Sorting-Based Selection Algorithms for Hypercubic Networks

P. Berthomé[1] *A. Ferreira*[1,2] *B. M. Maggs*[3]
S. Perennes[1] *C. G. Plaxton*[4]

Abstract

This paper presents several deterministic algorithms for selecting the kth largest record from a set of n records on any n-node hypercubic network. All of the algorithms are based on the selection algorithm of Cole and Yap, as well as on various sorting algorithms for hypercubic networks. Our fastest algorithm runs in $O(\lg n \lg^* n)$ time, very nearly matching the trivial $\Omega(\lg n)$ lower bound.

Keywords: Parallel algorithms, hypercube, selection.

Résumé

Dans cet article, nous présentons plusieurs algorithmes déterministes permettant de trouver le k-ième élément dans un ensemble ayant n enregistrements sur un réseau hypercubique quelconque. Tous ces algorithmes sont basés sur l'algorithme de sélection de Cole et Yap, ainsi que sur divers algorithmes de tri sur réseaux hypercubiques. Notre meilleur algorithme fonctionne en temps $O(\lg n \lg^* n)$, ce qui nous rapproche de la borne inférieure immédiate $\Omega(\lg n)$.

Mots-clés : Algorithmes parallèles, hypercube, sélection.

---

[1]Laboratoire de l'Informatique du Parallélisme, Institut IMAG, Ecole Normale Supérieure de Lyon, 46, Allée d'Italie, 69364 Lyon Cedex 07, France. This work has been supported by the French CNRS Coordinated Research Program on Parallelism C[3] and the French PRC/GDR MathInfo.

[2]On leave from Universidade de São Paulo, Brazil.

[3]NEC Research Institute, 4 Independence Way, Princeton, NJ 08540.

[4]Department of Computer Science, University of Texas at Austin, Austin, TX 78712. Supported by NSF Research Initiation Award CCR–9111591, Texas Advanced Research Program (TARP) Award #003658480, and the NEC Research Institute.

1 Introduction

Successive sampling techniques have previously been used to obtain efficient selection algorithms on certain idealized models of parallel computation. In this paper, we apply this methodology to obtain asymptotically fast selection algorithms for more practical models of parallel computation. In particular, the time bounds that we derive are applicable to any parallel machine in the class of hypercubic networks, which includes the hypercube, butterfly, cube-connected cycles, and shuffle-exchange.

1.1 Hypercubic networks

The selection algorithms described in this paper can be implemented on any hypercubic network. Let $n = 2^d$, where d is a nonnegative integer. In an n-node *hypercube*, each node has a distinct d-bit label. A node labeled $b_0 \cdots b_{d-1}$ has edges to those nodes whose labels differ from $b_0 \cdots b_{d-1}$ in exactly one bit position. An edge connecting two nodes whose labels differ in bit i is called a *dimension-i* edge. Each node has d neighbors, one for each dimension. A *subcube* of the hypercube is formed by fixing the bit values of the labels in some subset of the d dimensions of the hypercube, and allowing the bit values in the other dimensions to vary. In particular, for each subset $j_0, \ldots, j_{k-1}$ of the set of dimensions $\{0, \ldots, d-1\}$, and each set of bit values $v_0, \ldots, v_{k-1}$, there is a dimension-k subcube of the hypercube consisting of the $n/2^k$ nodes whose labels have value v_i in dimension j_i, $0 \leq i < k$, and the edges connecting those nodes.

The nodes in a hypercube represent processors, and the edges represent wires. Each processor has some local memory organized in $O(d)$-bit words. At each time step, a processor can send a word of data to one of its neighbors, receive a word of data from one of its neighbors, and perform a local operation on word-sized operands. In sorting and selection problems, the input consists of a number of $O(1)$-word *records*. Each record has an associated *key* that determines its rank in the entire set of records. We will assume throughout that all keys are unique. This may be done without loss of generality, since ties can always be broken in a consistent manner by appending the initial address (processor and memory location) of each record to its key.

All of the algorithms described in this paper use the edges of the hypercube in a very restricted way. At each time step, only the edges associated with a single dimension are used, and consecutive dimensions are used on consecutive steps. Such algorithms are called *normal* [6, Section 3.1.4]. The bounded-degree variants of the hypercube, including the butterfly, cube-connected cycles, and shuffle-exchange graph, can all simulate any normal hypercube algorithm with constant slowdown. For simplicity, we will describe all of the algorithms in terms of the hypercube.

1.2 Previous work

In [9], Valiant proved an $\Omega(\lg\lg n)$ lower bound on the time to find the largest record in a set of n records using n processors in the parallel comparison model. The lower bound implies a lower bound on the time to select the kth largest record as well. Valiant also showed how to find the largest record in $O(\lg\lg n)$ time. Cole and Yap [4] then described an $O((\lg\lg n)^2)$ selection algorithm for this model. The running time was later improved to $O(\lg\lg n)$ by Ajtai, Komlós, Steiger, and Szemerédi [1]. The comparisons performed by the latter algorithm are specified by an expander graph, however, making it unlikely that this algorithm can be efficiently implemented on a hypercubic network.

A different set of upper and lower bounds hold in the PRAM models. Beame and Hastad [2] proved an $\Omega(\lg n/\lg\lg n)$ lower bound on the time for selection in the CRCW comparison PRAM using a polynomial number of processors. Vishkin [10] discovered an $O(\lg n \lg\lg n)$ time PRAM algorithm that uses $O(n/\lg n \lg\lg n)$ processors. The algorithm is work-efficient (i.e., exhibits optimal speedup) because the processor time product is equal to the time, $O(n)$, of the fastest sequential algorithm for this problem. Cole [3] later found an $O(\lg n \lg^* n)$ time work-efficient PRAM algorithm.

For any p-processor hypercubic network, Plaxton [8] showed that selection from a set of n records requires $\Omega((n/p)\lg\lg p + \lg p)$ time in the worst case. The bound implies that a work-efficient algorithm is not possible.

This paper presents several algorithms for selecting the kth largest record from a set of n unordered records on an n-node hypercubic network. In Section 2.1 we describe an approximate selection algorithm. This algorithm is used as a subroutine in selection algorithms described in later sections. In Section 2.2 we describe an $O(\lg n \lg\lg n)$ time algorithm. The running time is improved to $O(\lg n \lg^{(3)} n)$ and then to $O(\lg n \lg^{(4)} n)$ in Sections 2.3 and 2.4, respectively. An $O(\lg n \lg^* n)$ algorithm is presented is Section 2.5. This improvement in the running time is obtained at the expense of using a non-uniform variant of the Sharesort algorithm [5] that requires a certain amount of preprocessing. Finally, in Section 2.6 we show how to avoid the non-uniformity introduced in Section 2.5. Thus, our best asymptotic result is a uniform selection algorithm with a running time of $O(\lg n \lg^* n)$.

2 Selection by successive approximation

2.1 Approximate selection

In this section, we develop an efficient subroutine for approximate selection based on the parallel comparison model algorithm of Cole and Yap [4]. There are two major differences. First, we use Nassimi and Sahni's sparse enumeration sort [7] instead of a constant time sort (as is possible in the parallel comparison model), and second we obtain a total running time that is proportional to the running time of the largest call to sparse enumeration sort, whereas in the Cole and Yap algorithm, the running time is proportional to the number of sorts ($O(\lg\lg n)$), each of which costs constant time.

As in the Cole-Yap algorithm, the approximate selection algorithm proceeds by successively sampling the given set of records. We define "sample 0" as the entire set of records. At the ith stage of the approximate selection algorithm, $i \geq 0$, a "sub-sample" is extracted from sample i. This sub-sample represents sample $i+1$, and will be a proper subset of sample i. Hence the sequence of sample sizes is monotonically decreasing. The sampling process terminates at a value of i for which the ith sample is sufficiently small that it can be sorted in logarithmic time (using sparse enumeration sort). From this final sample, we will extract lower and upper approximations to the desired record. A *lower approximation* to the record of rank k is a record with rank less than or equal to k. An *upper approximation* to the record of rank k is a record with rank greater than or equal to k. Our goal will be to obtain "good" upper and lower approximations in the sense that the ranks of our approximations will be close to k.

The following approach is used to extract sample $i+1$ from sample i. First, the records of sample i are partitioned into a number of equal-sized groups, and each group is assigned an equal fraction of the processors. Second, each group of records is sorted using sparse enumeration sort. The number of groups is determined in such a way that the running time of sparse enumeration sort is logarithmic in the group size. This is the case, for example, if sparse enumeration sort is used to sort m^2 records in a subcube with m^3 processors. Letting κ denote the group size, the third step is to extract approximately $\sqrt{\kappa}$ uniformly-spaced records (i.e., every $\sqrt{\kappa}$th record) from each group. The union of these extracted sets of size $\sqrt{\kappa}$ forms sample $i+1$. Note that the ratio of the size of sample i to that of sample $i+1$ is $\sqrt{\kappa}$.

Before proceeding, we introduce a couple of definitions.

Definition 2.1 The rank of a record α in a set S, $\mathrm{rank}(\alpha, S)$, is equal to the number of records in S that are strictly smaller than α. (Note that the record α may or may not belong to the set S.)

Definition 2.2 An r-sample of a set of records S is the subset $R \subseteq S$ consisting of those records whose ranks in S are multiples of r, i.e., $R = \{\alpha \in S \mid \mathrm{rank}(\alpha, S) = ir, 0 \le i < |S|/r\}$.

The input to our algorithm is a set S_0 of 2^δ elements concentrated in a subcube A of size $2^{\delta+x}$. (A set of records R is said to be *concentrated* in a subcube A if each of the $|R|$ lowest-numbered processors of A contains a unique element of R.) The factor by which the size of the subcube exceeds the size of S_0, 2^x, is called the *excess processor ratio*. In the first iteration, the records in S_0 are partitioned into $2^{\delta-2x}$ groups of size 2^{2x} and 2^{3x} processors are assigned to each group. Each group is then sorted in $O(x)$ time using sparse enumeration sort, and a 2^x-sample is taken from each group. The samples from all the groups are combined to form a new set S_1 containing $2^{\delta-x}$ elements. In general, after $i-1$ iterations, a set S_{i-1} of $2^{\delta-x(2^{i-1}-1)}$ records remain. In the ith iteration, set S_i is formed by partitioning the records of S_{i-1} into $g_{i-1} \stackrel{\text{def}}{=} 2^{\delta-x(3\cdot 2^{i-1}-1)}$ groups of size 2^{x2^i} and then extracting a $2^{x2^{i-1}}$-sample from each group. Since the ratio of the number of processors in A to $|S_{i-1}|$ is $2^{x2^{i-1}}$, we can assign $2^{3x2^{i-1}}$ processors to each group of size 2^{x2^i}, and each group can be sorted in $O(x2^i)$ time using sparse enumeration sort.

Lemma 2.1 The time to execute i iterations of the approximate selection algorithm is $O(x2^i)$.

Proof: The time is $\sum_{1\le j\le i} O(x2^j) = O(x2^i)$. □

Lemma 2.2 Let δ, δ', and δ'' denote integers satisfying $0 \le \delta' \le \delta$ and $0 \le \delta'' \le \delta'$. Let X denote a set of 2^δ records, and assume that X is partitioned into $2^{\delta-\delta'}$ sets X_k, $0 \le k < 2^{\delta-\delta'}$, of size $2^{\delta'}$. Let X' denote the union of the $2^{\delta''}$-samples of each of the X_k's. If record α has rank j in set X', then the rank of α in set X lies in the interval

$$(j2^{\delta''} - 2^{\delta-\delta'+\delta''}, j2^{\delta''}].$$

Proof: Let r_k denote the rank of α in the $2^{\delta''}$-sample extracted from set X_k, $0 \leq k < 2^{\delta-\delta'}$. Then the rank of α in set X_k lies in the interval $((r_k - 1)2^{\delta''}, r_k 2^{\delta''}]$, and so the rank of α in X belongs to

$$(\sum_{0 \leq k < 2^{\delta-\delta'}} (r_k - 1)2^{\delta''}, \sum_{0 \leq k < 2^{\delta-\delta'}} r_k 2^{\delta''}] = (j2^{\delta''} - 2^{\delta-\delta'+\delta''}, j2^{\delta''}],$$

since $j = \sum_{0 \leq k < 2^{\delta-\delta'}} r_k$. □

Corollary 2.2.1 Let record α have rank j in set S_i, for some $i \geq 1$. Then the rank of α in set S_{i-1} lies in the interval

$$(j2^{x2^{i-1}} - 2^{\delta - x(2^i-1)}, j2^{x2^{i-1}}].$$

Proof: A straightforward application of Lemma 2.2, with the variables δ, δ', and δ'' of the lemma replaced by the expressions $\delta - x(2^{i-1} - 1)$, $x(3 \cdot 2^{i-1} - 1)$, and $x2^{i-1}$, respectively. □

Lemma 2.3 Let record α belong to S_i and let j denote the rank of α in S_i, for some $i \geq 1$. Then the rank of α in S_0 lies in the range

$$(j2^{x(2^i-1)} - \sum_{0 \leq k < i} 2^{\delta - x2^k}, j2^{x(2^i-1)}].$$

Proof: The proof is by induction on i. The base case, $i = 1$, is a special case of Corollary 2.2.1.

Now let us assume that the claim holds inductively. Suppose that record α has rank j in S_i. Then by Corollary 2.2.1, the rank of α in the set S_{i-1} lies in the interval

$$(j2^{x2^{i-1}} - 2^{\delta - x(2^i-1)}, j2^{x2^{i-1}}].$$

Applying the induction hypothesis, the rank of record α in the set S_0 must be strictly greater than

$$(j2^{x2^{i-1}} - 2^{\delta - x(2^i-1)})2^{x(2^{i-1}-1)} - \sum_{0 \leq k < i-1} 2^{\delta - x2^k} \quad = \quad j2^{x(2^i-1)} - \sum_{0 \leq k < i} 2^{\delta - x2^k},$$

and at most

$$j2^{x2^{i-1}}2^{x(2^{i-1}-1)} \quad = \quad j2^{x(2^i-1)},$$

as required. □

Theorem 1 Let S denote a set of 2^δ records concentrated in a subcube Φ of size $2^{\delta+x}$ (x integer, $4 \leq x \leq \delta/2$), and let k be an integer, $0 \leq k < 2^\delta$. Then in $O(\delta)$ time it is possible to compute a subset S' of S and an integer k' such that the following conditions are satisfied:

(i) $|S'| = 2^{\delta-x+3}$,

(ii) $0 \le k' < |S'|$,

(iii) the record of rank k' in S' has rank k in S, and

(iv) S' is concentrated in Φ.

Proof: We begin by executing i iterations of the approximate selection algorithm where, as we shall see, $(\delta + x)/3 \le x2^i \le \delta + 1$. The approximate selection algorithm produces a set S_i of $2^{\delta - x(2^i-1)}$ records. By Lemma 2.1 the time is $O(x2^i) = O(\delta)$.

Next, the records in S_i are sorted using sparse enumeration sort. There are $2^{\delta-x(2^i-1)}$ records, $2^{\delta+x}$ processors, and an excess processor ratio of 2^{x2^i}. We choose i to be the smallest value such that the excess processor ratio is at least the square root of the number of records, $x2^i \ge \frac{1}{2}(\delta - x(2^i - 1))$. Solving for i yields $2^i \ge (\delta + x)/3x$ and $i = \left\lceil \lg \frac{\delta+x}{3x} \right\rceil$. The time for sparse enumeration sort is $O(\delta + x) = O(\delta)$.

We would now like to find two records, R_l and R_u in S_i, with ranks r_l and r_u in S_0, such that k belongs to the interval $(r_l, r_u]$ and $r_u - r_l$ is small. In the following, let $A = 2^{x(2^i-1)}$ and let $B = 2^{\delta-x+1}$. By Lemma 2.3, the key with rank j in S_i must lie in the interval $(jA - B, jA]$ in S_0. Let $j_l = \lfloor k/A \rfloor$, let $j_u = \lceil (k+B)/A \rceil$, and let R_l and R_u be the records in S_i with ranks j_l and j_u in S_i, respectively. Then $j_lA - B < r_l \le j_lA \le k$, and $k \le j_uA - B < r_u \le j_uA$. Note that $B = bA$, with b integer, and there exist integers α and β, $0 \le \beta < A$, such that $k = \alpha A + \beta$. Hence $j_u = \alpha + b + \lceil \beta/A \rceil \le \alpha + b + 1$, $j_l = \alpha$, and

$$\begin{aligned} r_u - r_l &\le (j_u - j_l)A + B \\ &\le (\alpha + 1 + b - \alpha)A + B \\ &= A + 2B. \end{aligned}$$

We will set S' to be the set of at most $A + 2B$ records in S_0 with ranks in $(r_l, r_u]$. Note that, given records R_l and R_u, it is straightforward to identify and concentrate the set S' in $O(\delta)$ time. For $A \le B$, we have $|S'| \le 3B < 2^{\delta-x+3}$. For $i = \left\lceil \lg \frac{\delta+x}{3x} \right\rceil$, the inequality $A \le B$ is satisfied since

$$\begin{aligned} x(2^i - 1) &\le x\left[2 \cdot \left(\frac{\delta + x}{3x}\right) - 1\right] \\ &= (\delta - x) - (\delta - 2x)/3 \\ &\le \delta - x, \end{aligned}$$

where the last inequality follows from the assumption that $x \le \delta/2$. The value of k' is determined by finding the rank of k in S', which can easily be done in $O(\delta)$ time. □

2.2 An $O(\lg n \lg \lg n)$ algorithm

Let us define $T(\delta, x)$ as the time required to select the kth element from a given set of 2^δ elements concentrated in a subcube of $2^{\delta+x}$ processors (for worst case k). Note that for $\delta' \le \delta$ and $x' \ge x$, we have $T(\delta', x') \le T(\delta, x)$; in what follows, we will occasionally make implicit use of this trivial inequality.

The subroutine corresponding to Theorem 1 gives

$$T(\delta, x) \leq T(\delta - x + 3, 2x - 3) + O(\delta) \tag{1}$$

for $4 \leq x \leq \delta/2$. For $x > \epsilon\delta$, where ϵ denotes an arbitrarily small positive constant, sparse enumeration sort implies that $T(\delta, x) = O(\delta)$. We are interested in obtaining an upper bound for $T(\delta, 0)$. Note that $T(\delta, 0) = \Theta(T(\delta, 4))$, since we can simulate a $2^{\delta+4}$-processor hypercube on a 2^{δ}-processor hypercube with only constant factor slowdown. By iterating the recurrence of Equation (1), we can obtain an upper bound for $T(\delta, 4)$. For $\delta \geq 8$, one application of the recurrence gives $T(\delta, 4) \leq T(\delta - 1, 5) + c\delta$ for some constant $c > 0$. For $\delta \geq 11$ we can apply the recurrence again to obtain $T(\delta, 4) \leq T(\delta - 3, 7) + 2c\delta$. In general, for $\delta \geq 2^i + 2^{i-1} + 5$, we can apply the recurrence i times to obtain $T(\delta, 4) \leq T(\delta - 2^i + 1, 2^i + 3) + ic\delta$ (this claim is easily verified by induction on i). For $\delta \geq 20$, we can set $i = \lfloor \lg \delta \rfloor - 1$ to obtain $T(\delta, 4) \leq T(\lfloor 3\delta/4 \rfloor + 1, \lceil \delta/4 \rceil + 3) + O(\delta \lg \delta) = O(\delta \lg \delta)$. Hence $T(d, 0) = O(d \lg d) = O(\lg n \lg \lg n)$.

This algorithm is essentially equivalent to that described by Plaxton in [8]. Prior to this algorithm, the best bounds known for selection on the hypercube were given by sorting algorithms.

2.3 An $O(\lg n \lg^{(3)} n)$ algorithm

Throughout this subsection, we will refer to the $O(\lg n \lg \lg n)$ selection algorithm of Section 2.2 as the "basic" algorithm. Our present goal is to improve the running time of the basic algorithm to $O(d \lg \lg d) = O(\lg n \lg^{(3)} n)$ by a simple modification. The basic algorithm consists of $O(\lg d)$ applications of the $O(d)$ approximation subroutine corresponding to Theorem 1. We will view each application of the approximation subroutine as a "phase" of the basic algorithm. In order to improve the performance of the basic algorithm, we will augment each phase in the following manner: before applying the approximation subroutine, we will partition the remaining data into subcubes of dimension δ', sort these subcubes completely, and extract a $2^{\delta''}$-sample from each subcube, where $\delta'' = \lceil \delta'/2 \rceil$. These samples are then passed on to the approximation subroutine for successive sampling.

The parameter δ' will be chosen in such a way that, given the excess processor ratio available at that particular phase, the sort can be completed in $O(d)$ time. The motivation for defining δ' in this manner is to balance the time spent on the initial sort with the $O(d)$ running time of the approximation subroutine. Sparse enumeration sort will be used to perform the initial sort in each phase.

We will now analyze the performance of each phase in greater detail. Before the phase, let S denote the set of remaining records, assume that $|S| = 2^{\delta}$, and assume that the excess processor ratio is 2^x, $x \geq 0$. At the beginning of the phase, we partition S into $2^{\delta-\delta'}$ sets of size $2^{\delta'}$, and sort each set in a subcube of dimension $\delta' + x$. We then extract a $2^{\delta''}$-sample from each sorted set, where $\delta'' = \lceil \delta'/2 \rceil$. Let S' denote the set of $2^{\delta-\delta''}$ records in the union of all of these samples. By Lemma 2.2, the key of rank j in S' has rank in the interval

$$(j2^{\delta''} - 2^a, j2^{\delta''}]$$

in S, where $a = \delta - \delta' + \delta''$. Accordingly, we will obtain a lower approximation for the kth record in S by computing a lower approximation (via Theorem 1) for the $\left\lfloor k2^{-\delta''} \right\rfloor$th record in

S'. Similarly, we will obtain an upper approximation for the kth record in S by computing an upper approximation for the $(\lceil k2^{-\delta''} \rceil + 2^{\delta-\delta'})$th record in S'.

By Theorem 1, in $O(\delta)$ time we can determine a set of at most 2^b records with contiguous ranks in S' that contains any desired rank, where $b = \delta - x - 2\delta'' + 3$. [To see this, apply Theorem 1 with the variables δ and x of the theorem replaced by the expressions $\delta - \delta''$ and $x + \delta''$, respectively.] In particular, we can obtain a lower approximation for the record of rank k' in S' that has rank strictly greater than $k' - 2^b$, and we can obtain an upper approximation with rank strictly less than $k' + 2^b$. Thus, in $O(\delta)$ time, we can determine:

(i) a lower approximation to the record of rank $\lfloor k2^{-\delta''} \rfloor$ in S' with rank strictly greater than $\lfloor k2^{-\delta''} \rfloor - 2^b$ in S', and

(ii) an upper approximation to the record of rank $\lceil k2^{-\delta''} \rceil + 2^{\delta-\delta'}$ in S' with rank strictly less than $\lceil k2^{-\delta''} \rceil + 2^{\delta-\delta'} + 2^b$ in S'.

With the bounds of the preceding paragraph, the aforementioned lower and upper approximations represent, respectively:

(i) a lower approximation to the record of rank k in S with rank strictly greater than $k - 2^{\delta''} - 2^a - 2^{b+\delta''}$ in S, and

(ii) an upper approximation to the record of rank k in S with rank strictly less than $k + 2^{\delta''} + 2^a + 2^{b+\delta''}$ in S.

Hence, within the same time bound we can identify a set of at most

$$z \stackrel{\text{def}}{=} 2 \cdot \left(2^{\delta''} + 2^a + 2^{b+\delta''}\right)$$

records with contiguous ranks in S and which contains the record of rank k in S. Observe that $a \geq \delta''$ and $a + 3 \geq b + \delta''$ (recall that $a = \delta - \delta' + \delta''$). Hence, we can conclude that $z \leq 2^{a+5}$.

Note that the initial application of sparse enumeration sort will run in $O(\delta)$ time if $\delta' = O(\sqrt{\delta x})$, since the running time of sparse enumeration sort is $O(\delta'(\delta' + x)/x) = O((\delta')^2/x)$. Accordingly, let us set $\delta' = \lceil c\sqrt{\delta} \rceil$ for some positive constant c. Note that for $x \geq 1$, $c > 1$, and δ sufficiently large, $2^{\delta-\delta'+\delta''+5} \leq 2^{\delta - \lceil \sqrt{\delta x} \rceil}$, and hence $z \leq 2^{\delta - \lceil \sqrt{\delta x} \rceil}$. As in Section 2.2 (where in fact we assumed that $x \geq 4$), we may assume that $x \geq 1$ without loss of generality.

Hence, the foregoing discussion has established the recurrence

$$\begin{aligned} T(\delta, x) &\leq T\left(\delta - \left\lceil \sqrt{\delta x} \right\rceil, x + \left\lceil \sqrt{\delta x} \right\rceil\right) + c'\delta \\ &\leq T\left(\delta, \left\lceil \sqrt{\delta x} \right\rceil\right) + c'\delta \end{aligned}$$

for $1 \leq x \leq \delta/2$ and some constant $c' > 0$. For $x \geq 1$ and $\lceil \sqrt{\delta x} \rceil \leq \delta/2$ we can iterate this recurrence to obtain

$$\begin{aligned} T(\delta, x) &\leq T\left(\delta, \left\lceil \sqrt{\delta \left\lceil \sqrt{\delta x} \right\rceil} \right\rceil\right) + 2c'\delta \\ &\leq T\left(\delta, \left\lceil \delta^{3/4} x^{1/4} \right\rceil\right) + 2c'\delta. \end{aligned}$$

More generally, for $x \geq 1$ and $\left\lceil \delta^{1-2^{-i}} x^{2^{-i}} \right\rceil \leq \delta/2$, $i \geq 0$, we can apply the recurrence i times to obtain

$$\begin{aligned} T(\delta, x) &\leq T\left(\delta, \left\lceil \delta^{1-2^{-i}} x^{2^{-i}} \right\rceil\right) + ic'\delta \\ &\leq T\left(\delta, \left\lceil \delta^{1-2^{-i}} \right\rceil\right) + ic'\delta. \end{aligned}$$

It is straightforward to verify that the recurrence can be applied $\lg\lg\delta + O(1)$ times, at which point we have

$$T(\delta, x) \leq T(\delta, y) + O(\delta \lg\lg\delta)$$

for some y with $\delta/2 < y < \delta$. Sparse enumeration sort implies that $T(\delta, y) = O(\delta)$ and hence $T(d, 0) = O(d \lg\lg d) = O(\lg n \lg^{(3)} n)$.

2.4 An $O(\lg n \lg^{(4)} n)$ algorithm

We can improve the time bound achieved in Section 2.3 by making use of the Sharesort algorithm of Cypher and Plaxton [5]. Several variants of that algorithm exist; in particular, detailed descriptions of two versions of Sharesort may be found in [5]. Both of these variants are designed to sort n records on an n-processor hypercubic network. The first algorithm runs in $O(\lg n (\lg\lg n)^3)$ time and the second algorithm, which is somewhat more complicated, runs in $O(\lg n (\lg\lg n)^2)$ time. The selection algorithm of this section will make use of Sharesort as a subroutine. For this purpose, either of the aforementioned variants of Sharesort may be used; this choice will not affect the overall running time by more than a constant factor. For the sake of concreteness, in the calculations that follow we will assume that the simpler $O(\lg n (\lg\lg n)^3)$ algorithm is used.

The only change to the algorithm of Section 2.3 is that in the initial phase, Sharesort will be used instead of sparse enumeration sort to perform the initial $O(d)$-time sort. With Sharesort, we can afford to set $\delta' = \Theta(d/(\lg d)^3)$, which is substantially larger than the $\Theta(\sqrt{d})$ bound achievable with sparse enumeration sort. For all phases subsequent to the first phase, however, we will make use of sparse enumeration sort. The reason is that, in the absence of a suitable processor-time tradeoff for the Sharesort algorithm, sparse enumeration sort is actually faster than Sharesort after the first phase (due to the large excess processor ratio created by the first phase). In Section 2.5, we will obtain an even faster selection algorithm by developing and applying an effective processor-time tradeoff for the Sharesort algorithm.

We will now analyze the running time of the selection algorithm of Section 2.3 when δ' is set to $\Theta(d/(\lg d)^3)$ in the first phase. The first phase establishes the inequality

$$T(d, 0) \leq T(d, \left\lceil d/(\lg d)^3 \right\rceil) + O(d).$$

Now $d/(\lg d)^3 = d^{1-2^{-i}}$ with $i = \lg\lg d - \lg^{(3)} d - O(1)$. Hence, the recurrence of Section 2.3 implies that $T(d, \lceil d/(\lg d)^3 \rceil) = O(d \lg^{(3)} d)$. Thus $T(d, 0) = O(d \lg^{(3)} d) = O(\lg n \lg^{(4)} n)$.

2.5 An $O(\lg n \lg^* n)$ algorithm

The improvement described in Section 2.4 resulted from applying Sharesort instead of sparse enumeration sort at the beginning of the first phase. Note, however, that all of the phases (including the first) continue to make extensive use of sparse enumeration sort. The calls to sparse enumeration sort made by each of the algorithms defined thus far may be partitioned into two classes: (i) those calls made within applications of Theorem 1, and (ii) those calls used to perform an $O(d)$-time sort (actually, a set of parallel $O(d)$-time sorts) before applying Theorem 1. The algorithm of Section 2.2 contains only calls of Type (i), since each phase consists solely of an application of Theorem 1. The algorithm of Section 2.3 contains both Type (i) and Type (ii) calls, since each phase consists of an $O(d)$-time sort followed by an application of Theorem 1. The algorithm of Section 2.4 is the same as the algorithm of Section 2.3, except that the Type (ii) call of the first phase is replaced with a call to Sharesort (causing the number of phases to be substantially reduced).

Could we obtain an even faster selection algorithm than that of Section 2.4 by replacing some or all of the remaining calls to sparse enumeration sort with calls to Sharesort? With regard to the Type (i) calls, the answer is no. Even if the Type (i) sorts were performed in optimal logarithmic time, the reduction in data (i.e., relevant records) between successive phases would not be improved significantly. The reason is that the amount of data that "survives" to the next phase is predominantly determined by the size of the subcubes sorted in the Type (ii) sorts. Thus, in all of the algorithms described in this paper, we will continue to make use of sparse enumeration sort to perform all of the sorts within applications of Theorem 1.

Now let us consider the Type (ii) calls. All of these calls to sparse enumeration sort will in fact be replaced with calls to a more efficient sorting algorithm in order to obtain the $O(\lg n \lg^* n)$ time bound claimed in the title of this section. As discussed in Section 2.4, we cannot obtain such a bound by merely replacing all of the Type (ii) calls to sparse enumeration sort with calls to one of the single-item-per-processor variants of Sharesort. Instead, we will proceed by developing a time-processor tradeoff for Sharesort, and then using the resulting algorithm to perform all of the Type (ii) sorts.

Theorem 2 Let n records be concentrated in a subcube of a p-processor hypercubic network $p^{2/3} \leq n \leq p$. There exists a non-uniform deterministic algorithm for sorting these records in time

$$O(\lg n(\lg\lg n - \lg\lg \tfrac{p}{n})). \tag{2}$$

Proof: As indicated in Section 2.4, there are a number of variants of the Sharesort algorithm of Cypher and Plaxton [5]. These algorithms differ solely in the way that the so-called *shared key sorting* subroutine is implemented. The shared key sorting problem represents a restricted version of the sorting problem; a formal definition of the shared key sorting problem will not be needed in this paper, and so will not be given. All variants of Sharesort make use of precisely the same recursive framework to reduce the problem of sorting to that of shared key sorting.

Perhaps the simplest variant of Sharesort runs in $O(\lg n \lg\lg n)$ time and relies upon an optimal logarithmic time shared key sorting subroutine. This particular result is mentioned in the original Sharesort paper [5] and more fully described by Leighton [6, Section 3.5.3]. Although it is the fastest of the Sharesort variants, this sorting algorithm suffers from the disadvantage that it is non-uniform. From the point of view of a user who would like to run this sorting algorithm on a particular hypercube of dimension d, what this non-uniformity implies is that a "setup" routine must be executed when the machine is first configured in order to generate a version of the algorithm that is capable of efficiently sorting any subcube of dimension less than or equal to d. Note that the setup routine need only be executed once in the lifetime of the machine (and not once per sort) and so this deficiency may not be considered overly severe. Unfortunately, the most efficient deterministic algorithms currently known for performing the setup task run in time that is doubly-exponential in d.

We will establish the validity of Equation 2 by developing a time-processor tradeoff for the $O(\lg n \lg\lg n)$ time, non-uniform variant of Sharesort.

As mentioned above, all variants of Sharesort are based on a particular system of recurrences. At the highest-level, sorting is performed recursively via *high-order merging* (i.e., merging n^{ϵ} sorted lists of length $n^{1-\epsilon}$ for some constant ϵ, $0 < \epsilon < 1$). The running time of Sharesort is dominated by the time required for high-order merging, which is itself performed recursively. Let $M(x, y)$ denote the task of merging x sorted lists of length y. One possible recurrence for performing the merge is (minor technical details related to integrality constraints are dealt with in [5] and will not be addressed here)

$$M(n^{1/5}, n^{4/5}) \le M(n^{4/45}, n^{16/45}) + M(n^{1/9}, n^{4/9}) + O(\lg n) + SKS(n), \tag{3}$$

where $SKS(n)$ denotes the time required to solve the shared key sorting problem. Note that all of the merge operations appearing in Equation (3) are of the form $M(x, x^4)$, so that it indeed provides a well-defined recurrence. For the $O(\lg n \lg\lg n)$ time, non-uniform variant of Sharesort, $SKS(n) = O(\lg n)$, and so the $SKS(n)$ term essentially disappears from the recurrence of Equation (3). In order to obtain the time bound of Equation (2), we will make use of additional processors in the following simple way: whenever a merging problem of the form $M(x, x^4)$ arises and x^5 is less than the excess process ratio, we will apply x^{10} processors to solve that merging subproblem in optimal $O(\lg x)$ time using sparse enumeration sort. A straightforward analysis shows that this modification to the $O(\lg n \lg\lg n)$ algorithm of Sharesort yields the sorting time bound of Equation (2). □

As suggested earlier, we now define the selection algorithm of the present section by modifying the $O(\lg n \lg^{(3)} n)$ algorithm of Section 2.3; all of the Type (ii) calls to sparse enumeration sort will be replaced with calls to the sorting algorithm of Theorem 2. In order to analyze the performance of this algorithm, one may simply repeat the analysis of Section 2.3, setting δ' to $\lceil c\delta/(\lg \delta - \lg x)\rceil$ for some sufficiently large positive constant c. Doing this, we obtain the recurrence

$$T(\delta, x) \le T\left(\delta, \left\lceil \frac{\delta}{\lg \delta - \lg x} \right\rceil\right) + c'\delta$$

for $1 \le x \le \delta/2$ and some constant $c' > 0$. For $x \ge 1$ and $\left\lceil \delta/\lg^{(i)}(\lg \delta - \lg x)\right\rceil \le \delta/2$, $i \ge 0$, we can apply the recurrence i times to obtain

$$T(\delta, x) \le T\left(\delta, \left\lceil \frac{\delta}{\lg^{(i)} \delta} \right\rceil\right) + ic'\delta.$$

In general, the recurrence can be applied $\lg^* d + O(1)$ times, and we find that $T(d, 0) = O(d \lg^* d) = O(\lg n \lg^* n)$.

2.6 A uniform $O(\lg n \lg^* n)$ algorithm

In Section 2.5, we proved the existence of an algorithm for selection that runs in $O(\lg n \lg^* n)$ time. However, as indicated in Section 2.5, that algorithm is non-uniform because it makes use of a non-uniform version of the Sharesort algorithm. In the present section we will establish the existence of a uniform selection algorithm with the same asymptotic complexity as the algorithm of Section 2.5.

The version of the Sharesort algorithm employed in Section 2.5 makes use of a non-uniform shared key sorting subroutine. The running time of the non-uniform shared key sorting subroutine is $O(\lg n)$, which is easily seen to be optimal. The fastest known uniform version of the shared key sorting subroutine takes $O(\lg n \lg\lg n)$ time, which leads to an $O(\lg n(\lg\lg n)^2)$ running time for the corresponding uniform variant of Sharesort [5]. In the following, we will show how to adapt this uniform version of Sharesort to obtain a uniform version of a "sparse" Sharesort, that is, an algorithm for sorting n records on $p \geq n$ processors. We will express the running time of SparseSharedKeySort() in terms of the two parameters n and p.

Theorem 3 Let n records be concentrated in a subcube of a p-processor hypercubic network $p^{2/3} \leq n \leq p$. There exists a uniform deterministic algorithm for sorting these records in time

$$O(\lg n(\lg\lg n - \lg\lg \tfrac{p}{n})^2). \tag{4}$$

Proof: To obtain the result, we have to show how to improve the time complexity of the shared key sorting procedure when we have a significant number of "extra" processors (i.e., $p \gg n$). Let $SSKS(n,p)$ be the time needed to solve the following *sparse shared key sorting problem*: Perform 2^b identical sorts of lists of size 2^a, with a and b such that $a + b = n$ and $b - a/2 = \Theta(a)$, on a hypercube of size p, $p \geq n$. By a similar analysis as that provided in Section 2.5, the theorem will follow if we can prove that

$$SSKS(n,p) = \lg n(\lg\lg n - \lg\lg \tfrac{p}{n}). \tag{5}$$

We now sketch certain minor modifications to the SharedKeySort() algorithm of [5] that will yield a "SparseSharedKeySort()" routine with the desired properties. The SharedKeySort() procedure consists of essentially two subroutine calls; these calls are made to the subroutines PlanRoute() and DoRoute(), respectively. As it happens, the recurrence leading to the $O(\lg n \lg\lg n)$ time complexity of the subroutine DoRoute() is similar to the recurrence governing the ShareMerge() subroutine discussed in Section 2.5. As a result, we can apply the same technique (cutting off the recurrence at an appropriate depth) to obtain the desired $O(\lg n(\lg\lg n - \lg\lg(p/n))$ time complexity for a version of the DoRoute() algorithm that makes use of $p \geq n$ processors.

On the other hand, it is not quite as easy to obtain the same time-processor tradeoff for PlanRoute(). Like DoRoute(), the running time of PlanRoute() is $O(\lg n \lg\lg n)$ (assuming n records and n processors). The difficulty is that the PlanRoute() recurrence takes the following form:

$$T(n) \leq T(\sqrt{n}) + O(\lg n \lg\lg n). \tag{6}$$

In this case, we find that $T(n) = O(\lg n \lg\lg n)$ regardless of the depth at which we may be able to cut off the recurrence. Instead, we can obtain the desired time-processor tradeoff for $\mathsf{PlanRoute}()$ by arguing that the $\lg\lg n$ factor appearing in Equation (6) can be replaced by a $\lg\lg n - \lg\lg(p/n)$ factor (assuming n records and $p \geq n$ processors). To see this, we observe that in the original $\mathsf{PlanRoute}()$ algorithm, the $O(\lg\lg n)$ factor corresponds to the number of "classes" into which the set of n input records is partitioned. This partitioning is performed via $O(\lg\lg n)$ successive calls to the $\mathsf{Balance}()$ subroutine. It is straightforward to argue that, given n records and $p \geq n$ processors, the number of calls to the $\mathsf{Balance}()$ subroutine, and hence the total number of classes, can be reduced to $O(\lg\lg n - \lg\lg(p/n))$. $\square$

We can now make use of Theorem 3 to obtain a uniform selection algorithm in the same way that Theorem 2 was used to define a non-uniform selection algorithm in Section 2.5. To determine the time complexity of this uniform selection algorithm, we can apply the analysis of Section 2.3 with δ' set to $\lceil c\delta/(\lg\delta - \lg x)^2\rceil$. We obtain the recurrence

$$T(\delta, x) \leq T(\delta, \left\lceil \frac{c\delta}{(\lg\delta - \lg x)^2} \right\rceil) + c'\delta$$

for $1 \leq x \leq \delta/2$, and some constant $c' > 0$. Applying this recurrence $\lg^*((\lg d)^2) + O(1) = \lg^* d + O(1)$ times, we find that $T(d, 0) = O(d \lg^* d) = O(\lg n \lg^* n)$.

3 Concluding Remarks

This paper has developed a number of asymptotically fast selection algorithms for hypercubic networks. Our analysis of these algorithms has focused on determining their running times to within a constant factor. In order to simplify the analysis, we have occasionally employed rather loose bounds, and so the multiplicative constants implicit in our O-bounds will be correspondingly pessimistic.

The time-processor tradeoff of Theorem 3, which led to our $O(\lg n \lg^* n)$ time selection algorithm, is likely to have other applications.

It is noteworthy that the algorithm devised by Cole and Yap for the powerful and abstract parallel comparison model has essentially pointed the way to the best known algorithms for realistic models of parallel computation.

References

[1] M. Ajtai, J. Komlós, W. L. Steiger, and E. Szemerédi. Deterministic selection in $O(\log\log n)$ parallel time. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, pages 188–195, May 1986.

[2] P. Beame and J. Håstad. Optimal bounds for decision problems on the CRCW PRAM. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 83–93, May 1987.

[3] R. Cole. An optimally efficient parallel selection algorithm. *IPL*, 26:295–299, 1988.

[4] R. Cole and C. K. Yap. A parallel median algorithm. *IPL*, 20:137–139, 1985.

[5] R. E. Cypher and C. G. Plaxton. Deterministic sorting in nearly logarithmic time on the hypercube and related computers. *Journal of Computer and System Sciences*, 47:501–548, 1993.

[6] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, and Hypercubes*. Morgan-Kaufmann, San Mateo, CA, 1991.

[7] D. Nassimi and S. Sahni. Parallel permutation and sorting algorithms and a new generalized connection network. *JACM*, 29:642–667, 1982.

[8] C. G. Plaxton. *Efficient Computation on Sparse Interconnection Networks*. PhD thesis, Department of Computer Science, Stanford University, September 1989.

[9] L. G. Valiant. Parallelism in comparison problems. *SIAM Journal on Computing*, 4:348–355, 1975.

[10] U. Vishkin. An optimal parallel algorithm for selection. In *Parallel and Distributed Computing,* Volume 4 of *Advances in Computing Research*, pages 79–86. JAI Press, Greenwich, CT, 1987.

Sorting-Based Selection Algorithms for Hypercubic Networks

P. Berthomé A. Ferreira B. M. Maggs S. Perennes C. G. Plaxton

Abstract

This paper presents several deterministic algorithms for selecting the kth largest record from a set of n records on any n-node hypercubic network. All of the algorithms are based on the selection algorithm of Cole and Yap, as well as on various sorting algorithms for hypercubic networks. Our fastest algorithm runs in $O(\lg n \lg^* n)$ time, very nearly matching the trivial $\Omega(\lg n)$ lower bound.

Keywords: Parallel algorithms, hypercube, selection.

Résumé

Dans cet article, nous présentons plusieurs algorithmes déterministes permettant de trouver le k-ième élément dans un ensemble ayant n enregistrements sur un réseau hypercubique quelconque. Tous ces algorithmes sont basés sur l'algorithme de sélection de Cole et Yap, ainsi que sur divers algorithmes de tri sur réseaux hypercubiques. Notre meilleur algorithme fonctionne en temps $O(\lg n \lg^* n)$, ce qui nous rapproche de la borne inférieure immédiate $\Omega(\lg n)$.

Mots-clés : Algorithmes parallèles, hypercube, sélection.