# Efficient Two-Level Mesh based Simulation of PRAMs

Martti Forsell
Dept. of Computer Science
University of Joensuu
FINLAND

Ville Leppänen
Dept. of Computer Science
University of Turku
FINLAND

Martti Penttonen
Dept. of Computer Science
University of Joensuu
FINLAND

## Abstract

*We consider time-processor optimal simulations of PRAM models on coated block meshes. A coated block mesh consists of $\pi$-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 time-processor optimally with moderate simulation cost. Using 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 $\pi$ provides a method to balance the construction with respect to the two techniques.*

**Keywords**: PRAM, shared memory machine, simulation, time-processor optimal, mesh, interconnection network.

# 1 Introduction

The standard approach to PRAM (defined in Section 2.1) implementation is to map the shared memory into a collection of $P$ 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. 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.

Diameter $\phi$ of a routing machinery usually determines a lower bound for the latency of messages. For many popular networks, $\phi = \Omega(\log P)$. If $N = O(P)$, the latency often determines the PRAM simulation cost. *Parallel slackness* [14] ($N > P$) can be used to balance the cost of local computation and global communication, and thus to improve the efficiency of PRAM simulations. In Valiant's XPRAM framework [14], 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 virtual processors. 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 *time-processor optimal*, if this is achieved.

## 1.1 Previous results

Many time-processor optimal PRAM simulation results have been proved for completely connected graphs ((i)-type) [4, 6, 12, 14]. In [6], Goldberg, Matias, and Rao provide so far the best (asymptotically smallest latency) time-processor optimal $P \log\log P$-processor EREW PRAM simulation.

If the $\phi$ 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 ((ii)-type), or that the routing machinery is separated from the processor&memory pairs and is considerably larger than $P$ ((iii)-type). 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 [14], 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 [13], or by requiring that $N = P^{1+\epsilon}$ for some $\epsilon > 0$ [14]. In [11], we prove (iii)-type results for EREW and CRCW PRAMs by taking the routing machinery to be a 2D mesh or a 3D mesh. Both for EREW and CRCW simulations, time-processor optimality is achieved, when $N = \Omega(P^{\frac{d}{d-1}})$. General leveled network results [7] imply that (iii)-type results can also be shown for several other networks.

### 1.2 Some critique

Because of routing machinery degree, the completely connected graphs are not very realistic with current technology. Also Valiant's hypercube result is based on some quite unrealistic assumptions. The degree $\log P$ makes it difficult to expand 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 unit time.

Type (iii) results also have a price to pay: The routing machinery is large in terms of the number of nodes. However, the hardware complexity ratio of a processor&memory pair and a routing machinery node might be significant, e.g., of one or two orders of magnitude. Due to non-constant connection length, the unit time assumption and scalability are problematic in the butterfly [1, 13] but not in our mesh based solution. In the 3D case, the routing machinery size is only $\sqrt{P}/6^{1.5}$ times $P$ [11]. If $P = 10^4$, this ratio is $\approx 7$. For the butterfly, the same ratio $\log P + 1 \approx 15$.

The proposed simulations differ also with respect to the required parallel slackness. To achieve constant simulation cost per PRAM processor, slackness level $\Omega(\phi)$ is required. The cost decreases as the slackness level increases. Supporting large number of threads per each processor is possible (Tera [2]: 128 threads per processor; SB-PRAM: $32 \times 32$ threads), but the thread requirement sets restrictions for successful programming, since achieving a processor utilization level requires certain amount of threads per processor. However, the actual "cost" of some slackness level is unclear.

### 1.3 Contributions

We propose an improvement to the coated mesh [11] in form of a coated block mesh. We group a bunch of routing machinery nodes to a *router block* by using a sparse matrix type solution within each router block. Consequently, the $\phi$ and slackness requirement decrease. Also the ratio of the number of elementary routing machinery nodes and processors improves by factor of $\pi^{\frac{1}{d-1}}$ in the $d$-dimensional case. The processors we divide to groups of $\pi$ processors, and integrate one shared memory module to each group. Each *processor&memory block* is considered as a small PRAM.

The idea in 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. Brute force solutions can be very efficient, if 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. (Similar design principle is recently applied in constructing practical expanders [3].) By properly choosing $\pi$, we hope to find solutions, where the cost of communication, memory access, and thread management primitives are properly balanced.

In Section 3, we show that the coated block mesh can efficiently simulate PRAM model. In asymptotic sense the same results can be derived from certain leveled network results [7]. However, the theme of this paper is to characterize the cost more precisely, and to observe that in practice it is very small. Our EREW simulation experiments indicate that the cost can be decreased below 2 in the 3D case, and to $1.1 \ldots 1.3$ in the 2D case. The latter means that 75% – 90% of the raw processing power can be given to arbitrary PRAM computations. In our results we assume that each PRAM processor makes an external memory reference at each step. This is hardly the case in practice, and thus achieving smaller simulation cost is possible.

## 2 Definitions

### 2.1 PRAM

The PRAM model consists of $N$ processors, $P_0, P_1, \ldots, P_{N-1}$, and a shared memory $M$ of size $m$. Each $P_i$ has some local memory and registers. A step of $P_i$ consists of a local operation, reading a shared memory location, *or* writing to a shared memory location. The phases of a step as well as steps are executed synchronously (system wide). A read instruction returns the value of memory location in question *before* the current step. In the EREW model concurrent reading and writing of any particular shared memory cell is forbidden, but each shared memory cell may be read and written once during a step. In the ARBITRARY CRCW model concurrent read and write operations are allowed. If two or more processors write to a memory location in a given step, one of the values is selected arbitrarily to become the new value.

## 2.2 Coated block mesh

Our *coated block mesh* (CBM) is a generalization of the coated mesh (CM) [11], which consists of a 2D mesh or a 3D mesh based routing machinery that is coated with processor&memory pairs (see Figure 1). In the CBM, $\pi$ processors and their memory modules are integrated to a processor&memory block (pm-block). Respectively, a bunch 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.

Formally, a $d$D coated block mesh $CBM(P, \pi, d, q)$ consists of $P/\pi$ pm-blocks $A_0, A_1, \ldots A_{P/\pi-1}$ of $\pi$ processors and one shared memory module, and $(P/(\pi 2d))^{d/(d-1)}$ router blocks $B_{i_1,i_2,\ldots,i_d}$, where $0 \leq i_j < (P/(\pi 2d))^{1/(d-1)}$ and $1 \leq j \leq d$. Each router block has $\pi$ 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 as a regular $d$D mesh.

### 2.2.1 Router block

We assume that the internal structure of a router block is a $d$-dimensional $\sqrt[d-1]{\pi}$-sided sparse matrix with $\pi$ router nodes. Each directed link has a buffer of size $q$ packets. A packet entering $B_{i_1,i_2,\ldots,i_d}$ via link $k$ on surface $j$, can leave the block only via link $k$ on some surface $l$. The function of router nodes is defined by the PRAM simulation algorithm.

Assume that the nodes of a router block 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 the 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 $f_j$ : $\{1, \ldots, \sqrt[d-1]{\pi}\}^{d-1} \mapsto \{0, \ldots, \pi-1\}$. For constructibility we want the connections within router blocks to be axial. 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 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. Consequently, a projective mapping of $g^{-1}$ with respect to any surface covers the whole surface. By defining $g$ and one $f_j$, we at the same time define rest of the $f_j$'s. Clearly, the definition of $f_0$ is irrelevant as long as it is bijective. We use $g_2^{-1}(k) = \{k, k\}$ in the 2D case, and $g_3^{-1}(\sqrt{\pi} \times k + l) = \{k+1, l+1, ((k+l) \bmod \sqrt{\pi}) + 1\}$ in the 3D case $(0 \leq k, l < \sqrt{\pi})$.

### 2.2.2 Processor&memory block

The pm-blocks "coat" the whole surface of router blocks based routing machinery. The degree of each processor (pm-block) is 1 ($\pi$). We assume that each processor can receive and send a packet in one time unit. For simplicity, we take the memory consistency model of the shared memory modules to be the same as the whole CBM is simulating. We do not define, how the pm-blocks are implemented, but assume that fulfilling any proper memory request takes unit time. If $\pi$ is a small constant, possible implementations are, e.g., a $\pi$-port memory [5], and a complete graph based $\pi$-processor DMM.

# 3 Simulation of PRAM models

A somewhat standard approach [13, 14] to simulate the PRAM models is to assume that the shared memory of the PRAM is mapped with a randomly chosen hash function $h \in \mathcal{H}$ into the distributed memory modules; in our case, into the local memory of pm-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 $h_{new} \in \mathcal{H}$, if $h$ turns out to be bad for the memory access patterns produced by the programs executed on the PRAM.

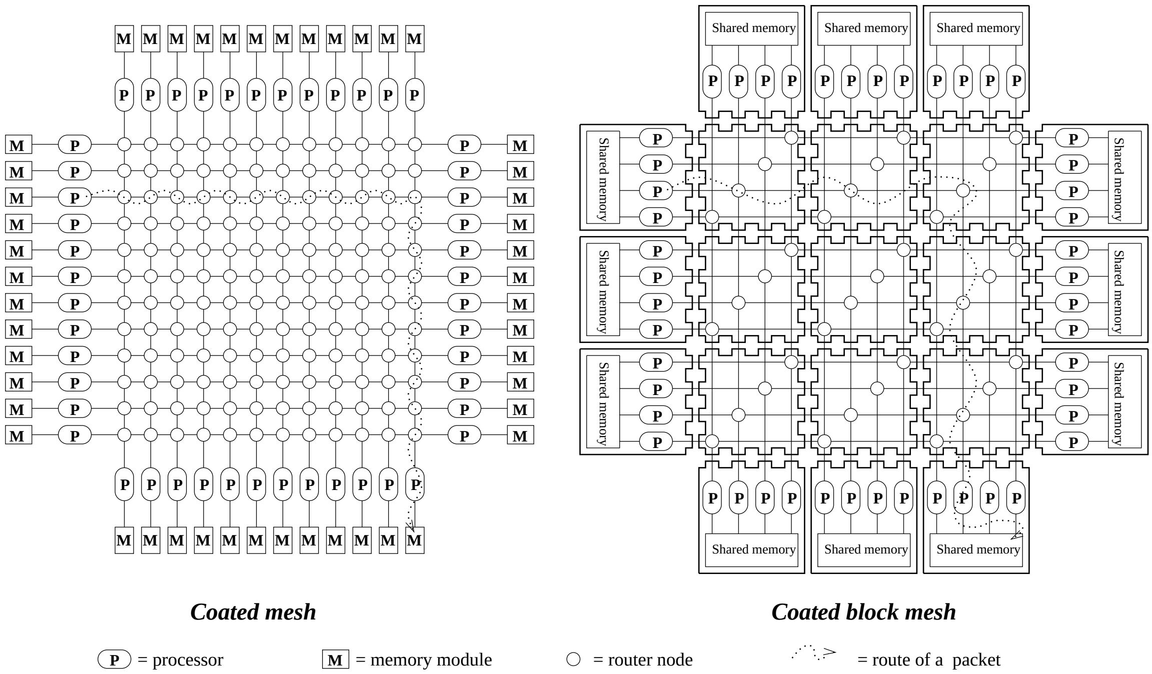


Figure 1: Coated mesh versus coated block mesh ($\pi = 4$) in 2-dimensional case.

Although each pm-block has the same memory consistency model as the simulated PRAM model, we still must 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 assume that the arriving requests are executed on fly, but simultaneously old memory values are temporarily preserved. Such a construction could be based on delayed write, where each memory cell is duplicated, and the delayed write instructions take effect after the synchronization point between separate PRAM steps is reached.

## 3.1 Simulation of EREW PRAM

The following algorithm is derived from the EREW simulation algorithm presented in [11] for the CM.

**Algorithm 3.1** *(EREW simulation on 2D CBM)* $\mathcal{P}_i$ *simulates PRAM processors* $P_{i\frac{N}{P}}, \ldots, P_{(i+1)\frac{N}{P}-1}$.

1. *Translate memory references to packets, and inject them uniformly to the routing machinery.*
2. *Route the requests greedily to their target, and execute the memory accesses.*
3. *Synchronize processors.*
4. *Build replies, and inject them uniformly to the routing machinery.*
5. *Route the replies back greedily.*
6. *Synchronize processors, and update virtual processor registers.*

In [11] we proved that using the above algorithm and polynomial hash functions, a 3D (2D) $P$-processor CM can simulate $P^{3/2}$-processor ($P^2$-processor) EREW PRAM time-processor optimally with probability $1 - P^{-\alpha \log\log P}$ for any $\alpha > 1$. Because the coated block mesh basically consists of $\pi$ separate coated meshes (with $P/\pi$ processors in each), the simulation succeeds with high probability in each of them 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* $CBM(P, \pi, d, q)$ *can simulate an* $N = P \times (\frac{P}{2d\pi})^{\frac{1}{d-1}}$*-processor EREW PRAM time-processor optimally using at most*

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

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

## 3.2 Simulation of CRCW PRAMs

In [11] we proved that 2D and 3D coated meshes can simulate the ARBITRARY CRCW time-processor optimally (with high probability), if $N = \Omega(P^{\frac{d}{d-1}})$. Extending the simulation result to the CBM is not as straightforward as in the EREW case, since CRCW simulation is based on sorting, and implementing sorting on the CBM is not trivial. However, it is enough to apply sorting within each of the $\pi$ logical coated meshes separately 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 [11].

**Theorem 3.3** *A $CBM(P, \pi, d, q)$ can simulate an $N = P \times (\frac{P}{2d\pi})^{\frac{1}{d-1}}$-processor* ARBITRARY *CRCW PRAM time-processor 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

A straightforward method to improve simulation cost is to increase the parallel slackness factor of simulation, i.e., the load $\frac{N}{P}$ of each physical processor. Next, we briefly discuss another method to improve the simulation cost.

Separating consecutive steps by naive synchronization attempts is 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) and vice versa. This technique has been successfully used for PRAM simulation on the butterfly [13] 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. 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 [7] – or more generally a directed acyclic graph (DAG), all nodes and processors receive exactly one synchronization packet via each input link and send exactly one via each output link.

The CBM is not directly a leveled network, but it can be seen as one, or more precisely, each CM can be seen as a virtual leveled network. Embedding a leveled network to a mesh is described in [7]. Their technique is improved in [10], and applied to coated meshes by embedding a DAG to a CM with duplicated connections. The idea is that 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. The above routing process in a natural way organizes all unidirectional connections of a CM and a CBM (with duplicated connections) to a DAG.

Continuous synchronization wave based EREW simulation can now be derived by setting one DAG from processors to memory modules (requests) and another from memory modules to processors (replies), and letting the synchronization wave reflect from one DAG to the other (see [10]). We denote such a coated block mesh by $CBM^+$.

# 5 Experimental results

We run about 400 experiments simulating altogether approximately 20 000 steps of an $N$-processor EREW PRAM on 2D and 3D $CBM^+$s. We assume that each unidirectional connection has a buffer of size $q$ packets, only the head of each queue resides at the receiving end, and a routing machinery node can only move packets from the heads of incoming queues to the tails of outgoing queues (if there is room). 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. More details about experiments and results can be found in [8].

### 5.1 Continuous EREW simulation

As a reference point for measuring the time to simulate a single PRAM step, we use the moment when the last "part" of a synchronization wave leaves the processors. The continuous simulation means that each processor processes the requests and replies (and the synchronization wave) in the order they arrive, and injects replies and new requests 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 $\phi$ of the routing machinery, the load $\frac{N}{P} = \lambda\phi$ per processor, $\pi$, and $q$ on $d$D CBM$^+$s. The simulation cost is calculated as an average time over 30 – 50 simulated EREW steps divided by the load.

#### 5.1.1 2-dimensional case

We run our 2D simulation experiments on $CBM^+(4\pi \times 10, \pi, 2, q)$ and $CBM^+(4\pi \times 50, \pi, 2, q)$, where the diameter $\phi$ is 40 and 200, respectively. The influence of buffer size $q$ on the simulation cost on $10 \times 10$ CBM$^+$ is shown in Figure 2. Even, when $q = 2$ the simulation cost decreases below 2. However, with buffers of size 16 or 32 one can achieve cost 1.1...1.2. The 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. Thus by applying $\pi = 32$ (instead of 1), one can decrease the diameter $\phi$ to $\frac{1}{32}$'th and consequently also the required load to achieve certain cost!

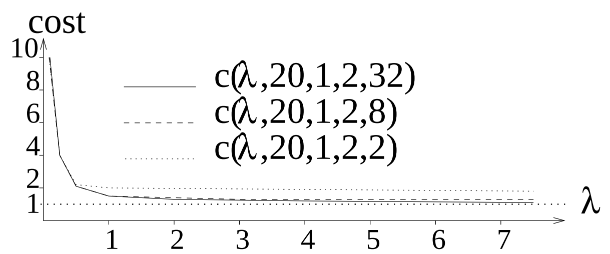


Figure 2: EREW simulation cost on 2D CBM$^+$.

The curves for $50 \times 50$ CBM$^+$ were practically identical for those of Figure 2 (occasionally off by 0.1). We conclude that the simulation cost depends strongly on $\lambda$ and $q$, but not on $\phi$. *Small* $\pi$-values seem to have minor effect on the simulation cost, and thus the advantage obtained via smaller diameter seems to translate linearly to smaller parallel slackness advantage. Clearly, the cost function begins to show asymptotic behavior, when $\lambda = 1 \ldots 2$.

#### 5.1.2 3-dimensional case

Our results concerning simulation of EREW on 3D CBM$^+$ are very similar to those for 2D CBM$^+$ (see [8] for details). Again, the simulation cost mainly depends on $q$ and $\lambda$. Small $\pi$-values and $\phi$ seem to have practically no effect on $c(\lambda, \phi, \pi, 3, q)$. Now the advantage provided by $\pi$ does not imply linearly smaller parallel slackness level, but only $\approx \sqrt{\pi}$ times smaller level. The curves show asymptotic behavior when $\lambda > 1$, and at best cost 1.68 was achieved (with $\lambda = 5$, $q = 32$). This is clearly worse than in the 2D case. The reason is that in the 3D CBM$^+$, the routing machinery bandwidth is simply too small [10] for all PRAM processors to make a shared memory reference at every step. Insufficient bandwidth makes the cost approach $1\frac{1}{2}$ in the 3D case.

## 6 Conclusions

We have proposed the coated block mesh as a platform for time-processor optimal implementation of PRAM models. The CBM is a generalization of the CM with respect to the integration degree $\pi$ 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.*
- The whole structure is *modularly extendible.* E.g., the hypercube lacks this property. The modularity also suggest fairly easy physical maintenance.
- The CBM is *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 small *constant length.*

When claiming our PRAM implementations time-processor 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 apparently more expensive in terms of hardware than the routing machinery nodes.

The CBM is an application of more general design method: It applies a theoretically elegant and scalable but rather inefficient solution on top of a non-scalable but efficient small scale solution. The blocks represent non-scalable but efficient solutions whereas the mesh structure represents the elegant part. The parameter $\pi$ provides a method to balance the cost of elementary operations: memory access, thread management, and communication with neighboring elements.

Our experiments show that by using the synchronization wave+DAG approach, the cost can be pushed below 2 in the 3D case, and very close to 1 in the 2D case. These results are based on the assumption that on each step each PRAM processor makes an external reference on every step! By processing threads that make local references during the "idle" cycles, further performance improvements can be expected.

The larger the $\pi$-value the smaller the diameter and the smaller the required parallel slackness level to achieve a certain simulation cost. Experiments indicate that load $\approx \phi$ is enough to achieve good performance in the 2D and 3D cases. To our surprise, even relatively large $\pi$-values (32) seem to have no influence on $c(\lambda, \phi, \pi, d, q)$. Thus, the smaller diameter advantage of the CBM over the CM translates directly to smaller parallel slackness requirement advantage.

### 6.1 Topics for further research

The two-level approach itself seems healthy, but can one apply it more efficiently in another context – i.e., would it be better to use other large scale and small scale methods. One could also study several variants of our coated block mesh. Instead of one $\pi$-degree memory module, we could have $\rho$ $\pi$-degree or $\pi/\rho$-degree memory modules per pm-block. We could also extend the degree of processors from 1 to $\rho$. That would make the delivery of packets harder but perhaps improve the constructibility of memory modules. If the router blocks are very big, we could consider locating $\tau$ pm-blocks on the surface of each instead of one. Supporting $\tau = \rho$ processor&memory blocks would be rather straightforward from the routing point of view.

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

[3] E.A. Brewer, F.T. Chong, and F.T. Leighton. Scalable Expanders: Exploiting Hierarchical Random Wiring. In *Proceedings, 26th Symposium on Theory of Computing*, 144 – 152, 1994.

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

[5] M.J. Forsell. Are Multiport Memories Physically Feasible? *Computer Architecture News*, 22(5):3 – 10, December 1994.

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

[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. Experimental PRAM on Coated Block Mesh Simulation Results, 1996. In preparation.

[9] V. Leppänen. On Implementing EREW Time-Processor Optimally on Mesh of Trees. *Journal of Universal Computer Science*, 1(1):23 – 34, 1995.

[10] V. Leppänen. Performance of Work-Optimal PRAM Simulation Algorithms on Coated Meshes, 1996. To appear in The Computer Journal.

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

[12] F. Meyer auf def Heide. Hashing Strategies for Simulating Shared Memory on Distributed Memory Machines. In *Parallel Architectures and Their Efficient Use, LNCS 678*, 20 – 29, 1992.

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

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