# Simulations of PRAM on Complete Optical Networks

Anssi Kautonen[1], Ville Leppänen[2], and Martti Penttonen[1]

[1] Department of Computer Science, University of Joensuu, B.O.Box 111, FIN-80101 Joensuu, FINLAND
[2] Department of Computer Science, University of Turku, Lemminkäisenkatu 14–18, FIN-20520 Turku, FINLAND

**Abstract.** We study time-processor optimal simulation of EREW PRAM model on a completely connected optical communication parallel computer (OCPC). We propose a new direct algorithm for the realization of $h$-relation, *penalty* algorithm, which is very simple and compares favorably with the Geréb-Graus and Tsantilas algorithm. We study also generalizations of the basic OCPC, where each processor has $\rho$ receivers instead of one and where $\pi$ processors are coupled to a small PRAM module. Our experiments indicate that in this new model the greedy routing algorithm runs without deadlock and fast. Indeed, simulation cost of 2, and even less, is achieved.

## 1 Introduction

Implementation of abstract PRAM models in time-processor optimal manner has been studied extensively in literature [4, 5]. A simulation is *time-processor optimal*, if a step of an $N$-processor PRAM is simulated in time $O(N/P)$ on a $P$-processor distributed memory machine (with high probability).

Anderson and Miller [1] were the first to study the implementation of PRAM (algorithms) and a special case of $h$-relation routing problem on OCPC (although they called it Local Memory PRAM). In *$h$-relation realization problem* each processor is the source and target of (at most) $h$ messages. In *OCPC protocol*, if two or more messages are simultaneously sent to a target, none of them gets delivered. The work of Anderson and Miller implies a realization of $h$-relation on a $P$-processor OCPC in time $O(h + \log P)$.

In [3], Goldberg, Matias, and Rao provide so far the best (asymptotically smallest delay) time-processor optimal simulation of a $P \log\log P$-processor EREW with delay $\log\log P$. Their simulation is rather complicated and therefore it is probably difficult to realize in practice.

A more practical approach is taken by Geréb-Graus and Tsantilas [2] who study direct $h$-relation realization algorithms using only one hash function. An $h$-relation realization algorithm is *direct*, if it sends only original messages and only to the original targets. In [2], Geréb-Graus and Tsantilas present so far the best direct $h$-relation realization algorithm, which works in time $O(h+\log P \log\log P)$.

Their algorithm yields a time-processor optimal EREW simulation for $h = \Omega(\log P \log\log P)$ with high probability.

We present a new direct $h$-relation realization algorithm, penalty algorithm, and experimentally compare its performance with the one presented by Gereb-Graus and Tsantilas. The experiments reported in Section 3 show that the penalty algorithm runs faster than the Gereb-Graus and Tsantilas algorithm, although we do not yet have a theoretical proof for it.

We also study the following *enhanced* OCPC($\pi$,$\rho$): (a) each processor has $\rho$ receivers instead of one, and (b) it is sufficient to deliver the message to a processor in destination area of size $\pi$ processors instead of one fixed target processor. The idea of destination area was used in [3]; here we assume that the destination area is realized as a small PRAM module of $\pi$-processors.

## 2 Routing algorithms

The simplest routing algorithm is the *greedy* routing algorithm that randomly selects one of the remaining packets and attempts to send it with probability 1. In [1], Anderson and Miller conclude that the greedy routing is not good for solving $h$-relation realization on an $OCPC(1,1)$, since certain memory modules can become saturated with requests and yield a deadlock (see simulations on OCPC(1,1) in Section 3).

**Greedy algorithm** for realizing $h$-relations
    **while** processor $P_j$ has packets **do**
        choose packet $\xi_j$ at random among unsent packets
        attempt to send $\xi_j$

As an alternative for the greedy algorithm we propose

**Penalty algorithm** for realizing $h$-relations
    **while** processor $P_j$ has packets **do**
        choose packet $\xi_j$ at random from the group of unsent packets
        **if** $1/(1 +$ number of failures in sending $\xi_j) >$ Random$[0 \ldots 1)$ **then**
            attempt to send $\xi_j$

The penalty algorithm is almost as simple as the greedy algorithm, but it has a mechanism for avoiding deadlock — clearly the probability of deadlock is 0, which is confirmed by experiments in Section 3). It also compares favorably with

**Gereb-Graus–Tsantilas algorithm for $h$-relation**
    **Procedure** $\mathsf{GGT}(\epsilon, h)$
        **for** $i = 0$ **to** $\log_{1/(1-\epsilon)} h$ **do** $\mathsf{Transmit}(\epsilon, (1-\epsilon)^i h)$
    **Procedure** $\mathsf{Transmit}(\epsilon, h)$
        **repeat** for $\frac{e}{1-\epsilon}(\epsilon k + max\{\sqrt{4\epsilon\alpha k \ln P}, 4\alpha \ln P\})$ times
            transmit a randomly chosen packet with probability $\frac{\#unsent\ packets}{k}$

The GGT algorithm has been proved to work in time $O(h + \log P \log\log P)$ with high probability. We don't have yet a similar proof for the penalty algorithm.

## 3 Experimental results

In this section we report some experiments carried out on a PRAM simulator. In simulations, we have done some simplifying assumptions.

- We assume that all instructions are `read` instructions that require two-way information passing.
- Data are stored at random addresses.
- There are 1024 processors, each running 4, 8, 16, 32 or 64 virtual processors (i.e. we assume slackness factor 4 to 64).
- The results are times proportional to the slackness factor, calculated as averages of 50 experiments.
- In the GGT routing algorithm, the parameters $\alpha = 1$ and $\epsilon = 1/2$ were used.
- Results of experiments with plain OCPC, and enhanced OCPC with 16 processor PRAM's and 4 receiver processors are shown.

In the first set of experiments we compared the greedy routing, the penalty routing, and the GGT routing algorithms, when we have a plain OCPC without additional receivers or multiprocessor modules. The results in Figure 1 show that the greedy routing deadlocks, while the penalty and GGT routing give equally good results.

In the second set of experiments, we assume that PRAM modules of 16 processors are available, and data is passed through randomly chosen processor in the target module. Figure 1 shows that greedy routing runs better than more sophisticated routing algorithms.

Finally, in the third set of experiments we test the effect of 4 receivers in processors. In Figure 1 we observe that there is no reason for using other than the greedy routing, and it runs very fast with four receivers per processor.

The success of greedy algorithm and the small influence of $\pi$ and $\rho$ on the GGT and penalty algorithms suggests that they do not send packets eagerly enough at the end of the routing process.

## 4 Conclusions

We have investigated the simulation of PRAM model on optically connected complete network. We observed that a small modification in the greedy algorithm prevents deadlocks and yields a simple and fast routing algorithm. Alternatively, a small enhancement in routing machinery makes the greedy routing deadlock-free and very fast — with a relatively small level of slackness, routing cost 2 per simulated PRAM processor can be achieved. In the experiments, we assumed that all PRAM processors generate a read request. If only half of the PRAM processors make a shared memory reference, the routing cost per simulated PRAM processor may drop closer to 1. We hope to provide analytic proofs in future work.

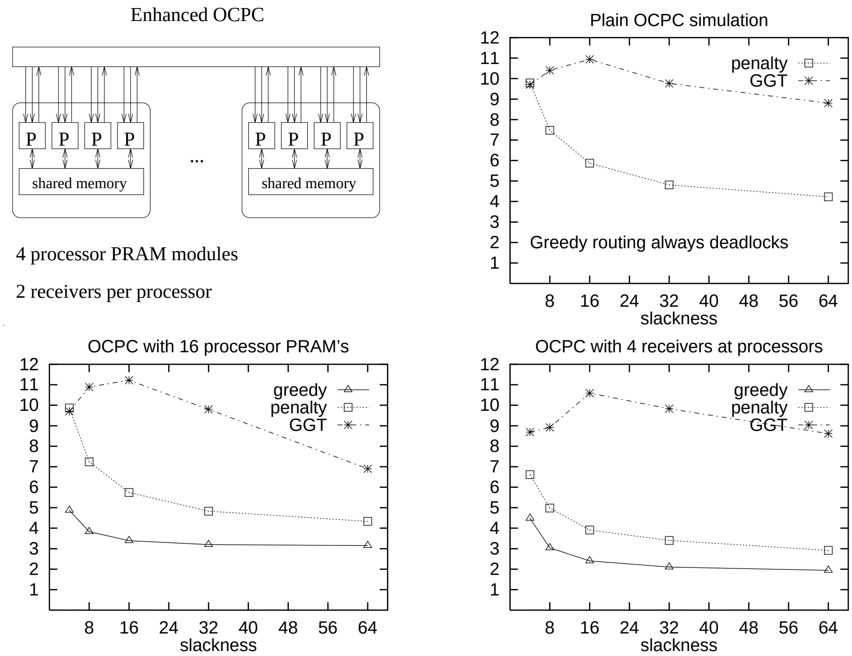


**Figure 1.** Enhanced OCPC. Cost of PRAM simulation without and with PRAM modules and multiple receivers enhancement.

## References

1. 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.
2. 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.
3. 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.
4. F. Meyer auf def Heide. Hashing Strategies for Simulating Shared Memory on Distributed Memory Machines. In F. Meyer auf der Heide, B. Monien, and A.L. Rosenberg, editors, *Proc. of Parallel Architectures and Their Efficient Use, First Heinz Nixdorf Symposium, LNCS 678*, pages 20 – 29. Springer-Verlag, 1992.
5. L.G. Valiant. General Purpose Parallel Architectures. In *Algorithms and Complexity, Handbook of Theoretical Computer Science*, volume A, pages 943–971, 1990.