Performance Analysis and Optimal Node-aware Communication for Enlarged Conjugate Gradient Methods

SHELBY LOCKHART, University of Illinois at Urbana-Champaign, USA
AMANDA BIENZ, University of New Mexico, USA
WILLIAM GROPP and LUKE OLSON, University of Illinois at Urbana-Champaign, USA

Krylov methods are a key way of solving large sparse linear systems of equations but suffer from poor strong scalability on distributed memory machines. This is due to high synchronization costs from large numbers of collective communication calls alongside a low computational workload. Enlarged Krylov methods address this issue by decreasing the total iterations to convergence, an artifact of splitting the initial residual and resulting in operations on block vectors. In this article, we present a performance study of an enlarged Krylov method, Enlarged Conjugate Gradients (ECG), noting the impact of block vectors on parallel performance at scale. Most notably, we observe the increased overhead of point-to-point communication as a result of denser messages in the sparse matrix-block vector multiplication kernel. Additionally, we present models to analyze expected performance of ECG, as well as motivate design decisions. Most importantly, we introduce a new point-to-point communication approach based on node-aware communication techniques that increases efficiency of the method at scale.

CCS Concepts: • **Mathematics of computing → Mathematical software performance**; **Solvers** • **Computing methodologies → Distributed algorithms**;

Additional Key Words and Phrases: Parallel, communication, node-aware, sparse matrix, collectives

ACM Reference format:
Shelby Lockhart, Amanda Bienz, William Gropp, and Luke Olson. 2023. Performance Analysis and Optimal Node-aware Communication for Enlarged Conjugate Gradient Methods. *ACM Trans. Parallel Comput.* 10, 1, Article 2 (March 2023), 25 pages.
https://doi.org/10.1145/3580003

This material is based in part upon work supported by the Department of Energy, National Nuclear Security Administration, under award numbers DE-NA0003963 and DE-NA0003966. This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications. This material is based in part upon work supported by the Department of Energy, National Nuclear Security Administration, under award number DE-NA0002374.
Authors' addresses: S. Lockhart, W. Gropp, and L. Olson, University of Illinois at Urbana-Champaign, Department of Computer Science, Urbana, Illinois 61801, USA; emails: {sll2, wgropp, lukeo}@illinois.edu; A. Bienz, University of New Mexico, Department of Computer Science, Albuquerque, New Mexico 87131, USA; email: bienz@unm.edu.

2329-4949/2023/03-ART2 $15.00
https://doi.org/10.1145/3580003

1 INTRODUCTION

A significant performance limitation for sparse solvers on large-scale parallel computers is the lack of computational work compared to the communication overhead [29]. The iterative solution to large, sparse linear systems of the form $Ax = b$ often requires *many* sparse matrix-vector multiplications and costly collective communication in the form of inner products; this is the case with the **conjugate gradient (CG)** method and with Krylov methods in general. In this article, we consider so-called *enlarged* Krylov methods [21], wherein block vectors are introduced to improve convergence, thereby reducing the amount of collective communication in exchange for denser point-to-point communication in the sparse matrix-block vector multiplication. We analyze the associated performance expectations and introduce efficient communication methods that render this class of methods more efficient at scale.

There have been a number of suggested algorithms for addressing the imbalance in computation and communication within Krylov methods, including communication avoidance [12, 26], overlapping communication and computation [16], and delaying communication at the cost of performing more computation [15]. Most recently, there has been work on reducing iterations to convergence via increasing the amount of computation per iteration and, ultimately, the amount of data communicated [20, 30, 31]. These approaches have been successful in reducing the number of global synchronization points; the current work is considered complementary in that the goal is reduction of the total *amount* of communication, which is achieved via message passage restructuring utilizing the MPI API.

In addition to reducing synchronization points, enlarged Krylov methods such as **enlarged conjugate gradient (ECG)** reduce the number of sparse matrix-vector multiplications by improving the convergence of the method through an increase in the amount of computation per iteration. This is accomplished by using block vectors, which results in an increase in (local) computational work but also an increase in inter-process communication per iteration. Consequently, the focus of this article is on analyzing the effects of block vectors on the performance of ECG and proposing optimal strategies to address the communication imbalances they introduce.

There are two key contributions made in this article.

(1) A performance study and analysis of an enlarged Krylov method based on ECG, with an emphasis on the communication and computation of block vectors. Specifically, we note how they re-balance the point-to-point and collective communication within a single iteration of ECG, shifting the performance bottleneck to the point-to-point communication.
(2) The development of a new communication technique for blocked data based on node-aware communication techniques that have shown to reduce time spent in communication within the context of sparse matrix-vector multiplication and algebraic multigrid [7, 9]. This new communication technique exhibits speedups as high as 60× for various large-scale test matrices on two different supercomputer systems, as well as reduces the point-to-point communication bottleneck in ECG.

These contributions are presented in Sections 3 and 4, respectively.

2 BACKGROUND

The CG method for solving a symmetric and positive definite system of equations, $Ax = b$, exhibits poor parallel scalability in many situations [12, 19, 26, 28]. In particular, the *strong* scalability is limited due to the high volume of collective communication relative to the low computational requirements of the method. The ECG method has a lower volume of collective communication and higher computational requirements per iteration compared to CG, thus exhibiting better strong

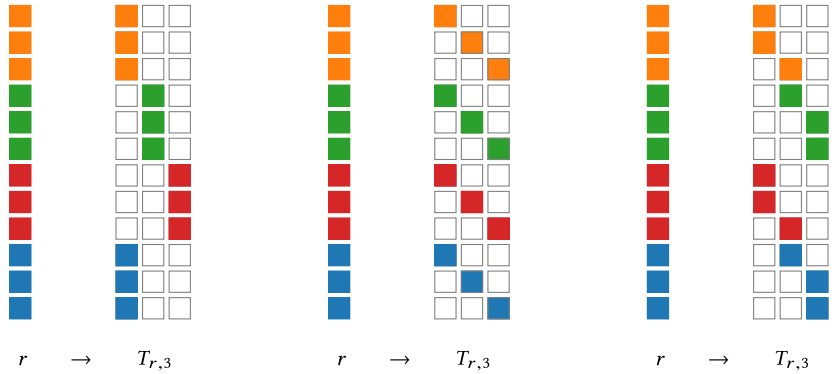

Figure 2.1. Three examples of $T_{r,t}$, with $t = 3$. In each case, r is a vector of length 12, decomposed into 12×3 block vector. The colors represent a case of four processors: orange, green, red, and blue.

scalability. In this section, we detail the basic structure of ECG, briefly outlining the method in terms of mathematical operations and highlighting the key differences from standard CG. A key computation kernel in both Krylov and enlarged Krylov methods is that of a sparse matrix-vector multiplication; we discuss node-aware communication techniques for this operation in Section 2.2.

Throughout this section and the remainder of the article, ECG performance is analyzed with respect to the problem described in Example 2.1.

Example 2.1. In this example, we consider a discontinuous Galerkin finite element discretization of the Laplace equation, $-\Delta u = 1$ on a unit square, with homogeneous Dirichlet boundary conditions. The problem is generated using MFEM [4] and the resulting sparse matrix consists of 1,310,720 rows and 104,529,920 nonzero entries. Graph partitioning is not used to reorder the entries, unless stated.

2.1 Enlarged Krylov Subspace Methods

Similarly to CG, ECG begins with an initial guess x_0 and seeks an update as a solution to the problem $Ax = b$, with the initial residual given by $r_0 = b - Ax_0$. Unlike CG, which considers updates of the form $x_k \in x_0 + \mathcal{K}_k$, where $\mathcal{K}_k$ is the Krylov subspace defined as

$$\mathcal{K}_k = \text{span}\{r_0, Ar_0, A^2 r_0, \ldots, A^{k-1} r_0\}, \tag{2.1}$$

ECG targets $x_k \in x_0 + \mathcal{K}_{k,t}$, where $\mathcal{K}_{k,t}$ is the *enlarged* Krylov space defined as

$$\mathcal{K}_{k,t} = \text{span}\{T_{r_0,t}, AT_{r_0,t}, A^2 T_{r_0,t}, \ldots, A^{k-1} T_{r_0,t}\}, \tag{2.2}$$

with $T_{r,t}$ representing a projection of the residual r (defined next). Notably, the enlarged Krylov subspace contains the traditional Krylov subspace: $\mathcal{K}_k \subset \mathcal{K}_{k,t}$ [20].

In Equation (2.2), $T_{r,t}$ defines a projection of the residual r (normally the initial residual r_0) from $\mathbb{R}^n \to \mathbb{R}^{n \times t}$ by splitting r across t subdomains. The projection may be defined in a number of ways, with the caveat that the resulting columns of $T_{r,t}$ are linearly independent and preserve the row-sum,

$$r = \sum_{i=1}^{t} (T_{r,t})_i, \tag{2.3}$$

where we denote the ith column of T as $(T)_i$. An illustration of multiple permissible splittings is shown in Figure 2.1.

Increasing the number of subdomains, t, increases computation from single vector updates to block vector updates of size $n \times t$. Additional uses of block vectors within ECG are outlined in detail in Algorithm 1. On Line 5, a (small) linear system is solved to generate the t search directions. In addition, the **sparse matrix-block vector (SpMBV)** product AP_k is performed at each iteration. The number of iterations to convergence is generally reduced from that required by CG, but the algorithm does not eliminate the communication overhead when the algorithm is performed at scale. Unlike CG, where the performance bottleneck is caused by the load imbalance incurred from each inner product in the iteration, ECG sees communication overhead at scale due to the communication associated with the SpMBV kernel (see Figure 3.3 in Section 3). We introduce a new communication method in Section 4 to improve this performance.

ALGORITHM 1: Enlarged Conjugate Gradient

```
1  r := b − Ax
2  P := 0, R := T_{r,t}, Z := R
3  while not converged
4      P_old := P
5      P := Z(Z^T AZ)^{−1/2}
6      c := P^T R
7      X := X + Pc
8      R := R − APc
9      if ‖ Σ_{i=1}^{t} (R)_i ‖ < tolerance
10         break
11     d := (AP)^T (AP)
12     d_old := (AP_old)^T (AP)
13     Z := AP − Pd − P_old d_old
14 x := Σ_{i=1}^{t} (X)_i
```

Additionally, Algorithm 1 can easily be updated to include preconditioning, represented by application of a preconditioning matrix M^{-1}. Here we summarize the necessary changes to Algorithm 1; for a full discussion of preconditioned ECG, see Reference [21]. Initialization of Z is updated to include application of the preconditioner to the split residual vector, $Z = M^{-1}R$, and a single step is added to each iteration in which the preconditioner is applied to the block of vectors AP before Line 11, $M^{-1}AP$. Consequently, Line 11 to 13 are then updated to include computations involving $M^{-1}AP$ as shown in Algorithm 2.

ALGORITHM 2: Preconditioning Updates to Enlarged Conjugate Gradient Iteration

```
11 d := (AP)^T (M^{−1}AP)
12 d_old := (AP_old)^T (M^{−1}AP)
13 Z := M^{−1}AP − Pd − P_old d_old
```

As noted by the authors in Reference [21], any preconditioner that can be applied to CG can be applied to ECG, but due to the block structure of the algorithm and the reduction in iterations to convergence, it may not be efficient to use preconditioners that introduce additional communication when performed in parallel, though they may be more robust. Hence, discussion of preconditioner choice and performance is excluded from this work as it is often problem specific, and the focus in this work is on the characterization and optimization of the performance of the ECG algorithm itself.

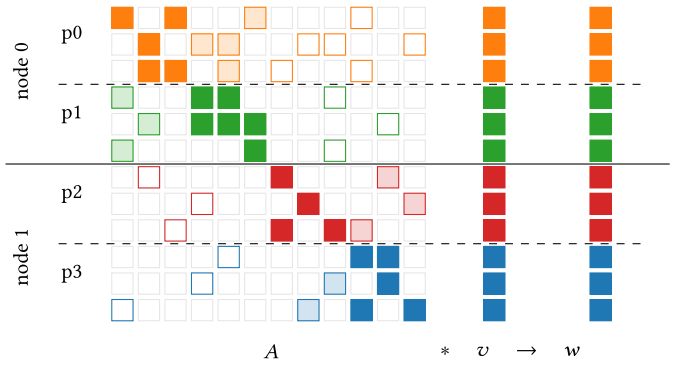

Figure 2.2. Communication and partitioning of a SpMV, $A \cdot v \rightarrow w$. With $n = 12$, matrix A and vectors v and w are partitioned across two nodes and four processors (p0, p1, p2, and p3), indicated by orange, green, red, and blue. A solid block, ■, represents the portion of the SpMV requiring only on-process values from v. A shaded block, ▩, represents the portion of the SpMV requiring on-node but off-process communication of values from v, and the outlined blocks, □, require values of v from processors off-node.

2.2 Node-aware Communication Techniques

Sparse matrix-vector (SpMV) multiplication, defined as

$$A \cdot v \rightarrow w, \tag{2.4}$$

with $A \in \mathbb{R}^{m \times n}$ and $v, w \in \mathbb{R}^n$, is a common kernel in sparse iterative methods. It is known to lack strong scalability in a distributed memory parallel environment, a problem stemming from low computational requirements and the communication overhead associated with applying standard communication techniques to sparse matrix operations.

There have been a number of techniques designed to reduce the communication costs of the parallel SpMV. Graph partitioning algorithms produce efficient data layouts that often lead to improved parallel partitions and system loads that reduce time spent in communication [13, 14, 24, 34]. Additionally, topology-aware task mapping addresses communication overhead via accurately mapping parallel partitions to supercomputer nodes [1, 27, 33]. While these methods can result in reduced communication times, they are often accompanied by costly set-up times or more complex matrix distributions. In the case of node-aware communication techniques, A, v, and w are generally considered to be partitioned row-wise across p processes with contiguous rows stored on each process (see Figure 2.2). In addition, the rows of A on each process are split into 2 blocks, namely on-process and off-process. The on-process block is the diagonal block of columns corresponding to the on-process portion of rows in v and w, and the off-process block contains the matrix A's nonzero values correlating to non-local rows of v and w stored off-process. This splitting is common practice, as it differentiates between the portions of a SpMV that require communication with other processes.

A common approach to a SpMV is to compute the local portion of the SpMV with the on-process block of A while messages are exchanged to obtain the off-process portions of v necessary for the local update of w. While this allows for the overlap of some communication and computation, it requires the exchange of many point-to-point messages, which still creates a large communication overhead (see Figure 2.3). MPI + X, or hierarchical parallelism, strategies can help mitigate this communication overhead by pairing a single MPI rank with multiple threads for local computation. These strategies allow for efficient overlap of communication and computation, yet they limit the number of available communicating processes per node [2, 5, 32]. While this can reduce the

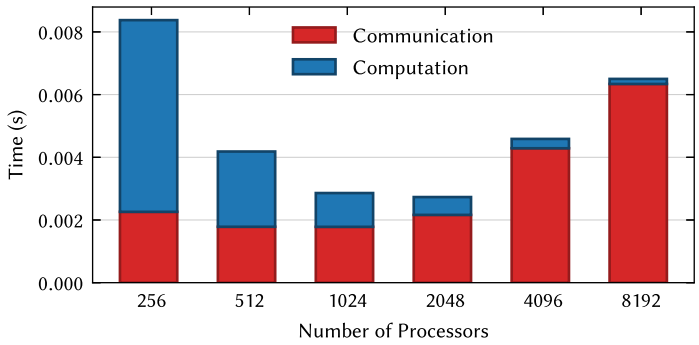

Figure 2.3. The time required for a single SpMV, split into communication and computation, for Example 2.1 run on Blue Waters with 16 processes per node.

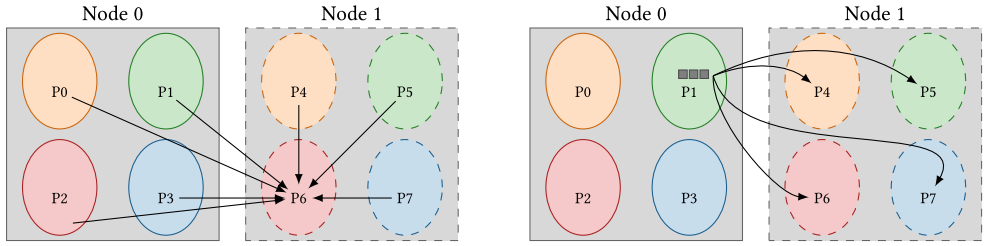

Figure 2.4. Standard communication. On the left, Node 0 injects multiple messages into the network, all to P6 on Node 1. On the right, P1 sends multiple messages containing the same data to each process on Node 1, resulting in "duplicate" data being sent across the network.

overall amount of time required for the SpMV over methods that use flat-MPI standard communication techniques, it typically does not reduce communication overhead as much as node-aware communication techniques with flat-MPI—detailed in Reference [9].

The inefficiency of the standard communication approach is attributed to two redundancies shown in Figure 2.4. First, many messages are injected into the network from each node. Some nodes are sending multiple messages to a single destination process on a separate node, creating a redundancy of messages. Second, processes send the necessary values from their local portion of v to any other process requiring that information for its local computation of w. However, the same information may already be available and received by a separate process on the same node, creating a redundancy of data being sent through the network.

Node-aware communication techniques [7, 9] mitigate these issues in both SpMVs and **sparse matrix-matrix (SpGEMM)** multiplication by considering node topology to further break down the off-process block into vector values of v that require on- or off-node communication; this decomposition is shown in Figure 2.2. As a result, costly redundant messages are traded for faster, on-node communication, resulting in two different multi-step schemes, namely three-step and two-step.

Three-step Node-aware Communication. Three-step node-aware communication eliminates both redundancies in standard communication by gathering all necessary data to be sent off-node in a single buffer. Efficient implementation of this method relies on pairing all processes with a

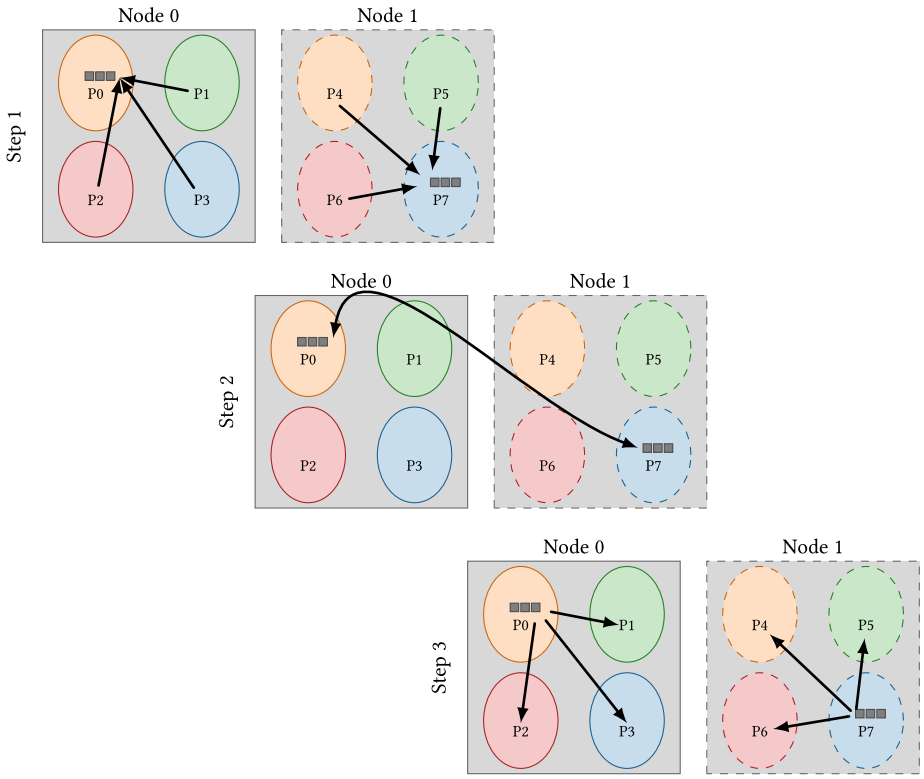

Figure 2.5. Three-step node-aware. In Step 1, all the data on Node 0 and Node 1 that need to be exchanged is collected in node local buffers on P0 and P7. In Step 2, the paired processes P0 and P7 exchange buffers. In Step 3, P0 and P7 redistribute the data to the correct receiving processes on Node 0 and Node 1, respectively.

receiving process on distinct nodes, thus ensuring that every process remains active throughout the entire communication scheme (see Figure 2.5).

First, all data to be sent to a separate node are gathered in a buffer by the single process paired with that node. Second, this process sends the data buffer to the paired process on the receiving node. Third, the paired process on the receiving node redistributes the data locally to the correct destination processes on-node. An example of these steps is outlined in Figure 2.5.

Two-Step Node-aware Communication. Two-step node-aware communication eliminates the redundancy of sending duplicate data from standard communication and decreases the number of inter-node messages but not to the same degree as three-step communication. In this case, *each* process exchanges the information needed by the receiving node with their paired process directly. Then the receiving node redistributes the messages on-node as shown in Figure 2.6. While multiple messages are sent to the same node, the duplicate data being sent through the network are eliminated. Hence, the number of bytes communicated with three-step and two-step node-aware schemes is the same, and often yields a significant reduction over the amount of data being sent through the network with standard communication.

2.2.1 Node-aware Communication Models. The *max-rate* model [22] is used to quantify the efficiency of node-aware communication throughout the remainder of Sections 2 and 3. For clarity, all modeling parameters referenced throughout the remainder of the article are defined in Table 1. The

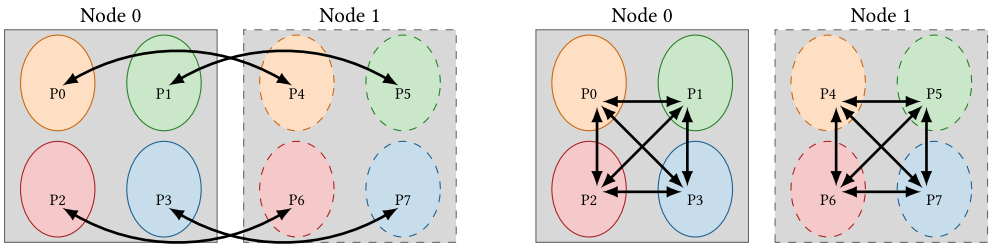

Figure 2.6. Two-step node-aware. Each process on Node 0 is paired with a receiving process on Node 1. In Step 1, each process on Node 0 sends the data needed by any process on Node 1 to their paired process on Node 1 and vice versa. Here, P0 is paired with P4, P1 with P5, P2 with P6, and P3 with P7. In Step 2, each process on Node 0 and Node 1 redistributes the inter-node data received to its final destination on node.

Table 1. Modeling Parameters

Parameter	Description	First use
p	number of processes	Section 2.2
nnz	number of nonzeros in A	Section 3
α	network latency	Equation (2.5)
s	maximum number of bytes sent by a process	Equation (2.5)
m	maximum number of messages sent by a process	Equation (2.5)
ppn	processes per node	Equation (2.5)
R_N	network injection rate (B/s)	Equation (2.5)
R_b	network rate (B/s)	Equation (2.5)
s_{proc}	maximum number of bytes sent by a process	Equation (2.8)
s_{node}	maximum number of bytes injected by a node	Equation (2.8)
$m_{\text{proc}\to\text{node}}$	maximum number of nodes to which a processor sends	Equation (2.8)
$m_{\text{node}\to\text{node}}$	maximum number of messages between two nodes	Equation (2.7)
$s_{\text{node}\to\text{node}}$	maximum size of a message between two nodes	Equation (2.7)

Standard communication modeling parameters are listed in the top portion of the table, with node-aware communication specific modeling parameters listed in the bottom portion.

max-rate model is an improvement to the standard postal model of communication, accounting for injection limits into the network. The cost of sending messages from a **symmetric multiprocessing (SMP)** node is modeled as

$$T = \alpha \cdot m + \max\left(\frac{\text{ppn} \cdot s}{R_N}, \frac{s}{R_b}\right), \tag{2.5}$$

where α is the latency, m is the number of messages sent by a single process on a given node, s is the number of bytes sent by a single process on a given SMP node, ppn is the number of processes per node, R_N is the rate at which a **network interface card (NIC)** can inject data into the network, and R_b is the rate at which a process can transport data.

In the case of on-node messages, the injection rate is not present and the max-rate model reduces to the standard postal model for communication

$$T = \alpha_\ell \cdot m + \frac{s}{R_{b,\ell}}, \tag{2.6}$$

where α_ℓ is the *local* or on-node latency and $R_{b,\ell}$ is the rate of sending a message on-node.

In Reference [9], the max-rate model is extended to two-step and three-step communication by splitting the model into inter-node and intra-node components. For three-step, the communication model becomes

$$T_{\text{total}} = \underbrace{\alpha \cdot \frac{m_{\text{node}\to\text{node}}}{\text{ppn}} + \max\left(\frac{s_{\text{node}}}{R_N}, \frac{s_{\text{proc}}}{R_b}\right)}_{\text{inter-node}} + \underbrace{2 \cdot \left(\alpha_\ell \cdot (\text{ppn} - 1) + \frac{s_{\text{node}\to\text{node}}}{R_{b,\ell}}\right)}_{\text{intra-node}}, \tag{2.7}$$

where $m_{\text{node}\to\text{node}}$ is the maximum number of messages communicated between any two nodes and $s_{\text{node}\to\text{node}}$ is the size of messages communicated between any two nodes.

For two-step, this results in

$$T_{\text{total}} = \underbrace{\alpha \cdot m_{\text{proc}\to\text{node}} + \max\left(\frac{s_{\text{node}}}{R_N}, \frac{s_{\text{proc}}}{R_b}\right)}_{\text{inter-node}} + \underbrace{\alpha_\ell \cdot (\text{ppn} - 1) + \frac{s_{\text{proc}}}{R_{b,\ell}}}_{\text{intra-node}}, \tag{2.8}$$

where s_{node} and s_{proc} represent the maximum number of bytes injected by a single NIC and communicated by a single process from an SMP node, respectively, and $m_{\text{proc}\to\text{node}}$ is the maximum number of nodes with which any process communicates.

The latency to communicate between nodes, α, is often much higher than the intra-node latency, α_ℓ, thus motivating a multi-step communication approach. In a two-step method, having every process on-node communicate data minimizes the constant factor $\max(\frac{s_{\text{node}}}{R_N}, \frac{s_{\text{proc}}}{R_b})$, which depends on the maximum amount of data being communicated to a separate node by a single process. In practice, a three-step method often yields the best performance for a parallel SpMV, since the amount of data being communicated by a single process is often small. As a result, moving the data to be communicated off-node into a single buffer minimizes the first term in Equation (2.7). These multi-step communication techniques minimize the amount of time spent in inter-node communication. We extend this idea to the block vector operation in Section 4.1.

3 PERFORMANCE STUDY OF ENLARGED CONJUGATE GRADIENT

In this section, we detail the per-iteration performance and performance modeling of ECG. A communication efficient version of Algorithm 1 is implemented in Raptor [10] and is based on the work in Reference [21]. Throughout this section and the remainder of the article, we assume an $n \times n$ matrix A with nnz nonzeros is partitioned row-wise across a set of p processes. Each process contains at most $\frac{n}{p}$ contiguous rows of the matrix A. In the modeling that follows, we assume an equal number of nonzeros per partition. In addition, each block vector in Algorithm 1—R, X, Z, AZ, P, and AP—is partitioned row-wise and with the same row distribution as A. The variables c, d, and d_{old} are size $t \times t$ and a copy of each is stored locally on each process. Tests were performed on the Blue Waters Cray XE/XK machine at the National Center for Supercomputing Applications at University of Illinois. Blue Waters [11, 25] contains a three-dimensional torus Gemini interconnect; each Gemini consists of two nodes. The complete system contains 22 636 XE compute nodes; each with two AMD 6276 Interlagos processors, and additional XK compute nodes unused in these tests.

3.1 Implementation

The scalability of a direct implementation of Algorithm 1 is limited [21]; however, this is improved by fusing communication and by executing the system solve in Line 5 in Algorithm 1 on each process. This is accomplished in Reference [21] by decomposing the computation of P into several steps as described in Algorithm 3. The $t \times t$ product $Z^T(AZ)$ is stored locally on every process in the storage space of c, as shown in Figure 3.1. The Cholesky factorization on Line 3 of Algorithm 3 is performed simultaneously on every process, yielding a (local) copy of C. Then each process

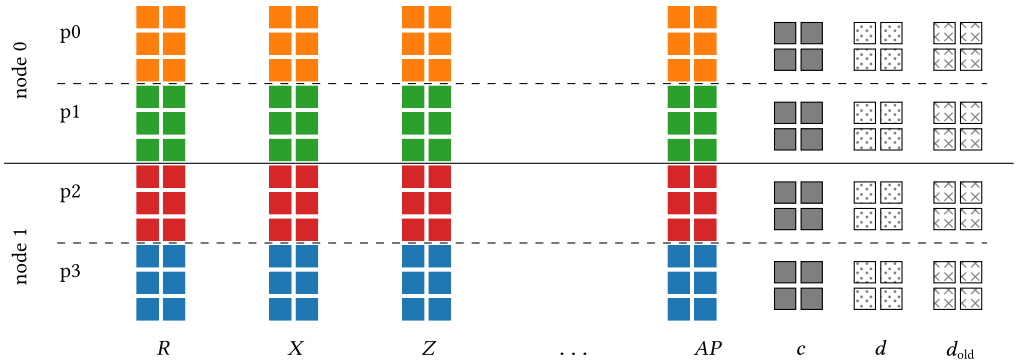

Figure 3.1. The above figure displays the partitioning of all the vectors and intermediary vectors, R, X, Z, AZ, P, and AP along with $t \times t$ working arrays c, d, and d_{old}, as in Algorithm 1, for $t = 2$.

performs a local triangular system solve using the local vector values of Z to construct the local portion of P (see Line 4). Similarly, an additional sparse matrix block vector product $AP = A * P$ is avoided by noting that AP is constructed using

$$AP \leftarrow \texttt{Triangular Solve with Multiple Right Sides of } AP * C = AZ,$$

since the product AZ and the previous iteration's $AP = A * P$ are already stored.

ALGORITHM 3: Calculating $P := Z(Z^T AZ)^{-1/2}$

1	$AZ \leftarrow A * Z$	[sparse matrix-block vector multiplication]
2	$Z^T AZ \leftarrow Z^T * AZ$	[block inner product]
3	$C^T C \leftarrow Z^T AZ$	[Cholesky factorization]
4	$P \leftarrow$ solve $P * C = Z$	[Triangular solve with multiple right sides]

Algorithm 4 summarizes our implementation in terms of computational kernels, with the on process computation in terms of floating point operations along with the associated type of communication. The remainder of the calculations within ECG consist of local block vector updates, as well as block vector inner products for the values c, d, and d_{old}. A straightforward approach is to compute these independently within the algorithm, resulting in four `MPI_Allreduce` global communications per iteration, as in Reference [21]. However, since the input data required to calculate c, d, and d_{old} are available on Line 6 in Algorithm 1 when c is computed, a single global reduction is possible. The implementation described in Algorithm 4 highlights a single call to `MPI_Allreduce` for all of these values and reducing them in the same buffer. This reduces the number of `MPI_Allreduce` calls to two per iteration. Additionally, the point-to-point communication required for the SpMBV is performed using the standard communication approach described in Section 2.2 and used in standard industry codes, such as PETSc [6].

From Algorithm 4, we note that computation and communication per iteration costs of ECG have increased over that of parallel CG, with computation in terms of floating point operations and the type of communication incurred next to each kernel. For our implementation, the number of collective communication calls to `MPI_Allreduce` has remained the same as CG (at two), but the number of values in the global reductions has increased from a single float in each of CG's global reductions to t^2 and $3t^2$. The singular SpMV from CG has increased to a SpMBV product, which does not increase the number of point-to-point messages, but does increase the size of the messages being communicated by the enlarging factor t. This SpMBV is performed the same as

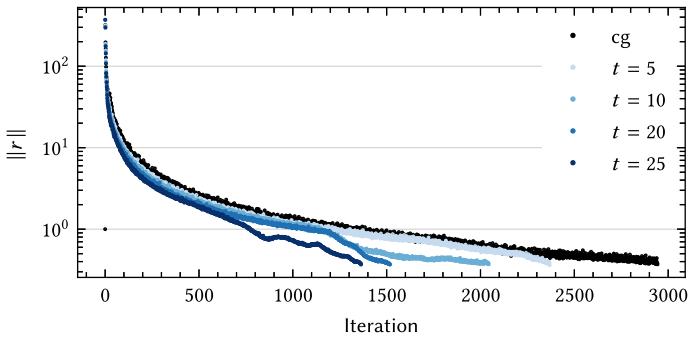

Figure 3.2. Residual history for CG and ECG with various enlarging factors t for Example 2.1.

ALGORITHM 4: Enlarged Conjugate Gradient by Kernel

1	SpMV			
2	Vector Initialization			
3	**for** $k = 1, \ldots$			
4	SpMBV	→	Flops: $2 \cdot \frac{\text{nnz}}{p} \cdot t$	Comm: point-to-point
5	Block inner product	→	Flops: $2 \cdot \frac{n}{p} \cdot t^2$	Comm: global all reduce
6	Cholesky decomposition	→	Flops: $\frac{1}{3} \cdot t^3$	
7	Triangular solves	→	Flops: $2 \cdot \frac{1}{2} \cdot t^2$	
8	Block inner product	→	Flops: $2 \cdot \frac{n}{p} \cdot t^2$	Comm: global all reduce
9	Block vector addition	→	Flops: $2 \cdot \frac{n}{p} \cdot t$	
10	Block vector axpy	→	Flops: $2 \cdot \frac{n}{p} \cdot t$	

the SpMV presented in Figure 2.2 with the local portion of the SpMBV computed while messages are exchanged to attain off-process data. Additionally, the local computation for *each kernel* has increased by a factor of t. ECG uses these extra per iteration requirements to reduce the total number of iterations to convergence, resulting in fewer iterations than CG as seen in Figure 3.2.

3.2 Per Iteration Performance

In Figure 3.3, we decompose the performance of a single iteration of ECG for Example 2.1 into (local) computation, point-to-point communication, and collective communication. Performance tests were executed on Blue Waters [11, 25]. Each test is the average of 20 iterations of ECG; reported times are the maximum average time recorded for any single process. At small scales, local computation dominates performance, while at larger scales, the point-to-point communication in the single SpMBV kernel and the collective communication in the block vector inner products become the bottleneck in ECG. Figure 3.3 also shows the time spent in a single inner product. While we observe growth with the number of processes, as expected, the relative cost (and growth) within ECG remains low. Importantly, increasing t at high processor counts does not significantly contribute to cost. This is shown in Figure 3.4, where the mean runtime for various block vector inner products all fall within each other's confidence intervals. This suggests that increasing t to drive down the iteration count will have little effect on the per iteration cost of the two calls to `MPI_Allreduce`, and in fact, will result in fewer total calls to it due to the reduction in iterations.

The remainder of this section focuses on accurately predicting the performance of a single iteration of ECG through robust communication performance models. In particular, SpMBV

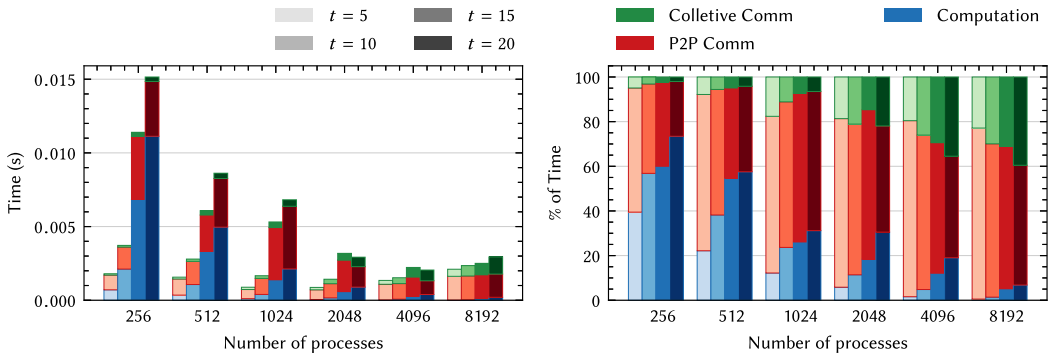

Figure 3.3. Time (left) and percentage of time (right) for a single iteration of ECG for various block vector sizes, t, and processor counts for Example 2.1. The three varying shades for each t value represent the amount of overall time spent in on-process computation, point-to-point communication, and collective communication.

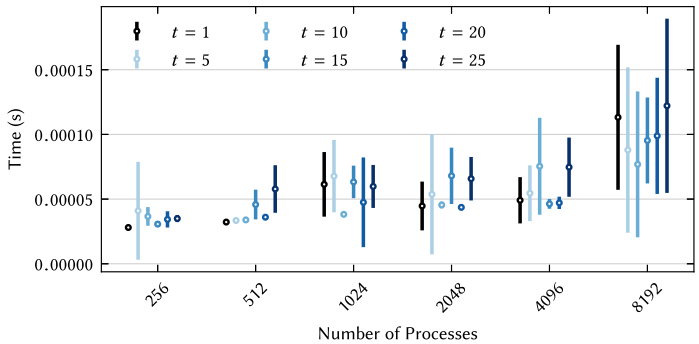

Figure 3.4. Time for a single inner product (right) for various block vector sizes, t, and processor counts for Example 2.1. Vertical lines denote the confidence intervals.

communication is addressed in detail in Section 4, where we discuss new node-aware communication techniques for blocked data.

3.3 Performance Modeling

To better understand the timing profiles in Figure 3.3, we develop performance models. Below, we present two different models for the performance of communication within a single iteration of ECG. First, consider the standard postal model for communication, which represents the maximum amount of time required for communication by an individual process in a single iteration of ECG as

$$T_{\text{postal}} = \underbrace{\alpha \cdot m + \frac{s \cdot t}{R_b}}_{\text{point to point}} + \underbrace{2 \cdot \alpha \cdot \log(p) + \frac{f \cdot 4 \cdot t^2}{R_b}}_{\text{collective}}, \tag{3.1}$$

where f is the number of bytes for a floating point number—e.g., $f = 8$. See Table 1 for a complete description of model parameters. As discussed in Section 2.2.1, this model presents a misleading picture on the performance of ECG at scale, particularly on current supercomputer architectures where SMP nodes encounter injection bandwidth limits when sending inter-node messages.

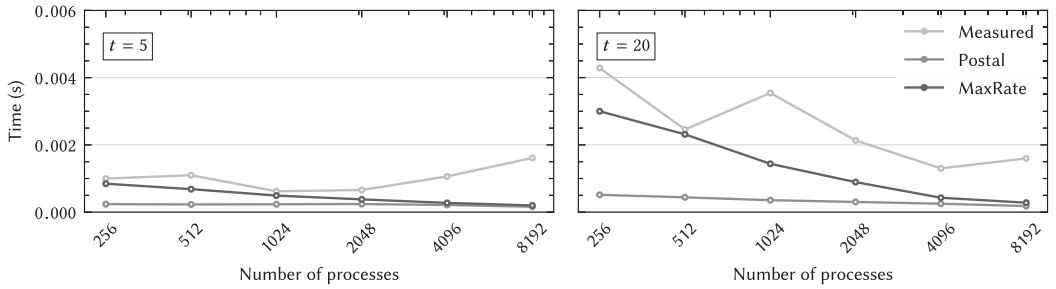

Figure 3.5. Max-rate model versus the postal model for the point-to-point communication in one iteration of ECG for Example 2.1 using various t. Measured runtimes are also included ($t = 5$ and $t = 20$ shown for brevity).

To improve the model, we drop in the max-rate model for the point-to-point communication, resulting in

$$T_{MR} = \underbrace{\alpha \cdot m + \max\left(\frac{\mathsf{ppn} \cdot s \cdot t}{R_N}, \frac{s \cdot t}{R_b}\right)}_{\text{point to point}} + \underbrace{2 \cdot \alpha \cdot \log(p) + \frac{f \cdot 4 \cdot t^2}{R_b}}_{\text{collective}}. \tag{3.2}$$

Figure 3.5 shows that the max-rate model provides a more accurate upper bound on the time spent in point-to-point communication within ECG. The term $2 \cdot \alpha \cdot \log(p) + \frac{f \cdot 4 \cdot t^2}{R_b}$ remains unchanged in Equations (3.1) and (3.2) to represent the collective communication required for the two block vector inner products. Each block vector inner product incurs latency from requiring $\log(p)$ messages in an optimal implementation of the `MPI_Allreduce`. More accurate models for modeling the communication of the `MPI_Allreduce` in the inner product exist, such as the logP model [17] and logGP model [3], but optimization of the reduction is outside the scope of this article, so we leave the postal model for representing the performance of the inner products.

Modeling the computation within an iteration of ECG is straightforward. The computation for a single iteration of ECG is written as the sum of the kernel floating point operations in Algorithm 4, which results in the following:

$$T_{comp} = \gamma \cdot \left((2t)\frac{\mathsf{nnz}}{p} + (4t + 4t^2)\frac{n}{p} + \frac{1}{2}t^2 + \frac{1}{3}t^3\right), \tag{3.3}$$

where γ is the time required to compute a single floating point operation. More accurate models, such as the roofline model, exist for predicting peak computational performance, as well as identifying computational bottlenecks [35]. However, because point-to-point communication is the overwhelmingly dominant cost in strong-scaling performance of ECG, we maintain a simple computational model. In total, we arrive at the following model for a single iteration of ECG:

$$T_{ECG} = \alpha \cdot m + \max\left(\frac{\mathsf{ppn} \cdot s \cdot t}{R_N}, \frac{s \cdot t}{R_b}\right) + 2 \cdot \alpha \cdot \log(p) + \frac{f \cdot 4 \cdot t^2}{R_b} + \gamma \cdot \left((2t)\frac{\mathsf{nnz}}{p} + (4t + 4t^2)\frac{n}{p} + \frac{1}{2}t^2 + \frac{1}{3}t^3\right). \tag{3.4}$$

Using this model, we can predict the reduction in the amount of time spent in point-to-point communication using the multi-step communication techniques presented in Section 2.2.1 for a single iteration of ECG for Example 2.1—see Table 2.

We see that ECG is still limited at large processor counts even when substituting the node-aware communication techniques to replace the costly point-to-point communication observed in Figure 3.3. The models do predict a large amount of speedup for most cases, however, when using three-step communication, suggesting that node-aware communication techniques can reduce

Table 2. Modeled Percentage of Time to Be Spent in Point-to-Point Communication for Multistep ("m-s") Compared against the Measured Percentage of Time Spent in Point-to-Point Communication for Standard ("std") in a Single Iteration of ECG for Example 2.1 with Varying t and Processor Counts on Blue Waters

(a) Two-Step Communication

procs →	256		512		1,024		2,048		4,096		8,192	
t	m-s	std	m-s	std	m-s	std	m-s	std	m-s	std	m-s	std
5	42.2	55.6	57.9	70.0	71.7	70.2	80.1	75.5	83.5	78.9	81.9	76.6
10	21.2	40.0	34.5	56.2	51.2	65.2	65.9	67.5	75.9	69.1	77.7	68.7
15	13.0	37.6	22.2	40.4	35.2	66.7	48.6	67.0	60.0	58.5	60.5	63.8
20	9.3	24.5	16.5	38.3	27.4	62.3	40.6	47.6	54.4	45.5	57.2	53.6

(b) Three-Step Communication

procs →	256		512		1,024		2,048		4,096		8,192	
t	m-s	std	m-s	std	m-s	std	m-s	std	m-s	std	m-s	std
5	29.6	55.6	43.0	70.0	54.8	70.2	64.2	75.5	72.3	78.9	71.4	76.6
10	14.6	40.0	24.4	56.2	35.9	65.2	49.5	67.5	65.6	69.1	68.5	68.7
15	9.2	37.6	15.8	40.4	23.9	66.7	34.4	67.0	49.9	58.5	50.8	63.8
20	6.7	24.5	11.9	38.3	18.6	62.3	28.7	47.6	45.9	45.5	48.8	53.6

Blue values correspond to values for which the percentage of time decreased by 5–20%, green values are a decrease of over 20%, and red values predict an increase over the original percentage.

the large point-to-point bottleneck observed in the performance study. While a large communication cost stems equally from the collective communication the `MPI_Allreduce` operations, their performance is dependent upon underlying `MPI` implementation and outside the scope of this article. We address the point-to-point communication performance further in Section 4 by analyzing it through the lens of node-aware communication techniques, optimizing them to achieve the best possible performance at scale.

4 OPTIMIZED COMMUNICATION FOR BLOCKED DATA

As discussed in Section 3, scalability for ECG is limited by the SpMBV multiplication kernel defined as

$$A \cdot V \to W, \tag{4.1}$$

with $A \in \mathbb{R}^{n \times n}$ and V, $V \in \mathbb{R}^{n \times t}$, where $1 < t \ll n$. Due to the block vector structure of X, each message in a SpMV is increased by a factor of t (see Figure 4.1). The larger messages associated with $t > 1$ increase the amount of time spent in the point-to-point communication at larger scales, making the SpMBV operation an ideal candidate for node-aware messaging approaches.

Additionally, the SpMBV kernel is a key computational kernel within block Krylov subspace methods [31]. While block Krylov methods are slightly different from *enlarged* Krylov methods, in that they solve a linear system with multiple right-hand sides, and *enlarged* Krylov methods project an initial residual into a larger subspace to solve a single system, they both require a SpMBV computational kernel. It is worth noting that the techniques presented in the following sections impact the parallel performance of block Krylov subspace methods via reduction of time spent in point-to-point communication. However, we do not analyze per iteration performance impacts outside the context of ECG.

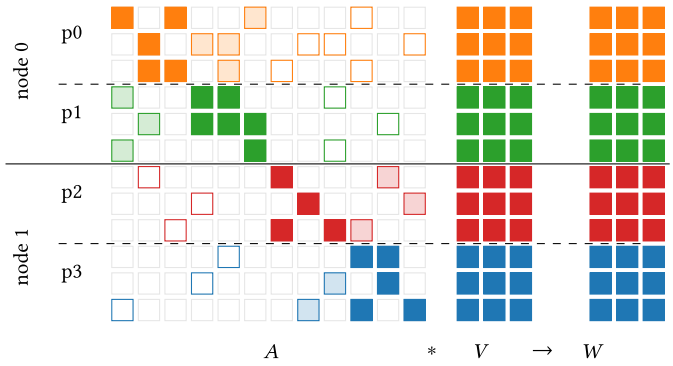

Figure 4.1. Sparse matrix-block vector multiplication (cf. Figure 2.2).

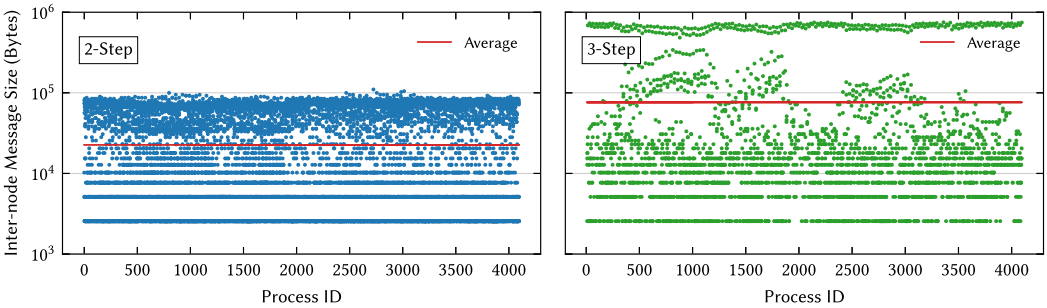

Figure 4.2. The inter-node message sizes across all processes for two-step and three-step communication when performing the SpMBV for Example 2.1 when $t = 20$ across 4,096 processes on Blue Waters. The average message size across all processes is marked by a red line.

4.1 Performance Modeling

Recalling the node-aware communication models from Section 2.2.1, we augment the two-step and three-step models with block vector size, t. As a result, the two-step model in Equation (2.8) becomes

$$T_{total} = \underbrace{\alpha \cdot m_{\text{proc}\to\text{node}} + \max\left(\frac{t \cdot s_{\text{node}}}{R_N}, \frac{t \cdot s_{\text{proc}}}{R_b}\right)}_{\text{inter-node}} + \underbrace{\alpha_\ell \cdot (\text{ppn} - 1) + t \cdot \frac{s_{\text{proc}}}{R_{b,\ell}}}_{\text{intra-node}}, \tag{4.2}$$

while the three-step model in Equation (2.7) becomes

$$T_{total} = \underbrace{\alpha \cdot \frac{m_{\text{node}\to\text{node}}}{\text{ppn}} + \max\left(\frac{t \cdot s_{\text{node}}}{R_N}, \frac{t \cdot s_{\text{proc}}}{R_b}\right)}_{\text{inter-node}} + \underbrace{2 \cdot \left(\alpha_\ell \cdot (\text{ppn} - 1) + t \cdot \frac{s_{\text{node}\to\text{node}}}{R_{b,\ell}}\right)}_{\text{intra-node}}. \tag{4.3}$$

In both models, t impacts maximum rate, a term that is relatively small for $t = 1$ and dominates the inter-node portion of the communication as t grows. Since messages are increased by a factor of t, the single buffer used in three-step communication quickly reaches the network injection bandwidth limits. Using multiple buffers, as in two-step communication, helps mitigate the issue, however more severe imbalance persists, since the amount of data sent to different nodes is often widely varying.

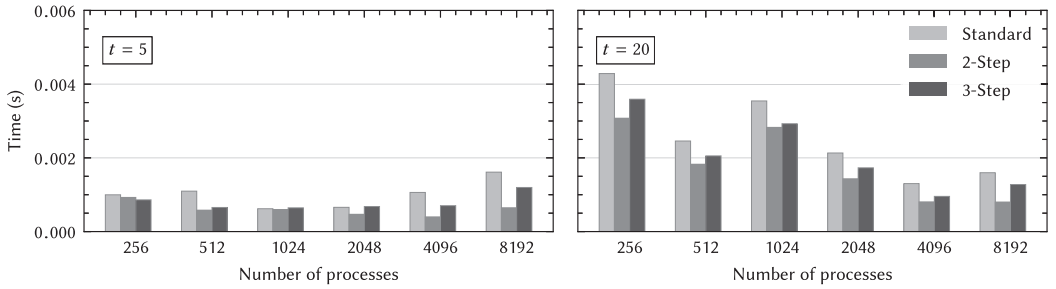

Figure 4.3. Strong scaling results for Example 2.1 using standard, two-step, and three-step communication in the SpMBV ($t = 5$ and $t = 20$ shown for brevity).

Figure 4.2 shows every inter-node message sent by a single process alongside the message size for Example 2.1 when performing the SpMBV kernel with 4,096 processes and 16 processes per node. We present the message sizes for three-step and two-step communication, noting that the overall number of inter-node messages decreases when using three-step communication, but the average message size sent by a single process increases.

For $t = 20$, the maximum message size nears 10^6 for three-step communication, while the maximum message size barely reaches 10^5 for two-step communication. Additionally, there is clear imbalance in the inter-node message sizes for two-step communication with messages ranging in size from 10^3 to 10^5 Bytes.

4.2 Profiling

We next apply the node-aware communication strategies presented for SpMVs and SpGEMMs in Section 2.2 to the SpMBV kernel within ECG.

Figure 4.3 displays the performance of standard, two-step, and three-step communication when applied to the SpMBV kernel for Example 2.1 with $t = 5$ and $t = 20$. The two-step communication appears to outperform the others in most cases. This is due to the large amount of data to be sent off-node that is split across many processes. We see two-step communication performing better due to the term $\alpha \cdot m_{\text{proc}\to\text{node}}$ in Equation (4.2) being smaller than the $\alpha \cdot m_{\text{node}\to\text{node}}$ term in the three-step communication model (Equation (4.3)) due to multiplication by the factor t. In fact, all counts measured in the ECG profiling section are now multiplied by the enlarging factor t. While the traditional SpMV shows speedup with three-step communication, we now see that two-step is generally the best fit for our methods as message size, and thus t, increases.

Next we consider node-aware communication performance results for a subset of the largest matrices in the SuiteSparse matrix collection [18] (matrix details can be found in Table 3). These were selected based on size and density to provide a variety of scenarios.

While two-step communication is effective in many instances, it is not always the most optimal communication strategy, as depicted in Figure 4.4. Unlike the results for Example 2.1, three-step and two-step communication do not always outperform standard communication, and in some instances (4,096 processes), for most values of t there is performance degradation. This is seen more clearly in Figure 4.5.

It is important to highlight cases where only a single node-aware communication technique results in performance deterioration over standard communication. Distinct examples include `Geo_1438` and `thermal2` on 4,096 processes. Both of these matrices benefit from two-step communication for $t = 10$ and 20, but performance degrades when using three-step communication. Another example is the performance of `ldoor` on 8,192 processes (see Figure 4.5). For $t = 5$ and 10,

Table 3. Test Matrices

matrix	rows/ cols	nnz	nnz/row	density
audikw_1	943,695	77,651,847	82.3	8.72e-05
Geo_1438	1,437,960	60,236,322	41.9	2.91e-05
bone010	986,703	47,851,783	48.5	4.92e-05
Emilia_923	923,136	40,373,538	43.7	4.74e-05
Flan_1565	1,565,794	114,165,372	72.9	4.66e-05
Hook_1498	1,498,023	59,374,451	39.6	2.65e-05
ldoor	952,203	42,493,817	44.6	4.69e-05
Serena	1,391,349	64,131,971	46.1	3.31e-05
thermal2	1,228,045	8,580,313	7.0	5.69e-06

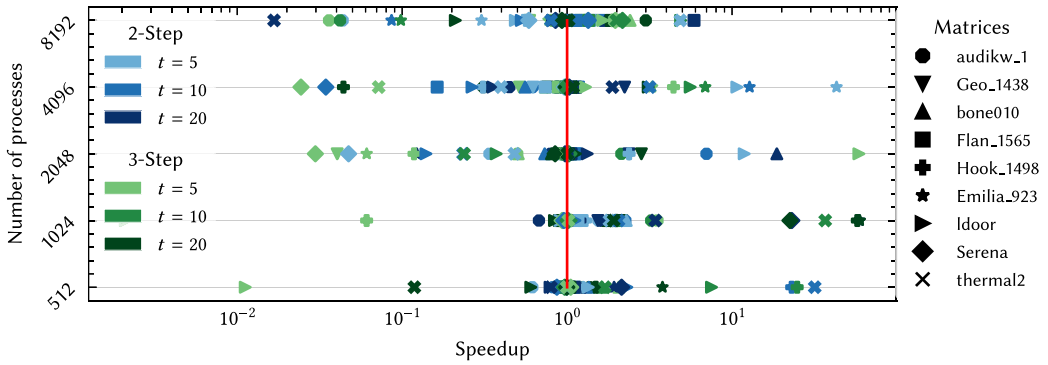

Figure 4.4. Speedup of two-step and three-step communication over standard communication in the SpMBV for various matrices from the SuiteSparse matrix collection on Blue Waters. The red line marks 1.0, or no speedup.

ldoor results in performance degradation using two-step communication, but up to 5× speedup over standard communication when using three-step communication. These cases highlight that while one node-aware communication technique underperforms in comparison to standard communication, the other node aware technique is still much faster. Using this as the key motivating factor, we discuss an optimal node-aware communication technique for blocked data in Section 4.3.

4.3 Optimal Communication for Blocked Data

When designing an optimal communication scheme for the blocked data format, the main consideration is the impact the number of vectors within the block have on the size of messages communicated. The effects message size and message number have on performance can vary based on machine, hence we present results for Blue Waters alongside Lassen [23]. Lassen, a 23-petaflop IBM system at Lawrence Livermore National Laboratory, consists of 792 nodes connected via a Mellanox 100 Gb/s Enhanced Data Rate InfiniBand network. Each node on Lassen is dual-socket with 44 IBM Power9 cores and 4 NVIDIA Volta GPUs (which are unused in our tests.)

In Figure 4.6, we view the effects placement of data and amount of data being communicated has on performance times for two different machines. This figure shows the amount of time required to send various numbers of bytes between two processes when they are located on the same node but different sockets (blue) and on the same node and same socket (red) for the machines Blue Waters (left) and Lassen (right). It also shows the amount of time required to communicate between two

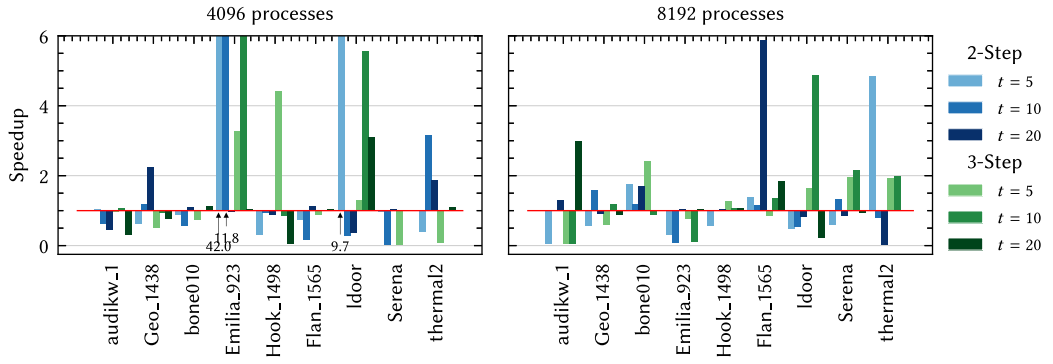

Figure 4.5. Speedup of two-step and three-step communication over standard communication in the SpMBV for various matrices on Blue Waters for 4,096 and 8,192 processes. The red line marks 1.0, or no speedup. Speedups greater than 6.0 are given via annotations on the plot.

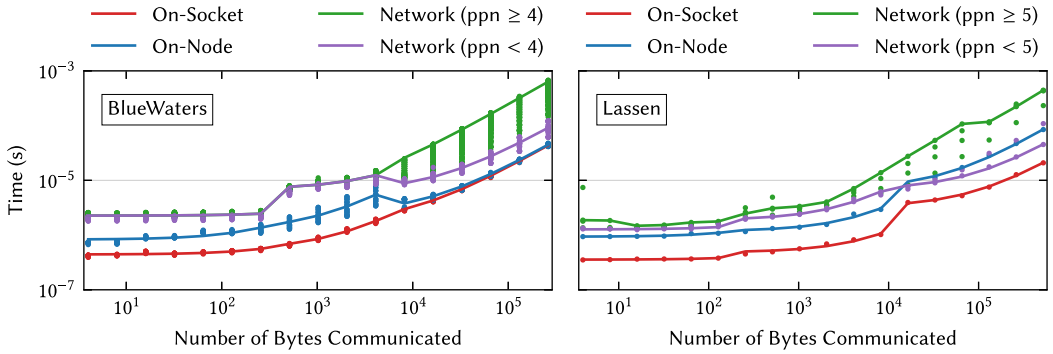

Figure 4.6. Communication time on-socket, on-node, or across the network on Blue Waters (left) and Lassen (right). Measured times are displayed as dots; solid lines represent max-rate model [22]. ppn represents the number of processors participating in communication. (The Blue Waters data were initially presented in Reference [8] and is replotted here.)

processes on separate nodes when there are less than four (Blue Waters) or five (Lassen) active processes communicating through the network at the same time or more than four or five processes communicating through the network at the same time. We see that as the number of bytes communicated between two processes increases, it becomes increasingly important whether those two processes are located on the same socket, node, or require communication through the network. For both machines, inter-node communication is fastest when message sizes are small, and there are few messages being injected into the network. On Blue Waters, intra-node communication is the fastest, with the time being dependent on how physically close the processes are located. For instance, when two processes are on the same socket, communication is faster than when they are on the same node, but different sockets. This is true for Lassen, as well, but cross-socket intra-node communication is not always faster than communicating through the network. Once message sizes exceed 10^4 bytes, and there are fewer than five processes actively communicating, inter-node communication is faster than two cross-socket intra-node processes communicating. For both machines, however, we see that once a large enough communication volume is reached, it becomes faster to split the inter-node data being sent across a subset of the processes on a single node due to network contention as observed in Reference [8].

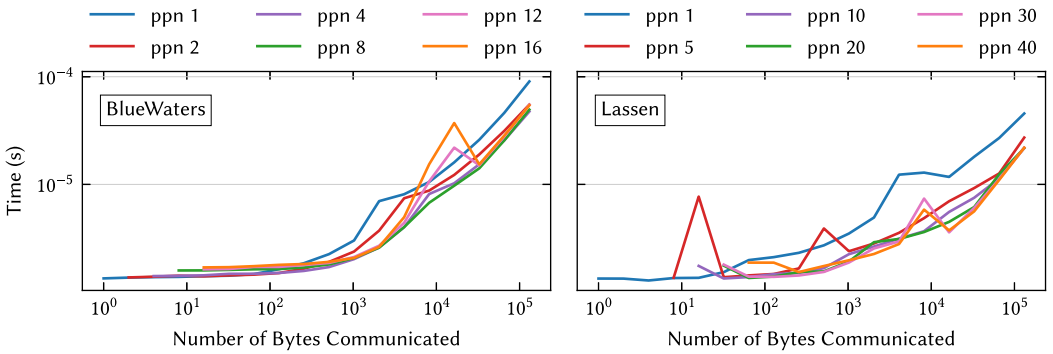

Figure 4.7. Time required to send data between two processes on separate nodes with the data communication split across ppn processes. Timings for Blue Waters and Lassen are presented in the left and right plots, respectively.

In addition to the importance of the placement of two communicating processes, the total message volume and number of actively communicating processes plays a key role in communication performance. While it is extremely costly for every process on a node to send 10^5 bytes as seen in Figure 4.6, there are performance benefits when splitting a large communication volume across all processes on a node, depicted in Figure 4.7. Blue Waters sees modest performance benefits when splitting large messages across multiple processes, whereas Lassen sees much greater performance benefits.

These observations help justify why three-step communication would outperform two-step communication, and vice versa in certain cases of the SpMBV profiling presented in Figure 3.5. Sending all messages in a single buffer becomes impractical when the block size, t, is very large, but having each process communicate with a paired process also poses problems when some of the inter-node messages being sent are still very large, as seen in Figure 4.2.

Motivated by the results above, we introduce an optimized multi-step communication process that combines the aspects of both the three-step and two-step communication techniques, and using three-step or two-step communication when necessary. We reduce the number of inter-node messages and conglomerate messages to be sent off node for certain cases when the message sizes to be sent off-node are below a given threshold, and we split the messages to be sent off-node across multiple processes when the message size is larger than a given threshold. Hence, each node is determining the most optimal way to perform its inter-node communication. As a result, this nodal optimal communication eliminates the redundancies of data being injected into the network, just as three-step and two-step do, but in some cases, it does not reduce the number of inter-node messages as much as three-step, and in fact can increase the number of inter-node messages injected by a single node to be larger than those injected by two-step, but never exceeding the total number of active processes per node.

The number of inter-node messages sent can be represented by the following inequality:

$$m_{\text{node}\to\text{node}} \leq n_{opt} \leq \max(m_{\text{proc}\to\text{node}}, \text{ppn}), \tag{4.4}$$

where n_{opt} is the number of inter-node messages injected by a single process for the optimal node-aware communication, and it is bounded below by the worst-case number of messages communicated in three-step communication (Equation (4.3)) and above by the worst-case number of messages communicated in two-step communication (Equation (4.2)) or ppn, whichever is greater.

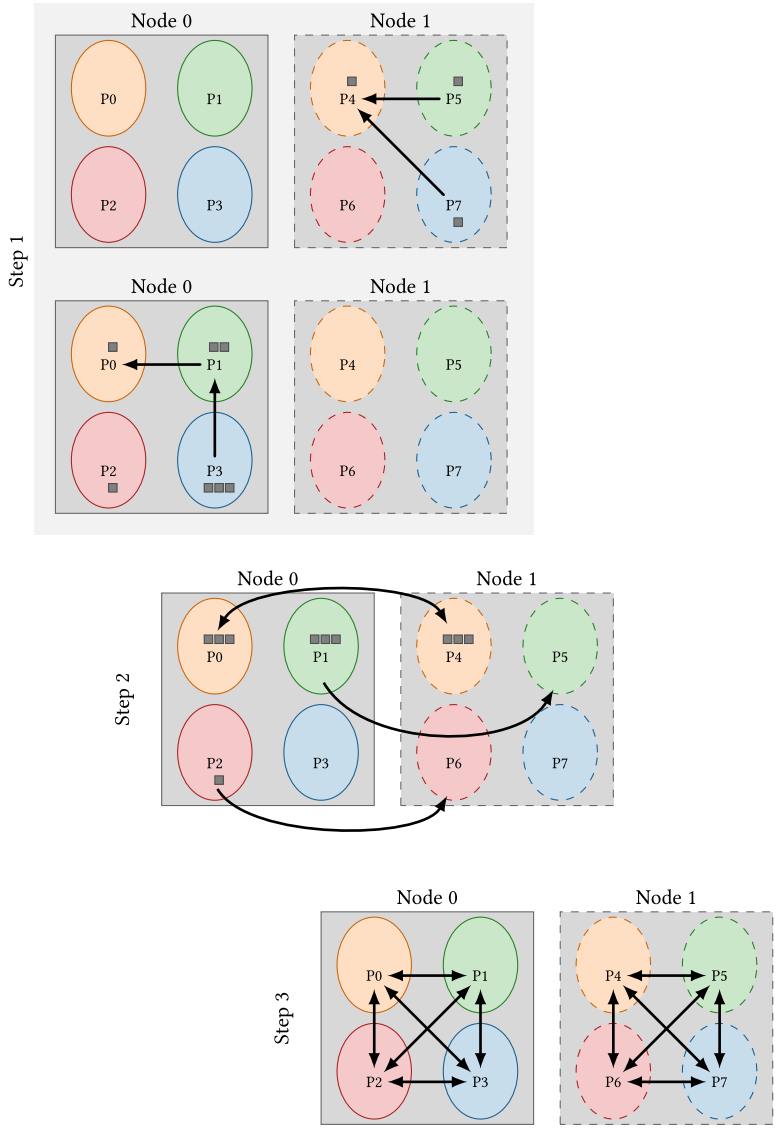

Figure 4.8. Node-aware communication on two nodes with four processes each.

The proposed process, excluding reducing the global communication strategy to three-step or two-step communication, is summarized in Figure 4.8. This figure outlines the nodal optimal process of communicating data between two nodes, each with four local processes.

Step 1: Each node conglomerates small messages to be sent off-node while retaining larger messages. Messages are assigned to processes in descending order of size to the first available process on node. This is done for every node simultaneously.

Step 2: Buffers prepared in step 1 are sent to their destination node, specifically to the paired process on that node with the same local rank. P0 exchanges data with P4, while P1 sends data to P5, and P2 sends data to P6.

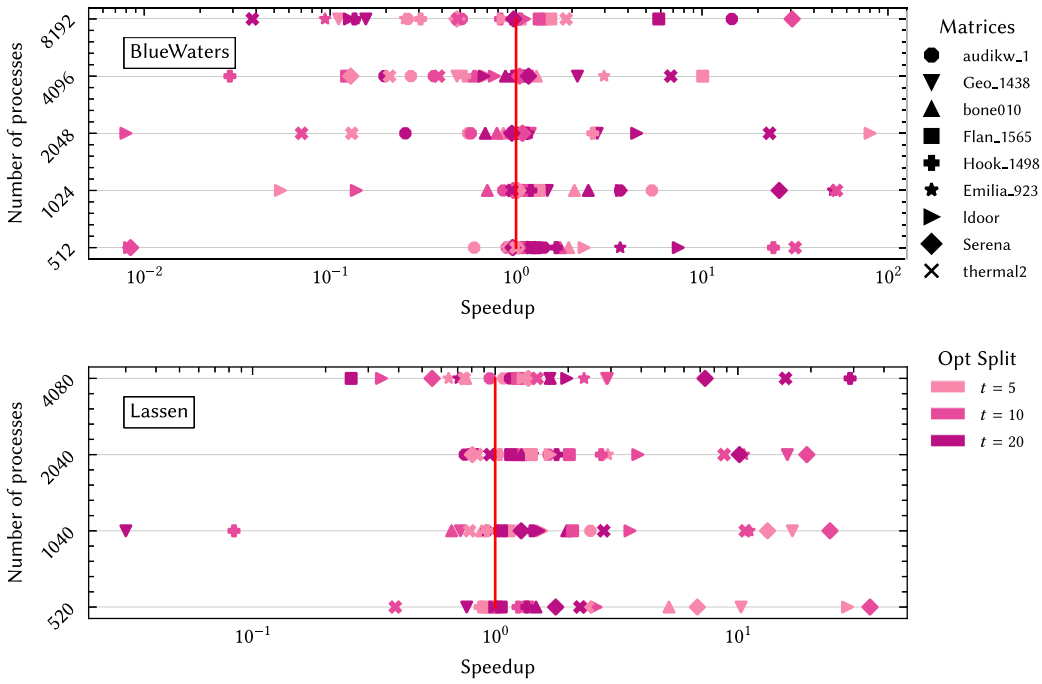

Figure 4.9. Speedup of optimal communication over standard communication (without reducing to a global communication strategy) in the SpMBV for various matrices from the SuiteSparse matrix collection on Blue Waters and Lassen. The red line marks 1 or no speedup.

Step 3: All processes on each individual node redistribute their received data to the correct destination processes on-node. In this step, all communication is local.

The resulting speedups for the two systems Blue Waters and Lassen are presented in Figure 4.9. Here the message size cutoff being used is the message size cutoff before the MPI implementation switches to the rendezvous protocol. For sending large messages between processes, the rendezvous protocol communicates an envelope first, then the remaining data are communicated after the receiving process allocates buffer space. It is necessary to use this protocol for large messages, but there is a slight slowdown over sending messages via the short and eager protocols that either include the data being sent as part of the envelope (short) or eagerly send the data if they do not fit into the envelope (eager). This cutoff is chosen because rendezvous is the slowest protocol and because the switch to the rendezvous protocol is approximately the crossover point when on-node messages become slower than network messages on Lassen (around 10^4 Bytes in Figure 4.6.)

Using this cutoff point, the method sees speedup for *some* test matrices and performance degradation for most on Blue Waters, which is consistent with the minimal speedup seen by splitting large messages across multiple processes in Figure 4.7. Additionally, it is likely that network contention is playing a large role in the Blue Waters results as the message sizes become large based on the findings in Reference [8], and due to each node determining independently how to send its own data without consideration of the size or number of messages injected by other nodes.

The nodal optimal communication performs better on Lassen than Blue Waters and aligns with expectations based on the combination of using three-step for nodes with small inter-node messages to inject into the network where on-node communication is faster than network communication (Figure 4.6) and splitting large messages where the benefits are much greater (Figure 4.7).

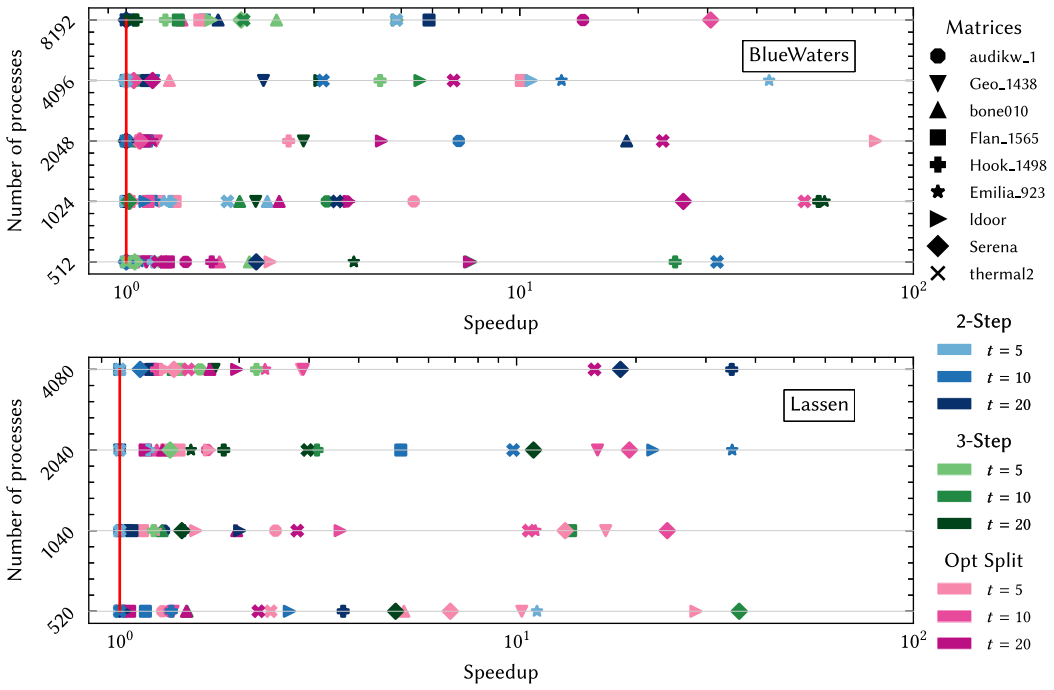

Figure 4.10. Speedup of tuned communication over standard communication in the SpMBV for various matrices from the SuiteSparse matrix collection on Blue Waters and Lassen. The red line marks 1.0 or no speedup.

While Blue Waters achieves higher speedups in some cases than Lassen, both systems see speedups as large as 60×. These results only present part of the overall communication picture, however. There are still cases where the global communication strategy should be reduced to three-step communication, two-step communication, or standard communication. This differs based on the two machines and the specific test matrix, but tuning between the techniques results in the most optimal communication strategy. Tuning comes at the minimal cost of performing four different SpMBVs during setup of the SpMBV communicator. Speedups over standard communication when using tuning to use the fastest communication technique are presented in Figure 4.10.

In the top plot of Figure 4.10, Blue Waters benefits in 33% of the cases from including the nodal optimal multi-step communication strategy. These results are expected based on Figure 4.9 where the benefits of nodal optimal multi-step communication were less than ideal. In fact, for 20% of the cases on Blue Waters, standard communication is the most performant, consistent with matrices of low density. The matrices `Geo_1438` and `ldoor`, which have the smallest density of the test matrices (Table 3), see minimal benefits from the multi-step communication techniques as the number of processes is scaled up due to the minimal amount of data being communicated.

We expected to see two-step and nodal optimal communication perform the best on Lassen due to the faster inter-node communication for smaller sized messages (Figure 4.6) and the benefits of splitting messages across many processes on node (Figure 4.7). These expectations are consistent with the results presented in the bottom plot of Figure 4.10; most test matrices saw the best SpMBV performance with nodal optimal communication (44% of the cases). The remainder of the test cases were divided almost equally between two-step, three-step, and standard communication for which technique was the most performant (approximately 18% of the cases, each).

Table 4. Measured Percentage of Time Spent in Point-to-point Communication for the Tuned Multistep ("m-s") Compared against the Measured Percentage of Time Spent in Point-to-point Communication for Standard ("std") in a Single Iteration of ECG for Example 2.1 with Varying t and Processor Counts on Blue Waters and Lassen

(a) Blue Waters

procs →	256		512		1,024		2,048		4,096		8,192	
t	m-s	std	m-s	std	m-s	std	m-s	std	m-s	std	m-s	std
5	54.7	55.6	34.1	70.0	66.2	70.2	70.6	75.5	70.2	78.9	41.7	76.6
10	31.8	40.0	39.3	56.2	34.6	65.2	48.4	67.5	59.7	69.1	32.2	68.7
15	9.3	37.6	18.2	40.4	30.4	66.7	31.9	67.0	38.4	58.5	26.0	63.8
20	7.5	24.5	15.0	38.3	26.3	62.3	46.0	47.6	30.8	45.5	29.7	53.6

(b) Lassen

procs →	240		520		1,040		2,040		4,080	
t	m-s	std	m-s	std	m-s	std	m-s	std	m-s	std
5	49.4	84.3	43.6	88.0	66.6	88.4	82.2	87.2	82.7	93.1
10	26.8	87.2	33.7	84.9	52.1	84.7	58.4	89.3	78.3	91.3
15	9.9	81.2	26.4	85.7	33.6	86.1	55.3	90.0	61.2	89.2
20	12.7	74.2	17.8	79.9	29.6	81.7	39.0	80.7	57.2	79.5

Blue values correspond to values for which the percentage of time decreased by 5–20%, green values are a decrease of 20–40%, and yellow values decreased by more than 40% from the original percentage.

Table 4 shows the benefits of using the tuned point-to-point communication over the standard communication in a single iteration of ECG for Example 2.1. Tuned communication reduces the percentage of time spent in point-to-point communication independent of system, though the performance benefits are typically best when more data are being communicated ($t = 20$ in Table 4(a) and Table 4(b)). For Blue Waters, the new communication technique results in point-to-point communication taking 20–40% less of the total time compared to the percentage of time when using standard communication (corresponding to the blue highlighted values in Table 4(a)). Performance benefits are much greater on Lassen where the tuned communication results in a decrease of more than 40% of the total iteration time compared to an iteration time with standard communication in most cases.

5 CONCLUSIONS

The enlarged ECG is an efficient method for solving large systems of equations designed to reduce the collective communication bottlenecks of the classical CG method. Within ECG, block vector updates replace the single vector updates of CG, thereby reducing the overall number of iterations required for convergence and hence the overall amount of collective communication.

In this article, we performed a performance study and analysis of the effects of block vectors on the balance of collective communication, point-to-point communication, and computation within the iterations of ECG. We noted the increased volume of data communicated and its disproportionate affects on the performance of the point-to-point communication; the communication bottleneck of ECG shifted to be the point-to-point communication within the SpMBV kernel. To address the new SpMBV bottleneck, we designed an optimal multi-step communication technique that builds on existing node-aware communication techniques and improves them for the emerging supercomputer architectures with greater numbers of processes per node and faster inter-node networks.

Overall, this article provides a comprehensive study of the performance of ECG in a distributed parallel environment, and introduces a novel point-to-point multi-step communication technique that provides consistent speedup over standard communication techniques independent of machine. Notably, the novel communication technique naturally extends to any iterative method in which there is a sparse matrix-block vector product kernel, such as block Krylov methods. Future work includes profiling and improving the performance of ECG on hybrid supercomputer architectures with computation offloaded to graphics processing units.

The software used to generate the results in this article is freely available in RAPtor [10].

REFERENCES

[1] Tarun Agarwal, Amit Sharma, A. Laxmikant, and Laxmikant V. Kalé. 2006. Topology-aware task mapping for reducing communication contention on large parallel machines. In *Proceedings of the 20th IEEE International Parallel & Distributed Processing Symposium*. IEEE, 10.

[2] Hasan Metin Aktulga, Chao Yang, Esmond G. Ng, Pieter Maris, and James P. Vary. 2014. Improving the scalability of a symmetric iterative eigensolver for multi-core platforms. *Concurr. Comput.: Pract. Exp.* 26, 16 (2014), 2631–2651.

[3] Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, and Chris Scheiman. 1997. LogGP: Incorporating long messages into the LogP model for parallel computation. *J. Parallel Distrib. Comput.* 44, 1 (1997), 71–79. DOI: https://doi.org/10.1006/jpdc.1997.1346

[4] Robert Anderson, Julian Andrej, Andrew Barker, Jamie Bramwell, Jean-Sylvain Camier, Jakub Cerveny, Veselin Dobrev, Yohann Dudouit, Aaron Fisher, Tzanio Kolev, Will Pazner, Mark Stowell, Vladimir Tomov, Ido Akkerman, Johann Dahm, David Medina, and Stefano Zampini. 2021. MFEM: A modular finite element methods library. *Comput. Math. Appl.* 81 (2021), 42–74. DOI: https://doi.org/10.1016/j.camwa.2020.06.009

[5] Allison H. Baker, Martin Schulz, and Ulrike M. Yang. 2010. On the performance of an algebraic multigrid solver on multicore clusters. In *International Conference on High Performance Computing for Computational Science*. Springer, 102–115.

[6] Satish Balay, Shrirang Abhyankar, Mark Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Alp Dener, Victor Eijkhout, W. Gropp, et al. 2019. *PETSc Users Manual*.

[7] Amanda Bienz, William Gropp, and Luke Olson. 2019. Node aware sparse matrix–vector multiplication. *J. Parallel Distrib. Comput.* 130 (08 2019), 166–178. DOI: https://doi.org/10.1016/j.jpdc.2019.03.016

[8] Amanda Bienz, William D. Gropp, and Luke N. Olson. 2018. Improving performance models for irregular point-to-point communication. In *Proceedings of the 25th European MPI Users' Group Meeting (EuroMPI'18)*. Association for Computing Machinery, New York, NY, 8 pages. https://doi.org/10.1145/3236367.3236368

[9] Amanda Bienz, William D. Gropp, and Luke N. Olson. 2020. Reducing communication in algebraic multigrid with multi-step node aware communication. *Int. J. High Perf. Comput. Appl.* 34, 5 (June 2020), 547–561. DOI: https://doi.org/10.1177/1094342020925535

[10] Amanda Bienz and Luke N. Olson. 2017. RAPtor: Parallel Algebraic Multigrid v0.1, Release 0.1. Retrieved from https://github.com/raptor-library/raptor.

[11] Brett Bode, Michelle Butler, Thom Dunning, Torsten Hoefler, William Kramer, William Gropp, and Wen-mei Hwu. 2013. The blue waters super-system for super-science. In *Contemporary High Performance Computing*. Chapman & Hall/CRC, 339–366. https://www.taylorfrancis.com/books/e/9781466568358.

[12] Erin Carson, Nicholas Knight, and James Demmel. 2013. Avoiding communication in nonsymmetric Lanczos-based Krylov subspace methods. *SIAM J. Sci. Comput.* 35, 5 (January 2013), S42–S61. DOI: https://doi.org/10.1137/120881191. arXiv:https://doi.org/10.1137/120881191

[13] Ümit V. Çatalyürek and Cevdet Aykanat. 1999. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. Parallel Distrib. Syst.* 10, 7 (1999), 673–693. DOI: https://doi.org/10.1109/71.780863

[14] Ümit V. Çatalyürek, Cevdet Aykanat, and Bora Uçar. 2010. On two-dimensional sparse matrix partitioning: Models, methods, and a recipe. *SIAM J. Sci. Comput.* 32, 2 (2010), 656–683. DOI: https://doi.org/10.1137/080737770

[15] A. T. Chronopoulos and C. W. Gear. 1989. s-step iterative methods for symmetric linear systems. *J. Comput. Appl. Math.* 25, 2 (February 1989), 153–168. DOI: https://doi.org/10.1016/0377-0427(89)90045-9

[16] S. Cools and W. Vanroose. 2017. The communication-hiding pipelined BiCGstab method for the parallel solution of large unsymmetric linear systems. *Parallel Comput.* 65 (July 2017), 1–20. DOI: https://doi.org/10.1016/j.parco.2017.04.005

[17] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. 1993. LogP: Towards a realistic model of parallel computation. *SIGPLAN Not.* 28, 7 (July 1993), 1–12. DOI: https://doi.org/10.1145/173284.155333

[18] Timothy A. Davis and Yifan Hu. 2011. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.* 38, 1, Article 1 (December 2011), 25 pages. DOI: https://doi.org/10.1145/2049662.2049663

[19] P. Ghysels and W. Vanroose. 2014. Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm. *Parallel Comput.* 40, 7 (July 2014), 224–238. DOI: https://doi.org/10.1016/j.parco.2013.06.001

[20] Laura Grigori, Sophie Moufawad, and Frederic Nataf. 2016. Enlarged Krylov subspace conjugate gradient methods for reducing communication. *SIAM J. Matrix Anal. Appl.* 37, 2 (January 2016), 744–773. arXiv:https://doi.org/10.1137/140989492

[21] Laura Grigori and Olivier Tissot. 2019. Scalable linear solvers based on enlarged Krylov subspaces with dynamic reduction of search directions. *SIAM J. Sci. Comput.* 41, 5 (January 2019), C522–C547. arXiv:https://doi.org/10.1137/18M1196285

[22] William Gropp, Luke N. Olson, and Philipp Samfass. 2016. Modeling MPI communication performance on SMP nodes. In *Proceedings of the 23rd European MPI Users' Group Meeting (EuroMPI'16)*. ACM Press, New York, NY, 41–50. DOI: https://doi.org/10.1145/2966884.2966919

[23] W. A. Hanson. 2020. The CORAL supercomputer systems. *IBM J. Res. Dev.* 64, 3/4 (2020), 1:1–1:10.

[24] Bruce Hendrickson and Tamara G. Kolda. 200. Graph partitioning models for parallel computing. *Parallel Comput.* 26, 12 (200), 1519–1534. DOI: https://doi.org/10.1016/S0167-8191(00)00048-X

[25] William Kramer, Michelle Butler, Gregory Bauer, Kalyana Chadalavada, and Celso Mendes. 2015. Blue waters parallel I/O storage sub-system. In *High Performance Parallel I/O*, Prabhat and Quincey Koziol (Eds.). CRC Publications, Taylor & Francis Group, 17–32.

[26] Hoemmen M. 2010. *Communication-Avoiding Krylov Subspace Methods*. Ph.D. Dissertation. University of California, Berkeley.

[27] Tania Malik, Vladimir Rychkov, and Alexey Lastovetsky. 2016. Network-aware optimization of communications for parallel matrix multiplication on hierarchical HPC platforms. *Concurr. Comput.: Pract. Exp.* 28, 3 (2016), 802–821.

[28] Lois Curfman McInnes, Barry Smith, Hong Zhang, and Richard Tran Mills. 2014. Hierarchical Krylov and nested Krylov methods for extreme-scale computing. *Parallel Comput.* 40, 1 (January 2014), 17–31. DOI: https://doi.org/10.1016/j.parco.2013.10.001

[29] Marghoob Mohiyuddin, Mark Hoemmen, James Demmel, and Katherine Yelick. 2009. Minimizing communication in sparse matrix solvers. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC'09)*. Association for Computing Machinery, New York, NY, Article 36, 12 pages. DOI: https://doi.org/10.1145/1654059.1654096

[30] Sophie M. Moufawad. 2020. s-step enlarged krylov subspace conjugate gradient methods. *SIAM J. Sci. Comput.* 42, 1 (January 2020), A187–A219. arXiv:https://doi.org/10.1137/18M1182528

[31] Dianne P. O'Leary. 1980. The block conjugate gradient algorithm and related methods. *Lin. Algebr. Appl.* 29 (February 1980), 293–322. DOI: https://doi.org/10.1016/0024-3795(80)90247-5

[32] Brian A. Page and Peter M. Kogge. 2018. Scalability of hybrid sparse matrix dense vector (spmv) multiplication. In *Proceedings of the International Conference on High Performance Computing & Simulation (HPCS'18)*. IEEE, 406–414.

[33] Jesper Larsson Träff. 2002. Implementing the MPI process topology mechanism. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'02)*. IEEE, Los Alamitos, CA, 1–14.

[34] Brendan Vastenhouw and Rob H. Bisseling. 2005. A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. *SIAM Rev.* 47, 1 (2005), 67–95. DOI: https://doi.org/10.1137/S0036144502409019

[35] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (2009), 65–76.

Received 10 March 2022; revised 15 December 2022; accepted 10 January 2023