Implementing the Hierarchical PRAM on the 2D Mesh: Analyses and Experiments*

George Chochia, Murray Cole and Todd Heywood

Department of Computer Science

The University of Edinburgh

King's Buildings, Edinburgh EH9 3JZ

Scotland

email: {gac,mic,thh}@dcs.ed.ac.uk

Abstract

We investigate aspects of the performance of the EREW instance of the Hierarchical PRAM (H-PRAM) model, a recursively partitionable PRAM, on the 2D mesh architecture via analysis and simulation experiments. Since one of the ideas behind the H-PRAM is to systematically exploit locality in order to negate the need for expensive communication hardware and thus promote cost-effective scalability, our design decisions are based on minimizing implementation costs. The Peano indexing scheme is used as a simple and natural means of allowing the dynamic, recursive partitioning of the mesh into arbitrarily-sized sub-meshes, as required by the H-PRAM. We show that for any sub-mesh the ratio of the largest manhattan distance between two nodes of the sub-mesh to that of the square mesh with an identical number of processors is at most 3/2, thereby demonstrating the locality preserving properties of the Peano scheme for arbitrary partitions of the mesh. We provide matching analytical and experimental evidence that the routing required to efficiently implement the H-PRAM with this scheme can be implemented cheaply and effectively.

*Work supported by EPSRC grant GR/J43295

1 Introduction

A model of parallel computation has a difficult task in that it must mediate between the conflicting requirements of *simplicity* of use for program design/analysis, and *reflectivity* of cost/resource details of realistic architectures. In other words, it must abstract away the implementation details inherent in architectural views, while truly reflecting the essential costs of parallel computation on those architectures [15]. The goal of a model is to bridge the gap between practice and theory; pragmatically, this requires its joint acceptance by both the systems and theory communities.

The PRAM model is accepted as a good tool for parallel algorithm research, but fails to even abstractly represent realistic architectures since it ignores communication and synchronization costs. In recognition of this fact, many other models have been proposed (see [5], and the references therein), none of which have yet caught on. In various ways they try to balance simplicity of use and reflectivity of architectural costs, and may be *roughly* classified into three types:

- PRAMs with locality (simplicity = shared memory, reflectivity = locality), e.g. H-PRAM, YPRAM, BDM, LPRAM, BPRAM.
- Global message passing models (simplicity = global address space, reflectivity = message passing), e.g. BSP, LogP, and many more.
- "Adjusted" PRAMs with "architectural" features, such as asynchrony (many variants of Asynchronous PRAMs) or memory queues to represent contention (QRQW PRAM).

Comparisons and critical analyses of various models may be found in [5, 6, 7]. With the exception of the LogP model, and to a small extent the BSP, results on these models have been theoretical. They have not received much attention on the part of the systems community.

This paper covers initial results of an investigation, via simulation experiments, of implementation details and performance of the Hierarchical PRAM (H-PRAM) model on a 2D mesh architecture. The H-PRAM, introduced in [7, 8], is the most general of the "PRAM with locality" type models (i.e. has the least restrictions on partitioning the memory into sub-memories). The H-PRAM is a PRAM whose instruction set is *extended* by a partition instruction, which allows the hierarchical organization of processors and memory into groups. Each group works as a separate synchronous "sub-PRAM", asynchronously and fully independently from the others. The model accounts for communication and synchronization costs, which are functions of the group size. An architecture implementing the H-PRAM must be able to hierarchically partition itself into any number of independent sub-architectures, where communication and synchronization costs are functions of sub-architecture size. The H-PRAM is reflective of the costs and resources of distributed memory architectures with recursively partitionable networks, thus H-PRAM programs would be architecture independent across this class of architectures.

In this paper, we are concerned with the practical, systems aspects of the implementation and performance of P-processor H-PRAM on a $\mathcal{P}$-processor mesh with $\mathcal{P} = P$. Our goal, towards which this paper is an initial contribution, is to determine the properties of *cost-effective* (cheapest possible) systems based on *scalable* architectures which can *efficiently* support the H-PRAM model. The thesis is that through the systematic exploitation of locality, represented abstractly at the programming level for ease of use, we can get high performance out of a scalable architecture without the need for exotic hardware. In other words, the use of locality should allow us to do more in systems software, reducing the need for expensive communication and synchronization hardware which inhibit cost-effective scalability. This philosophy should be contrasted with that implicit in the global message passing models above; creating a scalable, global address space requires significant investments in router technology at the very least.

The 2D mesh is one obvious choice of network if the criteria is cost-effective scalability. In terms of today's technology, it is relatively simple to construct, and ideal for VLSI implementation. It also fares well when considering future technology. Bilardi and Preparata [1] have shown that, under speed of light and device area/volume limitations, the mesh is the *only* scalable parallel architecture. Long term considerations are rather underappreciated in today's parallel computing research, but are very important with respect to proposed models of computation. Having a model of computation established and then made obsolete by technological advances is worse than not having an established model in the first place.

An argument against the the mesh is that it has a large diameter compared to many other networks. However, the H-PRAM allows us to substantially reduce or even eliminate the routing overhead of the underlying network at the price of memory reorganization. It is well known that there exists a large body of parallel algorithms where each processor, during a sufficiently long period of time, accesses data within a restricted domain. The H-PRAM captures this locality in the data access pattern via a partition instruction, and charges communication cost (for the purpose of performance prediction) as a function of local neighborhood size.

An argument against the use of communication metrics which are functions of the distances messages have to travel in a network is that in the current generation of parallel architectures the "start up" cost of getting messages from a processor onto the network dominates the network travel time. We believe this is a technological artifact, resulting from the concentration on routing network technology in development work between the previous and current generations, at the expense of the interface between the network and the processor nodes. This imbalance will be probably be addressed by the next generation; see [3] for example. Note also that under speed of light and device area limitations (see above) communication delay is solely a function of distance.

The paper is organized as follows. In Section 2 the H-PRAM model and some implementation details are briefly overviewed, setting the stage for the presentation of analysis and experiments. Section 3 concerns the mapping of the H-PRAM to the Peano-indexed mesh, giving the analytical results necessary for interpreting the experimental results, as well as being informative in their own right. Section 4 gives details of implementing routing on the Peano mesh, and the set-up of the resulting experiments. Finally, Section 5 presents the results of the experiments on the performance of a randomly partitioned H-PRAM when implemented on the 2D mesh. The simulation results are compared against the analytical predictions obtained earlier. Section 6 makes brief conclusions and outlines future work.

2 H-PRAM implementation

Structurally, the H-PRAM [7] is a PRAM which can recursively partition itself into sub-PRAMs, giving rise to a hierarchy of sub-PRAMs. A P-processor H-PRAM is a P-processor PRAM extended by a partition instruction:

partition { p_1: Algorithm-1 (parameter-list);
p_2: Algorithm-2 (parameter-list);
...
p_Q: Algorithm-Q (parameter-list) }

where the p_i are such that $\sum_{i=1}^{Q} p_i = P$.

The operation partitions the P processors of the original PRAM into disjoint subsets of p_i processors running the specified algorithms. Partitioning of processors causes the memory to be partitioned proportionately; thus "sub-PRAMs" are disjoint and cannot access any memory besides their own until the partition step terminates (this was called the "private H-PRAM" variant in [7]). The sub-PRAMs operate asynchronously from each other and synchronize at the termination of the partition instruction. Independently executing sub-PRAMs do not interact, i.e. all communication is restricted to occur within each sub-PRAM.

The sub-PRAM sub-algorithms may themselves have partition instructions, giving rise to a recursive partitioning of the H-PRAM. The hierarchical structure of the overall computation can be represented by a dynamically expanding and contracting tree, where nodes spawning children correspond to the starts of partition instructions and disappearing (leaf) nodes correspond to terminations of partitions and reactivation of the parent node.

Partitioning is *dynamic*, i.e. the parameters Q and p_i in the partition instruction need not be known in advance. While the algorithms in [8] are static, dynamic partitioning is more general, as it allows hierarchies to depend on data values.

Since an H-PRAM hierarchy will grow and contract according to data dependencies, the architecture must be able to partition into any number of arbitrarily-sized sub-architectures so that sub-PRAMs map to sub-architectures, where costs are functions of sub-architecture size. The evaluation of algorithm/program complexity for the H-PRAM is different than for the (unit cost) PRAM in that communication and synchronization operations are charged with their realistic cost as defined by the underlying architecture. See [7, 8] for complexity analysis details. Space does not allow a recapitulation here, but we can summarize this by saying that, under a set of very reasonable assumptions, H-PRAM algorithm analysis is like PRAM algorithm analysis except that a communication cost is charged to communication steps (synchronization cost is subsumed).

Each sub-PRAM in the hierarchy can be identified with respect to its level in a tree, and the order within the level. An arbitrary sub-PRAM at level i can be represented by a pair $\langle P_i, M_i\rangle$, where P_i is an ordered *set* of processors specified by indices, M_i an ordered *set* of memory locations, $i \geq 0$. $|M_0|$ is a multiple of $|P_0|$. $P_i \subseteq P_{i-1}$, $i \geq 1$ (i.e. some sub-PRAM at the next level up) such that the ordered set of processor indices $\{p|\ p \in P_i\}$ is a consecutive interval. There are no intersections in the processors belonging to different sub-PRAMs at the same level (private H-PRAM). $M_i \subseteq M_{i-1}$ is a subset of the shared memory locations available to P_i. Let L_p denote a set of memory locations with indices

$$\frac{|M_0|}{|P_0|}p, \ldots, \frac{|M_0|}{|P_0|}(p+1)-1$$

Then $M_i = \bigcup_{p \in P_i} L_p$.

Now consider the implementation of the H-PRAM on a distributed memory architecture whose network topology is defined as a graph G with diameter $d(G)$. Each node of the graph represents a pair: a processor and its local memory, and simulates some $p \in P_0$ of the H-PRAM. We will say that a topology is suitable for the H-PRAM implementation if there exists an indexing scheme I_P enumerating the nodes, such that

$$d(G') \leq c \cdot d(G'') \ , \tag{1}$$

where G' and G'' are any two disjoint subgraphs of G onto which the sub-PRAMs with $\mathcal{P}'$ and $\mathcal{P}''$, $\mathcal{P}' < \mathcal{P}''$, processors are mapped, where c is a constant.

Let G be a 2D mesh, $\mathcal{P}_0$ a set of processors of the mesh and $\mathcal{L}_p$, $p \in \mathcal{P}_0$ the local memory available to that processor. The simulated shared memory M_0 is a union of the local memories. The simulated processors P_0 and associated memory space L_p, $p \in P_0$ are mapped bijectively to the physical processors $\mathcal{P}_0$ and physical memory space $\mathcal{L}_p$, $p \in \mathcal{P}_0$ correspondingly. Each node of the mesh $p \in \mathcal{P}$ is assigned an index by means of the Peano indexing scheme. It will be shown below that Peano indexed mesh simulating the H-PRAM satisfies (1) with $c = \frac{3}{2-2/\sqrt{\mathcal{P}''}}$.

Before turning to the details of the mapping of the H-PRAM to the Peano mesh, however, we briefly need to cover the implementational issues of memory organization. The logical address space of each sub-PRAM is randomized onto the physical address space of the mesh by means of a pseudo-random hash function. Let v be the amount of memory actually used by the the algorithm in a sub-PRAM and let r be the space required to distribute this data in the physical memory using the hash function of the sub-PRAM. In general, $r \geq v$. The value of r depends on the hash function in use. If the hash function is not a bijection or *collision free*, additional locations will be required to resolve the collisions. For instance, the *perfect hash functions* of [4, 14] would require r to be $O(v)$. It is reasonable to expect that the requirements on a hash function should be not as strict if we simulate a PRAM on the topologies with diameter larger than $\log \mathcal{P}$. For instance, the requirement that a hash function must be computed in $O(\log \mathcal{P})$ time may now be excessive.

The obvious practical demand is that v should be as small as possible, ideally $v = r$. The class of 2-universal hash functions [2] computable in constant time suits this objective very well. In this case v is the minimal prime greater than r, and $v - r$ is known to be $O(\log^2 r)$ in the worst case. These hash functions give a much weaker theoretical guarantee that the memory requests are evenly distributed over the memory banks than do $\log \mathcal{P}$-universal hash functions. However, in the case of the mesh, it remains an open question whether their properties are sufficient in practice, i.e. it may very well be that shared memory simulation where the overhead associated with memory rehashing due to bad memory access patterns is much less than that of successful PRAM simulation cycles.

Whenever a partition step occurs, memory must be reorganized. The partition step is a union of two operations: moving of the data from the parent sub-PRAM into the simulated shared memory of the child sub-PRAMs and the replacement of the hash function. Consider variables in M_i which must be placed into a shared memory M_{i+1}. At first, each processor $p \in \mathcal{P}_{i+1}$ puts its fair share of these into its local memory in time $O\left(\frac{v_{i+1} \cdot \sqrt{|\mathcal{P}_i|}}{|\mathcal{P}_{i+1}|}\right)$, and then moves them to their correct destinations using a new hash function in time $O\left(\frac{v_{i+1} \cdot \sqrt{|\mathcal{P}_{i+1}|}}{|\mathcal{P}_{i+1}|}\right)$.

Some details, including costs and their subsumption by the charged costs in H-PRAM algorithm analysis, are discussed in [7]; other details pertaining to the constant factors involved are currently under investigation.

3 Partitioning the Peano mesh

This section presents the properties of Peano mesh partitionability, arriving at analytical results used in interpreting the experimental results in the following two sections.

The H-PRAM model does not impose any constraints on the number and sizes of the sub-PRAMs created in a partition step other than those imposed by the available number of processors. We need to be able to partition into any number of arbitrarily-sized sub-meshes, carried out dynamically at run-time. Though the sub-meshes will not generally be square, the sub-meshes must be of a shape that they well-approximate square sub-meshes of the same size, in that the diameter of a sub-mesh that a sub-PRAM maps to is $O(\sqrt{\mathcal{P}})$ for a $\mathcal{P}$-processor sub-PRAM. The Peano indexing scheme [9] nicely satisfies these criteria.

The indexing scheme I_P can be defined recursively for any $2^q \times 2^q$, $q \geq 0$ mesh by the process shown in Figure 1. The trick with the scheme is that any two nodes whose indices differ by d are at manhattan distance $O(\sqrt{d})$ in the mesh. There is a simple $O(q)$ step algorithm to convert the Peano index into Cartesian coordinates.

Given some segment of I_P, a subgraph of the mesh is defined, which we refer to as a *shape*. In the following, we give a classification of the shapes in terms of

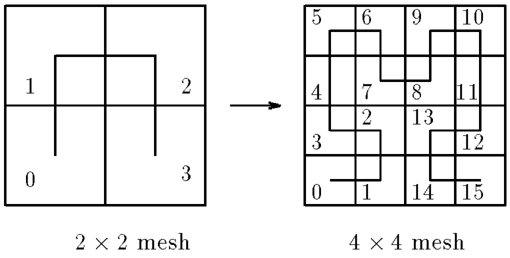

Figure 1: *The recursive rule for Peano indexing scheme. Each node of the mesh is shown as a square.*

primitive shapes and show that (1) above holds.

Definition 1 *A primitive shape is any shape that does not contain a 2×2 square.*

Twelve primitive shapes are shown in Figure 2. The complete set consists of these shapes and their reflections. Each primitive shape is identified by its type (shapes obtained by reflections have identical type). We will show that any shape has

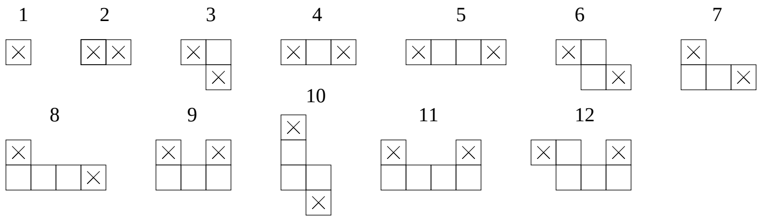

Figure 2: *Primitive shapes*

a type equivalent to one of the primitive shapes. Indeed, any shape contains a square of size $2^q \times 2^q$, $q \geq 0$, with the first node index a multiple of $2^q \times 2^q$. Let us call this *a maximal square*. In general, a shape contains more than one maximal square. The part of the Peano curve "before" the first maximal square is a shape of size less than the maximal square ("before" translates to: consisting of processors whose indices are less than those of processors within the maximal square). We can complement it by adding nodes along the Peano curve in the direction outside the shape, in order to form a (virtual) maximal square. This follows from the recursive definition of the Peano curve. The same is true for the last part of the Peano curve "after" the last maximal square. The original graph (shape) is a subgraph of that obtained by the complement process described above. The complemented shape must be congruent to one of the primitive shapes, since when reduced recursively, it becomes a shape of unit squares. As there are no other shapes of unit squares than primitive shapes the shape gets a unique type.

The other important characteristic, to which we devote the rest of this section, is the largest possible ratio of the distance between any two nodes within a shape to the square root of its size. For the primitive shapes in Figure 2, the most distant nodes in the mesh (going outside of the shape for type 9, 11, 12) are marked with crosses. Clearly, shapes 8 and 10 have the largest ratio of $4/\sqrt{5}$. In what follows we prove that shapes of this type have the asymptotically largest ratio compared to the rest. Consider the process shown in Figure (3) defining the shape of type 10. The longest distance between any pair of nodes, l_q, can be expressed by the

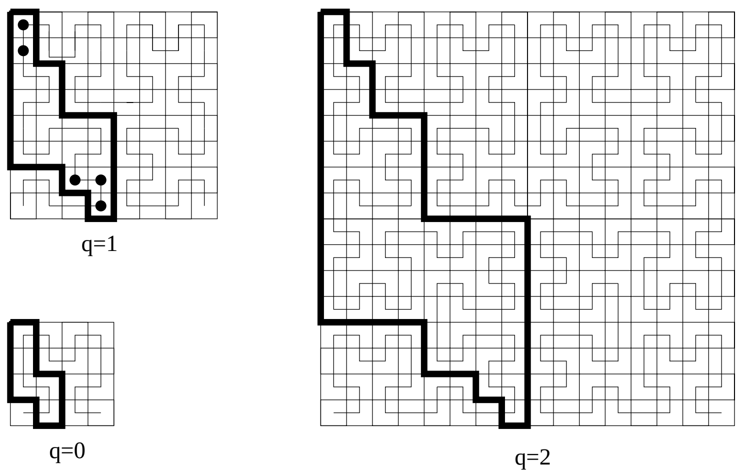

Figure 3: *Shape of type 10 is bounded by the bold line. Nodes belonging to the top and bottom maximal squares for $q = 1$ are marked with dots.*

recurrence

$$\begin{aligned} l_{q+1} &= 2 \cdot l_q + 2 \\ l_0 &= 4 \ , \end{aligned} \tag{2}$$

which has a solution: $l_q = 6 \cdot 2^q - 2$. The minimal size of a shape of type 10, i.e. the shortest length s_q of the Peano curve between nodes at distance l_q, can be found from

$$\begin{aligned} s_{q+1} &= 4 \cdot s_q - 3 \\ s_0 &= 5 \ , \end{aligned} \tag{3}$$

which has a solution $s_q = 4^{q+1} + 1$. These recurrences are obtained as follows. Consider valid shapes of type 10 with a property that the first and the last nodes are at the maximal distance from each other. Let us call these nodes *extremal.*

Consider the shape obtained from the primitive shape 10 after the first recursive step. Clearly, its size is $s_1 = 4 \cdot s_0$ and the maximal distance is equal to $l_1 = 2 \cdot l_0 + 2$. From the recursive rule defined in Figure (1) it is easy to see that the order in which four nodes of the 2×2 left upper square are enumerated is the same for the 4×4, 8×8, and larger meshes. If two largest indexed nodes are deleted at each new iterative step, we still have a shape of type 10 with one of the extremal nodes at the left upper corner. The process can be repeated at each iteration. Analogously, it is easy to check that the order in which four nodes of the 2×2 bottom right square are enumerated is the same for the 4×4, 8×8, and larger meshes. If one of the smaller indexed nodes is deleted at each new iterative step, we again have a shape of type 10 with the other of extremal nodes at the bottom right corner. Thus, in total, three nodes are deleted at each step, which is reflected in recurrence (3).

To ensure that the ratio obtained from (2) and (3) is the largest for shapes of type 10 we have to check that it is not less than that for any pair of nodes such that one of them belongs to the first (bottom) and the other to the last (top) maximal square. This is formulated in the following:

Lemma 1 *For shapes of type 10, the largest ratio of the distance to the square root of the size, within a mesh of size $2^{q+2} \times 2^{q+2}$, $q \geq 0$ is*

$$r_q^{max} = \frac{6 \cdot 2^q - 2}{\sqrt{4^{q+1} + 1}} < 3 - \frac{1}{2^q} \quad . \tag{4}$$

Proof: It is easy to see that the nodes with indices less than the minimal extremal node and larger than the maximal extremal node can not have a smaller ratio than that for the extremal nodes.

First we prove that given any node from the top maximal square ("top node") and bottom maximal square ("bottom node") the maximal ratio is achieved if the latter is an extremal node. The part of the curve within the bottom $2^q \times 2^q$ maximal square consists of two complete $2^{q-1} \times 2^{q-1}$ squares followed by the final part of the curve within the remaining $2^{q-1} \times 2^{q-1}$ square. The latter part in its turn consists of two complete $2^{q-2} \times 2^{q-2}$ squares and the final part within the remaining $2^{q-2} \times 2^{q-2}$ square and so on until the bottom extremal node. This structure follows from the construction of this shape. Thus, any non-extremal node belongs either to the first (most distant from the extremal) or to the second (nearest) $2^i \times 2^i$, $\quad 0 \leq i < q$, complete squares (see Figure 3).

Let l be the distance, s the length of the Peano curve between two nodes from maximal squares, Define l_δ to be the distance, s_δ the length of the Peano curve between a bottom node and bottom extremal node. It is sufficient to show that $\frac{l}{\sqrt{s}} < \frac{l+l_\delta}{\sqrt{s+s_\delta}}$ which is equivalent to

$$s_\delta < \frac{2 \cdot l_\delta \cdot s}{l} + \frac{l_\delta^2 \cdot s}{l^2} \tag{5}$$

The right hand is minimal if l is maximal and s is minimal. Maximal l is $6 \cdot 2^q - 2$, the distance between the extremal nodes, and s is greater than the number of nodes in three maximal squares between the top and bottom maximal squares, i.e. $3 \cdot 4^q$. If the bottom node is any node in the nearest square of size 4^i, then $l_\delta \geq 2^i$ and $s_\delta \leq 4^i + 2 \cdot \sum_{j=1}^{i} 4^{i-j}$. Thus $s_\delta < \frac{5}{3} \cdot 2^{2 \cdot i}$ whereas the right hand side is greater than $2^{q+i} + \frac{2^{2 \cdot i}}{12}$ i.e. greater than the left hand side for any $0 \leq i < q$. If the bottom node is any node in the distant square then for any node within it $l_\delta \geq 2 \cdot 2^i$ and $s_\delta \leq 2 \cdot 4^i + 2 \cdot \sum_{j=1}^{i} 4^{i-j}$, i.e. $s_\delta < \frac{8}{3} \cdot 4^i$. It is easy to check that (5) holds.

In the same way we can prove that, given any node from the bottom maximal square and the top maximal square, the bottom node and top extremal node has a larger ratio in comparison to it. Indeed, the part of the curve within the top maximal square can be represented as a sequence of $2^i \times 2^i$ squares, where $0 \leq i < q$. Hence in that case we have $l_\delta \geq 2^i$ and $s_\delta \leq \sum_{j=1}^{i} 4^{i-j}$, i.e. $s_\delta < \frac{4^i}{3}$. The inequality (5) holds in this case as well. ■

For any shape of type other than 10 we can find a recursive process that generates the shape with the largest ratio. Figure 4 shows these processes for types from 3 to 8. The types 1, 9, 11, and 12 can not have larger ratio compared to 3, 7, 8, and 10 respectively, because the latter have the same maximal distance but a smaller size. Shapes of type 2 contain a shape of type 10 with the same longest distance but smaller size for any $q \geq 1$. The ratios stated for types 3–8 in Figure 4 can be proven to be maximal for their types using the same method as above for type 10. Although type 8 has the same ratio as 10, it appears first time in the 8×8 mesh, whereas type 10 appears in the 4×4 mesh, i.e. the former is one iteration away from the latter. Thus, we can conclude that

Theorem 1 *The maximal ratio: distance to the square root of the size for any Peano shape within the mesh of size $2^{q+2} \times 2^{q+2}$, $q \geq 0$ is given by (4).*

From this and the fact that a $\mathcal{P}$ processor square has the minimal ratio $2 - 2/\sqrt{\mathcal{P}}$, we can conclude that inequality (1) holds with $c = \frac{3}{2-2/\sqrt{\mathcal{P}}}$. Indeed, from Theorem 1 it follows that the maximal possible longest distance for any $\mathcal{P}$ processor Peano shape of the mesh is less than $3 \cdot \sqrt{\mathcal{P}}$ whereas the minimal possible

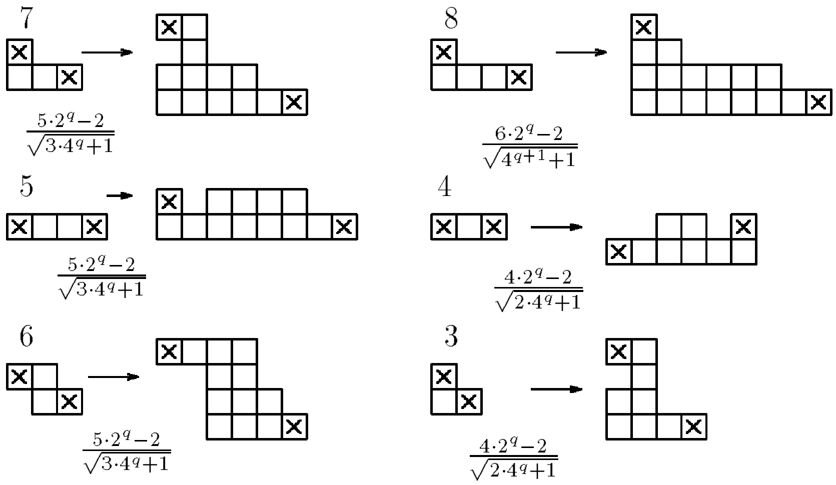

Figure 4: *The recursive definitions of the shapes with the largest ratio: distance to the square root of the size (shape of types 3–8). The ratio as a function of the recursion step $q \geq 0$ is shown below. Extremal nodes are marked with crosses.*

longest distance is $2 \cdot \sqrt{\mathcal{P}} - 2$. Setting $\mathcal{P}' = \mathcal{P}''$ in inequality (1) we find the value for c, which is asymptotically $3/2$ as $\mathcal{P} \rightarrow \infty$.

Most of this section involved a derivation of an upper bound on the ratio which can be represented analytically. In fact, it is possible to find the exact value of the maximal distance (and a ratio) over the shapes of the given size, via a *longest distance search* algorithm (see below).

4 Implementation: analysis and simulation

At the present time, we are primarily interested in *simple* routing schemes, in order to get an idea of the performance of a practical, cost-effectively scalable implementation of the H-PRAM. This effectively translates to simulating an EREW H-PRAM. Supporting concurrent reads/writes is related to the complexity of the routing algorithm and/or hardware mechanisms. We return to this in the conclusions. (For the same reason, we are not yet considering multithreading.)

Each of the sub-PRAMs comprising the EREW H-PRAM is an EREW PRAM (no two processors are allowed to reference the same memory location in one step). The shared memory M of the simulated EREW sub-PRAM is organized as a union of the local memories of the mesh processors simulating that sub-PRAM. Each local memory is assumed to consist of one memory bank, i.e. only one memory location can be fetched or stored in a single memory access cycle.

In the worst case, all memory references may occur to the same bank of memory. It was shown in [12] that given a randomly chosen k-universal hash function [2], where $k = O(\frac{\log \mathcal{P}}{\log\log \mathcal{P}})$, by means of which the memory is hashed, and any memory access pattern, then no more than $q = O\left(\frac{\log \mathcal{P}}{\log\log \mathcal{P}}\right)$ references fall into the same memory bank. Thus, in this case the routing problem we have is 1-to-q relation, i.e. at each simulated step each processor generates one request and receives at most q requests with high probability.

One of the questions we are looking to answer with the experiments is whether simple 2-universal, perfect hash functions may suffice for an efficient implementation of the H-PRAM on a mesh. As noted above, these functions possess the important feature that they can be implemented with a very small additional address space over that of the data being hashed; they are also quickly evaluable. The advantage of 2-universal hash functions is that there exist explicit such functions with a bijective mapping of the logical to physical addresses [2, 12].

We chose for our simulation the simplest two stage "greedy" routing algorithm (GRA) that routes the messages to the right column in the first stage and to the right row in the second. The first question that arises, given a partition of the H-PRAM into sub-PRAMS mapped to sub-meshes with different shapes, is whether the routing should stay within the bounds of the shape or not. This will be referred to as routing "in" and "out". Logically, sub-PRAMs are independent of each other, so one may think that the simulation of the sub-PRAMs should also be independent. This implies choosing routing "in". Below we show that routing "out" is preferable because it reduces the simulation time. If we ignore (just for the moment) the delay due to the link and memory contention, then, the only message delay is the distance a message has to go. The distance for routing "out" is not greater than that for routing "in" for any shape. The situation can be summarized with:

Theorem 2 *Let d_{in} and d_{out} be the longest distance for routing "in" and "out". Then for primitive shape 9, of size $s = 5$, $d_{in}/d_{out} = 2$; the asymptotic ratio for shapes of this type with $s \to \infty$ is at most $5/3$. Shapes of type 11 and 12 also have a ratio d_{in}/d_{out} greater than 1. Shapes of types 1–8 have $d_{in}/d_{out} = 1$.*

Proof: Obvious. ∎

The next question that arises is whether interference between independent sub-PRAMs may cause contention delays negating any advantages of routing "out". The following section gives experimental evidence that the delay due to link and memory contention is effectively bounded by a small (less than 7) additive constant to the total message routing delay. Moreover, it is independent of the size of a mesh and of a shape simulating a sub-PRAM. This is asymptotically negligible compared to the delay due to the larger multiplicative factors of Theorem 2 in the cases of shapes of type 9, 11, and 12.

The purpose of the simulation experiments carried out was to estimate the contribution of link and memory contention to the delay, i.e. the time required to simulate sub-PRAM steps on sub-meshes, for concurrently operating sub-PRAMs. The simulation is described by the tuple $\langle \mathbf{R},\mathbf{Q},\mathbf{H},\mathbf{P} \rangle$, where **R** is a routing scheme, **Q** – a queueing discipline, **H** – a hash function, and **P** – a problem. Although there exist a very large choice of deterministic and randomized algorithms (see e.g. [11]) for the routing **R** on the mesh with nice behavior in the worst case for problems like routing of permutations, there is no reason to apply more complicated schemes than the GRA given that we do not observe "bad" behavior. In general, when using these algorithms, we may expect the improvements for worst cases problems only.

We use the FIFO queueing discipline **Q** for input and output queues in each direction. Queue size has been varied from 1 to $O(\log n)$ for the $n \times n$-processor mesh. A communication protocol between the adjacent nodes has been implemented to prevent queue overflow.

Now consider the implementation of the hash functions. We use the following *local address* indexing scheme: assign the index 0 to the first free memory location of the local memory of the node with the smallest Peano index l belonging to $\mathcal{P}$-processor sub-PRAM, index 1 to the first available location of the node $l+1$, and so on until the node with index $l+\mathcal{P}-1$; followed by index $\mathcal{P}$ to the second available memory location of the node l and so on. Then the hash function $h(x)$ is a mapping of the logical address x to the local address index

$$h(x) = \left(\left(\sum_{\eta=0,k-1} a_\eta \cdot x^\eta \right) \; mod \; r \right) mod \; v \tag{6}$$

where $a_\eta \in [0, r-1]$ – uniformly distributed random variables, and $r \in Z_0^+$ is the nearest prime greater or equal to v (v defined in section 3). In our experiments k has been varied from 1 to $O(\log \mathcal{P})$.

It remains to specify the problem **P**. The H-PRAM was simulated for the problem of linear array references with a random stride, which is a typical access pattern in algorithms [8]. Each processor of a sub-PRAM generates a read request to the virtual address

$$(\zeta \cdot \# + \eta) \; mod \; v \quad ,$$

where ζ and η are random numbers with uniform distribution over $[0, v-1]$, and $\#$ is a processor index *relative to* the least Peano index of that sub-PRAM.

In the simulation experiments, the H-PRAM/Peano-mesh was randomly partitioned into sub-PRAMs (i.e. with a random number of processors in each disjoint sub-PRAM such that every processor is in some sub-PRAM). During the simulation the following statistics were collected: longest and average distances found for shapes of the given size, and worst and average delay required to complete a sub-PRAM step of the given size. The delay is expressed in the number of mesh neighbor-to-neighbor communication steps. Each simulated sub-PRAM was assigned a different number of instances of the problem, proportional to $1/\sqrt{\mathcal{P}}$, so that every sub-PRAM terminated approximately in the same time. This *overloading* of the small-sized sub-PRAMs sustains the level of link contention for messages crossing the boundaries of the shape.

5 Experimental Results

The experiments were done for $n \times n$, $4 \leq n \leq 16$ square meshes. We report results for 256 processor mesh. The experiments indicate that the number of the worst and average case delays does not change with n. Given that this remains unchanged for larger n, the number of communication steps needed to simulate a step of the sub-PRAM of $\mathcal{P}$ processors of any shape within a mesh of $2^{q+2} \times 2^{q+2}$ processors in the worst case can be probabilistically upper bounded by

$$2 \cdot r_q^{max} \cdot \sqrt{\mathcal{P}} + C \quad ,$$

where r_q^{max} is defined in (4) and C is a constant less then 7, found experimentally. The justification of this result is given below. Again, we note that a more accurate prediction than the analytic bound can be obtained by using a "longest distance" search algorithm. In this a case constant C must be added to the value of the longest distance for the given sub-PRAM size generated by the algorithm.

Figure 5 compares simulation results against the analytical results produced earlier. In a large number of cases we have a match between the longest distance predicted by the longest distance search algorithm and obtained in the simulation. Based on the results above, for a sub-PRAM of size $\mathcal{P}$, the upper bound on distance in any shape within the mesh of size $2^{q+2} \times 2^{q+2}$, $q \geq 0$ is $2 \cdot r_q^{max} \cdot \sqrt{\mathcal{P}}$. The appearance of the factor of two is due to the fact that the messages are read requests. The constant factor $2 \cdot r_q^{max}$ for a 16×16 mesh is equal to $\frac{44}{\sqrt{65}}$.

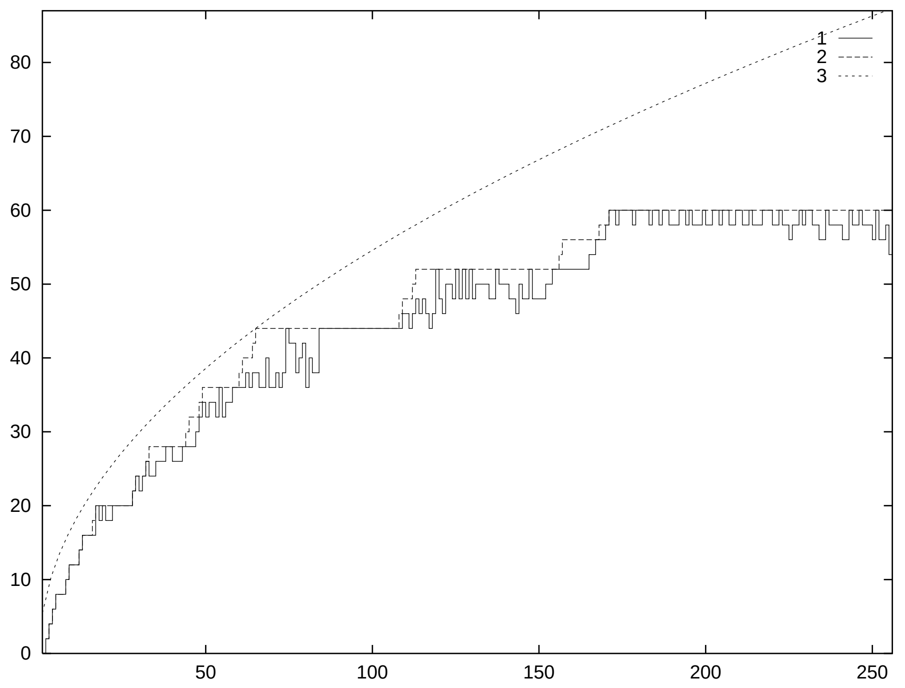

Figure 5: *1 – the longest distance found by random sampling, 2 – the longest distance found by the longest distance search algorithm, 3 – the analytical upper bound (all plotted against sub-PRAM size)*

Figure 5 demonstrates the relative discrepancy between the upper bound on the distance for 256 node H-PRAM, i.e. $\frac{44}{\sqrt{65}}\sqrt{\mathcal{P}}$ and that found by the longest distance search algorithm. At the point $\mathcal{P} = 65$ the two curves meet. This corresponds to the shape of type 10, shown in Figure 3 for $q = 2$. The third curve represents the longest distance found as a result of random partitioning of the H-PRAM when solving the problem P as described above.

Figure 6 represents the largest and average delay observed in a simulation of the H-PRAM on a mesh with queue size equal to two, plotted against sub-PRAM size. It is seen that the delay is only a small number of steps away from the distance curve. The more important observation is that the difference does not tend to vary with the size of the shapes. This difference is the number of delays due to link contention and is more clearly shown in Figure 7. The simulation results are in agreement with results due to Leighton [10], who showed that the random destination routing problem can be solved on a square mesh in $2n$ communication steps with the queue size equal to 4 with high probability under the furthest-first queueing discipline. Our results demonstrate that delay greater than 4 is exceptional. The worst case delay due to the link contention larger than 4 (note, that

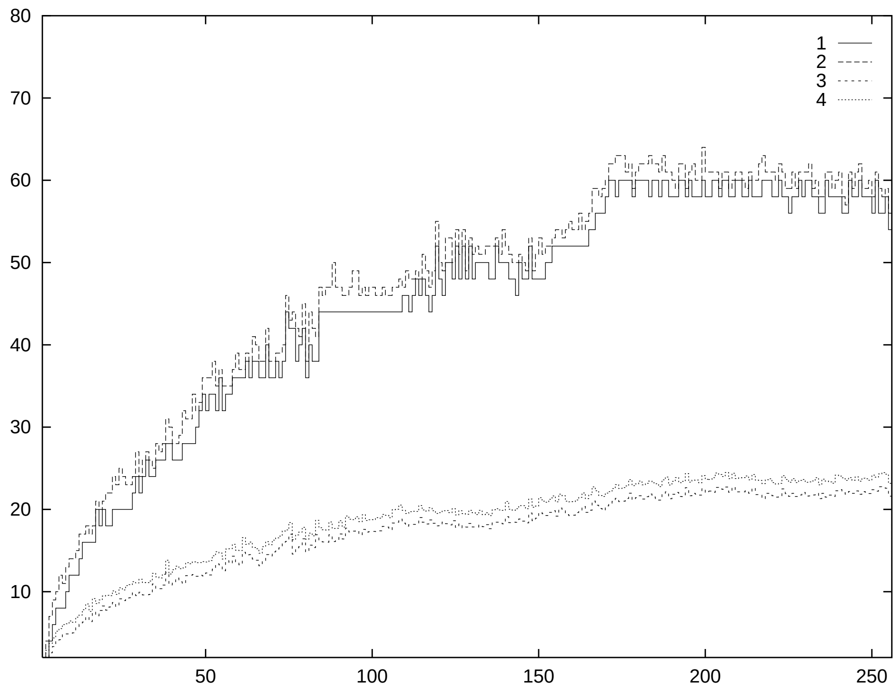

Figure 6: *Number of mesh communication steps (y axis) to complete a simulation of one step for the sub-PRAM of random shape with a given number of processors (x axis) within the 256 processor mesh. 1 – longest distance found (as 1 in Figure 5), 2 – the maximal number of steps to complete the simulation of the sub-PRAM (longest delay), 3 – average distance, 4 – average delay. Simulation results for 20,000 random partitions. Queue size = 2.*

the delay presented in Figure 7 is for messages traveling both to and from their memory destination) was observed in less then 5% of the cases, and the maximal delay is equal to 6 in 1.2% of the cases. Thus, the probability of observing the larger worst case delay is a rapidly decreasing function of its argument, independent of the shape and size of a sub-PRAM. The observed delay due to the link contention averaged over the number of runs is less than one for any sub-PRAM size. This suggests that any interference between the sub-PRAMs from routing "out" is negligible compared to their diameter, i.e. routing "out" will always be faster than routing "in".

It is worth remembering that our implementation requires only two one-element buffers: input and output to implement a queue. The experiments with larger buffers organized in a FIFO queue show no noticeable improvements in performance.

Another important outcome of the simulation is that varying the choice of k in the hash functions (6), which determines the minimal value of the probabilist-

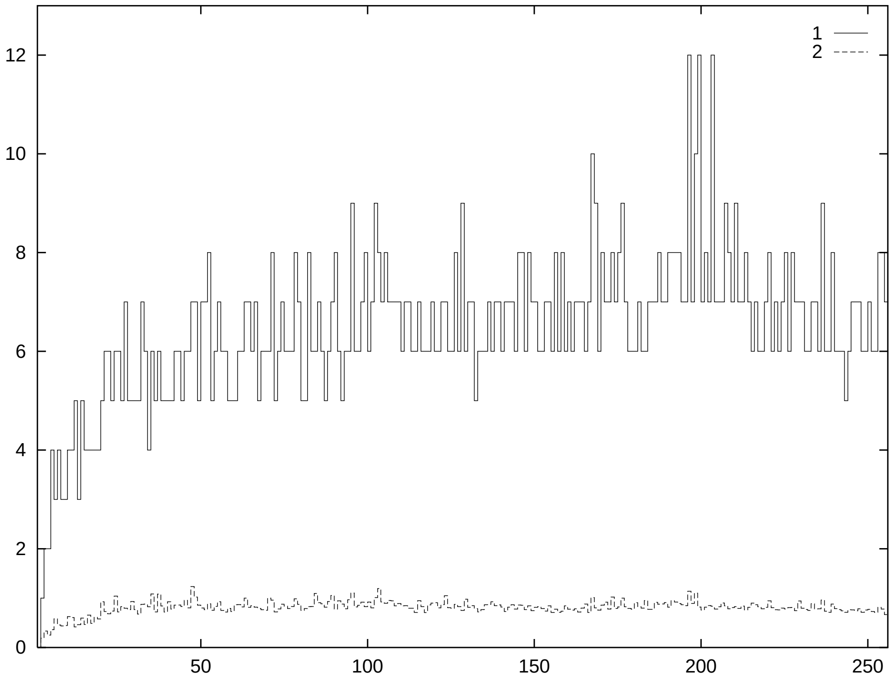

Figure 7: *1 – the largest observed number of link contention during one sub-PRAM simulation step, 2 – average delay due to the link contention. Queue size = 2.*

ically independent outputs for any k possible unequal inputs, did not lead to any noticeable reductions either in the value of the worst or average observed delay. This suggests that at least for the problem P under consideration, 2-universal hash functions are sufficient. This result is similar to results observed for the butterfly network simulating a PRAM [13].

6 Conclusions & Future Work

We have presented an analytical and experimental investigation of aspects of the implementation of the H-PRAM on a two dimensional mesh. Our technique exploits the Peano indexing scheme to preserve locality. We have demonstrated that the shapes of the sub-mesh areas induced by the scheme have satisfactory properties with respect to maximal path length. More specifically, we have developed a classification of the shapes in terms of 12 possible types and have demonstrated an exact upper bound on the longest distance within a shape for all these types as a function of the number of nodes in a shape.

These results allowed us to predict the number of communication steps needed to simulate a step of the sub-PRAM of any shape within a mesh of any size very accurately, provided that the delay due to the link contention is negligible compared to the longest distance. Our experimental evidence shows that this can indeed be the case. The worst case delay due to link and memory contention when simulating an H-PRAM step did not exceed 6 for all sub-PRAMs of different shapes simulated, for H-PRAMs/meshes of up to 256 processors. This suggests that the major contribution to the delay is due to the longest distance within a sub-PRAM that a message has to travel.

It is important to note that these results have been obtained for a simple, practical and cheaply implementable routing scheme, with two element FIFO queues in each direction and fast 2-universal hash functions requiring minimal additional memory for implementation. Recall from the introduction that our goal is cost-effective scalability.

We also note that our mechanism could be adapted for a variety of topologies, and emphasize the fact that it works on bounded degree networks like the mesh, where such mechanisms as parallel slackness [16] do not (see [1]) improve the performance by more than a constant factor. (Exploitation of parallel slackness, or multithreading, also imposes onerous and costly requirements on the router in logarithmic diameter topologies, which must be capable of dealing with $O(\log \mathcal{P})$ messages in a fixed time with growing $\mathcal{P}$.)

Future work in this project will consider extension of the scheme to cover issues of concurrent access to locations (i.e. CREW and CRCW variants). We hope to show that the extra burdens imposed can be addressed in software (for example, implementation via prefix computations and sorting, hence without expensive hardware) without crippling performance, due to the H-PRAM's systematic exploitation of locality to control costs. We are also investigating higher level aspects of our approach, including ways in which the conceptual tools involved can be embedded in a realistic programming language. This will lead on to experimentation with the hierarchical structures and memory access patterns generated by real applications.

In a wider context, we anticipate that the *partitioning* techniques introduced in this paper will prove beneficial for a very broad class of parallel algorithms, with respect to meshes, whether implemented through the H-PRAM abstraction or otherwise.

References

[1] Bilardi, G., Preparata, F.P., "Horizons of Parallel computation." in: A. Bensoussam, J.-P. Verjus (eds.), *Future Tendencies in Computer Science, Control and Applied Mathematics, Int. Conf. on the Occasion of the 25th Anniversary of INRIA*, LNCS 653, pp. 155-174, 1992.

[2] Carter, J.L., and Wegman, M.N., "Universal classes of hash functions", *J. Comput. Syst. Sci.*, 18:143-154, 1979.

[3] Choi, L. and Chien, A.A., "Integrating Networks and Memory Hierarchies in a Multicomputer Node Architecture", *International Parallel Processing Symposium*, 1994.

[4] Dietzfelbinger, M., Meyer auf der Heide, F., "A new universal class of hash functions and dynamic hashing in real time". In M.S. Paterson, editor, *Proc. of the 17th ICALP*, pp. 6-19. Springer, 1990. LNCS 443.

[5] Gibbons, P., "Models of Parallel Computation: An Overview", *DIMACS Workshop on Models, Architectures, and Technologies for Parallel Computation*, Sept. 1993, DIMACS Tech. Report 93-87, pp. 8-10 and 59-65.

[6] Heywood, T., and Leopold, C., "Models of Parallelism", *Proc. 2nd Leeds Workshop on Abstract Models for Highly Parallel Computers*, Oxford Univ. Press.

[7] Heywood, T., and Ranka, S., "A Practical Hierarchical Model of Parallel Computation. I. The Model", *Journal of Parallel and Distributed Computation*, 16, pp. 212-232, 1992.

[8] Heywood, T., and Ranka, S., "A Practical Hierarchical Model of Parallel Computation. II. Binary Tree and FFT Algorithms", *Journal of Parallel and Distributed Computation*, 16, pp. 233-249, 1992.

[9] Kaklamanis, C., and Persiano, G., "Branch-and Bound and Backtrack Search on Mesh-Connected Architectures", *Proc. 4th ACM Symp. on Parallel Algorithms and Architectures*, pp. 118-126, 1992.

[10] Leighton, T.,"Average case analysis of greedy routing algorithms on arrays" *Proc. ACM Symp. on Parallel Algorithms and Architectures*, pp. 2-10, 1990.

[11] Leighton, T., Methods for message routing in parallel machines", *24th Annual ACM Symp. of Theory of Computing*, pp. 77-96, 1992.

[12] Mehlhorn., K., and Vishkin, U.,"Randomized and Deterministic Simulations of PRAMs by Parallel Machines with Restricted Granularity", *Acta Informatica*, 21, pp. 339-374, 1984.

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

[14] Siegel, A., "On universal classes of fast high performance hash functions, their time-space tradeoff, and their applications", *Proc. of the 30th IEEE Ann. Symp. on Foundations of Computer Science*, pp. 20-25, 1989. Revised Version.

[15] Siegel, H.J., et. al.,"Report of the Purdue Workshop on Grand Challenges in Computer Architecture for the Support of High Performance Computing", *Journal of Parallel and Distributed Computing*, 16, pp. 199-211, 1992

[16] Valiant, L.G., "General purpose parallel architectures", *In J. van Leewen, editor, Handbook of Theoretical Computer Science, Vol. A: Algorithms and Complexity*, chapter 18, pages 943-971. Elsiever, Amsterdam, 1990.