The Parallel Hierarchical Memory Model

Ben H.H. Juurlink and Harry A.G. Wijshoff

High Performance Computing Division, Department of Computer Science
Leiden University, P.O. Box 9512, 2300 RA Leiden, The Netherlands
{benj,harryw}@cs.leidenuniv.nl

Abstract. Modern computer systems usually have a complex memory system consisting of increasingly larger and slower memory. Traditional computer models like the Random Access Machine (RAM) have no concept of memory hierarchy, making it inappropriate for an accurate complexity analysis of algorithms on these types of architectures.
Aggarwal et al. introduced the Hierarchical Memory Model (HMM). In this model, access to memory location x requires $f(x)$ instead of constant time. In a second paper they proposed an extension of the HMM called the Hierarchical Memory Model with Block Transfer (HMBT), in which a block of consecutive locations can be copied in unit time per element after the initial access latency.
This paper introduces two extensions of the HMBT model: the Parallel Hierarchical Memory Model with Block Transfer (P-HMBT), and the pipelined P-HMBT (PP-HMBT). Both models are intended to model memory systems in which data transfers between memory levels may proceed concurrently.
Tight bounds are given for several problems including dot product, matrix transposition and prefix sums. Also, the relationship between the models is examined. It is shown that the HMBT and P-HMBT are both strictly less powerful than the PP-HMBT. It is also shown that the HMBT and P-HMBT are incomparable in strength.
Index Terms — Hierarchical memory, data locality, algorithms.

1 Introduction

In order to keep up with the memory request rate of the central processing unit, modern computer systems usually have a complex memory system consisting of a relatively small amount of registers, followed by one or two levels of cache and main memory. The idea behind using such a hierarchical memory organization is that, by the principle of locality, most of the memory accesses can be made from the fast memory, so that the average access time is closer to the access time of the fast memory. Thus it is important in such an environment to reuse data as much as possible before returning it to slower memory. This property of an algorithm is usually called *temporal locality*. Another form of reuse is *spatial locality*. Spatial locality occurs when consecutive memory locations are accessed. This is important because the time to access one word may be long but several consecutive addresses can be transferred rapidly, since the memory can be organized in banks that allow one clock cycle for each access.

An important aspect for memory hierarchies is parallelism in block transfer, meaning that transfers at different levels may proceed concurrently and may be overlapped with processing. Traditionally, this represents one of the early uses of parallelism in computers. In the next section we introduce the Parallel Hierarchical Memory Model with Block Transfer (P-HMBT), that enables parallelism to be exploited between block transfers.

2 Definition of the Model

The most frequently used model of a computer is the *Random Access Machine* or RAM [3]. It consists of an accumulator, a read-only input tape, a write-only output tape, a program and a memory. The memory consists of a set of memory cells $\underline{A} = \{A_n : n \in I\!N\}$ and each cell is capable of holding an integer of arbitrary size. We will write n for A_n. There is no upper bound on the number of cells that can be used.

In the RAM there is no concept of memory hierarchy; each memory access is assumed to take one unit of time. This may be appropriate in cases where the size of the problem is small enough to fit in the main memory, but in the presence of registers, caches, disks and so on, this model may be inaccurate. In a more realistic model the memory access time is measured by a nondecreasing function $f : \underline{A} \to I\!N$.

In [1] the *Hierarchical Memory Model* (HMM) was proposed. The control is identical to that of the RAM, but access to location x requires $f(x)$ instead of constant time. In the HMM, however, there is no concept of block transfer to utilize spatial locality in algorithms. The *Hierarchical Memory Model with Block Transfer* (HMBT) proposed in [2], is defined as the HMM, but in addition a block copy operation $BC(x, y, l)$ is present. It copies the locations $x + i$ to $y + i$ for $0 \leq i \leq l$ and is valid only if the intervals $[x, x + l]$ and $[y, y + l]$ are disjoint. A block copy operation is assumed to take $\max(f(x), f(y)) + l$ time.

In the HMM and the HMBT, the memory can be viewed as consisting of a hierarchy of levels. A *memory level* is defined as a continuous array of memory cells $A_m \ldots A_n$ such that $f(A_i) = f(A_{i+1})$, for $m \leq i < n$, and $f(A_{m-1}) < f(A_m)$ and $f(A_n) < f(A_{n+1})$. We are now able to give a description of the P-HMBT model.

Definition 1. The concept of control, costs and block transfer of the P-HMBT are identical to the HMBT, except for the requirement that block transfers cannot cross level boundaries and can only take place between successive levels. In addition, block transfers are allowed to proceed in parallel between levels. Specifically, if $C_1, C_2, \ldots, C_m$ is a sequence of block transfers such that $C_i = BC(x_i, y_i, l_i)$, then these block transfers can be executed in parallel provided that all x_i's are in different levels. Such a parallel block transfer is written as $C_1 \| C_2 \| \ldots \| C_m$ and is assumed to take $\max_i\ t_i$ time, where $t_i = \max(f(x_i), f(y_i)) + l_i$.

We take $f(x) = \lfloor \log x \rfloor$ as a model for realistic memory systems. The choice for $\lfloor \log x \rfloor$ is technical, but we believe that it resembles real life well enough to give realistic qualitative predictions. In particular, it resembles memory systems consisting of increasingly larger and slower memory.

The function $f(x) = \lfloor \log x \rfloor$ partitions the memory into levels of continuous locations such that the i-th level $L(i)$ contains the memory locations $[2^i, 2^{i+1}-1]$. We will assume that concurrent block moves like $BC(x, y, l) \parallel BC(y, z, l)$ where x, y and z are in successive levels, cause no memory conflicts, since we can alternate between the odd numbered and even numbered levels. This will increase the running time by at most a constant factor. Alternatively, we can double the memory available to an algorithm by moving all data to the next lower level and use the upper half of each level as a buffer, without increasing the running time by more than a constant factor.

3 Bounds for Simple Problems

Consider the following problem which is called the *touch problem*. Initially, n inputs $a_1, a_2, \ldots, a_n$ are stored in memory locations $1, 2, \ldots, n$. An algorithm is said to touch the input a_i if, at some time during the execution of the algorithm, a_i is stored in location 1 of the hierarchical memory. The problem is to touch all inputs. This yields a lowerbound for various algorithms in which the entire input has to be examined. Examples of such algorithms include merging, searching a random sequence and computing the dot product of two vectors.

Theorem 2. *Any algorithm that touches n inputs on the P-HMBT with access cost function $f(x) = \lfloor \log(x) \rfloor$ requires $\Omega(n \log^*(n))$ time.*

Proof. We prove the theorem by using the same technique as in [2]. Let $b_i(t)$ be the least k such that a_i has been stored in memory location k during one of the first t steps of the computation. Define the *potential* at step t as $\phi(t) = \sum_{i=1}^{n} \log^*(b_i(t))$. The initial potential is $\phi(0) = \sum_{i=1}^{n} \log^*(i)$. When the algorithm terminates after T steps, each input has been stored in memory location 1 during one of the first T steps of the computation. Hence the final potential is $\phi(T) = \sum_{i=1}^{n} \log^*(1) = 0$. Thus $\phi(0) - \phi(T) = \sum_{i=1}^{n} \log^*(i) = \Omega(n \log^*(n))$.

We prove the theorem by showing that any move that takes time L on the P-HMBT decreases the potential by at most $O(L)$. From this the theorem follows. W.l.o.g. assume that $L = k + 2^k$ for some integer k. In L time units we can copy one location from level L of the hierarchical memory to level $L-1$, two locations from level $L-1$ to level $L-2$, ..., and $L-k = 2^k$ locations from level $k+1$ to level k. On level i, $1 \leq i \leq k$, we can only copy 2^{i-1} locations to the next memory level in this move (the size of the next higher level). The decrease in potential is maximal if we copy the last locations of level i to the first locations of level $i-1$. In other words, the following move decreases the potential as much as possible in L time units.

- On level i, $1 \leq i \leq k$, the memory locations $2^i + 2^{i-1} + j$, $0 \leq j \leq 2^{i-1} - 1$, are copied to locations $2^{i-1} + j$.

– On level i, $k+1 \le i \le L$, the locations $2^{i+1}-1-j$, $0 \le j \le L-i$, are copied to locations $2^{i-1}+j$.

It follows that the decrease in potential due to this move is at most

$$\begin{aligned}\Delta\phi &= \sum_{i=1}^{k}\sum_{j=0}^{2^{i-1}-1}(\log^*(2^i+2^{i-1}+j)-\log^*(2^{i-1}+j)) \\ &\quad + \sum_{i=k+1}^{L}\sum_{j=0}^{L-i}(\log^*(2^{i+1}-1-j)-\log^*(2^{i-1}+j)) \\ &\le \sum_{i=1}^{k}\sum_{j=0}^{2^{i-1}-1}(\log^*(2^{i+1})-\log^*(2^{i-1})) + \sum_{i=k+1}^{L}\sum_{j=0}^{L-i}(\log^*(2^{i+1})-\log^*(2^{i-1})).\end{aligned}$$

It can be shown that both terms are bounded by $O(L)$ [8]. □

Remark When describing algorithms for the P-HMBT model, it is generally assumed that the input is located in memory locations $1, 2, \ldots, n$ or in memory locations $n, n+1, \ldots, 2n-1$. Both initial situations are equivalent in the following sense. We can go from one situation to the other in time $O(n)$ by repeatedly moving each level concurrently to the next higher (lower) level. So, if an algorithm takes at least $\Omega(n)$ time, this does not affect the running time by more than a constant factor.

Theorem 3. *The following problems can be solved in time $O(n\log^*(n))$ on the P-HMBT with access cost function $f(x) = \lfloor\log(x)\rfloor$.*

(i) The touch problem,
(ii) searching a random sequence of size n, and
(iii) computing the dot product of two vectors of length n each.

Proof. (i) Initially, the $n = 2^m$ inputs are stored in locations $n, n+1, \ldots, 2n-1$, i.e. at level $\log(n)$ of the hierarchical memory. Treat it as a sequence of $n/\log(n)$ blocks B_i, $1 \le i \le n/\log(n)$, of length $\log(n)$ each. First move B_1 to level $\log(n)-1$ of the hierarchical memory and then concurrently move B_1 to level $\log(n)-2$ and B_2 to level $\log(n)-1$. Continue in this pipelined fashion until B_1 reaches level $\log\log(n)$ and repeat recursively on that block. Then move the next block from level $\log\log(n)+1$ to level $\log\log(n)$ and concurrently move each block on level i, $\log\log(n)+2 \le i \le \log(n)$, to the next higher level. Continue until all blocks have been processed. The running time $T(n)$ of this algorithm obeys the recurrence relation

$$\begin{aligned}T(n) &= n/\log(n)T(\log(n)) + O((\log(n)-\log\log(n))\log(n)) + O(n) \\ &= n/\log(n)T(\log(n)) + O(n),\end{aligned}$$

which solves to $T(n) = O(n\log^*(n))$. The same idea applies to *(ii)* and *(iii)*. □

4 Relationship Between the Models

In this section we explore the relationship between the HMBT model and the P-HMBT model. First the following definition is needed. A machine model M is said to be hierarchically weaker than M' ($M \preceq M'$) if each problem that can be solved on model M in time T can also be solved on model M' in time $O(T)$. We want to determine if HMBT $\preceq$ P-HMBT or P-HMBT $\preceq$ HMBT.

Theorem 4. *HMBT $\not\preceq$ P-HMBT.*

Proof. It can be shown that search operations require at least $\Omega(\log^2(n))$ time on the P-HMBT model, since at least one location must be accessed and access to location x requires $\sum_{i=0}^{\log x} i = \Theta(\log^2(x))$ time on this model. This holds regardless of the data structure used. On the HMBT, however, search operations can be performed in time $O(\log^2(n)/\log\log(n))$ by using a B-tree [4] where the buckets have size $\log(n)$. Each bucket is organized as a perfectly balanced binary search tree that is stored in continuous memory locations. To search the tree $O(\log(n)/\log\log(n))$ buckets must be searched. This can be done in time $O((\log\log(n))^2)$ per bucket by first moving each bucket to the first $O(\log(n))$ memory locations. Each bucket can be moved to fast memory in time $O(\log(n))$. Thus, search operations on this data structure can be performed in time

$$O(\log(n)/\log\log(n)((\log\log(n))^2 + \log(n))) = O(\log^2(n)/\log\log(n))$$

on the HMBT model. □

Lemma 5. *Any algorithm that transposes a $\sqrt{n}\times\sqrt{n}$ matrix requires $\Omega(n\log^*(n))$ time on a HMBT machine.*

Proof. The proof of this lemma can be found in [2]. □

Lemma 6. *A $\sqrt{n}\times\sqrt{n}$ matrix A can be transposed on a P-HMBT machine in optimal $O(n)$ time.*

Proof. Assume the input is in memory locations $n, n+1, \ldots, 2n-1$. The algorithm works essentially as follows. First, $A(1, 1 \ldots \sqrt{n})$ is moved to memory locations $2n, 2n+1, \ldots, 2n+\sqrt{n}-1$. Then, $A(1, 2 \ldots \sqrt{n})$ is moved to locations $4n, 4n+1, \ldots, 4n+\sqrt{n}-2$ and, concurrently, $A(2, 1 \ldots \sqrt{n})$ is moved to $2n+1, 2n+2, \ldots, 2n+\sqrt{n}$. After that, $A(1, 3 \ldots \sqrt{n})$ is moved to $8n, 8n+1, \ldots, 8n+\sqrt{n}-3$, $A(2, 2 \ldots \sqrt{n})$ to $4n+1, 4n+2, \ldots, 4n+\sqrt{n}-1$ and $A(3, 1 \ldots \sqrt{n})$ is moved to $2n+2, 2n+3, \ldots, 2n+\sqrt{n}+1$. The algorithm proceeds in this way until $A(i, j)$ is in memory location $2^j \cdot n + i - 1$ for $1 \le i, j \le \sqrt{n}$. Then the transposed matrix is transferred back to level $\log(n)$ of the hierarchical memory. A more formal description of this algorithm is given below.

for $i \leftarrow 0$ **to** $\sqrt{n}-1$ **do**
do steps (a) and (b) concurrently
(a) $BC(n+i\sqrt{n}, 2n+i, \sqrt{n}-1)$

(b) **for** $j \leftarrow 1$ **to** i **do concurrently**
$BC(2^j \cdot n + i - j + 1, 2^{j+1} \cdot n + i - j, \sqrt{n} - 1 - j)$
od
for $i \leftarrow 1$ **to** $\sqrt{n} - 1$ **do**
for $j \leftarrow i$ **to** $\sqrt{n} - 1$ **do concurrently**
$BC(2^j \cdot n + \sqrt{n} - (j - i), 2^{j+1} \cdot n + \sqrt{n} - (j - i) - 1, \sqrt{n} - 1 - j)$
od
for $i \leftarrow 0$ **to** $\sqrt{n} - 1$ **do**
do steps (a) and (b) concurrently
(a) $BC(2n, n + i\sqrt{n}, \sqrt{n} - 1)$
(b) **for** $j \leftarrow 1$ **to** $\sqrt{n} - 1 - i$ **do concurrently**
$BC(2^{j+1} \cdot n, 2^j \cdot n, \sqrt{n} - 1)$
od

The costs are $O(\sqrt{n})$ per block move. Thus the time for all these moves is bounded by $O(n)$. □

Theorem 7. *P-HMBT* $\not\preceq$ *HMBT.*

Proof. Follows directly from lemma 5 and lemma 6. □

Until now we assumed that only one block move per memory level was allowed in each step, and a next block copy operation could not start until all previous block moves had been completed. A more powerful model is obtained if these restrictions are removed. In the *pipelined* P-HMBT model (PP-HMBT), the only condition that must be satisfied is that during each cycle only one data word can be transferred from level i to level $i - 1$ or vice versa. As an example, consider the following sequence of block moves.

Initiate at cycle 0: $BC(y, x, 2) \parallel BC(z, y, 2)$
Initiate at cycle 3: $BC(y, x + 3, 2)$

In figure 1, the content of each level during each cycle is illustrated.

This example shows two important differences between the pipelined P-HMBT model and the "straight" P-HMBT model. First, a block copy operation involving a memory cell x may be started although the previous block move involving cell x has not been completed. Secondly, it is not required for a block move $BC(x, y, l)$ that all memory cells $[y, y + l]$ are free by the time the transmission of $[x, x + l]$ is started. Instead, if a data word d occupies a cell x at cycle k, then that cell must become free at cycle $k - 1$.

Obviously, P-HMBT $\preceq$ PP-HMBT, since any computation for the P-HMBT model is a legal computation on the PP-HMBT model and can be executed on the PP-HMBT in the same amount of time.

Theorem 8. *Any algorithm that takes time T on a HMBT machine, can be modified into an algorithm that takes time $O(T)$ on a pipelined P-HMBT machine. Hence, HMBT* $\preceq$ *PP-HMBT.*

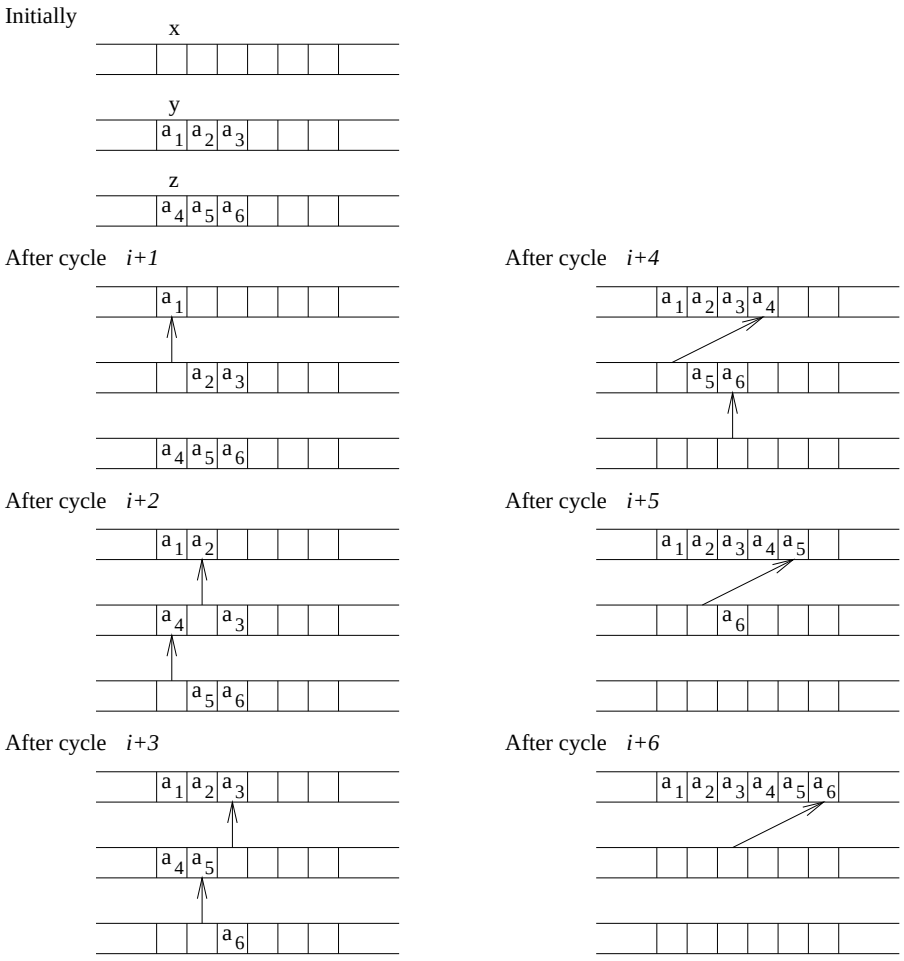


Fig. 1. Pipelining data transfers on the PP-HMBT machine.

Proof. Consider a block move $BC(x, y, l)$ and let $x > y$ (the case $x < y$ is treated similarly). The time for this move on the HMBT is $\lfloor \log(x) \rfloor + l$. This move is simulated as follows. Let $L(i)$ denote the i-th level of the hierarchical memory. The contents of memory locations $2^i, \ldots, 2^{i+1} - 1$ of the HMBT machine are stored in the lower part of $L(i + 2)$ in the pipelined P-HMBT. The upper part of each level is used as a buffer. Let $\alpha = \lfloor \log(y) \rfloor$ and let $\gamma = \lfloor \log(x) \rfloor$. Then, the block $B = [x, x + l]$ is contained in memory locations $2^\gamma \ldots 2^{\gamma+2} - 1$ of the HMBT machine (since $l < x$), and therefore in $L(\gamma + 2) \cup L(\gamma + 3)$. Copy the part of B that is in $L(\gamma + 2)$ to the buffer of $L(\gamma + 1)$ and then back to the lowest numbered locations in the buffer of $L(\gamma + 2)$. Move the part of B that is in $L(\gamma + 3)$ to the buffer of $L(\gamma + 2)$ to obtain a continuous block to be moved.

The time for these moves is $O(\lfloor \log(x) \rfloor + l)$.

Partition B into subblocks B_i, $\alpha + 2 \leq i \leq \gamma + 2$, where B_i contains the data words that have to go to $L(i)$. Move $B_{\alpha+2}, B_{\alpha+3}, \ldots, B_{\gamma+2}$ to the buffer of $L(\gamma+1)$ and *pipeline* $B_{\alpha+2}, B_{\alpha+3}, \ldots, B_{\gamma+1}$ to the buffer of $L(\gamma)$ (pipeline means that the transmission starts as soon as the first data word arrives). Continue in this way until each block B_i is in the buffer of $L(i-1)$. The time for these moves is bounded by

$$\lfloor \log(x) \rfloor + \lfloor \log(x) \rfloor - \lfloor \log(y) \rfloor + l - 1 = O(\lfloor \log(x) \rfloor + l).$$

Then, simultaneously copy each B_i back to $L(i)$. The time for this is also bounded by $O(\lfloor \log(x) \rfloor + l)$. Summing up over these three steps yields the desired bound. □

Theorem 9. *Touching n inputs, initially stored in locations $1, 2, \ldots, n$ of memory, can be done in time $O(n)$ on the pipelined P-HMBT model.*

Proof. At each computing cycle, a number of transmissions are initiated. The transmissions from $L(i)$ to $L(i-1)$ are initiated at cycles $k2^{i-1}$ for $k = 0, 1, 2, \ldots$. At cycles $k2^{i-1}$ with k is even the block copy operations

$$BC(2^i, 2^{i-1}, 2^{i-1} - 1), \text{ for } 1 \leq i \leq \log n,$$

are initiated (the lower half of each level is moved to the next higher level). When k is odd, the block copy operations

$$BC(2^i + 2^{i-1}, 2^{i-1}, 2^{i-1} - 1), \text{ for } 1 \leq i \leq \log n,$$

are initiated (the upper half of each level is moved to the next higher level). It is easily verified that with this sequence of block moves, exactly one data word is transferred between each pair of successive levels during each cycle. From this observation the time bound follows. It remains to be shown that no data is overwritten during execution. Consider two successive levels $L(i)$ and $L(i+1)$. The transmissions from $L(i+1)$ to $L(i)$ are initiated at cycles $k2^i$; location $2^{i+1} + j$, $0 \leq j \leq 2^i - 1$, is transferred to location $2^i + j$ during cycle $k2^i + i + 1 + j$ and location $2^{i+1} + j$, $2^i \leq j \leq 2^{i+1} - 1$, is transferred to location j during cycle $k2^i + i + 1 + j$. Likewise, the transmissions from $L(i)$ to $L(i-1)$ are initiated at cycles $k2^{i-1}$; location $2^i + j$, $0 \leq j \leq 2^i - 1$, is moved during cycle $k2^{i-1} + i + j$. In other words, if a cell x at level $L(i+1)$ is copied to a cell y at level $L(i)$ during cycle k, then that cell became free during cycle $k - 1$. □

This theorem shows that the PP-HMBT model is strictly more powerful than the HMBT model and the P-HMBT model.

5 More Results for the P-HMBT and PP-HMBT Model

Prefix Sums Let $\oplus$ be an associative operator over a domain D. The prefix sums problem is defined as follows. Given an array $A = (a_1, a_2, \ldots, a_n)$ of size n. It is required to compute the sums

$$s_i = a_1 \oplus a_2 \oplus \cdots \oplus a_i$$

for $i = 1, \ldots, n$. The prefix sums can be computed in $O(n)$ time on the RAM. From theorem 2, a lowerbound of $\Omega(n \log^*(n))$ follows for computing prefix sums on the P-HMBT. We will show that the prefix sums problem can be solved in time $O(n(\log^*(n))^2)$ on the P-HMBT.

Suppose that initially the array A is stored in memory locations $n, n+1, \ldots, 2n-1$, i.e. at level $\log(n)$ of the hierarchical memory. As in theorem 2, treat it as a sequence of $n/\log(n)$ blocks of length $\log(n)$ each.

1. Move these blocks, using the pipelining strategy, to higher levels until the first block reaches level $\log\log(n)$. Repeat recursively on that block and continue until all blocks have been processed.
   At this point we have computed the sums

   $$m_{ij} = a_{(i-1)\log n+1} \oplus a_{(i-1)\log n+2} \oplus \cdots \oplus a_{(i-1)\log n+j}$$

   for $1 \leq i \leq n/\log(n)$ and $1 \leq j \leq \log(n)$.
2. Copy $m_{i,\log n}$, $1 \leq i \leq n/\log(n)$, to memory locations $n/2 + i - 1$, i.e. to level $\log n - 1$. Call this array $Q = (q_1, q_2, \ldots, q_{n/\log n})$.
3. Move the array Q to level $\log(n/\log(n)) = \log(n) - \log\log(n)$ and call prefix sums recursively.
4. Add q_i, $1 \leq i \leq n/\log(n) - 1$ to $m_{i+1,j}$, $1 \leq j \leq \log(n)$. It can be verified that adding a scalar to an array of size m can be done in $O(m \log^*(m))$ time by adapting the algorithm given in the proof of theorem 2.

The running time $T(n)$ of this algorithm fulfills the recurrence

$$T(n) = n/\log(n)T(\log(n)) + T(n/\log(n)) + O(n\log^*(n)).$$

Since $T(2) = \textit{constant}$, it can be verified that $T(n) = O(n(\log^*(n))^2)$.

On a PP-HMBT machine, prefix sums can be performed in time $O(n)$ as follows. In location 1, the partial sum s_i is stored. The inputs are stored in the lower half of each level and are "fed" to location 1 using a similar scheme as for the touch problem. The outputs (i.e. the partial sums s_i for $1 \leq i \leq n$) are directed along the upper half of each level. It can be shown that this algorithm can be made to run in time $O(n)$ on the PP-HMBT model.

List Ranking Another important problem is *list ranking*. Given a linked list of n elements. The list ranking problem is to determine for each element its distance to the end of the list. Usually, the linked list is represented by two arrays C (the contents or data array) and S (the successor array). The first element in the list (the head) is stored in $C(1)$, and for $1 \leq i \leq n-1$, if the i-th element is stored in $C(j)$, then the $(i+1)$-th element is stored in $C(S(j))$. The list ranking problem can be solved in time $O(n)$ on the RAM.

Many parallel algorithms for the list ranking problem are *shortcut* based, meaning that the essential step in the execution is

$$S(i) \leftarrow S(S(i)).$$

The difficulty is to avoid that two consecutive elements are shortcut in parallel. A solution to this problem has been presented by Kruskal *et al.* [9]. We implement their algorithm where the numbers of processors p equals $\sqrt{n}$. The algorithm partitions the nodes of the list into a $\sqrt{n} \times \sqrt{n}$ array. Each processor is assigned one row of the array. The processors visit the columns of the array synchronously. If the successor of the node visited is not in the same column, the node is compacted with its successor, the node is marked and its successor is removed from the list (marked as deleted). This ensures that no two consecutive elements $C(i)$ and $C(S(i))$ are shortcut simultaneously. This procedure is repeated with the transpose of the array for the nodes that are not yet marked. In this step no further conflicts will occur, because if a node $C(i)$ is not yet marked, then $C(S(i))$ is in the same row as $C(i)$. After finishing this step at most $2/3n + O(1)$ nodes remain, because if a node has not been compacted with one of its successors, then both of its neighbors must have been compacted. Then, the remaining nodes are packed into an array of size $2/3n + O(1)$. This is done by doing a prefix sums with 0 assigned to the nodes that are marked deleted and 1 assigned to the nodes that are not deleted. The whole procedure is then applied recursively to the remaining nodes.

On the P-HMBT model this algorithm is implemented as follows. Assume that each node in the list is represented by a quadruple $V(i) = \langle C(i), S(i), M(i), D(i)\rangle$, where $C(i)$ and $S(i)$ have the usual meaning and where $M(i)$ ($D(i)$) is a single bit indicating whether a node is marked (deleted). Assume also that each quadruple exactly fits into one memory cell (this affects the running time only by a constant factor). Initially, the array V is stored in level $L(\log(n))$ of the hierarchical memory. The algorithm proceeds as follows.

1. Partition V into $\sqrt{n}$ blocks B_i, $1 \leq i \leq \sqrt{n}$, of size $\sqrt{n}$ each. For each of these blocks, repeat step (2).
2. Treat a block B_i as $\sqrt{n}/\log(n)$ blocks E_j of size $\log(n)$ each. For each of these blocks perform the following steps.
   (a) Move the block to level $L(\log\log(n))$ and copy the successor of each element to the array $NEXT$ located at level $L(\log(n) - 1)$.
   (b) Move the array $NEXT$ to level $L(\log\log(n) + 1)$ and compact each node with its successor provided that the successor is not in the same block B_i.

(c) Copy E_j and $NEXT$ back to level $L(\log(n))$.

3. Repeat steps (1) – (2) with the transpose of V.
4. Contract the remaining nodes into an array of size $2/3n + O(1)$. This is accomplished by doing a prefix sums on the nodes with 0 assigned to the nodes that are marked deleted and 1 assigned to the nodes that are not deleted.
5. Solve the resulting smaller list recursively.
6. Extend the solution to all elements of the original list.

This algorithm can be made to run in time

$$T(n) = T(2/3n) + O(n \log(n)),$$

which solves to $T(n) = O(n \log(n))$.

On a PP-HMBT machine, list ranking can be performed in optimal time $O(n)$ as follows. As for the P-HMBT, the input is divided into $\sqrt{n}$ blocks B_i of size $\sqrt{n}$ each, and each block is divided into $\sqrt{n}/\log n$ subblocks E_j of size $\log n$. Each block E_j is pipelined to location 1 of the hierarchical memory and each node is paired with its successor provided that the successor is not in the same block B_i. The cost per block are $O(\log n)$, i.e. in constant time per element. Again the input is permuted and these steps are repeated, which also requires $O(n)$ time. This implies that the running time $T(n)$ obeys the recurrence relation

$$T(n) = T(2/3n) + O(n) \ ,$$

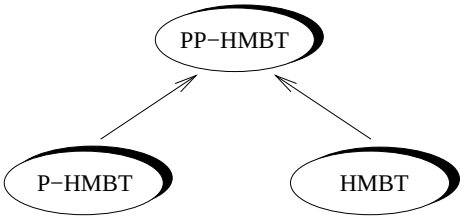
which solves to $T(n) = O(n)$.

For all problems considered in this paper, the running time on the PP-HMBT equals the RAM time. Whether this holds in general or for a certain class of problems, needs further investigation.

6 Summary and Concluding Remarks

In this paper we introduced two models for hierarchical memory that enables parallelism to be exploited between block transfers, meaning that transfers at different levels may proceed concurrently. Algorithms are given for, e.g., the touch problem, prefix sums and list ranking. Efficient algorithms in these models exhibit both temporal and spatial locality.

We examined the relationship between the proposed models P-HMBT and PP-HMBT and we showed that there is a partial ordering on these models and the HMBT proposed in [2]. This partial ordering adheres to the following diagram.



Finally, bounds are given for two important problems, namely prefix sums and list ranking. We noticed that parallel algorithmic techniques can also be used to obtain efficient algorithms for the P-HMBT model. This may not come as a surprise since many parallel algorithms have the following generic scheme (top-down design).

- Divide the problem at hand into p subproblems each taking approximately equal amount of time.
- Solve each subproblem in parallel.
- Combine the results.

Several directions for future research exist. For example, in existing computer systems the block size (cache line size, page size) at each level is usually fixed and the performance is greatly affected by this size [10]. It would be important to understand the behavior of algorithms in this setting. Also, implications for memory organizations in multiprocessor architectures will have to be investigated [5, 7].

Acknowledgement We want to thank Peter Knijnenburg for several useful suggestions.

References

1. A. Aggarwal, B. Alpern, A.K. Chandra, and M. Snir. A Model for Hierarchical Memory. In *19-th Annual ACM Symposium on Theory of Computing*, pages 305–314, May 1987.
2. A. Aggarwal, A.K. Chandra, and M. Snir. Hierarchical Memory with Block Transfer. In *28-th Symposium on Foundations of Computer Science*, pages 204–216, October 1987.
3. A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
4. A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.
5. A. Chin. Complexity Models for All-Purpose Parallel Computation. In A. Gibbons and P. Spirakis, editors, *Lectures on parallel computation*, chapter 14. Cambridge University Press, 1993.
6. J.W. Hong and H.T. Kung. I/O Complexity: The Red-Blue Pebble Game. In *Proc. of the 13-th Annual ACM Symp. on Theory of Computing*, pages 326–333, May 1981.
7. B.H.H. Juurlink and H.A.G. Wijshoff. Experiences with a Model for Parallel Computation. In *12th Annual ACM Symposium on Principles of Distributed Computing*, pages 87–96, August 1993.
8. B.H.H. Juurlink and H.A.G. Wijshoff. The Parallel Hierarchical Memory Model. Technical Report 93-33, Leiden University, 1993.
9. C.P. Kruskal, R. Rudolph, and M. Snir. The Power of Parallel Prefix. *IEEE Trans. on Comp.*, C-34(10), October 1985.
10. Alan Smith. Cache Memories. *ACM Computing Surveys*, 14(3), September 1982.