Improving Performance and Energy Efficiency of Matrix Multiplication via Pipeline Broadcast

Li Tan, Longxiang Chen, and Zizhong Chen
University of California, Riverside
{ltan003, lchen060, chen}@cs.ucr.edu

Ziliang Zong
Texas State University-San Marcos
zz11@txstate.edu

Rong Ge
Marquette University
rong.ge@marquette.edu

Dong Li
Oak Ridge National Laboratory
lid1@ornl.gov

***Abstract*—Boosting performance and energy efficiency of scientific applications running on high performance computing systems arise cruicially nowadays. Software and hardware based solutions for improving communication performance have been recognized as significant means of achieving performance gain and thus energy savings for such applications. As a fundamental component of most numerical linear algebra algorithms, improving performance and energy efficiency of distributed matrix multiplication is of major concerns. For such purposes, we propose a high performance communication scheme that fully exploits network bandwidth via non-blocking pipeline broadcast with tuned chunk size. Empirically, substantial performance gain up to 8.4% and energy savings up to 6.9% are achieved compared to blocking pipeline broadcast, and against binomial tree broadcast, performance gain up to 6.5% and energy savings up to 6.1% are observed on a 64-core cluster.**

Keywords*—*distributed matrix multiplication; performance; energy; pipeline broadcast; binomial tree broadcast; ScaLAPACK.

I. Introduction

The ever-growing number of cores and bandwidth of network for interconnected processors in a distributed-memory computing system provide unprecedented capability for large-scale computation. Boosting performance and energy efficiency of scientific applications by exploiting parallelism in high performance computing systems has become a significant issue, which motivates a large amount of hardware-related [1] [2] [3] [4] and software-based solutions [5] [6] [7] [8] [9] [10] [11] [12]. Among these approaches, improving communication efficiency has been recognized as the cornerstone of achieving performance gain during the execution of applications running on distributed-memory computing systems.

Matrix multiplication serves as a fundamental component of most numerical linear algebra algorithms like LU, Cholesky, and QR factorizations [13]. The classic algorithm of matrix multiplication on a distributed-memory computing cluster performs alternate broadcast and matrix multiplication on local computing nodes [14]. Different distributed broadcast algorithms have been devised and industrialized [15] [16] for high performance communication, including basic linear, chain, pipeline, binary tree, and binomial tree. Binomial tree and pipeline broadcast in general outperform other algorithms for different system configurations [17].

Regarding pipeline broadcast, two distinct types, non-blocking and blocking pipeline broadcast represent sending and receiving messages with shared CPU usage with other routines and dedicated CPU usage individually, which determines different resource utilization and slack pattern in the communication of distributed matrix multiplication. Many factors, including pipeline length and time latency to reach the maximal pipeline can impact the communication performance.

In this paper, we introduce a high performance communication scheme via pipeline broadcast to achieve performance and energy efficiency for distributed matrix multiplication. Specifically, we take advantage of a non-blocking pipeline broadcast scheme with tuned chunk size to boost performance of communication, with which network bandwidth is exploited more thoroughly compared to binomial tree broadcast. In summary, the contributions of this paper are as follows:

- We model and quantify the communication time complexity of binomial tree and pipeline broadcast, and analyze communication slack in two types of pipeline broadcast to achieve performance and energy efficiency in distributed matrix multiplication;
- The non-blocking pipeline broadcast with tuned chunk size is evaluated to achieve significant speed-up and energy savings compared to blocking pipeline broadcast and binomial tree broadcast on a 64-core cluster.

The rest of this paper is organized as follows. Section 2 discusses related work and section 3 details distributed matrix multiplication algorithms. We present high performance pipeline broadcast compared to binomial tree broadcast in section 4. We provide implementation details and evaluate our approach in section 5, and section 6 concludes the paper.

II. Related Work

Improving Communication Performance at System Level: Chan *et al.* [3] redesigned and reimplemented many MPI communication algorithms to achieve that one node can communicate with multiple nodes simultaneously with lower costs, rather than the traditional one-to-one at a time algorithms. Faraj *et al.* [7] presented a customized system that generates efficient MPI collectve communication routines via automatically-generated topology specific routines and performance tuning to achieve high performance consistently. Karwande *et al.* [6] presented an MPI prototype that supports compiled communication to improve performance of MPI communication routines, which featured that it allowed the user to manage network

978-1-4799-0898-1/13/$31.00 ©2013 IEEE

resouces to aggressively optimize communication. Hunold *et al.* [8] proposed a mechanism that automatically selected a suitable set of blocking factors and block sizes for pdgemm() to maximize performance. Our approach differs from these techniques, since it improves MPI communication performance via implementing a pipeline broadcast that maximizes the slack utilization, without modifying MPI communication routines and any parameters of the pdgemm() routine interface.

Improving Communication Performance at Algorithm Level: Solomonik *et al.* [4] mapped 2.5D dense linear algebra algorithms to allow the algorithms to exploit optimized line multicasts and reductions that cannot be leveraged in 2D algorithms. Speed-ups were achieved due to communication reductions from rectangular collectives over binomial tree broadcast. Ballard *et al.* [9] developed a novel parallel matrix multiplication algorithm based on Strassen's algorithm with minimized communication. Given the observation that the critical bottleneck of performance gain was the communication, the authors reached the lower bounds on communication costs and the experimental results scaled well. For improving performance of sparse matrix multiplication with most execution time comsumed on inter-processor communication, Ballard *et al.* [10] proved a new lower bound on the expected communication costs and proposed two new parallel algorithms to match the expected lower bound. Ballard *et al.* [11] observed that maintaining numerical stability via partial pivoting in LU factorization involved row interchanges leading to inefficient data access patterns. The authors introduced a shape morphing procedure that dynamically matches the data layout to the computation and demontrated that Gaussian Elimination can be performed in a communication efficient fashion.

Improving Pipeline Broadcast Performance: Desprez *et al.* [5] made efficient use of pipelining on LU factorization and a column-scattered data decomposition to derive precise variations of computational complexities, where a synchronous communication scheme and overlapping communication by computation were exploited. Watts *et al.* [1] addressed the problem of conducting pipelined broadcast on a mesh architecture. The proposed algorithm worked on meshes of any dimension with any number of nodes. Beaumont *et al.* [2] handled broadcast on heterogeneous platforms where messages were routed in a pipelined spanning tree fashion. Given that most systems supporting pipeline parallelism used a construct-and-run approach, Lee *et al.* [12] investigated on-the-fly pipeline parallelism where pipeline emerged as the program ran, and proposed simple linguistics for specifying a provably efficient scheduling algorithm in a work-stealing fashion. Our work focuses on analyzing the efficiency of binomial tree and pipeline broadcast and quantifying the communication costs.

III. Details of Distributed Matrix Multiplication Algorithms

Matrix multiplication is one fundamental operation of most numerical linear algebra algorithms for solving a system of linear equations, such as LU factorization, Cholesky factorization, and QR factorization [13]. Matrix multiplication is also widely used in many other areas, such as computer graphics, quantum mechanics, game theory, and economics. Various software libraries of numerical linear algebra for distributed multi-core cluster computing like ScaLAPACK [18] provide routines of matrix multiplication related computing. In this paper, we propose a high performance communication scheme to achieve performance gain in distributed matrix multiplication, which serves as a stepping stone for fulfilling further demands of improving performance efficiency in other linear algebra operations where matrix multiplication is frequently employed. As shown in Table I, the runtime percentage of matrix multiplication in the execution of the above three classic numerical linear algebra algorithms is significant, where nb denotes the block size for cache efficiency.

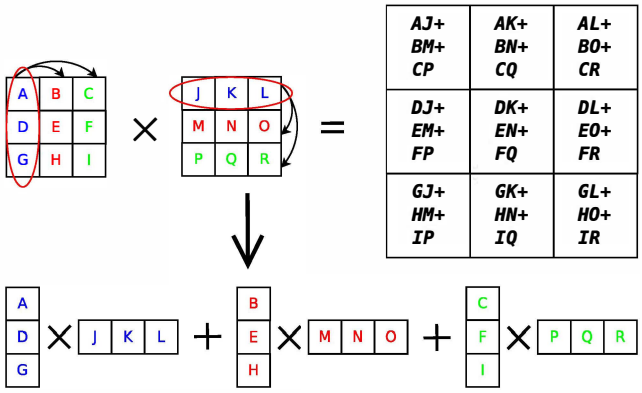

Fig. 1. A Distributed Matrix Multiplication Algorithm with a Global View.

TABLE I. Runtime Percentage of Matrix Multiplication.

Algorithm	Matrix Multiplication Runtime Percentage
Cholesky	92.0%, nb = 128
LU	89.5%, nb = 32
QR	72.4%, nb = 256

The matrix multiplication routines from ScaLAPACK are derived from the DIMMA (Distribution-Independent Matrix Multiplication) algorithm, an advanced version of SUMMA (Scalable Universal Matrix Multiplication Algorithm) [14]. The algorithm skeleton consists of three steps: (a) Distribute matrix elements into the process grid using a 2-D block cyclic distribution for load balancing; (b) broadcast local sub-matrices in a row-/column-wise way as a logical ring; and (c) perform local sub-matrix multiplication. Leveraging an optimized communication scheme, DIMMA outperforms SUMMA by eliminating slack from overlapping computation and communication effectively. We observe that additional performance improvement can be achieved by adopting an optimal communication scheme in accordance with system and application characteristics. We elaborate our idea of a high performance communication scheme in distributed matrix multiplication in the following section.

IV. High Performance Pipeline Broadcast

Due to distributing the global matrix evenly into the process grid for load balancing, we need to broadcast each row to all other rows and broadcast each column to all other columns to perform local matrix multiplication in parallel.

A. Binomial Tree and Pipeline Broadcast

There exist a large body of distributed broadcast algorithms to apply in cluster computing, where binomial tree and pipeline broadcast generally outperform other algorithms for different system configurations. In the original pdgemm() routine from ScaLAPACK atop different MPI implementations, different schemes like ring-based, binomial tree and pipeline broadcast

TABLE II. NOTATION IN BINOMIAL TREE AND PIPELINE BROADCAST.

P	Total number of processes in the communication
S_{msg}	Message size in one broadcast
BD	Network bandwidth in the communication
T_B	Total time consumed by the binomial tree broadcast
T_P	Total time consumed by the pipeline broadcast
T_b	Time consumed by one binomial tree broadcast
T_p	Time consumed by one pipeline broadcast
T_s	Network latency of starting up a communication link
T_d	Time consumed by transmitting messages
n	Number of chunks from dividing a message

are adopted depending on message size and other factors [18]. Table II lists the notation used in this section. Figure 2 (a) depicts how the algorithm works using a 3-round iteration on a 8-process cluster. We can see that in each round, a process sends messages complies with the following pattern:

- In Round 0, process P_0 (sender) sends a message to the subsequent available process P_1 (receiver);
- In Round j ($j > 0$), process P_i ($i \leq j,\ i = 0, 1, 2, \dots$) that is a sender/receiver in the precedent round sends a message to the subsequent available process.

In other words, in Round j, the number of senders/receivers is 2^j and thus the algorithm takes $logP$ rounds for the P^{th} process to receive a message. The communication time complexity can be modeled as:

$$T_B = T_b \times logP \qquad T_b = T_s + \frac{S_{msg}}{BD} \tag{1}$$

By substituting T_b, we obtain the ultimate time complexity formula of the binomial tree broadcast:

$$T_B = (T_s + \frac{S_{msg}}{BD}) \times logP \tag{2}$$

Pipeline broadcast works in a time-sliced fashion so that different processes simultaneously broadcast different message chunks as stages in pipelining, as shown in Figure 2 (b). Assume a message in the pipeline broadcast is divided into n chunks, when the pipeline is not saturated (the worst case), it takes $n+P-1$ steps for the P^{th} process to receive a message. We can model the time complexity of pipeline broadcast as:

$$T_P = T_p \times (n + P - 1) \qquad T_p = T_s + \frac{S_{msg}/n}{BD} \tag{3}$$

Similarly, we obtain the pipeline broadcast final time costs:

$$T_P = (T_s + \frac{S_{msg}/n}{BD}) \times (n + P - 1) \tag{4}$$

From Equations 2 and 4, despite the steps needed to receive a message, we can see that both T_B and T_P are the sum of T_s and T_d. In a cluster connected by an Ethernet/Infiniband switch, T_s is of the order of magnitude of μs, so T_s is negligible when S_{msg} is comparatively large. Therefore, Equations 2 and 4 can be simplied as follows:

$$T_B = \frac{S_{msg}}{BD} \times logP \quad T_P = \frac{S_{msg}}{BD} \times (1 + \frac{P-1}{n}) \tag{5}$$

It is clear that both T_B and T_P scale up as P increases, with fixed message size and fixed number of message chunks. However the pipeline broadcast outperforms the binomial tree broadcast with given P and message size by increasing n, since the ratio of binomial tree broadcast to pipeline broadcast approximates to $logP$ when n is large enough and $\frac{P-1}{n}$ becomes negligible. We experimentally observed the communication schemes in ScaLAPACK pdgemm() routines are not optimal in our experimental platform. Performance gain can be fulfilled by leveraging slack arising from the communication. Therefore, a pipeline broadcast scheme with tuned chunk size according to system characteristics is desirable.

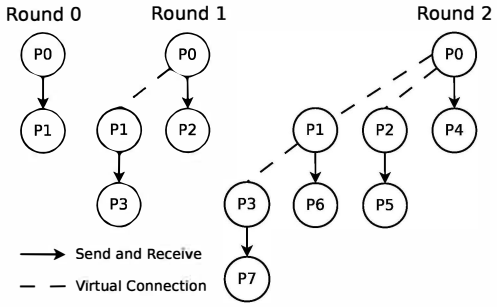

Fig. 2. Binomial Tree Broadcast Algorithm.

B. Two Pipeline Broadcast Schemes

Typically, there exist two types of pipeline broadcast schemes, non-blocking pipeline broadcast and blocking pipeline broadcast. Figures 3 and 4 show how messages are routed in the two broadcast schemes, where P1, P2, ..., P5 denote different processes, m1, m2, ..., m6 denote different messages, and T1, T2, ..., T11 denote different time slots with the same unit. Due to the use of different MPI communication routines, in the blocking pipeline method, each process can either send data or receive data when data is available, while in the non-blocking one, one process can send and receive data simultaneously without waiting for each other.

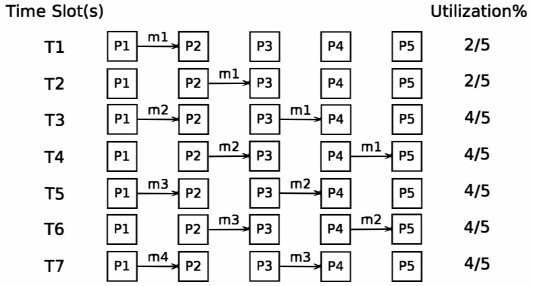

Fig. 3. Blocking Pipeline Broadcast Scheme.

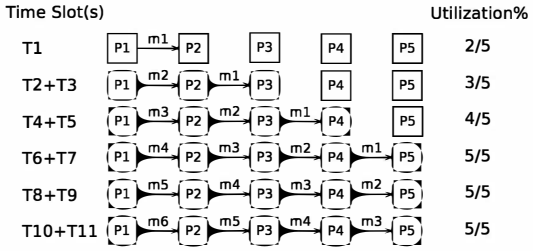

Fig. 4. Non-blocking Pipeline Broadcast Scheme.

Assuming that in the non-blocking scheme, the sending and receiving routines evenly split network bandwidth, we can see in Figures 3 and 4 that it takes twice execution time for P3 to finish receiving m1 from P2, since the sending routine in P2 shares network with the receiving routine. Consequently, in the blocking method, P5 finishes receiving the first message at the end of T4, while P5 in the non-blocking scheme finishes receiving the first message at the end of T7. We can model the relationship between time overhead and the i^{th} received message for the blocking and non-blocking schemes following the constraints of Equations 6, respectively:

$$T^B_{m_i} = 2i + 2 \qquad T^N_{m_i} = 2i + 5 \tag{6}$$

TABLE III. PIPELINE BROADCAST EFFICIENCY COMPARISON

Type of Pipeline Broadcast	Pipeline Is Full?	Max Pipeline Time Latency[1]	Average Utilization[2]
Blocking, Odd PL	No	$2(\lfloor \frac{PL}{2} \rfloor - 1)$	$\sim 1 - \frac{1}{PL}$
Blocking, Even PL	Yes[3]	$PL - 2$	$\sim 1 - \frac{1}{PL}$
Non-blocking, Odd PL	Yes	$2PL - 5$	$\sim 100\%$
Non-blocking, Even PL	Yes	$2PL - 5$	$\sim 100\%$

We can see that although receiving a message in the non-blocking scheme requires three more time units than that in the blocking scheme, both pipeline broadcast schemes have the same time complexity $O(i)$ for receiving a message.

C. Slack Analysis

Slack in pipeline broadcast of distributed matrix multiplication mostly arises from communication latency due to non-fully exploited network bandwidth. Figures 3 and 4 show a blocking and a non-blocking pipeline broadcast with an odd Pipeline Length (PL), respectively. Blocking and non-blocking pipelining with an even pipeline length are not presented due to similarity. We consider three factors to analyze possible slack in pipelining: (a) Saturated pipeline or not, (b) time latency to reach maximal pipeline, and (c) average network utilization in pipelining. In summary, we compare pipeline efficiency on the three factors of four types of pipeline broadcasts in Table III, where the following inferences are concluded:

- The pipe of the blocking pipeline broadcast is never saturated when the pipeline length is odd;
- With limited number of computing nodes and a great number of messages to send and receive, the average network utilization of the non-blocking scheme is higher than that of the blocking scheme;
- Time latency for reaching maximal pipeline (if not saturated) of the non-blocking method is smaller than that of the blocking method, when $PL < 3$, while is greater than that of the blocking method, when $PL \geq 3$. The boundary value of PL for maximal pipeline can be relaxed to greater values if only reaching the same network utilization is required.

Although on receiving messages, both schemes have similar time overhead and the same time complexity, according to the above inferences, the non-blocking scheme exploits network bandwidth more thoroughly than the blocking scheme in two aspects: Saturated pipeline or not and average network utilization. As for maximal pipeline time latency, the blocking scheme may be superior to the non-blocking scheme as PL is generally of the order of magnitude of tens or greater. Nevertheless, the gap for reaching maximal pipeline between two schemes shrinks if only reaching the same network utilization is considered, since pipeline in the blocking scheme can never be always saturated[3]. Therefore, the non-blocking pipeline broadcast is adopted for exploiting communication slack to improve performance of distributed matrix multiplication.

[1]The numbers are relative values instead of absolute values.

[2]We assume that the number of messages is considerably large, and the processes involved in communication have 100% network utilization individually while those not involved in communication have 0% network utilization individually. For simplicity, approximate arithmetic average values are calculated.

[3]Pipeline is not always saturated with alternate pipeline capacities between $1 - \frac{2}{PL}$ and 100% after saturated pipeline has been reached.

V. IMPLEMENTATION AND EVALUATION

We have implemented the high performance pipeline broadcast with tuned chunk size for improving performance efficiency of distributed matrix multiplication. Our implementation was accomplished by rewriting the pdgemm() routine based on the algorithm provided by ScaLAPACK [18], one widely used scalable numerical linear algebra libraries for distributed-memory clusters nowadays. In our implementation of pdgemm(), instead of using binomial tree broadcast for communication, we take advantage of tuning chunk size of pipeline broadcast to fully exploit possible slack during communication. For achieving optimal performance of computation, we employ the dgemm() routine provided by ATLAS [19], a linear algebra software library that automatically tunes performance according to configuration of the hardware. The rewritten pdgemm() has the same interface and is able to produce the same results as the pdgemm() of ScaLAPACK.

A. Experimental Setup

We applied the non-blocking pipeline broadcast to five matrix multiplications with different global matrix sizes to assess performance gain achieved from our approach. Experiments were performed on a computing cluster (HPCL) with an Ethernet switch consisting of 8 computing nodes with two Quad-core 2.5 GHz AMD Opteron 2380 processors (totalling 64 cores) and 8 GB RAM running 64-bit Linux kernel 2.6.32. In our experiments, time was measured using the MPI_Wtime() routine, and energy consumption was measured using PowerPack [20], a comprehensive software and hardware framework for energy profiling and analysis of high performance systems and applications. PowerPack was deployed and running at a meter node to collect energy costs on all involved hareware components on all 8 computing nodes of HPCL.

B. Performance Gain via Pipeline Broadcast

The optimal performance and energy costs can be achieved by applying the non-blocking pipeline broadcast with tuned chunk size in distributed matrix multiplication. We evaluated performance and energy efficiency of our high performance pipeline broadcast by contrasting our implementation of pdgemm() against pdgemm() with binomial tree broadcast and pdgemm() with the blocking pipeline broadcast on the HPCL cluster, respectively, as shown in Tables IV and V. The default block size 32 as in the pdgemm() routine of ScaLAPACK was adopted in our experiments.

Compared to the binomial tree pipelining, the non-blocking pipeline broadcast exploits network resources more completely via pipelined fine-grained message communication as illustrated in section 4A, where chunk size of sending and receiving messages is highly tuned. Specifically, performance gain can be maximized by increasing the number of message chunks from partitioning messages to send and receive into smaller pieces, so that the advantage of pipelining is fully utilized with the fine-grained communication. As Table IV shows (speed-ups are calculated by comparing time in the 4^{th} column with time in the 2^{nd} and the 3^{rd} column individually), execution time of pdgemm() with the non-blocking pipelining is 4.8% on average and up to 6.5% less than that with the binomial tree scheme. On the other hand, compared to the blocking pipeline broadcast, the non-blocking pipeline broadcast exploits network bandwidth more throughly due to the fully leveraged pipeline and higher average network utilization in pipelining

TABLE IV. NON-BLOCKING PIPELINE BROADCAST SPEED-UPS OVER BLOCKING PIPELINE AND BINOMIAL TREE BROADCAST (UNIT: SECOND)

Global Matrix Size	Execution Time with Binomial Tree Pipelining	Execution Time with Blocking Pipelining	Execution Time with Non-blocking Pipelining	Performance Gain in % 4^{th} vs 2^{nd}, 4^{th} vs 3^{rd}
7680	11.657	11.931	11.130	4.5%, 6.7%
10240	20.956	21.843	20.104	4.1%, 8.0%
12800	34.579	35.666	32.654	5.6%, 8.4%
15360	58.313	57.132	54.525	6.5%, 4.6%
17920	80.874	81.929	78.161	3.4%, 4.6%

TABLE V. NON-BLOCKING PIPELINE BROADCAST ENERGY SAVINGS OVER BLOCKING PIPELINE AND BINOMIAL TREE BROADCAST (UNIT: JOULE)

Global Matrix Size	Energy Costs with Binomial Tree Pipelining	Energy Costs with Blocking Pipelining	Energy Costs with Non-blocking Pipelining	Energy Savings in % 4^{th} vs 2^{nd}, 4^{th} vs 3^{rd}
7680	16486.7	16842.6	15692.4	4.8%, 6.8%
10240	31164.8	31908.9	29699.1	4.7%, 6.9%
12800	51896.5	52372.6	49269.2	5.1%, 5.9%
15360	88676.4	86147.8	83259.9	6.1%, 3.4%
17920	123986.3	124839.4	120006.1	3.2%, 3.9%

as discussed in section 4B. As shown in Table IV, the non-blocking scheme achieves 6.5% on average and up to 8.4% performance gain compared to the blocking scheme. As shown in the last column of Table IV, performance gain drops as matrix size increases. This is because when matrices expand in size, computation time increases faster than communication time, and thus performance gain from the communication can be covered up by performance loss from the computation. The trend does not manifest when matrix size is less than 15000.

C. Energy Savings via Pipeline Broadcast

In general, applications with less execution time consumes less energy. Table V presents energy consumption of the five matrix multiplications to show the effectiveness of the proposed approach on energy savings. As shown in Table V (energy savings are calculated by comparing energy costs in the 4^{th} column with energy costs in the 2^{nd} and the 3^{rd} column individually), non-blocking pipeline broadcast with tuned chunk size is evaluated to achieve 4.8% on average and up to 6.1% energy savings compared to binomial tree broadcast, and achieve 5.4% on average and up to 6.9% energy savings compared to blocking pipeline broadcast. Contrasting Tables IV and V, it is noteworthy that although energy consumption decreases monotonically as execution time drops, the variation of energy savings is not as much as that of performance gain.

VI. CONCLUSIONS

Exploiting parallelism in high performacne applications running on distributed-memory computing systems for performance and energy efficiency has been a challenging issue. Among effective software and hardware based solutions in the literature, exploiting slack in communication has been leveraged to provide substantial performance gain and thus energy savings. This paper proposes a non-blocking pipeline broadcast scheme with tuned chunk size for high performance communication of distributed matrix multiplication where network bandwidth is fully exploited. The experimental results on a 64-core cluster indicate the effectiveness of the proposed approach on saving time and energy compared to blocking pipeline broadcast and binomial tree broadcast.

ACKNOWLEDGMENTS

The authors would like to thank the HPCL Lab at the Marquette University for providing the distributed-memory computing cluster with PowerPack. This research is partly supported by US National Science Foundation, under the grants #CNS-1118043, #CNS-1116691, and #CNS-1304969.

REFERENCES

[1] J. Watts and R. van de Geijn, "A pipelined broadcast for multidemensional meshes," *Parallel Processing Letters*, 1995.

[2] G. Chen, K. Malkowski, M. Kandemir, and P. Raghavan, "Reducing power with performance constraints for parallel sparse applications," in *Proc. IPDPS*, 2005, pp. 1–8.

[3] E. Chan, R. van de Geijn, W. Gropp, and R. Thakur, "Collective communication on architectures that support simultaneous communication over multiple links," in *Proc. PPoPP*, 2006, pp. 2–11.

[4] E. Solomonik, A. Bhatele, and J. Demmel, "Improving communication performance in dense linear algebra via topology aware collectives," in *Proc. SC*, 2011, p. 77.

[5] F. Desprez, B. Tourancheau, and J. J. Dongarra, "Performance complexity of LU factorization with efficient pipelining and overlap on a multiprocessor," *Parallel Processing Letters*, 1994.

[6] A. Karwande, X. Yuan, and D. K. Lowenthal, "CC–MPI: a compiled communication capable MPI prototype for ethernet switched clusters," in *Proc. PPoPP*, 2003, pp. 95–106.

[7] A. Faraj and X. Yuan, "Automatic generation and tuning of MPI collective communication routines," in *Proc. ICS*, 2005, pp. 393–402.

[8] S. Hunold and T. Rauber, "Automatic tuning of PDGEMM towards optimal performance," in *Proc. Euro-Par*, 2005, pp. 837–846.

[9] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz, "Communication-optimal parallel algorithm for strassen's matrix multiplication," in *Proc. SPAA*, 2012, pp. 193–204.

[10] G. Ballard, A. Buluç, J. Demmel, L. Grigori, B. Lipshitz, O. Schwartz, and S. Toledo, "Communication optimal parallel multiplication of sparse random matrices," in *Proc. SPAA*, 2013, pp. 222–231.

[11] G. Ballard, J. Demmel, B. Lipshitz, O. Schwartz, and S. Toledo, "Communication efficient gaussian elimination with partial pivoting using a shape morphing data layout," in *Proc. SPAA*, 2013, pp. 232–240.

[12] I. A. Lee, C. E. Leiserson, T. B. Schardl, J. Sukha, and Z. Zhang, "On-the-fly pipeline parallelism," in *Proc. SPAA*, 2013, pp. 140–151.

[13] J. Choi, J. J. Dongarra, L. S. Ostrouchov, A. P. Petitet, D. W. Walker, and R. C. Whaley, "The design and implementation of the ScaLAPACK LU, QR and Cholesky factorization routines," *Scientific Programming*, vol. 5, no. 3, pp. 173–184, Aug. 1996.

[14] J. Choi, "A new parallel matrix multiplication algorithm on distributed-memory concurrent computers," in *Proc. HPC-Asia*, 1997, pp. 224–229.

[15] *MPICH*. http://www.mpich.org/.

[16] *Open MPI*. http://www.open-mpi.org/.

[17] D. M. Wadsworth and Z. Chen, "Performance of MPI broadcast algorithms," in *Proc. IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing, with IPDPS*, 2008.

[18] *ScaLAPACK - Scalable Linear Algebra PACKage*. http://www.netlib.org/scalapack/.

[19] *Automatically Tuned Linear Algebra Software (ATLAS)*. http://math-atlas.sourceforge.net/.

[20] R. Ge et al., "PowerPack: Energy profiling and analysis of high-performance systems and applications," *IEEE Trans. on PDS*, 2010.