Efficient Two-Level Mesh based Simulation of PRAMs

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

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: *Martti.Penttonen@cs.joensuu.fi*

Abstract

We consider work-optimal simulations of PRAM models on coated block meshes. A coated block mesh consist of π-processor blocks and $\pi \times \pi$ or $\sqrt{\pi} \times \sqrt{\pi} \times \sqrt{\pi}$ router blocks. The router blocks form a 2-dimensional or a 3-dimensional regular mesh, and the processor&memory blocks are located on the surface of the block mesh. As a generalization of the coated mesh, the 2-dimensional and 3-dimensional coated block meshes simulate EREW, CREW, and CRCW PRAM models work-optimally. Moreover, the simulation cost is very moderate. Using certain additional improvements and proper amount of parallel slackness, the cost can be decreased clearly below 2 routing steps per simulated PRAM processor.

The coated block mesh is actually an instance of a more general two-level construction technique, which uses a seemingly inefficient but scalable solution on top of a non-scalable but efficient solution. Within blocks (chips) brute force techniques are applied, whereas the mesh structure on top makes the whole construction modular, simple, and scalable. The parameter π provides a method to balance the construction with respect to the available technology.

1 Introduction

The standard approach to PRAM (whose definition is given in Section 2.1) implementation is to map the shared memory into a collection of P

*This work was possible due to a grant provided by the Computer Science Dept. of the University of Turku.

distributed memory modules, to attach a memory module to each real processor, and to map the N virtual processors of the PRAM to the P real processors of the parallel computer. Each real processor is assumed to simulate at most $\lceil N/P \rceil$ *virtual processors* (threads). The operations on shared 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 atomic consistency.

The diameter ϕ of the routing machinery usually determines a lower bound for the latency of messages. For many popular interconnection networks, the graph theoretic diameter is $\Omega(\log P)$. If $P = N$, then the latency often determines the cost of PRAM simulation. This kind of simulation results have been proved for many interconnection networks, see *e.g.* [14, 16, 19, 20].

Multithreading ($N > P$) can be used to balance the local computation and the global communication, and thus to improve the efficiency of PRAM simulations. In Valiant's XPRAM framework [22], the simulation is done in supersteps of length L, where $L = \Omega(N/P)$. During a superstep, the real 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, which is used 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 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.

1.1 Previous results

Work-optimal PRAM simulation results have been proved for completely connected graphs (type (i)-results) in many papers [3, 4, 7, 22]. In [7], Goldberg, Matias, and Rao provide so far the best (asymptotically smallest latency) work-optimal $P \log\log P$-processor EREW PRAM simulation. Some of the simulation results are recently surveyed in [8, 17].

If the diameter ϕ of the routing machinery is not $O(1)$, then to hide the delay we must either assume that each node can deal with $\Omega(\phi)$ routing requests at each step (type (ii)-results), or assume that the routing machinery is separated from the processor&memory pairs and is considerably larger than P (type (iii)-results). In the latter case, the nodes only need to make simple routing decisions, and therefore it might be acceptable to ignore their hardware complexity and work.

In [22], Valiant proves a (ii)-type EREW-result for a P-node binary hypercube. He also gives a (iii)-type EREW-result for a P-input butterfly. Similar results are also established for most of the CRCW PRAMs in two ways: By extending the routing machinery with combining [20], or by requiring that $N = P^{1+\epsilon}$ for some $\epsilon > 0$ [22]. In [15], we prove type (iii)-results for EREW and CRCW PRAMs by taking the routing machinery to be a 2-dimensional mesh or a 3-dimensional mesh. Both for EREW and CRCW simulation, the work-optimality is achieved, when $N = \Omega(P^{\frac{d}{d-1}})$. In [9], we prove (iii)-type EREW-result for 2-dimensional and 3-dimensional meshes of trees. There,

$N = \Omega(P \log P)$.

1.2 Some critique

Because of the degree, the type (i)-results on completely connected graphs are not very realistic with current technology. This is much better taken into consideration in the mesh of (optical) buses [6, 21] (MOB), which in the d-dimensional case is essentially a Cartesian product of d completely connected graphs. The degree of nodes is now only $d\sqrt[d]{P}$. In theory, at least, one should remember that the physical diameter of a P-processor system is $\Omega(\sqrt[3]{P})$, and the velocity of light restricts the propagation of messages. In practice however, the message production rate and the size of P may make the unit time communication assumption acceptable.

Valiant's type (ii)-result is also based on some quite unrealistic assumptions. The degree $\log P$ makes it difficult to extend the hypercube physically. The lengths of connections increase, when the system is scaled up. Moreover, it is somewhat unfair to assume that a processor can deal with all of its $\log P$ input and output links in time unit.

The results based on ignoring the complexity of simple nodes have also a price to pay: The routing machinery is large in terms of the number of routing machinery nodes. However, the hardware complexity ratio of a processor&memory pair and a routing machinery node might be significant, *e.g.*, of one order of magnitude. The unit time assumption and scalability are also problematic in the butterfly. However, our mesh based solution is physically scalable, since the connections are of constant length. Moreover, in the 3-dimensional case, the size of the routing machinery is only $\sqrt{P}/6^{1.5}$ times P [15]. If $P = 10^4$, then this ratio is approximately 7. Notice that in the butterfly case, the ratio is $\log P + 1$, which is approximately 15 for $P = 10^4$.

The proposed work-optimal simulations differ also with respect to the required multithreading level (threads per processor), which typically is $\Omega(\phi)$, where ϕ is the graph theoretic diameter of underlying routing machinery. The number of required routing steps per thread decreases as the multithreading level of processors increases. Yet, it is clear that the multithreading should be as small as possible. Not because it is impossible to support a large number of threads in hardware[1], but the thread requirement sets restrictions for successful programming.

1.3 Contributions

In [10], we concluded the EREW simulation cost to be approximately $10 \ldots 14$ routing steps per simulated processor on a 3-dimensional coated mesh (see Section 2), when the load is approximately $0.7 \ldots 1 \times \sqrt{P}$. The thread requirement is due to a relatively large logical diameter $\sqrt{P}/2$. By *grouping* together a bunch of routing machinery nodes, the diameter, as well as the load requirement, decreases. We group nodes to a *router block* by using a sparse matrix type solution within

[1] The Tera computer system supports 128 threads per each physical processor. Each thread consists of 40 registers (64-bits). The SB-PRAM [1] supports 32×32 threads per each physical processor.

each router block. Although the ratio of the number of elementary routing machinery nodes and processors is already acceptable in the 3-dimensional coated mesh, it is further improved in the coated block mesh (by factor of $\pi^{\frac{1}{d-1}}$ in the d-dimensional case).

Compared to the coated mesh, the processors are divided to groups of π processors, and one shared memory module is integrated to each group. Each such *processor&memory block* is considered as a small PRAM that can be implemented using special constructions, *e.g.*, π-port memory technique [5]. The idea of both the router and processor&memory blocks is that in small scale one can use special constructions (brute force) without introducing any serious restrictions on the overall scalability of the system. Brute force solutions can be very efficient, if they are applied within chips. By integrating somewhat costly small scale solutions to an elegant large scale solution, we hope to produce an efficient and scalable overall architecture solution. Large scale solutions tend to be slow in small scale whereas fast special solutions are often non-scalable. By properly choosing (depending on available technology) the parameter π, we hope to find solutions, where the cost of communication, memory access, and thread management primitives are properly balanced with respect to the requirements of PRAM simulations.

Our EREW simulation experiments on our coated block mesh indicate that the simulation cost can be decreased below 2 in the 3-dimensional case, and to $1.1 \dots 1.3$ in the 2-dimensional case. In the 2-dimensional case, this means that 75% – 90% of the raw processing power of the system is in good use, although the stepwise synchronous execution is guaranteed automatically and nothing is assumed about the locality of the data. Our results are based on the assumption that each PRAM processor makes an external memory reference at each step. This is hardly the case in practice, and thus one can hope to further improve the simulation cost by designing the processors so that higher priority is given for non-local memory references.

In the next section, we give a detailed definition of the PRAM models and the coated block mesh. We discuss our PRAM simulations on the coated block mesh in Section 3, and speculate with methods to improve the simulations in Section 4. In Section 5, we provide experimental results of the simulation cost for certain load values. Finally, in Section 6 we present observations about the feasibility of this kind of PRAM implementations, and suggest some topics for further study.

2 Coated block mesh architecture

Our coated block mesh uses small PRAM modules as building blocks, and is designed especially for implementing the PRAM models. Next, in Section 2.1 we give a short definition of the PRAM models, and in Section 2.2 we define our coated block mesh by comparing it to the coated mesh [15].

2.1 PRAM

The 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 ($\in \{0, 1, \ldots, N-1\}$). 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. A read instruction returns the value of the memory location in question *before* the current step. Here we discuss only the following PRAM models, although the results are valid for many other PRAM models, too.

EREW: Concurrent reading and concurrent writing of any shared memory cell is forbidden. However, N concurrent memory operations can be performed, and each shared memory cell may be read and written (once) 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.

2.2 Coated mesh versus coated block mesh

Informally, the coated mesh [15] consists of a 2-dimensional or 3-dimensional mesh based routing machinery that is coated with processor&memory pairs (see Figure 1). The coated block mesh is a generalization of the coated mesh. There π processors and their memory modules are integrated to a processor&memory block (pm-block). Respectively, a punch of routing machinery nodes are integrated to a router block. The coated block mesh is then composed of blocks as the coated mesh is composed of router nodes and processor&memory pairs. In a way, the routing task is partially moved from the routing machinery to the pm-blocks. Another way to view the coated block mesh is to consider π separate coated meshes, which are merged by integrating corresponding routing machinery nodes and processor&memory pairs together.

Formally, the d-dimensional *coated block mesh* $\mathcal{CBM}(P, \pi, d)$ consists of P/π pm-blocks $A_0, A_1, \ldots A_{P/\pi-1}$ of π processors and one shared memory module, and $\left(\frac{P}{\pi 2d}\right)^{\frac{d}{d-1}}$ router blocks $B_{i_1,i_2,\ldots,i_d}$, where $0 \leq i_j < \sqrt[d-1]{P/(\pi 2d)}$, and $1 \leq j \leq d$. Each router block has π bidirectional links on each of its $2d$ surfaces, and is capable of receiving and sending one message per link in one time unit. The router blocks $B_{i_1,i_2,\ldots,i_d}$ are connected in the shape of a d-dimensional regular mesh, and corresponding links between neighboring router blocks are connected to each other.

2.2.1 Router block

For simplicity, we assume that the internal structure of a router block is a d-dimensional $\sqrt[d-1]{\pi}$-sided sparse matrix with π router nodes. Moreover, we assume that there is a queue of size q associated to each link, and packets entering router block $B_{i_1,i_2,\ldots,i_d}$ via link k on surface j, can leave the block only via link k on some surface l. The function of each

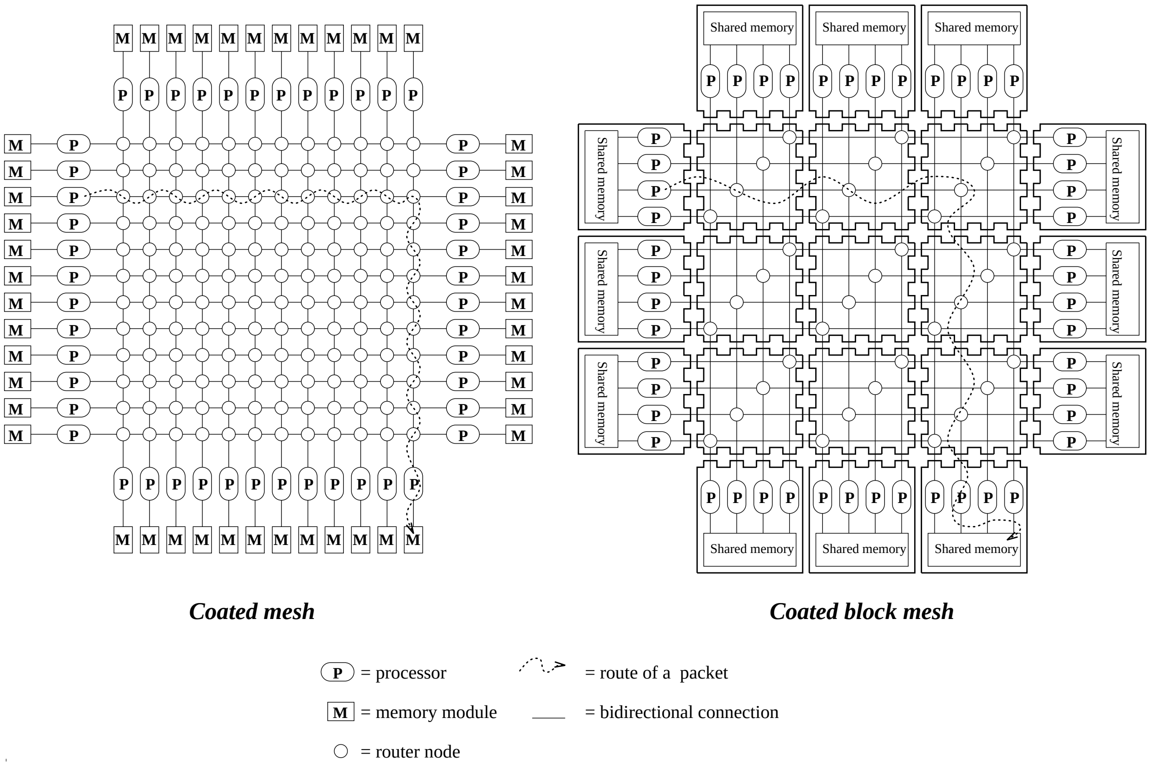

Figure 1: Coated mesh versus coated block mesh in 2-dimensional case. In the coated block mesh, the parameter π has value 4. Dotted splines represent the "same" route in both architectures.

router node is a design parameter that is defined by the simulation algorithm.

Assume that router nodes and their I/O-entries are located in a d-dimensional $\sqrt[d-1]{\pi}+2$-sided space. Then, the I/O-entries are on positions $(x_1, x_2, \ldots, x_d)$, where exactly one of $x_1, x_2, \ldots, x_d$ is either 0 or $\sqrt[d-1]{\pi}+1$. The I/O-entries having same coordinate x_i equal 0 (or $\sqrt[d-1]{\pi}+1$) form surface S_{2i} (respectively S_{2i+1}). The indexing of I/O-entries on surface S_j is a mapping $f_j : \{1, \ldots, \sqrt[d-1]{\pi}\}^{d-1} \mapsto \{0, \ldots, \pi-1\}$.

The internal structure of a router block is a sparse matrix, but for constructibility we would like to connections to be axial from each router node to each of the $2d$ I/O-entries that it is connected. Therefore, we define the indexing on surfaces S_{2i} and S_{2i+1} to be mirror images of each other – *i.e.*, $f_{2i}(a_1, a_2, \ldots, a_{d-1}) = f_{2i+1}(a_1, a_2, \ldots, a_{d-1})$ for each $0 \leq i \leq d$. To complete the definition of the internal structure of a router block, we also assign an indexing g to the internal nodes. Partial injective mapping g is a mapping from $\{1, \ldots, \sqrt[d-1]{\pi}\}^d$ to $\{0, \ldots, pi-1\}$. There is one requirement concerning g: Each router node x and its projective image on any of the $2d$ surfaces must have the same index, *i.e.*, for all $1 \leq i \leq d$:

$$g^{-1}(k) = \{x_1, x_2, \ldots, x_d\} \Longrightarrow f_{2i}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_d) = k.$$

Consequently, a projective mapping of g^{-1} with respect to any surface covers the whole surface. Notice that by defining g and one f_j, *e.g.* f_0, we at the same time define the rest of the f_j's. Clearly, the definition of f_0 is irrelevant as long as it is bijective.

Now, the question is that is there such a partial mapping g_d in the d-dimensional case for $d = 2$ and $d = 3$? Clearly, $g_2^{-1}(k) = \{k, k\}$ is such a mapping. It is not hard to prove that in the 3-dimensional case we can use, *e.g.*,

$$g_3^{-1}(\sqrt{\pi} \times k + l) = \{k + 1, l + 1, ((k + l) \ mod \ \sqrt{\pi}) + 1\},$$

where $0 \leq k, l < \sqrt{\pi}$.

2.2.2 Processor&memory block

The pm-blocks are on the surface of the router blocks based routing machinery. Together they "coat" the whole routing machinery. Each pm-block consists of π processors and one π-port shared memory module. The out-degree (and the in-degree) of each processor is 1, and thus the degree of a single pm-block is π. We assume that each processor can receive and send a packet during one time step.

For simplicity, we take the memory consistency model of our shared memory modules to be the same as the model that the whole coated block mesh is simulating. We do not define, how the π-processor pm-blocks are implemented, but just assume that fulfilling any memory request properly, with respect to the simulated PRAM model, takes unit time. We take π to be a small constant, and therefore possible implementations are, *e.g.*, a π-port memory [5], and a complete graph based π-processor distributed memory machine. In the previous, we naturally assume that $\pi, P, d, \sqrt[d-1]{P/(\pi 2d)} \in I\!N$, and $P \equiv 0 \ mod \ \pi$.

3 Simulation of PRAM models by CBM

A somewhat standard approach [20] to simulate the PRAM models is to first assume that the shared memory of the PRAM is distributed with a randomly chosen hash function h into the distributed memory modules; in our case, into the shared memory modules of π-processor processor&memory blocks. Then, the simulation is done by translating the memory references to packets, which are routed via the routing machinery to targets. Replies are generated to the read requests, and routed back to senders of the read requests. Finally, the system is completed with a synchronization mechanism, which guarantees that the simulation of consecutive steps maintains the atomic memory consistency of the PRAM.

The idea in shared memory hashing is to initially arrange the shared memory according to a randomly chosen hash function h, and to rearrange the memory by choosing a new hash function h_{new}, if h turns out to be bad for the memory access patterns produced by the programs executed on the PRAM. This way one hopes to guarantee an even distribution of packet destinations with high probability.

3.1 Memory consistency of processor&memory blocks

Each π-processor pm-block has a shared memory module, whose consistency model is the same as the model that we are simulating. Unfortunately, having same consistency model does not automatically[2] guarantee the right consistency model with respect to the whole PRAM simulation. We have to ensure that the write requests will not influence the read requests on the current step. We can think of several approaches to solve the problem. Here, we simply assume that the memory modules execute arriving requests on fly, but preserve temporarily the old memory values (overwritten by a write request) using a special construction to satisfy possibly later arriving read requests. Such a construction could be based on *delayed write*, where each memory cell is duplicated, and the delayed write instructions take effect after a global triggering criteria is fulfilled. Clearly, the synchronization point between separate PRAM steps can be used as the triggering criteria.

3.2 Simulation of EREW PRAM

Since the coated block mesh is a generalization of the coated mesh, one can expect the PRAM simulation methods used in the coated mesh to apply to the coated block mesh as well. The following Algorithm 3.1 is derived from the EREW simulation algorithm presented in [15] for the coated mesh.

Algorithm 3.1 *(EREW simulation on 2-dimensional $\mathcal{CBM}$)*
Each processor $\mathcal{P}_i$ simulates PRAM processors $P_{iN/P+j}$, where $i \in \{0, 1, \ldots, P-1\}$, and $j \in \{0, 1, \ldots, N/P-1\}$.

1. *Build a request packet corresponding to each external reference, and inject the requests uniformly to the routing machinery.*
2. *Route the requests greedily to their target, and on arrival to the target, execute each memory access.*
3. *Synchronize processors.*
4. *Build a reply to each read request, and inject the replies uniformly to the routing machinery.*
5. *Route the replies back greedily.*
6. *Synchronize processors, and update virtual processor registers.*

In [15], we proved that using the above algorithm a 3-dimensional (2-dimensional) P-processor coated mesh, whose nodes can temporarily store $O(1)$ messages, can simulate $P^{3/2}$-processor (P^2-processor) EREW PRAM work-optimally with probability of the form $1-P^{-\alpha \log\log P}$ for any $\alpha > 0$. Because the coated block mesh basically consists of π separate coated meshes (with P/π processors in each), the simulation succeeds with high probability in each of the π coated meshes simultaneously, if $\pi = O(\frac{P}{\pi})$. By "high probability", we mean a probability of the form $1 - P^{\gamma}$ for some constant $\gamma > 1$. Clearly, the coated mesh result can be extended to Theorem 3.2.

Theorem 3.2 *A $\mathcal{CBM}(P, \pi, d)$ can simulate an $N = P \times (\frac{P}{2d\pi})^{\frac{1}{d-1}}$-processor EREW PRAM work-optimally using at most*

$$\begin{cases} 36 + o(1), & if\ d = 2; \\ 78 + o(1), & if\ d = 3 \end{cases}$$

[2] Unless our consistency model is the sequential consistency model [18] (or some weaker model).

routing steps per simulated PRAM processor with high probability, if $\pi = O(\sqrt{P})$.

Besides the theoretical result of Theorem 3.2, we give experimental simulation results in Section 5. Our experiments mainly study the load-simulation cost dependence on a more practical variant of the $\mathcal{CBM}$.

3.3 Simulation of CRCW PRAMs

In [15], we proved that 2-dimensional and 3-dimensional coated meshes can simulate many CRCW PRAM models, especially the Arbitrary model. In order to be work-optimal (with high probability), the simulation requires that $N = \Omega(P^{\frac{d}{d-1}})$. Extending the simulation result to the $\mathcal{CBM}$ is not as straightforward as in the EREW case, since our simulation is based on sorting, and implementing sorting on block mesh is not trivial. However, it is enough to apply sorting within each of the π logical coated meshes and to maintain the memory consistency. We state Theorem 3.3 without proof. Proving it requires similar assumptions as in the EREW case. For details of the CRCW simulation algorithm, see the details the simulation on the coated mesh [15].

Theorem 3.3 *A* $\mathcal{CBM}(P, \pi, d)$ *can simulate an* $N = P \times (\frac{P}{2d\pi})^{\frac{1}{d-1}}$*-processor* Arbitrary *CRCW PRAM work-optimally using at most*

$$\begin{cases} 66 + o(1), & if\ d = 2; \\ 153 + o(1), & if\ d = 3 \end{cases}$$

routing steps per simulated PRAM processor with high probability, if $\pi = O(\sqrt{P})$.

4 Improvements

Next, we briefly discuss two methods to improve the PRAM simulations. Besides the following modifications, one can trade simulation time to hardware, *e.g.*, by applying the delayed load method [2].

4.1 Moving threads

An other kind of improvement is the moving threads approach [12]. The idea is simply to move threads instead of moving read and write requests and replies. Besides providing a method to keep the work load of the processors automatically balanced [12], the moving threads allow us to avoid routing the replies. Assuming that it is not essentially more expensive to move threads than to move packets, the cost of routing drops to half and consequently also the cost of EREW simulation. Unfortunately, in the CRCW case the moving threads approach will not decrease the routing cost to half, because it is necessary to decombine the combined read requests. An interesting observation is that in the ERCW case no decombining is needed, and it is possible to do the combining on the route (in combining queues) and still save in the routing time. If we do not want to save information on the network nodes about combined threads that write to the same shared memory location, we can simply route the "eliminated" thread to any available target.

4.2 Synchronization wave

Separating consecutive steps by synchronization attempts is quite expensive, and we would like to apply a more flexible synchronization method. Such a technique is *synchronization wave.* A synchronization wave is sent by the processors (sources) to the memory modules (destinations) or vice versa. This technique has been successfully used for PRAM simulation on the butterfly [20] and on the mesh of trees [9]. 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, where the actual packets could go. However, when a 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. The synchronization wave may not bypass any actual packets and vice versa. When a synchronization wave sweeps over a leveled network 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 coated block mesh is not directly a leveled network, but it can be seen as one, as described in [13].

When the synchronization wave approach is applied with the moving threads, the task of the processors in the PRAM simulation becomes very simple: The processors process arriving threads freely and the atomic consistency is automatically preserved as long as the synchronization wave reflects between the processors.

5 Experimental results

We run about 400 experiments simulating altogether approximately 20 000 steps of an N-processor EREW PRAM on 2-dimensional and 3-dimensional coated block meshes. In Sections 5.1 and 5.2 we describe the test setting and results of those EREW simulation experiments. More details about experiments and results can be found in [11]. There, we also give ERCW and CRCW PRAM simulation results.

5.1 EREW test setting

We run our simulation experiments on a modified version of the $\mathcal{CBM}$, where all the connections within the routing machinery are duplicated. This allows us to organize all the unidirectional connections between routing machinery nodes so that they form a leveled network. Packets entering an incoming connection x of node y at level l can enter only an outgoing connection z, which is at some level $k > l$. In practice, each routing machinery node has d unidirectional "injection" and d unidirectional "removal" edges (one for each axis), and d bidirectional "normal" edges. A packet is routed by first choosing randomly an intermediate target from the closer half of the pile of nodes axial to the source processor. After arriving to the first intermediate target, the packet advances along the normal edges (greedy XY-routing) and moves to the second randomly chosen intermediate target at the closer half of the pile of nodes axial to the target processor. Finally, the packet is routed to the target by using the removal edges. A more detailed description of the leveled network construction and the routing process is given in [11].

To each unidirectional connection we attach a queue of size q. We assume that only the head of each queue resides on the receiving end, while others reside at the sending end. During a routing step, a routing machinery node can only move packets from the head of incoming queues to the tail of outgoing queues. Moreover, in the 2-dimensional and 3-dimensional cases we introduce d internal queues of size 1 (for packets turning on place) so that each queue can receive packets from at most 4 queues and send them to at most 4 queues. During a routing step, a routing machinery node can move several (up to 4) packets to the tail of an outgoing queue, if there is room for all of them. None of the inserted packets is available at the receiving end during the current step. Naturally, each processor can receive and send at most 1 packet per routing step.

In practice, we expect a randomly chosen hash function to turn typical EREW memory reference patterns to almost random ones. However, being short of typical EREW reference patterns, we did choose the destinations of memory references randomly (using Unix function `random`). The only additional assumption about the computation is that the number of PRAM processors remains the same throughout the computation.

5.2 Cost of continuous moving threads based simulation

Here, we present only results based on the moving threads approach in a continuous simulation situation, where the simulation of consecutive steps is kept separate by a synchronization wave. Other results are given in [11]. As a reference point for measuring the time to simulate a single PRAM step, we use the moment when the last "part" of the synchronization wave leaves the processors. The continuous routing means that processor processes threads (and the synchronization wave) in the order they arrive, and injects them back to the routing machinery as soon as possible. Next, we study the dependence of simulation cost $c(\lambda, \phi, \pi, d, q)$ on the diameter ϕ of the routing machinery, the load $\frac{N}{P} = \lambda\phi$ per processor, π, and q on 2-dimensional and 3-dimensional coated block meshes. The simulation cost is calculated as average simulation time of one EREW step divided by the load.

5.2.1 2-dimensional case

We run our 2D simulation experiments on above described variant of $\mathcal{CBM}(4\pi \times 10, \pi, 2)$ and $\mathcal{CBM}(4\pi \times 50, \pi, 2)$, where the diameter ϕ of routing machinery is 20 and 100, respectively. The influence of queue size q on the simulation cost on 10×10 $\mathcal{CBM}$ is shown in Figure 2. Even, when $q = 2$ the simulation cost decreases below 2. However, with queues of size 16 or 32 one can archive cost $1.1 \ldots 1.2$. Observe that cost 1 is the best one can achieve. The curves of Figure 2 are obtained by using $\pi = 1$. To our surprise, the same curves for $\pi = 4, 8, 16,$ and 32 are identical, or off by cost 0.1. This means that by grouping processors to groups of 32, instead of 1, one can decrease the diameter

ϕ to $\frac{1}{32}$'th and consequently also the required load to achieve certain cost!

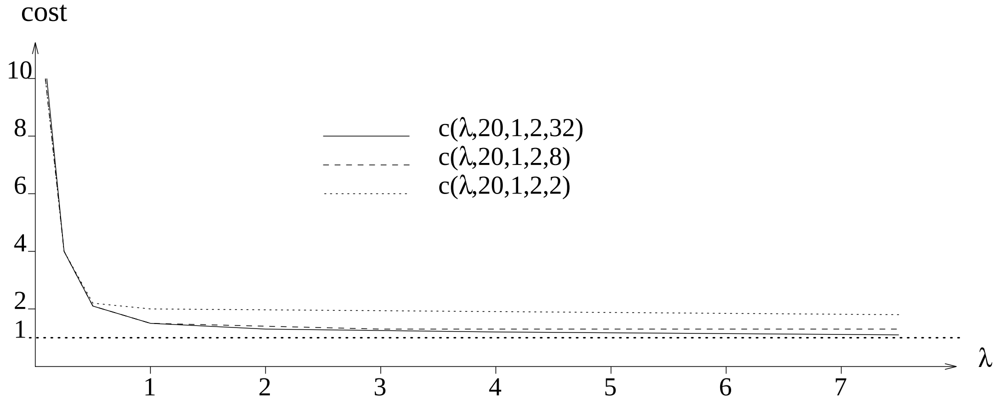

Figure 2: EREW simulation cost on 2-dimensional coated block mesh.

Similarly, the curves for 50×50 $\mathcal{CBM}$ were practically identical for those of Figure 2 (occasionally off by 0.1). Based on our experiments, we conclude that the simulation cost depends strongly on λ and q, but not on ϕ. *Small* π-values also seem to have minor effect on the simulation cost, and thus *the advantage obtained via smaller diameter translated linearly to a smaller multithreading advantage*. Clearly, the cost function begins to show asymptotic behavior, when $\lambda = 1 \ldots 2$.

5.2.2 3-dimensional case

Our results concerning simulation of EREW on 3-dimensional $\mathcal{CBM}$ are very similar to those for 2-dimensional $\mathcal{CBM}$. Again, the simulation cost mainly depends on q and λ. Small π-values and ϕ seem to have practically no effect on $c(\lambda, \phi, \pi, 3, q)$. However, now the advantage provided by π does not imply linearly smaller multithreading level, but only $\approx \sqrt{\pi}$ times smaller multithreading level. The cost function $c(\lambda, 30, 1, 2, 16)$ for $10 \times 10 \times 10$ $\mathcal{CBM}$ is shown in Figure 3. The curve for $30 \times 30 \times 30$ $\mathcal{CBM}$ is almost identical. As can be seen, the cost shows asymptotic behavior, when $\lambda > 1$. The cost approaches to $1.7 \ldots 1.8$, which is clearly worse than in the 2-dimensional case. Increasing q to 32 provides only minor improvement.

We believe that the overall bandwidth of the routing machinery with respect to the number of processors is too small in the 3-dimensional case. Doubling the bandwidth of the routing machinery might be the answer, if the cost $1.8 \ldots 2.0$ is too high.

6 Conclusions

In this paper, we have proposed the coated block mesh as a platform for work-optimal implementation of the PRAM models. The coated block mesh is a generalization of the coated mesh with respect to a parameter: The integration degree π of processing and memory elements. The coated block mesh itself has many desirable properties.

- As a mesh based construction, it has a *regular* and *feasible layout*.

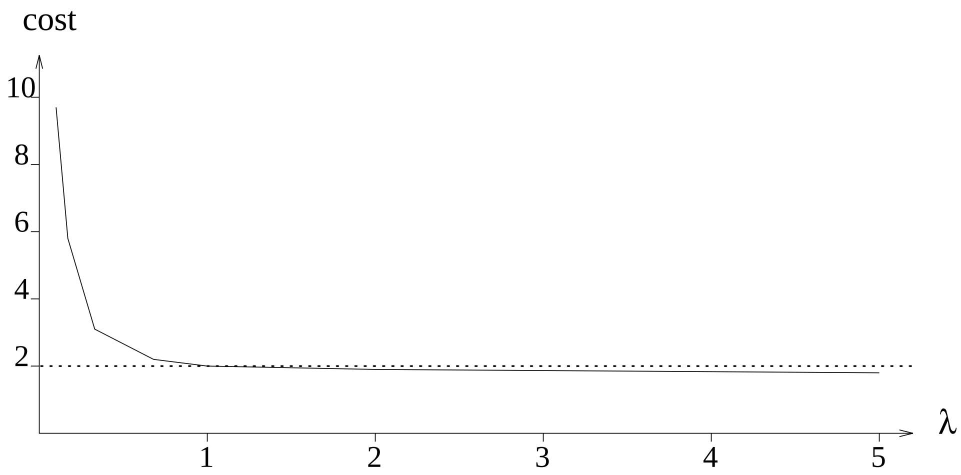

Figure 3: EREW simulation cost in the 3-dimensional case: $c(\lambda, 30, 1, 2, 16)$.

- The whole structure is clearly *modularly extendible*. For example, the hypercube lacks this property due to increasing degree of processing nodes. The lego-like modularity of the coated block mesh also suggest fairly easy physical maintenance.
- The coated block mesh is also *truly scalable* in the sense that x-folding the physical system (x times more processors) provides x times more computational power, if the processors can support approximately $x^{\frac{1}{d-1}}$ times more threads per processor.
- The connections between logical neighbors (processors/nodes) are of *constant length*. For example, in the butterfly the length of connections increases as the system is scaled up. This makes physical extending difficult. Moreover, it can limit the maximum clockrate of the system.

When claiming our PRAM implementations work-optimal, we ignore the work and hardware complexity of the router blocks. If $d = 3$, then the ratio of router and processor&memory blocks is $\frac{\sqrt{P}}{6^{1.5} \cdot \sqrt{\pi}}$. If $\pi = 4$ and $P = 86400$, then this ratio is only 10. On the other hand, the processors, and especially the memory modules, are clearly more expensive in terms of hardware than the routing machinery nodes.

6.1 Advantage of coated block mesh over coated mesh

The coated block mesh is an application of more general method to design efficient systems. It is a two-level system, which applies a theoretically elegant and scalable but rather inefficient solution on top of a non-scalable but efficient small scale solution. The blocks (and π-port memories) represent non-scalable but efficient solutions, and the mesh structure, of course, represents the elegant part. The parameter π clearly provides a method to balance the $\mathcal{CBM}$ and the cost of its elementary operations (memory access, thread management primitives, communication with neighboring elements) with respect to the available technology.

Our experiments on a variant of the $\mathcal{CBM}$ show that by using the moving threads approach, the simulation cost can be pushed below 2 in the 3-dimensional case, and very close to 1 in the 2-dimensional case. Observe that these results are based on the assumption that on each step each thread makes an external reference. By processing threads that make local references during the "idle" cycles, one can hope to further improve the performance of the processors.

The larger the π-value of a coated block mesh the smaller the diameter, and the smaller the required multithreading level to achieve certain simulation cost. Experiments indicate that load $\approx \phi$ is enough to achieve good performance in the 2-dimensional and 3-dimensional cases. To our surprise, even relatively large π-values (32) seem to have no influence on $c(\lambda, \phi, \pi, d, q)$. Thus, the smaller diameter advantage of the $\mathcal{CBM}$ over the $\mathcal{CM}$ translates directly to smaller multithreading requirement advantage (proportional to $\pi^{\frac{1}{d-1}}$)!

6.2 Topics for further research

It is quite obvious that in small scale (*e.g.*, within a chip) one can apply different (brute force type) solutions than in large scale. The two-level approach itself seems healthy, but can one apply it more efficiently in an other context – *i.e.*, would it be better to use other large scale (*e.g.*, the butterfly) and small scale methods (*e.g.*, the crossbar)?

One could also study several variants of our coated block mesh. For example, instead of one π-degree shared memory module, we could have ρ π-degree or π/ρ-degree shared memory modules per processor&memory block. At the same time we could extend the degree of processors from 1 to ρ. This way, the delivery of packets would become harder but the constructibility of memory modules would perhaps improve.

If the router blocks are very big, we could consider locating τ processor&memory blocks on the surface instead of just one. This could give us a third parameter τ, which might help us in finding a construction, where the utilization of the three resources (processors, memory and communication capacity) is in balance. Supporting $\tau = \rho$ processor&memory blocks would be rather straightforward from the routing point of view.

As an opposite to the shared memory module approach, we could simply group π processor&memory pairs to a loosely coupled block. This would make the routing process harder, since the number of targets would increase. Consequently, it suggests that one should use a crossbar style solution within the router blocks.

References

[1] F. Abolhassan, R. Drefenstedt, J. Keller, W.J. Paul, and D. Scheerer. On the Physical Design of PRAMs. In J. Buchmann, H. Ganzinger, and W.J. Paul, editors, *Informatik – Festschrift zum 60. Geburtstag von Guenter Hotz*, pages 1 – 19. Teubner Verlag, 1992.

[2] 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.

[3] 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.

[4] 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.

[5] M.J. Forsell. Are Multiport Memories Physically Feasible? *Computer Architecture News*, 22(4):47 – 54, September 1994.

[6] 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.

[7] 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.

[8] T.J. Harris. A Survey of PRAM Simulation Techniques. *ACM Computing Surveys*, 26(2):187 – 206, June 1994.

[9] V. Leppänen. On Implementing EREW Work-Optimally on Mesh of Trees. Technical Report R-94-9, University of Turku, Computer Science Department, 1994. Submitted for publication.

[10] V. Leppänen. Performance of Four Work-Optimal PRAM Simulation Algorithms on Coated Meshes, 1994. To appear in The Computer Journal.

[11] V. Leppänen. Experimental PRAM on Coated Block Mesh Simulation Results, 1995. In preparation.

[12] V. Leppänen. Moving Threads: Automatic Load Balancing, 1995. In preparation.

[13] V. Leppänen. Moving Threads based Parallel Computation, 1995. In preparation.

[14] 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.

[15] V. Leppänen and M. Penttonen. Work-Optimal Simulation of PRAM Models on Meshes. Technical Report R-94-1, University of Turku, Computer Science Department, January 1994. To appear in Nordic Journal on Computing.

[16] F. Luccio, A. Pietracaprina, and G. Pucci. A Probabilistic Simulation of PRAMs on a Bounded Degree Networks. *Information Processing Letters*, 28:141–147, 1988.

[17] 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.

[18] D. Mosberger. Memory Consistency Models. *Operating Systems Review*, 17(1):18 – 26, January 1993.

[19] M. Palis, S. Rajasekaran, and D.S.L. Wei. Emulation of a PRAM on Leveled Networks. In *International Conference on Parallel Processing, Part I*, pages 418 – 421, 1991.

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

[21] 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.

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