Separation and Lower Bounds for ROM and Nondeterministic Models of Parallel Computation*

MING LI AND YAACOV YESHA

Department of Computer and Information Science, Ohio State University, Columbus, Ohio 43210

For the first time we separate the widely used shared-memory models of parallel computation, COMMON(m), ARBITRARY(m), and PRIORITY(m), for small m (the communication width), without the restrictive assumption that each processor has access only to one input. Rather we follow S. A. Cook and C. Dwork (1982, in “Proceedings, 14th ACM Symp. on Theory of Computing, 1982,” pp. 231-233) and U. Vishkin and A. Wigderson (1985, SIAM J. Comput. 14, No. 2, 303-314) who assume that the inputs are given in a read only memory (ROM). The previous separation results of F. Fich, P. Ragde, and A. Wigderson (1984, in “Proceedings, 3rd ACM Symp. on Principles of Distrib. Comput., 1984,” pp. 179-184) and F. Fich, F. Meyer, P. Ragde, and A. Wigderson (1985, in “Proceedings, 17th ACM Symp. on Theory of Comput., 1985,” pp. 48-58) (Familiarity with these papers would be helpful.) do assume that each processor knows only one input. We introduce a new technique to separate COMMON(m) with ROM and ARBITRARY(1) (even without ROM). We also generalize a technique of (Fich, Ragde, and Wigderson, op cit.) to separate ARBITRARY(m) with ROM and PRIORITY(1) (even without ROM). These two separation results are tight. Fich, Ragde, and Wigderson (ibid.) have previously obtained tight separation results for the corresponding models without ROM. Also we settle a conjecture of Vishkin and Wigderson about the parallel time (depth) needed to compute PARITY nondeterministically on a PRIORITY(1). They conjecture that the lower bound is \( \Omega(\sqrt{n}) \) which is the same as the deterministic tight lower bound. We prove a nondeterministic upper bound of \( O(n^{1/3}) \). We also prove a tight lower bound of \( \Omega(\sqrt{n}) \) for PARITY on nondeterministic PRIORITY(1) without ROM and a lower bound of \( \Omega(\log \log n) \) for PARITY on nondeterministic PRIORITY(1) with ROM.

1. INTRODUCTION

In this paper, we prove new separation results between shared memory models of parallel computation. The models we consider are variants of the parallel RAM (PRAM), which differ from each other in the way they restrict simultaneous writing into the same shared memory address. As

* This work was supported in part by the National Science Foundation under Grant DCR-8606366.

0890-5401/87 $3.00
Copyright © 1987 by Academic Press, Inc.
All rights of reproduction in any form reserved.
pointed out in (Fich et al. 1984) there are algorithms on all of these models in the literature. Hence it is important to know whether these models differ in power.

A PRAM consists of a set of processors $P(i), i = 1, 2, \ldots$ which are random access machines (RAMs), a collection of shared memory cells $C(i), i = 1, 2, \ldots$ and $n$ read only input cells (ROM), $X(1), X(2), \ldots, X(n)$. Each step of the computation consists of four phases:

1. each processor reads from some ROM cell;
2. each processor reads from some shared memory cell;
3. each processor performs a computation;
4. each processor may attempt writing into some shared memory cell.

Whenever more than one processor simultaneously attempt writing into the same shared memory cell there is a write conflict. The various variants of the PRAM differ in the way they handle write conflicts. These variants are:

1. CREW (concurrent read exclusive write). In this model it can never happen that at the same step more than one processor attempt writing into the same shared memory cell.
2. COMMON. In this model it is required that all the processors which at the same step attempt writing into the same shared memory cell write the same value.
3. ARBITRARY. In this model, among all the processors which at the same step attempt writing into the same shared memory cell, an arbitrary one succeeds.
4. PRIORITY. In this model, among all processors which at the same step attempt writing into the same shared memory cell, the one with minimum index succeeds.

If the PRAM computes a function, the depth of the PRAM is the number of parallel steps used to compute the function.

All the above models are widely used for implementing parallel algorithms. For example, (Hirschberg et al., 1979) use CREW, Shiloach and Vishkin, 1981; Galil, 1984) use COMMON, (Shiloach and Vishkin, 1982) use ARBITRARY, and (Awerbuch and Shiloach, 1983) use ARBITRARY and PRIORITY.

As mentioned in (Vishkin and Wigderson 1985), the case in which the number of shared memory cells is 1 is of particular importance. (Vishkin and Wigderson, 1985) point out that the “Ethernet” can be considered as a PRAM with only one shared memory cell. They also mention that (Gottlieb et al., 1983; Kuck, 1977; Vishkin, 1980) imply that minimizing
the size of shared memory may amount to hardware feasibility of the parallel machine.

Having this in mind, we continue, in the line of research of (Cook and Dwork, 1982; Vishkin and Wigderson, 1985; Fich et al., 1984; Fich et al., 1985), to prove lower bounds on depth and separation results for models with one shared memory cell or a constant number of shared memory cells. For $X$ any of the above models, let $X(m)$ be the model with $m$ shared memory cells. While (Cook and Dwork, 1982; Vishkin and Wigderson, 1985) only consider models with ROM, (Fich et al., 1984; Fich et al., 1985) assume that the inputs are not given in a ROM. Rather, they assume that processor $P(i)$ can only read input $X(i)$ for $i = 1, 2, ..., n$. For models without ROM, (Fich et al., 1984) give tight separation results between $\text{COMMON}(m)$ without ROM and $\text{ARBITRARY}(1)$ without ROM, and between $\text{ARBITRARY}(m)$ without ROM and $\text{PRIORITY}(1)$ without ROM. (Fich et al., 1985) give a tight separation between $\text{COMMON}(m)$ without ROM and $\text{PRIORITY}(m)$ without ROM for $m \leq n^\varepsilon$, $\varepsilon < 1$. However, the situation for models with ROM turns out to be much more complicated. (Note that in the above previous results $m = o(n)$, hence the shared memory is not large enough to save even a constant fraction of the input.) It can be easily shown that the models with ROM are more powerful than models without ROM, if the amount of shared memory is $o(n)$. Hence we introduce a new technique which for the first time enables us to separate between $\text{COMMON}(m)$ with ROM and $\text{ARBITRARY}(1)$ (even without ROM), and generalize a technique of (Fich et al., 1984) to separate between $\text{ARBITRARY}(m)$ with ROM and $\text{PRIORITY}(1)$ (even without ROM). Both separations are tight. We also note that, for instance, some functions which require $\log n$ depth on $\text{COMMON}(1)$ without ROM can be computed in constant depth on $\text{COMMON}(1)$ with ROM. As pointed out in (Vishkin and Wigderson, 1985), the introduction of ROM in models with $o(n)$ shared memory is similar to a read only input tape in off-line $o(n)$ space bounded Turing machines. A major motivation to study PRAMs with ROM is to distinguish between communication and information sharing.

We then proceed to investigate nondeterministic PRAMs. (Vishkin and Wigderson, 1985) ask about lower bounds for nondeterministic PRAMs and conjecture that the $\Omega(\sqrt{n})$ lower bound on the depth of a $\text{PRIORITY}(1)$ with ROM which computes the PARITY function holds also for nondeterministic $\text{PRIORITY}(1)$ with ROM. We show an $O(n^{1/3})$ upper bound for PARITY on this model. We then prove a tight lower bound on the depth for computing PARITY on nondeterministic $\text{PRIORITY}(1)$ without ROM, and also prove a lower bound on depth for PARITY on $\text{PRIORITY}(1)$ with ROM. Again, the case with ROM is much more difficult.
SEPARATION AND LOWER BOUNDS

TABLE I
Separation between Deterministic Model with ROM

<table>
<thead>
<tr>
<th>A function is computable in constant depth on</th>
<th>Requires</th>
<th>Results in Section</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \Omega \left( \frac{\log n}{\log(m + 1)} \right) ) on</td>
<td></td>
</tr>
<tr>
<td>ARBITRARY(1)</td>
<td>COMMON(m) with ROM and ( n^a ) processors ((a &lt; 2))</td>
<td>3</td>
</tr>
<tr>
<td>PRIORITY(1)</td>
<td>ARBITRARY(m) with ROM</td>
<td>4</td>
</tr>
</tbody>
</table>

We summarize our results in Tables I and II.

2. PRELIMINARIES AND DEFINITIONS

We start by defining deterministic models of synchronous parallel computers with shared memory. (See also Cook and Dwork, 1982; Vishkin and Wigderson, 1985; Fich et al., 1985.) The models consist of a collection of processors which can read from and write into shared memory. The only communication among the processors is through the shared memory. The various models differ in the way they resolve write conflicts in which more than one processor attempt writing into the same shared memory cell simultaneously. By \( N \) we denote the set of positive integers \( \{1, 2, \ldots\} \).

**DEFINITION 2.1.** A PRIORITY PRAM (PRIORITY, in short) consists of a set \( P = \{ P(1), P(2), \ldots \} \) of processors, a number \( n \) of inputs, \( n \) read only input cells (ROM) \( X(1), X(2), \ldots, X(n) \), a set \( C(1), C(2), \ldots \) of shared memory cells, an alphabet \( \Sigma \) (usually infinite), and a depth \( T \). Each processor \( P(i) \) has a set of states \( Q_i \) and functions:

**TABLE II**
Lower Bounds for Parity on NPRIORITY(1)

<table>
<thead>
<tr>
<th>Lower bound on</th>
<th>Is</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPRIORITY(1) (no ROM)</td>
<td>( \Omega(\sqrt{n}) ) (tight)</td>
<td>5</td>
</tr>
<tr>
<td>NPRIORITY(1) with ROM</td>
<td>( \Omega(\log \log n) )</td>
<td>5</td>
</tr>
</tbody>
</table>

*Note.* A prefix \( N \) denotes nondeterministic model. The above results generalize to any constant number of shared-memory cells.
The model operates in steps. Step $t$ for $t = 1, 2, \ldots, T$ occurs in time period $t$. At time $t = 0$ the input cell $X(i)$ ($i = 1, 2, \ldots, n$) contains the $i$th input $x_i$, all the shared memory cells contain a distinguished symbol $a \in \Sigma$, and every processor $P_i$ is in a distinguished state $q_{i,0}$ in $Q_i$, which is called an initial state.

At step $t$ each processor $P(i)$ is in state $q_{i,t}$ in $Q_i$ and each shared memory cell $C(j)$ contains a symbol $S_{i,t}$ in $\Sigma$. The $q_{i,t}, S_{i,t}$ for $t = 1, 2, \ldots, T$ are determined from the $q_{i,t-1}, S_{i,t-1}$ as follows:

1. For all $i$, $P(i)$ reads from input cell $X(XREAD_i(q_{i,t-1}))$.
2. For all $i$, $P(i)$ reads from shared memory cell $C(CREAD_i(q_{i,t-1}))$.
3. For each $j \in N$ determine the set $WR_{j,t}$ of indices of the processors which attempt writing into shared memory cell $C(j)$ at time $t$. Formally, let: $WR_{j,t} = \{i \mid WRITE_i(q_{i,t})$ is of the form $(j, b)$ for some $b \in \Sigma\}$. Then for each $C(j)$ such that $WR_{j,t}$ is nonempty find the minimum index in $WR_{j,t}$. This is going to be the index of the processor which will actually write into $C(j)$ at step $t$. Formally for all $j$ such that $WR_{j,t}$ is nonempty, let $WR_{j,t} = \{i \mid WRITE_i(q_{i,t})$ is of the form $(j, b)$ for some $b \in \Sigma\}$. Then for each $C(j)$ such that $WR_{j,t}$ is nonempty find the minimum index in $WR_{j,t}$. This is going to be the index of the processor which will actually write into $C(j)$ at step $t$. Formally for all $j$ such that $WR_{j,t}$ is nonempty, let $WR_{j,t} = \{i \mid WRITE_i(q_{i,t})$ is of the form $(j, b)$ for some $b \in \Sigma\}$. Then for each $C(j)$ such that $WR_{j,t}$ is nonempty find the minimum index in $WR_{j,t}$. This is going to be the index of the processor which will actually write into $C(j)$ at step $t$. Formally for all $j$ such that $WR_{j,t}$ is nonempty, let $WR_{j,t} = \{i \mid WRITE_i(q_{i,t})$ is of the form $(j, b)$ for some $b \in \Sigma\}$. Then for each $C(j)$ such that $WR_{j,t}$ is nonempty find the minimum index in $WR_{j,t}$. This is going to be the index of the processor which will actually write into $C(j)$ at step $t$. Formally for all $j$ such that $WR_{j,t}$ is nonempty, let $WR_{j,t} = \{i \mid WRITE_i(q_{i,t})$ is of the form $(j, b)$ for some $b \in \Sigma\}$. Then for each $C(j)$ such that $WR_{j,t}$ is nonempty find the minimum index in $WR_{j,t}$. This is going to be the index of the processor which will actually write into $C(j)$ at step $t$. Formally for all $j$ such that $WR_{j,t}$ is nonempty, let $WR_{j,t} = \{i \mid WRITE_i(q_{i,t})$ is of the form $(j, b)$ for some $b \in \Sigma\}$. Then for each $C(j)$ such that $WR_{j,t}$ is nonempty find the minimum index in $WR_{j,t}$. This is going to be the index of the processor which will actually write into $C(j)$ at step $t$. Formally for all $j$ such that $WR_{j,t}$ is nonempty, let $WR_{j,t} = \{i \mid WRITE_i(q_{i,t})$ is of the form $(j, b)$ for some $b \in \Sigma\}$. Then for each $C(j)$ such that $WR_{j,t}$ is nonempty find the minimum index in $WR_{j,t}$. This is going to be the index of the processor which will actually write into $C(j)$ at step $t$. Formally for all $j$ such that $WR_{j,t}$ is nonempty, let $WR_{j,t} = \{i \mid WRITE_i(q_{i,t})$ is of the form $(j, b)$ for some $b \in \Sigma\}$. Then for each $C(j)$ such that $WR_{j,t}$ is nonempty find the minimum index in $WR_{j,t}$. This is going to be the index of the processor which will actually write into $C(j)$ at step $t$. Formally for all $j$ such that $WR_{j,t}$ is nonempty, let $WR_{j,t} = \{i \mid WRITE_i(q_{i,t})$ is of the form $(j, b)$ for some $b \in \Sigma\}$. Then for each $C(j)$ such that $WR_{j,t}$ is nonempty find the minimum index in $WR_{j,t}$. This is going to be the index of the processor which will actually write into $C(j)$ at step $t$. Formally for all $j$ such that $WR_{j,t}$ is nonempty, let $WR_{j,t} = \{i \mid WRITE_i(q_{i,t})$ is of the form $(j, b)$ for some $b \in \Sigma\}$. Then for each $C(j)$ such that $WR_{j,t}$ is nonempty find the minimum index in $WR_{j,t}$. This is going to be the index of the processor which will actually write into $C(j)$ at step $t$. Formally for all $j$ such that $WR_{j,t}$ is nonempty, let $WR_{j,t} = \{i \mid WRITE_i(q_{i,t})$ is of the form $(j, b)$ for some $b \in \Sigma\}$. Then for each $C(j)$ such that $WR_{j,t}$ is nonempty find the minimum index in $WR_{j,t}$. This is going to be the index of the processor which will actually write into $C(j)$ at step $t$. Formally for all $j$ such that $WR_{j,t}$ is nonempty, let $WR_{j,t} = \{i \mid WRITE_i(q_{i,t})$ is of the form $(j, b)$ for some $b \in \Sigma\}$. Then for each $C(j)$ such that $WR_{j,t}$ is nonempty find the minimum index in $WR_{j,t}$.

4. For all $i$, processor $P(i)$ changes state according to the values $P(i)$ has read from the ROM and from the shared memory. Formally $Q_{i,t} = NEXTSTATE_i(q_{i,t-1}, x_u, S_{v,t-1})$, where $u = XREAD_i(q_{i,t-1}), v = CREAD_i(q_{i,t-1})$.

In the above definition, whenever $i$ is in $WR_{j,t}$ we say that processor $P(i)$ attempts writing into shared-memory cell $C(j)$ at time $t$. By (3) above, for each $t$ and $j$, among all the processors which attempt writing into $C(j)$ at step $t$, the one with the minimum index (which we call $W_{j,t}$) succeeds. There are other models which differ in the way the processor which succeeds in writing is determined:
SEPARATION AND LOWER BOUNDS

ARBITRARY. An arbitrary processor among those processors which attempt writing into the same memory cell in the same step is selected.

COMMON. This model is restricted in such a way that all the processors which at the same step attempt writing into the same shared-memory cell, attempt to write the same value and this value will be written into that memory cell.

CREW (concurrent read exclusive write). This model is restricted in such a way that for all \( t \) and \( j \) at most one processor attempts writing into \( C(j) \) at step \( t \). If there is one such processor it will write.

For \( t = 0, 1, \ldots \) the memory map is the vector \( H_t = \{ S_{j,t} \mid j = 1, 2, \ldots \} \), where \( S_{j,t} \) is the contents of \( C(j) \) at time \( t \). The history through step \( T \) of the computation of the parallel model \( M \) on input \( (x_1, x_2, \ldots, x_n) \) is the vector \( H_0, H_1, \ldots, H_T \) which results by letting \( x_i \) be the contents of \( X(i) \) (\( 1 \leq i \leq n \)) and determining \( q_{i,t}, S_{j,t} \) (\( t = 0, 1, \ldots, T \)) as above.

Less powerful variants of the above models were studied in (Fich et al., 1985; 1984). For \( M \) any of the above models, \( M \) without ROM will denote the corresponding model for which the number of processors is equal to the number \( n \) of inputs, and such that rather than reading the input from a ROM, the cell \( X(i) \) belongs to \( P(i) \) which knows its contents \( x_i \). But \( P(i) \) cannot read from \( X(j) \) for \( j \neq i \).

For proving upper bounds, we sometimes only describe an algorithm for the model without ROM, since the corresponding model with ROM can simulate a depth \( T \) model without ROM in depth \( T + 1 \), using \( n \) processors and the same amount of memory cells. At the first step simply let \( P(i) \) read \( x_i \) from \( X(i) \).

Let \( M \) be any of the above models, with or without ROM. Let \( D \) be any domain, and let \( f \) be any function defined on \( D^n \). We say that \( M \) computes \( f \) in depth \( T \) if on every input \( \bar{x} \) in \( D^n \), \( M \) will have \( f(\bar{x}) \) written in \( C(1) \) at step \( T \). Formally, \( S_{1,T} = f(\bar{x}) \). It is important to notice that for the ARBITRARY model we require that \( f(\bar{x}) \) will be written in \( C(1) \) at step \( T \), no matter which processors succeeded in writing at each step.

As usual, a language \( L, L \subseteq D^n \) is recognized by \( M \), if \( M \) computes its characteristic function.

Another important consideration is the communication width (width, for short). For \( M \) any of the above models, \( M(m) \) will denote model \( M \) restricted to having only \( m \) shared memory cells \( C(1), C(2), \ldots, C(m) \).

We now define nondeterministic models of parallel computation. Using \( M \) for any of the previously defined models, \( NM \) will denote its nondeterministic variant. The nondeterministic variant of \( M \) is a generalization of \( M \), obtained by fixing a constant \( d \geq 2 \) (called the branching factor of \( M \)) and such that the next state function for each processor is a function whose range is all \( d \)-tuples of states. Formally, using the notation
of Section 2, \( \text{NEWSTATE}_i \) is a function from \( Q_i \times \Sigma \times \Sigma \) into \( Q_i' \). Also \( q_{i,t} \) may be any element of \( \text{NEWSTATE}_i \) \( (q_{i,t-1}, x, S_{i,t-1}) \). If \( \text{NEWSTATE}_i(q_{i,t-1}, x, S_{i,t-1}) = (e_1, e_2, \ldots, e_d) \) and \( q_{i,t} = e_j \) \( (1 \leq j \leq d) \), we say that at step \( t \), \( P(i) \) chose branch number \( j \). We also use the word \textit{computation} to denote a particular action of \( M \) on some input, by having \( P(i) \) choose some branch number at step \( t \), for \( i = 1, 2, \ldots, t \).

We also define a subset \( \Sigma_A \) of the symbols in \( \Sigma \) to be the set of \textit{accepting symbols}. If \( M \) has depth \( T \), a computation of \( M \) is called \textit{accepting} if at step \( T \), \( C(1) \) contains an accepting symbol. \( M \) of depth \( T \) is said to \textit{compute a function} \( f \) if all symbols in \( \Sigma_A \) are of the form \( \langle A, y \rangle \), where \( A \) is a special character, and, for all input \( \tilde{x} \).

(i) there is an accepting computation of \( M \) on \( \tilde{x} \) for which at step \( T \), \( C(1) \) contains \( \langle A, f(\tilde{x}) \rangle \);

(ii) every accepting computation of \( M \) on \( \tilde{x} \) has at step \( T \), \( \langle A, f(\tilde{x}) \rangle \) in \( C(1) \).

Now let \( L \) be any language, say \( L \subseteq D^n \). We say that the nondeterministic model \( M \) \textit{accepts} \( L \) \textit{in depth} \( T \) if on any input in \( L \) there exists at least one accepting computation of \( M \), and for any input not in \( L \) there is no accepting computation.

### 3. A Separation between COMMON\((m)\) with ROM and ARBITRARY\((1)\)

In this section, for the first time we prove a tight separation result between \( \text{COMMON}(m) \) with ROM and ARBITRARY\((1)\). The separation is tight up to a constant factor. Fich \textit{et al.} (1985) previously proved a separation result between \( \text{COMMON}(m) \) \textit{without} ROM and PRIORITY\((m) \) \textit{without} ROM for \( m \leq n^\varepsilon \), \( \varepsilon < 1 \). Their separation relies on the following theorem.

**Theorem 3.1** (Fich \textit{et al.}, 1985). Let \( f \) be a surjective function from \( \Sigma^n \) \( (\Sigma = \{0, 1\}) \) onto \( R \). Then any \( \text{COMMON}(1) \) \textit{without} ROM which computes \( f \) requires depth at least \( \log_3 |R| \).

They consider the function \( \text{INDEX}(x_1, x_2, \ldots, x_n) \) which is defined on \( \Sigma^n \) \( (\Sigma = \{0, 1\}) \) as follows: \( \text{INDEX}(x_1, x_2, \ldots, x_n) = \max \{ j \mid x_i = 0 \text{ for all } 1 \leq i < j \} \). Since \( \text{INDEX} \) is surjective onto \( \{1, 2, \ldots, n + 1\} \), \( \text{INDEX} \) requires at least \( \log_3 n \) steps on \( \text{COMMON}(1) \) \textit{without} ROM, by Theorem 3.1. Since \( \text{INDEX} \) can be computed in constant depth on PRIORITY\((1)\), the separation result of Fich \textit{et al.} (1985) follows. We now show that Theorem 3.1 is not true for \( \text{COMMON}(1) \) with ROM. Consider the
function \( I_1(x_1, x_2, \ldots, x_n) \) defined as follows: if there exists \( i \) (\( 1 \leq i \leq n - 1 \)), such that \( x_i = 1 \) and \( x_{i+1} = 0 \) then \( I_1(x_1, x_2, \ldots, x_n) = n + 1 \). Otherwise, \( I_1(x_1, x_2, \ldots, x_n) = \text{INDEX}(x_1, x_2, \ldots, x_n) \). \( I_1 \) is surjective onto \( \{1, 2, \ldots, n + 1\} \). However, we prove the following

**FACT.** \( I_1 \) can easily be computed in constant depth on a \( \text{COMMON}(1) \) with \( \text{ROM} \), using \( n \) processors.

**Proof.** In parallel, \( P(i) \) reads \( X(i) \) (\( i = 1, 2, \ldots, n \)). Then in parallel, \( P(i) \) reads \( X(i+1) \) (\( i = 1, \ldots, n - 1 \)). Then any processor \( P(i) \) which reads \((X(i), X(i+1)) = (1, 0)\) writes \( n + 1 \) into \( C(1) \). Then all the processors read from \( C(1) \). If they read the value \( n + 1 \) then the computation halts. Otherwise the (unique) processor \( P(i) \) which reads \((X(i), X(i+1)) = (0, 1)\) writes \( i + 1 \) into \( C(1) \), or \( P(1) \) writes 1 if \( X(1) = 1 \).

Fich et al. (1984) previously proved a tight separation between \( \text{COMMON}(m) \) without \( \text{ROM} \) and \( \text{ARBITRARY}(1) \) which does not rely on Theorem 3.1. However, a \( \text{ROM} \) makes the situation much more complicated, and the technique of (Fich et al., 1984) apparently does not generalize to the case of having \( \text{ROM} \). We have to introduce a new technique to prove the lower bound for \( \text{COMMON}(m) \) with \( \text{ROM} \). The technique is based on an adversary argument which is used to prove a lower bound on \( \text{COMMON}(1) \) with \( \text{ROM} \) for the language of threshold-2-function:

\[
L_2 = \{(x_1, x_2, \ldots, x_n) \mid x_i \in \{0, 1\} (1 \leq i \leq n) \text{ and for some } i, j, 1 \leq j < i \leq n, x_i = x_j = 1\}.
\]

It is easy to see that \( L_2 \) can be recognized on \( \text{ARBITRARY}(1) \) with \( n \) processors, even without \( \text{ROM} \), in constant depth. We also note that a \( \text{COMMON}(1) \) with \( \text{ROM} \) and \( \Omega(n^2) \) processors can recognize \( L_2 \) in a constant depth, by reading all pairs \((x_i, x_j)\). If, however, the number of processors \( q \) satisfies \( q \leq n^a, 0 < a < 2 \), we prove a lower bound of

\[
\Omega((2 - a) \log n)
\]
on the depth of a \( \text{COMMON}(1) \) with \( \text{ROM} \) recognizing \( L_2 \). Hence \( L_2 \) cannot be recognized in constant depth on a \( \text{COMMON}(1) \) with \( \text{ROM} \) and at most \( n^a \) processors with \( a < 2 \). Hence we get the separation result.

**Theorem 3.2.** Any \( \text{COMMON}(1) \) with a \( \text{ROM} \) and \( q = n^a \) processors \((0 < a < 2)\) which recognizes \( L_2 \) requires depth \( \Omega((2 - a) \log n) \).

**Proof.** Let \( M \) be a \( \text{COMMON}(1) \) with \( \text{ROM} \) which recognizes \( L_2 \). We will define, for \( t = 0, 1, \ldots \), sets \( S_t, PA_t, \) and \( I_t \). \( S_t, \) and \( PA_t \) represent con-
straints on the input, which are imposed by an adversary at step $t$. $I_t$ is the set of all inputs which satisfy the constraints represented by $S_t$ and $PA_t$. The constraints are chosen by the adversary in such a way that

1. All inputs in $I_t$ have the same history through step $t$.
2. Furthermore, if $t$ is not large enough, then $I_t$ will have to contain an input in $L2$ and an input not in $L2$, which implies that $M$ cannot have depth $t$.

More specifically $PA_t$ is a subset of $\{1, 2, \ldots, n\} \times \{1, 2, \ldots, n\}$, and $S_t$ is a subset of $\{1, 2, \ldots, n\}$ and $I_t$ is always defined by

$$I_t = \{ (x_1, x_2, \ldots, x_n) \mid x_i \in \{0, 1\} \text{ for } i = 1, 2, \ldots, n \text{ and for some } i \in S_t, x_i = 1, \text{ and for all } f \text{ not in } S_t, x_f = 0, \text{ and there is no pair } (j, k) \text{ in } PA_t, \text{ such that } x_j = x_k = 1 \}.$$

We construct $S_t$ and $PA_t$ by induction on the number of steps $t$. For $t = 1, 2, \ldots$ we will always have $S_t \subseteq S_{t-1}$ and $PA_t \subseteq PA_{t-1}$. Hence $I_t \subseteq I_{t-1}$ ($t = 1, 2, \ldots$).

**The Induction Hypothesis.**

1. All inputs in $I_{t-1}$ have the same history through step $t-1$.
2. All pairs $(j, k)$ such that $j \neq k$ and some processor on some input $(x_1, x_2, \ldots, x_n) \in I_{t-1}$ reads during the first $t-1$ steps from ROM positions $X(j)$ and $X(k)$ such that $x_j = x_k = 1$, are in $PA_{t-1}$.

We define

$$S_0 = \{1, 2, \ldots, n\} \quad \text{and} \quad PA_0 = \phi.$$

Assuming that $S_{t-1}$ and $PA_{t-1}$ are defined and satisfy the induction hypothesis, define $PA_t$ and $S_t$ according to the behavior of $M$ on the inputs from $I_{t-1}$ during steps 1 through $t$. Let

$$PA_t = PA_{t-1} \cup \{ (j, k) \mid j \neq k \text{ and on some input } (x_1, x_2, \ldots, x_n) \text{ in } I_{t-1}, \text{ some processor, during steps 1 through } t, \text{ reads ROM positions } X(j) \text{ and } X(k), \text{ and } x_j = x_k = 1 \}. \quad (2)$$

of the induction hypothesis for $t-1$ would imply (2) of the induction hypothesis for $t$, provided that $I_t \subseteq I_{t-1}$. Clearly always $(j, k) \in PA_t$, if and only if $(k, j) \in PA_t$.

Now if processor $P(l)$ on input $(x_1, x_2, \ldots, x_n)$ in $I_{t-1}$ reads from ROM positions $X(i_1), X(i_2), \ldots, X(i_{t-1})$ during steps 1 through $t-1$, by definition of $PA_{t-1}$, $I_{t-1}$ and the induction hypothesis, if $(f, k)$ are distinct indices in $\{i_1, i_2, \ldots, i_{t-1}\}$ we cannot have $x_f = x_k = 1$. Let, for $r = 1, 2, \ldots, t-1$, $I_{t-1}[l, r] = \{ (x_1, x_2, \ldots, x_n) \mid (x_1, \ldots, x_n) \in I_{t-1}, \text{ and at step } r, \text{ } P(l) \text{ on input } (x_1, \ldots, x_n) \text{ reads from a ROM cell which contains a } 1 \}.$
Claim 1. \(|PA| \leq qt^2\).

Proof. Since on any input in \(I_{t-1}\), \(P(l)\) cannot read a value 1 from more than one ROM cell during steps 1 through \(t-1\) and the history on all inputs in \(I_{t-1}\) is the same, we conclude that all inputs in \(I_{t-1}[l, r]\) cause \(P(l)\) to go through the same sequence of states during steps 1 through \(t-1\). Hence on all inputs in \(I_{t-1}[l, r]\), \(P(l)\) will read from the same ROM locations during steps 1 through \(t\). By the above argument, each processor \(P(l)\) can contribute at most \(2(t-1)\) pairs to \(PA\), which are not in \(PA_{t-1}\). Since \(|PA_0| = 0\) we get by induction \(|PA| \leq qt(t-1) \leq qt^2\) for \(t = 1, 2, \ldots\).

We now would like to define \(S_t\) in such a way that all inputs in \(I_t\) will have the same history through step \(t\) and such that \(I_t \subseteq I_{t-1}\). Define

\[
I_t = \{ (x_1, x_2, \ldots, x_n) \mid (x_1, \ldots, x_n) \text{ in } I_{t-1} \text{ and for all } (j, k) \text{ in } PA, (x_j, x_k) \neq (1, 1) \}.
\]

Intuitively, every input \((x_1, x_2, \ldots, x_n)\) in \(I_{t-1}\) such that \(x_j = x_k = 1\) for some \((j, k) \in PA\), is already confirmed to be in \(L_2\) by \(M\), and cannot help in our adversary argument. We now consider three cases.

Case 1. On some input \(\tilde{x} = (x_1, x_2, \ldots, x_n)\) in \(I_t\), some processor during steps 1 through \(t\) reads only 0's from the ROM, and writes in step \(t\). Let \(U\) be the set of ROM positions from which \(P(l)\) has read on input \(\tilde{x}\) during steps 1 through \(t\). Let \(S_t = S_{t-1} - U\). Since \(|U| \leq t\), we have \(|S_t| \geq S_{t-1} - t\). Also, \(S_t \subseteq S_{t-1}\) and \(PA_{t-1} \subseteq PA_t\). Hence \(I_t \subseteq I_{t-1}\). Now let \(\tilde{x}' = (x'_1, x'_2, \ldots, x'_n)\) be any input in \(I_t\). Since \(I_t \subseteq I_{t-1}\), \(\tilde{x}\) and \(\tilde{x}'\) have the same history through step \(t-1\). Hence by induction on the step number, the set of ROM locations from which \(P(l)\) reads on \(\tilde{x}'\) is also \(U\). Also \(x'_i = 0\) for all \(i\) in \(U\). Hence on \(\tilde{x}'\), \(P(l)\) writes at step \(t\) the same values as on \(\tilde{x}\). We conclude that all inputs in \(I_t\) have the same history through step \(t\).

Case 2. There is a set \(U \subseteq S_{t-1}\) such that \(|U| \leq |S_{t-1}|/2\), and by fixing all input positions in \(U\) to 0 no processor will write at step \(t\) on any input in \(I_t\), which has value 0 in all these fixed positions. Finally, if we let \(S_t = S_{t-1} - U\), no processor will write at step \(t\) on any input in \(I_t\). Hence all inputs in \(I_t\) have the same history through step \(t\). Clearly \(I_t \subseteq I_{t-1}\).

Case 3. Neither one of the Cases 1, 2 holds. Consider the subset \(S'_t\) of \(S_{t-1}\) defined by \(S'_t = \{ i \mid i \text{ in } S_{t-1} \text{ and on some input } (x_1, \ldots, x_n) \text{ in } I_t \text{ such that } x_i = 1 \text{ some processor reads from ROM position } X(i) \text{ during one of the first } t \text{ steps, and writes at step } t \}\). Suppose that \(|S'_t| < |S_{t-1}|/2\). Then, since Case 1 does not hold, Case 2 must hold with \(U = S'_t\). We conclude that \(|S'_t| \geq |S_{t-1}|/2\). Let
$W_t = \{(i, j) | \text{on some input in } I_t \text{ some processor by step } t \text{ read a 0 from } X(i) \text{ and 1 from } X(j) \text{ or read 0 from } X(j) \text{ and 1 from } X(i)\}.$

**Claim 2.** $|W_t| \leq 2qt^2$.

**Proof.** Since there are $q$ processors, it is enough to show that each processor contributes at most $2t^2$ pairs to $W_t$. This is true since no processor read a 1 from two distinct ROM locations on any input in $I_t$. Hence there are at most $t$ possibilities for the vector of locations from which the processor read, and each vector contributes at most $2t$ pairs to $W_t$.

As a guide in constructing $S_t$, we define $i, j$ in $S_t$ such that $i \neq j$ as independent if $(i, j)$ and $(j, i)$ are not in $PA_t \cup W_t$.

Now we will construct $S_t$ by keeping an index $i_0$ in $S_t$ which participates in a minimum number of pairs in $PA_t \cup W_t$, and then deleting from $S_t$ all indices $i$ such that $(i_0, i), (i, i_0) \in PA_t \cup W_t$. Formally,

$$S_t = S_t - \{i \mid (i_0, i), (i, i_0) \in PA_t \cup W_t\}.$$

The selection of $i_0$ is done as follows: For $i \in S_t$, let $w(i)$ be the number of pairs of $PA_t \cup W_t$ in which $i$ is a member. Now, consider $\sum_{i \in S_t} w(i)$. Each pair in $PA_t \cup W_t$ contributes at most 2 to this sum, and $|PA_t \cup W_t| \leq 3qt^2$. Hence there exists an index $i_0$ in $S_t$ which is a member in at most $6qt^2/|S_t|$ pairs in $PA_t$. Define $S_t$ using $i_0$ as above. According to our previous notation

$$I_t = \{(x_1, \ldots, x_n) \mid \text{for some } i \in S_t, x_i = 1, \text{for all } j \not\in S_t, x_j = 0, \text{and for all } (k, l) \in PA_t, (x_k, x_l) \neq (1, 1)\}.$$

**Claim 3.** All inputs in $I_t$ have the same history through step $t$.

**Proof.** Using the induction hypothesis, all inputs in $I_t$ have the same history through step $t - 1$. By definition of $S_t$ and $S_t'$, there must be an input $\tilde{x} = (x_1, \ldots, x_n)$ in $I_t$ and some processor $P(l)$ such that on input $\tilde{x}$, $P(l)$ reads a 1 from $X(i_0)$ during steps 1 through $t$, and writes at step $t$. Suppose that $\{j_1, j_2, \ldots, j_r\}$ ($r \leq t$) are the distinct ROM locations from which $P(l)$ has read during the first $t$ steps on input $\tilde{x}$, such that $j_1 = i_0$.

Then by definition of $i_0$ and $W_t$, $j_2, \ldots, j_r$ are not in $S_t$. Hence for all $\tilde{y} = (y_1, \ldots, y_n)$ in $I_t$, $y_{j_k} = 0$ ($k = 2, \ldots, r$). Hence on all inputs $(y_1, \ldots, y_n)$ in $I_t$ for which $y_{i_0} = 1$, $P(l)$ reads the same values from the ROM during the first $t$ steps, and hence writes the same value at step $t$ on all of them.

Now consider an input $\tilde{y} = (y_1, \ldots, y_n)$ in $I_t$ such that $y_{i_0} = 0$ and $y_{i_1} = 1$ for some $i_1$ in $S_t$. Some processor $P(l')$ on input $\tilde{y}$ reads from distinct ROM locations $\{k_1, \ldots, k_r\}$ during the first $t$ steps, where $k_1 = i_1$, and writes
at step $t$. Since $i_i \in S_i$, $i_0$ is not in $\{k_1, ..., k_r\}$ by the definitions of $i_0$ and $W_i$. Let $\bar{y}'$ be the input obtained from $\bar{y}$ by changing the value of $y_{i_0}$ to 1. Since $(i_0, i)$ is not in $PA_i$ for all $i$ in $S_i$, $\bar{y}' \in I_i$. On $\bar{y}'$, $P(l')$ will still write at step $t$ the same value as on $\bar{y}$, hence $\bar{y}$, $\bar{y}'$ have the same history through step $t$. But then, by the previous discussion, all inputs in $I_i$ have the same history through step $t$.

**Claim 4.** In all three cases, if $t < |S_{i-1}|/8 \sqrt{q}$, then $|S_i| \geq |S_{i-1}|/4$.

**Proof.** In Case 1, $|S_i| \geq |S_{i-1}| - t \geq 7 |S_{i-1}|/8 \geq |S_{i-1}|/4$ (since $t < |S_{i-1}|/8$). In Case 2, $|S_i| \geq |S_{i-1}|/2$. In Case 3 (using $|S''_i| \geq \frac{1}{2} |S_{i-1}|$),

$$|S_i| \geq |S''_i| \geq \frac{6qt^2}{|S''_i|} \geq \frac{3 |S_{i-1}|^2}{32 |S''_i|} \geq \frac{3 |S_{i-1}|^2}{2} - \frac{3 |S_{i-1}|^2}{32} \geq \frac{5 |S_{i-1}|}{16} \geq \frac{|S_{i-1}|}{4}.$$  

**Claim 5.** There exists $n_0$ such that for all $n \geq n_0$ and $0 < t < ((2 - a)/8) \log n$, $|S_i| \geq n/4^t$ and $|S_i|^2 \geq 4 |PA_i|$.

**Proof.** We will choose $n_0$ such that $(2 - a) \log n < n^{(2 - a)/4}$ for all $n \geq n_0$. We now prove the claim by induction. $|S_0| = n$. Suppose $|S_{i-1}| \geq n/4^{t-1}$. Then

$$|S_{i-1}| \geq \frac{n}{4^{(2-a)\log n/8}} = \frac{n}{2^{(2-a)\log n/4}}$$

$$= \frac{n}{n^{(2-a)/4} = n^{(2-a)/4}}$$

$$= n^{(2-a) + 2a/4} = n^{a/2} n^{(2-a)/4}.$$  

Hence

$$\frac{|S_{i-1}|}{8 \sqrt{q}} \geq \left(\frac{n^{a/2} n^{(2-a)/4}}{8n^{a/2}}\right) \frac{n^{(2-a)/4}}{8} > \frac{(2-a)}{8} \log n > t.$$  

Hence by Claim 4, $|S_i| \geq |S_{i-1}|/4 \geq n/4^t$. Now, $n/4^t \geq n^{(2-a)/4} \log n/8 = n^{a/2} n^{(2-a)/4}$. Hence $|S_i|/8 \sqrt{q} > n^{(2-a)/4} / 8 > ((2-a)/8) \log n > t$. Hence $|S_i|^2 \geq 4qt^2 \geq 4 |PA_i|$.
CLAIM 6. If \( n \geq n_0 \) and \( 0 < t < (2 - a)/8 \log n \), then \( |S_i| (|S_i| - 1) > |PA_i| \).

Proof. Since \( |S_i|^2 > 4qt^2 \geq 4, |S_i| \geq 2 \). So

\[
|S_i| (|S_i| - 1) = |S_i|^2 - |S_i|
\]

\[
= |S_i|^2 - \frac{|S_i|^2}{|S_i|}
\]

\[
\geq |S_i|^2 - \frac{|S_i|^2}{2}
\]

\[
= \frac{|S_i|^2}{2} \geq 2 |PA_i|.
\]

If \( |PA_i| = 0 \), then \( \frac{1}{2} |S_i|^2 > 1 > |PA_i| \). If \( |PA_i| > 0 \), then \( \frac{1}{2} |S_i|^2 > 2 |PA_i| > |PA_i| \)  

The following claim completes the proof of the theorem.

CLAIM 7. If \( n \geq n_0 \) and \( T < (2 - a)/8 \log n \), then \( M \) cannot recognize \( L_2 \) in depth \( T \).

Proof. By Claim 6 \( |S_T|(|S_T| - 1) > |PA_T| \). Hence there are indices \( i, j \) in \( S_T, i \neq j \), such that \( (i, j) \) and \( (j, i) \) are not in \( PA_T \). Consider the inputs \( x = (x_1, x_2, ..., x_n) \) and \( x' = (x'_1, ..., x'_n) \) such that: \( x_i = 1, x_k = 0 \) for \( k \neq i \), \( x'_i = x'_j = 1, x'_k = 0 \) for \( k \) not in \{i, j\}. \( \bar{x}, \bar{x}' \) are in \( I_T \), hence have the same history through step \( T \). But \( \bar{x} \) is in \( L_2 \), while \( \bar{x}' \) is not. Hence \( M \) cannot recognize \( L_2 \) in depth \( T \).

Since \( L_2 \) can easily be recognized in constant depth on an ARBITRARY(1) with \( n \) processors, we have

THEOREM 3.3. \( L_2 \) separates COMMON(1) with ROM and \( n^a \) processors, \( 0 < a < 2 \), from ARBITRARY(1) with \( n \) processors, with or without ROM.

We also note that for \( 1 \leq a < 2 \) processors the bound is tight up to a multiplicative constant, since COMMON(1) with at least \( n \) processors can recognize \( L_2 \) in depth \( O(\log n) \) by using binary search (See Sect. 4) to find the minimum \( i \) such that \( x_i = 1 \), and then having any \( P(j) \) such that \( j \neq i \) and \( x_j = 1 \) write into the shared-memory cell. In fact, with ROM, a COMMON(1) with \( n/\log n \) processors is sufficient.

We now generalize our result to more memory cells.
COROLLARY 3.4. For all \( m \geq 1 \) and \( 1 \leq a < 2 \) there is a tight \( \Omega(\log n/\log(m + 1)) \) separation between COMMON\((m)\) with ROM and \( n^a \) processors, and ARBITRARY\((1)\) with \( n^a \) processors.

Proof. The separation is tight since (Fich et al., 1984) show how COMMON\((m)\) with \( q \) processors can simulate one step of ARBITRARY\((1)\) with \( q \) processors in \( O(\log q/\log(m + 1)) \) steps. In our case \( q = n^a \), hence \( \log q = a \log n \). Theorem 3.2 gives the separation for \( m = 1 \). The generalization of Theorem 3.2 to \( m > 1 \) is done by a similar adversary argument. At step \( t \), we try to prevent any processor from writing into \( C(i) \) at step \( t \) by fixing a not too large fraction of the unfixed input positions to 0. For each \( C(i) \) for which this is impossible, find a set \( S'_j(i) \) of unfixed input positions such that by having a value 1 will cause some processor to write into \( C(i) \) at step \( t \). Since the previous case did not hold for those \( C(i) \), one of those \( S'_j(i) \) must have \( |S'_j(i) - \bigcup_{j \neq i} S'_j(j)| \) large enough so that we can fix all the positions in \( S'_j(j) \) (for all \( j \neq i \)) to 0 and proceed as in the proof of Theorem 3.2.

4. SEPARATION BETWEEN ARBITRARY\((m)\) WITH ROM AND PRIORITY\((1)\)

In this section we show for the first time that a certain simple function which can be computed in constant depth on PRIORITY\((1)\), even without ROM, requires depth at least \( T \), where \( T + \log T \geq \log n \) on ARBITRARY\((1)\) with ROM, and, in general, \( \Omega(\log n/\log(m + 1)) \) depth on ARBITRARY\((m)\) with ROM.

(Fich et al., 1984) shows an \( \Omega(\log n) \) tight separation between ARBITRARY\((1)\) without ROM and PRIORITY\((1)\) and, in general, an \( \Omega(\log n/\log(m + 1)) \) tight separation between ARBITRARY\((m)\) and PRIORITY\((1)\). We generalize their technique to prove a tight separation between ARBITRARY\((m)\) with ROM and PRIORITY\((1)\) even without ROM. For a given ARBITRARY\((1)\) with ROM, the adversary fixes bits of the input to prevent any computation of a certain depth from being able to compute the function.

The bound we obtain for ARBITRARY\((1)\) with ROM is tight up to an additive constant, and does not depend on the number of processors.

In the following theorem, by "All the inputs in some collection \( S \) have the same history through step \( t \)" we will mean: There are indices \( l_1, l_2, ..., l_s \) such that on all inputs in \( S \) there is a possible computation in which processor \( l_i \) will actually write in step \( j \) (\( l_i = 0 \) means no processor attempts writing at step \( j \) on any input in \( S \)) and the history will be the same on all inputs in \( S \).

THEOREM 4.1. If \( INDEX(x_1, x_2, ..., x_n) \) is computed in depth \( T \) on ARBITRARY\((1)\) (with ROM) then \( T + \log T + 1 \geq \log n \).
We use an adversary argument, which generalizes the technique of (Fich et al., 1984) to the case of having ROM. Let $M$ be an ARBITRARY(1) which computes $\text{INDEX}(x_1, x_2, ..., x_n)$. We define subsets $S_0, S_1, ..., S_t$ of $\Sigma^n (\Sigma = \{0, 1\})$ such that for all $t$, all inputs in $S_t$ have the same history through step $t$. These subsets are defined by induction. $S_0 = \Sigma^n (\Sigma = \{0, 1\})$ is the set of all possible inputs. We inductively define subsets $K_t, J_t$ of $\{1, 2, ..., n\}$, where $K_t$ is the set of input positions with values fixed to 0 by step $t$, and $J_t$ is the set of input positions with value fixed to 1 by step $t$. We will also define indices $l_1, l_2, ..., l_t$ by induction.

We will let $J_0 = K_0 = \emptyset$ and for $t = 0, 1, ..., S_t = \{(x_1, x_2, ..., x_n) | x_i \in \{0, 1\} (j = 1, 2, ..., n) \text{ and } x_i = 0 \text{ for all } i \in K_t \text{ and } x_i = 1 \text{ for all } i \in J_t\}$.

$S_t, K_t, J_t$ are defined inductively from $S_{t-1}, K_{t-1}, J_{t-1}$. We inductively assume that all inputs in $S_{t-1}$ have the same history through step $t-1$, where processor $l_{t-1}$ actually writes at step $j$ on all those inputs $(1 \leq j \leq t-1)$. If $l_{t-1} = 0$ then no processor attempts writing.

Let $A_{t-1} = \{1, 2, ..., n\} - (K_{t-1} \cup J_{t-1})$. Let $L_{t-1}, R_{t-1}$ be such that $A_{t-1} = L_{t-1} \cup R_{t-1}$, every index in $L_{t-1}$ is lower than any index in $R_{t-1}$ and $|L_{t-1}| = \frac{1}{2} |A_{t-1}|$. Intuitively, $L_{t-1}$ is the low half of $A_{t-1}$ and $R_{t-1}$ is the high half of $A_{t-1}$. Several cases arise.

Case 1. At step $t$ no processor writes on any input in $S_{t-1}$. Then let $S_t = S_{t-1}, K_t = K_{t-1}, J_t = J_{t-1}$. Clearly then by induction all inputs in $S_t$ have the same history through step $t$. Let $l_t = 0$.

If however, at step $t$ some processor writes on some input in $S_{t-1}$, then to every input $\bar{x} = (x_1, x_2, ..., x_n)$ in $S_{t-1}$ and every processor $P(l)$ which writes on $\bar{x}$ at step $t$ there correspond two sets: $U(\bar{x}, l) = \{i | i \in A_{t-1}, x_i = 0 \text{ and during steps } 1 \text{ through } t P(l) \text{ has read from } X(i)\}$; $V(\bar{x}, l) = \{i | i \in A_{t-1}, x_i = 1 \text{ and during steps } 1 \text{ through } t P(l) \text{ has read from } X(i)\}$. Clearly $|U(\bar{x}, l) + V(\bar{x}, l)| \leq t$. We now have

Case 2. For all $\bar{x}$ in $S_{t-1}$ and $l$ as above, $V(\bar{x}, l) \cap L_{t-1}$ is nonempty. Intuitively, this means that each processor, in order to write at step $t$ on input $(x_1, x_2, ..., x_n)$ in $S_{t-1}$ and every processor $P(l)$ which writes on $\bar{x}$ at step $t$ there correspond two sets: $U(\bar{x}, l) = \{i | i \in A_{t-1}, x_i = 0 \text{ and during steps } 1 \text{ through } t P(l) \text{ has read from } X(i)\}$; $V(\bar{x}, l) = \{i | i \in A_{t-1}, x_i = 1 \text{ and during steps } 1 \text{ through } t P(l) \text{ has read from } X(i)\}$. Clearly $|U(\bar{x}, l) + V(\bar{x}, l)| \leq t$. We now have

Case 3. There exists $\bar{x}, l$ as above such that $V(\bar{x}, l) \cap L_{t-1}$ is empty. Intuitively this means that all bits $x_i$ of $\bar{x}$ which were read by $P(l)$ by step $t$ and are in the lower half of the unfixed positions have value 1. In this case, the adversary fixes all the bits in this lower half to 0, thus making sure that no processor will write at step $t$ on any input. Formally: $K_t = K_{t-1} \cup L_{t-1}, J_t = J_{t-1}$. Clearly all inputs in $S_t$ have the same history through step $t$. Let $l_t = 0$. 
in \( R_{t-1} \) to 0. Formally, \( K_t = K_{t-1} \cup U(\bar{x}, l) \cup (R_{t-1} - V(\bar{x}, l)) \), \( J_t = J_{t-1} \cup V(\bar{x}, l) \). Let \( l_t = l \).

Since \( S_t \subseteq S_{t-1} \), by induction all inputs in \( S_t \) have the same history through step \( t-1 \), \( P(l) \) will read from the same ROM positions on all inputs in \( S_t \), and will read the same values. Hence all inputs in \( S_t \) have the same history through step \( t \).

We now count the number of positions which are unfixed by step \( t \). This number is \( |A_t| \), which we denote by \( u_t \). Now \( u_0 = |A_0| = n \). In Case 1 we have \( u_t = u_{t-1} \). In Case 2 \( u_t \geq (u_{t-1})/2 \). In Case 3 \( u_t \geq \frac{1}{2}(u_{t-1} - 1) - t \). Now we prove by induction that \( u_t \geq n/2^t - 2t \). Since \( u_0 = n \) this assertion is true for \( t = 0 \). Assuming that

\[
\begin{align*}
u_{t-1} &\geq \frac{n}{2^{t-1}} - 2(t-1) \\
u_t &\geq \frac{1}{2} (u_{t-1} - 1) - t \geq (n/2^{t-1} - 2(t-1) - 1)/2 - t \\
&= \frac{n}{2^t} - (t-1) - \frac{1}{2} - t \\
&= \frac{n}{2^t} - 2t + \frac{1}{2} \geq \frac{n}{2^t} - 2t.
\end{align*}
\]

Now if \( t + \log t + 1 < \log n \) then \( 2^t2t < n \), hence \( n/2^t > 2t \), hence \( n/2^t - 2t > 0 \), hence \( u_t - |A_t| > 0 \). So \( A_t \) is nonempty which means that there is at least one unfixed position. Now let \( i_t = \min(A_t) \). By the above construction, for every \( j \in J_t \) we have \( i_t < j \). In other words, no input position which is lower than some position in \( A_t \) is ever fixed to 1.

Let \( S_t(0) = \{ (x_1, x_2, \ldots, x_n) \mid (x_1, \ldots, x_n) \in S_t \text{ and } x_{i_t} = 0 \} \), \( S_t(1) = \{ (x_1, x_2, \ldots, x_n) \mid (x_1, x_2, \ldots, x_n) \in S_t \text{ and } x_{i_t} = 1 \} \). \( S_t(0) \) and \( S_t(1) \) are both nonempty:

\[
\begin{align*}
\text{INDEX}(\bar{x}) &= i_t \quad \text{for all } \bar{x} \text{ in } S_t(1), \\
\text{INDEX}(\bar{x}) &> i_t \quad \text{for all } \bar{x} \text{ in } S_t(0).
\end{align*}
\]

Hence, if \( M \) computes \( \text{INDEX} \) in depth \( T \), and \( T + \log T + 1 < n \) there are two inputs \( \bar{x}_0, \bar{x}_1 \) with \( \text{INDEX}(\bar{x}_0) \neq \text{INDEX}(\bar{x}_1) \) having the same history through step \( T \), a contradiction.

Now, \( \text{INDEX} \) can clearly be computed in a constant depth on \( \text{PRIORITY}(1) \), even without ROM. Hence we get a separation result. Next we show that the lower bound obtained for \( \text{INDEX} \) is tight, up to an additive constant.
Theorem 4.2. Whenever $T + \log T \geq \log n$, INDEX($x_1, x_2, ..., x_n$) can be computed on COMMON(1) with ROM, and hence by an ARBITRARY(1) with ROM, in depth $T + 2$, using $n$ processors. For example $T$ can be $\log n - \log \log n + 1$.

Proof. The COMMON(1) will perform a binary search for $\min\{i \mid x_i = 1\}$ with a speed-up using the ability of each processor to read any ROM cell $X(j)$ in one step. For the sake of simplicity, we demonstrate the idea for the case in which $n = 2^k + j$, $1 \leq k \leq 2^k$. In this case $T + \log T = 2^k + k = 2^k + j = \log n$. The proof of the general case is an obvious generalization. The COMMON(1) with ROM which computes INDEX($x_1, x_2, ..., x_n$) in at most $T + 2$ steps uses the following idea.

Assume that $X(1), X(2), ..., X(n)$ and $P(1), P(2), ..., P(n)$ are ordered from left to right.

The following obvious PARALLEL BINARY SEARCH algorithm can for $n = 2^u$ in $\log n + 1$ steps find an interval $J_h$ of the form $J_h = [(h - 1)2^u - 1, h2^u]$ for some $h$, $1 \leq h \leq 2^u$ such that INDEX($x_1, x_2, ..., x_n$) is in $J_h$ unless INDEX($x_1, ..., x_n$) = $n + 1$. (As usual $x_i$ is the contents of ROM cell $X(i)$.)

PARALLEL BINARY SEARCH

(1) In parallel $P(i)$ reads from $X(i)$ ($i = 1, 2, ..., n$)

(2) Initially all processors have $J^{(1)} = [1, n] = [1, 2^u]$

(3) Successively, for $r = 1, 2, ...$: Let $K$ be leftmost half of the interval $J^{(r)}$. For all $i$ in $K$ $P(i)$ in parallel write $r$ into the common memory cell $C(1)$, provided $X(i)$ contains 1. Then all the processors read from $C(1)$. If $C(1)$ contains $r$ then let $J^{(r+1)} = K$. Otherwise $J^{(r+1)}$ is the rightmost half of $J^{(r)}$. When $r = l$, let $J_h = J^{(l)}$.

(End PARALLEL BINARY SEARCH)

If we would have a COMMON(1) without a ROM it looks like we would have to use $l = u$ and the binary search would take $u + 1 = \log n + 1$ steps. Fich et al. (1984) have obtained for COMMON(1) without a ROM a lower bound of $\log_2 n$. We now show how the ROM can be used to compute the INDEX in $2^k + 1 = T + 1$ steps. We choose $l = T = 2^k$, while $u = \log n = 2^k + j$. The idea is that in parallel to the execution of PARALLEL BINARY SEARCH, for $h = 1, 2, ..., 2^u$, $P((h - 1)2^u - 1 + 1)$ can all in $2^u - 1 = 2^u + j - 2^u = 2^u - 2^u = T$ steps read in parallel $X(i)$ for $i = (h - 1)2^u - 1 + 1, (h - 1)2^u + 1, ..., h2^u - 1$ in that order, thus determining $\min\{i \mid x_i = 1\}$. Now at step $T + 1$ an integer $h_0$ is determined by the binary search, such that INDEX($x_1, ..., x_n$) is in $J_{h_0}$ unless INDEX($x_1, ..., x_n$) = $n + 1$. At this point $P((h_0 - 1)2^u - 1 + 1)$ knows INDEX($x_1, x_2, ..., x_n$) and writes it into $C(1)$.


Theorem 4.2 implies that the lower bound of Theorem 4.1 is tight up to an additive constant. Also note that for \( n = 2^k + j \), \( j \leq k \), a straightforward binary search approach with \( l = \log n = 2^k + j \) will take \( 2^k + j + 1 \) steps, while our algorithm will use at most \( 2^k + 2 \) steps.

We now generalize our separation to \( m \) shared memory cells.

**Corollary 4.3.** For all \( m \geq 1 \) there is an \( \Omega(\log n / \log(m+1)) \) depth separation between \( \text{ARBITRARY}(m) \) with ROM and \( \text{PRIORITY}(1) \), and this separation is tight.

**Proof.** Theorem 4.1 shows the separation for \( m = 1 \). The case \( m > 1 \) is a generalization of Theorem 4.1. The main idea is to divide the unfixed input positions into at most \( m + 1 \) (rather than 2) consecutive blocks. We omit the details. The fact that the separation is tight follows from a \( \log n / \log(m+1) \) simulation of \( \text{PRIORITY}(1) \) by \( \text{ARBITRARY}(m) \), given in (Fich et al., 1984).

---

5. **LOWER AND UPPER BOUNDS FOR COMPUTING PARITY ON NONDETERMINISTIC PRIORITY(1)**

Vishkin and Wigderson (1985) ask about the lower bound for computing PARITY on NPRIORITY(1) (with ROM) and conjectured an \( \Omega(\sqrt{n}) \) lower bound. In this section, we introduce new and interesting techniques which enable us to establish lower bounds on the depth of NPRIORITY(1) without ROM, and NPRIORITY(1) with ROM which compute PARITY. We also disprove the conjectured \( \Omega(\sqrt{n}) \) lower bound of op cit. by presenting an \( O(n^{1/3}) \) upper bound for computing PARITY on NPRIORITY(1) with ROM. We start by considering NPRIORITY(1) without ROM.

**Theorem 5.1.** It requires \( \Omega(\sqrt{n/\log d}) \) time to compute the function PARITY by an NPRIORITY(1) without ROM, where \( n \) is the number of processors and inputs, and \( d > 1 \) is the branching factor.

**Proof.** We actually prove that the above depth is required even for accepting the language of even parity strings. For each input of even parity, fix one accepting computation.

We will partition the inputs according to different accepting computations which are chosen for each input. Step \( t \) of a computation is characterized by the triple \((i_t, \text{input}, b)\), where \( i_t \) is the index of the processor that succeeds in writing at step \( t \) (w.l.o.g., \( i_t = 1 \) if no processor writes in step \( t \)). \( b \) is a vector \((d_1, d_2, ..., d_t)\) meaning that processor \( P(i_t) \) at
step \( k \) \((k = 1, 2, \ldots, t)\) used branch \( d_k \), and the input is the input bit to \( P(i) \).
In this proof, by computation we will mean a sequence of triples as above.

The number of such triples at level \( k \) is \( 2n^d_k \), where \( n \) is the number of all possible choices of \( i \); \( 2 \) is for two possible binary inputs; \( d_k \) is the number of different sequences of branching numbers which may be used by \( P(i) \) up to level \( k \).

**Claim.** At level \( k \), there is a group \( G_k \) of \( 2^n/2n^k2^k d^{k(k+1)/2} \) inputs of even parity having the same accepting computation up to step \( k \). That is, the first \( k \) tuples for the accepting computations fixed to all the inputs in \( G_k \) are the same.

**Proof** (of claim). There are \( 2^n/2 \) inputs of even parity. The number of distinct computations up to step \( k \) is at most \( \prod_{j=1}^{k-1} 2n^d_j = n^k2^k d^{k(k+1)/2} \). The claim follows by picking up the largest set of inputs corresponding to one computation.

Now we use this claim to prove Theorem 5.1. Consider the accepting computation defined in the claim. Assume that the computation depth is \( t \). If \( |G_\ell| \geq 2 \), then at least two different inputs of even parity have the same accepting computation. Fix two such inputs \( I_1, I_2 \). \( I_1 \) differs from \( I_2 \) in at least one position. Choose such a position \( p \) in \( I_1 \). Change its value to the opposite value (\( I_2 \)'s corresponding value). This will not change the history for the following reasons:

(a) This bit is not one of the inputs for processors \( i_1, i_2, \ldots, i_\ell \), since those are already fixed to be the same for \( I_1 \) and \( I_2 \).

(b) The processor \( P(p) \) will not influence the history since for both \( I_1 \) and \( I_2 \), \( P(p) \) did not do so.

(c) Other processors (except for \( P(p) \)) will not be influenced since the history of the computation is the same and their input bits are not changed.

Therefore, we conclude that the NPRIORITY(1) accepts an input of odd parity, a contradiction. Hence we have to have \( |G_\ell| < 1 \). That is,

\[
\frac{2^n}{2n^t2^t d^{t(t+1)/2}} \leq 1.
\]

So, \( 2^n \leq n^t 2^{t+1} d^{t(t+1)/2} \),

\[
n \leq t \log n + t + 1 + \frac{t(t+1)}{2} \log d
\]

\[
t = \Omega(\sqrt{n/\log d}).
\]
We now show a matching upper bound for PARITY on N_PRIORITY(1) without ROM.

**Theorem 5.2.** PARITY can be computed in depth $O(\sqrt{n/\log d})$ on N_PRIORITY(1) without ROM.

**Proof.** We consider the case $d=2$. The generalization to $d > 2$ is obvious. For simplicity, assume $n = k^2$. In $k$ parallel steps the $k$ processors $P(k), P(2k), \ldots, P(n)$ each nondeterministically guesses $k$ bits. Specifically, $P(jk)$ guesses $y_j = (y_{j1}, y_{j2}, \ldots, y_{jk})$ $(j = 1, 2, \ldots, k)$, and computes $Z_j = \sum_{i=1}^{k} y_{ji} \pmod{2}$ as it guesses the $y_j$'s. Then, sequentially, for $j = 1, 2, \ldots, k$, $P(jk)$ writes its guess into the shared-memory cell, and in parallel for $l = 1, 2, \ldots, k$, $P((j-1)k + l)$ writes a special symbol $R$ into the shared memory if $x_{(j-1)k + l} \neq y_{jl}$. Every processor reads from the shared memory. If any processor reads $R$, the computation halts. After these $2k$ parallel steps, if no processor writes $R$ into the shared memory, it is confirmed that $(x_1, x_2, \ldots, x_n) = (y_{11}, y_{12}, \ldots, y_{1k}, y_{21}, y_{22}, \ldots, y_{2k}, \ldots, y_{k1}, \ldots, y_{kk})$. Hence $\sum_{j=1}^{n} x_j = \sum_{j=1}^{k} Z_j \pmod{2}$.

Now $P(k)$ writes $Z_1$ into the shared memory. Then for $j = 2, \ldots, k$ in this order, $P(jk)$ reads the contents of the shared memory, adds $Z_j$ to it (mod 2), and writes the result into the shared memory. After these $k$ steps, $Z = \sum_{j=1}^{k} Z_j \pmod{2}$ is written into the shared memory. $P(1)$ reads $Z$ and writes $\langle A, Z \rangle$ into the shared memory.

We now show that PARITY can be computed on N_COMMON(1) with ROM in $O(n^{1/3})$ depth, thus disproving a conjecture of Vishkin and Wigderson (1985).

**Theorem 5.3.** PARITY can be computed in depth $O(n^{1/3}/\log d)$ on N_COMMON(1) with ROM, using $n^{2/3}$ processors and branching factor $d$.

**Proof.** For simplicity of exposition we assume $n = u^3$ and $d = 2$. Partition the ROM positions into $u^2$ equal distinct groups of $u$ bits each. In $u = n^{1/3}$ parallel steps each of the $u^2$ processors reads the bits in one of those groups and computes their sum modulo 2. Different processors read different groups. Let $P(i)$ have sum $y_i$ $(i = 1, 2, \ldots, u^2)$. The output should be $\sum_{i=1}^{u^2} y_i \pmod{2}$. Using the algorithm of Theorem 5.2, this sum can be computed nondeterministically in $\sqrt{u^2} = u = n^{1/3}$ parallel steps.

In the following proof, a labeled hypergraph $(V, E)$ is given by a set $V$ of vertices and a set $E$ of edges where every edge is of form $e = \langle (v_1, v_2, \ldots, v_a), w \rangle$ for $v_i$ $(i = 1, 2, \ldots, a)$ being distinct members of $V$ (a is called the size of $e$) and $w$ is an (arbitrary) label. The degree of a vertex is the number of distinct edges on which it is incident. See (Berge, 1973) for standard terminology for hypergraphs.
Theorem 5.4. It requires $\Omega(\log \log n/\log d)$ depth to compute the parity function of $n$ bits by an NPRIORITY$(1)$ with ROM and with branching factor $d$ and $n$ processors.

Proof. Again, we prove that the lower bound holds even if the NPRIORITY$(1)$ only accepts the language of even-parity strings. The output is presented in $C(1)$, the only shared memory cell, when $M$ stops. The input is presented in the ROM. For each input $X = (x_1, x_2, \ldots, x_n)$ of even parity, we fix one accepting computation $COMP_X$. A computation $COMP_X$ is characterized as triples $triple_k = (i_k, (y_1, y_2, \ldots, y_k), (d_1, d_2, \ldots, d_k))$ for $k = 1, 2, \ldots, T$, which have the following meaning. At the $k$th step of the computation $COMP_X$, triple$_k$ is used. Processor $i_k$ read input bit $y_j$ (from triple$_k$), took branch $d_j$ (from triple$_k$) at step $j$ for $1 \leq j \leq k$ and at the $k$th step either $i_k = 1$ and no processor writes, or $P(i_k)$ succeeds in writing. This definition of a computation is similar to that given in Theorem 5.1. Notice that for a fixed computation on $X$, $COMP_X$ uniquely determines the history of this computation and the addresses accessed by $P(i_k)$ up to step $k$ for $k = 1, 2, \ldots, T$. In the following, for each input $X$ of even parity, we consider only a fixed accepting computation $COMP_X$. Similar to the claim in Theorem 5.1, we have the following claim.

Claim 1. There is a group $G_k$ of at least

$$\frac{2^n}{f(n, d, k)},$$

where $f(n, d, k) = 2n^k \prod_{j=1}^{k} (2d)^j$,

inputs of even parity having the same accepting computation $COMP$ up to step $k$. That is, the first $k$ triples of the accepting computations fixed to the inputs in $G_k$ are the same.

Proof. The proof is similar to the proof of the claim in Theorem 5.1. There is only one difference. In this proof, we have a ROM to store the input, and every processor can look at many bits of the input. This is why we have a vector of $(y_1, y_2, \ldots, y_k)$. Up to step $k$, we have $k$ triples. A triple at step $j$, $1 \leq j \leq k$, can have $n2^j d^j$ values. There are $2^{n-1}$ even-parity inputs. So there is a group $G_k$ of inputs that have the same $COMP$ up to step $k$ of size at least

$$\frac{2^n}{f(n, d, k)},$$

where $f(n, d, k) = 2n^k \prod_{j=1}^{k} (2d)^j$.

Up to here the proof has been similar to Theorem 5.1. A naive approach would be to try to claim that if two inputs of even parity have the same computation, then, as in Theorem 5.1, we can change the parity of one
input (by changing a bit) without changing the computation. But observe that we cannot change a bit as freely as before because the input bits are in the ROM which can be read by many processors. This is why this proof turns out to be much more complicated than the proof of Theorem 5.1.

W.l.o.g., we assume that all inputs take precisely $T$ steps. Now fix the COMP of depth $T$ with the maximum $G_T$ as in Claim 1. So all inputs in $G_T$ have computation COMP. By fixing COMP we also fixed the history $H_1, H_2, \ldots, H_T$, where $H_i = \langle A, a \rangle$. We construct a labeled hypergraph $HG = (V, E)$ as follows:

1. $V = \{ p | p $ is a position in ROM holding an input bit $\}$. $|V|=n$.
2. $E = \{ (q_1, q_2, \ldots, q_a), (w_1, w_2, \ldots, w_a) | a = (2d)^j \text{ there is a processor } P(l) \text{ such that with the fixed history } H_1, H_2, \ldots, H_{j-1}, (q_1, \ldots, q_a) \text{ is the collection of all ROM positions from which } P(l) \text{ can read up to step } j \text{ on any input using any branch numbers through step } j, \text{ and such that if position } q_i \text{ contains } w_i (1 \leq i \leq a), P(l) \text{ has no choice but to change COMP by succeeding in writing a value different from } H_j \text{ at step } j \}$.

The purpose of each hyperedge is to state that certain inputs with “bad” bit combinations in some locations would force some processor to change COMP. The size of the edges at step $j$ is at most $(2d)^j$ because at most $(2d)^j$ positions in ROM can be possibly reached by one processor. W.l.o.g., we assumed that $a = (2d)^j$.

**Claim 2.** For a depth $T$ computation, each processor can cause at most

$$s(T, d) = \sum_{j=1}^{T} 2^{(2d)^j} \leq T 2^{(2d)^T}$$

hyperedges in $HG$.

**Proof.** At step $j$, each processor can possibly reach at most $(2d)^j$ fixed positions in the ROM. This is true because the history is fixed.

By Claim 2, $|E| \leq ns(T, d)$. Hence the sum of the degrees of all the nodes of $HG$ is at most $|E| (2d)^T = ns(T, d)(2d)^T$, since each hyperedge can be incident upon at most $\text{len}(T, d) = (2d)^T$ nodes. So the average degree of each node in $V$ is at most $s(T, d)(2d)^T$. Therefore at least $n/2$ nodes in $V$ have degree at most $\text{deg}(T, d) = 2s(T, d)(2d)^T$. Let

$$V_{\text{free}} = \{ p_i | p_i \in V \text{ and degree } (p_i) \leq \text{deg}(T, d) \}.$$

We have $|V_{\text{free}}| \geq n/2$. W.l.o.g., we assume $|V_{\text{free}}| = n/2$. Let

$$V_{\text{fix}} = V - V_{\text{free}}.$$
So \( |V_{\text{fix}}| = n/2 \). Now we assign values to the locations in \( V_{\text{fix}} \) in such a way that they are consistent with as many members of \( G_T \) as possible. Among the \( 2^{2n} \) possible assignments, we choose the assignment, \( F \), that is consistent with a maximum number of members in \( G_T \). By Claim 1, there are at least \( 2^n/f(n, d, T) \) inputs in \( G_T \). Hence there is an assignment \( F: V_{\text{fix}} \rightarrow \{0, 1\} \) of values to the positions in \( V_{\text{fix}} \) such that there are at least \( 2^{n/2}/f(n, d, T) \) even-parity inputs in \( G_T \), where \( G_F \) is the collection of all even-parity inputs in \( G_T \) which are consistent with \( F \). Also let \( I_F \) be the set of all inputs of even parity which are consistent with \( F \).

We now localize the edges and restrict \( HG \) to have the vertex set \( V_{\text{free}} \). Define

\[
HG|_{V_{\text{free}}} = (V_{\text{free}}, E|_F),
\]

where \( E|_F \) is defined by the rules:

(a) If \( \langle \bar{e}, \bar{w} \rangle \in E \) and no position fixed by \( F \) (i.e., in \( V_{\text{fix}} \)) belongs to \( \bar{e} \), then \( \langle \bar{e}, \bar{w} \rangle \in E|_F \).

(b) For \( \langle \bar{e}, \bar{w} \rangle = \langle (e_1, \ldots, e_c), (w_1, \ldots, w_f) \rangle \), if the locations \( e_i, \ldots, e_c \) are fixed by \( F \), then

(i) if \( F \) fixes \( e_i \) to \( w_j \) for \( j = 1, 2, \ldots, c \), then we create a new hyperedge \( \langle \bar{e}', \bar{w}' \rangle \) by simply deleting the components \( e_i, \ldots, e_c, w_i, \ldots, w_f \) from \( \langle \bar{e}, \bar{w} \rangle \) and put \( \langle \bar{e}', \bar{w}' \rangle \) into \( E|_F \);

(ii) otherwise, do nothing. That is, throw away the hyperedge \( \langle \bar{e}, \bar{w} \rangle \).

(c) Only the hyperedges constructed in (a) and (b) are in \( E|_F \).

From the above discussion, we conclude,

**Claim 3.** (i) Each node in \( V_{\text{free}} \) has a degree at most \( \text{deg}(T, d) \) in the new graph \( HG|_{V_{\text{free}}} \); (ii) With the assignment \( F \), among the total \( 2^{n/2} \) inputs of even parity that are consistent with \( F \), at least \( 2^{n/2}/f(n, d, T) \) of them cause COMP.

We now bound the number of inputs in \( I_F \) which cannot have a computation COMP (and hence are not in \( G_F \)). Note that only those inputs that are inconsistent with all the hyperedges may have a computation COMP. Consider \( HG|_{V_{\text{free}}} \). If there is a node in \( V_{\text{free}} \) that has degree zero, which means that there is no restriction on this position \( p \) at all, then this bit can be either 1 or 0 without affecting the COMP, and as in Theorem 5.1, we can reach a contradiction easily. Therefore we assume that every one of the \( n/2 \) nodes has some hyperedges incident on it. So there are
at least \( n/2(2d)^T \) edges in \( E|_F \). Let \( J = I_F \). For any set \( J \) of inputs and hyperedge \( \langle e, w \rangle = \langle \{q_1, \ldots, q_a\}, \{w_1, \ldots, w_a\} \rangle \), define
\[
J_{\langle e, w \rangle} = \{ \bar{x} | \bar{x} \in J \text{ and for some } i (1 \leq i \leq a) \text{ location } q_i \text{ in } \bar{x} \text{ does not have value } w_i \}.
\]

We repeat the following process until \( E|_F \) is empty:

1. Choose an arbitrary hyperedge \( \langle e, w \rangle \) that remains.
2. From \( V_{\text{free}} \), delete all the nodes that are incident upon this edge.
3. Delete all the hyperedges incident on the nodes deleted in (2) from \( E|_F \).
4. \( J \leftarrow J_{\langle e, w \rangle} \).

This process can be repeated at least \( g(n, d, T) = n/2 \deg(T, d) \text{len}^2(T, d) \) times: Since each edge can be adjacent to at most \( \text{len}(T, d) \) nodes and each node has degree no more than \( \deg(T, d) \), only \( \deg(T, d) \text{len}(T, d) \) hyperedges can be deleted each time.

Each time the above process is repeated, a hyperedge disjoint from all the previously chosen hyperedges is chosen. So at least
\[
r(T, d) = \frac{2^{\text{len}(T, d)}}{2^{\text{len}(T, d)} - 1}
\]
of the remaining inputs in \( J \) cannot have a computation COMP. That is, \( J \) is reduced by a factor of \( r(T, d) \). Therefore up to step \( T \), we have at most
\[
|J| = \frac{2^{n/2}}{(r(T, d))^{g(n, d, T)}}
\]
inputs that may have a computation COMP. But by Claim 3, this must be greater than or equal to
\[
2^{n/2}/2n^T(2d)^{T^2}.
\]
That is,
\[
\left( \frac{2^{\text{len}(T, d)}}{2^{\text{len}(T, d)} - 1} \right)^{g(n, d, T)} \leq 2n^T(2d)^{T^2},
\]
where \( g(n, d, T) = n/2 \deg(T, d) \text{len}^2(T, d) \).

Now it is not hard to see that for a fixed \( d \) \( T \) cannot be a constant. Taking logarithm on both sides of (1), we obtain,
\[
g(n, d, T) \log \left( \frac{2^{\text{len}(T, d)}}{2^{\text{len}(T, d)} - 1} \right) \leq 1 + T \log n + T^2 \log 2d.
\]
CLAIM 4. There exists a constant $C$ such that for all $X \geq 1,$

$$\log \left( \frac{2^X}{2^X - 1} \right) \geq \frac{C}{2^X}.$$ 

Proof. By a standard calculation. \hfill \blacksquare

By Claim 4,

$$g(n, d, T) \log \left( \frac{2^{\text{len}(T,d)}}{2^{\text{len}(T,d)} - 1} \right) \geq \frac{g(n, d, T)C}{2^{\text{len}(T,d)}}. \quad (3)$$

From (2) and (3), we must have

$$\frac{g(n, d, T)C}{2^{\text{len}(T,d)}} \leq 1 + T \log n + T^2 \log 2d$$

or

$$Cg(n, d, T) \leq (1 + T \log n + T^2 \log 2d) 2^{\text{len}(T,d)}.$$ 

Again taking the logarithm on both sides,

$$\log(Cg(n, d, T)) \leq \text{len}(T, d) + O(\log[T^2 \log n \log d]) \quad (4)$$

Since $\text{len}(T, d) = (2d)^T,$ $\deg(T, d) \leq 2T2^{(2d)^T}(2d)^T,$ and $g(n, d, T) = n/2T2^{(2d)^T}(2d)^{3T},$ the LHS of (4) is greater than or equal to

$$\log n - \log T - (2d)^T - 3T \log 2d - 2. \quad (5)$$

Combining (4) and (5), we must have, for some $C,$

$$\log n \leq 2(2d)^T + O(\log[T^2 \log n \log d]) + \log T + 3T \log 2d + 2$$

$$\leq C(2d)^T \quad (\text{assuming } 2^T > \log \log n)$$

$$T = \Omega(\log \log n/\log d). \quad \blacksquare$$

6. CONCLUDING REMARKS

We have proved separation results between parallel models with ROM. As mentioned in (Vishkin and Wigderson, 1985), input availability can affect the complexity of problems and if we assume that the inputs are given in a ROM, they are available to all the processors and then we can concentrate on the communication among the processors. We also treat nondeterministic models. Many lower bounds for deterministic models
allow in each step an arbitrary amount of local computation (i.e., no shared memory cells accessed) by each processor. The reason for doing so is that the model used satisfies the minimal set of requirements, and the lower bounds still hold without restricting the power of each RAM or requiring some uniformity.

For nondeterministic models, an arbitrary amount of local computation at each basic step may result in an unbounded branching factor (the private computation is a binary tree of unbounded depth). Since any realistic model would have only a fixed number of nondeterministic choices in one step, it definitely makes sense to bound the branching factor by a constant, while still allowing each branch to be arbitrarily long as long as it does not hurt the lower bound. A model with an unbounded branching factor is unrealistic and each processor can, in one step, guess an unbounded number of bits. On such a model, any function can be computed on an NCOMMON(1) without ROM in one step, by having one processor guess all the input in one step and write it into the shared memory, and then any processor whose input is not consistent with this guess objects.

As for deterministic models, an obvious open question is to show that COMMON(f(n)), ARBITRARY(f(n)), and PRIORITY(f(n)) have different powers for any f(n). In a subsequent paper (Li and Yesha, 1986), we partially answer this question by showing that with inputs in ROM, COMMON(n^ε), ARBITRARY(n^ε), and PRIORITY(n^ε) are different for ε < 1. However, these separations are not as tight as the separations for ROM models with one shared-memory cell in Theorems 3.2 and 4.1, or as the separations for models without ROM with n^ε shared memory cells in (Fich et al., 1985).

ACKNOWLEDGMENTS

We thank Eitan Gurari, Xin He, and Tim Long for useful discussions. Thanks also to Chanderjit Bajaj and Marty Marlatt for their help.

RECEIVED October 28, 1985; ACCEPTED April 18, 1986

REFERENCES


Vishkin, U. (1982), Parallel-design space distributed-implementation space (PDDI) general purpose computer, RC 9541, IBM T.J. Watson Research Center, Yorktown Heights, NY.