Performance of Work-Optimal PRAM Simulation Algorithms on Coated Meshes

Ville Leppänen*
Department of Computer Science, University of Turku
Lemminkäisenkatu 14-18, Datacity, FIN-20520 Turku, FINLAND
E-mail: *Ville.Leppanen@cs.utu.fi*, FAX +358 21 333 8600, Tel. +358 21 333 8676

December 3, 1995

Abstract

We study the effect of varying the multithreading level of processors in work-optimal PRAM simulation algorithms on coated meshes. A coated mesh consists of a mesh connected routing machinery and P processor&memory pairs that form a coat on the routing machinery. The algorithms studied are based on greedy routing, sorting, improved virtual leveled network technique, combining queues method, and synchronization wave. Our results show that increasing the multithreading level considerably improves the simulation cost. The cost can be decreased below 5 routing steps per P simulated PRAM processors. In case of one algorithm, even costs 1.1 ... 2 are achieved.

1 Introduction

Work-optimal simulation of PRAM models means that a constant fraction of the aggregate power of processors can be given to (arbitrary) PRAM computations. We study the efficiency of five PRAM simulation algorithms on a structure called coated mesh. This is interesting, since (a) the coated mesh (rigid definition is given in Section 1.1) is a very realistic architectural model of parallel computation; (b) it allows work-optimal implementation of certain PRAM models by using Valiant's parallel slackness concept [15] (*i.e.*, by using multithreading); (c) the PRAM models have been very popular platforms in parallel algorithm design [5]; and (d) although several work-optimal PRAM simulations have been developed [4, 9, 10, 11, 15], relatively little attention has been paid on the exact simulation cost and the realisticness of proposed simulations.

The coated mesh is a mesh based routing machinery that is "coated" with processor&memory pairs. It is an attractive architectural model, since due to constant length connections it is truly scalable. As a mesh based construction, it is very regular and has a natural layout. The difference compared to an ordinary mesh is that the mesh nodes are much more simpler in the coated mesh – basically, they just forward packets. Ordinary meshes cannot simulate PRAM models work-optimally because of insufficient routing capacity compared to the diameter [8]. The idea in introducing a P-processor coated mesh with diameter δ is to increase the routing capacity (from $O(P)$) to $O(P\delta)$. By using Valiant's XPRAM framework, this allows work-optimal PRAM simulation.

The capacity is obtained by having $O(\delta)$ routing machinery nodes per each physical processor. However, the work and hardware complexity of routing machinery nodes is seen negligible, since they are considerably simpler than processor&memory pairs. The same reasoning is used in the connection of p-direct butterfly [15] that is analogous structure with the coated mesh. It uses the butterfly instead of the mesh and has shorter logical diameter but longer wires and more irregular physical layout.

All algorithms, except one, that we study in this paper can be proved to be work-optimal. However, the presented proofs do not take increasing of the multithreading level (how many threads, or virtual PRAM processors, per each real processor) properly into consideration. In the following, we shall refer to the multithreading level simply by *load*, and call *overloading* the increasing of load. We show that overloading can be used to significantly improve the efficiency of all the studied simulation algorithms. This is due to the simulation algorithms having parts whose cost is independent of the load. Another major reason behind improving is that relative cost of routing phases decreases as the load increases. A routing machinery whose capacity is $O(\delta)$ times larger than P enables, in principle, that

*This work was financially supported by the Academy of Finland.

random routing tasks complete in time $O(\delta + load)$. The relative cost of routing phase $O(\frac{\delta+load}{load})$ thus decreases as the load increases. When load exceeds the routing machinery capacity (per processor), the relative cost can be expected to show asymptotic behavior.

Is high multithreading level an unrealistic assumption? Tera [3] and SB-PRAM [1, 2] support 128 and 1024 threads per each physical processor, respectively. Supporting even more is not unrealistic from hardware point of view, since a thread basically requires only few tens of words of memory. From algorithm design point of view, the answer is not as clear. However, the facts that several problems have work-optimal NC-class PRAM-solutions [5] and parallel algorithms are often developed in practice for large modeling problems [13], suggest that high multithreading level is not an unrealistic assumption. Work-optimal NC-class algorithms use polylogarithmic time. Since work-optimality is defined with respect to sequential work (often polynomial wrt. the size of problem), the multithreading level is clearly high. Besides, the threads can originate from several concurrently running PRAM applications.

Next in Sections 1.1 and 1.2, we define the coated mesh and the PRAM models, respectively. Then we give an overview of the five PRAM simulation algorithms that work on coated meshes. In Section 3, we show how overloading improves their efficiency. A small comparison with work-optimal PRAM implementations proposed for the p-direct butterfly is made in Section 4. We draw conclusions in Section 5 and propose some open problems.

1.1 Coated mesh

We call *coated mesh* a mesh based routing machinery (of shape $\sqrt{Q} \times \sqrt{Q}$ or $\sqrt[3]{Q} \times \sqrt[3]{Q} \times \sqrt[3]{Q}$), which is coated with processor&memory pairs (see Figure 1). The processors are attached only to the nodes on the surface of the routing machinery. Each node of the routing machinery

- has a constant amount of local memory for computation and for storing packets;
- has at most Δ unidirectional links towards each neighboring node;
- is capable of forwarding at most one packet from each incoming edge (current size of output queue restricts forwarding) in unit time;
- has a buffer of size q packets for each outgoing edge (we assume only the "head" of each output buffer to reside at the receiving node); and
- is connected with bidirectional links to at most one processor per direction.

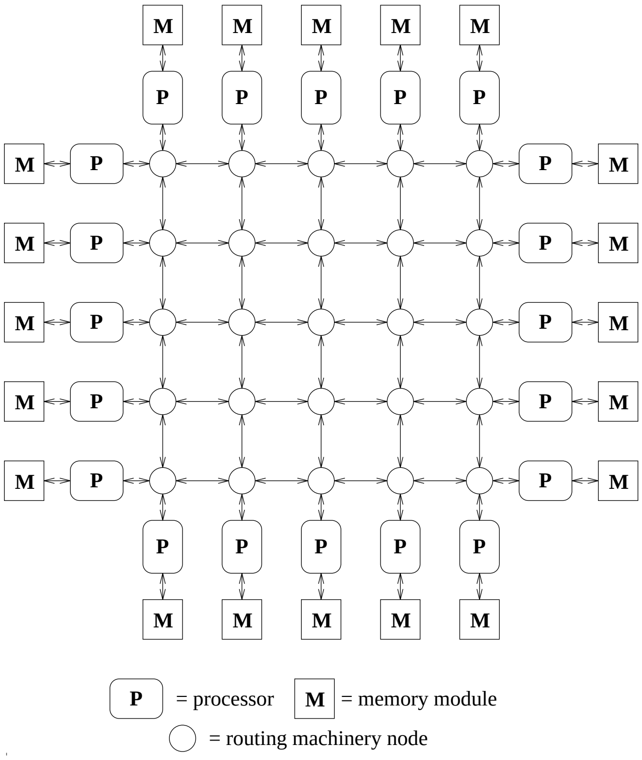

Figure 1: A 2-dimensional coated mesh.

We assume that each physical processor $\mathcal{P}_i$ is attached to a memory module $\mathcal{M}_i$. The routing machinery we take to be either a $\sqrt{Q} \times \sqrt{Q}$ mesh or a $\sqrt[3]{Q} \times \sqrt[3]{Q} \times \sqrt[3]{Q}$ mesh. In the 3D (2D) case, there are altogether $P = 6Q^{2/3}$ ($4\sqrt{Q}$) processor&memory pairs on 6 (4) surfaces. We denote such a d-dimensional coated mesh with $\mathcal{CM}(s^d, q, \Delta)$, where s is the side length – *i.e.*, $s^d = Q$. By $\mathcal{CM}^+(s^d, q, \Delta)$ we denote an extension, where the queues are combining queues – *i.e.*,

- the edge queues can combine, without increasing the size of the packet, in unit time an incoming packet with another packet already in the queue and referring to the same shared memory location (both packets must be either read or write requests);
- there is enough memory attached to each edge to store the necessary information to do decombining (split a reply into two replies); and
- the queues can do the decombining.

The clockrate of the routing machinery might be higher than that of the processors (by time multiplexing the usage of a physical link, one can create more virtual links). The routing machinery could be toroidal like that used in the Tera machine [3]. This would shorten the length of expected route.

1.2 PRAM models

The PRAM models consists of N processors and a shared memory M of size m. Each processor has some local memory, and it knows its own unique identifier PID (Processor IDentifier). The PIDs are integers in the range $0, 1, \ldots, N-1$. We denote the processors with $P_0, P_1, \ldots, P_{N-1}$. A step of a PRAM processor consists of a local operation, reading a shared memory location, *or* writing to a shared memory location. The phases of a step are executed synchronously, and each processor is assumed to finish the current step before any of the processors begins the next one. Here we discuss only the following PRAM models.

EREW: Concurrent reading or writing of any shared memory cell is forbidden. However, each shared memory cell may be read and written during the same step.

ARBITRARY CRCW : Concurrent read and write operations of the shared memory are allowed. If two or more processors write to the same 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.

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.

2 Simulation algorithms

The technique to simulate PRAM models on distributed memory machines is well-known [11, 15]. The shared memory of the PRAM is mapped into the memory modules of the coated mesh with a randomly chosen hash function h, and each real processor is assumed to simulate $\lfloor N/P \rfloor$ or $\lceil N/P \rceil$ PRAM processors. The only problem in the simulation is how to efficiently route read and write requests (caused by references to the shared memory) and replies in the routing machinery while preserving the atomic consistency of the PRAM.

In Sections 2.1 and 2.2, we give high level descriptions of two EREW simulation algorithms. In Section 2.3, we describe a sorting based CRCW simulation algorithm. Then, in Sections 2.4 and 2.5 we give two CRCW simulation algorithms, which are based on the combining queues method and on the virtual leveled network technique.

2.1 Two-phase EREW simulation

We do the simulation of an N-processor EREW PRAM on a P-processor coated mesh with Algorithm 2.1. It attaches two intermediate targets for each packet: One from pile of nodes attached to source and target processors. A *pile* of nodes related to some processor is a set of routing machinery nodes that form the closer half of an array of nodes perpendicular to the processor. The related processor is connected to one of those nodes. Each routing machinery node belongs to at most d piles.

We route packets from first intermediate target to the second with a greedy routing algorithm R_g that "corrects" the current coordinates of packets one by one in a predetermined order of axis (X, Y, Z). On arrival to the second intermediate target, the packets are forwarded greedily along removal edges to their target processor. Besides ordinary bidirectional connections, the coated mesh is assumed to have unidirectional *removal edges* from each routing machinery node towards all the processors whose pile it belongs. The algorithm uses queuing discipline Q_{ff} (farthest-first) to determine, which packet is sent first in case there are many candidates.

We assume that each processor can receive all the packets destined to it during one simulation round, and all the nodes can store the packets injected to the routing machinery before actually beginning a routing phase. For simplicity, N/P is an integer in the following.

Algorithm 2.1 (2-phase EREW simulation) *Each processor $\mathcal{P}_i$ ($i \in \{0, 1, \ldots, P-1\}$) simulates PRAM processors $P_{iN/P+j}$ ($j \in \{0, 1, \ldots, N/P-1\}$).*

1. *Translate instructions to packets, choose a random intermediate target from target pile for each packet, and inject packets evenly to the routing machinery.*
2. *Route the packets to targets via their second intermediate target by using R_g and Q_{ff}.*
3. *Execute the memory accesses and generate replies.*
4. *Ensure that all packets have been received.*
5. *Inject replies to the routing machinery.*
6. *Route replies to targets via the original intermediate nodes by using R_g and Q_{ff}.*
7. *Ensure that all packets have been received and update registers.*

A coated mesh $\mathcal{CM}(\sqrt{N}^2, O(\log N), 2)$ (respectively a $\mathcal{CM}(\sqrt[3]{N}^3, O(\log N), 2)$) can provably [9] simulate with Algorithm 2.1 an N-processor EREW PRAM work-optimally by using at most $36 + o(1)$ ($78 + o(1)$) routing steps per $P = 4\sqrt{Q}$ ($P = 6Q^{\frac{2}{3}}$) PRAM processors, with high probability. By "high probability" we mean probability $1 - N^{-k}$ for any fixed k (the choice of k affects $o(1)$). Expression "x routing steps per P

simulated processors" tells the cost of each PRAM operation. In the following, we shall call this the *relative cost of simulation* algorithm $\mathcal{G}(N, P, d, q)$, or simply the relative cost.

A relative cost of 36, or more, is too high. A method to improve it is overloading – having $N \gg Q$. The relative cost of injection phases (Phases 1 and 5) is almost independent of Q and N. For $N \gg Q$ their expected relative cost is close to 2 and with high probability it is less than 3. The synchronization phases (Phases 4 and 7) are independent of N – *i.e.*, their relative cost decreases as N increases. However, due to difficulties of exactly analyzing routing situations, where $N \gg Q$, we provide experimental results in Section 3. (Notice that the routing capacity of $\mathcal{CM}(\sqrt[d]{Q}^d, q, \Delta)$ can be up to $Qd\Delta$.)

2.2 Continuous EREW simulation

Assume the coated mesh to be of type $\mathcal{CM}(s^d, q, 4)$. There are two logical networks: One for delivering the requests and another for delivering replies. The logical networks are analogous, and in both of them, each node is connected with 2 bidirectional links to each neighbor. Besides the normal bidirectional connections and removal edges, there are unidirectional *injection edges* towards the center along each axis.

The purpose of having 4 bidirectional connections between neighboring nodes is to build a virtual acyclic directed graph (VDAG) from processors to memory modules and vice versa. Such a VDAG can be used to implement more sophisticated synchronization mechanism: *synchronization wave* [2, 11]. The idea is that when a source has sent all its packets on their way, it sends a synchronization packet. Synchronization packets from various sources push on the actual packets, and spread to all possible paths that the actual packets could go. When a (logical) node receives a synchronization packet from one of its inputs, it waits, until it has received a synchronization packet from all of its inputs, then it forwards the synchronization wave to all of its outputs. While waiting, the node forwards other packets. The synchronization wave may not bypass any actual packets and vice versa. When a synchronization wave sweeps over a VDAG based routing machinery, all nodes (and processors) receive exactly one synchronization packet via each input link and send exactly one via each output link.

The injection and removal edges together with R_g routing protocol define a VDAG for both logical networks in a natural way. The VDAG nature of routing prevents deadlocks. The synchronization wave approach makes PRAM simulation more flexible: The processors and memory modules may process arriving packets freely and the atomic consistency is preserved as long as the synchronization wave reflects between the memory modules and processors. Obviously, the processors can achieve a high utilization level.

Algorithm 2.2 (Continuous EREW simulation)

1. *Translate instructions to packets and attach two randomly chosen intermediate targets for each: One from source pile and the other from target pile.*
2. *Inject the packets, and a synchronization packet after them, to the routing machinery via injection edges.*
3. *Route the packets to first intermediate target by using injection edges, then to second intermediate target by using normal connections (with R_g and Q_{ff}), and finally to the target by using removal edges. The synchronization wave proceeds as explained.*
4. *Execute the memory accesses, generate replies, and inject them to the reply network. Also let the synchronization wave reflect to the reply network.*
5. *Route replies as normal packets.*
6. *On arrival to the target, update registers, fetch next instruction, and repeat the above. Let the synchronization wave reflect to the request network.*

All the above steps are executed simultaneously in pipelined manner.

By using the virtual leveled network routing results proved in [7] Algorithm 2.2, can be proved to function work-optimally, if h is chosen randomly from a class of polynomial hash functions and $q = \Omega(s)$. The latter requirement is due to transforming the VDAG to a virtual leveled network by adding virtual nodes to edges. The virtual edges can be simulated simply by having larger buffers. Whether the results in [7] can be extended to VDAGs so that q remains a constant is an open problem. Our experimental results suggest that constant size buffers are sufficient in practice.

2.3 Sorting based CRCW simulation

Algorithm 2.3 is obtained by "embedding" a CRCW on EREW simulation algorithm [5] into Algorithm 2.1. It works on a $\mathcal{CM}(s^d, q, 2)$ as Algorithm 2.1 but requires that the routing machinery nodes can do many elementary operations in addition to forwarding packets.

Algorithm 2.3 (CRCW using sorting)

1. *Translate instructions to packets and inject them to the routing machinery.*
2. *Sort packets according to the target.*
3. *Nodes save packets temporarily and find a representative for each group of packets with the same target.*

4. Route each representative to the target processor via a random node on the target pile with R_g and Q_{ff}.

5. Execute memory accesses and build replies.

6. Ensure that all packets are routed to their target.

7. Inject replies to the routing machinery.

8. Route replies back to the creators of representatives with R_g and Q_{ff}.

9. Ensure that nodes have received all replies.

10. Spread received values to processors having the same target and build replies to the original requests.

11. Sort new replies according to the injection point of the original request.

12. Route replies to processors and update registers.

The representative of a group of ARBITRARY write requests is obtained by choosing one (*e.g.*, the first) of the requests. In [9], we have proved that with Algorithm 2.3 a coated mesh $\mathcal{CM}(\sqrt{N}^2, O(\log N), 2)$ (respectively a $\mathcal{CM}(\sqrt[3]{N}^3, O(\log N), 2)$) can simulate an N-processor ARBITRARY CRCW PRAM work-optimally with relative simulation cost at most $66+o(1)$ $(153+o(1))$, with high probability. The results are based on Kunde's sorting results [6]. Again, we have difficulties in characterizing exactly the effect of overloading on the relative simulation cost when $N \gg Q$.

2.4 Combining queues methods

Algorithm 2.1 can be almost used for CRCW simulation simply by replacing the normal queues with combining queues. The following Algorithm 2.4 shows how Algorithm 2.1 needs to be modified.

Algorithm 2.4 (CRCW with combining queues)

Assume the coated mesh type $\mathcal{CM}^+(\sqrt[d]{N}^d, O(\log N), 2)$. Only the differences to Algorithm 2.1 are shown.

2. Route the packets to their target via a random node on target pile by using combining on route and storing information on nodes for decombining (when and where the combined packets came).

6. Repeat the routing process of Phase 2 in reverse order and do decombining on route.

7. Update registers.

Algorithm 2.4 can clearly simulate ARBITRARY CRCW model. Although experimental results in Section 3 indicate Algorithm 2.4 to behave very well, the algorithm is not known to be provably work-optimal.

2.5 Improved virtual leveled network

In a (directed) *leveled network*, each directed edge connects a node on some level i to another node on level $i+1$. A leveled network is assumed to consist of L levels. Mesh as such is not a leveled network, but it can be seen as a virtual leveled network [7]. Our reason to embed a virtual leveled network to a mesh is due to the routing results proved in [7] that can be used to derive PRAM simulation results. Notice that only the routing machinery is seen as a leveled network, not the whole coated mesh.

We do not use the virtual leveled network construction presented in [7] for toroidal multidimensional arrays as such, since it has depth $L = 2d\sqrt[d]{Q} - d$ in the d-dimensional case. Next, we describe how that construction can be improved to have depth $L = d\sqrt[d]{Q} + O(d)$ even without the toroidal connections. The idea is to move the entry points, though which packets enter from plateau i to plateau $i+1$, from the side to the middle of each axis.

Imagine the mesh to consist of $2d$ virtual plateaus – each node is simulating $2d$ virtual nodes. For simplicity assume that $\sqrt[d]{Q} \equiv 1 \ mod\ 2$, and let $\Upsilon = (\sqrt[d]{Q} - 1)/2$. In the following, $1 \leq j \leq d$. Denote mesh node $(x_1, x_2, \ldots, x_d)$ on plateau s, where $0 \leq x_j < \sqrt[d]{Q}$ and $1 \leq s \leq 2d$, with $(s, (x_1, x_2, \ldots, x_d))$. On plateau $2i-1$, there is a directed edge from $(2i-1, (x_1, \ldots, x_i, \ldots, x_d))$ to $(2i-1, (x_1, \ldots, x_i+1, \ldots, x_d))$, if $x_i < \Upsilon$, and to $(2i-1, (x_1, \ldots, x_i - 1, \ldots, x_d))$, if $x_i > \Upsilon$. Similarly, there is a directed edge from node $(2i, (x_1, \ldots, x_i, \ldots, x_d))$ to node $(2i, (x_1, \ldots, x_i - 1, \ldots x_d))$, if $0 < x_i \leq \Upsilon$, or to node $(2i, (x_1, \ldots, x_i + 1, \ldots x_d))$, if $\Upsilon \leq x_i < \sqrt[d]{Q} - 1$. From plateau $2i$ to $2i+1$ packets enter via connections between nodes $(2i, (x_1, x_2, \ldots, x_d))$ and $(2i+1, (x_1, x_2, \ldots, x_d))$ for all $0 \leq x_j < \sqrt[d]{Q}$ and $1 \leq i < d$. Similarly, node $(2i-1, (x_1, \ldots, x_{i-1}, \Upsilon, x_{i+1}, \ldots, x_d))$ is connected to $(2i, (x_1, \ldots, x_{i-1}, \Upsilon, x_{i+1}, \ldots, x_d))$ for all $1 \leq i \leq d$.

It is relatively easy to prove that by using

$$f_L(s, (x_1, x_2, \ldots, x_d)) = s - 1 + \sum_{k=1}^{\lfloor \frac{s}{2} \rfloor} \alpha(x_k) + \sum_{k=1}^{d} \beta(x_k),$$

where $\alpha(x) = |x - \Upsilon|$ and $\beta(x) = \Upsilon - \alpha(x)$, to define level numbers for virtual nodes, the whole constructions turns out to be a leveled network. If $\sqrt[d]{Q} \equiv 0 \ mod\ 2$, entry points $\Upsilon_1 = \sqrt[d]{Q}/2$ and $\Upsilon_2 = \Upsilon_1 - 1$ should be used instead of Υ. In fact, the depth of the above construction can be reduced by modifying the way packets enter from plateau s to plateau $s+1$.

Clearly, there is a unique path from a node on plateau 1 to any node on plateau $2d$. We call R_{ivln} a routing algorithm that routes packets from level to level according to above described improvement; keeps all the packets passing through (virtual) nodes sorted according to key $\{random\ rank, destination\ address\}$ (queuing discipline Q_{rp}); uses ghost packets as in [7]; and combines packets with the same target whenever they meet on route. We denote a coated mesh capable of the above operations with $\mathcal{CM}^\bullet(\sqrt[d]{Q}^d, q, \Delta)$. Although in R_{ivln} each node simulates $2d$ virtual nodes, it is not troublesome, since the communication needs are balanced so that the capacity of normal bidirectional connections is sufficient. The following algorithm is derived from Algorithm 2.1 by using our improved virtual leveled network construction.

Algorithm 2.5 (R_{ivln} based CRCW simulation)

2. *Route the packets to random intermediate node on target pile with R_{ivln} and Q_{rp} by using combining on route and storing information on nodes for decombining (when and where the packets came). Thereon route packets to target processors along removal edges.*

6. *Repeat the routing process of Phase 2 in reverse order and do decombining on route.*

7. *Update registers.*

As a consequence of [7, Theorem 2.11] it can be shown that Algorithm 2.4 on $\mathcal{CM}^\bullet(\sqrt{N}^2, O(1), 3)$ and $\mathcal{CM}^\bullet(\sqrt[3]{N}^3, O(1), 3)$ can simulate an N-processor ARBITRARY CRCW work-optimally by using $O(1)$ routing steps per P simulated PRAM processors. However, no exact expression for the relative cost is known to us.

3 Experimental comparison

In our experiments, we used meshes of side length 10, 30, and 50 for the 3D coated mesh (Q = 1000, 27000, 125000 and P = 600, 5400, 15000), and 50 and 150 for the 2D coated mesh. We varied the *overloading factor* $b = \frac{N}{Q}$ between 1 and 120, although we made experiments only on values 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 60, and 120 of b. The *load of each processor* (which is $Load = \frac{N}{P} = b\sqrt{Q}/4$ for the 2D coated mesh, and $b\sqrt[3]{Q}/6$ for the 3D coated mesh) was at most 200 in the 3D case, and at most 750 in the 2D case. Altogether, about 30 000 routing experiments were conducted on more than 300 different parameter value combinations of q, Q, b, and the simulation algorithm. For each experimented parameter combination, at least 30 experiments were made. The curves shown later are Bézier splines drawn via the measured points.

We measured the efficiency of simulation algorithm $\mathcal{A}$ in terms of *measured relative simulation cost* $G(s^d, b, q, \mathcal{A})$. We made our experiments on randomly generated patterns (EREW), and patterns produced randomly on the basis of distributions of Figure 2 (CRCW). The distributions of Figure 2 are, of course, only a tiny fraction of all possible distributions but we believe them to be representative enough.

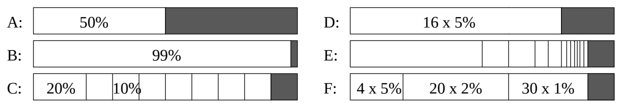

Figure 2: Distributions used in combining experiments. A gray area means randomly chosen targets, and the white areas each represent an amount (proportional to the area) of read requests with a common target memory location. An expression like "16 × 5%" is a shorthand for 16 areas of size 5%.

Some of the algorithms described in Section 2 have a phase where a global ensuring that all packets have reached their destination is required. We assume this to be implemented so that after T_1 routing steps a global check over the routing machinery is calculated, and if some packets are still in the routing machinery, the checking is repeated after $T_2 - T_1$ routing steps. A global check can be calculated in time $d\sqrt[d]{Q} + O(1)$. Time T_1 should be chosen so that only a small fraction of typical routing tasks require second checking. We have found T_1 = "average routing time + 2× standard deviation of the routing time" to be a good measure. In the cost calculation, T_2 is simply the maximum measured routing time.

3.1 Experiments with buffer size

Proofs related to the algorithms described in Section 2 do not give good exact bounds for the buffer size. Some buffers being occasionally full during the routing process does not decline the overall routing capacity significantly. One is tempted to believe that small buffer size is adequate in practice. On the other hand, it is not clear how big role the buffer size has in the cost of the whole construction.

Figure 3 describes simulation experiments made with Algorithm 2.1. As can be seen, the curves for different buffer sizes almost overlap each other: Obviously buffers of size 4 are adequate. Queues with higher capacity seem to make the routing to succeed faster, but the effect is quite negligible. On the basis of Figure 3, we conclude that the effect of overloading factor b on relative simulation cost is much larger than that of buffer size q. Similar, observations were made for all other simulation algorithms. For simplicity, we mainly use $q = 4$ in the following.

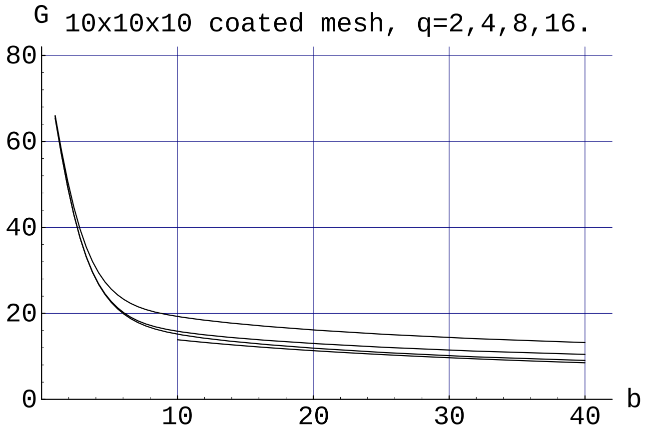

Figure 3: Influence of buffer size on the simulation cost. The curves are from the highest to the lowest: $G(10^3, b, 2, Alg.2.1)$, $G(10^3, b, 4, Alg.2.1)$, $G(10^3, b, 8, Alg.2.1)$, and $G(10^3, b, 16, Alg.2.1)$. Large buffers will not significantly improve the relative simulation cost.

3.2 EREW simulation results

3.2.1 Algorithm 2.1

In our experiments with Algorithm 2.1, we assume that after the injection phase each mesh node has exactly b packets with randomly chosen targets. The routing itself was done with Q_{fifo} and R_g using the principle that the incoming queues of each node are processed in cyclic order (by selecting a random starting point each time), and only the first packet of each queue is moved to the target queue, if there is room for that packet. We were only interested of Q_{fifo} because of its simplicity.

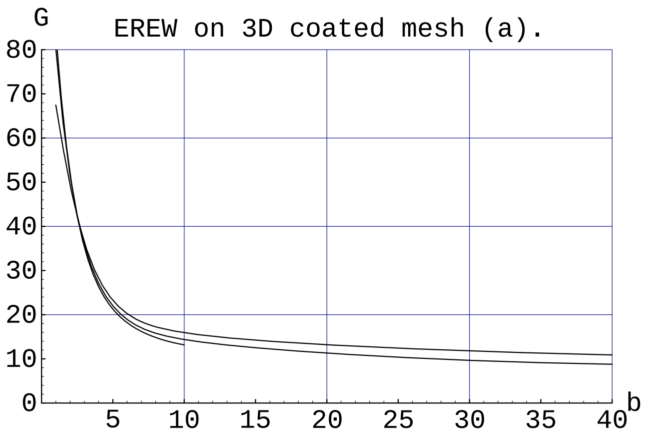

Figure 4: Algorithm 2.1 on 3D coated mesh. The shortest curve is $G(50^3, b, 4, Alg.2.1)$, and the two almost overlapping curves are $G(10^3, b, 4, Alg.2.1)$ and $G(30^3, b, 4, Alg.2.1)$ (the latter is in the middle).

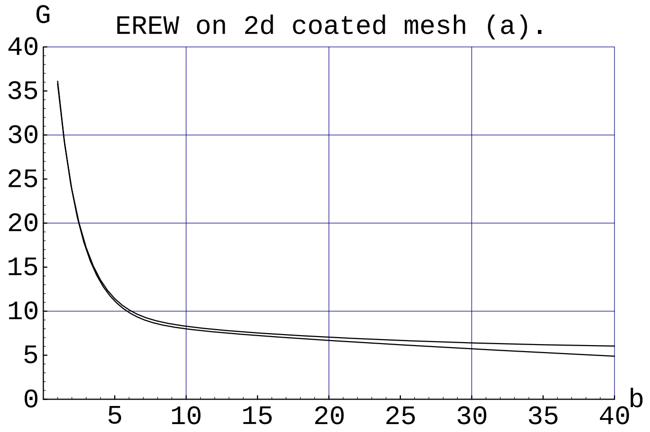

Figure 5: Algorithm 2.1 on 2D coated mesh. The overlapping curves are $G(50^2, b, 4, Alg.2.1)$ and $G(150^2, b, 4, Alg.2.1)$ (lower).

Figures 4 and 5 describe routing experiments made with EREW simulation Algorithm 2.1. The message is clear: Overloading significantly improves the efficiency up to $b \approx 5 \ldots 6$ in both cases. After that the relative cost improves slowly. Surprisingly, the curves are not completely independent of Q. The larger the mesh the better the efficiency with fixed q and b. Also increasing q has a positive effect on the relative cost. The results with the same q and b seem to yield better results in the 2D case. However, the relative simulation cost mainly depends on b.

As can be seen in Figures 4 and 5, the relative cost of Algorithm 2.1 can be pushed below **10** both for the 2D and 3D coated meshes. In fact at $b = 40$, we measured relative costs 4.9 and 8.8 for a $\mathcal{CM}(150^2, 4, 2)$ and a $\mathcal{CM}(30^3, 4, 2)$, respectively. We achieved even better results by increasing q. At $q = b/2$ and $b = 120$, relative costs 3.4 and 6.3 were achieved for a $\mathcal{CM}(50^2, q, 2)$ and a $\mathcal{CM}(10^3, q, 2)$, respectively. Our experiments suggest that by using only small buffers and properly overloading the coated mesh, about 7–8-fold speedup over the results of [9] can be achieved ($36 \longrightarrow 5$ and $78 \longrightarrow 9$). By using larger buffers, over 10-fold speedups seem achievable.

Overloading decreases the relative cost, since the relative cost of synchronization phases (Phases 4 and 7 of Algorithm 2.1) is approximately $2/b$. The relative cost of injection phases is almost independent of b – the cost of them is approximately 2–3. The cost of routing depends on *load* and the diameter of routing machinery δ, and is approximately $(c_d \times load + \delta)/load$. When load is very high, the relative cost of synchronization phases is negligible and the cost of injection phases is just over 2. Therefore it seems that $c_2 \approx 1$ but the same does not seem to hold for c_3.

3.2.2 Algorithm 2.2

In the test setting of Algorithm 2.2, the processors initially have *load* packets with randomly chosen targets. The two intermediate targets are chosen randomly as explained in Algorithm 2.2, and again queuing discipline Q_{fifo} is used. Synchronization wave separates consecutive steps, and a PRAM step is taken to be completed when last part of the synchronization wave (related to that PRAM step) leaves the processors. That moment is used to measure the time to simulate a PRAM step.

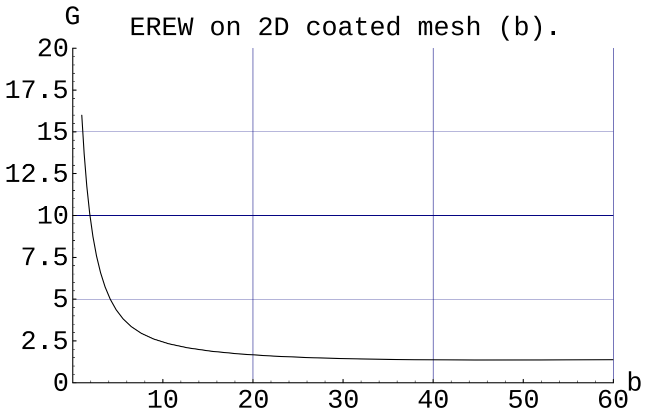

Figure 6: $G(50^2, b, 4, Alg.2.2)$.

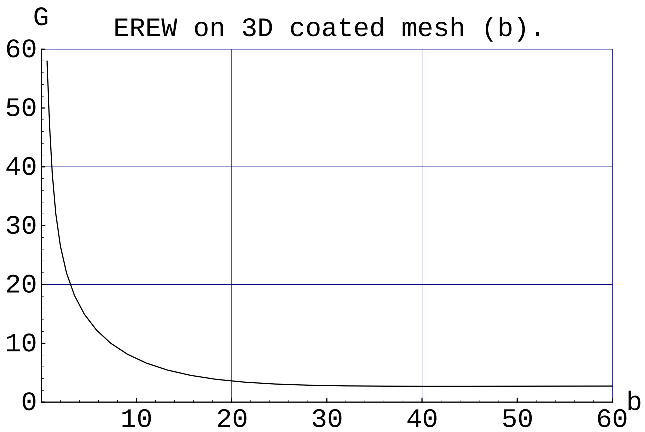

Figure 7: $G(10^3, b, 4, Alg.2.2)$.

Figures 6 and 7 show EREW simulation results obtained from experiments made with Algorithm 2.2 in 2D and 3D cases. Again the main observation is that efficiency improves considerably as b increases, up to $b \approx 10$. The reasons behind the phenomenon are the same as before. Since the relative cost of synchronization wave is small, the relative cost of simulation decreases rapidly **below** 2–3 when b increases.

On a $\mathcal{CM}(50^2, 4, 4)$ we measured relative cost 1.32 at $b = 40$. Doubling the size of queues yielded relative cost 1.17 at the same point. On a $\mathcal{CM}(50^2, 32, 4)$ we even measured relative cost 1.09 at point $b = 80$. Obviously in the 2D case, the relative cost quite freely approaches 1 as b and q are increased. In the 3D case, a $\mathcal{CM}(10^3, 4, 4)$ yielded relative cost 2.7 at $b = 36$ whereas doubling the queue size improved to relative cost to 2.1. On a $\mathcal{CM}(10^3, 32, 4)$ we even measured relative cost 1.68 at point $b = 180$. Although in the 3D case, the cost decreases below 2, it seems not to freely approach 1.

The reason is that in the 3D case the routing machinery bandwidth turns out to be insufficient. The bisection width of $\mathcal{CM}(s^d, q, \Delta)$ is (at most) Δs^{d-1}. However, Algorithm 2.2 uses the connections of $\mathcal{CM}(S^d, q, 4)$ for six different purposes. When routing packets form the first intermediate target to the second by using only normal bidirectional connections, the bisection width of that part of the algorithm is only s^{d-1}. On average half of the packets produced by the other half of the processors cross the bisection. Therefore the bisection width should be at least $2ds^{d-1}/(2 \cdot 2)$. In the 2D case this holds but not in the 3D case. In fact, the bisection width argument suggests the relative cost of Algorithm 2.2 to converge towards 3/2 on $\mathcal{CM}(s^3, q, 4)$ – the bisection bandwidth should be somehow increased.

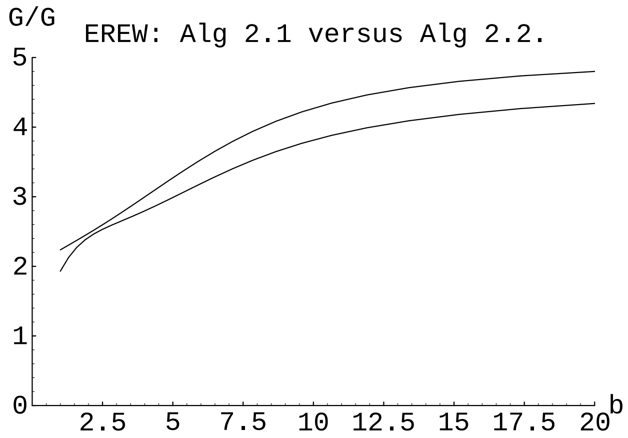

Figure 8: Algorithm 2.1 versus Algorithm 2.2. The curves (higher and lower) are

$$\frac{G(50^2, b, 4, Alg.2.1)}{G(50^2, b, 4, Alg.2.2)} \; and \; \frac{G(10^3, b, 4, Alg.2.1)}{G(10^3, b, 4, Alg.2.2)},$$

respectively.

Algorithm 2.2 gives clearly better results than Algorithm 2.1 but it also requires approximately twice as complex routing machinery. Is it still better if this is accounted? Figure 8 shows that Algorithm 2.2 is more than twice as good as Algorithm 2.1 in terms of relative cost. On the other hand, a $\mathcal{CM}(s^d, q, 2\Delta)$ is less than twice as costly as a $\mathcal{CM}(s^d, q, \Delta)$ in terms of hardware, and therefore Algorithm 2.2 is truly better than Algorithm 2.1.

3.3 CRCW simulation results

The purpose of sorting in Algorithm 2.3 is to transform CRCW style routing problem into EREW style problem. Therefore, we estimated the cost of Algorithm 2.3 directly on basis of EREW experiments made with Algorithm 2.1 (all except routing phases are deterministic). The cost of Algorithms 2.4 and 2.5 were estimated by conducting a set of experiments with patterns of Figure 2. Destination for packets were then chosen by first generating a distribution and then assigning a target for each packet randomly according to the distribution. Basically, all the same assumptions concerning processing of queues are as in the previous section. Further, the routing machinery nodes are assumed to "precombine" packets with the same target – after or during the injection phase but before the routing phase.

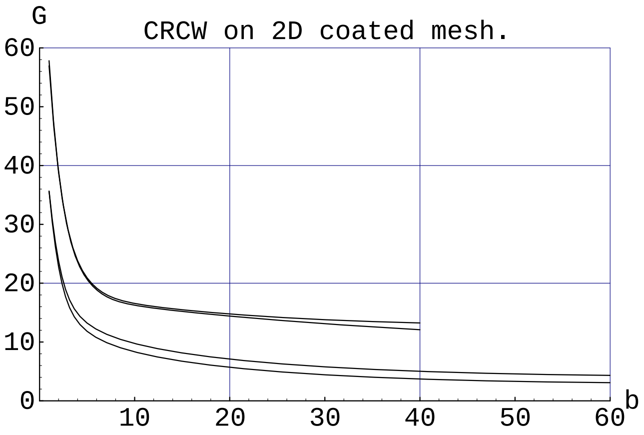

Figure 9: CRCW simulation cost on 2D coated mesh. Two highest curves (overlapping almost completely) are $G(50^2, b, 4, Alg.2.3)$ and $G(150^2, b, 4, Alg.2.3)$. The lowest curve is $G(50^2, b, max(3, \frac{b}{2}), Alg.2.4)$, and $G(50^2, b, max(3, \frac{b}{2}), Alg.2.5)$ is in the middle.

Comparison of the three CRCW simulation algorithms is made in Figures 9 and 10 by showing the curves for measured relative cost. The measured relative cost is now calculated as an average over all the experiments made with patterns of Figure 2.

First impression of Figures 9 and 10 is that overloading clearly improves the efficiency. In the 2D case, the increase of b seems to provide a relatively modest im-

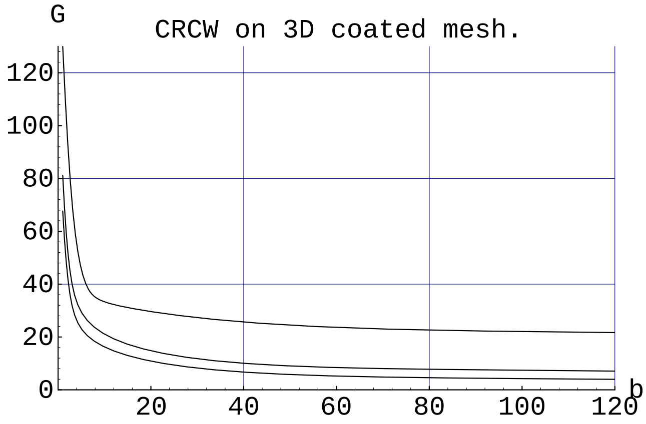

Figure 10: CRCW simulation cost on 3D coated mesh. The curves are from the highest to the lowest: $G(10^3, b, 4, Alg.2.3)$, $G(10^3, b, max(3, \frac{b}{2}), Alg.2.5)$, and $G(10^3, b, max(3, \frac{b}{2}), Alg.2.4)$.

provement after $b \approx 5 \ldots 6$. The same point for the 3D coated meshes seems to be $b \approx 7 \ldots 9$.

In both cases, the relative simulation cost decreases below 5 for Algorithm 2.4. At point $b = 10$ a $\mathcal{CM}^+(50^2, b/2, 2)$ achieved relative cost 6.0, and at point $b = 60$ cost 3.1. Similarly, a $\mathcal{CM}^+(10^3, b/2, 2)$ achieved relative costs 10.5, 5.1, and 4.0 at points $b = 10, 40$, and 120, respectively. Algorithm 2.5 did not quite achieve the same cost. On a $\mathcal{CM}^\bullet(50^2, b/2, 3)$ we measured relative costs 7.5, 5.6, and 4.3 at points $b = 10$, 20, and 60. For a $\mathcal{CM}^\bullet(10^3, b/2, 3)$, we measured relative costs 14.7 and 7.1 at points $b = 10$ and $b = 60$, respectively. The sorting based Algorithm 2.3 was clearly the worst. At best it achieved relative cost 13.2 on a $\mathcal{CM}(50^2, 4, 2)$ at point $b = 40$, and relative cost 21.7 at point $b = 120$ on a $\mathcal{CM}(10^3, 4, 2)$.

We also observed that pattern A produces the worst result for almost all methods. Obvious conclusion is that the more we have packets to combine the faster we can solve the routing problem.

We conclude that overloading can be used to improve the efficiency of Algorithm 2.3 by 5–7-fold. In many cases, the improved virtual leveled network technique proves out to be better than the sorting based solution. However, the combining queues method seems to be even more effective: It seems to cost about as much as an EREW simulation based on Algorithm 2.1! An obvious remaining question is that can one modify the combining queues method to work on Algorithm 2.2?

3.4 Summary of observations

The larger the buffers the better the simulation time. For relatively small values of b (*e.g.*, $b \leq 10$), buffer size

$q = 4$ gives good results. Values $b > 10$ will not significantly improve the relative cost. As expected, the relative cost in simulation algorithms is almost independent of the side length $\sqrt[d]{Q}$. The smaller dimensionality d the better relative cost with respect to b, since the routing capacity with respect to the expected work (expected length of route $\times$ number of packets) is better the smaller the d.

In the EREW case Algorithm 2.2 is nearly optimal, but in the CRCW case all the algorithms leave a lot to hope for. Sorting based solution is too costly. The combining queues method works very well, but the simplicity of $\mathcal{CM}^{\bullet}$ over $\mathcal{CM}^{+}$ makes the virtual leveled network based approach more attractive.

4 Mesh versus butterfly

The original stimulus to study the coated meshes was to find alternatives for butterfly based work-optimal PRAM constructions [1, 2, 12, 15]. In [9], we presented such constructions for the coated meshes, although the constant factors left a lot to hope for. The advantage of P-direct butterfly over the coated meshes is a small graph-theoretic diameter $O(\log P)$, which also allows the overloading factor to be asymptotically smaller. However, the problem with the butterfly is its physical layout – the largest physical distance between two nodes is of the same order for both interconnection types: Some connections between logical neighbors must necessarily be of length $\Omega(\sqrt[3]{P}/\log P)$ in any 3D layout of P-direct butterfly [9]. In fact, Thompson's VLSI area-time complexity result [14] implies wire length bound $\Omega(P/\log^2 P)$ for any 2D VLSI layout. Naturally, physical layouts for the butterfly are not as regular as for the mesh.

On the other hand, in order to avoid some timing and load balancing issues one might have to operate on the condition of the length of longest wires. This would restrict the exploitable clockrate and/or scalability. The effect of long wires can be reduced by scattering buffers on long wires at regular intervals, and by pipelining the communication over long wires. This increases the logical diameter and at the same time changes the whole structure towards the mesh.

The coated meshes do not have such severe layout problems. Yet, given sufficient amount of parallel slackness, the coated meshes can be made to simulate PRAM models with relatively small cost. In [2], the relative cost was made very close to 1 for the P-direct butterfly by using many hardware improvements. Those improvements could also be applied to the coated meshes to further improve the efficiency.

Interesting questions that remain are the size of routing machinery, the hardware complexity of nodes and processors, and the amount of parallel slackness required. We would very much like to see a hardware comparison between the 3D coated mesh and the P-direct butterfly. The sufficient parallel slackness $Load$ was concluded to be approximately $10 \times \sqrt[3]{Q}/6$ for the 3D coated mesh. For $Q = 10^6$, this is less than 200, and for $P = 10^6$, it is approximately 650. Thus, even for a relatively large 3D coated mesh the amount of parallel slackness should not be a serious obstacle.

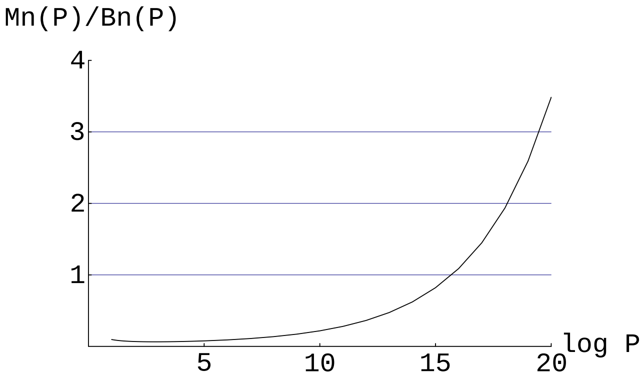

Figure 11: Ratio of routing machinery sizes. Denote by $Bn(P) = P(\log P + 1)$ the number of routing machinery nodes in an ordinary P-direct butterfly implementation. Respectively, denote by $Mn(P) = (P/6)^{1.5}$ the size of a P-processor 3D coated mesh routing machinery. Surprisingly, $Mn(P) < Bn(P)$ for $P \leq 2^{15}$.

An interesting aspect is the number of nodes in the routing machinery (=extraneous hardware). If no attempt to integrate nodes is made, then the routing machinery of the P-direct butterfly has $P(\log P + 1)$ nodes, whereas the 3D coated mesh implementation has $\left(\frac{P}{6}\right)^{\frac{3}{2}}$ nodes. It is quite surprising to notice that the 3D coated mesh actually has less nodes for $P \leq 2^{15}$. Even, when $P = 2^{20}$, the ratio of nodes is only 3.3:1 in favor of the butterfly construction (see Figure 11). Unfortunately, for the 2D coated mesh the corresponding ratios are clearly worse.

5 Conclusions

We have provided experimental evidence that EREW and CRCW PRAM models can be simulated on the 2D and 3D coated meshes with relative cost clearly below 10 – even below 5 – routing steps. In the EREW case, Algorithm 2.2 was found to be nearly optimal, giving relative costs below 2 – even relative cost 1.1 was achieved. Of the three CRCW simulation algorithms we found the combining queues based algorithm to be the best in terms of the number of required routing steps. We leave as an open problem the question whether or not this holds when the hardware requirements (implied by the algorithms) are taken into account. We also would like to know whether the combining queues method can be beaten?

Acknowledgments

The author would like to thank anonymous referees for helpful comments and Martti Penttonen also for encouragement.

References

[1] F. Abolhassan, R. Drefenstedt, J. Keller, W.J. Paul, and D. Scheerer. On the Physical Design of PRAMs. *The Computer Journal*, 36(8):756 – 762, 1993.

[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] 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.

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

[5] R.M. Karp and V. Ramachandran. Parallel Algorithms for Shared-Memory Machines. In *Algorithms and Complexity, Handbook of Theoretical Computer Science, Volume A*, pages 869–932. Elsevier Science Publishers B.V., 1990.

[6] 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.

[7] F.T. Leighton, B.M. Maggs, A.G. Ranade, and S.B. Rao. Randomized Routing and Sorting on Fixed-Connection Networks. *Journal of Algorithms*, 17(1):157 – 205, July 1994.

[8] 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.

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

[10] F. Meyer auf def Heide. Hashing Strategies for Simulating Shared Memory on Distributed Memory Machines. In F. Meyer auf der Heide, B. Monien, and A.L. Rosenberg, editors, *Proc. of Parallel Architectures and Their Efficient Use, First Heinz Nixdorf Symposium, LNCS 678*, pages 20 – 29. Springer-Verlag, 1992.

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

[12] 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.

[13] H.J. Siegel, S. Abraham, B. Bain, K.E. Batcher, T.L. Casavant, D. DeGroot, J.B. Dennis, D.C. Douglas, T-Y. Feng, J.R. Goodman, A. Huang, H.F. Jordan, J.R. Jump, Y.N. Patt, A.J. Smith, J.E. Smith, L. Snyder, H.S. Stone, R. Tuck, and B.W. Wah. Report of Purdue Workshop on Grand Challenges in Computer Architecture for the Support of High Performance Computing. *Journal of Parallel and Distributed Computing*, 16(3):199 – 211, 1992.

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

[15] L.G. Valiant. General Purpose Parallel Architectures. In *Algorithms and Complexity, Handbook of Theoretical Computer Science*, volume A, pages 943–971, 1990.