9 PRAM Simulations

A PRAM (*parallel random access machine*) consists of n processors $P_1, \ldots, P_n$ and a *shared memory* of p memory cells, numbered from 0 to $p-1$ (see Figure 1).

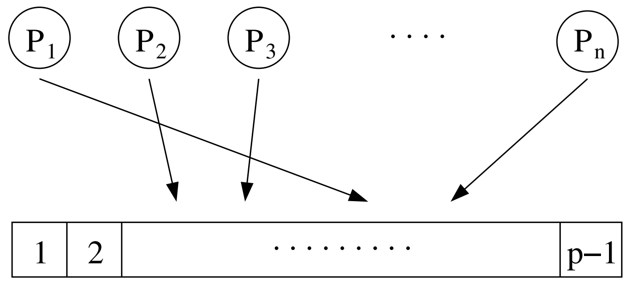

Figure 1: The structure of a PRAM.

Every processor can read or rewrite an arbitrary memory cell in each step. The processors work in a synchronous way. Each step consists of three phases. In the first phase, each processor can do an internal computation, in the seond phase all read requests are done, and in the third phase all write requests are done.

If several processors want to access the same memory cell at the same time step, then the following access rules may be used:

- ER (*exclusive read*): the simultaneous reading of the same cell is forbidden.
- CR (*concurrent read*): the simultaneous reading of the same cell is allowed.
- EW (*exclusive write*): the simultaneous writing to the same cell is forbidden.
- CW (*concurrent write*): the simultaneous writing to the same cell is allowed.

For instance, an EREW PRAM is a PRAM that forbids simultaneous reading and writing to the same cell, and a CRCW PRAM is a PRAM that allows simultaneous reading and writing to the same cell. In the latter case, however, write conflicts may occur. To resolve these conflicts, special rules like the ARBITRARY rule have been used. The ARBITRARY rule says that an arbitrary processor may win when several contend to write to the same cell at the same time.

The PRAM model is a very powerful parallel computational model in which it is easy to design parallel algorithms. However, it is very difficult to actually build a PRAM. Much easier to realize are machines with a *distributed memory*, so-called *distributed memory machines* or DMMs.

A DMM consists of n processors $Q_1, \ldots, Q_n$ and n memory modules $M_1, \ldots, M_n$. Each memory module M_i consists of q memory cells and a communication window. The processors are connected to the modules by a network (see Figure 2).

We again assume that the processors work in a synchronized way. Each time step consists of three phases that work as for the PRAM model above. Each processor can access an arbitrary

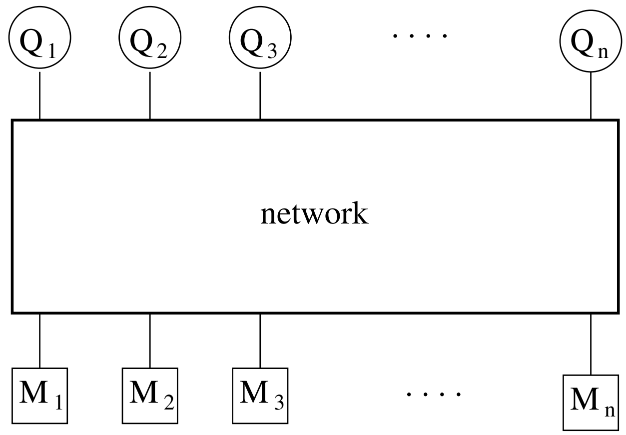

Figure 2: The structure of a DMM.

memory cell of an arbitrary module, but has to do it via the communication window of that module. Hence, requests to a single module are sequentialized.

If the network is a butterfly network or a hypercube network, we say we have a Butterfly DMM or a Hypercube DMM. If the network is complete, we have to define how to handle the situation that many requests arrive at a module at the same time step.

- ARBITRARY DMM: If at least one request arrives at a module, then any one of these is processed and the rest is rejected.
- c-COLLISION DMM: If at most c requests arrive at a module, then all of them are processed, and otherwise none.

Our aim will be to find efficient strategies for the simulation of a PRAM with n processors by a DMM with n processors. If the number of memory cells, p, is much larger than n, this is not an easy problem, because it may happen that many cells that are accessed by the processors in one step are located in the same memory module. Hence, our aim will be to find strategies that ensure that the requests can be distributed among the memory modules as evenly as possible. To simplify this task, we will only look at how to simulate EREW PRAMs in the following.

9.1 A simple simulation strategy for EREW PRAMs

Let $U = \{0, \dots, p-1\}$ be the set of all memory cells. Our trick will be to distribute these memory cells among the modules via a *hash function* $h : U \to \{1, \dots, n\}$ that behaves like a truely random function. That is, for any two (or more) elements $x, y \in U$ with $x \neq y$ we will assume that h assigns modules to x and y chosen independently and uniformly at random. (We will show how this can be achieved in the next section.) In this case, we have the following property.

Theorem 9.1 *Any read or write step of an EREW PRAM can be simulated by an* ARBITRARY *DMM in* $O(\frac{\log n}{\log\log n})$ *steps with high probability.*

Proof. Since we have an EREW PRAM, we know that each read or write step must consist of requests to different memory cells. Let $c_1, \ldots, c_n$ be the memory cells that the processors want to access in a step. Then it holds for any fixed i and k that

$$
\begin{aligned}
&\Pr[\text{at least } k \text{ cells } c_i \text{ are in module } M_i] \\
&\quad = \Pr[\bigcup_{i_1 < \ldots < i_k} (\text{cells } c_{i_1}, \ldots, c_{i_k} \text{ are in module } M_i)] \\
&\quad \leq \sum_{i_1 < \ldots < i_k} \Pr[\text{cells } c_{i_1}, \ldots, c_{i_k} \text{ are in module } M_i] \\
&\quad = \binom{n}{k} \left(\frac{1}{n}\right)^k \leq \frac{n^k}{k!} \cdot \left(\frac{1}{n}\right)^k \\
&\quad = \frac{1}{k!} .
\end{aligned}
$$

Hence, the probability that there is a module M_i with at least k cells c_i is at most $n/k!$. If $k = \Theta(\frac{\log n}{\log \log n})$ is large enough, then this probability is in the order of $1/n^{\Theta(1)}$. □

Hence, our simple random mapping strategy achieves a slowdown of $O(\frac{\log n}{\log \log n})$. Next we present a technique that allows to reduce this slowdown even further.

9.2 An advanced simulation strategy for EREW PRAMs

Suppose now that the modules have enough space to allow us to store every memory cell of the PRAM twice. Then we choose two hash functions h_1 and h_2 to distribute the memory cells among the modules. That is, for each memory cell x, $h_1(x)$ and $h_2(x)$ provide the two modules in which copies of x can be found. However, to have many copies of a memory cell may cause a consistency problem: suppose that in a write request only one copy is updated and in a subsequent read request only one copy is read. Then it can happen that the copy that is read is outdated, which can lead to a failure of the PRAM simulation. To solve this problem, we assume that every time a write request is issued, both copies are updated. Then a read request is allowed to choose any of the two copies without causing inconsistencies. Since read requests are usually much more frequent than write requests, this strategy of ensuring consistency is reasonable.

We show now that the freedom of being able to choose any one of the two copies of a memory cell to satisfy a read request can substantially improve the simulation time.

Theorem 9.2 *Any read step of an EREW PRAM can be simulated by an 4-*Collision *DMM with two copies per cell in $O(\log \log n)$ steps with high probability.*

Proof. Suppose that we allow the 4-Collision DMM $k = d \cdot \log \log n$ rounds to handle all read requests of a PRAM step. Let $c_1, \ldots, c_n$ be the cells the PRAM wants to read in a step. We will show that if d is a sufficiently large constant, then the DMM will be able to handle all of these requests in k rounds with high probability.

Suppose on the contrary that k steps do not suffice to achieve this goal. Then there must be a module M_i at round k that received more than 4 requests, among them say $c_{i_1}, \ldots, c_{i_4}$. Since these requests were still alive in round k, their alternative modules must have received at least 4 other requests in round $k-1$. We continue the argument with these requests to find modules with at least 4 other requests in round $k-2$. This is repeated until we arrive at step 1 of the simulation. The whole process of recording modules with too many requests creates a directed 4-ary tree, called *witness tree of depth k*, in which the internal nodes represent modules and the leaves and edges represent cells c_i. The direction of the edge tells which module stores the first copy (the one at the head of the edge) and which the second copy (the one at the tail of the edge) of the cell (see Figure 3).

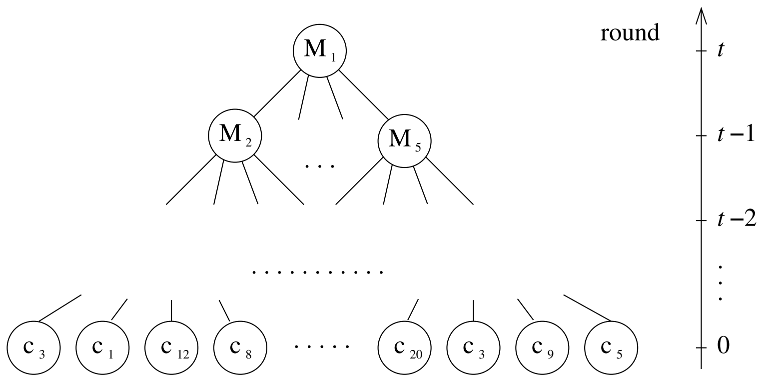

Figure 3: The structure of a witness tree.

So if the simulation fails, then the assignments of the cells c_i must create a witness tree of depth k. Negating this, we obtain that if the cells c_i do not form a witness tree of depth k, then the simulation must be successful. Thus, if we could show that such a witness tree can only exist with a low probability, the theorem would follow.

Thus, it remains to bound the probability that a witness tree of depth k exists. First, we present a bound on the number of ways the cells c_i can be used to form a witness tree.

Lemma 9.3 *There are at most $n^{\sum_{i=0}^{k-1} 4^i} \cdot (\binom{n}{4} \cdot 2^4)^{\sum_{i=1}^{k} 4^i}$ ways for the requested cells to form a witness tree of depth k.*

Proof. Each module has n possible choices, and for each module we have $\binom{n}{4}$ choices to assign cells to the edges below it and 2^4 choices for the direction of these edges. □

Suppose that in a given witness tree all modules are different. Then we distinguish between two cases. If all cell-copy pairs associated with the endpoints of the edges are different, then the probability that the assignment of cells and modules to the witness tree is correct is equal to

$$n^{-\sum_{i=1}^{k-1} 4^i} \cdot n^{-\sum_{i=1}^{k} 4^i} .$$

The first bound represents the probability that the lower endpoints of the edges are correct (in a sense that the corresponding copy of the cell is really stored in the chosen module), and the second bound represents the probability that the upper endpoints of the edges are correct. If the cell-copy pairs associated with the endpoints of the edges are *not* different, then the probabiity that the assignment of cells and modules to the witness tree is correct is zero (either the same copy must be stored in two different modules, which is not possible, or a cell-copy pair is counted twice as a witness for the overload of a module, which cannot appear in our counting). Thus, for the case that in a witness tree all modules are different we arrive at an upper bound for the probability of at most

$$
\begin{aligned}
n^{\sum_{i=0}^{k-1} 4^i} \cdot \left(\binom{n}{4} \cdot 2^4 \right)^{\sum_{i=1}^{k} 4^i} \cdot n^{-\sum_{i=1}^{k-1} 4^i} \cdot n^{-\sum_{i=1}^{k} 4^i} &\leq n \cdot \left(\frac{2^4}{4!} \right)^{\sum_{i=1}^{k} 4^i} \\
&\leq n \cdot \left(\frac{2}{3} \right)^{4^k} \leq n \cdot \left(\frac{1}{2} \right)^{4^k/2}
\end{aligned}
$$

that the mapping of modules and cells to the witness tree is correct. This is polynomially small for $k = O(\log \log n)$.

It remains to look at the situation that the selected modules are not all different. For this situation it can be shown that we still have a polynomially small probability of having a correct mapping of modules and cells to the witness tree. The details can be found in the paper by Meyer auf der Heide, Scheideler, and Stemann. □

The nice side effect of using the c-Collision DMM for the simulation of a PRAM step is that if the distribution of requests among the modules is successful, it guarantees that every module has to process at most c requests in each simulated step. We note, however, that it can happen that the requests cannot be distributed among the modules with the help of the c-Collision rule. (Consider, for instance, the case that all copies of c requests are stored in the same module.) In this case, some emergency rule may have to be used. Fortunately, if $c \geq 4$ as in the theorem above, this will not happen with high probability.

9.3 Simulations of PRAMs by realistic networks

Obviously, any network that is an (n, k)-simulator (concerning the model presented in Assignment 8) can simulate any step of the DMM with slowdown $O(k)$. Hence, it is easy to transfer the results above also to DMMs using realistic networks.

9.4 References

- M. Dietzfelbinger and F. Meyer auf der Heide. How to simulate a dictionary in a complete network. In *Proceedings of the 22nd ACM Symposium on Theory of Computing*, pp. 117–127, 1990.

- R. Karp, M. Luby, and F. Meyer auf der Heide. Efficient PRAM simulation on a distributed memory machine. In *Proceedings of the 24th ACM Symposium on Theory of Computing*, pp. 318–326, 1992.

- F. Meyer auf der Heide, C. Scheideler, and V. Stemann. Exploiting storage redundancy to speed up randomized shared memory simulations. *Theoretical Computer Science* 162, July 1996.