Scalability of Hybrid SpMV with Hypergraph Partitioning and Vertex Delegation for Communication Avoidance

Brian A. Page, Peter M. Kogge
Dept. of Computer Science and Engineering
Univ. of Notre Dame, Notre Dame, IN 46556, USA
Email: (bpage1,kogge)@nd.edu

***Abstract*—Communication overhead has been identified as the primary factor in overall performance degradation for sparse and irregular problems such as SpMV. Many works have shown significant communication reductions, but only for matrices with specific characteristics and by dramatically reworking the computations. This study develops and evaluates a communication avoiding distributed heterogeneous implementation for strong scaling of SpMV on the Sierra supercomputer architecture. To address the far bigger matrices characteristic of real problems, we utilize a hypergraph partitioning package HYPE to determine workload distribution and reduce inter-node communication. Additionally we investigated the performance impact of performing hypergraph partitioning on scale free graphs which had undergone a vertex delegation pre-processing step. We achieved up to 97% reduction in average message size per process at scale when using the HYPE partitioner. Despite this we show how optimizing SpMV on existing GPU architectures does provide increased computational performance, yet does not address the dominant communication overhead factor at scale despite attempts to avoid communication where possible.**

***Keywords*-Strong Scalability, Hybrid SpMV, Multi GPU, Sierra Systems, Communication Overhead, HPC, MPI;**

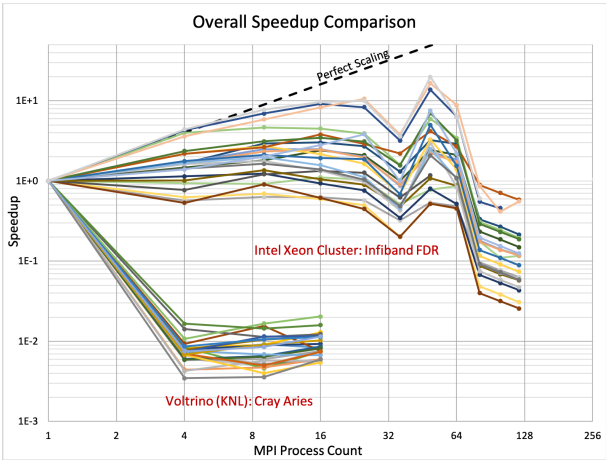


Fig. 1: Overall Speedup on Intel Xeon [7] and Intel KNL Clusters [8]. Each line represents a single benchmark matrix used in both studies, with speedup evaluation considering both computation and communication time measurements. 1 MPI rank per socket with the Intel cluster having 2 MPI ranks per node, and the KNL cluster having 1 MPI rank per node.

I. Introduction

Dense linear algebra boasts well documented efficient algorithms and performance models for nearly all modern architectures. In contrast, sparse linear algebra operations remain a field rife for deeper optimization efforts. The product of a sparse matrix and a dense vector (**SpMV**) is a key part of many codes from disparate areas. For instance, SpMV constitutes the bulk of the High Performance Conjugate Gradient (HPCG) [1] code that has become an alternative to LINPACK for rating supercomputers. It is also used extensively in its general sparse-matrix sparse-matrix form, in linear solvers such as HYPRE [2], and finite element method applications such as PGFem3D [3], [4]. Additionally when matrix operations are changed from floating multiply and add to other non-numeric functions, it becomes an essential part of many graph kernels [5], and is a key function in the GRAPHBLAS spec [6].

In earlier studies [7], [8] we examined strong scaling of SpMV in a hybrid Message Passing Interface (MPI) + Multi-threading (OpenMP) environment for a variety of architectures and moderate problem sizes. These studies showed that when strong scaling is attempted (fixed matrix size but increasing processor count), SpMV performance *degradation*, not improvement, can be seen for all matrices, often with relatively few processes. While strong scaling improves computational performance, network communication among participating MPI processes drastically reduced any overall speedup. Fig. 1 illustrates these effects for the Intel Xeon and Intel Xeon Phi Knights Landing (KNL) systems evaluated earlier.

Based on earlier findings, which show that communication is the dominant factor in overall performance, our goal for this paper is to evaluate the strong scaling behavior of iterative SpMV in which communication is deliberately avoided or reduced wherever possible. Therefore we developed and tested a multi-GPU hybrid and heterogeneous SpMV kernel using MPI, OpenMP, and the NVIDIA cuSPARSE [9] library on the Sierra supercomputer architecture. To reduce communication we perform a *hypergraph partitioning* on all benchmark sparse matrices since it has been shown to reduce communication by upwards of 60% compared to traditional graph partitioning [10]. Additionally in an effort to magnify the benefits of hypergraph partitioning for matrices which posses scale-free characteristics we perform an additional pre-partitioning step

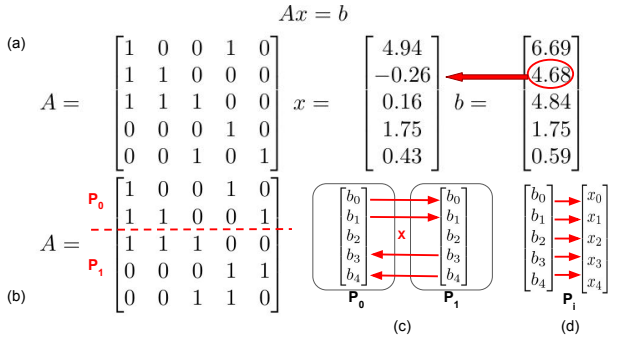


Fig. 2: Distributed Iterative SpMV Communication Pattern

in which vertex delegation marks and removes very high degree rows (vertices) from the matrix before partitioning. We compare results from both implementations with a greedy balanced workload partitioning.

In most studies on SpMV so far, the focus has been on problems that "fit" in a shared memory space or which do not require distributed processing. Yet increasingly there are problems which are simply "too big" for single node systems, requiring multiple nodes just to store matrix and dense vector data in memory. In this study we focus on analyzing the scalablity of very large matrices, each containing several billion non-zeros, incapable of computation on a single node.

Our analysis shows that despite our efforts to reduce and or avoid communication, when possible, the remaining communication still caused substantial overall performance degradation despite observing a computational speedup during strong scaling.

In organization, Section II provides background. Section III describes the spectrum of sparse matrices considered. Section IV overviews the experimental platform, workload balancing and distribution, and hybrid GPU algorithm. Finally Section V evaluates the results, and Section VI concludes.

II. Background

A. Iterative SpMV Overview

Performing iterative SpMV in a distributed environment adds additional complexities such as communication overhead which much be factored into overall performance. Communication requirements are tied to the workload partitioning of rows as well as the non-zero structure of the sparse matrix. Fig. 2 demonstrates the communication associated with distributed SpMV, given a sparse matrix A is multiplied by the dense vector x with its row results being placed into the result vector b. In many applications such as HPCG, SpMV is called iteratively on the same A, with each result vector reused in some way for the x in the next iteration. Thus we assume in our study that the computed b must be placed back in essentially the same order across the cluster as the original x.

Fig. 2b shows a possible partitioning of A among two MPI processes. Entire rows are assigned to their respective process P_i along with the elements in x corresponding to the column id $col[i]$ of each non-zero in the rows. A has been distributed along with possibly overlapping subsets of the dense vector x. Computation of rows proceeds locally on each process and row results are placed into their respective result vectors. As mentioned previously, the dense vector is updated with the row results after every iteration to reflect iterative applications. However x has been distributed with potentially multiple copies of each element existing on different processes. In Fig. 2c we can see the communication pattern for the given sparse matrix and its partitioning. P_0 must send its $b[0]$ and $b[1]$ to P_1 due to P_1 containing a row that requires $x[0]$ and $x[1]$. Similarly, P_1 sends $b[3]$ and $b[4]$ to P_0. Note that the row result for $b[2]$ is never sent because the corresponding element $x[2]$ is only used by P_1. Lastly each process updates its x vector from its b vector which contains row results from the latest SpMV iteration.

B. SpMV on GPUs

Given the memory bandwidth constraint on SpMV, minimizing references is critical. Various storage schemes exist for maintaining a compressed version (does not contain zero entries) of the sparse matrix in memory. Compressed Sparse Row (CSR) format is a commonly used storage format in which the non zeros within a given matrix are kept in three one-dimensional vectors: row, column, and value. Many-core architectures such as GPUs provide memory access channels capable of higher sustained bandwidth and therefore can potentially aid in the performance of sparse memory bound problems such as SpMV. Several studies have analyzed SpMV on GPUs, with the majority exploring the impact of novel matrix compression techniques. Such methods capitalize on the architectural nuances these platforms provide [11], [12]. Even so most of these methods focus on single node or architecture specific implementation, do not attempt strong scaling, and or use matrices which are relatively small in size and number of non-zeros. CSR has been the basis for many of them [13], [14], [15], [16], [17].

Many other studies attempt to obtain improved performance by leveraging alternative storage formats (such as COO, CSR, DIA, ELL, HYB, BCSR, etc.) [18], [19], [20], [21], [16]. For instance ELLPACK (ELL) and Hybrid (HYB), developed for use with GPUs, have exhibited greatly improved single processor performance over CSR.

In addition to storage formats, libraries designed to perform linear algrebraic operations, such as NVIDIA CUDA basic linear algebra subprograms (CUBLAS) and NVIDIA CUDA sparse matrix library (cuSPARSE), are used extensively in machine learning, computational fluid dynamics, and computational sciences [9]. While single-GPU codes are common place, multi-GPU implementations of SpMV appear to be rare, with distributed multi-GPU codes even more so. Single GPU implementations of sparse linear algebra have demonstrated increased efficiency and performance [22], [23]. However modern supercomputing systems feature multiple many-core CPUs along with multiple GPUs. We believe that a strong scaling evaluation of SpMV must therefore include exploration into multiple GPUs per node, as well as multiple nodes.

For multi-node implementations, the distribution of both matrix data and result vectors, especially as process counts grow, directly affects the volume and size of inter-process communication. While more exotic in-memory matrix storage formats may provide some performance increase for SpMV computation, they do not alter the overall communication requirement. In this study we focus on the impact of communication on overall performance, therefore we chose to implement SpMV using the CSR format due to its implementation simplicity.

C. Sierra Supercomputer Architecture

Sierra at Lawrence Livermore National Laboratory is one of the latest in a series of leading-edge Advanced Simulation and Computing (ASC) Program supercomputers. Built by IBM in partnership with NVIDIA Corporation and Mellanox Technologies, Sierra is a heterogeneous supercomputer that uses IBM Power9 CPUs along with NVIDIA Tesla V100 (Volta) Tensor Core GPUs. Sierra's heterogeneous architecture utilizes the enormous parallelism delivered by its GPUs to accelerate scientific computing applications while providing greatly enhanced energy efficiency over previous systems. Sierra is the first production system of its kind produced for the NNSA and is currently one of the most promising architectures for future, exascale computing solutions [24].

Our experiments were run on the Lassen supercomputer which though smaller in size is architecturally identical to Sierra, and available to the research community. Its interconnect is a Mellanox Infiniband EDR connected via a tapered fat tree switch network with dual links per node. Lassen comprises 684 compute nodes, each with dual-socket 22 core Power9s and 4 Nvidia V100s (Volta). Each Volta is connected via NVIDIA NVLink system interface, providing a peak bi-directional bandwidth of up to 100GB/s. Furthermore each GPU contains 5,120 CUDA cores along with 16GB high bandwidth memory version 2 (HBM2) memory at up to 900GB/s memory bandwidth [25]. While Power9 CPU performance is something that may play a role in some overall application performance, for this study it only handles messaging and GPU management.

III. Benchmark Matrices

The matrices used for our earlier studies, pictured in Fig. 1, were largely from the SuiteSparse Matrix Collection[1] [26]. Our focus in this study is on much larger matrices where strong scaling would be of most value, in addition to being required due to resource limitations. Table I lists characteristics of the matrices we chose. All but the synthetic matrix are real data sets with varying size, number of non-zeros per row, and non-zero structure. The PGFem3d multiscale finite element solver [3], [4] in particular is relevant to the class of solvers in use in an on-going large multi-scale simulation project. The synthetic matrix was generated to conform to the dimension and nnz values shown in Table I. Each row's non-zeros were assigned to columns randomly generating a random structure and column-wise degree distribution.

[1]Currently hosted at https://sparse.tamu.edu/

TABLE I: Benchmark Matrix Suite

Matrix	Rows	NNZ	NNZ%	NNZ_{row}
com-Friendster	65,608,366	3,612,134,270	8.39E-7	55.1
PGFem3D_stiff	35,859,280	2,877,137,749	6.71E-6	84.5
synth_rand	40,000,000	4,000,000,000	2.5E-6	100
MOLIERE_2016	30,239,687	6,669,254,694	7.29E-6	220.5

Fig 3 illustrates the vertex degree distribution for each benchmark matrix. The synthetic matrix is not shown due to its absolute uniformity with each row having exactly 100 non-zeros. Meanwhile the social network *com-Friendster* matrix exhibits a power law distribution indicating it is scale-free. This characteristic is common among real world graphs, especially those of social networks [27], and often prove troublesome for graph analytics due to their extremely large vertex degree disparity [28]. We discuss our attempts to mitigate the affects of high degree vertices in Section IV-C.

IV. Experimental Setup

A. Distributed Multi-GPUs SpMV

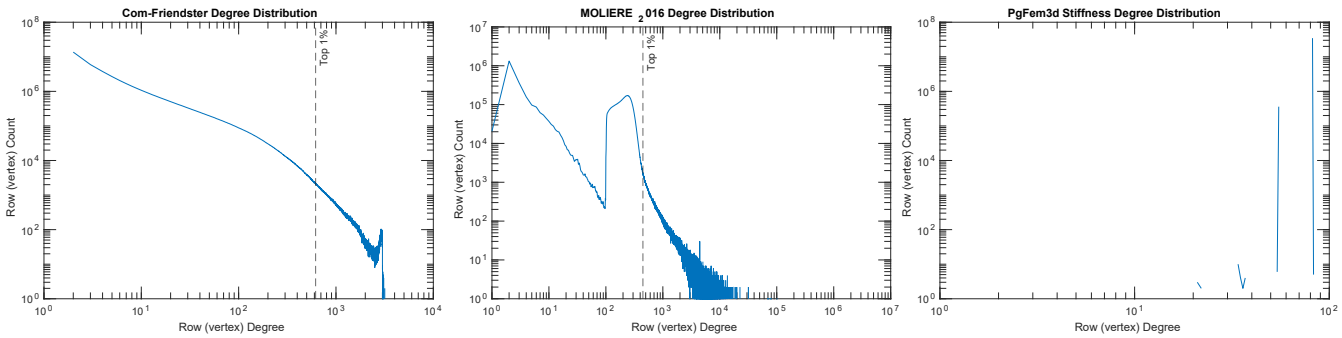
To perform this study we developed a multi-GPU, multi-node hybrid implementation using the cuSPARSE library [9]. cuSPARSE provides CUDA sparse basic linear algebra subprogram kernels which provides improved performance over CPU-only alternatives.

Our heterogeneous code disperses computation across multiple GPUs, effectively performing several disjoint sub problems simultaneously within each node. To achieve this we assign a single MPI rank to each cluster node, and use OpenMP threads to perform CUDA calls on their corresponding GPU. Each OpenMP thread was bound to a single Power9 core according to its thread id such that it would reside on the socket associated with its corresponding GPU's system interconnect. This assisted performance by eliminating the need for CPU-to-GPU memory allocation and copies to incur cross socket communication overhead.

A local CSR representation is generated on every node, using the non-zeros assigned to it as a result of matrix partitioning. The local CSR is then partitioned further depending on the number of GPUs selected, in Lassen's case one partition for each of the 4 Volta V100s present on every node. The cuSPARSE library is not designed explicitly for use within a multi-GPU setting. Instead our implementation creates 4 disjoint sub problems, each with their own GPU, memory buffers, and kernel context on every rank. OpenMP threads then call the cuSPARSE kernel on their respective GPU. Once all local GPU computation has completed we collect and reorder GPU results on each host (node) to form the single result vector for each node (MPI rank). Row results are then distributed throughout the system and used to update the dense vector ahead of the next SpMV iteration.

B. Workload Partitioning with Hypergraphs

Determining an optimal workload distribution is a common and significant issue. With respect to SpMV various



(a) com-Friendster (b) MOLIERE_2016 (c) PgFem3d Stiffness

Fig. 3: Benchmark Matrix Row Degree Distribution

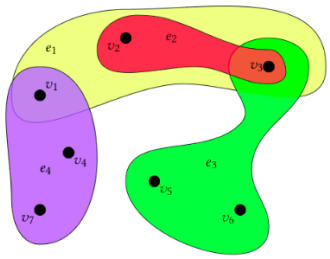
partitioning schemes have been developed in an effort to obtain improved workload uniformity, reduce computation time, decrease cache misses, and even optimize for new multi-core and many-core architectures. Various forms of the well known 1D and 2D matrix partitioning methods were evaluated in previous studies [7], [8]. These methods perform matrix partitioning based on structural characteristics of the input matrix such as symmetry, or being clustered along the main diagonal. We implemented a more generalized approach which is independent of specific matrix characteristics and requires no apriori knowledge about the input matrix.

Many applications utilize hypergraphs to represent the interconnection of vertex properties within a given data set. Hypergraphs contain hyperedges which, unlike traditional graph edges that connect only two vertices, can join any number of vertices. All vertices belonging to the vertex set of a hyperedge share some property as defined by the data set and application in question. Additionally, a vertex may exist in multiple hyperedges, that is it has properties in common with more than one set of vertices. Formally hypergraph partitioning is the process of finding a partitioning of a hypergraph such that some cost function, such as net cut, or fanout (k-1) is minimized. It is used in many fields such as VLSI design [29], and database storage shard reduction [30].

Any sparse matrix can be treated as a hypergraph by considering an arbitrary $row[i]$ as a hyperedge $h[i]$, and the column id of the non-zeros within that $row[i]$ as vertices belonging to $h[i]$. Partitioning of a hypergraph representation of sparse matrices for use in SpMV have been shown to reduce communication by up to 60%, as it more accurately depicts the communication pattern required by row result updates [10].

Fig. 4 shows a sample sparse matrix A and its transpose A^T. If a hypergraph is generated using A then its rows are treated as hyperedges, with the column ids of the rows' non-zero values treated as the vertices in each hyperedge's vertex set. Therefore the unifying property of each hyperedge's *vertices* is that they all belong to the same row of A. Conversely if the transpose of a sparse matrix is used to generate a hypergraph than the columns become hyperedges and row ids of the non-zeros within each column are the vertices within them. The visual representation of the hypergraph in Fig. 4 illustrates the higher-dimensional and overlapping

$$A = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$



Partition	Row Id
1	1 2 3
2	5 6
3	1 4 7

Fig. 4: Hypergraph Partitioning of a Sparse Matrix

behavior inherent to hyperedges. Our implementation performs partitioning on hypergraphs generated using the transpose of each benchmark matrix. This was important because we chose to use CSR storage format for local matrix data, which can obtain improved cache performance when all non-zeros of a row are contiguous in memory.

With respect to SpMV, entire row results are computed and the elements requiring that result in subsequent iterations would be co-located with the row itself when possible. This behavior enables hypergraph partitioning to effectively eliminate the need for communication of row results where such co-location were possible, or allow for reducing the number of partitions to which a result must be distributed.

Hypergraph partitioning is a difficult problem, with balanced $k-way$ hypergraph partitioning being NP-hard. As a result we used the **HYPE** hypergraph partitioner [31] which uses neighborhood expansion for efficiently determining optimal vertex assignment. HYPE performs a k-way partitioning by analyzing the vertices within each hyperedge and generating a minimal *core set* of vertices with which to calculate similarity. Additionally HYPE seeks to minimize the K-1 metric, the number of times that neighboring vertices are assigned to different partitions, for the given graph.

Fig. 5 shows the HYPE partitioning quality for the bench-

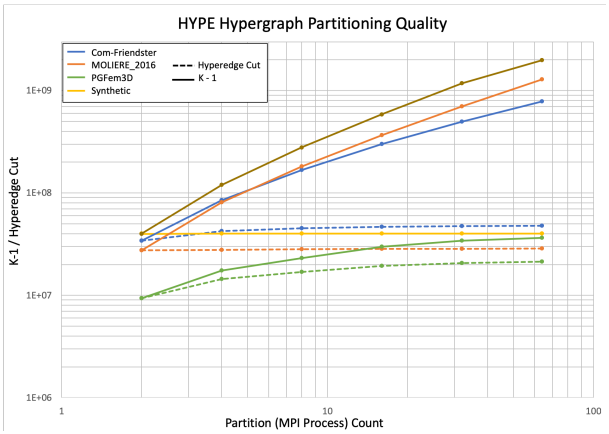


Fig. 5: Hypergraph Partitioning Quality. For all tests, MPI process count equals cluster node count.

mark matrices at all MPI Processes counts evaluated (lower is better). As the number of partitions increases, K-1 increases for all matrices. This was expected due to the increasing likelihood that a hyperedge will have its vertices placed on separate partitions in order for HYPE to maintain a balanced partitioning. However Fig. 5 also shows that the number of hyperedge cut (spread across more than one partition) increases much slower, appearing to level off after approximately 16 processes for all matrices.

In our tests HYPE produced partitions with a hyperedge cut 20%-40% lower than the total row count of each matrix. This indicates that before the message volume resulting from fanout by the remaining vertices, we have potentially reduced communication by 40% or more.

C. Vertex Delegation

Since we are treating the sparse matrices as graphs for the purposes of partitioning, we looked at some additional graph optimization techniques. Graphs containing high degree vertices (hubs), such as *scale-free* graphs which follow a power law degree distribution, can pose significant storage and communication requirements when performing operations on the graph. Current graph partitioners fail to efficiently partition graphs with very high degree vertices without generating considerable workload imbalance due to the exceedingly large edge list associated with very high degree vertices. To alleviate this imbalance a technique, known as *vertex delegation* [28], spreads vertices with a degree above some threshold across multiple partitions and was shown to be particularly useful for the traversal of scale-free graphs.

In vertex delegation local copies of a hub vertex are created on each partition and then only the edges between the hub vertex and those non-delegated vertices assigned to the partition are assigned. By doing so the vertex, its edge list, and its computational workload has been distributed across multiple partitions, thereby generating a more balanced workload.

Computational workload of a graph depends on the operation being performed, For this study, computational workload pertains to the distribution of non-zeros in a highly populated row across multiple MPI processes. Each delegated row's partial sum must be reduced into a single solution which must then be communicated back to all partitions for the next SpMV iteration. This may improve communication overhead for SpMV since in addition to reducing imbalance, vertex delegation can also reduce communication between partitions by greatly increasing the number of intra-partition operations, and decreasing off partition updates [28].

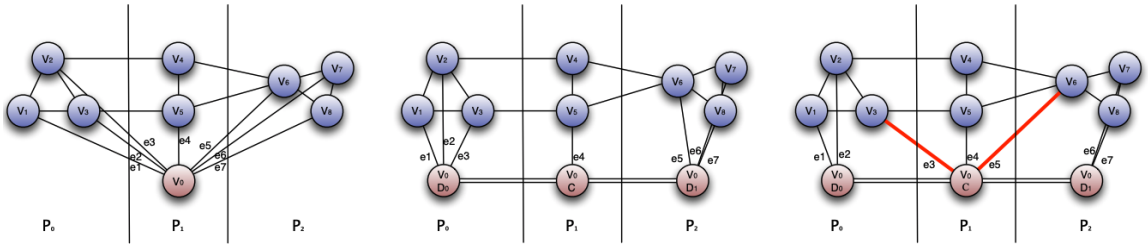
Fig. 6a diagrams a sample graph containing a relatively high-degree vertex. If we were to perform 1D partitioning on the given graph, we must decide which partition is assigned the edges associated with V_0. If Partition P_2 were assigned V_0 and all of its edges, P_2 must update the vertices in P_1 and P_3 which are adjacent to it. Furthermore the placement of a hub's edge list onto a single node may create considerable workload imbalance along with increased communication overhead associated with edge and or vertex updates.

To alleviate load imbalance, vertex delegation generates local copies of V_0 and distributes its edge list on each partition as seen in Fig. 6b. The computational workload for updates to the hub vertex has now been distributed, resulting in a more balanced workload. Computation is performed using the local state of each hub vertex. After computation, the local states of each hub are reduced to form a uniform state across all partitions once again. This can dramatically reduce storage requirements by eliminating duplication of edges on all partitions involved, as well as allow for superior workload balancing of *scale-free* graphs on distributed systems.

Finally Fig. 6c illustrates how the distributed edge list could be further refined to create a superior workload balancing. It is important to note that this improvement comes at a cost of increased communication.

As shown in Fig. 3 the com-Friendster matrix exhibits a power law degree distribution, and MOLIERE_2016 appears to have a quasi power law distribution. Because of these characteristics we elected to generate an additional partitioning in which we perform vertex delegation on these matrices prior to performing hypergraph partitioning with the HYPE partitioner. Our implementation creates delegates from the top 1% of vertices according to their degree. The vertical dashed lines seen in Fig. 3a and 3b indicate where this selection occurs based on the degree distribution, with all vertices to the right of this line becoming delegates. When performing vertex delegation on com-Friendster and MOLIERE_2016 we observed that due to the power law behavior many vertices (rows) had all corresponding edges removed, therefore disconnecting them from the graph. The HYPE partitioning software does not allow for disconnected vertices (empty rows) to exist in the hypergraph when performing partitioning. Consequently these *free* vertices were also removed. The remaining matrix was then partitioned using the HYPE hypergraph partitioning software [31].

For our distributed SpMV evaluations when loading the matrix into memory according to the partition assignment, it was possible to assign free vertices to any partition arbitrarily since the hub vertex they are adjacent to was guaranteed to have a local copy present on every partition. This allowed



(a) 1D Partitioning

(b) Distributed Delegates Partitioning. P_1 contains the *controller*.

(c) Distributed Delegates Partitioning after balancing.

Fig. 6: Vertex Delegation of Graphs with High Degree Hubs from [28].

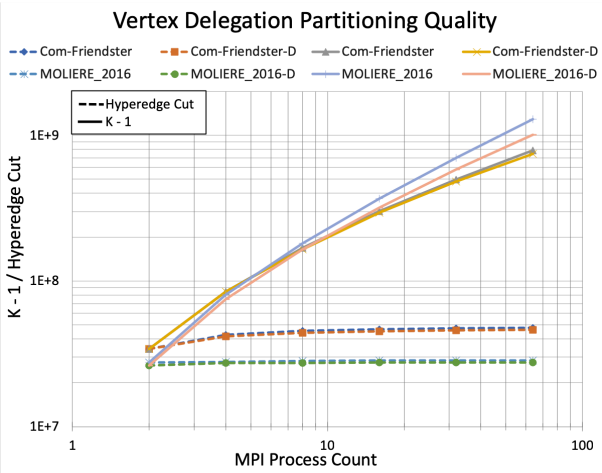


Fig. 7: Hyperedge Cut and K-1 Metric for Hypergraph Partitioning with Vertex Delegation. HYPE partitioning is denoted by the matrix name, while *-D* suffix indicates hypergraph partitioning with vertex delegation. For all tests, MPI process count equals cluster node count.

us to determine the partition assignment of free vertices as $P_i = V_i \mod |P|$. The resulting partition quality can be seen in Fig. 7. We observed that hyperedge cut increases as the number of partitions k increases for com-Friendster and MOLIERE_2016. Additionally Fig. 7 also shows the $K-1$ metric increasing for both matrices. This behavior was expected since the fanout of an arbitrary hyperedge increases, in general as K increases, as the number of vertices associated with that hyperedge are more likely to have been assigned to different partitions.

D. Greedy 1D Partitioning

In this study we compare performance observed using *hypergraph partitioning* and *hypergraph partitioning with vertex delegation* with that of a greedy 1-dimensional partitioning. This greedy 1D partitioning is similar to the balance workload distribution method in [7]. The goal of this method is to create a near uniform non-zero work load distribution across all partitions in an efficient manner.

To achieve near uniform partitioning, rows are sorted based on their length (vertices sorted by degree), then assigned to partitions in descending order. Rows are always assigned to the partition with the current lowest number of non-zeros assigned to it. All non-zeros belonging to a row are assigned to the same partition in order to take advantage of cache behavior and hopefully improve performance. This also prevents the need to communicate partial sums for a row throughout the system, instead only final row results must be distributed when necessary.

For the remainder of this paper we will refer to this partitioning method as the *greedy* method.

E. Result Accumulation

For interprocess communiation we utilize the IBM Spectrum MPI [32] library supported on Sierra and Lassen systems. IBM Spectrum MPI is based on Open MPI 3.0.0 and maintains similar functionality.

In our iterative distributed SpMV tests, after an MPI rank has completed its computation it must then update each rank which requires its row results in so as to allow them to update their dense vectors prior to the next iteration. For our implementation the matrix partitioning is known by every rank, enabling each rank to know precisely which row results to send to as well as what to receive from other ranks.

To ascertain the impact on communication overhead a particular communication method might have, we chose to implement several methods for all matrices and process counts: MPI_Allgather, MPI_Allreduce, and Remote Direct Memory Access (RDMA) via MPI_Get. Several studies [33], [34], [35] have evaluated the performance and behaviour of various global collectives, such as Allgather and Allreduce, in addition to implementing optimization methods. These method are still commonplace and therefore serve as a good baseline implementation for computation and communication performance. While the selection of the algorithm used internally within a collective can be explicitly set by the developer, MPI does attempt to select the best algorithm based on message size, process count, etc. In this study we made no effort to optimize MPI_Allreduce and MPI_Allgather and allowed MPI to use default parameters.

RDMA provides one-sided communication which depending implementation, and can eliminate the need for acknowledgements and or overly complex synchronization events. RDMA allows each MPI process to place row results into a shared memory window from which other processes may directly access only the data which they need to update. Remote ranks perform the MPI_Get operation to access the remote row results and copy only the data they require, directly into main memory reducing communication overhead

by eliminating one memory copy [36]. Furthermore, the total volume of communication required to ensure all processes are up to date can be reduced since each process transfers only the row results they need to update elements in their local dense vector.

In our implementation each process P_i begins by copying updates from P_{i-1}. In doing so, each process is actively performing at most 1 MPI_Get at a time, while simultaneously responding to only one processes get operation. By making these changes, we observed an order of magnitude reduction in communication times, thanks to eliminating wait times due to communication bottle-necking.

V. Evaluation

A. Distributed Performance

When running experiments, we tracked computation and communication times separately. Time measurements for SpMV computation include only the time required to perform the multi-GPU cuSPARSE CSR based SpMV and do not include memory copies of matrix data between host and device, or any result updates via MPI.

Fig. 8 shows the computation and communication times for all tests. The data shown is the average of 10 SpMV iterations per process count and partitioning scheme selected. Computation times decreased for nearly all matrices and partitioning types as $P \to \infty$. Yet the rate at which it decreased was dependent on the matrix and partitioning type selected. Moreover with the exception of com-Friendster, we saw that computation times were nearly identical regardless of matrix partitioning method, and that this behavior was seen among all benchmark matrices. This indicates that our efforts to eliminate communication were not generating computational load imbalances across nodes.

In Figures 8a and 8d we can see that the MOLIERE_2016 and Synthetic matrices did not benefit from hypergraph partitioning with respect to either communication reduction or improved computational performance. Interestingly these two matrices experienced the same behavior for all partitioning methods. Results for the scale-free Com-Friendster graph in Fig. 8b not only show over an order of magnitude lower communication times using hypergraph partitioning with vertex delegation along with the corresponding computation time experiencing a similar reduction compared to other methods. Finally Fig. 8c shows that PGFem3D experienced reduced communication times when using hypergraph partitioning over that of the greedy method. Furthermore computation times were nearly identical for both methods.

As seen in Fig. 9, we observed a wide range of speedup behavior when testing our benchmark matrices using up to 3 workload distribution methods. As MPI process count increases so does the number of Volta v100 GPUs being utilized for computation, at a ratio of 4 GPUs per MPI process. Correspondingly the per GPU problem size decreases as $P/4$ and greatly benefits from the additional parallelism offered by the greater GPU count.

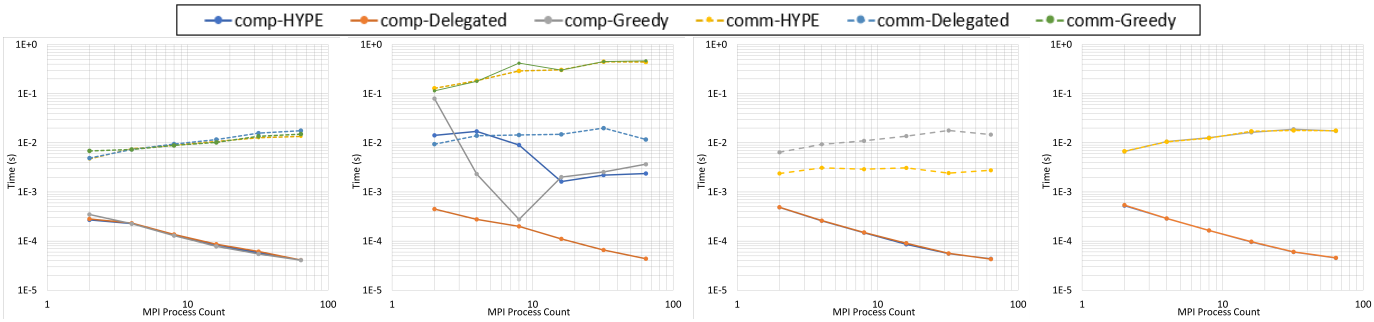
From our results we observed increased performance thanks to the increased parallelism provided by larger GPU counts at scale. In Fig. 9 we can see that nearly all matrices experienced increased speedup as system size increased regardless of partitioning method used. Com-Friendster saw superlinear speedup at $P = 4$ and again at $P = 16$, after which performance began to decline. This behavior was seen for both HYPE and Greedy partitioning of Com-Friendster. Due to the scale-free nature of the Com-Friendster graph the vast majority of rows contain only one edge (non-zero) while a few rows contain millions of edges. The HYPE partitioniner attempts to create a balanced partitioning with respect to vertex count and not the number of non-zeros. It is highly likely that non-zero workload imbalance allows for improved cache performance when access non-zero data for computation. As system size increases non-zero workload becomes increasingly balanced while thread overhead and the lock-step execution penalty degrades performance.

B. Communication Overhead

This study's primary focus was to reduce communication so that improved scalability of hybrid SpMV would be achieved. During testing the RDMA communication method experienced the lowest communication times, regardless of benchmark matrix, partitioning method, or MPI process count, with RDMA being at least 1 order of magnitude and up to 2-3 orders of magnitude faster than MPI_Allgather and MPI_Allreduce respectively, Figs. 8, 10, and 12 plot only the lowest communication measurements which used RDMA.

Fig. 10 illustrates the effectiveness of the hypergraph based partitioning methods approach by showing the average message size reduction compared to the greedy approach. The greedy partitioning method made no effort to reduce communication but rather to balance workload uniformly across all processes with complete disregard for precisely which rows or columns were assigned. Conversely the hypergraph based methods analyze the structure of the matrix in order to perform higher quality assignment of rows/vertices. We saw that all benchmark matrices experienced some level of message size reduction compared to the greedy method when hypergraph partitioning was used.

MPI message latency is a function of message size with larger messages having higher latency. When strong scaling, in an ideal setting, the message size per process pair decreases at scale, allowing lower message latency as process counts increase. Fig. 11 depicts the results of the Ohio State University Mico-Benchmark [37] for MPI_Allgather, MPI_Alreduce, and MPI_Get latency on the Lassen system. In our mutli-GPU with hypergraph partitioning study we used the MPI *get* RDMA operation from every other process, for a total of $P^2 - P$ total RDMA operations. MPI_Get has lower latency than the MPI_Allgather and MPI_Allreduce collectives used in an earlier multi-GPU study on Lassen, except at $P = 2$ and $P = 4$, and even then only at message sizes of less than 8KB. For all other message sizes MPI_Get experienced superior latency.



(a) MOLIERE_2016 (b) com-Friendster (c) PGFem3D Stiffness (d) Synthetic

Fig. 8: Observed Computation and Communication Times. Computation times have solid lines, and communication dashed lines.

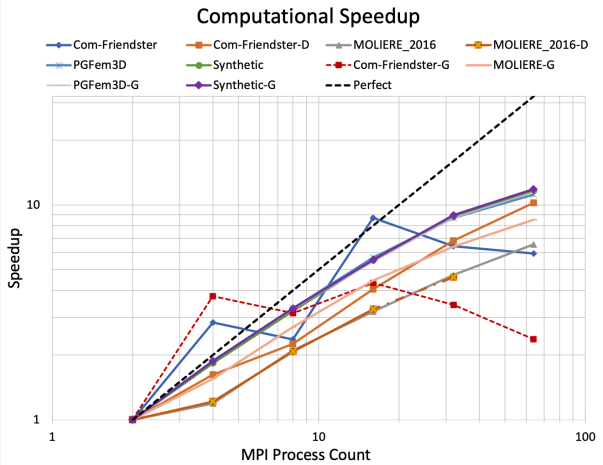


Fig. 9: Computational Speedup. HYPE partitioning is denoted by the matrix name with no trailing character, the *-D* designator denotes hypergraph partitioning with vertex delegation, and *-G* our greedy method. For all tests, MPI process count equals cluster node count.

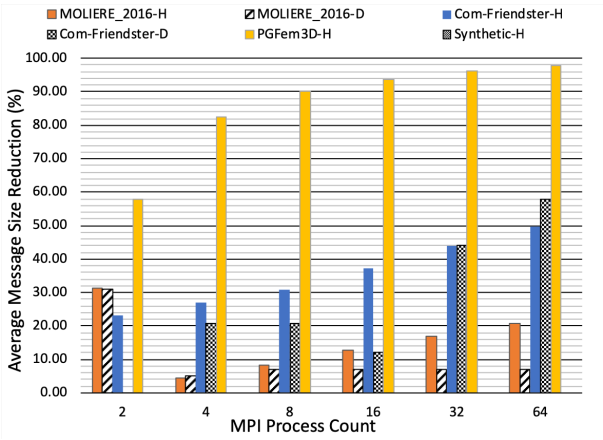


Fig. 10: Average Message Size Reduction (vs Greedy Method). HYPE partitioning is denoted by the *-H* suffix, while *-D* denotes hypergraph partitioning with vertex delegation. Com-Friendster had 0.0% reduction for 2 MPI processes while the synthetic matrix achieved 0.0% reduction in communication for all process counts. For all tests, MPI process count equals cluster node count.

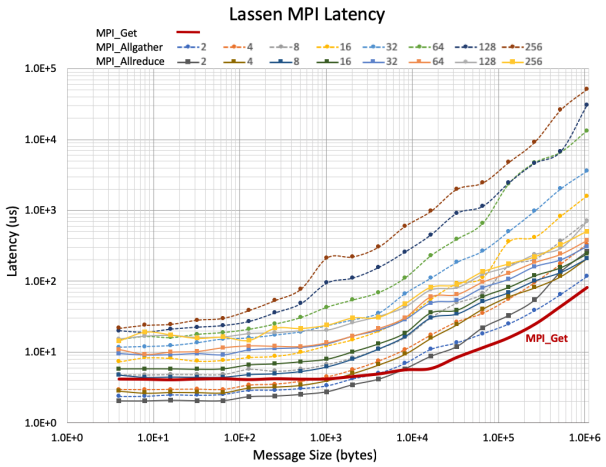


Fig. 11: Lassen MPI Message Latency. Latency benchmark utilized 1 MPI process count per node.

The latency benchmark results on Lassen, Fig. 11 indicate that at a minimum any RDMA message will require approximately 3 microseconds (μs) to send an 1KB message and increasing with message size. With respect to distributed SpMV message size is determined by the number of rows in the benchmark matrix being evaluated.

As seen in Fig. 8 observed computational times were in between 1E-3 and 1E-5 seconds. This means that depending on the input matrix, and MPI process count, its is possible that even a single message could have a latency longer then the compute portion of the SpMV iteration. Additional messaging requirements, either increased message volume or message size, only exacerbate this further. Because of this communication must be reduced nearly entirely to prevent it from driving overall performance.

Fig. 10 shows that for all matrices except for MOLIERE_2016, when using hypergraph partitioning with vertex delegation, we see increased communication reduction as $P \to \infty$. In contrast, the com-Friendster graph which also used vertex delegation was able to reduce average message size per process by 57.8% at $P = 64$. We believe this discrepancy in vertex delegation's effectiveness to be due its non-zero structure. Com-Friendster's scale-free degree distribution means that vertex delegation generates larger numbers of *free vertices* which can be placed on any partition

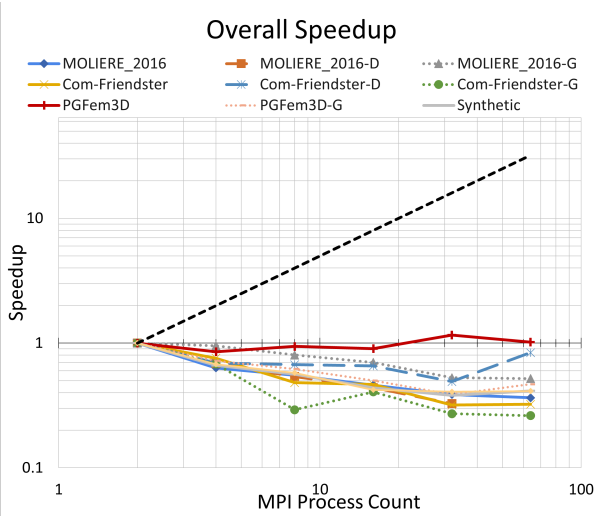


Fig. 12: Overall Speedup (Computation and Communication). HYPE partitioning is denoted by the matrix name with no trailing character, the *-D* designator denotes hypergraph partitioning with vertex delegation, and *-G* our greedy method. For all tests, MPI process count equals cluster node count.

without incurring a communication penalty for row updates. We believe that while MOLIERE_2016 may have a quasi-scale-free degree distribution, its actual structure does not allow for the generation of large numbers of free vertices in the same manner as a com-Friendster. In fact, we can see that MOLIERE_2016 experienced better communication reduction with hypergraph partitioning and no vertex delegation, at nearly all process counts.

Unexpectedly Fig. 10 shows that the PGFem3D stiffness matrix had the best communication reduction for all matrices, process counts, and partitioning types. PGFem3D with hypergraph partitioning had a minimum average message size reduction of 57.83% at $P = 2$ and a maximum of 97.86% at $P = 64$. This means that the average message size between any two processes during the update phase of SpMV has almost been eliminated entirely. Again we believe this to be due to the non-zero structure of the matrix itself. Finite element matrices often represent physical structures which contain highly localized elements, meaning a partitioning method which takes vertex adjacency similarity into account can partition these matrices with very high effectiveness.

Lastly, for all process counts, the synthetic matrix saw no change its average message size per process compared to that of the greedy partitioning method. The random non-zero distribution within each row means that on average every process must retrieve a result for every row assigned to all other processes. Hypergraph partitioning had no effect on this behavior and therefore did not reduce communication for the synthetic matrix.

C. Workload Partitioning Impact

It would be reasonable to surmise that reduced communication along with reduced computation times, would indicate improved performance for SpMV in a distributed environment. As we have seen in previous studies, communication is the dominant factor in overall performance at scale. Fig. 12 indicates the overall speedup for our benchmark matrices across all partitioning methods we evaluated. As can be seen, when communication overhead is taken into account when evaluating performance, nearly all tests performed poorly. This is even more evident when looking at the the computational speedup of MOLIERE_2016 in Fig. 9 vs its overall speedup in Fig. 12. MOLIERE_2016 exhibited substantial computational speedup across all process counts. Despite this it experienced the worst overall performance at scale.

In fact the only test results with a speedup of 1 was for the PGFem3D matrix when partitioned with the HYPE partitioning software. Its performance remained stable as we increased MPI process counts, whereas most other tests saw increasingly poor overall performance. Driving this degradation is the disparity in the time required for each portion of the code. While compute times decrease in general as $P \to \infty$ they do not decrease as fast as communication times are increasing. It may be possible for additional scaling to allow some matrices to experience eventual stabilizing of overall performance. This would likely occur after several orders of magnitude increase in hardware requirements, therefore making it impractical

VI. Conclusion

This study attempted to increase the scalability of SpMV by implementing a distributed heterogeneous and multi-GPU implementation of SpMV code which deliberately reduced inter-process communication. We specifically chose matrices which would require the use of a distributed system to be computed over and thus require the addition of communication overhead as would be seen in real world problems. The computation portion of overall performance did dramatically decrease over prior studies, indicating that the partitioning helped in improving computation time for strong scaling despite the irregular memory access pattern inherent to SpMV.

Regarding communication, we were also successful in reducing communication volume by up to 97% using our partitioning approaches. However, even with the drastic reduction in communication thanks to workload partitioning, overall performance of distributed and or hybrid SpMV continues to be dominated by communication, not computation. The imbalance is off by orders of magnitude.

That being said, in the case of the PgFem3D matrix we did see relatively consistent speedup, though it was only 1X. This is important as all previous studies observed dramatic reductions in overall performance which continued to decrease at scale. Therefore for some data sets it may be possible to increase system size in an effort to accommodate larger matrices without experiencing decreased overall performance.

Lastly these results say something relatively profound about modern high end architectures. With modern computational nodes such as employed here, there is no network interconnect currently in existence which can provide message performance good enough to decrease communication overhead so that it is comparable with computational performance at scale. Since

strong scaling of irregular problems is key to many applications, we have gone far overboard towards the computation side, and need to rethink.

It is apparent now that in the general case the only way to alleviate the performance degradation associated with the strong scaling of sparse problems is to dramatically change, improve, or eliminate communication. New architectures, such as the EMU migrating thread architecture, are being developed which move computation to where the data is stored therby eliminating traditional inter-process messaging and have been able to provide improved scaling performance of SpMV [38].

VII. Acknowledgements

This work was supported in part by the Department of Energy, NNSA, under the Award No. DE-NA0002377 as part of the Predictive Science Academic Alliance Program II, in part by NSF grant CCF-1642280, and in part by the University of Notre Dame. In addition we wish to thank Roger Pearce and the Center for Applied Scientific Computing (CASC) for the use of the Lassen at Lawrence Livermore National Laboratory, provided by the National Nuclear Security Administration's Advanced Simulation and Computing (ASC) program.

References

[1] J. Dongarra and M. Heroux, "Toward a new metric for ranking high performance computing systems," Sandia National Labs, Sandia Report SAND2013 4744, June 2013. [Online]. Available: http://www.sandia.gov/~maherou/docs/HPCG-Benchmark.pdf

[2] R. D. Falgout and U. M. Yang, "hypre: A library of high performance preconditioners," Berlin, Heidelberg, pp. 632–641, 2002.

[3] M. Mosby and K. Matouš, "Hierarchically parallel coupled finite strain multiscale solver for modeling heterogeneous layers," *Int. Journal for Numerical Methods in Engineering*, vol. 102, no. 3-4, pp. 748–765, 2015. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.4755

[4] ——, "Computational homogenization at extreme scales," *Extreme Mechanics Letters*, vol. 6, pp. 68 – 74, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2352431615300134

[5] J. Kepner and J. Gilbert, *Graph Algorithms in the Language of Linear Algebra*, J. Kepner and J. Gilbert, Eds. Society for Industrial and Applied Mathematics, 2011.

[6] [Online]. Available: http://graphblas.org/index.php?title=Graph_BLAS_Forum

[7] B. A. Page and P. M. Kogge, "Scalability of hybrid sparse matrix dense vector (spmv) multiplication," *Int. Conf. on High Performance Computing & Simulation*, Jul 2018. [Online]. Available: http://par.nsf.gov/biblio/10064735

[8] ——, "Scalability of hybrid spmv on intel xeon phi knights landing," *Int. Conf. on High Performance Computing & Simulation*, Jul 2019. [Online]. Available: https://par.nsf.gov/biblio/10109480

[9] M. Naumov, "Incomplete-lu and cholesky preconditioned iterative methods using cusparse and cublas," 2011.

[10] U. V. Catalyurek and C. Aykanat, "Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 7, pp. 673–693, July 1999.

[11] C. Hong, A. Sukumaran-Rajam, B. Bandyopadhyay, J. Kim, S. E. Kurt, I. Nisa, S. Sabhlok, U. V. Çatalyürek, S. Parthasarathy, and P. Sadayappan, "Efficient sparse-matrix multi-vector product on gpus," New York, NY, USA, p. 66–79, 2018. [Online]. Available: https://doi.org/10.1145/3208040.3208062

[12] A. Ashari, N. Sedaghati, J. Eisenlohr, and P. Sadayappan, "An efficient two-dimensional blocking strategy for sparse matrix-vector multiplication on gpus," New York, NY, USA, pp. 273–282, 2014. [Online]. Available: http://doi.acm.org/10.1145/2597652.2597678

[13] E.-J. Im, "Optimizing the performance of sparse matrix-vector multiplication," *thesis, Berkeley*, p. 132, 2000.

[14] M. Martone, S. Filippone, P. Gepner, and M. Paprzycki, "Use of Hybrid Recursive CSR/COO Data Structures in Sparse Matrix-Vector Multiplication," *Computer Science and Information Technology (IMCSIT), Proc. of the 2010 Int. MultiConf. on*, pp. 327–335, 2010. [Online]. Available: http://ieeexplore.ieee.org/xpl/login.jsp?tp=\&arnumber=5680039

[15] B. Yang, S. Gu, T.-x. Gu, C. Zheng, and X.-p. Liu, "Parallel Multicore CSB Format and Its Sparse Matrix Vector Multiplication *," no. 91130024, pp. 1–8, 2014.

[16] N. Sedaghati and S. Parthasarathy, "Characterizing Dataset Dependence for Sparse Matrix-Vector Multiplication on GPUs," pp. 17–24, 2015.

[17] N. Bell and M. Garland, "Efficient Sparse Matrix-Vector Multiplication on CUDA," *Nvidia Technical Report*, pp. 1–32, 2008.

[18] A. Buluç, J. T. Fineman, M. Frigo, J. R. Gilbert, and C. Leiserson, "Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks," pp. 233–244, 01 2009.

[19] X. Sun, Y. Zhang, T. Wang, X. Zhang, L. Yuan, and L. Rao, "Optimizing spmv for diagonal sparse matrices on gpu," pp. 492–501, 09 2011.

[20] F. Vázquez, E. M. Garzon, J. Martinez, and J. J Fernández, "The sparse matrix vector product on gpus," vol. 2, 01 2009.

[21] N. Bell and M. Garland, "Efficient sparse matrix-vector multiplication on CUDA," NVIDIA Corporation, NVIDIA Technical Report NVR-2008-004, Dec. 2008.

[22] S. Lin and Z. Xie, "A jacobi_pcg solver for sparse linear systems on multi-gpu cluster," *The Journal of Supercomputing*, vol. 73, no. 1, pp. 433–454, Jan 2017. [Online]. Available: https://doi.org/10.1007/s11227-016-1887-4

[23] H. Ltaief and S. Tomov, "Gpu technology conf." 2012. [Online]. Available: http://on-demand.gputechconf.com/gtc/2012/presentations/S0042

[24] "Sierra," *Advanced Simulation and Computing*, Jan 2018. [Online]. Available: https://asc.llnl.gov/content/assets/docs/sierra-fact-sheet.pdf

[25] N. Corporation, "Nvidia tesla v100 gpu architecture," NVIDIA Corporation, NVIDIA Technical Report WP-08608-001_v1.1, Aug. 2017. [Online]. Available: https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

[26] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, pp. 1:1–1:25, Dec. 2011. [Online]. Available: http://doi.acm.org/10.1145/2049662.2049663

[27] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999. [Online]. Available: https://science.sciencemag.org/content/286/5439/509

[28] R. Pearce, M. Gokhale, and N. M. Amato, "Faster parallel traversal of scale free graphs at extreme scale with vertex delegates," Piscataway, NJ, USA, pp. 549–559, 2014. [Online]. Available: https://doi.org/10.1109/SC.2014.50

[29] C. Ababei, N. Selvakkumaran, K. Bazargan, and G. Karypis, "Multi-objective circuit partitioning for cutsize and path-based delay minimization," New York, NY, USA, pp. 181–185, 2002. [Online]. Available: http://doi.acm.org/10.1145/774572.774599

[30] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," pp. 343–348, June 1999.

[31] C. Mayer, R. Mayer, S. Bhowmik, L. Epple, and K. Rothermel, "HYPE: massive hypergraph partitioning with neighborhood expansion," *CoRR*, vol. abs/1810.11319, 2018. [Online]. Available: http://arxiv.org/abs/1810.11319

[32] I. Corporation, "Ibm spectrum mpi version 10 release 1.0.2," IBM Corporation, IBM Technical Report GC27-8265-01, Jan. 2016.

[33] J. Pješivac-Grbović, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. J. Dongarra, "Performance analysis of MPI collective operations," *Tertiary Education and Management*, vol. 10, no. 2, pp. 127–143, 2004.

[34] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in MPICH," *Int. Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, 2005.

[35] A. Lastovetsky, M. O'Flynn, and V. Rychkov, "Optimization of collective communications in heteroMPI," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 4757 LNCS, pp. 135–143, 2007.

[36] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda, "High Performance RDMA-Based MPI Implementation over InfiniBand," pp. 295–304, 2003.

[37] [Online]. Available: http://mvapich.cse.ohio-state.edu/benchmarks/

[38] B. A. Page and P. M. Kogge, "Scalability of sparse matrix dense vector multiply (spmv) on a migrating thread architecture," *10th. Int. Workshop on Accelerators and Hybrid Exascale Systems (AsHES)*, May 2020.