Tight Bounds for a Distributed Selection Game with Applications to Fixed-Connection Machines

*C. Greg Plaxton**

Department of Computer Science
University of Texas at Austin
Austin, TX 78712

Abstract

We define a distributed selection game that generalizes a selection problem considered by Kosaraju [7]. We offer a tight analysis of our distributed selection game, and show that the lower bound for this abstract communication game directly implies near-tight lower bounds for certain selection problems on fixed-connection machines. For example, we prove that any deterministic comparison-based selection algorithm on an $(n/\log n)$-processor bounded-degree hypercubic machine requires $\Omega(\log^{3/2} n)$ steps in the worst case. This lower bound implies a non-trivial separation between the power of bounded-degree hypercubic and expander-based machines. Furthermore, we show that the algorithm underlying our tight upper bound for the distributed selection game can be adapted to run in $O((\log^{3/2} n) \cdot (\log\log n)^2)$ steps on any $(n/\log n)$-processor hypercubic machine.

1 Introduction

The worst-case complexity of selection has been studied on a wide variety of models of parallel computation. In general, the complexity of selection is expressed in terms of two parameters: n, the number of input keys, and p, the number of processors. For distributed memory machines, we assume that each input key is provided at exactly one processor, and that each processor initially holds either $\lfloor n/p \rfloor$ or $\lceil n/p \rceil$ input keys. (The latter assumption can typically be relaxed when establishing lower bounds.) It is possible to treat the desired output rank k, $0 \leq k < n$, as a third parameter in the complexity bounds of any selection algorithm. More often, however, the parameter k is eliminated from the complexity bounds by considering worst-case k, which can normally be taken to be $\lfloor n/2 \rfloor$. We have eliminated k in this manner from all of our bounds.

Since linear-time sequential algorithms are known for selection (see [2], for example), it is natural to ask for what ranges of n and p it is possible to achieve work-optimal selection on a given model of parallel computation. For example, Cole [4] has developed an EREW PRAM algorithm for selection that is work-optimal when n exceeds p by an $\Omega((\log p) \cdot \log^* p)$ factor. This result precisely characterizes the complexity of selection on the EREW PRAM, except for a tiny range in which an $O(\log^* p)$ gap remains.

This paper focuses on determining the complexity of selection on certain classes of fixed-connection machines, especially the class of *bounded-degree hypercubic machines* (e.g., the butterfly, cube-connected cycles, and shuffle-exchange). As noted in [11], the sorting result of Leighton [8] and the token distribution result of Peleg and Upfal [10] imply that a selection algorithm designed for the PRAM by Vishkin [12] (which is work-optimal when n exceeds p by an $\Omega((\log p) \cdot \log\log p)$ factor) can be implemented to run in $O(n/p + (\log p) \cdot \log\log(n/p))$ time on a certain class of bounded-degree expander networks. For the interesting special case $p = n/\log n$, where the information-theoretic and diameter-based lower bounds are both $\Omega(\log n)$, this result yields a near-optimal time bound of $O((\log n) \cdot \log\log n)$.

We introduce a distributed selection game in which a set A of $a \cdot b$ distinct keys is initially evenly-distributed amongst a players, an integer k is given, $0 \leq k < a \cdot b$, and the players engage in successive rounds of communication to determine the key of rank k in A. We introduce a third parameter, c, to bound the communication allowed in each round of the game. The resulting game is formally defined in Section 2 as the (a, b, c)-selection game.

The (a, b, c)-selection game generalizes a selection problem considered by Kosaraju [7]. Kosaraju observed that a lower bound for selection on an n-

*This research was supported by the National Science Foundation under Grant No. CCR–9504145 and the Texas Advanced Research Program under Grant No. ARP–93–00365–461. Email: `plaxton@cs.utexas.edu`.

processor tree can be derived by studying the problem of selection on a linear array where the two processors at the endpoints of the array each initially hold a sorted list. (Remark: The main result of [7] is not this lower bound but rather a near-tight upper bound.)

More formally, let processors $0, \ldots, c$ be arranged in a linear array, assume that processor 0 (resp., c) initially holds a sorted list X (resp., Y) of b keys, $X \cap Y = \emptyset$, and assume that our task is to determine the key of some given rank k in $X \cup Y$ (e.g., the median). At each step of the computation, a processor of the array can transmit at most one key to each neighboring processor. Kosaraju sketches a simple $\Omega(c \cdot \log_c b)$ lower bound argument for this problem. The idea is that the set of c keys transmitted from one endpoint of the linear array during the first c steps cannot depend on the set transmitted from the other endpoint. Using this observation, it can be shown that after c steps an adversary can "give away information" about the input lists X and Y to reduce the original selection problem to a similar selection problem involving a pair of sorted lists of size $\Theta(b/c)$. In particular, the new selection problem satisfies the property that no copies of relevant keys exist at internal (i.e., non-endpoint) processors of the array.

A slightly different way to think about this lower bound argument is to imagine a game in which two players each have a sorted list of b keys, and are allowed to send one another copies of up to c keys in each "round" of computation. The adversary argument sketched above yields a lower bound of $\Omega(\log_c b)$ rounds for this problem. In formulating our definition of the (a, b, c)-selection problem, we are seeking to generalize this two-player selection game to an a-player game with the following property: Any lower bound for the the number of rounds needed to solve the (a, b, c)-selection problem should be useful for proving lower bounds for selection on fixed-connection machines with significantly higher interconnectivity (e.g., higher bisection bandwidth) than the linear array or tree. For our lower bound applications, we have in mind a fixed-connection machine where the a "players" correspond to a disjoint groups of processors, the parameter b corresponds to a lower bound on the number of keys initially stored in any group of processors, and the parameter c corresponds to an upper bound on the number of keys that can enter or leave any single group of processors in d steps, where d is the minimum distance (number of hops) between any two processors belonging to distinct groups.

Our analysis of the general (a, b, c)-selection game is substantially more complicated than the analysis sketched above for the special case $a = 2$. Why is the general game apparently so much more complicated to analyze? In the 2-player game, we can argue that after each round, all keys relevant to solving the remaining selection problem (after the adversary gives away information) reside in the endpoints of the linear array. In the a-player game, the adversary cannot in general afford to give away information to all players about all keys that have ever been involved in a single communication step. As a result, our analysis must cope with a complex structure of "shared" partial ordering information related to keys that have been involved in one or more communication steps, but for which the overall rank has not yet been given away.

Our bounds for the (a, b, c)-selection game lead to new upper and lower bounds for selection on bounded-degree hypercubic machines for what is arguably the most interesting range of the parameters n and p. Previous work on this problem has addressed the following two "extreme" cases:

- For $n \leq p$, Berthomé et al. [1] have proven that Cole's EREW PRAM algorithm [4] can be adapted to obtain a normal hypercube algorithm running in $O((\log n) \cdot \log^* n)$ time. This upper bound holds for any hypercubic machine. While this algorithm is not work-optimal, it comes within a $\log^* n$ factor of the trivial diameter-based $\Omega(\log n)$ lower bound. Thus, the complexity of selection is all but completely determined for $n \leq p$. (Furthermore, for $n \leq p^{1-\epsilon}$, $\epsilon > 0$, the complexity of selection is known to be precisely $\Theta(\log n)$.)
- For $n \geq p \cdot \log^2 p$, Plaxton [11] has proven that the complexity of normal hypercube algorithms for selection is $\Theta((n/p) \cdot \log\log p)$. This bound holds for any comparison-based algorithm running on any hypercubic machine (bounded-degree or otherwise). Thus, work-optimal selection algorithms are almost, but not quite, achievable for $n \geq p \cdot \log^2 p$.

In this paper, we consider the remaining range, namely, where $p \leq n \leq p \cdot \log^2 p$. Letting $n = p \cdot (\log p)^\alpha$, $0 \leq \alpha \leq 2$, our results may be summarized as follows. First, we establish a lower bound of $\Omega((\log p)^{1+\alpha/2})$. Second, we establish a near-tight upper bound of $O((\log p)^{1+\alpha/2} \cdot (\log\log p)^2)$.

The rest of the paper is organized as follows. In Section 2, we define the (a, b, c)-selection game. In Section 3, we introduce the notion of a pre-post tree, which plays an important role in our lower bound technique. In Section 4, we analyze a token game played on a tree that is used in our main lower bound argument. In Section 5, we prove a lower bound on the complexity of the (a, b, c)-selection game. In Section 6, we prove a matching upper bound on the complexity of the (a, b, c)-selection game. In Section 7, we apply our results for the (a, b, c)-selection game to obtain new upper and lower bounds for selection on various fixed-connection machines.

2 A Distributed Selection Game

Fix positive integers a, b, and c. In this section, we define a distributed selection game that we refer to as the (a, b, c)-*selection game*. The input to the (a, b, c)-selection game is a set A of $a \cdot b$ distinct keys drawn from some totally ordered universe, and an integer k, $0 \le k < a \cdot b$. The output of the (a, b, c)-selection game is the key of rank k in A.

The input set A is provided in the form of a collection of a disjoint size-b sets A_j, $0 \le j < a$. There are a *players* numbered 0 through $a-1$, and player j initially holds the set A_j. It is convenient to assume that each key in A has a unique *name*. Multiple copies of a given key may be generated during the course of the game; all such copies have the same name. The goal of the players is to determine the name of the key of rank k in A.

The computation proceeds in *rounds*. At the beginning of round i, $i \ge 0$, each player j holds some set of keys $A_{i,j} \subseteq A$. (Thus $A_{0,j} = A_j$.) We call $A_{i,j}$ the *hand* of player j at the beginning of round i. At the beginning of round i, there is another set of keys $B_i \subseteq A$ that we call the *pile*. The initial pile is empty, that is, $B_0 = \emptyset$. At the beginning of some round $i > 0$, a given key x in A may belong to many hands, and may also belong to the pile; as indicated in the preceding paragraph, all such copies of the key x have the same name.

Each round consists of three phases. In phase 0 of round i, the following information is supplied to every player: (i) the names and relative order of the keys in each hand, and (ii) the names and relative order of the keys in the pile. Let Π_i denote the set of all permutations of A that are consistent with this information. In phase 1 of round i, each player j chooses two sets of keys $A'_{i,j}$ and $B'_{i,j}$ such that $A'_{i,j} \subseteq A_{i,j}$, $B'_{i,j} \subseteq B_j$, and $|A'_{i,j}| + |B'_{i,j}| \le c$. In phase 2 of round i, the hands and pile for round $i+1$ are formed as follows:

$$\begin{aligned} A_{i,j+1} &= A_{i,j} \cup B'_{i,j}, \text{ and} \\ B_{j+1} &= B_j \cup \left(\bigcup_{0 \le j < a} A'_{i,j} \right). \end{aligned}$$

The elements of the sets $A'_{i,j}$ and $B'_{i,j}$ are chosen sequentially in phase 1, via a series of $|A'_{i,j}|+|B'_{i,j}|$ *read* or *write* steps. In a read step, player j selects a key x in B_i to be added to $B'_{i,j}$, and is given the rank of x in $A_{i,j}$. In a write step, player j selects a key x in $A_{i,j}$ to be added to $A'_{i,j}$, and is given the rank of x in B_i.

It is straightforward to prove by induction on i that a permutation of A belongs to Π_i if and only if it is consistent with all comparison information gathered up to and including phase 0 of round i. The algorithm terminates in the earliest round i for which every permutation in Π_i has the same key of rank k, that is, as soon as the desired key of rank k is uniquely determined in Π_i.

3 Pre-Post Trees

In this section we introduce a data structure called a *pre-post tree*. Pre-post trees will be used in Section 5 to represent certain sets of permutations.

A pre-post tree is a rooted tree with an associated set of keys. The children of each node in the tree are totally ordered. Each key in the set resides at a unique node in the tree. Each key is designated as either a *preorder*, *postorder*, or *wildcard* key. The set of preorder (resp., postorder) keys associated with a given node is totally ordered. For any node α of a pre-post tree, let $preorder(\alpha)$ (resp., $postorder(\alpha)$) denote the ascending sequence of preorder (resp., postorder) keys of α, and let $wildcard(\alpha)$ denote the set of wildcard keys of α.

Note that any subtree of a pre-post tree is also a pre-post tree. We now describe a non-deterministic *pre-postorder traversal* algorithm that can be applied to any pre-post tree. The output of the algorithm is a permutation of the keys associated with the given pre-post tree. For a pre-post tree with root node α, the output of a pre-postorder traversal is the concatenation of the following sequences: (i) $preorder(\alpha)$, (ii) an arbitrarily chosen subset of $wildcard(\alpha)$ in arbitrary order, (iii) for each child β of α (in ascending order), the output of a pre-postorder traversal of the subtree rooted at β, (iv) the remaining keys of $wildcard(\alpha)$ in arbitrary order, and (v) $postorder(\alpha)$.

Let a given pre-post tree T have n nodes numbered from 0 to $n-1$, and assume that node i contains w_i wildcard keys. Note that a pre-postorder traversal of T generates one of a set of $\prod_{0 \le i < n} (w_i + 1)!$ possible permutations.

Definition 3.1 *For any pre-post tree T with associated set of keys X, let $\Pi(T)$ denote the set of all* *permutations of X that could be generated by a pre-postorder traversal of T.*

Let x belong to *wildcard*(α) for some node α in a given pre-post tree T. We *fix x left (resp., right)* by designating x as a preorder key instead of a wildcard key, and appending (resp., prepending) x to the sequence *preorder*(α) (resp., *postorder*(α)). The result of such an operation is a new pre-post tree T' such that $\Pi(T') \subseteq \Pi(T)$.

Let T be a pre-post tree with associated set of keys X, π be a permutation drawn from the set $\Pi(T)$, x be a key in X residing at node α of T, y be a key in X residing at node β of T, and node γ be the least common ancestor of α and β in T. If γ is not equal to either α or β, then the relative order of keys x and y in π is uniquely determined in $\Pi(T)$; thus, comparing keys x and y would yield no new information about the permutation π.

Now assume without loss of generality that $\gamma = \alpha$; furthermore, if $\alpha = \beta = \gamma$ and exactly one of x and y is a wildcard key, then assume without loss of generality that x is the wildcard key. Under these assumptions, the relative order of keys x and y in π is uniquely determined in $\Pi(T)$ if and only if x is not a wildcard key. Thus, if x is a wildcard key and we are given the relative order of keys x and y in π, it is no longer true that π could be any permutation in $\Pi(T)$. In other words, if we let Π' denote the new set of possible permutations π (i.e., given the relative order of x and y), then $\Pi(T) \not\subseteq \Pi'$. On the other hand, it is easy to verify that if x is given to precede (resp., follow) y in π, then the tree T' obtained by fixing x left (resp., right) satisfies $\Pi(T') \subseteq \Pi'$.

The foregoing remarks suggest the following strategy for designing an adversary argument to lower bound the running time of a given comparison-based selection algorithm. Initially, the adversary restricts the input permutation to be drawn from the set $\Pi(T_0)$ associated with some (carefully constructed) pre-post tree T_0. In general, once the algorithm has made $i \geq 0$ comparisons, the adversary constructs a pre-post tree T_i such that the following two properties hold: (i) every permutation in $\Pi(T_i)$ is consistent with the outcomes of the first i comparisons, and (ii) either i is large (i.e., the lower bound has been attained) or the desired output key is not uniquely determined in $\Pi(T_i)$. After i comparisons, the adversary gives away information by guaranteeing that the input permutation is drawn from $\Pi(T_i)$.

The adversary constructs tree T_{i+1} from tree T_i by considering the keys involved in comparison i, $i \geq 0$, and determining whether or not the outcome of the comparison is uniquely determined in $\Pi(T_i)$. If so, T_{i+1} is set to T_i. If not, T_{i+1} is obtained from T_i by fixing (either left or right) one of the two keys involved in the comparison, as discussed above.

4 A Token Game

In this section we define and analyze a particular *token game*. The analysis of this game is used in the proof of our main technical lemma, Lemma 5.1 of Section 5.

Let $b = (9 \cdot c)^d$ for some nonnegative integer d. Let T be a complete $(9 \cdot c)$-ary tree of height h. Initially, each node of height i in T holds exactly $(36 \cdot c)^i$ *tokens*, each of which is "unmarked". Let $T_0 = T$ and for all integers i and j such that $0 \leq j < i$, define:

$$f(i,j) = \begin{cases} c \cdot (9 \cdot c)^{j-i} & \text{if } i-d \leq j < i, \text{ and} \\ c/b & \text{if } 0 \leq j < i-d. \end{cases}$$

Consider the following h-round game between two players 0 and 1. Each round consists of two phases. In phase 0 of round i, player 0 "marks" a subset of the tokens in T_i subject to the constraint that at most an additional $f(h-i,j)$ fraction of the $(9 \cdot c)^{h-i-j} \cdot (36 \cdot c)^j$ tokens of height j in T_i are marked, $0 \leq j < h-i$. (Remark: No restriction is placed on the number of keys marked in the root of T_i.) In phase 1 of round i, player 1 specifies a node α_i in T by selecting a random child of the root of T_i. Tree T_{i+1} is then defined as the subtree of T rooted at α_i.

Player 1 *succeeds* in round i if and only if at most $1/3$ of the tokens in node α_i have been marked by player 0 by the end of round i. Fix the algorithm (deterministic or randomized) used by player 0, and let X_i denote the random variable that is equal to 1 if player 1 succeeds in round i, and to 0 otherwise. Let $X = \sum_{0 \leq i < h} X_i$.

Lemma 4.1 *For the token game described above, we have*

$$E[X] = \Omega(\min\{h, b/c\}).$$

Proof: Let $Y_{i,j}$ be a random variable corresponding to the fraction of tokens in node α_i (also a random variable) marked by player 0 in round j, $0 \leq j \leq i$. Let $Y_i = \sum_{0 \leq j \leq i} Y_{i,j}$. (Note that $X_i = 1$ if and only if $Y_i \leq 1/3$.) By the definition of a round, and using linearity of expectation, we find that $E[Y_{i,j}] \leq f(h-j, h-i-1)$. Hence,

$$\begin{aligned} E[Y_i] &\leq \sum_{0 \leq j \leq i} f(h-j, h-i-1) \\ &= \sum_{0 \leq j \leq i-d} f(h-j, h-i-1) \end{aligned}$$

$$+ \sum_{i-d<j\le i} f(h-j, h-i-1)$$

$$= \left(\sum_{0\le j\le i-d} c/b \right) + \sum_{i-d<j\le i} c\cdot(9\cdot c)^{j-i-1}.$$

Now $\sum_{i-d<j\le i} x^{j-i-1} \le \sum_{i>0} x^{-i} = 1/(x-1)$ for $x > 1$. Furthermore, letting $h' = \min\{h, \lfloor b/(8\cdot c)\rfloor\}$, we have $\sum_{0\le j\le i-d} c/b \le h'\cdot c/b$ for all i such that $0 \le i < h'$. Substituting these bounds into the preceding inequality, we obtain

$$\begin{aligned} E[Y_i] &\le h'\cdot c/b + c/(9\cdot c - 1) \\ &\le 1/8 + 1/8 \\ &= 1/4, \end{aligned}$$

$0 \le i < h'$. Thus, $\Pr[Y_i \ge 1/3] \le 3/4$ and we find that $\Pr[X_i = 1] = E[X_i] \ge 1/4$, $0 \le i < h'$. Hence,

$$\begin{aligned} E[X] &\ge \sum_{0\le i<h'} E[X_i] \\ &\ge h'/4 \\ &= \min\{h, \lfloor b/(8\cdot c)\rfloor\}/4 \\ &= \Omega(\min\{h, b/c\}). \end{aligned}$$

■

5 A Lower Bound for (a, b, c)-Selection

In this section we establish Theorem 1, stated below. In Section 7 we show that Theorem 1 directly implies new selection lower bounds for certain fixed-connection machines.

Theorem 1 *Any deterministic (a, b, c)-selection algorithm requires*

$$\Omega\left(\min\left\{b/c, \log_c(a\cdot b)\right\}\right)$$

rounds in the worst case.

The proof of the following lemma makes use of an adversary argument of the kind discussed at the end of Section 3.

Lemma 5.1 *Any deterministic (a, b, c)-selection algorithm requires $\Omega(\min\{b/c, \log_c(a)\})$ rounds in the worst case.*

Proof: For ease of exposition, assume that $b = (9\cdot c)^d$ for some nonnegative integer d. Let T denote a complete $(9\cdot c)$-ary tree of height $d+h$, where h is a parameter to be specified momentarily. Associate $(36\cdot c)^i$ of the input sets A_j with each node of height $d+i$ in T, $0 \le i \le h$, under the restriction that no set is associated with more than one node. Since there are a input sets A_j, the height h must be chosen to satisfy the inequality

$$\begin{aligned} a &\ge \sum_{0\le i\le h} (36\cdot c)^i \cdot (9\cdot c)^{h-i} \\ &= \Theta(36\cdot c)^h. \end{aligned}$$

We choose h as large as possible while satisfying the above inequality; thus, $h = \Theta(\log_c a)$.

We now modify our initial assignment of keys to the nodes of tree T according to the following procedure, which is applied in parallel to each set A_j that has been assigned to some node of T. Let A_j be such a set, and assume that A_j is currently assigned to node α of height $d+i$, $0 \le i \le h$. Let $\langle \alpha_i : 0 \le i < b\rangle$ denote the sequence of b height-i descendant nodes of α. (The sequence is ordered from "left to right".) We now move the key of rank ℓ in A_j from node α to node α_ℓ, $0 \le \ell < b$.

Let T' denote an arbitrarily chosen subtree in the set of b subtrees of height h in T. Note that each node of height i in T' has $(36\cdot c)^i$ associated keys. Let A' denote the set of all keys associated with nodes in T'. Our assignment of keys to T' is easily seen to satisfy the following properties (which we refer to as properties 1 and 2 below), where T'' ranges over all subtrees of T', A'' is the set of all keys associated with nodes in T'', h'' is the height of T'', and X ranges over all input sets A_j:

1. No two keys in $X \cap A''$ are assigned to the same node of T''.
2. Suppose that $|X \cap A''| \ne 0$. Then all keys in $X \cap A''$ are assigned to nodes of the same height, say h^*, in T''. Furthermore, $|X \cap A''| = (9\cdot c)^i$ where $i = \max\{d, h'' - h^*\}$.

We now present an adversary argument that establishes the lemma. Let us interpret T' as a pre-post tree by making every key in A' a wildcard key. Our adversary argument will proceed in rounds, corresponding to the rounds of computation that occur in the (a, b, c)-selection game. During the course of the argument, many of the keys in A' will become preorder or postorder keys.

For any subtree T'' of T' with associated set of keys A'', let us define the phrase *the problem is confined to T''* to mean that the adversary "gives away information" by stipulating that the input permutation A and desired rank k satisfy the following properties: (i) the keys in A'' have contiguous ranks in A, (ii) each key in $A \setminus A''$ has a given rank in A, (iii) the desired key of rank k in A is the median of A'', and (iv) the relative order of the keys in A'' is given by some permutation in $\Pi(T'')$. In addition, we require that every permutation of A consistent with the aforementioned conditions (i) through (iv) also be consistent with any other comparison information the algorithm has learned so far. Before round 0, the problem is confined to T'.

We prove by induction that the adversary can be designed in such a way that the following invariants hold at the beginning of each round i, $i \geq 0$, until the game ends:

1. The problem is confined to some subtree T'' of T'.
2. At most 1/3 of the keys in the root of T'' are non-wildcard keys.
3. No key in B_i is a wildcard.
4. No key in $A_{i,j} \setminus A_j$ is a wildcard, $0 \leq j < a$.
5. The adversary argument continues to *mimic the token game* in the sense defined in the following paragraph.

Invariant 5 requires additional explanation. Note that the tree T' is closely related to the tree used in the token game of Section 4: (i) both are complete $(9 \cdot c)$-ary trees of height h, and (ii) interpreting the keys of tree T' as "tokens", both have the same number of tokens at each node of a given height. Furthermore, there is a correspondence between the process governing the "marking" of tokens in the token game and the process by which wildcard keys in A' become non-wildcard (i.e., preorder or postorder) keys in our adversary argument. More specifically, any deterministic algorithm for the (a, b, c)-selection game corresponds to some deterministic strategy for player 0 in the token game: The probability distribution determining the locations of the wildcard and non-wildcard keys in the sequence of $i+1$ subtrees T'' appearing in invariant 1 above is identical, for some deterministic strategy for player 0, to the probability distribution determining the locations of the marked tokens in the sequence of $i + 1$ subtrees occurring in the token game that consists of T_0 followed by the sequence of subtrees that are defined in phase 1 of those rounds where player 1 succeeds. As a result, we will be able to make use of Lemma 4.1 in the argument that follows.

Before proceeding with the proof by induction of invariants 1 to 5, we now complete the definition of the adversary. In the accompanying discussion of the adversary construction, we will assume that invariants 1 to 5 hold at the beginning of a given round i.

In phase 0 of round i, the adversary does nothing. In phase 1 of round i, the adversary processes each of the c sets of up to a read and write steps as follows.

Read/Write Claim: Each read or write step by player j forces the adversary to *commit to fix* (either left or right) at most one key in the set A_j, and no other keys. In saying that the adversary commits to fix a key x left (resp., right), we mean that the key x is declared to be the smaller (resp., larger) of the two keys in the corresponding comparison, but the pre-post tree T' is not yet updated by fixing x left (resp., right). (The updating of T' occurs during phase 2.) The read/write claim follows from: (i) the general procedure for resolving comparison queries in pre-post trees given in Section 3, (ii) property 1 of pre-post tree T', (iii) invariant 3, and (iv) invariant 4. Remark: If a write step by player j involves a wildcard key x in A_j, the adversary always commits to fix x (either left or right), even if the procedure for resolving comparison queries would not require such a commitment; this is done in order to preserve invariant 3.

Orthogonality Claim: Let us denote by $C(j)$ the "left/right" sequence of length at most c corresponding to the commitments made by the adversary with respect to player j during phase 1 as described in the read/write claim above. Then the behavior of any given player j during phase 1 does not depend on $C(j')$ for any $j' \neq j$. The correctness of the orthogonality claim is an immediate consequence of the definition of a round given in Section 2.

In phase 2, the adversary first updates the pre-post tree T' by fixing left (resp., right) each key committed to being fixed left (resp, right) during phase 1, with the exception of any keys associated with the root of the subtree T'' defined in invariant 1. (These keys will be fixed momentarily.) By the orthogonality claim and property 2 of tree T', the wildcard keys associated with root of subtree T'' can be fixed left or right in any manner, without contradicting the commitments made (and just enforced in T') to keys associated with nodes below the root. The adversary exploits this freedom with respect to the root of T'' to re-establish invariant 1 with respect to some proper subtree of T'' (which plays the role of T'' in the next round). This subtree, call it T''', is chosen in a manner that corresponds to the token game of Section 4: The adversary follows a randomly chosen downward path in T'' until a node α is reached where at most 1/3 of the associated keys are non-wildcard keys; subtree T''' is then defined as the subtree rooted at α. (The game ends if the adversary would otherwise attempt to extend the random downward path below a leaf node in T''.) The adversary then confines the problem to T'''. That it is possible to confine the problem to T''' may be seen as follows: (i) at least 3/4 of the keys in T'' are located in the root of T'' (due to the fact that the number of keys at each height $\ell + 1$ is 4 times the number at height ℓ), (ii) at least 2/3 of the keys at the root are wildcard keys at the beginning of round i by invariant 2, (iii) hence the desired key of rank k in A, which is the median of the keys in T'' by invariant 1, can be forced to be the key of *any* particular rank in *any* proper subtree T^* of T'', by appropriately fixing all remaining wildcard keys located at nodes outside T^*.

We now proceed with the proof by induction. It is straightforward to prove that invariants 1 through 5 hold for $i = 0$, the base case. The full details of the induction step are not difficult, but are somewhat lengthy, and so will be postponed to the full version of the paper. For the present exposition, we simply point out which facts are used to re-establish that each of invariants 1 through 5 hold at the beginning of round $i + 1$:

1. The re-establishment of invariant 1 follows from the argument given at the end of the preceding paragraph.
2. The re-establishment of invariant 2 follows from the procedure used by the adversary in phase 2 of round i to select the node α (the root of the "new" subtree T'').
3. The re-establishment of invariant 3 follows from invariant 3 and the fact that every key "written" to the pile B_{i+1} in phase 2 of round i is fixed either left or right in phase 2 of round i.
4. The re-establishment of invariant 4 follows from invariants 3 and 4 since every key "read" by player j during round i is taken from pile B_i.
5. The re-establishment of invariant 5 follows from invariant 5, property 2 of the tree T', the read/write claim, and the bound of c on the number of read/write steps made by each player in a round.

This completes the inductive proof of invariants 1 through 5.

By invariant 5 and Lemma 4.1, the expected number of rounds in the game is $\Omega(\min\{h, b/c\})$. (The expectation is taken with respect to the random path-building decisions made by the adversary; the algorithm itself is deterministic.) This completes the proof of Lemma 5.1, since the algorithm requires $\Omega(\min\{h, b/c\})$ rounds with non-zero probability. ■

The following lemma is due to Kosaraju [7].

Lemma 5.2 *For $a \geq 2$, any deterministic algorithm for (a, b, c)-selection requires $\Omega(\log_c b)$ rounds in the worst case.*

We now make use Lemmas 5.1 and 5.2 to prove Theorem 1, stated at the beginning of this section. For $b \leq a$, the lower bound of Lemma 5.1 is equivalent to the lower bound of Theorem 1. Furthermore, for $b \geq a$ the lower bound of Lemma 5.2 is equivalent to the lower bound of Theorem 1 since $b/c = \Omega(\log_c b)$ and $\log_c b = \Theta(\log_c (a \cdot b))$.

6 A Matching Upper Bound for (a, b, c)-Selection

In this section we prove the following theorem, which matches the lower bound given in the preceding section. Although this theorem does not *directly* imply any upper bounds for realistic models of parallel computation such as fixed-connection machines, we show in Section 7 that the algorithm given in the proof of Theorem 6 can in fact be efficiently implemented on any hypercubic machine. Moreover, the running time of the resulting algorithm is extremely close to the corresponding lower bound implied by Theorem 1.

Theorem 2 *There is a deterministic algorithm for (a, b, c)-selection that uses*

$$O\left(\min\left\{b/c, \log_c (a \cdot b)\right\}\right)$$

rounds in the worst case.

The idea behind the algorithm of Theorem 2 is that each player maintains a set of *live* keys that could still possibly represent the desired key of rank k in A. Initially, all keys are live. The algorithm iteratively applies the same two-round procedure in each pair of subsequent rounds until the game ends. At the beginning of phase 1 of a general round $2 \cdot i$, $i \geq 0$: (i) each player j holds a set $L_{i,j}$ of live keys such that $L_{i,j} \subseteq A_j$, (ii) the set L_i defined as $\cup_{0 \leq j < a} L_{i,j}$ spans a contiguous interval of ranks in A that includes the desired rank k, and (iii) the rank, say k_i, in L_i of the desired key of rank k in A is known to all players. The correctness of this claim is immediate for $i = 0$.

During phase 1 of round $2 \cdot i$, $i \geq 0$, each player j writes a set of c evenly-spaced *splitter* keys of the set $L_{i,j}$ (e.g., every key with rank in $L_{i,j}$ equal to some integer multiple of $\lceil |L_{i,j}|/c \rceil$) to the pile. (Any previous contents of the pile are not used by the algorithm and may be ignored.) When the relative order of keys in the pile is revealed in phase 0 of round $2 \cdot i + 1$, it is straightforward to argue that the players can determine a pair of keys x and y in the pile such that: (i) x has rank r_x less than or equal to k_i in L_i (i.e., x is a *lower approximation* to the desired key), (ii) y has rank r_y greater than or equal to k_i in L_i (i.e., y is an *upper approximation* to the desired key), (iii) $r_y - r_x \approx |L_i|/c$. (The technique is the same as that used in analyzing the linear-time selection algorithm of Blum et al. [2].)

During phase 1 of round $2 \cdot i + 1$, each player reads the lower and upper approximation keys x and y. During phase 0 of round $2 \cdot i + 2$, each player can determine the ranks r_y and r_x precisely. If $r_x = k_i$ or $r_y = k_i$, then we are done. Otherwise, the set $L_{i+1,j}$ is defined as the set of all keys in A_j with ranks strictly between x and y, k_{i+1} is set to $k_i - r_x - 1$, and the algorithm continues. It is easy to verify that k_{i+1} and the $L_{i+1,j}$'s satisfy the properties claimed above to hold at the start of phase 1 of round $2 \cdot (i + 1)$.

Because the set of live keys is cut by a factor of approximately c in each pair of rounds, the total number of rounds is $O(\log_c(ab))$.

An upper bound of $O(b/c)$ is trivial to achieve by a different algorithm. Note that each player j can move a copy of every key in A_j to the pile within $\lceil b/c \rceil$ rounds. Once the pile contains all of the keys, the entire input permutation of A is known to all players, and so the desired key of rank k is uniquely determined.

By interleaving the $O(\log_c(ab))$ algorithm with the $O(b/c)$ algorithm we can obtain an algorithm that runs in $O(\min\{b/c, \log_c(ab)\})$ rounds. This completes the proof of Theorem 2.

7 Applications to Fixed-Connection Machines

In this section we state a number of upper and lower bounds for selection problems on fixed-connection machines. We begin with the lower bounds, which follow directly from Theorem 1. Since the results are straightforward applications of Theorem 1, the proofs have been omitted.

The input conventions and other assumptions concerning the model of computation used in our lower bound arguments are as follows: (i) each input key is provided at exactly one processor, (ii) the only operations allowed on keys are copy and comparison, and (iii) each processor can transmit at most one key to each neighboring processor in a single step. (Remark: Assumption (iii) could be replaced by an assumption that each processor can transmit at most one key to a single neighboring processor in a single step; such a revised assumption would affect Definition 7.1 below but would not otherwise affect any of the results stated in this section.) For simplicity, we will also assume that: (iv) each processor initially holds either $\lfloor n/p \rfloor$ or $\lceil n/p \rceil$ input keys. In most cases of interest it is not difficult to relax assumption (iv).

Note that our lower bounds make no assumption concerning: (i) the amount of non-key information transmitted to a neighboring processor in a single step, and (ii) the amount of local computation (including key comparisons) performed in a single step. Note that the $\Omega((n/p) \cdot \log\log p)$ lower bound established for selection on a wide class of fixed-connection machines (including all hypercubic machines) in [11] differs in the sense that only one local key comparison is allowed at each step.

Definition 7.1 *Fix n and p. An (a, b, c, d)-*machine *is a p-processor fixed-connection machine that contains a disjoint "groups" of processors such that: (i) $b = s \cdot \lfloor n/p \rfloor$ where s denotes the least number of processors in any single group, (ii) c is an upper bound on the number of keys that can be sent out of any single group in d steps, and (iii) d denotes the minimum distance (number of hops) between any two processors in distinct groups.*

Theorem 3 *For any (a, b, c, d)-machine, selection requires $\Omega(d \cdot \min\{b/c, \log_c(ab)\})$ steps in the worst case.*

Definition 7.2 *A p-processor bounded-degree fixed-connection machine is $(f(p), g(p))$-*partitionable *if and only if the processors of the machine can be partitioned into $\Theta(p/f(p))$ disjoints sets of size $\Theta(f(p))$ such that $O(g(p))$ keys can be sent out of any single set of processors in any one time step.*

Lemma 7.1 *Any p-processor bounded-degree fixed-connection machine that is $(f(p), g(p))$-partitionable with $\log(p/f(p)) = \Theta(\log p)$ is a $(\Theta(p/f(p)), \Theta(f(p) \cdot n/p), \Theta(g(p) \cdot \log p), \Theta(\log p))$-machine.*

The class of bounded-degree hypercubic machines can be divided into "butterfly-like" (e.g., butterfly, cube-connected cycles) and "shuffle-exchange-like" (e.g., shuffle-exchange, de Bruijn) machines [9]. The lemma stated below is straightforward to prove for butterfly-like machines, but is quite non-trivial for shuffle-exchange-like machines. The proof for shuffle-exchange-like machines is due to Cypher [5].

Lemma 7.2 *Every p-processor bounded-degree hypercubic machine is $(f(p), f(p)/\log p)$-partionable for all $f(p) = O(p)$.*

For the following two theorems, let $n = p \cdot (\log p)^\alpha$, $0 \le \alpha \le 2$.

Theorem 4 *For any p-processor bounded-degree hypercubic machine, selection requires $\Omega((\log p)^{1+\alpha/2})$ steps in the worst case.*

Remark: The proof of Theorem 4 makes use of Lemma 7.2 with $f(p) = 2^z$ where $z = (\log p)^{1-\alpha/2}$.

The algorithm underlying Theorem 2 can be efficiently implemented as a normal hypercube algorithm to obtain:

Theorem 5 *There is a selection algorithm for any p-processor bounded-degree hypercubic machine that uses $O((\log p)^{1+\alpha/2} \cdot (\log\log p)^2)$ steps in the worst case.*

Remark: This algorithm makes use of the Sharesort sorting algorithm of Cypher and Plaxton [6]. Sharesort is a normal hypercube algorithm which, for $n = p$, runs in $O((\log p) \cdot (\log\log p)^2)$ time on any hypercubic machine. For $n \ge p$, this immediately implies an upper bound of $O((n/p) \cdot (\log p) \cdot (\log\log p)^2)$ for any hypercubic machine.

8 Concluding Remarks

It is worth noting that any improvement in the upper bound of $O((\log p) \cdot (\log\log p)^2)$ for sorting p records on a p-processor hypercubic machine would imply a corresponding improvement in the upper bound of Theorem 5. In particular, an optimal $O((n/p) \cdot \log p)$ sorting algorithm for hypercubic networks would improve the upper bound of Theorem 5 to $O((\log p)^{1+\alpha/2})$, matching the lower bound of Theorem 4.

The factor of $(\log\log p)^2$ in the bound of Theorem 5 can be improved slightly with existing techniques. The current bound assumes that a (one item per processor) sorting subroutine is used to determine each successive pair of lower and upper approximation keys x and y within the algorithm of Section 6. However, a (one item per processor) weighted selection algorithm is sufficient to determine this pair of keys. Chen and Gupta [3] have recently established an $O((\log p) \cdot \log^* p)$ upper bound for weighted selection. By making use of this weighted selection algorithm, we can improve the upper bound of Theorem 5 by a factor proportional to the square root of the ratio $(\log\log p)^2$ over $\log^* p$, that is, to $O((\log p)^{1+\alpha/2} \cdot (\log\log p) \cdot \sqrt{\log^* p})$.

Acknowledgments

The author would like to thank Bob Cypher and Rajmohan Rajaraman for their valuable comments on this work.

References

[1] P. Berthomé, A. Ferreira, B. M. Maggs, S. Perennes, and C. G. Plaxton. Sorting-based selection algorithms for hypercubic networks. In *Proceedings of the 7th International Parallel Processing Symposium*, pages 89–95, April 1993.

[2] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *JCSS*, 7:448–461, 1973.

[3] D. Z.. Chen and A. K. Gupta. Weighted selections on coarse-grain hypercubes. In *Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing*, October 1995. To appear.

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

[5] R. E. Cypher. Theoretical aspects of VLSI pin limitations. *SIAM J. Comput.*, 22:58–63, 1993.

[6] R. E. Cypher and C. G. Plaxton. Deterministic sorting in nearly logarithmic time on the hypercube and related computers. *JCSS*, 47:501–548, 1993.

[7] S. R. Kosaraju. Pipelining computations in a tree of processors. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 184–189, October 1989.

[8] F. T. Leighton. Tight bounds on the complexity of parallel sorting. *IEEE Transactions on Computers*, C–34:344–354, 1985.

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

[10] D. Peleg and E. Upfal. The token distribution problem. *SIAM J. Comput.*, 18:229–243, 1989.

[11] C. G. Plaxton. On the network complexity of selection. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 396–401, October 1989.

[12] 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.