# Algorithms for Parallel Memory II: Hierarchical Multilevel Memories

Jeffrey Scott Vitter[1]
Elizabeth A.M. Shriver[2]

**Technical Report No. CS-92-05**
August 1992

[1]Department of Computer Science, Box 1910, Brown University, Providence, RI 02912
[2]Courant Institute New York University, New York, 10012

# Algorithms for Parallel Memory II: Hierarchical Multilevel Memories*

*Jeffrey Scott Vitter*†

Dept. of Computer Science
Brown University
Providence, R. I. 02912–1910

*Elizabeth A. M. Shriver*‡

Courant Institute
New York University
New York, N. Y. 10012

August 27, 1992

**Abstract**

In this paper we introduce parallel versions of two hierarchical memory models and give optimal algorithms in these models for sorting, FFT, and matrix multiplication. In our parallel models, there are $P$ memory hierarchies operating simultaneously; communication among the hierarchies takes place at a base memory level. Our optimal sorting algorithm is randomized and is based upon the probabilistic partitioning technique developed in the companion paper for optimal disk sorting in a two-level memory with parallel block transfer. The probability of using $\ell$ times the optimal running time is exponentially small in $\ell(\log \ell) \log P$.

*A summarized version of this research was presented at the *22nd Annual ACM Symposium on Theory of Computing*, Baltimore, MD, May 1990.

†Support was provided in part by National Science Foundation Presidential Young Investigator Award grants CCR–8906419 and CCR–9047466 with matching funds from IBM, by NSF research grants DCR–8403613 and CCR–9007851, by Army Research Office grant DAAL03–91–G–0035, and by the Office of Naval Research and the Defense Advanced Research Projects Agency under contract N00014–91–J–4052 ARPA order 8225.

‡This work was done in part while the author was at Brown University supported by a Bellcore graduate fellowship and at Bellcore.

# Contents

# 1 Introduction

Large-scale computer systems contain many levels of memory—ranging from very small but very fast registers, to successively larger but slower memories, such as multiple levels of cache, primary memory, magnetic disks, and archival storage. An elegant hierarchical memory model was introduced by Aggarwal, Alpern, Chandra, and Snir [1] and further developed by Aggarwal, Chandra, and Snir [2] to take into account block transfer. All computation takes place in the central processing unit (CPU), one instruction per unit time. Access to memory takes a varying amount of time, depending on how low in the memory hierarchy the memory access is. Optimal bounds are obtained in [1,2] for several sorting and matrix-related problems.

In this paper we investigate the capabilities of *parallel* memory hierarchies. In the next section, we define two uniform memory models, each consisting of $P$ hierarchical memories connected together at their base levels. The $P$ hierarchical memories can be either of the two types [1,2] mentioned earlier.

For each model, we develop matching upper and lower bounds for the problems of sorting, FFT, and matrix multiplication, which are defined in Section 3. The algorithms that realize the optimal bounds for sorting are applications of the optimal disk sorting algorithm developed in the companion paper [12] for a two-level memory model with parallel block transfer. We apply the partitioning technique of [12] to the one-hierarchy sorting algorithms of [1,2]. Intuitively, the hierarchical algorithms are optimal because the internal processing in the corresponding two-level algorithms is efficient. The main results are given in Section 4 and are proved in Sections 5–10. Conclusions and open problems are discussed in Section 11.

# 2 Parallel Hierarchical Memory Models

A *hierarchical memory model* is a uniform model consisting of memory whose locations take different amounts of time to access. The basic unit of transfer in the hierarchical memory model HMM [1] is the record; access to location $x$ takes time $f(x)$. The BT model [2] represents a notion of block transfer applied to HMM; in the BT model, access to the $t+1$ records at locations $x-t$, $x-t+1$, ..., $x$ takes time $f(x)+t$. Typical access cost functions are $f(x) = \log x$ and $f(x) = x^{\alpha}$, for some $\alpha > 0$. A model similar to the BT model that allows pipelined access to memory in $O(\log n)$ time was developed independently by Luccio and Pagli [5].

We can think of a memory hierarchy as being organized into discrete levels, as shown in Figure 1 for HMM; for each $k \geq 1$, level $k$ contains the $2^{k-1}$ locations at addresses $2^{k-1}$, $2^{k-1}+1$, ..., $2^k - 1$. We restrict our attention to *well-behaved* access costs $f(x)$, in which $f(x)$ is nondecreasing and there are constants $c$ and $x_0$ such that $f(2x) \leq cf(x)$, for all $x \geq x_0$. For such $f(x)$, access to any location on level $k$ takes $\Theta(f(2^k))$ time. The access cost functions $f(x) = \log x$ and $f(x) = x^{\alpha}$, for some $\alpha > 0$, are well-behaved.[1] For example, if $f(x) = \log x$, access to any location on level $k$ takes time $\approx k$.

Both the HMM and BT hierarchical memory models can be augmented to allow parallel data transfer. One possibility is to have a discretized hierarchy, in which each memory

[1]For simplicity of notation, we use $\log x$, where $x \geq 1$, to denote the quantity $\max\{1, \log_2 x\}$.

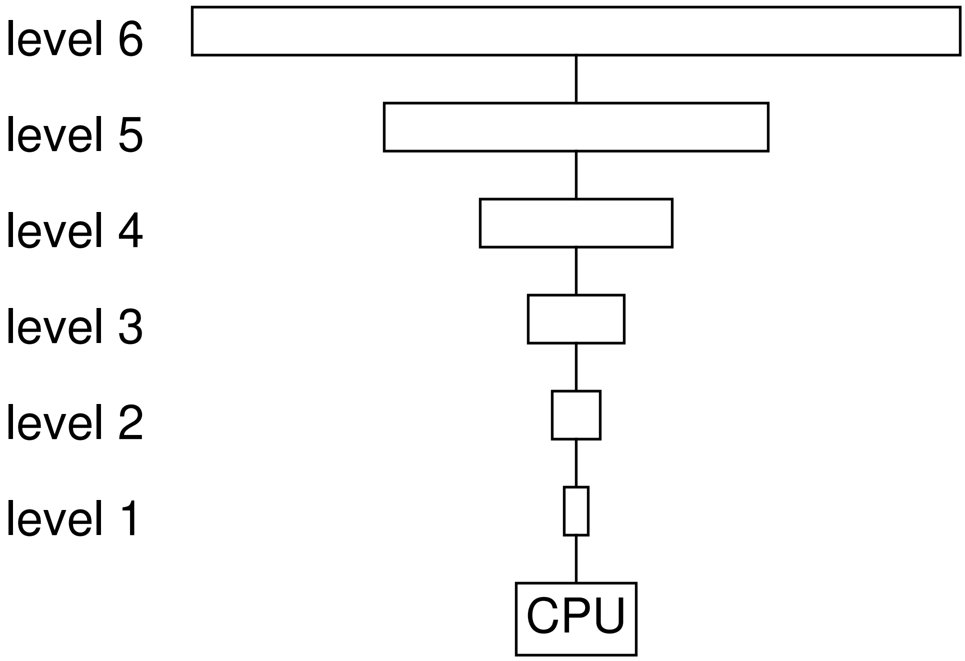


Figure 1: The HMM hierarchical memory model [1].

component is connected to $P$ larger but slower memory components at the next level. A cleaner and more feasible extension, which we adopt in this paper, is to have $P$ separate memories connected together at the base level of each hierarchy. Our model is pictured in Figure 2.

More specifically, we assume that the $P$ hierarchies can each function independently. Communication between hierarchies takes place at the *base memory level* (level 1), which consists of location 1 from each of the $P$ hierarchies. We assume that the $P$ base memory level locations are interconnected via a network such as a hypercube or cube-connected cycles so that the $P$ records in the base memory level can be sorted in $O(\log P)$ time (perhaps via a randomized algorithm [10]) and so that two $\sqrt{P/2} \times \sqrt{P/2}$ matrices stored in the base memory level can be multiplied in $O(\sqrt{P}\,)$ time. We denote by P-HMM and P-BT the $P$-hierarchy variants of the hierarchical memory models HMM and BT, as described above.

We shall refer to the $P$ locations, one per hierarchy, at the same relative position in each of the $P$ hierarchies as a *track*, by analogy with the two-level disk model [12]. The $i$th track, for $i \geq 1$, consists of location $i$ from each of the $P$ hierarchies. In this terminology, the base memory level is the track at location 1. The *global memory locations* (which refer collectively to the $P$ hierarchies combined) are numbered track by track. That is, the global memory locations in track 1 are numbered 1, 2, $\ldots$, $P$; the global memory locations in track 2 are numbered $P+1$, $P+2$, $\ldots$, $2P$; and so on.

# 3 Problem Definitions

The following definitions of sorting, FFT, and standard matrix multiplication are essentially those of the companion paper [12].

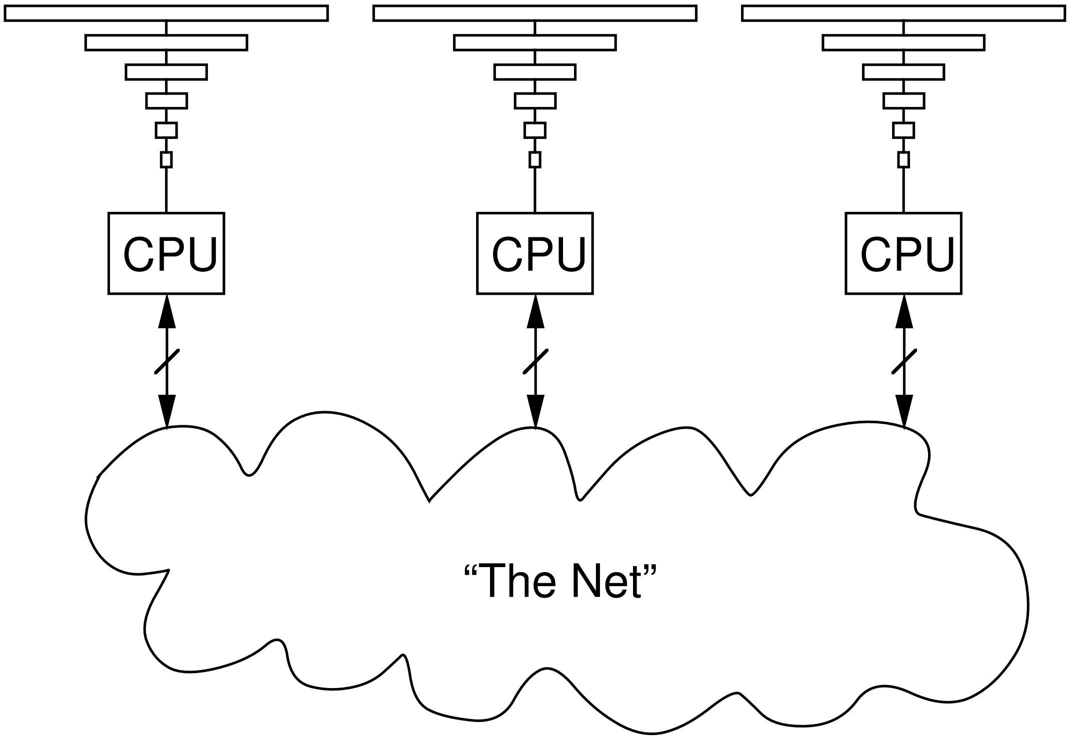


Figure 2: A parallel hierarchical memory. The $P$ individual memory hierarchies can be either of type HMM or of type BT. The $P$ CPUs can communicate among one another via the network; we assume, for example, that the $P$ records stored in their accumulators can be sorted in $O(\log P)$ time.

**Sorting**

Problem Instance: The $N$ records are stored in the first $N$ global memory locations.

Goal: The $N$ records are stored in sorted non-decreasing order in the first $N$ global memory locations.

**Fast Fourier Transform (FFT)**

Problem Instance: Let $N$ be a power of 2. The $N$ records are stored in the first $N$ global memory locations.

Goal: The $N$ output nodes of the FFT digraph are "pebbled" (as explained below), and the $N$ records are stored in the first $N$ global memory locations.

The FFT digraph consists of $\log N + 1$ columns each containing $N$ nodes; column 0 contains the $N$ input nodes, and column $\log N$ contains the $N$ output nodes. Each non-input node has indegree 2, and each non-output node has outdegree 2. We shall denote the $i$th node $(0 \leq i \leq N-1)$ in column $j$ $(0 \leq j \leq \log N)$ in the FFT digraph by $n_{i,j}$. For $j \geq 1$ the two predecessors to node $n_{i,j}$ are nodes $n_{i,j-1}$ and $n_{i \oplus 2^{j-1}, j-1}$, where $\oplus$ denotes the exclusive-or operation on the binary representations. (Note that nodes $n_{i,j}$ and $n_{i \oplus 2^{j-1}, j}$ each have the same two predecessors).

The $i$th node in each column corresponds to record $R_i$. We are allowed to pebble node $n_{i,j}$ if its two predecessors $n_{i,j-1}$ and $n_{i\oplus 2^{j-1},j-1}$ have already been pebbled; the cost of the pebbling is the cost of accessing the records $R_i$ and $R_{i\oplus 2^{j-1}}$ corresponding to the two predecessors. Intuitively, the FFT problem can be phrased as the problem of pumping the records up and down the hierarchies in a way that permits the computation implied by the FFT digraph.

**Standard Matrix Multiplication**

Problem Instance: The elements of two $k \times k$ matrices, $A$ and $B$, where $2k^2 = N$, are each stored in the first $N$ global memory locations.

Goal: The product $C = A \times B$, formed by the standard matrix multiplication algorithm that uses $O(k^3)$ arithmetic operations, is stored in the first $N/2$ global memory locations.

# 4 Main Results

The fundamental problem that arises in trying to take full advantage of parallel transfer in these models is how to distribute records among the $P$ memory hierarchies so that each hierarchy is kept "busy." We shall show later how the randomized distribution sort algorithm of [12] for a two-level memory model with parallel block transfer can be used as a basic building block to get optimal sorting algorithms for the hierarchical models. The lower bounds for P-HMM and P-BT are based upon the approach used in [1] and [2].

**Theorem 1** *In the P-HMM model, the time for sorting and the FFT is*

$$\Theta\left(\frac{N}{P}\log N \log\left(\frac{\log N}{\log P}\right)\right) \qquad \text{if } f(x) = \log x;$$

$$\Theta\left(\left(\frac{N}{P}\right)^{\alpha+1} + \frac{N}{P}\log N\right) \qquad \text{if } f(x) = x^{\alpha},\ \ \alpha > 0.$$

*The upper bound for sorting is given by a randomized algorithm for the first two cases; the probability of using more than $\ell$ times the optimal number of I/Os falls off exponentially in $\ell(\log \ell)\log P$. In the sorting lower bound for the $f(x) = x^{\alpha}$ case, the $(N/P)\log N$ term requires the comparison model of computation. The time for multiplying two $k \times k$ matrices together using the standard algorithm is*

$$\Theta\left(\frac{k^3}{P}\right) \qquad \text{if } f(x) = \log x;$$

$$\Theta\left(\frac{k^3}{P}\right) \qquad \text{if } f(x) = x^{\alpha},\ \ 0 < \alpha < \tfrac{1}{2};$$

$$\Theta\left(\frac{k^3}{P^{3/2}}\log k + \frac{k^3}{P}\right) \qquad \text{if } f(x) = x^{\alpha},\ \ \alpha = \tfrac{1}{2};$$

$$\Theta\left(\left(\frac{k^2}{P}\right)^{\alpha+1} + \frac{k^3}{P}\right) \qquad \text{if } f(x) = x^{\alpha},\ \ \alpha > \tfrac{1}{2}.$$

**Theorem 2** *In the P-BT model, the time for sorting and the FFT is*

$$\begin{array}{ll} \Theta\left(\frac{N}{P}\log N\right) & \text{if } f(x)=\log x; \\ \Theta\left(\frac{N}{P}\log N\right) & \text{if } f(x)=x^{\alpha},\ \ 0<\alpha<1; \\ \Theta\left(\frac{N}{P}\log N\log\frac{N}{P}\right) & \text{if } f(x)=x^{\alpha},\ \ \alpha=1; \\ \Theta\left(\left(\frac{N}{P}\right)^{\alpha}+\frac{N}{P}\log\frac{N}{P}\log P\right) & \text{if } f(x)=x^{\alpha},\ \ \alpha>1. \end{array}$$

*The upper bounds for the first two cases of sorting are given by a randomized algorithm; the probability of using more than $\ell$ times the optimal number of I/Os falls off exponentially in $\ell(\log\ell)\log P$. The $(N/P)\log N$ terms in the sorting lower bounds require the comparison model of computation. The time for multiplying two $k\times k$ matrices together using the standard algorithm is*

$$\begin{array}{ll} \Theta\left(\frac{k^3}{P}\right) & \text{if } f(x)=\log x; \\ \Theta\left(\frac{k^3}{P}\right) & \text{if } f(x)=x^{\alpha},\ \ 0<\alpha<\frac{3}{2}; \\ \Theta\left(\frac{k^3}{P^{3/2}}\log k+\frac{k^3}{P}\right) & \text{if } f(x)=x^{\alpha},\ \ \alpha=\frac{3}{2}; \\ \Theta\left(\left(\frac{k^2}{P}\right)^{\alpha}\right) & \text{if } f(x)=x^{\alpha},\ \ \alpha>\frac{3}{2}. \end{array}$$

In Sections 5–10 we prove Theorems 1 and 2. In the process we also develop optimal algorithms in the P-HMM and P-BT models for matrix addition, the "touch" problem, and the other so-called "simple" problems in [1,2]. Our techniques can be extended to get optimal algorithms for other problems of [1,2,12], such as searching, generating permutations, and permutation networks.

Our optimal P-HMM and P-BT algorithms for sorting and FFT are applications of the optimal algorithms of [12] for the two-level model with parallel block transfer, applied to the HMM and BT algorithms given in [1,2]. The optimality of the resulting P-HMM and P-BT algorithms reflects the fact that the internal processing done by the two-level algorithms on which they are based is very efficient, both sequentially and in parallel.

# 5 Sorting in P-HMM

In this section, we derive the matching upper and lower bounds for sorting in the P-HMM model given in Theorem 1. We prove that the randomized distribution sort algorithm we develop is simultaneously optimal for all well-behaved access cost functions, as defined in Section 2.

The sorting algorithm is a modified version of the optimal distribution sort algorithm for the one-hierarchy HMM [1]. For simplicity, we assume that records have distinct key values;

this assumption is satisfied, for example, if we append to the key field of each record the original global memory location of the record. Distribution sort works by forming a set of $S-1$ partitioning elements $b_1$, $b_2$, ..., $b_{S-1}$, for some $S \geq 2$, and breaking up the file into the $S$ buckets defined by the partitioning elements. The $j$th bucket consists of all the records $R$ in the file whose key values are in the range

$$b_{j-1} < key(R) \leq b_j,$$

where for convenience we define $b_0 = -\infty$ and $b_S = +\infty$. Each bucket is then sorted recursively. The final sorted order is the concatenation of the $S$ sorted buckets.

The key component of our P-HMM algorithm is the partitioning technique of [12], which we use to spread the records in each bucket evenly among the $P$ memory hierarchies so that the next level of recursion can proceed optimally. The partitioning technique is actually two techniques—Phase 1 and Phase 2—each with its own range of applicability. In Phase 1, when $N \geq P^{3/2}/\ln P$, a randomized approach akin to hashing is used to distribute the records evenly among the hierarchies. One intuition why it works well is that in hashing when the load factor is large enough (at least logarithmic in the number of slots in the hash table), the items are evenly distributed; by that, we mean that the largest-populated slot has roughly the same number of items as an average-sized slot. However, when $N < P^{3/2}/\ln P$, the distribution is no longer even. In this case, we use the Phase 2 partitioning technique, motivated by routing on the FFT. Both Phase 1 and Phase 2 work with overwhelming probability in their respective ranges of applicability.

First we develop some useful notation akin to that of [12], but simplified for our purpose: *Hierarchy striping* is a programming technique in which the $P$ hierarchies are coordinated so that at any given time the memory locations accessed by the $P$ hierarchies form a track. Hierarchy striping has the effect of making the parallel hierarchies act like a single hierarchy in which $P$ records can be stored at each location.

We maintain the pointers $last\_write_{j,k}$ and $next\_write_k$, for $1 \leq j \leq S$, $1 \leq k \leq P$, to keep track of the $S$ buckets formed in Phase 1 of the algorithm below. The pointer $last\_write_{j,k}$ points to the last location in the $k$th hierarchy written to by the $j$th bucket. The pointer $next\_write_k$ points to the next unwritten location in the $k$th hierarchy.

The final carryover we use from [12] is a simplified notion of *diagonal*, for use in Phase 2, when $N < P^{3/2}/\ln P$. For simplicity of exposition, let us assume that $N$ and $P$ are powers of 2. Every diagonal contains $P^2/N$ records from each of the $N/P$ tracks of the hierarchical memory. The $i$th diagonal consists of the following $P$ records: the $k$th track, for $1 \leq k \leq N/P$, contributes the $P^2/N$ locations

$$\left((k+i-2) \bmod \frac{N}{P}\right)\frac{P^2}{N}+1, \quad \left((k+i-2) \bmod \frac{N}{P}\right)\frac{P^2}{N}+2,$$
$$\ldots, \quad \left((k+i-2) \bmod \frac{N}{P}\right)\frac{P^2}{N}+\frac{P^2}{N}.$$

The sorting algorithm works as follows:

1. We assume without loss of generality that the $N$ records are situated in level $\lceil \log(N/P) \rceil + 1$ on the $P$ hierarchies. If $N \leq P$, we sort the file in the base memory level. Otherwise we do the following steps:

2. We subdivide the file of $N$ records into $t = \lceil \min\{\sqrt{N}, N/P\} \rceil$ subsets, $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_t$, each of size $\lfloor N/t \rfloor$ or of size $\lfloor N/t \rfloor + 1$. We sort each $\mathcal{G}_i$ recursively, after bringing its records to level $\lceil \log(N/Pt) \rceil + 1$ of the hierarchical memory.

3. [Find partitioning elements.] We determine the number $S$ of partitions and choose the $S-1$ partitioning elements as follows: Let us define

$$x = \begin{cases} \dfrac{\sqrt{N}}{\ln N} & \text{if } P^2 \le N; \\ \dfrac{\sqrt{P}}{\ln^2 N} & \text{if } P^{3/2}/\ln P \le N < P^2; \\ \dfrac{2N}{P} & \text{if } P < N < P^{3/2}/\ln P. \end{cases}$$

   We form a set $A$ of at least $N/\log N$ elements consisting of the the key value of every $\lfloor \log N \rfloor$th record from each $\mathcal{G}_i$. We sort $A$ using two-way merge sort with hierarchy striping. We set $S$ to be $\lfloor |A| / \lfloor |A|/x \rfloor \rfloor + 1 \approx x$. We define the $j$th partitioning element $b_j$, for each $1 \le j \le S-1$, to be the $j\lfloor |A|/x \rfloor$th smallest element of $A$. We move the partitioning elements to level $\lceil \log(S/P) \rceil + 1$.

4. [Phase 1 or Phase 2?] If $N \ge P^{3/2}/\ln P$, we do Step 5 corresponding to Phase 1; otherwise we do Steps 6 and 7, corresponding to Phase 2.

5. [Phase 1.] For each $1 \le i \le t$ in sequence, we partition $\mathcal{G}_i$ by processing it in sorted order. We move $P$ records at a time to the base memory level and determine which bucket each record belongs to by merging in the partitioning elements. The partitioning elements can be processed $P$ elements at a time, so that the merging proceeds optimally; when the next track of partitioning elements is needed, it is read into the base memory level. We then randomly scramble the $P$ records and write the records back to level $\lceil \log(N/P) \rceil + 1$. If the record written in the $k$th hierarchy belongs to the $j$th bucket $\mathcal{S}_j$, we update the pointers $\mathit{last\_write}_{j,k}$ and $\mathit{next\_write}_k$, which are stored in hierarchy $k$, so that all the records of each bucket are linked together in the hierarchy, which is necessary for the next recursive application of the algorithm. We then proceed to Step 8.

6. [Phase 2—Pass 1.] We scramble the $N$ records, $P$ at a time, by reading the file to the base memory level, track by track, randomly permuting the $P$ records there, and writing them back to level $\lceil \log(N/P) \rceil + 1$.

7. [Phase 2—Pass 2.] We move $P$ records at a time to the base memory level, one diagonal at a time, as defined above. For each diagonal of $P$ records, we partition the records into buckets by sorting the $P$ records and the partitioning elements. (The number of partitioning elements is $2N/P \le P$, for $P \ge 4$.) We write the $P$ records back to level $\lceil \log(N/P) \rceil + 1$, cycling through the buckets; if a bucket is empty, a dummy record is written. Let $d$ be the least common multiple of $P$ and $S$. After every $d/S$ cycles of $P$ writes, we skip over the next hierarchy before beginning the next cycle of $P$ writes.

8. [Sort recursively.] For each $1 \le j \le S$ in sequence, we sort the $j$th bucket $\mathcal{S}_j$ recursively, after bringing the records of $\mathcal{S}_j$ to level $\lceil \log(N/SP) \rceil + 1$ of the hierarchical memory.

(With high probability, the records in each bucket are distributed evenly among the $P$ hierarchies, and thus each bucket can be accessed in $O((N/SP)f(N/P))$ time.) The sorted list of $N$ records consists of the concatenated sorted buckets.

The value of $x$ in Step 3, which determines the number of partitioning elements, is chosen so that the partitioning analysis from the companion paper [12] can be modified to show that the records with high probability are distributed evenly among the buckets.

## 5.1 Logarithmic Access Cost

Let us first consider the case $f(x) = \log x$ of Theorem 1.

**Theorem 3** *The time used by the above algorithm to sort $N \geq P$ records in the P-HMM model with $f(x) = \log x$ is*

$$O\left(\frac{N}{P}\log N \log\left(\frac{\log N}{\log P}\right)\right)$$

*with overwhelming probability. In particular, the probability that the number of I/Os used is more than $\ell$ times the average is exponentially small in $\ell(\log \ell)\log P$.*

*Proof*: Let $T(N)$ be the time used by this algorithm to sort a file of $N$ records. For $N \geq P^2$, the time needed in Step 2 to subdivide the set of $N$ records, move the records to the correct level, and sort the $\lceil\sqrt{N}\rceil$ subsets $\mathcal{G}_i$ is

$$\sqrt{N}\,T(\sqrt{N}\,) + O\left(\frac{N}{P}\log\frac{N}{P}\right).$$

The execution time for the two-way merge sort used in Step 3 to sort $n$ elements is $O((n/P)\log n(\log(n/P) + \log P))) = O((n/P)\log^2 n)$. Since $n \approx N/\log N$, the resulting time to find the $S-1$ partitioning elements is $O((N/P)\log N)$. The method of choosing the partitioning elements guarantees that the size $N_j$ of the $j$th bucket is at most $2N/(S-1)$, for each $1 \leq j \leq S$; the proof is along the lines of Lemmas 3 and 4 in the companion paper [12]. The time needed to partition the file in Steps 5, 6, and 7 is $O((N/P)\log(N/P) + (N/P)\log P) = O((N/P)\log N)$. The analysis in the companion paper [12] can be modified to show that with high probability the records in each bucket are distributed evenly over the $P$ hierarchies, so that the time for sorting the buckets recursively in Step 8 is with high probability

$$\sum_{1\leq j\leq S} T(N_j) + O\left(\frac{N}{P}\log\frac{N}{P}\right),$$

where $\sum_{1\leq j\leq S} N_j = N$ and $N_j \leq 2N/S$ for each $1 \leq j \leq S$. Hence, for $N \geq P^2$, with high probability we have

$$T(N) = \sqrt{N}\,T(\sqrt{N}\,) + \sum_{1\leq j\leq \lceil\sqrt{N}/\ln N\rceil} T(N_j) + O\left(\frac{N}{P}\log N\right).$$

If $N < P^2$, there will be at most two more applications of Phase 1 and one application of Phase 2, each phase taking $O((N/P)\log N)$ time with high probability. The remaining subfiles will have size at most $P$ and can be sorted in the base memory level in time $O((N/P)\log P)$ time. This yields as desired the time bound

$$T(N) = O\left(\frac{N}{P}\log N \log\left(\frac{\log N}{\log P}\right)\right)$$

with high probability. The probability bounds quoted in Theorem 3 follow from those in [12]. □

The following lower bound matches the the algorithm's running time in Theorem 3, and thus the algorithm is optimal. It is interesting to note that this lower bound, as well as several others in this paper, ignores the cost of the network communication and considers only the cost of memory access.

**Theorem 4** *The time required to sort $N \geq P$ records in the P-HMM model with $f(x) = \log x$ is*

$$\Omega\left(\frac{N}{P}\log N \log\left(\frac{\log N}{\log P}\right)\right).$$

*Proof*: Let $A$ be a sorting algorithm that is optimal in the P-HMM model. Let us define the "sequential time" of $A$ to be the sum of its time costs for each of the $P$ hierarchies; the sequential time of $A$ is at most $P$ times its running time. Following the approach in [1], we can imagine superimposing onto the P-HMM-type hierarchical memory a sequence of two-level memories. For each $M$ in the range $P \leq M < N$, we superimpose on the P-HMM an internal memory of size $M$ and one infinite-sized disk.

By [3], the I/O complexity of sorting $N$ records with one disk, no blocking, and an internal memory of size $M$ is

$$T_M(N) = \Omega\left(\frac{N\log N}{\log M} - M\right). \tag{1}$$

The "$-M$" term permits $M$ records to reside initially in the internal memory. In each individual hierarchy, every transfer done by $A$ that corresponds to an I/O with respect to an internal memory of size $M$ contributes

$$\delta f\left(\frac{M}{P}\right) = f\left(\frac{M+1}{P}\right) - f\left(\frac{M}{P}\right)$$

to its sequential time. In other words, if we let $T_f(N)$ denote the sequential time for $A$, we have

$$T_f(N) \geq \sum_{P\leq M<N} \delta f\left(\frac{M}{P}\right) T_M(N). \tag{2}$$

For $f(x) = \log x$, we have $\delta f(M/P) = \log((M+1)/M) = \Theta(1/M)$. Plugging this and (1) into (2) we get

$$T_f(N) = \Omega\left(\sum_{P\leq M<N}\left(\frac{N\log N}{M\log M} - 1\right)\right) = \Omega\left(N\log N\log\left(\frac{\log N}{\log P}\right)\right).$$

Dividing the sequential time $T_f(N)$ by $P$ gives us the desired lower bound. □

## 5.2 Other Access Costs

First we show the lower bound corresponding to case $f(x) = x^\alpha$, for $\alpha > 0$, of Theorem 1:

**Theorem 5** *The time required to sort $N \geq P$ records in the P-HMM model with $f(x) = x^\alpha$, for $\alpha > 0$, is*

$$\Omega\left(\left(\frac{N}{P}\right)^{\alpha+1} + \frac{N}{P}\log N\right).$$

*The $(N/P)\log N$ term depends on using the comparison model of computation.*

*Proof*: We apply the same approach as in Theorem 4, except that we use $f(x) = x^\alpha$. Substituting $\delta f(M/P) = \Theta(M^{\alpha-1}/P^\alpha)$ and (1) into (2), we get

$$T_f(N) = \Omega\left(\sum_{P \leq M < N}\left(\frac{M^{\alpha-1}N\log N}{P^\alpha \log M} - \left(\frac{M}{P}\right)^\alpha\right)\right) = \Omega\left(\frac{N^{\alpha+1}}{P^\alpha}\right).$$

Dividing the sequential time $T_f(N)$ by $P$ gives us the first term of the desired lower bound. The second term follows from the $N\log N$ lower bound for sorting in the comparison model of computation. □

The next theorem shows that the sorting algorithm given earlier in Section 5 is *uniformly optimal* (in the language of [1]) in that it is optimal for all well-behaved access costs $f(x)$, as defined in Section 2. In particular, it follows that the sorting algorithm's running time for the case $f(x) = x^\alpha$, where $\alpha > 0$, meets the lower bound of Theorem 5 and is therefore optimal. The following theorem, when combined with Theorem 4, also gives an alternate proof of Theorem 3.

**Theorem 6** *The algorithm given at the beginning of Section 5 is optimal for all well-behaved access costs $f(x)$.*

*Proof*: We use a parallelized version of the approach of [1], combined with the lower bound strategy of Theorems 4 and 5. Let $T_{M,P}(N)$ be the average number of I/O steps done by the sorting algorithm with respect to an internal memory of size $M$, where we allow each hierarchy to move simultaneously, in a single I/O step, a record between internal memory and external memory. From the algorithm, for $N \geq M^2$, we get the recurrence

$$T_{M,P}(N) = \sqrt{N}\, T_{M,P}(\sqrt{N}\,) + \sum_{1 \leq j \leq \sqrt{N}/\log N} T_{M,P}(N_j) + \frac{N}{P},$$

with high probability, where $\sum_{1 \leq j \leq S} N_j = N$ and $N_j \leq 2N/S$, for $1 \leq j \leq S$. When $N < M^2$, we have $T_{M,P}(N) = O(N/P)$ with high probability. This recurrence can be solved by iteration to get

$$T_{M,P}(N) = O\left(\frac{N}{P}\frac{\log N}{\log M}\right) \tag{3}$$

with high probability, which by (1) is within an $O(M/P)$ additive term of optimal. The communication time used by the algorithm in the base memory level is

$$O\left(\frac{N}{P}\frac{\log N}{\log P}\log P\right) = O\left(\frac{N}{P}\log N\right). \tag{4}$$

Putting together these last two facts, we find that the extra time used by the algorithm over and above the lower bound resulting from (2) is

$$
\begin{aligned}
& O\left(\frac{N}{P}\log N + \sum_{P \le M < N} \frac{M}{P}\delta f\left(\frac{M}{P}\right)\right) \\
& \qquad = O\left(\frac{N}{P}\log N + \frac{N}{P} f\left(\frac{N}{P}\right) - f(1) - \frac{1}{P}\sum_{P \le M < N} f\left(\frac{M+1}{P}\right)\right) \\
& \qquad = O\left(\frac{N}{P}\log N + \frac{N}{P} f\left(\frac{N}{P}\right)\right) \qquad (5)
\end{aligned}
$$

with high probability. The first term in (5) corresponds to the lower bound that arises from the comparison model of computation. The second term in (5) is the time to "touch" all the records in the file (that is, bring all the records at least once to the base memory level) when the access cost $f(x)$ is well-behaved, and thus it is dominated by the lower bound resulting from (2). It follows that the sorting algorithm is optimal. □

# 6 Sorting in P-BT

In this section we show the matching upper and lower bounds quoted in Theorem 2 in Section 4 for sorting in the P-BT model. The following useful lemma is a parallel version of a theorem in [2].

**Lemma 1** *The time to "touch" a list of $n \ge P$ elements or to merge two sorted lists of $n \ge P$ elements in the P-BT model is*

$$
\begin{array}{ll}
O\left(\frac{n}{P}\left(\log^* \frac{n}{P} + \log P\right)\right) & \textit{if } f(x) = \log x; \\
O\left(\frac{n}{P}\left(\log\log \frac{n}{P} + \log P\right)\right) & \textit{if } f(x) = x^\alpha,\ \ 0 < \alpha < 1; \\
O\left(\frac{n}{P}\log n\right) & \textit{if } f(x) = x^\alpha,\ \ \alpha = 1; \\
O\left(\left(\frac{n}{P}\right)^\alpha + \frac{n}{P}\log P\right) & \textit{if } f(x) = x^\alpha,\ \ \alpha > 1.
\end{array}
$$

*Proof Sketch*: Touching the elements is a special case of merging, so we restrict our attention to the merging bound. The lists are stored on the $P$ hierarchies in such a manner that they are striped across the tracks. We merge the two lists one track at a time, accessing all $P$ hierarchies. To do the merging, we use $3P$ stacks, three stacks per hierarchy. A stack can be maintained in each individual hierarchy with an amortized cost per operation of

$$
\begin{array}{ll}
O\left(\log^* \dfrac{n}{P}\right) & \text{if } f(x) = \log x; \\
O\left(\log\log \dfrac{n}{P}\right) & \text{if } f(x) = x^\alpha,\ \ 0 < \alpha < 1; \\
O\left(\log \dfrac{n}{P}\right) & \text{if } f(x) = x^\alpha,\ \ \alpha = 1; \\
O\left(\left(\dfrac{n}{P}\right)^{\alpha-1}\right) & \text{if } f(x) = x^\alpha,\ \ \alpha > 1,
\end{array}
$$

where $n$ is the number of operations [2]. The cost of merging two lists of $P$ elements in base memory is $\log P$. □

## 6.1 Access Cost $f(x) = x^\alpha$, where $\frac{1}{2} \le \alpha < 1$

Let us first consider the access cost function $f(x) = x^\alpha$, where $\frac{1}{2} \le \alpha < 1$. We can access the levels within our hierarchies optimally if we read and write $(x/P)^\alpha$ elements at a time when we access global location $x$.

Our optimal P-BT sorting algorithm is a modified version of the one-hierarchy algorithm presented in [2]. The key component of the algorithm is our use of the partitioning technique of [12] to spread the records in each bucket evenly among the $P$ hierarchies. For brevity we present only the portion of the P-BT sorting algorithm whose description differs from the P-HMM sorting algorithm of Section 5.

2. Same as Step 2 of the P-HMM sorting algorithm, except that we subdivide the file of $N$ records into $t = \lceil (N/P)^{1-\alpha} \rceil$ subsets, so that each subset will contain about $P(N/P)^\alpha$ records, for purposes of optimal transfer.

3. [Find partitioning elements.] We determine the number $S$ of partitions and choose the $S-1$ partitioning elements as follows: Let us define

$$
x = \begin{cases}
\dfrac{(NP)^\alpha}{\ln(NP)} & \text{if } P^{(1+\alpha)/(1-\alpha)} \le N; \\[2ex]
\dfrac{\sqrt{P}}{\ln^2 N} & \text{if } P^{3/2}/\ln P \le N < P^{(1+\alpha)/(1-\alpha)}; \\[2ex]
\dfrac{2N}{P} & \text{if } P < N < P^{3/2}/\ln P.
\end{cases}
$$

   We form a set $A$ of at least $N/\log N$ elements consisting of the key value of every $\lfloor \log N \rfloor$th record from each $\mathcal{G}_i$. We form $A$ by touching each $\mathcal{G}_i$ and accumulating the desired key values recursively, with each hierarchy processing up to $\lceil N/P \rceil$ elements of the $N$ elements in the file. We sort $A$ using two-way merge sort by recursively applying the algorithm presented in Lemma 1. We set $S$ to be $\lfloor |A| / \lfloor |A|/x \rfloor \rfloor + 1 \approx x$. We define the $j$th partitioning element $b_j$ to be the $j \lfloor |A|/x \rfloor$th smallest element of $A$. We move the partitioning elements to level $\lceil \log(S/P) \rceil + 1$.

8. [Reposition buckets and sort recursively.] For each $1 \le j \le S$ in sequence, we reposition the elements within the $j$th bucket $\mathcal{S}_j$ so that they are stored in contiguous memory locations in each hierarchy. For each $1 \le j \le S$ in sequence, we sort the $j$th bucket $\mathcal{S}_j$ recursively, after bringing the records of $\mathcal{S}_j$ to level $\lceil \log(N/SP) \rceil + 1$ of the hierarchical memory. (With high probability, the records in each bucket are distributed evenly among the $P$ hierarchies.) The sorted list of $N$ records consists of the concatenated sorted buckets.

In Step 5, we use the "touch" algorithm of Lemma 1 to process records $P$ at a time. We partition the records into buckets by merging the partitioning elements with each sorted subset $\mathcal{G}_i$ .

**Theorem 7** *The time used by the above algorithm to sort $N \ge P$ records in the P-BT model with $f(x) = x^\alpha$, for $0 < \alpha < 1$, is*

$$O\left(\frac{N}{P} \log N\right)$$

*with overwhelming probability. In particular, the probability that the number of I/Os used is more than $\ell$ times the average is exponentially small in $\ell(\log \ell) \log P$.*

*Proof*: Let $T(N)$ be the time used by this algorithm to sort a file of $N$ records. For $N \ge P^{(1+\alpha)/(1-\alpha)}$, the time needed in Step 2 to subdivide the set of $N$ records, move the subsets to faster memory, and sort the subsets $\mathcal{G}_i$ is bounded by

$$t'\, T\left(\frac{N}{t'}\right) + O\left(t'\left(\frac{N}{P}\right)^\alpha + \frac{N}{P}\right) \quad = \quad \left(\frac{N}{P}\right)^{1-\alpha} T\left(P\left(\frac{N}{P}\right)^\alpha\right) + O\left(\frac{N}{P}\right).$$

where $t' = (N/P)^{1-\alpha}$, $t = \lceil t' \rceil$. The time for touching each $\mathcal{G}_i$ and accumulating the $N/\log N$ elements of set $A$ in Step 3 is $O((N/P)\log\log(N/P))$ [2]. Using Lemma 1, we can show that the time for the two-way merge sort used in Step 3 to sort $n$ elements is $O((n/P)\log n(\log\log n + \log P))$. Since $n \approx N/\log N$, the resulting time to find the $S$ partitioning elements is $O((N/P)(\log\log N + \log P))$. The method of choosing the partitioning elements guarantees that the size $N_j$ of the $j$th bucket is at most $2N/S$, for each $1 \le j \le S$. By Lemma 1, the time needed to partition the file in Steps 5, 6, and 7 is $O(t(N/tP)(\log\log N + \log P) + t(S/P)\log\log S) = O((N/P)(\log\log N + \log P))$. The data movement to reposition the buckets in Step 8 can be done by the same method used by the one-hierarchy algorithm [2], that is, by computing the generalized matrix transposition for each hierarchy independently; the time needed is thus $O((N/P)(\log\log(N/P))^4)$ with high probability. The analysis in the companion paper [12] can be modified to show that with high probability the records in each bucket are distributed evenly over the $P$ hierarchies, so that the time for sorting the buckets recursively in Step 8 is with high probability

$$\sum_{1 \le j \le S} T(N_j) + O\left(\frac{N}{P} \log\log N\right),$$

where $N_j$ is the size of the $j$th bucket $S_j$, $\sum_{1 \le j \le S} N_j = N$, and $N_j \le 2N/S$ for each $j$. Hence, for $N \ge P^{(1+\alpha)/(1-\alpha)}$, with high probability we have

$$T(N) = \left(\frac{N}{P}\right)^{1-\alpha} T\left(P\left(\frac{N}{P}\right)^\alpha\right) + \sum_{1 \le j \le \lceil (NP)^\alpha / \ln(NP) \rceil} T(N_j) + O\left(\frac{N}{P}\left(\log\log\frac{N}{P}\right)^4 + \frac{N}{P}\log P\right),$$

and $N_j \leq 2N^{1-\alpha}P^{-\alpha}\ln(NP)$ for each $j$.

If $N < P^{(1+\alpha)/(1-\alpha)}$, there will be at most a constant number of applications of Phase 1 and one application of Phase 2, each phase taking $O((N/P)\log N)$ time with high probability. The remaining subfiles will have size at most $P$ and can be sorted in the base memory level in $O((N/P)\log P)$ time with high probability. This yields as desired the time bound

$$T(N) = O\left(\frac{N}{P}\log N\right)$$

with high probability. The probability bounds follow from those derived in [12]. □

A lower bound of $\Omega((N/P)\log N)$ time for sorting with $f(x) = x^\alpha$, for $\frac{1}{2} \leq \alpha < 1$, follows from the well-known lower bound for sorting in a RAM under the comparison model of computation.

### 6.2 Other Access Cost Functions

Since the above algorithm is optimal for the access cost function $f(x) = x^{1/2}$, it is also optimal for $f(x) = \log x$ and $f(x) = x^\alpha$, for $0 < \alpha < \frac{1}{2}$.

Now let us consider the access cost function $f(x) = x^\alpha$, where $\alpha \geq 1$. We sort by a simple application of divide-and-conquer merge sort. The upper bound of Theorem 2 follows by using the algorithm of Lemma 1 for merging two sorted lists.

The lower bound in Theorem 2 for the case $\alpha = 1$ follows by a modification of the argument in [2] for $P = 1$. The first term in the lower bound in Theorem 2 for the case $\alpha > 1$ is the time needed to access the farthest elements in memory, and the second term is dominated by the lower bound for $\alpha = 1$.

## 7 FFT in P-HMM

The P-HMM lower bounds proved in Theorem 4 and 5 for sorting apply also to the FFT computation. This follows immediately by substituting the phrase "FFT" for "sorting" in the proofs of the theorems. What remains to prove the FFT bounds in Theorem 1 is to give the FFT algorithm that meets these bounds.

We can perform the FFT when $N \geq P^2$ using the following well-known technique that mimics somewhat the recursive decomposition used in Theorem 3. (The reader is referred to Section 5 in [12] for a discussion of FFT and the shuffle-merge technique.)

1. We start by computing $\sqrt{N}$ FFTs. The $i$th FFT is computed on the $i$th contiguous group of $\sqrt{N}/P$ tracks.
2. We shuffle-merge the records to form $\sqrt{N}$ new contiguous groups of $\sqrt{N}/P$ tracks. For each $1 \leq i \leq \sqrt{N}$, the $i$th new group consists of the $i$th record from each of the original $\sqrt{N}$ groups.
3. We finish the computation by doing $\sqrt{N}$ FFTs, one for each of the new groups.

If $P < N < P^2$, we do $N/P$ FFTs, each FFT of size $P$, followed by a shuffle-merge, followed by FFTs of size $N/P$.

**Theorem 8** *The time used by the above algorithm to compute an $N$-input FFT with $f(x) = \log x$ is*

$$O\left(\frac{N}{P}\log N \log\left(\frac{\log N}{\log P}\right)\right).$$

*Proof*: Steps 1 and 3 take $\sqrt{N}\,T(\sqrt{N}\,)$ time. The shuffling step can be done in linear time $O((N/P)\log(N/P))$ once each group is "shifted" by an appropriate offset. By "shift," we mean that every record that is stored in the $k$th hierarchy is moved to hierarchy $1 + (k + \mathit{offset} - 1) \bmod P$. The $i$th group is shifted by $\mathit{offset} = (i-1) \bmod P$. The shifting can be done in time $O((N/P)\log P)$. This gives us

$$T(N) = 2\sqrt{N}\,T(\sqrt{N}\,) + O\left(\frac{N}{P}\log N\right),$$

which yields the desired bound. □

The proof of Theorem 6 carries over to show that the above algorithm is uniformly optimal for all well-behaved access cost functions.

# 8 FFT in P-BT

In this section we apply the results of Section 7 to prove the FFT portion of Theorem 2. The lower bounds for sorting apply also to FFT. The FFT algorithm that meets these bounds is based on the FFT algorithm of Section 7. The shuffling is done by touching the records in each group, using the touching algorithm of [2] applied to the hierarchies independently. For each $P$ records of a group that are touched at the same time, the records are shifted by the offset amount while in the base level memory.

# 9 Standard Matrix Multiplication in P-HMM

## 9.1 Upper Bounds

Before we present our optimal standard matrix multiplication algorithm in the P-HMM model, we first present a lemma that we need to show optimality.

**Lemma 2** *The time used to add two $k \times k$ matrices, where $k > P$, is*

$$O\left(\frac{k^2}{P}\log\frac{k}{P}\right) \qquad \text{if } f(x) = \log x;$$

$$O\left(\left(\frac{k^2}{P}\right)^{\alpha+1}\right) \qquad \text{if } f(x) = x^\alpha,\ \ \alpha > 0.$$

*Proof Sketch*: Two matrices can be added by touching the corresponding elements of the matrices simultaneously, using the naive touching algorithm applied to the hierarchies independently. Once two elements are in base memory level together, they can be added. □

We use the following divide-and-conquer algorithm, as for two-level memories [12]:

1. If $k \leq \sqrt{M}$, we multiply the matrices internally. Otherwise we do the following steps:
2. We subdivide $A$ and $B$ into 8 $k/2 \times k/2$ submatrices: $A_1$–$A_4$ and $B_1$–$B_4$.

$$A = \begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix}; \qquad B = \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix}.$$

   We reposition the records so that $A_1$–$A_4$ and $B_1$–$B_4$ are each stored in row-major order.
3. We recursively multiply the 8 pairs of submatrices.
4. We add the 4 pairs of submatrices which resulted from the above multiplications, giving $C_1$–$C_4$.
5. We reposition $C_1$–$C_4$ so that $C$ is stored in row-major order.

The repositioning of the matrices can be done in the same time as the touching problem. We define $T(k)$ to be the time used by the algorithm to multiply two $k \times k$ matrices together. For $f(x) = \log x$, we get from Lemma 2

$$T(k) = 8\,T\left(\frac{k}{2}\right) + O\left(\frac{k^2}{P}\log\frac{k}{P}\right).$$

Using the stopping condition $T(\sqrt{P}) = \sqrt{P}$, we get the desired upper bound $O(k^3/P)$ of Theorem 1. The upper bounds for the other access cost functions follow by using the other cases of Lemma 2.

## 9.2 Lower Bounds

The standard matrix multiplication algorithm given earlier for the logarithmic access cost is *uniformly optimal*, that is, it is optimal for all well-behaved access cost functions. This can be proved using the same approach as in Theorem 6. By [11], the I/O complexity of multipling two $k \times k$ matrices with one disk, no blocking, and an internal memory of size $M$ is

$$T_M(k^2) = \Omega\left(\frac{k^3}{\sqrt{M}} - M\right). \tag{6}$$

Let $T_{M,P}(N)$ be the average number of I/O steps done by the standard matrix multiplication algorithm with respect to an internal memory of size $M$, where we allow each hierarchy to move simultaneously, in a single I/O step, a record between internal memory and external memory. From the algorithm, for $k > \sqrt{M}$, we get the recurrence

$$T_{M,P}(k) = 8\,T_{M,P}\left(\frac{k}{2}\right) + \frac{k^2}{P}.$$

For smaller $k \leq \sqrt{M}$, we have $T_{M,P}(k) = O(k/P)$. The solution of this recurrence is

$$T_{M,P}(k) = O\left(\frac{k^3}{P\sqrt{M}}\right),$$

which by (6) is within an $O(M/P)$ additive term of optimal. The time used by the algorithm in base memory level computations is

$$O\left(\frac{k^3}{P\sqrt{P}}\sqrt{P}\right) = O\left(\frac{k^3}{P}\right).$$

Therefore, the extra time used by the algorithm over and above the optimal amount is

$$O\left(\frac{k^3}{P} + \sum_{P \le M < k^2} \frac{M}{P}\delta f\left(\frac{M}{P}\right)\right) \quad = \quad O\left(\frac{k^3}{P} + \frac{k^2}{P} f\left(\frac{k^2}{P}\right)\right). \tag{7}$$

The first term in the right-hand side of (7) is bounded by the number of operations performed, and the second term is the time required to access all the elements; thus the running time is within a constant factor of optimal.

## 10 Standard Matrix Multiplication in P-BT

Before we present the P-BT algorithm that yields the optimal bound quoted in Theorem 2 for standard matrix multiplication, we first present a useful lemma.

**Lemma 3** *The time used to add two $k \times k$ matrices, where $k > P$, is*

$$\begin{array}{ll} O\left(\frac{k^2}{P}\log^* \frac{k^2}{P}\right) & \text{if } f(x) = \log x; \\ O\left(\frac{k^2}{P}\log\log k\right) & \text{if } f(x) = x^\alpha,\ \ 0 < \alpha < 1; \\ O\left(\frac{k^2}{P}\log k\right) & \text{if } f(x) = x^\alpha,\ \ \alpha = 1; \\ O\left(\left(\frac{k^2}{P}\right)^\alpha\right) & \text{if } f(x) = x^\alpha,\ \ \alpha > 1. \end{array}$$

*Proof Sketch*: We apply the same approach as in Lemma 2. Two matrices can be added by touching the corresponding elements of the matrices simultaneously, using the touching algorithm of [2] applied to the hierarchies independently. Once two elements are in base memory level together, they can be added. The resultant matrix moves to slower memory in the same manner as the two matrices being added moved to faster memory, only backwards. □

Let us consider the access cost function $f(x) = x^\alpha$, where $0 < \alpha < 1$. The algorithm presented in Section 9 can be adapted to run on the P-BT model. The repositioning of the matrices can be done in the same time as the touching problem. We define $T(k)$ to be the time used by the algorithm to multiply two $k \times k$ matrices together. It is easy to see that

$$T(k) = 8\,T\left(\frac{k}{2}\right) + O\left(\frac{k^2}{P}\log\log k\right).$$

Using the stopping condition $T(\sqrt{P}) = \sqrt{P}$, we get the desired upper bound $O(k^3/P)$ of Theorem 2 for the access cost function $f(x) = x^\alpha$, where $0 < \alpha < 1$. The lower bound of $\Omega(k^3/P)$ clearly holds since the number of operations performed is $\Theta(k^3)$.

Since the above algorithm is optimal for the access cost function $f(x) = x^{1/2}$, it is also optimal for $f(x) = \log x$. The upper bounds for the remaining cases of Theorem 2 follow by applying the other cases of Lemma 3. The lower bound for the access cost function $f(x) = x^\alpha$, where $\alpha = 3/2$, for the BT model [2] can be modified for the P-BT model. When $\alpha > \frac{3}{2}$ we get a lower bound of $\Omega((k^2/P)^\alpha)$ since that is the time needed to access the farthest elements in memory.

# 11 Conclusions

We have presented optimal hierarchical memory algorithms for sorting and matrix-related problems that take advantage of multiple hierarchies. The sorting algorithm is a randomized version of distribution sort, using the partitioning technique of the companion paper [12], which was developed for optimal sorting on two-level memories with parallel block transfer.

**Addendum.** Recently an alternative two-level sorting algorithm that is both optimal and deterministic was developed by Nodine and Vitter [9]. The algorithm is based on merge sort and thus does not seem to provide optimal P-HMM and P-BT algorithms when applied to hierarchical memory. Subsequently, Nodine and Vitter developed optimal sorting algorithms for P-HMM and P-BT based on distribution sort that are deterministic [7,8].

Another interesting type of hierarchical memory is introduced in [4]. Parallel hierarchies of this type are studied in [6].

**Acknowledgments.** We thank Mark Nodine for several helpful comments.

# References

[1] A. Aggarwal, B. Alpern, A. K. Chandra, and M. Snir, "A Model for Hierarchical Memory," IBM Watson Research Center, Technical Report RC 15118, October 1989, also appears in *Proceedings of 19th Annual ACM Symposium on Theory of Computing*, New York (May 1987), 305–314.

[2] A. Aggarwal, A. Chandra, and M. Snir, "Hierarchical Memory with Block Transfer," *Proceedings of 28th Annual IEEE Symposium on Foundations of Computer Science* (October 1987), 204–216.

[3] A. Aggarwal and J. S. Vitter, "The Input/Output Complexity of Sorting and Related Problems," *Communications of the ACM* (September 1988), 1116–1127, also appears in *Proceedings of 14th Annual International Colloquium on Automata, Languages, and Programming (ICALP)*, Lecture Notes in Computer Science 267, Springer-Verlag, Berlin, 1987.

[4] B. Alpern, L. Carter, and E. Feig, "Uniform Memory Hierarchies," *Proceedings of 31st Annual IEEE Symposium on Foundations of Computer Science* (October 1990).

[5] F. Luccio and L. Pagli, "A Model of Sequential Computation Based on a Pipelined Access to Memory," *Proceedings of the 27th Annual Allerton Conference on Communication, Control, and Computing* (September 1989).

[6] M. H. Nodine and J. S. Vitter, "Large-Scale Sorting in Uniform Memory Hierarchies," *Journal of Parallel and Distributed Computing*, special issue.

[7] M. H. Nodine and J. S. Vitter, "Optimal Deterministic Sorting on Parallel Memory Hierarchies," Department of Computer Science, Brown University, Technical Report , August 1992.

[8] M. H. Nodine and J. S. Vitter, "Optimal Deterministic Sorting on Parallel Disks," Department of Computer Science, Brown University, Technical Report , August 1992.

[9] M. H. Nodine and J. S. Vitter, "Large-Scale Sorting in Parallel Memories," *Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures* (July 1991).

[10] J. H. Reif and L. G. Valiant, "A Logarithmic Time Sort on Linear Size Networks," *Journal of the ACM* 34 (January 1987), 60–76, also appears in *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, Boston (April 1983), 10–16.

[11] J. Savage and J. S. Vitter, "Parallelism in Space-Time Tradeoffs," in *Advances in Computing Research, Volume 4*, F. P. Preparata, ed., JAI Press, 1987, 117–146, also appears in *Proceedings of the International Workshop on Parallel Computing and VLSI*, Amalfi, Italy (May 1984), P. Bertolazzi and F. Luccio, ed., Elsevier Science Press, 1985, 49–58.

[12] J. S. Vitter and E. A. Shriver, "Algorithms for Parallel Memory I: Two-Level Memories," Department of Computer Science, Brown University, Technical Report CS–90–21, September 1990.