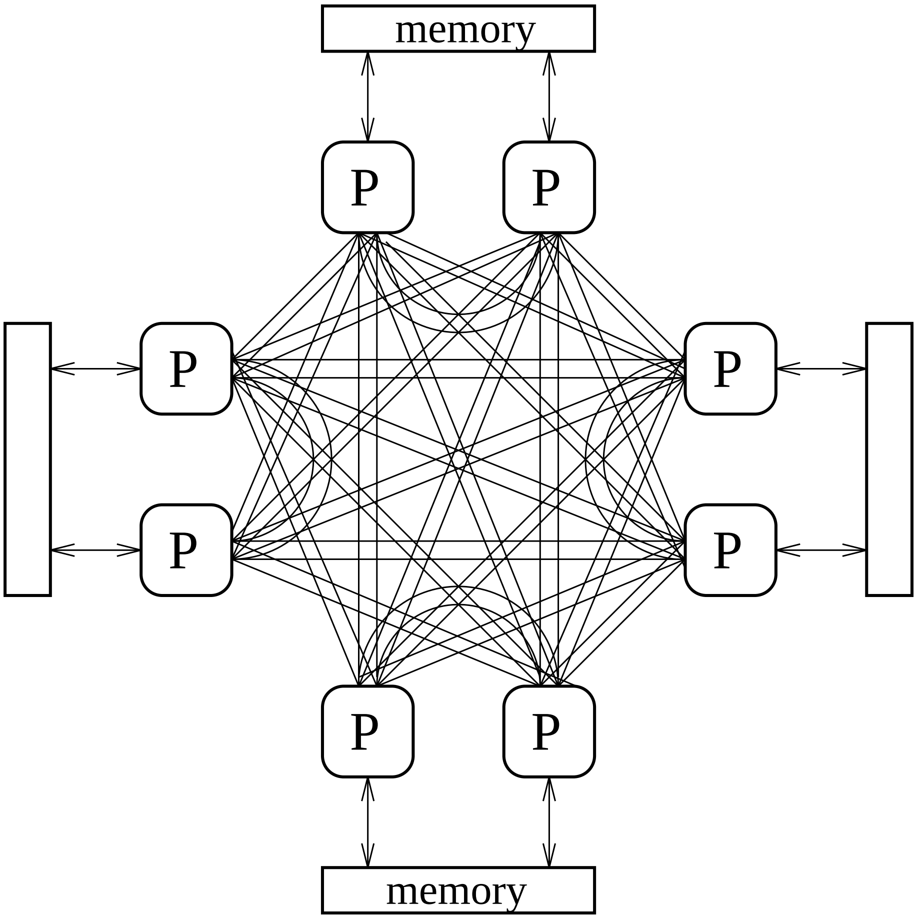


Figure 1: DMM of eight processors in four PRAM modules of size two, with two receivers in each processor.

We do not yet have a theoretical criterion for a good penalty function. Lemma 3.1 gives a lower bound for penalty. Too low penalty leads to too eager sending and collisions, while too high penalty leads to passivity, when no packets are not even tried. In somewhat analogous situation, [7] and [8] suggest linear penalty, while our experiments documented in Section 5 suggest penalties between logarithmic and linear, depending on the configuration.

# 4 Simulator

For the comparison of different routing protocols and different machine models, we have implemented a simulator. In the simulator, the following parameters can be adjusted:

- the number $p$ of processors,
- the slackness $s$, i.e. the number of PRAM processors simulated by a DMM processor
- the routing protocol: the greedy routing without failure penalty, or with logarithmic, square root, linear, or quadratic penalty,
- the number $r$ of receivers (or communication channels) per processor,
- the size $m$ of PRAM modules.

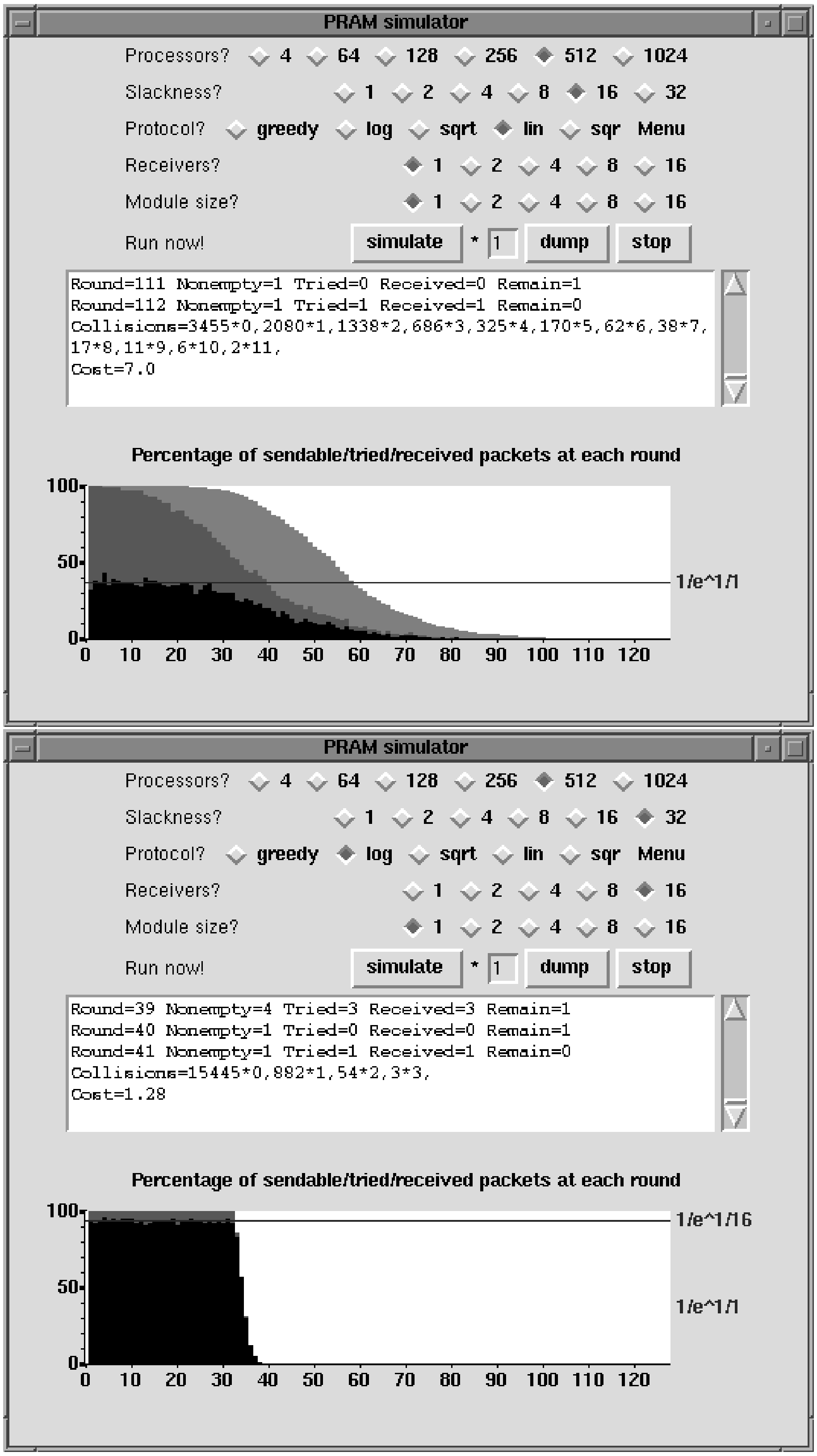


Figure 2: The user interface of the PRAM simulator. Two cases with different number of receivers and penalty.

The simulator was implemented in Prolog (BinProlog 5.25), and the user interface, see Figure 2, in Tcl/Tk (versions 7.5/4.1). It is available by anonymous ftp from `ftp.cs.joensuu.fi` in directory `pub/Software/pram/PRAMonDMM.tar.gz`. The choice of implementation language makes it easy to experiment with different protocols. On the other hand, our easy implementation does not allow for parameters (the number of processors, slackness, etc) much higher than seen in the next section.

The simulator assumes that all instructions are read or write instructions to a random address. At the beginning of a simulation round, $s$ packets with random addresses in the range $0, \ldots, p-1$, are generated. At a simulation step, each processor chooses a packet at random. If the protocol allows, it is sent, i.e. stored in the input buffer of a randomly chosen receiver of the target processor. At the end of the simulation step, and only those packets that are the only in the input buffer are accepted.

The main output of the simulator is the simulation cost, i.e. the proportion of time needed to route $s$ packets to the slackness $s$. This proportion should be as close to 1 as possible. At each simulation step, the simulator shows graphically the numbers of nonempty processors having still packets to be sent, the number of packets that are chosen for sending at this round, and the number of packets that are received successfully. For reference, in the graph there is a line drawn at proportion $1/\sqrt[r]{e}$, where $r$ is the number of receivers. Notice that $1/\sqrt[r]{e}$ is the probability that a ball falls alone in a box, when $p$ balls are thrown at random to $rp$ boxes. At the end of a simulation round, the simulator reports, how many of the packets failed $0, 1, 2, \ldots$ times before a successful sending.

# 5 Experiments

We carried out two sets of experiments to get an idea about the effect of algorithmic and hardware options.

In the first set of experiments we have fixed the number of processors to 512 and the slackness to 16. We used the plain greedy routing protocol, and the greedy protocol with $\log x$, $\sqrt{x}$, $x$, and $x^2$ penalty, where $x$ is the number of failures of the packet being sent. Accordingly, we used sending probabilities as indicated in Table 1.

| $x$ | greedy | $\log x$ | $\sqrt{x}$ | linear | $x^2$ |
|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1/2 | 1/2 | 1/4 |
| 3 | 1 | 1/2 | 1/2 | 1/3 | 1/9 |
| 4 | 1 | 1/2 | 1/2 | 1/4 | 1/16 |
| 5 | 1 | 1/3 | 1/3 | 1/5 | 1/25 |
| 6 | 1 | 1/3 | 1/3 | 1/6 | 1/36 |
| 7 | 1 | 1/3 | 1/3 | 1/7 | 1/49 |
| 8 | 1 | 1/3 | 1/3 | 1/8 | 1/64 |
| 9 | 1 | 1/3 | 1/3 | 1/9 | 1/81 |
| 10 | 1 | 1/3 | 1/4 | 1/10 | 1/100 |

Table 1: Sending probabilities with different penalties.

Actually, we also tried penalties even higher than quadratic, but they gave results that were uniformly much worse than those found here. As architectural enhancements, we let the module size, the number of receivers, or their product to grow to 16. The number of experiments was 10 in each case, unless deadlock occurred (and simulation was interrupted). The number of experiments is rather low, but in general there was very little variation in running times, so there was no motivation for more extensive experimentation. It should be remembered, however, that in case of plain greedy protocol the deadlock is possible even in those cases that succeeded in our experiments. The results of these experiments are shown in Table 2.

| modules | receivers | greedy | log | sqrt | lin | sqr |
|---|---|---|---|---|---|---|
| 1 | 1 | ↑ | 25 | 12 | 6.9 | 12 |
| 2 | 1 | ↑ | 8.5 | 6.1 | 5.7 | 12 |
| 4 | 1 | ↑ | 4.8 | 4.6 | 5.8 | 9.8 |
| 8 | 1 | 14 | 4.1 | 4.7 | 5.7 | 9.3 |
| 16 | 1 | 3.9 | 4.1 | 4.5 | 5.5 | 9.4 |
| 1 | 2 | ↑ | 3.5 | 3.3 | 3.8 | 7.1 |
| 1 | 4 | 1.7 | 2.0 | 2.3 | 2.8 | 3.5 |
| 1 | 8 | 1.5 | 1.6 | 1.8 | 2.3 | 2.5 |
| 1 | 16 | 1.3 | 1.4 | 1.5 | 2.0 | 2.2 |
| 2 | 2 | 3.0 | 2.8 | 3.1 | 3.9 | 6.7 |
| 4 | 2 | 2.2 | 2.5 | 2.9 | 3.7 | 6.1 |
| 2 | 4 | 1.7 | 2.0 | 2.2 | 2.8 | 3.7 |
| 4 | 4 | 1.7 | 1.9 | 2.1 | 2.8 | 4.2 |

Table 2: Simulation cost for different routing protocols, module numbers, and receiver numbers. The number of processors is 512 and the slackness is 16. The symbol ↑ shows cases where deadlocks occurred.

Another set of experiments was carried out to find out a sufficient amount of slackness. In these experiments, the number of processors was 512, and routing with $\log x$ penalty was used. The right amount of slackness depends, of course, on the routing protocol. The log penalty routing was chosen, because it was found fast and safe at the first set of experiments. These experiments are depicted in Figure 3.

## 6 Conclusions

Even though the number and range of experiments was rather low, the results were so uniform that we dare to draw some conclusions. However, they should be read with due cautiousness until there is theoretical support to these results.

- The plain greedy algorithm is the fastest if there is enough routing capacity. However, it is dangerous, because there is the possibility of deadlocks.
- The logarithmic penalty yields slightly higher simulation cost than the plain greedy routing, but is robust.
- If there is no extra capacity, such as multiple receivers or PRAM modules, linear penalty seems to be the best. It is not reasonable to use penalties higher than linear.
- In general, low penalty means eager sending and may cause collisions. High penalty prevents collisions, but may lead to inertia when no packets are even tried to send.
- Increasing the capacity with multiple receivers helps a lot.
- PRAM modules do no speed-up the simulation much, obviously because they do not increase the capacity of the routing machinery.
- A proper (e.g. logarithmic) amount of slackness helps to hide the latency of routing.
- With a little help from the side of technology, i.e. the facility of optical transmitters switchable at the rate of the clock of the processor, and efficient context switching in the processor,

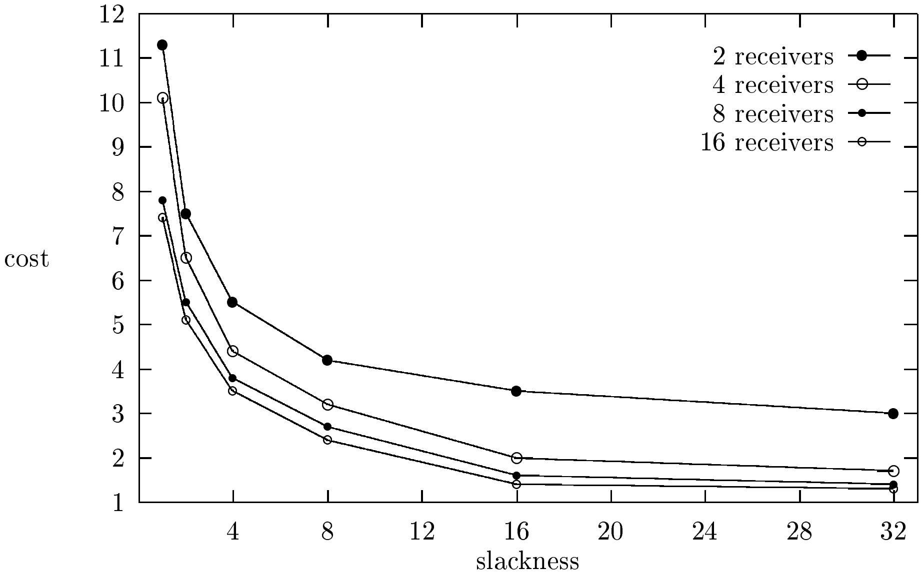


Figure 3: Simulation cost as function of slackness and the number of receivers. The number of processors is 512 and routing has $\log x$ penalty.

efficient simulation of the PRAM looks feasible. A simulation cost of 1.5 or below seems possible.

**Acknowledgement.** We are grateful to Ville Leppänen and Jorma Sajaniemi for many valuable comments on the manuscript, which have helped to improve this report.

# References


[1] M. Adler, J. Byers, and R. Karp. Scheduling Parallel Communication: The $h$-Relation Problem. In *Proceedings of Symposium on Mathematical Foundations of Computer Science, MFCS'95*, LNCS 969:1 – 20, 1995.

[2] R.J. Anderson and G.L. Miller. Optical Communication for Pointer Based Algorithms. Technical Report CRI-88-14, Computer Science Department, University of Southern California, LA, 1988.

[3] M.J. Forsell. Are Multiport Memories Physically Feasible? *Computer Architecture News*, 22(5):3 – 10, December 1994.

[4] M. Geréb-Graus and T. Tsantilas. Efficient Optical Communication in Parallel Computers. In *SPAA'92, 4th Annual Symposium on Parallel Algorithms and Architectures, San Diego, California*, pages 41 – 48, June 1992.

[5] L.A. Goldberg, M. Jerrum, T. Leighton, and S. Rao. A Doubly Logarithmic Communication Algorithm for the Completely Connected Optical Communication Parallel Computer. In *SPAA'93, 5th Annual Symposium on Parallel Algorithms and Architectures, Velen, Germany*, pages 300 – 309, June 1993.

[6] L.A. Goldberg, Y. Matias, and S. Rao. An Optical Simulation of Shared Memory. In *SPAA'94, 6th Annual Symposium on Parallel Algorithms and Architectures, Cape May, New Jersey*, pages 257 – 267, June 1994.

[7] L.A. Goldberg, P.D. MacKenzie. Analysis of practical backoff protocols for contention resolution with multiple servers. Manuscript, 1996.

[8] J. Håstad, T. Leighton, and B. Rogoff. Analysis of backoff protocols for multiple access channels. *SIAM J. Comput.* 25:740–774, 1996

[9] A. Kautonen. Experiments on Simulating PRAM on Complete Network and on Mesh of Buses. Technical Report A-1995-6, University of Joensuu, Department of Computer Science, 1995.

[10] A. Kautonen, V.Leppänen, and M. Penttonen. Simulations of PRAM on complete optical networks. *Proc. Euro-Par'96*, LNCS 1124:307-310.

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

[12] D.A. Patterson. Observations on Massive Parallelism Trends and Predictions for 1995 to 2000. In *DIMACS Workshop on Models, Architectures, and Technologies for Parallel Computation*, pp. 66-77, 1993.

[13] P. Raghavan, and E. Upfal. Stochastic Contention Resolution With Short Delays. In *Proc. STOC 95*, 1995.

[14] A.G. Ranade. How to emulate shared memory. *Journal of Computer and System Sciences*, 42:307–326, 1991.

[15] L.G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33:103-111, 1990.

[16] U. Vishkin. A case for the PRAM as a standard programmer's model. In F. Meyer auf der Heide, B. Monien, and A.L. Rosenberg (eds), *Proc. of Parallel Architectures and Their Efficient Use, First Heinz Nixdorf Symposium*, LNCS 678: 11-19, Springer-Verlag, 1992.