Dynamic Randomized Simulation of Hierarchical PRAMs on Meshes*

Todd Heywood†
Department of Computer Science
University of Edinburgh
Edinburgh EH93JZ, Scotland
thh@dcs.ed.ac.uk

Claudia Leopold
Fakultät für Mathematik und Informatik
Friedrich-Schiller-Universität Jena
07740 Jena, Germany
claudia@minet.uni-jena.de

Abstract

The Hierarchical PRAM (H-PRAM) [5] model is a dynamically partitionable PRAM, which charges for communication and synchronization, and allows parallel algorithms to abstractly represent general locality. In this paper we show that the H-PRAM can be implemented efficiently on a two-dimensional mesh. We use the Peano indexing scheme to hierarchically partition the mesh. Multiple sub-PRAMs of the H-PRAM are simulated on irregular sub-meshes. For an H-PRAM program of cost T, the overall CRCW H-PRAM simulation runs in time constant in T with high probability. The simulation is dynamic, i.e. it does not depend on prior knowledge of a program's specific hierarchical configuration, which may be data dependent.

1 Introduction

In parallel computing, a model of computation has a difficult task. It must mediate between the conflicting requirements of abstraction (ease of use) for program design/analysis, and 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 [13].

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 recognizance of this fact, many other models have been proposed (see [2], 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 motivating the study of the H-PRAM is given elsewhere in detail [3, 5], so we will not discuss these issues here. See also [2].

The Hierarchical PRAM (H-PRAM), introduced in [5] and [6], 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

*1995 Aizu International Symposium on Parallel Algorithm/Architecture Synthesis
†Supported by EPSRC grant GR/J43295

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 aim of this paper is to demonstrate that a P-processor H-PRAM can be *implemented* efficiently on a P-processor mesh (the reflectivity shown in [5] is rather abstract). This is important since a model of computation needs to consider the long-term in addition to the current generation of architectures, and considerations of the physical limitations of parallel computation (e.g. [1]) indicate that future architectures will be mesh-like.

The analysis in this paper is up to constant factors that are small compared to the rest of the formula under consideration. However, we cannot use the standard big Oh notation since, for analysis simplification purposes, we bound the number of processors from above (by the number of particles in the universe). Use of big Oh notation requires that no upper bound be placed on the values that the parameters may take. We mark this distinction by using the notation $\hat{O}()$ instead of $O()$.

The H-PRAM simulation given in this paper is an algorithm that can interpret a given H-PRAM *program instance*. We distinguish "program instance" from "program" since we are only concerned about those instances of a program that are relevant to practice. Specifically, the set of possible inputs to the simulation algorithm is a set of program instances and the input size corresponding to a specific input is the H-PRAM cost complexity of that program instance. The notion "program" would imply a non-fixed cost complexity of the simulation depending on the size of the input to the program being simulated, that we are not interested in. However, in the interest of avoiding clumsy phrasing the rest of the paper will use the term "program", assuming that the reader understands this to mean "program instance".

As the simulation algorithm cannot know in advance how the H-PRAM hierarchy will grow and contract according to data dependencies, it is necessary that the mapping of the H-PRAM to the mesh be a dynamic one. In other words, the mesh must be able to partition into any number of arbitrarily-sized sub-meshes so that sub-PRAMs map to sub-meshes. Further, 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 sub-PRAM maps to is $\hat{O}(\sqrt{p})$ for a p-processor sub-PRAM. The Peano indexing scheme [7] nicely provides these criteria. This paper gives a mapping of the H-PRAM to a Peano-indexed mesh, and then considers a randomized H-PRAM simulation where sub-PRAMs are simulated on irregular sub-meshes via an adaptation of Ranade's randomized PRAM simulation on a butterfly [11] (the details of this are omitted because of page limitations, but can be found in the full version [4]). It then considers the recursive simulation of multiple sub-PRAMs in parallel, assuming the existence of a high probability sub-PRAM on sub-mesh simulation method. The overall CRCW H-PRAM simulation runs in time $\hat{O}(T)$ for an input program of H-PRAM complexity T with high probability, even though all the concurrently operating sub-PRAMs only achieve their time bound probabilistically.

Section 2 of the paper reviews the H-PRAM and outlines the Peano indexing scheme for the mesh. Section 3 gives the results on simulating PRAMs on irregular sub-meshes and analyzes the overall simulation.

2 Dynamic mapping of the H-PRAM to a mesh

Structurally, the H-PRAM [5] is a PRAM which can recursively partition itself into sub-PRAMs, giving rise to a hierarchy of sub-PRAMs (in this paper, any kind of CRCW PRAM). A P-processor H-PRAM is a P-processor PRAM extended by a partition instruction:

```
partition {  p1: Algorithm-1 (parameter-list);
             p2: Algorithm-2 (parameter-list);
             ...
             pQ: Algorithm-Q (parameter-list) }
```

where the p_i are positive integers with $\sum_{i=1}^{Q} p_i = P$. (A notational note: we use upper case P and M to denote the processor and memory resources of the H-PRAM, and lower case p and m to denote the resources of a sub-PRAM in the hierarchy.)

The operation partitions the P processors of the original PRAM into disjoint subsets of p_i processors running the specified algorithms. We assume partitioning of processors causes the memory to be partitioned proportionately ($M/P = m/p$), this was called the "private H-PRAM" variant in [5]. 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 series–parallel graph, where "forks" correspond to the starts of partition instructions and "joins" correspond to the terminations of them [5].

We assume the case of dynamic partitionability here, i.e. the parameters Q and p_i in the partition instruction need not be known in advance. While the algorithms in [6] are static, dynamic partitioning is more general, as it allows hierarchies to depend on data values.

Before defining the H-PRAM cost measure, we need to fix some terminology. In this paper, the term *step* of a sub-PRAM refers to a step *in that sub-PRAM*. In contrast, a partition *hyperstep* includes the activities in all sub-PRAMs hierarchically produced by a partition instruction. A step is either a usual PRAM step, where each processor may independently carry out a computation, communication or idle operation, or it is a partition step, i.e. the "invocation" of a partition hyperstep. To avoid confusion, the notion *cost* will always be tied to the charged H-PRAM cost of an algorithm, in contrast to the *simulation time*.

The evaluation of algorithmic 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 [5, 6] for complexity analysis details. Summarizing for the purposes of this paper, we charge $\sqrt{p}$ for *every* step in a p-processor sub-PRAM (whether communication, computation, or partition).

The complexity of an H-PRAM program is defined the obvious way: Costs caused by successive steps within the same sub-PRAM sum up, while the cost of a partition hyperstep is equal to the maximal cost of its sub-sub-PRAM programs. The cost of the sub-sub-PRAM programs is determined recursively, in the same manner. In this paper, we will also use an equivalent, but sometimes more convenient, definition determining the cost T of a program:

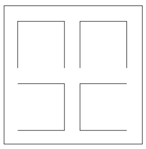
$$T = \max_{\text{paths}} \left(\sum_{\text{steps on that path}} \sqrt{\begin{array}{l}\text{size of sub-PRAM}\\ \text{executing that step}\end{array}} \right)$$

where a path is any path through the series–parallel graph representing the program structure.

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. This freedom for the programmer, on the other hand, brings along difficulties for the simulation task. In contrast to the realization of the YPRAM [14], we cannot solely rely on the fact that a square sub-mesh can be recursively partitioned into four square sub-meshes. Instead, 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, each p-processor sub-mesh must have diameter $\hat{O}(\sqrt{p})$, and permit PRAM simulation on it in $\hat{O}(\sqrt{p})$ mesh routing steps per PRAM step. The Peano indexing scheme (see [7]) allows us to meet these criteria.

Consider meshes of size $2^n \times 2^n$, integer n. This restriction is not serious, as a $2^n \times 2^n$ -processor mesh can simulate any $q \times q$ -processor mesh ($2^n \leq q < 2^{n+1}$) in time $O(1)$ per step [8].

The Peano scheme indexes a mesh according to the pattern



which is recursively applied, such that an 8×8 -mesh is indexed

22	23	26	27	38	39	42	43
21	24	25	28	37	40	41	44
20	19	30	29	36	35	46	45
17	18	31	32	33	34	47	48
16	13	12	11	54	53	52	49
15	14	9	10	55	56	51	50
2	3	8	7	58	57	62	63
1	4	5	6	59	60	61	64

The trick with the scheme is that any two nodes whose indices differ by d are at Manhattan distance $\hat{O}(\sqrt{d})$ in the array.

Consider a partition instruction partitioning a p-processor sub-PRAM into $p'_1, p'_2, \dots p'_Q$ -processor sub-sub-PRAMs. Assume the p-processor sub-PRAM is mapped to contiguously-indexed mesh nodes $v_1 \dots v_1 + p$. Then the p'_1 -processor sub-PRAM is mapped to mesh nodes $v_1 \dots v_1 + p'_1 - 1$, the p'_2 -processor sub-PRAM to $v_1 + p'_1 \dots v_1 + p'_1 + p'_2 - 1$, etc. Note that this very simple method works with hierarchically nested partitions, and can be applied dynamically.

3 A randomized H-PRAM simulation

In the full version of this paper [4], we describe how the sub-meshes constructed in Section 2 can simulate a sub-PRAM by proving the following Theorem.

Theorem 3.1 *Provided the memory is hashed via a universal hash function, each step of a p-processor m-memory sub-PRAM can be simulated on a sub-mesh (consisting of contiguously-indexed processors) of a Peano-indexed mesh in time $\hat{O}(\sqrt{p})$ with probability $1 - p^{-k}$, for any k. The memory requirements per mesh node are $\hat{O}(m/p)$.*

It is quite obvious that a simulation with the properties stated in this Theorem exists. A straightforward simulation would map everything (memory locations/requests) into a square "kernel" of the sub-mesh. CRCW PRAM simulation in a square mesh can be done by sorting and combining the requests in a prephase, and afterwards simulating an EREW PRAM. For EREW PRAM simulation, the (square) mesh is modeled by a leveled network [9], and Ranade's technique of universal hashing combined with deterministic constant–queue–size routing [9], [10], [11] applied. In the full paper, we show that the leveled network can be associated with the whole sub-mesh (not only the kernel). Additionally, we show how to omit the prephase by combining during routing, while guaranteeing constant queue size. (The bound derived e.g. in [11] is actually an asymptotic bound, but as the suppressed constant is relatively small, we can also take it as an $\hat{O}()$-bound.)

In the following, we assume to be given any method leading to Theorem 3.1, e.g. the method described in the full version or some other method. We will prove that, given a set of reasonable assumptions on the properties of a time T CRCW H-PRAM program, the overall H-PRAM simulation runs in $\hat{O}(T)$ time with high probability, even though the concurrently operating sub-PRAMs only achieve their time bounds probabilistically. Note that the assumptions restrict the applicability of the analysis method only, not those of the simulation. The assumptions are:

1. We are to simulate a P-processor M-memory H-PRAM program on a P-processor $\hat{O}(M)$-memory mesh. The mesh can be hierarchically partitioned into any number of arbitrarily-sized sub-meshes. Provided the memory is initially hashed, each p-processor $\hat{O}(m)$-memory sub-mesh is able to simulate a p-processor m-memory sub-PRAM such that
   - each sub-PRAM step is simulated in time $\hat{O}(\sqrt{p})$ with probability at least $1 - 1/p^3$, and in time $\hat{O}((m/p)\sqrt{p})$ otherwise, and
   - the simulation time of each step is statistically independent from other steps.
2. For all sub-PRAMs: number of steps $= \hat{\Omega}((m/p) \cdot$ number of partition steps).

3. For all sub-PRAMs: $p \geq m/p \geq 45$.

4. For all partition instructions dividing a p-processor sub-PRAM into $p'_1, p'_2, \ldots p'_Q$ -processor sub-PRAMs, $p/p'_i \geq 16$ holds for all $1 \leq i \leq Q$.

5. Depth of the hierarchy created by the H-PRAM program ≤ 67.

Assumption 1 is justified by the previous section and Theorem 3.1. The value $1 - 1/p^3$ is just a special case of the result of Theorem 3.1 (setting $k = 3$), and simulation time $\hat{O}((m/p)\sqrt{p})$ is guaranteed by switching to deterministic simulation if a step takes too long. Statistical independence can be maintained by choosing a new hash function and rehashing after each unsuccessful step. In the case of a "bad" hash function (i.e., a function that hashes more than $\hat{O}(\log p)$ PRAM memory cells into the same group of processors in secondary hashing, see [11] App. A), we first execute the next PRAM instruction (deterministically) before trying another hash function (to avoid a step takes arbitrarily long). As a hash function is bad with probability $1/p^3$ only, Assumption 1 follows.

In H-PRAM simulation, whenever a sub-PRAM is created its memory needs to be hashed, and when it finishes its memory needs to be unhashed. Also, when a partition instruction is initiated, the sub-PRAM memory needs to be unhashed, and when the instruction terminates its memory needs to be rehashed. Each hashing/unhashing has cost $\hat{O}((m/p)\sqrt{p})$. Assumption 2 secures that the cost of these hashings/unhashings is subsumed by the charged H-PRAM cost. Assumption 2 slightly restricts the set of H-PRAM programs covered by the analysis of the simulation method. Programs which undertake memory management on m elements (which would appear to be the normal case for non-trivial problems since programs need to organize memory to take advantage of locality) will necessarily execute $\hat{\Omega}(m/p)$ memory reference steps prior to every partition instruction, implying Assumption 2.

If $m/p \geq 45$ is not fulfilled in Assumption 3, switch to deterministic simulation, which then takes time $\hat{O}(\sqrt{p})$. The requirement $p \geq m/p$ secures that the cost of hashing does not become too large with respect to its frequency. Hashing in small PRAMs would be a waste; one could switch to deterministic simulation and charge $\hat{O}(p)$ per step. Assumption 3 touches on the question of whether communication cost is best represented by number of processors p or memory per processor m/p when $p \leq m/p$ (i.e. interprocessor communication and I/O).

Assumptions 4 and 5 do not seem to be essential, but reflect some rough estimations employed by our proof technique. In any case, we expect most H-PRAM programs to be structured such that partition instructions either fulfill Assumption 4 or violate it for all sub-PRAMs created in the partition. In the second case, Assumption 4 can be established by omitting the partition and running the sub-PRAMs concurrently in the p-processor PRAM. This modification increases the cost per sub-PRAM instruction from $\sqrt{p'_i}$ to $\sqrt{p} < 4\sqrt{p'_i}$, and hence the cost of the program by at most the factor 4. Taking into account that the number of H-PRAM processors can not be higher than the number of particles in the universe (10^{80} as stated in [1]), Assumption 5 is a consequence of Assumption 4 since $\log_{16} 10^{80} \leq 67$.

The analysis given in this section is somewhat simplified in that it assumes certain values to be integers, and others to be multiples of some other values, though this need not be the case. The exact analysis is given in the full version of the paper [4].

We will *analyze* an *idealized* simulation characterized by the following properties.

1. Assumption 1 is interpreted as follows: The cost of simulating the sub-PRAM step is *exactly* $s\sqrt{p}$ (in the "successful" case) or $s(m/p)\sqrt{p}$ (in the "unsuccessful" case), for some fixed constant s. Also, assume the successful case occurs with probability *exactly* $1/p^3$.

2. Whenever a "straightline sub-program segment" (i.e. with no partition instruction) of cost l has simulation time less than $2sl$, let the simulation take time $2sl$.

Note that the idealized simulation is not performed, but is introduced for purposes of *analysis* only. In the present context, we understand the simulation as an algorithm whose input is an H-PRAM program (instance) and that needs a certain amount of running time, depending on the input size which is the cost T of the program (instance). The only properties of the *idealized* simulation we are interested in are that it is operating on the same set of inputs and takes at least the same time on all inputs as does the *real* simulation (which is clearly the case). The semantics of the idealized simulation is irrelevant here, in fact it can be any meaningless algorithm.

The most important technical result of this section will be the following Theorem.

Theorem 3.2 *For each H-PRAM program B of cost T, there exists an H-PRAM program A of cost $\hat{O}(T)$ such that*

- *A does not contain partition instructions, and*
- *The probability A is simulated in $\hat{O}(T)$ is at most the probability that B is simulated in $\hat{O}(T)$.*

Theorem 3.2 forms the basis to establish our main result:

Theorem 3.3 *Under Assumptions 1-5, an H-PRAM program of cost T can be simulated on the mesh in time $\hat{O}(T)$ with high probability. The probability goes to 1 as T goes to infinity.*

Proof: From Theorem 3.2, consider the corresponding H-PRAM program A which does not have partition instructions, i.e. a PRAM program of cost $T' = \hat{O}(T)$. Let $T' = t(M/P)\sqrt{P}$, for any $t \geq 1$, and allow the simulation to take time $cT = c'sT'$. The bound is met if there are at most $t(c'-1)$ rehashings. The claim follows from

$$\mathrm{Prob}(\#\text{rehashings} \geq t(c'-1))$$
$$\leq \left(\frac{t(M/P)(e/P^3)}{t(c'-1)}\right)^{t(c'-1)} \leq \left(\frac{e}{P^2(c'-1)}\right)^{t(c'-1)}.$$

where the first inequality derives from the Chernoff bound stated in [12]. ■

3.1 Proof of Theorem 3.2

First, Lemma 3.4 will restrict the set of H-PRAM programs which need to be considered in the proof of Theorem 3.2. We say that a sub-PRAM is *immediately created* in a partition step, if it is created there, and not on deeper levels of the hierarchy.

Lemma 3.4 *For every H-PRAM program B of cost T, there exists an equivalent program B' of cost $\hat{O}(T)$ such that for all partition instructions called in p-processor m-memory sub-PRAMs of B' holds:*

$$\textit{total cost of the partition hyperstep} \geq (m/p) \cdot \sqrt{p}.$$

Proof Sketch: We construct B' by inserting idle steps with total cost $m/p\sqrt{p}$ into all sub-PRAMs being immediately created in a partition step of a p-processor sub-PRAM of B. Clearly, B' fulfils the requirement. Cost $\hat{O}(T)$ is a consequence of Assumption 2. ■

The proof of Theorem 3.2 will be by stepwise reduction. In each step, we reduce the simulation of one program to that of another program taking the same or a higher simulation time with at least the same probability. The approach resembles that of the "idealized simulation" discussed above in that the programs constructed by the reductions are not simulated, but are introduced for purposes of analysis only. There are two types of reduction, described by the following two Lemmas.

Lemma 3.5 *(Reduction 1) Consider two Cases:*

(a) Straightline $p'_1, p'_2, \ldots p'_Q$ -processor sub-PRAM programs running in parallel, sometimes synchronizing with each other. The distances l_i between synchronizations fulfil $(m/p)\sqrt{p} \leq l_i \leq 2(m/p)\sqrt{p}$ and $\sum l_i = l$.

(b) Straightline $p'_1, p'_2, \ldots p'_Q$ -processor sub-PRAM programs running in parallel with (maximal) cost l.

Let the random variable Z_a denote the simulation time in Case (a), and Z_b in Case (b). Then,

$$\mathit{Prob}(Z_b \geq Z) \leq \mathit{Prob}(Z_a \geq Z), \quad \text{for all } Z \in R.$$

Lemma 3.5 can easily be shown recalling the definition of H-PRAM cost and observing that the simulation time is the maximal simulation time over all paths. As the programs of Cases (a) and (b) consist of the same operations, the set of program paths in (b) is simply a subset of those in (a). Hence, the simulation in Case (a) cannot be faster than in Case (b), implying $\text{Prob}(Z_b \geq Z) \leq \text{Prob}(Z_a \geq Z)$.

The second reduction is not straightforward, and will be proven later.

Lemma 3.6 *(Reduction 2) Consider two Cases:*

(a) A straightline p-processor sub-PRAM program of cost l where $(m/p)\sqrt{p} \leq l \leq 2(m/p)\sqrt{p}$, and

(b) Straightline $p'_1, p'_2, \ldots p'_Q$-processor sub-PRAM programs running in parallel with (maximal) cost l and $\sum_{i=1}^{Q} p'_i = p$.

Let the random variable Z_a denote the simulation time in Case (a), and Z_b in Case (b). Then

$$\mathit{Prob}(Z_b \geq Z) \leq \mathit{Prob}(Z_a \geq Z), \quad \text{for all } Z \in R. \tag{1}$$

Given Lemmas 3.5 and 3.6, Theorem 3.2 can be proven as follows. We transform B into A through a sequence of intermediate programs C_1, D_1, C_2, D_2 etc. We obtain C_1 from B by applying the first reduction (Lemma 3.5) to all partition hypersteps *at the deepest hierarchy level*, and D_1 by applying the second reduction (Lemma 3.6) to the segments those partition hypersteps were transformed into. We obtain the other C_i and D_i the same way, proceeding toward the root of the hierarchy. According to the reductions, the probability that the programs are simulated in time $\hat{O}(T)$ can only decrease from program B to C_1 to D_1 to $\ldots$ to A. To see that the transformations are constructive, note that in Theorem 3.2, A can be any meaningless program. To construct D_i from C_i, execute steps of the desired cost in the sub-PRAM one level above in the hierarchy (and cancel the partition hyperstep). To construct C_{i+1} from D_i (or B), observe that each value $l \geq (m/p)\sqrt{p}$ can be represented as a sum of values l_i from the range $(m/p)\sqrt{p} \leq l_i \leq 2(m/p)\sqrt{p}$. Note that Lemmas 3.5 and 3.6 as given in this paper are simplified versions of those in the full paper [4]. The cost l cannot in general be the same for Cases (a) and (b). The reason is that PRAM steps are indivisible, such that we need $\sqrt{p'_i}|l$ and $\sqrt{p}|l$. Briefly, the extended versions of Lemmas 3.5 and 3.6 in [4] allow the cost in Case (a) to be slightly higher than in Case (b). Correspondingly, the transformations slightly increase the cost of the programs. Fortunately, the increase is very small, a factor of at most $(1 + 7p/12m)(1 + p/m)$ per H-PRAM hierarchy level. By Assumption 3, the cost of A is larger than that of B by at most a multiplicative factor of $1.0355^{\text{hierarchy-depth}}$, which can be considered constant under Assumption 5. This completes the proof of Theorem 3.2. ■

The rest of this section focuses on proving Lemma 3.6 (Reduction 2). We restrict ourselves to the case $p'_1 = p'_2 = \ldots = p'_Q = p'$, which can be easily generalized.

We denote the simulation time of the i-th sub-PRAM in Case (b) by Z_b^i. Furthermore, we denote the number of rehashings in Case (a) by X_a, and the number of rehashings in the sub-PRAMs of Case (b) by X_b^i (for sub-PRAM i).

In Case (a), we have $(l/\sqrt{p}) - X_a$ steps of simulation time $s\sqrt{p}$ each, and X_a steps of simulation time $s(m/p)\sqrt{p}$, i.e.

$$Z_a(X_a) = \max\{s(l + X_a((m/p) - 1)\sqrt{p}), 2sl\}.$$

Similarly,

$$Z_b^i(X_b^i) = \max\{s(l + X_b^i((m/p) - 1)\sqrt{p'}), 2sl\}$$

Let $Z = Z(r) = s(l + r((m/p) - 1)\sqrt{p})$, for $r \in R$. Note that from Property 2 of the idealized simulation, we have $\text{Prob}(Z_a \geq Z(r)) = \text{Prob}(Z_b \geq Z(r)) = 1$, for $Z(r) \leq 2sl$. Hence, it suffices to prove Lemma 3.6 for $Z > 2sl$, which implies $r \geq 2$ (as $l \geq (m/p)\sqrt{p}$, from Lemma 3.4). We have

- $\text{Prob}(Z_a \geq Z(r)) = \text{Prob}(X_a \geq \lceil r \rceil)$, and

- $\text{Prob}(Z_b^i \geq Z(r)) = \text{Prob}(X_b^i \geq r\sqrt{p/p'})$

The X_b^i are independent random variables which are identically distributed. Let X_b denote any representative of these identical variables. Then,

$$\text{Prob}(Z_b \geq Z(r)) \leq (p/p')\text{Prob}(X_b \geq r\sqrt{p/p'})$$

Inequality (1) thus translates into

$$(p/p')\text{Prob}(X_b \geq r\sqrt{p/p'}) \leq \text{Prob}(X_a \geq \lceil r \rceil)$$

i.e. it suffices to show for $r \geq 3$

$$(p/p')\text{Prob}\left(X_b \geq (r-1)\sqrt{p/p'}\right) \leq \text{Prob}(X_a \geq r)$$

Using the above mentioned Chernoff bound, and a similar inequality on the right, this requirement transforms into

$$\frac{p}{p'}\left(\frac{el}{(r-1)\sqrt{p/p'}p'^3\sqrt{p'}}\right)^{(r-1)\sqrt{p/p'}} \leq \frac{1}{e}\left(\frac{l}{rp^3\sqrt{p}}\right)^r \tag{2}$$

The proof of inequality (2) is omitted here for brevity. It is a purely technical proof, using Assumptions 3 and 4, and the restrictions on r and l.

References

[1] G. Bilardi, F. P. Preparata, "Horizons of Parallel Computation," in: A. Bensoussan, 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] P. Gibbons, "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.

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

[4] T. Heywood and C. Leopold, "Dynamic Simulation of the Hierarchical PRAM Model on Mesh Architectures" (Full version), manuscript, September 1994.

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

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

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

[8] C. P. Kruskal, L. Rudolph, M. Snir, "A Complexity Theory of Efficient Parallel Algorithms", *Theoretical Computer Science*, 71, pp. 95-132, 1990.

[9] T. Leighton, B. Maggs, A. Ranade, S. Rao, "Randomized Routing and Sorting on Fixed-Connection-Networks", *Journal of Algorithms*, Vol. 17, No. 1, pp. 157-205, 1994.

[10] V. Leppänen, "PRAM Computation on Mesh Structures", Dissertation, University of Turku, 1993.

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

[12] S. Sen, “Random Sampling Techniques and Parallel Algorithm Design”, *Synthesis of Parallel Algorithms* Chapter 9, J. Reif, Ed., Morgan-Kaufmann, 1993.

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

[14] P. de la Torre, C. P. Kruskal,“Towards a Single Model of Efficient Computation in Real Parallel Machines”, *Proc. Parallel Architectures and Languages Europe PARLE*, LNCS 505, pp.6-24, 1991