The Economics of Large-Memory Computations

Clark D. Thomborson
Computer Science Department
University of Auckland
Private Bag 92019, Auckland
New Zealand
tel: (+64 9) 3737 599 x5357
fax: (+64 9) 3737 453
cthombor@cs.auckland.ac.nz

March 17, 1997

Abstract

We propose, and justify, an economic theory to guide memory system design, operation, and analysis. Our theory treats memory random-access latency, and its cost per installed megabyte, as fundamentals. We introduce incentives in our economic theory, and side-constraints in our analytic model of hierarchical memory, to ensure sufficient memory bandwidth and processor speed in any "well-formed" system of a given latency and size.

Our theory suggests that computer users should be charged a "rental" cost, proportional to their use of the total capacity in a hierarchical memory system. This rental cost is a natural unit for algorithmic analysis and, we submit, is a rational basis for pricing.

We use our theory to compare the cost/performance of various large-memory organizations such as PoPCs ("piles of PCs"), NOWs ("networks of workstations"), SMPs ("shared memory multiprocessors"), MPPs ("massively parallel processors"), and even Cray-class vector supercomputers.

Suggested conference track: Performance of Parallel and Distributed Computing Systems.

1 Introduction

The cost of a high-performance computer system is, in many cases, determined more by its memory subsystem (caches, RAM, disk, and the connecting buses) than by its instruction-processing subsystem (CPUs, FPUs, ALUs) [7]. When designing efficient software, memory usage and sharing patterns can be more important considerations than minimizing instruction counts, as noted by many authors, recently including [5, 13, 9]. Traditional algorithmic analysis, however, is based on counting instructions under an increasingly-misleading assumption of "unit-cost" memory accesses. Users have become accustomed to paying for their computations by the CPU-second, even though CPU resources are available at incredibly-low price on desktop systems, albeit with much less memory support than on supercomputers or mainframes.

Our analyses are based on, and serve to justify, the novel idea that computational "work" W can best be measured as the product of the total number of references R multiplied by the size S of the memory space in which these memory references are made:

$$W = RS \tag{1}$$

Our R is a count of references to appropriately-sized blocks, with no temporal locality between these block-references. By "no temporal locality," we mean that the block addresses are uniformly distributed in the space of size S. By "appropriately-sized blocks," we mean that we would increment our reference count R by $\lceil n/B \rceil$ every time a user's code read or wrote n consecutive bytes in the address space. The block size B must, of course, be carefully chosen so that it is appropriate for any contemporary memory technology that could be used to build a system of capacity S. In Section 3 of this paper, we address this difficulty, developing an analytic model of memory latency, bandwidth, and parallelism that matches typical large-memory systems in existence today.

Well-designed user codes exhibit temporal locality. We capture this effect in our analysis by requiring that memory systems be "hierarchical" in their performance characteristics, in the following sense. A memory of capacity S must be supported by a smaller, faster memory which in turn is supported by smaller, faster memories. The recursion ends at the CPU registers. The supporting memories should service enough of the temporally-local references into S that the computer system is latency-bottlenecked on the large, slow (size-S) memory rather than the smaller, faster layers.

Our analyses are based on the belief that most computational problems of economic importance can best be solved on latency-bottlenecked computational systems. We do not analyze CPU bottlenecks and memory-bandwidth bottlenecks, because processor and memory parallelism is relatively easy to increase in contemporary, scalable, computer systems. Latency reduction is, in comparison, an expensive proposition; as persuasively argued in [17], latency forms a fundamental "wall" on system performance.

A narrower reading of our work is independent of a belief in the fundamental nature of latency. In this reading, our economic analyses apply only to codes that are latency-bottlenecked on computer systems that are well-modeled by the method we describe in Section 3. For CPU-bottlenecked codes, traditional measures of algorithmic work (in computational steps or CPU-seconds) are more appropriate. For bandwidth-bottlenecked codes, it would be appropriate to charge users on the basis of their contribution to bus contention or memory saturation.

Yet another valid, but still narrower, reading of our work would largely ignore our analytic model of Section 3. Such a reading would consider only the immediate practical implications of our characterizations of Work and Quality on computer hardware system design, operating system design, and benchmark design. Our metrics would be viewed, in this reading, as an incidental addition to the standard measures of system performance, where each measure is to be considered on its own merits depending on the situation. We have written Sections 1 and 2 to be independent of our analytic model, to aid in such a narrow reading.

For latency-bottlenecked codes, our measure of computational work $W = RS$ is the natural choice. A convenient unit for W on contemporary systems is megareferences · megabyte, or equivalently terareferences · byte. We will abbreviate this unit as a "treb". For example, a million random accesses into a one-gigabyte database is 1000 trebs of work. This is the same amount of work as a thousand references into a one-terabyte database, or a billion references into a one-megabyte database.

To convert trebs to dollars, we must consider the cost and latency of the memory fabric. For example, a large high-performance disk memory (and its associated DRAM and CPU subsystems) might have a total purchase cost of \$0.10 per megabyte. We might plan to amortize this purchase cost over a million seconds of continuous use, or a few weeks, bearing in mind its rapid obsolescence, the impossibility of attaining 100% utilization, and the costs of capital, maintenance and operation. We should thus "rent" disk memory at \$0.10 per megabyte per 10^6 seconds, that is, at 1E-13 dollars per byte·second. If a random access into this memory (to an appropriately-sized block) takes ten milliseconds, our charge should be 1E-15 dollars per byte·reference, or more succinctly, \$0.001/treb.

We formalize the analysis of the previous paragraph by defining the "quality" Q of a memory system as

$$Q = \frac{1}{LC'} \tag{2}$$

where L is its random-access latency (in seconds), and C' is its per-byte purchase cost C amortized over a million seconds. Please note that wee apostrophize C' to indicate that it is a time-derivative of the purchase-cost C. The constant of proportionality in our conversion from C to C' could be adjusted to acheive any desired amortization rate other than our assumed million-second writeoff.

The dollar value V of performing latency-bottlenecked work W on a system of quality Q is

$$V = W/Q \tag{3}$$

under a rational economic analysis.

A convenient unit for expressing Q is thousands of trebs per dollar. We call this unit a "qual." Our formula $Q = 1/(LC')$ thus has the units of milliquals if L is measured in seconds and C' is expressed in dollars per megabyte per megasecond, so that C' is numerically equal to the purchase price C in dollars per megabyte. Alternatively, if L is measured in milliseconds and (as above) C is in dollars per megabyte, $1/(LC)$ evaluates directly into quals.

As indicated above, a contemporary disk memory delivers approximately 1 qual. By way of comparison, a DRAM memory system with 200 nanosecond ($= 2 \cdot 10^{-4}$ millisecond) latency might cost \$10/megabyte, thereby delivering $1/(2 \cdot 10^{-4} \cdot 10) = 500$ quals. That is, a latency-bottlenecked, large-memory, computation will cost about five hundred times less if you can manage to run it as a DRAM-bound job rather than from disk. It will also proceed to completion much, much faster, although we hasten to add that completion rate is a secondary consideration in most of our economic analyses.

Our analysis thus far has neglected one important aspect of modern computing, namely parallelism. A well-designed code will mitigate a latency bottleneck by running many memory accesses simultaneously. We capture this effect in our model with the notion of "effective quality," described below.

We say that a multithreaded (or otherwise parallelised) code has an effective quality of

$$Q_{\text{eff}} = PQ \tag{4}$$

where P is the effective degree of memory parallelism, and Q is the quality of the memory system. Our notation is meant to suggest that the multithreaded code could be charged as though it is running in single-threaded mode on an imaginary memory system of quality Q_{eff}. Another, equally valid, interpretation is that users who multithread their code appropriately will receive a "discount" of $Q_{\text{eff}}/Q = P$.

For example, a user who rents a hundred gigabytes of space, distributed over ten disk devices, should be allowed to run $P = 10$ disk operations "for the price of one" if these operations are issued concurrently to separate devices. If the operations were not issued concurrently, or not issued to separate devices, then the user's effective parallelism P would of course be less than 10.

We are now able to make some rough calculations of the effective quality of typical large-memory architectures available today. For example, a 1024-processor SGI Origin 2000 (S2MP) has a latency L of approximately 2 microseconds into its shared DRAM of at most 2 TB. The effective parallelism into shared DRAM may be as large as $P_{\max} = 4096$, because up to four cache misses can be outstanding per processor at any time [5]. The cost C of this DRAM (and its supporting buses, CPUs, etc.) is, we would imagine, about \$50/MB in large quantities; that is, a 2 TB machine might cost \$100 million. These figures give $Q = 1/LC = 10$ quals for single-threaded (that is, untuned) 2 TB computations. The maximum effective quality $Q_{\max}$ for this architecture is some 4096 times larger, or 40960 quals. The largest actually-acheivable quality Q_{eff} will, of course, depend critically on code design and on architectural constraints that may limit effective parallelism into DRAM. Still, if a computation obtains an effective parallelism of more than 200 on this 10-qual shared-memory architecture (yielding $Q_{\text{eff}} = 2000$), it is economical in comparison to a workstation with 4-way interleaved ($Q_{\max} = 2000$ qual) DRAM.

In Table 1, we have listed our best current estimates for the maximum capacity S in gigabytes, latency L in seconds, system cost C (divided by the maximum capacity) in \$/MB, and the guaranteed not-to-be-exceeded "speed-of-light" effective parallelism $P_{\max}$ for a range of contemporary architectures. We have calculated the quality $Q = 1/(LC')$ for these architectures: this can be interpreted as a lower bound on the effective price-performance for latency-bound, single-threaded computations of size at most S. We have also calculated an upper-bounding $Q_{\max}$ that would be observed if some code actually achieved an effective memory parallelism of $P_{\max}$ on that architecture.

Unless our estimates are wildly inaccurate, one implication is clear: cost-sensitive users with problems requiring 50 GB or more memory should con-

	$S_{\max}$	L	C	$P_{\max}$	Q	Q_{eff}
Workstation DRAM	0.5	2E-7	10	4	500	2000
Workstation disks (4)	8	0.01	0.1	4	1	4
NOW (100 workstations; DRAM)	50	2E-5	20	100	2.5	250
Pile of PCs (100 disks)	200	0.01	0.1	100	1	100
SGI S2MP (1024 PEs; DRAM)	2000	2E-6	50	4096	10	40960
Cray T3E/900 (2048 PEs; DRAM)	4000	2E-6?	50?	2048?	10?	20480?
Cray T932 (32 PEs; SRAM)	8000	1E-7?	500?	8192?	100?	81920?

Table 1: Minimal quality Q and maximum $Q_{\max}$ for contemporary architectures with storage capacity $S_{\max}$ gigabytes, latency L seconds, cost C \$/MB, and (maximum) effective parallelism $P_{\max}$.

sider developing parallelizable code for one of the supercomputers, rather than for a NOW ("network of workstations") or a PoPC ("pile of PCs"). Also, any form of parallelism can yield cost-effective computations in comparison with non-parallel systems, as noted succinctly by Wood and Hill [16]. The wide gap between Q and $Q_{\max}$ for supercomputers should of course be a matter of concern: depending on the memory parallelism P actually acheived, a supercomputer may be an excellent or a very poor choice from a cost-performance standpoint.

It should also be clear from our table that the biggest difference between a NOW and a PoPC is not in the cost of performing work W on these platforms: their memory quality is remarkably similar, in our estimation. The rate at which work progresses on a NOW is about 0.01/2E-5 = 500 times faster than on a PoPC, which would be an important concern to most users.

Our economic bias should be clear by now: we would choose a computer system on the basis of price-performance Q_{eff}, subject to feasibility constraints on $S \leq S_{\max}$ and perhaps on rate L. Less cost-sensitive users would choose on the basis of L, with feasibility constraints on S and Q_{eff}. Our economic theory would cover either case, however for simplicity in exposition we will assume that the reader of this paper shares our economic bias.

PoPC zealots, and others worried by the implications of our Table 1, are welcome to contact us to suggest more appropriate values for their favorite architectures. Our intent here is not to disparage any particular system architecture. Rather we want only to develop a model and theory under which disparate architectures may be compared.

In Section 2, we make more careful definitions, and explore some of the implications of our charging model on system design and operation. Section 3 examines a wide range of contemporary architectures, analyzing their latency, bandwidth, and parallelism characteristics under our model of "well-formed" memory hierarchies. Section 4 contains a summary of our contributions, and an outline of the research frontier in this area.

2 Definition and Implications of Work and Quality

We define latency L using the "back-to-back load" notion of the lmbench benchmark [12, 11]. That is, our L is the multiplicative inverse of the rate at which a chain of pointers can be followed through memory. We do not subtract a CPU clock period from the back-to-back load time, however, as in lmbench's output routines. Also, we do not follow lmbench's fixed-stride assumption. Instead, we assume that the addresses of the cells in the chain are random variables, unpredictable by the CPU or the memory system, uniformly distributed over the entire address range of the memory layer. This suggests that our L is somewhat larger than the unit-stride latency measured by lmbench. Our L is also smaller than some worst-case latencies measurable by lmbench, if that worst-case is encountered only at a small fraction of the possible strides.

The definitional confusion over L, described briefly above, arises because we must make some appropriate assumption about temporal and spatial locality of addressing sequences, in order to make a meaningful definition of latency. Our definition is based on the theoretical model of Section 3, although it may also be justified informally as follows. If there is spatial locality in a reference stream, then it should be accomodated efficiently by choosing an appropriate blocksize. If there is temporal locality in a reference stream, then it should be accomodated efficiently by caching in smaller, faster layers. Thus, to a good first approximation, there should be very little spatial or temporal locality in the reference streams seen in each layer of large-scale, hierarchical memory systems, beyond that captured by the blocksize and caching mechanisms.

We are not overly concerned with modelling stride-k performance, for $k > 1$, as a phenomenon that is somehow distinct from other forms of memory parallelism. For example, we would count a stride-8 load of vector length 100 as 100 independent one-word loads. Most high-performance memory systems will exhibit large effective parallelism P under such conditions. We fully appreciate that predicting effective parallelism, given a particular code, system, and system load, is a very difficult question and seems to require very detailed, system-specific models at present. Our goal in this paper is not to predict effective parallelism P, but merely to point out some of the economic ramifications of the variability in P.

A final, and fundamental, difference between our latency L and that measured by lmbench is that we would measure a block-load latency, where the blocksize B is suitable to the memory layer in question. We will return to this point later in this section, for now noting only that in Section 3 we will define the appropriate blocksize as $B = L/G$, where G is the average "gap" (measured in seconds/byte) [4] in the data stream resulting from the memory operation. An appropriate blocksize, thus, is one in which the memory operation runs at about 50% efficiency, incurring a latency of L seconds and a transfer-time of BG seconds. Our nominal blocksize B is thus equal to the $n_{1/2}$ parameter in common usage when characterizing memory systems performance.

For convenience, we would measure L only on read operations. We would insist, however, that write operations have similar latency, otherwise we

would increase L to the extent necessary to obtain a "well-formed" memory system.

Our analytic variable C' is the cost per megabyte per second of whatever memory device is used to implement the memory layer being analyzed. We suggest the use of a fully-depreciated cost including an allowance for the buses, backplanes, etc. that are required for memory upgrades. For example, a commodity DRAM chip might cost \$10/megabyte at the present. In a computer center installation, it might be appropriate to use a C' for this chip of, roughly, a microdollar per megabyte per second, that is $C' = 10^{-6}$ in our usual units of dollars per megabyte per second. If the chip is rented continuously at this charge, its purchase price would be recovered in about three months (10^7 seconds). In another few months, it should be possible to recover the cost of the motherboard, etc.; thus, within a year, assuming a minimum 50% occupancy, the chip will be "turning a profit."

Memory in office workstations has low usage rates, so the C' in such systems might be scaled upwards in inverse proportion to the "expected occupancy" of memory rentals. In this paper, however, we will assume for simplicity that all memory architectures would have similar "occupancy rates." As noted in Section 1, we somewhat arbitrarily choose 10^6 seconds as our amortization base, thereby giving office workstations, and other low-utilization installations, a possibly-unjustified advantage over high-utilization systems in our rough calculations. More accurate C' could be obtained, for any given installation, by more careful economic analysis.

An innovative implication of defining quality $Q = 1/(LC')$ is that it provides a metric to compare and contrast organizations for large, scalable computing systems. It is not at all difficult to calculate a rough value of Q for any memory system design: we need to know only its purchase cost per megabyte and the latency. All other factors being equal, the organization with the larger Q for a particular memory size of economic importance will be preferred. The devil is in the details of course: what are appropriate side-constraints on memory bandwidth, processor power, and hierarchical memory performance, so that the different computing systems are indeed comparable? Other researchers, of course, have considered this question of "balance" between various performance measures, notably McCalpin [10].

In this section we will assume, perhaps naively, that contemporary architectures are reasonably well-designed, and thus that side-constraints are unnecessary. However, if our $Q_{\text{eff}} = P/(LC')$ metric of effective quality, for jobs with memory parallelism P, is used to design or market systems in the future, then side-constraints and/or other performance metrics are an absolute necessity. Otherwise we may find ourselves purchasing a system with huge amounts of low-latency, highly-parallel memory, without sufficient memory bandwidth and CPU resources to accomplish any useful task.

Probably the best approach to defining and enforcing side-constraints, other than the analytic technique suggested in the next section, or the balance formulas developed by other authors, would be to develop a benchmark suite of problems with known work W. As a simplistic example, we could model a large, batched, database-query problem as a million 1000-way gather operations ($R = 10^9$) on a 100 gigabyte dataset ($S = 10^{11}$) composed of a hundred billion 1000-byte records. The addresses in these op-

erations should obey some reasonable (perhaps Zipfian) locality-of-reference rule, to permit some (small) speedup from caching the results of prior gathers. Also, the addresses for each gather operation should be computed as a function (perhaps XOR) on the result of the previous gather, to prevent more than 1000-fold memory parallelism. This benchmark would require at least $W = RS = 10^9 \cdot 10^{11} = 10^8$ trebs (trillions of reference-bytes) of work on any computer, as long as we all agree that a "reasonable" blocksize for a 100 GB memory must be at least 1000 bytes. Somewhat more work W may be required on inefficient systems, as discussed below.

A system costing C dollars that completes 10^8 trebs of work in T seconds must be delivering $10^8/(C'T/10^6)$ trebs/dollar, that is, its effective quality Q_{eff} is $10^{11}/(C'T)$ quals. Note: as mentioned in Section 1, a qual is 1000 trebs/dollar; and as discussed in this section, we amortize system purchase costs over 10^6 seconds.

If bottlenecks such as inefficient operating-system paging policies, memory bandwidth, synchronization, or address-translation impede the workflow, then this will be reflected in the Q_{eff} calculated for the known-W benchmark suite. These bottlenecks could be discovered and perhaps repaired more easily, we believe, if the W measure were reported by work-sensitive system accounting routines.

Our work-sensitive vision of system accounting is easily described. Each user should be notified of the work $W = RS$ done on their behalf by the operating system during each reporting interval, which might conveniently be a second, an hour or a day. In order to allow such a report to be made, the computer system must count each user's memory references on each layer of the memory hierarchy, and estimate their memory occupancy S_i at the time of each reference at memory layer i. The work-update rule is thus $W_i = W_i + S_i$, triggered whenever a reference occurs on layer i. Ideally, this statistic would also be gathered for each of the user's process groups, processes and threads to aid in performance tuning.

The nonlinearity of our work-measurement rule may be somewhat surprising: it suggests, for example, that users with large memory allocations should be charged more for each cache miss than are users with small memory allocations. The cost-sensitive user would certainly notice, and complain, about the charges under this scheme if their job were run with an inefficiently-large memory allocation! Similarly, if their job were run with an inefficiently-small memory allocation, they would incur extra charges under this regime due to excessive page-fault activity. To put it another way, charging by work-done would put pressure on designers to deliver systems capable of work-efficient operation. It would also put pressure on users to run codes that are capable of being run efficiently, and to choose systems that can run their codes efficiently.

Our favourite proposal for system charges is somewhat simpler to explain and to implement. Each user should "rent" their memory capacity at a fixed charge per megabyte-second. This rental charge should be set high enough to cover an appropriate amount of "active usage" of this memory. Roughly speaking, someone who is renting half of a memory layer (say, occupying half of the capacity of a paging device) should get half of the total service (e.g. page faults/second) available from that layer. In our economic model,

it is easy to establish an appropriate rental charge. Someone renting space S should be charged C' dollars per megabyte-second. This user should be assured of the possibility of obtaining up to P/L memory references per second, at a latency of L, if their code is properly tuned to take advantage of the available parallelism. Exact values of P and L may be difficult to obtain, but roughly-appropriate values could be obtained by analytic means (see our Section 3), an appropriate benchmark, or by observation of the effective quality being delivered on this system in some prior period.

Our "rental" scheme for charging has the virtue of simplicity, but it does pose some risks. The user must trust the computer center to provide systems capable of work-efficient operation, and to operate these systems in a work-efficient manner. The computer center must trust the user, or better, their resource scheduler, not to allow CPU-bound and bandwidth-bound jobs to impede the progress of latency-bound ones. Only the latter "pay the rent." The others will get a "free ride" under this accounting scheme, implying that our rental scheme will only be attractive to computer centers if and when most jobs of economic importance are latency-bottlenecked. We will return to this question of market analysis again, briefly, at the end of this section after discussing one other charging scheme that is admissible under our economic model.

A work-sensitive operating system should, we believe, collect statistics on the effective memory quality $Q_{\text{eff}} = P/(LC')$ being delivered on each layer of memory to each of a user's threads, processes, and process groups. This will help knowledgeable users understand the charges they incur, and perhaps give them enough information to tune their codes. The value C' is a constant, depending only on the layer in question. The value P is an estimate of the effective parallelism (queue length, or number of outstanding requests) for that thread, process or group on that memory layer. Ideally, this would be evaluated at the time of each reference-completion, along with the latency L incurred by that reference and the time t since the previous reference-completion. A suitable update rule would then be $Q = (1 - t/T)Q + tP/(TLC)$, where T is an appropriately-large constant for sporadic time-averaged reporting. Each time this report is printed, the knowledgeable user will also want to see the current total W and the difference in W since the last report. These last two statistics are the analogues, in the work-accounting scheme, of total CPU-seconds and %CPU in traditional system accounting.

To aid in code-tuning, quality $Q = P/(LC)$ values might be averaged over distinct, fixed-length, time intervals and reported as a time-series to a user interested in profiling a task's memory performance. If work $W = RS$ values were also averaged and reported over the same intervals, then a knowledgeable user would be able to spot time periods in which their code was not running efficiently. Additional statistics, especially the corresponding time series for P, L and S would help to diagnose the problem.

These measurements of Q and W seem to us to be feasible for all existing systems, with the possible exception of the fastest (processor cache) layers of the memory hierarchy. If measurements on the fast layers are infeasible, we would recommend estimating them by inference from the usual CPU-second measurement, under the assumption that the CPU is fully saturating its

caches at all times.

We can now state our third, and most complex, proposal for a rational charging structure. Each user could be assessed S/Q dollars each time a memory reference is made on their behalf, on each layer of memory for which S and Q statistics are being collected.

The alert reader may have noticed that, in two of our charging schemes, we are proposing to charge users for their memory usage on all layers. This is an apparent shift from our fundamental economic analysis, which in essence suggested that the cost of a system could be amortized over the usage of only its largest (more precisely, its most costly) memory layer.

One justification for charging on all layers rather than on one is that it allows us to charge a little bit for (the cache references of) CPU-bound jobs with tiny memory footprints. Such jobs should, under our analysis, be run so far "in the background" that they do not materially affect the progress of some huge-memory job whose presence justifies the large capital expense of a large-memory system. This may not be possible, or actually economic, on contemporary systems, due to (uninformed??) market demand for CPU cycles. Furthermore, the tiny CPU-hogs don't "really" belong on an ideally-configured large-memory system, as should be clear from our economic analysis, but systems can never be ideally configured for all the memory load patterns they will encounter. There will, on occasion, be more CPU cycles, cache capacity, DRAM capacity, etc. on any large-memory system than can efficiently be used by the large-memory jobs.

From a theoretician's perspective, there is only a "constant factor difference" in our charging regimes. As long as there are only a constant number of chargeable memory layers, and as long as the most-expensive layer has a non-zero load average, then it doesn't matter "to within a constant factor" whether we charge only for usage on the most-expensive layer, or for usage on all layers. If the most-expensive layer has a tiny load average, and if it is the major determinant of system cost, then the system is hopelessly uneconomical and cannot be salvaged by renting its less-expensive layers under any rational charging regime.

The choice of the most appropriate charging scheme should thus, in our opinion, be left to market analysts and systems designers. We are satisfied to point out several rational alternatives, each of which has varying virtues. Our favourite space-rental scheme is quite simple. Our work-charging scheme (under an assumed constant Q) is arguably inappropriate when running custom codes, because the effective Q for these codes will be difficult to estimate accurately. It might, however, be appropriate for users running standardized, well-tuned codes with predictable Q: perhaps linear-program solvers might fit in this category. Our final proposal, that of charging by S/Q with both S and Q being runtime estimates, has many non-linearities hence will not be easily comprehended. It has the signal virtue, from the users' perspective, of putting responsibility squarely on the shoulders of the computer center to find some economic way to run any code, no matter how wildly variable its memory demands may be.

3 An Analytic Model of Memory

The key variables in our performance model are space S, latency L, and effective parallelism P. We obtain an economic model by introducing a constant of proportionality C', and defining quality $Q = 1/(LC')$.

There are many other performance variables of interest, notably gap G and blocksize B. Single-stream memory bandwidth is $1/G$ and blocksize $B = L/G$, to a good first approximation, so either G or B is a dependent variable.

A complete model of performance would also consider the various "overheads" for address translation, synchronization, interprocess communication, etc. At the risk of severe oversimplification, we will ignore the overhead variables in this section, concentrating only on analytic models for space, latency, available parallelism, and gap. Because we have ignored overheads, the actually-obtained parallelism in any realistic setting will always be somewhat less than the available parallelism P predicted by our model.

We suggest that our model be judged on its simplicity, accuracy, and number of independent variables. Ideally, there would be just one independent variable S. It does not take many observations of actual systems to discover, however, that any model that predicts L from S will be woefully inaccurate. Some perfectly-reasonable multi-gigabyte systems are based on disk technology, and therefore slow. Others are based on DRAM, and are therefore more expensive but much faster.

The next hope is to predict P and G given L and S. We tentatively conclude that this is possible to accuracy within a factor of ten or so, at the present time, at least for systems based on high-performance microprocessor chips. The outlook is unclear for our model on vector processors such as the Cray T932; and we certainly must make at least minor adjustments to handle low-end PCs (based on low-performance microprocessors). This section should best be read, then, as a report on work-in-progress.

We note at this juncture that our analytic model has heaps of competition. The influential LogP model [4] has many merits, but tells nothing about how to predict G given L and P. We could nonetheless use the LogP model to justify a charging structure for a single layer of memory (one with blocksize L/G), if we charged the o overhead to some "CPU" budget.

A second "gap" parameter has been proposed recently, to widen the applicability of the LogP model [1]. We could use the resulting LogGP model to handle two-layer hierarchies with two different blocksizes, one for each "gap" value. Another non-hierarchical model was recently proposed by Hambrusch and Khokhar [6] for coarse-grained parallel machines. Maggs, Matheson, and Tarjan give an excellent survey of the state of the art, along with a tentative argument that two parameters would suffice to model current systems. Their parameters are the number of processors P and the interval I, which we read as $I = L \approx G$, implying $B = O(1)$. Thus, although this model is not quite accurate enough for our taste, it would support a rough version of our economic analysis.

We make special note of the PMH (parallel memory hierarchy) model [3] and the earlier UMH (uniform memory hierarchy) [2], as our extensive discussions with some of its authors have heavily influenced our thinking

on this subject. We part company, slightly, with Alpern and Carter by suggesting that there are $\log \log S$ layers in a memory hierarchy of size S, rather than $\log S$.

Our model is, in some respects, an extension to the one recently published by Jacob, Chen, Silverman and Mudge [8]. They characterize available memory technology with two parameters, essentially our L and C, then show how optimally to construct a cost-effective hierarchy by considering only latency effects. Our model additionally predicts blocksizes and parallelism, which could be used to introduce feasibility constraints into their analysis.

In our model, as noted above, there are $\log \log S$ layers in a memory of size S. We must establish appropriate bases for the logarithms. The size of the first layer is fixed by the size of the register set of whatever CPU is used as a building-block. For this reason, we assume that $S_0 = 256$ bytes is an appropriate value for the first layer, giving us a model with $\log_\delta \log_{256} S$ layers for some as-yet-unknown δ.

It is at least convenient for the user, and we believe essential for efficient programming practice, that there be an integral number of levels in the hierarchy. In this writing, we will not attempt to justify our assertions about efficient programming, referring the interested reader to our other publications, notably [14]. Our present thinking is that δ is approximately $\sqrt{2}$ in most contemporary architectures, so we define

$$h = \lceil \log_{\sqrt{2}} \log_{S_0} S \rceil \tag{5}$$

with

$$\delta = (\log_{256} S)^{1/h} \tag{6}$$

and

$$S_0 = 256 \tag{7}$$

for microprocessor-based systems. For the Cray T932, the value $S_0 = 8192$ is probably more appropriate.

With δ defined as above, we can define the sizes of the intermediate layers in our hierarchy:

$$S_i = S_0^{(\delta^i)} = (S_{i-1})^\delta \tag{8}$$

This yields, as desired, $S_h = S$.

In hierarchical models of memory, it is customary (and seems accurate) to work on a principle of "self-similarity": that layer i behaves much like layer $i-1$, except that the blocks are somewhat bigger, the latency is somewhat larger, and the bandwidth is typically somewhat smaller. Accordingly, if latency L_i is defined as the latency for a transfer from layer i to layer $i+1$ (be careful here! different authors have different conventions!), we write

$$L_i = L_0(S_i/S_0)^\alpha \tag{9}$$

where an appropriate value of α can be determined by noting that $L_{h-1} = L$ and

$$L_0 = 10^{-8} \quad \text{seconds} \tag{10}$$

for contemporary high-performance microprocessors, yielding

$$\alpha = \frac{\ln(L/L_0)}{(\ln 256)\delta^{h-1}} \tag{11}$$

Similarly, we can define blocksizes B_i as power functions on a current technological parameter B_0 (the size, in bytes, of a minimally-efficient transfer to register from L1 cache) and some as-yet-unknown β:

$$B_i = B_0(S_i/S_0)^{\beta} \tag{12}$$

with

$$B_0 = 8 \tag{13}$$

for microprocessor chips and, apparently, Cray-class supercomputers (due to their gather-scatter facility).

We have found, from observation of existing systems, that

$$\beta = \alpha/2 \tag{14}$$

is a reasonably-close approximation to current design: systems with fourfold less latency seem to have about half as much gap (that is, twice as much bandwidth). Note that $G_i = L_i/B_i$, by definition, so the gap behaves as

$$G_i = (L_0/B_0)(S_i/S_0)^{\alpha-\beta} \tag{15}$$

We have also observed, tentatively, that the maximum available parallelism seems to have a power-function exponent $(1-\beta)/2$ with constant of proportionality near unity:

$$P_i = (S_i)^{(1-\beta)/2} \tag{16}$$

On theoretical grounds, this is not an unreasonable design constraint. It implies that parallelism grows as the square root of the total number of blocks in a layer.

Almost certainly, actually-existing memory technology will not allow cost-efficient system constructions that exactly match the prescriptions of our model. In particular, we are quite aware of the long-standing and problematic "access gap" between the DRAM and disk layers of memory. Furthermore, our analytic model will be invoked for several different users with differing S (and possibly Q) simultaneously on the same system. Such users should each be given an appropriate computing surface, that is, each user address space on a multiprogrammed system should have layer capacities, latencies and available parallelism congruent with our analytic model. We would thus need to prove a set of "simulation" lemmas and theorems, and to develop operating systems and hardware to realize these simulations, before claiming that our analytic model is fully successful.

In this paper, we will restrict ourselves to making the appropriate definitions, leaving the proofs and implementations for later work. We define a "well-formed" memory hierarchy of size S as one that is capable of meeting or exceeding the computing surface (L_i, P_i, G_i, S_i, $0 \le i \le h$) defined above for any set s_j, q_j of user demands such that $\sum_j s_j \le S$ and $\max_j(q_j) = Q$. Trivially, a hierarchy "meets or exceeds" a computing surface if it has equal or lower latency, equal parallelism, equal or less gap, and equal or greater available size on some layer $i' \le i$. Not quite so trivially, a hierarchy with performance ($l_{i'}$, $p_{i'}$, $g_{i'}$) on any layer i' can be "time-sliced" to meet parameters (L_i, P_i, G_i) if $l_{i'} \le L_i$ and $p_{i'} l_{i'}/g_{i'} \ge P_i L_i/G_i$.

There isn't much point in proving simulation results on an unverified model. We turn now to the question of accuracy.

We developed our model, admittedly with some danger of "overfit," by considering the systems similar to the ones appearing in Table 1. The interested reader may wish to access our Excel spreadsheets, through our homepage `http:\\cs.auckland.ac.nz\~cthombor\`, to try out our model for various parameter settings.

In Table 2, we present the relevant data from our NOWDRAM.XL spreadsheet, parameterized on $S = 2^{37} \approx 140$ gigabytes of distributed DRAM and $L = 20$ microseconds of latency into this size-S space. We might hope to implement this system with a couple of hundred high-performance workstations, each configured with about 500 MB of DRAM and a very-high performance network interface.

In Table 2, we can recognize the CPU registers in Layer 0. Their parameters are fixed by our assumptions about L_0, B_0 and the functional form of P_i: there is no dependence on L or S. Note that our model specifies 7.3-way parallelism and 800 MB/sec bandwidth for individual 1-word register-cache transfers; the reality, at least for the R10000 CPU, is 4-way parallelism. Layers 1 and 2 are recognizable as (quite modestly-sized) cache footprints; we note, in passing, that surprisingly small caches are also found in the Jacob-Chen-Silverman-Mudge model [8]. Layer 3 might be the TLB-mapped portion of DRAM. Layer 4 is the remainder of processor-local DRAM; the 1.2 microsecond latency of transfers between Layer 3 and Layer 4 seems slightly high in our model, but not by more than a factor of three. The modelled bandwidth for these transfers to DRAM, 87 MB/sec, is accurate for some contemporary workstations but somewhat low for others. (We will discuss the parallelism to DRAM in the next paragraph.) Layer 5 is our shared, non-local DRAM space. Note that our model predicts a bisection bandwidth of 15.3 GB/sec, an effective parallelism of 854 (about 4 transfers pending per processor, assuming about 200 processors), and about 18 MB/second of bandwidth per transfer. This bandwidth is probably a factor of four too high, even for an aggressive NOW design: it would imply that each processor's network interface can handle an aggregate of about 72 MB/second.

Our analytic treatment of parallelism may have confused some readers. Perhaps it is best to explain it by example, referring to Table 2 under the assumption that Layer 0 of that table is accurate. The natural implementation of the NOW under this assumption would use about $854/7.3 = 117$ processors, each capable of handling 7.3 threads. The system should be organized into groups of about $144/7.3 = 20$ processors to form about $854/144 = 6$ independent layer-4 implementations, that is, to provide for each processor group a coherent, shared address space of 1.6E8 bytes providing a load/store latency of 1.2 microseconds from layer 3.

So far, so good: instead of imagining our NOW to be built from about 117 single-processor workstations, we can imagine it to be built from six twenty-processor workstations. This, however, is not the usual interpretation of the NOW architectural paradigm; nor is a memory bandwidth of 10.6 GB/sec, shared among 20 processors, a good description of a contemporary workstation.

i	S_i	L_i	B_i	1E-6/G_i	P_i	1E-9P_i/G_i
0	256	1E-8	8	800	7.3	5.8
1	1868	3.1E-8	14	454	14.8	6.7
2	2.8E4	1.5E-7	30	210	39	8.2
3	1.1E6	1.2E-6	87	74	144	10.6
4	1.6E8	2E-5	360	18	854	15.3
5	1.4E11					

Table 2: Analytic model for a network of 200 workstations, with $S = 2^{37} =$ 140 gigabytes and latency $L = 20$ microseconds. Our model parameters are $\alpha = 0.570$, $\beta = 0.285$, $\delta = 1.358$. We tabulate the layer number i, the size S_i in bytes, the latency L_i in seconds, the blocksize B_i in bytes, the serial bandwidth $1/G$ in megabytes per second, the maximum available paralellism P_i, and the maximum bandwidth available for parallel transfers 1E-9P_i/G_i in gigabytes per second.

Alternatively, we could build a usable NOW that does not provide coherency below layer 5 at the latencies suggested by Table 2. Applications developers for such a NOW should be warned away from relying on low-latency coherence at layer 4, at peril of obtaining low effective parallelism and hence low efficiency. If low-latency coherence is not required at layer 4, it could be implemented by 117 fully-independent banks of DRAM, each of size 1.6E8 bytes. The memory bandwidth into each bank is 10.6 GB/sec, which is a factor of six beyond the fastest contemporary workstations, but the other values are well within reach.

Our Table 2 prescribes parallelism 39 for transfers between layer 2 and layer 3. In a NOW built of single-processor workstations coherent only at layer-5 latencies, bandwidths and blocksizes, we must either provide effective parallelism 39 between layers 2 and 3 for each processor, or "time-slice" into a layer-3 memory that is considerably faster than 150 nanoseconds. Fortunately a solution is at hand here: we could implement layer 3 from SRAM rather than DRAM, that is, we would provide each processor in our NOW with approximately 1.1 MB of secondary cache. If this cache had, say, 15 nanosecond latency, then we could provide a "39-way parallel, 150-nanosecond" performance on layer 3 as follows: with parallelism $4(150/15) =$ 40 by ten-way "time-slicing" of references on a processor that allows four outstanding L1 misses without a stall.

Transfers between layer 1 and layer 2, in Table 2, occur at parallelism 14.8 and latency 3.1E-8. These figures are consistent with a 4-way parallel processor core and a layer 2 implementation by a $4(3.1\text{E-}8)/14.8 = 8$ nanosecond L1 cache of capacity 28 KB.

As noted above, our model has no predictive power for transfers between layer 0 and layer 1. As microprocessor technology changes, the constants B_0 and L_0 in our model should be adjusted, and possibly a new parameter P_0 should be introduced.

We conclude that all our parameters for the NOW are within a factor of six of being accurate. The tightest performance constraint is on the

bandwidth into non-TLB-mapped DRAM memory. We also note that NOW programmers should not assume coherence at layers 4 or below will occur at the modelled latencies of those layers.

Our findings are similar on other architectural paradigms: we believe that our analytic model would be accurate to within a factor of ten or so on all current architectures based on high-performance microprocessors. The interested reader is invited to visit our homepage `http:\\cs.auckland.ac.nz\~cthombor\` to take copies of our spreadsheets NOWDISK.XL (for shared memory built from disks on a NOW), POPC.XL (for shared memory built from disks on a "pile of PCs"), WORKDRAM.XL (for single-processor workstation DRAM), WORKDISK.XL (for single-processor workstation disk, not an economical choice for latency-bottlenecked computations), ORIGIN1024.XL (a maximally-configured SGI Origin 2000 S2MP), T3E.XL (a maximally-configured Cray T3E/900), and T932.XL (a maximally-configured T932, a 32-processor vector architecture). The last is still giving us trouble; your comments are especially invited here.

4 Summary and Open Questions

We have defined a novel concept of work $W = RS$ as the product of random-access block-references into a space of size S. Strictly speaking, this concept is only applicable to latency-bottlenecked tasks. If appropriate side-constraints on CPU and memory bandwidth are enforced, however, work W would be a fully-general measure. Alternatively, work W should be seen as the analogue (for latency-bottlenecked tasks) of CPU-seconds and total memory bandwidth (for CPU-bottlenecked and bandwidth-bottlenecked tasks, respectively).

We have defined memory quality $Q = 1/(LC')$ where L is the latency and C' is the fully-amortized per-byte cost of memory. We further defined $Q_{\text{eff}} = P/(LC')$ as the effective quality of a computational task acheiving (memory) parallelism P. This notion of cost-performance is generally appropriate for latency-bottlenecked systems, that is, for systems with sufficiently-inexpensive memory bandwidth and CPU resources. In other settings, Q and Q_{eff} should be viewed as analogues of traditional measures of cost-performance, for example MFLOPS/$, MOPS/$ and MBW/$ for FPU-, CPU- and memory bandwidth-bottlenecked systems respectively.

We made rough estimates of the quality of several current architectures, including piles of PCs (PoPCs), networks of workstations (NOWs), shared-memory multiprocessors, etc. These estimates could be refined by someone with more knowledge of the actual parameters of existing systems. More interestingly, we plan to develop benchmarks to measure quality directly. We require assistance here: please contact us if you are interested in this project.

Our notion of memory quality supports many economic analyses. We briefly outlined its implications for charging regimes at computation centers. In particular, we toyed with an extreme position, that computation centers should "rent" memory space at a fixed charge per byte-second, making no extra charge for standard "computational services" on this space (references, bandwidth, CPU cycles, etc.). In the end, we washed our hands of this

matter, in the belief that the choice among rational (and even irrational) charging regimes is best left to market analysts and systems designers. We would, however, be very interested in collaborating on future research in this area.

We sketched some performance monitoring techniques that would, we believe, be useful to anyone interested in developing work-efficient codes. We intend to write user-level libraries to provide some primitive support for these techniques on standard-issue workstations and, perhaps, on an SGI Power Challenge. We heartily welcome others' advice and assistance on such projects, and on more ambitious schemes that would require assistance from hardware or OS kernel routines.

We defined an analytic model of performance that seems capable of predicting hierarchical bandwidths, parallelism, latencies and sizes, given top-level memory size S and latency L, to within a factor of ten, for existing systems based on high-performance microprocessors. Although our model is still somewhat sketchy in its details, it gives concrete support to the notion that there exists an "appropriate blocksize" for a random reference. Lacking such a model, or some other (benchmark-based?) agreement on blocksizes, our definitions of work W and quality Q would be incomplete. We invite your comments and criticisms of this model, and suggestions for its application. In particular, we would be very interested to explore the connections, and possible cross-fertilization, between our model and the model and analysis of Jacob, Chen, Silverman and Mudge [8].

We intend, in future work, to explore some of the ramifications of our performance model on algorithmic design and analysis. Some of our planned techniques are prefigured in our recent technical report [14], in particular our reliance on prior results in other models [3, 2, 15].

References

[1] Albert Alexandrov, Mihai Ionescu, Klaus Schauser, and Chris Scheiman. LogGP: Incorporating long messages into the LogP model. In *Proceedings of the 7th Annual Symposium on Parallel Algorithms and Architectures (SPAA)*, 1995.

[2] B. Alpern, L. Carter, E. Feig, and T. Selker. The uniform memory hierarchy model of computation. *Algorithmica*, 12:72–109, 1994.

[3] Bowen Alpern and Larry Carter. Towards a model for portable parallel performance: Exposing the memory hierarchy. In *Portability and Performance for Parallel Processors*. John Wiley & Sons. to appear.

[4] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. LogP: Towards a realistic model of parallel computation. In *Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming*, pages 1–12, 1993.

[5] Jeff Fier. Origin2000 (tm) performance tuning and optimization. Technical Report 007-3430-001, Silicon Graphics, Inc., http://www.sgi.com/techpubs/, 1996.

[6] Susanne Hambrusch and Ashfaq Khokhar. C^3: An architecture-independent model for coarse-grained parallel machines. *Journal of Parallel and Distributed Computing*, 32(2):139–154, 1996.

[7] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., 1990.

[8] Bruce Jacob, Peter Chen, Seth Silverman, and Trevor Mudge. An analytical model for designing memory hierarchies. *IEEE Transactions on Computers*, 45(10):1180–1193, October 1996.

[9] Anthony LaMarca and Richard Ladner. The influence of caches on the performance of sorting. In *Proceedings of the Eight Annual ACM-SIAM Symposium on Discrete Algorithms*, January 1997.

[10] John D. McCalpin. A survey of memory bandwidth and machine balance in current high performance computers. *IEEE TCCA Newsletter*, December 1995.

[11] Larry McVoy. lmbench homepage, 1997. http:// reality.sgi.com/ employees/ lm/lmbench/lmbench.html.

[12] Larry McVoy and Carl Staelin. lmbench: Portable tools for performance analysis. In *Proceedings of the 1996 Usenix Conference*, To appear.

[13] Shashi Shekhar, Sivakumar Ravada, Vipin Kumar, and Douglas Chubb. Parallelizing a GIS on a shared address space architecture. *IEEE Computer*, 29(12):42–48, December 1996.

[14] Clark Thomborson. When virtual memory isn't enough. Technical Report CS-TR-136, Computer Science Department, Auckland University, November 1996.

[15] J.S. Vitter and E.A.M. Shriver. Algorithms for parallel memory, ii: Hierarchical multilevel memories. *Algorithmica*, 12:148–169, 1994.

[16] David Wood and Mark Hill. Cost-effective parallel computing. Technical Report CS-TR-94-1245, Computer Sciences Department, University of Wisconsin-Madison, 1994.

[17] William Wulf and Sally McKee. Hitting the memory wall: Implications of the obvious. *Computer Architecture News*, 23(1):20–24, March 1995.