

Characterizing the Performance of Node-Aware Strategies for Irregular Point-to-Point Communication on Heterogeneous Architectures

Shelby Lockhart¹, Amanda Bienz¹, William D. Gropp¹, Luke N. Olson¹

^a*University of Illinois at Urbana-Champaign, Department of Computer Science, Urbana, 61801, IL, USA*

^b*University of New Mexico, Department of Computer Science, Albuquerque, 87131, NM, USA*

Abstract

Supercomputer architectures are trending toward higher computational throughput due to the inclusion of heterogeneous compute nodes. These multi-GPU nodes increase on-node computational efficiency, while also increasing the amount of data to be communicated and the number of potential data flow paths. In this work, we characterize the performance of irregular point-to-point communication with MPI on heterogeneous compute environments through performance modeling, demonstrating the limitations of standard communication strategies for both device-aware and staging-through-host communication techniques. Presented models suggest staging communicated data through host processes then using node-aware communication strategies for high inter-node message counts. Notably, the models also predict that node-aware communication utilizing all available CPU cores to communicate inter-node data leads to the most performant strategy when communicating with a high number of nodes. Model validation is provided via a case study of irregular point-to-point communication patterns in distributed sparse matrix-vector products. Importantly, we include a discussion on the implications model predictions have on communication strategy design for emerging supercomputer architectures.

Keywords: performance modeling, GPU, data movement, CUDA-aware, GPUDirect, MPI, parallel, communication, sparse matrix

1. Introduction

Modern parallel supercomputers exhibit increasingly higher computational throughput due to the inclusion of multiple GPUs per node — see ?? . These GPUs operate on much higher data volumes concurrently than previous CPU-only clusters, yet the issue of communication bottlenecks persists and is exacerbated in a multi-node–multi-GPU setting. While the high computational intensity of modern supercomputers is driving a new era of applications, the volume of data communicated between compute units has also increased, creating new obstacles for data movement performance.

In this paper, we focus on irregular point-to-point communication, which generates performance bottlenecks in parallel iterative solvers and graph algorithms due to the prevalence of sparse matrix operations and unstructured mesh computations [? ?]. We aim to characterize the performance of various irregular point-to-point communication strategies using MPI within heterogeneous compute environments via performance modeling, which suggests the extension of node-aware communication strategies for inter-CPU communication (discussed in ??) onto heterogeneous architectures.

Node-aware communication schemes utilize the relative location of communicating processes and exchange costly data flow paths for lower cost alternatives [?]. There has been extensive research on the development of these communication schemes for inter-CPU communication [? ? ? ?], as

well as, initial results demonstrating potential benefits of staging GPU data through host processes before exchanging inter-node messages when message counts are high [?]. However, there has not been a study on the potential benefits of using node-aware communication techniques for inter-GPU communication on modern heterogeneous architectures. In this work, we provide an overview of existing node-aware communication techniques for inter-CPU communication and extend them to inter-GPU communication. While there are many potential paths for data movement on heterogeneous architectures, we consider the communication paths available via the MPI API and only consider device specific optimizations, such as utilizing CUDA Multi-Process Service (MPS) to allow multiple MPI ranks to copy data from a single GPU, as a comparison.

In ?? , we present modeling parameters for all potential data flow paths between CPUs and GPUs, which are then used within performance models to predict the cost of various node-aware communication schemes when implemented on heterogeneous architectures in ?? . Models are first validated via comparison against the performance of communication within a sparse matrix-vector product, then further modeling results suggest that for large message counts, optimal performance is achieved when GPU data is staged through a host process and split across multiple processes before communicating through the network.

Furthermore, ?? provides a study of the techniques modeled in ?? when applied to the irregular point-to-point communication patterns in distributed sparse matrix-vector multiplication (SpMV) on heterogeneous architectures, further validating model predictions. Finally, ?? provides a discussion on the im-

*Corresponding author

56 plications model predictions and benchmark results have on the 89
 57 future of communication strategy design for emerging super- 90
 58 computer architectures, alongside a summary of the presented 91
 59 results.
 60

The following provides a summary of paper contributions:
 61

1. performance models for node-aware communication on 94
 heterogeneous architectures — ???;
2. performance predictions for common irregular point-to- 98
 point communication scenarios using the developed mod- 99
 els — ??;
3. benchmarks of irregular point-to-point communication pat- 100
 terns found within distributed sparse matrix-vector mul- 102
 tiplication — ??; and 103
4. remarks on future communication design for emerging 104
 supercomputer architectures — ??.

71 2. Background

72 2.1. Modern Architectures

73 Many current large-scale supercomputers consist of hetero- 111
 74 geneous nodes containing multiple GPUs connected to a sin- 112
 75 gle CPU per socket with two sockets per node. In the case 113

Summit. However, these compute nodes will consist of a single 92
 59 socket housing an AMD EPYC CPU connected to four AMD 93
 Instinct 250X GPUs via AMD Infinity Fabric with a Slingshot 94
 network. Additionally, the National Center for Supercomputing 95
 Applications (NCSA) will boast more expansive compute 96
 nodes consisting of four to eight AMD A100 GPUs connected 97
 to a dual AMD 64-core 2.55 GHz Milan processor per compute 98
 node in their upcoming system Delta².
 99

100 Both current and future supercomputers boast heterogeneous 101
 59 architectures with multiple paths for data movement between 102
 60 two GPUs. Two connected GPUs either exchange data directly 103
 61 or stage through the host CPU by first copying data to CPU 104
 62 memory, then transferring data from the local CPU to the host 105
 63 CPU of the receiving GPU, and finally copying received data 106
 64 to the destination GPU. The process of staging data through 107
 65 the host CPU can be used for any set of communicating GPUs 108
 66 independent of their relative locations. However, device-aware 109
 67 data movement paradigms, such as CUDA-aware MPI using 110
 68 GPUDirect [?] on Lassen, remove the necessity of copy- 111
 69 ing data to the host CPU and allow data retrieval directly from 112
 70 device memory, even in the case of inter-node data transfers. 113
 The addition of device-aware technologies increases the num- 114
 71 ber of potential data movement paths necessitating the use of 115
 72 robust performance modeling to determine communication 116
 73 bottlenecks, as well as, design optimal communication 117
 74 strategies.
 118

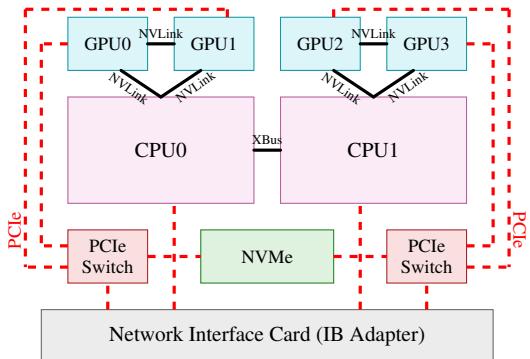


Figure 2.1: Lassen compute node.

75 of the Lassen supercomputer, each socket consists of a single 126
 76 IBM Power9 CPU connected to two NVIDIA V100 GPUs [?]
 77] (see ??), while the Summit supercomputer has a single IBM 127
 78 Power9 CPU connected to three NVIDIA V100 GPUs [?].
 79 For both machines, each CPU has 20 available cores, CPUs 128
 80 and GPUs are connected via NVLink. Furthermore, CPUs are 129
 81 connected directly to the Infiniband (IB) adapter through PCIe 130
 82 lanes, while GPUs are connected to the IB adapter via a PCIe 131
 83 switch system connecting the GPUs to NVMe and the adapter.
 84 Nodes are connected via Mellanox EDR 100G InfiniBand in a 132
 85 non-blocking fat tree topology. Upcoming Department of En- 133
 86 ergy exascale machines, Frontier [?] and El Capitan¹, will 134
 87 have nodes with a similar structure to those found in Lassen and 135

114 2.2. Modeling Data Movement

115 Throughout this paper, we rely on the *max-rate* model as the 116
 59 basis for communication modeling [?]. The max-rate model 117
 60 is an improvement to the standard postal model of communi- 118
 61 cation, accounting for injection limits into the network. The 119
 62 traditional postal model estimates the cost of communicating 120
 63 a message between two symmetric multiprocessing (SMP) nodes 121
 64 as
 122

$$T = \alpha + \beta \cdot s \quad (2.1)$$

123 where α is the latency, β is the per-byte transfer cost, and s is 124
 59 the number of bytes being communicated. The max-rate model 125
 60 adds parameters for injection-bandwidth limits and the number 126
 61 of actively communicating processes, resulting in the following 127
 62 time estimation,
 128

$$T = \alpha \cdot m + \max \left(\frac{\text{ppn} \cdot s}{R_N}, \frac{s}{R_b} \right) \quad (2.2)$$

129 where α is again the latency, m is the maximum number of mes- 130
 63 sages sent by a single process on a given node, s is the maxi- 131
 64 mum number of bytes sent by a single process on a given SMP, 132
 65 ppn is the number of processes per node, R_N is the rate at which 133
 66 a network interface card (NIC) can inject data into the network, 134
 67 and R_b is the rate at which a process can transport data. When 135
 68 $\text{ppn} \cdot R_b < R_N$, this model reduces to the postal model.
 136

137 For inter-CPU communication, additional improvements are 138
 69 available to the max-rate model within the context of irregular 140
 70

¹<https://www.llnl.gov/news/el-capitan-testbed-systems-rank-among-top-200-worlds-most-powerful-computers>

²<https://www.ncsa.illinois.edu/research/project-highlights/delta>

136 point-to-point communication. Additional hardware and soft-¹⁷⁹
 137 ware overhead penalties are represented in the LogP model[?],¹⁸⁰
 138 which is extended to include long message costs in the LogGP¹⁸¹
 139 model[?]. Additionally, models for queue search times and
 140 network contention have been shown to be important for ac-¹⁸²
 141 curately predicting performance of point-to-point communica-¹⁸³
 142 tion [?]. These models provide penalty parameters which have
 143 been shown to increase the accuracy of standard communica-¹⁸⁴
 144 tion models, as well as, motivate design decisions within the
 145 context of node-aware communication techniques. Node-aware
 146 communication strategies are discussed in more detail in ??.

147 The max-rate model also applies to inter-GPU communica-¹⁸⁵
 148 tion [?]. Here, the noted difficulty in reaching injection
 149 bandwidth limits with inter-GPU communication is due to the
 150 low number of communicating GPUs per node. Additionally,
 151 for large message counts, performance benefits are observed [?]
 152 when staging communication between GPUs through host
 153 CPUs. In ??, the models for inter-GPU irregular point-to-point
 154 communication are presented.

155 2.3. Node-Aware Communication

156 Node-aware communication techniques for irregular point-¹⁸⁶
 157 to-point communication have been designed within the con-
 158 text of sparse matrix-vector multiplication (SpMV) and sparse
 159 matrix-matrix multiplication (SpMM) [?]. Due to their low
 160 computational requirements, sparse matrix operations often in-
 161 cur a large communication overhead when performed in a par-
 162 allel distributed setting, highlighting the limitations of standard
 163 communication practices.

164 There are two redundancies that occur with standard com-¹⁸⁷
 165 munication, namely: a message redundancy and a data redun-¹⁸⁸
 166 dancy, illustrated in ???. First, each node injects many mes-¹⁸⁹
 167 sages into the network; for example, some nodes send multiple
 168 messages to a single process on the destination node creating¹⁹⁰
 169 message redundancy. Second, processes send their local data to¹⁹¹
 170 every destination process, independent of whether they had sent¹⁹²
 171 the same local data to another process on the same node; hence¹⁹³
 172 a redundancy in data being sent through the network. The ma-

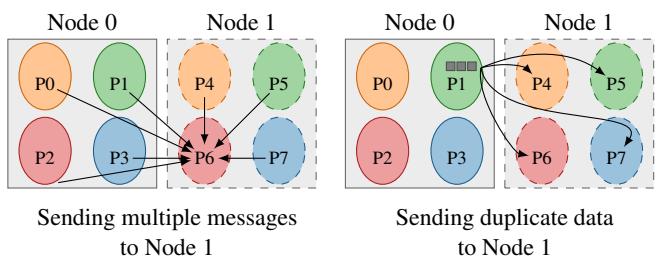


Figure 2.2: Standard communication. On the left, Node 0 injects multiple mes-²⁰⁰
 201 sages into the network, all to P6 on Node 1. On the right, P1 sends all high-
 202 highlighted data to multiple processes on Node 1, leading to redundant messages.
 203

172 jority of node-aware communication work has been done within²⁰⁴
 173 the context of CPU to CPU communication with a subset of this
 174 work later replicated for GPU to GPU communication. There
 175 are three types of node-aware communication for CPU to CPU
 176 communication, each eliminating all or some of the redundan-²⁰⁵
 177 cies introduced by standard communication.²⁰⁶

2.3.1. 3-Step

3-Step node-aware communication, first introduced in [?]
 1, eliminates both redundancies in standard communication by
 gathering all necessary data to be sent off-node in a single buffer.
 Pairing all processes with a receiving process on distinct nodes
 ensures efficiency of the method by making sure every process
 remains active throughout the communication scheme. First,

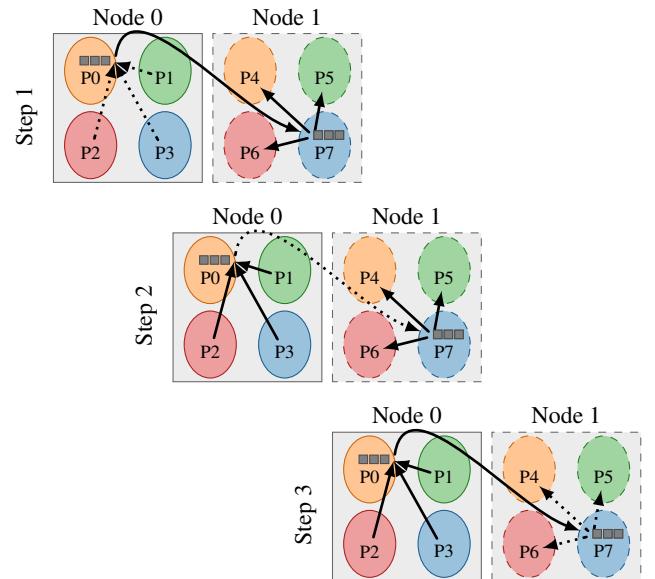


Figure 2.3: 3-Step node-aware. In Step 1, all data on Node 0 that needs to be sent to Node 1 is collected in a buffer on P0, the process paired to send and receive from Node 1. In Step 2, P0 sends this buffer from Node 0 to P7, the receiving process on Node 1. In Step 3, P7 redistributes the data to the correct receiving processes on Node 1. Dotted lines,, depict the action performed in each step.

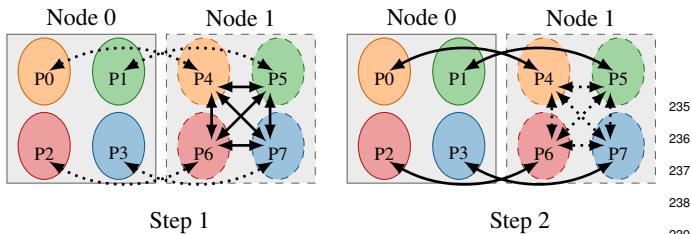
all messages sent to a separate node are gathered in a buffer by the single process associated with the node. Secondly, this process sends the data buffer to the paired process on the receiving node. Thirdly, the paired process on the receiving node redistributes the data to the correct destination processes on-node. An example of these steps is outlined in ??.

As noted in [?], the method can be extended to include further breakdown of data exchanges to include intra-socket data communication before the intra-node communication phase. However, we expect minimal performance benefits in extending the communication strategy throughout the entire node hierarchy for CPU to CPU communication. Instead, this strategy is adopted for GPU to GPU communication in [?], where the full hierarchy of the node is utilized to achieve optimal performance due to the fast data transfer rates of socket-level GPU interconnects on Summit [?]. In addition, recent work on utilizing neighborhood collectives in conjunction with the 3-Step node-aware communication strategy further reduces communication overhead in sparse solvers [?].

2.3.2. 2-Step

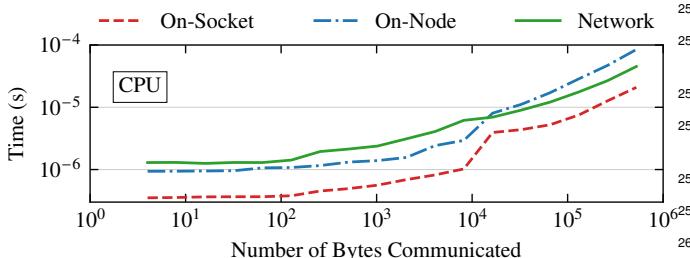
When communicating high data volumes between nodes, 3-Step communication can see limitations as the single buffer

208 communicating data grows extremely large, thus motivating a
 209 2-Step node-aware technique as in [?]. The 2-Step technique
 210 eliminates the redundancy of sending duplicate data through the
 211 network, but does not reduce the redundancy of multiple mes-
 212 sages being sent between nodes. In 2-Step, *each* process ex-
 213 changes information needed by the receiving node with their
 214 paired process directly, followed by the receiving node redis-
 215 tributing the messages on-node, as shown in ???. Overall, the
 216 total number of bytes communicated with 3-Step and 2-Step
 217 communication techniques is the same, but the number and size
 of inter-node messages differs.



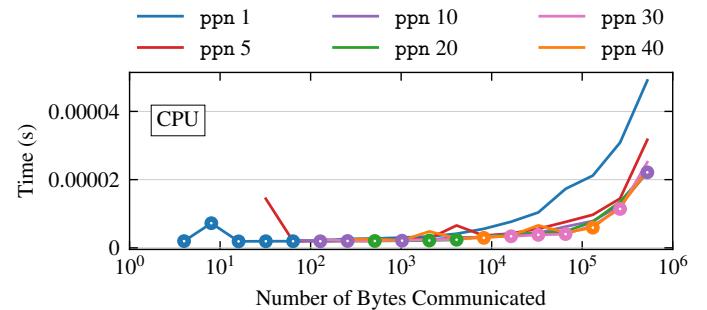
240
 241 Figure 2.4: 2-Step node-aware. Each process on Node 0 is paired with a re-
 242 ceiving process on Node 1. In Step 1, each process on Node 0 sends the data
 243 needed by any process on Node 1 to its paired process on Node 1. Here, P0 is sending to P4, P1 to P5, P2 to P6, and P3 to P7. In Step 2, each process on Node 1 redistributes the data received from Node 0 to the destination on Node 1. Dotted lines,, depict the action performed in each step.
 244

245
 246
 247 2.3.3. *Split*
 248 3-Step and 2-Step communication show a drastic difference in performance in communicating on-node versus inter-node messages [?], particularly on more traditional networks, e.g., the now retired BlueWaters system. Yet this is not always the case for more recent interconnects, such as on Lassen, which shows varying performance for inter-node versus intra-node communication depending on the amount of data being communicated — see ???. In addition to network communication being



254
 255 Figure 2.5: The amount of time required to send data between two processes distinguishing between where the two processes are physically located on the same socket, the same node and separate sockets, and separate nodes requiring network communication.
 256

257 faster than on-node communication for large message sizes, the CPUs used in current supercomputers have high numbers of cores (for example, the IBM Power9 has 40 available cores on Lassen, and the Delta system has 64 available cores on each AMD Milan processor), making splitting large data volumes across all available cores more performant than when the entire data volume is sent by a single process — see ???.
 258



259 Figure 2.6: The amount of time required to send various amounts of data between two distinct nodes when splitting the data across varying numbers of processes per node (ppn). Minimum times circled.
 260

261 Split communication, as introduced in [?], addresses the variable performance of 3-Step and 2-Step node-aware communication on modern supercomputers. This communication technique balances the performance of multi-step communication by splitting the communicated inter-node data into messages of size `message_cap`, followed by a distribution across some number of on-node processes before being injected into the network. Pseudo-code of the setup is provided in ?? with communication parameters defined in ???. Here, we detail the operations summarized in ??.

262 **Line 8** The algorithm begins by splitting inter-node messages by their origin node (on-node or off-node).

263 **Line 9** A *local* communicator is created for exchanging all messages with origin on-node.

264 **Line 10** All messages with origin off-node are split into lists according to their origin node.

265 **Lines 11** Parameters, such as the number of nodes from which this node receives, the *maximum* amount of data being received from a single other node, and the *total* amount of data being received from any node by this node, are determined.

266 **Lines 12–17** In this block of the algorithm, the appropriate `message_cap` is determined.

267 **Lines 12–13** First, the maximum amount of data being received from any node is checked to determine if it is smaller than the user provided `message_cap`. When this occurs, every node's data should be sent in a single message.

268 **Lines 14–17** Otherwise, if the total inter-node data volume being communicated divided by the provided `message_cap` is greater than the active number of processes per node, then the `message_cap` is increased to be the total inter-node data volume divided by the number of on-node processes.

269 **Line 18** On-node processes are assigned inter-node messages to receive in descending order of size, starting with local rank 0. Inter-node messages to be sent are assigned in

Algorithm 1: Setup for split communication.

```

1 Input: l_recv [list of messages to receive]
2         comm [world communicator]
3         message_cap [user-defined message cap size]
4 Output: local_comm [on-node subcommunicator]
5         local_Rcomm [redistribution subcommunicator]
6         global_comm [off-node subcommunicator]
7         local_Scomm [on-node sending subcommunicator]
8 Split messages by origin, off-node and on-node
9 local_comm  $\leftarrow$  Create on-node communicator
10 Split off-node messages by node
11 Set parameters in ??
12 if max_IN_recv_size < message_cap
13   Conglomerate all inter-node receives by node
14 else
15   if  $\frac{\text{total\_IN\_recv\_vol}}{\text{message\_cap}} > \text{PPN} \ \& \ \text{num\_IN\_nodes} < \text{PPN}$ 
16     Set message_cap =  $\lceil \frac{\text{total\_IN\_recv\_vol}}{\text{PPN}} \rceil$ 
17   Split inter-node receives to max size message_cap
18 Set on-node receive order (descending by size)
19 local_Rcomm  $\leftarrow$  Create redistribution communicator (receive)
20 global_comm  $\leftarrow$  Create inter-node communicator
21 local_Scomm  $\leftarrow$  Create redistribution communicator (send)

```

Parameter	Description
message_cap	maximum message size when splitting communicated inter-nodal data volumes
total_IN_recv_vol	total amount of data being received by this node from any other node in Bytes
max_IN_recv_size	maximum amount of data being received from a single other node in Bytes
num_IN_nodes	number of nodes from which this node is receiving any messages
PPN	processes per node

Table 1: Split communication parameters.

Algorithm 2: Split communication.

- 1 Perform local_comm communication.
 - 2 Perform local_Scomm data redistribution.
 - 3 Perform global_comm inter-node communication.
 - 4 Perform local_Rcomm data redistribution.
-

inter-node data to be sent by this node to the local processes responsible for sending the inter-node messages.

?? provides the steps for performing Split communication once the relevant communicators have been created. While ?? and the four stages of node-aware communication in ?? would ultimately be hidden from the user, ?? demonstrates the flexibility of the communication technique. Depending on the computation in which Split communication is being used, ?????? of ?? can be overlapped with various pieces of the computation — details of performance gains when overlapping computation with node-aware communication can be found in [?]. Furthermore, in [?], the inter-node message size cutoff is determined by the rendezvous protocol based on communication modeling for Lassen, but it is observed that the message size cutoff can be determined via tuning or any other chosen criteria. Similarly, we use a message size cutoff of three in ?? to demonstrate the multi-step technique when communicating between two nodes with four processes each.

272 the reverse order starting with local rank PPN-1. This in 289 combination with the message splitting ensures that all 290 processes are active during communication. 291

275 **Line 19** A *local* communicator is created for redistributing all 292 received inter-node data to its final destination processes 293 on-node. 294

278 **Line 20** A *global* communicator is created for exchanging inter-296 node messages based on send and receive message as-297 signment in **Line 18**. 298

281 **Line 21** A *local* communicator is created for redistributing all 300

Splitting communication eliminates the data redundancy from

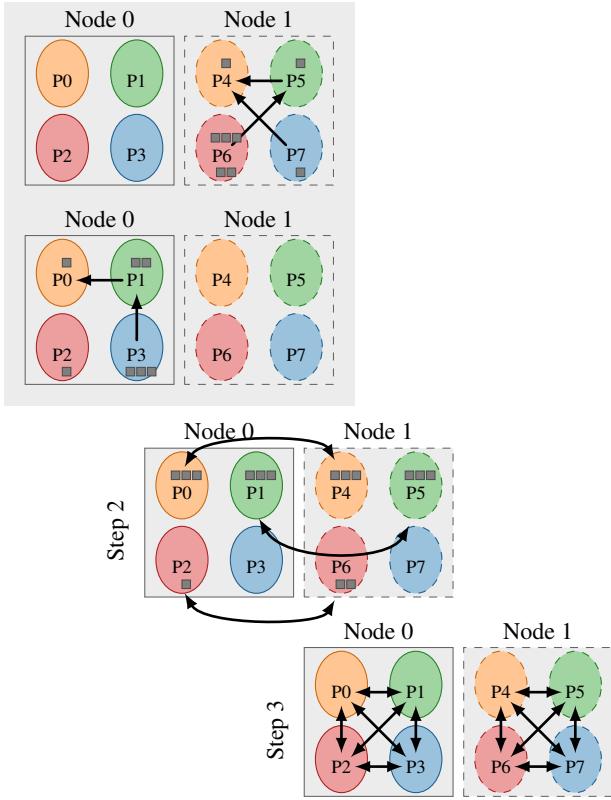


Figure 2.7: Split node-aware. Here, data is communicated between two distinct nodes: Node 0 and Node 1, each with 4 local processes, denoted $P\#$. In Step 1, each node conglomerates small messages to be sent off-node, splits messages based on a message cap of 3, and retains messages approximately the size of the message cap (?? ??). In Step 2, the buffers prepared in Step 1 are sent to their destination node, specifically to the paired process on that node (?? ??). For Step 3, all processes redistribute their received data to the correct destination processes on-node (?? ??).

standard communication, but does so with varying numbers of inter-node messages (as determined by the total data volume being sent to another distinct node). Within the context of a sparse matrix-block vector multiplication, this scheme yields up to 60 \times speedup over standard communication techniques. The goal of this work is to consider approaches similar to the split communication strategy within the context of heterogeneous architectures.

2.4. Distributed Sparse Matrix-Vector Multiplication

Throughout the paper, we utilize the irregular point-to-point communication patterns induced by sparse matrix-vector multiplication (SpMV) to test the performance potential of node-aware communication strategies within the context of GPU to GPU communication, as well as provide further model validation. A SpMV, defined as

$$A \cdot v \rightarrow w \quad (2.3)$$

with $A \in \mathbb{R}^{m \times n}$ and $v, w \in \mathbb{R}^n$, is a common kernel in sparse iterative methods. Distributed SpMVs performed on GPUs currently face many obstacles in performance including computational inefficiencies of the local SpMV portion on each GPU,

packing and unpacking communication buffers, strategically overlapping computation and communication, etc. [?? ?]. There are multiple potential solutions to these problems, many of which are still currently being researched [?? ?].

Because the presented work focuses on general communication strategies, we do not attempt to optimize these portions of the distributed SpMV. Instead our performance tests focus solely on benchmarking the irregular point-to-point communication that occurs in the distributed kernel, characterizing the performance of multiple communication strategies for various communicated data volumes and message counts on a heterogeneous architecture.

2.4.1. Testing Setup

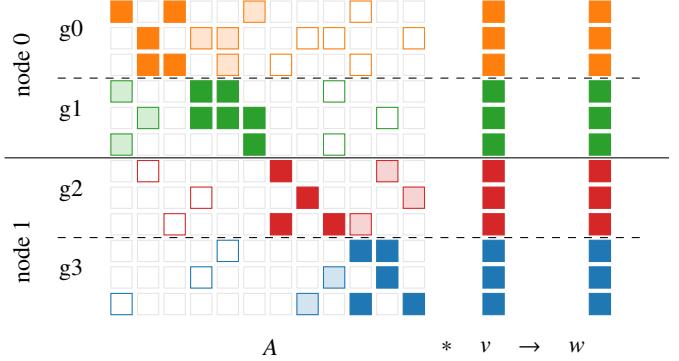


Figure 2.8: Partitioning of a SpMV, $A \cdot v \rightarrow w$, with $n = 12$. Matrix A and vectors v and w are partitioned across two nodes, four GPUs (g_0, g_1, g_2, g_3). Solid blocks, \blacksquare , represent the portion of the SpMV requiring on-GPU values from v . Shaded blocks, \blacksquare , require on-node but off-GPU communication of values from v . Outlined blocks, \square , require values of v from GPUs off-node.

All performance tests presented in ??? correspond to a distributed SpMV with A , v , and w partitioned row-wise across g GPUs with contiguous rows stored on each GPU (see ??). In addition, the rows of A on each GPU are presumed to be split into two blocks, namely on-GPU and off-GPU. The on-GPU block is the diagonal block of columns corresponding to the on-GPU portion of rows in v and w , and the off-GPU block contains the matrix A 's nonzero values correlating to non-local rows of v and w stored off-GPU. This splitting is common practice, as it differentiates between the portions of a SpMV that require communication, as well as making the distributed kernel a perfect case study for node-aware communication performance on heterogeneous architectures. Because our key goal is to characterize irregular point-to-point communication performance independent of the distributed operation in which it is included, all presented benchmarks throughout the paper focus on the communication patterns induced by the distributed SpMV and not the computational aspects of the operation. We would like to note that within the context of a distributed SpMV, optimal performance depends on some combination of communication and computation overlap. However, optimizing the entire distributed SpMV operation lies outside the scope of this paper, thus timings for the computational portion and on-device kernel details are excluded.

3. Modeling Parameters for Communication

When data is moved between two GPUs on separate nodes using MPI, the data can be moved in one of two ways:

Device-aware: data is sent directly from the sending GPU through the NIC and the network to the receiving GPU without being copied to the host CPU; and

Staged-through-host: data is copied to the host CPU before being sent through the NIC and the network to the receiving GPU's host CPU then copied to the receiving GPU.

Because both of these involve moving data through the GPU and possibly the CPU, it is important to consider the cost of transmitting data through all possible data flow paths involving the CPU or GPU.

Throughout this section and the remainder of the paper, results are presented for the Spectrum MPI implementation on Lassen [?]. In [?], it is shown that Lassen and Summit [?] demonstrate similar performance using Spectrum MPI (there, the MPI implementation is optimized for use on the two DOE machines), therefore results for a single machine are provided. Moreover, each of the presented model parameters is the result of ping-pong and node-pong timings collected through Bench-⁴⁰⁶ Press³, a node architecture-aware library used for benchmarking data movement performance on large-scale systems. The ping-pong and node-pong tests are performed for 1000 iterations and averaged; each model parameter is then given by a linear least-squares fit to the collected data.⁴¹⁰

We use the postal model presented in ?? to model the time required for sending a single message between two CPUs or two GPUs, with the measured parameters for Lassen presented in ???. The α and β parameters are separated based on where the two processes are physically located with respect to one another, namely on the same socket, on different sockets but the same node, or separate nodes. In addition, the parameters are split further based on messaging protocol:⁴¹¹

short fits in the envelope so the message is sent directly to the receiving process;

eager assumes adequate buffer space is already allocated by the receiving process; or

rendezvous requires the receiving process to allocate buffer space for the message before the data is sent.

The short protocol has been excluded from the GPU messaging parameter portion of ?? because this protocol is not used⁴¹³ in device-aware communication on Lassen. Furthermore, the⁴¹⁴ inter-GPU parameters are split further into whether GPUDi-⁴¹⁵ rect technologies were enabled (GDR) or disabled (No GDR),⁴¹⁶ demonstrating the benefits of utilizing GPUDirect technologies⁴¹⁷ for device-aware communication.⁴¹⁸

Because staging data through a host process requires copy-⁴¹⁹ ing data to the sending host CPU and from the receiving GPU's⁴²⁰

			on-socket	on-node	off-node
inter-CPU	Short	α	3.67e-07	9.25e-07	1.89e-06
		β	1.32e-10	1.19e-09	6.88e-10
	Eager	α	4.61e-07	1.17e-06	2.44e-06
		β	7.12e-11	2.18e-10	3.79e-10
inter-GPU GDR	Rend	α	3.15e-06	6.77e-06	7.76e-06
		β	3.40e-11	1.49e-10	7.97e-11
	Eager	α	1.87e-06	2.02e-05	8.95e-06
		β	5.79e-11	2.15e-10	1.72e-10
inter-GPU No GDR	Rend	α	1.82e-05	1.93e-05	1.10e-05
		β	1.46e-11	2.39e-11	1.72e-10
	Eager	α	4.15e-05	4.27e-05	4.56e-05
		β	6.08e-09	6.24e-09	6.03e-10
	Rend	α	5.54e-05	5.75e-05	8.72e-05
		β	8.96e-11	8.11e-11	7.96e-11

α [sec] β [sec/byte]

Table 2: Measured parameters for inter-CPU and inter-GPU communication (with and without GPUDirect enabled) on Lassen.

host process, measured parameters for `cudaMemcpyAsync` are included in ?? with distinction between whether the copy is using a single process or four processes to move data from the device. We assume that all data copies will occur on-socket, and we do not consider cases with more than four processes pulling data from a single GPU at a time as there was no observed benefit in splitting data copies further across multiple processes — see ??.

		HostToDevice	DeviceToHost
1 proc	α	1.30e-05	1.27e-05
	β	1.85e-11	1.96e-11
4 proc	α	1.52e-05	1.47e-05
	β	5.52e-10	1.50e-10
8 proc	α	3.10e-05	3.03e-05
	β	7.88e-11	6.21e-11
10 proc	α	3.85e-05	3.81e-05
	β	9.43e-11	1.05e-10

α [sec] β [sec/byte]

Table 3: Measured parameters for `cudaMemcpyAsync` on Lassen.

In addition to considering the postal model for inter-CPU and inter-GPU communication, the max-rate model presented in ?? is required for accurately predicting the performance of staging GPU data through a host process when using more than a single process per node. Therefore, the measured injection bandwidth limit for inter-CPU communication is presented in ???. The inter-GPU injection bandwidth limit is excluded, as these limits are not reached with the four available GPUs per node on Lassen. Using the measured modeling parameters, we now model the performance of various communication strategies based

³<https://github.com/bienz2/BenchPress>

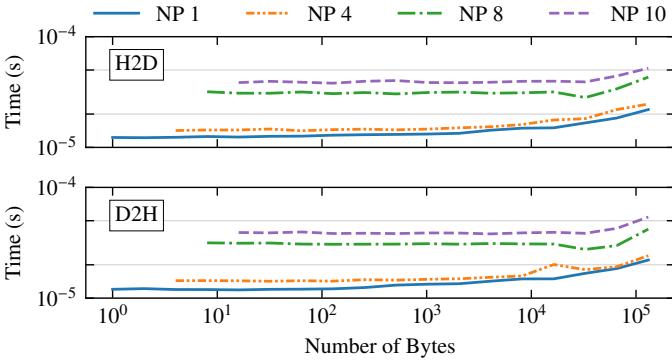


Figure 3.1: The time required to copy various amounts of data from a single GPU using cudaMemcpyAsync when splitting the copy across NP processes. HostToDevice (H2D) and DeviceToHost (D2H) timings presented.

	R_N^{-1} [bytes/sec]
inter-CPU	4.19e-11

Table 4: Measured parameter for injection bandwidth limits on Lassen.

424 on the node-aware techniques discussed in ??.

425 4. Modeling Node-Aware Strategies for Inter-Node 426 Communication

427 In this section, we present performance models for existing
428 node-aware strategies using device-aware and staged-through-
429 host communication for inter-node data exchanges on Lassen,
430 though these models do extend to any machine with two sockets
431 per node. For each node-aware case, the models are divided
432 into the time spent in on-node communication (????), off-node
433 communication (??), and data copies, in the case of staged-
434 through-host communication (see ??). The models themselves
435 do not consider the removal of duplicate data discussed in ??,
436 as the amount of duplicate data injected into the network is
437 operation and problem dependent. However, adapting the input
438 parameters for the models to reflect the removal of duplicate
439 data is straightforward and done in ??.

440 Performance is modeled for standard communication and
441 all node-aware communication strategies discussed in ???. We
442 consider staged-through-host and device-aware communication
443 for all of the strategies except for the split strategies, for which
444 device-aware communication does not apply. “Split + MD”
445 first copies data to a single host process, then splits the inter-
446 node data to be communicated across multiple processes via
447 extra on-node inter-CPU messages. “Split + DD” uses dupli-
448 cate device pointers to copy data from a GPU to multiple host
449 process, reducing the number of on-node messages required to
450 split the inter-node data volume being communicated. Each
451 GPU is assumed to have a single host process except in the case
452 of “Split + DD”. For reference, the modeled communication
453 strategies are listed in ??.

	Staged-through-host	Device-aware
Standard	✓	✓
3-Step	✓	✓
2-Step	✓	✓
Split + MD	✓	
Split + DD	✓	

Table 5: Modeled communication strategies.

454 4.1. Modeling On-Node Communication for 3-Step and 2-Step

455 For 3-Step communication, all data originating on any GPU
456 on node k with a destination of any GPU on node l is first gathered
457 locally. In the worst-case scenario, all GPUs on node k
458 must contribute data for node l , requiring communication among
459 all GPUs per node. This is modeled as

$$T_{\text{on}}(s) = (\text{gps} - 1)(\alpha_{\text{on-socket}} + \beta_{\text{on-socket}} \cdot s) + \text{gps} \cdot (\alpha_{\text{on-node}} + \beta_{\text{on-node}} \cdot s) \quad (4.1)$$

460 where gps is the GPUs per socket and s is the maximum mes-
461 sage size sent by any single GPU.

462 The last step of both 2-Step and 3-Step communication in-
463 volves redistributing data received via inter-node communica-
464 tion to its final destination GPU on-node. The worst case sce-
465 nario for both strategies occurs when all of the data received
466 via inter-node communication needs to be redistributed to ev-
467 ery other GPU on-node. This is also modeled with ??, with s
468 representing the maximum received inter-node message size.

469 4.2. Modeling On-Node Communication for Split

470 The split strategies require copying all data on node k with
471 destination of any GPU on node $l \neq k$ to the host processes be-
472 fore distributing the data across some number of on-node pro-
473 cesses. Finally, each process sends data through the network.
474 For large inter-node message sizes, the worst-case scenario oc-
475 curs when a single GPU contains all data to be sent off-node
476 with a data size large enough that it is split across all on-node
477 processes. In the case of Lassen, there are a maximum of 40
478 on-node processes, therefore distributing the data would require
479 an additional 19 on-socket messages and 20 off-socket/on-node
480 messages if a single host process per GPU were being used.
481 Generalizing the split strategy to any architecture using multi-
482 ple host processes with duplicate device pointers yields

$$T_{\text{on-split}}(s, \text{ppg}) = \left(\frac{\text{pps}}{\text{ppg}} - 1 \right) \cdot (\alpha_{\text{on-socket}} + \beta_{\text{on-socket}} \cdot s) + \left(\frac{\text{pps}}{\text{ppg}} \right) \cdot (\alpha_{\text{on-node}} + \beta_{\text{on-node}} \cdot s) \quad (4.2)$$

483 as the modeled time, where ppg is the number of host processes
484 per GPU, and pps is the processes per socket, and s is the total
485 data volume to be communicated inter-node split across ppg .

486 Similar to the worst-case scenario for 3-Step and 2-Step
487 on-node communication, the worst-case redistribution scenario

488 for the split strategies is equivalent to ???. In this case, a single GPU must redistribute all received inter-node data to every
489 other GPU on-node; here, s represents the total data volume
490 received via inter-node communication split across ppg.
491

492 4.3. Modeling Off-Node Communication

493 For the off-node communication portion of each of the multi-
494 step communication strategies, the max-rate model ?? is used
495 for routines that are staged-through-host, and the postal model ??
496 is used for device-aware routines. For the max-rate model, the
497 time spent in off-node communication is given by

$$T_{\text{off}}(m, s) = \alpha_{\text{off-node}} \cdot m + \max\left(\frac{s_{\text{node}}}{R_N}, s \cdot \beta_{\text{off-node}}\right) \quad (4.3)$$

498 for a number of messages to be communicated, m , and a max-
499 imum number of bytes sent by a single process, s_{proc} where
500 s_{node} is the maximum number of bytes injected into the net-
501 work by any single node. For device-aware communication,
502 this reduces to the postal model

$$T_{\text{off-DA}}(m, s) = \alpha_{\text{off-node}} \cdot m + s \cdot \beta_{\text{off-node}}. \quad (4.4)$$

504 4.4. Copy Parameter for Staged-through-Host Communication

505 The time required to copy data between the CPU and GPU
506 is given by

$$T_{\text{copy}}(s_{\text{send}}, s_{\text{recv}}) = \alpha_{\text{H2D}} + \beta_{\text{H2D}} \cdot s_{\text{send}} + \alpha_{\text{D2H}} + \beta_{\text{D2H}} \cdot s_{\text{recv}}. \quad (4.5)$$

507 where s_{send} is the initial data copied from the source GPU, and
508 s_{recv} is the final data copied to the destination GPU.

509 For all communication strategies except splitting with
510 duplicate device pointers, a single process copies all data from
511 a corresponding GPU. In the case of splitting with duplicate
512 device pointers, we set the number of processes copying data
513 simultaneously to four in our model. Parameters for both a sin-
514 gle host process copying data and four host processes copying
515 data simultaneously are presented in ??.

516 4.5. Model Validation

517 ?? presents the full models for the various communication
518 strategies given in ??, which combine the preceding sub-models,
519 with extra model parameters defined in ?? for clarity.

520 We provide a brief validation of the models via performance
521 of the communication pattern induced by sparse matrix-vector
522 multiplication (SpMV) with the audikw_1 matrix from the Suite-
523 Sparse matrix collection [?]. The matrix has 943 695 rows
524 and columns, and a nonzero density of 8.72e-05 with the as-
525 sociated sparsity pattern in ???. Due to the high number of
526 nonzero entries in the top rows and first columns of the ma-
527 trix, the communication pattern associated with a SpMV for
528 audikw_1 incurs high numbers of on-node and inter-node com-
529 munication, therefore it is a perfect test case for validating the
530 models which model the worst-case on-node communication

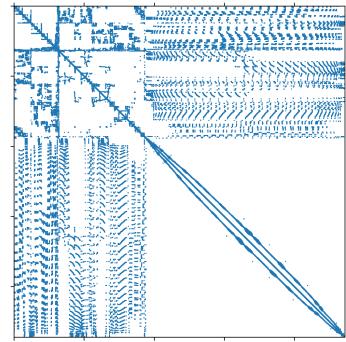


Figure 4.1: Sparsity pattern for the audikw_1 matrix.

scenarios for each of the communication strategies. Modeling the worst-case on-node scenario can result in over-prediction of actual runtimes, as such, these models are not designed to provide a fine-grained prediction. Their purpose is to predict which communication scheme will perform fastest, a task at which they succeed. It is worth noting that the models are easily adaptable to modeling the exact on-node communication that would occur for a given application, should a more fine-grained model be desired. This would require knowledge of problem partitioning, as well as, the communication load of every participating process.

?? depicts the measured times (solid lines) for SpMV communication alongside model predictions (dotted lines) with minimum performing and *minimum model-predicted* times are circled. Presented measured times are the *maximum* average time required for communication by any *single* process for 1000 test runs. In the standard communication cases, the modeled times are an order of magnitude higher than actual measured times, but for the node-aware communication models, the predicted times provide a tight upperbound, generally on the same order of magnitude as the measured performance. In ??, we use these models to predict the performance of common irregular point-to-point communication scenarios.

4.6. Modeled Performance

?? presents the modeled performance for common scenarios with irregular point-to-point communication, namely, a node sending a modest number of inter-node messages (32) and a large number of inter-node messages (256), with messages distributed evenly across on-node GPUs. Because the node-aware performance models are dependent upon the number of destination nodes, the models are split further, modelling if the data was being sent to 4 nodes/ 16 off-node GPUs (??) or 16 nodes/ 64 off-node GPUs (??). Note that the number of GPUs to which data is being communicated does not reflect overall problem partitioning. It simply models the cases where the maximum number of GPUs with which any one node would need to communicate is 16 GPUs on 4 nodes or 64 GPUs on 16 nodes.

For each of these scenarios, we model the amount of time required for each node to send their messages to the destination nodes using standard communication. This modeled performance is compared against that of the various node-aware

Communication Strategy		Model
Standard	Staged-through-host	Max-rate model ??
	Device-aware	Postal model ??
3-Step	Staged-through-host	$T_{\text{off}}(m_{\text{node} \rightarrow \text{node}}, s_{\text{node} \rightarrow \text{node}}) + 2 \cdot T_{\text{on}}(s_{\text{node} \rightarrow \text{node}}) + T_{\text{copy}}(s_{\text{proc}}, s_{\text{node} \rightarrow \text{node}})$
	Device-aware	$T_{\text{off-DA}}(m_{\text{node} \rightarrow \text{node}}, s_{\text{node} \rightarrow \text{node}}) + 2 \cdot T_{\text{on}}(s_{\text{node} \rightarrow \text{node}})$
2-Step	Staged-through-host	$T_{\text{off}}(m_{\text{proc} \rightarrow \text{node}}, s_{\text{proc}}) + T_{\text{on}}(s_{\text{proc}}) + T_{\text{copy}}(s_{\text{proc}}, s_{\text{node} \rightarrow \text{node}})$
	Device-aware	$T_{\text{off-DA}}(m_{\text{proc} \rightarrow \text{node}}, s_{\text{proc}}) + T_{\text{on}}(s_{\text{proc}})$
Split	Staged-through-host + MD	$T_{\text{off}}(m_{\text{proc} \rightarrow \text{node}}, s_{\text{node}/\text{ppn}}) + 2 \cdot T_{\text{on-split}}(s_{\text{node}}, 1) + T_{\text{copy}}(s_{\text{proc}}, s_{\text{node} \rightarrow \text{node}})$
	Staged-through-host + DD	$T_{\text{off}}(m_{\text{proc} \rightarrow \text{node}}, s_{\text{node}/\text{ppn}}) + 2 \cdot T_{\text{on-split}}(s_{\text{node}}, 4) + T_{\text{copy}}(s_{\text{proc}}, s_{\text{node} \rightarrow \text{node}})$

Table 6: Communication models. (Extra parameters defined in ??)

Parameter	Description
s_{proc}	max # of bytes sent by a single process/ GPU
s_{node}	max # of bytes injected by a single node
$s_{\text{node} \rightarrow \text{node}}$	max # of bytes sent between any two nodes
$m_{\text{proc} \rightarrow \text{node}}$	max # of nodes to which a processor sends
$m_{\text{node} \rightarrow \text{node}}$	max # of messages between any two nodes

Table 7: Extra modeling parameters.

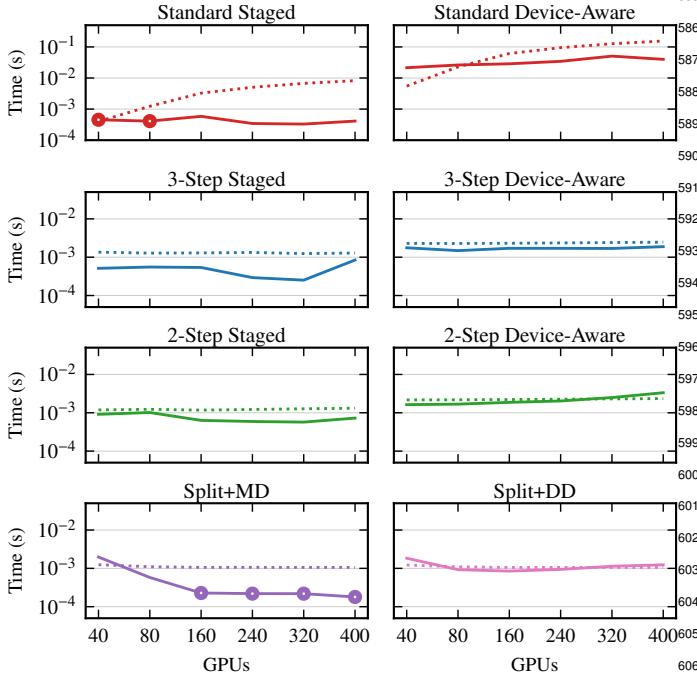


Figure 4.2: Model validation. Solid lines, —, depict measured times, and dotted lines,, depict model predictions. Circles are used to highlight the minimum performing communication strategy, accurately predicted by the models.

strategies where the messages are split and/or agglomerated accordingly. There are two cases presented for the 2-Step strategy, considering if every GPU on the source node is sending

data to every GPU on the destination node (2-Step All), or if all the messages being sent to the destination node are from a single active GPU on the source node (2-Step 1). The message size cap for the split strategies is taken to be the same that was used in [?] and is the message cap used for switching to the rendezvous protocol on Lassen.

In ??, we present the minimum modeled times on the top rows, excluding the 2-Step 1 approaches, as they present the *best-case* scenario for 2-Step communication, which does not often occur in practice. However, we do think it is important to present these modeled times in order to depict a comprehensive picture of node-aware communication's potential, hence they are included in the bottom rows of the plots. For large message counts (256 Inter-Node Messages plots in ??) and for message sizes greater than 10^3 Bytes, device-aware 2-Step 1 is predicted to perform best, indicating that for high inter-node data volumes, if the on-node data was distributed such that every GPU on a given node k was communicating with a distinct GPU on destination node l , 2-Step communication would be best. This is consistent with the observed performance of the application-specific hierarchical communication in [?]. Now, we include discussion of the circled minimum times excluding the 2-Step 1 performance predictions.

In the case of a small number of messages injected into the network to a small number of nodes (??), 3-Step and standard communication are observed as the most performant with Split+MD communication replacing 3-Step as the most performant for 16 nodes (??). In both cases, the staged-through-host strategies predict the best performance until message sizes grow extremely large ($> 10^4$ Bytes), where standard device-aware communication is modeled to be best. Device-aware communication is also modeled to be best for large message sizes when a node is injecting a large number of messages into the network. However, due to the high message volume, 3-Step and 2-Step device-aware strategies are predicted to have the optimal performance, due to their reduction in messages sent compared to standard communication.

Staged-through-host node-aware communication techniques model the best performance independent of number of destination nodes for all message sizes up to 10^4 Bytes. When commu-

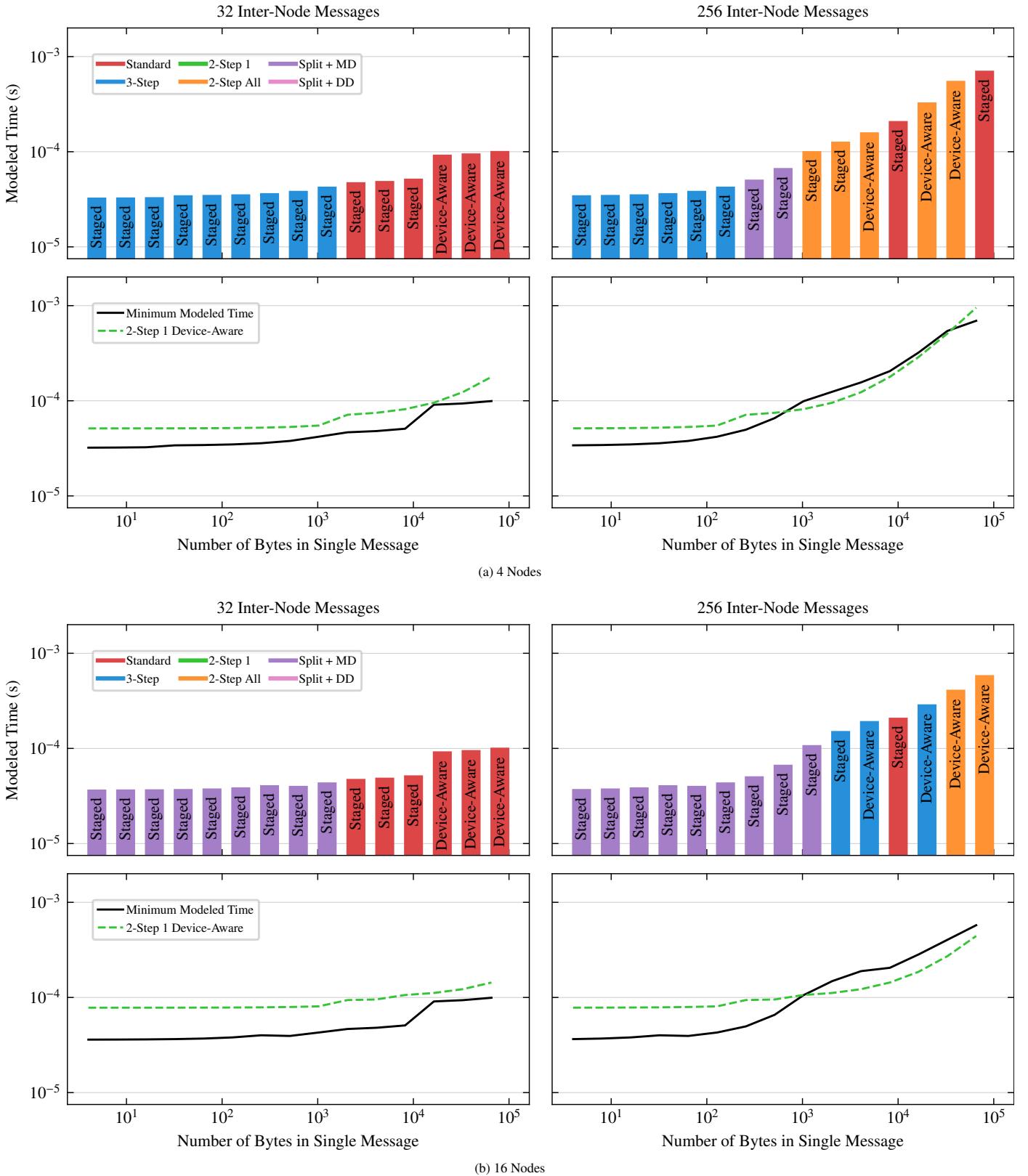


Figure 4.3: The modeled time to send data from a single node to **4 nodes** (top) and **16 nodes** (bottom), where data from the sending node is sent via 32 or 256 messages distributed evenly across all on-node GPUs when using Standard communication. Minimum modeled times (excluding the 2-Step *best-case* scenario, 2-Step 1) are presented with bars on the top rows. A comparison of the minimum modeled times to the 2-Step 1 case is presented on the bottom rows.

nicipating with a small number of nodes (??), 3-Step and 2-Step₆₆₉ communication are often predicted to be the most performant,₆₇₀ while Split+MD communication is predicted to be the most per-₆₇₁ formant when communicating with a larger number of nodes₆₇₂ (??). This can be attributed to the use of all available processes₆₇₃ on-node (40 in the case of Lassen), so that each individual pro-₆₇₄ cess is injecting fewer messages into the network than in the₆₇₅ case of 3-Step or 2-Step communication, where there is only a₆₇₆ single process paired with each GPU (4 in the case of Lassen).₆₇₇

The device-aware node-aware strategies models present rel-₆₇₈ atively large costs. However, this is unsurprising, considering₆₇₉ the high overhead for inter-GPU communication on-socket and₆₈₀ on-node (as indicated by the measured parameters in ??). The₆₈₁ only cases for which device-aware node-aware strategies have₆₈₂ improved performance over staged-through-host techniques is₆₈₃ when the communicated inter-node data volume is *extremely*₆₈₄ large, or assumed to have an optimal communication pattern₆₈₅ (as in 2-Step 1).₆₈₆

Concerning the removal of duplicate data, there should be₆₈₇ no impact on performance for small numbers of inter-node mes-₆₈₈ sages. A performance impact is noticed primarily when their₆₈₉ is communication of a high inter-node data volume via a high₆₉₀ number of messages. Once message sizes grow past 10^3 Bytes₆₉₁ in standard communication for all high message count models,₆₉₂ removing duplicate data impacts which node-aware communi-₆₉₃ cation strategy could be most performant. Seeing as there is₆₉₄ very little difference in modeled performance for removing du-₆₉₅ plicate data, these modeled times are excluded for brevity.₆₉₆

Overall, the staged-through-host node-aware communica-₆₉₇ tion strategies model the best predicted performance for com-₆₉₈ munication patterns requiring a high number of inter-node mes-₆₉₉ sage exchanges. In ??, we benchmark the performance of the₇₀₀ communication strategies within the context of sparse matrix-₇₀₁ vector multiplication, verifying model predictions.₇₀₂

5. Benchmarking Sparse Matrix-Vector Multiplication Communication Patterns

In this section, we present performance results for the vari-₆₅₁ ous communication strategies discussed throughout ???. When₆₅₂ applied to the communication patterns of a single distributed₆₅₃ SpMV – see ??, For each of the strategies, each GPU is as-₆₅₄ sumed to have a *single host process*, except in the case of “Split₆₅₅ + DD” where *four host processes* are used to copy data from₆₅₆ each GPU. The number of host processes used to copy data₆₅₇ from any GPU is distinct from the number of processes used to₆₅₈ communicate inter-node data, in the case of the split strategies.₆₅₉ For the split strategies, after data is copied from each GPU via₆₆₀ some number of host processes, inter-node communicated data₆₆₁ is potentially partitioned across up to all available 40 processes₆₆₂ on-node (the maximum number of on-node processes/cores for₆₆₃ Lassen.) Our test matrices are a subset of the largest matrices₆₆₄ in the SuiteSparse matrix collection [?]. For each benchmark,₆₆₅ we performed 1000 test runs and present the *maximum* average₆₆₆ time required for communication by any *single* process. The₆₆₇ presented results reflect *actual* measurements on Lassen.₆₆₈

5.1. Results

?? displays the distributed SpMV communication bench-
mark performance times for each communication strategy pre-
sented in ?? for each SuiteSparse matrix. Presented beneath
each plot is the number of GPUs on which the SpMV is par-
titioned, the maximum number of nodes to which any single
node is communicating (Recv Nodes), and the communicated
inter-node message volume for standard communication.

Consistent with the majority of model predictions for large
inter-node message volumes, the staged-through-host commu-
nication strategies exhibit far faster performance than the device-
aware communication strategies. However, it is worth noting
that device-aware 3-Step and device-aware 2-Step are typically
much faster than standard device-aware communication. In the
case of the *thermal2* matrix, which exhibited a high inter-node
message volume for standard communication, the gap between
the device-aware node-aware strategies and staged-through-host
communication strategies is smaller than for other matrices.
Additionally, “Split + DD” consistently performed worse than
“Split + MD”, consistent with modeled predictions. This is
unsurprising considering the latency associated with using
duplicate device pointers (~1.5e-05 in ??) is much higher than
the latency of sending on-socket messages (~3.7e-07~3.2e-06
in ??) to distribute data being sent from a single GPU across
multiple ranks.

The majority of the presented results are similar to the model
prediction plots (??), where the fastest communication strategy
was typically predicted to be one of the staged-through-host
strategies: “Split + MD”, Standard, or “3-Step”. “Split + MD”
exhibits the minimal performing time in most cases, except for
smaller counts of participating GPUs (40 or 80 in the case of
audikw_1, *Serena*, *ldoor*, *thermal2*), or for low inter-node
message counts (*bone010*, *Geo_1438*) in which standard com-
munication becomes more optimal.

Overall, staged-through-host node-aware communication strate-
gies demonstrate the best performance for the majority of bench-
marks, with “Split + MD” typically being the fastest, consistent
with model predictions in ??.

6. Conclusions and Future Work

The advancement of parallel computers has introduced the
design of supercomputers with heterogeneous compute nodes
due to the inclusion of multiple GPUs per node. For distributed
applications, this typically results in larger communicated data
volumes, as each compute unit can now operate on a larger par-
tition of the problem. In addition to increased data volumes, the
inclusion of multiple GPUs per node has increased the com-
plexity of determining optimal data movement paths, partic-
ularly in the case of inter-node irregular point-to-point commu-
nication. In this work, we characterized the performance of ir-
regular point-to-point communication between GPUs via mod-
eling and introduced node-aware communication strategies to
inter-node communication on heterogeneous architectures. Our
models suggested the use of staged-through-host node-aware
communication strategies, specifically split methods were indi-
cated as potential top performers. These results were confirmed

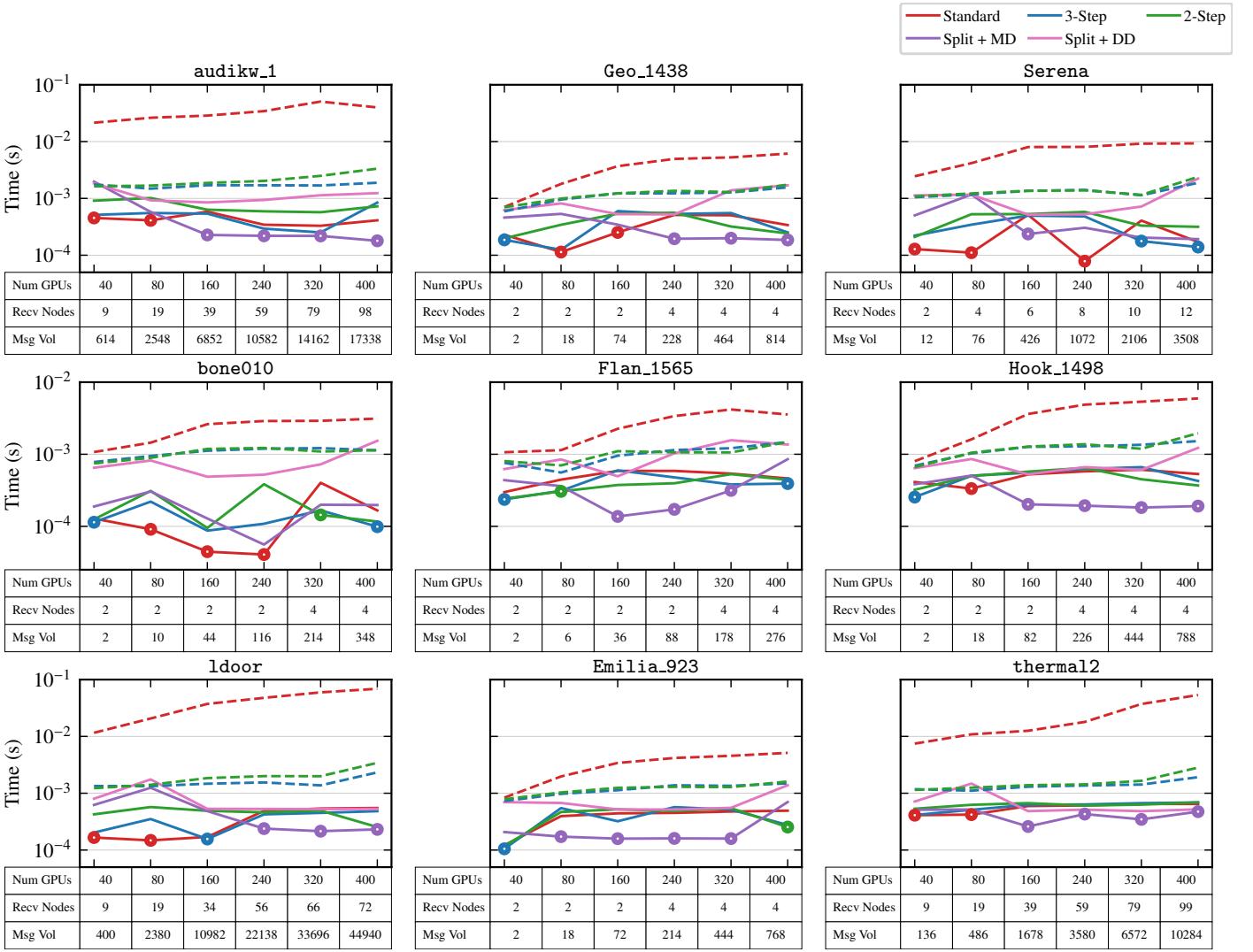


Figure 5.1: The measured time spent in irregular point-to-point communication for a distributed SpMV for various SuiteSparse matrices. Number of GPUs across which the problem was partitioned and standard communication maximum number of connected nodes for any single node (Recv Nodes) and message volume included beneath plots. Solid lines, —, depict staged-through-host communication, and dashed lines, - - -, depict device-aware communication. Minimum times circled for convenience.

724 by a performance study on distributed SpMVs which saw split₇₄₁
 725 node-aware communication performing best in most cases, and₇₄₂
 726 typically much faster than standard device-aware communica-₇₄₃
 727 tion.

728 Additionally, our work provides important groundwork on
 729 designing efficient communication strategies for the next
 730 generation of supercomputers. Future exascale machine architec-₇₄₄
 731 tures will include higher CPU core counts per node, alongside₇₄₆
 732 higher bandwidth interconnects (e.g., on Frontier, El Capitan,₇₄₇
 733 or Delta), two parameters that largely affect the performance of
 734 node-aware communication strategies. Based on the presented
 735 models, Split communication strategies will likely be the most
 736 efficient communication techniques to take advantage of the
 737 high bandwidth interconnects, but distributing data to be com-
 738 municated across a larger number of on-node CPU cores could
 739 pose performance constraints. Because the models presented
 740 in ?? naturally extend to architectures with single socket nodes,

future work includes plans to begin modeling the performance of machines resembling the next generation DOE exascale machines.

Acknowledgements

This material is based in part upon work supported by the Department of Energy, National Nuclear Security Administration, under Award Number DE-NA0003963 and DE-NA0003966.