Granularity of parallel memories*

Kurt Mehlhorn
Fachbereich 1o
Universität des Saarlandes
66oo Saarbrücken
West Germany

Uzi Vishkin
Courant Institut
New York
University
251 Mercer Street
New York,
N.Y. 1oo12

A 83/1o

Granularity of parallel memories*

Kurt Mehlhorn
Fachbereich 10
Universität des Saarlandes
6600 Saarbrücken
West Germany

Uzi Vishkin
Courant Institute
New York University
251 Mercer Street
New York, N. Y. 10012

Abstract

Consider algorithms which are designed for shared memory models of parallel computation in which processors are allowed to have fairly unrestricted access patterns to the shared memory. General fast simulations of such algorithms by parallel machines in which the shared memory is organized in modules where only one cell of each module can be accessed at a time are proposed.

The paper provides a comprehensive study of the problem. The solution involves three stages:

(a) Before a simulation, distribute randomly the memory addresses among the memory modules.

(b) Keep several copies of each address and assign memory requests of processors to the 'right' copies at any time.

(c) Satisfy these assigned memory requests according to specifications of the parallel machine.

*A preliminary version of this paper was presented at the 9th Workshop on Graghtheoretic Concepts in Computer Science (WG-83), Fachbereich Mathematic, Universitat Osnabruck, June 1983.

Oct., 83

I. Introduction

Consider algorithms designed for models of parallel computation in which processors have access to a shared memory and parallel machines in which this shared memory is organized in modules where only one cell of each module can be accessed at a time.

The problem of simulating such algorithms on these machines is the problem of granularity of parallel memories (granularity, in short). Every intuitive idea for coping with the granularity problem has to be analyzed for alternate formal settings of assumptions for both the model of parallel computation and the parallel machine. In order to overcome this difficulty we present our main ideas on a setting of assumptions which enables us to simplify the presentation by limiting the discussion to the actual problem which is overcome by our ideas. Extensions of the ideas to alternate models of computation and machines are given later in the paper.

We study ways by which the second parallel machine model below can simulate the first. Both machine models employ p processing elements (PE's or processors) which operate synchronously and N common memory cells. In the first model each processor has access to each of the N cells in each time unit. We forbid only the case where two (or more) processors seek access to the same cell at the same time. This is the Exclusive-Read Exclusive-Write Parallel Random Access Machine (EREWPPAM). It is based on [Lev, Pippenger and Valiant]. Our second model of computation is called Module Parallel Computer (MPC). The common memory of size N is partitioned into m memory modules. Say that at the beginning of a cycle of this model the processors issue R_j requests for addresses located in cells of module j, $0 < j < m-1$. Let $R_{max} = \max \{|R_j| / 0 < j < m-1\}$. Then the requests for each module are queued in some order and satisfied one at a time. So a cycle takes R_{max} time. We assume that immediately after a simulation of a cycle is finished every processor knows it. Figure 1 illustrates the difference between the EREWPRAM and the MPC. The problem of simulating efficiently one cycle of the EREWPRAM by the MPC is taken as the definition of the granularity problem in the next two chapters which include the main contribution of this paper. When N = m the MPC can simulate a cycle of the EREWPRAM in one time unit while when $N > mp$ a naive simulation may result in R_{max} as large as p.

The survey paper [Kuck] emphasizes the importance of the granularity problem. It reports about the considerable attention this problem has received in the literature by mentioning fourteen papers that dealt with it. Most of these papers suggest strategies for partitioning the memory addresses among the modules for algorithms that either have access patterns which are known in advance or have access patterns in successive time units which satisfy some probabilistic assumptions. Our attitude is completely different. We present solutions and analyses for the general problem of simulation. They do not depend on the access behaviour of the algorithms being simulated. This is in sharp contrast to both classes of past research mentioned above. In this spirit [Vishkin and Wigderson] observed that in a few general cases the idea of dynamically changing location of addresses among modules throughout the performance of an algorithm enables efficient simulations utilizing only a moderate number of modules ($m=p$).

Our research is motivated by the Ultracomputer project. The NYU-Ultracomputer group ([Gottlieb et al.]) believes that a machine using 4096 processors and 4096 memory modules will be available by 1990. The MPC represents actually, an abstract Ultracomputer design which idealizes only one point: the interconnection of processors and memory modules. A significant part of this project involves heuristics for the granularity problem. The Ultracomputer is a general-purpose parallel computer that may be used for any parallel algorithm. Our general solutions are, therefore, of particular relevance to its design.

The present paper is similarly motivated by the parallel-design distributed-implementation (PDDI) machine, proposed in [Vishkin 82]. The PDDI machine forms a counterpart to the Ultracomputer which differs from it mainly at the following point. Its interconnection network, between processors and memories, performs well in the worst case while the interconnection network of the Ultracomputer performs well in the average.

Part of this research can also be motivated by some data base applications; where, for instance, there are p processes (transactions), m servers (resources, disks) and N files distributed among the servers, so that each server may serve at most one process at a time (a resource is locked by a transaction). The case where a processor may require only one file at a time readily fits our framework. However, a few alternative assumptions regarding how many files can be required simultaneously by the same process may reflect different circumstances. It might be interesting to investigate which such assumptions fits our framework and possible extensions to others. We do not elaborate on this motivation any more in this paper.

We provide a three stage study of the granularity problem. The ideas of each stage can be applied separately or in conjunction with the others. The first stage is designed to keep us 'out of trouble', in the first place, in the average case. The key idea behind the proposed approach is to utilize universal hasing in the simulating machine. The MPC itself picks at random a hashfunction from an entire class of hashfunctions, instead of a specific hashfunction. This is shown to keep memory contention low. The idea behind the second stage is to keep several copies of each memory address in distinct memory modules. This idea, in conjunction with fast algorithms for picking the 'right' copy of each address request, is shown to decrease memory contention in the worst case, for the less fortunate cases of the first stage. Our above definition of the granularity problem made the third stage somewhat indistinct. In simulations of other models than the EREWPRAM by the MPC or other machines, the problem of simulation is not completely solved by specifying for each address request the module that satisfies it. Problems like scheduling the requests for a module (in case queues are not available) or combining simultaneous requests for the same address in the same module may arise. They are solved in the third stage. Chapters II, III, and IV discuss the respective stages.

We wish to point out a typical difficulty that we had to cope with in all stages. Every solution we suggest is beneficial only if we combine it with an efficient parallel algorithm. By efficient we mean that it is fast and does not use too much local or common memory. Note that since the worst R_{max} that we want to improve is p, our algorithms have to be significantly faster than that.

II 1. A Probabilistic Simulation

In this section we begin to study a simple probabilistic simulation of PRAMs on MPCs. Consider a PRAM with p processing elements and a shared memory of size N. Also consider an MPC with p processing elements, and a shared memory of size N which is divided into m modules. More precisely, let memory module MM_j, $0 \leq j < m$, contain all (physical) addresses a with $0 \leq a < N$ and $a \bmod m = j$.

Our probabilistic simulation is based on universal hashing as introduced by Carter/Wegman. Let H be a subset of S_N, the full set of permutations of $[0...N-1]$. We use elements of H to make the connection between logical and physical addresses. More precisely, we proceed as follows:

Initialization: Choose $h \in H$ at random and store h in every processing element of the MPC. The i-th PE of the MPC will run the same program as the i-th PE of the PRAM to be simulated. We maintain the invariant that cell h(a) of the MPC has the same content as cell a of the PRAM for $0 \leq a \leq N-1$.

Step by Step Simulation:

Let a_i be the (logical) address generated by the i-th PE of the MPC. Apply h to a_i and obtain (physical) address $b_i = h(a_i)$. Issue a request for memory cell b_i. This describes the behavior of the i-th PE of the MPC, $1 \leq i \leq p$. Memory module MM_j, $0 \leq j < m$, collects all requests for cells in MM_j and serves them sequentially. When all requests are served the next cycle of the PRAM is simulated.

Of course, the quality of the simulation described above depends crucially on class H of permutations used in the simulation. Note that the simulation is probabilistic because h is chosen at random from class H. We want

1) H to be small; because every PE needs additional local memory of $O(\log|H|)$ bits (assuming a suitable encoding) to store an element $h \in H$.

2) random elements of H to be easy to generate; because this will hold the cost of the initialization phase small.

3) elements $h \in H$ to be easy to evaluate; because this determines the cost of translating from logical to physical addresses.

4) the length of the queues arising in the simulation to be short; because they essentially determine the quality of the simulation.

We will next study expected queue length in more detail. Let $S = \{a_1, a_2, \ldots, a_p\} \subseteq [0\ldots N-1]$ be a set of p addresses. Let $h \in H$ be a permutation. Define

$$R_j(h,S) = |\{a \in S;\ h(a) \bmod m = j\}|, \quad 0 < j < m,$$

$$R_{max}(h,S) = \max_{0 < j < m} R_j(h,S), \text{ and}$$

$$R_{max} = \max_{\substack{S \subseteq [0\ldots N-1] \\ |S|=p}} \sum_{h \in H} R_{max}(h,S)/|H|$$

$R_j(h,S)$ is the length of the queue in front of memory module MM_j when permutation $h \in H$ is used and set S of addresses is issued by the processors. $R_{max}(h,S)$ is the length of the longest queue in front of any memory module under the same conditions. Next $\sum_{h \in H} R_{max}(h,S)/|H|$ is the expected value of $R_{max}(\ ,S)$. Finally, R_{max} is the worst case of that value taken with respect to all possible sets of p addresses. In other words, R_{max} is the worst case (with respect to addresses) expected (with respect to random elements of H) length of the longest queue.

We want R_{max} to be small. What can we expect? In order to get a feeling for R_{max} we will briefly study a limiting case first: $H = S_N$, the full set of permutations of N elements. Note that class S_N is much too large (NlogN bits local memory would be required in every processor) to be practically useful.

Theorem 1: Let $H = S_N$ and let $m > p$. Then

$$R_{max} < \min(\log m/\log (m/p),\ \log m/\log\log m) + 1$$

Proof: Let $S = \{a_1,\ldots,a_p\} \subseteq [0\ldots N-1]$ be arbitrary. Let $p_{k,j}$ be the probability that at least k elements of S are mapped into memory module j by a random element $h \in S_N$. Then $p_{k,j} < \binom{p}{k}(1/m)^k$ since images of different addresses are independent and uniformly distributed. Let p_k be the probability that at least k elements of S are mapped into some memory module. Then

$$p_k < p_{k,0} + p_{k,1} + \ldots + p_{k,m-1} < \binom{p}{k}(1/m)^{k-1}.$$

Hence

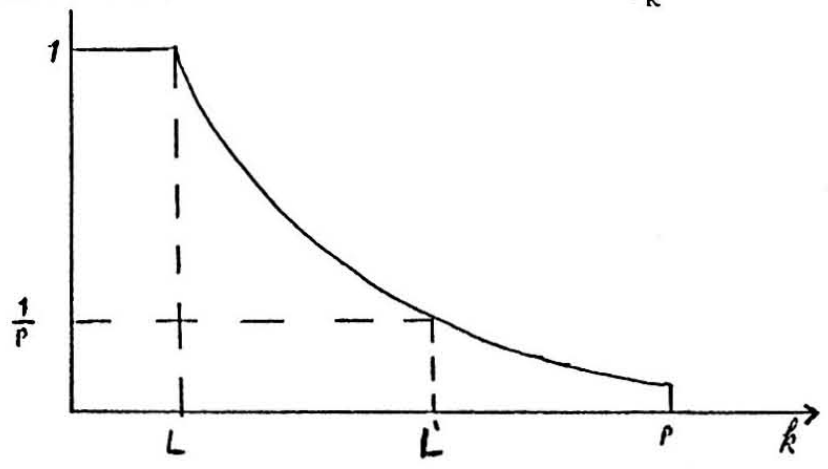
$$R_{max} = \sum_{k\geq 1} p_k < \sum_{k\geq 1} \min(1,\binom{p}{k}(1/m)^{k-1})$$

$$< \sum_{k\geq 1} \min(1,(m/k!)(p/m)^k)$$

Since $(p/m)^k(m/k!) < 1$ for $k > \min(\log m/\log m/p), \log m/\log\log m)$ and decreases exponentially as a function of k for larger k, we have

$$R_{max} < \min(\log m/\log (m/p), \log m/\log\log m) + 1 \quad \bullet$$

We infer from theorem 1 that $R_{max} < 1 + \log m/\log\log m$ if $m > p$ and $H = S_N$ and that $R_{max} < 2 + 1/\varepsilon$ if $m = p^{1+\varepsilon}$ and $H = S_N$. Unfortunately, S_N is too large for our purposes. However, close inspection of the proof of theorem 1 suggests methods for finding smaller classes of H which yield essentially the same value of R_{max} as $H = S_N$. The diagram below shows a plot of p_k as a function of k. Our bound on p_k decreases exponentially for



$k > L := \min(\log m/\log(m/p), \log m/\log \log m)$. In particular $p_k < 1/p$ for values of k exceeding L' where L' is only slightly larger than L. Quantity R_{max} is the area under curve p_k. The following simple observation is crucial for the sequel: quantity R_{max} increases by at most one if p_k were equal to $1/p$ for $k > L'$. Thus the bound on R_{max} shown in Theorem 1 stays essentially true if we only know that $p_{k,j} < \binom{p}{k}(1/m)^k$ for $k < L'$ and that $p_{k,j}$ is non-increasing as a function of k. We will explore this approach in the next section.

II 2. On the Expected Length of the Longest Chain in Universal Hashing.

Universal hashing was introduced by Carter/Wegman. They showed that very small classes of hash functions suffice to obtain an expected case behavior which is similar to the one of ordinary hashing. We show in this section that universal hashing is also competitive with respect to expected worst case behavior.

Def [CW]. Let $c \in \mathbb{R}$, $h \in \mathbb{N}$, $N \in \mathbb{N}$, $m \in \mathbb{N}$. A multiset $H \subseteq \{h;\ h:[0\ldots N-1] \to [0\ldots m-1]\}$ is c strongly k universal if for all $a_1,\ldots,a_k \in [0\ldots N-1]$, pairwise distinct, and all $b_1,\ldots,b_k \in [0\ldots m-1]$,

$$|\{h \in H;\ h(a_i) = b_i \text{ for } 1 \leq i \leq k\}| \leq c \cdot |H|/m^k \circ$$

As in the preceeding section, let $p \in \mathbb{N}$ and let $S \subseteq [0\ldots N-1]$, $|S| = p$. For $h \in H$ let

$$R_{max}(h,S) = \max_{0 \leq j < m} |\{a \in S;\ h(a) = j\}|,$$

and let

$$R^p_{max} = \max_{\substack{S \subseteq [0\ldots N-1] \\ |S|=p}} \sum_{h \in H} R_{max}(h,S)/|H|$$

We are now in a position to state the observation following Theorem 1 as

Theorem 2: Let H be a c strongly k universal multiset of functions from $[0\ldots N-1]$ to $[0\ldots m-1]$. Then

$$R^p_{max} \leq k + cpm(p/m)^k/k!$$

for all $p \in \mathbb{N}$.

Proof: Let S $[0\ldots N-1]$, $|S| = p$ be arbitrary. Let $p_i(S)$ be the probability that $R_{max}(h,S) > i$, i.e. $p_i(S) = |\{h \in H;\ R_{max}(h,S) > i\}|/|H|$. Then $1 > p_1 > p_2 > p_3 > \ldots$ and

$$R^p_{max} = \max_{\substack{S \subsetneq [0\ldots N-1] \\ |S|=p}} \sum_{i=1}^{p} p_i(S)$$

$$k + p \max_{\substack{S \subsetneq [0\ldots N-1] \\ |S|=p}} p_k(S) .$$

Next observe that $p_k(S) < p_{k,0}(S) + p_{k,1}(S) + \ldots + p_{k,m-1}(S)$ where $p_{k,j}(S)$ is the probability that at least k elements of S are mapped onto j. For fixed $a_1,a_2,\ldots,a_k \in S$ (pairwise distinct) we have

$$|\{h \in H;\ h(a_\ell) = j \text{ for } 1 < \ell < k\}| < c|H|/m^k$$

since H is c strongly k universal. Hence $p_{k,j}(S) < c\binom{p}{k}/m^k < c(p/m)^k/k!$ and $p_k(S) < cm(p/m)^k/k!$ for all S. ●

Before we give some examples of strongly universal classes we recall the following lemma.

Lemma (Carter/Wegman). If $H \subsetneq \{h; h: [0\ldots N-1] \to [0\ldots N-1]\}$ is c strongly k universal and $r: [0\ldots N-1] \to [0\ldots m-1]$ is such that $|r^{-1}(j)| < \lceil N/m \rceil$ for all j, $0 < j < m$, then multiset

$$\hat{H} = \{r \circ h; h \in H\}$$

is $\hat{c}$ strongly k universal where $\hat{c} = (m \lceil N/m \rceil/N)^k c$.

Proof: Let $a_1,\ldots,a_k \in [0\ldots N-1]$ be pairwise distinct and let $b_1,\ldots,b_k \in [0\ldots m-1]$. Then there are at most $\lceil N/m \rceil^k$ tuples $c_1,\ldots,c_k \in [0\ldots N-1]$ with $r(c_i) = b_i$ for $1 < i < k$. For every such tuple $(c_1,\ldots,c_k)$ there are at most $c|H|/m^k$ functions $h \in H$ with

$h(a_i) = c_i$ for $1 \le i \le k$. Thus $\hat{H}$ is $\hat{c}$ strongly k universal with $\hat{c} = ([N/m]/(N/m))^k c$. ∎

We will next give some examples. Applications 1 and 2 are based on the fact that there are small doubly and triply transitive permutation groups. A set H of permutations of set X is transitive if for all $a,b \in X$ there is $h \in H$ such that $h(a) = b$. It is doubly (triply) transitive if it contains a permutation replacing any whatever given ordered pair (triple) of elements in X by any whatever ordered pair (triple) of elements in X. The reader may consult [Carmichael, Chapter VI] for a detailed discussion. Application 3 uses the fact that a polynomial of degree k is fixed by its values at k+1 points, i.e. a random polynomial of degree k maps a set of k+1 points into a random set.

Application 1. Let N be a prime and let $H_1 = \{h; h(x) = (ax+b) \bmod N$ for some $a,b \in [0..N-1]$, $a \ne 0\}$, let $r(x) = x \bmod m$ and let $\hat{H}_1 = \{r \circ h; h \in H_1\}$. Since for every $x_1,x_2,y_1,y_2 \in [0...N-1]$, $x_1 \ne x_2$, $y_1 \ne y_2$ there is exactly one pair $a,b \in [0...N-1]$ such that $y_1 = ax_1 + b \bmod N$ and $y_2 = ax_2 + b \bmod N$ class H_1 is 1 strongly 2 universal and hence $\hat{H}_1$ is $(m[N/m]/N)^2$ strongly 2 universal. For $m = p^3$ we obtain $R^p_{max} \le 2 + 4/2! = 4$; note that $(m[(N/m)]/N) \le 2$. Finally, observe that H_1 is a set of permutations.

Application 2. Let N be a prime. For $a,b,c,d \in [0...N-1]$ with $ad-bc \ne 0 \bmod N$ define $h_{a,b,c,d}: [0...N-1] \cup \{\infty\} \to [0...N-1] \cup \{\infty\}$ by

$$h_{a,b,c,d}(x) = \begin{cases} a/c & \text{if } x = \infty \\ \infty & \text{if } x = d/c \\ (ax-b)/(cx-d) \bmod N & \text{otherwise} \end{cases}$$

Note that division is well-defined since the integers mod N are a field. Also the first two clauses in the definition coincide if $c = 0$. It is known (a proof can be found in [Carmichael, Chapter VI]) that

$h_{a,b,c,d}$ is a permutation and that set $H_2 = \{h_{a,b,c,d};\ a,b,c,d \in [0...N-1],\ ad-bc \neq 0\}$ is a triply transitive group of permutations, i.e. for all x_1,x_2,x_3 and $y_1,y_2,y_3 \in [0..N-1] \cup \{\infty\}$, x_1,x_2,x_3 and y_1,y_2,y_3 pairwise distinct, there is exactly one $h \in H_2$ with $\hat{h}(x_i) = y_i$ for $1 < i < 3$. Thus H_2 is 1 strongly 3 universal and hence $H_2 = \{roh;\ h \in H_2\}$ and $r(x) = x \bmod m$ is 8 strongly 3 universal. Note that $(m[N/m])^3 < 8$. Thus for $m = p^2$ we have $R^p_{max} < 3 + 8/3! = 13/3$. Finally, we want to mention that $h_{a,b,c,d}(x)$ can be evaluated efficiently using Euklid's algorithm. (cf. [AHU], p. 300-302). More precisely, $h_{a,b,c,d}(x)$ can be computed in $O(\log N)$ arithmetic steps.

Application 3: Let N be a prime, let k be an integer and let $H_3 = \{h;\ h(x) = \sum_{0<i<k} a_i x^i \bmod N$ for some $a_i \in [0...N-1]$ and $a_i \neq 0$ for some $i > 1\}$ be the set of all polynomials of degree at most k-1. Since for every $x_1,...,x_k$ and $y_1,...,y_k \in [0...N-1]$, $x_1,...,x_k$ pairwise different, there is at most one non-trivial polynomial h of degree at most k-1 with $h(x_i) = y_i$ for $1 < i < k$ we conclude that H_3 is 1 strongly k universal. Also $\hat{H}_3 = \{roh;\ h \in H_3\}$ is $\hat{c}$ strongly k universal with $\hat{c} = (m\ [N/m]/N)^k < (1 + m/N)^k$. Thus for $m = p^{1 + 2/(k-1)}$ we have

$$R^p_{max} < k + \hat{c}\ p^{2+2/(k-1)}\ p^{-[2/(k-1)]k}/k!$$

$$< k + \hat{c}/k!$$

Another interesting choice is $k = 3 \ln p/\ln \ln p$ and $m = p$. Then

$$R^p_{max} < k + \hat{c}\ p^2/k! = O(\ln p/\ln \ln p).$$

Application 3 should be compared with Theorem 1. It states that if the class of hash functions is restricted to the set of all polynomials of degree at most $3 \ln p/\ln \ln p$ (note that there are only $N^{3\ln p/\ln \ln p}$ such functions) then the expected worst case behavior is as good as if we use all hash functions (and there are N^N of those). Similarly, if $m = p^{1+\epsilon}$ and $k = 1 + 2/\epsilon$ for some $\epsilon > 0$ and if the class of hash functions is restricted to the set of polynomials of degree at most $k - 1$ (note that there are only $N^{k-1} = N^{2/\varepsilon}$ such functions) then the expected worst case behavior is almost as good (except for a multiplicative factor of two) as if we use all hash functions (and here are N^N of those). Unfortunately, class H_3 is not a class of permutations and hence class H_3 cannot be directly used in simulating PRAMs by MPCs.

Example 4: Example 4 is slightly more difficult to treat; it is not a direct application of Theorem 2. Let $N = 2^n$ and $m = 2^b$. Then $[0..N-1]$ can be identified with the bit vectors of length n, i.e. $[0..N-1] \simeq \{0,1\}^n$. The bit vectors of length n form a vector space of dimension n over the field of two elements. In a vector space we can use linear transformations to map any set of (linearly independent) vectors into any other set of vectors. This suggests to consider $H_4 = \{h;\ h: \{0,1\}^n \to \{0,1\}^n$ and $h(x) = Mx$ for some n by n (0,1) - invertible matrix $M\}$.

Here matrix multiplication is over the field of two elements. It is important for the application in Section II 3 that H_4 is a set of permutations. As before, let $r(x) = x \bmod m$ and $\hat{H}_4 = \{r\circ h;\ h \in H_4\}$.

Before we analyze the behavior of class H_4 we show that elements of $\hat{H}_4$ are easy to find by a probabilistic algorithm. More precisely, we show that a significant fraction of all (0,1)-matrices is invertible.

Lemma 1: $|H_4| = \prod_{0 \le i \le n-1} (2^n-2^i) > 2^{(n^2)}/e^{7/5}$

Proof: Lemma 1 is well known and can be found for example in E. Artin: Geometric Algebra. We include the very short proof for the sake of completeness. Choose M column by column. When columns 1 to i have been chosen (and are linearly independent) then column i+1 must be different from all linear combinations of columns 1 through i. Hence there are 2^n-2^i choices for column i+1. Thus

$$|H_4| = (2^n-1)(2^n-2)\dots(2^n-2^{n-1})$$

$$= 2^{(n^2)} \prod_{i=1}^{n} (1-2^{-i})$$

Next observe that

$$\prod_{i=1}^{n} (1-2^{-i} > e^{\sum_{i=1}^{n} \ln(1-2^{-i})}$$

$$> e^{-(7/5)\sum_{i=1}^{n} 2^{-i}}$$

since $\ln(1-x) > -7x/5$ for $0 < x < 1/2$

$$> e^{-7/5} \qquad \bullet$$

Lemma 1 shows that at least 25% of all (0,1)-matrices are invertible. Hence invertible (0,1)-matrices can be found by taking a few random (0,1)-matrices and checking for singularity by Gaussian elimination. We will next show that H_4 is a good class of hash functions.

For $x_1,x_2,\ldots,x_k$ a set of vectors we write $\dim(x_1,x_2,\ldots,x_k)$ to denote the dimension of the space spanned by $x_1,\ldots,x_k$. Also, if $x \in \{0,1\}^n$ we write x_b for the b-dimensional vector consisting of the last b components of x.

<u>Lemma 2</u>: Let $a_1,\ldots,a_k \in \{0,1\}^n$, pairwise different. Let $d = \dim(a_1-a_2,\ldots,a_1-a_k)$. Then

$$|\{M;\ (Ma_1)_b = (Ma_2)_b = \ldots = (Ma_k)_b\}| < 2^{(n^2)}/m^d$$

<u>Proof</u>: If $(Ma_1)_b = \ldots = (Ma_k)_b$ then $(M(a_1-a_i))_b = 0$ for $2 < i < k$. Assume w.l.o.g. that $a_1-a_2,\ldots,a_1-a_{d+1}$ are linearly independent. Let X be the n by d matrix whose columns are $a_1-a_2,\ldots,a_1-a_{d+1}$. Let X_1 be the first d rows of X and let X_2 be the remaining n-d rows. Assume w.l.o.g. that X_1 is non-singular. Let M′ be the matrix consisting of

the first b rows of M. Let M_1 consist of the first d columns of M' and let M_2 consists of the remaining n-d columns. Then

$$M_1X_1 + M_2X_2 = 0$$

or

$$M_1 = - M_2X_2X_1^{-1}$$

Thus M_1 is determined by the choice of M_2 and hence there are at most $2^{n^2-bd} = 2^{n^2}/m^d$ matrices M such that $(Ma_1)_b = \dots = (Ma_k)_b$. Recall that $m = 2^b$. ●

Lemma 3: Let $S = \{a_1,\dots,a_p\} \subseteq \{0,1\}^n$. Let $t_{k,\ell}$ be the number of subsets of S of cardinality k and dimension ℓ. Then

$$t_{k,\ell} < \binom{p}{\ell}\binom{2^\ell}{k-\ell}$$

In particular, $t_{k,\ell} = 0$ for $2^\ell + \ell < k$.

Proof: Any subset of cardinality k and dimension ℓ can be written as a set of ℓ linearly independent elements plus a set of $k-\ell$ vectors which are linear combinations of the first ℓ elements. There are only $\binom{p}{\ell}$ choices for the first set and only $\binom{2^\ell}{k-\ell}$ choices for the second set. ○

We are now in a position to estimate the behavior of class $\hat{H}_4$. Let $S = \{a_1,a_2,\dots,a_p\} \subseteq \{0,1\}^n$ be arbitrary. As above, let

$$p_k = \text{prob}(R_{max}(h,S) < k) = |\{h \in \hat{H}_4 \ ; \ R_{max}(h,S) > k\}|/|\hat{H}_4|$$

Lemma 4: a) $p_1 > p_2 > p_3 > \dots$

b) $p_k = cm \sum_{\ell > 0} t_{k,\ell}\, m^{-\ell}$ where $c = e^{7/5}$

c) $p_k < 2cm(p2^k/m)^{\log k-1}$ for $2 < k < \log m/p-1$

Proof: a) obvious

b) We have

$$p_k < \sum_{\ell>0} \sum_{\substack{A\subseteq S \\ |A|=k \\ \dim A=\ell}} |\{h \in \hat{H}_4;\ h \text{ maps all points of } A \text{ into the same location}\}|/|\hat{H}_4|$$

$$< \sum_{\ell>0} \sum_{\substack{A\subseteq S,\ |A|=k \\ \dim A=\ell}} c/m^{\ell-1}$$

by Lemma 1,2 and the observation that $\dim A=\ell$, $A=\{a_1,\dots,a_k\}$ implies $\dim(a_1-a_2,\dots,a_1-a_k) > \ell-1$

$$< c \sum_{\ell>0} t_{k,\ell}\ m^{1-\ell}$$

by the definition of $t_{k,\ell}$. This proves part b).

c) For $k > 2$ we have $t_{k,\ell} = 0$ for $\ell < \log k-1$. Hence by part b) and lemma 3.

$$p_k < cm \sum_{\log k-1<\ell<k} \binom{p}{\ell}\binom{2^\ell}{k-\ell} m^{-\ell}$$

$$< cm \sum_{\log k-1<\ell<k} (p2^k/m)^\ell$$

$$< cm\ (p2^k/m)^{\log k-1} \sum_{\ell>0} (p2^k/m)^\ell$$

$$< 2cm(p2^k/m)^{\log k-1}$$

since $p2^k/m < 1/2$ for $k < \log(m/p) - 1$. ∘

Lemmas 1 to 4 directly lead to

Theorem 3: Let $N = 2^n$, $m = 2^b$ and let $\hat{H}_4$ be defined as above.

a) there is a function $f\colon \mathbb{N} \to \mathbb{R}$, $f(p) = O(p^{\varepsilon})$ for all $\varepsilon > 0$ such that for $m \geq pf(p)$ we have:

$$R^p_{max} \leq 20 \log p/(\log(10 \log p))^2 + O(1)$$

b) Let $\varepsilon > 0$. For $m = p^{1+\varepsilon}$ we have:

$$R^p_{max} \leq O(1)$$

Proof: Note first that

$$R^p_{max} = \sum_{1 \leq k \leq p} p_k \leq k_1 + pp_{k_1}$$

for every k_1, $1 \leq k_1 \leq p$. From Lemma 4,c we conclude further

$$R^p_{max} \leq k_1 + 2\, cmp(p2^{k_1}/m)^{\log k_1 - 1}$$

where $c = e^{7/5}$

a) Let $f(p) = \max(2^{2^{10}}, 2^{10} \log p/\log(10 \log p))$.

Then $f(p) = O(p^{\varepsilon})$ for all $\varepsilon > 0$. Let $k_1 = \log f(p)/\log \log f(p)$.

Then $k_1 \leq (\log f(p))/10$ and hence

$$p2^{k_1}/m \leq p\ f(p)^{1/10}/p\ f(p) = f(p)^{-9/10}$$

Also $\log k_1 - 1 = \log \log f(p) - \log \log \log f(p) - 1 \geq (\log \log f(p))/2$ since $\log \log f(p) \geq 10$. Thus

$$pp_{k_1} < 2cmp\ f(p)^{-9 \log \log f(p)/20}$$

$$< 2c\ p^2 f(p)\ f(p)^{-9 \log \log f(p)/20}$$

$$< 2c\ 2^{2 \log p + \log f(p) - 9 \log f(p) \log \log f(p)/20}$$

$$< 2c\ 2^{2 \log p - 7 \log f(p) \log \log f(p)/20} \quad \text{since } \log \log f(p) > 10$$

$$= O(1) \quad \text{since } 7 \log f(p) \log \log f(p)/20 > 2 \log p$$

for sufficiently large p. Thus

$$R^p_{max} < \log f(p)/\log \log f(p) + O(1)$$

$$< 20 \log p/(\log (10 \log p))^2 + O(1)$$

b) Let $m = p^{1+\varepsilon}$. Then

$$R^p_{max} < k_1 + 2cp^{2+\varepsilon}(2^{k_1}/p^{\varepsilon})^{\log k_1 - 1}$$

for all k_1, $1 < k_1 < p$. Let k_1 be such that $2+\varepsilon = \varepsilon(\log k_1 - 1)$, i.e. $k_1 = 2^{2+2/\varepsilon}$. Then

$$R^p_{max} < k_1 + 2c\ p^{2+\varepsilon - \varepsilon(\log k_1 - 1)}\ 2^{k_1(\log k_1 - 1)}$$

$$= 2^{2+2/\varepsilon} + 2c\ 2^{(1+2/\varepsilon)2^{2+2/\varepsilon}}$$

$$= O(1) \qquad \bullet$$

Theorem 3 states that the performance of class $\hat{H}_4$ is "close" to the performance of the full set of permutations. More precisely, if $m = p^{1+\varepsilon}$ then $R^p_{max} = O(1)$ in both cases. Of course, the constants involved are dramatically different. Also $R^p_{max} = O(\log p/\log \log p)$ can be achieved by class $\hat{H}_4$ for $m = p\ f(p)$ where $f(p)$ grows slower than any root of p. If the full set of permutations is used there $RP_{max} = O(\log p/\log\log p)$ can be achieved for $m = p$.

Why doesn't class H_4 behave well in the case $m = p$? The answer comes in two parts. Firstly, the bounds derived in lemmas 3,4 and hence in Theorem 3 are not sharp. Secondly and more significantly, class H_4 has a certain deficiency. Multiplying by a random, invertible matrix hashes a set of <u>independent</u> vectors very well, however, it does not do that well on a set of linearly dependent vectors. A variant of Lemma 1 can be used to show that the expected dimension of a <u>random</u> set $a_1,\dots,a_n$ of vectors in $\{0,1\}^n$ is very large, namely $n - O(1)$. Unfortunately, this observation is of no use since we are dealing with worst case behavior.

We will now discuss a possibility to overcome this problem. As above, let $N = 2^n$, $m = 2^b$. Assume also that $q = N-c$ for some small constant c is a prime. For $a \in [1\dots q-1]$ let

$$h_a: [0\dots 2^n-1] \to [0\dots 2^n-1]$$

be defined by

$$h_a(x) = \begin{cases} (ax) \bmod q & \text{if } x < q \\ x & \text{if } q \le x \le N-1 \end{cases}$$

Note that h_a is a permutation. We conjecture

<u>Conjecture</u>: Let $a_1,\dots,a_p \in \{0,1\}^n \equiv [0\dots 2^n-1]$ be arbitrary. For $a \in [1\dots q-1]$ let $\dim_a$ be the dimension of set $h_a(a_1),\dots,h_a(a_p)$. Then

$$\sum_{1 \le a < q} \dim_a/(q-1) \ge \min(n,p)-f$$

for some small constant f. Moreover,

$$|\{a;\ 1 \le a < q \text{ and } \dim_a = \min(n,p) - f - i\}| \le qp^{-i}$$

for all $i \ge 1$.

Informally, the conjecture states that a random function h_a turns a set of p vector $a_1,\ldots,a_p$ into a (nearly) independent set of vectors. Moreover, it is very unlikely that the dimension of the resulting set of vectors is much less than the expected dimension.

Consider class

$$\hat{\hat{H}}_4 = \{h\colon \{0,1\}^n \to \{0,1\}^b ;\ h(x) = (M\cdot h_a(x)) \bmod 2^b$$

for some a, $1 \le a < q$, and some invertible n by n (0,1)-matrix.}.

We next show that class $\hat{\hat{H}}_4$ is a very good class of hash functions provided that the conjecture is true.

<u>Theorem 4</u>: Let $\hat{\hat{H}}_4$ be defined as above and let $m \ge 2p$. If the conjecture above is true then

$$R^p_{\max} \le k + 3cm\, p^{f+1}/k!$$

for all k, $f < k \le \min(n,p)$. Here $c = e^{7/5}$.

<u>Proof</u>: Let $S = \{a_1,\ldots,a_p\} \subseteq [0..N-1]$. As above, let $p_k = \operatorname{prob}(R_{\max}(h,S) \ge k)$. Then

$$p_k = \sum_{a=1}^{q-1} |\{h \in \hat{H}_4 ;\ h \text{ maps at least k points of } h_a(S) \text{ into the same location}\}|/|\hat{\hat{H}}_4|$$

$$< \sum_{a=1}^{q-1} \sum_{\ell>0} \sum_{\substack{A \subsetneq S \\ |A|=k \\ \dim h_a(A)=\ell}} |\{h \in \hat{H}_4 \,;\, h \text{ maps all points of } h_a(A) \text{ into the same location}\}| / |\hat{\hat{H}}_4|$$

$$< \sum_{a=1}^{q-1} \sum_{\ell>0} \sum_{\substack{A \subsetneq S \\ |A|=k \\ \dim h_a(A)=\ell}} cm^{1-\ell}/q$$

by Lemma 2

$$= \sum_{\substack{A \subsetneq S \\ |A|=k}} \sum_{\ell>0} |\{a;\ \dim h_a(A) = \ell\}| cm^{1-\ell}/q$$

$$< \sum_{\substack{a \subsetneq S \\ |A|=k}} \sum_{\ell=0}^{\min(n,k)-f-1} cm^{1-\ell} p^{-(\min(n,k)-f-\ell)}$$

$$+ \sum_{\substack{A \subsetneq S \\ |A|=k}} cm^{1-(\min(n,k)-f)}$$

$$< 2c\binom{p}{k} m \cdot p^{f-k} + c\binom{p}{k} m^{1+f-k}$$

since $k < n$ and $m > 2p$ by assumption

$$< 3c\binom{p}{k} m\, p^{f-k} = 3cmp^f/k!$$

since $m > 2p$ and $k > f$ by assumption. The proof is now completed since $R^p_{max} < k + pp_k$ for all k. ∘

Theorem 4 has an interesting consequence. Assume that $m = 2p$ and $n > \log p$. The latter assumption is certainly realistic. Let $k = (f+2) \log p/\log \log p$. Then $R^p_{max} = (f+2) \log p/\log \log p + 6$. This is basically the same behavior as the behavior of the full class of permutations; cf. Theorem 1.

II 3. Probabilistic Simulations Revisited.

We will now apply the results of Section II 2 to the probabilistic simulation described in Section II 1. We assume throughout this section that operations on addresses, like multiplying by an integer, take unit time.

Theorem 1: Let $m = p^3$. Then a $T(n)$ - time bounded PRAM with p PE's and N memory cells, can be simulated by a randomiced MPC with p PE's and m memory modules and total memory of size N+2p in time $O(T(n))$.

Proof: We use the simulation as described in Section II 1 except that permutation π is chosen from class H_1. Every processor needs two additional storage cells to store π. Also one step of the PRAM takes expected time O(1) on the MPC. ○

Theorem 2: Let $m = p^2$. Then a $T(n)$ time bounded PRAM with p PE's and N memory cells can be simulated by a randomized MPC with p PE's m memory modules and total memory size N+4p in time $O(T(n) \log N)$.

Proof: Replace H_1 by H_2 in the proof of Theorem 1 and observe that the functions in H_2 can be evaluated in time $O(\log N)$. ○

Theorem 3: a) Let $m = p$. Then a $T(n)$ time bounded PRAM with p PE's and N memory cells can be simulated by a randomized MPC with p PE's, m memory modules and total memory size $(N+p)\log p$ in time $O(T(n) \log p)$.
b) Let $\varepsilon > 0$ and let $m = p^{1+\varepsilon}$. Then a $T(n)$ time bounded PRAM with p PE's and N memory cells can be simulated by a randomized MPC with p PE's, m memory modules and total memory size $(1 + 2/\varepsilon)(N+p)$ in time $O(T(n))$.

Proof: a) We use the simulation as directed in Section II 1 except that π is chosen from class H_3 with $k = 3 \log p/\log \log p$. Then every processor requires $O(\log p)$ cells to store π. There is one additional problem now: class H_3 not a class of permutations. Rather $\pi \in H_3$ might map up to $\log p/\log \log p$ distinct points into the same cell. It can certainly map no more distinct points into the same cell since $\pi \in H_3$

is a nontrivial polynomial of degree at most $\log p/\log\log p$. Therefore the simulating MPC has $\log p/\log\log p$ copies of every memory cell. A memory access is made by sending the original PRAM address a and the modified address $\pi(a)$ to the appropriate memory module. We can then build up a balanced tree (say) for all addresses which are mapped to the same address by π. Thus access time within a memory module might be as large as $O(\log\log p)$. Also the expected number of concurrent accesses to the same module is $O(\log p/\log\log p)$ by Section II 2, application 3 and hence it takes an expected number of $O(\log p)$ MPC-steps to simulate our PRAM step. Also evaluation of a hash function in H_3 takes $O(\log p)$ steps.
b) The proof is completely analogous to the proof of part a). We choose $k = 1 + 2/\varepsilon$. Then at most $2/\varepsilon$ distinct points are mapped into the same cell. Thus totoal memory size is $(1 + 2/\varepsilon)(N + p)$ and it takes an expected number of $O(2/\varepsilon \log 2/\varepsilon)$ MPC steps to simulate one PRAM step. ●

Theorem 4: a) There is a function $f\colon \mathbb{N} \to \mathbb{R}$ with $f(p) = O(p^{\varepsilon})$ for all $\varepsilon > 0$ such that: if $N = 2^n$, $m = 2^b > p\,f(p)$ then a $T(n)$ time bounded PRAM with p PE's and memory size N can be simulated on a randomized MPC with p PE's, m memory modules and total memory size $N + p \log N$ in time $O((\log N)^3 + T(n)(\log p/(\log\log p)^2 + \log N))$.

b) Let $\varepsilon > 0$ be fixed. If $m = p^{1+\varepsilon}$ then the time bound reduces to $O\big((\log N)^3 + T(n)\log N\big)$

Proof: Replace H_1 by H_4 in the proof of Theorem 1 and observe that an element of H_4 can be stored in $\log N$ words of length $(\log N)$ each. Thus every processor needs $\log N$ additional memory cells. A random element of H_4 can be chosen as follows. Generate $\log N$ random bitstrings of length $\log N$ each. (in time $O(\log N)^2$) and check whether the (0-1) matrix M generated in this way is invertible. This takes time $O((\log N)^3)$ on a sequential machine. Also $O(1)$ tries suffice on the average. Thus choosing a random element of H_4 takes time $O((\log N)^3)$. Next observe that it takes time $O(\log N)$ to multiply an log N by log N matrix by a vector. The time bounded is now an easy consequence of Theorem 3 of Section II 2. •

We do not feel that theorems 1 to 4 are best possible. They complement each other in that they optimize different parameters of the problem. Theorems 1, 2 and 4 present solutions with only a. very moderate increase in total memory size (O(1) additional cells per processor in theorems 1 and 2 and O(log N) in Theorem 4) and only a moderate increase in running time (a multiplicative factor of O(1) in Theorem 1 and O(log N) in theorems 2 and 4). However, all three theorems require a non-linear number of memory modules thus increasing the size of the interconnection network. Theorem 4a is our best result in that respect.

On the other hand, both parts of Theorems 2 provide us with solutions with a small number of memory modules (p in part a and $p^{1+\epsilon}$ in part b) and modest increases of running time (O(log p)) in part a and O(1) in part b). However, both parts force us to increase total memory size considerably.

Theorem 4 of the previous section has the potential (if the conjecture were true) of combining most advantages of the other schemes. With only two memory modules per processor and only log N additional memory cells per processor it achieves a slowdown of O(log p/log log p).

III. Efficient Deterministic Simulations

Say that each of the N addresses is contained in exactly c of the m memory modules for some integer c(c>1). Each such distribution of addresses is called a c-partitioning.

We may break each cycle of the EREW PRAM into two halves: one includes all read instructions from the common memory, while the other includes all the write instructions of the cycle. This enables us to classify the cycles into reading cycles and writing cycles without multiplying the running time by more than a factor of two. We argue, in the paper, that the worst case time for simulating writing cycles worsens only a little while the worst case time for simulating reading cycles improves a lot. Many simulations of programs for the

Ultracomputer have been run. The ratio between read instructions and write instructions that relate to the common memory was around 8:1. This fact is important since in case a store instruction into the common memory is executed we need access to all copies of the memory address, while in case a fetch instruction is executed one of the copies would suffice. See more on writing cycles in the last chapter. Our main concern is to study the advantages and limitations of the copying approach for simulating reading cycles. Our analysis applies also to input addresses (which are only read).

We start this chapter by studying some limitations of the copying approach. Then, a specific c-partitioning is proposed. For this c-partitioning we give upper bounds for the optimal R_{max} achievable. We include also two sections that discuss how to compute efficiently good assignments of address requests to modules. The first of these sections deals with polynomial-time sequential algorithms for computing optimal assignments, namely with minimum R_{max}. The second section suggests fast parallel algorithms that give low (but not necessarily minimum) R_{max}. For the purpose of fast simulation we only need algorithms of the second kind. The proposed c-partitioning is very efficient when this stage is applied separately. However, an alternative c-partitioning which combines well with the first stage is considered subsequently. The last section of this chapter demonstrates that a number of copies c is useful when $\binom{m}{c}$ is much larger than N.

Note that the number of simultaneous requests for memory addresses is $\leq$ p. In order to use one parameter less, and thereby simplify the presentation, we consider the most difficult case only; i.e., where this number is p. It will be straightforward to extend our results to cases where this number is < p.

III 1. Lower Bounds.

Assume that some c-partitioning is given. Problem: Find a lower bound for the optimal worst case time delay (R_{max}) for any p reading requests for memory addresses.

Let $1 \leq i_1 < i_2 \ldots < i_k \leq m$ be k modules and $A_{i1,i2,\ldots,ik}$ be the set of all memory addresses such that all their c copies are contained in memory modules $i_1, i_2, \ldots, i_k$. We are looking for a (very unfortunate) set of p memory addresses such that all their copies are contained in a minimum number of modules. More formally, we are looking for a minimum size subset of modules $\{i_1,i_2,\ldots,i_k\}$ such that $\# A_{i1,i2,\ldots ik} > p$.

Claim. The equation $\sum \# A i_1,i_2,\ldots,i_k = \binom{m-c}{k-c} N$ holds for any k, $c < k < m$. The summation on the left hand side is on all $\binom{m}{k}$ subsets of k modules. The right hand side gives an alternative evaluation which is based on the fact that each memory address is contained in c modules; fixing k-c additional modules (in any of the $\binom{m-c}{k-c}$ possibilities) gives a subset of k modules containing this address. This completes the proof of the claim.

Hence, there is at least one subset of k modules that contains

$$\binom{m-c}{k-c} N/\binom{m}{k} = k\ldots(k-(c-1))N/(m\ldots(m-(c-1)))$$

elements. We are looking for the minimum k such that

$$(1) \qquad p < k\ldots(k-(c-1))N/(m\ldots(m-(c-1))).$$

Denote this k by K_p. This implies that

OBSERVATION 1. $R_{max} > p/K_p$.

If $N = \binom{m}{c}x$ for some integer x, then from inequality (1) we get $p < k\ldots(k-(c-1))x/c!$ or $p < \binom{k}{c}x$. This implies

OBSERVATION 2. For $N = \binom{m}{c}x$, $R_{max} > p/K_p$ where K_p is the smallest k satisfying $p < \binom{k}{c}$.

It is shown later that the last lower bound meets exactly an upper bound on R_{max} for a specific c-partitioning that we propose. Therefore, we delay presentation of explicit evaluations until this later discussion.

III 2. The Proposed c-partitioning

Our suggested c-partitioning is simple. For $N < \binom{m}{c}$ and an address i, $0 < i < N$, take the i-th subset of c modules (out of m in the lexicographic order) and put a copy of address i in each of the c modules. If $(x-1)\binom{m}{c} < N <= x\binom{m}{c}$, for some integer x, then partition the addresses into x approximately equal subsets (layers) and fix the c-partitioning of each of the x layers separately as for $N < \binom{m}{c}$.

Remark: For an address i it takes $O(c^2)$ time to compute the subset of c modules containing its copies. Clearly, $i_1' = i \pmod{([N/\binom{m}{c}]))}$ is the serial number of i in its layer. The minimum $i_1 (> c)$ such that $\binom{i_1}{c} > i'$ implies that module number $i_1 - 1$ contains a copy of address i. The computation of i_1, takes $O(c)$ time. This is because we get a constant difference approximation to i_1 by Stirling formula and explicit presentation of i_1. Denote $i_2' = i_1 - \binom{i_1 - 1}{c}$. The minimum $i_2 (> c-1)$ such that $\binom{i_2 - 1}{c-1} > i_2'$ implies that module number $i_2 - 1$ contains a copy of address i. This takes $O(c-1)$ time; and so on.

III 3. Upper Bounds

Theorem 1. Let t be an integer, $N = x\binom{m}{c}$, $S_f = \min\{s / \binom{s}{c}x > (t-1)s+1\}$ and $r = S_f(t-1)+1$. If $S_f < m$ then: (a) There exist r address requests which cause memory contention of at least $t (R_{max} > t)$ for any assignment of requests to copies. (b) For any r-1 requests, however, it is possible to get $R_{max} < t-1$.

Proof : We show (b) first. Say, in contradiction, that a set of r-1 requests is given such that the best R_{max} obtainable is t. (It will be easy to modify our argument if the best R_{max} that can be obtained is greater than t). Among all possible ways to partition requests among our modules which imply $R_{max}=t$ choose these that assign minimum number of modules with exactly t requests. Then, restrict further the choice to a partition where a module with the smallest serial number possible is assigned with t requests. So we have t requests for addresses in some module. All these addresses have other copies in other modules. Each of these modules is assigned with at least t-1 requests(!). Let us denote a lower bound on the number of modules containing all copies of these t addresses by S_1, then we have, so far, requests for at least $(t-1)S_1+1$ addresses. Denote by S_2 a lower bound on the number of modules containing all copies of the $(t-1)S_1+1$ addresses. It should be clear how to define $S_3, S_4, \ldots$ (note that we actually choose S_1-1, S_2-S_1, $S_3-S_2, \ldots$, modules in the corresponding steps of this process)

Claim 1. Each module which enters the scene is assigned with at least t-1 requests.

Proof. Our sequence satisfies the following. There exists a directed path of the following form from the first module (the one we started with, which is assigned with t requests) to each module which is counted by the S_i numbers; the nodes along the path are modules; and there is a directed edge from module A to module B if module A is assigned with some address request and another copy of this address is located in module B. By propagating requests along this path we may decrease the number of requests assigned to the first module by one, increase the number of requests assigned to the last module on the path by one and not change the number of requests assigned to any other module. Thus, the existence of a module as in claim 1, which is assigned with less than t-1 requests contradicts the choice of the partitioning of requests among modules above.

$$S_1 = \min\{s/\tbinom{s}{c}x > t\},\ S_{i+1} = \min\{s/\tbinom{s}{c}x > S_i(t-1)+1\},\ \text{for } i > 1.$$

Claim 2. The sequence (of integers) $\{S_i\}$ converges to S_f.

Proof. Recall $s_f \leq m$. If $S_i < S_f$ then S_{i+1} satisfies $S_i < S_{i+1} \leq S_f$. If $s_i = s_f$ then $s_{i+1} = s_f$.

So we could not start with less than $s_f(t-1)+1$ requests. A contradiction.

Proving (a) is easy. Take r $(= s_f(t-1)+1)$ requests that all their copies are in s_f modules. This completes the proof of Theorem 1.

The Connection with the lower bound. We showed that for any r address requests the smallest number of modules to contain their copies is the smallest k satisfying $r \leq \tbinom{k}{c}x$, which is exactly our S_f. Then we concluded that R_{max} is at least the smallest integer satisfying $R_{max} \geq r/S_f$. Here this integer is t which is exactly the best R_{max} achievable. So, the upper bound and the lower bound are exactly equal.

For any p requests let us establish an explicit upper bound on the best R_{max} achievable. Denote this R_{max} by t. Let x be the integer such that

$$(x-1)\binom{m}{c} < N < x\binom{m}{c} .$$

Similar to the proof of Theorem 1 we restrict the choice of the assignments of requests to modules. Then, we start with a module which is assigned with t requests. Now, s is the minimum number of modules that have to be assigned with at least t-1 requests in order not to enable us decrease t in the first module. The following inequality holds from similar reasons to the proof of Theorem 1, $x\binom{s}{c} > (t-1)s+1$. It implies

$$x\ s^c/c! > (t-1)s + s > (t-1)c!/x)^{1/c-1}$$

Together with the fact $p > (t-1)s+1$ we get

$$(t-1)((t-1)c!/x)^{1/c-1} < p \text{ implying}$$

$$(t-1)^{c/c-1} < p/(c!/x)^{1/c-1} ,$$

$$(t-1)^c < xp^{c-1}/c! \text{ and}$$

$$t < 1 + (xp^{c-1}/c!)^{1/c} .$$

Thus,

Theorem 2. $R_{max} < 1 + ((p^{c-1}/c!)\lceil N/\binom{m}{c}\rceil)^{1/c}$. ($\lceil x\rceil$ denotes the smallest integer i such that $i > x$)

III 4. Optimal Assigment.

The problem of optimal assignment to the 'right' copy of each requested memory address, in order to minimize R_{max}, is solvable in polynomial time. We actually solve the problem for any distribution of copies among modules (not only c-partitionings), in Figure 2. For more on the Max-Flow problem, see [Even].

An alternative solution.

We have to assign p requests to modules . Assign one request at a time. Say that i out of the p address requests, for some $0 \leq i < p$, got already (temporary) assignment to modules. Assume that these i requests are assigned to modules in a way which minimizes R_{max}, with respect to them. Denote this R_{max} by $R_{max,i}$. We show how to extend this assignment to another temporary assignment for one more request, in a way which achieves minimum R_{max} with respect to the i+1 requests. Our algorithm and proof are similar to the proof of Theorem 1. Consider the following auxiliary directed graph. Nodes represent modules. There is an edge from module A to module B if module A is assigned with a request for some address a while B contains another copy of a. The algorithm for extending the assignment of i requests to an assignment of i+1 requests is as follows.

(1) Assign the (i+1)-st address request to a module M_1 containing one of the copies of the address. (Add to the auxiliary digraph c-1 edges from this module to the other modules that contain copies of this address.)

(2) If M_1 is now assigned with $\leq R_{max,i}$ requests then we are done. (It is impossible to add requests and decrease R_{max}.)

(3) If M_1 is now assigned with $R_{max,i} + 1$ requests search the auxiliary digraph for a module which is assigned with less than $R_{max,i}$ requests and is reachable through a directed path from M_1. (Any efficient search can be utilized here. For instance Breadth First Search) If such a module is found then propagate request assignments along a path in the digraph from M_1 to this module. (This results by the following change: the number of requests assigned to this module increases by one. Note, however, that it is still $\leq R_{max,i}$. The number of request assigned to all other module is exactly the same as it was for the i requests.)

(4) If no such module is found in (3) do nothing (the request is assigned to M_1).

The only thing that has to be proved is that step (4) is correct. In other words: why is it impossible to assign these i+1 requests with

$R_{max} = R_{max,i}$ rather than $R_{max} = R_{max,i} + 1$? Assume, in contradiction, that there is an assignment of these i+1 requests with $R_{max} = R_{max,i}$. In this contradictory assignment M_1 is assigned with $< R_{max,i}$ requests. Therefore, there exists a request for some address a, which is assigned to M_1 in our assignment (the result of step (4)) but not in the contradictory assignment. Hence, the auxiliary graph contain an edge from M_1 to M_2 - the node representing the module that the request for a is assigned to it in the contradictory assignment. By step (3), M_2 has $R_{max,i}$ request assigned to it in our assignment. Define S, an auxiliary set of modules, to include presently M_1 and M_2. In our assignment both modules of S have together $2R_{max,i}+1$ requests assigned to them while in the contradictory assignedment there are at most $2R_{max,i}$ such requests. Therefore, there exist a request for some address a_2 which is assigned to a module of S in our assignment but not in the contradictory assignment where this request is assigned to another module not in S, say M_3. This implies the existence of an edge in the auxiliary graph from this module in S to M_3. By step (3) M_3 has $R_{max,i}$ requests assigned to it in our assignment. Add M_3 to S. Similarly we show that the set S can grow infinitely large. The number of modules is finite, so we got a contradiction. Therefore, the assignment achieved by our algorithm yields the minimum R_{max}.

III 5. Fast Parallel Approximation Algorithms.

The above mentioned optimal assignment algorithms are of general theoretical interest and might be relevant for bata-base applications as mentioned in the introduction. However, the simulation requires fast parallel algorithms for the assignment problem even if the optimal result is not achieved (remember that we are after fast simulation of one EREWPRAM cycle). This poses a challenge of a new kind.

Lemma. Let $N < \binom{m}{c}$ be the number of addresses. For p address requests we can achieve $R_{max} < cp^{c-1/c}$ in parallel time $O(c^2+c \log p)$. For c=2, we can do it in constant parallel time.

Remark. The claim for c=2 needs a little bit stronger assumptions about the MPC; like, a memory request can be dropped by the requesting processor, or, alternatively, a memory module can discard memory requests sent to it if their location in the module's queue exceeds a certain point.

Proof . By induction on c.
For $c=2$ start with any assignment of address requests to copies. For all modules where the number of requests is $> \sqrt{p}$ (there exists $< \sqrt{p}$ such modules) switch all requests above the $\sqrt{p}$ line (by doing that for all requests that were not responded in the first $\sqrt{p}$ time units of the cycle which is being simulated, we avoid computation of partial sums for $c=2$) to the other copy. Second copies belong to pairwise distinct modules. Therefore, each module gets no more than $\sqrt{p}$ requests for second copies. Now, we are ready for the inductive step. For c copies cut in the first copy over $p^{(c-1/c)}$. Get (at most $p^{(1/c)}$) sets of address request to be switched to another copy. The other $c-1$ copies for each set relate like $c-1$ copies of a subset of $\binom{m-1}{c-1}$ addresses. The somewhat tedious exact implementation details are left to the interested reader. The following outline will be helpful for this. The decision where is the $p^{(c-1/c)}$ line is based on scheduling the requests for the modules in a way which utilizes both sorting and partial sums and described in detail in the next chapter for a similar purpose. The scheduling of requests to second copies is done separately in each set of requests, that were switched from the same module, and so is the scheduling for later copies which is done separately for even more refined sets. The definition of such a refined set (at each transition from i to i-1 copies and application of the inductive step) includes all address requests that passed through the same sequence of modules but have not yet been satisfied. The partial sums computation for such set meets exactly the partial sums computation scheme described in the next chapter. (Here, we do not need the assumptions of the remark for $c=2$ since switched requests are not sent in the first place.) Therefore, we can apply the induction. Let A_i be sizes of switched sets then

$$\sum (c-1)A_i^{((c-2/(c-1))} < (c-1)p^{1/c}p^{((c-1/c))^{((c-2)/(c-1))}} = (c-1)p^{((c-1)/c)}$$

The inequality holds because the left hand side is maximized when all the A_i numbers are equal. The c^2 size is due to the need to compute the subset of modules that contain a copy of an address by each processor using its local memory. The $\log^2 p$ is due to computations of partial sums which is required at each of the c steps of the induction.

Theorem 3. We can achieve $R_{max} < c(p^{c-1}[N/\binom{m}{c}])^{1/c}$ in parallel time $O(c^2+c \log^2 p)$.

Proof. Let x be the integer such that $(x-1)\binom{m}{c} < N < x\binom{m}{c}$.
Apply the lemma for each of the x layers, separately. Let p_i be the number of requests for address in the i-th layer $1 \leq i \leq x$. Then we get

$$R_{max} < \sum_{1}^{x} R_{max}(i) < \sum_{1}^{x} cp_i^{(c-1)/c} < x\, c(p/x)^{(c-1)/c} =$$
$$(p^{(c-1)}x)^{1/c} < c(p^{c-1}[N/\binom{m}{c}])^{1/c}.$$

$R_{max}(i)$ is the R_{max} obtained for each layer. The first inequality is obvious. The second is implied by the lemma. The third is because its left hand side is maximized when all the p_i are equal. The scheme of partial sums computation of the next chapter should be used here as well.

III 6. Connection with the First Stage.

The proposed c-partitioning above poses some difficulties in combining it with the probabilistic simulation of the previous chapter. There, a copy of address a_i could have been found in module $h(a_i)(\mathrm{mod}\ m)$. While in this chapter the set of modules that contain copies of address $h(a_i)$ was selected differently. Each address is assigned to a set of modules containing its copies (according to the definition of the proposed c-partitioning at the beginning of Section III 2) where no function like the remainder mod m seems to be involved in determining any member of this set. (For instance, module $h(a_i)(\mathrm{mod}\ m)$ may not have a copy of address a_i). This is the reason why for the purpose of combining this stage with the previous one we propose an alternative c-partitioning. Its small disadvantage is that it gives a little bit inferior results than the first c-partitioning. Its big advantage is that it suits to be a second stage following Chapter II. Note that in the sequel we always omit the hashfunction h and refer to the address a_i only. When the solution of this section is set to follow Chapter II address a_i should be replaced by $h(a_i)$.

The alternative c-partitioning. For $N < m!/(m-c)!$ and address i, $0 < i < N$, the first copy of i is in module $i \pmod m$, the second copy is in module $i \pmod{(m-1)}$ of the remaining $m-1$ modules and so on. Namely the j-th copy, $1 < j < c$, is in module $i \pmod{(m-j+1)}$ of the $m-j+1$ modules not occupied by the first $j-1$ copies.

Example. Let $m = 10$ $c = 3$ and $i = 1$. The first copy of i is in module 5 ($15 \pmod{10} = 5$), the second is in module 7 ($15 \pmod 9 = 6$ and since module 5 is occupied, module 7 corresponds to 6) and the third is in module 9 ($15 \pmod 8 = 7$ and module 5 and 7 are occupied).

If $(x-1)(m!/(m-c)!) < N < x(m!/(m-c)!)$ for some integer x then partition the addresses into x approximately equal subsets (layer) and fix the c-partitioning of each of the x layers as for $N < m!/(m-c)!$.

Remark. For an address i it takes $O(c \log c)$ time to compute the subset of c modules containing its copies. Clearly, $i_1' = i \pmod{[N(m-c)!/m!]}$ is the serial number of i in its layer. Find the modules one at a time. Create a 2-3 tree for the modules that were chosen so far. (For more on 2-3 trees, see [Aho, Hopcroft and Ullman].) By keeping in each internal node of the tree information about the number of unoccupied modules among its leaf-descendents we can identify the module of the next copy and update the tree in time $O(\log c)$. The simple additional details are omitted. In the cases where $N = xc!\binom{m}{c} (= x\ m!/(m-c)!)$ for some integer x, Observation 2 and Theorem 1 still hold since $\#A_{i_1,i_2,\ldots i_k} = x\ c!$ for any subset $\{i_1,i_2,\ldots,i_k\}$ of k modules in both c-partitionings (using the notations of the lower bounds section). In order to shorten the paper we reconstructed here only the upper bound analogues to Theorem 2. This is done in an informal way since it follows the same lines as the proof of Theorem 2.

For any p requests we want to find an upper bound on the best R_{max} achievable. Let x be the integer such that $(x-1)m!/(m-c)! < N < x\ m!/(m-c)!$. For some $R_{max} = t$ and a module assigned with t requests denote by s the minimum number of modules (including the first module) that have to be assigned with at least $t-1$ request in order not to enable us decrease t in the first module. The following two inequalities hold:

(1) $\quad x\ s!/(s-c)! > (t-1)s + 1.$

(2) $\quad p > (t-1)s + 1.$

(1) implies $x\ s^c > (t-1)s$ and $s > (t/x)^{c-1}$. This and (2) imply

$$(t-1)((t-1)/x)^{1/c-1} < p, \quad (t-1)^c < x\ p^{c-1} \text{ and } t < 1 + (x\ p^{c-1})^{1/c}$$

Theorem 2′. For the alternative c-partitioning

$$R_{max} < 1 + (p^{c-1}[N(m-c)!/m!])^{1/c}.$$

The analogue to Theorem 3 will be

Theorem 3′. For the alternative c-partitioning we can achieve

$$R_{max} < c(p^{c-1}c![N(m-c)!/m!])^{1/c}$$

in parallel time $O(c \log c + c \log^2 p)$.

Proof.

The only new idea in this proof is the following. We show that all addresses that fall in the same layer of our alternative c-partitioning can be efficiently further partitioned into c! 'sublayers'. Note that the $m!/(m-c)!$ addresses of one layer (there is at most such number) hit every subset of c modules exactly c! times each time in the c modules are hit in a different order of the copies. Let $(i_1,i_2,\ldots,i_c)$ be the c modules that contain the respective first, second,..., c-th copy of address i that belongs to layer ℓ for some i and ℓ. Denote by $N(j)$ the cardinality of $\{i_k | k<j \text{ and } i_k<i_j\}$. Note that these cardinalities can be observed upon searching the 2-3 tree mentioned above for i_j without changing the $O(\log_c c)$ time estimate. Obviously $0 \leq N(j) < j$. Define $L(i_1,i_2,\ldots,i_c) = \sum (j-1)!N(j)$ to be the sublayer of address i in layer ℓ. The only address in this sublayer which is contained in modules $i_1,i_2,\ldots,i_c$ is i. Therefore, the N addresses form altogether $c![N(m-c)!/m!]$ sublayers. Applying the lemma similar to the proof of Theorem 3 to each of this sublayers completes the proof of the theorem. The $O(c \log^2 p)$ size represents the computation of partial sums as in the proof of the lemma.

III 7. Some Nonconstructive Upper Bounds.

In this section we demonstrate a way for utilizing a number of copies c where $\binom{m}{c} >> N$. A typical counting argument that provides for non-constructive c-partitionings having the desirable property of enabling small R_{max} is presented. We leave the problem of constructing such efficient c-partitionings open.

We first need the following theorem:

Theorem 4: Let p and m be as before. Let d be a positive integer, where $p \leq dm$. Assume that a c-partitioning is given such that for every subset of size p of the N addresses, the set of modules that contain all c copies of these p addresses is of size $\geq [p/d]$. Then it is possible to get $R_{max} \leq d$ for every p memory requests.

Proof. (Similar to the proof of correctness of the alternative sequential algorithm.) Assume, in contradiction, that there exists a set of $p(\leq dm)$ requests that causes $R_{max}>d$. By a similar technique we choose an assignment such that some module is assigned with $\geq$ d+1 requests. They must have copies in one more module. This second module must be assigned with $\geq$ d request. So far we have $\geq 2d + 1$ requests. They must have copies in $\geq 3 = [\frac{2d+1}{d}]$ modules. The third module is also assigned with $\geq$ d requests, and so on. So we get that this set of requests was of size $\geq dm + 1$. A contradiction.

This is, actually an extension of Hall's Theorem (cf. [Even]) which is given for the case d = 1. An upper bound on the portion of c-partitionings that do not have the favorable property described in the theorem is

$$I_p = \sum_{1\leq k\leq p} \binom{N}{k} \binom{m}{[\frac{k}{d}] - 1} \frac{c\frac{N}{m}([\frac{k}{d}]-1)!}{(c\frac{N}{m}([\frac{k}{d}]-1)-ck)!} \frac{(cN-ck)!}{(cN)!}$$

assuming that N is divisible by m. (Note, that the definition of a c-partitioning is extended to the case where more than one copy of an address is contained in the same module.) This upper bound reminds to some extent the proof of the existence lemma of "expander bipartite graphs" given in page 298 of [Pippenger].

If we prove for some values of N, m, p, d and c that I_p (the portion of "bad" c-partitionings) is smaller than one then we gave a (non-constructive) existence proof of a "good" c-partitioning.

Appendix I examplifies a result that can be derived from this upper bound. We get that if $N = p^3$, $m = p^2$, $d = 1$ and p is "large enough" than there exists a "good" 10-partitioning.

In a similar way to the considerations above we may obtain an upper bound of 1-b on the portion of bad c-partionings, for some positive constant b. Since we do not have a way to construct a "good" c-partitioning, we can choose one at random and use it till the first time it fails. Namely, if we find for the first choice a set of requests for which it is impossible to get $R_{max} = d$, then we choose another c-partitioning and so on. Obviously, with high probability, we find after a small number of trials a "good" c-partitioning.

IV. Efficient Low-level Simulations.

The purpose of this paper is to present ways for simulations of shared memory models of parallel computation which allow fairly unrestricted access patterns of processors to shared memory cells by machines in which the memory is organized in modules where only one cell of each module can be accessed at a time. As was mentioned in the introduction the paper envisions a three stage analysis and solution to the problem. The combination of the two earlier stages brings us to a point where each processor of the MPC simulating machine specifies in each cycle being simulated both an address request and the module that was chosen to satisfy this request. The problem is how to complete the simulation. The choice of the EREWPRAM and the MPC for presentation of our ideas for the second two stages is due to the fact that the simulation of the former by the latter distinguished both the problems and solutions. Among other things, it was helpful to make the need for a third stage indistinct and thereby focusing on the previous stages. This is the reason that here we switch to other models of computation. Instead of the EREWPRAM we take a more permissive model of computation, the concurrent-read concurrent-write (CRCW)PRAM. In this model several processors are allowed to read simultaneously from the same memory location. If several processors try to write simultaneously into the same memory location the lowest numbered processors succeeds. This model is based on [Goldschlager] and [Shiloach and Vishkin]. We substitute the MPC machine by a weaker machine named non-queued (NQ)-MPC. Here only one address request may arrive at each module at each time unit. Besides that the NQ-MPC is similar to the MPC.

We wish to simulate one cycle of the CRCWPRAM. Assume that each processor of the NQ-MPC is assigned already both with an address request from the common memory and the memory module which has to satisfy this address request. Assume also that $m > p$, namely the number of modules is at least as big as the number of processors. Our proposed solution resembles the simulation of a CRCWPRAM by a EREWPRAM in [Vishkin 83]. Unfortunately, our present problem requires not only the circumvention of access conflicts to the same memory location but also to the same module. Whenever the considerations are similar to this simulation we shorten the presentation. The first step of the simulation is:

(1) Sort in parallel the p triples specified below in the lexicographic order.

Each processor enters the triple:

(the serial number of the module assigned to its address request, the serial number of the address request itself , its own serial number).

The NQ-MPC sorts them using Batcher sorting algorithm in time $O(\log^2 p)$. It does not need more than size p shared memory. This is achieved by using one cell of each module. The result is that all requests for the same module (resp. the same address of the same module) appear in successive locations of the sorted vector. Let us call the set of such successive locations interval (resp. subinterval) denoted $I(M)$(resp. $SI(M,a)$). M and a represent indeterminants corresponding to modules and addresses, respectively. Note that subintervals are sorted by the serial number of the processors.

(2) The serial number of each subinterval relative to the other subintervals in the sorted vector is computed. Denote it by $\#SI(M,a)$.

A processor is allocated to each triple. By comparison with the triple in the preceding place of the sorted vector the processor finds out whether its triple is the smallest in its subinterval. If yes it is chosen to 'represent' the subinterval as will be seen later; let us call the triple in this case, a subinterval-triple. Each processor that is (resp. not) allocated to a subinterval triple enters one (resp. zero) to a simple (O[log p]) time partial-sums computation. This results with the required subinterval serial number associated with the subinterval triple.

(3) The serial number of each subinterval which is smallest in its interval ($\min_b(\#SI(M,b))$) is 'broadcasted' to all other triples of the interval.

The number $\#SI(M,a) - \min_b(\#SI(M,b))+1$ is the serial number of the subinterval SI(M,a) relative to other subintervals of interval I(M). The broadcasting is done in [log p] pulses. In pulse i, $1 \leq i \leq [\log p]$, each processor that knows, already, the serial number of the smallest subinterval of the interval of its triple writes it into the 2^{i-1} successor of the triple in the sorted vector if it belongs to the same interval. It is simple to see (see [Vishkin 83]) that all triple get the appropriate message and each module is accessed by one processor at a time.

(4) The processor of each subinterval-triple performs the requested access to the right module in time corresponding to the serial number computed in Step (3) above.

In case, we simulate a writing cycle (recall the classification of the previous chapter into reading and writing cycles) we are done. In case a reading cycle is simulated we finish by:

(5) The content read by the processor of each subinterval-triple is broadcasted to all other triple of this subinterval.

The broadcasting is done by a similar technique to Step (3). Broadcasting for the same purpose is used in [Vishkin 83]. The $O(\log^2 p)$ time for sorting clearly dominates the time complexity of this simulation. It is appropriate to mention at this point the new O(log p) sorting algorithms (for p processors) given by [Ajtai, Komlos and Szemeredi] and [Reif and Valiant]. In the first algorithm the time is multiplied by a large constant, while the second is a probabilistic algorithm and the constant factor is smaller.

V. Summary

We give above a detailed description of each component of our solution. Here we would like to give a high level description of a simulation of the CRCWPRAM by the NQ-MPC.

The CRCWPRAM (resp. NQ-MPC) is permissive (resp. restrictive) relative to the spectrum of other permissive (resp restrictive) models of computation that appear in the literature. This enables us to go again through the main notions of our solution in order to summarize them in a uniform fashion.

Stage one assumes a class H of hashfunction. Pick at random one of them say h. This hashfunction implies the location of the c copies of each address a_i. By the alternative c-partitioning of stage two the j-th copy is in module $h(a_i)(\mathrm{mod}\ m-(j-1))$ of the modules that were not occupied by the first j-1 copies ($1 \leq j \leq c$). The step by step simulation starts with each processor of the NQ-MPC machine specifying an address request for read or write like its corresponding CRCWPRAM processor. Now we have to split into reading and writing cycles. For reading cycles Theorem 3′ gives an algorithm for allocation of address requests to modules. A scheduling of the requests to time units of the simulated cycle and a way to transmit the read request to the modules and the response back to the processors are described in the previous chapter. For writing cycles each address request is assigned to all its c copies. We do not do worse than c times (the time for one reading cycle). The reader is invited to by observing that we can access: first copies, then second copies, and so on.

It is interesting to note that both the Ultracomputer and the PDDI machine are able to compute partial sums in the same time it takes for the processors to access the memory. So in both machines the term $\log^2 p$ in the time evaluation of theorems 3 and 3′ and the preceding lemma can be omitted. They also allow simultaneous access by several processors to the same cell of a module. It is resolved in the same way as in the CRCWPRAM within the same time as accessing the memory. So most of the discussion of Chapter IV is irrelevant for both machines.

Acknowledgement

Helpful discussions with A. Gottlieb, M. Snir, P. Spirakis and A. Wigderson are gratefully acknowledged. We thank Kevin McAuliffe for the information about the ratio between read and write instructions achieved in simulations of the Ultracomputer Project.

References

[Aho, Hopcroft and Ullman] Aho, A.V., Hopcroft, J.E. and Ullman, J.D., The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA 1974.

[Ajtai, Komlos and Szemeredi] Ajtai, M., Komlos, J. and Szemeredi, E., "An O(nlog n) sorting network", Proc. Fifteenth ACM Symposium on Theory of Computing, 1983, pp. 1-9.

[Carmichael] Carmichael, R.D., Groups of finite orders, Dover Publications, 1956.

[Carter and Wegman] Carter, J.L. and Wegman, M.N., "Universal classes of hash functions", Proc. Nineth ACM Symposium on Theory of Computing, 1977, pp. 106-112.

[Even] Even, S., Graph Algorithms, Computer Scince Press, Potomac, Maryland, 1979.

[Goldschlager] Goldschlager, L.M., "A Unified Approach to Models of Synchronous Parallel Machines," Proc. Tenth ACM Symposium on Theory of Computing, 1978, pp. 89-94.

[Gonnet] Gonnet, G.H., "Expected lengh of the longest probe sequence in hash code searching", JACM 28, 1981, 289-304.

[Gottlieb et al.] Gottlieb, A., Grishman, R., Kruskal, C.P., McAuliffe K.P., Rudolph, L. and Snir, M., "The NYU Ultracomputer-Designing a MIMD Shared Memory Parallel Machine", IEEE Trans. on Comp. c-32,2, 1983, pp. 175-189.

[Kuck] Kuck, D.J., "A survey of parallel machine organization and programming", Computing Surveys 9, 1, 1977, 29-59.

[Lev, Pippenger and Valiant] Lev, G., Pippenger, N. and Valiant, J.G., "A fast parallel algorithm for routing in permuting networks", IEEE Trans. on Computers c-30,2, 1981, pp. 93-100.

[Pippenger] Pippenger, N., "Superconcentrators", SIAM J. on Computing 6,2, 1977, pp. 298-304.

[Rabin] Rabin, M.O., "Probabilistic algorithms", in Algorithms and Complexity (J.F. Traub, Ed.), Academic Press, New York, 1976.

[Reif and Valiant] Reif, J. and Valiant, L.J., "A logarithmic time sort for linear size networks", Proc. Fifteenth ACM Symposium on Theory of Computing, 1983, pp. 10-16.

[Shiloach and Vishkin] Shiloach, Y., and Vishkin, U.,"Finding the maximum, merging and sorting in a parallel computation model", J. of Algorithms 2,1, 1981, pp. 88-102.

[Vishkin82] Vishkin, U., "Parallel-Design space Distributed - Implementation space (PDDI) general purpose computer", RC 9541, IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, 1982.

[Vishkin83] Vishkin, U., "Implementation of simultaneous memory address access in models that forbid it", J. of Algorithms 4,1, 1983, pp. 45-50.

[Vishkin and Wigderson] Vishkin, U. and Wigderson, A., "Dynamic parallel memories", TR-72, Dept. of Computer Science, Courant Institute, NYU, 1983.

Appendix 1.

The following example may illustrate what are the results that one may expect out of Section III 7. No special effort was made to get the best possible result in the subsequent computation.

Example: Let $N = p^3, m = p^2$ and $d \equiv 1$. Then,

$$I_p < \sum_{1\le k\le p} \binom{p^3}{k}\binom{p^2}{k} \frac{(cpk)!}{(cpk-ck)!}\,\frac{(cp^3-ck)!}{(cp^3)!}$$

$$= \sum_{1\le k\le p} \binom{p^3}{k}\binom{p^2}{k}\binom{cpk}{ck}\Big/\binom{cp^3}{ck}$$

Since

$$\binom{cp^3}{ck} > \binom{p^3}{k}\binom{p^2}{k}\binom{(c-1)p^3-p^2}{(c-2)k}$$

we get

$$< \sum_{1\le k\le p} \binom{cpk}{ck}\Big/\binom{(c-1)p^3-p^2}{(c-2)k}$$

Since

$$\binom{cpk}{ck} < \frac{(cpk)^{ck}}{(ck)!}$$

and

$$\binom{(c-1)p^3-p^2}{(c-2)k} > \frac{\left((c-1)p^3-p^2-(c-2)k\right)^{(c-2)k}}{((c-2)k)!}$$

we get

$$< \sum_{1\le k\le p} \frac{(ck)!}{((c-2)k)!}\,\frac{(cpk)^{ck}}{\left((c-1)p^3-p^2-(c-2)k\right)^{(c-2)k}}$$

Now $\frac{(ck)!}{((c-2)k)!} < (ck)^{2k}$; and for p "large enough" $p^3 > p^2 + (c-2)k$ where c is a constant and $k < p$. So

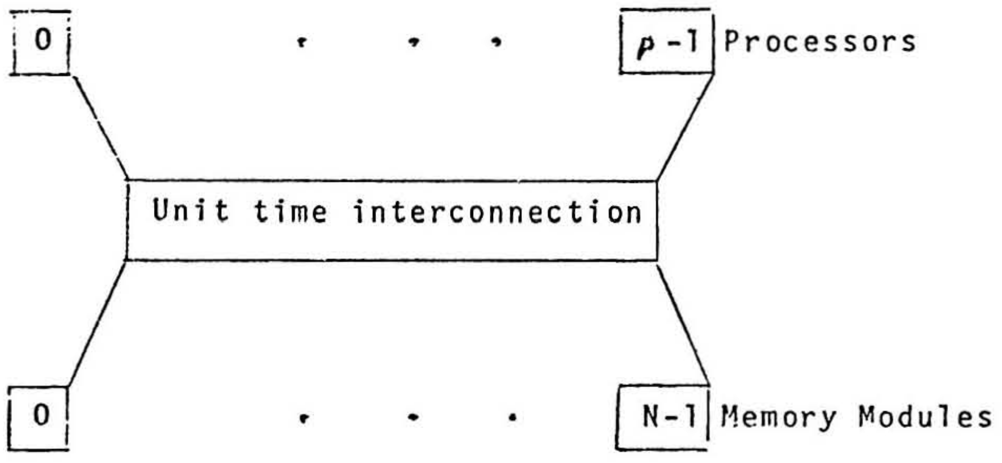
$$< \sum_{1 < k < p} \frac{(ck)^{2k}(cpk)^{ck}}{((c-2)p^3)^{(c-2)k}}$$

Call each element L_k. Let $c = 10$. For large enough p

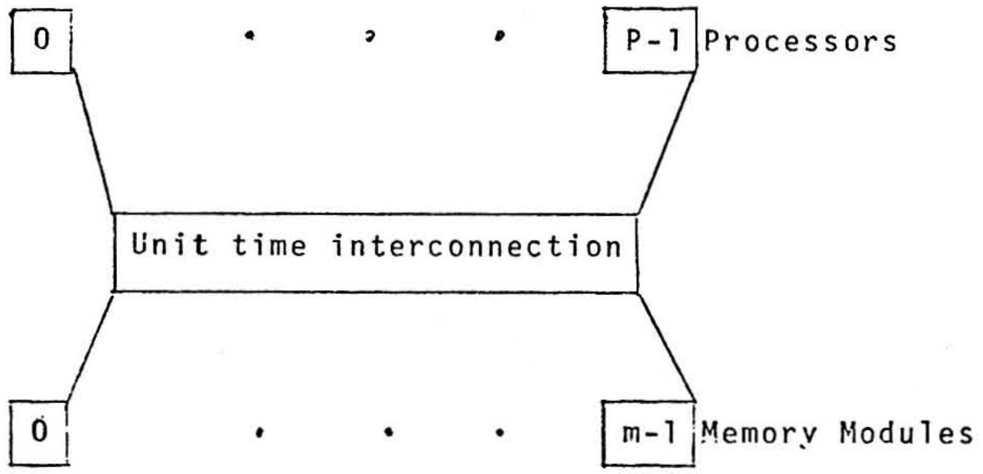
$$L_1 = \frac{10^2(10p)^{10}}{(8p^3)^8} < 1/p$$

Since it is easy to verify that the derivate of L_k with respect to k $(1 < k < p)$ is negative for large enough values of p, we get $\sum_{1 \leq k \leq p} L_k < 1$. So, we proved the existence of a 10-partitioning such that for these parameters gives always a memory contention of one.

THE MODELS OF COMPUTATION



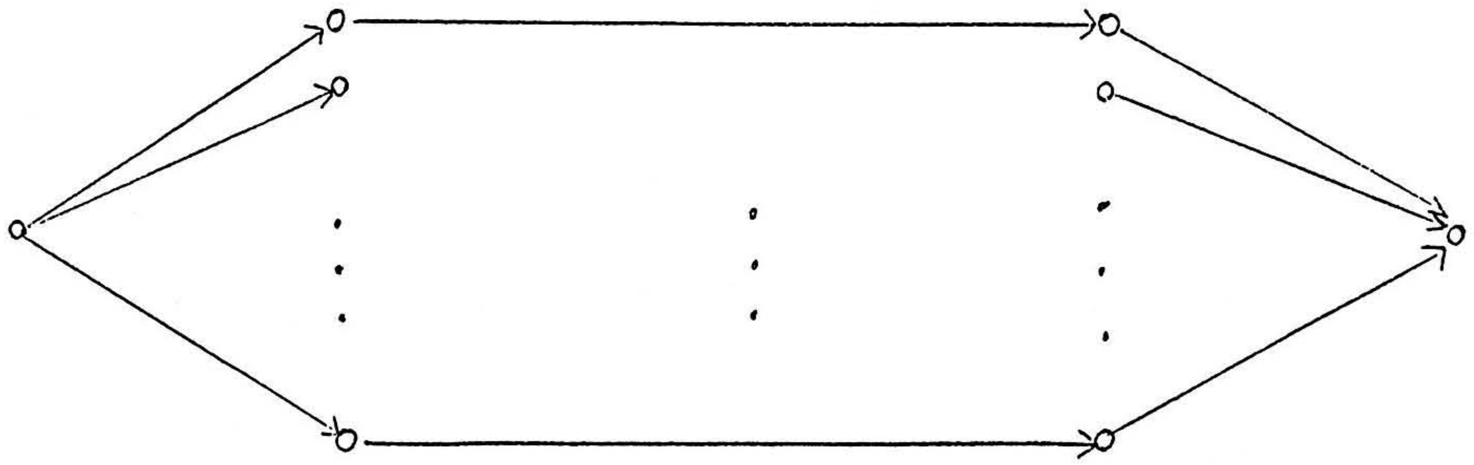
EREW PRAM



Module Parallel Computer
(MPC)

(FIGURE 1)

FINDING OPTIMAL ASSIGNMENT TO COPIES BY UTILIZING INSTANCES OF MAX-FLOW



source — capacity 1 — r nodes representing address requests — capacity 1 edge from address-node to module-node if the module contains a copy of the address — nodes representing modules — capacity z — sink

The optimal assignment algorithm:

find (by binary search) minimum z that enables flow of r.

(FIGURE 2)