Quantifying Locality Effect in Data Access Delay: Memory logP

Kirk W. Cameron
Dept. of Computer Science and Engineering
University of South Carolina
Columbia, SC 29208
kcameron@cse.sc.edu

Xian-He Sun
Department of Computer Science
Illinois Institute of Technology
Chicago, IL 60616
sun@cs.iit.edu

Abstract

The 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 in the model 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 to predict and analyze the latency of memory copy, pack and unpack. We use the model to isolate contributions of hardware, middleware, and software to data transfers on Intel- and MIPS-based platforms.

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.

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**[4]** approximate memory communication in parallel systems with a fixed overhead parameter (o), the reciprocal of the bandwidth between application and network buffers[1]. 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.

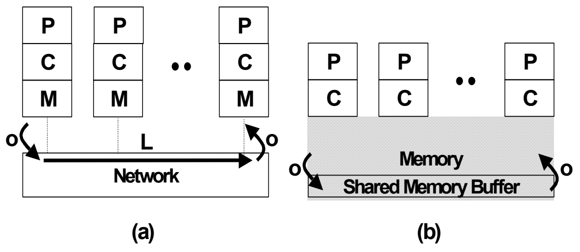

Fig. 1. P=Proc, C=Cache, M=Memory. Hardware parameterizations of communication costs using LogP are effective in parallel systems (a) when network communication dominates overall cost. As the impact of memory communication on overall communication increases, exemplified in shared memory (b), the effects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

Communication cost in shared memory architectures (Figure 1b) is dominated by memory communication. Years of research addressing the cpu-memory gap 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 (illustrated in Figure 1b) is not sufficient since the effective latency overlap of a hardware implementation is application and system dependent. 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.

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

Memory communication cost will increase in proportion to overall communication cost given current technological trends[6]: Future generations of distributed systems will consist of commodity, shared memory components with fast interconnects. 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.

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.

Future algorithm designs on distributed systems must optimize memory and network communication costs through balanced inter- and intra-node communication to achieve efficiency. Existing parallel programs do not exhibit good performance on distributed systems[1]. A large class of parallel scientific applications (e.g. climate models) stands to benefit from development of predictive models of distributed computation that incorporate system software characteristics and encourage multiple levels of achievable latency overlap.

In the remainder of this paper, we present our approach to estimating the cost of memory communication performance by augmenting the LogP model of parallel computation to estimate cost in a hierarchical memory subsystem. The resulting memory logP model provides a simple and useful 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. Motivating example

Array boundary exchange[2] common in scientific applications 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[3] by column (Fortran) or row (C). Thus transmitting rows in Fortran or columns in C involves accessing non-contiguous data elements. Figure 2 illustrates the varied cost of transmission. The critical data path is dependent upon source and destination, data size and distribution, and system implementation. We perform simple experiments to quantify this cost.

On a distributed memory machine as depicted in Figure 1a, the impact of non-contiguous accesses will severely impact the memory communication overhead. To quantify the cost of such strided accesses on memory communication, we performed a simple experiment on a Pentium III based Beowulf described in detail later. The

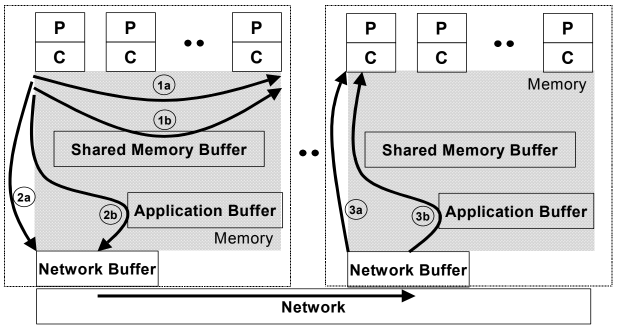

Fig. 2. Memory communications within shared memory (1a-b) and to/from the network buffer in distributed communication (2a-b/3a-b) follow critical paths dependent upon data size, data distribution, and system implementation.

[2] 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.
[3] We assume contiguous or physically adjacent to mean successive virtual addresses mapped to successive physical addresses.

code in Figure 3 illustrates a simple unpacking algorithm for copying a contiguous array of data in sbuffer[] into a non-contiguous array of data in rbuffer[] at stride s. We ignore loop overhead in our discussion and assume a 2-D array of 1024 x 1024 elements of type double (8 bytes on Pentium III) stored in row-major order. Measuring unpacking time for a single row at a stride of 1 element gives .0083 seconds - the ideal throughput achieved by this algorithm on this machine using an optimizing compiler since spatial locality is optimal. Unpacking time for a stride of 4 (32 bytes for fp elements on Intel Pentium III) is .062 seconds – a six-fold increase in the execution time due solely to the decreases in the spatial locality that the compiler and architectural implementation were unable to compensate for.

```
for (k=0,j=0;
     k<(s*numarrayelements);
     k=k+s,j++)
       rbuffer[k] = sbuffer[j];
```

Fig. 3. Simple unpacking algorithm.

On shared memory machines, the impact of non-contiguous accesses will severely impact the overall communication overhead (since network communication is not present). We perform a simple measurement of the time to send contiguous and non-contiguous data within and across nodes on an SGI Origin 2000 machine using a blocking send and measuring delay on the sender side. This blocking send of a 1KB message of contiguous data without packing (stride of 8 bytes for fp on MIPS) measured .176us while the same amount of data requiring packing with a 16 byte stride took .551us. This three-fold increase in overall communication within a single SMP was due to the cost of memory communication of the non-contiguous data. The delay becomes four times the memory communication cost when data needs to be moved to the network buffer for transport across SMPs.

For large-scale scientific applications with scalable workloads designed to fully utilize a processor's allocated memory, the additional memory communication time may outweigh the network communication latency. This is especially true as disk accesses (another level in the memory hierarchy) are required. In this paper, we will focus only on problem sets that fit in DRAM, leaving incorporation of the disk access portion of the memory hierarchy to future work.

3. Related work

Our model attempts to predict algorithm performance based on measurable system parameters. 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 [5].

Many computational models of parallel performance provide simple quantification of communication performance [8]. The Hierarchical Memory Model (HMM) applies the characteristics of memory hierarchies to network communication. Cost estimates are accurate for very large sets of streaming data, but ignore the network attributes common to parallel and distributed systems. Our work focuses on 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 [10] 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 [9] and is the subject of our modeling efforts. These models ignore the effects of the possibly significant impact of middleware implementation and application data distribution on memory communication in distributed systems.

The memory logP model presented in this paper addresses the apparent convergence of distributed architectures to clusters of shared memory machines. Application of previous parallel communication models to such distributed systems may result in less than optimal algorithm design. While maintaining simplicity, our model attempts to incorporate memory hierarchy performance in models of communication. It provides two notable contributions: 1) Incorporates system and application characteristics in memory communication cost. 2) Separates memory communication into inherent, unavoidable system overhead and additional latency that has overlap potential encouraging efficient algorithm design.

4. The memory logP model

Full derivation of the model can be found in the associated technical report **[2]**. Under the memory logP model, processors communicate via explicit loads and stores that drive implicit replication across the hierarchy. Although the cost of memory communication could be estimated as the average hardware transfer rate across the hierarchy, such cost estimates would ignore the system and application dependent latency overlap. Figure 4 provides an illustrative view of the succeeding discussion.

To incorporate system 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 o parameter of memory logP.

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:

l: 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, $l=f(s,d)$. Improvements in system application, middleware, and hardware that increase latency overlap may reduce this parameter.

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, o remains constant with size. Improvements in system middleware and hardware that increase data throughput may reduce this parameter.

g: the gap, defined as the minimum time interval between consecutive message receptions at a processor. The reciprocal of g corresponds to the available per-processor bandwidth for a given implementation of data transfer on a given system. We follow common simplification of $o=g$ in our model since o captures the entire cost of data movement (e.g. 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.

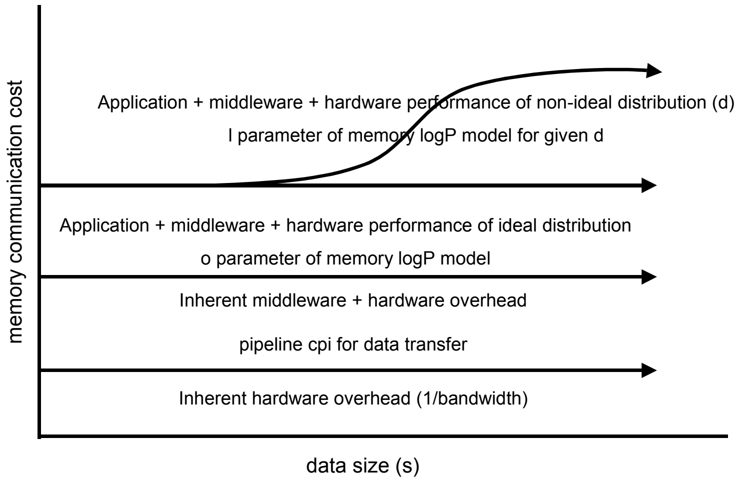
memory communication cost
Application + middleware + hardware performance of non-ideal distribution (d)
l parameter of memory logP model for given d
Application + middleware + hardware performance of ideal distribution
o parameter of memory logP model
Inherent middleware + hardware overhead
pipeline cpi for data transfer
Inherent hardware overhead (1/bandwidth)
data size (s)

Fig. 4. A graphical view of performance bounds using the memory logP model of communication. The o and l parameters characterize additional delay due to middleware and application implementation.

Similar to LogP and its extensions, all parameters are measured as multiples of the processor clock cycle. As in the LogGP model, 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 all our experiments, word size is 4 bytes for integer and 8 bytes for floating point, so multiplying by 4 or 8 respectively will provide measurements in terms of cycles per word or double word.

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.

5. 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 memory 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 logP model is similar while the parameters have different contextual meaning. Cost of a single word transfer of w_m bytes is $w_m*(o + l)$. We approximate the overhead as the cost of data movement for contiguous data (the case with l=0). Data placement causes additional cache misses. Thus l is a function of data size and access pattern.

LogP and memory logP 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 $(o_m+l) + (L_n/w_n) + (o_m+l)$ 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+l is the average cost between packing on the source processor and unpacking on the target processor. The combination of the memory logP model with the LogP model of parallel computation effectively replaces the o_n of the LogP model with $w_n*(o_m+l)$.

6. Experimental details

We utilized a 32-node Beowulf at the University of South Carolina. Each node consists of a 933 MHz Pentium III Processor with 1GB available main memory running Redhat Linux version 2.4.7-10. Separate non-blocking 16KB data and instruction L1 caches are available on chip. Each processor is additionally equipped with a 256KB Advanced Transfer L2 Cache on-die running at core speed. Cache and page block sizes are 32 bytes and 4096 bytes respectively. Load misses at L1 and L2 were measured as 7 and 70 cycles respectively. The interconnection network consists of two separate standard 10/100 Mbit/s switched Ethernet channels connected by a router.

We also modeled parameters on the SGI Origin 2000 at NCSA that utilizes a cc-NUMA architecture running the IRIX version 6.5.14 operating system. Each node contains two MIPS R10000 processors; each running at 195MHz, and 32kB two-way set associative, two-way interleaved primary (L1) cache. An off-chip 4MB secondary unified cache is present as well. Cache and page block sizes are 32 and 4096 bytes respectively. Load misses at L1 and L2 were measured as 12 and 90 cycles respectively. The achievable remote memory bandwidth on Craylink interconnect is 624MB/sec in each direction, which adds a 165ns off-node penalty and 110ns per hop.

We gathered all of our measurements by augmenting an existing tool that is part of the MPICH distribution. mpptest [7] 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 and unpacking. All experimental results were run a minimum of 20 times by the tool to ensure dependable results. Furthermore we ran all tests twice to ensure no system wide perturbations affected results.

Our general experimental method was to direct copy one array into another allocating only the necessary space. We varied which arrays were contiguous and noncontiguous to simulate direct copy, packing, and unpacking in memory systems while varying stride and data size. We also varied the data type (integer and double) and the system (Beowulf IA32 , and SGI Origin 2000). We measured strides between word size and 128K for data sizes varying from word size to 128K. For ease of discussion, results are presented for strides up to 2048 bytes. Measurements beyond this stride are similar and discussed in context.

7. Model verification

Verification of our approach entails illustrating our ability to measure the l and o parameters and utilizing these values to explain memory communication cost. Figure 5 provides measurements for memory communication for direct copy of non-contiguous integer data using a simple array copy optimized by the native compiler. These measurements depict 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 o parameter (the bottom most line with stride = 4 = word size) is constant with increasing stride indicating a scalable method of data transfer. The o term is the cost associated with movement of contiguous data utilizing the data transmission algorithm.

Estimation of the o parameter requires measurement of contiguous data transmission, a relatively simple task. We expect the o parameter will be constant as problem size and strides increase; that a scalable transmission method is chosen. Recent work **[11]** 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 most scientific computing only exhibits few non-contiguous data access patterns, we can predict memory communication latency (*l*) for regular communication schemes and optimize the performance of scientific computing 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.

8. 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

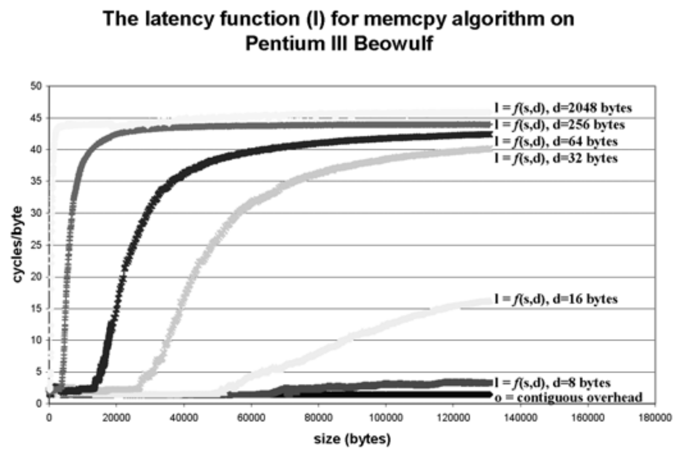

Fig. 5. The taxonomy of the memory logP model for the cost of memcpy of non-contiguous data illustrates the magnitude of delay relative to the contiguous overhead (o) and the LogP overhead (o_n where o_n<o).

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 **[3]** to approximate the resulting cache hit rates at each level of the memory hierarchy. The average memory access time can be expressed as

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

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, …, M. P_j depends on the characteristics of the memory hierarchy and the data access pattern. Both are known for memory communication. We can create an estimate of the cost for l based on approximation of these stack distance curves, or the hit rates at each level in the cache. At present we perform a manual curve fitting to illustrate the l function can be approximated as a fitted function of the cache and block size, and the data size and stride distance. 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.

Our simple estimation trades accuracy for simplicity. In practice however, we believe algorithm designers are interested in bounding costs such as L in the LogP model. The original o_n parameter (o of the LogP model) is a lower bound on hardware overhead. The predictive model of (o_m + l) is intended to provide an upper bound on overhead incorporating software and middleware. Despite many assumptions regarding the cache hierarchy (such as full associativity, no computational overlap, no TLB effect) our predictive model can be accurate within +80% and –60%. Figure 6 provides selected predictions for integer memcpy on the Pentium III Beowulf.

9. Other measurements

We can use the memory logP taxonomy to confirm some generally understood characteristics of memory hierarchies. Direct experimental measurement confirms the unpacking portion of the data transfer dominates the cost of the native memory copy for non-contiguous data. This indicates that for these systems it is sufficient to model pack as the o parameter (l=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.

We performed experiments separating the pack and unpack (i.e. memory communication gather and scatter) costs of data transfer. For packing, writes are to contiguous addresses resulting in deviations from contiguous overhead cost only when TLB misses dominate for very large data and stride sizes. Most additional latency (l) is due to the cost of writing data to a non-contiguous address space. These data access patterns confound the write-back buffering abilities of the memory subsystem implementation.

Figure 5 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 6 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.

To illustrate 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 IA32 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.

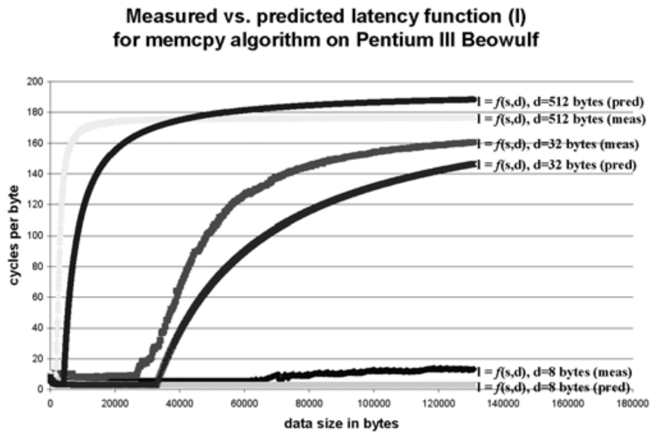

Fig.6. Predicting the miss rates with a simple curve fitting constrained to the size characteristics of the memory hierarchy and the data characteristics of size (s) and distribution (d) can be fairly accurate when TLB misses are not present and accesses are regular.

10. Conclusions

We have illustrated the need for inclusion of memory communication performance in models of point-to-point communication. To that end we derived the memory logP model of memory communication through application and augmentation of the LogP model of parallel computation to the memory hierarchy. The parameters of the memory logP model can be simply measured and analyzed. The model is generally applicable to any memory communication, but prediction is likely only for very regular memory access patterns.

We used the model to characterize, bound, and predict memory performance. The result of these techniques is a more accurate estimate of overall communication performance. We practically applied our techniques to two architecturally distinct systems, an IA32 Beowulf and the MIPS-based SGI Origin 2000. The resulting measurements for the o parameter quantified the scalability of the copying algorithm. Additionally, simple stack distance curve prediction was shown to be practically accurate (within +80% and –60%).

Use of stack distance curves at present ignores TLB misses, a source of inaccuracy when page thrashing occurs. Another limitation of the present model concerns coupling with network delay since buffering makes concatenation with LogP less than straightforward. We will address this limitation in future work.

[1] 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.

[2] K. W. Cameron, "Memory logP and its implications," *USC CSCE Technical Report # 2002-001, available at http://www.cse.sc.edu/~kcameron/prof.html*, 2002.

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

[4] 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*, vol. 39, pp. 78-85, 1996.

[5] X. Du and X. Zhang, "Memory hierarchy considerations for cost-effective cluster computing," *IEEE Transactions on Computers*, vol. 49, pp. 915-933, 2000.

[6] I. Foster, "The Grid: A New Infrastructure for 21st Century Science," *Physics Today*, vol. 55, pp. 42-49, 2002.

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

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

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

[10] L. G. Valiant, "A Bridging Model for Parallel Computation," *Communications of the ACM*, vol. 33, pp. 103-111, 1990.

[11] 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, Fort Lauderdale, FL, 2002.