LOWER BOUNDS FOR THRESHOLD AND SYMMETRIC FUNCTIONS IN PARALLEL COMPUTATION

Yossi Azar*

Computer Science Department
Stanford University
Stanford, CA 94305-2140

Abstract

We consider the family of decision problems of the threshold languages L_g. A threshold language L_g is the set of n bit vectors having at least $g(n)$ "1"s. Using a new technique for controlling the size and structure of a hypergraph by a potential function, we prove lower bounds for these decision problems on a PRIORITY PRAM with m shared memory cells and any polynomial number of processors. The lower bounds are almost tight for the admissible range ($m \leq n^{\epsilon}$). By combining our results with the results of Vishkin and Wigderson and the results of Li and Yesha we are able to show a complexity gap between an m cell PRIORITY PRAM having an exponential (or unlimited) number of processors and one having only a polynomial number. A consequence of our results is that PRIORITY PRAM and ARBITRARY PRAM with m shared memory cells and any given polynomial number of processors have the same power (up to a small factor) for computing symmetric functions.

* Supported by a Weizmann Fellowship and by contract ONR N00014-88-K-0166.

1. Introduction

This paper considers the PRAM model with limited shared memory. A PRAM consists of processors $P(i)$, $i = 1, ..., p$, and shared memory cells $C(i)$, $i = 1, ..., m$, through which the processors communicate. Since we assume that the number of shared memory cells (m) is smaller than the input size (n), there are also n read only input cells (ROM) $X(1), ..., X(n)$. There is a hierarchy among the different PRAM models in accordance with the way they resolve write conflicts. The strongest model is the PRIORITY model, where the minimal index processor (among those attempting to write to the same cell) succeeds. We prove lower bounds for the PRIORITY model with m shared memory cells, denoted by PRIORITY(m). In order to get more powerful lower bounds we assume that each cell in the shared memory can accommodate strings of arbitrary length.

For a given function $g = g(n)$, the threshold language L_g (see [LY2]) is defined by

$$L_g = \{(x_1, .., x_n) | x_i \in \{0, 1\} \ (1 \leq i \leq n) \ \text{ and } \ \sum_{i=1}^{n} x_i \geq g(n)\}$$

By symmetry we can always assume that $g(n) \leq n/2$. The language L_g, where $g(n) = n/2$, is called the MAJORITY language. The complexity of the family of threshold languages has immediate implications on the complexity of symmetric functions.

1.1 Background and Previous Work.

The first to consider this model were Vishkin and Wigderson [VW]. They proved an $\Omega(\sqrt{n/m})$ lower bound for computing MAJORITY ($g = n/2$) on PRIORITY(m). Their lower bound does not depend on the number of processors and is tight for all $m \leq n/\log^2 n$. The upper bound can be easily obtained using $\sqrt{nm}$ processors. For general $g(n)$ their method produces an $\Omega(\sqrt{g(n)/m})$ lower bound. They argue that the case of a single shared memory cell ($m = 1$), or of a constant number, is not only interesting from the theoretical point of view, but is also well founded in practice. For example, the "Ethernet" can be considered as a PRAM with a single shared memory cell. They also point out that in [GGKMRS],[K] and [V] it is implied that the size of the shared memory may determine the hardware feasibility of the parallel machine. More lower bounds for PRAM with small shared memory appear in [B],[FMW],[FRW] and others.

Li and Yesha made more progress on analyzing the complexity of threshold languages in this model. In [LY2], they considered the threshold languages L_g for $g = o(n)$. They noticed that the $\Omega(\sqrt{g})$ lower bound ($m = 1$) can be matched using $\binom{n}{g^{0.5}}$ processors. Since this number is exponential in n, they suggested that it is of interest to find more accurate bounds for smaller number of processors. They present a better lower bound only for the special case where the number of processors is *linear* and $g = o(n^{1/4})$. Specifically, they showed that for $g = o(n^{1/4})$, a PRIORITY(m) with $O(n)$ processors requires depth $\Omega(g/m)$ to recognize L_g. For $m = 1$, a matching upper bound can be easily obtained.

However, the general problem remained open, i.e. to find the complexity of the family of the threshold languages for any polynomial number of processors. Moreover, the lower bound in [LY2] holds only for the subfamily where $g = o(n^{1/4})$. Thus, it is desirable to determine the complexity over the entire range of g. From the fact that the threshold can be found for any g in $O(\sqrt{n})$ steps, even for $m = 1$ and using only $\sqrt{n}$ processors, (see [VW]), it follows that the lower bound of $\Omega(g)$ cannot be obtained for every $g = o(n)$. Hence, it is of interest to know for which range the $\Omega(g)$ lower bound holds and how it changes for bigger g.

1.2 Our Results.

We address all these problems and prove a lower bound for the general case. We assume without loss of generality that $g \leq n/2$, since the complexity of g and $n - g$ is the same because of symmetry. We prove that $\Omega(g/m)$ rounds are necessary for solving the L_g decision problem for any polynomial number of processors and for the admissible range of g. More precisely, we show that for any $\epsilon > 0$, and for any constant $c > 0$, a PRIORITY(m) with $O(n^c)$ processors requires depth $\Omega(\min(\frac{g}{m}, \frac{n^{1/2-\epsilon}}{m}))$. This bound is tight for $m = 1$ and $g \leq n^{1/2-\epsilon}$, since it can easily be obtained by using n/g processors. It is almost tight (up to a factor of $n^{2\epsilon}$) for the remaining range of g and for all $m \leq n^{\epsilon}$. This follows from the existence of a simple algorithm that runs using one shared memory cell in $O(n^{1/2})$ rounds and $n^{1/2}$ processors, and due to the fact that the lower bound is reduced by a factor of at most m for m cell PRAMs, compared to a single shared memory cell.

Our results thus determine the complexity of the family of the threshold problems when a polynomial number of processors is available. Moreover, we conclude that the running time of an algorithm that uses a polynomial number of processors will not be better than the running time of one that uses only a linear number of processors. Our results also show that the complexity of the problem with a polynomial number of processors is different from the complexity in the exponential case. Thus, there is a quadratic gap between the cases. For example, for $m = 1$ and exponential number of processors the complexity is $\Theta(\sqrt{g})$; however, for the polynomial range, we prove that it behaves in the following curious way. It is g for $g \leq n^{1/2}$, and it becomes $n^{1/2}$ for the remaining domain up to a factor of less than n^{ϵ}.

By combining our results with those of [LY2], we show that for any polynomial number of processors and $m \leq n^{\epsilon}$, PRIORITY(m) and ARBITRARY(m) require the same time complexity (up to a factor of at most $n^{2\epsilon}$) for computing any symmetric function. For $m = 1$ we prove that the two models are equivalent, for a large family of symmetric functions. These results are based on the strong connection between computing symmetric functions and threshold decision problems (see [LY2]).

Let us briefly elaborate on the techniques that are used to prove our main lower bound. The main new tool is the use of a potential function on a dynamic hypergraph which changes at every step according to the actions of the processors. The set of vertices of the hypergraph corresponds to the ROM cells. The hyperedges reflect, at each step, the main set of constraints defining the legal inputs at that step. We perform several types of transformations on the hypergraph so as to maintain the following invariant: all the legal inputs have the same history. The construction also guarantees that the set of legal inputs contains at least one input in L_g and at least one which is not. Interestingly, the structure and the size of the hypergraph are controlled by the potential function. These structures also serve to control the information that each processor can deduce from the ROM and the shared memory.

2. The Main Lower bound

Each step of the PRAM computation consists of four phases: each processor $P(i)$ reads from some shared memory cell $C(j)$, reads from some ROM cell $X(k)$, performs a computation and may try to write into some shared memory cell $C(l)$. The actions and the next state of each processor depend on the current state, the values read from the ROM and the shared memory. See [VW] and [LY] for formal definitions.

The strongest model, the PRIORITY model, is considered. Thus, the minimal index processor succeeds in writing when write conflicts occur. Recall that PRIORITY(m) denotes the model with m shared memory cells (width m). Since each round of PRIORITY(m) can be simulated by $2m$ rounds of PRIORITY(1), all the lower bounds of $m = 1$ hold for PRIORITY(m) when divided by $2m$. Thus, we can concentrate in the case where $m = 1$.

We state our main theorem.

The Main Theorem. *Let k be any positive integer. PRIORITY(1) requires $\Omega(g/k)$ rounds to recognize L_g using p processors when the following inequality holds:*

$$p[2^{k+4}g]^{2k+1} \leq n^{k+1} \tag{2.1}$$

Corollary 1. *Let $\epsilon > 0$ and $d = o(\log n)$. Then $\Omega(g/d)$ rounds are needed in order to recognize L_g using $p = O(n^d)$ processors for $g \leq n^{1/2-\epsilon}$.*

Proof: *For $g \leq n^{1/2-\epsilon}$ we can choose $k = Cd$ where C is large enough constant such that (2.1) holds for large enough n.*

An important special case of Corollary 1 is when d is constant.

Corollary 2. *For any polynomial number of processors $\Omega(g)$ rounds are needed to recognize L_g for $g \leq n^{1/2-\epsilon}$.*

A matching upper bound can be easily obtained using a linear number of processors as follows. Associate a processor with each ROM cell. At each step each processor whose input bit is 1 tries to write its identity into the shared memory cell. Then it reads from this cell and checks if it succeeded in writing. If it did succeed, then it halts. An input is in L_g if and only if the algorithm is alive for at least g steps.

Thus, we cannot improve the asymptotic running time using any polynomial number of processors compared to the case of using only a linear number of them. The results also separate, as we already mentioned, between the power of a polynomial number of processors and the power of an exponential number, since the complexity in the latter case is $\Theta(\sqrt{g})$.

Note that the complexity is monotonic (up to a constant factor) in g for $g \leq n/2$ since we can pad the input vector by 1's. we conclude a lower bound for the remaining range of g.

Corollary 3. *For any polynomial number of processors $\Omega(n^{1/2-\epsilon})$ rounds are needed to recognize L_g for $g \geq n^{1/2-\epsilon}$.*

This bound is almost tight as it is easy to find the exact threshold (to sum up the bits) in $O(n^{1/2})$ rounds using $n^{1/2}$ processors as follows. Partition the input into $n^{1/2}$ blocks, each of size $n^{1/2}$, and associate a processor with each block. In the first $n^{1/2}$ rounds each processor sums up the bits in its block. In the next $n^{1/2}$ rounds each processor in turn adds its result to the shared memory cell. This results in the bit sum.

In fact, we can further decrease the n^{ϵ} gap between the upper and lower bound for $g \geq n^{1/2-\epsilon}$. By choosing $k = \Theta(\sqrt{\log n})$ we can obtain from our main theorem an $\Omega(g/k)$ lower bound for $g = n^{1/2}/2^{\Theta(\sqrt{\log n})}$. This yields a gap of $O(\sqrt{\log n})$ for $n^{1/2-\epsilon} \leq g \leq n^{1/2}/2^{\Theta(\sqrt{\log n})}$ and for $g > n^{1/2}/2^{\Theta(\sqrt{\log n})}$ the gap is reduced to at most $2^{\Theta(\sqrt{\log n})}$.

Corollary 1 combined with the result of [VW] yields the following lower bound.

Corollary 4. *For $g \leq n^{1/2-\epsilon}$ and $d = o(\log n)$, $\Omega(g/d + \sqrt{g})$ rounds are needed to recognize L_g using $O(n^d)$ processors.*

This result matches up to a constant factor the [LY2] upper bound of $O(g/d + \sqrt{g})$ using $\binom{n}{d} = O(n^d)$ processors.

Recall that PRIORITY(1) can simulate each step of PRIORITY(m) in at most $2m$ steps. This yields the following corollary.

Corollary 5. *All the above lower bounds divided by $2m$ apply to PRIORITY(m). Thus, for $m \leq n^{\epsilon}$ the bounds are tight up to a factor of at most $n^{2\epsilon}$ (in fact up to at most n^{δ} for any $\delta > \epsilon$).*

3. The proof of the Main Theorem

In this section we prove the main theorem. In the first part the main inductive hypothesis is presented and proved. In the second part the proof of the main theorem is completed using the main inductive hypothesis and some of the transformations defined in the first part.

First let us assume that $p \geq n^{1/2}$, as for the easy case $p \leq n^{1/2}$ the complexity for every g is clearly $\Theta(n/p)$ which is more than what the theorem claims in this case. Let k be an arbitrary positive integer.

We start with some definitions. Define the history of a computation through t steps as a vector $H_1, ..., H_t$, where H_i is the contents of the shared memory at step i. For $t = 0, 1, ...$ the adversary defines collection of inputs I_t on which M has the same history through step t ($I_t \subseteq I_{t-1}$ for $t \geq 1$). Whenever $t < g/k$ then I_t contains one input in L_g and one not in L_g. Thus the algorithm cannot recognize L_g in depth t.

Let $I_0 = \{0,1\}^n$. We will define sets $B_t \subseteq \{1, ..., n\}$, $B_{t-1} \subseteq B_t$. The hypergraph is defined on the vertices $\{1, ..., n\}$. At stage t the edges of the hypergraph will be $F_t = \cup_{i=1}^{k+1} F_t^i$, where F_t^i is a set of subsets (hyperedges), each of size i, of the vertices. Let $B_0 = \phi$ and $F_0 = \phi$.

Intuitively B_t is the set of indices of cells whose input bits are fixed to be 1 at the end of step t. A hyperedge (sometimes referred to as edge) in the hypergraph corresponds to a constraint on the input. The cells, whose indices are the vertices of the hyperedge, cannot all contain 1. In particular, F_t^1 is the set of indices (edges of size 1) of the cells whose bits are fixed to be 0. Furthermore, we do not let any processor see more than k 1's from the input cells (in addition to the 1's in the set B_t which are known to all the processors). This is done by constructing the hypergraph edges (constraints). Let

$$q = \frac{n}{g 2^{k+4}} \ .$$

Define a potential function W on the hypergraph F,

$$W(F) = \sum_{i=1}^{k+1} q^{-(i-1)} |F^i| \ .$$

The main inductive hypothesis for the end of step $t-1$ is the following:

(I1) All the inputs in I_{t-1} have the same history through step $t-1$.

(I2) On all the inputs in I_{t-1} and for any processor $P(j)$ the set $\{i | i \notin B_{t-1}$ and $P(j)$ read a value 1 from $X(i)$ during steps 1 to $t-1\}$ has a cardinality of at most k.

(I3) $|B_{t-1}| \leq k(t-1)$.

(I4) $W(F_{t-1}) \leq q^{-k} \frac{p}{k!} (t-1)^{k+1} + 2^k q(t-1) + (t-1)^2$.

(I5) For any $s < r \leq k+1$ and all j_i, $1 \leq i \leq r$, if $[\{j_1, ..., j_r\} \in F_{t-1}^r$ and for $1 \leq i \leq s$ $j_i \in B_{t-1}]$ then $\{j_{s+1}, ..., j_r\} \in F_{t-1}^{r-s}$.

(I6) $I_{t-1} = \{(x_1, ..., x_n) \in I_0 | \ x_j = 1$ for all j in B_{t-1}, and $x_{j_1} x_{j_2} \cdots x_{j_i} = 0$ for any $1 \leq i \leq k+1$ and all $\{j_1, ..., j_i\} \in F_{t-1}^i\}$.

Clearly the main inductive hypothesis holds for $t-1 = 0$. We have to prove it for t assuming it is true for $t-1$. We start step t by a **constraint-adding** stage. The goal of this stage is to satisfy (I2) for the end of step t. For that we add to F_{t-1}^{k+1} a set A_t of hyperedges, each of size $k+1$. Let

$A_t = \{\{j_1, ..., j_{k+1}\} |$ for $i = 1, ..., k+1$, j_i is not in B_{t-1} and on some input in I_{t-1} some processor read 1 from all the $X(j_i)$ during steps 1 through $t\}$.

We would like to estimate the size of A_t in order to evaluate the change in W. Let $R_t(l)$ be all the sets $\{j_1, ..., j_{k+1}\}$ such that the j_i's are not in B_{t-1} and on some input in I_{t-1} processor $P(l)$ read a value 1 from $X(j_i)$ for all $1 \leq i \leq k+1$ during steps 1 to t.

For $1 \leq t_1 < t_2 < ... < t_k \leq t-1$ let $I^l_{t-1}(t_1, ..., t_k)$ be all the inputs $x \in I_{t-1}$ such that on input x processor $P(l)$ read at step t_i, for all $1 \leq i \leq k$, a value 1 from some ROM location $X(j_i)$, j_i is not in B_{t-1} and all the j_i's are different.

Claim 1. *All the inputs in $I^l_{t-1}(t_1, ..., t_k)$ contribute at most one set to $R_t(l)$.*

Proof: The basic idea of the proof is to prove by inductive on j, $j \leq t$ that during steps 1 to j on all the inputs in $I^l_{t-1}(t_1, ..., t_k)$, the sequence of ROM locations which $P(l)$ has read is the same, and unless $j = t$ their values are also the same. The claim follows easily from this assertion and from (I2).

The assertion is clearly true for $j = 0$. Assume that the assertion holds up to step $j-1$. Thus, all the inputs in $I^l_{t-1}(t_1, ..., t_k)$ have the same history and on all these inputs the same sequence of values was read by $P(l)$ from the ROM. Thus, at step j on these inputs, $P(l)$ will read from the same location (denoted by j^*) from the ROM. It is left to show that $P(l)$ will also read the same value when $j < t$.

We consider four cases. If $j = t_i$ for some $i = 1, .., k$ then by definition the value is always 1. If $j^* \in B_{t-1}$ then by (I6) the value is also necessarily 1. If $j^* = t^*_i$ for some $i = 1, .., k$ where t^*_i is the common (by induction) location in the ROM that $P(l)$ read from at step t_i on all the inputs $I^l_{t-1}(t_1, ..., t_k)$, then clearly the value is 1 for all these inputs. Otherwise, the value has to be 0 as a value 1 would contradict (I2) since $t^*_1, ..., t^*_k, j^*$ would be a sequence of $k+1$ different places not in B_{t-1} that $P(l)$ read a value 1 from. $\square$

Since $A_t \subseteq \cup_{l=1}^{p} R_t(l)$ then Claim 1 implies that

$$|A_t| \leq p\binom{t-1}{k} \leq \frac{p}{k!}(t-1)^k \ .$$

Let G'_t be the hypergraph after this stage. Hence

$$W(G'_t) \leq W(F_{t-1}) + q^{-k}\frac{p}{k!}(t-1)^k \ .$$

We need to remark that at almost any point in the proof the hypergraph might have redundant edges, i.e., edges which correspond to redundant constraints. Specifically, if for some hyperedges X and Y, $X \subset Y$ then we may omit Y. This does not change the legal inputs and does not increase the potential function as well. However, assumption (I5) should be interpreted in the following way. The subset $\{j_{s+1}, ..., j_r\}$ (in (I5)) is not required to be an edge in the hypergraph, but rather some subset of it is required to be an edge.

We define on the hypergraph F the **weighted sunflower transformation** as follows. For a current hypergraph denote by $d_s^r\{j_1, ..., j_s\}$ the number of hyperedges of size r that contain $\{j_1, ..., j_s\}$ as a subset ($s < r$). If for some s, some set $\{j_1, ..., j_s\}$ which is not an edge in the current F satisfies

$$\sum_{r=s+1}^{k+1} q^{-(r-1)} d_s^r\{j_1, ..., j_s\} > q^{-(s-1)}$$

then we add a new edge $\{j_1, ..., j_s\}$ to the current F and omit all the edges that contain this set as a subset. By that we create a current F. We repeat the above transformation on the current hypergraph as much as possible in any arbitrary order. Clearly, the number of hyperedges decreases at each transformation and therefore the process is final. The potential function W on F does not increase during this process, since the potential of a new edge is at most the sum of the potentials of the edges that it has replaced. Let G_t denote the hypergraph at the end of this process. Thus,

$$W(G_t) \leq W(G_t') \leq W(F_{t-1}) + q^{-k}\frac{p}{k!}(t-1)^k .$$

Let $I_t' = \{(x_1, ..., x_n) \in I_0 |\ x_j = 1$ for all j in B_{t-1}, and $x_{j_1}x_{j_2}...x_{j_i} = 0$ for any $1 \leq i \leq k$ and all $\{j_1, ..., j_i\} \in G_t^i\}$.

Clearly $I_t' \subseteq I_{t-1}$ since the constraint-adding stage as well as the weighted sunflower transformations could just restrict the legal inputs. Now there are 2 possible cases; The first is that no processor writes at step t on any input in I_t'. In this easy case we let $B_t = B_{t-1}$. The general case is when there exists a processor that writes at step t on some input in I_t'. Consider the set of processors which write at step t for some input in I_t'. Let $P(l)$ be the minimum index processor in this set. Suppose that $P(l)$ writes at step t on $x \in I_t'$. Let U_t, V_t respectively, be the set of ROM locations from which $P(l)$ has read a value 0, 1 respectively, on input x by step t. Clearly $|U_t| \leq t$. In order to force $P(l)$ to write on all legal inputs (and by that (I1) will hold at the end of step t) we need to fix the bits of V_t to be 1 and of U_t to be 0. We let $B_t = B_{t-1} \cup V_t$; later in the proof (after the spreading transformation) we will force the bits of U_t to be 0 by adding each bit as an edge to the hypergraph. By the definition of B_t and I_t' and by (I2),(I5) (both for $t-1$), we can easily conclude that

$$|B_t - B_{t-1}| \leq k$$

and thus (I3) will hold at the end of step t.

Now the edges (constraints) in G_t^i should be changed in accordance with the new information in order to satisfy (I5). This is called the **spreading transformation**. For any s let $Z = \{j_1, ..., j_s\}$ be a set of any s elements in $B_t - B_{t-1}$. If it is a subset of any edge in G_t^r ($r > s$) then we must create a new edge of the $r-s$ remaining elements, and add it to the hypergraph. Moreover, since

$$\sum_{r=s+1}^{k+1} q^{-(r-1)} d_s^r\{j_1,\ldots,j_s\} \leq q^{-(s-1)}$$

(otherwise, it would contradict the weighted sunflower transformations) then

$$\sum_{r=s+1}^{k+1} q^{-((r-s)-1)} d_s^r\{j_1,\ldots,j_s\} \leq q^{-(s-1)} q^s = q \ .$$

Hence, the total potential of the edges that were created by the set Z is at most q. We perform the spreading transformation simultaneously for all the subsets of $B_t - B_{t-1}$ (at most 2^k subsets), and conclude that the potential function is increased by at most $2^k q$. Define F_t to be the hypergraph after the spreading transformation union with $|U_t| \leq t$ sets, each of size one, of all the individual elements of U_t. Clearly F_t satisfies (I5). Moreover, each of these one element sets adds at most one to the potential function and thus

$$W(F_t) \leq W(G_{t-1}) + 2^k q + t \ .$$

Therefore

$$W(F_t) \leq W(F_{t-1}) + q^{-k}\frac{p}{k!}(t-1)^k + 2^k q + t \ ,$$

which yields (I4) for the end of step t. Finally, we define I_t according to (I6) with t instead of $t-1$.

This completes step t. One can easily verify that the main inductive hypothesis holds for t since each assumption was satisfied at some point in the proof and remained satisfied henceforth.

Proving the lower bound using the main inductive hypothesis. We will prove that the algorithm cannot stop in t steps for $t < g/k$. Fix some $T < g/k$. We first look at the following input; $x_i' = 1$ for $i \in B_T$ and $x_i' = 0$ otherwise. Clearly this input belongs to I_T. By (I3), $|B_T| < g$ and therefore it is not in L_g. Constructing an input in I_T which is also in L_g is more complicated. We start with $x_i'' = 1$ for $i \in B_T$, $x_i'' = 0$ for $i \in F_T^1$. We need to find $g - |B_T| \leq g$ other locations with value 1 that will be consistent with all the constraints of F_T. More precisely, we have to find an **independent set** of size g in the hypergraph, which is a set of vertices of size g such that each edge of the hypergraph contains at least one vertex not in this set. If such an independent set exists we will set the input bits that correspond to those vertices to be 1 (in addition to the input bits of the set B_t), and the remaining vertices to 0. This will define an input in I_T which is in L_g and will complete the proof.

First one can easily check that inequality (2.1) yields (as $p \geq n^{1/2}$) that

$$g \leq n^{1/2}/32$$

and by definition of q ($q = \frac{n}{g2^{k+4}}$) and inequality (2.1)

$$q^{-k}\frac{p}{k!}g^{k+1} \leq n/16 \ .$$

Hence, easy computation shows that for $t \leq g$,

$$W(F_t) \leq n/4 \ .$$

Constructing an independent set consists of g steps. These steps are independent of the actions of the processors after step T. However, for simplicity of notation these pseudo-steps are referred to as steps $t = T+1, ..., T+g$. At each such step $t = T+1, ..., T+g$ we perform the following operations. First the weighted sunflower transformations are performed on the current hypergraph F_{t-1} and this results in an hypergraph G_t. Then a vertex j (not in B_{t-1} or G_t^1) is chosen as will be described later. Let $B_t = B_{t-1} \cup \{j\}$ and set the corresponding input bit to 1. Finally we perform the spreading transformation for this vertex, i.e. the vertex is omitted from each hyperedge containing it, and thus the size of each such hyperedge is decreased by 1.

It is left to show that at every pseudo-step t it is possible to choose a new vertex $\{j\}$ to add to B_{t-1}. That means that for any $T+1 \leq t \leq T+g$ we need to show that there is a vertex whose bit can be set to 1 consistently. Note that (I5) holds for the hypergraph G_t also for these pseudo-steps and therefore the constraints (edges) that contain vertices in B_{t-1} are redundant. Moreover, the edges in $\cup_{i>1}G_t^i$ that contain vertices from G_t^1 are redundant as well. Thus, each vertex j which is not in $B_{t-1} \cup G_t^1$ can be added to B_{t-1} to continue the process since the input $\{x_i = 1,\ i \in B_{t-1} \cup \{j\}$ and $x_i = 0$ otherwise$\}$ is legal in the current hypergraph. Nevertheless, we still have to show that such a vertex j always exists, i.e, $|B_{t-1}| + |G_t^1| < n$. To this end we first observe that

$$|B_{t-1}| = |B_T| + (t-1-T) < g + g \leq 2n^{1/2}/32 \leq n/16 \ .$$

Moreover, the potential function increases by at most q at each of these g pseudo-steps due to the spreading transformation. Since $gq \leq n/16$ and the potential function was at most $n/4$ at the end of step T we conclude that during the g pseudo-steps $W(F_t) \leq n/4 + n/16$. However, the potential of each edge in $F_t^1 \subseteq F_t$ is 1, and each edge in F_t has a non-negative potential. Hence,

$$|G_t^1| \leq |F_t^1| \leq W(F_t) \leq n/4 + n/16$$

for the current hypergraph. Therefore

$$|B_{t-1}| + |G_t^1| \leq n/16 + n/4 + n/16 < n$$

and such a j exists. Hence, for any $T < g/k$ we found two inputs with the same history through the T steps, one of which is in L_g and the other is not. Thus, we cannot recognize L_g in less than g/k steps which completes the proof of the main theorem.

4. The Complexity of Computing Symmetric Functions

Comparing the relative power of models is known to be an important question. It is known that PRIORITY is strictly stronger than ARBITRARY (see [FMW],[FRW] for models without a ROM and [LY],[LY1],[FLRY] for models with a ROM).

The question is whether this is true for symmetric functions. Note that there is a strong connection between the threshold decision problem and computing symmetric functions. In [LY1] it is shown that PRIORITY(1) and ARBITRARY(1), both without ROM, are equivalent for computing symmetric functions of boolean inputs.

Using the results from the previous section with results from [LY2], we extend their results and prove that for any polynomial number of processors, (in fact up to $n^{o(\log n)}$), PRIORITY(m) and ARBITRARY(m), where $m \leq n^{\epsilon}$, both with ROM and the same number of processors, have the same power, up to a factor of at most $n^{2\epsilon}$, for computing any symmetric function. For $m = 1$ we show that the two models are equivalent, up to a constant factor, for a large family of the symmetric functions. In [LY2] it is proved only for linear number of processors and smaller class of symmetric functions or for a large enough number of processors.

For any symmetric function f on n bits we associate a function $\overline{f}$ such that $\overline{f}(i) = j$ whenever $f(\overline{x}) = j$ for $|\overline{x}| = i$ ($|\overline{x}| = \sum_{i=1}^{n} x_i$). The threshold of f was defined in [LY2] by

$$|f| = \min\{h + l | \overline{f} \text{ is a constant on the closed interval } [h, n-l]\} .$$

They showed that a lower bound for L_g is also a lower bound for computing a symmetric function f with threshold $|f| = g$. Thus using the result from the previous section we conclude the following theorem.

Theorem 4.1. *Let f be any symmetric function. For any $\epsilon > 0$ PRIORITY(m) with ROM and polynomial number of processors requires $\Omega(\min(\frac{|f|}{m}, \frac{n^{1/2-\epsilon}}{m}))$ to compute f.*

It is quite easy to design (see [LY2]) an algorithm in ARBITRARY(1) that matches the bound up to a constant factor for $m = 1$ and $|f| \leq n^{1/2-\epsilon}$. Thus for any f and $m \leq n^{\epsilon}$ the bounds are tight up to at most $n^{2\epsilon}$, (in fact up to n^{δ} for any $\delta > \epsilon$). Moreover, we conclude the following theorem.

Theorem 4.2. *Let $\epsilon > 0$ be any number and $m \leq n^{\epsilon}$. For any polynomial number of processors, PRIORITY(m) and ARBITRARY(m), both with ROM and the same number of processors, have the same power for computing all symmetric functions on n bits, up to a factor of at most n^{δ} for any $\delta > \epsilon$. For $m = 1$ and $|f| \leq n^{1/2-\epsilon}$ the two models are equivalent up to a constant factor.*

Theorems 4.1 and 4.2 can be extended in the obvious way to the range of superpolynomial number of processors, i.e. when the number of processors is $O(n^d)$ where $d = o(\log n)$.

Acknowledgement

I would like to thank N. Alon for helpful discussions and suggestions, that encouraged me to improve the original results. I would also like to thank D. Koller for helpful remarks.

References

[Be] P. Beame, Lower bounds in parallel machine computation, Ph.D. Thesis, University of Toronto 1986.

[FLRY] F. Fich, M. Li, R. Ragde, and Y. Yesha, On the power of concurrent-write PRAMs with Read-Only Memory, Information and Control, 83:2 (1989) pp. 234-244.

[FMW] F. Fich, F. Meyer auf der Heide, and A. Wigderson, Lower bounds for parallel random access machines with unbounded shared memory, Advances in Computing Research, 1987 pp. 1-15.

[FRW] F. Fich, P. Ragde, and A. Wigderson, Relation between concurrent write models of parallel computation, SIAM J. Comput. 17:3 (1988) pp. 606-627.

[GGKMRS] A. Gottlieb, R. Grishman, C. Kruskal, K. Mcauliffe, L. Rudolf and M. Snir, The NYU Ultracomputer Designing a MIND shared memory parallel machine, IEEE Trans. Comput. C-32 (1983) pp. 175-189.

[K] D.Kuck, A survey of parallel machine organization and programming, Computing Surveys 9 (1977) pp. 29-52.

[LY] M. Li and Y. Yesha, Separation and lower bounds for ROM and nondeterministic models of parallel computation, Information and Control, 73:2 (1987) pp. 102-128.

[LY1] M. Li and Y. Yesha, New lower bounds for parallel computation, 18th ACM Ann Symp. on Theory of Computing, Berkeley, 1986 pp. 177-187.

[LY2] M. Li and Y. Yesha, Resource bounds for parallel computation of threshold and symmetric functions. J. of Comp. and System Science, to appear.

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

[VW] U. Vishkin and A. Wigderson, Trade-offs between depth and width in parallel computation, SIAM J. Computing, Vol. 14. No 2 (1985), pp. 303-314.