On the Network Complexity of Selection

C. Greg Plaxton*
Department of Computer Science
Stanford University
Stanford, CA 94305

Abstract

The selection problem is to determine the kth largest out of a given set of n keys, and its sequential complexity is well known to be linear. Thus, given a p processor parallel machine, it is natural to ask whether or not an $O(n/p)$ selection algorithm can be devised for that machine. The main result of this paper is an $\Omega((n/p)\log\log p + \log p)$ lower bound for selection on any network that satisfies a particular low expansion property. The class of networks satisfying this property includes all of the common network families such as the tree, multi-dimensional mesh, hypercube, butterfly and shuffle exchange. When n/p is sufficiently large (for example, greater than $\log^2 p$ on the butterfly, hypercube and shuffle exchange), this result is matched by the upper bound presented in [4].

1 Introduction

This paper is concerned with deriving a lower bound on the complexity of the selection problem for a certain large class of networks. Given a set S of n keys and an integer k, $0 \leq k < n$, the selection problem is to determine a key with rank k in S. If all of the keys are distinct, there will be a unique key with rank k. The lower bound will apply to the special case of the selection problem where all keys are distinct and the key being sought is the median of S, that is, $k = \lfloor n/2 \rfloor$. The only operations allowed on keys are copy and comparison.

Given a network corresponding to the graph $G = (V, E)$, let the *expansion* of any subset U of the vertices (processors) be defined as $|\Gamma(U)|/|U|$, where $\Gamma(U)$ denotes the set of vertices in $V \setminus U$ that are adjacent (connected by a communication channel) to at least one vertex in U. The *expansion of a network* is the minimum over all $U \subseteq V$ such that $1 \leq |U| \leq |V|/2$ of the expansion of U.

The main result of this paper is an $\Omega((n/p)\log\log p + \log p)$ lower bound for selection on any network that satisfies a particular low expansion property defined in Section 5. The class of networks satisfying this property includes all of the common network families such as the tree, multi-dimensional mesh, hypercube, butterfly and shuffle-exchange. The lower bound is proven in Sections 4 and 5. When n/p is sufficiently large (for example, greater than $\log^2 p$ on the hypercube and shuffle-exchange), the lower bound is tight to within a multiplicative constant. The matching upper bound is provided by the algorithm Select presented in [4]. Section 3 contains a brief description of this algorithm.

Due to space limitations, a number of proofs and additional remarks have been omitted from this version of the paper. The statement of a theorem or lemma for which the proof has been omitted is immediately followed by a □. The reader is referred to [5] for a complete version.

*This work was supported in part by a grant from the AT&T Foundation, NSF grant DCR-8351757 and ONR grant N00014-88-K-0166. The author is primarily supported by a 1967 Science and Engineering Scholarship from the Natural Sciences and Engineering Research Council of Canada.

2 The Network Model

A p processor fixed interconnection network may be viewed as an undirected graph, where vertices correspond to processors and edges correspond to communication channels. Each processor has an infinite local memory. There is no global memory; processors communicate with one another by sending and receiving data over the channels provided by the network. In order to discuss the time complexity of an algorithm it is necessary to define exactly what operations can be performed in a single unit of time, or *time step*. For establishing asymptotic upper bounds, it is realistic to assume that:

1. Memory is configured in $O(\log p)$ bit words.
2. In a single time step, a processor can send and/or receive $O(1)$ words of data and perform $O(1)$ CPU operations on word-sized operands.

For the particular problem of selection it is necessary to make additional assumptions regarding the size of the keys and the operations that are allowed to be performed on them. It will be assumed that a key can be stored in a constant number of words and that the only operations allowed on keys are copy and comparison. It will also be assumed that $n = O(p^c)$ for some positive constant c. Having bounded n in this manner, it may be assumed without loss of generality that all of the keys are distinct [4].

The selection algorithm presented in [4] obeys all of the above restrictions. Section 3 contains a brief outline of that algorithm along with a summary of its asymptotic performance on a variety of networks. The lower bound for selection proven in this paper applies under a strictly more powerful model of network computation than that defined above. In particular, the only restrictions enforced by this model are the following:

1. Each processor can send and/or receive at most one key per time step.
2. The only operations allowed on keys are copy and comparison, and each processor can perform at most one such operation per time step.

Note that an unlimited amount of computation and communication involving data other than keys can be performed in each time step. Under this model, it will be proven that any selection algorithm running on a network satisfying a particular low expansion property requires $\Omega((n/p)\log\log p + \log p)$ time steps in the worst case.

The two models of network computation defined in this section will be referred to as the *upper bound model* and *lower bound model*, respectively.

3 Upper Bounds

For the EREW PRAM, Cole has exhibited an algorithm with a running time of $O(n/p + \log p \log^* p)$ [2]. Note that Cole's algorithm provides optimal speedup for $n = \Omega(p \log p \log^* p)$. The network algorithm Select presented in [4] achieves nearly optimal speedup when the ratio n/p is sufficiently large. To be precise, assume that a particular network is capable of sorting $n = p$ keys located one per processor in time T_1 and can perform broadcasting and summing operations in time T_2. Then the running time of algorithm Select may be written as

$$O((n/p)\log\log p + (T_1 + T_2 \log p)\log(n/p)). \qquad (1)$$

Briefly, algorithm Select operates as follows. Initially, there are n/p keys at each processor, all of which could potentially be the key x being sought, that is, the key of rank k in the entire set S of n keys. Select operates in $\log(n/p)$ stages, and at each stage the number of "live" keys (keys that could be x) is reduced by a constant fraction. The difference is that Select achieves this reduction at every processor without redistributing the keys.

The elimination of keys at each stage is based upon determining a particular key y with rank in S "close" to that of x, as follows. First, each processor computes the median of the set of live values that it holds using an efficient sequential algorithm. The resulting set of p medians is then sorted, and a binary search is performed over the set of sorted medians to determine that median with rank in the set of all live items that is closest to the rank of x. This median is the desired key y and, as proven in [4], it can be used to eliminate half of the live items at every processor. In order to facilitate the $\log p$ rank computations performed by the binary search over the set of p medians, the ith stage begins by spending $O(2^{-i}(n/p)\log\log p)$ time performing local preprocessing of the set of live items at each processor. The corresponding sequential tradeoff between preprocessing time and search time is well understood, see [1] and [3]. The running time of the ith stage is readily shown to be $O(2^{-i}(n/p)\log\log p + T_1 + T_2 \log p)$, which implies the overall time bound for Select stated in Equation (1).

Consider the performance of algorithm Select on a number of common network families. For the butterfly, hypercube and shuffle exchange, $T_1 = O(\log^2 p)$ and $T_2 = \Theta(\log p)$, so the second term in Equation (1) is $\log^2 p \log(n/p)$, and for $n = \Omega(p \log^2 p)$ the running time of Select is $O((n/p)\log\log p)$. For the d-dimensional mesh (d constant), $T_1 = T_2 = \Theta(p^{1/d})$, so the first term dominates for $n = \Omega(p^{1+1/d}\log^2 p/\log\log p)$. Let T_3 denote the time required for a given network to perform selection over $n = p$ keys located one at each processor. If $T_3 \log p < T_1$ then the second term in Equation (1) can be reduced to $(T_2 + T_3)\log p \log(n/p)$ by implementing each of the $\log(n/p)$ binary searches over p medians with $\log p$ selection operations rather than a single sort.

4 A Restricted Lower Bound

An $\Omega((n/p)\log\log p - \log p/\log\log p)$ time lower bound will now be proven for selection on a complete network with restricted capability. In order to simplify the exposition, it will be assumed that n and p are both powers of 2, $n \geq p$. Recall that S denotes the set of n keys, and that there are initially n/p keys located at each of the p processors. The lower bound established in this section applies under the model of computation defined below.

Definition 4.1 The *restricted lower bound model* is equivalent to the lower bound model defined in Section 2, with the following additional restriction. For every $R \subseteq S$, if R is initially assigned to a set of processors X, then at most $|X|$ comparisons per time step can involve keys from R.

The motivation for the restricted lower bound model is that networks with poor expansion properties suffer from a similar, albeit less severe, inability to spread out a concentrated set of data in order to apply more processors to it. In particular, if the size of the neighborhood of a given set of processors X is $\alpha|X|$ where $\alpha = o(1/\log\log p)$, then in $O((n/p)\log\log p)$ time not even a constant fraction of the $|X|n/p$ keys initially located in the set X can have been moved or copied to processors outside of X. Of course, the restricted lower bound model is, by itself, quite unrealistic. Furthermore, it is unclear whether or not selection can be performed in $O((n/p)\log\log p)$ time on this model, even assuming the complete network. However, the preceding observation indicates that it may be possible to transfer a lower bound for the complete network operating under the restricted lower bound model to a realistic network operating under the lower bound model. This is precisely the plan that will be followed in the rest of Section 4 and in Section 5.

The remainder of this section is concerned with proving an $\Omega((n/p)\log\log p - \log p/\log\log p)$ lower bound on the running time of any algorithm for computing the median on the complete network operating under the restricted lower bound model. The proof makes use of an adversary argument. At each time step, the algorithm indicates what set of at most p comparisons it would like to make, and the adversary resolves these comparisons sequentially. Of course, the algorithm must respect the rules of the restricted lower bound model, and the adversary must resolve comparisons in a manner that is consistent with at least one total ordering of the keys. A procedure by which the adversary resolves comparisons will be defined later. Aside from this procedure, which resolves comparisons as they occur, the proof proceeds in *stages* consisting of a number of consecutive time steps. Let λ denote the number of stages; it will turn out that $\lambda = \Theta(\log p/\log\log p)$. The ith stage begins at time $t_i = \max_{0\le j\le i}\lfloor (n/p)d_j\rfloor - j$ and ends at time t_{i+1}, where $d_i = \frac{1}{2}\log(i+1)$, $0 \le i < \lambda$. Note that $t_0 = 0$ and $t_{i+1} \ge t_i$, $0 \le i < \lambda$. The following pair of technical lemmas will be useful for bounding the amount of work that can be performed in a single stage, and up to a particular stage.

Lemma 4.1 There are at most $\frac{1}{2\ln 2}\frac{n}{p(i+1)}$ time steps in the ith stage. □

Lemma 4.2 The starting time of the ith stage, t_i, satisfies the inequality $t_i \le \frac{1}{2}(n/p)\log(i+1)$, $0 \le i < \lambda$. □

At the beginning of each stage, the adversary guarantees that the algorithm cannot yet have computed the median by making sure that more than one key could still be the median. At the same time, the adversary will generally give away the outcomes of a certain number of comparisons. This is done in order to simplify the lower bound argument. Note that giving away such additional information can only help the algorithm.

At the beginning of stage 0 (i.e., at time step 0), certain information is given to the algorithm for free. Several definitions are necessary in order to describe this information. Let a *concentrated block* of processors be defined as a set of processors B such that the $|B|n/p$ keys initially located in the set B have contiguous ranks in the set of all keys S. It will be convenient to define the concept of a *block at level* i, $0 \le i < \lambda$, which is a certain type of concentrated block. All blocks at level i contain the same number of processors, s_i, and they are pairwise disjoint. The set of all p processors is the unique block at level 0, so $s_0 = p$. Note that the block at level 0 is trivially a concentrated block. The adversary guarantees that every block at level i will contain exactly $b_i = 2^{\lceil \log(i+1)\rceil}$ blocks at level $i+1$, $0 \le i < \lambda - 1$. The size of the blocks at level $i+1$ is such that the union of the blocks at level $i+1$ within a given block B at level i contains $\gamma|B|$ processors. In other words, $s_{i+1} = \gamma s_i/b_i$ and hence

$$s_i = \frac{\gamma^i p}{b_0 \cdots b_{i-1}},$$

$0 \le i < \lambda$. Setting γ to the reciprocal of some power of 2 will ensure that the s_i's are integer-valued as long as $p \ge c^i i!$ for some constant c. Taking logarithms, this requirement becomes $\log p \ge i\log i + O(i)$, which is asymptotically satisfied for certain values of λ in the range $\Theta(\log p/\log\log p)$.

The term *block* will be used to refer to a set of processors that is a block at level i for some i. Note that every block has a unique level. Aside from guaranteeing that the initial distribution of the keys contains a tree of blocks as described above, the adversary gives away additional information at time 0. First, the algorithm is given the ids of the processors that make up each block. Since the blocks at level i are pairwise disjoint concentrated blocks, the ordering of the keys induces a total order over the set of blocks at level i. This total order over the blocks at level i is also given to the algorithm at time 0, $0 \le i < \lambda$.

Like blocks, processors and keys are assigned unique level numbers. The *level of a processor* is i if and only if the highest level block that it is contained in is at level i. The *level of a key* is given by the level of the processor that it is initially contained in. Note that a processor or key at level i is contained in exactly one block at level j, $0 \le j \le i$.

The adversary guarantees that the level of the median key will be as high as possible, namely $\lambda - 1$. Furthermore, the adversary will operate in such a way that the algorithm cannot have determined which block at level $i+1$ contains the median key until the beginning of stage i, $0 \le i < \lambda - 1$. Now recall that $\lambda = \Theta(\log p/\log\log p)$, and note that the ith stage does not begin before time $\lfloor \frac{1}{2}(n/p)\log(i+1)\rfloor - i$. Hence, an adversary argument with the characteristics described above will provide a $\frac{1-o(1)}{2}(n/p)\log\log p - O(\log p/\log\log p)$ time lower bound for selection.

At any given time during the computation, the algorithm will know the outcomes of some subset of the $\binom{n}{2}$ possible key comparisons. Some of this information will have been obtained by making actual comparisons, and some of it will have been given away by the adversary. In any event, the distribution of this information amongst the processors is not an issue because the restricted lower bound model allows an unbounded amount of non-key data to be transferred during each time step. In other words, it may be assumed that at all times, every processor is aware of all of the comparison information that has been gathered thus far.

Definition 4.2 If the outcomes of all $n-1$ comparisons involving a particular key are known to the algorithm, that key is said to be *dead*. Keys that are not dead are *live*.

As mentioned earlier, the adversary will give away certain information at the beginning of the ith stage, $0 \le i < \lambda$. After this information has been given away, and before the ith stage begins, the construction of the adversary will guarantee that the following invariants hold:

1. Let D_i and $L_i = S \setminus D_i$ denote the sets of dead and live keys, respectively, and note that the ranks in S of the keys in D_i have all been determined. There are integers $j < \lfloor n/2 \rfloor$ and $k > \lfloor n/2 \rfloor$ such that the set of ranks in S of the keys in D_i is exactly $\{0,\ldots,j-1\} \cup \{k,\ldots,n-1\}$. Thus, the set of ranks in S of the keys in L_i is $\{j,\ldots,k-1\}$.

2. The set L_i is a subset of a single block at level i. Thus, every key in L_i is of level j for some $j \geq i$.
3. The key being sought, namely the median of S, is also the median of L_i.
4. Let A_i denote the set of all level i keys in L_i. All pairs of keys in A_i are incomparable. Furthermore, $|A_i| \geq (1-\gamma)(n/p)s_i/(i+1)$.
5. Let $B_i = L_i \setminus A_i$ denote the set of all keys in L_i with level strictly greater than i. The set B_i is exactly the set of keys in some block at level $i+1$.
6. Every key in A_i is incomparable to every key in B_i.

It will now be proven by induction on i that Invariants 1 to 6 are all satisfied at the beginning of the ith stage, $0 \leq i < \lambda$. This claim is easy to check for stage 0, with L_i being S, A_i being the set of $(1-\gamma)n$ level 0 keys, and $B_i = S \setminus A_i$ being the set of keys in the lone block at level 1 (note that $b_0 = 2^{\lceil \log 1 \rceil} = 1$). The induction hypothesis is that the invariants hold at the beginning of the ith stage, $0 \leq i < \lambda - 1$. It remains to be proven that the invariants hold at the beginning of stage $i+1$. In order to do so, the procedure by which the adversary resolves comparison queries made by the algorithm up to t_{i+1} must first be defined.

Consider a single comparison made by the algorithm, between keys x and y of levels j and k, respectively. Without loss of generality, assume that $j \leq k$. If either x or y is a dead key then the outcome of the comparison is forced, and the adversary responds accordingly. Similarly, if y does not belong to the same block at level j as x, the response is forced. Otherwise, y belongs to the same block at level j as x, and the two cases $i < j$ and $i = j$ will be considered separately.

If $i = j$ then x belongs to A_i and y belongs to either A_i or B_i. The adversary alternately resolves queries of this type by saying that x is smaller than (larger than), not only y, but the entire set of live tokens in $A_i \cup B_i$. Let A_i^L denote the set of keys in A_i that are known to the algorithm to be less than B_i, and let $A_i^H = A_i \setminus A_i^L$ denote the set of keys that are greater then B_i. Note that the median of S must reside in the set B_i since the adversary has guaranteed that it will be a level $\lambda - 1$ key. Also, the median of S is the median of L_i by Invariant 3 (induction hypothesis). Hence, the partition of A_i into A_i^H and A_i^L must be determined by the algorithm in order to compute the median of L_i. Note that the response made by the adversary kills off the key x (i.e., it becomes a dead key), allows the algorithm to determine exactly one new member of either A_i^L or A_i^H, and preserves the property that the remaining live keys in A_i are all incomparable to one another and to B_i. Furthermore, the algorithm's knowledge of A_i^L and A_i^H grows at an equal rate.

The set of keys A_i are contained in a set of at most $(1-\gamma)s_i$ processors, so Definition 4.1, Lemma 4.1 and Invariant 4 (induction hypothesis) imply that only a fraction $\frac{1}{2 \ln 2}$ of the keys in A_i could have taken part in a comparison during the ith stage. Hence, at the end of stage i, the algorithm could at best have determined that $\frac{1}{4 \ln 2}|A_i|$ of the keys in A_i belong to A_i^L, and that a similar number belong to A_i^H. At least $\left(1 - \frac{1}{2 \ln 2}\right)|A_i|$ of the keys in A_i are still incomparable to one another and to B_i. Using the inequality of Invariant 4, and the fact that $|B_i| = (n/p)s_{i+1} = \gamma(n/p)s_i/b_i$, one finds that the ratio $|A_i|/|B_i|$ must be at least $(1-\gamma)/\gamma$. The preceding discussion all but completes the proof of the following lemma.

Lemma 4.3 For sufficiently small choices of γ, any of the level $\lambda - 1$ keys in B_i could still be the median of L_i at the end of stage i.

Proof: As argued above, the median of L_i must be a level $\lambda - 1$ key in B_i. It is sufficient to prove that the size of the set of keys known to reside in A_i^L (A_i^R) plus the size of the set B_i is less than $|L_i|/2$. At the end of the ith stage, these sizes are at most $\frac{1-\gamma}{4 \ln 2}|L_i|$ and $\gamma|L_i|$, respectively. Thus, the lemma holds for $\gamma < \frac{2 \ln 2 - 1}{4 \ln 2 - 1} \approx 0.218$. As mentioned earlier, it is convenient to set γ to the reciprocal of a power of 2; $\frac{1}{8}$ is an appropriate choice. □

The formulation of the adversary will now be completed. In order to determine how to resolve a comparison query in the remaining case, $i < j$, the adversary consults a *comparison tree* that it has been maintaining for the block at level j containing x. The adversary maintains such a comparison tree for every block. A comparison tree of the same sort was used by Borodin et al. [1] to obtain an easy (though not their strongest) sequential tradeoff between preprocessing time and search time in a partial order. A comparison tree is a binary tree with *tokens* placed at certain nodes. The comparison tree for a block B at level j contains $(1-\gamma)(n/p)s_j$ tokens corresponding to the keys of level j in B, and b_j tokens corresponding to the blocks of level $j+1$ in B. When it is important to distinguish these two types of tokens, they will be referred to as *key tokens* and *block tokens*, respectively. At time 0, the key tokens are all placed at the root and the block tokens are placed, one per node, on the b_j nodes at depth $\log b_j = \lceil \log(j+1) \rceil$. Note that every key corresponds to a key token in the comparison tree of exactly one block.

To resolve the comparison query between keys x and y in the case $i < j$, the adversary locates the key token x' corresponding to x in the comparison tree T associated with the block at level j containing x. The adversary also locates the token y' corresponding to y in the same comparison tree. If $j = k$, this will be a key token; otherwise, it will be the block token corresponding to the unique block of level $j+1$ that contains y. Having located tokens x' and y' in the comparison tree T, the adversary resolves the comparison between keys x and y as follows. Let x' and y' reside in nodes u and v of the tree T, respectively, and let w be the least common ancestor of u and v. If w is not equal to either u or v, then one of u and v must be located in the left subtree of w and the other must be located in the right subtree of w. In this case, the adversary resolves the comparison by stating that the key corresponding to the token in the left subtree is smaller. No tokens are moved. Consider the following pair of lemmas, which summarize the main properties of a comparison tree.

Lemma 4.4 All key tokens lying on a single downward path from the root correspond to incomparable keys. ∎

Lemma 4.5 If some downward path from the root contains both a key token x' and a block token y', then the key corresponding to x' is incomparable to every key in the block corresponding to y'. ∎

Thus, if w is equal to either u or v, the adversary must commit itself to one or more new inequalities. If w is equal to both u and v, then x (say) is said to be the smaller key, x' is moved to the left child of w, and y' is moved to the right child of w. If w is equal to u but not v, then if v is located in the left subtree of w, x is said to be larger, and x' is moved to the right child of w. The remaining cases are treated similarly. Now consider the following lemma.

Lemma 4.6 The adversary resolves comparison queries in a consistent manner. ∎

The inductive step of the proof can now be addressed. It remains to be shown that at the end of stage i, the adversary can determine subsets A_{i+1}, B_{i+1} and $L_{i+1} = A_{i+1} \cup B_{i+1}$ of the set B_i, and kill off all remaining live keys in $S \setminus L_{i+1}$, in such a way that Invariants 1 to 6 are once again satisfied. By Invariant 5 (induction hypothesis), the set B_i corresponds to some block B at level $i+1$. To define the sets needed for stage $i+1$ the adversary goes to the comparison tree T associated with block B. The number of key tokens in this tree is equal to $(1-\gamma)(n/p)s_{i+1}$, and these tokens reside in a set of $(1-\gamma)s_{i+1}$ processors. Let Λ denote the average depth of the key tokens in T. Observe that every comparison made by the algorithm increments the depth of at most two tokens. Hence, Definition 4.1 and Lemma 4.2 imply that $\Lambda \le \frac{1}{2}\log(i+2)$ at time t_{i+1}. Let P denote the set of b_{i+1} paths in T from the root to the initial position of each of the b_{i+1} block tokens in T, that is, all paths of depth $\lceil \log(i+2) \rceil$. Let a_j denote the number of key tokens at depth j in T at time t_{i+1}. By a simple averaging argument, some path in P must contain at least

$$\sum_{0 \le j \le \lceil \log(i+2) \rceil} a_j 2^{-j}$$

key tokens. This sum is clearly minimized by moving the tokens downward in a uniform fashion. Hence, the bound on Λ implies that some path q in P contains at least a fraction $2^{-\log(i+2)} = \frac{1}{i+2}$ of the key tokens. The adversary assigns A_{i+1} to the set of key tokens on the path q, and assigns B_{i+1} to the block at level $i+2$ corresponding to q. Finally, the adversary kills off all remaining live keys in $S \setminus L_{i+1}$ in such a way that the median of S is also the median of L_{i+1}; Lemma 4.3 implies that this is always possible. All of the keys in A_{i+1} are incomparable by Lemma 4.4. Every key in A_{i+1} is incomparable to every key in B_{i+1} by Lemma 4.5. Thus, the construction of the adversary ensures that Invariants 1 to 6 all hold at the beginning of stage $i+1$, $0 \le i < \lambda - 1$, and the proof by induction is complete. The following theorem summarizes the main result of this section.

Theorem 4.1 Any selection algorithm for the complete network running under the restricted lower bound model requires $\frac{1-o(1)}{2}(n/p)\log\log p - O(\log p/\log\log p)$ time. ∎

5 The Network Lower Bound

The purpose of this section is to prove an $\Omega((n/p)\log\log p + \log p)$ time lower bound for selection on certain realistic networks. The lower bound will apply under the powerful lower bound model defined in Section 2, and will be obtained by making suitable modifications to the proof of Theorem 4.1. Consider the following definition.

Definition 5.1 Let $\mathcal{N}(\alpha, \beta)$ denote the class of all network families $\mathcal{F}$ for which, given any p processor network in $\mathcal{F}$, it is possible to construct all of the blocks at levels less than β (as defined in Section 4) in such a way that every block has expansion at most α, where α and β may depend on p.

A careful examination of the proof of Theorem 4.1 reveals that there are only two points at which the definition of the restricted lower bound model is invoked; the rest of the proof applies to the full-fledged lower bound model. The first use of Definition 4.1 is in the proof of Lemma 4.3, and the second use leads to the existence of a suitable set A_{i+1} in the induction step. In both cases, Definition 4.1 provides an upper bound of $(1-\gamma)s_i$ on the number of comparisons per time step involving the set of level i keys of a particular block at level i.

Theorem 5.1 Let $\mathcal{F}$ be a network family belonging to the class $\mathcal{N}(o(1/\log\log p), \beta)$. Then any selection algorithm for $\mathcal{F}$ has a running time of at least $\frac{1}{2}(n/p)\log\beta - \beta$ under the lower bound model.

Proof: In the following argument, let X denote a generic block at level i, $0 \le i < \beta - 1$. Let Y denote the union of the blocks at level $i+1$ in X, and let $Z = X \setminus Y$. Let Y' and Z' denote the sets of keys initially residing in Y and Z, respectively. By the remarks preceding the statement of the theorem, it is sufficient to prove that the adversary construction of Section 4 can be revised in such a way that the two applications of Definition 4.1 can be avoided.

The construction of the adversary will be augmented in the following manner. Let T denote the comparison tree associated with block X. At any given time step, each of the $|Z'|$ key tokens in T is said to be either *bad* or *good*. A good key token is one for which no copy of the corresponding key has ever left the set of processors Z. Thus, all key tokens are initially good, and bad key tokens never become good again. How many good key tokens in tree T can become bad in a single time step? There are only two ways for a good key token to become bad. One way is for a copy of the corresponding key to be transmitted to a processor outside of X. Since block X has expansion $o(1/\log\log p)$, the number of good key tokens that can become bad in this manner is $o(|X|/\log\log p)$ per time step. The other way for a key token to become bad is for a copy of the corresponding key to be transmitted to a processor in Y. Since Y is the union of a number of disjoint sets with expansion $o(1/\log\log p)$, the number these events is $o(|Y|/\log\log p)$ per time step. By construction, $|Z|$ is a constant fraction of $|X|$, so the total number of key tokens that can become bad in a single time step is $o(|Z|/\log\log p)$. Now the lower bound argument only runs for $O((n/p)\log\log p)$ time steps, during which time the number of bad tokens that can be generated is $o(|Z|n/p) = o(|Z'|)$. In other words, the number of good key tokens in T is $(1-o(1))|Z'|$ at all times in the range of interest.

The final modification to the adversary construction of Section 4 is as follows. In the induction step, the adversary now extracts the set A_{i+1} from the set of good key tokens only. In a single time step, at most $|Z|$ comparisons can be made involving keys in the subset of Z' corresponding to the good key tokens in T, since processors outside of Z do not have copies of any of these keys. Thus, the argument of Section 4 goes through with the inequality of Invariant 4 weakened by a factor of $1-o(1)$, the fraction of all key tokens that are guaranteed to be good. Now consider the proof of Lemma 4.3. At the beginning of the ith stage, all of the key tokens corresponding to A_i are good, and they reside in a set of $|Z|$ processors. Using an expansion argument as above, Lemma 4.1 implies that at most a $o(1)$ fraction of the key tokens corresponding to A_i can become bad during the ith stage. This minor effect is of no consequence since γ has already been set to a value that is bounded away from its maximum acceptable value. A similar comment applies to the effect of the weakened inequality in Invariant 4. $\Box$

Corollary 5.1.1 Let $\mathcal{F}$ be an $\Omega(\log p)$ diameter network family belonging to the class $\mathcal{N}(o(1/\log\log p), \Theta(\beta))$. Then any selection algorithm for $\mathcal{F}$ has a running time of at least $\frac{1}{2}(n/p)\log\beta + \Omega(\log p)$ under the lower bound model.

Proof: This bound follows immediately from Theorem 5.1 and the additional observation that any selection algorithm for a network of diameter d has a running time of at least $\lfloor d/2 \rfloor$ under the lower bound model. $\Box$

5.1 The Hypercube

Throughout this section, let ϵ denote an arbitrarily small positive constant. A decomposition of the hypercube will now be defined that proves the hypercube network family belongs to $\mathcal{N}(o(1/\log\log p), \Theta(\log^{1/3-\epsilon} p))$. Given a hypercube with $p = 2^d$ processors, let $q = \lfloor d^\delta \rfloor$ or $\lfloor d^\delta \rfloor - 1$, whichever is odd. The exponent δ is a parameter between 0 and 1 to be determined later. Let $r = \lfloor d/q \rfloor$, and divide the first qr bits of each processor id into r *fields* of q contiguous bits. The ith field determines the ith bit of an r-bit *condensed id* according to the following rule, $0 \le i < r$. If the majority of the q bits in the ith field are 0, then the ith bit of the condensed id is 0; otherwise, it is a 1. Note that since q is odd there will always be a strict majority of either 0's or 1's. By symmetry, 2^{d-l} processors will belong to any *condensed subcube* of dimension $r-l$ obtained by fixing the values of l bits in the condensed id, $0 \le l < r$.

Lemma 5.1 The expansion of a condensed subcube of dimension l is $O(lq^{-1/2})$. $\Box$

Since $l \le r$, $q = \Theta(d^\delta)$ and $r = \Theta(d^{1-\delta})$, Lemma 5.1 implies that the expansion of every condensed subcube is $O(rq^{-1/2}) = O(d^{1-3\delta/2})$. If δ is chosen to be any constant strictly between $2/3$ and 1, then the expansion of every condensed subcube will be $O(\log^{-\epsilon} p) = o(1/\log\log p)$. Furthermore, it should be clear that the condensed subcube structure can be used to construct the tree of blocks required by the lower bound argument of Section 4, at least to a certain depth, since the size of every block is a power of 2. All that remains is to determine the maximum possible value of β as a function of p. The relevant inequality is $c^\beta \beta! \le 2^r$ for some constant c, which is satisfied for $\beta = \Theta(d^{1-\delta-\epsilon/2})$. Setting $\delta = 2/3 + \epsilon/2$ gives $\beta = d^{1/3-\epsilon}$, as claimed above. Slightly finer calculations allow this ϵ to be replaced by $o(1)$. Hence, Corollary 5.1.1 implies the following result.

Theorem 5.2 Any selection algorithm for the hypercube has a running time of at least $\frac{1-o(1)}{6}(n/p)\log\log p + \Omega(\log p)$ under the lower bound model.

5.2 Other Networks

The preceding decomposition also works for the shuffle-exchange, since it is easy to prove that Lemma 5.1 remains valid. Hence, the lower bound of Theorem 5.2 applies to the shuffle-exchange. Similar comments apply for the butterfly network.

Low flux networks such as the tree and multi-dimensional mesh can be easily decomposed into a large number of equal-sized components with very poor expansion. In such cases, β can be increased so that the lower bound of Theorem 5.2 applies with an improved multiplicative constant of $\frac{1-o(1)}{2}$.

References

[1] A. Borodin, L. J. Guibas, N. A. Lynch, and A. C. Yao. Efficient searching using partial ordering. *IPL*, 12:71–75, 1981.

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

[3] R. M. Karp, R. Motwani, and P. Raghavan. Deferred data structuring. *SIAM J. Comput.*, 17:883–902, 1988.

[4] C. G. Plaxton. Load balancing, selection, and sorting on the hypercube. In *Proceedings of the 1st Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 64–73, June 1989.

[5] C. G. Plaxton. On the network complexity of selection. Technical Report STAN–CS–89–1276, Stanford University, Department of Computer Science, August 1989.