Memory-aware Communication – an Experimental Study with MPI*

Surendra Byna
Department of Computer Science, Illinois Institute of Technology, Chicago, IL
renbyna@iit.edu

Kirk W. Cameron
Department of Computer Science and Engineering, Univ. of South Carolina, Columbia, SC
kcameron@cse.sc.edu

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

Abstract

Assuming network transfer is the dominant factor of communication, current communication models estimate only network related delays and are inadequate to address other performance factors such as memory access delay. Modern computer architectures employ complex hierarchical memory systems to compensate for the performance gap between processor and memory, nonetheless significant delays occur when transmitting non-contiguous data. This study revisits the common assumption that memory access patterns are a negligible factor of communication. Our experimental results show non-contiguous memory access may increase the communication cost multi-fold in a cluster environment, confirming the need for rethinking MPI implementation and parallel algorithm design, and motivating discussion of a memory-aware communication model.

1. Introduction

The processor–memory performance gap [3] is a bottleneck for high performance computer systems. Many architectural advances in memory hierarchy design have targeted this gap in an effort to increase the effective bandwidth of memory access. Techniques that overlap latency, reduce cache misses and increase the instruction throughput of the processor are common in commodity architectures. Implementations include features such as larger and faster hierarchical caches, blocked data transfer, non-blocking transfers, superscalar and out-of-order execution, and branch prediction. Despite these technical advances, memory performance remains the dominant contributor to execution time in sequential codes.

The communication performance of parallel programs is not immune to the effects of memory latency especially in the case of non-contiguous message communication, where a message distributed at various locations of memory. Non-contiguous message communication is common in parallel computing. However, assuming network is the dominant factor of communication, memory factor has been largely ignored in current communication models. Our experimental results show that recent technology advances have changed the communication cost distribution. Memory access pattern has become an increasingly important factor of communication. Non-contiguous communication may increase communication overhead by multi-fold in a cluster environment. The high cost of non-contiguous communication requires a rethinking of parallel algorithm design, MPI [11, 9] implementation, and calls for a memory aware communication model.

Message passing of non-contiguous data types is common in parallel programming. The spatial locality of data impacts the performance of parallel algorithms such as the ocean grid solver and Barnes-Hut [6] and other domain decomposition based algorithms. The ocean grid solver exchanges data along horizontal and vertical boundaries. In many domain partition based solutions the boundary data is contiguous and spatial locality is optimal in the cache. When the boundary is non-contiguous (e.g. column boundary in a row-ordered language implementation), the amount of cache misses increase based on the contiguity of the data. The Barnes-Hut application initially operates on adjacent particles with good spatial locality for communication. As the simulation progresses, particles travel through physical space decreasing the spatial locality of communication causing additional cache-related delays. Another example, transmission of a sub-matrix may require a series of non-contiguous accesses incurring more memory latency than contiguous accesses of the same size.

* This research was supported in part by national science foundation under NSF grant EIA-0130673, ANI-0123930, ACI-0130458, and by Army Research Office under ARO grant DAAD19-01-1-0432.

Transmissions of data in such cases often utilize the message-passing model, a widely used and accepted parallel programming interface [11].

To quantify the impact of non-contiguity on communication, we study the total cycles, cache misses, and load/store instructions for contiguous and non-contiguous MPI point-to-point communication in this research. In the remainder of this paper, we discuss previous attempts at modeling communication performance. Next, we describe the details of non-contiguous message communication in MPI. Section 4 addresses the effects of memory access on a parallel application's point-to-point communication performance. Section 5 describes the experimental setup and finally, in Section 6 we present our experimental results.

2. Related work

Much research exists regarding the communication cost of message passing for contiguous data. Dongarra et. al. [18] provides a good overview of message passing performance issues and measurements. Gropp and Lusk [8] provide the mpptest tool for measuring MPI message performance accurately. This tool is currently part of the MPICH distribution [20] and through experimentation can help tweak parameters in MPI implementations for specific platforms. Few studies have targeted the performance of non-contiguous data. Ashworth [21] 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.

Existing parallel communication models focus on network interface communication delay. LogP [5] is a popular, realistic model for parallel computation, which provides a basis for many succeeding models. L is the upper bound for the latency, or delay in communicating a message from its source network buffer to the destination network buffer. o, the overhead, is the amount of time the processor is busy during the transmission or reception of a message. g, the gap, is defined as the minimum time interval between consecutive message receptions. P is the number of processor/memory modules. The LogP model is accurate for fixed-size, small messages. LogGP [1], where G represents the gap per byte, extends LogP for systems that provide hardware support for long messages. Both models ignore network contention since it is a difficult factor to quantify in estimates of point-to-point communication. For the special case of active messages, contention is quantified as the time an interrupt spends queued at the target processor. This motivates an additional C parameter for contention, the LoGPC model [2]. Model accuracy and complexity increase with the number of parameters.

The above models approximate the communication latency over the network quite well. None of them addressed memory access delay at the communication end points. In our effort to project the inevitability of introducing a new communication model, we study the effect of memory access time as a parameter in this paper. Moreover, previous models and measurements average overhead at the sender and receiver using half of round-trip time as the total communication cost. Since the overhead depends on the message data distribution at the sender and receiver, overhead may be asymmetrical. Round-trip time measurements are thus imprecise for non-contiguous data. They are also imprecise in the context of NUMA architectures, where the memory access delay is not same for all the processors.

3. Non-contiguous message passing communication in MPI

Two methods in MPI can be used to communicate non-contiguous data. The first is to manually pack all the blocks of data into a contiguous buffer at the sender node and send this data block to the receiver. At the receiver the user has to manually unpack to retain the structure of the data. Here after this method is termed as "user pack/unpack" method in this paper. This has the disadvantage of requiring additional memory-to-memory copy operations at both nodes even when the communication subsystem has scatter-gather capabilities. To reduce tedious job of sending non-contiguous data, MPI implementation provides a mechanism called "derived datatypes". Derived datatypes are a means to describe layouts of data in memory. A general datatype is constructed as a sequence of basic datatypes and a sequence of integer displacements. The basic datatypes are MPI_INT (integers in C), MPI_FLOAT, MPI_DOUBLE, and MPI_CHAR. The displacements need not be positive, distinct or in increasing order. A type map is the combination of data types and displacements. This type map, together with a base address buf, specifies a communication buffer: the communication buffer consists of n entries, where the i-th entry is at address buf + dispi. [11]

In MPICH implementation, type map of a derived data type can be represented as a data type tree. Leaf nodes correspond to basic datatypes and are characterized by their size in number of bytes. The internal nodes correspond to derived types, and are described by their repetition count (for vectors), their extent, size and their children with

associated displacements. The number of children depends on the type of the node. For instance, a structure or indexed type has as many children as there are components in the structure, while a vector has only one child.

The tree representation suggests how a type map can be reconstructed and used when packing an instance of a derived datatype to a communication buffer. By a simple recursive procedure the tree is traversed from root to leaves computing the correct offsets in both user and communication buffers during the descent. When a leaf is reached copying of the leaf data takes place. The repetition count of each internal type node determines how many times this node's sub-trees are visited. The overhead with the use of derived datatypes lies in traversing the data structure that stores type map. Traff [16] makes an effort to reduce this recursive traversal of the tree data type in his 'flattening on the fly' technique. In any algorithm, the overhead consists of the following elements:

1. Time to traverse the data structure to find the next displacement.
2. Time to calculate the next memory block location
3. Access that memory location, which may contain cache miss penalty due to the large message and page faults or the misses due to the spatial locality of the data.

All the non-contiguous memory communication operations can be divided into three types. [8]

1. Fixed length block of data with fixed stride
2. Variable length block with fixed stride
3. Fixed length block with variable stride.

In our experiments we used the first method, where the fixed length data with fixed stride is passed between two processors.

4. Overhead in message passing

In parallel program communication of non-contiguous data, cache misses cause additional non-overlapped stall time. Cache misses are related to the message size and the data distribution. In our analysis, we study the memory costs in message-passing point-to-point communication. Our observation projects that the send/recv time of a message in message passing communication is a combination of startup time, time to copy the data from application memory to either the buffer or to the buffer on network interface card from where the data would be transferred. Portions of latency will be overlapped and must be removed.

$$T_{send/recv} = t_{st} + t_{ac} + t_{mem} + t_{buf} - t_{ol} \quad (1)$$

t_{st} is the startup time for communication such as sending a request to start the communication. t_{ac} is the time to calculate the address of the memory location from where the next data block has to be copied. This cost is significant when the data is non-contiguous. t_{mem} is the cost of the copying the message from the sender's application memory to the local/NIC buffer or from the receiver's local/NIC buffer into its application memory. This includes the cache miss penalty. If the message size is greater than the buffer space available, the sender blocks (assuming blocked communication) until the buffer is free (t_{buf}). t_{ol} is present if there is any overlapping between the above mentioned times. Due to the latest scalar processing technology more than one instruction can be processed within one cycle. In that case some overlapping would exist where some of t_{ac} can be overlapped by t_{mem}. This is a simple formula of costs that contribute to the overhead at an end point. Finding out each cost is the real problem for communication model.

The major factor of variation in the above formula is memory access time for the distributed data. In this case, MPI calculates the memory address of the next block when derived datatypes are used. Non-contiguous memory accesses cause cache misses where the resulting penalties that constitute additional overhead. In this paper, we conduct exhaustive experimentation to project the amount of memory effect based on non-contiguity. Overhead costs may be asymmetric. For example in a row-ordered programming model, a column of a matrix sent is received as a row at the receiver. The memory access patterns differ at send/receive ends of transmission so the memory access overhead is different for the same size message. In Equation 1, the send and receive time are same when data is distributed similarly.

As mentioned in section 3, non-contiguous messages can be passed using MPI specified derived datatypes, and user pack/unpacking method, where user manually packs the data at the sender and unpacks at the receiver. The tradeoff is between the extra memory, tedious packing/unpacking job of the user pack/unpack method and the degraded performance of derived datatypes. In user pack/unpack technique, the time to calculate the next memory location is easy as the user provides those values. The cost of derived datatypes depend on how many times it needs to access the tree data structure that contains the block information and stride information of message.

The other overhead includes copying blocks of data into a separate contiguous memory location. The performance of long and contiguous messages is better as they try to avoid the use of buffering between the sender user memory and receiver user memory. In that way, the additional memory copying time is saved. Current implementations of derived

data types are still using the buffer before sending a message, which increases the communication overhead. In DSM machines such as SGI Origin 2000, the message is directly copied into the shared memory, and the receiver accesses that directly.

5. Experimental setup

5.1. SGI Origin 2K Architecture

Distributed Shared-Memory (DSM) multiprocessors provide the convenience of shared memory programming with a scalable design. The SGI Origin 2000 at NCSA utilizes a cc-NUMA architecture running the IRIX version 6.5.14 operating system. The interconnection network for 128 processors is a 5th degree hypercube with 4 processors (2 nodes) per router. High-speed, dedicated Craylink interconnects link nodes. 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. As long as the communication is between nodes within a hypercube, per hop latency is zero, but the communication to an outer cube node causes increase in latency.

A directory based tree protocol maintains cache-coherence. A complex memory hierarchy reduces the impact of memory latency. Each node contains two MIPS R10000 processors [12]; 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 MIPS R10000 is a four-way superscalar RISC processor. The machine used in testing has 48 195MHz MIPS R10000 processors, with 14 GB main memory. The available local memory access bandwidth is 680MB/sec in each direction. SGI O2K machine is selected as our platform because of the availability of library to access the hardware counters. As our study is mainly concentrating on the effect of local memory references this can be generalized for commodity cluster architectures.

5.2. Hardware counters

All the commodity processors provide hardware performance counters to measure and validate the processor architecture. 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). The overhead of cache misses on SGI Origin 2000 is measured [mucc97] as 1 cycle for register access, 2-3 cycles for an L1 cache hit, 7~13 cycles for an L1 cache miss, 60~200 cycles for an L2 cache miss. This shows that the overhead increases massively if a L2 cache misses occur. We have chosen these counters to study the memory effect on communication as they reflect the memory operations for any processor.

5.3. Performance testing program

We used a program, which is similar to mpptest [8] programs that measures the performance of contiguous blocking, non-blocking and non-contiguous message communication. In this program we inserted hardware counter measuring routines before and after each MPI communication calls to measure number of cycles, cache misses, total instructions, load/store instructions. Messages are passed between two processors. All these tests are performed on NCSA's SGI Origin 2000 machine with the environment specified earlier in this section. In non-contiguous message passing program, the sender processor sends a block of a matrix, with various strides for various message sizes. We defined a data type called columntype, which collects the blocks, each of block size 1, which are a stride apart. We measured the primary cache misses, load/store instructions to show how memory references are affecting communication performance with data distribution and message size.

6. Results and analysis

Our experiments are divided into three categories.

1. Blocking communication overhead of contiguous and non-contiguous data, with various message sizes ranging from 64bytes to 1024kB.
2. Blocking and non-blocking communication cost for contiguous and non-contiguous data, with various message sizes
3. Blocking communication overhead of non-contiguous data, with the use of derived datatypes and user pack/unpack method.

We measured the primary cache misses, load/store instructions to show how memory references are affecting communication performance with data distribution and message size.

6.1. Contiguous message vs. Non-contiguous message passing

Fig. 1 shows how the number of cycles is increasing with message size at sender processor, with a fixed stride of 16. When the message size is less, most of the message is within the cache and the number of cache misses is less. Increase in the message size, causes more memory operations (load/store) and when the total message size is more than the cache memory available, cache misses increase. As explained in the section 4, with the number of cache misses increase, the amount of memory stall time increases affecting communication time. Until the message size is below 256kB increase in number of cycles is below the proportionate factor due to the processor's pipelining. After 256kB the sender starts blocking to wait for a matching receive before sending a message. That causes extra waiting time at the sender. In a separate experiment we measured the number of load/store instructions. Number of loads and stores increase with the message size, which proves that the overhead at a sender/receiver is a parameter of memory operations. The number of load and store instructions also increases below the proportionate factor with the message size until the explicit blocking at the sender starts. When the buffering starts load/stores increase more than the message size increment factor.

The reasons for extra cycles in the case of non-contiguous message passing are increase in cache misses and blocking property of MPI blocking send. We investigated further how cache misses are affecting the total number of cycles. Fig. 2 shows the number of cache misses information. As we have stated earlier, after 256kB message, the number of cache misses are increasing at a greater number. Due to this the total cost of communication

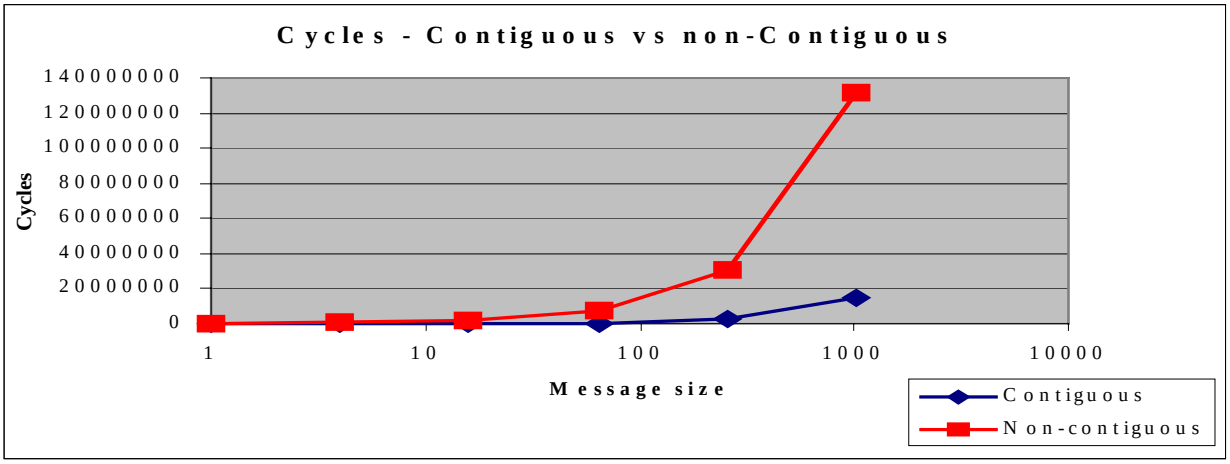

Fig. 1 Number of cycles for contiguous vs. non-contiguous message passing

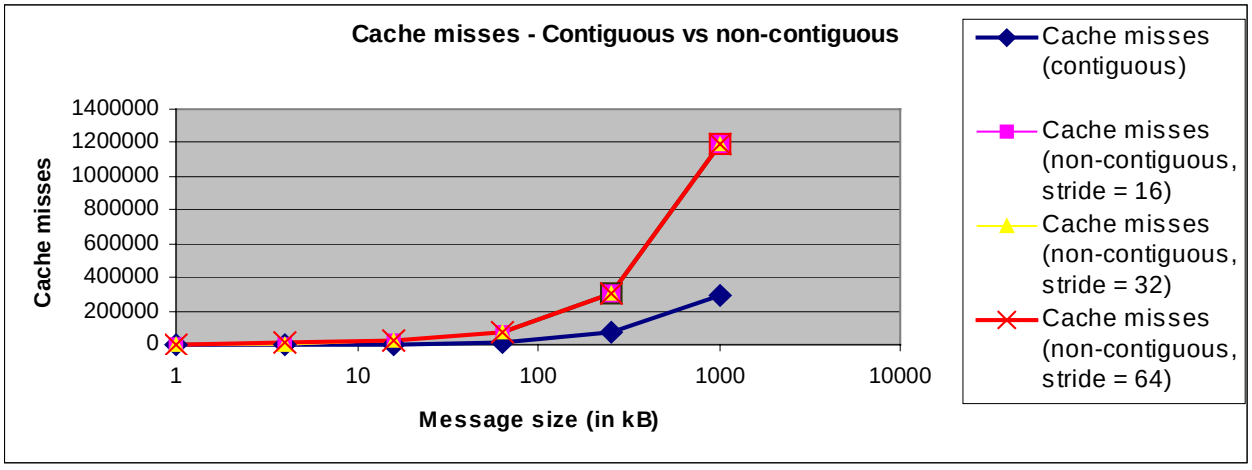

Fig. 2 Number of primary cache misses for contiguous vs. non-contiguous message passing

from sender to receiver cannot be considered as a constant as previous models are assuming. We have investigated further into the next level of cache, secondary cache to observe the contribution at that hierarchy level. Fig. 3 shows how the secondary cache misses affects the local communication. Each secondary level cache misses costs approx. 200 cycles, which means the total cost increases massively with secondary miss. The misses are very high at larger message sizes.

6.2. Blocking vs. Non-blocking

In our experiments, we have measured the total number of cycles and primary cache misses in order to study the impact of memory on these communication patterns. MPI_Send is used for blocking send and MPI_Isend is used for non-blocking communication. We observed that until the message size is 64kB, the difference between blocking and non-blocking communication is insignificant. But from 256kB, the communication is insignificant. But from 256kB, the blocking communication overhead is very high compared to non-blocking communication for both contiguous and non-contiguous message data.

Fig. 4 shows that the number of cycles or cache misses for blocking communication are almost equal or a bit higher than that of non-blocking communication. But, for non-contiguous messages, the ratio of blocking to non-blocking communication overheads is 3 and 10 at message sizes, 256kB and 1GB. This is because, blocking send starts buffering at this point and it causes extra memory copy operations, which is not necessary for non-blocking send. If the buffer cannot accommodate to store the entire message, then the message is sent in parts, which add up to the buffer wait time. This illustrates the need of including buffer

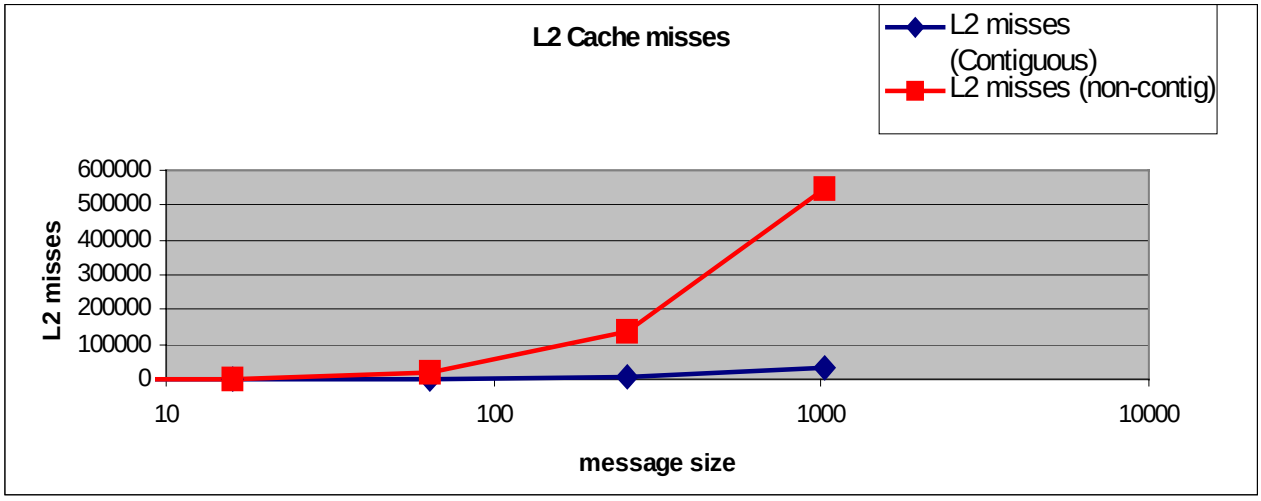

Fig. 3 Number of secondary cache misses for contiguous vs. non-contiguous message passing

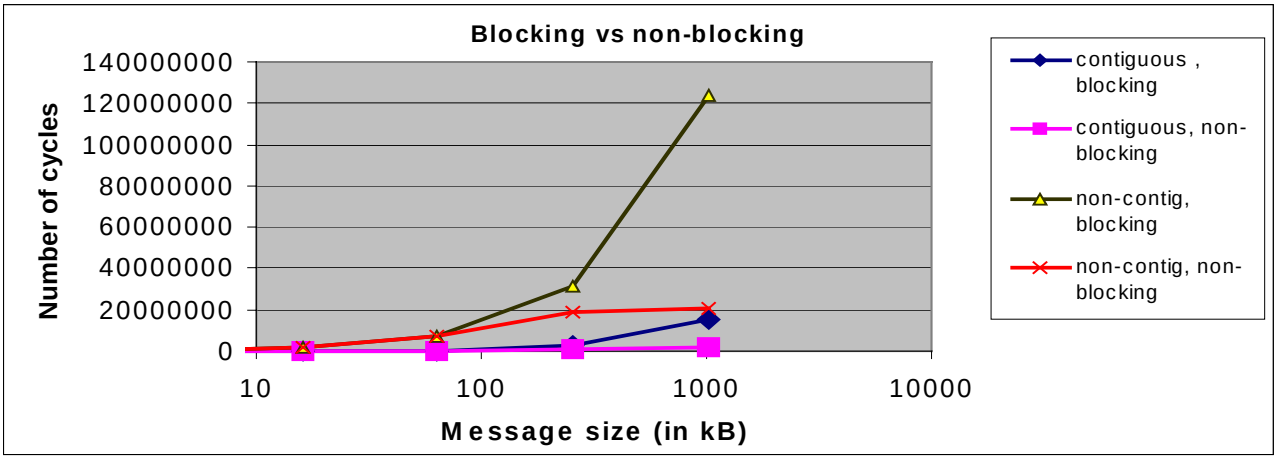

Fig. 4 Blocking vs. non-blocking performance for contiguous and non-contiguous messages

waiting time as a factor of communication overhead. The increase of memory effect with message size depends on the buffering protocols and the implementation of the communication calls.

6.3. User pack/unpack vs. Derived Datatypes

In other experiments, we have measured the number of instructions, loads, stores and the primary cache misses for contiguous and non-contiguous message sending, with varying stride and blocking communication. In non-contiguous communication, we used pack/unpack method such as PVM, and MPI derived datatypes. But derived datatypes proved to have more overhead cost, which degrades the performance. We observed that the increase of overhead with the stride is not very high. But the difference in memory effect between contiguous and non-contiguous communication is large. The difference between pack/unpack method and derived datatypes method is also very high. This shows that the derived datatypes are increasing the overhead due to the displacement computation. In derived datatypes method, the stride effect starts increasing when the stride is more than the block size of the cache.

Measurements for sending a contiguous block of 4 kB are: 43,230 cycles, 9,058 graduated instructions, 3,119 load instructions, 2,390 store instructions, and 1231 L1 Misses. Table 1 and Table 2 illustrate the number of memory parameters for the point-to-point communication sending performance of non-contiguous data in two methods, the MPI based derived datatypes and user packing (packed manually) method. As explained earlier, in MPI derived datatypes, the data is directly copied from the various addresses dynamically. In user packing, the user packs all the non-contiguous data into one buffer and sends it. All the counters, (cycles, graduated instructions, loads and store instructions) are very high for derived datatypes method. Moreover the overhead is increasing with the stride (16, 32, and 64) gradually. This increment factor depends on cache size and its block size. With the user packing the increase is very small, which is almost negligible. According to our measurements, the ratio of performance with MPI derived datatypes and user packing is at least 4. This increases with the stride, as the user packing doesn't cost any further increase with the stride. This is mainly due to the excessive cost to compute the next data block in non-contiguous data using derived datatypes. This is the tradeoff between the user's effort to pack and the performance. The measurements for larger message sizes proved the same, that the derived data communication cost is far greater than the user packing method, for fixed block size and varying message size non-contiguous data. Based on these results we state that the time to traverse the tree data type to obtain the displacements is pretty high. If the data structure that stores the displacements is made simple, then the performance could become as good as that of the user pack/unpack method. In both cases the number of memory references is very high compared to that of sending a contiguous message. This incites a necessity of a communication model that is aware of memory references. A further study is in progress to break down the overhead costs, which enables to observe the distribution of tree traversal cost.

7. Conclusion

It was believed that data allocation is not a noticeable factor of communication in a cluster-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 operations are a major factor in communication cost at a sender or a receiver, especially when data is non-contiguous. In blocking communication, the number of memory operations increase proportionately with message size and non-contiguity until there is no blocking. With the added blocking the overhead cost increases drastically. Experimental results show that non-contiguous data allocation can increase communication overhead ranging from 4 to 10 times. The ratio of blocking to non-blocking communication overheads is 3 to 10. Performance with user packing is better than MPI derived datatypes by 3 to 8 times. This is proved by the variation of number of cycles, cache misses, load/store instructions. As the performance evaluation of parallel applications give great insight into its design to parallel programmers, it is important to project the communication overhead at seder/receiver, so that the design can be improved.

This drastic performance degradation of MPI leads many programmers to use their own functions instead of using MPI. It's the responsibility of performance analysts to provide convenience to the programmers, so that a consistent performance is achieved and the cost of hiring highly knowledgeable programmers. We have introduced the concept of memory-aware communication to emphasize the data allocation effect on communication. This study proves that there is a necessity of introducing a new communication model that deals with these memory references. MPI implementation is a major success for the message-passing paradigm. MPI provides a convenient way to handle the non-contiguous messages. The current MPI implementations do not fully aware the data access factor in communication. These need to be re-engineered and improved by using better flattening techniques.

8. Future work

We are currently developing memory aware communication model for parallel computing. We practically applied our techniques to two architecturally distinct systems, an IA32 Beowulf and the MIPS-based SGI Origin 2000. Using that model we accurately (within +80% and –60%) predicted the cost of regular packing and unpacking algorithms for varying data types and architectures. Our other objective is to improve the performance of MPI derived datatypes implementation, by observing memory operations. This work is progressing in collaboration with Argonne National Laboratory. Currently we are working on breaking down the costs defined in equation (1). Based on these observations of our experiments we have developed Memory-logP model, which predicts parallel application communication latency considering data access delay. l parameter of Memory-logP model is the additional cost caused due to the non-contiguous accesses. More details about this model can be found in [21].

9. Acknowledgements

We thank Dr. Bill Gropp, Argonne National Laboratory, and Dr. Eric Salo, developer of SGI MPI, who helped us in understanding MPI implementation. We are grateful to NCSA, which allowed us to run our programs on their SGI Origin 2000 machine.

10. References

[1] 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.

[2] 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.

[3] John L Hennessy and David A Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufman, CA, 1996.

[4] C.K. Chow, "On Optimization of Storage Hierarchies", *IBM J.Research and Development*, pp. 194-203, May 1974.

[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] D. Culler, J.P. Singh, A. Gupta: *Modern Parallel Computer Architecture*, 1997, Morgan Kaufmann

[7] Xing Du, Xiaodong Zhang, Zhichun Zhu, "Memory hierarchy considerations for cost-effective cluster computing", *IEEE Transactions on Computers*, Vol. 49, No. 9, 2000.

[8] William Gropp, Ewing Lusk, and Deborah Swider "Improving the Performance of MPI Derived Data types ", *in the Proceedings of MPIDC'99*

[9] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum, "A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard", *Parallel Computing* 1996

[10] Bruce L. Jacob, Peter M. Chen, Seth R. Silverman, Trevor N. Mudge. "An Analytical Model for Designing Memory Hierarchies", *IEEE Trans. Computers*, vol. 45, no. 10, pp. 1180-1194, October 1996.

[11] Message Passing Interface Forum: "MPI: A message passing interface standard", *International Journal of Supercomputing Applications*, 8(3/4), 1994.

[12] 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

[14] James Laudon and Daniel Lenoski. The SGI Origin: A ccNUMA Highly Scalable Server. In *Proceedings of the24th Annual International Symposium on Computer Architecture (ISCA'97)*, Denver, CO, USA. pp.241-251

[15] X.H. Sun and K. Cameron, "A Statistical-Empirical Hybrid Approach to Hierarchical Memory Analysis," *Proc. of Euro-Par 2000, Lecture Notes in Computer Science, Springer, Sept. 2000.*

[16] Jesper Larsson Träff, Rolf Hempel, Hubert Ritzdorf, Falk Zimmermann. Flattening on the Fly: efficient handling of MPI derived datatypes. *In Recent Advances in Parallel Virtual Machine and Message Passing Interface. 6th European PVM/MPI Users' Group Meeting*, volume 1697 of Lecture Notes in Computer Science, pages 109-116, 1999

[17] T. A. Welch, "Memory Hierarchy Configuration Analysis", *IEEE Trans. on Computers*, vol. 27, no. 5, pp. 408-415, May 1978

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

[19] W. Gropp and E. Lusk, "A High Performance MPI Implementation on a Shared Memory Vector Supercomputer," *Parallel Computing*, Vol. 22 No. 11, pp 1513-1526, January 1997.

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

[21] K. Cameron, X.H. Sun, "Memory logP and its implications", *SCS Laboratory Technical reports*, Illinois Institute of Technology, Chicago. URL: www.cs.iit.edu/~scs

Stride	Cycles	Graduated Instructions	Loads	Stores	L1 Misses
16	570,943	651,383	114,292	55,886	5,954
32	712,683	726,577	132,641	132,577	8,190
64	1,018,968	874,800	167,225	76,208	12,688

Table. 1 Hardware counter values for non-contiguous messages sent using derived datatypes

Stride	Cycles	Graduated Instructions	Loads	Stores	L1 Misses
16	145,963	128,106	32,132	18,839	1,779
32	140,150	128,292	32,063	18,903	1,797
64	149,629	128,782	31,438	19,031	1,947

Table. 2 Hardware counter values for non-contiguous messages sent using user pack/unpack