Predicting and Evaluating Memory Communication Performance[1]

Kirk W. Cameron[2], Member, *IEEE*, Xian-He Sun[3], Senior Member, *IEEE*,

Surendra Byna[3], Student Member, *IEEE*, Rong Ge[2], Student Member, *IEEE*

Abstract–Application of hardware-parameterized models to distributed systems can result in omission of key bottlenecks such as the full cost of inter-node communication in a shared memory cluster. However, inclusion of message characteristics and complex memory hierarchies may result in impractical models. Nonetheless, the growing gap between memory and CPU performance combined with the trend toward large scale clustered shared memory platforms implies an increased need to consider the impact of local memory communication on parallel processing in distributed systems. We present a simple and useful model of point-to-point memory communication for use in performance prediction and evaluation. We illustrate the utility of the model in three ways: 1) to predict and analyze the latency of memory copy, pack and unpack in complex memory hierarchies, 2) to isolate latency delays due to middleware overhead in an MPI implementation and 3) to more accurately predict the computational cost of scatter operations in a distributed system. We use our model to isolate the contributions of hardware, middleware, and software to data transfers on Intel- and MIPS-based platforms.

Index Terms–performance modeling and prediction, memory hierarchy, distributed systems

1 Introduction

Communication promises to remain a critical bottleneck in the performance of distributed applications for many generations of architectures. A communicated message must be moved from the source's local memory to the target's local memory. *Memory communication* is the transmission of data to/from user space from/to the local network buffer (or shared memory buffer). *Network communication* is data movement between source and target network buffers. Communication cost consists of the sum of memory and network communication times.

---

[1] An earlier version of this paper appears in Proceedings of the 2003 International Parallel and Distributed Processing Symposium, Nice, France.
[2] Department of Computer Science and Engineering, University of South Carolina, Columbia, SC 29208.
[3] Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616.

Memory communication overhead is the time the CPU is engaged in local data movement during which no other work can be accomplished. Figure 1a shows how models such as LogP [7] approximate memory communication in parallel systems with a fixed overhead parameter (o), the reciprocal of the bandwidth between application and network buffers[4]. Generally, when network communication (L) dominates cost, LogP cost prediction is accurate since ignoring the impact of message size and distribution on memory communication overhead is reasonable.

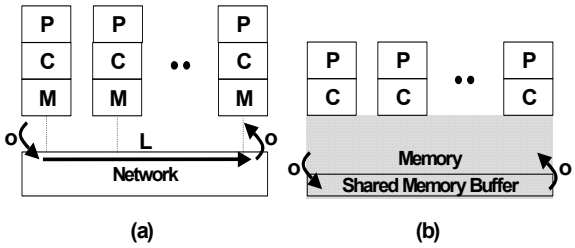

Fig. 1. P=Proc, C=Cache, M=Memory. Hardware parameterizations of communication costs using LogP are effective in parallel systems (a) when network communication dominates overall cost. As the impact of memory communication on overall communication increases, exemplified in shared memory (b), effects of data size and distribution are too significant to ignore.

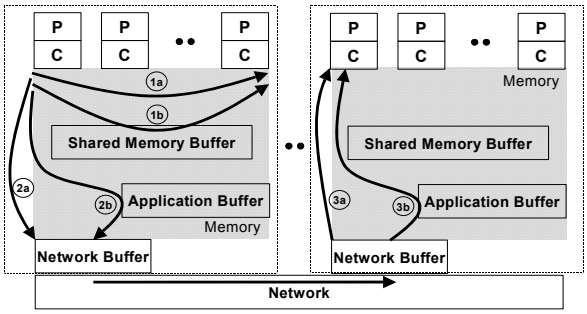

Fig. 2. Memory communications within shared memory (1a-b) and to/from the network buffer in distributed communication (2a-b/3a-b) follow critical paths dependent upon data size, data distribution, and system implementation.

Communication cost in shared memory architectures (Figure 1b) is dominated by memory communication. Years of research addressing the cpu-memory gap [28] have resulted in very complex memory hierarchies that overlap latency with useful work. Using a single hardware bandwidth parameter to model memory communication performance in a hierarchy is not sufficient since the effective latency overlap of a hardware implementation is application and system dependent. More accurate models of communication that incorporate these characteristics are warranted when memory communication has significant impact, however resulting models must remain simple despite the complexity of current memory hierarchies in order to be of use in

[4] We employ a common simplification of the LogP model, letting o=g for simplicity in our discussion.

algorithm design (e.g. user- and application-level prediction and analysis) or in system software development (e.g. time-saving abstractions/predictions in a parallel compiler or system simulator). In this paper we focus on use of our models in algorithm design and analysis though their applicability is more general.

Memory communication cost will increase in proportion to overall communication cost given current technological trends: Future generations of distributed systems will consist of commodity, shared memory components with fast interconnects [5, 19]. Improvements in memory speed will continue to lag behind improvements in processor and network-interconnect technologies. Further algorithm development may take advantage of network latency tolerance via technologies such as active messages [26].

There are two compelling reasons to incorporate data characteristics into models of memory communication cost in distributed systems. First, the number of transmissions between source and target user space in shared memory (see Fig. 2 1a-b) or between user memory space and the network buffer in distributed transmission (see Fig. 2 2a-b), will vary with data size, data distribution, and system implementation. Reasons for re-buffering include system design considerations such as finite sized network buffers and system software implementations such as temporary storage of data for packed transmission to the network or shared memory buffer.

Second, by identifying costs that exhibit overlap potential, we encourage efficient algorithm design. In the LogP model, overhead (i.e. memory communication) is the cost that cannot be avoided. Latency (i.e. network communication) is the cost that has overlap potential. The LogP model encourages overlap of network latency in algorithm design. However, when data distribution is considered, the aforementioned two-copy problem notwithstanding, memory communication cost increases with additional misses in the memory hierarchy attributable to

poor data locality. This additional latency has the potential for overlap depending on system design such as non-blocking memory accesses or aggressive prefetching and algorithm characteristics.

A large class of parallel scientific applications (e.g. climate models like the Ocean simulation [8]) stand to benefit from predictive models of distributed computation that incorporate system software characteristics and encourage multiple levels of achievable latency overlap. For example, array boundary exchange[5] in Ocean classically requires array data transmission of rows and columns between processing elements. The mapping of a 2-D domain to the linear virtual address space in a given memory implementation typically involves grouping elements contiguously[6] by column (Fortran) or row (C). Thus transmitting rows in Fortran or columns in C involves accessing non-contiguous data elements. We measured a six-fold increase in execution time over the contiguous case for a simple boundary exchange of a 1024 x 1024 array of doubles with a 32-byte stride on our 32-node Pentium III Beowulf. For a similar exchange in shared memory we measured a three-fold increase in overall communication time within SMPs on the SGI Origin 2000 architecture. Costs will be much more severe for I/O bound codes[15].

Existing parallel programs do not exhibit good performance on distributed systems[3]. Future algorithm designs on distributed systems must optimize memory and network communication costs through balanced inter- and intra-node communication to achieve efficiency. In the remainder of this paper, we present our approach to estimating the cost of memory communication by augmenting the LogP model of parallel computation to estimate cost in a hierarchical memory subsystem. The resulting memory ***log***P model provides a simple and useful

[5] The astute reader will note we are oversimplifying boundary exchange problems by ignoring concepts such as ghost cells and alternating communications in favor of a simplified discussion.
[6] We assume contiguous or physically adjacent to mean successive virtual addresses mapped to successive physical addresses.

cost estimate of memory communication performance in terms similar in meaning to the LogP model of parallel computation. While such an approach is currently disjoint from the LogP model of parallel computation, it motivates another bridging model of distributed computation that couples both models to incorporate the memory hierarchy in estimates of point-to-point communication.

2 The memory ℓogP model

We begin with a summary of the model leaving the detailed description of its derivation to the following section. Under the memory ℓogP model, processors communicate via explicit loads and stores that drive implicit replication across the hierarchy. Figure 3 provides an illustrative view of the succeeding discussion. To incorporate system and application characteristics in the model, the o parameter is the cost of ideally distributed data transfers. Since most systems exploit spatial locality, the system-dependent o cost is average, unavoidable overhead and represents the best case for data transfer on a target system. The pipeline cpi (cycles per instruction under perfect cache conditions) of the data transfer algorithm is the lower bound for o – where only changes to the micro-architecture will improve upon the overhead. Costs greater than the pipeline cpi are due to hardware and middleware implementation characteristics such as imperfect cache implementations and less than optimal data transfer implementation.

Additional delays in memory communication are functions of the system implementation and the application characteristics of data size (s) and data distribution (d). The effective latency after overlap (ℓ) is a function of these parameters. The lower-case ℓ ("ell") symbolizes the realized reduction in transfer cost due to latency overlap in contrast to the upper bound on network

latency (L) of the original LogP model. This parameter is bounded above by the cost of data transfers without overlap and bounded below by the $\boldsymbol{o}$ parameter of memory $\boldsymbol{\ell o g}$P.[7]

The implicit small message size w_n of the LogP model is replaced with w_m, the word size of the instruction set architecture for the machine under study. We formally characterize memory communication cost under four parameters:

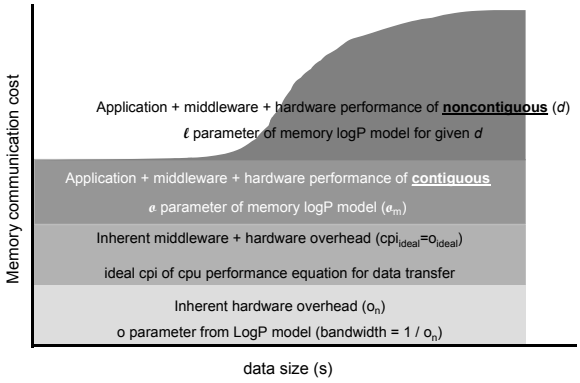

Fig. 3. Performance bounds with the memory $\boldsymbol{\ell o g}$P model of communication. The $\boldsymbol{o}$ and $\boldsymbol{\ell}$ parameters characterize additional delay due to middleware and application implementation.

$\boldsymbol{\ell}$: the effective latency, defined as the length of time the processor is engaged in the transmission or reception of a message due to the influence of data size (s) and distribution (d) for a given implementation of data transfer on a given system, $\boldsymbol{\ell}=f(s,d)$. Improvements in system application, middleware, and hardware that increase latency overlap may reduce this parameter.

$\boldsymbol{o}$: the overhead, defined as the length of time that a processor is engaged in the transmission or reception of an ideally distributed message for a given implementation of data transfer on a given system. During this time the processor cannot perform other operations. For scalable implementations, $\boldsymbol{o}$ remains constant with size. Improvements in system middleware and hardware that increase data throughput may reduce this parameter.

$\boldsymbol{g}$: the gap, defined as the minimum time interval between consecutive message receptions at a processor. The reciprocal of $\boldsymbol{g}$ corresponds to the available per-processor bandwidth for a given implementation of data transfer on a given system. We follow common simplification of $\boldsymbol{o}=\boldsymbol{g}$ in

[7] The definition of the $\boldsymbol{\ell}$ ("ell") parameter is purposely subtle. It symbolizes reduction via overlap from the upper bound on memory latency (L). A system with no overlap ability has $\boldsymbol{\ell}=L$; such systems are rare today. We use $\boldsymbol{script}$ to denote memory $\boldsymbol{\ell o g}$P parameters.

our model since o captures the entire cost of data movement (i.e. no assist overhead is present). We do not eliminate g since we believe future systems may not follow this simplifying assumption (e.g. multi-threaded architectures).

P: the number of processor/memory modules. We currently consider only point-to-point communication in the memory hierarchy, so *P=1*. We have results that show this is equivalent to point-to-point communication in small SMPs. Gather/scatter in this context means collective packing and unpacking operations on data. With *P>1* it refers to collective communication. We avoid this confusion with our assumption. We relax this assumption in Section 6.5.

Similar to LogP and its extensions [2, 10, 13, 17], all parameters are measured as multiples of the processor clock cycle. As in the LogGP model [2], it is most natural to express o and ℓ in terms of the overhead per byte instead of per word since word definitions may intuitively have different meanings to different audiences. For this reason we will express results in terms of cycles per byte.

In our discussion, we assume typical load/store architectures with hierarchical memory implementations that utilize state-of-the-art latency hiding techniques to achieve high rates of instruction-level parallelism. Our analyses target memory communication performance so we will ignore effects of branching and icache, assuming copying algorithms exhibit perfect branch prediction and perfect icache hit rates. For memory communications involving data transfers across the memory hierarchy, these assumptions are reasonable. Our predictions are at the application level, so non-deterministic characteristics of memory access delay at the micro-architecture level are not considered. We assume deterministic access delay and use minimum average values as inputs to our model. As is customary, we assume the receiving processor may

access a message only after the entire message has arrived. At any given time a processor can either be sending or receiving a single message.

3 Model Derivation

3.1 Single buffer transfers

The LogP model of parallel computation consists of four parameters. L is an upper bound on the communication latency. o is the overhead of communication. g is the bandwidth at the network interface. P is the number of processors. Communication is modeled as point-to-point messages between processors. Sending a small message between two processors requires $o + L + o$ cycles. o is the average overhead on the send and receive ends of communication and L is an upper bound on the latency between the two processors across the network. The o and g parameters of the LogP model are fixed for a given machine small message size, w_n.

Sending a longer message of B bytes requires $\lceil B/w_n \rceil$ messages. The associated cost is $o + max\{g,o\} * \lceil B/w_n \rceil + L + o$. LogP first characterized the CM-5 architecture, which had a significant message dispatch cost (g-o) at the network interface. Succeeding generations of architectures have reduced this dispatch cost and the terms of LogP are typically reduced to o=g.

Culler and Singh [8] suggest LogP can be applied to model any data transfer. Data movement across a memory hierarchy occurs as a series of data transfers between cache levels (or buffers). The instruction set architecture of the target system has a fixed "message size" for load and store instructions (4 bytes in a 32-bit ISA). We assume there is instruction overhead for movement of data between buffers (caches) in the memory hierarchy per message (w_m bytes of data) movement. The resulting cost of a single word data transfer between two buffers in a cache hierarchy is

$T_{word} = O + L$ *(1).*

Here *O* and *L* are the *o* and *L* parameters of the original LogP model. This formula is accurate for a single word transfer in a memory hierarchy. Such transfers will incur hardware operational overhead to cause the data movement (instruction overhead or *O*) and the full hardware latency (*L= 1/bandwidth*) between the two buffers.

However, for multiple word transfers in one direction (i.e. a single type of instruction load or store) between buffers the cost estimate becomes dependent on the context of execution. Using the LogP model as described by Equation 1 would lead to large overestimates of memory communication cost since it ignores the effective overlap inherent in block transfers and instruction execution. We can attempt to compensate for this overlap by reducing the terms of Equation 1. The cost for a single word in a multiple word transfer of fixed type is

$T_{word} = O+L-overlap$ *(2).*

While the *O* and *L* parameters of Equation 2 will be fixed values for the given instruction type, the overlap parameter will vary with every word, even for the same instruction type depending on the software context. The cost attributed to overhead may at times be a single cycle, but under another instruction sequence hardware pipelining of instructions may reduce the cost. For latency, the cost depends on the percentage of the block transfer that contains useful information. The amount of overlap varies with the system and application implementation. However, Equation 2 is based upon the LogP model of hardware parameters, so the *overlap* should be a measurable hardware constant. Strict application of Equation 2 (and hence LogP) will lead to inaccurate cost predictions when overlap varies as in today's complex memory hierarchies.

We modify Equation 2 to compensate for this limitation by defining overhead and latency in single buffer transfers as achieved performance on a given architecture and system. Let O_j and L_j be the average full hardware costs associated with data transfer overhead and latency for data transfers between buffer *j* and *j-1*. Also, let α be the percentage of instruction execution overlap achieved by a given implementation during the *W* word data transfer and β be the percentage of latency overlap achieved by a given implementation during the *W* word data transfer. Then the average cost per word for transmission of *W* words of fixed type between buffers *j* and *j-1* is

$$T_{word} = (1-\alpha_j)O_j + (1-\beta_j)L_j \qquad (3).$$

3.2 Multiple buffer transfers

For modeling data transfers in the memory hierarchy, we must extend these models to incorporate multiple buffers, and instruction types. Since we are currently interested in modeling data transfers, we only consider load and store instruction types of word granularity. As mentioned, we assume register-register or load-store architecture, however our model is generally applicable. As such, we verify and apply our model to register-register (e.g. MIPS) and register-memory (e.g. IA-32) architectures in Section 6.

Extending Equation 3 to multiple buffer transfers over a single instruction type involves a summation over the number of buffers (*M = number of levels in the memory hierarchy*).

$$T_{word} = \sum_{j=1}^{M} (1-\alpha_j)O_j + (1-\beta_j)L_j \qquad (4)$$

Here the subscript *j* denotes the associated costs and observed overlap percentage between levels *j* and *j-1* in the hierarchy. Equation 4 can be used to estimate communication cost of a single load/store in a stream of loads or stores. Interleaving of loads and stores complicates the cost estimate. α_j and β_j values for loads and stores will be different. Loads cause inbound

transfers across the hierarchy while stores cause outbound transfers across the hierarchy. Latency hiding effects differ significantly to take advantage of the inherent properties of inbound and outbound transfers. Additionally, with interleaving, loads may cause stores that were previously buffered in the hierarchy and stores may cause loads of the data to be modified. This skews the individual memory latency of a single load or store.

To address these problems, we simplify Equation 4 to account for average values for our parameters. Averages are more intuitive since programmers view message communication at larger than word granularity and more practical by reduction in the number of parameters.

We previously assumed word granularity of data transfer. In general purpose memory hierarchies, data transfer across hierarchy levels is through implicit replication, meaning overhead is observed at the start of the instruction only – not with each additional transfer. It follows that the overlap parameter α_j also applies only to the reduced term. Thus the cost as seen by the processor for a transfer across multiple hierarchy levels is intuitively a single cost of $\sum_{j=1}^{M}(1-\alpha_j)O_j = (1-\alpha)O$. Equation 4 becomes

$$T_{word} = (1-\alpha)O + \sum_{j=1}^{M}(1-\beta_j)L_j \qquad (5).$$

α is the percentage of instruction overlap achieved in a W word transfer, β_j is the percentage of overlap achieved in a W word transfer between levels j and j-1, L_j is the hardware transfer latency between levels j and j-1 in the hierarchy, and O is the average hardware overhead cost for a single transfer instruction.

Equation 5 expresses transfer time overall per word transfer as the sum of the effective overhead per word-granular instruction ((1-α)*O) and the effective latency. Effective latency is

the difference of the total latency across the hierarchy and the cost of data accesses overlapped with useful work.

Applying Equation 5 to a set of instruction types requires summation over the types. For S types of instructions the transfer cost per word becomes

$$T_{word} = \sum_{i=1}^{S}(1-\alpha_i)O_i + \sum_{j=1}^{M}(1-\beta_{ij})L_{ij} \quad (6).$$

While types may vary beyond loads and stores, we can generally separate instructions into inbound and outbound buffer movement categories. The overhead and latency costs can be separated and grouped.

$$T_{word} = (1-\alpha_{in})O_{in} + (1-\alpha_{out})O_{out} + \sum_{j=1}^{M}(1-\beta_{in,j})L_{in,j} + \sum_{j=1}^{M}(1-\beta_{out,j})L_{out,j} \quad (7).$$

Equation 7 expresses overhead cost as the sum of overlap and overhead products by type. If we use average overhead over all instruction types O_{in+out} and average instruction overlap over all instruction types, α_{in+out}, we can simplify Equation 7. We further observe that the hardware latency (L_j) between levels j and $j+1$ is independent of instruction type. This reduces Equation 7 to

$$T_{word} = (1-\alpha_{in+out})O_{in+out} + \sum_{j=1}^{M}(1-\beta_{in,j})L_j + \sum_{j=1}^{M}(1-\beta_{out,j})L_j \quad (8).$$

Interleaving of loads and stores will affect the overall transfer time through effective changes in α_{in+out} and will be fixed for a given code implementation that interleaves loads and stores (e.g. a given data copying algorithm). The effects of interleaving on data latency overlap ($\beta_{in,j}$ and $\beta_{out,j}$) are dynamic depending on transfer size and distribution.

While Equation 8 is useful for derivation and discussion, in practice such measurements are not separable by loads and stores. For practical reasons, we combine $\beta_{in,j}$ and $\beta_{out,j}$ into a single

parameter β_j characterizing the average latency overlap at level j in the hierarchy for all inbound and outbound data transfers. This parameter can be approximated as miss rates at level j in the hierarchy so redundant transfers (e.g. write-back related collisions) will be accounted assuming means exist to estimate or measure miss rates. Equations 6-8 provide our rational for redefining the parameters of Equation 5 in the following:

$$T_{word} = (1-\alpha_{in+out})O_{in+out} + \sum_{j=1}^{M}(1-\beta_j)L_j \qquad (9).$$

Equation 9 gives the cost for interleaved loads and stores in hierarchical memory as a function of software-related overlap parameters α and β and hardware-related cost parameters O and L. This equation provides the cost of a memory copy transfer for W words across a M-level hierarchy for typical load/store architectures with the ability to overlap useful work with data latency.

For data transmission across the memory hierarchy the overhead is dominated by data instructions of word granularity. Assuming the values in Equation 9 are expressed in cycles, this is an expression of the cpu performance equation [18] from Patterson and Hennessey. This formula describes the achieved speed of the processor as a combination of pipeline cpi without memory influence ($cpi_{ideal} = (1-\alpha_{in+out})O$) and the additional cpi due to memory access delay ($cpi_{mem} = \sum_{j=1}^{M}(1-\beta_j)L_j$).

$$cpi = cpi_{ideal} + cpi_{mem} \qquad (10)$$

If we apply Equation 10 to contiguous data transfers, the lowest cost bound is cpi_{ideal} as the amount of data transferred increases. Any additional memory cost (cpi_{mem}) is due to poor implementation either in hardware, middleware or (with regard to data size) application.

For noncontiguous data transfers cpi_{ideal} does not change assuming the same transfer algorithm is used from the contiguous case[8]. Any increase in the cost will be due to additional misses that are a function of the data size and distribution given by the application and their interaction with the hardware and middleware implementation.

$$cpi_{contig} = cpi_{ideal} + cpi_{mem} \quad (11)$$

$$cpi_{noncontig} = cpi_{ideal} + cpi_{mem} + cpi_{mem'} \quad (12)$$

$$cpi_{mem'} = \sum_{j=1}^{M} (1 - \beta_j')L_j) \quad (13)$$

$$T_{word} = cpi_{memcpy} = cpi_{contig} + \sum_{j=1}^{M} (1 - \beta_j')L_j \quad (14)$$

$cpi_{mem'}$ is the cost due to the additional non-overlapped latency specified as the product of the noncontiguous overlap percentage β_j' and the associated hardware latencies at each level j in the hierarchy.

There are practical reasons for this reduction. Typically, the cost of contiguous accesses will approach cpi_{ideal} since data distribution is optimal and hierarchies exploit data locality. cpi_{contig} is readily measurable as well.

On the other hand, $cpi_{noncontig}$ is a motivation of this work: to quantify and separate the additional cost of noncontiguous transfers in the context of memory communication. By separating the contiguous cost, the additional noncontiguous cost becomes a function of the data distribution (i.e. memory access rates). Under regular distributed access patterns, as observed in the example in Section 2, these costs are predictable.

[8] We ignore non-deterministic effects of out-of-order execution on the processor.

Lastly, by simplifying Equation 14, we have completed the discussion that links application and augmentation of the LogP model to memory hierarchies while maintaining simplicity. Equation 14 can be expressed in terms of the previously defined memory ***log***P model:

$$T_{word} = cpi_{contig} + \sum_{j=1}^{M} (1 - \beta_j')L_j = \boldsymbol{o} + \boldsymbol{\ell} \qquad (15)$$

The assignment of the $\boldsymbol{o}$ and $\boldsymbol{\ell}$ parameters to measurable or predictable quantities (cpi_{contig} and $cpi_{mem'}$ respectively) makes the model practical. These practical estimates agree with the theoretical description of the memory ***log***P parameters presented in Figure 3.

4 Usage of the model

As mentioned, LogP estimates point-to-point communication cost as $o + L + o$ for transfers of w_n bytes, where o is a lower bound on memory communication and L is an upper bound on network latency. Sending a longer message of B bytes requires $\lceil B / w_n \rceil$ messages. The associated cost is $o + max\{g,o\} * \lceil B / w_n \rceil + L + o$. Estimating the cost of point-to-point communication in the memory ***log***P model is similar while the parameters have different contextual meaning. Cost of a single word transfer of w_m bytes is $w_m * (\boldsymbol{o} + \boldsymbol{\ell})$. We approximate the overhead as the cost of data movement for contiguous data (the case with $\boldsymbol{\ell}=0$). Data placement causes additional cache accesses. Thus $\boldsymbol{\ell}$ is a function of data size and access pattern. (Equations 11-15 confirm this observation.)

LogP and memory ***log***P are disjoint; we assume all memory communication occurs prior to network communication with no overlap (e.g. the network buffer is infinite in size). Coupling the two models into a single model is currently being studied. Despite this, the models can be applied separately to estimate cost by normalizing with respect to the network packet size. The

projected cost per byte is $(o_m+\ell) + (L_n/w_n) + (o_m+\ell)$ using *n* to denote network parameters and *m* to denote memory parameters. This assumes no additional buffer copying other than data movement from user space to the network or shared memory buffer. Additional buffer copies would incur a cost of o_m per byte. We also assume $o_m+\ell$ is the average cost between packing on the source processor and unpacking on the target processor. The combination of the memory ***loag***P model with the LogP model of parallel computation effectively replaces the o_n of the LogP model with $w_n*(o_m+\ell)$.

5 Experimental details

We measured model parameters on several machines: a 32-node Beowulf at the University of South Carolina with 933 MHz Pentium III and 256K L2 cache with 10/100 Ethernet interconnect; the 128-node SGI Origin 2000 at NCSA with 195 MHz MIPS R10000 and 4MB L2 cache with Craylink interconnect; and the Titan IA-64 Linux Cluster at NCSA with 800 MHz Intel Itanium I 32KB L1, 96KB L2, and 4MB L3 caches with Myrinet 2000 interconnect technology for low latency transmissions.

We gathered all of our measurements by augmenting an existing tool that is part of the MPICH distribution. mpptest [11] provides platform independent, reproducible measurement of message passing experiments such as ping-pong and memory copy. It can be used to benchmark systems for determining MPICH platform dependent parameters. We modified mpptest to allow variations in stride access and array data structures for data movement operations such as direct copy, packing, unpacking, and broadcast. All experimental results were run a minimum of 20 times by the mpptest tool to ensure dependable results. Furthermore we ran all tests twice to ensure no system wide perturbations affected results. We assume no additional costs due to contention for resources in our study.

6 Experimental results

6.1 Model verification

Verification of our approach entails illustrating our ability to measure the ℓ and α parameters and using these values to explain and predict memory communication cost. Figure 4a provides source-node performance measurements for memory communication (direct copy of non-contiguous integer data) using a simple array copy optimized by the native compiler. These measurements depict throughput in cycles/byte for the direct copy of an integer array of varying sizes (x axis) and strides (separate curves) on a Beowulf machine. The curves for strides > 4 bytes represent the ℓ function. The influence of the memory hierarchy on ℓ is obvious and significant in the series of plateaus observed. Also the α parameter (the bottom most line with stride = 4 = word size) is constant with increasing stride indicating a scalable method of data transfer. (In contrast, Section 6.4 identifies scalability limitations of data transfer in MPI derived data types.) The α term is the cost associated with movement of contiguous data using the data transmission algorithm.

Estimation of the α parameter requires measurement of contiguous data transmission, a relatively simple task. We expect the α parameter will be constant as problem size and strides increase; that a scalable transmission method is chosen. Other work [23, 27] indicates that the overhead for packing and unpacking of MPI derived data types in implementations such as MPICH does not scale well.

Measuring the ℓ parameter directly requires running experiments varying message size and contiguity. After subtracting the ideal overhead, the ℓ function remains. Predicting the ℓ parameter in general is not an easy task. Given the regular nature of memory communication performance and the fact that many scientific computing codes exhibit strided data access

patterns, we can predict memory communication latency (ℓ) for regular communication schemes and optimize the performance of scientific computing applications with pre-measured ℓ and o values. Using pre-measured values to choose the best possible implementation, while practically important, is straightforward technically. For the sake of brevity, we only discuss general prediction mechanisms in the next section.

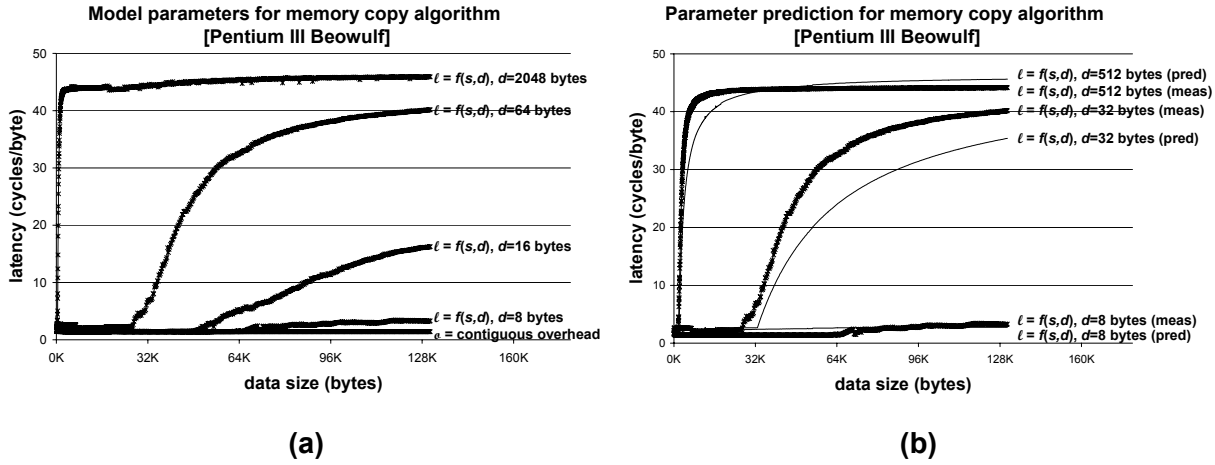

Fig. 4. (a) Parameterization of the memory logP model for the cost of memcpy for non-contiguous data illustrates the magnitude of delay relative to the contiguous overhead (o) and the LogP overhead (o_n where $o_n<o$). (b) Predicting the miss rates and memory $\ell o g$P parameters with a simple curve fitting constrained to the size characteristics of the memory hierarchy and the data characteristics of size (s) and distribution (d) can be fairly accurate when TLB misses are not present and accesses are regular.

6.2 Parameter prediction

The regular patterns identified in our initial analysis of direct copy encouraged us to explore cost prediction of the ℓ function. This technique is specific to memory communication and cannot be generally applied. Measurement of the o term is relatively simple and so prediction is not truly warranted since o can often be approximated with a single constant as suggested in our model. For relatively simple copying schemes algorithm dependent predictive cost estimates of the ℓ term can be developed. For less regular packing structures, it will be necessary to directly

measure the ℓ function for a target system. Whether using predictive measurements of ℓ or direct measurements, applied cost estimation is identical once o and ℓ are identified.

Our approach to prediction is based on two key observations. First, efficient data copying involves regular data access patterns. Second, latency hiding is accomplished primarily through blocked communication between hierarchy levels making additional delays dependent upon the data access pattern and data size. Techniques such as out-of-order execution or loads-under-miss will not significantly impact the performance beyond that observable in the measured value o, the contiguous overhead.

We attempt to use stack distance curves [6, 14] to approximate the resulting cache access rates at each level of the memory hierarchy. The average memory access time can be expressed as

$$\ell = \sum_{j=2}^{M}(1-P_j)T_j = T_1 + T_2\int_{S_1}^{\infty} p(x)dx + T_3\int_{S_2}^{\infty} p(x)dx + \ldots + T_M\int_{S_{M-1}}^{\infty} p(x)dx \qquad (16)$$

where P_j and T_j are the access probability and the average access time to the memory hierarchy at level j where $j = 2, \ldots, M$. P_j depends on the characteristics of the memory hierarchy and the data access pattern. Both are known for memory communication. We currently ignore the effects of TLB although inaccuracy at large data size and strides would indicate TLB effect can be significant and should be included. Figures 4b and 5b (discussed further in the next subsection) depict results for a manual curve fitting and illustrate ℓ can be approximated as a fitted function of the cache and block size, and the data size and stride distance.

6.3 Memory copy analysis

We can use memory ℓogP to confirm some generally understood characteristics of data copying in complex memory hierarchies. We performed experiments separating the pack and unpack (i.e. memory communication gather and scatter) costs of data transfer. Direct

experimental measurements confirm the unpacking portion of the data transfer dominates the cost of the native memory copy for non-contiguous data. Data access patterns confound the write-back buffering abilities of the memory subsystem implementation. This indicates that for these systems it is sufficient to model pack as the α parameter ($\ell=0$) except for very small transfers where prefetch overhead dominates average cost per byte. Modeling unpack or memory copy requires measurement or prediction of the additional latency due to mismatches between the contiguity of the application and the middleware and architectural implementation.

Figure 4a provides measured data for varied access patterns on the native memory copy implementations of the Linux-based Beowulf machine. Plateaus are present as the size of each level of cache is exhausted. This figure depicts saturation of the L2 cache as the stride and data sizes increase. By removing the contiguous contribution from the measured latency, we provide taxonomy of the contributions to stalls for memory copy and unpacking algorithms.

Figure 4b shows select results from our simple prediction curve fitting approach. Most approximations are quite close (within +80% and –60%). The predictions are more accurate for larger data sizes since small deviations from the absolute values for small data sizes lead to large differences in the relative error. Inaccuracy becomes worse as problem sizes grow very large since we currently ignore the effects of TLB.

Figure 5 illustrates the portability of our approach; we repeated the experiments for the direct copy algorithm on the MIPS R10000 architecture of the SGI Origin 2000. Our data and predictions are system dependent, but our findings are similar. The associated error rates for prediction are slightly better than the error rates for the IA-32 architecture. This could be for any number of reasons including the compiler and architecture. The key observation here is that our

analysis technique and the predictive method can be useful and accurate on distinctive memory subsystems despite the simplicity of our approach.

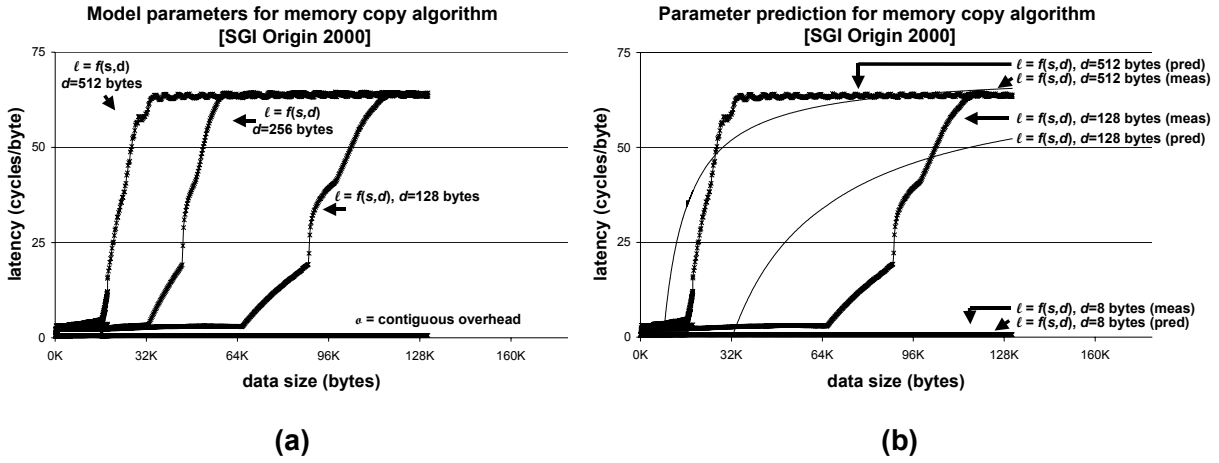

Fig. 5. Results for (a) parameterization and (b) simple curve fitting of the memory *ℓog*P parameters on the MIPS-based Origin 2000.

6.4 MPI overhead analysis

In practice, code developers often make performance decisions based on experience (i.e. previous trial and error). For example, the MPI standard provides derived data types for use by application programmers. This abstraction is designed to ease parallel programming by alleviating user's of the task of packing and unpacking spatially distributed data structures prior to data transmission. Seasoned MPI developers often assume such abstractions will negatively impact performance. In this section, we turn our attention to communication in distributed shared memory using MPI, though our techniques are applicable to any communication library. We use the memory ***ℓog***P model to systematically analyze the performance of MPI derived data types.

In scientific applications, it is common for arrays of data to be transferred as in the boundary exchange problem discussed in Section 1. Figure 6a shows memory ***ℓog***P parameter measurements for transfers of various n x n matrices of doubles (8 bytes) on the SGI Origin 2000

implemented using derived data types and standard MPI packing routines. Matrices of 128KB, 512KB, 2MB, 8 MB, 128MB, and 512MB are studied with dimensions of n=128, 256, 512, 1024, 2048, 4096, and 8192 respectively.

The $\boldsymbol{o}$ parameter of memory $\boldsymbol{\ell og}$P provides the lower bound on (best case of) memory communication cost and reflects the properties of the memory hierarchy, increasing as cache boundaries are crossed with regard to data size (s). If the other latencies for communication ($\boldsymbol{\ell}$ and L) are eliminated or overlapped, the $\boldsymbol{o}$ cost becomes the bottleneck. Notably, the $\boldsymbol{o}$ parameter is nearly as large as L for larger data sizes despite the fact that contiguous accesses are aggressively optimized (e.g. prefetching) in most memory hierarchies. The L parameter of LogP varies slightly as cost is affected by memory overhead; strict application of LogP implies using the maximum value (worst case) of L for prediction. For matrices measured of 512KB or less, the network latency (L) is the bottleneck. For matrices measured of 2MB or larger, the $\boldsymbol{\ell}$ parameter of memory $\boldsymbol{\ell og}$P or the latency due to poor spatial locality becomes the dominant bottleneck of communication for this algorithm using derived data types. Total memory communication cost ($\boldsymbol{o} + \boldsymbol{\ell}$) actually dominates overall communication cost at even smaller sizes.

In Figure 6b, we illustrate how simple changes to an MPI packing routine can enhance the performance of communication observable in the memory $\boldsymbol{\ell og}$P model. In this figure, dotted lines separate pairs of stacked bars. The first stacked bar in a given pair is the parameterized cost for a simple implementation of MPI packing. The second stacked bar in a given pair, labeled “opt”, is the parameterized cost for a version of MPI packing that uses array padding and cache blocking to improve spatial locality. The result of the optimization is to skew the point at which memory communication dominates network communication for this algorithm to larger array sizes. These types of optimizations reduce the latency ($\boldsymbol{\ell}$) due strictly to spatial locality of the data set for this

algorithm. This approach has limits in reducing the ℓ parameter of memory ℓogP in memory communication. Though currently ignored by the theoretical coupling of memory ℓogP and LogP as mentioned in Section 4, varying the intermediate buffer size policy for MPI communication will also affect the measured ℓ parameter. Smaller buffers will decrease throughput and increase ℓ. Larger buffers will have the opposite effect. This type of experimentation is the subject of future work.

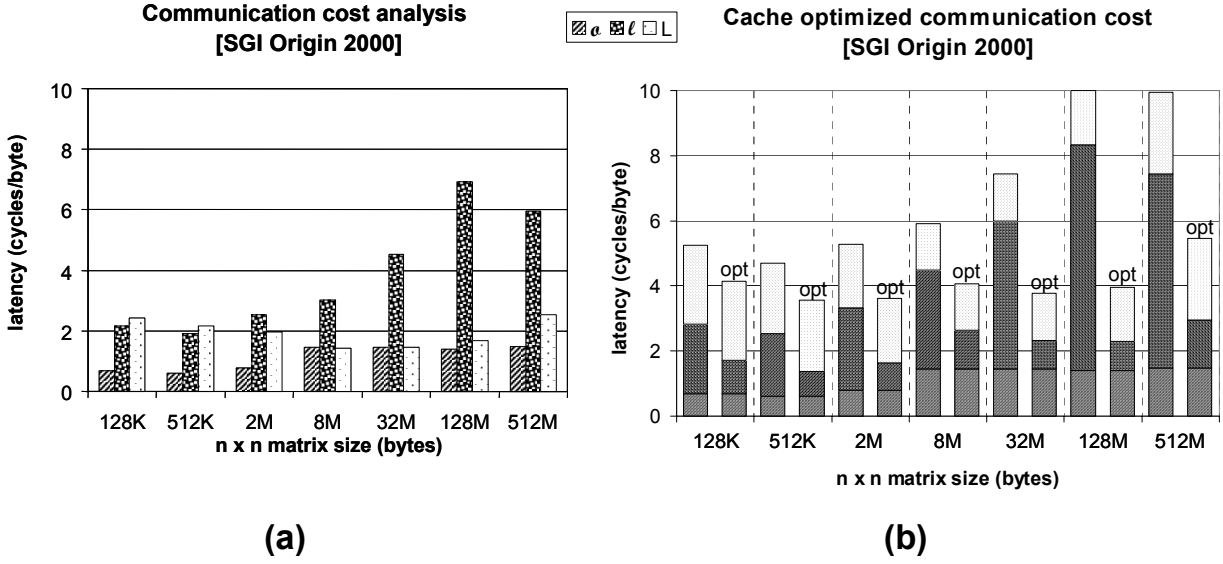

Fig. 6.The memory ℓogP model is useful for quantifying the costs of MPI communications in a distributed shared memory machine. These figures depict MPI_Send results for transferring matrices implemented as derived data types. (a) The costs of memory communication quickly dominate network communication. (b) Optimizing the overhead of derived data types can reduce the ℓ parameter, but overall communication is still significantly affected by overhead from memory communication.

6.5 Algorithm cost prediction

Under the simplifying assumptions of Section 5, in this section we illustrate use of memory ℓogP to theoretically and practically analyze a simple broadcast algorithm This example serves several purposes: 1) It highlights the differences inherent to algorithm design when application- and system-dependent memory throughput is considered. In this instance LogP underestimates

the resulting communication cost. 2) It illustrates the cost in complexity inherent to our model. 3) It quantifies the impact of memory communication for simple algorithm cost models. This discussion underscores the pros and cons of memory ***ℓogg***P and LogP: tradeoffs of complexity and accuracy in the design process similar to PRAM versus LogP must be considered.

Figure 7 shows the cost of a simple broadcast under both models in a system composed of four processing nodes. Under shared memory, *L=0* since there is no network latency. With distributed memory, *L* is fixed across both models as the maximum network transmission latency. Using the traditional assumption of *o=g*, a point-to-point communication under LogP is *(P-1)*max(o,g)+2o+L=(P-1)*o+2o+L=(P+1)*o+L*. This is the hardware overhead delay for each successive message followed by the final receive overhead that is not overlapped with other transmissions.

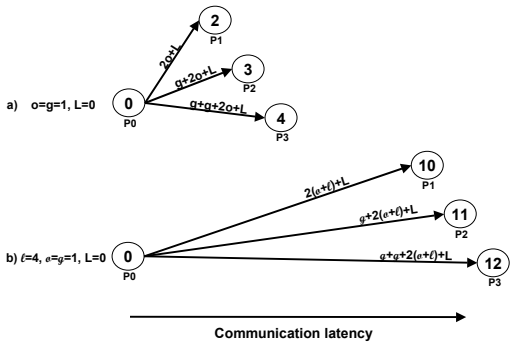

Fig. 7. Communication cost predictions under LogP (a) and ad hoc memory *ℓogg*P + LogP (b) for simple broadcast in a 4-way SMP.

Using the ad hoc memory ***ℓogg***P + LogP model as discussed in Section 4, the cost prediction is similar. For memory ***ℓogg***P we temporarily lift our *P=1* assumption. The resulting point-to-point communication cost is *(P-1)*max(**o**,**g**)+2(**o**+**ℓ**)+L=(P-1)***o**+2(**o**+**ℓ**)+L=(P+1)***o** + 2**ℓ** + L*. This assumes ***o***=***g*** as in the LogP analysis. If we assume the ***o*** cost of memory ***ℓogg***P is close to the LogP overhead parameter (we saw a counter-example of this in the previous section under MPI), the noticeable difference among the cost estimates is the *2**ℓ*** cost given by memory ***ℓogg***P. Current trends in memory and system performance imply the need to (at least for some codes and systems) consider the memory hierarchy. This *2**ℓ*** cost will be significant when *L=0*. When *L>0*,

the degree of network latency overlap and the speed of the network will determine the impact of the 2ℓ cost.

Figure 8 quantifies the costs of memory communication using the memory ***log***P model for an 8-way broadcast on two systems using an algorithm similar to Figure 7b on the SGI Origin 2000 and the Titan IA-64 cluster. Once again dotted lines separate groups of stacked bars. Each stacked bar in a given group corresponds to message data sizes increasing from left to right for 1K, 2K, 4K, and 8K bytes (the IA-64 set includes data points for 16K bytes). Each group corresponds to message distributions (or strides) of 8, 256, 2048, 8192 and 16384 bytes. Each bar is divided between total memory communication overhead, $(P+1)*o$ and latency, 2ℓ.

We use $(P+1)*o$ and 2ℓ instead of simply o and ℓ for continuity with our previous discussion and to illustrate the magnitude of difference between the hardware overhead parameter of LogP and the memory ***log***P parameters o and ℓ for memory communication in the broadcast. If we approximate the hardware cost (o_n) as the minimum software overhead for transmitting a contiguous message (i.e. the lowest observed value for the memory ***log***P overhead parameter), we can conservatively quantify the cost of memory communication. In Figure 8, the horizontal line of a given stacked bar dividing the overhead cost separates this estimated LogP overhead, $(P+1)*o_n$, in the bottom portion of $(P+1)*o$ from the measured memory ***log***P overhead, $(P+1)*o_m$ in the upper portion of $(P+1)*o$. From our assumptions (and in fact) $o_n<o_m$, and from our observations, $o_n<\ell$ by as much as a factor of *70x* for the SGI Origin 2000 and as much as *30x* for the IA-64 Cluster. o and ℓ values are obtained dividing $(P+1)*o$ and 2ℓ by $P+1=8$ and *2* respectively.

Other trends are evident as well in the data from Figure 8. Inner group trends indicate the influence of the overhead parameter on performance. For increasing sizes, transfer rates improve

for strides of 2048 bytes or less on the SGI Origin 2000 and strides of 2048 bytes or less on the IA-64 cluster. This quantifies the effect of aggregating the overhead by increasing the amount of data sent, resulting in lower cost per byte transferred. As locality worsens however, at larger stride sizes, these effects are overcome by additional penalties for memory copying the sparsely distributed data. As shown, the overheads remain fairly consistent as the 2ℓ parameter dominates the cost per byte of transfer for larger data sizes and strides on both systems. The exception to this trend is overhead for messages larger than 16K bytes on the IA-64 cluster that incur additional costs due to limitations of the cache hierarchy and middleware implementation for smaller message sizes. Such costs were the focus of analyses in Section 6.4.

The cost of network latency (L) can be very large for some systems and dominate even these high costs of memory communication. However, in the case of shared memory systems where $L=0$ and for systems with fast interconnect or the ability to overlap network communication, the impact of the costs ignored under LogP and quantified in Figure 8 will be significant for the reasons discussed in Section 1.

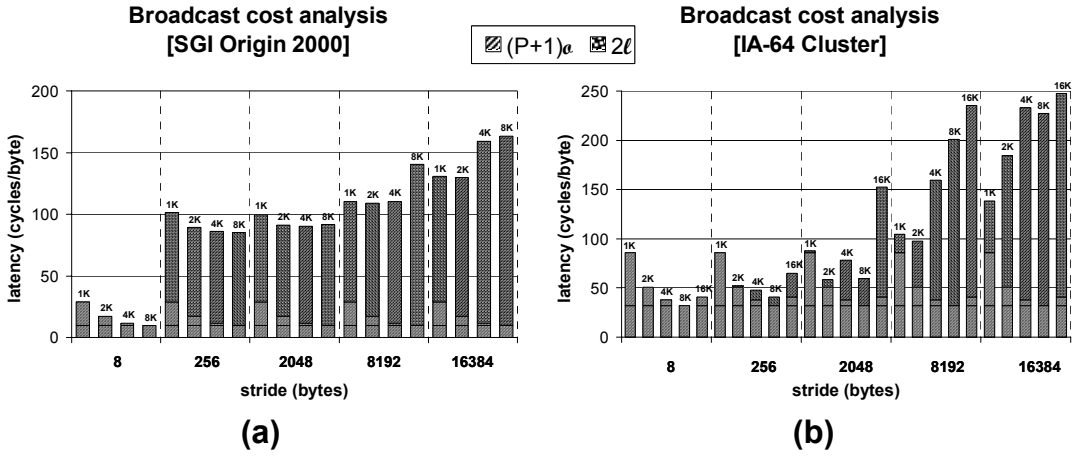

Fig. 8. The memory communication cost of an 8-way broadcast. The impact of overhead (P+1)o and latency 2ℓ becomes more significant as spatial locality decreases. For shared memory systems, these costs dominate total communication cost. For distributed memory systems with fast or overlapping network communications, these costs are significant.

7 Related work

Analytical techniques to predict cache performance can be used to estimate model parameters as desired. But, accurate models of memory hierarchy performance are necessarily complicated [1, 9]. Other models that attempt to capture the effects of spatial locality have been proposed. Of these, Sivasubramaniam et al. [20] and Stricker and Gross [21] address memory communication. The former consider spatial locality in a CC-NUMA architecture applying the traditional LogP model to estimate communication cost for optimizing simulation performance. Our approach redefines the LogP parameters to capture the effects of software and middleware on memory communication.

Stricker and Gross advocate hardware/compiler support to eliminate buffer copying in parallel communication based on a copy-transfer model. This model considers the achieved bandwidth in a single-level cache hierarchy for various message distributions and their effect on memory communication cost. The parameters of the copy-transfer model represent fine-grain bandwidth measurements along each critical data path for native system transmissions. This model could be used to complement the memory ***ℓog***P model by approximating parameters at finer granularity than that presented herein – at the expense of further complexity.

Our approach also uses throughput-based parameterization, but at a higher level of abstraction and under a different characterization scheme. Memory ***ℓog***P separates ideal and data-dependent latency for performance evaluation and prediction of memory communication costs. Our abstraction allows more complex hierarchies to be considered simply and more ready coupling with traditional algorithm models such as LogP. Our motivation is to capture the effects of the interaction between algorithm, system software, and hardware implementation. These other

papers make important contributions and provide additional support to our argument that memory communication is indeed significant and relevant in emerging architectures.

Many computational models of parallel performance provide simple quantification of communication performance [16]. The Hierarchical Memory Model (HMM) [4, 12] applies the characteristics of memory hierarchies to network communication. Cost estimates are accurate for very large sets of streaming data [22, 25], but ignore the network attributes common to parallel and distributed systems. Our work provides impetus for combining hierarchical memory performance with estimates of network communication cost, distinguishing the two approaches to models of point-to-point communication.

Other models attempt to bridge different system characteristics to fully model the communication path. Bridging models such as LogP and the bulk synchronous parallel (BSP) model [24] incorporate multiple aspects of parallel systems. Both models quantify communication latency and bandwidth. LogP is widely used since it additionally incorporates asynchronous behavior and communication overhead – thus it has been the subject of previous extensions and is the subject of our modeling efforts. All of these models ignore the effects of the possibly significant impact of middleware implementation and application data distribution on memory communication in distributed systems.

8 Conclusions

Application of previous parallel communication models to distributed systems may result in less than optimal algorithm design. To address this problem, we derived the memory ***log***P model. The parameters of the memory ***log***P model are simply measured and analyzed and provide two notable contributions: 1) Incorporate system and application characteristics in memory communication cost. 2) Separate memory communication into inherent, unavoidable system

overhead and additional latency that has overlap potential encouraging efficient algorithm design. The result is a more accurate estimate of overall communication performance.

We used our techniques on an IA-32 Beowulf, a MIPS-based SGI Origin 2000, and an IA-64 cluster to: 1) Quantify and predict the cost of memory copy operations critical to communication transfers with accuracies varying from +80% to –60%. 2) Quantify and isolate the middleware-induced bottlenecks in an MPI implementation of derived data types. 3) Quantify and predict the memory communication costs of a simple broadcast algorithm – illustrating costs of 30x and 70x are ignored by hardware-based model parameters.

Our techniques show promise in performance evaluation and prediction, yet there are some limitations. Our analyses were limited to regular access patterns; prediction is more cumbersome for irregular patterns. Coupling memory ℓogP and LogP is presently an ad hoc model of computation. The additional costs of further buffering due to large messages and policy decisions can be incorporated in the ℓ parameter, but the impact on model complexity is unclear at present and needs further study. We intend to extend memory ℓogP to large messages much as LogP was extended to LogGP once the original model was developed. We also believe memory ℓogP will be useful for prediction in heterogeneous systems and intend to pursue research in this area.

[1] D. H. Albonesi and I. Koren, "A mean value analysis multiprocessor model incorporating superscalar processors and latency tolerating techniques," *International Journal of Parallel Programming*, 24 (3), pp. 235-63, 1996.

[2] A. Alexandrov, M. F. Ionescu, K. Schauser, and C. Scheiman, "LogGP: Incorporating Long Messages into the LogP model," in proceedings of Seventh Annual Symposium on Parallel Algorithms and Architecture, pp. 95-105, Santa Barbara, CA, 1995.

[3] G. Allen, T. Dramlitsch, I. Foster, T. Goodale, N. Karonis, M. Ripeanu, E. Seidel, and B. Toonen, "Supporting Efficient Execution in Heterogeneous Distributed Computing Environments with Cactus and Globus," in proceedings of SC 2001, Denver, CO, 2001.

[4] B. Alpern, L. Carter, E. Feig, and T. Selker, "The Uniform Memory Hierarchy Model of Computation," *Algorithmica*, 12 (2/3), pp. 72-109, 1994.

[5] C.-C. Chang, G. Czajkowski, C. Hawblitzel, and T. v. Eicken, "Low latency communication on the IBM RISC System/6000 SP," in proceedings of ACM/IEEE Supercomputing, Pittsburgh, PA, 1996.

[6] E. G. Coffman and P. J. Denning, *Operating Systems Theory*. Englewood Cliffs, N.J.: Prentice-Hall, 1973.

[7] D. E. Culler, R. Karp, D. A. Patterson, A. Sahay, E. Santos, K. Schauser, R. Subramonian, and T. von Eicken, "LogP: A Practical Model of Parallel Computation," *Communications of the ACM*, 39 (11), pp. 78-85, 1996.

[8] D. E. Culler, J. P. Singh, and A. Gupta, *Parallel Computer Architecture: a hardware/software approach*. San Fancisco, CA: Morgan Kaufmann Publishers, 1999.

[9] X. Du and X. Zhang, "Memory hierarchy considerations for cost-effective cluster computing," *IEEE Transactions on Computers*, 49 (9), pp. 915-33, 2000.

[10] M. I. Frank, A. Agarwal, and M. K. Vernon, "LoPC: Modeling Contention in Parallel Algorithms," in proceedings of Sixth Symposium on Principles and Practice of Parallel Programming, pp. 276-87, Las Vegas, NV, 1997.

[11] W. Gropp and E. Lusk, "Reproducible Measurements of MPI Performance," in proceedings of PVM/MPI '99 User's Group Meeting, pp. 11-8, 1999.

[12] T. Hey and J. Ferrante, *Portability and Performance for Parallel Processing*. New York: John Wiley and Sons Ltd., 1993.

[13] F. Ino, N. Fujimoto, and K. Hagihara, "LogGPS: A Parallel Computational Model for Synchronization Analysis," in proceedings of PPoPP '01, pp. 133-42, Snowbird, Utah, 2001.

[14] B. L. Jacob, P. M. Chen, S. R. Silverman, and T. N. Mudge, "An analytical model for designing memory hierarchies," *IEEE Transactions on Computers*, 45 (10), pp. 1180-94, 1996.

[15] M. Kandemir, J. Ramanujam, and A. Choudhary, "Improving the performance of out-of-core computations," in proceedings of 1997 International Conference on Parallel Processing, pp. 128-36, Bloomingdale, IL, 1997.

[16] B. M. Maggs, L. R. Matheson, and R. E. Tarjan, "Models of Parallel Computation: A Survey and Synthesis," in proceedings of 28th Hawaii International Conference on System Sciences (HICSS), pp. 61-70, Honolulu, HI, 1995.

[17] C. A. Moritz and M. I. Frank, "LoGPC: Modeling Network Contention in Message-Passing Programs," in proceedings of SIGMETRICS '98, pp. 254-63, Madison, WI, 1998.

[18] D. A. Patterson and J. L. Hennessy, *Computer Architecture: A quantitative approach*, 3rd ed. San Fancisco, CA: Morgan Kaufmann Publishers, 2003.

[19] F. Petrini, W.-C. Feng, A. Hoisie, S. Coll, and E. Frachtenberg, "The Quadrics Network (QsNet): High-Performance Clustering Technology," *IEEE Micro*, 22 (1), pp. 46-57, 2002.

[20] A. Sivasubramaniam, A. Singla, U. Ramachandran, and H. Venkateswaran, "Abstracting Network Characteristics and Locality Properties of Parallel Systems," in proceedings of HPCA 1995, pp. 54-63, Raleigh, NC, 1995.

[21] T. Stricker and T. Gross, "Optimizing Memory System Performance for Communication in Parallel Computers," in proceedings of ISCA 95, pp. 308-19, Santa Margherita Ligure, Italy, 1995.

[22] R. Thakur, W. Gropp, and E. Lusk, "Optimizing non-contiguous accesses in MPI-IO," *Parallel Computing*, 28 (2), pp. 83-105, 2002.

[23] J. L. Traff, R. Hempel, H. Ritzdorf, and F. Zimmermann, "Flattening on the Fly: Efficient handling of MPI derived datatypes," in proceedings of PVM/MPI '99, pp. 109-16, 1999.

[24] L. G. Valiant, "A Bridging Model for Parallel Computation," *Communications of the ACM*, 33 (8), pp. 103-11, 1990.

[25] J. S. Vitter and E. A. M. Shriver, "Algorithms for Parallel Memory II: Hierarchical Multilevel Memories," *Algorithmica*, 12 (2/3), pp. 148-69, 1994.

[26] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Shauser, "Active Messages: A Mechanism for Integrated Communication and Computation," in proceedings of 19th Annual International Symposium on Computer Architecture, pp. 256-66, Gold Coast, Australia, 1992.

[27] J. Worringen, A. Gaer, and F. Reker, "Exploiting transparent remote memory access for non-contiguous and one-sided communication," in proceedings of Workshop for communication architectures in clusters (CAC 02) at IPDPS '02, pp. 163b, Fort Lauderdale, FL, 2002.

[28] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *Computer Architecture News*, 23 (1), pp. 20-4, 1995.