Work-Optimal Simulation of PRAM Models on Meshes

Ville Leppänen*
Department of Computer Science
University of Turku, FINLAND
E-mail: *Ville.Leppanen@cs.utu.fi*

Martti Penttonen*
Department of Computer Science
University of Joensuu, FINLAND
E-mail: *penttonen@cs.joensuu.fi*

Research Report R-94-1, University of Turku
January 3, 1994

Abstract

In this paper we consider work-optimal simulations of PRAM models on coated meshes. Coated meshes consist of a mesh connected routing machinery with processors on the surface of the mesh. We prove that coated meshes with 2-dimensional or 3-dimensional routing machinery can work-optimally simulate EREW, CREW, and CRCW PRAM models. The general idea behind this simulation is to use Valiant's XPRAM approach, and ignore the work-complexity of simple nodes of the routing machinery.

1 Introduction

There are a wide variety of approaches to parallelism in general [40], and even to general purpose parallelism [39] – reflecting the prevailing uncertainty of the correct approach. One model aiming at general purpose parallelism is the PRAM (Parallel Random Access Machine) model, which is a natural generalization of the classical RAM model. It consists of N processors, each of which may have some local memory and registers, and a global shared memory of size m. A step of PRAM is often seen to consist of three phases: writing into the shared memory, performing some local computation, and reading from the shared memory. There are two synchronizing levels in the operation of the PRAM: stepwise and phasewise synchronization. This means that the execution of read and write requests of every processor is tightly synchronized and a very strict consistency of memory is achieved. Therefore, the programmer has more to rely on with the PRAM model than with the other consistency models [42][1]!

The PRAM model has been a very popular platform for the design of parallel algorithms in the literature. In spite of its popularity, there have been only modest interest in actually implementing PRAM. The approach in the long-running Ultracomputer project [23] is PRAM-style, but the machine itself must still be regarded experimental[2]. A remarkable effort of implementation is being taken at the University of Saarbrücken [2, 1, 16]. The underlying interconnection network in that 128-processor machine is the butterfly. Other efforts to implement PRAMs also exist [38].

*This work was financially supported by the Academy of Finland. A preliminary version of this report was published in Proceedings of the Seventh Finnish Symposium on Computer Science in Liperi, Joensuu.

[1]In [42] the term "PRAM" denotes Pipelined RAM. The consistency model of our PRAM is the static atomic consistency, where the operation interval is a phase of a step. We note that [42] concludes to favor weaker consistency models, because the implementation is argued to be faster, but does not attempt to take into consideration the pragmatic effect of consistency models.

[2]A 16-processor prototype Ultra III is described in [7]. The machine is based on Omega-network.

1.1 Work-optimal implementations

The "standard" approach to the implementation is that the shared memory is mapped into a collection of w distributed memory modules, and the N virtual processors are mapped to P real processors of the parallel computer. Each real processor is simulating at most $\lceil N/P \rceil$ virtual processors, and the operations on global memory are translated to messages, which are routed from the sources to the targets via a routing mechanism. The simulation itself is done stepwise in order to make easier the preservation of the atomic consistency.

Often, it is assumed that a distributed memory module is attached to every real processor (i.e., $P = w$), and the processors and the connections between them are seen to form the routing mechanism. In this paper, we also make this assumption. Clearly, the diameter – which is $\Omega(\log P)$ for many popular networks – of the routing machinery, and the requirement of strict consistency restrict the efficiency of implementations.

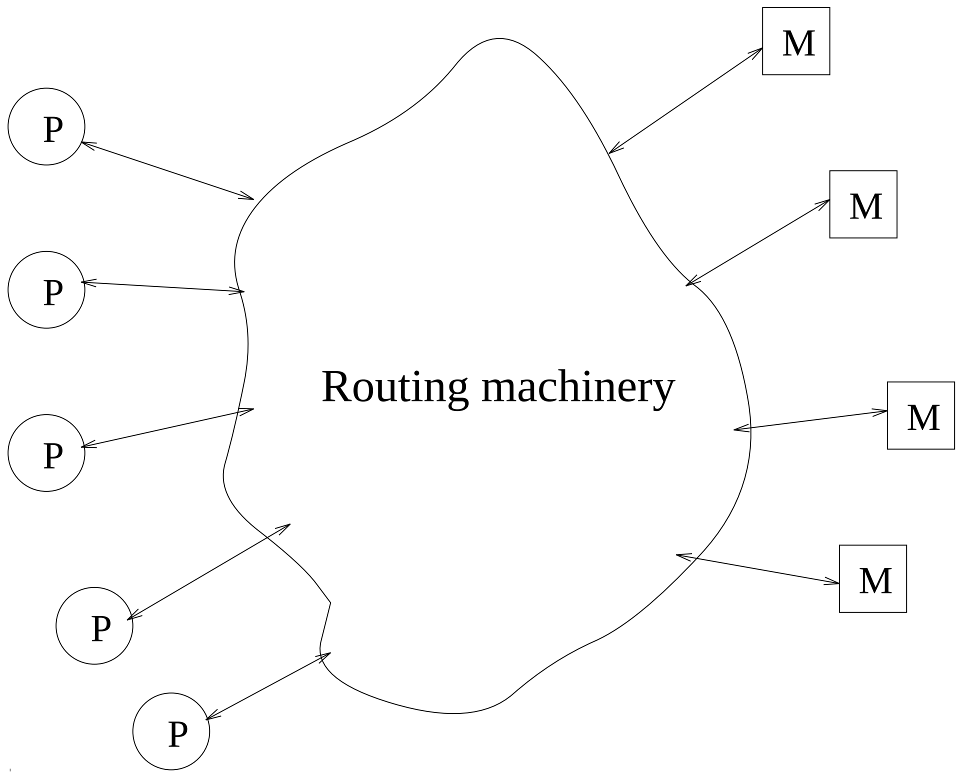

Figure 1: One interpretation of the XPRAM approach.

Valiant [50] proposed a solution to this inefficiency in the form of BSP (Bulk Synchronous Parallel) model. The idea is simply to balance the local computation and the global communication. One way to do the balancing is the XPRAM framework [51], which assumes that $N > P$, and the simulation is done in supersteps of length L, where $L = \Omega(N/P)$. During a superstep, the processors asynchronously do local computations, and communicate with each other in order to simulate the current step of the $\lceil N/P \rceil$ virtual processors that they each simulate. A superstep ends to a barrier synchronization, the purpose of which is to check if all the processors have managed to simulate their virtual processors, or if another superstep is needed. Balancing is done by choosing N, L, and the routing machinery $\mathcal{R}_{N,P}$ so that, all the communication requests of the N virtual processors can be delivered in time $O(N/P)$ (with high probability). The simulation is said to be *work-optimal*, if this is achieved, since then a PRAM program using sufficient amount of virtual processors can constantly achieve a constant fraction (independent of the application and the number of processors) of the peak performance. Remark 1.1 states, when an implementation can be work-optimal.

Remark 1.1 *(See e.g. [27].) Work-optimality can only be achieved on networks, where the expected routing time and amortized (pipelined) hash function evaluation time per processor are constants.*

Next, we classify some of the presented work-optimal PRAM simulation results to (i) those achieved on a completely connected interconnection network, (ii) to those achieved on Mesh Of (Optical) Buses (MOB), and (iii) to those obtained by ignoring the hardware and the work-complexity of the routing machinery components. Some of the simulations results are recently surveyed in [6] (see also [22]).

1.1.1 Ignoring restrictions of physical space

The implementation of the PRAM models on completely connected networks has been studied e.g. in papers [41, 49, 3, 5, 30, 51, 27, 19, 12, 10, 20, 52, 22].

In [51], Valiant showed how a P-processor $\mathcal{RCN}$ (Rejecting Complete Network: the memory module of a processor accepts at most one memory request at a given step; informs about rejection in constant time) can simulate $EREW(P \log P)$ (and weaker) PRAM models work-optimally with high probability. In [20] it is further shown, how a P-processor $\mathcal{RCN}$ can route $\log\log P$-relations work-optimally with high probability. (An *h-relation* means a packet routing situation, where each processor has at most h packets and each processor is the target of at most h packets.) Unfortunately, this does not directly yield a work-optimal $EREW(P \log\log P)$ simulation, since one cannot turn arbitrary memory reference patterns to $O(\log\log P)$-relations (with high probability) by applying only one hash function. A work-optimal simulation of all CRCW PRAMs on a P-processor $\mathcal{SCN}$ (Seclusive Complete Network: only one request is allowed per memory module at any given step) in time $O(P^{\epsilon})$ ($\epsilon > 0$) with high probability was given in [30]. Another work-optimal simulation of PRIORITY PRAM on P-processor $\mathcal{ACN}$[3] (Arbitrary Complete Network: all memory modules accept one, arbitrarily selected, memory request at any given step) was described in [11]. It works in time $O(\log P)$ with high probability. (Note, that $\mathcal{RCN}$ results are valid for $\mathcal{ACN}$ too.)

Using several hash functions and copies of each shared memory cell, it was shown in [12] that a P-processor c-Collision[4] DMM (completely connected, each memory module either rejects all the inputs or accepts them; the inputs are rejected, if there are more than c of them) can simulate a $EREW(P \log P)$ work-optimally with high probability. Similarly, using several hash functions, an improved work-optimal simulation of EREW PRAM on a variant of P-processor $\mathcal{ACN}$ in time $O(\log\log P \log^* P)$ with high probability was constructed in [27]. Recently, Goldberg, Matias, and Rao [22] proved a work-optimal $EREW(P \log\log P)$ simulation for P-processor $\mathcal{RCN}$ using three hash functions simultaneously.

1.1.2 Ignoring the interconnection length

A very tempting model called Mesh Of (Optical) Buses, MOB, was presented in [46]. In the MOB, the processors are arranged in the form of mesh structure, and connected

[3]Valiant [51] denoted with S*PRAM our $\mathcal{RCN}$, and with SPRAM our $\mathcal{SCN}$. In [20] our $\mathcal{RCN}$ is known as OCPC (Completely Connected Optical Communication Parallel Computer) . $\mathcal{SCN}$ is also known as DCM [30] and MPC [41]. In [27] $\mathcal{ACN}$ is a DMM with ARBITRARY write conflict resolution rule.

[4]This model is also motivated by optics. The recognition of several beams of light coming from different directions is obviously not essentially harder than recognizing just one.

to one optical bus per each dimension (each optical bus can be seen as a completely connected graph). What makes optical buses so different from electrical buses is that several beams of light, using different wavelength bands of light, can travel simultaneously in the same space (e.g. within an optical waveguide) without interfering each other (this technique is called WDM – Wavelength Division Multiplexing). Connections between the processors are formed with optical transmitters and receivers. The receivers operate on fixed wavelength bands of light, while the transmitters are tunable (the receivers could also be tunable).

In [46], Rao conjectures that h-relations can be routed work-optimally on a P-processor 2-dimensional MOB (and obviously on 3-dimensional too), if $h = O(\log P)$. This result implies that the MOB can simulate work-optimally $EREW(P \log P)$ PRAM, since if the memory is distributed according to $\mathcal{H}_\zeta$ (defined in Section 3.3; $\zeta = O(\log P)$), then every reference pattern is a $O(\log P)$-relation with high probability [26, 51]. The details of this kind of work-optimal simulation are recently proved in [21].

1.1.3 Ignoring hardware and work complexity of simple nodes

If the diameter of the routing machinery is not constant, then in order to achieve work-optimality the capacity of the routing mechanism must be considerably larger than P. This might mean that the number of routing machinery nodes is asymptotically greater than P. However, in that case it might be acceptable to ignore the hardware complexity of the routing mechanism, since the components can be very simple.

In [51], Valiant showed that when $L = \log N$, a P-input butterfly and a P-node binary hypercube can be used as a routing machinery $R_{P \log P, P}$ to provide a work-optimal simulation of EREW PRAM with high probability. Similar results are established for most of the CRCW PRAMs in two ways: by extending the routing machinery with on route combining mechanism [44], or by requiring that $N = P^{1+\epsilon}$ for some $\epsilon > 0$ [51]. In [30] some of the above results were also given. An implementation of the ideas[5] of [44, 45, 51] is described in [2, 1, 16].

In addition to the above work-optimal simulations, it is easy to construct others simply by taking some routing mechanism $R_{N,P}$, which has P inputs and outputs, and can deliver N messages in time $O(N/P)$ from input to outputs (with high probability).

1.1.4 Some remarks

What are we allowed to ignore? We feel that none of the simulations on completely connected networks is very practical. Although small complete networks might be feasible and optics might enlarge "small" a little bit, we do not consider this to be directly the way to obtain general purpose computing power. The results might, however, be very useful through some other construction, e.g. the MOB.

The assumption of tunable transmitters is essential in the MOB, and the turning speed might be a severely limiting factor. Of course, there are other possible problems, like reliability [43], in the MOB as well as in optics in general. Yet, we would again like to point out that the communication within a bus is assumed to take place in one time unit, which of course is an unrealistic assumption (but may be realistic in practice).

The results based on ignoring the complexity of simple nodes try to make the PRAM approach as a realistic and tempting alternative, but at the same time they ignore (almost completely) the restricting nature of our 3-dimensional world. One simply cannot find a layout with logarithmic physical diameter for a network with graph-theoretically logarithmic diameter. Moreover, the essential assumption in the work-optimal simulation of the

[5] The work-optimal simulation on the butterfly can be derived from Ranade's construction by assuming only one row of the nodes of the network to be processors. This way the routing mechanism becomes $O(\log P)$ times larger than the number of physical processors P.

PRAMs on the hypercubes in [51], that a processor can operate with all of its $O(\log P)$ inputs and outputs in one time unit, can be considered unfair.

1.2 Our way to achieve work-optimality

The laws of the nature (or more precisely, the restrictions implied by contemporary or foreseeable technologies) turn the implementation of the PRAM models as an optimization problem of the layout of the processors and the routing mechanism as well as the routing algorithms. Since it is difficult to take all the issues into account at the same time, and the practicality of the PRAM approach is still to be solved, we will consider a very simple and realistic approach to the routing mechanism by fixing it to be a 2-dimensional or 3-dimensional coated mesh. Notice that this is not the same as a crossbar, since packets traverse from node to node in the routing machinery – not from processor directly to any node within a column (or row) and then again directly to the target processor.

In order to achieve work-optimality, the 2D (or 3D) routing mechanism needs to be $O(P)$ (respectively $O(\sqrt{P})$) times larger than the number of physical processors P. In paper [34] our objective is to find out, what is the "optimal" load for the routing mechanism. In this paper, we only prove upper bounds (constants) for the number of routing steps required per P PRAM processors to simulate PRAM models. Before we continue, it should be noted that the Tera computer [4] uses a 3-dimensional toroidal mesh as a routing mechanism, which is $O(\sqrt{P})$ times larger than P.

We proceed by discussing in Section 2, why we select mesh as the routing mechanism, and illustrate the unrealistic assumptions and difficulties of certain other approaches. In Section 3, we give a closer definition of routing mechanism, and the PRAM models to be simulated. Routing on our mesh based routing machinery is described along with some proofs in Section 4. A description of the simulation of EREW PRAM and other PRAM models is given in Section 5. Finally, in Section 6 we compare the presented solution to the butterfly based solution, and suggest some open problems.

2 Why mesh?

It is clear that a massive general purpose computer should scale up well, and be based on technologically realistic assumptions. Two- and three-dimensional mesh structures clearly have these properties. Those have a natural layout, the number of neighbors is independent of the size of the mesh, the utilization of area and volume are good, and the lengths of interconnections do not grow, when the structure is scaled up. Additionally, mesh structures are modularly extensible, very regular, and even the idea of replacing faulty modules does not seem unrealistic.

The reason, why mesh structures are often seen undesirable, is the diameter $O(d\sqrt[d]{Q})$ of a regular d-dimensional mesh. Completely connected networks are not desirable either, since the degree of a node should be a small constant. The constant degree inevitable implies logarithmic diameter for the network. Many such networks exist and are studied extensively in the literature – e.g. butterfly, multibutterfly, cube-connected-cycles (CCC), hypercube, and shuffle-exchange networks. The problem with these network topologies is that the assumption of constant time communication between neighbors becomes more and more unrealistic as the size scales up [53, 18]. Also the scalability and modular extensibility properties are much more limited than in the case of the mesh structures.

Let us assume (in favor of logarithmic networks) that wires take no volume. Then e.g. the butterfly takes volume $\Omega(\sqrt[3]{Q})$, which implies that some[6] of the wires must have

[6] In [53] it is even proved that the average length of wires in any 3D-layout of a CCC or a binary hypercube must be at least $\Omega(\sqrt[3]{Q}/\log Q)$.

length $\Omega(\sqrt[3]{Q}/\log Q)$, whereas a 3D-mesh of equal size has wires of constant length. Similar argumentation leads to bound $\Omega(\sqrt{Q}/\log Q)$ in the case of 2D-layouts. In fact, the area-time complexity result of Thompson [48] implies that some wires must be of length $\Omega(Q/\log^2 Q)$ in any 2D-layout of the butterfly (similar bound holds for many other logarithmic networks). The reason why this is troublesome is that, if we try to utilize the fact that some of the wires may be short, we are easily lead to difficult load-balancing and timing issues. For example, in the PRAM implementation project [16] it is preferred to operate in the terms of the length of the longest wire in the layout of the butterfly. Actually, the shortening of wires by scattering buffers to long wires at regular intervals was also considered in [16]. In a way this can be seen as a step towards the mesh structure. The benefit of it depends on factors like the capacity of wires, the time to initialize and terminate a transmission, the length of longest wire, and the time to make a routing decision on a node.

We consider the physical length of longest connection significant, since it might restrict the overall clockrate. This is important, since parallelism aims to hook up a huge number of processors, and increasing the clockrate has been (and will perhaps be for awhile) the easiest way to increase the computational power of a machine.

Currently, many of the factors favor butterfly-like routing machinery, especially when the size of the routing machinery is small. However, the relative simplicity and straightforward layout of mesh structure together with the fact that short interconnections allow higher clockrates make mesh-based routing machinery as a tempting alternative. *It should also be noted that the achievable computational power (in terms of PRAM operations per second) does not only depend on the diameter, but also on the throughput of the routing machinery, the available parallel slackness, and the ability of the processors and the routing machinery to tolerate high slackness.*

3 Preliminaries

From now on, N denotes the number of PRAM processors, P the number of real processors, and Q the number of nodes in the routing machinery.

3.1 Coated mesh

Definition 3.2 *A d-dimensional $n_1 \times n_2 \times \ldots \times n_d$* ***mesh*** *is a graph $G_{mesh}^{n_1 \times n_2 \times \ldots \times n_d} = (V, E)$, where*

$$V = \left\{ V_{p_1,p_2,\ldots,p_d} \middle| 0 \leq p_i \leq n_i - 1, 1 \leq i \leq d \right\}$$

is a set of $Q = n_1 n_2 \ldots n_d$ (almost) identical nodes, and

$$E = \left\{ (V_{p_1,p_2,\ldots,p_d}, V_{q_1,q_2,\ldots,q_d}) \middle| \sum_{i=1}^{d} |p_i - q_i| = 1, 0 \leq p_i, q_i \leq n_i - 1, 1 \leq i \leq d \right\}$$

defines the connections between nodes (no wrap-around connections allowed). The degree of mesh is $2d$, and the diameter is $\sum_{i=1}^{d}(n_i - 1)$ (which is $d\sqrt[d]{Q} - d$, if $n_i = n_j$ for all i and j).

We call *coated mesh* the above defined routing machinery $G_{mesh}^{n_1 \times n_2 \times \ldots \times n_d}$, which is coated with processors, and attach the processors to the routing machinery with mapping $\delta : \{\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_{P-1}\} \mapsto V \times \{1, \ldots, d\} \times \{up, down\}$. We further assume that only the nodes on the "surface" can be connected to the processors (see Figure 2). In addition to the interconnection topology of Definition 3.2 we assume that each node $V_{i_1,i_2,\ldots,i_d}$ of the routing machinery

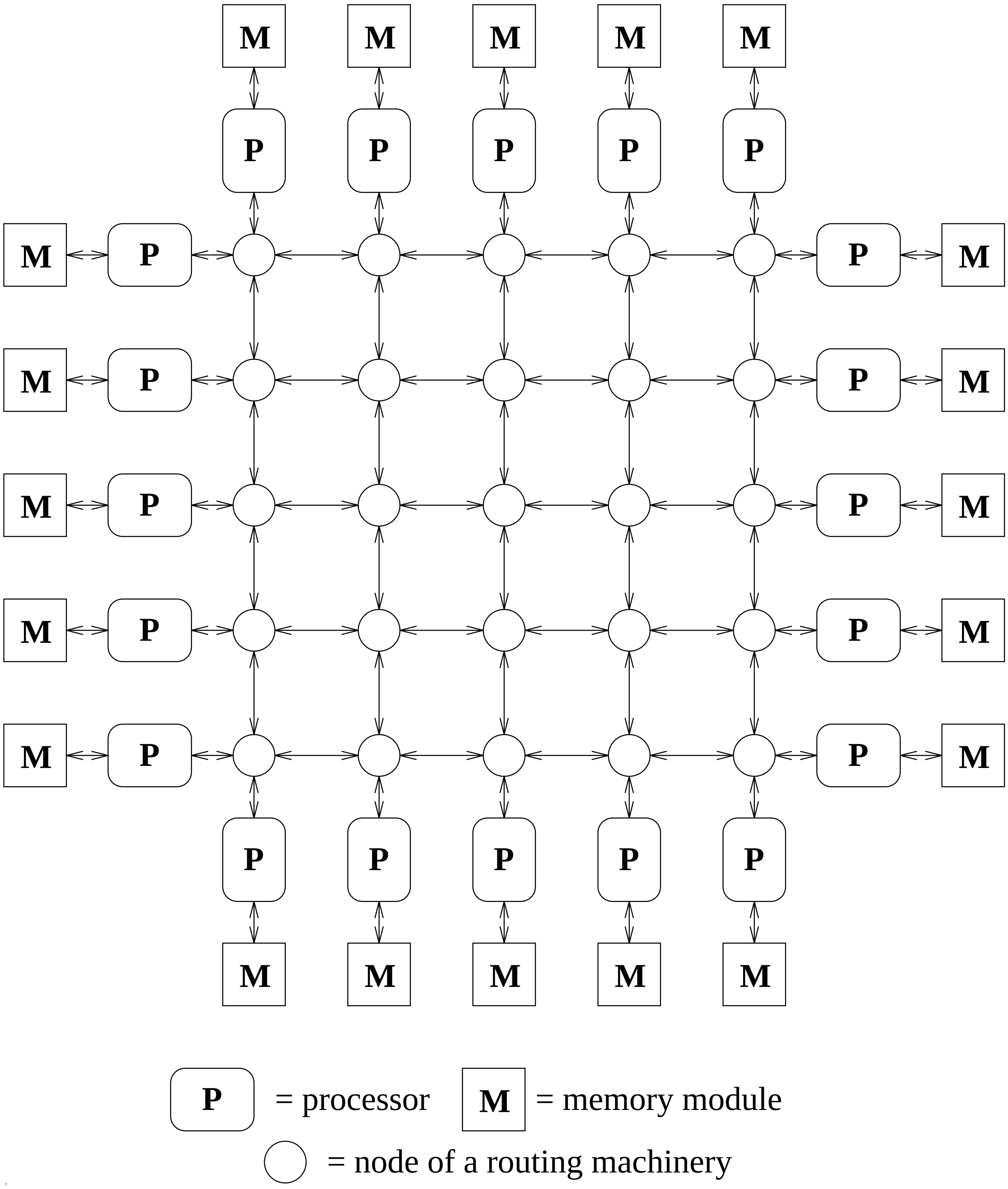

Figure 2: A 2-dimensional coated mesh.

- has d parallel bidirectional links between each neighboring node[7];
- has constant amount of local memory for computation and for storing c_{store} packets;
- is capable of receiving a constant number c_{in} of packets ($O(\log N)$ bits) from each incoming edge, sending a constant number c_{out} of packets along each outgoing edge, and do some local computation (routing decisions) in one time unit;
- has a q-packet queue for each incoming edge;
- is be able to notice, if the queue of a neighboring node is full; and
- can be connected with bidirectional links to at most one processor or routing machinery node per direction.

Notice, that it is not necessary to restrict the last property to only 1 processor altogether, since the links to the processors replace the links to the neighboring nodes, and although a larger number of packets goes through a node connected to many processors, it is possible to design the routing algorithm (actually the routing situation) so that the nodes with 2 or 3 neighboring processors (in the case of 2-dimensional and 3-dimensional routing machinery) will not become a bottleneck.

Remark 3.3 *The routing machinery and the processors might operate at different clockrates. This should be easy to take into consideration in the simulation results, although we will not attempt to do so.*

Remark 3.4 *During one network cycle, the nodes of the routing machinery could transmit more than one packet to each neighbor, if the bandwidth of the communication channels is large (fast connections, several wires, ...). We do not attempt to model this either. If the nodes can transmit x packets to each neighbor, then the network clockrate can be seen to be x times faster than that of the processors. The (possibly) smaller clockrate of processors might not cause any problems, if there are e.g. several wires along which the processors can interact with the routing machinery.*

3.1.1 Distributed memory machines

We assume that each physical processor $\mathcal{P}_i$ is attached to a memory module M_i of size $m' + m_l$. For the mapping of shared memory cells into the memory modules we reserve m' cells. The rest, m_l are reserved for the function of the processor – in particular we assume that a processor can temporarily store $c_{receive}$ messages there. The routing machinery we assume to be either a 2D mesh or a 3D mesh. Further, we assume that the 2D routing machinery is of size $\sqrt{Q} \times \sqrt{Q}$, and there are $P = 4\sqrt{Q}$ processors connected to the outer surface of the routing machinery (any δ_{2D} satisfying this can be used). We denote such a coated mesh with $\mathcal{C}M^{2D}(Q,q)$. In the case of 3D routing machinery, we assume that there are processors on all 6 surfaces. We denote such a $\sqrt[3]{Q} \times \sqrt[3]{Q} \times \sqrt[3]{Q}$ coated mesh ($P = 6Q^{2/3}$) with $\mathcal{C}M^{3D}(Q,q)$. Again we leave δ_{3D} otherwise unspecified.

3.2 PRAM

PRAM model consists of N processors and a shared memory M of size m. Each processor has some local memory and registers, and it knows its own unique identifier PID (Processor IDentifier). PIDs are integers in the range $0, 1, \ldots, N-1$. We denote the processors with $P_0, P_1, \ldots, P_{N-1}$.

The control of the processors is usually thought to work in MIMD fashion, although most PRAM algorithms gain their power from SIMD-style computation. The location

[7] The number of parallel links can be reduced to 1 as we have done in [37, 33, 34], but the proofs come a little bit more complicated, and the routing times increase.

of the program is usually left unspecified, but it can be thought to reside in the shared memory. We leave the I/O-issues unspecified on purpose.

A step of a PRAM processor in PRAM algorithms is often seen to consists of the following three phases:

- do a certain amount of local computation (on the registers),
- read the contents of a shared memory location; and
- write into a shared memory location.

A possible fourth phase might be fetching of next instruction. The execution of phases as well as steps are assumed to be tightly synchronized (i.e., no processor will start next phase/step until all the processors have finished the current phase/step). All the phases are assumed to take unit time (independent of N and m). *In this paper we make the simplification that during one step a processor can participate in at most one of the phases.*

Typically, the PRAM models are characterized to be of type EREW, CREW, ERCW, or CRCW. Other kind of PRAM models have been proposed also in the literature. The letters 'E' (Exclusive) and 'C' (Concurrent) define the access modes of allowed read ('R') and write ('W') instructions on per memory cell per step basis. EREW and CRCW are the most popular – ERCW has received only a little attention in the literature, although in [2] it is considered to be easier to implement than the more popular (and perhaps natural) CREW.

The meaning of concurrent reading is immediately clear, whereas the meaning of concurrent writing is not. Several write-conflict resolution rules have been proposed according to which the CRCW and ERCW models can be divided into submodels. We will explicitly discuss only Arbitrary and Strong CRCW PRAM models, but the results shown in this paper are valid for other CRCW and ERCW models too [33].

Arbitrary ([47]) If two or more processors write into a memory location in a given step, then one of the values is selected arbitrarily to become the new value. Nothing is known about the selection.

Strong ([30]) A special Read-Modify-Write operation replaces the concurrent write (and read). In a given step each processor P_i' can issue a Read-Modify-Write operation $< \oplus_i, arg1_i, arg2_i >$. Either $arg1_i$ or $arg2_i$ is a memory location v_i. The effect of all these operations is as if each processor would, sequentially in the increasing order of PIDs, read the old value of v_i, and update it to $arg1_i \oplus_i arg2_i$ (remember that either $arg1_i$ or $arg2_i$ is actually v_i!).

3.3 Hashing

The objective of hashing the shared memory into the local memories is to guarantee that only a small amount of packets (generated from memory references) is destined to any memory module during a step. Guaranteeing this in the worst-case sense is difficult, since the memory access patterns are generally not known beforehand, and if $M = \Omega(P^2)$ then the memory module of some processor has $\Omega(P)$ shared memory cells in its custody, which all can be referenced during one PRAM step.

A better way to deal with the problem is to guarantee an even distribution of destinations "only" with high probability [8]. The idea is to initially arrange shared memory according to randomly chosen $h \in \mathcal{H}$ (where $\mathcal{H}$ is some proper family of hash functions), and to rearrange the memory by choosing a new hash function $h_{new} \in \mathcal{H}$, if h turns out to be bad for the memory access patterns generated by the program(s).

Choosing a proper hash functions for the emulation of shared memory has been studied extensively in e.g. [41, 26, 14, 11, 28, 17]. In this context, hash function h is often seen as a mapping from $\mathbb{Z}_m \mapsto \mathbb{Z}_m$, which is then used to define mappings $own : \mathbb{Z}_m \mapsto \{0, \ldots, P-1\}$ and $loc : \mathbb{Z}_m \mapsto \mathbb{Z}_{m'}$ in the following way

$$own(x) = (h(x)\ div\ m')\ mod\ P$$

and

$$loc(x) = h(x)\ mod\ m'.$$

It is obvious that h should be bijective (and consequently a permutation). Unfortunately, most of the hash functions considered in the literature are not bijective, and colliding values must be resolved in some way, by secondary hashing for example. Secondary hashing[8] defines an unique location within $own(x)$ for each x.

Definition 3.5 *Let us denote with $\mathcal{H}_\zeta$ and $\mathcal{H}_{mult}$ the families of hash functions*

$$\mathcal{H}_\zeta = \left\{ h \middle| h(x) = (\textstyle\sum_{0 \le i < \zeta} a_i x^i\ mod\ p)\ mod\ m; 0 < a_i < p, m \le p = O(m) \right\},$$

where p is a prime and $\zeta \ge 2$; and

$$\mathcal{H}_{mult} = \left\{ h \middle| h(x) = ax\ mod\ m; 0 < a < 2^k, a \equiv 1\ mod\ 2, m = 2^k \right\}.$$

In [44], family $\mathcal{H}_{O(\log N)}$ was used for the simulation of PRAM models on the butterfly. Although good results are proved for the behavior of family $\mathcal{H}_{O(\log N)}$, the time to evaluate $\zeta = O(\log N)$-degree polynomials is high[9], and hash functions of that family are seldom bijective – i.e., secondary hashing is needed. Hash functions of the family $\mathcal{H}_{mult}$ [13] can easily be proved to be bijective, if $m = 2^k$ for some $k \in \mathbb{N}$, since $gcd(a, 2^k) = 1$.

The reorganizing of memory is an expensive operation, and therefore the triggering criteria of reorganizing must be chosen very carefully. For the reorganizing we apply the methods used on the butterfly [44, 28]. If the processors have an extra shade memory of size $O(m/P)$, then it takes at most

$$T^{2D}_{rehash}(P, m) = O(m + P^2)$$

$$(respectively T^{3D}_{rehash}(N) = O(m + P^{3/2}))$$

routing steps to rehash according to $\mathcal{H}_{mult}$ or $\mathcal{H}_\zeta$ on a $\sqrt{N} \times \sqrt{N}$ coated mesh (respectively on a $\sqrt[3]{N} \times \sqrt[3]{N} \times \sqrt[3]{N}$ coated mesh). Simply "move" the memory cells one by one from each memory module in turn. In fact, it would be possible to do rehashing faster with high probability, but the above times are good enough for our purposes.

Definition 3.6 *Whenever we say "with high probability", we mean "with probability at least $1 - N^{-\beta}$ for some β, where $\beta > 1$".*

Practical experiments [45, 28, 17] seem to indicate that simpler families (e.g. $\mathcal{H}_{mult}$) behave surprisingly well. It is easy to prove based on Chernoff bounds that the expected module congestion – the maximum number of different packets (out of P packets) destined for any processor – is $O(\log P)$ for a randomly selected h and a randomly selected access pattern. However, this does not necessarily guarantee a good behavior, since the access patterns generated by the program(s) need not be random. In our proofs, we will use the family $\mathcal{H}_\zeta$ because of Lemma 3.7 and Lemma 3.8. We will not deal with the secondary hashing problem of $\mathcal{H}_\zeta$ – the methods used in [44, 24] apply in our case, too.

[8] Secondary hashing might not actually require hashing, but just remapping within the memory module. In that case, it might be necessary to send M_k in messages instead of $loc(M_k)$.

[9] However, it is possible to reduce the cost with overloading and pipelining.

Lemma 3.7 *[15] If a randomly chosen $h \in \mathcal{H}_\zeta$ is used for hashing a set S of unique memory locations into s modules, then for $|S| = n$ and for all i $(0 \leq i < s)$:*

$$Pr(b_i \geq u) \leq \begin{cases} c \cdot e^{u-1}/u^u \cdot (n/s)^u, & if\ 1 \leq u < \zeta, \\ c \cdot e^{\zeta-1}/u^\zeta \cdot (n/s)^\zeta, & if\ \zeta \leq u, \end{cases}$$

where $b_i = |\{x \in S \mid h(x) = i\}|$, and $c = (\lceil p/s \rceil \cdot s/p)^\zeta \leq 2^\zeta$.

Lemma 3.8 *[30, Corollary 4.20] If a randomly chosen $h \in \mathcal{H}_\zeta$ is used for hashing a set S of unique memory locations into s modules, for which $|S| = n \geq \zeta s/2$, then for all i $(0 \leq i < s)$:*

$$Pr(b_i > n/s + \epsilon) \leq \frac{\zeta}{2} \left(\frac{e\zeta n}{2s\epsilon^2} \right)^{\frac{\zeta}{2}},$$

where $b_i = |\{x \in S \mid h(x) = i\}|$, and $\ln e = 1$.

Remark 3.9 *Although Lemma 3.7 and Lemma 3.8 deal with remainder classes (the b_i's), their results are valid for any consecutive segment χ of n/s items, since if $h_1(x) = (\sum_{0 \leq j < \zeta} a_j x^j \bmod p) \bmod m$ maps set S to bucket b_i, then there exists such k that $h_2(x) = ((\sum_{0 \leq j < \zeta} a_j x^j) + k \bmod p) \bmod m$ maps χ to bucket b_i.*

4 Routing packets

Our simulation requires that we inject up to N packets into the routing machinery that delivers them. In [36] PRAM influenced routing situations were studied, although there it was assumed that initially and finally packets reside at the nodes of the routing machinery. In that case the basic greedy routing algorithm, routing packets by "correcting" the coordinates of the packets in a certain order of axes, was found good in practice. However, the sources and the destinations are now on the surface of the mesh.

One of the good properties of the basic greedy routing algorithm is that it is deadlock free. If we would apply the basic greedy algorithm directly to packets entering from different directions, then the usage of different parts of the routing machinery would not be uniform (approximately half of the packets would start by crawling along the surface). On the other hand, if we simultaneously apply different order of axes to packets coming from different directions, then the deadlocks become possible. That is why we devote one set of channels (a separate network) for each pair of (opposite) target surfaces. Now, the packets destined to a surface that is perpendicular to the ith axis are easily delivered with the basic greedy routing algorithm R_{cbg}, if the order of axes is fixed to $i+1, \ldots, d, 1, \ldots, i$. Deadlocks are not possible, because routing to different surfaces can be seen as routing in separate networks. We call *Out of Mesh Routing Problem* (OMRP) a problem, where each node of the routing machinery contains at most one packet, and the packets are destined to processors on the surface by $\mathcal{H}_\zeta$ addressing.

4.1 Analysis for 2-dimensional coated meshes

In this section we prove that R_{cbg} solves the OMRP on a 2-dimensional coated mesh so that the packets are delayed at most $O(\log N)$ routing steps with high probability. We use the *wide-channel* proof technique of Leighton [32], and Lemmas 3.7 and 3.8 to evaluate the probabilities. Notice that the 2D-OMRP is symmetric with respect to X- and Y-surfaces, and therefore we study only packets going perpendicular to the Y-axis. For simplicity we assume that there is one packet per node i.e. $N = Q$ – we will use only N in the following. The numbering of processors is actually irrelevant in the 2D-OMRP.

Lemma 4.10 *[32, Lemma 1.10] For all $T, \Delta, x > 0$, if x packets cross an edge ρ during a window of Δ steps $[T+1, \ldots, T+\Delta]$ in the standard routing model, then there is a $t \geq 0$ for which at least $x+t$ packets cross ρ during the interval $[T+1-t, \ldots, T+\Delta]$ in the wide-channel model.*

In the *wide-channel model* an arbitrary number of packets can cross any edge at any step, and thus packets are never delayed. The *standard model* simply restricts this number to 1 per each directed edge. The idea in the wide-channel technique is to use Lemma 4.10 to bound the number of packets crossing an edge (in the wide-channel model), in a certain time interval, below the size of the time interval for each edge with high probability. We show that all intervals of the size $\Omega(\log N)$ have the desired property with high probability.

For the time being, assume that the queues are of unlimited size. As a consequence of that, packets are never delayed while they move along the X-axis or at turn from the X-axis to the Y-axis. Packets are only delayed at the edge queues of the Y-axis by some other packets traversing to the same target. Denote with $\Upsilon(x, n)$ the probability that *at least* x out of n packets are destined (by $h \in \mathcal{H}_\zeta$) to a particular column. The proof of Lemma 4.11 mainly consists of evaluating the probability expression $\Upsilon(x, n)$ for certain x and n.

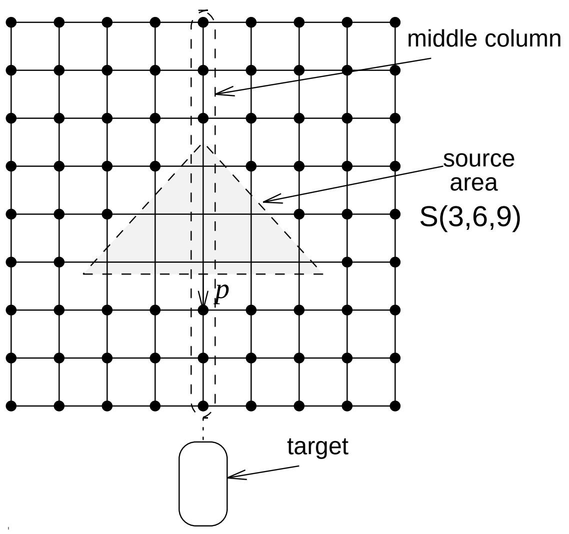

Figure 3: The source and target area related to ρ.

Consider an edge ρ in the middle column of a $\sqrt{N} \times \sqrt{N}$-mesh at height i from the top (see Figure 3). Denote with $S(\Delta', i, \sqrt{N})$ the set of nodes above i and at distance Δ' or less from ρ. We study the middle column, since then as little as possible of the triangular area goes over the boundaries. In the following, we only need the information that $S(\Delta', i, \sqrt{N}) \leq min(\Delta'^2, N)$.

Lemma 4.11 *There exist such constants $\delta, \beta, \gamma > 0$ that the (up to) N unique packets of the 2D-OMRP can be routed on a $\sqrt{N} \times \sqrt{N}$ coated mesh with R_{cbg} and the farthest-first queuing protocol Q_{ff} to their target so that any particular packet is delayed at least Δ steps with probability at most*

$$\begin{cases} N^{-\gamma \log\log N}, & if\ \Delta = \delta \log N\ and\ \zeta = \beta \log N; and \\ N^{-\gamma}, & if\ \Delta = \delta \frac{\log N}{\log\log N}\ and\ \zeta = \beta \frac{\log N}{\log\log N}. \end{cases}$$

Proof. Without loss of generality we deal only with the downward traffic. Consider an edge ρ at height i. Applying Lemma 4.10 (like in [32]), we know that at least Δ packets cross ρ during steps $T+1,\ldots,T+\Delta$ in the standard model only if there is some $t\geq 0$ for which $\Delta+t$ or more packets cross ρ during steps $T+1-t,\ldots,T+\Delta$ in the wide-channel model. Clearly, the source area, where the packets crossing ρ are from, is of size at most $S(t+\Delta,i,\sqrt{N})$.

$$
\begin{aligned}
&Pr(\textit{at least } \Delta \textit{ packets cross } \rho \textit{ during steps } T+1,\ldots,T+\Delta \textit{ in standard model})\\
&\quad\leq \sum_{t\geq 0}\Upsilon(t+\Delta, S(t+\Delta,i,\sqrt{N})) = \sum_{0\leq t\leq N-\Delta}\Upsilon(t+\Delta,S(t+\Delta,i,\sqrt{N}))\\
&\quad= \sum_{\Delta\leq\Delta'\leq\sqrt{N}/\log N}\Upsilon(\Delta',S(\Delta',i,\sqrt{N})) + \sum_{\sqrt{N}/\log N<\Delta'\leq N}\Upsilon(\Delta',S(\Delta',i,\sqrt{N})).
\end{aligned}
$$

The idea from now on is to use Lemmas 3.7 and 3.8 to evaluate $\Upsilon(\ldots)$ for all values of t.

Case (i). In the first sum, we can apply Lemma 3.7 with values $s=4\sqrt{N}$, $n\leq\Delta'^2$, and $u=\Delta'$.

$$
\begin{aligned}
&\sum_{\Delta\leq\Delta'\leq\frac{\sqrt{N}}{\log N}}\Upsilon(\Delta',S(\Delta',i,\sqrt{N}))\\
&\quad\leq \sum_{\Delta\leq\Delta'\leq\frac{\sqrt{N}}{\log N}}\Upsilon(\Delta',\Delta'^2)\leq 2^{\zeta}\sum_{\Delta\leq\Delta'\leq\frac{\sqrt{N}}{\log N}}\left(\frac{e\cdot n}{u\cdot s}\right)^{min(\zeta,u)}\\
&\quad\leq 2^{\zeta}\sum_{\Delta\leq\Delta'\leq\frac{\sqrt{N}}{\log N}}\left(\frac{e\Delta'}{4\sqrt{N}}\right)^{min(\zeta,\Delta')}\leq 2^{\zeta}\sum_{\Delta\leq\Delta'\leq\frac{\sqrt{N}}{\log N}}\left(\frac{e}{4\log N}\right)^{min(\zeta,\Delta')}\\
&\quad\leq\begin{cases} N^{-\gamma'\log\log N}, & \textit{if } \Delta=\delta\log N \textit{ and } \zeta=\beta\log N; \textit{and}\\ N^{-\gamma'}, & \textit{if } \Delta=\delta\frac{\log N}{\log\log N} \textit{ and } \zeta=\beta\frac{\log N}{\log\log N}.\end{cases}
\end{aligned}
$$

for some constants γ', δ, and β.

Case (ii). The other sum is calculated in the same way: now we use Lemma 3.8 instead of Lemma 3.7. Let $s=4\sqrt{N}$, and $\epsilon=\Delta'/2$. Applying of Lemma 3.8 requires that $n/s+\epsilon\leq\Delta'$. Therefore, we can use approximation $n={\Delta'}^2$ (enlarging the source area increases the probability) for $\sqrt{N}/\log N\leq\Delta'\leq\sqrt{N}$, and value $n=N$ for $\Delta'>\sqrt{N}$.

$$
\begin{aligned}
&\sum_{\frac{\sqrt{N}}{\log N}\leq\Delta'\leq N}\Upsilon(\Delta',S(\Delta',i,\sqrt{N}))\\
&\quad\leq\sum_{\frac{\sqrt{N}}{\log N}\leq\Delta'\leq\sqrt{N}}\frac{\zeta}{2}\left(\frac{e\zeta n}{2s\epsilon^2}\right)^{\frac{\zeta}{2}}+\sum_{\sqrt{N}<\Delta'\leq N}\frac{\zeta}{2}\left(\frac{e\zeta n}{2s\epsilon^2}\right)^{\frac{\zeta}{2}}\\
&\quad\leq\sum_{\frac{\sqrt{N}}{\log N}\leq\Delta'\leq\sqrt{N}}\frac{\zeta}{2}\left(\frac{e\zeta{\Delta'}^2}{2\cdot 4\sqrt{N}\cdot(\Delta'/2)^2}\right)^{\frac{\zeta}{2}}+\sum_{\sqrt{N}<\Delta'\leq N}\frac{\zeta}{2}\left(\frac{e\zeta N}{2\cdot 4\sqrt{N}\cdot(\Delta'/2)^2}\right)^{\frac{\zeta}{2}}\\
&\quad\leq\frac{\zeta}{2}\sum_{\frac{\sqrt{N}}{\log N}\leq\Delta'\leq N}\left(\frac{e\zeta}{2\sqrt{N}}\right)^{\frac{\zeta}{2}}\leq\begin{cases} N^{-\gamma''\log\log N}, & \textit{if } \Delta=\delta\log N \textit{ and } \zeta=\beta\log N; \textit{and}\\ N^{-\gamma''}, & \textit{if } \Delta=\delta\frac{\log N}{\log\log N} \textit{ and } \zeta=\beta\frac{\log N}{\log\log N}.\end{cases}
\end{aligned}
$$

for some constants γ'', δ, and β.

■

Clearly, the delay bound of Lemma 4.11 sets an upper bound for the sufficient size of the queues. Now, since the delay some packets can encounter while being routed along edges of the routing machinery is at most $O(\Delta)$ with high probability, there are d pairs of surfaces, only $O(N^2)$ edges, and $O(N)$ time windows to consider, we have Theorem 4.12.

Theorem 4.12 *There exist constants $\delta, \beta, \gamma > 0$ such that a $\mathcal{CM}_{\mathcal{R}}^{2D}(N, O(\Delta))$ solves the 2D-OMRP with R_{cbg} and Q_{ff} in $T_{route}^{2D,coated}(N) = 2\sqrt{N} + O(\Delta)$ routing steps with probability at least*

$$\begin{cases} 1 - N^{-\gamma \log\log N}, & if\ \Delta = \delta \log N\ and\ \zeta = \beta \log N; and \\ 1 - N^{-\gamma}, & if\ \Delta = \delta \frac{\log N}{\log\log N}\ and\ \zeta = \beta \frac{\log N}{\log\log N}. \end{cases}$$

4.2 3-dimensional coated mesh

Based on above, it should be no surprise that we can also prove Theorem 4.13. Grounds for the proof can be found from [33].

Theorem 4.13 *There exist constants $\delta, \beta, \gamma > 0$ such that a $\mathcal{CM}_{\mathcal{R}}^{3D}(N, O(\Delta))$ solves the 3D-OMRP with R_{cbg} and Q_{ff} in $T_{route}^{3D,coated}(N) = 3\sqrt[3]{N} + O(\Delta)$ routing steps with probability at least*

$$\begin{cases} 1 - N^{-\gamma \log\log N}, & if\ \Delta = \delta \log N\ and\ \zeta = \beta \log N; and \\ 1 - N^{-\gamma}, & if\ \Delta = \delta \frac{\log N}{\log\log N}\ and\ \zeta = \beta \frac{\log N}{\log\log N}. \end{cases}$$

4.3 Routing the packets back

In the case of a read instruction, the packet fetching the data must also be routed back to the source. The greedy algorithm R_{cbg} can, of course, be applied in this case too, but the results just proved for the OMRP are not directly valid for this problem. In fact, if we would store certain kind of information concerning the movement of the packets, then we could route the packets back by repeating the movement of packets in completely reverse order. This requires additional hardware, and is not necessary, since we can prove Theorem 4.16 using Lemma 4.14 and Lemma 4.15.

Lemma 4.14 *[32, Lemma 1.5] An N-node linear array whose each node contains an arbitrary number of packets, but for which there are at most one packet destined for each node, can route the packets in $N-1$ routing steps using the basic greedy routing and Q_{ff}.*

Routing the packets back is also symmetric. Therefore, we only study the packets that are on either of the sides that are parallel with X-axis. We call these packets *x-packets*.

Lemma 4.15 *The x-packets determined with randomly selected $h \in \mathcal{H}_\zeta$ can be routed back to their source row on a $\sqrt{N} \times \sqrt{N}$ coated mesh with R_{cbg} and Q_{ff} within time $\sqrt{N} + o(\sqrt{N})$ with probability at least*

$$\begin{cases} 1 - N^{-O(\log\log N)} & if\ \zeta = O(\log N) \\ 1 - N^{-O(1)} & if\ \zeta = O(\log N / \log\log N). \end{cases}$$

Proof. Without loss of generality we can consider only packets traversing downwards. Consider some column η, and divide it into two half. Denote with ψ the processor connected to the lower half of η. Now, we can apply Lemma 3.8 with values $s = 4\sqrt{N}$, $n = N/2$, and $\epsilon = n/s$ and draw the conclusion that more than $\sqrt{N}/2^2$ packets out of $\sqrt{N}/2^1$ rows is destined to ψ with probability at most

$$\begin{cases} N^{-O(\log\log N)} & if\ \zeta = O(\log N); \\ N^{-O(1)} & if\ \zeta = O(\log N / \log\log N). \end{cases}$$

This means that, if replies are send with Q_{ff}, and Q_{ff} is applied within rows and columns separately, then the last of the replies going from ψ to the upper half of the mesh enter there by time $\sqrt{N}/2 + \sqrt{N}/4$ with high probability. The replies going to the closer half of the mesh clearly reach their destination within time $\sqrt{N}/4 + \sqrt{N}/4 + \sqrt{N}/2 = \sqrt{N}$ with high probability.

Now, the proof can be completed by observing that the problem with packets entering to the upper half is the same as the original problem. The above procedure can be repeatedly applied to areas of size $N/2^i$, where $i \le \log \frac{\sqrt{N}}{\zeta} - 1$ (this condition is set by Lemma 3.8). Let j be the last i satisfying the condition. The upmost $\sqrt{N}/2^j = O(\zeta)$ rows receive at most $O(\zeta)$ packets by Lemma 3.8. Clearly, they all can be delivered to their targets in $O(\zeta)$ steps. Thus, we have completed the proof of Lemma 4.15.

■

Theorem 4.16 *There exist such constants $\gamma, \beta > 0$ that a*

$$\begin{cases} \mathcal{C}M^{2D}_{\mathcal{R}}(N, q = O(\log N)) \\ \mathcal{C}M^{2D}_{\mathcal{R}}(N, q = O(\frac{\log N}{\log\log N})) \end{cases}$$

can route the x-packets determined with randomly selected $h \in$

$$\begin{cases} \mathcal{H}_{\beta \log N} \\ \mathcal{H}_{\beta \frac{\log N}{\log\log N}} \end{cases}$$

back to their source node using $R^{O(\log N)}_{cbg}$ (with predetermined intermediate targets) and Q_{ff} in

$$T^{2D,coated}_{route\ back}(N) = 2\sqrt{N} + o(\sqrt{N})$$

routing steps with probability at least

$$\begin{cases} 1 - N^{-\gamma \log\log N} & if\ \zeta = O(\log N) \\ 1 - N^{-\gamma} & if\ \zeta = O(\log N/\log\log N), \end{cases}$$

respectively.

Proof. Let $\zeta = \beta \log N$, and route the packets first along the Y-axis and then along the X-axis. Case, where $\zeta = \beta \frac{\log N}{\log\log N}$ is proved similarly.

Lemma 4.15 implies that each packet reaches its destination row within $\sqrt{N} + o(\sqrt{N})$ steps with probability at least $1 - N^{-\gamma' \log\log N}$ for some constant γ'. The queue size $O(\log N)$ is enough, since according to Lemma 3.7 ($n = \sqrt{N}$ and $m = \sqrt{N}$) at least $u = \zeta$ packets were sent to any given column from any given row with probability at most

$$\begin{aligned} Pr(b_i \ge u) &\le \frac{2e^{\zeta - 2}}{u^{\zeta - 1}} \\ &< \left(\frac{e}{\beta \log N}\right)^{\beta \log N - 1} \\ &< N^{-(\beta - \frac{1}{\log N}) \log\log N} \\ &\le N^{-\gamma'' \log\log N}, \end{aligned}$$

if $\beta > e$ and $\gamma'' \le \beta - \frac{1}{\log N}$.

Thus, if queues are of size at least $\zeta = \Omega(\log N)$, none of the packets traversing along columns is delayed (with high probability) because of receiving queues being full. Now, all we need to observe is that according to Lemma 4.14 all the packets reach their destination in the rows within $\sqrt{N}$ steps after turning to that row.

■

Similar proof techniques can be applied to 3-dimensional coated mesh to prove Theorem 4.17.

Theorem 4.17 *There exist such constants $\gamma, \beta > 0$ that a*

$$\begin{cases} \mathcal{C}M^{3D}_{\mathcal{R}}(N, q = O(\log N)) \\ \mathcal{C}M^{3D}_{\mathcal{R}}(N, q = O(\frac{\log N}{\log\log N})) \end{cases}$$

can route the packets determined with randomly selected $h \in$

$$\begin{cases} \mathcal{H}_{\beta \log N} \\ \mathcal{H}_{\beta \frac{\log N}{\log\log N}} \end{cases}$$

back to their source using R^q_{cbg} (with predetermined intermediate targets) and Q_{ff} in

$$T^{3D,coated}_{route\ back}(N) = 3\sqrt[3]{N} + o(\sqrt[3]{N})$$

routing steps with probability at least

$$\begin{cases} 1 - N^{-\gamma \log\log N} & if\ \zeta = O(\log N) \\ 1 - N^{-\gamma} & if\ \zeta = O(\log N / \log\log N), \end{cases}$$

respectively.

5 Simulation

The technique to simulate PRAM models on distributed memory machines is well-known (see e.g. [44, 51]). Let us first assume that the simulated machine is EREW. We map arbitrarily the N virtual processors of the EREW to the real processors so that each is simulating $\lfloor N/P \rfloor$ or $\lceil N/P \rceil$ virtual ones. Similarly, we map the virtual shared memory of the PRAM into the local memories of the simulating distributed memory machine $\mathcal{M}$ with some hash function h. The simulation itself is done stepwise in order to make easier the preservation of atomic consistency. The simulation of local computation of the PRAM causes no problems. Write and read instructions of the PRAM are simply translated to write and read (and reply) packets, which are destined to the processors on whose custody the referenced shared memory cell is. After the translation, the packets are injected into the routing machinery, which routes them to their target. The arrival of a write packet simply causes a value to be updated, whereas a read packet causes a generation of a reply (which is at some time later injected into the routing machinery). Clearly, the atomic consistency of the shared memory can be guaranteed, if the execution of the next phase is not started before the current is finished, and if an arriving write packet creates a backup copy of the value to be updated (if necessary) for the use of possibly later arriving read packet.

As stated in [54] there can be 3 kinds of congestion in a routing machinery that should be avoided: congestion caused by (a) packets using the same channel (wire), (b) packets referencing to the same memory cell, and (c) packets destined to the same processor/memory module in general. Later we shall see that combining is the method to solve (b), while randomness provided by the hashing of the shared memory is a solution (together with uniform distribution of simulated virtual processors) to (a) and (c).

5.1 Simulating EREW PRAM model

We simulate an N-processor EREW PRAM on a $P = 4\sqrt{Q}$-processor coated mesh $\mathcal{C}M^{2D}_{\mathcal{R}}(Q, q)$ with Algorithm 5.18. In it we assume that processors can receive all the packets destined to them during one simulation round ($c_{receive}$), and all the nodes can store the packets injected to the routing machinery before actually beginning the routing phase (c_{store}). Of course, $c_{receive}$ and c_{store} have some fixed size, and since it is possible that all the packets are destined to some specific node, we can guarantee (only) with high probability that $c_{receive}$ and c_{store} are no exceeded.

The memory references are translated to *packets*, which are then routed via the routing machinery. The read request packet, corresponding to PRAM processor $P_{iN/P+j}$ reading M_k to register R_l, is of the form

$$request\{\Gamma(i,j), i, j, own(M_k), loc(M_k)\},$$

where i defines the source processor; and $\Gamma(i,j)$ defines the node, where the packet is moved in the injection phase before applying routing algorithm R_{cbg}. We assume that simulating processors preserve information concerning pairs (j,l). The generated *request* is routed to the processor $\mathcal{P}_{own(M_k)}$, and when it has fetched the *value* at $loc(M_k)$, it routes the read reply packet

$$reply\{\Gamma(i,j), i, j, value\},$$

back to the processor $\mathcal{P}_i$ simulating $P_{iN/P+j}$ via $\Gamma(i,j)$. Respectively, when the processor $\mathcal{P}_i$ simulating $P_{iN/P+j}$ wants to write R_l to M_k, it sends the write packet

$$write\{\Gamma(i,j), iN/P+j, R_l, own(M_k), loc(M_k)\}$$

to $\mathcal{P}_{own(M_k)}$ via $\Gamma(i,j)$.

Algorithm 5.18 *(EREW simulation on 2-dimensional coated mesh)*
Each processor $\mathcal{P}_i$ simulates PRAM processors $P_{iN/P+j}$, where $i \in \{0,1,\ldots,P-1\}$, and $j \in \{0,1,\ldots,N/P-1\}$. We fix the mapping of real processors[10] δ_{2D}: $\{\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_{P-1}\} \mapsto \{V_{0,0}, \ldots, V_{0,\sqrt{Q}}, \ldots, V_{\sqrt{Q},\sqrt{Q}}, \ldots, V_{\sqrt{Q},0}, \ldots, V_{1,0}\} \times \{X, Y\} \times \{down, up\}$ *to*

$$\delta_{2D}(\mathcal{P}_i) = \begin{cases} (V_{0,i}, Y, down), & if\ 0 \le i < \sqrt{Q}; \\ (V_{\sqrt{Q}, i\ mod\ \sqrt{Q}}, Y, up), & if\ \sqrt{Q} \le i < 2\sqrt{Q}; \\ (V_{i\ mod\ \sqrt{Q}, 0}, X, down), & if\ 2\sqrt{Q} \le i < 3\sqrt{Q}; and \\ (V_{i\ mod\ \sqrt{Q}, \sqrt{Q}}, X, up), & if\ 3\sqrt{Q} \le i < 4\sqrt{Q}. \end{cases}$$

In fact, any mapping will do as long as it can be calculated in constant time.

We use Γ_{2D} to define the nodes (w.l.o.g. assume that $\sqrt{Q} \equiv 0\ mod\ 2$), where we inject packet j of processor $\mathcal{P}_i$:

$$\Gamma_{2D}(i,j) = \begin{cases} V_{i,(\frac{\sqrt{Q}}{2} - 2j_r - 1 - \lambda(i+1+(j\ div\ \frac{\sqrt{Q}}{2})))} & if\ 0 \le i < \sqrt{Q}; \\ V_{i-\sqrt{Q},(\frac{\sqrt{Q}}{2} + 2j_r + \lambda(i+(j\ div\ \frac{\sqrt{Q}}{2})))} & if\ \sqrt{Q} \le i < 2\sqrt{Q}; \\ V_{(\frac{\sqrt{Q}}{2} - 2j_r - 1 - \lambda(i+(j\ div\ \frac{\sqrt{Q}}{2})))i-2\sqrt{Q}}, & if\ 2\sqrt{Q} \le i < 3\sqrt{Q}; and \\ V_{(\frac{\sqrt{Q}}{2} + 2j_r + 1 - \lambda(i+(j\ div\ \frac{\sqrt{Q}}{2})))i-3\sqrt{Q}}, & if\ 3\sqrt{Q} \le i < 4\sqrt{Q}. \end{cases}$$

where $j_r = j\ mod\ \frac{\sqrt{Q}}{2}$ and $\lambda(i) = i\ mod\ 2$. Both

$$\Gamma_{2D}(\{0, \ldots, 2\sqrt{Q}-1\}, \{0, \ldots, \sqrt{Q}/2 - 1\})$$

and

$$\Gamma_{2D}(\{2\sqrt{Q}, \ldots, 4\sqrt{Q}-1\}, \{0, \ldots, \sqrt{Q}/2 - 1\})$$

[10] Recall the definition of δ from page 6. Instead of $1, 2, 3, \ldots$ we use letters X, Y, Z, ...

define a chess board, and together they cover the whole mesh. Notice that

$$\Gamma_{2D}(\{0,\ldots,2\sqrt{Q}-1\},\{k\sqrt{Q}/2,\ldots,(k+1)\sqrt{Q}/2-1\})$$
$$=$$
$$\Gamma_{2D}(\{2\sqrt{Q},\ldots,4\sqrt{Q}-1\},\{(k+1)\sqrt{Q}/2,\ldots,(k+2)\sqrt{Q}/2-1\})$$

for all $k \in I\!N$.

Consider the EREW PRAM processor $P_{iN/P+j}$ *reading* M_k *to* R_l*, writing* R_l *to* M_k*, or executing a local operation. It is simulated by the coated mesh as follows:*

1. *Build the packet corresponding to the instruction:*
   case *instruction type* ***of***
   read: $request\{\Gamma_{2D}(i,j), i, j, own(M_k), loc(M_k)\}$.
   write: $write\{\Gamma_{2D}(i,j), iN/P+j, R_l, own(M_k), loc(M_k)\}$.
   other: do not construct any packet, just simulate the local operation.
2. *Inject the packet (at most* N/P *per processor) to location* $\Gamma_{2D}(i,j)$ *of the routing machinery.*
3. *Route the packet greedily to processor* $\mathcal{P}_{own(M_k)}$ *using* R_{cbg} *and* Q_{ff}.
4. *Execute the memory access by*
   case *instruction type* ***of***
   read: Fetch memory value $value_k$ *from (the backup copy of)* $loc(M_k)$
   write: Make a backup copy of $loc(M_k)$*, and write* R_l *at* $loc(M_k)$.
5. *Synchronize processors, and broadcast to all processors*
   $max\{C^i_{received} | \mathcal{P}_i \text{ has received } C^i_{received} \text{ packets}\}$.
6. *For each read request, build reply packet* $reply\{\Gamma_{2D}(i,j), i, j, value_k\}$.
7. *Inject the s'th reply packet of* $\mathcal{P}_{own(M_k)}$ *to location* $\Gamma_{2D}(own(M_k), s)$ *of the routing machinery.*
8. *Route reply-packet back to* $\mathcal{P}_i$ *via* $\Gamma_{2D}(i,j)$ *using* R_{cbg} *(in reverse order of axes) and* Q_{ff}.
9. *Synchronize processors.*
10. *Use the reply packets to update registers.*

Theorem 5.19 *A coated mesh* $\mathcal{CM}^{2D}_{\mathcal{R}}(N, O(\log N))$ *with* $PCS(G^{\sqrt{N}\times\sqrt{N}}_{mesh}, R^{O(\log N)}_{cbg}, Q_{ff})$ *and* $\mathcal{H}_{O(\log N)}$ *addressing, can simulate an* N*-processor EREW PRAM work-optimally using at most* $36+o(1)$ *routing steps per* P *PRAM processors with high probability.*

Proof. (Notice that $Q = N$.) Algorithm 5.18 clearly simulates one step of an N-processor EREW PRAM, if $c_{receive} = \frac{2N}{P}$ and $c_{store} = 2$ are not exceeded. If during the algorithm some of the limits are exceeded, then some special action is taken to clear the situation – e.g. we might route all the packets one by one in $O(N\sqrt{Q}) = O(N^{3/2})$ steps. As a consequence of Lemma 3.8 a processor receives more than $\frac{2N}{P}$ packets with probability

$$\begin{aligned}
Pr(b_i > n/m + \epsilon) &= Pr(b_i > N/P + N/P) \\
&\le \frac{\zeta}{2}\left(\frac{e\zeta N}{2P(N/P)^2}\right)^{\frac{\zeta}{2}} \\
&\le \frac{\zeta}{2}\left(\frac{e\zeta P}{2N}\right)^{\frac{\zeta}{2}} \\
&\le \frac{\zeta}{2}\left(\frac{2e\zeta}{\sqrt{Q}}\right)^{\frac{\zeta}{2}}
\end{aligned}$$

$$\begin{aligned} &\le \frac{\zeta}{2}\left(\frac{1}{\zeta}\right)^{\zeta} \\ &= Q^{-\gamma \log\log N} \end{aligned}$$

for some constant $\gamma > 0$, if $\zeta = O(\log N)$ and $\sqrt{Q} > 2e\zeta^3$. (If $\zeta = O(\log Q/\log\log Q)$, then the corresponding probability is $Q^{-\gamma}$.) None of the processors receive more than $2N/P$ packets with high probability, and thus when replies are injected back to routing machinery, each node receives at most

$$2 \times \frac{2N/P}{\sqrt{Q}/2} = 2$$

packets with high probability. (Of course we could assume higher limits to obtain better confidence.)

Phases 1, 4, 6 and 10 of Algorithm 5.18 can be done with constant work per PRAM processor (if the amortized evaluation time of ζ-degree polynomials is constant (pipelining!).), and require no routing steps. In phase 2, each processor injects up to N/P packets to routing machinery in

$$T^{2D}_{inject}(N/P) = max(N/P, \sqrt{Q}/2)$$

routing steps. The number of routing steps to accomplish phase 3 ($T^{2D}_{route}(Q)$) and phase 8 ($T^{2D}_{back}(Q)$) were already evaluated in Section 4. Phase 5 basically requires only one prefix sum calculation, and can be accomplished using $T^{2D}_{sync}(Q) = 2\sqrt{Q} + O(1)$ routing steps (see [35]). Naturally, we make the first attempt to accomplish phase 5 after the time, when it is very likely that the routing phase has been accomplished (and therefore the effect of failing in the first attempt will have insignificant influence on the amortized number of routing steps of the simulation). The synchronization in phase 9 is done as in phase 5. Using the argumentation presented in Remark 5.20, we can conclude that the impact of rehashing on the amortized number of routing steps is also insignificant. Thus, the number of routing steps required by simulation Algorithm 5.18 per P PRAM processors is

$$\begin{aligned} &\frac{T^{2D}_{inject}(N/P) + T^{2D}_{route}(Q) + T^{2D}_{sync}(Q) + T^{2D}_{inject}(2N/P) + T^{2D}_{back}(Q) + T^{2D}_{sync}(Q)}{N/P} \\ &= \frac{\frac{1}{2}\sqrt{Q} + 2\sqrt{Q} + o(\sqrt{Q}) + 2\sqrt{Q} + \frac{1}{2}\sqrt{Q} + 2\sqrt{Q} + o(\sqrt{Q}) + 2\sqrt{Q}}{\sqrt{Q}/4} \\ &= 36 + o(1) \end{aligned}$$

with high probability.

■

The actual cost of simulation per P PRAM processors is, of course, larger than 36, because the construction of the packets requires some work. On the other hand, we do not know the relative cost of routing step and an ordinary operation of the processors.

Remark 5.20 *In Algorithm 5.18 and Theorem 5.19 memory reorganization caused by bad hashing and possible failure in the first synchronization attempt was not considered, because its effect on the medium execution time is negligible. Indeed, if N is large enough, $m = N^{\alpha}$ for some constant α, and rehashing is done, when routing is not accomplished in time $2\sqrt{N} + o(\sqrt{N})$, then based on Section 3.3 the effect of rehashing on medium routing time is*

$$T^{2D}_{rehash} \times N^{-\gamma \log\log N} = O(N^{\alpha - \frac{1}{2} - \gamma \log\log N}) = O(1)$$

for some properly chosen γ. On the other hand, if the first synchronization attempt is made at some time $T_1 = 2\sqrt{N} + o(\sqrt{N})$ and the second attempt is made at time $T_2 = 2N$ (all packets have necessarily reached their target by T_2, and thus no synchronization attempt is actually needed), the effect of failing in the first attempt is

$$N^{-\gamma \log\log N} \times (T^{2D}_{sync} + 2N - 2\sqrt{N} - o(\sqrt{N})) = O(1).$$

5.1.1 Some remarks

Above, we assumed that all the packets could be loaded into the routing machinery. If that is not possible (in case of overloading) – i.e., $N > Qc_{store}$, then we divide the routing into several rounds of the above routing algorithm. If $c_{receive}$ is large enough (e.g. about $2N/P$), then it might not be wise to generate replies straight away, since some of the processors have more than $c_{store}\sqrt{Q}/2$ replies to deliver with extremely high probability (and then it would take more than 1 round to reply). A better approach is to first route all the packets to their targets in several rounds, and then route the replies in about the same number of rounds.

Notice that if we would have processors only on one side of the routing machinery, then the routing cost per P PRAM processors would only be about $\frac{1}{4}$'th! It is not, however, the best way to improve performance, since the number of processors is smaller and the overloading factor is higher. What was said in [35] about bus based synchronization, clearly applies to coated meshes too. If $T^{2D}_{sync}(Q) << 2\sqrt{Q}$ (see e.g. [25] for fast optical bus based synchronization), then the number of routing steps required per P PRAM processors would reduce about 20% in Theorem 5.19. It is also good to remember that the clockrate of the routing machinery can be much smaller than that of the processors (see Remarks 3.3 and 3.4). If the clockrate of the routing machinery is x times faster (and the bandwidth of injection and removal operations is sufficient), then the cost of simulation per P PRAM processors can in principle be divided by (approximately) x.

Yet, another method to improve the cost in practice is overloading. We could improve the cost in Theorem 5.19 simply by routing $N > Q$ packets in several rounds (Q packets per round). Furthermore, we believe that even better improvements can be achieved by routing considerably more than Q packets in each routing phase. Unfortunately, we are unable to prove analytical results for this kind of overloaded routing. However, the experiments made in [33, 34] indicate large improvements in performance. We give some estimations of the improvements in Section 6.

5.2 Simulation of CREW and CRCW PRAMs

Of course, the remarks made about EREW simulation on the 2-dimensional coated mesh apply to CRCW simulations too. The following algorithm requires that the nodes[11] of the routing machinery can do many elementary operation in addition to forwarding packets.

Algorithm 5.21 *(CRCW simulation on 2-dimensional coated mesh)*
Consider a PRAM processor $P_{iN/P+j}$ reading M_k to R_l or writing R_l to M_k, or executing some other local operation. Using the same notations as in Algorithm 5.18 we can construct a simulation of one step of the PRAM model with the following phases.

1. *Build a packet corresponding to the instruction:*
   case** instruction type **of
   read: $request\{\Gamma_{2D}(i,j), i, j, own(M_k), loc(M_k)\}$;
   write: $write\{\Gamma_{2D}(i,j), iN/P + j, R_l, own(M_k), loc(M_k)\}$;
   other: *generate no packet, just simulate the local operation.*

[11] Does the simplicity of nodes become questionable?

2. *Inject the packet (at most N/P per processor) to location $\Gamma_{2D}(i,j)$ of the routing machinery.*
3. *Sort the packets lexicographically according to $own(M_k)$ and $loc(M_k)$ into the snake-like indexing.*
4. ***case** packet type **of***
   read: *save request packet temporarily;*
   if** node $V_{i',j'}$ is in the center of a group with common target **then
   build $request_1\{V_{i',j'}, own(M_k), loc(M_k)\}$;
   write: *save write packet temporarily;*
   if** node $V_{i',j'}$ is in the center of a group with common target **then
   build $write_1\{V_{i',j'}, V, own(M_k), loc(M_k)\}$,
   where V is obtained from the values of the group by
   applying appropriate associative reduction operation on the values.
5. *Route $request_1$ and $write_1$ packets to processor $P_{own(M_k)}$ with R_{cbg} and Q_{ff}.*
6. *Execute the memory access by*
   case** packet type **of
   read: *Fetch memory value $value_k$ from (the backup copy of) $loc(M_k)$;*
   write: *Make a backup copy of $loc(M_k)$, and write R_l at $loc(M_k)$.*
7. ***if** packet type read or* STRONG *write **then** build reply packet $reply_1\{value_k, V_{i',j'}\}$.*
8. *Synchronize processors, and broadcast to all processors*
$$max\{C^i_{received} | \mathcal{P}_i \text{ has received } C^i_{received} \text{ packets}\}.$$
9. *Inject the s'th reply packet of $\mathcal{P}_{own(M_k)}$ to location $\Gamma_{2D}(own(M_k), s)$ of the routing machinery.*
10. *Route $reply_1$ packets back to $V_{i',j'}$ with R_{cbg} and Q_{ff}.*
11. *Synchronize nodes.*
12. *Spread $value_k$ to processors having the common target $loc(M_k)$ at $P_{own(M_k)}$.*
13. *Build packet $reply\{V_{i'',\alpha(i'',j'')}, i, j, value'_k\}$ (using $value_k$, an intermediate valued produced by associative reduction operation, and the saved packet), where $\Gamma_{2D}(i,j) = V_{i'',j''}$ and*
$$\alpha(i,j) = \begin{cases} j, & \text{if } i \equiv 0 \text{ mod } 2 \\ \sqrt{N} - j & \text{if } i \equiv 1 \text{ mod } 2. \end{cases}$$
14. *Sort the generated reply-packets according to $V_{i'',\alpha(i'',j'')}$ into the snake-like indexing.*
15. *Route the reply-packets back to processors.*
16. *Use the reply packets to update registers.*

Theorem 5.22 *A coated mesh $\mathcal{CM}^{2D}_{\mathcal{R}}(N, O(\log N))$ with $PCS(G^{\sqrt{N}\times\sqrt{N}}_{mesh}, R^{O(\log N)}_{cbg}, Q_{ff})$ and $\mathcal{H}_{O(\log N)}$ addressing, can simulate an N-processor CREW and all the presented CRCW PRAM models work-optimally using at most $66 + o(1)$ routing steps per P PRAM processors with high probability.*

Proof. Before we can conclude that Algorithm 5.21 can be used to simulate CRCW (and CREW) PRAM models, we need to explain the model dependent phases 4, 6, 7, 12, and 13. How to accomplish read requests in those phases is clear, since the middle point of a group with equal target can be found in $\sqrt{Q}$ steps ([35]). If the write model is ARBITRARY, then any of the packets can be selected as a representative in phase 4.

The write model STRONG is the hardest to implement (that is why we choose it). We recall that in article [30] Kruskal, Rudolph, and Snir show, how the strong primitive can be reduced to prefix computations (multiplications) of 2×2-matrices, and a multiplication of a 2×1-matrix. On the other hand, in [35] it is shown, how prefix computations can be folded into compressing (phase 4) and spreading (phase 12). Updating the memory locations in phase 12 is easy, since as a consequence of compressing we have the both values on which we need to apply the required operation. Building a reply packet for STRONG write operation at phase 13 is also straightforward, since the value to be sent can be calculated during the spreading phase. We instruct readers to look at [30, Section 4.5] and [35, Section 3.4] for closer details.

Phases 1, 6, 7, 13 and 16 of the algorithm clearly require no routing steps. The injection phase 2 requires $T^{2D}_{inject}(N/P)$ routing steps, and the other injection phase 9 respectively requires $T^{2D}_{inject}(2N/P)$ steps. Sorting at phases 3 and 14 requires $T^{2D}_{sort}(Q) = 2.5\sqrt{Q}+o(\sqrt{Q})$ (deterministic) routing steps [31][12]. Finding out the center of a segment at phase 4 can be done in $T^{2D}_{compress}(Q) = 2\sqrt{Q}+O(1)$ routing steps [35]. Routing at phases 5, 10, and 15 takes $T^{2D}_{route}(Q)$, $T^{2D}_{back}(Q)$, and $T^{2D}_{inject}(N/P)$ routing steps (phases 15 and 2 are reverse operations), respectively. Synchronization at phases 8 and 11 requires $T^{2D}_{sync}(Q)$ routing steps, and spreading of packets at phase 12 requires $T^{2D}_{spread}(Q) = \sqrt{Q} + O(1)$ steps [35]. Thus, the expected number of routing steps required per P PRAM processors is

$$
\begin{aligned}
& \frac{T^{2D}_{inject}(N/P) + T^{2D}_{sort}(Q) + T^{2D}_{compress}(Q) + T^{2D}_{route}(Q) + T^{2D}_{sync}(Q) + T^{2D}_{inject}(2N/P)}{N/P} \\
& + \frac{T^{2D}_{back}(Q) + T^{2D}_{sync}(Q) + T^{2D}_{spread}(Q) + T^{2D}_{sort}(Q) + T^{2D}_{inject}(N/P)}{N/P} \\
& = \frac{\frac{1}{2}\sqrt{Q} + \frac{5}{2}\sqrt{Q} + o(\sqrt{Q}) + \sqrt{Q} + 2\sqrt{Q} + o(\sqrt{Q}) + 2\sqrt{Q} + \frac{1}{2}\sqrt{Q}}{\sqrt{Q}/4} \\
& + \frac{2\sqrt{Q} + o(\sqrt{Q}) + 2\sqrt{Q} + \sqrt{Q} + \frac{5}{2}\sqrt{Q} + o(\sqrt{Q}) + \frac{1}{2}\sqrt{Q}}{\sqrt{Q}/4} \\
& = 66 + o(1).
\end{aligned}
$$

What was said in the proof of Theorem 5.19 about the effect (on the expected number of routing steps) of rehashing and failing to accomplish routing tasks within certain time bounds, applies to Algorithm 5.21 too (i.e., the effect can be proved to be negligible).

■

5.3 Simulation on 3-dimensional coated meshes

It is easy to see, how to extend Algorithm 5.18 and Algorithm 5.21 to 3-dimensional coated meshes. Theorem 5.23 and Theorem 5.24 state the time complexity of such extensions. The theorems are based on bounds

$$T^{3D}_{inject}(x) = max(x, \frac{1}{2}\sqrt[3]{Q}),$$

$T^{3D}_{sync}(Q) = 3\sqrt[3]{Q}+O(1)$ [35], $T^{3D}_{sort}(Q) = 4.5\sqrt[3]{Q}+o(\sqrt[3]{Q})$ [31], $T^{3D}_{compress}(Q) = 3\sqrt[3]{Q}+O(1)$ [35], and $T^{3D}_{spread}(Q) = \frac{3}{2}\sqrt[3]{Q} + O(1)$ [35].

Theorem 5.23 *A $\mathcal{CM}^{3D}_{\mathcal{R}}(N, O(\log N))$ with $PCS(G^{\sqrt[3]{N}\times\sqrt[3]{N}\times\sqrt[3]{N}}_{mesh}, R^{O(\log N)}_{cbg}, Q_{ff})$ and $\mathcal{H}_{O(\log N)}$ family of hash functions, can simulate an N-processor EREW PRAM work-optimally using at most $78 + o(1)$ routing steps per P PRAM processors with high probability.*

[12] Recently, in [29] a deterministic algorithm for sorting in time $2\sqrt{Q} + o(\sqrt{Q})$ is presented, but to use it requires that the keys are from $[0, \ldots, N]$ (which they are not).

Proof.

$$\frac{T^{3D}_{inject}(N/P) + T^{3D}_{route}(Q) + T^{3D}_{sync}(Q) + T^{3D}_{inject}(2N/P) + T^{3D}_{back}(Q) + T^{3D}_{sync}(Q)}{N/P}$$
$$= \frac{\frac{1}{2}\sqrt[3]{Q} + 3\sqrt[3]{Q} + o(\sqrt[3]{Q}) + 3\sqrt[3]{Q} + \frac{1}{2}\sqrt[3]{Q} + 3\sqrt[3]{Q} + o(\sqrt[3]{Q}) + 3\sqrt[3]{Q}}{\sqrt[3]{Q}/6}$$
$$= 13 \times 6 + o(1) = 78 + o(1)$$

■

Theorem 5.24 *A $\mathcal{CM}^{3D}_{\mathcal{R}}(N, O(\log N))$ with $PCS(G^{\sqrt[3]{N}\times\sqrt[3]{N}\times\sqrt[3]{N}}_{mesh}, R^{O(\log N)}_{cbg}, Q_{ff})$ and $\mathcal{H}_{O(\log N)}$ family of hash functions, can simulate an N-processor CREW and all the presented CRCW PRAM models work-optimally using at most $153 + o(1)$ routing steps per P PRAM processors with high probability.*

Proof.

$$\frac{T^{3D}_{inject}(N/P) + T^{3D}_{sort}(Q) + T^{3D}_{compress}(Q) + T^{3D}_{route}(Q) + T^{3D}_{sync}(Q) + T^{3D}_{inject}(2N/P)}{N/P}$$
$$+ \frac{T^{3D}_{back}(Q) + T^{3D}_{sync}(Q) + T^{3D}_{spread}(Q) + T^{3D}_{sort}(Q) + T^{3D}_{inject}(N/P)}{N/P}$$
$$= \frac{\frac{1}{2}\sqrt[3]{Q} + \frac{9}{2}\sqrt[3]{Q} + o(\sqrt[3]{Q}) + \frac{3}{2}\sqrt[3]{Q} + +3\sqrt[3]{Q} + o(\sqrt[3]{Q}) + 3\sqrt[3]{Q} + \frac{1}{2}\sqrt[3]{Q}}{\sqrt[3]{Q}/6}$$
$$+ \frac{3\sqrt[3]{Q} + o(\sqrt[3]{Q}) + 3\sqrt[3]{Q} + \frac{3}{2}\sqrt[3]{Q} + \frac{9}{2}\sqrt[3]{Q} + o(\sqrt[3]{Q}) + \frac{1}{2}\sqrt[3]{Q}}{\sqrt[3]{Q}/6}$$
$$= 153 + o(1).$$

■

6 Conclusions

In this paper we presented a classification of work-optimal simulation results; proved that PRAM influenced routing situations can be solved on coated meshes almost within the diameter of routing machinery; and proposed the implementation of PRAM models on very realistic coated mesh structures. We proved that 2-dimensional and 3-dimensional coated meshes can simulate work-optimally EREW, CREW, ARBITRARY CRCW and STRONG CRCW PRAM models using only a constant number of routing steps per P simulated PRAM processors – i.e., the time to accomplish memory references asymptotically equals to the number of virtual processors each real processor is simulating. We could have proved the same result for other CRCW models too.

The constants provided by our proofs bounding the simulation time are relatively high, but experiments based on overloading alone (ignoring improvements such as fast buses for synchronization, toroidal mesh structure, wide channels, duplicating the connections of routing machinery, possibly faster clockrate of routing machinery etc) show that the constants can be made small – as low as 10 on a very reasonable load $N/P = d\sqrt[d]{N}$, and on a larger load even lower – both for 2-dimensional and 3-dimensional coated meshes, and for EREW, CREW, and CRCW simulations; see [33, 34].

In [2] a detailed analysis of work-optimal butterfly based PRAM implementation is presented. This construction can be seen as an improved version of Ranade's Fluent Machine [45], and it uses some of the improvements mentioned in the previous paragraph. If we have interpreted the results of [2] correctly, the number of routing steps required per P PRAM processors is 1 (or 2). Our constant (≈ 10) is worse, but perhaps not significantly. The differences are so small that the network cycle times and the overall regularity will obviously play a significant role. In this light, the constant length interconnections of mesh make it a very noteworthy alternative to the butterfly.

Another interesting aspect in the work-optimal implementations of PRAM by the butterfly and the 3-dimensional coated mesh is the number of nodes in the routing machinery. If no attempt to integrate nodes is made (as in [9] for the butterfly), the routing machinery of the butterfly based implementation has $(P+1)\log P$ nodes, whereas the 3-dimensional coated mesh implementation has $(P/6)^{\frac{3}{2}}$ nodes. It is quite surprising to notice that the 3-dimensional coated mesh actually has fewer nodes for $P \leq 2^{15}$. Even for $P = 2^{20}$ the ratio of nodes is only 3.3:1 in favor of the butterfly construction. It might well be that when the coated butterfly construction becomes better in terms of the number of nodes, then the lengths of connections and the overall regularity, the scalability, and the simplicity of the coated mesh play a more significant role than the number of nodes!

In spite of great efforts, the implementation of PRAM models must still be considered as an open problem. Hopefully, our investigation in mesh based architectures will contribute in future implementation proposals. For further study we would like to propose the following problems:

1. improve and simplify the presented PRAM implementations;
2. present an analytic prove that overloading the greedy routing algorithm on PRAM influenced simulation situation significantly improves the efficiency;
3. prove that simpler families of hash functions can be used instead of $\mathcal{H}_\zeta$; and
4. present an exact comparison of the butterfly and the coated mesh based work-optimal PRAM implementations in terms of hardware costs per provided computational power.

Acknowledgments

Authors would like to thank F. Meyer auf der Heide for pointing out an error in an earlier version of this report.

References

[1] F. Abolhassan, J. Keller, and W.J. Paul. On the Cost-Effectiveness and Realization of the Theoretical PRAM Model. Technical Report SFB Report 09/1991, Universität des Saarlandes, W-6600, Saarbrucken, Germany, 1991.

[2] F. Abolhassan, J. Keller, and W.J. Paul. On the Cost-Effectiveness of PRAMs. In *Proceedings, 3rd IEEE Symposium on Parallel and Distributed Computing, ACM Special Interest Group on Computer Architecture, and IEEE Computer Society*, pages 2 – 9, 1991.

[3] H. Alt, T. Hagerup, K. Mehlhorn, and F.P. Preparata. Deterministic Simulation of Idealized Parallel Computers on More Realistic Ones. *SIAM Journal of Computing*, 16(5):808–835, 1987.

[4] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith. The Tera Computer System. *ACM SIGARCH Computer Architecture News, International Conference on Supercomputing 1990*, 18(3):1 – 6, September 1990.

[5] 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.

[6] F. Meyer auf der Heide. Hashing Strategies for Simulating Shared Memory on Distributed Memory Machines. Technical Report TR-RF-93-006, University of Paderborn, Computer Science Department, June 1993.

[7] R. Bianchini, S. Dickey, J. Edler, G. Goodman, A. Gottlieb, R. Kenner, and J. Wang. The Ultra III Prototype. Technical Report Ultracomputer note 185, New York University, 1993. Also published in IPPS'93.

[8] J.L. Carter and M.N. Wegman. Universal Classes of Hash Functions. *Journal of Computer and System Sciences*, 18:143–154, 1979.

[9] D. Cross, R. Drefenstedt, and J. Keller. Reduction of Network Cost and Wiring in Ranade's Butterfly Routing. *Information Processing Letters*, 45(2):63 – 68, February 1993.

[10] M. Dietzfelbinger. Simulations Between Different Models of Parallel Computers. In *Proceedings, 9th International Conference on Fundamentals of Computation Theory, FCT'93*, pages 16 – 30, August 1993.

[11] M. Dietzfelbinger and F. Meyer auf der Heide. How to Distribute a Dictionary in a Complete Network. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 117 – 127, 1990.

[12] M. Dietzfelbinger and F. Meyer auf der Heide. Simple, Efficient Shared Memory Simulations. In *SPAA'93, 5th Annual Symposium on Parallel Algorithms and Architectures, Velen, Germany*, pages 110 – 119, June 1993.

[13] M. Dietzfelbinger, T. Hagerup, J. Katajainen, and M. Penttonen. A Reliable Randomized Algorithm for the Closest-Pair Problem, 1993. Manuscript.

[14] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Mayer auf der Heide, H. Rohnert, and R.E. Tarjan. Dynamic Perfect Hashing: Upper and Lower Bounds. In *IEEE Symposium on Foundations of Computer Science 29*, pages 524 – 531, 1988. A revised version of the paper will appear in *SIAM Journal on Computing.*

[15] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R.E. Tarjan. Dynamic Perfect Hashing: Upper and Lower Bounds. Technical Report TR-RI-91-77, University of Paderborn, Computer Science Department, January 1991.

[16] R. Drefenstedt and D. Schmidt. On the Physical Design of Butterfly Networks for PRAMs. In *Proceedings, Fourth Symposium on the Frontiers'92*, pages 202 – 209. IEEE Computer Society Press, USA, 1992.

[17] C. Engelmann and J. Keller. Simulation-Based Comparison of Hash Functions for Emulated Shared Memory. In *Proceedings of PARLE'93 Parallel Architectures and Languages Europe, Springer LNCS 694*, pages 1 – 11, 1993.

[18] Y. Feldman and E. Shapiro. Spatial Machines: A More Realistic Approach to Parallel Computation. *Communications of the ACM*, 35(10):60 – 73, 1992.

[19] 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.

[20] 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.

[21] L.A. Goldberg, M. Jerrum, T. Leighton, and S. Rao. Doubly Logarithmic Communication Algorithms for Optical Communication Parallel Computers, 1994. Submitted to SIAM Journal on Computing.

[22] 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*, June 1994.

[23] A. Gottlieb, R. Grishman, C.P. Kruskal, K.P. McAuliffe, L. Rudolph, and M. Snir. The NYU Ultracomputer – Designing an MIMD Parallel Machine. *IEEE Transactions on Computers*, 32:175–189, 1983.

[24] T. Hagerup and C. Rüb. Randomized Simulation of PRAMs on Processor Networks. Technical Report Parallele Algorithmen (WS 91/92), Universität des Saarlandes, Fachbereich 14, Informatik, January 1992.

[25] M.H. Davis Jr. and U. Ramachandran. Synchronization Primitves on an Optical Broadcast Ring. Technical Report GIT-CC-93/08, College of Computing, Georgia Institute of Technology, January 1993.

[26] A.R. Karlin and E. Upfal. Parallel Hashing – an Efficient Implementation of Shared Memory. In *Proceedings, 18th Annual ACM Symposium on Theory of Computing*, pages 160–168, 1986.

[27] R.M. Karp, M. Luby, and F. Meyer auf der Heide. Efficient PRAM Simulation on a Distributed Memory Machine. In *Proceedings, 24th Annual ACM Symposium on Theory of Computing*, pages 318 – 326, 1992.

[28] J. Keller. Hashing and Rehashing in Emulated Shared Memory. Technical Report CS-R9240, CWI, 1992.

[29] D. Krizanc. Interger Sorting on a Mesh-Connected Array of Processors. *Information Processing Letters*, 47(6):283 – 289, October 1993.

[30] C.P. Kruskal, L. Rudolph, and M. Snir. A Complexity Theory of Efficient Parallel Algorithms. *Theoretical Computer Science*, 71:95–132, 1990.

[31] M. Kunde. Concentrated Regular Data Streams on Grids: Sorting and Routing Near to the Bisection Bound. In *Proceedings, 32th Annual Symposium on Foundations of Computer Science*, pages 141 – 150, 1991.

[32] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays • Trees • Hypercubes*. Morgan Kaufman, San Mateo, CA, 1992.

[33] V. Leppänen. PRAM Computation on Mesh Structures. Technical Report R-93-9, University of Turku, Computer Science Department, November 1993. Ph.Lic. thesis.

[34] V. Leppänen. Experimental Results for Four Work-Optimal PRAM Simulation Algorithms on Coated Meshes. Technical Report R-94-2, University of Turku, Computer Science Department, January 1994.

[35] V. Leppänen and M. Penttonen. Simulation of PRAM Models on Meshes. Technical Report R-93-4, University of Turku, Computer Science Department, August 1993.

[36] V. Leppänen and M. Penttonen. Simulation of PRAM Models on Meshes. In *Proceedings, Parallel Architectures and Languages Europe, PARLE'94, Lecture Notes in Computer Science 817*, pages 146 – 158, July 1994.

[37] V. Leppänen and M. Penttonen. Work-Optimal Simulation of PRAM Models on Meshes. In *Proceedings of the Seventh Finnish Symposium on Computer Science in Liperi, Joensuu*, pages 91 – 119, January 1994.

[38] J.K. Littlejohn. Dynamic Bulk Synchronous Parallelism. Technical Report EPCC-SS92-16, Edinburgh Parallel Computing Centre, 1992. 18pp.

[39] W.F. McColl. General Purpose Parallel Computing. In A M Gibbons and P Spirakis, editors, *Lectures on Parallel Computation. Proc. 1991 ALCOM Spring School on Parallel Computation*, Cambridge International Series on Parallel Computation, pages 337–391. Cambridge University Press, 1993.

[40] W.F. McColl. Special Purpose Parallel Computing. In A M Gibbons and P Spirakis, editors, *Lectures on Parallel Computation. Proc. 1991 ALCOM Spring School on Parallel Computation*, Cambridge International Series on Parallel Computation, pages 261–336. Cambridge University Press, 1993.

[41] K. Mehlhorn and U. Vishkin. Randomized and Deterministic Simulations of PRAMs by Parallel Machines with Restricted Granularity of Parallel Memories. *Acta Informatica*, 21:339–374, 1984.

[42] D. Mosberger. Memory Consistency Models. Technical Report TR 93/11, Department of Computer Science, The University of Arizona, 1993.

[43] J.J. Olsen. *Control and Reliability of Optical Networks in Multiprocessors.* PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, May 1993.

[44] A.G. Ranade. How to Emulate Shared Memory. *Journal of Computer and System Sciences*, 42:307–326, 1991.

[45] A.G. Ranade, S.N. Bhatt, and S.L. Johnsson. The Fluent Abstract Machine. In *Proceedings, 5th MIT Conference on Advanced Research in VLSI*, pages 71–93, 1988.

[46] S.B. Rao. Properties of an Interconnection Architecture based on Wavelength Division Multiplexing. Technical Report TR-92-009-3-0054-2, NEC Research Institute, Princeton, January 1992.

[47] Y. Shiloach and U. Vishkin. A $\log n$ Parallel Connectivity Algorithm. *Journal of Algorithms*, 3:57–63, 1982.

[48] C.D. Thompson. Area-Time Complexity for VLSI. In *Proceedings, 11th Annual ACM Symposium on Theory of Computing*, pages 81 – 88, 1979.

[49] E. Upfal and A. Wigderson. How to Share Memory in a Distributed Systems. *Journal of the ACM*, 34:116–127, 1987.

[50] L.G. Valiant. A Bridging Model for Parallel Computation. *Communications of the ACM*, 33(8):103–111, 1990.

[51] L.G. Valiant. General Purpose Parallel Architectures. In *Algorithms and Complexity, Handbook of Theoretical Computer Science*, volume A, pages 934–971, 1990.

[52] L.G. Valiant. A Combining Mechanism for Parallel Computers. Technical Report TR-24-92, Harvard University, Aiken Computation Laboratory, 1992.

[53] P.M.B. Vitányi. Locality, Communication, and Interconnect Length in Multicomputers. *SIAM Journal on Computing*, 17(4):659 – 672, August 1988.

[54] R. Lyndon White. A Scalable Implementation of Globally-Shared Queues. Technical Report uwa-TR-93/3, The University of Western Australia, Department of Computer Science, April 1993.