A Deterministic Poly(log log N)-Time N-Processor Algorithm for Linear Programming in Fixed Dimension

Miklos Ajtai[*] and Nimrod Megiddo[†]

February 1992; revised November 1994

Abstract. It is shown that for any fixed number of variables, the linear programming problems with n linear inequalities can be solved deterministically by n parallel processors in sub-logarithmic time. The parallel time bound (counting only the arithmetic operations) is $O((\log\log n)^d)$ where d is the number of variables. In the one-dimensional case this bound is optimal. If we take into account the operations needed for processor allocation, the time bound is $O((\log\log n)^{d+c})$ where c is an absolute constant.

1. Introduction

The general linear programming problem is known to be P-complete [6] so it is interesting to investigate the parallel complexity of special cases. One important case is when the number of variables (the dimension) d is fixed while the number of inequalities n grows. Megiddo [11] showed that this problem can be solved in $O(n)$ time for any fixed d. Clarkson [4] and Dyer [8] improved the dependence of the constant on d. The general search technique proposed in [11] provides poly-logarithmic algorithms with n processors for any fixed d (see [12]). Deng [5] gave an $O(\log n)$ algorithm with $n/\log n$ processors for the case $d = 2$. It was not known previously whether the problem can be solved in $o(\log n)$ time with n processors for any $d > 1$. Alon and Megiddo [3] showed, however, that on a probabilistic CRCW-PRAM with n processors the problem can be solved by a Las Vegas algorithm almost surely in constant time. In this paper we show for the first time that for any d the problem can be deterministically solved in $O((\log\log n)^d)$ time on n processors, if we count only the arithmetic operations. If we take into account the steps necessary for processor allocation then our time bound is $O((\log\log n)^{d+c})$, where

[*]IBM Almaden Research Center, 650 Harry Road, San Jose, California 95120-6099.

[†]IBM Almaden Research Center, 650 Harry Road, San Jose, California 95120-6099, and School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel. Research supported in Part by ONR Contracts N00014-91-C-0026 and N00014-94-C-0007.

c is an absolute constant. We describe our model of computation in section 3. We note that the simple case of $d = 1$ is equivalent to the problem of finding the maximum of n elements, which requires $\Omega(\log \log n)$ time on n processors.

2. Preliminaries

We first review some known facts about expander graphs. Let $G = (V, E)$ be any graph. For any nonnegative integer r, denote by $G^r = (V, E^r)$ a graph where $(u, v) \in E^r$ if and only if in G there exists a path of length less than or equal to r from u to v. For any $S \subset V$, the r-neighborhood of S, $N_r(S) = N_r(S; G)$, is defined to be the set of all vertices v such that either $v \in S$ or there exists a $u \in S$ with $(u, v) \in E^r$.

Definition 2.1. A graph $G = (V, E)$ is called *an expander with expansion coefficient* α if for every $S \subset V$ such that $|S| \leq \frac{1}{2}|V|$, we have $|N_1(S)| \geq \alpha|S|$.

The work of Gabber and Galil [9] provides for every $n = m^2$, $m = 1, 2, ...$, an explicit construction of a 6-regular graph which is a 1.03-expander. For any sufficiently large n we can get a 1.02-expander graph on n vertices with maximum degree less than 7, by taking first 6-regular 1.03-expander on n' vertices, where n' is the smallest square greater than n and then discarding $n' - n$ arbitrary vertices from it. Let $\epsilon = 0.02$

Proposition 2.2. *For every sufficiently large positive integer n and for every positive integer r, there exists a graph $G = (V, E)$ on n vertices with maximum degree less than 7^r, such that for every $S \subset V$,*

$$|N_1(S; G)| > \min\{(1 + \epsilon)^r |S|, n/2\} \ .$$

Proof: Let G_0 be a graph on n vertices with a maximum degree of 6 with expansion coefficient $\alpha > 1.1$. As we have noted already an explicit construction for such a graph is given in [9]. In section 6 we describe how the construction can be carried out on a $CRCW - PRAM$ with n processors in a constant number of steps where each processor can perform arithmetic operations on numbers not greater than n. Let $G = (G_0)^r$. The maximum degree d_r in G_r is not greater than $\sum_{i=1}^{r} 6^i < 7^r$. Moreover, for every j, if $|N_j(S; G)| \leq n/2$ then $|N_{j+1}(S; G)| > (1 + \epsilon)|N_j(S; G)|$, and the proof follows by induction. ∎

Corollary 2.3. *In the graph $(G_0)^r$, if A and B are sets of vertices with cardinalities greater than $n/(1 + \epsilon)^{r/2}$, then there exists an edge between them.*

Proof: It follows from Corollary 2.2 that

$$|N_{r/2}(A;G_0)| > \min\{(1+\epsilon)^{r/2}|A|, n/2\} = \min\{n, n/2\} = n/2$$

and similarly, $N_{r/2}(B;G_0) > n/2$. This implies that $N_{r/2}(A;G_0) \cap N_{r/2}(B;G_0) \neq \emptyset$ and hence in G_0 there is a path of length less than or equal to r between A and B or, equivalently, an edge of $(G_0)^r$. ▮

Let $c = 2\log_{1+\epsilon} 7$, so that $(1+\epsilon)^{c/2} = 7$.

Proposition 2.4. *For every sufficiently large n and for every positive integer r, there exists a graph G of degree less than $d = 7^r$ with the following property: if $t^c = d$, then every two disjoint sets A and B of vertices of G, such that $|A| = |B| = n/t$, are connected by an edge.*

Proof: The proof follows directly from Corollary 2.3 since $t = d^{1/c} = 7^{r/c} = (1+\epsilon)^{r/2}$. ▮

Corollary 2.5. *In the expander of Proposition 2.4, for every two disjoint sets of vertices A and B of vertices such that $|A| = |B| = 2n/t$, there exist more than n/t edges between A and B.*

Proof: Suppose, on the contrary, that the number of edges between A and B is less than or equal to n/t. Let A_1 be the subset of A consisting of those vertices that are not adjacent to any vertex in B. Since $|A_1| \geq n/t$, by Proposition 2.4 there is an edge between A_1 and B, hence a contradiction. ▮

3. Linear programming in the plane

Consider the linear programming problem with two variables in the form:

$$(P^2)\qquad \begin{array}{rll} \text{Minimize} & y & \\ \text{subject to} & y \geq a_i x + b_i & (i \in N_+) \\ & y \leq a_i x + b_i & (i \in N_-) \\ & \ell \leq x \leq h\ , & \end{array}$$

where $|N_+| + |N_-| = n$ and $\{\ell, h\} \subset [-\infty, \infty]$. Any two-variable linear programming problem can be reduced to this form in $O(\log\log n)$ time with n processors. The algorithm proposed by Dyer [7] and Megiddo [10] provides a method of discarding 1/4 of the set of constraints with an effort of computing one median and two maxima in sets of at most n elements. It was shown by Ajtai, Komlós, Steiger, and Szemerédi [2] that selection can be done in $O(\log\log n)$ time in Valiant's parallel comparison tree model with n processors. The selection steps of our algorithm are implemented in this model. All the other steps can be implemented on a CRCW-PRAM. Deng [5] gave a parallel algorithm which runs in $O(\log n)$ time, using $O(n/\log n)$ processors. In fact, his algorithm applies the procedure of [7; 10] until the number of remaining constraints allows for computation of the entire convex hull in $O(\log n)$ time with $O(n/\log n)$ processors. Such an approach cannot yield an $o(\log n)$ time bound. Our approach discards *increasing* proportions of the set of remaining constraints without resorting to the computation of the entire convex hull of the remaining set at a relatively early stage.

Suppose we are left with the lines

$$L_i = \{(x,y) \mid y = a_i x + b_i\} \quad (i \in N'_+ \cup N'_-)$$

where $N'_+ \subset N_+$ and $N'_- \subset N_-$. Let $n = |N_+| + |N_-|$ denote the revised number of constraints, and we continue to employ p processors (where p is the initial number of constraints). We now describe how a large number of constraints can be further discarded. Denote $q = p/n$ (where p is the initial number of processors). The treatment of the two classes of constraints is very similar so we describe just the case of N'_+.

Let d be the largest power of 7 that is smaller than $q^{c/(c+1)}$. Let t be defined by $t^c = d$. Consider an expander graph $G = (V, E)$ of degree smaller than d, with vertices corresponding to the lines in N'_+, that has the properties asserted in Proposition 2.4 and Corollary 2.5.

For every edge $(i,j) \in E$ $(i,j \in N'_+)$, if L_i and L_j are not parallel, consider the intersection coordinate $\xi_{ij} = -(b_i - b_j)/(a_i - a_j)$. Denote by C the set of these intersection points. Obviously, $|C| \le \frac{1}{2}nd < p$, so all the points in C can be computed in constant time with our p processors. Partition C into intervals by points $-\infty = a_0 < x_1 < x_2 < \cdots < x_s = \infty$, so that each interval $[x_{i-1}, x_i]$ $(i = 1, \ldots, s)$ contains less than n/t points. This can be achieved with $s < nd/(n/t) = t^{c+1} = q$.

Proposition 3.1. *If A and B are disjoint sets of lines, each containing at least $2n/t$ elements, then there does not exist a k $(1 \le k \le s)$ such that $\xi_{ij} \in [x_{k-1}, x_k]$ for all $L_i \in A$ and $L_j \in B$.*

Proof: The proof follows from the fact that the number of such intersection points that are also in C is at least n/t by Corollary 2.5, while each interval contains less than n/t points of C. ∎

Given the set C, we wish to determine in which of the intervals $[x_{k-1}, x_k]$ an optimal solution x^* might lie. Denote

$$f_+(x) = \max\{a_i x + b_i \mid i \in N'_+\}$$
$$f_-(x) = \min\{a_i x + b_i \mid i \in N'_-\} \ .$$

Note that an optimal solution must satisfy $x^* \in [\ell, h]$, $f_+(x^*) \le f_-(x^*)$, and $f_+(x^*)$ is minimal. Since $f_+(x)$ and $f_+(x) - f_-(x)$ are convex, we can test any value of x with at most three computations of a maximum in a set of cardinality n (see [10]), and conclude with one of the following possibilities: (i) the problem has no feasible solution, (ii) x is an optimal solution, (iii) if x^* is an optimal solution, then $x^* > x$, (iv) if x^* is an optimal solution, then $x^* < x$. Since $p/s > n$, we have at least n processors per point x_k, so in $O(\log\log n)$ time we can locate the optimum in one of our intervals.

Suppose we have identified an interval $[u, v]$ where the minimum is attained, and there are less than n/t intersection points of C over $[u, v]$. Consider the orders induced on the set of lines by their intersections with the lines $\{x = u\}$ and $\{x = v\}$. Call them the u-order the v-order, respectively. For every k ($k = 1, \ldots, n$), denote by U_k the set of the k lowest lines in the u-order and denote by V_k the set of the k lowest lines in the v-order.

Proposition 3.2. *For every k ($k = 1, \ldots, n$), the symmetric difference*

$$U_k \ominus V_k = (U_k \setminus V_k) \cup (V_k \setminus U_k)$$

contains at most $4n/t$ lines.

Proof: Since $|U_k \setminus V_k| = |V_k \setminus U_k|$, it follows that if, on the contrary, $|U_k \ominus V_k| > 4n/t$, then

$$|U_k \setminus V_k| = |V_k \setminus U_k| > 2n/t \ .$$

Note that all the intersections ξ_{ij} of a line $L_i \in U_k \setminus V_k$ with a line $L_j \in V_k \setminus U_k$ must be in $[u, v]$, in contradiction to 3.1 ▮

.

Let $\bar{U_k}$ and $\bar{V_k}$ be the complements of U_k and V_k, respectively, in the set of all n lines.

Proposition 3.3. *For $k = n - 4n/t$, there exists at least one line in $\bar{U_k} \cap \bar{V_k}$.*

Proof: The claim is trivial if $U_k = V_k$; otherwise, since $U_k \ominus V_k = \bar{U_k} \ominus \bar{V_k}$, by Proposition 3.2,

$$|\bar{U_k} \cap \bar{V_k}| = |\bar{U_k} \cup \bar{V_k}| - |U_k \ominus V_k| \geq (4n/t + 1) - 4n/t = 1 \ .$$

▮

Enter Figure 2 here

Proposition 3.4. *Let $k = n - 4n/t$. If L is any line in $\bar{U}_k \cap \bar{V}_k$, then over the interval $[u, v]$, the line L lies above at least $n - 8n/t$ lines.*

Proof: The line L lies above every other line, except possibly for some lines in $\bar{U}_k \cup \bar{V}_k$; but $|\bar{U}_k \cup \bar{V}_k| \leq 8n/t$. ∎

Corollary 3.5. *The number of lines can be reduced from n to no more than $8n/t$ in $O(\log\log n)$ time.*

Proof: A line $L \in \bar{U}_k \cap \bar{V}_k$ $(k = n - 4n/t)$ can be found as follows. Compute the set $\bar{U}_k$ by selecting the k'th smallest element relative to the u-order, and then find the line L which is the maximum in the $\bar{U}_k$ relative to the v-order. Given L, we can compare it with all of the other lines and discard those lines which are smaller in both orders. The number of remaining lines will be at most $8n/t$. ∎

Theorem 3.6. *The linear programming problem with two variables can be solved in $O((\log\log n)^2)$ time.*

Proof: The scheme we have described so far reduces the number of constraints from n to $8n/t(n)$, so it works only when $8n/t(n) < n$. In order to satisfy the latter condition, we start the algorithm by running a constant number of iterations of the algorithm of [7; 10], where 1/4 of the set of constraints is discarded in each iteration. This constant number is determined from $t(n) = (p/n)^{1/(c+1)} > 8$, *i.e.*, $n/p < 8^{-c-1}$ and the number is $\log_{3/4} 8^{-c-1}$.
Recall that $t = t(n) = (p/n)^{1/(c+1)}$, so the value of n is reduced in one iteration to

$$n' = 8n^{1+1/(1+c)}p^{-1/(c+1)} ,$$

so the next value of t is

$$t' = \left(\tfrac{1}{8}p^{1+1/(c+1)}n^{-1-1/(1+c)}\right)^{1/(c+1)} = 8^{-1/(c+1)}t^{1+1/(1+c)} .$$

Thus, after k iterations, the value of t is

$$t^{(k)} = \beta^{1+\gamma+\cdots+\gamma^{k-1}}t^{\gamma^k} ,$$

where $\beta = 8^{-1/(c+1)}$ and $\gamma = 1+1/(c+1)$, and this implies that the number of iterations is $O(\log\log n)$. ∎

4. The three-dimensional case

The linear programming problem with three variables is formulated as follows.

$$(P^3) \qquad \begin{array}{rll} \text{Minimize} & z & \\ \text{subject to} & z \geq a_ix + b_iy + c_i & (i \in N_+) \\ & z \leq a_ix + b_iy + c_i & (i \in N_-) \\ & 0 \leq a_ix + b_iy + c_i & (i \in N_0) , \end{array}$$

where $|N_+| + |N_-| + |N_0| = n$.

As in the two-dimensional case, we proceed by discarding increasing proportions of the set of constraints. The three sets N_+, N_-, and N_0 can be handled independently. We describe only the processing of N_+. Suppose we are currently left with n planes

$$P_i = \{(x,y,z) \mid z = a_ix + b_iy + c_i\} \quad (i \in N'_+ \subset N_+)$$

and there are p processors. For every pair of planes (P_i, P_j), if the planes are not parallel, let

$$L_{ij} = \{(x,y) \mid a_ix + b_iy + c_i = a_jx + b_jy + c_j\} ,$$

i.e., L_{ij} is the projection of the line of intersection of P_i and P_j into the (x,y)-plane.

We now use an expander graph $G = (N'_+, E)$ of maximum degree less than t_1^c (the dependence of t_1 on n and p will be explained later) whose vertices represent the planes, with the property that if A and B are disjoint subsets of N'_+, each of cardinality of at least $2n/t_1$, then the number of edges between A and B is at least n/t_1. Let

$$D = \{L_{ij} \mid (i,j) \in E\} \ .$$

Note that the number of pairs (i,j) such that $L_{ij} \in D$ is less than nt_1^c.

We now use an expander graph $G' = (D, E')$ (*i.e.*, the vertices correspond to the lines in D) with maximum degree less than t_2^c (the dependence of t_2 on n and p will be explained later) so that between any two sets, each of cardinality of at least $2|D|/t_2$, the number of edges is at least $|D|/t_2$. In view of Corollary 2.5, this is true if $t_2 \le (p/(nt_1^c))^{1/(c+1)}$, since $|D| \le nt_1^c$.

Denote by C the set of intersection points of pairs of lines corresponding to the edges of G'. We proceed as in the two-dimensional case (as if D were the total set of lines). We partition the x-axis into t_2^{c+1} intervals in the same way, and find one interval $[u,v]$ of the partition such that, without loss of generality, the given instance of (P^3) may be restricted to the stripe[1] $\{(x,y) \mid u \le x \le v\}$. The testing algorithm that we use to select the required stripe is described in detail in [11; section 4, pp. 123–126]. We can decide with this testing algorithm where is the solution of (P^d) relative to a given hyperplane, that is, whether there is a solution on the hyperplane and if not then which of the two halfspace determined by the hyperplane contains the solution. The algorithm uses only the solution of three instances of the d-1-dimensional problem if the hyperplane is in the d-dimensional space. (We will use this algorithm for the solution of the d-dimensional problem too). Now we may conclude that the required stripe can be selected by solving at most 3 instances of the 2-dimensional problem. We also know that the two orders on D (induced by the intersection points with $\{(x,y) \mid x = u\}$ and $\{(x,y) \mid x = v\}$) are almost the same, in the sense that for every k ($k = 1, \ldots, |D|$),

$$|U_k \ominus V_k| \le 4|D|/t_2$$

(see Proposition 3.2).

[1]Note that we may have $u = -\infty$ or $v = \infty$.

Enter Figure 3 Here

Consider the following two partitions of D into $r < t_2/8$ intervals of length $\delta > 8|D|/t_2$: (i) intervals $I_1, \ldots, I_r$ relative to the u-order, and (ii) intervals $J_1, \ldots, J_r$ relative to the v-order.

Proposition 4.1. *For every k $(k = 1, \ldots, r)$, $I_k \cap J_k \neq \emptyset$.*

Proof: Since $|U_k \ominus V_k| \leq 4|D|/t_2$ $(k = 1, \ldots, |D|)$, and

$$I_k = U_{k\delta} \setminus U_{(k-1)\delta} \quad \text{and} \quad J_k = V_{k\delta} \setminus V_{(k-1)\delta} \ ,$$

it follows that

$$|I_k \ominus J_k| \leq |(U_{k\delta} \ominus V_{k\delta}) \cup (U_{(k-1)\delta} \ominus V_{(k-1)\delta})| \leq 2(4|D|/t_2) < \delta \ ,$$

so

$$|I_k \cap J_k| \geq |I_k \cup J_k| - |I_k \ominus J_k| > 0 \ .$$

∎

It follows that for each k ($k = 1, \ldots, r$), there exists a line $\ell_k \in I_k \cap J_k$, so that the members of $M = \{\ell_1, \ldots, \ell_r\}$ do not intersect in the stripe $\{(x,y) \mid u \leq x \leq v\}$. These lines partition the stripe into "trapezoids."

Remark 4.2. A suitable set M can be constructed on a CRCW-PRAM in $O(\log\log n)$ time. First, the intervals are constructed by solving r selection problems. Suppose each member of D knows the intervals it belongs to relative to the two orders. Each member of $I_k \cap J_k$ now attempts to write its name in a cell representing ℓ_k, and one succeeds.

Proposition 4.3. *There are at most 5δ pairs (i,j) (not necessarily in E) such that the trapezoid bordered by the ℓ_k and ℓ_{k+1} is intersected by L_{ij}.*

Proof: The proof follows from the fact that this trapezoid may intersect only the lines in the set

$$I_k \cup I_{k+1} \cup J_k \cup J_{k+1} \cup (U_{(k+1)\delta} \ominus V_{(k+1)\delta}) \cup (U_{(k-1)\delta} \ominus V_{(k-1)\delta})$$

whose cardinality is at most $4\delta + 2(4|D|/t_2)$. ∎

Next, we identify one trapezoid to which the problem (P^3) may be restricted without loss of generality. Furthermore, we can divide this trapezoid into two triangles and restrict attention to one of them which we denote by T.

Proposition 4.4. *If $5\delta < n/t_1$, then for every pair (A, B) of disjoint sets of planes, such that $|A|, |B| > 2n/t_1$, there is at least one line L_{ij} such that $P_i \in A$ and $P_j \in B$, and L_{ij} does not intersect T.*

Proof: We know that for such A and B, there are at least n/t_1 lines L_{ij} in D such that $P_i \in A$ and $P_j \in B$. On the other hand, by Proposition 4.3 T is intersected by at most 5δ lines L_{ij}. ▮

Thus, we need to choose t_1 and t_2 so that $40|D|/t_2 < n/t_1$. Hence, we require $40t_1^c/t_2 < 1/t_1$, *i.e.*, $40t_1^{c+1} < t_2$, and since $t_2 < (p/|D|)^{1/(c+1)}$, it suffices that

$$t_1 < \tfrac{1}{40}\left(\frac{p}{n}\right)^{\frac{1}{(c+1)(c+2)}} \quad \text{and} \quad t_2 = \left(\frac{p}{n}\right)^{\frac{1}{(c+2)}} .$$

At each of the three vertices[2] of T, there is a natural linear order on the set of hyperplanes induced by the z-coordinate. We may apply the argument we used in the two-dimensional case to any pair of orders as we did with the u-order and the v-order. For every k ($k = 1, \ldots, n$), let U_k, V_k, and W_k denote the sets of k lowest planes relative to these three orders.

Proposition 4.5. *For every k ($k = 1, \ldots, n$), the set*

$$S_k = (U_k \ominus V_k) \cup (U_k \ominus W_k) \cup (V_k \ominus W_k)$$

contains at most $12n/t_1$ elements.

Proof: Each member of $U_k \setminus V_k$ intersects each member of $V_k \setminus U_k$ in T, each member of $V_k \setminus W_k$ intersects each member of $W_k \setminus V_k$ in T, and each member of $W_k \setminus U_k$ intersects each member of $U_k \setminus W_k$ in T. Since $|U_k \setminus V_k| = |V_k \setminus U_k|$, $|V_k \setminus W_k| = |W_k \setminus V_k|$, and $|W_k \setminus U_k| = |U_k \setminus W_k|$, it follows from our choice of t_1 that the cardinality of each of these six sets is not greater than $2n/t_1$. ▮

Proposition 4.6. *For $k = n - 12n/t_1$, $\bar{U}_k \cap \bar{V}_k \cap \bar{W}_k \neq \emptyset$.*

Proof: If $U_k = V_k = W_k$ the claim is trivial; otherwise,

$$|\bar{U}_k \cap \bar{V}_k \cap \bar{W}_k| \geq |\bar{U}_k \cup \bar{V}_k \cup \bar{W}_k| - |S_k| \geq (12n/t_1 + 1) - 12n/t_1 = 1 .$$

▮

[2]The case of an "infinite" triangle can be easily handled too.

Proposition 4.7. *If $k = n - 12n/t_1$ and $P \in \bar{U}_k \cap \bar{V}_k \cap \bar{W}_k$, then over the triangle T, the plane P lies above at least $n - 36n/t_1$ planes.*

Proof: The plane P lies above every plane in $U_k \cap V_k \cap W_k$, but

$$|U_k \cap V_k \cap W_k| = n - |\bar{U}_k \cup \bar{V}_k \cup \bar{W}_k| \geq n - 3(12n/t_1) \ .$$

∎

Enter Figure 5 Here

Corollary 4.8. *The number of planes can be reduced from n to no more than $36n/t_1$ in $O((\log\log n)^2)$ time.*

Proof: The sets $\bar{U}_k$, $\bar{V}_k$, and $\bar{W}_k$ can be computed with a selection algorithm and then it can be decided separately (and simultaneously) for each member of the union of these sets whether it satisfies the conditions of Proposition 4.7. Since we need to solve linear programming problems with two variables and we have enough processors for solving all these problems in parallel, Theorem 3.6 applies, so the effort for one iteration is $O((\log\log n)^2)$. ∎

Theorem 4.9. *The linear programming problem with three variables can be solved in* $O((\log\log n)^3)$ *time.*

Proof: The value of n is reduced in one iteration to $36n/t_1$, where $t_1 = (p/n)^{1/((c+1)(c+2))}$. As in the proof of Theorem 3.6, the next value of t_1, is

$$t_1' = 36^{-1/((c+1)(c+2))} t_1^{1+1/((c+1)(c+2))} ,$$

so the number of iterations is $O(\log\log n)$. This implies our claim. ∎

5. The general d-dimensional case

We now consider the general linear programming problem with d variables, which we formulate as follows.

$$(P^d) \qquad \begin{array}{rll} \text{Minimize} & y & \\ \text{subject to} & y \geq \boldsymbol{a}_i^T\boldsymbol{x} + b_i & (i \in N_+) \\ & y \leq \boldsymbol{a}_i^T\boldsymbol{x} + b_i & (i \in N_-) \\ & 0 \leq \boldsymbol{a}_i^T\boldsymbol{x} + b_i & (i \in N_0) \ , \end{array}$$

where $\boldsymbol{a}_i \in R^{d-1}$ ($i \in N_+ \cup N_- \cup N_0$ and $|N_+| + |N_-| + |N_0| = n$).

5.1 Hyperplane queries

In general, our algorithm works recursively in the dimension. First, as explained in [11], a linear programming algorithm for problems with $d-1$ variables can be used as an oracle for deciding the position of the set of optimal solutions, if any, relative to any given hyperplane. More precisely, it can be used to solve the following problem:

Problem 5.1. Given an instance of (P^d) and a hyperplane $H = \{\boldsymbol{x} \in R^{d-1} \mid \boldsymbol{a}^T\boldsymbol{x} = b\}$, decide whether (i) the optimal solutions of (P^d), if any, may be assumed to lie in $H_+ = \{\boldsymbol{x} \in R^{d-1} \mid \boldsymbol{a}^T\boldsymbol{x} > b\}$, (ii) the optimal solutions of (P^d), if any, may be assumed to lie in $H_- = \{\boldsymbol{x} \in R^{d-1} \mid \boldsymbol{a}^T\boldsymbol{x} < b\}$, or (iii) a final conclusion can be reached that either H contains an optimal solution, the problem is unbounded on H, or the problem is infeasible.

The conclusion in (iii) is reached when the solution of (P^d) with the additional constraint $\boldsymbol{x} \in H$ yields a solution of (P^d), or the problem is infeasible and the "amount of infeasibility" is minimized on H.

5.2 Locating the solution in a "small" simplex

We now introduce a problem that plays the key role in the algorithm, but we first need to define an oracle for minimizing a function:

Definition 5.2. Consider a function $f : R^{d-1} \to R \cup \{-\infty\}$. By an *oracle for* f we mean a mechanism that, when presented with a hyperplane H in R^{d-1}, returns information in one of the following forms: (i) either the minimum of f lies in H_+ or f is unbounded from below on H_+, (ii) either the minimum of f lies in H_- or f is unbounded from below on H_-, (iii) either the minimum of f lies in H or f is unbounded from below on H.

Problem 5.3. Given are: an oracle for a function f as in Definition 5.2, a number p (of processors), and hyperplanes $H_k = \{(\boldsymbol{x}, y) \in R^d \mid y = \boldsymbol{a}_k^T \boldsymbol{x} + b_k\}$ $(k = 1, \ldots, n)$. Either find some hyperplane H in R^{d-1} such that the minimum of f lies in H (or f is unbounded on H), or find d halfspaces $F_k = \{\boldsymbol{x} \in R^{d-1} \mid \boldsymbol{c}_k^T \boldsymbol{x} + d_k \geq 0\}$ $(k = 1, \ldots, d)$ such that

(i) either the minimum of f lies in the "simplex"[3] $\Delta = F_1 \cap \cdots \cap F_d$ or f is unbounded from below on Δ, and

(ii) at most n/t $(t = t(p/n; d))$ pairs (H_i, H_j) of hyperplanes intersect over Δ. (The value of $t(p/n; d)$ will be derived later.)

Enter Figure 6 Here

[3]In general, this intersection may be unbounded.

Remark 5.4. When $d = 2$, the polyhedron is an interval which may extend to infinity in one direction. In higher dimensions, the polyhedron can be either a simplex or a simplicial cone. In any case, there will be at most d linear orders on the set of H_k's such that if H_i is above H_j in each of these linear orders, then H_i lies above H_j at every point of Δ.

Recall that in the case $d = 2$, we located an interval with $t_1 = (p/n)^{1/(c+1)}$, and in the case $d = 3$, we located a "triangle" T such that the number of pairs of planes that intersected over T was at most n/t_1, where $t_1 = (p/n)^{1/((c+1)(c+2))}$.

Denote

$$L_{ij} = \{\boldsymbol{x} \in R^{d-1} \mid \boldsymbol{a}_i^T \boldsymbol{x} + b_i = \boldsymbol{a}_j^T \boldsymbol{x} + b_j\} .$$

If H_i and H_j are not parallel, then L_{ij} is a hyperplane in R^{d-1}.

Let $G = (V, E)$ be an expander graph with maximum degree less than τ_1^c (the value of τ_1 will be determined later), whose vertices correspond to the hyperplanes H_k, with the property that every two disjoint subsets $A, B \subset V$ of cardinality n/τ_1 are connected by an edge. Hence, if A and B are subsets of V of cardinality at least $2n/\tau_1$, then the number of edges between A and B is at least n/τ_1. Let

$$D = \{L_{ij} \mid (i, j) \in E\} .$$

We have $|D| < n\tau_1^c$. Consider the function $f' : R^{d-2} \to R \cup \{-\infty\}$ defined by

$$f'(x_1, \ldots, x_{d-2}) = \inf_{x_{d-1}} f(x_1, \ldots, x_{d-1}) .$$

An oracle for f (in the sense of Definition 5.2) provides an oracle for f' when we extend any hyperplane H in R^{d-2} into a hyperplane in R^{d-1} described by the same equation. Thus, recursively, we can either (i) find a hyperplane $H \subset R^{d-2}$ which contains the minimum of f', and hence its extension into a hyperplane in R^{d-1} contains the minimum of f, or (ii) find $d-1$ halfspaces $F_k = \{\boldsymbol{x} \in R^{d-2} \mid \boldsymbol{c}_k^T \boldsymbol{x} + d_k \geq 0\}$ ($k = 1, \ldots, d-1$) with the properties described in Problem 5.3 with respect to the hyperplanes L_{ij} in D:

(i) either the minimum of f lies in $\Delta = F_1 \cap \cdots \cap F_{d-1}$ or f is unbounded from below on Δ, and

(ii) at most $|D|/\tau_2$ ($\tau_2 = t(p/|D|, d-1)$) pairs of L_{ij}'s from D intersect over Δ.

For $k = 1, \ldots, d-1$, let F'_k be the halfspace in R^{d-1} parallel to the (x_{d-1})-axis, obtained by extending the halfspace F_k into R^{d-1}. Thus, the polyhedron Δ is extended into a polyhedral "cylinder" Δ^o in R^{d-1} which contains the minimum of f. Furthermore, the number of pairs of L_{ij}'s intersecting Δ^o is at most $|D|/\tau_2$. Consider the $d-1$ linear orders induced on the set of L_{ij}'s by their x_{d-1} values at the $d-1$ vertices of Δ (or at infinity as explained above). For $j = 1, \ldots, d-1$ and $k = 1, \ldots, |D|$, denote by U_k^j the set of the k lowest hyperplanes relative to the j'th order.

Proposition 5.5. *For any i and j $(1 \le i < j \le d-1)$ and for every k $(k = 1, \ldots, |D|)$,*

$$|U_k^i \ominus U_k^j| \le 4|D|/\tau_2 \ .$$

Proof: As in Proposition 3.2,

$$U_k^i \ominus U_k^j = (U_k^i \setminus U_k^j) \cup (U_k^j \setminus U_k^i)$$

so if, on the contrary,

$$|U_k^i \ominus U_k^j| > 4|D|/\tau_2 \ ,$$

then

$$|U_k^i \setminus U_k^j| = |U_k^j \setminus U_k^i| > 2|D|/\tau_2 \ .$$

Since all the intersections of members of $U_k^i \setminus U_k^j$ with members of $U_k^j \setminus U_k^i$ intersect Δ^o, we reach a contradiction. ∎

We now consider $d-1$ partitions of D into $r < \tau_2/(4(d-1)(d-2))$ intervals of length $\delta > 4(d-1)(d-2)|D|/\tau_2$: for $i = 1, \ldots, d-1$, a partition into intervals $I_1^i, \ldots, I_r^i$ relative to the i'th order.

Proposition 5.6. *For every k $(k = 1, \ldots, r)$,*

$$I_k^1 \cap I_k^2 \cap \cdots \cap I_k^{d-1} \neq \emptyset \ .$$

Proof: We have

$$|U_k^i \ominus U_k^j| \le 4|D|/\tau_2 \quad \text{and} \quad I_k^j = U_{k\delta}^j \setminus U_{(k-1)\delta}^j \ .$$

Since

$$I_k^1 \cap I_k^2 \cap \cdots \cap I_k^{d-1} = \left(I_k^1 \cup I_k^2 \cup \cdots \cup I_k^{d-1}\right) \ \setminus \ \bigcup_{i<j} \left(I_k^i \ominus I_k^j\right) \ ,$$

and

$$I_k^i \ominus I_k^j \ \subseteq \ (U_{k\delta}^i \ominus U_{k\delta}^j) \cup (U_{(k-1)\delta}^i \ominus U_{(k-1)\delta}^j) \ ,$$

it follows that

$$\begin{aligned} \left|I_k^1 \cap I_k^2 \cap \cdots \cap I_k^{d-1}\right| &\ge \delta - \sum_{i<j} \left|U_{k\delta}^i \ominus U_{k\delta}^j\right| - \sum_{i<j} \left|U_{(k-1)\delta}^i \ominus U_{(k-1)\delta}^j\right| \\ &\ge \delta - 2\binom{d-1}{2} \cdot 4|D|/\tau_2 > 0 \ . \end{aligned}$$

∎

It follows that for each k $(k = 1, \ldots, r)$, there exists a hyperplane $L_k^* \in I_k^1 \cap \cdots \cap I_k^{d-1}$, such that the members of $M = \{L_1^*, \ldots, L_r^*\}$ do not intersect in the cylinder Δ^o. These hyperplanes partition Δ^o into "prisms."

Proposition 5.7. *There are at most $(2d-1)\delta$ pairs (i,j) such that the prism bordered by L_k^* and L_{k+1}^* is intersected by L_{ij}.*

Proof: The prism may be intersected only by hyperplanes in the set:

$$\bigcup_{j=1}^{d-1}\left(I_k^j \cup I_{k+1}^j\right) \;\cup\; \bigcup_{i<j}\left(U_{(k+1)\delta}^i \ominus U_{(k+1)\delta}^j\right) \;\cup\; \bigcup_{i<j}\left(U_{(k-1)\delta}^i \ominus U_{(k-1)\delta}^j\right) ,$$

whose cardinality is at most

$$2(d-1)\delta + 2\binom{d-1}{2}\cdot 4|D|/\tau_2 < (2d-1)\delta \ .$$

∎

We now find one prism that may be assumed to contain the minimum of f. In this process we might find a hyperplane which contains the minimum. We then divide the prism into $d-1$ "simplices" and restrict attention to one of them, which we now denote by Δ'.

Proposition 5.8. *If $2d^3|D|/\tau_2 < n/\tau_1$, then for every pair (A,B) of disjoint sets of hyperplanes H_k, such that $|A|,|B| > 2n/\tau_1$, there exists at least one L_{ij} such that $H_i \in A$ and $H_j \in B$, and L_{ij} does not intersect Δ'.*

Proof: We choose $\delta > 4(d-1)(d-2)|D|/\tau_2$ such that $(2d-1)\delta < 2n/\tau_1$. For A and B that satisfy our conditions, there are at least n/τ_1 L_{ij}'s in D such that $H_i \in A$ and $H_j \in B$. On the other hand, Δ' is intersected by at most $(2d-1)\delta$ L_{ij}'s. Under the assumption of the proposition, the latter is less than n/τ_1. ∎

Thus, we will choose τ_1 so that $2d^3|D|/\tau_2 < n/\tau_1$. On the other hand, $|D| < n\tau_1^c$, so it suffices that

$$\tau_1 < \left(\frac{\tau_2}{2d^3}\right)^{\frac{1}{c+1}} .$$

We are thus led to the expression for $\tau(d) = t(p/n, d)$ as follows. First, $\tau(1) = q^{1/(c+1)}$ (where $q = p/n$)). Next,

$$\tau(d) = \left(\frac{\tau(d-1)}{2d^3}\right)^{\frac{1}{c+1}} .$$

It follows that

$$\tau(d) = \frac{q^{\epsilon^d}}{2^{\epsilon+\cdots+\epsilon^{d-1}}\prod_{k=1}^d k^{3\epsilon^{d+1-k}}} ,$$

where $\epsilon = 1/(c+1)$ (not the ϵ of Section 3).

5.3 Discarding constraints

After we have located the solution of our linear programming problem in a small simplex, we can discard a large number of constraints as follows. As in the previous sections, we consider members of the three sets N_+, N_- and N_0 separately. Suppose we are left with a set $N'_+ \subseteq N_+$ of n hyperplanes, and we have now found a "simplex" Δ in R^{d-1}, which is known to contain the solution, and over which at most $(p/n)^{\tau(d)}$ pairs of hyperplanes intersect. We also know d linear orders over N'_+ (at "vertices" of Δ, such that if a hyperplane H lies above another hyperplane H' in all these orders, then H lies above H' over the entire set Δ. For $j = 1, \ldots, d$ and $k = 1, \ldots, n$, denote by U^j_k the set of the k lowest hyperplanes relative to the j'th order.

Proposition 5.9. *For every k ($k = 1, \ldots, n$), the set*

$$S_k = \bigcup_{i<j} (U^i_k \ominus U^j_k)$$

contains at most $d(d-1)n/\tau(d)$ elements.

Proof: The proof is similar to that of Proposition 4.5; each of sets $U^i_k \ominus U^j_k$ contains at most $2n/\tau(d)$ elements. ▮

Proposition 5.10. *For $k = n - d(d-1)n/\tau(d)$, $\bigcap_{j=1}^d \bar{U}^j_k \neq \emptyset$.*

Proof: The proof is similar to that of Proposition 4.6:

$$\left| \bigcap_{j=1}^{d} \bar{U}^j_k \right| \geq \left| \bigcup_{j=1}^{d} \bar{U}^j_k \right| - |S_k| \geq n - k - d(d-1)n/\tau(d) \ .$$

▮

Proposition 5.11. *If $k = n - d(d-1)n/\tau(d)$ and $H \in \bigcap_{j=1}^d \bar{U}^j_k$, then over the simplex Δ, the hyperplane H lies above at least $n - d^2(d-1)n/\tau(d)$ hyperplanes.*

Proof: The hyperplane H lies above every member of $\bigcap_{j=1}^d U^j_k$, but

$$\left| \bigcap_{j=1}^{d} U^j_k \right| = n - \left| \bigcup_{j=1}^{d} \bar{U}^j_k \right| \geq n - d^2(d-1)n/\tau(d).$$

▮

Corollary 5.12. *The number of planes can be reduced from n to no more than $d^2(d-1)n/\tau(d)$ in one phase where $(d-1)$-variable linear programs are solved in parallel, each with a linear number of processors.*

Theorem 5.13. *For any fixed d, the linear programming problem with d variables and n inequalities can be solved with n processors in $O((\log\log n)^d)$ time.*

Proof: Denote

$$C = C(d) = \frac{1}{2^{1+1/c}d^3(d!)^3} \ .$$

It follows from what we have proven that the value of n can be reduced in one iteration to $n/t(d)$, where

$$t(d) = t(d;p/n) = C(d)\left(\frac{p}{n}\right)^{\epsilon^d} \ .$$

After one iteration the new value of $t = t(d)$ is $t' = t^{1+\epsilon^d}$, so after k iterations, $t^{(k)} = t^{(1+\epsilon^d)^k}$. ▮

Remark 5.14. We note that the algorithm has to start with a constant number of iterations that reduce the number of constraints by discarding constant proportions until we get $t > 1$, *i.e.*, if initially $n = p = m$, we need to reduce n until $n < mC^{\epsilon^{-d}}$. In terms of d, the constant proportion is $O(3^{-d^2})$, hence the constant number iterations is $O(-\log C(d)3^{d^2})$, *i.e.*, $O(3^{d^2}d\log d)$. Then, the variable number of iterations is $O(\log\log n/(\log(1+\epsilon))$, *i.e.*, $O((c+1)^d\log\log n)$.

6. The problem of processor allocation

To describe our computational model, first we recall Valiant's comparison tree model used for measuring the complexity of sorting, or selection problems. We formulate it in a way which is suitable for further generalizations. Assume that we have an ordered set with n abstract elements and $n/2$ processors (n is even). At each step each processor receives two elements, compares them, and reports the result to a central processor. The central processor receives the results, and based on all the information received, decides which comparison should be made and by whom during the next round of comparisons. There is no restriction on the computing ability of the central processor. The sorting/selection problem is solved in k rounds of comparison if after k rounds the central processor knows the answer to the question. For example, if $a_1,\ldots,a_n$ are the elements of the ordered set (not necessarily ordered in this way), then the computation may start by sending a_{2i-1}, a_{2i}, to processor i ($i = 1,\ldots,n$) and after the first round of comparisons the central processor may decide that processor 1 gets a_7, a_{10}, processor 2 gets a_3, a_9, etc.

The motivation of this model is that we want only to count the number of comparisons and not the amount of computation necessary to decide which set of comparisons will be made in the next round.

We may generalize this model for the solution of linear programming problems. Assume that the coefficients of the constraints in the problem are elements of an abstract ordered field. Each constraint contains a constant number of these abstract elements as coefficients. At the beginning, the coefficients of each constraint are stored at a single processor. Different constraints are stored at different processors. We assume also that at the beginning of each step each processor holds a constant number of these abstract elements. During a single step, the processor may perform a constant number of arithmetic operations on them, compare the resulting elements, and report the results of the comparisons to the central processor. The central processor, using the reported results of the comparisons, redistributes the elements among the processors and decides which arithmetic operations and comparisons are done in the next step. (The central processor never gets the abstract elements themselves, only the results of the comparisons.) Again, this model measures only the number of arithmetic operations and comparisons necessary for the solution of the linear programming problem and not the amount of computation necessary to decide which arithmetic operations and comparisons must be performed. (We will give later a more realistic model which measures this as well.) We get a clearer picture if we separate the arithmetic operations and the comparisons, that is, we assume that in each round either only arithmetic operations or only comparisons are performed. This way we can measure separately both the number of arithmetic operations and comparisons required for a solution.

Our result in this computational model is that the d-dimensional linear programming problem can be solved in a way that the number of rounds where we perform only arithmetic operations is $O((\log\log n)^d)$, and the number of rounds where only comparisons are performed is $O((\log\log n)^{d+1})$.

To be able to measure the amount of computation needed to decide which arithmetic operations or comparisons to perform, we will use a CRCW-PRAM machine. We will not, however, describe every step of our computation in this model. We will assume that the selection steps are done in Valiant's comparison tree model. Our algorithm in the CRCW model described below can be performed in $O((\log\log n)^{d+c})$ steps where c is an absolute constant.

The CRCW-PRAM model that we use in this paper consists of processors that communicate with each other according to the following rules. If there are m processors, then the processors are numbered from 1 to m. The number assigned to a processor will be called its *address*. Each processor has a constant number of registers, which may contain positive integers not greater than m. In each step processors may (simultaneously) read the contents of the first register of any processor. We assume that when processor i reads the contents of the first register of processor j, then the number j is contained in the second register of processor i. Alternatively the processors may try simultaneously to write into the first register of any processor. (If more than one processor attempts to write in the same register the one with the smallest address succeeds.) These kinds of steps will be the read/write steps of the processors. Between these steps the processors can perform a constant number of arithmetic operations on the contents of their own registers.

To handle the real numbers given in the constraints of the linear programming problem, we assume that each processor also has a constant number of registers each containing a real number. We consider real numbers here as elements of an abstract ordered field, so the processors may only perform the arithmetic operation on them and may compare them, but the binary bits of the real numbers are not directly available for the processors. We assume that the same rules of reading and writing are valid for these type of registers as for the ones containing integers. Between two read/write steps each processor is allowed to perform a constant number of operations on the real numbers contained in its registers. We now describe how can we handle certain specific problems in this model.

Expander graphs. The expander graphs that we use were constructed by Gabber and Galil and are described in [9]. If $n = m^2$, then the vertices of the graph are ordered pairs of positive integers (i, j) where $0 \leq i, j < m$. The neighbors of the vertex (i, j) can be computed in a constant number of arithmetic operations modulo m starting from the numbers i, j. We will represent such a graph in the following way. Each vertex will be associated with a processor and the neighbors of that vertex will be listed in the registers of the processor. Therefore, if the number of vertices is not greater than the number of processors, this expander graph can be computed in a constant number of steps on our CRCW-PRAM machine.

Power of a graph. For certain steps of the algorithm we will need a family of graphs where the maximum degree is not bounded by a constant. In the following we may assume that graphs may have multiple edges. This makes their representation easier in our model and it is suitable also for our applications. The graphs will be represented in the following way. If the maximum degree is d then each vertex v will be associated with a set T_v of processors of size d. (We assume that the addresses of these processors form an interval of length d.) Each vertex v therefore has an address: the address of the first processor in T_v. The addresses of the neighbors are stored in the processors contained in T_v (the same address may occur several times). If G_1 and G_2 are graphs with the same set of vertices, then their product G_1G_2 is a graph where the edges between x and y are defined in the following way: for each vertex z and each pair of edges e, f so that e connects x and z, and f connects z and y there is a separate edge $E_{z,e,f}$ connecting x and y. It is easy to see that if two graphs G_1, G_2 are represented this way then in a constant number of steps we may compute a representation of their product, provided that the number of processors is large enough for the representation of the product. Consequently, if G is a graph then we may compute the representation of G^k in poly($\log k$) steps. We will always have $k \leq \log n$ (where n is the total number of processors), so we will have that G^k can be always constructed in poly($\log\log n$) steps.

Prime numbers. For certain steps of our algorithm we will need prime numbers. If we have n processors, all of these prime numbers are smaller that $\sqrt{n}$. We may actually compute all of the prime numbers up to $\sqrt{n}$ in a constant number of steps in our model in the following way. We divide the processors into $\sqrt{n}$ intervals, each of length $\sqrt{n}$. The ith interval has to decide whether the number i is a prime or not. Since there are at least i processors in the interval, they can decide this in a constant number of steps: the jth processor is checking whether j is a divisor of i. After this we may assume that if $k \leq \sqrt{n}$, then the kth processor knows whether k is a prime or not, and if necessary then other processors may read this information from its register. If processor j needs a prime from an interval I, contained in $[0, \sqrt{n}]$, then each processor whose address is a prime in this interval tries to write its address in the first register of processor i. If there is a prime in I then the smallest one will appear in the register of j.

All of the steps of the algorithm except the selection procedures can be implemented on a CRCW-PRAM. Apart from the specific problems mentioned above (expander graphs, powers of graphs, primes) there is only one step, namely, discarding constraints, whose implementation is not immediate. We solved the linear programming problem by discarding an increasing proportion of the remaining constraints. In a typical step of the iteration we assume that the remaining n constraints are stored in an array of n cells $R[1], \ldots, R[n]$, and we discard at least $n(1-1/s)$ $(1 \leq s \leq n)$ of them. In order to continue with the algorithm, we need the remaining n/s constraints to be stored in an array whose size is essentially not larger than n/s. More precisely, we need an algorithm for the following problem:

Problem 6.1. Given an array $R[1], \ldots, R[n]$ and a subset $H \subset \{1, 2, \ldots, n\}$ such that $|H| = n/s$, move the contents of each $R[i]$ $(i \in H)$ to some $R[j(i)]$ so that $j(i) \leq n/s^{1-\epsilon}$ and $j(i) \neq j(i')$ for all $i, i' \in H$ $(i \neq i')$.

Proposition 6.2. *For every fixed $\epsilon > 0$, there exists an algorithm for Problem 6.1 which runs in $O(\log\log n)$ time on an n-processor* CRCW-PRAM.

Proposition 6.2 implies that wherever we originally reduced the number of constraints from n to n/s, we will be able to reduce it on a CRCW-PRAM from n to $n/s^{1-\epsilon}$. This does not affect the upper bounds given in the previous sections.

In the following we will assume that each processor has a register which may contain a single element of a set A. We will say that this element is handled by the processor. We suppose that throughout the computation each element of A is handled by a single processor (which may be different from step to step) and each processor handles at most one element. We also assume that the processors are ordered in some arbitrary way. The *rank* of a processor is its position relative to the given ordering. According to this definition, if we say that we took the elements of A to the first k processors, then we mean that after executing the algorithm, each element of A will be handled by a processor whose rank is at most k, and distinct elements of A will be handled by distinct processors.

Definition 6.3. For any $\eta > 0$ and any positive integers c, n, s, let $\Phi_{\eta,c}(n, s)$ denote the following proposition: *There exists an algorithm that runs in c time units, so that if there are n processors and $|A| \leq n/s$, then after running the algorithm, the number of those elements of A which are not handled by one of the first $n/s^{1-\eta}$ processors is smaller than $n/s^{1+\eta}$.*

Proposition 6.4. *For every sufficiently small $\eta > 0$, there exists a positive integer c such that for all n and s, $\Phi_{\eta,c}(n, s)$ is true.*

The proof will be given later.

Proposition 6.5. *Proposition 6.4 implies Proposition 6.2.*

Proof: It suffices to prove Proposition 6.2 for every sufficiently small $\epsilon > 0$, since for smaller ϵ's the statement of the proposition is stronger. Given a sufficiently small $\epsilon > 0$, let $\eta = \epsilon/2$. Assuming Proposition 6.4 is true, $\Phi_{\eta,c}(n, s)$ is true. So, there exists an algorithm as explained in Definition 6.3. When we iterate this algorithm, then after each iteration the number of those elements of A which are not handled by one of the first $n/s^{1-\epsilon}$ processors decreases from n/t to $n/t^{1+\eta}$. After $O(\log \log n)$ iterations, every element of A will be at the place claimed in Proposition 6.2. Note at the first step $t = s$. ∎

Remark 6.6. It suffices to prove Proposition 6.4 for every $s > s_0$, where s_0 is an arbitrary constant. Indeed, it is possible to simulate ns_0 processors with n processors in a constant number of steps, so we may always assume that the number of processors is at least $s_0|A|$.

We use the following proposition in the proof of Proposition 6.4:

Proposition 6.7. *For every $\epsilon > 0$, there exist $\epsilon' > 0$ and $c > 0$ such that for all positive integers s, if we have s processors and $|A| \leq s^{1-\epsilon}$, then in c steps we can move all of the elements of A to the first $s^{1-\epsilon'}$ processors.*

Our goal is to show that Proposition 6.7 implies Proposition 6.4.

Let $\epsilon > 0$ be any small constant. (Later we will give an upper bound on ϵ.) We now assume that $|A| = n/s$, and the elements of A are stored at processors with rank not greater than n. For the sake of simplicity, we assume that s is an integer and n is divisible by s, but the proof remains valid in general with minor modifications. We partition the set of processors into n/s intervals of length s. Let A' be the set of all elements of A which occur in an interval where the number of elements from A is less then $s^{1-\epsilon}$. It follows from Proposition 6.7 that there exist $\epsilon' > 0$ and $c > 0$ such that for each interval of this type we can move the elements of the interval to the first $s^{1-\epsilon'}$ processors of this interval in time c. In this way, all of the elements of A' can be taken to processors with ranks smaller than $\frac{n}{s}s^{1-\epsilon'} = ns^{-\epsilon'}$.

Remark 6.8. We note that in order to perform the described step (with regard to moving the elements of A'), we do not have to count the number of elements in the intervals. We simply attempt to apply the algorithm of Proposition 6.7, and we succeed in the intervals that contain the elements of A'.

Let X denote the set of the remaining intervals and let A'' denote the set of those elements of A which are at processors belonging to an interval from X (after we have performed the steps based on Proposition 6.7.) The set A'' may have essentially the same size as A.

For the next step of the algorithm, we need the following proposition which is a consequence of the existence of explicitly constructible expander graphs:

Proposition 6.9. *There exist a positive integer r, a $\delta, \gamma \in (0,1)$ and an algorithm that constructs for any positive integers k and m, a symmetric nonnegative $m \times m$ matrix $\boldsymbol{B}$ of integer entries such that:*

(i) *The largest eigenvalue of $r^{-k}\boldsymbol{B}$ is 1, and the only eigenvector with this eigenvalue is $\frac{1}{\sqrt{m}}\boldsymbol{e}$ (where $\boldsymbol{e} = (1,\ldots,1)^T$).*

(ii) *All the other eigenvalues of $r^{-k}\boldsymbol{B}$ lie in the interval $[0, \gamma^k]$.*

(iii) *If k is sufficiently large relative to r and γ, and if $\boldsymbol{v} = (v_1, \ldots, v_m)^T$ is a $(0,1)$-vector such that $\boldsymbol{e} \cdot \boldsymbol{v} \le m/r^{2k}$, then $\|r^{-k}\boldsymbol{B}\boldsymbol{v}\|_2 \le r^{-\delta k}\|\boldsymbol{v}\|_2$.*

Proof: The results of Gabber and Galil [9] imply that there is an integer $d > 1$ and there is exists an explicit construction (for every m) of an $m \times m$ symmetric matrix $\boldsymbol{D}$ with nonnegative integer entries such that:

(i) d is an eigenvalue of $\boldsymbol{D}$, and the only eigenvector with eigenvalue d is $\frac{1}{\sqrt{m}}\boldsymbol{e}$.

(ii) All the eigenvalues of $\boldsymbol{D}$ lie between $-d$ and d.

Let $\boldsymbol{F} = \boldsymbol{D} + d\boldsymbol{I}$ where $\boldsymbol{I}$ is the identity matrix. We get the eigenvalues of $\boldsymbol{F}$ by adding d to the eigenvalues of $\boldsymbol{D}$. If $r = 2d$ and $\gamma/2d$ is the second largest eigenvalue of $\boldsymbol{F}$, then it is easy to see that the matrix $\boldsymbol{B} = \boldsymbol{F}^k$ is as stated in parts (i) and (ii) of the proposition. We will show that (iii) is a consequence of the above.

Let $\boldsymbol{v}$ be a $(0,1)$-vector such that $\boldsymbol{e} \cdot \boldsymbol{v} \leq m/r^k$. Denote by W the linear subspace of all the vectors orthogonal to $\boldsymbol{e}$. Represent $\boldsymbol{v}$ as

$$\boldsymbol{v} = \mu\boldsymbol{e} + \boldsymbol{w}$$

where $\boldsymbol{w} \in W$. Let $\boldsymbol{v}^2, \ldots, \boldsymbol{v}^m$ be orthogonal eigenvectors of $r^{-k}\boldsymbol{B}$, all orthogonal to $\boldsymbol{e}$, with eigenvalues $\lambda_2, \ldots, \lambda_m$, respectively ($0 \leq \lambda_i \leq \gamma^k$ for $i = 2, \ldots, m$), and represent $\boldsymbol{w} = \alpha_2\boldsymbol{v}^2 + \cdots + \alpha_m\boldsymbol{v}^m$. Obviously,

$$\|r^{-k}\boldsymbol{B}\boldsymbol{v}\| \leq \|r^{-k}\boldsymbol{B}\mu\boldsymbol{e}\| + \|r^{-k}\boldsymbol{B}\boldsymbol{w}\| \ .$$

Now,

$$\begin{aligned} \|r^{-k}\boldsymbol{B}\boldsymbol{w}\| &= \left\|\sum_{i=2}^{m} \alpha_i r^{-k}\boldsymbol{B}\boldsymbol{v}^i\right\| = \left\|\sum_{i=2}^{m} \alpha_i \lambda_i \boldsymbol{v}^i\right\| \\ &\leq \gamma^k \left\|\sum_{i=2}^{m} \alpha_i \boldsymbol{v}^i\right\| = \gamma^k\|\boldsymbol{w}\| \leq \gamma^k\|\boldsymbol{v}\| \ . \end{aligned}$$

Since

$$\|r^{-k}\boldsymbol{B}\mu\boldsymbol{e}\| = \|\mu\boldsymbol{e}\| = |\mu|\sqrt{m} \ ,$$

it follows that

$$\|r^{-k}\boldsymbol{B}\boldsymbol{v}\| \leq \mu\sqrt{m} + \gamma^k\|\boldsymbol{v}\| \ .$$

Now, $\boldsymbol{v}$ is a $(0,1)$ vector, so $\|\boldsymbol{v}\|^2 = \boldsymbol{e} \cdot \boldsymbol{v}$. On the other hand, $\boldsymbol{e} \cdot \boldsymbol{w} = 0$, so

$$\|\boldsymbol{v}\|^2 = \boldsymbol{e} \cdot (\mu\boldsymbol{e}) = \mu m$$

and we have

$$\mu\sqrt{m} \leq m^{-1/2}\|\boldsymbol{v}\|^2 \ .$$

Consequently,

$$\|r^{-k}\boldsymbol{B}\boldsymbol{v}\| \leq m^{-1/2}\|\boldsymbol{v}\|^2 + \gamma^k\|\boldsymbol{v}\| \leq (m^{-1/2}\|\boldsymbol{v}\| + \gamma^k)\|\boldsymbol{v}\| \ .$$

Since $\boldsymbol{v}$ is a $(0,1)$-vector and $\boldsymbol{e} \cdot \boldsymbol{v} \leq m/r^{2k}$, we have $\|\boldsymbol{v}\| \leq (mr^{-2k})^{1/2}$. This implies that

$$\|r^{-k}\boldsymbol{B}\boldsymbol{v}\| \leq (m^{-1/2}(mr^{-2k})^{1/2} + \gamma^k)\|\boldsymbol{v}\| = (r^{-k} + \gamma^k)\|\boldsymbol{v}\| \ \leq r^{-\delta k}\|\boldsymbol{v}\|$$

if k is sufficiently large , where $\delta > 0$ depends only on γ and r. ∎

In the intervals of X there are at least $s^{1-\epsilon}$ elements from A''. We apply Proposition 6.9 with $m = n/s = |X|$. We pick k so that $s^{1/8} < r^k < s^{1/4}$; this is possible since $s > s_0$. We take a graph G on the vertex set X whose matrix is $\boldsymbol{B}$. This graph may contain both loops and multiple edges. We try to move the elements of A'' along the edges. More precisely, an element of A'' which is at a processor in an interval I will be moved to a processor of an interval which is connected to I by an edge of the graph. Distinct elements from the same interval I may move to distinct intervals.

Let h be the number of intervals which contain more than $s^{1-\epsilon}$ elements from A''. We will later prove the following:

Proposition 6.10. *It is possible to move the elements of A'' in a constant number of steps so that the new arrangement has the following property: if K is the set of intervals which contain more than $s^{1-\epsilon}$ elements from A'', then $|K| < s^{-\delta/5}h$, where δ is the constant defined in Proposition 6.9.*

We first show that, assuming Proposition 6.7 is true, Proposition 6.10 implies Proposition 6.4.

Proof of Proposition 6.4: Let A_0 be the set of those elements from A'' which are in an interval belonging to K. Let $\lambda = \delta/5$. By proposition 6.10 there exists $\epsilon > 0$ such that

$$|A_0| < |A|s^{-(1-\epsilon)}s^{-\lambda}s = |A|s^{\epsilon-\lambda} \leq |A|s^{-\lambda/2} .$$

So, if we pick $0 < \eta < \lambda/2$, then all the elements of A_0 can be included among the exceptional $n/s^{1+\eta}$ elements of Proposition 6.4.

All the remaining elements, *i.e.*, the elements of $A'' \setminus A_0$, are now in intervals where the number of elements is smaller than $s^{1-\epsilon}$. Thus, applying again the algorithm based on Proposition 6.7, we put every element of A at the required place. (We assume that $\eta < \epsilon/2$.) To complete the proof, we have to show that the elements of A'' can be moved in the manner described.

Let E be the set of edges of the graph G on the vertex set X, associated with the matrix $\boldsymbol{B}$ in proposition 6.10. Property (i) of the matrix $\boldsymbol{B}$ implies that the degrees of all the vertices are equal to $\ell = r^k$. We partition E into ℓ 1-factors and each interval I into s/ℓ classes of equal sizes. We associate with each edge $e \in E$, connecting the intervals I, J, with a pair of classes $I_e \in I$ and $J_e \in J$. Using the partition of E into 1-factors, we may define the pairs (I_e, J_e) so that each class occurs in exactly one pair associated with some edge e. Let ι_e be a one-to-one correspondence between the sets I_e and J_e. In each step we swap the contents of the processors x and $\iota_e(x)$ for all $e \in E$ and $x \in I_e$. This can be done in a single step since each processor takes part in a single swap. The essential change is as follows. If in a pair $(x, \iota_e(x))$ exactly one processor contained an element of A, then this remains true but the element from A will be in the other processor. (If either both of them or none of them contained an element of A, then this situation prevails.)

Originally, we had only intervals where the processors containing elements from A'' had either density at least $s^{-\epsilon}$ or density 0. Suppose that after performing a step described above, the set A'' will have density u_I in the interval I $(0 < u_I < 1)$. Let $\boldsymbol{w} = (w_I)_{I \in X}$ be the vector consisting of the original densities, and let $\boldsymbol{u} = (u_I)_{I \in X}$ be the vector of densities after one step has been performed. Our definitions imply that $\boldsymbol{u} = r^{-k}\boldsymbol{B}\boldsymbol{w}$. Let $\boldsymbol{v} = (v_I)_{I \in X}$ be the "characteristic" vector of $\boldsymbol{w}$, *i.e.*,

$$v_I = \begin{cases} 1 & \text{if } w_I \neq 0 \\ 0 & \text{if } w_I = 0 \ . \end{cases}$$

By the nonnegativity of $\boldsymbol{B}$, $\boldsymbol{u}$, $\boldsymbol{v}$, and $\boldsymbol{w}$, we have $\|\boldsymbol{u}\| \leq \|r^{-k}\boldsymbol{B}\boldsymbol{v}\|$. To give an upper bound on $\|\boldsymbol{u}\|$, we want to use property (iii). The condition "k is sufficiently large" holds because of the assumptions $r^k > s^{1/8}$ and $s > s_0$. Also,

$$\sum v_I = |K| \leq |A|/s^{1-\epsilon} = (n/s)/s^{1-\epsilon} \leq |X|/s^{1-\epsilon} \ .$$

Since $m = |X|$ and $r^{2k} < s^{1/2} < s^{1-\epsilon}$, the requirements of (iii) in Proposition 6.9 are met. According to the conclusion there, we have $\|\boldsymbol{u}\| \leq r^{-\delta k}\|\boldsymbol{v}\| \leq r^{-\delta k}h^{1/2}$, where h is the number of nonzero components of $\boldsymbol{v}$. On the other hand $\|\boldsymbol{u}\| \geq s^{-\epsilon}|K|^{1/2}$. Therefore, $s^{-\epsilon}|K|^{1/2} \leq r^{-\delta k}h^{1/2}$. Since $r \geq s^{1/8}$, we conclude that $|K| \leq s^{2\epsilon-\delta/4}h \leq s^{-\delta/5}h$ (here we assumed that $\epsilon < \delta/20$). ∎

We will need the following in the proof of Proposition 6.7.

Proposition 6.11. *There is a positive c such that for every integer $s > 0$, if we have s processors, $s^{1/8} > |A|$, and the elements of A are at the first $s^{1/2}$ processors, then all the elements of A can be moved to the first $s^{1/4}$ processors in c steps.*

We will use the following well-known concepts in the proof. If K is a field, then the *affine plane* over K, $K \times K$, consists of all the ordered pairs of elements of K. A subset $L \subseteq K \times K$ is a line if there exist $a, b, c \in K$ so that $L = \{\langle x, y\rangle | ax + by + c = 0\}$. We say that two lines have the same direction if they are parallel, that is, they do not intersect. A direction is a maximal set of parallel lines. *Proof:* Suppose $p = s^{1/4}$ is a prime. (If $p = s^{1/4}$ is not a prime then let p be an arbitrary prime between $\frac{1}{2}s^{1/4}$ and $s^{1/4}$.) We associate with each of the first $s^{1/2}$ processors a point in the affine plane with p^2 elements. There are $p = s^{1/4}$ directions (maximal sets of parallel lines) on the plane but less than $(s^{1/8})^2 = s^{1/4}$ pairs formed from the elements of A. Therefore, there is a direction so that each line of this direction contains at most one element from A. Since the number of processors is s, we can actually find such a direction in a constant number of steps. Indeed, we associate with each line e a processor P_e, and with each pair of points $\langle p_1, p_2\rangle$ ($p_1 \neq p_2$), a processor Q_{p_1,p_2}. If both points p_1, p_2 contain an element of A, then the processor $Q(p_1, p_2)$ attempts to write in the register of processor P_e, where e is the line determined by the points p_1 and p_2. After this step, each processor P_e knows whether the line e contains more than one element of A and in the next step in will try in a similar way to transmit this information to processors associated with directions. Using the direction where each line contains at most one point from A, we may easily move A to the first $s^{1/4}$ processors. ∎

Proof of Proposition 6.7: If ϵ, ϵ' and c are positive, then denote by $\Psi(\epsilon, \epsilon', c)$ the following statement: *for all positive integers s, if the elements of A ($|A| \leq s^{1-\epsilon}$) are given on s processors, then in c steps we can move all the elements to the first $s^{1-\epsilon'}$ processors.*

Proposition 6.7 is the following assertion:

$$(\forall \epsilon > 0)(\exists \epsilon' > 0)(\exists c > 0)\ \Psi(\epsilon, \epsilon', c)\ .$$

Let $\Theta(\epsilon)$ be the statement:

$$(\exists \epsilon' > 0)(\exists c > 0)\ \Psi(\epsilon, \epsilon', c)\ .$$

We have to prove that for all $\epsilon \in (0, 1)$, $\Theta(\epsilon)$.

We will present a strictly increasing continuous function $f : (0, 1) \to (0, 1)$ so that for each $\epsilon \in (0, 1)$ we have: $\Theta(f(\epsilon)) \Rightarrow \Theta(\epsilon)$. Moreover, we will give an $\epsilon_0 \in (0, 1)$ with $\Theta(\epsilon_0)$. The existence of these objects implies Proposition 6.7 since the properties of the function f guarantee that for every $\epsilon > 0$ there is a positive integer i such that $f^{(i)}(\epsilon) > \epsilon_0$, where $f^{(i)}$ is the ith iterate of f. Therefore (using the fact that if $\alpha < \beta$, then $\Theta(\alpha) \Rightarrow \Theta(\beta)$), we have

$$\Theta(\epsilon_0) \Rightarrow \Theta(f^{(i)}(\epsilon)) \Rightarrow \Theta(f^{(i-1)}(\epsilon)) \Rightarrow \cdots \Rightarrow \Theta(\epsilon)\ .$$

Now we prove $\Theta(\epsilon_0)$ with $\epsilon_0 = 7/8$. More precisely, we prove $\Psi(7/8, 1/4, c)$ for some c. Assume that $|A| \leq s^{1-\epsilon_0} = s^{1/8}$. Let Z be a set of $s^{1/2}$ elements. We associate each processor with an element of $Z \times Z$. Let

$$Y = \{y \in Z \mid (\exists z \in Z)(\text{processor } \langle y, z\rangle \text{ stores an element of } A)\}\ .$$

We can actually find the elements of Y in a constant number of steps. Note that Y is represented as the set $Y \times \{z_0\}$ for an arbitrary $z_0 \in Z$. Also, $|Y| \leq |A| \leq s^{1/8}$. Let $\bar{A}$ be a new set (disjoint from A) whose elements have to be handled by the processors. Suppose that the elements of $\bar{A}$ are at the processors belonging to Y. We may assume that $Z \times \{z_0\}$ are the first $s^{1/2}$ processors, so the elements of $\bar{A}$ are at the first $s^{1/2}$ processors. Applying Proposition 6.11, we may move the elements of $\bar{A}$ to the first

$s^{1/4}$ processors. We may also assume that at the end, each processor containing an $a \in \bar{A}$ also contains the address of the processor where a was initially. This implies that coming back to the original set A, we may move simultaneously all the elements of A inside the sets $Z \times \{z\}$ (for each $z \in Z$), so that each element of A will move to a processor $\langle v, z\rangle$, where v is among the first $s^{1/4}$ elements of Z. Consequently, A is on the first $s^{3/4}$ processors, which concludes the proof of $\Theta(\epsilon_0)$. Thus, we have reduced the proof of Proposition 6.7 to the question of existence of a function f with properties given in the following proposition. ∎

Proposition 6.12. *There is a strictly increasing continuous function $f : (0, 1) \to (0, 1)$ such that for every $\epsilon \in (0, 1)$, $\Theta(f(\epsilon)) \Rightarrow \Theta(\epsilon)$.*

For the proof of this proposition we will need the following:

Proposition 6.13. *If $\alpha > 0$ is sufficiently small, then for every $\xi > 0$, there is a positive c such that for all s, if $|A| = s^\alpha$ and the elements of A are on s processors, then they can be moved in c steps to the first $s^{\alpha+\xi}$ processors.*

Remark 6.14. We will use the following consequence of the proof of $\Theta(\epsilon_0)$: if $\alpha > 0$ is sufficiently small and $|A| \le s^\alpha$, then we can move $|A|$ to the first $s^{3/4}$ processors in a constant number of steps. Iterating this step, we get the following: *for all $\beta > 0$, if $\alpha > 0$ is sufficiently small, $|A| \le s^\alpha$, and the set A is given on s processors, then we can move A to the first s^β processors in a constant number of steps.*

Assume now that a sufficiently small $\beta > 0$ is fixed and $\alpha > 0$ is sufficiently small with respect to β. According to the previous remark, we may assume that A is on the first s^β processors. Let $m = s^\beta$. We now have the following situation: the elements of A are given on m processors but we have $m^{1/\beta}$ extra processors that we can use for the computation. Therefore, we can simulate an unlimited fan-in constant depth Boolean circuit of size $m^{\beta/2}$ with m inputs. We associate the input nodes with the m processors. The value of the input will be 1 if there is an element of A at the processor, and 0 otherwise. Using the following theorem (see [1]), we are able to count approximately the number of elements of A in a constant number of steps:

Theorem 6.15. *There exist positive c and d such that for all positive integers n and $a \le n$, there is an (explicitly constructed) unlimited fan-in Boolean circuit C with n inputs, with depth d and of size at most n^c, so that for each input sequence $\boldsymbol{x}$, if $|\boldsymbol{x}|$ denotes the number of 1's in the sequence, then we have the following:*

(i) *If $|\boldsymbol{x}| \le (1 - (\log n)^{-1})a$, then $C(\boldsymbol{x}) = 0$, and*

(ii) *If* $|\boldsymbol{x}| \geq (1+(\log n)^{-1})a$, *then* $C(\boldsymbol{x})=1$.

We will also use the following easy proposition from [1] about mod p polynomials.

Definition 6.16. Let p be a prime number and let K_p be a field with p elements. Assume that f is a polynomial of degree k with coefficients in K_p. We define a map

$$h_f^p : K_p \times K_p \to K_p$$

as follows. If $\langle u, v\rangle \in K_p \times K_p$ then $h_f^p(\langle u, v\rangle) = u - f(v)$.

Proposition 6.17. *If* $X \subseteq K_p \times K_p$ *and* $|X| \leq p^{1-2/(k+1)}$, *then there exists a polynomial* f *of degree* k *with coefficients in* K_p, *such that for all* $y \in K_p$,

$$|\{x \in K_p \times K_p \mid h_f^p(x) = y\}| \leq k\ .$$

We use this proposition with $k=4$, and we assume that $K_p \times K_p$ is the set of processors, and X is the subset of processors where the elements of A are sitting. Proposition 6.17 for $k=4$ implies that if $|A| < p^{3/5}$ and the elements of A are located in the first p^2 processors, then using p^{10} processors, we can move the elements of A to the first $4p$ processors in a constant number of steps. Since we have enough processors, we may check all of the polynomials of degree 4 simultaneously and find a polynomial f satisfying the conclusions of the proposition. Using this polynomial, we can move the points in A to the first $4p$ processors. Iterating this step and using Remark 6.14, we get the following:

Proposition 6.18. *If* $\alpha > 0$ *is sufficiently small and* $|A| \leq s^\alpha$, *then using* s *processors, the elements of* A *can be moved to the first* $s^{2\alpha}$ *processors.*

Proof of Proposition 6.13: Assume now that $|A| \leq s^\alpha$, where $\alpha > 0$ is sufficiently small. According to Proposition 6.18, we may assume that A is already in the first $|A|^2$ processors and we may continue in the following way: assume that p is a prime with $|A| < p < 2|A|$. Since we are able to count approximately, we can find such a prime in a constant number of steps. (Since we can count approximately we can find in a constant number of steps an interval which is contained in $(|A|, 2|A|)$ and is of length at least $|A|/2$. By the Prime Number Theorem, if $|A|$ is sufficiently large, such an interval always contains a prime.) We put the first p^2 processors on the affine plane with p^2 elements. There are only p^2 ordered pairs formed from the elements of A, and there are p directions (maximal set of parallel lines) on the plane. Therefore, there must be a direction so that the number of ordered pairs from A contained in the lines of this direction is at most p. Since $\alpha > 0$ is sufficiently small, and A is on the first $s^{2\alpha}$ processors, we may actually find a direction (using approximate counting) where the number of ordered pairs is less than, say, $2p$. Let $a_1, \ldots, a_p$ be the numbers of elements of A on the lines with this direction. We have $\sum_{i=1}^{p} a_i^2 \leq 2p$.

Let $\mu > 0$ be sufficiently small. First, we consider the lines with $a_i \leq p^{\mu}$. According to Proposition 6.18, within each line of this type all of the elements of A can be moved to the first $p^{2\mu}$ processors. Therefore, every element of A contained in a line of this type can be moved to an array of size $pp^{2\mu} = 4s^{\alpha+2\mu} \leq s^{\alpha+3\mu}$.

We claim that the number of elements of A on lines with $a_i > p^{\mu}$ is at most $p^{1-\mu}$. Indeed, the minimum of $\sum' a_i^2$ (where $\sum'$ stands for $\sum_{a_i > p^{\mu}}$) subject to $\sum' a_i =$ constant is attained when all the a_i's are equal, and in this case we get the claimed result.

Thus, in one step we decreased the number of elements of A not in our array by a factor of p^{μ}. Continuing this process, in a constant number of steps we will have all but $s^{\alpha/2}$ elements of A in an array of the required size. According to Proposition 6.18, the remaining $s^{\alpha/2}$ elements can be moved to an array of size s^{α}, which completes the proof of Proposition 6.13. ∎

Proof of Proposition 6.12: We first define f. Let $\alpha \in (0,1)$ so that Proposition 6.13 holds for α. For the remainder of the proof, we consider α as fixed. Let $f(\epsilon) = \epsilon(1 + \alpha(1-\epsilon))$. Clearly, $f(\epsilon) \in (0,1)$ for all $\epsilon \in (0,1)$. Moreover, f is continuous and strictly increasing in this interval. Apart from these, we will need only the following property of f: for all $\epsilon \in (0,1)$,

$$1 - \epsilon - (1-\alpha)(1-f(\epsilon)) \leq \alpha - \alpha^2\epsilon \ .$$

Indeed,

$$\begin{aligned} 1 - \epsilon - (1-\alpha)(1-f(\epsilon)) &= \alpha - \epsilon + \epsilon(1+\alpha(1-\epsilon))(1-\alpha) \\ &\leq \alpha - \epsilon + \epsilon(1+\alpha)(1-\alpha) = \alpha - \alpha^2\epsilon \ . \end{aligned}$$

Suppose $\epsilon \in (0,1)$ is fixed and $\Theta(f(\epsilon))$. We wish to show that $\Theta(\epsilon)$ holds. We divide the set of s processors into intervals of size $s^{1-\alpha}$. Let R be the set of all intervals of this type and for all $\kappa > 0$, let $S_\kappa \subseteq R$ be the set of those intervals where the number of elements from A is at most $(s^{1-\alpha})^{1-\kappa}$. $\Theta(f(\epsilon))$ implies that for each $I \in S_{f(\epsilon)}$, the elements of A can be moved to the first $(s^{1-\alpha})^{1-\epsilon_1}$ processors in c_1 steps, where ϵ_1 and c_1 depend only on $f(\epsilon)$ (and so only on ϵ.) Therefore, all of the elements of A contained in intervals of $S_{f(\epsilon)}$ can be moved to the first $s^{\alpha}(s^{1-\alpha})^{1-\epsilon_1} = s^{1-(\epsilon_1 - \alpha\epsilon_1)}$ processors in a constant number of steps. So, we have to consider only those elements of A that are contained in processors outside $S_{f(\epsilon)}$. (We first apply the algorithm described here to all the intervals, and later we work only with those elements of A that are in intervals where the algorithm did not work.)

Since each interval of $R \setminus S_{f_\epsilon}$ contains at least $(s^{1-\alpha})^{1-f(\epsilon)}$ elements of A and $|A| \leq s^{1-\epsilon}$, we have that $|R \setminus S_{f(\epsilon)}| \leq s^{1-\epsilon}/s^{(1-\alpha)(1-f(\epsilon))}$. This and the inequality proved after the definition of f imply that $|R \setminus S_{f(\epsilon)}| \leq s^{\alpha-\alpha^2\epsilon} = (s^\alpha)^{1-\alpha\epsilon}$. According to Proposition 6.13, using all of the s processors, we can construct a one-to-one mapping τ of $R \setminus S_{f(\epsilon)}$ into the set of the first $(s^\alpha)^{1-(\alpha\epsilon/2)}$ intervals of R. Moving all the elements of A from an interval I to the processors of the interval $\tau(I)$, we are able to move all of the elements of A to processors in the first $(s^\alpha)^{1-(\alpha\epsilon/2)}$ intervals. Thus, we have moved the elements of A to the first $s^{1-\alpha}(s^\alpha)^{1-(\alpha\epsilon/2)} = s^{1-(\alpha\epsilon/2)}$ processors, and so $\Theta(\epsilon)$ holds with $\epsilon' = \alpha\epsilon/2$. This completes the proof of Proposition 6.12 which was the last step in the proof of Proposition 6.2 ∎

.

References

[1] M. Ajtai, "Approximate Counting with Uniform Constant Depth Circuits," in: Advances in Computational Complexity Theory, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 13, Jin-Yi Cai, Ed., Amer. Math. Soc., 1993, pp. 1–20.

[2] M. Ajtai, J. Komlós, W. L. Steiger, and E. Szemerédi, "Optimal parallel selection has complexity $O(\log \log N)$," *J. Comp. Sys. Sci.* **38** (1989) 125–133.

[3] N. Alon and N. Megiddo, "Parallel linear programming in fixed dimension almost surely in constant time," *J. ACM* **41** (1994) 422–434.

[4] K. L. Clarkson, "Linear programming in $O(n \times 3^{d^2})$ time," *Information Processing Letters* **22** (1986) 21–27.

[5] X. Deng, "An optimal parallel algorithm for linear programming in the plane," *Information Processing Letters* **35** (1990) 213–217.

[6] D. Dobkin, R. J. Lipton, and S. Reiss, "Linear programming is log space hard for P," *Information Processing Letters* **8** (1979) 96–97.

[7] M. E. Dyer, "Linear time algorithms for two- and three-variable linear programs," *SIAM J. Comput.* **13** (1984) 31–45.

[8] M. E. Dyer, "On a multidimensional search technique and its application to the Euclidean one-center problem," *SIAM J. Comput.* **15** (1986) 725–738.

[9] O. Gabber and Z. Galil, "Explicit Construction of Linear-Sized Superconcentrators" *J. Comput. Sys. Sci.* **22** (1981) 407–420.

[10] N. Megiddo, "Linear-time algorithms for linear programming in R^3 and related problems," *SIAM J. Comput.* **12** (1983) 759–776.

[11] N. Megiddo, “Linear programming in linear time when the dimension is fixed,” *J. ACM* **31** (1984) 114–127.

[12] N. Megiddo, “Dynamic location problems,” *Annals of Operations Research* **6** (1986) 313–319.

[13] R. M. Tanner, “Explicit concentrators from generalized N-gons,” *SIAM J. Alg. Disc. Meth.* **5** (1984) 287–293.