SPEEDING UP RANDOM ACCESS MACHINES BY FEW PROCESSORS

(Preliminary Version)

Friedhelm Meyer auf der Heide

FB 20-Informatik, Johann Wolfgang Goethe Universität Frankfurt

6000 Frankfurt a.M.

Fed. Rep. of Germany [1)]

Abstract: Sequential and parallel random access machines (RAMs, PRAMs) with arithmetic operations + and - are considered. PRAMs may also multiply with constants. These machines work on integer inputs. It is shown that, in contrast to bit orientated models as Turing machines or log-cost RAMs, one can in many cases speed up RAMs by PRAMs with few processors. More specifically, a RAM without indirect addressing can be uniformly sped up by a PRAM with q processors by a factor $\frac{(\log\log q)^2}{\log q}$. A similar result holds for nonuniform speed ups of RAMs with indirect addressing. Furthermore, certain networks of RAMs (such as k-dimensional grids) with q processors can be sped up significantly with only $q^{1+\varepsilon}$ processors. Nonuniformly, the above speed up can even be achieved for arbitrary bounded degree networks (including powerful networks such as permutation networks or Cube-Connected Cycles), if only few input variables are allowed. It is previously shown by the author, that the speed ups for RAMs are almost best possible.

[1)] This research was done at the IBM Research Laboratory, San Jose, CA, USA.

Introduction

Parallel random access machines (PRAMs) are a widely accepted model of parallel computation. Many algorithms are known in this model which show that sometimes surprisingly strong speed ups of certain sequential algorithms are possible. On the other hand many problems look inherently sequential, i.e. there seems to be no significant speed up possible, at least when only few processors are allowed. It is known [PR] that with many, namely 2^t processors, one can speed up t steps of a Turing machine by a factor $\frac{\log\log\log t}{\log\log t}$, but no speed ups are known with poly(t) processors.

In this paper we show that such speed ups are possible for RAMs with operations + and -, uniform cost measure, and inputs given integer by integer, not bit by bit. We show that such RAMs without indirect addressing (storage addresses are functions in the ***number***, but not in the ***values*** of the inputs) can be sped up by PRAMs with q processors by a factor $\frac{(\log\log q)^2}{\log q}$, if the PRAMs can multiply with constants. Here ***uniform*** means that , if the RAM computes $f: N^* \to N^*$ in time T(n) (n = #(input variables)), then the PRAM needs time $T(n)\frac{(\log\log q)^2}{\log q}$. We also can speed up RAMs with indirect addressing in a similar way, but only to the expense of nonuniformity, i.e. we need a new PRAM for each new number n of input variables. This is one of the examples where fast algorithms can be designed to the expense of nonuniformity. Other, more surprising examples are the fast nonuniform algorithms for the knapsack or traveling salesman problem [M1], [M2], or the fast nonuniform simulations of probabilistic by deterministic computations [BG], [A], [M3].

We furthermore show that even certain networks of RAMs can be sped up. If q RAMs are for example connected to a k-dimensional grid, a PRAM with $q^{1+\varepsilon}$ processors can uniformly speed it up by a factor $\frac{(\log\log q)^2}{(\log q)^\delta}$, where $\delta = \frac{1}{k+1}$. Nonuni-

formly, we even can speed up arbitrary bounded degree networks, including powerful networks as permutation networks or Cube-Connected Cycles, as long as the number of input variables is very small relative to the number of processors.

The paper is organized as follows. In section I we define our computation models in more detail and state our results. In section II we show how to speed up straight line versions of RAMs. It turns out that, because only operations + and - are allowed, we can achieve dramatic speed ups with few processors. This result is used in the next two sections in order to show the simulations of RAMs and networks.

I. Definitions and Results

A random access machine (RAM) consists of a program and an infinite set of registers labelled by 1,2,..., each able to store one integer. Initially the first n registers contain the input consisting of n integers, finally the first m registers contain the output, m integers. In one step, a RAM1 can execute a direct storage access (i.e. the address of the accessed register only depends on the ***number***, but not on the ***values*** of the input variables, and its computation only uses constant addresses independent on n), write a constant into register 1, add or subtract two contents of registers, or execute an If-question (if (content of register 1) > 0 then ... else...). A RAM2 can, in addition to the above, execute indirect storage accesses, i.e. use any computed value as an address.

A PRAM1 (PRAM2) with q processors P1,...,Pq consists of q RAM1s (RAM2s), the processors, and an infinite shared memory consisting of registers labelled with 1,2,.., each able to store one integer. In addition to its RAM1 (RAM2) capabilities each processor can access directly (and also indirectly) the shared memory. Each processor can also multiply a content of a register with a constant, where

a number is constant, if it appears in the program, or, inductively, is the sum or product of two constants. The PRAMs are assumed to be synchronized. We allow to concurrently read in the same register of the the shared memory, but no concurrent write is allowed. (This means that for our simulations we do not need the very strong concurrent write versions of PRAMs, compare [FMRW].)

Straight line RAM1s or RAM2s are those RAM1s or RAM2s which do not use If-questions.

A network M of q RAM1s or RAM2s consists of q processors (RAM1s or RAM2s) which are partially connected according to a (communication) graph. In one step each processor can, in addition to its sequential capabilities, read an information from some register of a neighboring processor. M has no shared memory. If $s:N\to N$ is such that each Pi can reach at most s(t) different processors along paths of length at most t in the communication graph, then s is the ***spreading function*** of M.

A machine as described above is uniform (and T(n) time bounded), if it computes a function $f:N^*\to N^*$ (in time T(n), where n denotes the respective number of input variables). A family of such machines is nonuniform (and T(n) time bounded), if it contains a machine for each n which computes $f|_{N^n}$ (in time T(n)). Well known examples of nonuniform computation models are Boolean and algebraic circuits and all types of computation trees or branching programs.

In this paper we show the following results.

Theorem 1 : A uniform PRAM1 with q processors can simulate a T(n) time bounded RAM1 in time $O(T(n)\frac{(\log\log q)^2}{\log q})$.

Theorem 2 : A family of nonuniform PRAM1s with q processors can simulate a T(n) time bounded RAM2 in time $O((T(n)+n)\frac{(\log\log q)^2}{\log q})$.

Theorem 3 : A uniform PRAM1 with $q^{1+\varepsilon}$ processors can simulate a T(n) time bounded network of q RAM1s with spreading function $O(t^k)$ in time $O(T(n)\frac{(\log\log q)^2}{(\log q)^\delta})$, where $\delta=\frac{1}{k+1}$.

Theorem 4 : Let the number n of input variables the following machines deal with be a constant. A nonuniform PRAM1 with $q^{1+\varepsilon}$ processors can simulate T steps of a bounded degree network of q RAM1s in time $O(T\frac{(\log\log q)^2}{\sqrt{\log q}})$. (n is considered constant in this O-notation.)

Remark 1 : A k-dimensional grid has spreading function $O(t^k)$. Thus theorem 3 applies to it.

Remark 2 : In [M4] the author has shown that the simulations from the theorems 1 and 2 are almost best possible (up to a factor $\log\log q$).

II. Speeding up straight line RAMs

In this section we show that straight line RAM1s can be sped up significantly.

Lemma 1 : A PRAM1 with t^3 processors can simulate t steps of a straight line RAM1 M in $O((\log t)^2)$ steps.

Proof: Assume for a moment that we have computed all at most t addresses used in M. We sort them and compute their ranks, such that equal numbers have the same rank. This can be done in time $O(\log t)$ with t processors [AKS]. For our purpose even the $O((\log t)^2)$ sorting algorithm from [B] is good enough. We now replace each address by its rank. This obviously does not change the computation and has the advantage that we only access the registers 1,...,t.

Let $\vec{c}_i \in N^{t+1}$ be the vector of the contents of the t registers before step i, $c_{i,0}=1$. Then for each operation of M there is a (t+1)x(t+1) matrix A such that $\vec{c}_{i+1}=A\vec{c}_i$.

where i is the time when the operation is executed. For example, if the operation is "store the sum of registers i and j into register i", then $A=E+E_{ij}$, where E is the unity matrix and E_{ij} has a one at position ij and zeros everywhere else. As the first component of $\bar{c}_i$ is 1, we also can create a constant c in register j by the matrix $E+cE_{j0}-E_{jj}$, etc. Thus we can compute in constant time the t matrices $A_1,\ldots,A_t$ (each with the help of t^2 processors) such that $\bar{c}_{t+1}=A_tA_{t-1}\cdots A_1\bar{c}_1$. But in this way we certainly can compute $\bar{c}_{t+1}$, i.e. simulate M with t^3 processors, in time $O((\log t)^2)$, because with t^2 processors we can execute a matrix multiplication in time $O(\log t)$, and because of the associativity of this operation we get the whole product in $\log t$ stages each consisting of at most t parallel matrix multiplications. (Here we need that the processors may multiply with constants.)

It remains to show how to compute the addresses mentioned in the beginning of the proof. This can be done by the same algorithm, if we consider n to be the input variable. For this program the addresses are by the definition of direct addressing only constants which appear explicitly in the program and need not be computed. Thus we get all addresses in time $O((\log t)^2)$, which completes the simulation. q.e.d.

III. Simulating RAMs by PRAMs

We first prove theorem 1.

Let t be chosen such that $2^t t^4 \le q$. Assume we have simulated some number of steps. We now show how to simulate the next t steps. A computation of length t is a sequence of instructions M executes if the results of the If-questions are fixed. There are at most 2^t such computations, $C_1,\ldots,C_{2^t}$. For each of them we reserve t^4 processors. Let $C=C_i$ be fixed. C is a RAM1. For each $t' \le t$ we now use t^3 processors to simulate the prefix of length t' of C. This can be done in time

$O((\log t)^2)$ by lemma 1. Now we check whether C is the computation actually executed by M. This can be done in constant time because we know all prefixes and therefore all values determining the If-questions. The above we do for all C_i in parallel. Now the bunch of processors which has identified the right computation updates the registers maintaining the storage of M. This can be done in constant time because at most t changes are necessary and we have enough processors.

Thereby we have simulated t steps of M in $O((\log t)^2)$ steps. As we may choose $t \sim \log q$ theorem 1 follows. q.e.d.

Now we prove theorem 2.

We unroll the T(n)-time bounded RAM2 for a fixed number n of input variables to a computation tree as described in [M2]. In such a tree, a node representing an arithmetic operation or a direct storage access has one child, a node representing an If-question has two children, one for each possible outcome of the question, a node representing an indirect storage access has s+1 children $v_0, \ldots, v_s$, where s denotes the number of previously accessed registers. The i-th child stands for the case that the i-th of the previously accessed registers is accessed now, i=1,...,s. The 0-th child stands for the case that the register accessed now was never accessed before. (For a detailed description of this computation tree see [M2].)

As shown in [M2] each path of this tree can be looked upon as a straight line RAM1, if we ignore the nodes associated with If-questions.

In order to apply the idea from the last simulation we try to simulate all computations of length t from some time on, for a given parameter t. But as in this case the degree of the tree can be roughly T(n), we would need $T(n)^t t^4$ processors in order to simulate all possible computations of length t as in the previous simulation. Therefore there is no (non-constant) speed up possible, if the

number of processors is polynomial in T(n).

Thus we need an additional idea to obtain a speed up as demanded in theorem 2. We first remove all those branches from the computation tree which are not followed by any input. If during this procedure, a node representing an indirect storage access looses all but one of its children, this storage access was ***redundant***, i.e. for all inputs passing through this node the time when the accessed register was previously accessed is the same, or for all these inputs the accessed register was never accessed before. In this case we can clearly replace the indirect by a direct storage access. Note that these modifications can only be done to the expense of nonuniformity!

Now let t be chosen such that $p \leq 2^t t^4$. Suppose we have simulated some number of steps of the RAM2 M. We now want to simulate the next t steps. We have seen above that we do not have enough processors to simulate all computations of length t in parallel. Therefore we only simulate those computations which follow the 0-th branch at each (nonredundant) indirect addressing, i.e. which assume that the accessed register was never accessed before. As there are at most 2^t such computations, we can simulate all of them and all their prefixes in $O((\log t)^2)$ steps, as described in the previous computation.

Now we check whether one of these computations is correct, i.e. whether on one of them all the If questions are answered correctly and whether the choices of the 0-th branches at indirect addressings were correct. If we find such a computation we have simulated t steps in $O((\log t)^2)$ steps and are done.

Otherwise we locate the computation and the time when the first mistake (at an indirect addressing) occured. This can be done in constant time, because we have all the prefixes of the computations and enough processors.

Now we restart the simulation from this time on. First we simulate the indirect addressing correctly (with one processor in constant time). Then we start a new

phase of simulating t steps. By this algorithm we clearly simulate M.

Each phase of trying to simulate t steps needs $O(\log(t)^2)$ steps. The number of phases we now have to simulate is not only $\frac{T(n)}{t}$, but $\frac{T(n)}{t}+m$, where m denotes the number of mistakes, i.e. the number of unsuccessful attempts to simulate t steps. The following lemma shows that we do not make too many such mistakes and thereby implies theorem 2 .

Lemma 2 : During the simulation, at most n mistakes occur (n=number of input variables).

Proof : Let B be the set of inputs following the computation of M we consider up to (not including) the indirect read, where we made a mistake. Let L be the affine subspace with smallest dimension containing B. For an input $\bar{x} \in B$ let $f(\bar{x})$ be the address used in M at this indirect storage access. Suppose that the same cell was previously accessed at time t with address $g(\bar{x})$. Let now A contain all those inputs $\bar{y}$, for which $f(\bar{y})=g(\bar{y})$ holds. One easily checks that f and g are linear functions (see [M4]). As the indirect read is nonredundant we know that $f|_B \neq g|_B$. On the other hand, $f|_A = g|_A$ by construction. Thus $A \subset L' := L \cap \{\bar{y} \in R^n : f(\bar{x})=g(\bar{x})\}$. Thus each mistake reduces the dimension of the set of inputs following the computation we consider. After at most n mistakes this set has dimension 0, i.e. it consists of at most one point. But in this case there are no further nonredundant indirect storage accesses possible. Therefore our simulation makes no further mistakes. q.e.d.

IV. Simulating networks by RAMs

We first prove theorem 3. Let M be a network of RAM1s P1,...,Pq with spreading function s. Let p (>q) be the number of processors of the simulating PRAM1. Choose t such that $(t\, s(t))\,4^{2t\, s(t)} \leq \frac{p}{q}$. We again simulate t steps of M. For each Pi

we reserve $(t\,s(t))^4 2^{t\,s(t)}$ processors. We now use an idea from [M5] to simulate t steps of M by only considering relatively small parts of M. Let Mi be the subnetwork of M consisting of all those processors which are connected via a path of length at most t to Pi. Mi has at most s(t) processors. In [M5] it is shown that after t computation steps of M and Mi, the configurations of Pi in M and Mi are the same. Thus it is sufficient to simulate M1,...,Mq for t steps. As each Mi only has s(t) processors, we can simulate it by a RAM1 in $t\,s(t)$ steps. By theorem 1 we can simulate this RAM1 by a PRAM1 with $(t\,s(t))^4 2^{t\,s(t)}$ processors in time $O(\log(t\,s(t))^2)$ steps. Because of our choice of t we have enough processors to execute these simulations for all Mi in parallel. Thus we have simulated t steps of M in $O(log\,(t\,s(t))^2)$ steps. If $s(t)=O(t^k)$ for some k, we can choose $t\sim(\varepsilon \log q)^\delta$ with $\delta=\frac{1}{k+1}$ and obtain a PRAM with $q^{1+\varepsilon}$ processors which simulates T(n) steps of M in time $O(T(n)\frac{(\log\log q)^2}{(\log q)^\delta})$. q.e.d.

Now, in order to sketch the proof of theorem 4, we try to apply the same ideas to simulate arbitrary networks of RAM1s with q processors and degree bound c, say. Such a network has spreading function c^t. Let t be fixed. We specify it later. We define the Mi's as in the last proof. But now their number of processors can be exponential in t, namely up to c^t. We now interpret Mi as a PRAM1 with c^t processors which makes t steps. Combining ideas from [DL] and [M4] shows that this PRAM can nonuniformly be simulated in $d=O(2^n t^2)$ steps by a RAM1. Now we apply again theorem 1 to speed up this RAM1. With $d^4 2^d$ processors we can simulate Mi in $O((n+\log t)^2)$ steps. Thus $q d^4 2^d$ processors can simulate t steps of M in $O((n+\log t)^2)$ steps. Now choose $t\sim\sqrt{(\frac{\varepsilon \log q}{2^n})}=\Omega(\sqrt{\log q})$ because n is a constant. Then $d=q^\varepsilon$. Therefore we can simulate $\Omega(\sqrt{\log q})$ steps of M in $O((\log\log q)^2)$ steps using $q^{1+\varepsilon}$ processors. q.e.d.

References

[A] L. Adleman, Two theorems on random polynomial time, 19th IEEE-FOCS, 1978, 75-83.

[AKS] M. Ajtai, J. Komlos, E. Szemeredi, An $O(n\log n)$ sorting network , 15th ACM-STOC, 1983, 1-9.

[B] K. Batcher, Sorting networks and their applications, AFIPS spring joint computing conference 32, 1968, 307-314.

[BG] C. H. Bennett, J. Gill, Relative to a random oracle, $P^A \neq NP^A \neq co\text{-}NP^A$ with probability 1, SIAM J. on Comp. 10, 1981, 96-113.

[DL] D. Dobkin, R. Lipton, Multidimensional search problems, SIAM J. on Comp. 5, 1976, 181-186.

[FMRW] F. Fich, F. Meyer auf der Heide, P. Ragde, A. Wigderson, One , two, three ... infinity: lower bounds for parallel computation, 17th ACM-STOC, 1985, 48-58.

[M1] F. Meyer auf der Heide, A polynomial linear search algorithm for the n-dimensional knapsack problem, J. ACM 31(3), 1984, 668-676.

[M2]_____,Fast algorithms for n-dimensional restrictions of hard problems, 17th ACM-STOC, 1985, 412-420.

[M3]_____, Simulating probabilistic by deterministic algebraic computation trees, to appear in TCS.

[M4]_____, Lower bounds for solving linear Diophantine equations on several parallel computational models, to appear in Information and Control.

[M5]_____, Efficient simulations among several models of parallel computers, to appear in SIAM J. on Comp.

[PR] W. Paul, R. Reischuk, On alternation II, Acta Informatica 14, 1980, 391-403.