Optimizing Performance on Linux Clusters Using Advanced Communication Protocols: How 10+ Teraflops Was Achieved on a 8.6 Teraflops Linpack-Rated Linux Cluster

Manojkumar Krishnan and Jarek Nieplocha

Computational Sciences and Mathematics Division
Pacific Northwest National Laboratory

Abstract. Advancements in high-performance networks (Quadrics, Infiniband or Myrinet) continue to improve the efficiency of modern clusters. However, the average application efficiency is as small fraction of the peak as the system's efficiency. This paper describes techniques for optimizing application performance on Linux clusters using Remote Memory Access communication protocols. The effectiveness of these optimizations is presented in the context of an application kernel, dense matrix multiplication. The result was achieving over 10 teraflops on HP Linux cluster on which LINPACK performance is measured as 8.6 teraflops.

1 Introduction

Linux clusters offer openness, flexibility, low cost, and reliability, and achieve high efficiency and high productivity. The high-performance networks, such as Myrinet, Quadrics, and InfiniBand improve a cluster's overall efficiency over Ethernet-based networks. However, the average application efficiency is still only a small fraction of the peak. Even for computationally intensive benchmarks, such as LINPACK, those do not stress the network as much as many scientific applications, approaching peak performance is difficult. For example, the theoretical peak performance (R_{peak}) of the HP Linux cluster at Pacific Northwest National Laboratory [1] and the IBM Linux cluster at Barcelona Supercomputer Center [1] is 11.6 and 31.4 teraflops respectively, where as the maximum LINPACK [2] performance (R_{max}) is only 8.6 and 20.5 teraflops respectively. Most of the scientific applications achieve performance lower than R_{max} [3].

This problem has its sources in multiple areas including hardware and software. One of the primary factors degrading the application performance is the growing gap between 1) CPU and network speed 2) CPU and memory speed[4]. In spite of significant progress in the commodity networks, the gap between processor speed and inter-processor communication performance is growing. For example, in 1990 on the NCUBE/2 massively parallel system employing a 1MFLOP/s processor and 2.5 MB/s network bandwidth, the message-passing latency ranged from 65-130µs. On the other hand, the latest 1.5 GHz Itanium-2 processor is rated at 6GFLOP/s and is employed in

Linux clusters connected with networks (e.g., Quadrics QsNet^{II}) that support ~3μs latency and 850 MB/s network bandwidth at the MPI [5] layer. During the last 14 years, the processor speed and network bandwidth is improved by a factor of 10^4 and $10^{2\text{-}4}$ respectively, and latency is improved only by a factor of 10. This growing gap is not specific to commodity clusters only. For example, the Cray X1 processor (MSP) mode is rated at 12.8GFLOP/s while the MPI latency is just ~9 μs. The processor/memory performance gap is growing at a similar rate. Each memory access cost on average 10 or even hundreds of processor cycles[4]. Although increasingly large caches can help reduce the performance gap, they only work for applications that can reuse cached data and/or exploit data locality. Therefore, this growing gap between CPU-network and CPU-memory is a fundamental problem that requires attention in the design of communication models as well as scalable parallel algorithms.

In this paper, we discuss techniques for addressing some of the implications of the technology trends, and illustrate how they could be used in practice by using an example of the dense matrix multiplication operation. We also advocate the remote memory access (RMA) communication model because of its simplicity and good hardware support on modern networks it possesses certain characteristics important for reducing the performance gap between system peak and application performance. This gap can be reduced by combining quality implementation of the communication interfaces with algorithms capable of exploiting locality and using optimal types of memory for its communication buffers. The techniques and protocols discussed in this paper are: 1) zero-copy protocol 2) network latency hiding through effective non-blocking communication 3) explicit control of data locality and task mapping e.g., exploiting shared memory within SMP nodes and RMA across network 4) reducing communication contention in access to the data.

In this paper, we present a case where a parallel matrix multiplication kernel, effectively uses these techniques to achieve improved performance on Linux clusters. In many scientific applications as well as in HPL (high performance LINPACK), parallel dense matrix multiplication is one of the most important linear algebra operations. The experimental results on the HP cluster with Quadrics QsNet^{II} network demonstrate that incorporating these techniques and protocols in the matrix multiplication operation indeed offer real and measurable performance improvements and help close the gap between peak and observed performance. For example, on 1849 processors, we achieved efficiency of 88.2% of the theoretical peak performance on the HP Linux cluster at PNNL, where as the maximum LINPACK efficiency recorded on this cluster is only 74%. Our parallel matrix multiplication uses sequential *dgemm* from vendor optimized math library (HP-mlib) for Itanium^2, which performs at 93.5% dgemm efficiency.

This paper is organized as follows: Section 2 outlines RMA communication model. Section 3 described the techniques and fast communication protocols that can be used in an application to achieve optimum performance. Section 4 describes a matrix multiplication kernel and how these techniques are incorporated in this kernel, and presents experimental results, and the paper is concluded in Section 5.

2 RMA Communication Model

Remote memory access (RMA) operations facilitate an intermediate programming model between message passing and shared memory. This model combines some advantages of shared memory, such as direct access to shared/global data, and the message-passing model, namely the control over locality and data distribution. Certain types of shared memory applications can be implemented using this approach. In some other cases, remote memory operations can be used as a high-performance alternative to message passing [6]. On many modern platforms, RMA is directly supported by hardware and is the lowest-level and often most efficient communication paradigm available [7]. RMA is sometimes considered a form of message passing; however, an important difference over the MPI-1 message-passing model is that RMA does not require explicit receive operation and thus offers increased asynchrony of data transfers (see Figure 1).

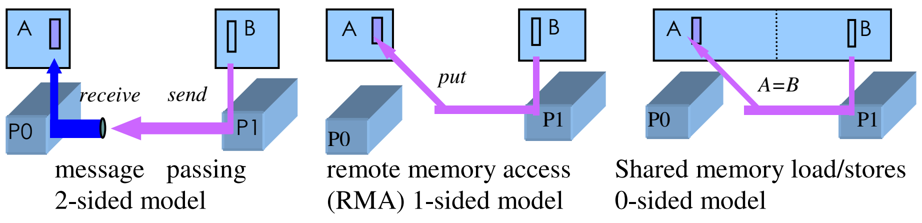


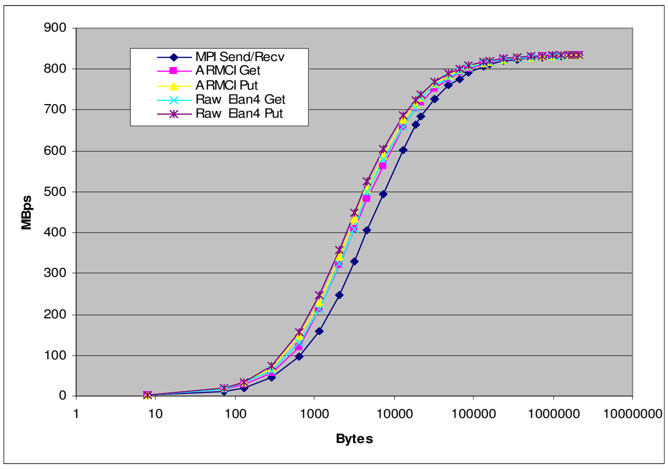
Fig. 1. Taxonomy of communication models.

In our experiments, we used a portable RMA interface called aggregate remote memory copy interface (ARMCI). A portable RMA interface is needed both for developing applications and for creating a communication layer for libraries and compiler run-time systems, especially for the (re)emerging global address space languages. ARMCI was developed to serve the latter purpose [7-9] by complementing MPI-2 [10], which targets application developers and imposes certain rules and restrictions on data access (e.g., window serialization, access epochs) or progress rules that are absent in vendor-specific interfaces such as the Cray SHMEM [11], IBM LAPI [12], and Quadrics Elan [13].

3 Techniques for Optimizing Communication Performance

3.1 Asynchronous RMA Communication

The traditional RMA communication facilitates data transfers between a buffer of a local processor and another location in the remote processor memory. However, cooperation with the remote processor is not required to complete the data transfer. The RMA model is closely aligned with RDMA capabilities of modern networks (Infiniband, Myrinet, VIA, Elan), which provide hardware support to read from or write to remote memory locations. With the exception of Elan, which offers virtual memory RDMA, the networks listed above require the source and destination buffers to be registered with the network adapter in advance of the communication. Registration allows the network adapter driver to establish virtual memory translations and lock the buffers in physical memory. RDMA is a simple communication model that enables network adapters on two ends of the network to complete data transfer asynchronously and avoid remote-host processing. This simplicity makes RDMA, in most cases, the highest performance data-transfer mechanism available. If communication buffers are registered, RMA operations such as put or get map directly to the RDMA write or put operations supported by the hardware. Since the RMA model does not require a remote processor to match message tags or deal with early message arrivals, as required in message passing, RMA can achieve higher performance on these networks, as well.



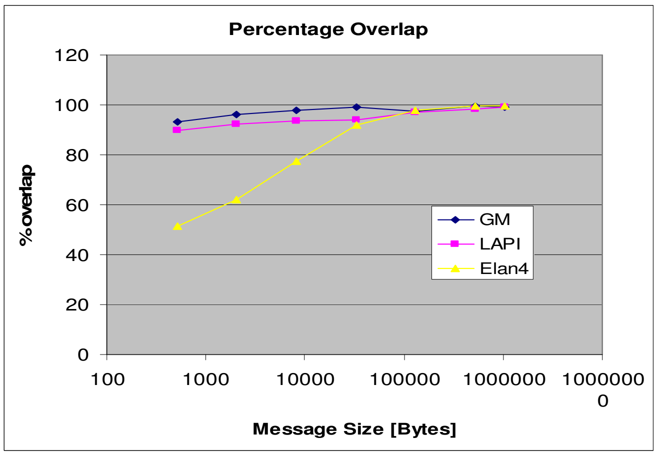


Fig. 2. (a) Bandwidth in ARMCI Put/Get in comparison to Raw Elan4 Get/Put and MPI Send/Recv on the Elan4 cluster**. (b)** Percentage overlap for increasing message sizes for MPI and ARMCI on various platforms (SP, Elan4, GM).

The first benchmark is designed to demonstrate the performance of ARMCI Put and Get operations, which is based on RMA programming model, on the HP IA64 Linux cluster with Quadrics QsNet^{II} network. It also demonstrates the effectiveness of the low overhead implementation of these operations over the native communication protocols (Elan4). Figure 2a compares the performance of ARMCI Put/Get with MPI Send/Recv and the Native RDMA Put/Get on the Quadrics Elan-4 cluster. The benchmark used here clearly shows that ARMCI operations introduce very low overhead over the native Quadrics protocols. ARMCI does quite well compare to MPI since it avoids the usual tag-matching overheads that the message-passing libraries have to provide.

3.2 Latency Hiding

Latency hiding (or latency tolerance) can be accomplished through different techniques, including overlapping communication with computation [14] by the use of nonblocking communication [15, 16] and zero-copy protocols. Another technique, is coalescing small put/get messages (i.e., aggregation) [17] into larger ones to eliminate startup cost for as many messages as possible and to improve network utilization. The availability of non-blocking RMA operations presents additional opportunities for overlapping data transfers and computations. Although pre-fetching and post-storing instructions are often supported by the shared memory hardware and are exploited by compilers to overlap computations with data movement, a scientific programmer on shared memory systems typically faces difficulties when attempting to manage explicitly overlapping of computations and communication due to the lack of precise APIs. Such explicit non-blocking APIs are present in the most RMA interfaces.

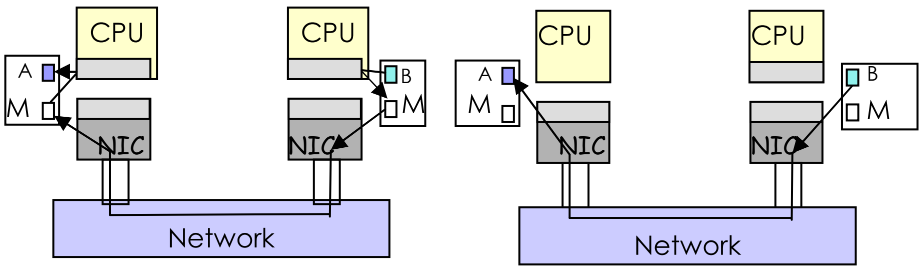
3.2.1 Non-blocking Communication Interfaces

Nonblocking operations initiate a communication call and then return control to the application. The user who wishes to exploit nonblocking communication as a technique for latency hiding by overlapping communication with computation implicitly assumes that progress in communication can be made in a purely computational phase of the program execution when no communication calls are made. Unfortunately, that assumption is often not satisfied in practice in MPI-- the availability of nonblocking API does not guarantee that overlapping communication with computation is always possible [18]. The RMA interfaces of the high performance networks (GM, ELAN, LAPI) include nonblocking interfaces and provide good potential for overlapping communication with computations.

We conducted a test to demonstrate the effectiveness of RMA non-blocking communications. The overlap microbenchmark deals with overlapping communication with computation, and it was performed in the context of available RMA based network protocols (IBM-SP (LAPI), Linux cluster with Myrinet (GM) and Quadrics (Elan4) interconnects). The percentage overlap is measured as the amount of time of a nonblocking (data transfer) call that can be overlapped with useful computation without increasing the overall benchmark time. We consider this a good measure of the effectiveness of a non-blocking call. The computation is incorporated in the program in the form of a delay. Increasing computation is gradually inserted between the initiating nonblocking get call and the wait completion call. As we keep increasing the computation, at some point the sum of the nonblocking call issue overhead and computation would exceed the idle CPU time, so the total benchmark running time would increase. This point gives us the maximum possible overlap. We performed this experiment on two nodes, with one node issuing the nonblocking *get* for data located on the other and then waiting for the transfer to be completed in the *wait* call. From Figure 2b, we observe that the RMA model offers a higher level of overlap on these networks. On Elan4, the overlap efficiency is not close to 100% up to 64KB because, the latency to transfer data up to 64KB is very low (e.g. 13 micro seconds for 8K message size) and overhead of the non-blocking and wait call initiation times with respect to latency is high (approximately 3 microseconds).

3.2.2 Zero-Copy Protocols

The zero-copy communication is increasingly important due to the growing gap between processor and memory speeds: memory b/w is also a precious resource shared in SMP clusters between multiple processors. When implemented it allows remote CPU to work on its own computations rather than be interrupted and involved in data copying on behalf of another processor (see Figure 3b). RMA model is well suited for zero-copy implementation on the networks such as Quadrics, Infiniband and Myrinet, since both source and destinations are known and memory buffers can be preregistered. In that case put/get operations are handled by DMA engines on the NIC, thus avoiding multiple memory copies (see Figure 3a) on local as well as on the remote CPU. Thus there is more opportunity for overlapping communication as both the local and remote CPU can do useful computations while data transfer is taking place.



Copy-based, high CPU involvement

Zero-copy, low CPU involvement e.g., Quadrics

Fig. 3. Data Movement Schemes: **(a)** copy-based protocol and **(b)** zero-copy protocol.

For example, the communication protocols used to optimize blocking transfers of data from non-registered memory by pipelined copy and network communication through a set of registered memory buffers [7] can achieve very good performance by tuning the message fragmentation in the pipeline [19]. However, pipelining is not effective for non-blocking communication as memory copy requires the active host CPU involvement and therefore reduces the potential for effective overlapping communication with computation. To increase the overlap, we expanded the use of direct (zero-copy) protocols on networks that require memory registration, such as Myrinet.

3.3 Data Locality and Shared Memory

The fastest communication protocols available on modern high-performance clusters are shared memory within the SMP nodes and RDMA between the nodes. To take full performance advantage of these protocols, exploiting data locality information is critical in implementation of communication protocols but we also advocate it at the application level. Specifically, with appropriate task mapping information, applications can in fact use shared memory to communicate between tasks within the same node and RMA between the nodes.

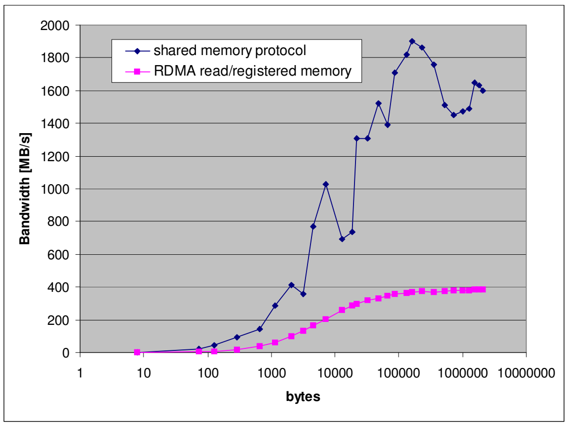


Fig. 4. Performance of get operation between RDMA read from registered memory and within the SMP node using shared memory.

Performance benefits of using registered shared memory is demonstrated below for a Linux cluster employing dual 1GHz Itanium-2 nodes interconnected with the Mellanox, Infiniband "Cougar" cards. Figure 4 shows the performance of a get operation implemented on top of shared memory and implemented using an RDMA read operation (memory is registered), both within the SMP node. Because a single network adapter must move the data across the PCI bus twice within an SMP node, the RDMA read bandwidth is about half of the rate for that operation across the network. The shared memory communication profile is slightly affected by caching of data and directly follows performance of the memory copy operation.

4 Example - a parallel matrix multiplication

In the previous section, optimization mechanisms to improve the overall performance of scientific applications were discussed along with system level tests to validate the effectiveness of the optimization mechanisms and advanced communication protocols. The effectiveness of these protocols is evaluated by incorporating these techniques in the context of an RMA based parallel matrix multiplication algorithm. For the comparison, we used the *pdgemm* routine from PBLAS/ScaLAPACK Version 1.7, which uses message passing communication model.

```
for i=0 to s-1 {
    for j=0 to s-1 {
        Initialize all elements of $C_{ij}$ to zero (optional)
        for k=0 to s-1 {
            $C_{ij} = C_{ij} + A_{ik} \times B_{kj}$
        }
    }
}
```

Fig. 5. Block matrix multiplication for matrices N×N and block size N/s × N/s.

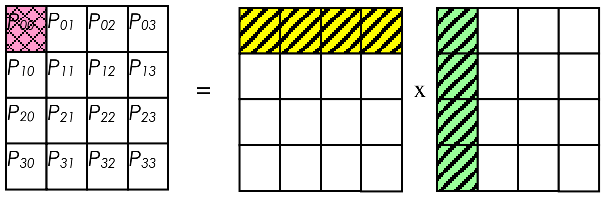


Fig. 6. Matrix distribution example. In a 4 x 4 process grid, process P_{00} needs blocks of matrix A from P_{00}, P_{01}, P_{02}, and P_{03}, and blocks of matrix B from P_{00}, P_{10}, P_{20}, and P_{30}.

For many scientific applications, matrix multiplication is one of the most important linear algebra operations. At the high level, the parallel matrix multiplication algorithm follows the serial block-based matrix multiplication (see Figure 5 and 6) by assuming the regular block distribution of the matrices A, B, and C. Each process accesses the appropriate blocks of the matrices A and B to multiply them together with the result stored in the locally owned part of matrix C.

In principle, the overall sequence of block matrix multiplications can be similar to that in Cannon's algorithm. However, unlike Cannon's algorithm, where skewed blocks of matrix A and B are shifted using message-passing to the logically neighboring processors, our approach fetches these blocks independently, as needed, without requiring any coordination with the processors that own the matrix blocks. This is possible thanks to the use of RMA or shared memory access protocols. In addition, the specific sequence in which the block matrix multiplications are executed is determined dynamically at run time to more efficiently schedule and overlap communication with computations. The absence of sender-receiver synchronization/coordination (such in Cannon's algorithm) based on message passing makes the overall algorithm more asynchronous and thus more suited for the execution environments where the computational threads share a CPU with other processes and system daemons (e.g., on commodity clusters). This is because synchronization amplifies performance degradations due to the nonexclusive use of the processor by the application. In this algorithm, we start computations by accessing matrix A and B blocks within the same SMP node through shared memory as suggested in 3.3 while nonblocking RMA calls are issued for getting matrix blocks on remote nodes.

4.1 Role of Zero-copy and Non-blocking Protocols

Zero-copy non-blocking is used in this application to transfer matrix blocks, the communication can be overlapped with computation, as shown in Figure 7. To demonstrate the effectiveness of zero-copy protocol, we evaluated what degree zero-copy RMA communication affected the performance of matrix multiplication. As zero-copy protocol is default in Quadrics elan4 network, we used Myrinet to investigate the role zero-copy protocols play in performance of the matrix multiplication. The new matrix multiplication algorithm was tested with enabling and disabling the zero-copy implementation [19] of the ARMCI get operation. Figure 8 shows that zero-copy protocol is very important for performance of the new algorithm. This test is performed on the Linux cluster with Myrinet, using 1) blocking and non-blocking communications, and 2) zero-copy protocol disabled and enabled. These results show that the performance benefit of using nonblocking communications is amplified when the zero-copy protocol is enabled. This is because; the remote host CPU cycles are not taken away when overlapping communication with computation since the NICs are able to transfer the data between the user buffers across the network. We were able to overlap more than 90% of the communication with computation, thus improving the overall application performance.

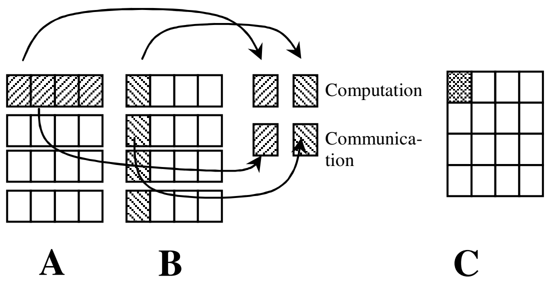


Fig. 7. Using two sets of buffers to overlap communication and computation in matrix multiplication.

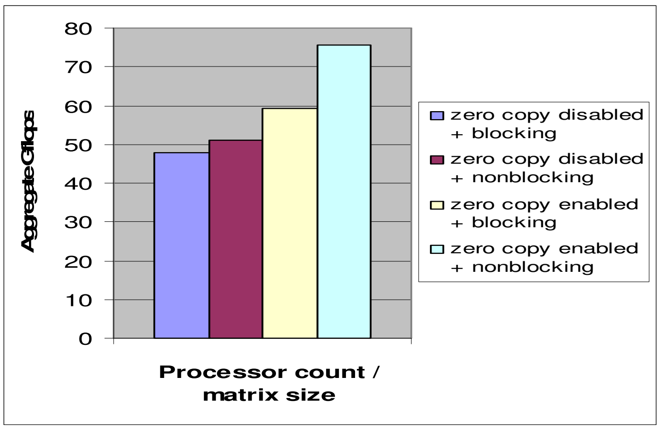
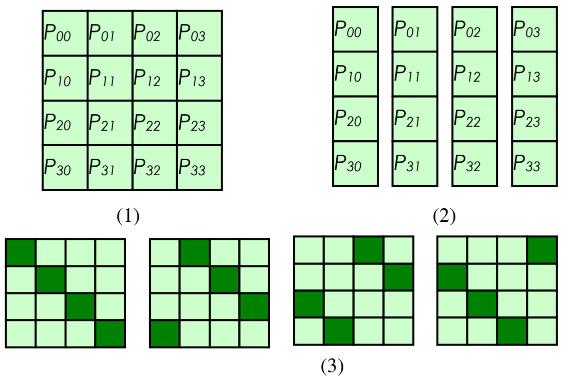


Fig. 8. Performance of the matrix multiplication on Linux Cluster (Myrinet Interconnect) with enabled or disabled zero-copy protocol.

4.2 Communication Contention Management

An effective communication pattern in parallel algorithms reduces the network comunication contention on clusters and thus effectively utilizing network bandwidth. In our parallel matrix multiplication algorithm we used a network contention algorithm (NCA) as shown in Figure 9. For example, consider a matrix *A* that is distributed on a 4 x 4 processor grid (as shown in Figure 9 (1) on a 4-way SMP cluster, i.e., node 1 has processors P_{00}, P_{10}, P_{20}, *and* P_{30}; node 2 has P_{01}, P_{11}, P_{21}, *and* P_{31}; etc., as shown in Figure 9 (2). To compute its locally owned matrix *C*, a processor needs the corresponding rows and columns of matrix *A* and *B* respectively, as shown in Figure 7. i.e., processor P_{00} needs blocks of matrix *A* from P_{00}, P_{01}, P_{02}, and P_{03}, and blocks of matrix B from P_{00}, P_{10}, P_{20}, and P_{30}. If the network contention algorithm is not used, processors P_{00}, P_{10}, P_{20}, and P_{30} get a block from P_{01}, P_{11}, P_{21}, and P_{31}, respectively in the first step. Thus all the 4 processors are sharing the network bandwidth between node1 and node2. If the diagonal shift algorithm is used instead, then processors P_{00}, P_{10}, P_{20}, and P_{30} get a block from P_{00} (node0), P_{11} (node1), P_{22} (node2 and P_{33} (node3, respectively in the first step, thus reducing the contention. Figure 9 (3) represents the pattern of getting blocks by processors in node 1.

The communication load in the multiplication kernel is spread uniformly using a network contention algorithm as shown in Figure 9, thus improving network bandwidth utilization. Figure 10 shows the effect of incorporating network contention algorithm (NCA) in blocking and non-blocking flavors of matrix multiplication on an 8000 size matrix multiplication on Quadrics HP Linux cluster at PNNL. PBLAS/ScaLAPACK (which uses MPI) matrix multiplication numbers are also shown in the Figure 10. From Figure 10, it is demonstrated that as the number of processors increase the non-blocking and NCA version of matrix multiplication scales extremely well when compared to others. As the number of processors increase, even the blocking with NCA version of matrix multiplication performs better than the non-blocking version without NCA.



P_{00} P_{01} P_{02} P_{03}
P_{10} P_{11} P_{12} P_{13}
P_{20} P_{21} P_{22} P_{23}
P_{30} P_{31} P_{32} P_{33}
(1)
(2)
(3)

Fig. 9. Pattern of getting blocks on a 4-way SMP cluster to reduce communication contention.

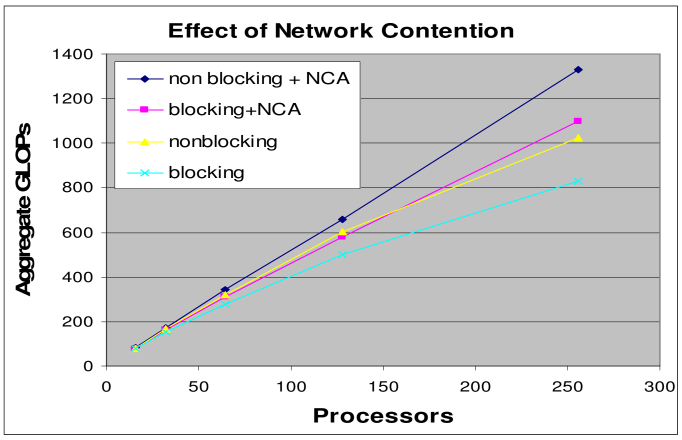


Fig. 10. Effect of network contention algorithm (NCA) in parallel matrix multiplication.

4.3 Performance and Scalability

Experiments were conducted for various problem (matrix) sizes and processor counts on the HP Linux cluster based on 980 dual 1.5-GHz Intel Itanium^2 nodes and Quadrics QsNet^{II} network at PNNL. The (square) matrix sizes range from 2000 till 40000 for up to 1849 processors. Due to operational constraints, we could not run on the whole machine (1960 processors). Figure 11 shows the performance numbers of the optimized matrix multiplication (called SRUMMA [20, 21]) and *pdgemm* from

PBLAS/ScalaPACK. SRUMMA achieved a maximum of 9.7 teraflops on 1849 processors, where as PBLAS achieved lower than 8 teraflops. For a problem size of 40000 on 1849 CPUs, our implementation achieves a parallel *dgemm* efficiency of 88.2% of the theoretical machine peak performance on the HP Linux cluster at PNNL, where as the PBLAS *pdgemm* efficiency is 74%. Extrapolating these numbers for 1960 processors, we would get 10.3 teraflops, which recorded a maximum LINPACK performance is 8.6 teraflops. Thus, using advanced communication protocols optimal application performance can be obtained on Linux Clusters. The vendor optimized sequential *dgemm* (mlib from HP) is 92.9%.

Figure 12 presents the performance differences for square matrices, no transpose; matrix ranks range from 200 to 8000 on Quadrics Linux cluster. As shown in Figure 12, the SRUMMA outperforms *pdgemm* and scales better, with the most profound gains seen are due to the incorporation of these optimizations. For example on the Linux cluster, it is faster by 20% to 60% in most of the cases. In most of the cases, we were able to overlap 90% of the communication with computation.

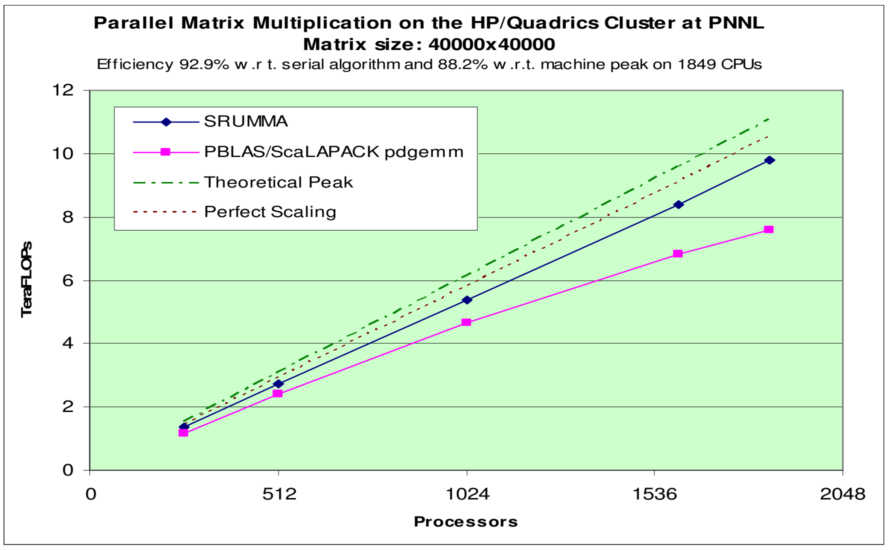


Fig. 11. Parallel Matrix Multiplication (SRUMMA) performance on the HP/Quadrics cluster at PNNL

5 Conclusion

This paper described communication performance optimizations that are relevant to modern clusters. We showed impact of these optimizations on performance of parallel matrix multiplication kernel. It achieved 10.3 teraflops on a 8.6 teraflop (R_{max}) Linux supercomputer. For a problem size of 40000 on 1849 CPUs, parallel *dgemm* efficiency of 88.2% of the theoretical peak performance was measured, whereas the

PBLAS *pdgemm* efficiency was 74%. We found remote memory access (RMA) communication model valuable because of its simplicity and good hardware support on modern networks it possesses certain characteristics important for reducing the performance gap between system peak and application performance.

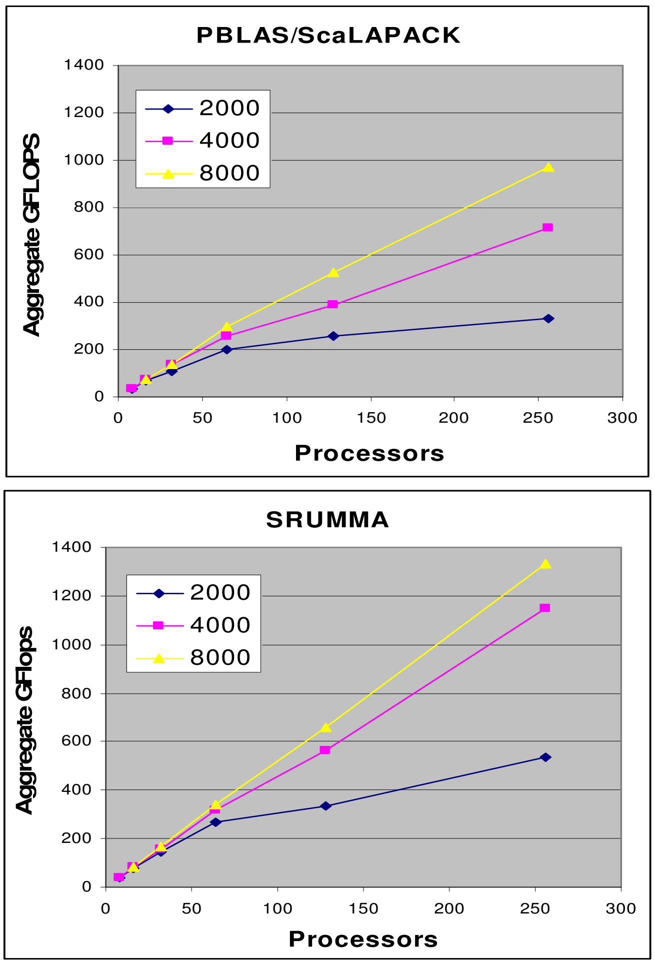


Fig. 12. SRUMMA vs. PBLAS pdgemm.

References

[1] Top500. Top 500 Supercomputer List. www.top500.org.

[2] J. Dongarra, "The LINPACK Benchmark: An Explanation," in proceedings of Proceedings of the 1st International Conference on Supercomputing, 1987.

[3] Olaf Lubeck, Adolfy Hoisie, Federico Bassetti, Kirk Cameron, Yong Luo, and H. Wasserman, " ASCI Application Performance and the Impact of Commodity Processor Architectural Trends," in proceedings of International Workshop on Innovative Architecture, 1998.

[4] W. Wulf and S. McKee, "Hitting the memory wall: Implications of the obvious," in *ACM Computer Architecture News*, 1995.

[5] MPI. The Message Passing Interface Forum. www.mpi-forum.org.

[6] C. Guiang, K. Milfeld, and A. Purkayastha, "Remote memory operations of Linux clusters: expressiveness and efficiency of current implementation," in proceedings of 3rd LCI International Conference on Linux Clusters, St. Petersburg, Florida, 2003.

[7] J. Nieplocha, V. Tipparaju, A. Saify, and D. K. Panda, "Protocols and strategies for optimizing performance of remote memory operations on clusters," in proceedings of Communication Architecture for Clusters (CAC'02) Workshop, held in conjunction with IPDPS '02, 2002.

[8] J. Nieplocha and B. Carpenter, "ARMCI: A Portable Remote Memory Copy Library for Distributed Array Libraries and Compiler Run-time Systems," in proceedings of RTSPP of IPPS/SDP'99, 1999.

[9] J. Nieplocha, E. Apra, J. Ju, and V. Tipparaju, "One-Sided Communication on Clusters with Myrinet," *Cluster Computing*, vol. 6, pp. 115-124, 2003.

[10] MPI-2, "Message Passing Interface Forum. MPI-2: Extensions to the Message-Passing Interface."

[11] R. Bariuso and A. Knies, *SHMEM's User's Guide*: Cray Research, Inc., 1994.

[12] G. Shah, J. Nieplocha, J. Mirza, C. Kim, R. Harrison, R. K. Govindaraju, K. Gildea, P. DiNicola, and C. Bender, "Performance and experience with LAPI-a new high-performance communication library for the IBM RS/6000 SP," in proceedings of International Parallel Processing Symposium IPPS/SPDP, 1998.

[13] F. Petrini, S. Coll, E. Frachterberg, and A. Hoisie, "Peformance Evaluation of the Quadrics Interconnection Network," *Journal of Cluster Computing*, 2003.

[14] V. Strumpen and T. L. Casavant, "Exploiting Communication Latency Hiding for parallel network computing: model and analysis," in proceedings of PDS, 1994.

[15] S. B. Baden and S. J. Fink, "Communication overlap in multi-tier parallel algorithms," in proceedings of Proceedings of Supercomputing, 1998.

[16] E. Culler, A. Dusseau, S. Goldstein, A. Krishna-murthy, S. Lumetta, T. Eicken, and K. Yelick, "Parallel programming in Split C," in proceedings of Proc. Supercomputing, 1993.

[17] D. Pham and C. Albrecht, "Optimizing Message Aggregation for Parallel Simulation on High Performance Clusters," in proceedings of 7th Intern. Symposium on Modeling Analysis and Simulation of Computer and Telecommunication Systems, 1999.

[18] J. B. White and S. W. Bova, "Where's the overlap? Overlapping communication and computation in several popular mpi implementations," in proceedings of Third MPI Developers' and Users' Conference, 1999.

[19] R. Y. Wang, A. Krishnamurthy, R. P. Martin, T. E. Anderson, and D. E. Culler, "Modeling and Optimizing Communication Pipelines," in proceedings of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, 1998.

[20] M. Krishnan and J. Nieplocha, "SRUMMA: a matrix multiplication algorithm suitable for clusters and scalable shared memory systems," in proceedings of Parallel and Distributed Processing Symposium, 2004.

[21] M. Krishnan and J. Nieplocha, "Optimizing Parallel Multiplication Operation for Rectangular and Transposed Matrices," in proceedings of 10th IEEE International Conference on Parallel and Distributed Systems (ICPADS'04). 2004.