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

December 19, 1991

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.

1 Introduction

In many 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

[*]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 by the Office of Naval Research and the Defense Advanced Research Projects Agency under contract N00014–83–K–0146 and ARPA order 6320, amendment 1.

[†]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.

performance on such a computer, it is often necessary for the algorithm designer to take into account the physical characteristics of the 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 HMM model of Aggarwal, Alpern, Chandra, and Snir [AAC] allows access to individual location x in time $f(x)$. The BT model of Aggarwal, Chandra, and Snir [ACSa] 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 $\alpha > 0$.[1] 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 [LuP]. Optimal sorting algorithms for each of these models have been developed [AAC, ACSa, LuP].

In this paper we concentrate on a newer hierarchical memory model introduced by Alpern, Carter, and Feig [ACF, ACSb], called the uniform memory hierarchy (UMH), which offers an alternative model of blocked multilevel memories. In the $\mathrm{UMH}_{b(\ell)}$ model (for integer constants $\alpha, \rho \geq 2$), the ℓth memory level (as illustrated in Figure 1) consists of $\alpha\rho^{\ell}$ blocks, each of size ρ^{ℓ}; 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 time $\rho^{\ell}/b(\ell)$. The CPU resides at level 0.

A model for parallel hierarchies was introduced by Vitter and Shriver, in which P hierarchies are connected at their base level via an interconnection network as shown in Figure 2. Communication between the P hierarchies takes place at the *base memory level* (call it level 0), which consists of location 1 from each of the P hierarchies. The P base memory level locations are interconnected via a network such as the hypercube or cube-connected cycles so that the P records in the base memory level can be sorted

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

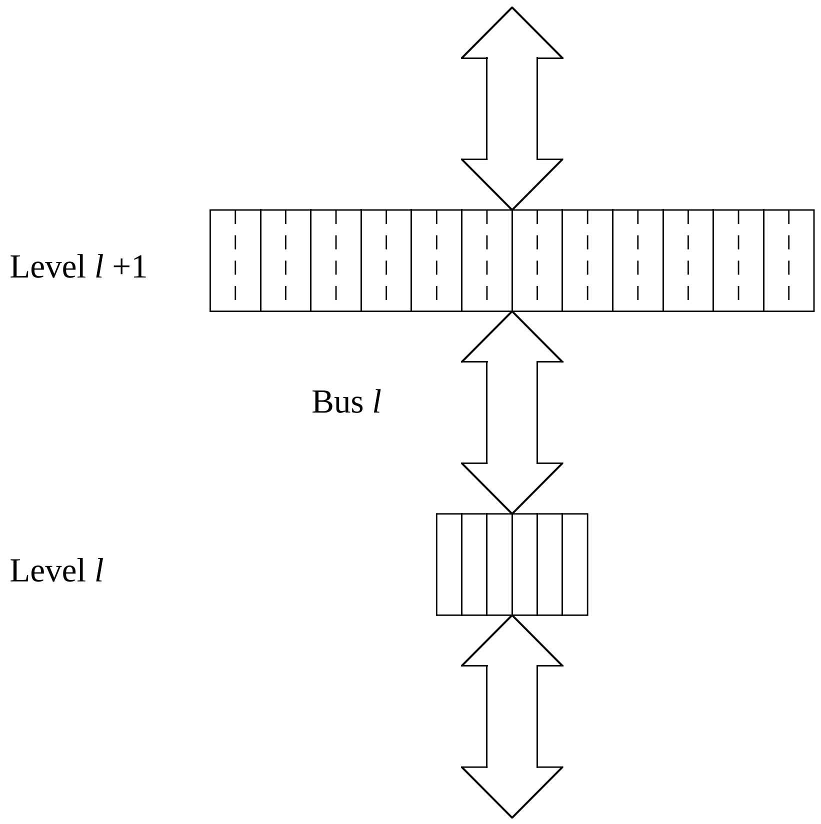

Figure 1: The uniform memory hierarchy (UMH), pictured here for the case $\alpha = 3$, $\rho = 2$. The ℓth memory level contains $\alpha\rho^\ell$ blocks, each of size ρ^ℓ. It is connected via buses to levels $\ell - 1$ and $\ell + 1$. Each level ℓ block can be randomly accessed and transferred to level $\ell + 1$ at a *bandwidth* of $b(\ell)$ (that is, in $\rho^\ell / b(\ell)$ time).

in $O(\log P)$ time (perhaps via a randomized algorithm [ReV]). Vitter and Shriver introduced optimal randomized sorting algorithms for P-HMM and P-BT [ViSa]. The algorithms were based on their randomized two-level partitioning technique applied to the optimal single-hierarchy algorithms for HMM and BT developed in [AAC, ACSa].

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 [ACF].) The initial input of N elements resides at level $s = \lceil \frac{1}{2} \log_\rho \frac{N}{\alpha P} \rceil$.

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

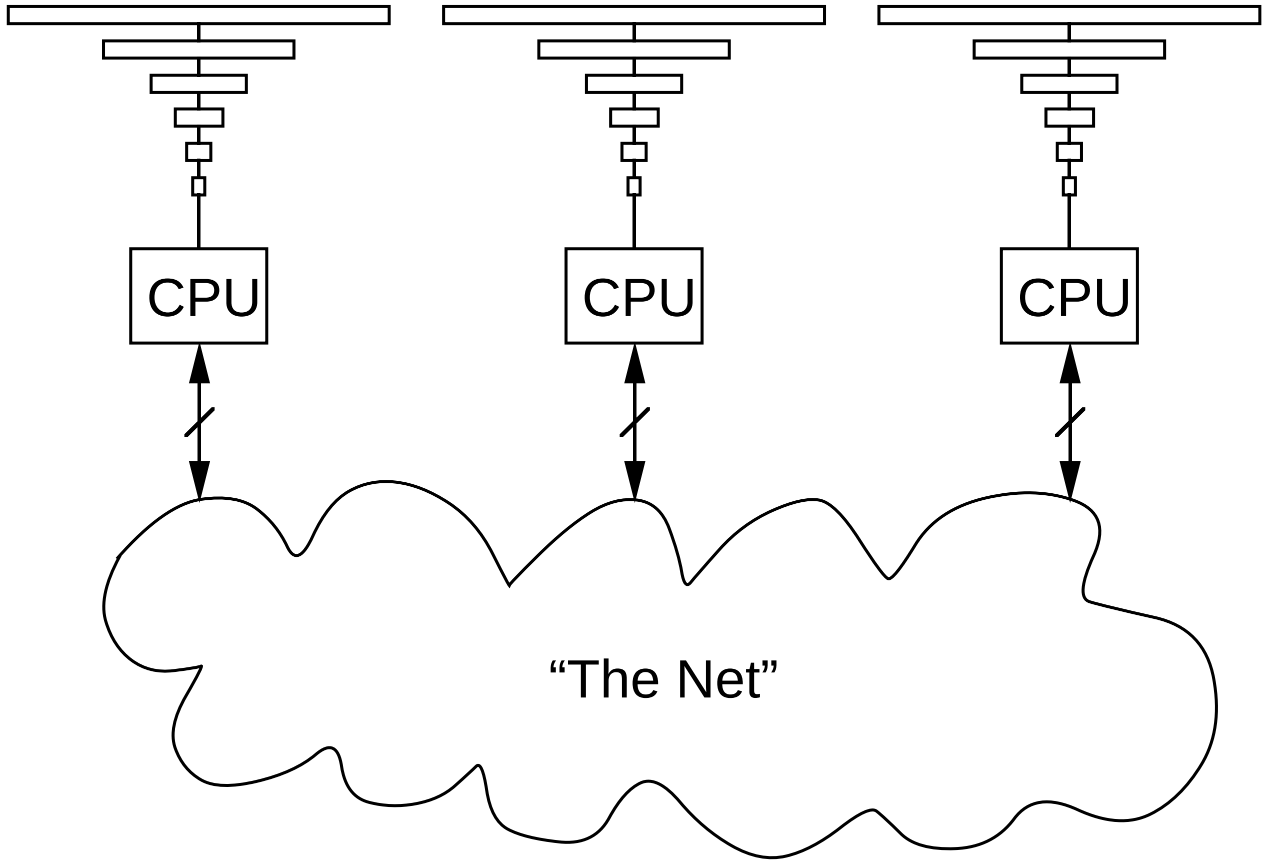

Figure 2: A parallel hierarchical memory. The P individual memory hierarchies are all of the same type, such as HMM, BT, or UMH. The P CPUs can communicate among one another via the interconnection network (which can be a hypercube or cube-connected cycles, for example).

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*.

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 $b(\ell) = 1/(\ell + 1)$.

Recently other authors announced an efficient UMH sorting algorithm for the case $b(\ell) = 1/(\ell + 1)$, based on the optimal two-level distribution sort algorithm of [AgV], but their $\mathrm{UMH}_{1/(\ell+1)}$ algorithm turned out to be inefficient, with a running time of $\Omega(N \log^c N)$, for $c > 3$. Whether or not an $O(N \log N)$-time $\mathrm{UMH}_{1/(\ell+1)}$ algorithm exists is still open.

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 [ACF] 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 [Akl] 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 [Lei] 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 $\mathbf{UMH}_1$

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 (after a small initial delay and with 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. At every time step after the 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 subblock 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 subblock 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. The number of wasted CPU cycles is only $O(\log N) = o(N \log N)$, so the schedule is parsimonious. $\square$

2.2 Sorting in P-UMH

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

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

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

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

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 P processors, the P-UMH sorting time can be at most P times faster, giving a $\Omega((N/P) \log N)$ 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 $s = \lceil \frac{1}{2} \log_\rho \frac{N}{\alpha P} \rceil$. It then follows that it takes $\Omega((N/P)^{1+c/2})$ time 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 [ViSb]. 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 $\sqrt{x} + \sqrt{y} + b$. It can be shown that the amount of time taken by a UMH to do the same transfer, with multiple levels of the hierarchy active concurrently, is

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

which is bounded by $c(\sqrt{x} + \sqrt{y} + b)$ for all $c \geq \rho/\sqrt{\alpha}$.

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 that operations in the algorithm process all the elements at any level in the hierarchy consecutively. It may be convenient for the algorithm to describe 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 [ViSb] all meet this constraint.

For the second case $b(\ell) = 1/(\ell + 1)$, the upper bound is related to the P-HMM approach for $f(x) = \log x$ [ViSb]. The P-HMM algorithm needs to be modified to reblock the buckets prior to sorting them recursively by substituting Step 8 of the P-BT algorithm of [ViSb] into the P-HMM algorithm. 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.

The upper bound third case $b(\ell) = \rho^{-c\ell}$ makes use of an algorithm based on deterministic, two-way merge sort. This algorithm gives rise to the recurrence relation

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

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 $\Theta((N/P)\log N)$ by a $\log((\log N)/\log P)$ 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 [ACF] 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 [ViSa]); accordingly the upper and lower bounds combine in an interesting way several techniques from [AAC, ACSa, AgV, ViSa].

Theorem 3 *The running times mentioned in Theorem 2 are matching upper and lower bounds for sorting in P-RUMH. The algorithms for nonconstant P 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 RUMH$_{1/(\ell+1)}$ by HMM with access cost function $f(x) = \log x$. Specifically, any block transfer of $\rho^{\ell-1}$ elements from level ℓ to level ℓ' where $\ell > \ell'$ will take an amount of time that is

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

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 RUMH$_{1/(\ell+1)}$ running time. Hence, the lower bound for P-HMM for $f(x) = \log x$ given in [ViSb] also holds for P-RUMH$_{1/(\ell+1)}$. $\square$

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

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

Proof: We prove the lower bounds using an approach similar to that of [ViSb]. Let us define the "sequential time" of a P-SUMH algorithm to be the sum of its time costs for each of the P hierarchies. The sequential time can be at most P 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 [AgV, ViSb], in the following way: For $1 \le \ell \le \frac{1}{2}\log_\rho(N/\alpha P)$, the ℓth two-level memory has one disk, internal memory size $M_\ell = P\alpha(\rho^{2(\ell+1)} - 1)/(\rho^2 - 1)$, 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

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

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

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

as shown in [AgV]. 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:

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

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 P. The $b(\ell) = \rho^{-c\ell}$ case in addition 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 [ViSb], 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. $\square$

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, or whether deterministic algorithms can be found to replace the randomized ones.

References

[AAC] A. Aggarwal, B. Alpern, A. K. Chandra, and M. Snir, "A Model for Hierarchical Memory," *Proceedings of 19th Annual ACM Symposium on Theory of Computing* (May 1987), 305–314.

[ACSa] 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.

[AgV] 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)*, LNCS 267, Springer-Verlag, Berlin, 1987.

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

[ACF] B. Alpern, L. Carter, and E. Feig, "Uniform Memory Hierarchies," *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science* (October 1990), 600–608.

[ACSb] B. Alpern, L. Carter, and T. Selker, "Visualizing Computer Memory Architectures," *Proceedings of the 1990 IEEE Visualization Conference Foundations of Computer Science* (October 1990).

[Lei] T. Leighton, "Tight Bounds on the Complexity of Parallel Sorting," *IEEE Transactions on Computers* C-34 (April 1985), 344–354, also appears in *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, (April 1983), 71-80.

[LuP] 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).

[ReV] 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* (April 1983), 10–16.

[ViSa] J. S. Vitter and E. A. M. Shriver, "Optimal Disk I/O with Parallel Block Transfer," *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing* (May 1990), 159–169.

[ViSb] J. S. Vitter and E. A. M. Shriver, "Algorithms for Parallel Memory II: Hierarchical Multilevel Memories," Brown University, CS-90-22, September 1990.