Parallel Processing Letters
© World Scientific Publishing Company

ISOLATING COSTS IN SHARED MEMORY COMMUNICATION BUFFERING

SURENDRA BYNA
Department of Compuer Science, Illinois Institute of Technology
Chicago, IL 60616, USA

KIRK W. CAMERON
Department of Computer Science and Engineering, University of South Carolina
Columbia, SC 29208, USA

and

XIAN-HE SUN
Department of Compuer Science, Illinois Institute of Technology
Chicago, IL 60616, USA

ABSTRACT
Communication in parallel applications is a combination of data transfers internally at a source or destination and across the network. Previous research focused on quantifying network transfer costs has indirectly resulted in reduced overall communication cost. Optimized data transfer from source memory to the network interface has received less attention. In shared memory systems, such memory-to-memory transfers dominate communication cost. In distributed memory systems, memory-to-network interface transfers grow in significance as processor and network speeds increase at faster rates than memory latency speeds. Our objective is to minimize the cost of internal data transfers. The following examples illustrating the impact of memory transfers on communication, we present a methodology for classifying the effects of data size and data distribution on hardware, middleware, and application software performance. This cost is quantified using hardware counter event measurements on the SGI Origin 2000. For the SGI O2K, we empirically identify the cost caused by just copying data from one buffer to another and the middleware overhead. We use MPICH in our experiments, but our techniques are generally applicable to any communication implementation.

1. Introduction

Computers continue to increase in complexity. Hierarchical memories, superscalar pipelining, and out-of-order execution have improved system performance at the expense of simplicity. Redesigned compilers allow applications to take advantage of new architectures and execute more efficiently.

Unfortunately, compilers are limited in their ability to increase performance. For instance, the compiler is often unaware of subtle characteristics of cache hierarchies. Optimal performance is typically achieved through a combination of optimized compilation and algorithm redesign through system dependent analysis.

Communication in parallel systems increases complexity tremendously. As a result, parallel compilation is even less fruitful than its sequential ancestor. Optimizing performance in such environments relies more heavily on the use of tools to provide performance data for analysis.

Distributed systems rely on middleware to address the interoperability of heterogeneous software and hardware implying additional complexity. The interaction between hardware, software and middleware is not well understood. Thus, the optimizing abilities of distributed compilers are very limited. Optimizing performance in such environments will greatly rely on tools for system dependent analysis.

Communication costs in such environments are a function of the critical data path (see Fig. 1). 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.

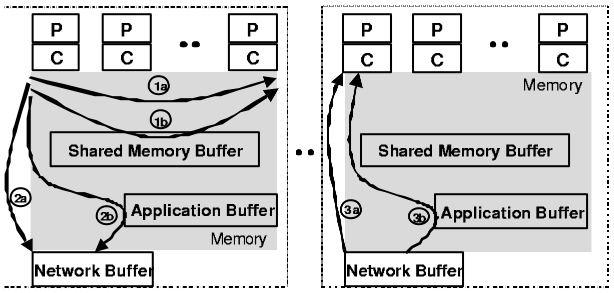

Fig. 1. Memory communications within shared memory (1a-b) and to/from the network buffer in distributed communication (2a-b/3a-b) follow critical paths dependent upon data size, data distribution, and system implementation.

Unfortunately, the same layers of middleware that enable distributed processing (e.g. MPI, PVM) convolute the critical data path. Fig. 1 illustrates the possible critical paths of data for communication in a simple distributed shared memory machine or cluster. The chosen path (and cost of the communication) depends on the destination, the data size and distribution, and the system implementation of the middleware.

Consider communication in a cluster of shared memory computers. Depending on the underlying communication scheme (e.g. MPI, PVM), the chosen implementation (e.g. MPICH [1]), and the system architecture design, different communication buffers will be used along the critical data path. For small data sizes, communication proceeds without application-level buffering. For large data sizes, data buffering at the application level will occur. Buffering will also occur at the network level.

The relative cost of memory communication is increasing. Processor speeds continue to outpace memory latency improvements. As the gap widens, memory performance becomes an even larger portion of overall execution time. It is very important to limit the cost of memory accesses. As mentioned, compilers can help, but given the complexity of present and future distributed systems the first step toward this solution is to identify the critical path of memory communication in a real system and quantify the cost.

Due to the increasing importance of memory, in this paper we seek to describe a scientific approach to empirically determine the critical path of memory communication. Our approach follows traditional methods of characterizing memory hierarchies applied to memory communication. We additionally apply a model of memory communication cost to characterize the effects of locality on observed buffer transmission. Our goal is to identify and quantify the costs of memory communication automatically. We test our methodology on a cluster of SMPs to show its usefulness and verify correctness.

2. Related Work

Gropp and Lusk [1] provide the mpptest tool for measuring MPI message performance accurately. The mpptest tool can be used for limited buffer identification. Our work couples this basic approach with traditional methods of cache hierarchy evaluation to automate buffering cost identification.

Limited work has targeted the memory communication cost of non-contiguous data. Ashworth [3] provides an application specific benchmark for non-contiguous communication in regular-partitioned, grid-based, distributed finite difference models. This work is solely empirical and contextually specific, drawing no general conclusions regarding non-contiguous communication performance for message passing applications.

Much research has focused on the network communication cost of message passing for contiguous data. Dongarra et. al. [4] provides a good overview of message passing performance issues and measurements. Existing parallel communication models like LogP [5] focus on network communication delay with limited consideration of memory communication. Later derivates maintain this focus, extending LogP to long messages [6] and active messages [7].

More recently, LogP was extended to incorporate advanced memory communication cost. Similar to the aforementioned extensions, memory logP [8] trades increased model complexity for improved model coverage of communication cost on evolving architectures. Since this research focuses on enabling automated buffer identification, the additional model complexity is acceptable and necessary to quantify the individual contributions of contiguous and non-contiguous data.

Memory logP formally characterizes 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). 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. 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. P: the number of processor/memory modules. Point-to-point communication in the memory hierarchy implies P=1. More information about memory logP model can be found in [8]. Locality-conscious programming using cache blocking and array padding provide 118% speedup on average for this matrix transpose algorithm verse the MPICH derived data type version [17]. This shows a broad scope for improving the performance of middleware.

All the experiments are performed on SGI Origin 2000 at NCSA, which uses a cc-NUMA architecture running the IRIX version 6.5.14 operating system. Each node contains two MIPS R10000 processors [10]; each running at 195MHz, and 32kB two-way set associative, two-way interleaved primary (L1) cache and off-chip secondary cache of 4MB. The MIPS R10000 processor has two on-chip 32-bit registers to count 30 distinct hardware events. In our experiments we have measured the events related to total cycles (event 0), graduated instructions (event 17), memory data loads graduated (event 18), memory data stores graduated (event 19), L1 cache misses (event 25), L2 cache misses (event 26).

3. Quantifying Communication Cost

Communication cost for sending a data segment depends on architectural parameters (e.g. cache capacity) and code characteristics (e.g. data distribution) as explained in the memory-logP model. Typically a message transmission involves data collection, data copying to the network buffer and data forwarding to the receiver. When a data distribution is not contiguous, typically it is collected into a contiguous buffer before copying to the network buffer (see Fig. 1). This intermediate copying is costly as data sizes and strides increase resulting in additional capacity and conflict misses to the cache [12]. This can be done without extra buffer copying by directly copying to the network buffer. However the performance degrades further due to poor utilization of network buffer. Strided accesses decrease the efficiency of cache hierarchies designed to exploit locality (capacity misses). Caches with less than full associativity, often a small power of 2, suffer from mapping collisions under certain access patterns (conflict misses).

The parameters of the memory logP model capture architecture and code characteristics. Memory cost of transferring data of a specific size is a combination of unavoidable overhead (o), and effective latency (l) - a function of data size and distribution. Additional network latency after removing overlap exists in passing a message between two processors. Total communication time increases with memory latency (l). In our micro benchmark experiments, memory performance worsens with an increase in stride. We use the memory-logP model to enumerate the memory hierarchy performance so that a developer can improve the performance of those parts of code with locality-conscious optimizations. Quantifying the memory communication costs is the first step in bottleneck identification. Succeeding analysis can identify system buffer-related parameters.

Fig. 3 and Fig. 4 illustrate the communication cost and cause of 16-dword (16 x 8 bytes), strided data transfers using MPI Blocking Sends. The contribution of memory communication to total communication is obvious. As message sizes increase for fixed strides, data transfer time increases (Fig. 3) from additional conflict and capacity misses (Fig. 4) in the memory communication. The rate of cost increase is dependent upon the data distribution and the memory hierarchy characteristics.

Estimation of the o parameter of the memory logP model requires measurement of contiguous data transmission, a relatively simple task using micro-benchmarking techniques. We expect the o parameter increases proportionately as problem size and strides increase; that a scalable transmission method is chosen. Recent work [13] indicates that the overheads for packing and unpacking of MPI derived data types in implementations such as MPICH do 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.

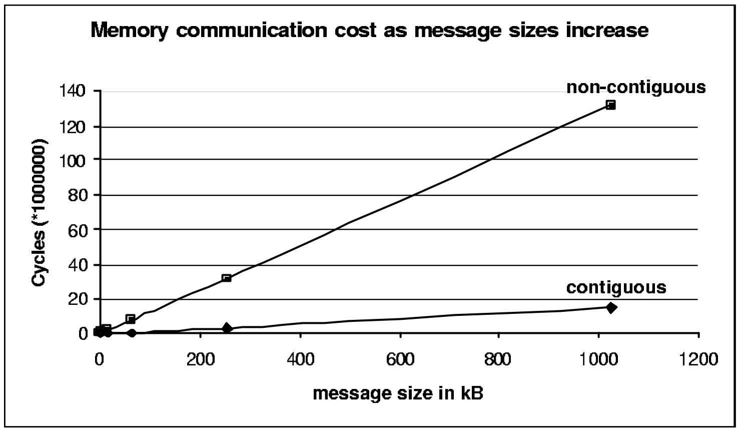

Fig. 2. Total cost for sending contiguous (stride 1) and non-contiguous (stride 16) messages. The costs are similar at low data sizes and it increases a lot once the data does not fit into the cache or when TLB thrashing occurs.

4. Identifying Memory Communication Buffers

Memory hierarchies are complex. System middleware (e.g. MPICH) provides abstractions (e.g. derived data types) to simplify distributed programming hiding the details of data transfer from the user. Determining the particular costs of memory communication is non-trivial due to the complex interaction between application, middleware, and hardware. However, to optimize performance, application developers must understand the full cost of communication. Using the quantifiable parameters of the memory logP model and micro-benchmark experiments, it is possible to identify buffer copies in shared memory architectures.

Specifically, we test various data sizes and strides iteratively and observe the largest gap among the successive hardware counter values after consideration of

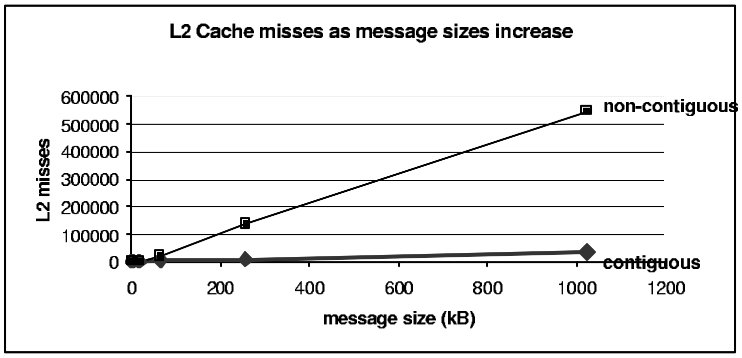

Fig. 3. L2 cache misses for sending contiguous (stride 1) and non-contiguous (stride 16) messages. Each L2 cache miss costs between 60 to 200 cycles.

experimental variation. These gaps or significant changes pinpoint policy decisions in the case of MPI codes (application buffers) and memory hierarchy characteristics (implicit buffering). At the memory hierarchy level, our approach is similar to that of traditional micro-benchmarking techniques [14, 15] used to identify general cache characteristics. We additionally verify our analyses with hardware counter data; this is particularly important for identifying application-level buffers.

Inefficient memory communication is not limited to exploitation of the memory hierarchy. Fig. 5 shows the increases in the latency parameter (l) with additional layers of overhead caused by middleware such as MPICH implementation is measured. It has been our experience that code developers targeting performance generally avoid certain abstractions such as derived data types since they understand the overhead resulting from such abstraction negatively and significantly impacts performance. Fig. 5 affirms this intuition. Fig. 6 shows the classification of various costs including (l) , (o) and other latency. Collective costs of middleware are very high with the increase of message size. This depicts that the magnitude of the cost differential is truly system and application dependent. Hence, a more scientific approach to determine when to use abstractions such as derived data types would involve determining the exact cost for a specific application-architecture combination. This is the purpose of our techniques and the original motivation behind this work.

Comparing the performance of communication for contiguous and strided messages can isolate the overhead caused by additional buffering. The cost of sending a contiguous message between two processors is a combination of data transfer overhead (o) and network latency. Sending a strided message has extra overhead due to poor exploitation of memory hierarchy and additional buffer copying. Optimally, the cost of copying strided data into a contiguous buffer is the same as the cost of packing it using MPI implementation. Additional middleware induced overhead is separated by subtracting the costs of sending contiguous message and that of

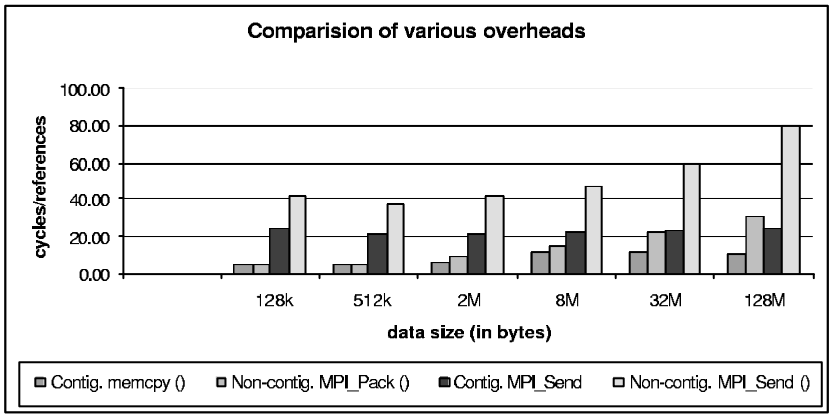

Fig. 4. Comparison of cost for various implementations of transpose algorithm. Contig.memcpy () : Copying data from one buffer to another using memcpy (). This is the basic overhead (o) to copy contiguous data. Non-contig. MPI_Pack () : This packs columns of matrix using MPI_Type_vector (). This cost is a combination of (o) and (l). Contig.MPI_Send () sends a contiguous message over network to another processor. This includes the cost of small contiguous copying overhead (o) and the cost for network transfer and software overhead of MPI_Send. Non-contig. MPI_Send () packs a non-contiguous data to transpose and sends to the receiver. This cost includes the packing overhead, copying data from memory to the network buffer and the network cost.

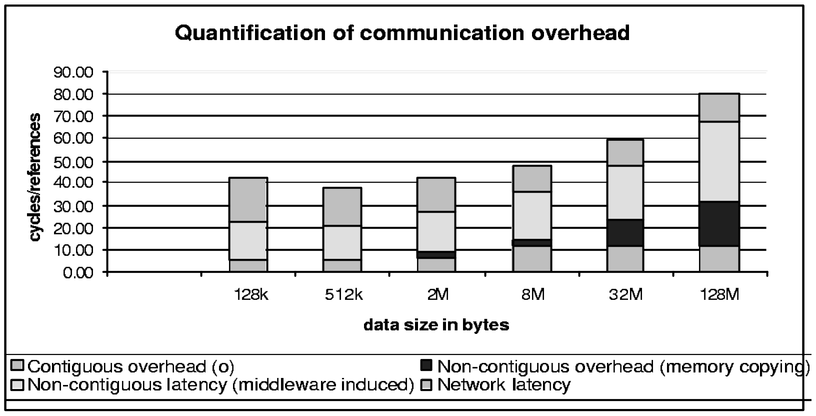

Fig. 5. From previous figure Contig. memcpy () = o, MPI_Send () contains the overhead due to copying of data from memory to the network buffer and network latency. Non-contig MPI_Send () has the additional buffer copying cost over MPI_Send () costs.

packing from the total cost of sending strided message. Fig. 6 depicts the partition of these costs. This is an empirical method of separating all the costs in memory communication. Presentation of these costs provides a developer with an insight into exploiting the advanced memory hierarchies, and to decide which critical data path is optimal to use.

5. Conclusions and Future Work

It was believed that data allocation is not a noticeable factor of communication in a parallel computing environment. All the existing parallel programming models consider cost of memory access either constant or negligible. Through our experimental testing, and case studies, in this research we have shown that memory communication is a function of data size and distribution. The performance degrades by a factor 10 times even with a small stride of 16. Communication performance can be improved more than 115% by using memory friendly optimizations external to compilers. This portrays a large scope for improvement of communication dominant applications and compilers.

Memory communication can be caused by many factors, under utilization fast CPUs with multiple levels of memory hierarchy, data distribution and various copying overheads between buffers. Application developers need to be aware of underlying architectures to develop high performance programs. But lack of documentation regarding the memory hierarchy and buffering schemes for various architectures is a source of difficulty in optimizing applications. Identification of these implementation based communication overheads is a part of enumerating the memory communication costs. In this paper we have presented an approach to determine the critical data path scientifically along with the memory access overhead as described earlier.

Towards modeling this memory communication, we have developed memory logP model [8]. 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%).

After recognizing the points of performance degradation, the next step is to optimize the performance. A program developer who is aware of all these details can be able to produce better solutions. But to achieve uniform and optimized performance there should be libraries and automatic development of these solutions to ease the burden on the programmer. Our future work is moving towards achieving these goals. One of our objectives is to improve the performance of MPI derived datatypes implementation, by observing memory operations. This work is progressing in collaboration with Argonne National Laboratory.

References

[1] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum, "A High-Performance, Portable Implementation of the MPI Message Passing Interface Stan-

dard", *Parallel Computing*, 1996

[2] W. Gropp and E. Lusk, "High Performance MPI Implementation on a Shared Memory Vector Supercomputer", *Parallel Computing 1996*

[3] M. Ashworth, "The OCCOMM Benchmarking Guide, Version 1.2", http://www.dl.ac.uk/TCSC/CompEng/OCCOMM/, *March 1996*

[4] J. Dongarra and T. Dunigan, "Message Passing Performance of Various Computers", *Concurrency: Practice and Experience*, Vol. 9 No. 10, pp 915-926, 1997

[5] D.Culler et al. "LogP: Towards a Realistic Model of Parallel Computation", *In Proceedings of Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 1-12, San Diego, California, May 1993

[6] A.Alexandrov, MF Ionescu, KE Schauser, and C. Scheiman. "LogGP: Incorporating Long Messages into the LogP Model - One Step Closer Towards a Realistic Model for Parallel Computation", *In Proc. Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 95–105, Santa Barbara, CA, July 1995

[7] Csaba Andras, Moritz Matthew, I. Frank. "LoGPC: Modeling Network Contention in Message-Passing Programs", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 12, No. 4, April 2001

[8] K. Cameron, X.H. Sun, "Quantifying Locality Effect in Data Access Delay: Memory logP", *International Parallel and Distributed Processing Symposium (IPDPS)*, 2003

[9] SGI MIPSpro family compilers, http://www.sgi.com/developers/devtools/languages/mipspro.html

[10] National Center for Supercomputing Applications Archives, (NCSA) "Understanding Performance on the SGI Origin 2000" NCSA Online document, URL: http://archive.ncsa.uiuc.edu/SCD/Perf/Tuning/Tips/Tuning.html

[11] Shirley Moore, Nils Smeds, "Performance tuning using hardware counter data", *Presentation at SuperComputing*, 2001, Nov. 01

[12] Monica Lam, Edward E. Rothberg and Michael E. Wolf, " The cache performance of blocked algorithms", *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991

[13] 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*, 2002, Fort Lauderdale, FL, 2002

[14] R. H. Saavedra , R. S. Gaines , M. J. Carlton, Micro benchmark analysis of the KSR1, *Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, p.202-213, December 1993, Portland, Oregon, United States

[15] R. Clint Whaley, Antoine Petiet and Jack Dongarra, "Automated Empirical Optimizations of Software and the ATLAS project", *Parallel Computing*, Vol. 27, Num. 1-2, pp. 3-25, 2001

[16] Message Passing Interface (MPI) Standard Specification, http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html

[17] S. Byna, K. Cameron, X-H. Sun, "Quantifying Memory Communication", *Submitted for publication.*