Poly-log parallel algorithms for LP with an application to exploding flying objects*

Nimrod Megiddo
Carnegie-Mellon University
and Tel Aviv University

November 1982

Abstract

The main results of this paper are the following: (i) LP is solvable in poly-log time when the number of variables is fixed (despite the general problem being log-space hard for P). (ii) Probabilistic parallel algorithms are designed with the aid of a probabilistic constant-time algorithm for maximum-finding (on the WRAM model) and random searches. We solve d-variable LP in $O(\log n(\log\log n)^{d-2})$ time, probabilistically; the deterministic algorithms run in $O(\log^{d-1} n \log\log n)$ time. (iii) We provide sequential algorithms for the following problem: Given n objects, flying at various speeds on straight line trajectories in R^3, find the smallest size of a bomb required to destroy all of them at once, thereby determining the optimal time and place for the explosion. Relying on parallel algorithms for LP-type problems, we solve the optimal explosion problem with (sequential) complexity as low as $O(n \log^4 n \log\log n)$ deterministically, and $O(n \log^2 n (\log\log n)^2)$ probabilistically.

*Supported in part by the National Science Foundation under grant no. ECS-8121741.

1. Introduction

The general LP problem is log-space hard for P [DLR]. So, we do not expect to solve LP in parallel in poly-log time. It is interesting to ask whether LP with a fixed number d of variables is solvable in poly-log time. The answer is not trivial since (i) both the simplex algorithm [Da] and the Khachiyan-type algorithms [K, YL] are inherently sequential, and (ii) even the special case of the max-flow problem is log-space hard for P [GSS]. Nevertheless, we provide an affirmative answer in the present paper. This stems from our previous (sequential) algorithms for LP [M3]. The obvious parallel versions of those algorithms are improved, and we also incorporate randomization for even further improvements. For $d=2$ we obtain an interesting probabilistic algorithm which runs almost surely (a.s.) in $O(\log n)$ time. This then generalizes to $O(\log n(\log\log n)^{d-2})$ algorithms for any fixed d. The deterministic algorithms run in $O(\log^{d-1} n \log\log n)$ time.

An interesting feature of this paper is the exploitation of a parallel constant-time probabilistic algorithm for the maximum [M4], in a probabilistic sequential algorithm for a problem of optimal explosion. The latter is defined as follows. Given n objects, moving on straight line trajectories in 3-space, find the smallest radius of a ball which contains all the objects at any one time. This ball determines the smallest size of a bomb that is required to destroy all the objects in one shot, thereby indicating the optimal time and place for the explosion. This problem provides motivation for studying the parallel complexity of the static problem of the smallest ball enclosing n given points in 3-space. The latter is closely related to 4-variable LP (see [M2, M3]). Actually, the algorithms for the two problems are very similar. The relationship between the dynamic and the static problems is along the lines

developed in [M1]. We remark that the dynamic problem in 3-space is not equivalent to the static problem in 4-space. The static problem is solvable in any fixed dimension in linear-time [M3]. Problems of computational geometry in R^3 are obviously much harder than in R^2. Even a seemingly simple problem, namely, recognizing whether any two of n given lines in R^3 intersect, is not known to be solvable (sequentially) in $o(n^2)$ time [Do]. It is therefore interesting that our dynamic smallest ball problem can be solved (sequentially) in $O(n \log^4 n \log\log n)$ time and, probabilistically, in $O(n \log^2 n (\log\log n)^2)$ time. These algorithms rely on our parallel algorithms for LP and static smallest ball problem.

2. Existence of poly-log algorithms for d-variable LP

The existence of poly-log algorithms for d-variable LP follows from our (sequential) linear-time algorithms [M3]. A parallel implementation runs in $O(\log^d n)$ for any d. However, we improve this bound substantially in the present paper. Our paper on the sequential case [M3] is currently not in the form of a publication (see [M2] for $d \leq 3$). The result is certainly a non-trivial extension of [M2] and necessitates at least a brief outline here.

The algorithm recurses on the dimension. It works as follows. First, the constraints are classified into three types and within each type we arrange the constraints in disjoint pairs. For each pair we compute an equation of a critical hyperplane. All this work requires constant-time on n processors. The next step requires partitioning the hyperplanes into two sets according to the slope of their intersection with the (x_1,x_2)-subspace. This involves finding the median slope. We can do it deterministically in $O(\log n \log\log n)$ time [BH, V]. Probabilistically, the slope of a random hyperplane turns out to be asymptotically as good as the median. Given the partition of the hyperplanes, we match those with "high" slope with those with "low" slope

(the matching is not perfect in the probabilistic case). For each pair of matched hyperplanes, we produce two other hyperplanes: one by eliminating x_1 from the equations and another one by eliminating x_2. This also takes constant-time. Let x^* denote the solution point. The basic idea is that we obtain information about the position of x^* relative to $B \cdot n$ hyperplanes ($B = B(d) > 0$), thereby discarding $B \cdot n$ constraints. Repeating this procedure $O(\log n)$ times reduces the number of constraints in such a way that we can then solve the problem in one step.

The information about $B \cdot n$ hyperplanes is acquired recursively (relative to d). We first obtain information about $B(d-1) \cdot n$ of the hyperplanes from which x_1 was eliminated and then about $(B(d-1))^2 n$ of their counterparts, i.e., those with x_2 eliminated. This step requires, by induction, $O(\log^{d-1} n)$ time, so that we can solve the d-dimensional problem in $O(\log^d n)$ time. Improvements will be discussed below.

3. Improvement of the two-variable algorithm

We now describe a parallel algorithm for 2-variable LP which runs in $O(\log n \log\log n)$ time, and hence improves upon the $O(\log^2 n)$ bound which follows from the general case. By induction, this also improves the bound for general dimension. Assume, for simplicity, our problem is to minimize y subject to $y \geq a_i x + b_i$ $(i=1,\ldots,n)$. This problem encompasses the typical operations required for solving any 2-variable LP. Let $f(x) = \text{Max}\{a_i x + b_i : i = 1,\ldots,n\}$, so we are seeking an x^* where f is minimized.

Our algorithm is based on Valiant's parallel maximum-finding [V], utilizing a technique we have developed in [M1]. The algorithm consists of $O(\log\log n)$ phases. Phase k starts with a subset $N_k \subset \{1,\ldots,n\}$ for which it is known that $f(x^*) = \text{Max}\{a_i x^* + b_i : i \in N_k\}$ (even though x^* itself is

not known). Moreover, an interval $[\alpha_k,\beta_k]$ is known such that $\alpha_k \leq x^* \leq \beta_k$ and $f(x) = \text{Max}\{a_i x+b_i: i\in N_k\}$ for all $x \in [\alpha_k,\beta_k]$. During phase k we form n pairs of lines from N_k. The pairs are determined by a graph on $n_k = |N_k|$ nodes and n arcs, which consists of pairwise disjoint cliques whose cardinalities differ by at most one. At the end of the phase each clique contributes precisely one line to the next set N_{k+1}. For each pair we find the intersection point of the participating lines. The amount of work so far is constant (see [SV]). A single evaluation of f at an $x \in [\alpha_k,\beta_k]$ takes $O(\text{loglog } n_k)$ time on n_k processors. Thus, we can simultaneously evaluate f at n/n_k points of $[\alpha_k,\beta_k]$ in $O(\text{loglog } n_k)$ time. This implies that, once the intersection points who lie in $[\alpha_k,\beta_k]$ are sorted (requires $O(\log n)$ time), we can in $O(\log n/\log(n/n_k))$ steps (each consisting of n/n_k f-evaluations) find a subinterval $[\alpha_{k+1}, \beta_{k+1}] \subset [\alpha_k,\beta_k]$ which contains x^* and no intersection point lies in its interior. Since $n_k = O(n/2^{2^k})$, it follows that the entire effort of phase k is $O(\log n + (\text{loglog}(n/2^{2^k}))(\log n)/2^k)$. Since $k \leq \text{loglog}n$, it follows that the total effort is $O(\log n \text{ loglog}n)$.

4. Probabilistic two-variable algorithm

In this section we indicate a probabilistic $O(\log n)$ algorithm for 2-variable LP which can run on the WRAM model. It is based on finding the maximum of n numbers a.s. in $O(1)$ time on the WRAM model [M4]. Specifically, we take a random sample of $\sqrt{n}$ lines and find all their points of intersection. This takes constant-time. We now wish to locate x^* in an interval $[\alpha,\beta]$ that contains no intersection point in its interior, so that the sample-maximum will be known at x^* (and in fact over the entire $[\alpha,\beta]$). This is carried out by a random search over the set of intersection points. Specifically, pick an intersection point x at random and evaluate $f(x)$

(a.s. in constant-time) and the one-sided derivatives $f_-(x)$, $f_+(x)$ (also a.s. in constant-time). Discard a portion of the set according to the position of x relative to x^*, as learned from $f_-(x)$ and $f_+(x)$ (see [M2]). It follows that this search will take an expected number of $O(\log n)$ evaluations. Knowing the line L which determines the sample-maximum over $[\alpha,\beta]$, we now find the intersection points of L with all the other lines, and search for an interval $[\alpha',\beta'] \subset [\alpha,\beta]$ which contains x^* and no such intersection point in its interior. This is done essentially in the same way we do the first search, i.e., in $O(\log n)$ time. Now we know which lines lie above the sample-maximum at x^*. Their number is expected to be $O(\sqrt{n})$ and we repeat the entire procedure unless there are less than $\sqrt{n}$ lines left. However, with probability approaching 1 (as n tends to infinity), this will not repeat more than twice. When we are left with less than $\sqrt{n}$ lines, we find the minimum directly in one search step (i.e., $O(\log n)$ time). The entire algorithm takes an expected $O(\log n)$ time.

5. Deterministic and probabilistic algorithms for d variables

For improving the 3-variable case we need to consider the 2-variable case with any number $p(n \leq p \leq n^2)$ of processors. Let $r = 2p/n$. The maximum can be found in $O(\log((\log n)/\log r))$ time [SV, V]. Using this result, we solve 2-variable LP deterministically in $O(\log(n/r)\log\log(n/r) + \log\log(n/r)\cdot\log n/\log r)$ time and probabilistically in simply $O(\log n/\log r)$ time. These bounds imply that the 3-variable case is solvable deterministically in $O(\log^2 n \log\log n)$ time and probabilistically in $O(\log n \log\log n)$ time. These bounds are then generalized to $O(\log^{d-1} n \log\log n)$ deterministically, and $O(\log n(\log\log n)^{d-2})$ probabilistically. The difference, which becomes more and more dramatic as d increases, is due to the fact that a random

search (which is asymptotically as good as a deterministic one) does not require median-finding. The latter is a relatively costly operation when we are dealing with deterministic parallel algorithms.

6. The optimal explosion problem

We consider the following problem: Given are n objects, flying in 3-space at various speeds with straight line trajectories. Find a time and a place at which a smallest ball encloses all the objects. This ball determines the size of the smallest bomb that can destroy all the objects, and the optimal time and place for the explosion. This problem is not equivalent to that of finding the smallest ball enclosing n points in 4-space. Also, a related problem of finding the smallest ball that touches all the n trajectories (i.e., each object touches the ball at some time) is easier than our present problem. The "static" problems of smallest balls can be solved in linear-time, whenever the dimension is fixed, by methods resembling those of [M3]. A seemingly related problem, namely, recognizing whether any two of n given lines in R^3 intersect, is currently not known to be solvable in $o(n^2)$ time [Do]. It is thus interesting that our optimal explosion problem can be solved in (sequential) $O(n \log^4 n \log\log n)$ time deterministically and $O(n \log^2 n (\log\log n)^2)$ probabilistically. However, these low-order complexities follow by combining two powerful results: (i) the general scheme of using parallel algorithms in the design of serial ones, as proposed in [M1], and (ii) fast parallel algorithms for the static smallest ball problem in R^3. The bound can be improved if randomization is allowed.

Following is a brief sketch of the algorithm. Let $F(t)$ denote the radius of the smallest ball enclosing the objects at time t. The function $F(t)$ is convex and we know how to evaluate it at any t in $O(n)$ time [M3].

Moreover, we develop in the present paper a parallel algorithm for evaluating $F(t)$ in $O(\log^3 n \log\log n)$ time with n processors. Denote the minimum by t^*. Simulating this parallel algorithm, we can run in $O(\log^3 n \log\log n)$ stages with t not specified but confined to an interval $[\alpha_k, \beta_k]$ during stage k. Typically, there will also be a value $\gamma_k (\alpha_k < \gamma_k < \beta_k)$ such that $F(\alpha_k) > F(\gamma_k)$ and $F(\beta_k) > F(\gamma_k)$ (and therefore $\alpha_k < t^* < \beta_k$). The task of each processor is the same for all $t \in [\alpha_k, \gamma_k)$ and for all $t \in [\gamma_k, \beta_k]$. However, each processor may produce a small number (independent of n) of critical values of t which determine how the algorithm should proceed in the succeeding stage. Denoting these values by $\alpha_k = t_1 < \ldots < t_m = \beta_k$, ($\gamma_k = t_\ell$ for some ℓ), we then search for j such that $F(t_{j-1}) > F(t_j)$ and $F(t_{j+1}) > F(t_j)$. The next interval is $[\alpha_{k+1}, \beta_{k+1}] = [t_{j-1}, t_{j+1}]$. Since F is convex, we can perform a "Fibonacci search" over the set $\{t_1, \ldots, t_m\}$ which amounts to $O(\log m)$ evaluations of F. However, it follows that $m \leq 18n$, since each processor may need to compare two pairs of polynomials of t of degree 9 (one pair over $[\alpha_k, \gamma_k)$ and another over $[\gamma_k, \beta_k]$), thereby producing at most 18 critical values. The effort per stage is therefore $O(n \log n)$ and the total is $O(n \log^4 n \log\log n)$. We should mention that this is in fact the number of times we may need to solve single-variable polynomial equations of degree not greater than 9. However, the effort involved in solving a single equation is independent of n.

A probabilistic algorithm, based on probabilistically finding a static smallest ball in $O(\log n (\log\log n)^2)$ parallel time, is utilized in designing a probabilistic $O(n \log^2 n (\log\log n)^2)$ sequential algorithm for the explosion problem.

References

[BH] A. Borodin and J. E. Hopcroft, "Routing, merging and sorting on parallel models of computation," STOC 1982, 338-344.

[Da] G. B. Dantzig, Linear programming and extensions, Princeton University Press, Princeton, NJ, 1963.

[Do] D. Dobkin, private communication, November 1982.

[DLR] D. Dobkin, R. J. Lipton and S. Reiss, "Linear programming is log-space hard for P," Inf. Proc. Lett 8 (1979) 96-97.

[GSS] L. M. Goldschlager, R. A. Shaw and J. Staples, "The maximum flow problem is log space complete for P," Theor. Comp. Sci. 21 (1982) 105-111.

[K] L. G. Khachiyan, "A polynomial algorithm in linear programming," Soviet Math. Dokl. 20 (1979) 191-194.

[M1] N. Megiddo, "Applying parallel computation algorithms in the design of serial algorithms," FOCS 1981, 399-408.

[M2] N. Megiddo, "Linear-time algorithms for linear programming in R^3 and related problems," FOCS 1982, 329-338.

[M3] N. Megiddo, "Solving linear programming in linear time when the dimension is fixed," research report, April 1982.

[M4] N. Megiddo, "Parallel algorithms for finding the maximum and the median almost surely in constant-time" (preliminary report) October 1982.

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

[V] L. G. Valiant, "Parallelism in comparison problems," SIAM J. Comput. 4 (1975) 348-355.

[YL] B. Yamnitsky and L. A. Levin, "An old linear programming algorithm works in polynomial time," FOCS 1982, 327-328.