Improving communication performance in dense linear algebra via topology aware collectives

Edgar Solomonik
Dept. of Electric Engineering and Computer Science
University of California at Berkeley
Berkeley, CA 94720
solomon@eecs.berkeley.edu

Abhinav Bhatele
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA 94551
bhatele@llnl.gov

James Demmel
Dept. of Electric Engineering and Computer Science
University of California at Berkeley
Berkeley, CA 94720
demmel@eecs.berkeley.edu

ABSTRACT

Recent results have shown that topology aware mapping reduces network contention in communication-intensive kernels on massively parallel machines. We demonstrate that on mesh interconnects, topology aware mapping also allows for the utilization of highly-efficient topology aware collectives. We map novel 2.5D dense linear algebra algorithms to exploit rectangular collectives on cuboid partitions allocated by a Blue Gene/P supercomputer. Our mappings allow the algorithms to exploit optimized line multicasts and reductions. Commonly used 2D algorithms cannot be mapped in this fashion. On 16,384 nodes (65,536 cores) of Blue Gene/P, 2.5D algorithms that exploit rectangular collectives are significantly faster than 2D matrix multiplication (MM) and LU factorization, up to 8.7x and 2.1x, respectively. These speed-ups are due to communication reduction (up to 95.6% for 2.5D MM with respect to 2D MM). We also derive LogP-based novel performance models for rectangular broadcasts and reductions. Using those, we model the performance of matrix multiplication and LU factorization on a hypothetical exascale architecture.

Categories and Subject Descriptors

G.1.0 [**Numerical Analysis**]: General—Parallel algorithms

General Terms

Algorithms, Performance

Keywords

communication, mapping, interconnect topology, performance, exascale

1. INTRODUCTION

SC '11, November 12-18, 2011, Seattle, Washington USA

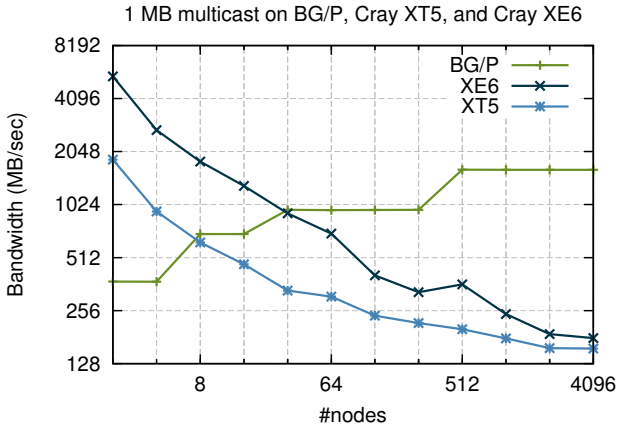


Figure 1: BG/P uses topology aware multicasts to utilize a high fraction of node bandwidth.

A multicast is a broadcast of data from one processor to a selected subset of all other processors. This operation is used in many algorithms, especially in dense linear algebra. Figure 1 shows that the performance of a multicast varies dramatically depending on whether it exploits the topology of the underlying interconnect network. The bandwidth of a 1 MB multicast *drops* by a factor of as much as 30x as the number of processors grows from 2 to 4096 on a Cray XE6 (Hopper), but *grows* by a factor of 4.3x on the Intrepid Blue Gene/P (BG/P). On BG/P, for processor counts that are powers of two, the communication network connects the processors with a 3D torus (i.e. cuboid). Cuboid partitions enable BG/P's Deep Computing Messaging Framework (DCMF) [15] to use specialized 'rectangular' algorithms. Rectangular algorithms saturate all the bandwidth available in all the network links simultaneously. In contrast, the Cray XT5 (Jaguar) and XE6 use non-cuboid partitions and a less efficient (binomial) multicast algorithm that leads to more contention. If we use a binomial multicast protocol on BG/P, a similar degradation in bandwidth is observed as on XT5 and XE6.

This difference in bandwidth raises several questions:

1. Can we map higher level algorithms to cuboids in order to take advantage of the better performance of rectangular multicasts and other collective operations?

2. Can we develop a model to accurately predict the performance of various collective operations and of algorithms that use them?

3. What do our predictions tell us about the performance of these algorithms on future exascale architectures? How important will it be to use topology aware collectives and algorithms on these machines?

To answer the first question, we will use the recently introduced class of 2.5D dense linear algebra algorithms [18] for matrix multiplication (MM) and LU decomposition. The conventional algorithms for these problems [7] store one copy of the input matrix (or matrices) spread over a virtual 2D mesh of $\sqrt{P} \times \sqrt{P}$ processors. These algorithms attain known lower bounds on the total communication (number of words moved between processors [4]). Each processor performs $O(n^3/P)$ arithmetic operations and sends or receives $O(n^2/\sqrt{P})$ words from other processors. 2.5D algorithms similarly perform $O(n^3/P)$ arithmetic operations per processor. 2.5D algorithms store $c > 1$ copies of the input matrix on a virtual 3D mesh of $\sqrt{P/c} \times \sqrt{P/c} \times c$ processors. Redundant memory usage allows 2.5D algorithms to reduce the total communication by a factor of $\sqrt{c}$ compared to 2D algorithms. 2.5D algorithms attain a correspondingly smaller lower bound on communication (this lower bound is a decreasing function of the amount of memory used per processor).

Further, we see that 2.5D algorithms naturally map to the cuboid processor meshes on BG/P, unlike the conventional 2D algorithms. This means 2.5D algorithms not only communicate $\sqrt{c}$ times less data than 2D algorithms, they can communicate this data much faster. As a result, 2.5D algorithms solve small problems more efficiently (achieve strong scaling), since these are more communication bound than large problems. On 65,536 cores, for a matrix of dimension only $n = 2^{12}$, our 2.5D MM algorithm still maintains 24% efficiency (8.7x faster than 2D MM).

To answer the second question, we use the LogP model [8] to model both multicasts and reductions. We derive the first performance model of rectangular multicasts on cuboid networks. By taking into account topology of the algorithm and architecture, we are able to produce a detailed and reliable model. We also provide a model for the more generally applicable but slower binomial tree method. We compare our performance predictions with DCMF measurements and attain good agreement.

To answer the third question, we model the performance of 2D and 2.5D MM and LU on a model of a future exascale machine. Our models for communication collectives allow us to predict the performance of communication within these dense linear algebra algorithms. Our model predicts modest efficiency improvements for MM (from 80% for 2D MM with a binomial multicast, to 87% for 2.5D MM with binomial multicast, to 95% for 2.5D MM with rectangular multicasts). We predict much better efficiency improvements for LU without pivoting (from 20% to 42% to 80%).

The rest of the paper is organized as follows. Section 2 describes how multicasts can take advantage of a mesh interconnect to attain the bandwidth shown in Figure 1. Section 3 briefly describes how 2.5D matrix multiplication and 2.5D LU work (see [18] for details) and presents the performance of our implementation. We analyze the overheads of idle time and communication time and show that 2.5D algorithms with rectangular collectives reduce both. Section 4 presents and validates the multicast and reduction performance models. Section 5 describes the exascale models, and Section 6 discusses future work.

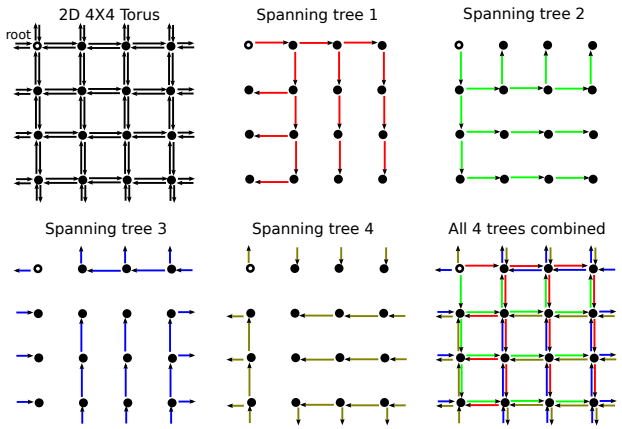


Figure 2: The four disjoint spanning trees used by a rectangular algorithm on a 4x4 processor grid

2. COLLECTIVE COMMUNICATION PRELIMINARIES

Generic multicast and reduction algorithms typically propagate a message down a spanning tree (e.g. binary or binomial) of nodes. To improve bandwidth and network utilization, large messages are pipelined down multiple spanning trees. Each tree spans the full list of nodes but in a different order [5, 13, 16]. However, if any of the tree edges pass through the same physical links, network contention creates a bottleneck in bandwidth utilization.

A topology aware rectangular multicast on a cuboid partition can utilize all the links on the network without any contention. On a mesh of dimension d, rectangular protocols form d edge-disjoint spanning trees (each having a different 'color'). On a torus of dimension d, rectangular protocols form $2d$ colors/spanning trees. For each tree, the root sends unique packets down one of $2d$ combinations of dimension and network direction. On a ring of processors, the two edge-disjoint spanning trees are simply the two directions of the bidirectional network. On a 2D torus of processors, four edge-disjoint spanning trees are formed by routing in different dimensional order ($x \rightarrow y$, $y \rightarrow x$) and in different directions ($-x \rightarrow -y$, $-y \rightarrow -x$). These four trees are displayed in Figure 2. For higher dimensional meshes, rectangular algorithms form d edge-disjoint dimensional orders by performing $d-1$ circular shifts on some initial ordering ($D_1 \rightarrow D_2 \rightarrow \ldots D_d$, $D_2 \rightarrow D_3 \rightarrow \ldots D_d \rightarrow D_1$, etc.).

These topology aware multicasts are valid only if the partition is a cuboid. This condition requires not only that the machine scheduler allocates cuboid partitions but also that algorithms and applications perform multicasts on cuboid partitions. We evaluate the utility of topology aware collectives by designing and implementing algorithms that can exploit these collectives. We also derive new performance models to shed light on the scaling characteristics of these collectives and our new algorithms.

3. COLLECTIVES IN DENSE LINEAR ALGEBRA

Typical dense linear algebra algorithms such as those implemented in ScaLAPACK [7] are two-dimensional (2D). Such algorithms subdivide the matrix into blocks producing a 2D rectangular virtual topology. Most collective communication happens in the form of line multicasts or reductions within this 2D virtual topology or 2D grid. To utilize rectangular algorithms for line collectives, each row and column of the 2D grid must be contiguous on the physical network mapping. Therefore, given a 2D mesh or toroidal partition, 2D algorithms are fit to use any dimensional collectives available. However, some of the largest supercomputers have 3D networks currently. On a 3D grid, a contiguous embedding of a 2D virtual grid requires one of three dimensions of the physical network to be folded entirely into one of the dimensions of the 2D virtual grid. Mapping a rectangular 2D grid of an arbitrary aspect ratio to a 3D cuboid of different aspect ratios in this fashion is usually infeasible. So, 2D algorithms typically cannot utilize rectangular algorithms on 3D networks.

Certain dense linear algebra algorithms do have 3D mappings. A classical 3D algorithm is a matrix multiply on a 3D cube [1, 2, 6]. This algorithm requires a $\sqrt[3]{P} \times \sqrt[3]{P} \times \sqrt[3]{P}$ partition, and the results end up on a different face of the cube than the input started on. The 3D algorithm uses $P^{1/3}$ copies of the matrices, increasing the memory footprint by a factor of $P^{1/3}$. Such tight restrictions on the processor layout and memory availability often makes this 3D algorithm an impractical solution.

Recently, the first 2.5D algorithms were developed for LU and for matrix multiplication [18]. These algorithms perform less communication than their 2D counterparts by replicating the inputs c times and using more memory. They are valid on any $\left(\sqrt{P/c}\right) \times \left(\sqrt{P/c}\right) \times c$ grid, *for any* $c \leq \sqrt[3]{P}$. These virtual topologies map relatively easily to the 3D partitions allocated by Blue Gene/P. 2.5D algorithms can map to a more general set of cuboid partitions by using multiple processes per node. The intra-node processes can be folded into any given physical network dimension. We map 2.5D algorithms to utilize contiguous line multicasts and reductions. In fact, *all communication in 2.5D MM and LU algorithms can be done using topology aware rectangular collectives.*

3.1 2.5D algorithms

2.5D algorithms utilize a 3D partition by replicating the input matrices so that each of c layers holds a copy. Then parts of the computation are done on each layer. Communication and synchronization among layers is needed to combine the results computed on each layer.

3.1.1 2.5D communication lower bounds

2.5D algorithms attempt to use a factor of c more memory than necessary to lower the communication cost. The lower bound on the number of words communicated per processor in a class of linear algebra algorithms that includes MM and LU (proven in [4]) is

$$W = \Omega\left(\frac{n^2}{\sqrt{c \cdot P}}\right) \tag{1}$$

The corresponding lower bound on the number of messages, i.e. latency cost is

$$S = \Omega\left(\sqrt{P/c^3}\right) \tag{2}$$

Communication optimal 2D algorithms satisfy these lower bounds for $c = 1$. 2.5D algorithms generalize 2D algorithms for $c \in \{1, 2, \ldots, \lfloor \sqrt[3]{P} \rfloor\}$. Our 2.5D MM algorithm satisfies all communication lower bounds for any c, while 2.5D LU satisfies only the bandwidth lower bound (Eq. 1). A tight lower bound for the communication latency in 2.5D LU is proven in [18].

Algorithm 1: $[C] = $ 2.5D-SUMMA(A,B,n,P,c)

Input: square $n \times n$ matrices A, B distributed so that Π_{ij0} own $\frac{n}{\sqrt{P/c}} \times \frac{n}{\sqrt{P/c}}$ blocks A_{ij} and B_{ij}

Output: square $n \times n$ matrix $C = A \cdot B$ distributed so that Π_{ij0} own a $\frac{n}{\sqrt{P/c}} \times \frac{n}{\sqrt{P/c}}$ block C_{ij}

forall $i, j \in [0, \sqrt{P/c} - 1], k \in [0, c - 1]$ **do**

/ replicate input matrices */*

  Π_{ij0} do a **line multicast** of A_{ij} and B_{ij} to Π_{ijk}

  for $t = k \cdot \sqrt{P/c^3}$ **to** $(k + 1) \cdot \sqrt{P/c^3}$ **do**

    Π_{itk} do a **line multicast** of A_{it} to Π_{ijk}

    Π_{tjk} do a **line multicast** of B_{tj} to Π_{ijk}

    $C_{ijk} := C_{ijk} + A_{it} \cdot B_{tj}$

  end

  Π_{ijk} sum C_{ijk} with a **line reduction** to Π_{ij0}

end

3.1.2 2.5D matrix multiplication

We present a new version of 2.5D matrix multiplication, that is based on SUMMA [21] rather than Cannon's algorithm (presented in [18]). If the matrices are square, each layer performs n/c rank-1 updates. The results are combined with a sum-reduction among layers. The rank-1 updates are combined into rank-b updates, where $b = \sqrt{c} \cdot n/\sqrt{P}$ for a square matrix. We can generalize this algorithm to non-square matrices A (m-by-k) and B (k-by-n). The size of each update (b) should correspond to the smallest blocking factor among A and B in the dimension corresponding to k.

Algorithm 1 describes the 2.5D SUMMA algorithm for square matrices on a $\left(\sqrt{P/c}\right) \times \left(\sqrt{P/c}\right) \times c$ grid indexed by Π_{ijk}. This algorithm has the same asymptotic costs as the Cannon-based 2.5D MM algorithm presented in [18]. However, this version communicates with multicasts rather than sends, allowing the implementation to exploit topology-aware collectives.

This 2.5D MM algorithm achieves the bandwidth lower bound (Eq. 1) [18] and has a latency cost of

$$S = O\left(\sqrt{P/c^3} + \log(c)\right)$$

This latency cost reaches the lower bound in Eq. 2 modulo the $\log(c)$ term.

3.1.3 2.5D LU factorization

2.5D LU works by accumulating the Schur complement on each of c layers [18]. However, more inter-layer communication and synchronization is required than in the case of 2.5D MM. We will not describe 2.5D LU in full detail. The

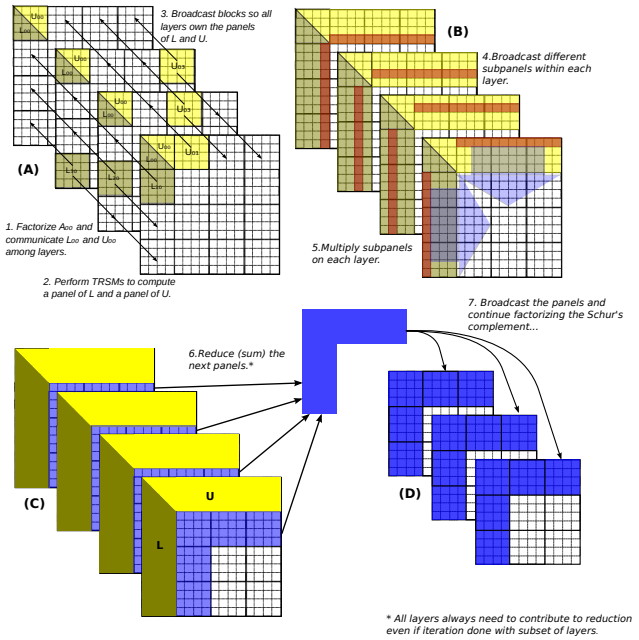


Figure 3: 2.5D LU algorithm work-flow

full algorithm and analysis can be found in [18]. At a high level, the algorithm functions as follows (also see Figure 3)

(A) The top layer factorizes the top left diagonal block, $A_{11} = L_{11}U_{11}$ and performs a **line multicast** of L_{11} and U_{11} to the other layers. The other layers update L_{21} and U_{12}. The intra-layer updates and factorizations have 2D mappings and are able to utilize **line multicasts** for all communication.

(B) Each layer performs different thin panel Schur complement updates on A_{22}.

(C) The Schur complement of the next panel of A_{22} is subtracted using a **line all-reduce**. After the all-reduce, all layers own the correct Schur complement panels.

(D) Steps A-C are repeated until the entire matrix is factorized.

This 2.5D LU algorithm achieves the communication bandwidth lower bound in Eq. 1. The communication is minimized along the critical path, rather than simply the total volume as done by a 3D LU algorithm introduced in [12]. The 2.5D LU algorithm has a latency cost of

$$S = O\left(\sqrt{P \cdot c}\right)$$

This latency cost grows with larger c, contrary to the behavior of 2.5D MM. Due to the nature of the diagonal dependencies in LU, any approach to lowering this latency cost would increase bandwidth cost [18].

3.1.4 2.5D LU with CA-pivoting

2.5D LU can also be done with pivoting as shown in [18]. In order to maintain a theoretically optimal latency cost, communication-avoiding (CA), tournament pivoting [10] is used, rather than partial pivoting. Tournament pivoting in 2.5D LU requires the following extra work:

1. A tournament is done for each column of width $O(n/\sqrt{pc})$. The tournament is done within a block of height $O(n/\sqrt{p/c})$ on each layer, then over the best rows of the layers.

2. For each block-column of width $O(n/\sqrt{pc})$, we pivot and update a block-column of width $O(n/\sqrt{p/c})$, to which the smaller block-column belongs.

3. After each factorization of a block column of width $O(n/\sqrt{p/c})$, the remainder of the matrix is pivoted and updated throughout layers.

For a detailed description and complexity analysis of 2.5D LU with pivoting see [18].

3.1.5 Implementation

We implemented 2.5D MM and LU using OpenMP for shared-memory parallelization and MPI. We also developed a pure DCMF version to make sure the collective protocols MPI was using were optimal for our uses and to remove any potential MPI overhead. However, this optimization did not yield noticeable speed-ups. So, we will detail only MPI performance, which is more reproducible and portable.

We did not attempt exploiting overlap between communication and computation. Our goal was to reduce communication time rather than to hide it. A good production implementation should try to do both. For 2.5D LU, we did try to expose as much parallelism as possible by using a block-cyclic layout and pipelining step A. All layers start performing updates as soon as the corner processor on the first layer factorizes its block.

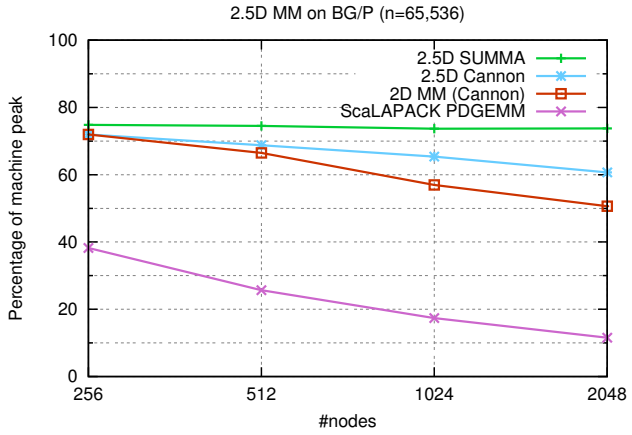
3.2 Performance of 2.5D algorithms

Mapping 2.5D algorithms to utilize BG/P rectangular in each dimension is aided by folding the 4 cores into one of the physical dimensions. For example, a 2048-node partition is a $8 \times 8 \times 32$ cuboid. In order to do our strong scaling study on this partition, we ran 2.5D algorithms with 4 processes per node (vn mode). We folded this nodal dimension into the physical X dimension. This folding generated a virtual square (32×32) XZ layer. The Y dimension was the 'k' dimension (the number of layers, c). We collected strong-scaling results by running on sub-partitions along the Y dimension ($c = 1$ for 256 nodes, to $c = 8$ for 2048 nodes). On 16,384 nodes (65,536 cores), 2.5D algorithms used $c = 16$ layers to map precisely to the $16 \times 32 \times 32$ physical network of nodes. For 2D runs, we used either 1 process per node (smp mode) or 2 processes per node (dual mode) if the number of nodes in the partition is an odd power of two.

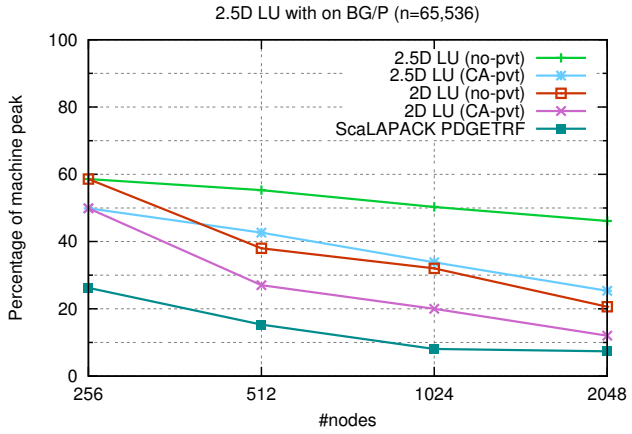
3.2.1 Strong scaling

Figures 4(a) and 4(b) show strong scaling from 256 to 2048 nodes of BG/P for 2.5D matrix multiply and 2.5D LU, respectively. Our new SUMMA-based 2.5D matrix multiplication algorithm achieves virtually perfect strong scaling. The efficiency of the 2D algorithms deteriorates due to the rising relative communication cost. 2D Cannon and SUMMA cannot reap the benefit of rectangular collectives. 2D SUMMA performs worse because binomial collectives perform worse than the point-to-point sends used by Cannon's algorithm.

We also observe improved strong scaling behavior in 2.5D LU 4(b). Our implementation of 2.5D LU generalizes and outperforms the 2D algorithm with CA-pivoting and with no pivoting.

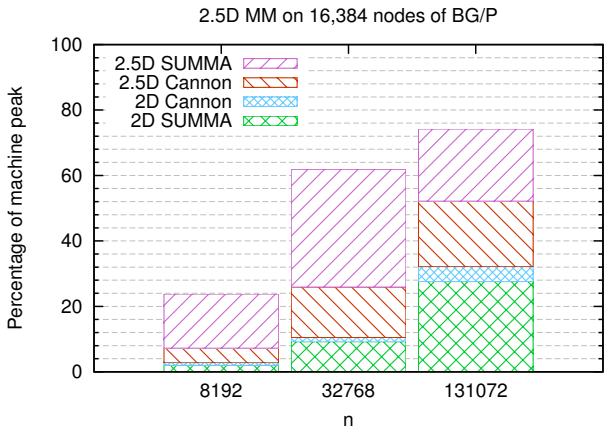


(a) Matrix multiplication

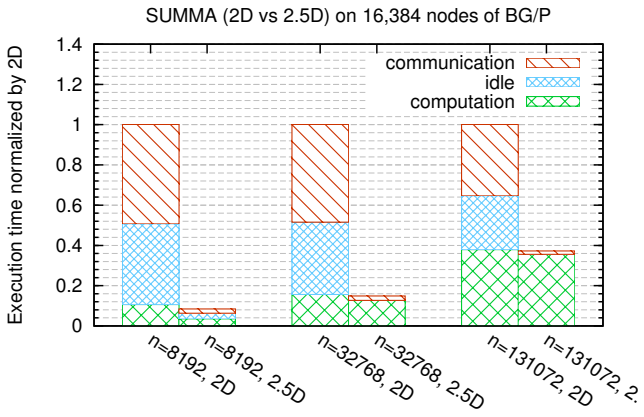


(b) LU factorization

Figure 4: Strong scalability of 2.5D algorithms on BG/P



(a) Efficiency of 2.5D matrix multiplication



(b) Execution time comparison (down is good).

Figure 5: Performance of 2.5D matrix multiplication on 16,384 nodes of BG/P

3.2.2 *Efficiency analysis at scale*

On 16,384 nodes (65,536 cores), the benefit of avoiding communication grows 5(a). 2.5D SUMMA retains a 24% efficiency running on matrix as small as $n = 8192$. This efficiency yields a 8.7X speed-up over the best 2D algorithm (Cannon's algorithm). Note that a speed-up of over 2x is only possible by reducing communication rather than overlapping communication and computation. For the largest problem size tested $n = 131,072$, 2.5D SUMMA runs at 76% efficiency, achieving 0.16 Petaflop/s.

Figure 5(b) compares 2D SUMMA with 2.5D SUMMA normalizing with respect to the performance of 2D SUMMA. We scale the proportion of total time all processors spend in computation, idling, and communicating to this slow-down. We calculate total communication time as the time spent in MPI calls and use barriers to extract synchronization time. 2.5D SUMMA communicates less by performing a minimal amount of steps and utilizing line collectives. During a matrix multiplication of dimension ($n = 131,072$), these techniques reduce the communication overhead by 95.6%.

For $n = 8192$, 2.5D SUMMA also spends less time performing on-node computation. We attribute the reduction in computation time to the increased block-size of the sequential multiplications. For $n = 8192$, the 2D algorithm uses a block size of $n/\sqrt{P} = 64$, while the 2.5D algorithm works with blocks of size $n/\sqrt{P/c} = 256$. The sequential efficiency of DGEMM is significantly higher for the larger block-size.

Figure 6 demonstrates the communication reduction we achieve for LU factorization, with and without pivoting. For the largest problem size tested, $n = 131,072$, 2.5D LU reduces the communication time by 81.9% without pivoting and 85.9% with CA-pivoting (with respect to 2D LU). 2.5D LU also reduces the amount of time the supercomputer spends idle, since the communication time along the critical path of the execution is reduced. These reduction in communication and idle time yield a 2.1x efficiency speed-up for both 2.5D LU versions.

4. MODELING COMMUNICATION COLLECTIVES

Rectangular algorithms can reduce communication time in dense linear algebra operations significantly on current supercomputers. We model the performance of rectangular collectives in order to evaluate their scaling behavior.

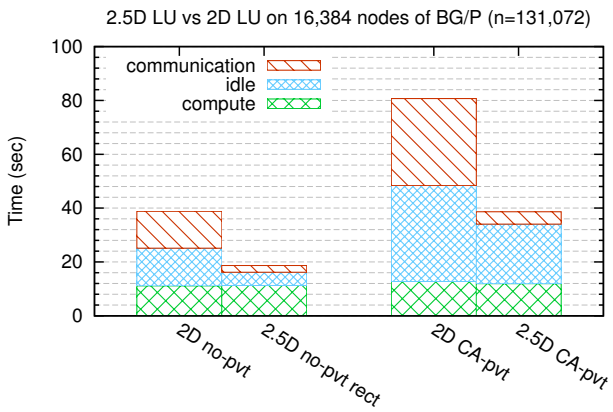


Figure 6: Time to compute a dense LU factorization on 16,384 nodes

4.1 Basic assumptions and terminology

We base our assumptions to reflect current architectures and an idealized messaging implementation.

1. We assume the LogP performance model for messaging.
2. We assume m, the message size, is big and gear our analysis towards understanding bandwidth cost of collectives. Bandwidth cost is more relevant for dense linear algebra. Latency costs are heavily dependent on network architecture and implementation.
3. We restrict our analysis to multicasts and reductions on k-ary d-cube networks.
4. We assume the networks are bidirectional tori for brevity. However, our models can be trivially modified to meshes.
5. We assume that a DMA device and wormhole routing are used for messaging.

We build a model from the following parameters:

- **L** is the physical network latency cost as defined by the LogP model.
- **o** is the overhead of sending and receiving a message as defined by the LogP model.
- **d** is the number of dimensions of the k-ary d-cube network.
- **g** is the reciprocal of the bandwidth achieved by a single message. We assume a single message achieves at best the unidirectional bandwidth of a single link.
- **P** is the number of processors.
- **m** is the message size in bytes.
- $\boldsymbol{\gamma}$ is the flop rate of a node.
- $\boldsymbol{\beta}$ is the memory bandwidth of a node.

4.2 Rectangular multicast model

Rectangular multicasts function by pipelining packets of a message down multiple edge-disjoint spanning trees. Each spanning tree traverses the network in a different dimensional order. As shown in figure 2, all sends are near-neighbor and no contention occurs.

A rendezvous send is the protocol of choice for large message sizes on modern networks. Rendezvous messaging establishes a handshake by sending a small *eager send* to exchange buffer information. Once the handshake is established, the receiver pulls the data with a one-sided *get* operation. The latency cost of an eager send under the LogP model is $2o + L$. A one-sided get requires an extra hop to start the transaction but does not incur overhead on the sender side. Therefore, the latency cost of a get is $o + 2L$. The cost of sending a rendezvous message of size m_r incurs both the eager send and get latency costs,

$$t_r = m_r \cdot g + 3o + 3L \tag{3}$$

A naive version of a rectangular multicast would synchronously send a $m_r = \frac{m}{2d}$ sized message down both directions of each dimension. Such a protocol would achieve single link bandwidth at best. A more aggressive rectangular protocol can overlap the sends in each of the $2d$ directions/dimensions. This rectangular protocol overlaps the network latency (L) and bandwidth (g) costs of each direction. However, the sequential overhead suffered by the sender grows in proportion to the number of trees. Therefore, the start-up cost of a rectangular multicast (the time it takes the root to send off the entire message) is

$$\begin{aligned} t_s &= (m/2d) \cdot g + (2d-1) \cdot o + (3o + 3L) \\ &= (m/2d) \cdot g + 2(d+1) \cdot o + 3L \end{aligned} \tag{4}$$

The multicast does not complete until all nodes receive the entire message. So the multicast finishes when the last packet travels all the way to the farthest node of its spanning tree. The last packet leaves the root at time t_s (Eq. 4). To get to the farthest node of any dimensional tree takes $d \cdot P^{1/d}$ hops. A rendezvous transaction handshake (Eager Send) must be established with the next node at each hop, so the cost per hop is $2o + L$. Over all hops, the overhead of the path of the packet is

$$t_p = d \cdot P^{1/d} \cdot (2o + L) \tag{5}$$

Combining t_s (Eq. 4) and t_p (Eq. 5) gives us an estimate of the time it takes to complete a multicast

$$\begin{aligned} t_{rect} &= t_s + t_p \\ &= \frac{m}{2d} \cdot g + 2(d+1) \cdot o + 3L + d \cdot P^{1/d} \cdot (2o + L) \end{aligned} \tag{6}$$

To review, our model of the cost of a multicast (Eq. 6) is composed of

1. The bandwidth term, $(m/2d) \cdot g$ – the time it takes to send the full message out from the root.
2. The start-up overhead, $2(d+1) \cdot o + 3L$ – the overhead of setting up the multicasts in all dimensions.
3. The per hop overhead, $d \cdot P^{1/d} \cdot (2o + L)$ – the time it takes for a packet to get from the root to the farthest destination node.

4.3 Rectangular reduction model

Reductions behave similarly to multicasts. A multicast tree can be inverted to produce a valid reduction tree. However, at every node of this tree, it is now necessary to apply some operator (do computational work) on the incoming data. We assume that the reduction operator is the same

single flop operation applied to each element of the same index (e.g. sum-reduction). We assume that the elements are double-precision values. The packet size (size of pipelined chunks) must be larger than the value size (size of values on which to apply the reduction operator) to pipeline the reduction.

We adopt the reduction model from the multicast model. However, we need to account for the extra computation and access to memory involved in the application of the reduction operator. The amount of computational work done in each spanning tree node differs depending on whether the node is a leaf or an internal node of the spanning tree. However, summing over all trees, each node receives at most m bytes of data and sends at most m bytes. Therefore, the total amount of computational work on a node is simply the reduction operator applied once on two arrays of size m bytes.

Applying the operator on 2 arrays of m bytes, requires reading $2m$ bytes, writing m bytes, and performing $m/8$ flops (8 bytes per double-precision value). The bandwidth and computational costs are effectively overlapped on modern processors. Given a DMA device, the computation can also be overlapped with the network bandwidth. So, we adjust the start-up time t_s (Eq. 4) to

$$t_{sr} = \max\left(\frac{m}{8\gamma}, \frac{3m}{\beta}, \left(\frac{m}{2d}\cdot g\right)\right) + 2(d+1)\cdot o + 3L \quad (7)$$

The per hop cost t_p (Eq. 5) should be the same for a reduction as a multicast. We build a reduction model by combining t_p and the new start-up cost t_{sr} (Eq. 7),

$$\begin{aligned} t_{red} &= t_{sr} + t_p \\ &= \max\left(\frac{m}{8\gamma}, \frac{3m}{\beta}, \left(\frac{m}{2d}\cdot g\right)\right) + 2(d+1)\cdot o + 3L \\ &\quad + d\cdot P^{1/d}\cdot(2o+L) \end{aligned} \quad (8)$$

4.4 Binomial tree multicast model

Binomial tree multicasts are commonly used as generic algorithms for multicasts and reductions [9, 17, 19]. Binomial collectives models have been written for the LogGP model [3] and for the Hockney model [11] in [19]. Here, we construct a slightly modified binomial tree multicast model under the LogP model. Our model reflects the DCMF binomial multicast implementation on BG/P.

A binomial tree multicast has $\log_2(P)$ stages. In each stage, the multicast root sends the message to a node in a distinct sub-partition. Each sub-partition recursively applies the algorithm on a smaller subtree. A message can be pipelined into these stages to exploit multiple links simultaneously. We assume that there is no network contention in the tree. Modeling contention is non-trivial and out of the scope of this paper.

The binomial tree is unbalanced. The first established sub-tree is the deepest. Therefore, the overhead of the first packet traversing the deepest sub-tree dominates the overhead of the root establishing handshakes with each of the sub-trees. So the cost of a binomial tree multicast is

$$t_{bnm} = \log_2(P)\cdot((m/2d)\cdot g + 2o + L) \quad (9)$$

4.5 Validation of multicast model

Our rectangular collectives models are based purely on architectural parameters. We can validate the quality of our assumptions and analysis by comparing our performance prediction with an actual implementation of rectangular algorithms on a Blue Gene/P machine. We implemented our benchmarks using DCMF [15]. Subtracting the performance of DCMF Get (cost: $o+L = 1.2\,\mu$s) from DCMF Put (cost: $o + 2L = 2.0\,\mu$s) as reported in [15], we get $L = .8\,\mu$s, $o = .4\,\mu$s. The inverted achievable link bandwidth is known to be $g = 1/375$ s/MB.

We validate performance for rectangular algorithms of tori of different dimension. This analysis is important since it justifies the portability of the model among k-ary d-cubes of different dimension d. In particular, algorithms that perform subset multicasts (e.g. line multicasts/reductions in dense linear algebra), operate on lower dimensional partitions. We benchmark collectives on processor partitions of dimension $d \in \{1, 2, 3\}$. Lower dimensional toroidal partitions are generated by extracting sub-partitions of a BG/P 3D torus partition.

Figure 7 details the performance of rectangular and binomial DCMF Broadcast on a ring of 8 BG/P nodes. Our performance model (t_{rect}) matches the performance of the rectangular DCMF algorithms (DCMF rectangle dput) closely.

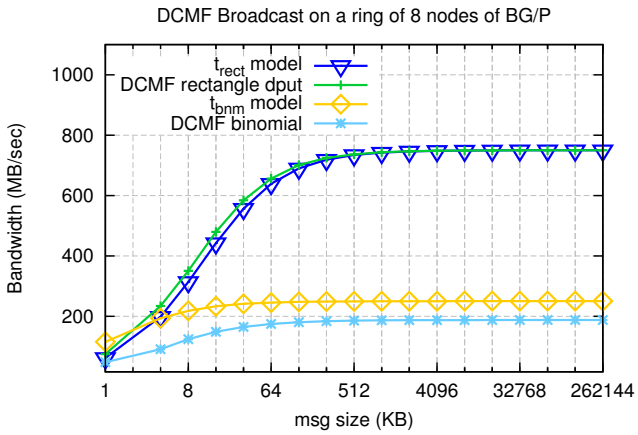


Figure 7: Multicast on a 8-node toroidal ring of processors

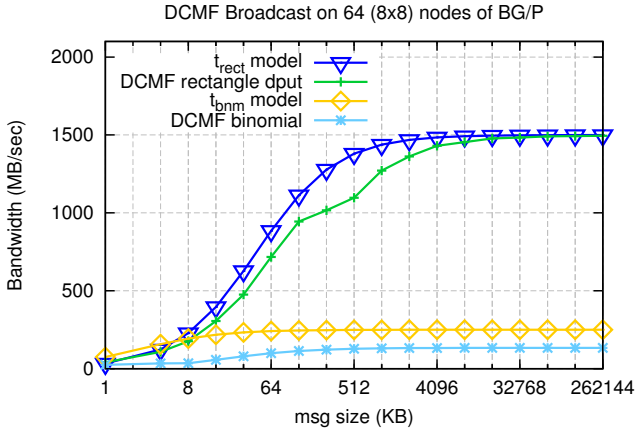


Figure 8: Multicast on a 8-by-8 node toroidal grid of processors

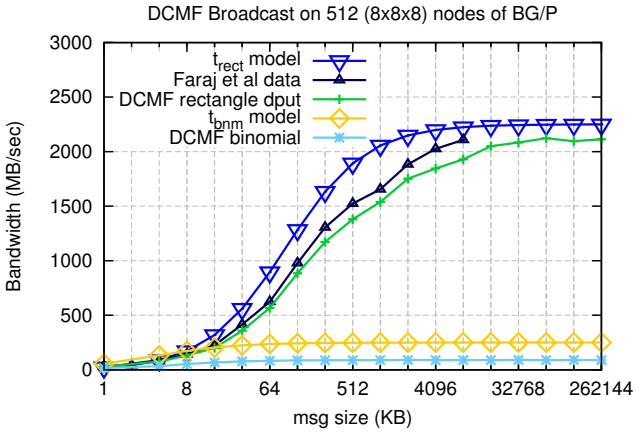


Figure 9: Multicast on a 8-by-8-by-8 node toroidal grid of processors

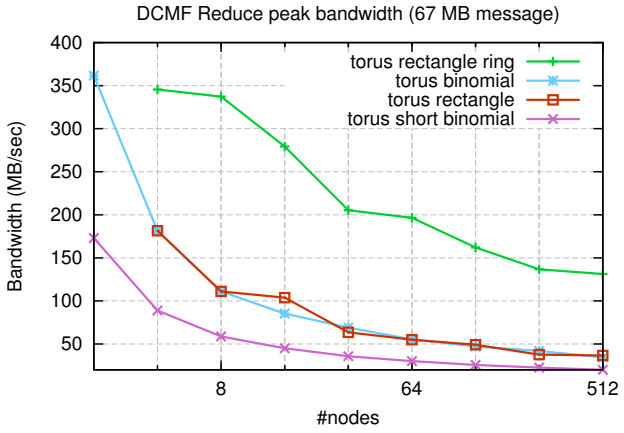


Figure 10: DCMF Reduce performance for different processor counts

Our binomial performance model (t_{bnm}) overestimates the actual achieved bandwidth (DCMF binomial).

The bandwidth of the binomial multicast peaks at exactly half of link bandwidth. On eight nodes, a binomial algorithm must form three trees and has two available links (directions). Our model does not consider contention and assumes the three trees share all available node bandwidth. Since two links are available, the model predicts a peak of two thirds of the link bandwidth. However, two of the trees get mapped to the same physical link of the root, creating contention over the link. As a result, the binomial protocol achieves only half of the link bandwidth.

On a 2D toroidal sub-partition, the available bandwidth as well as the overheads increase. Figure 8 shows that our model (t_{rect}) matches the performance of the rectangular, one-sided DCMF protocol (DCMF rectangle dput) fairly well. The observed data seems to take a dip in performance growth for messages larger than 128 KB. This may be due to the buffer exceeding the size of the L1 cache, which is 64 KB. The binomial model (t_{bnm}) overestimates performance again due to contention.

On a 3D partition (Figure 9), a similar overhead is even more pronounced than on a 2D partition. However, we see that the performance data from Faraj et al. [9] achieves higher bandwidth than the best data we were able to collect (DCMF rectangle dput). Perhaps the difference is due to our usage of DCMF or the use of a lower level interface with less overhead by Faraj et al. [9]. We also found that the performance of multicasts varied substantially depending on the buffer-size of the receive FIFO buffer. We sampled data over a few different buffer-sizes and selected the best performing data-points. Perhaps the data in [9] was collected with more intensive control for such parameters.

Comparisons between the idealized rectangular performance and actual observed data show that implementation overhead grows as a function of dimension. Higher dimensional rectangular algorithms stress the DMA implementation by utilizing multiple links. The hardware and software have to manage communication through multiple links simultaneously.

The binomial multicast performance suffers severely from network contention. It is particularly difficult to build a general contention model since it is dependent on the semantics of spanning tree construction. Rectangular algorithms have *no contention*. This feature improves their scalability with comparison to binomial and other tree-based generic algorithms.

4.6 Analysis of reduction model

A BG/P node has a memory bandwidth of $\beta = 13.6$ GB/s and a flop rate of $\gamma = 13.6$ Gflop/s. If fully overlapped, the computational overhead involved in a reduction on BG/P should hide entirely behind the cost of interprocessor communication. Running in smp mode (one process per node), BG/P should then achieve the same reduction performance as multicast performance.

The observed rectangular reduction performance on BG/P is radically inferior to the multicast performance. Figure 10 shows that the performance of all the DCMF protocols never exceeds the link bandwidth and *deteriorates* on larger partitions. On the other hand, Figure 1 shows that the performance of the rectangular DCMF Broadcast protocol *grows* with partition size.

In order to verify the feasibility of an efficient rectangular reduction on BG/P, we implemented our own 1D rectangular protocols. Our protocols used near neighbor MPI Isend and Irecv operations. Such an implementation incurs a very heavy latency overhead since a new handshake and buffer allocation needs to happen for every pass of a packet. We found that we had to use a packet size of 256 KB in order to be able to sufficiently offset this overhead (DCMF operates with 256 byte packets).

We pipelined the reduction (posted the next Irecv before computing the operator) to overlap computation and communication in the reduction protocol. We used the smp version of ESSL DAXPY to apply the reduction operator in parallel within the node. We enabled interrupts to achieve overlap, which incurred a noticeable overhead. Figure 11 shows the performance of this *first version*.

To avoid the overhead of interrupts, we also implemented a *second version*. This implementation offloads two (one for each direction of the ring) non-master OpenMP threads to apply the operator. Meanwhile, the master thread calls MPI Waitall immediately. We posted MPI Isends from worker

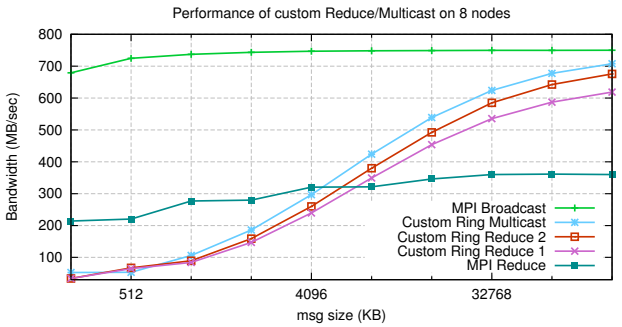


Figure 11: Performance of our implementation of 1D reduce and multicast vs MPI

Total flop rate (γ)	10^{18} flop/s
Total memory	32 PB
Node count (P)	262,144
Node interconnect bandwidth (g)	.06 s/GB
Latency overhead (o)	250 ns
Network latency (L)	250 ns
Topology	3D Torus
Size of dimensions	$64 \times 64 \times 64$

Table 1: Predicted architecture characteristics of an Exaflop/s machine

threads, while the master was in the MPI Waitall. This implementation invoked the single-threaded version of ESSL DAXPY from multiple threads.

Figure 11 compares the performance of these two custom implementations with MPI Broadcast/Reduce (here MPI performs as well as any of the underlying DCMF protocols). We see that our implementations of 1D reduction and multicast (Custom Ring Multicast/Reduce) can achieve a higher bandwidth than the MPI Reduce for large enough message sizes. Our reduction performance (Custom Ring Reduce 2) lags behind the multicast performance only mildly (Custom Ring Multicast). Since we are using a large packet size, the computation done along the path is considerable. The performance of these custom made protocols demonstrates that rectangular reductions can perform nearly or equally as well as multicasts. However, a powerful enough processor and good overlap between communication and computation are necessary conditions.

5. COLLECTIVES AT EXASCALE

The performance of binomial and tree-based collectives deteriorates with increased partition size due to increased tree depth and contention. In particular, contention and branch factor of trees bounds the peak achievable bandwidth. The only cost of rectangular collectives that grows with partition size is the increased depth of spanning trees. To examine the scaling of collectives, we model performance of raw collectives and dense linear algebra algorithms on a potential exascale architecture.

5.1 Exascale architecture

Table 1 details the parameters we use for an exascale machine. These parameters are derived from a report written at a recent exascale workshop [20].

We assume the exascale machine is a 3D torus. Our analysis and conclusions would not change significantly for a torus of slightly higher dimension. However, rectangular collectives are not applicable to a switched network. In particular, the Dragonfly network architecture [14] seems to be a suitable exascale switched network topology.

5.2 Performance of collectives at exascale

We model the performance of collectives on 1D, 2D, and 3D sub-partitions of the exascale machine. Figure 12 details the performance of rectangular and binomial multicasts for these partitions. The peak bandwidth of binomial multicast

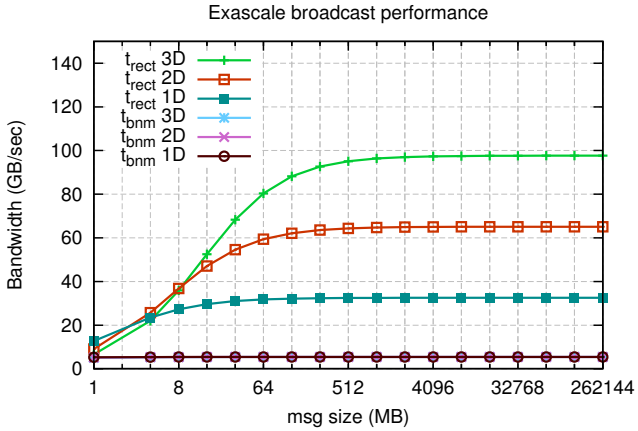


Figure 12: Performance of rectangular and binomial multicasts on a potential exascale architecture

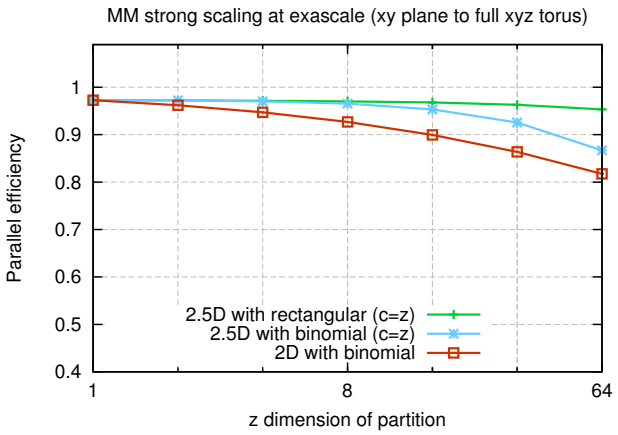
stays the same for each partition since the number of trees increases proportionally to the number of available links. In practice, this peak bandwidth would actually deteriorate on larger partitions due to increased network contention.

Figure 12 also demonstrates that to achieve peak bandwidth, large messages are required (1 GB for a 3D partition). We assume that intra-node concurrency is utilized hierarchically. A single process per node performs all inter-node communication.

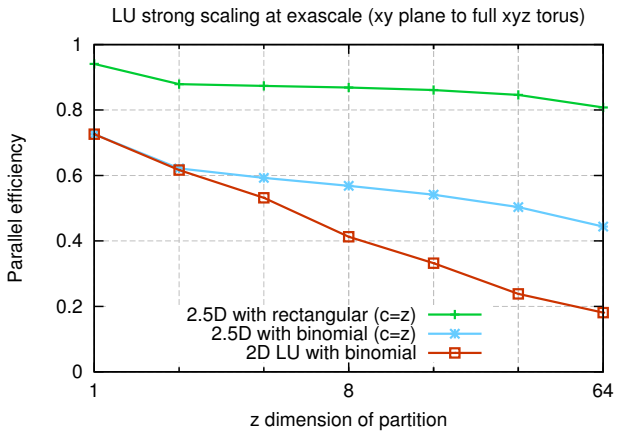
5.3 Performance of MM at exascale

We model performance of matrix multiply with rectangular and binomial collectives at exascale. Our 2D and 2.5D MM algorithms own a single block of each matrix and always communicate in messages that contain the whole block. For the problem we study, this message size should be entirely bandwidth bound for both 1D rectangular and binomial protocols. So we model the multicast and reduction in the 2.5D algorithm using the maximum bandwidth achieved by these protocols in Figure 12. We assume the shifts (sends) in the algorithm achieve the single link peak bandwidth.

Figure 13(a) demonstrates the strong scaling of 2.5D MM with rectangular and binomial collective protocols and 2D MM with a binomial protocol. We scale from a single plane of the 3D torus ($z = 1$) to the entire machine ($z = 64$). We use the largest matrix size that fits in memory on a single



(a) Matrix multiplication



(b) LU factorization

Figure 13: 2.5D MM/LU achieve better efficiency at exascale than their 2D counterparts. The results show strong scaling from full memory on the first plane ($z = 1$) of the machine to the full machine ($z = 64$).

plane ($n = 2^{22}$). We calculate the computational time (t_f) based on the peak flop rate. We express parallel efficiency in terms of t_f and the communication time t_c,

$$efficiency = \frac{t_f}{t_f + t_c}$$

Evidently, matrix multiplication is dominated by computation at this scale. The 2.5D algorithm and rectangular collectives only make a difference when we are using most of the machine. This result is consistent with the performance of MM on BG/P, where significant speed-ups were achieved only for smaller problem sizes.

5.4 Performance of LU at exascale

LU factorization requires more careful modeling, since the block-cyclic layout makes the message sizes variable throughout the algorithm. Further, the 2.5D LU algorithm performs collectives on messages of different sizes and to different partitions depending on the stage of the algorithm. The LU communication costs are itemized in Appendix B of [18]. We scale these communication costs based on the message size and partition size they operate on using Eq. 6 (rectangular collectives) and Eq. 9 (binomial collectives).

Our LU exascale study uses the same problem size and measures the efficiency in the same way as the MM study (previous section). Figure 13(b) details the scaling from a plane to the full machine for a matrix size of $n = 2^{22}$. We model only LU without pivoting. We see that using the 2.5D LU algorithm increases performance significantly even without rectangular collectives. 2.5D LU with rectangular collectives achieves very good parallel scalability with comparison to the 2D algorithm. When using all available nodes ($z = 64$), 2.5D LU with rectangular collectives reduces communication time by 95% and achieves a speed-up of 4.5x over 2D LU.

6. FUTURE WORK

2.5D algorithms and rectangular collectives reduce interprocessor communication and achieve speed-ups for large-scale communication bound problems. However, achieving a good topology-aware mapping for these algorithms puts constraints on the job scheduler. The job scheduler must allocate contiguous and topology-aware partitions. A simpler and more aggressive partition scheduler can achieve higher utilization of the machine. This paper is a case study of how topology-aware partitions can be effectively utilized. In order to complete a comparative study, we plan to study contention more closely on BG/P and Cray systems. A better understanding of contention would also extend our binomial collective model.

There are also promising directions for new 2.5D algorithms and applications of rectangular collectives. We will be developing a more general set of algorithms and a framework that performs topology-aware mapping and replication. 2.5D algorithms should be particularly useful for quantum chemistry applications, which perform tensor contractions that often reduce to distributed matrix multiplications. We plan to attempt to study these applications and to incorporate 2.5D algorithms where possible.

Acknowledgments

The first author was supported by a Krell Department of Energy Computational Science Graduate Fellowship, grant number DE-FG02-97ER25308. This research was supported in part by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). It was also supported by U.S. Department of Energy grants numbered DE-SC0003959, DE-SC0004938, DE-FC02-06-ER25786 and DE-SC0001845. This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. This document was released by Lawrence Livermore National Laboratory for an external audience as LLNL-CONF-491442.

7. REFERENCES

[1] R. C. Agarwal, S. M. Balle, F. G. Gustavson, M. Joshi, and P. Palkar. A three-dimensional approach

to parallel matrix multiplication. *IBM J. Res. Dev.*, 39:575–582, September 1995.

[2] A. Aggarwal, A. K. Chandra, and M. Snir. Communication complexity of PRAMs. *Theoretical Computer Science*, 71(1):3 – 28, 1990.

[3] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. LogGP: incorporating long messages into the LogP modelone step closer towards a realistic model for parallel computation. In *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*, SPAA '95, pages 95–105, New York, NY, USA, 1995. ACM.

[4] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing communication in linear algebra. *To appear in SIAM J. Mat. Anal. Appl*, 2011.

[5] M. Barnett, D. G. Payne, R. A. van de Geijn, and J. Watts. Broadcasting on meshes with worm-hole routing. Technical report, Austin, TX, USA, 1993.

[6] J. Berntsen. Communication efficient matrix multiplication on hypercubes. *Parallel Computing*, 12(3):335 – 342, 1989.

[7] L. S. Blackford, J. Choi, A. Cleary, E. D'Azeuedo, J. Demmel, I. Dhillon, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK user's guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[8] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: towards a realistic model of parallel computation. In *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPOPP '93, pages 1–12, New York, NY, USA, 1993. ACM.

[9] A. Faraj, S. Kumar, B. Smith, A. Mamidala, and J. Gunnels. MPI collective communications on the Blue Gene/P supercomputer: Algorithms and optimizations. In *High Performance Interconnects, 2009. HOTI 2009. 17th IEEE Symposium on*, pages 63 –72, 2009.

[10] L. Grigori, J. W. Demmel, and H. Xiang. Communication avoiding Gaussian Elimination. pages 29:1–29:12, 2008.

[11] R. W. Hockney. The communication challenge for MPP: Intel Paragon and Meiko CS-2. *Parallel Computing*, 20(3):389 – 398, 1994.

[12] D. Irony and S. Toledo. Trading replication for communication in parallel distributed-memory dense solvers. *Parallel Processing Letters*, 71:3–28, 2002.

[13] S. L. Johnsson and C.-T. Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. Comput.*, 38:1249–1268, September 1989.

[14] J. Kim, W. J. Dally, S. Scott, and D. Abts. Technology-driven, highly-scalable dragonfly topology. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 77–88, Washington, DC, USA, 2008. IEEE Computer Society.

[15] S. Kumar, G. Dozsa, G. Almasi, P. Heidelberger, D. Chen, M. E. Giampapa, B. Michael, A. Faraj, J. Parker, J. Ratterman, B. Smith, and C. J. Archer. The deep computing messaging framework: generalized scalable message passing on the Blue Gene/P supercomputer. In *Proceedings of the 22nd annual international conference on Supercomputing*, ICS '08, pages 94–103, New York, NY, USA, 2008. ACM.

[16] P. McKinley, Y.-J. Tsai, and D. Robinson. Collective communication in wormhole-routed massively parallel computers. *Computer*, 28(12):39 –50, Dec. 1995.

[17] J. Pješivac-Grbović, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. J. Dongarra. Performance analysis of MPI collective operations. *Cluster Computing*, 10:127–143, June 2007.

[18] E. Solomonik and J. Demmel. Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms. Technical Report UCB/EECS-2011-10, EECS Department, University of California, Berkeley, Feb 2011.

[19] R. Thakur, R. Rabenseifner, and W. Gropp. Optimization of collective communication operations in MPICH. *International Journal of High Performance Computing Applications*, 19(1):49–66, Spring 2005.

[20] J. Torrellas. Architectures for extreme-scale computing, nov. 2009.

[21] R. A. Van De Geijn and J. Watts. SUMMA: scalable universal matrix multiplication algorithm. *Concurrency: Practice and Experience*, 9(4):255–274, 1997.