Large-Scale Sorting in Uniform Memory Hierarchies

Jeffrey Scott Vitter[1]
Mark H. Nodine[2]

Technical Report No. CS-92-02
January 1992

[1]Department of Computer Science, Box 1910, Brown University, Providence, RI 02912
[2]Motorola Cambridge Research Center, One Kendall Square, Building 200, Cambridge, MA 02139

Large-Scale Sorting in Uniform Memory Hierarchies*

Jeffrey Scott Vitter[†] and *Mark H. Nodine*[‡]

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

October 6, 1992

Abstract. We present several efficient algorithms for sorting on the uniform memory hierarchy (UMH), introduced by Alpern, Carter, and Feig, and its parallelization P-UMH. We give optimal and nearly-optimal algorithms for a wide range of bandwidth degradations, including a parsimonious algorithm for constant bandwidth. We also develop optimal sorting algorithms for all bandwidths for other versions of UMH and P-UMH, including natural restrictions we introduce called RUMH and P-RUMH, which more closely correspond to current programming languages.

---

*An earlier version of this research appeared in shortened form in "Large-Scale Sorting in Parallel Memories," *Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '91)*, Hilton Head, SC, July 1991, 29–39.

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

‡Support was provided in part by an IBM Graduate Fellowship, by a National Science Foundation Presidential Young Investigator Award CCR–9047466 with matching funds from IBM Corporation, by National Science Foundation grant CCR–9007851, and by the U.S. Army Research Office under grant DAAL03–91–G–0035.

1 Introduction

Input/output (I/O) has become a bottleneck in many large-scale problems. This bottleneck arises because advances in disk technology have not kept pace with those for processor technology. The problem is further aggravated by the current trend towards using several processors in parallel. The simplest expedient for overcoming the bottleneck is to use many disks in parallel.

The traditional model where I/O takes place between internal memory and an external drive, such as a disk or tape, focuses on only one small portion of the I/O problem. In most large-scale computer systems, memory progresses from very small but very fast registers to successively larger but slower components, such as several layers of cache, primary memory, disks, and archival storage. In order to achieve optimal performance on such a computer system, it is often necessary for the algorithm designer to take into account the physical characteristics of the entire memory hierarchy. Unfortunately, there are too many possible variables to consider (e.g., the block size of each level, the number of blocks at each level, the bandwidth between one level and the next) to allow the design of general algorithms; hence some degree of abstraction of the memory hierarchy is required.

Several interesting and elegant hierarchical memory models have been proposed recently to model the many levels of memory typically found in large-scale computer systems. The Hierarchical Memory Model (HMM) of Aggarwal, Alpern, Chandra, and Snir [1] views the entire memory hierarchy as a linear address space and allows access to individual location x in time $f(x)$. We normally think of $f(x)$ as a non-decreasing function, such as $\log x$ or x^α, for some $\alpha > 0$,[1] depending on the characteristics of the memory hierarchy.

One important effect that is missing from the HMM model but that is generally present in memory hierarchies is that it is often almost as fast to transfer a whole block of data as it is to retrieve a single data element. This effect happens in disks, for example, because the amount of time needed to seek to a given sector is typically much larger than the amount of time needed to transfer the sector once the read/write

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

head is in place. Accordingly, data are usually transferred in large units of *blocks*. The Block Transfer (BT) model of Aggarwal, Chandra, and Snir [2] represents a notion of block transfer applied to HMM; in the BT model, access to the $t+1$ records at locations $x-t$, $x-t+1$, $\ldots$, x takes time $f(x)+t$. 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 [8]. Optimal sorting algorithms for each of these models have been developed [1,2,8].

In this paper, we concentrate on a newer hierarchical memory model introduced by Alpern, Carter, and Feig [5,6], called the Uniform Memory Hierarchy (UMH), which offers an alternative model of blocked multilevel memories. Let us consider a hierarchy comprising several levels of memory models, where the ℓth level has the following parameters associated with it:

$$\begin{aligned} s_\ell &= \text{\# of elements that fit in a block} \\ n_\ell &= \text{\# of blocks} \\ b_\ell &= \text{\# of elements that can move from level } \ell \text{ to level } \ell+1 \text{ per unit time} \end{aligned}$$

This model has enough parameters that it could be adjusted to suit any memory hierarchy. Unfortunately, the very flexibility in the parameters makes the model too detailed to be of theoretical or practical interest. A programmer would like to be able to design an algorithm that could work without modification in a wide range of memory hierarchies. In the $\text{UMH}_{b(\ell)}$ model (for integer constants $\alpha, \rho \geq 2$), the ℓth memory level (as illustrated in Figure 1) consists of $n(\ell) = \alpha\rho^\ell$ blocks, each of size $s(\ell) = \rho^\ell$; it is connected via buses to levels $\ell-1$ and $\ell+1$. Each individual block on level ℓ can be randomly accessed as a unit and transferred to or from level $\ell+1$ at a *bandwidth* of $b(\ell)$; that is, each block transfer takes $\rho^\ell/b(\ell)$ time units. We have switched from subscripts on s, n, and b to emphasize that there is a functional relationship. Thus, in the UMH model, the number of blocks and the size of each block on a given level is a multiplicative factor of ρ larger than those on the previous level, and the bandwidth is a well-behaved function of the level. A block on one level corresponds to a sub-block in the next level.

Figure 1: The uniform memory hierarchy (UMH), pictured here for the case $\alpha = 3$, $\rho = 2$. The ℓth memory level contains $\alpha\rho^\ell$ blocks, each of size ρ^ℓ. It is connected via buses to levels $\ell - 1$ and $\ell + 1$. Each level ℓ block can be randomly accessed and transferred to level $\ell + 1$ at a *bandwidth* of $b(\ell)$ (that is, in $\rho^\ell / b(\ell)$ time). This figure shows only the blocks (and sub-blocks for level $\ell + 1$); the individual records are not shown.

We can assign a linear ordering to the addresses in a UMH. The CPU resides at level 0. Level 0 will contain locations $1, \ldots, \alpha$, level 1 will contain locations $\alpha + 1, \ldots, \alpha + \alpha\rho^2$, and so on.

Table 1 gives a comparison between a real system, the IBM RISC System/6000 memory hierarchy, and a UMH model whose parameters are chosen to approximate the architecture. The UMH model captures the flavor of the real memory hierarchy, although there are some differences. In this paper, we will consider three bandwidth functions: $b(\ell) = 1$, $b(\ell) = 1/(\ell + 1)$, and $b(\ell) = \rho^{-c\ell}$. The constant bandwidth case is the fastest one that makes any sense theoretically, although it is unrealistic in practice. The exponentially decaying function approximates the real bandwidth

	RISC/6000			UMH			
level	n_ℓ	s_ℓ	b_ℓ	n_ℓ	s_ℓ	$1/(\ell+1)$	$\rho^{-\ell/2}$
4	64K	4K	—	64K	4K		
3	12K	512	0.001*	8K	512	0.25	0.044
2	128	512	17*	1024	64	0.33	0.125
1	512	16	1.1	128	8	0.5	0.35
0	32	1	1	16	1	1	1

Table 1: Comparison of the IBM RISC System/6000 memory hierarchy with a UMH whose parameters are chosen to approximate it. The RISC quantities labeled with an asterix (*) are approximate. The parameters used in the UMH model are $\alpha = 16$ and $\rho = 8$.

better than the linearly decaying one. Level 2 (and its connection to level 3) seems to be somewhat out of kilter from the rest of the hierarchy; discounting that level results in a linearly decaying bandwidth that fits as well as the exponential one.

A model for parallel hierarchies was introduced by Vitter and Shriver, in which H hierarchies are connected at their base level via an interconnection network as shown in Figure 2. Communication between the H hierarchies takes place at the *base memory level* (call it level 0), which consists of location 1 from each of the H hierarchies. The H base memory level locations are interconnected via a network such as the hypercube or cube-connected cycles so that the H records in the base memory level can be sorted in $O(\log H)$ time (perhaps via a randomized algorithm [10]). The parallel versions of HMM and BT are called P-HMM and P-BT, respectively. Vitter and Shriver introduced optimal randomized sorting algorithms for P-HMM and P-BT [12] to sort with the following bounds:

$$T_{\text{P-HMM}} = \begin{cases} \Theta\left(\frac{N}{H}\log N \log\left(\frac{\log N}{\log H}\right)\right) & \text{if } f(x) = \log x; \\ \Theta\left(\left(\frac{N}{H}\right)^{\alpha+1} + \frac{N}{H}\log N\right) & \text{if } f(x) = x^\alpha,\ \alpha > 0; \end{cases}$$

Figure 2: A parallel hierarchical memory. The H individual memory hierarchies are all of the same type, such as HMM, BT, or UMH. The H CPUs can communicate among one another via the interconnection network (which can be a hypercube or cube-connected cycles, for example).

$$T_{\text{P-BT}} = \begin{cases} \Theta\left(\dfrac{N}{H}\log N\right) & \text{if } f(x) = \log x \\ \Theta\left(\dfrac{N}{H}\log N\right) & \text{if } f(x) = x^\alpha,\ 0 < \alpha < 1; \\ \Theta\left(\dfrac{N}{H}\log N \log \dfrac{N}{H}\right) & \text{if } f(x) = x^\alpha,\ \alpha = 1; \\ \Theta\left(\left(\dfrac{N}{H}\right)^\alpha + \dfrac{N}{H}\log\dfrac{N}{H}\log H\right) & \text{if } f(x) = x^\alpha,\ \alpha > 1, \end{cases}$$

The algorithms were based on their randomized two-level partitioning technique applied to the optimal single-hierarchy algorithms for HMM and BT developed in [1, 2].

We can consider parallel UMH hierarchies (analogous to P-HMM and P-BT), and we call the resulting model P-UMH. (This is fundamentally different from the parallel type of UMH called UPHM mentioned in [5].) Each level ℓ in each of the H hierarchies of the P-UMH holds $\alpha\rho^{2\ell}$ records, for a total of $\alpha H\rho^{2\ell}$ records on level ℓ. Accordingly, we assume that the initial input of N elements resides at level $s = \lceil \frac{1}{2}\log_\rho \frac{N}{\alpha H}\rceil$, which is the smallest level that holds all the elements.

A UMH or P-UMH "program" consists of a schedule of choreographed block transfers and computations. If a RAM program that runs in $T(N)$ steps can be scheduled in UMH in $\sim T(N)$ time, the program is said to be *parsimonious*; note that the constant factor must be 1. If the UMH program runs in time $O(T(N))$, it is said to be *efficient*. A UMH program whose running time is within a constant factor of best possible for that problem in the UMH model is said to be *optimal*. We can define "parsimonious", "efficient", and "optimal" analogously for P-UMH by comparing to a PRAM program.

When we consider sorting in UMH, we assume that each memory location can hold one of the records to be sorted. In Section 2 we give optimal and near-optimal sorting algorithms for UMH and P-UMH for a wide range of bandwidth rates $b(\ell)$, and we present a parsimonious schedule for merge sort for the case $b(\ell) = 1$. In Section 3 we also introduce a natural and easy-to-program restriction of UMH, called random-access UMH (or RUMH), for which we have optimal upper and lower bounds for all bandwidths and amounts of parallelism, and a sequential model of UMH called SUMH, for which we do the same.

2 Sorting in UMH and its Parallelization

Optimal sorting in $O(N \log N)$ time in UMH is possible only when the bandwidth $b(\ell)$ at level ℓ is $\Omega(1/\ell)$, or else the time required just to access the N records will be greater than $O(N \log N)$. Many buses may be active simultaneously in the UMH model, so conceivably it is possible to sort in $O(N \log N)$ time even with small bandwidth $b(\ell) = 1/(\ell + 1)$. The FFT computation, which is similar to sorting, can be done in $O(N \log N)$ time with bandwidth $b(\ell) = 1/(\ell + 1)$. Whether or not an $O(N \log N)$-time $\mathrm{UMH}_{1/(\ell+1)}$ algorithm exists is a big open problem.

In this section we give a near-optimal sorting algorithm for the small bandwidth case $b(\ell) = 1/(\ell+1)$, and optimal sorting algorithms for several other bandwidths. For the special case of constant bandwidth, we present a parsimonious algorithm. Since optimal sorting seems to require nonoblivious UMH programs, the oblivious UMH model of [5] must be modified in a reasonable way. In Theorem 1, we assume that the ℓth level of the hierarchy can initiate a transfer from the $(\ell+1)$st level without involving the CPU when one of its blocks becomes empty. In the remaining theorems, we assume that the CPU can originate the transfer of a block at level ℓ given the address of the block, with suitable delay.

The fastest oblivious algorithm we have found for sorting in $\mathrm{UMH}_{1/(\ell+1)}$ is based on a simple schedule of Batcher's bitonic sort [4] where each of the $\log^2 N$ parallel time steps is implemented in $O(N \log N)$ time for an overall running time of $O(N \log^3 N)$. It is also possible to schedule a recursive version of Columnsort [7] on $\mathrm{UMH}_{1/(\ell+1)}$ in a manner that is efficient with respective to the RAM algorithm, but this observation is not very useful since both algorithms have running time that is $O(N \log^c N)$, where $c \approx 3.4$.

2.1 Parsimonious sorting in UMH_1

Theorem 1 *A variant of merge sort can be scheduled in* UMH_1 *parsimoniously, assuming* $\alpha \geq 2$ *and* $\alpha\rho \geq 6$.

Proof: The basic idea is to schedule a systolic binary merge sort in such a way that the CPU is always kept busy (except for a small initial delay and a small final delay for propagating the results back). After the initial delay, the CPU (level 0) reads one element from each of the two lists. The amount of time needed to move the first element from each of the two lists down to the CPU is $2\sum_{0 \leq k < s} \rho^k = O(\sqrt{N})$, where $s = \frac{1}{2}\log_\rho(N/\alpha)$ is the starting level. At every time step after this initial delay, the CPU writes the smaller element to the output list and then reads the next element from the list that had the smaller element at the previous step. We use a double-buffering scheme so that level ℓ, for $\ell \geq 1$, contains room for two blocks from each of the two lists being merged. It also has two blocks for the output list. When level $\ell - 1$ requests a sub-block from one of the lists, and this request causes level ℓ's buffer to be emptied, then level ℓ requests the next block from level $\ell+1$. In this way, level ℓ always has at least one sub-block for level $\ell-1$ available on demand. The output blocks fill up at a known rate, so they can be scheduled in advance (again using double-buffering to keep an empty sub-block available for writing from level $\ell-1$). At the end of each list, we immediately begin to send a new list down for the next merge. The CPU can keep track of how many elements have been read from each list, so that when one list is finished it knows to copy the rest of the other list to the output. There will be a final delay of $O(\sqrt{N})$ as the last results propagate back to the starting level. The number of wasted CPU cycles is only $O(\sqrt{N}) = o(N \log N)$, so the schedule is parsimonious. □

2.2 Sorting in P-UMH

In the previous subsection, we were rather explicit in describing the choreography of data movement within the UMH. The remainder of the paper uses simulations of P-HMM and P-BT algorithms. Accordingly, we have to state how to derive a choreography from simulating one of these algorithms. The difficulty in choreographing the data movement is in making effective use of the blocking on the UMH. We therefore restrict ourselves to simulating algorithms that access blocks of data (even though the P-HMM model does not take advantage of those access patterns). When a block of data is moved from one level ℓ of a UMH to a lower level ℓ', we choreograph this movement by sending the first block from level ℓ to level $\ell-1$. We start sending the second block from level ℓ to level $\ell-1$ while simultaneously sending the first block to level $\ell-2$ sub-block by sub-block. Since we consider only non-increasing bandwidth functions, the space occupied by the first block on level $\ell-1$ will always have been completely freed by the time we are ready to send the third block from level ℓ to level $\ell-1$. The net effect is that we can keep the bus between levels ℓ and $\ell-1$ always busy. In transferring to a level $\ell' > \ell$, we adopt the same tactics in reverse: we time the transfers from the lower levels so as always to keep the bus between levels $\ell'-1$ and ℓ' busy.

Theorem 2 *Distribution sort algorithms can be scheduled on P-UMH with the following running times. The algorithms for nonconstant H for the first two bandwidth cases are randomized.*

$$\begin{array}{ll} \Theta\left(\frac{N}{H}\log N\right) & \text{if } b(\ell)=1; \\ O\left(\frac{N}{H}\log N \log\left(\frac{\log N}{\log H}\right)\right) & \text{if } b(\ell)=\frac{1}{\ell+1}; \\ \Theta\left(\left(\frac{N}{H}\right)^{1+c/2}+\frac{N}{H}\log N\right) & \text{if } b(\ell)=\rho^{-c\ell},\ c>0. \end{array}$$

Proof: The lower bound for the first case $b(\ell)=1$ follows from the conventional $\Omega(N\log N)$ serial bound for sorting on a RAM. With H processors, the P-UMH sorting time can be at most H times faster, giving a $\Omega((N/H)\log N)$ lower bound. The best known lower bound for the case $b(\ell)=1/(\ell+1)$ is the same as when $b(\ell)=1$. To prove the lower bound when $b(\ell)=\rho^{-c\ell}$, we assume that the N records reside initially in level $s=\lceil\frac{1}{2}\log_\rho\frac{N}{\alpha H}\rceil$. To move the N items from level s to level $s-1$, we need to move N/ρ^s blocks, each of which takes time $\rho^s/b(s)=\rho^{(c+1)s}$ with at most H blocks transferred simultaneously. It follows that it takes time

$$\frac{N\rho^{cs}}{H}=\frac{N}{H}\rho^{(c/2)\log_\rho(N/\alpha H)}=\frac{N}{H}\left(\frac{N}{\alpha H}\right)^{c/2}=\Omega\left(\left(\frac{N}{H}\right)^{1+c/2}\right)$$

to get the N records from level s to level $s-1$, thus completing the lower bound for the third case.

The algorithm that achieves the upper bound for the first case $b(\ell)=1$ is based on a simulation of the P-BT algorithm for access cost function $f(x)=\sqrt{x}$ given in [11]. The time for the simulation is bounded by a constant times the P-BT running time. Let us consider moving any $b+1$ elements from locations $x-b,\ldots,x$ to locations $y-b,\ldots,y$ in the BT model with access cost function $f(x)=\sqrt{x}$. The amount of time taken for this transfer is $\sqrt{x}+\sqrt{y}+b$. We now compute the amount of time taken by a UMH to do the same transfer with multiple levels of the hierarchy active concurrently. We define the level function

$$\mathrm{lev}(x)=\left\lceil\frac{1}{2}\log_\rho\frac{x}{\alpha}\right\rceil,$$

which is the level on which location x appears in a single UMH. Let us assume for the time being that all of the elements in the block $x-b,\ldots,x$ occur on the same level and that $x>y$. At most two partly-filled blocks will be transferred from any level in the range $\mathrm{lev}(y)+1,\ldots,\mathrm{lev}(x)$. Level $\mathrm{lev}(x)-1$ will receive its first elements after time $\rho^{\mathrm{lev}(x)-1}$ (we assume that the whole block has to arrive before any of the elements are accessible). It can then start sending elements down to level $\mathrm{lev}(x)-2$. Level $\mathrm{lev}(y)$ will start getting its first elements after a delay of

$$\sum_{\mathrm{lev}(y)\le k<\mathrm{lev}(x)} \rho^k = \frac{\rho^{\mathrm{lev}(x)}-\rho^{\mathrm{lev}(y)}}{\rho-1}.$$

Since

$$\rho^{\mathrm{lev}(x)} = \rho^{\log_\rho(x/\alpha)/2} = \sqrt{\frac{x}{\alpha}},$$

the delay before the first elements arrives is

$$\frac{\sqrt{x/\alpha}-\sqrt{y/\alpha}}{\rho-1} = \frac{\sqrt{x}-\sqrt{y}}{\sqrt{\alpha}(\rho-1)}.$$

Bad blocking imposes an additional delay of $\rho^{\mathrm{lev}(x)-1}=\sqrt{x/\alpha}/\rho$, and each additional element after the first block imposes a delay of 1. Thus, the total amount of time that is needed is no more than

$$\frac{\sqrt{x}-\sqrt{y}}{\sqrt{\alpha}(\rho-1)} + \frac{\sqrt{x}}{\sqrt{\alpha}\rho} + b,$$

which is bounded by $c(\sqrt{x}+\sqrt{y}+b)$ for all $c\ge 2/\sqrt{\alpha}$. The case where $x<y$ is handled analogously and is likewise bounded by $c(\sqrt{x}+\sqrt{y}+b)$. Thus, the P-UMH time is within a constant factor of the P-BT time with cost access function $f(x)=\sqrt{x}$, which is $O((N/H)\log N)$.

The other simulations presented in this paper are a bit trickier, since they require that effective use be made of blocking in the UMH simulation, and therefore that the algorithm meet certain constraints. A sufficient constraint is for the operations in the algorithm to process all the elements at any level in the hierarchy consecutively. It may be convenient for the algorithm to consider the elements as comprising groups that may be unrelated to the block size for that level, but as long as all the elements are accessed consecutively, the intermediate levels of the hierarchy can act as buffers to allow reblocking to occur as needed without losing efficiency, as long as $\alpha \geq 3$. The algorithms given in [11] all meet this constraint.

For the second case $b(\ell) = 1/(\ell+1)$, the upper bound is related to the P-HMM approach for $f(x) = \log x$ [11]. The P-HMM algorithm needs to be modified to reblock the buckets prior to sorting them recursively. Immediately prior to the recursive call, the P-HMM algorithm has the buckets spread out evenly among the hierarchies; the elements of any given bucket are not necessarily contiguous on any of the hierarchies, but rather are linked together. We insert a step into the P-HMM algorithm that permutes the elements on each hierarchy so that each bucket appears in contiguous locations; an algorithm to perform this generalized transposition permutation was described in [2]. The cost of accessing an element at location x in the HMM model is $\log x$; the amortized cost of accessing the same element in the UMH model when an entire block is brought to the base memory level is $\log(x/\alpha)/\log\rho$, which is within a constant factor of the HMM cost. Thus, the overall cost is

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

just as for P-HMM with cost function $f(x) = \log x$.

The upper bound third case $b(\ell) = \rho^{-c\ell}$ makes use of an algorithm based on deterministic, two-way merge sort. We can use the same algorithm as we used in the proof for Theorem 1. The exact choreography is unimportant, since the overall transfer time of any transfer will be dominated by the amount of time needed to move to and from the highest level involved. This algorithm gives rise to the recurrence relation

$$S(N) = \begin{cases} 2S\left(\frac{N}{2}\right) + O\left(\left(\frac{N}{H}\right)^{c/2+1}\right) & \text{if } N > H; \\ O(\log N) & \text{if } N \leq H, \end{cases}$$

which gives the stated bound. □

The algorithms are optimal, except for the middle $b(\ell) = 1/(\ell+1)$ case, which is off from the best known lower bound of $\Theta((N/H)\log N)$ by a $\log((\log N)/\log H)$ factor.

3 Sorting in SUMH and RUMH and their Parallelizations

The UMH model can be difficult to program because many buses can be active simultaneously. An earlier version of [5] introduced a *sequential* UMH model, appropriately called SUMH, that allowed at most one bus to be active at a time. However, the SUMH restriction can be regarded as too severe, since it forfeits much power of the UMH model.

We introduce the following more natural and less severe restriction that fits in nicely with feasible and easy-to-use programming languages: We require that the UMH program correspond exactly to a RAM program in which the RAM instruction set is augmented with a block move command that can move t contiguous memory elements in time t, for arbitrary t. Each such block transfer can be implemented in UMH by a coordinated series of transfers in which several buses are simultaneously active but cooperating on that single transfer. We call this natural variant of UMH the *random-access* UMH model, or simply RUMH. For example, a block of $\sqrt{N}$ elements can be moved from the top memory level all the way down to the CPU (or anywhere in between) in $\sim \sqrt{N}$ time in UMH_1 and RUMH_1, but it requires $\Theta(\sqrt{N} \log N)$ time in SUMH_1.

The parallel versions of RUMH and SUMH are called P-RUMH and P-SUMH, respectively. Theorems 3 and 4 give matching upper and lower bounds for sorting in the RUMH and SUMH models and their parallelizations. The structures of the formulas in Theorems 3 and 4 suggest several different relationships between the RUMH and SUMH models on the one hand and the HMM, BT, and two-level models on the other hand (cf. Theorems 5 and 6 in [12]); accordingly the upper and lower bounds combine in an interesting way several techniques from [1,2,3,12].

Theorem 3 *The running times mentioned in Theorem 2 are matching upper and lower bounds for sorting in P-RUMH. The algorithms for nonconstant H for the first two bandwidth cases are randomized.*

Proof: The upper bounds all follow directly from the proof of Theorem 2, since all the algorithms given there are P-RUMH algorithms.

The lower bounds for $b(\ell) = 1$ and $b(\ell) = \rho^{-c\ell}$ are the same as and follow directly from those for P-UMH. When $b(\ell) = 1/(\ell+1)$, we can prove a tight lower bound by simulating $\text{RUMH}_{1/(\ell+1)}$ by HMM with access cost function $f(x) = \log x$. We compute the amount of time that it takes to transfer a block of $\rho^{\ell-1}$ elements from level ℓ to level ℓ' where $\ell > \ell'$. The amount of time before any elements start arriving at level ℓ' is

$$\sum_{\ell' \le k < \ell} \frac{\rho^k}{b(k)} = \sum_{\ell' \le k < \ell} (k+1)\rho^k.$$

Since

$$\sum_{0 \le k \le n} k a^k = \frac{a^{n+1}((a-1)(n+1) - a) + a}{(a-1)^2},$$

the amount of time is

$$\frac{\rho^{\ell}((\rho-1)\ell - \rho) - \rho^{\ell'}(\rho-1)\ell' - \rho)}{(\rho-1)^2} + \frac{\rho^{\ell} - \rho^{\ell'}}{\rho - 1} = \Theta(\ell\rho^{\ell-1}).$$

The extra time needed after the first elements have arrived at level ℓ' is no more than $\ell\rho^{\ell-1}$, so the overall time needed is $\Theta(\ell\rho^{\ell-1})$. Doing the same move in the HMM model requires time at most

$$\rho^{\ell-1}(\log(\alpha\rho^{2\ell}) + \log(\alpha\rho^{2(\ell'+1)})) = O(\ell\rho^{\ell-1}),$$

so the simulation by HMM is bounded by a constant times the $\text{RUMH}_{1/(\ell+1)}$ running time. Hence, the lower bound for P-HMM for $f(x) = \log x$ given in [11] also holds for $\text{P-RUMH}_{1/(\ell+1)}$. □

Theorem 4 *The following bounds are matching upper and lower bounds for sorting in P-SUMH. The algorithms for nonconstant H for the first two bandwidth cases are randomized.*

$$\Theta\left(\frac{N}{H}\log N \log\left(\frac{\log N}{\log H}\right)\right) \qquad \text{if } b(\ell) = 1;$$

$$\Theta\left(\frac{N}{H}\log N \log\frac{N}{H}\right) \qquad \text{if } b(\ell) = \frac{1}{\ell+1};$$

$$\Theta\left(\left(\frac{N}{H}\right)^{1+c/2} + \frac{N}{H}\log N\right) \qquad \text{if } b(\ell) = \rho^{-c\ell},\ c > 0.$$

Proof: We prove the lower bounds using an approach similar to that of [11]. Let us define the "sequential time" of a P-SUMH algorithm to be the sum of its time costs for each of the H hierarchies. The sequential time can be at most H times the P-SUMH running time. We superimpose on the P-SUMH model a sequence of one-disk, two-level memories of the type studied in [3,11], in the following way: For $1 \le \ell \le \frac{1}{2}\log_\rho(N/\alpha H)$, the ℓth two-level memory has one disk, internal memory size $M_\ell = H\alpha(\rho^{2(\ell+1)} - 1)/(\rho^2 - 1)$, and block size $B_\ell = \rho^\ell$. An I/O in the ℓth two-level memory corresponds to a single block transfer between levels ℓ and $\ell + 1$ in one of the hierarchies in the P-SUMH model, which requires sequential time $C_\ell = B_\ell / b(l)$. Substituting various bandwidth functions gives us

$$C_\ell = \begin{cases} B_\ell & \text{if } b(\ell) = 1; \\ \Theta(B_\ell \log B_\ell) & \text{if } b(\ell) = \dfrac{1}{\ell + 1}; \\ B_\ell^{1+c} & \text{if } b(\ell) = \rho^{-c\ell},\ c > 0. \end{cases}$$

The minimum number of I/Os required for sorting in the ℓth two-level memory is

$$\Omega\left(\frac{N}{B_\ell}\frac{\log(N/B_\ell)}{\log(M_\ell/B_\ell)} - \frac{M_\ell}{B_\ell}\right),$$

as shown in [3]. Each such I/O contributes C_i to the sequential time in the P-SUMH model, since in the P-SUMH model only one level can be active at a time in each hierarchy. This gives a lower bound on the P-SUMH sequential time:

$$T(N) = \Omega\left(\sum_{1 \le \ell \le \frac{1}{2}\log_\rho(N/\alpha H)} C_\ell \left(\frac{N}{B_\ell}\frac{\log(N/B_\ell)}{\log(M_\ell/B_\ell)} - \frac{M_\ell}{B_\ell}\right)\right).$$

We get the desired lower bound on the P-SUMH time by substituting the values of M_ℓ, B_ℓ, and C_ℓ for the three cases into the above summation, and then dividing by H. The $b(\ell) = \rho^{-c\ell}$ case additionally requires the use of the the conventional $N \log N$ serial bound for sorting.

The upper bounds for the first two cases $b(\ell) = 1$ and $b(\ell) = 1/(\ell + 1)$ are achieved by simulating the optimal P-HMM algorithm of [11], for access cost functions $f(x) = \log x$ and $f(x) = \log^2 x$, respectively. Since a UMH in each case can simulate an HMM with the appropriate cost function in a running time that is at most a constant times the HMM time, the P-HMM bound holds for the P-UMH simulation. The upper bound for the $b(\ell) = \rho^{-c\ell}$ case is achieved by the same deterministic merge sort as the previous theorems. □

4 Conclusions

We have given optimal or near-optimal sorting algorithms for UMH and its parallelization that we have introduced called P-UMH. We have derived tight matching upper and lower bounds for sorting in the restricted models RUMH and SUMH and their parallelizations. Some of the algorithms are randomized. The RUMH model is particularly useful because it is easy to visualize and it matches well with current programming languages and compilers.

An interesting open problem is whether it is possible to sort in $O(N \log N)$ time with the $\mathrm{UMH}_{1/(\ell+1)}$ model. The related FFT computation can be done in $\mathrm{UMH}_{1/(\ell+1)}$ in $O(N \log N)$ time. Another open problem is whether a parsimonious oblivious algorithm can be found to replace our non-oblivious one in UMH_1.

Addendum After submitting this paper, Nodine and Vitter [9] have developed an optimal deterministic sorting algorithm for the P-HMM and P-BT parallel memory hierarchies, assuming that the interconnection network at the base level consists of a CRCW PRAM to allow optimal internal sorting. This improves upon the optimal randomized algorithms of [11]. With such an interconnection network, the deterministic algorithms in [9] can then be used in place of those of [11] in the simulations in this paper, thus transforming all the randomized algorithms in this paper into deterministic ones.

Acknowledgments We thank the referees for their helpful comments.

References

[1] Aggarwal, A., Alpern, B., Chandra, A. K. and Snir, M., A model for hierarchical memory, *Proceedings of 19th Annual ACM Symposium on Theory of Computing*, New York, NY (May 1987), 305–314.

[2] Aggarwal, A., Chandra, A. and Snir, M., Hierarchical memory with block transfer, *Proceedings of 28th Annual IEEE Symposium on Foundations of Computer Science*, Los Angeles, CA (October 1987), 204–216.

[3] Aggarwal, A. and Vitter, J. S., The input/output complexity of sorting and related problems, *Communications of the ACM* (September 1988), 1116–1127.

[4] Akl, S. G., *Parallel Sorting Algorithms*, Notes and Reports in Computer Science and Applied Mathematics #12, Academic Press, Inc., Orlando, 1985.

[5] Alpern, B., Carter, L. and Feig, E., Uniform memory hierarchies, *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, St. Louis, MO (October 1990), 600–608.

[6] Alpern, B., Carter, L. and Selker, T., Visualizing computer memory architectures, *Proceedings of the 1990 IEEE Visualization Conference Foundations of Computer Science* (October 1990).

[7] Leighton, T., Tight bounds on the complexity of parallel sorting, *IEEE Transactions on Computers* C-34 (April 1985), 344–354.

[8] Luccio, F. and Pagli, L., A model of sequential computation based on a pipelined access to memory, *Proceedings of the 27th Annual Allerton Conference on Communication, Control, and Computing*, Allerton, IL (September 1989).

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

[10] Reif, J. H. and Valiant, L. G., A Logarithmic time sort on linear size networks, *Journal of the ACM* 34 (January 1987), 60–76.

[11] Vitter, J. S. and Shriver, E. A. M., Algorithms for Parallel Memory II: Hierarchical Multilevel Memories, Brown University, CS–92–05, 1992, also appears in summarized form in Optimal disk I/O with parallel block transfer, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, Baltimore, MD (May 1990) 159–169.

[12] Vitter, J. S. and Shriver, E. A. M., Algorithms for parallel memory I: two-level memories, Brown University, CS–92–04, 1992, also appears in summarized form in Optimal disk I/O with parallel block transfer, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, Baltimore, MD (May 1990) 159–169.