Optimizing Communication for a 2D-Partitioned Scalable BFS

Jeffrey Young
School of Computer Science
Georgia Institute of Technology
Atlanta, Georgia, USA
Email: jyoung9@gatech.edu

Julian Romera, Matthias Hauck, Holger Fröning
Institute of Computer Engineering
Ruprecht-Karls University of Heidelberg
Germany
Email: romera@stud.uni-heidelberg.de,
matthias.hauck@sap.com,
holger.froening@ziti.uni-heidelberg.de

***Abstract*—Recent research projects have investigated partitioning, acceleration, and data reduction techniques for improving the performance of Breadth First Search (BFS) and the related HPC benchmark, Graph500. However, few implementations have focused on cloud-based systems like Amazon's Web Services, which differ from HPC systems in several ways, most importantly in terms of network interconnect.**

This work looks at optimizations to reduce the communication overhead of an accelerated, distributed BFS on an HPC system and a smaller cloud-like system that contains GPUs. We demonstrate the effects of an efficient 2D partitioning scheme and allreduce implementation, as well as different CPU-based compression schemes for reducing the overall amount of data shared between nodes. Timing and Score-P profiling results demonstrate a dramatic reduction in row and column frontier queue data (up to 91%) and show how compression can improve performance for a bandwidth-limited cluster.

I. Introduction

Basic graph algorithms support a large number of important search operations like betweenness centrality and shortest paths computations that are used to analyze social networks and provide efficient mappings for transportation routes and networks. Among these fundamental algorithms, Breadth First Search (BFS), is one of the most relevant graph operations due to its prevalence as the base of other search-oriented algorithms. At the same time, BFS is also one of the most difficult graph algorithms to parallelize due to the difficulty of partitioning the initial graph and distributing updates between devices or nodes. This limitation is most clearly seen with level-synchronous BFS, where the search operation proceeds in a synchronous fashion across all involved processes and each iteration requires exchanging an update frontier.

Previous work [1] has looked at techniques to partition the graph, apply accelerators to speed up the search operation [2], and reduce the overall amount of computation [3]. While some of this work has looked at compressing the amount of data to be sent to reduce data transfer time [2], [4], there has not yet been a deep analysis of compression techniques and their effects on the performance of BFS for distributed and low-bandwidth clusters. This analysis is an important part of optimizing BFS for cloud computing-style clusters like Amazon's EC2, which typically eschew high-bandwidth InfiniBand for limited-bandwidth 1 or 10 Gbps Ethernet.

This paper starts with an optimized BFS algorithm designed for a single node with multiple GPUs and then it adds multiple optimizations including an improved 2D partitioning scheme, optimized allreduce phase, and CPU-based compression of the frontier queue. Moreover, this work provides a detailed analysis of several compression techniques and an argument for a specific type of compression to best reduce the size of frontier queue exchanges in a multi-node, distributed BFS implementation. While this paper may not propose the fastest distributed implementation of BFS, we assert that understanding the approaches and costs of compression of data exchange can help improve future implementations that combine multiple optimizations for accelerators and MPI-based clusters.

Specific contributions of this paper include the following:

- A hierarchical 2D/1D partitioning strategy geared towards supporting large GPU clusters
- An exploration of compression techniques and a tailored allreduce variant to reduce data volume
- An evaluation and discussion of our partitioning scheme and data reduction techniques

Our investigation is organized as follows: Section II describes previous optimizations while Section III describes the effects of 2D partitioning and Section IV describes a variable-sized allreduce operation. Section V details several different compression techniques and advocates for the *Frame-of-Reference (FOR)* compression technique to best reduce the size of data communicated between nodes. Experimental results are detailed in Section VI and future extensions are then discussed in Section VII.

II. Background and Related Work

In 2010, the Graph500 benchmark was initially released to help measure the performance of high performance systems on irregular, graph-based algorithms, Breadth First Search (BFS) is the most commonly implemented and reported Graph500 algorithm that is used to compute a system's place in the semi-annual Graph500 listing. Results are measured in terms of time to construct an input graph where the initial set of edges or "scale" is referred to by the exponent in the edge calculation 2^{scale} and also in terms of the time to execute several BFS searches on the resultant, pseudo-random graph. Due to the complexity of performing distributed BFS and low intra-node

TABLE I
FEATURE COMPARISON BETWEEN THIS AND OTHER GRAPH500 WORKS

Optimizations	This Work	SC11	ISC12	SC12	ISC14	SC15
2D decomposition	✓	✓	✓	✓	✓	✓
vertex sorting			✓	✓		
direction optimization					✓	
data compression, GPGPU			✓	✓		
data compression, CPU	✓					✓
SpMV with pop counting					✓	
adaptive data representation					✓	
overlapped communication		✓	✓	✓	✓	✓
shared memory					✓	
shared visited				✓		
GPGPU	✓	✓	✓	✓		
NUMA-aware					✓	

latencies, the systems that fare best at this benchmark tend to be large shared memory machines.

A. Existing Optimizations for Distributed BFS

In Table I we compare this work with more recent state-of-the-art implementations, specifically those for distributed clusters.

The cited works SC11, ISC12, SC12 were proposed by Ueno et al. [5], [6], [2] and were part of a larger effort at TITECH to optimize BFS for a GPU cluster. ISC14, proposed by Yasui et al. [7], is ranked number 1 in the Graph500 as of May, 2016 and focuses its optimizations on CPU clusters. Other notable work includes multiple winning Graph500 runs with the BlueGene/Q system in 2013 and SC15 [8], [9]. BlueGene includes many novel communication mechanisms like wavefront communication and hardware-supported put/get based updates that give it very good performance, but this work does not compare directly with this system since it has many non-commodity hardware features.

The optimization mechanisms referenced in Table I are: *(1) 2D decomposition* to reduce network data traffic, *(2) Vertex sorting* to increase the cache hit ratio when accessing the visited bitmap in the Predecessor reduction phase, *(3) Direction optimization* to optimize the frontier queue strategy (top-down vs. bottom-up) according to the frontier queue size, *(4) Data compression on GPGPU* to decrease the size of data transferred for each iteration as well as GPU-based compression/ decompression routines, and *(5) Data compression on CPU* that uses CPU SIMD-based compression and decompression routines, *(6) Sparse vectors with pop counting* to accelerate Sparse matrix vector (SpMV) multiplications by using *popc* assembly instructions, *(7) Adaptive data representation* to improve data locality by adding an exploration phase, *(8) Overlapped communication* to take advantage of communication waiting time by overlapping it with other computation, *(9) Shared Visited* to reduce the size of the Frontier Queue through the use of a Bitmap, *(10) GPGPU* to use the parallelization provided by General Purpose Graphic Processor Units (GPGPU) to implement the BFS algorithm, and *(11) NUMA-aware*, where the memory used by a search task is bound to the local memory of the related processor.

Our implementation focuses on evaluating the effects of three of these optimizations, GPGPU (10), 2D decomposition (1), and Data compression on CPU (5), so we discuss background and related work for these in more detail here.

B. GPGPU

Initial versions of BFS for GPU focused on algorithms that took advantage of the unique features of one GPU in a node. The work in [10] introduced the concept of hierarchical frontier queues and kernel implementations. Soon after in [11], the concept of user-defined warps was used to map BFS to the GPU while reducing branch divergence for irregular graphs.

Merrill's multi-GPU, single node implementation [12] introduced several new concepts including a hybrid approach to handle latency- or bandwidth-sensitive portions of irregular graphs and an efficient frontier queue expansion and contraction scheme. This particular implementation achieved a single-node performance of 8.3 GTEPS across 4 GPUs. The implementation in this paper is based off of the open-source version of this code.

While Ueno et al's implementation in [2], [6] currently combines the most number of optimizations (and thus typically performs the best for commodity clusters), other important GPGPU related work includes [13] and [14]. In these projects, performance was improved by performing a pruning of next-level frontier set (and resultant inter-node communication) using well-balanced workloads across all threads on a GPU. This optimization reduced the overall communication and enabled their algorithm to scale to 3 GTEPS across 128 GPUs. Later codes using Kepler GPUs achieved 800 GTEPS on 4096 GPUs [15]. As we show in Section VI, while our work does not include all potential optimizations, the evaluation and usage of compression and 2D decomposition can be a key step to improving scalability in clusters.

C. 2D Decomposition

The most important work in decomposing graphs for BFS has been done by Buluc and Madduri [1]. In addition, techniques like using node-local flash memory [16] can be used to support larger graphs and require fewer overall nodes with moderate (~40%) degradation. This work also includes storing the adjacency matrix as a sorted edge list, allowing for an equal number of mostly contiguous edges on each node.

D. Data compression

Recent work has led to the development of a variety of modern and efficient data indexing techniques that operate in near real-time over structures containing large integer sequences.

In this work we will focus on two main families of compression algorithms. These algorithm families are discussed in more detail in [17], but the main families are *Frame-of-Reference (FOR)* and *Variable Byte*. The compression algorithms will also be referred to as *codecs* or *schemes* throughout this document. The FOR family of codecs employs techniques such as *Frame-of-Reference* [18], *Delta compression or differential coding* [19], *Bit packing* [20] and *Patched coding or exceptions* [21]. The *Variable Byte* codec family, known in other literature with names such as as VByte or varint, is based upon the *Prefix suppression (PS)* compression technique [22].

For our compression, we choose the following algorithms from the *FOR* family: *PFOR* [21], *Para-PFOR* [23], *S4-BP128*

[20], [24]. From the *Variable Byte* family, we choose varint-G8IU [25] and maskedVByte [26].

Other schemes such as *Variable-length Quantity (VLQ)* and *Word-aligned Hybrid (WAH)* have special importance as they have been the focus of other compression works in the Graph500 context [6], [2], [27]. The latter is especially interesting as it focuses on *bitmap-based* transfer compression.

III. Hierarchical Partitioning for Scalability

There are two major goals a distributed implementation of an algorithm should achieve: scalability in terms of performance and also in terms of problem size with increased compute node count. For Graph500, we want to be able to tackle larger scale factors and reach higher TEPS rates by using more compute nodes. However, larger scale factors are limited by memory consumption and especially by the limited amount of memory on each GPU. The initial BFS implementation we started from [12] requires two different sets of resident data: the static graph and the working set with the result and supporting data structures of the BFS. The base graph for our partitioned implementation is stored as an adjacency matrix using the compress sparse row (CSR) format.

To partition the input matrix for BFS, 1D or 2D partitioning schemes are typically used. With a 1D scheme, the matrix is sliced by row, and with a symmetric 2D partitioning scheme [1] the matrix can be sliced by both row and column while also trying to balance the amount of graph vertices per node. This partitioning is complicated by the design of commodity GPU clusters with multiple GPUs per node because of their heterogeneous communication layers: internode communication using Ethernet or InfiniBand and intranode communication that currently is built around PCI Express.

We address this heterogeneity by implementing a hybrid 2D/1D partitioning. This means that symmetric 2D partitioning is used to divide the input graph between nodes while 1D partitioning is used within the node to partition data between GPUs. Many earlier-generation GPU systems, such as our test system KIDS, have a non-square number of GPUs per node (3 for KIDS), which limits opportunities for a symmetric 2D partitioning. Thus, our hybrid partitioning scheme manages the challenges of asymmetric, low-device count GPU systems while also allowing us to take advantage of the theoretical guarantees of a symmetric 2D partitioning.

IV. Optimizing Allreduce for Set-Operations

The most important part of the frontier queue communication for a 2D distributed BFS is the column communication. This is because column communication requires a reduction of the frontier queue, as opposed to row communication that can simply be realized by a broadcast. This reduction function combines the frontier queues from all partitions of a column, so that each compute node receives a copy of the complete frontier queue. To reduce work and data volume, vertex IDs need to be unique, which effectively makes the reduction a set-union operation that merges variable-sized frontier arrays and eliminates duplicate IDs.

Unfortunately, MPI collectives currently do not support variable-sized arrays. For this reason, we implemented a specialized allreduce collective that adapts an algorithm made for MPI-style allreduce operations. This variable-sized allreduce distributes the work and communication across the nodes of a column and results in enhanced utilization [28].

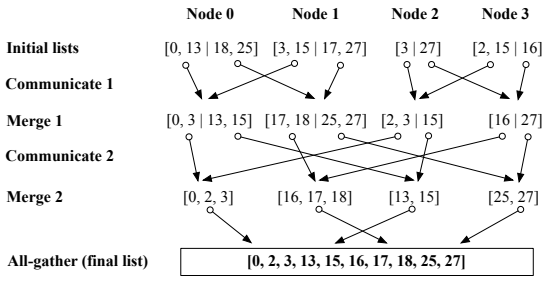


Fig. 1. Variable Allreduce Example

As shown in Figure 1, each node's data array is split in each iteration and then during the main reduction phase, one of the partners processes the lower half while the other partner handles the upper half. At first, direct neighbors communicate, but with every iteration the distance doubles until it is N/2. If the total number of nodes is not a power of two then some nodes will need to send their full array to other nodes so the remaining nodes can partner up.

The consequence of this method is that all N nodes send and receive data in a bidirectional fashion and better utilize the available network bandwidth. In addition, the communication pattern of every node N forms a tree with M levels and N as a root. After every iteration, the size of the processed data chunk halves, and reduced array chunks need to be transferred to the other nodes by an allscatter operation at the end.

To use variable-sized input arrays with a set-union operation, the allreduce algorithm also needs an additional modification: instead of splitting an array in half, the ID range between the global minimum ID and the maximum ID of the communicator needs to be split. In our graph processing setting, the maximum ID is known beforehand because we know the minimum and maximum ID of each column slice. Alternatively, it is also possible to estimate the minimum and maximum IDs by an allreduce min/max operation on the minimum and maximum IDs across the nodes of a communicator.

V. Compression of the Frontier Queue

The achieved data reduction depends on both the characteristics of the source data and the chosen compression algorithm, and the maximum achievable compression is given by Shannon's Entropy, ($H(X)$). The synthetically generated low-degree graphs used in Graph500 enable the use of formats like Compressed Spare Row (CSR) or Compressed sparse column (CSC) to represent the adjacency matrix. These data formats allow SpMV multiplications to generate bitmaps represented as integers. The transmitted frontier queue of the BFS is then composed of a sequence of these integers, and the characteristics of this sequence, including $H(X)$, are a determining factor of compression efficacy. Other important factors include the compression efficacy of the selected algorithm over the source data. As a preliminary analysis of the compression, we intercept a large-sized (30M integers) random frontier queue

TABLE II
CHARACTERISTICS OF THE SAMPLE FRONTIER QUEUE SEQUENCE

	Random sample of a frontier queue
Integer sequence distribution	Uniform (slightly skewed)
Empirical entropy ($H(x)_{empirical}$)	14.58-bit
Integer size	32-bit
Integer Range	[1-65532]
Total integers in sample	29899
Sequence	sorted, increasing order
Mean \ maximum gap distance	2.19 \ 21
Mean \ maximum element in sequence	32789.66 \ 65532

from the distributed BFS transfer phase. Even though a unique trace may not be significant enough to entirely state the nature of the BFS communication, it does give an insight into its numerical characteristics. To perform the analysis we use [29], and as shown in Table II, this distribution is quite similar to a uniform probability distribution.

The analysis of this sample shows an ascending, sorted sequence, that is separated by low-distance gaps but where no integer is dramatically larger than the rest. The empirical entropy of this sample shows a compressibility of $32bit/14.5bit = 45.31\%$. These characteristics match *a priori* the requirements for efficient Frame-of-Reference compression. With the same sample sequence, we run a benchmark to identify compression-ratios and speeds (Table III).

TABLE III
COMPRESSION-RATIO (*ratio*), COMPRESSION & DECOMPRESSION SPEEDS IN MILLIONS OF INTEGERS/SEC (*MI/s*) FOR *FOR* AND *VByte* CODECS

Ratio (%)	bits/integer	C_{speed} MI/s	D_{speed} MI/s	Codec
45.5	*14.5*			$H(x)_{empirical}$
47.2	15.1	$3.207 \cdot 10^3$	$4.701 \cdot 10^3$	S4-BP128 [20]
56.1	17.9	$0.182 \cdot 10^3$	$1.763 \cdot 10^3$	varint-G8IU [25]
68.6	21.9	$0.587 \cdot 10^3$	$0.381 \cdot 10^3$	variableByte [20]
68.6	21.9	$0.629 \cdot 10^3$	$0.282 \cdot 10^3$	maskedvByte [20]
100.0	32.0	$3.399 \cdot 10^3$	$4.535 \cdot 10^3$	memcpy

In Frame-of-Reference, the compression-ratio and speeds are related by gap distances. For *Variable Byte* and *VLQ* codecs, the performance depends on the integer size. Thus, for this random sample with low gap distances and high average sizes the *FOR* S4-BP128 codec outperforms the *VByte* codecs, leading to our further evaluation using S4-BP128.

VI. PERFORMANCE ANALYSIS

A. Experimental Setup

The BFS tests were performed on two GPU clusters, Creek at the University of Heidelberg, and the Keeneland Initial Delivery System (KIDS) at Georgia Tech. Creek has a single socket CPU, so its two GeForce GPUs can be used at the same time by exchanging data with peer-to-peer memory accesses via NVIDIA's GPUDirect [30], [31]. On the other hand, Creek only has a 1 Gbps Ethernet link, which limits performance for multi-node experiments. KIDS is composed of 110 HP SL390 nodes, and each node contains two Westmere CPUs and 3 M2090 Fermi GPUs. All nodes within the KIDS system are connected by QDR InfiniBand. Tests were run using CUDA 6.5, GCC 4.8.2, and zone analysis was primarily performed using the Score-P analysis toolkit [32].

While latency and bandwidth constraints for PCIe-based devices and slower inter-node networks are the main limiting factors for GPU-based BFS, other hardware factors also come into play within a node. Each node in the Keeneland test system has two CPU sockets, and each socket also has its own PCI Express hub and is connected by Intel's proprietary QPI interconnect. The implications of this type of architecture are two-fold: 1) The PCI Express bus limits the available performance for a hybrid CPU/GPU system because the bus's bandwidth is vastly slower than the compute and memory bandwidth of the GPU. 2) The heterogeneity of the architecture limits the effectiveness of GPU-specific data movement optimizations like NVIDIA's GPUDirect [30] and GPUDirect RDMA [31]. The QPI link connecting each socket means that many data transfers must interact with the CPU and host operating system rather than being able to transfer data directly between GPUs. Optimizations performed in Merrill's single-node version were highly dependent on using GPUDirect, so to avoid these issues many of our tests use a single GPU per node or avoid using GPUDirect for intra-node data exchange.

B. Effects of 2D Partitioning on Scalability

The first set of results look at the effect of the added 2D partitioning strategy and the optimized allreduce operation. Figures 2 and 3 show the phase breakdown for BFS, scale 23, and overall TEPS scaling for 4 to 49 nodes on KIDS. While the scale 26 size graph achieves up to 1.4 GigaTEPS,

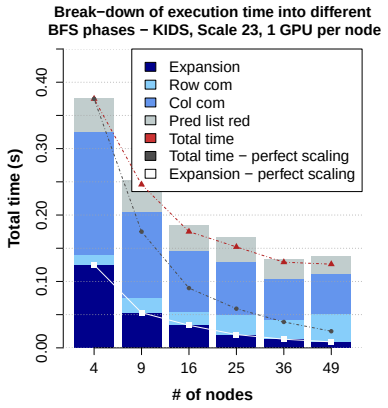


Fig. 2. BFS - Time breakdowns with 2D partitioning

or GTEPS, the phase breakdown for the 2D partitioned graph demonstrates that at lower node counts the expansion on GPU dominates the runtime and at higher node counts the row and column communication time dominates the overall runtime. These limits on TEPS scaling show that additional techniques are required to reduce the amount of intra-node data movement.

C. Effects of Compression Techniques

As mentioned previously, an initial analysis of compression techniques showed that the S4-BP128 codec provides the best tradeoff of compression/decompression time and overall compression ratio. By using Score-P to annotate timing regions for compression, decompression, and MPI communication calls, we can illustrate how effective this CPU-based codec is for reducing data size and data movement.

In Table IV, 36 node and 49 node runs both have a reduction in data size of approximately 85% while the 64 node runs reduce the amount of row and column data by 91% before transferring it to other nodes. While the row and column data

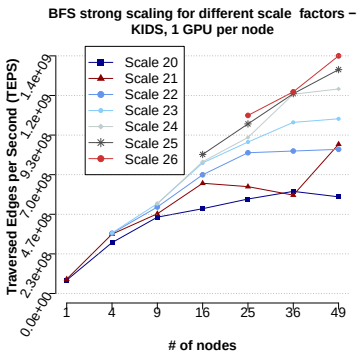


Fig. 3. BFS - TEPS with 2D partitioning

TABLE IV
KIDS - REDUCTION OF TRANSFERRED DATA FOR ROWS AND COLUMNS, VARYING THE NUMBER OF NODES

	data (bytes)	compressed (bytes)	reduction (%)
36 gpus			
Scale22	$1.80311 \cdot 10^{10}$	$2.521333 \cdot 10^{9}$	86.01
Scale27	$4.71471 \cdot 10^{11}$	$6.638748 \cdot 10^{10}$	85.91
49 gpus			
Scale22	$2.08890 \cdot 10^{10}$	$3.112144 \cdot 10^{9}$	85.10
Scale27	$5.47627 \cdot 10^{11}$	$8.312558 \cdot 10^{10}$	84.82
64 gpus			
Scale22	$2.37315 \cdot 10^{10}$	$2.032943 \cdot 10^{9}$	91.43
Scale27	$6.21408 \cdot 10^{11}$	$5.576142 \cdot 10^{10}$	91.02

size is reduced by 91% on average across all node sizes, the predecessor reduction phase remains uncompressed. For example, for Scale 27 the column communication size is reduced from $3.631 \cdot 10^{11}$ to $3.323 \cdot 10^{10}$ bytes while the reduction phase stays constant at $5.396 \cdot 10^{11}$ bytes. As Figure 4(a) and 4(b) demonstrate on the Creek system, this means that as the row and column communication time is reduced, the predecessor reduction time takes a larger portion of the overall runtime. On KIDS with Ethernet networking, 64 GPUs and Scale 27 runs show that row communication time is reduced from 9294.10 to 2259.14 seconds (75.69%), column communication time is reduced from 32,486.71 to 4370.89 seconds (86.54%), but there is only a .24% reduction in the predecessor reduction time (81251.16 to 81050.82 seconds).

However, looking at Figures 4(c) and 4(d) show that runtime *increases* with compression-optimized communication over InfiniBand. While both versions are substantially faster than using Ethernet for all communication, the time for row communication with Scale 27 and 64 nodes increases by 9.0% (3485.29 to 3833.48 seconds), column communication increases by 18.6% (1356.49 to 1666.83 seconds), and predecessor reduction time increases by 88.1% (1031.0 to 8721.87 seconds) in the compressed versus uncompressed scenario.

To further investigate this, we look more closely at the Score-P profiling data and the actual time for compression and decompression operations on each node. Figure 5(a) clearly illustrates the issue where compression is responsible for a 76% to 93% reduction in overall runtime when using Ethernet while compression results in a 5% to 24% increase in runtime when InfiniBand is deployed. Both scenarios 5(b) have a 91% reduction in the amount of row and column data being sent over each network, but a deeper analysis shows that the compression and decompression time is relatively low at 20% for Scale 27 and 64 nodes. In this scenario, 111.75 seconds were used for the actual encoding out of a total of 670.40 seconds for compression-related calls over the full 64 BFS iteration run. 80% of the compression overhead is due to data type conversions required before compression and after decompression. This overhead is much less than the penalty imposed by data movement on slow networks like 1 Gbps Ethernet, but it quickly becomes a bottleneck when a faster 40 Gbps network is used.

Data type conversion overheads occur due to the FOR codec we selected and the design of the GPU BFS code itself. The GPU BFS code uses 64-bit signed integers to support large numbers of vertex labels and to represent unvisited vertices as negative values. However, the FOR codec we selected for our evaluation requires unsigned 32-bit integers. Each compression/decompression cycle incurs this conversion cost, which leads to large overheads that impact scalability.

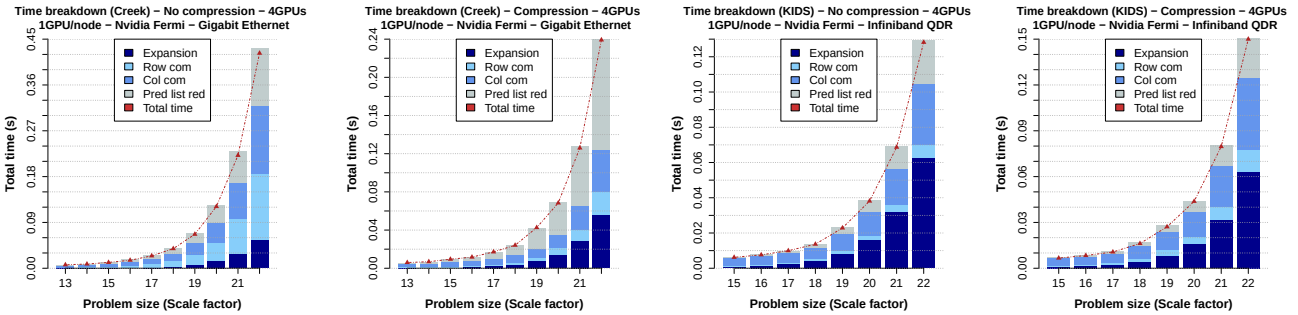
This conversion overhead does not hinder improved TEPS and timing for Ethernet systems but does reduce overall TEPS for InfiniBand systems. Figures 6(a) and 6(c) show an improvement in overall TEPS and reduction of runtime where the 4 node Creek system with a Scale 22 input can achieve up to .28 GTEPS and complete in 240 ms as compared to .16 GTEPS and 430 ms for the uncompressed version. For the same scale and 4 nodes on KIDS, the uncompressed code achieves .52 GTEPS while the compressed version tops out at .45 GTEPS. For a Scale 27 graph and 64 nodes (not shown), the compressed code reaches 2.58 GTEPS but the uncompressed version reached 3.26 GTEPS.

VII. Future Work and Optimizations

The addition of the proposed partitioning and compression optimization schemes provides a large benefit for clusters with limited-bandwidth networks. However, there are limitations that could be improved by building on recent related research.

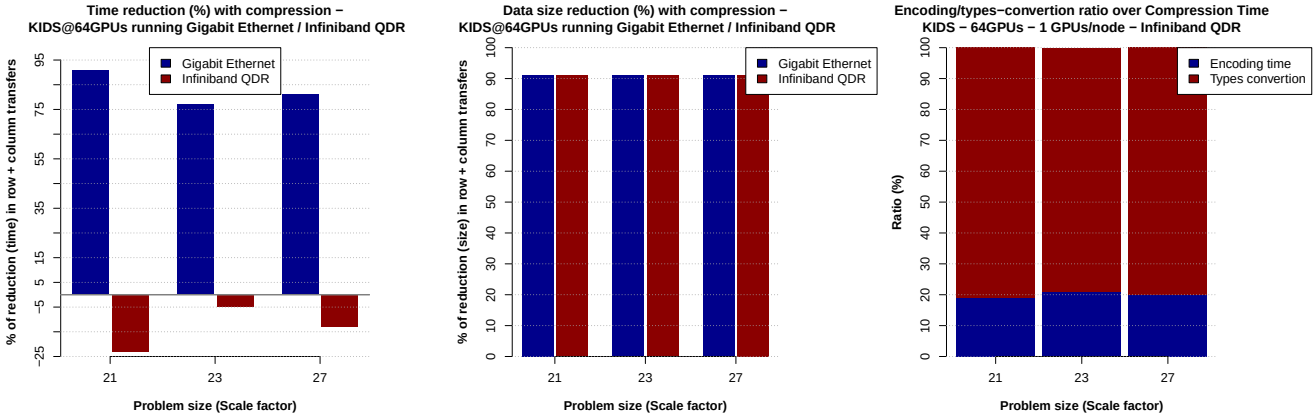
Our top priority would be to try to match our BFS 64-bit integer data types for vertex labels with a 64-bit compression codec to avoid the overhead seen in results on the KIDS cluster. In addition, by starting from an distributed systems algorithm like [2], we could avoid some of the encountered issues in our base, single-node code like limited size scalability and memory overheads. By using a NUMA-aware library [7], we could further optimize data movement between PCI Express and Ethernet, InfiniBand, or NVLink interconnects. Additionally, the custom allreduce that is used in this work has a corner case where it requires additional handling if the $Rows \cdot Columns$ is an odd number. A recent, GPU-related algorithm [33] is likely to provide performance improvement as it does not have to handle this same corner case.

Finally, one of the most promising related developments to this work focuses on compressing the graph itself [34]. Ligra+ is currently implemented on shared memory systems, but it is likely that its graph compression techniques could augment our proposed compression techniques to increase each GPU's data set size and to further reduce data movement time.



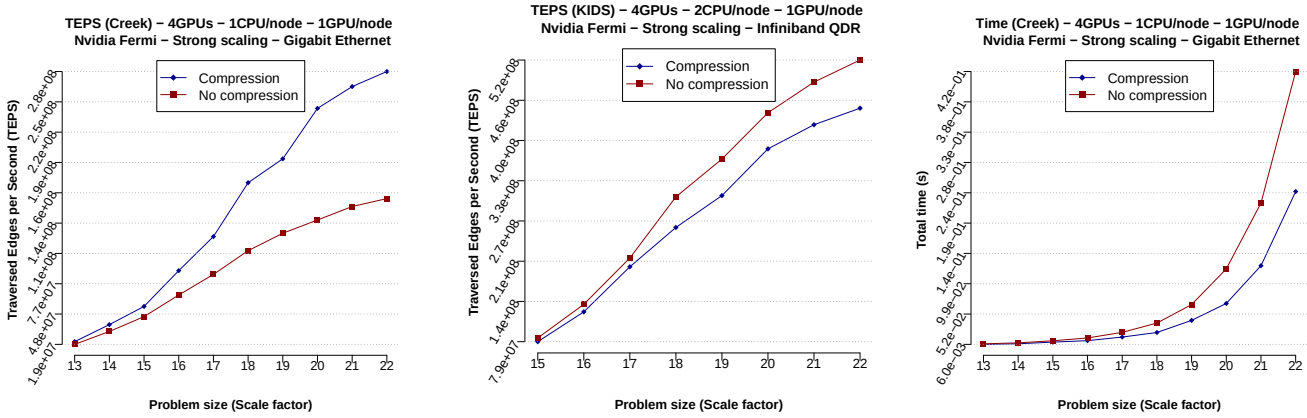
(a) Gig. Ethernet. No compression (b) Gig. Ethernet. Compression (c) InfiniBand. No compression (d) InfiniBand. Compression

Fig. 4. Time Breakdowns on Creek and KIDS Clusters



(a) time reduction (%) (b) size reduction (%) (c) Real encoding time / types conversion ratio

Fig. 5. Compression Time, Data, and Overhead Breakdowns



(a) TEPS. Gigabit Ethernet (Creek) (b) TEPS. InfiniBand (KIDS) (c) Time. Gigabit Ethernet (Creek)

Fig. 6. Impacts of Compression on TEPS

VIII. Conclusion

This work presents an investigation into the design and potential performance limiters for a 2D distributed, GPU-based implementation of BFS. Results on a 4 node and 64 node GPU cluster indicate that compression is very important in reducing communication and data movement between nodes, especially for low-bandwidth networks. For high-performance networks, however, we observe that the time required for compression can outstrip the savings from reduced data transmission time. We pointed out several techniques that would help to address this computational overhead, including performing compression on the GPU to address the limited-bandwidth PCIe interconnect. Additionally, dynamic compression selection could be implemented in conjunction with a performance model of compression overheads, thus applying compression only when beneficial for a given situation.

REFERENCES

[1] A. Buluc and K. Madduri, "Parallel breadth-first search on distributed memory systems," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. New York, NY, USA: ACM, 2011, pp. 65:1–65:12. [Online]. Available: http://doi.acm.org/10.1145/2063384.2063471

[2] K. Ueno and T. Suzumura, "Parallel distributed breadth first search on gpu," in *High Performance Computing (HiPC), 2013 20th International Conference on*, dec 2013, pp. 314–323.

[3] S. Beamer, K. Asanović, and D. Patterson, "Direction-optimizing breadth-first search," *Scientific Programming*, vol. 21, no. 3-4, pp. pp. 137–148, 2013.

[4] H. Lv, G. Tan, M. Chen, and N. Sun, "Compression and Sieve: Reducing Communication in Parallel Breadth First Search on Distributed Memory Systems," *ArXiv e-prints*, aug 2012.

[5] K. Ueno and T. Suzumura, "2d partitioning based graph search for the graph500 benchmark," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, may 2012, pp. 1925–1931.

[6] K. Ueno, T. Suzumura, and Toyotaro, "Highly scalable graph search for the graph500 benchmark," in *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '12. New York, NY, USA: ACM, 2012, pp. 149–160. [Online]. Available: http://doi.acm.org/10.1145/2287076.2287104

[7] Y. Yasui and K. Fujisawa, "Fast and scalable numa-based thread parallel breadth-first search," in *High Performance Computing Simulation (HPCS), 2015 International Conference on*, jul 2015, pp. 377–385.

[8] F. Checconi and F. Petrini, "Massive data analytics: The graph 500 on ibm blue gene/q," *IBM J. Res. Dev.*, vol. 57, no. 1, pp. 111–121, Jan. 2013. [Online]. Available: http://dx.doi.org/10.1147/JRD.2012.2232414

[9] X. Que, F. Checconi, F. Petrini, X. Liu, and D. Buono, "Exploring network optimizations for large-scale graph analytics," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: ACM, 2015, pp. 26:1–26:10. [Online]. Available: http://doi.acm.org/10.1145/2807591.2807661

[10] L. Luo, M. Wong, and W. M. Hwu, "An effective GPU implementation of breadth-first search," in *Proceedings of the 47th Design Automation Conference*, ser. DAC '10. New York, NY, USA: ACM, 2010, pp. 52–55. [Online]. Available: http://doi.acm.org/10.1145/1837274.1837289

[11] S. Hong, S. K. Kim, T. Oguntebi, and K. Olukotun, "Accelerating CUDA graph algorithms at maximum warp," in *Proceedings of the 16th ACM symposium on Principles and practice of parallel programming*, ser. PPoPP '11. New York, NY, USA: ACM, 2011, pp. 267–276. [Online]. Available: http://doi.acm.org/10.1145/1941553.1941590

[12] D. Merrill, M. Garland, and A. Grimshaw, "Scalable gpu graph traversal," in *ACM SIGPLAN Notices*, vol. 47, no. 8. ACM, 2012, pp. 117–128.

[13] M. Bernaschi, M. Bisson, E. Mastrostefano, and D. Rossetti, "Breadth first search on APEnet+," in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, nov 2012, pp. 248–253.

[14] E. Mastrostefano and M. Bernaschi, "Efficient breadth first search on multi-GPU systems," *J. Parallel Distrib. Comput.*, vol. 73, no. 9, pp. 1292–1305, sep 2013. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2013.05.007

[15] M. Bisson, M. Bernaschi, and E. Mastrostefano, "Parallel distributed breadth first search on the kepler architecture," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, no. 99, pp. 1–1, 2015.

[16] R. Pearce, M. Gokhale, and N. M. Amato, "Scaling techniques for massive scale-free graphs in distributed (external) memory," in *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, ser. IPDPS '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 825–836. [Online]. Available: http://dx.doi.org/10.1109/IPDPS.2013.72

[17] J. Romera, "Optimizing communication by compression for multi-gpu scalable breadth-first search," Master's thesis, Ruprecht-Karls University of Heidelberg, 2016.

[18] J. Goldstein, R. Ramakrishnan, and U. Shaft, "Compressing relations and indexes," in *Data Engineering, 1998. Proceedings., 14th International Conference on*, feb 1998, pp. 370–379.

[19] W. K. Ng and C. V. Ravishankar, "Block-oriented compression techniques for large statistical databases," *IEEE Trans. Knowl. Data Eng.*, vol. 9, no. 2, pp. 314–328, 1997. [Online]. Available: http://dx.doi.org/10.1109/69.591455

[20] D. Lemire, L. Boytsov, and N. Kurz, "SIMD Compression and the Intersection of Sorted Integers," *Software: Practice and Experience*, vol. 46, no. 6, pp. 723–749, jun 2016.

[21] M. Zukowski, S. Heman, N. Nes, and P. Boncz, "Super-scalar ram-cpu cache compression," in *Data Engineering, 2006. ICDE '06. Proceedings of the 22nd International Conference on*, apr 2006, pp. 59–59.

[22] M. Zukowski, "Balancing vectorized query execution with Bandwidth-Optimized storage," Ph.D. dissertation, Universiteit van Amsterdam, sep 2009.

[23] N. Ao, F. Zhang, D. Wu, D. S. Stones, G. Wang, X. Liu, J. Liu, and S. Lin, "Efficient parallel lists intersection and index compression algorithms using graphics processing units," *Proc. VLDB Endow.*, vol. 4, no. 8, pp. 470–481, may 2011. [Online]. Available: http://dx.doi.org/10.14778/2002974.2002975

[24] D. Lemire and L. Boytsov, "Decoding billions of integers per second through vectorization," *Software: Practice and Experience*, vol. 45, no. 1, pp. 1–29, jan 2015.

[25] A. A. Stepanov, A. R. Gangolli, D. E. Rose, R. J. Ernst, and P. S. Oberoi, "Simd-based decoding of posting lists," in *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, ser. CIKM '11. New York, NY, USA: ACM, 2011, pp. 317–326. [Online]. Available: http://doi.acm.org/10.1145/2063576.2063627

[26] J. Plaisance, N. Kurz, and D. Lemire, "Vectorized vbyte decoding," in *Proceedings of the First International Symposium on Web Algorithms*, ser. SWIG'15, 2015. [Online]. Available: http://arxiv.org/abs/1503.07387

[27] H. Lv, G. Tan, M. Chen, and N. Sun, "Compression and Sieve: Reducing Communication in Parallel Breadth First Search on Distributed Memory Systems," *ArXiv e-prints*, aug 2012.

[28] R. Rabenseifner, "A new optimized mpi reduce algorithm," High-Performance Computing-Center, University of Stuttgart, nov 1997. [Online]. Available: http://www.hlrs.de/mpi/myreduce.html

[29] M. L. Delignette-Muller and C. Dutang, "fitdistrplus: An R package for fitting distributions," *Journal of Statistical Software*, vol. 64, no. 4, pp. 1–34, 2015. [Online]. Available: http://www.jstatsoft.org/v64/i04/

[30] G. Shainer, A. Ayoub, P. Lui, T. Liu, M. Kagan, C. R. Trott, G. Scantlen, and P. S. Crozier, "The development of mellanox/NVIDIA GPUDirect over InfiniBand: a new model for GPU to GPU communications," *Journal of Computer Science - Research and Development*, vol. 26, no. 3-4, pp. 267–273, jun 2011. [Online]. Available: http://dx.doi.org/10.1007/s00450-011-0157-1

[31] S. Potluri, K. Hamidouche, A. Venkatesh, D. Bureddy, and D. Panda, "Efficient inter-node MPI communication using GPUDirect RDMA for InfiniBand clusters with NVIDIA GPUs," in *2013 42nd International Conference on Parallel Processing (ICPP)*, oct 2013, pp. 80–89.

[32] A. Knüpfer, C. Rössel, D. Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. Malony *et al.*, "Score-p: A joint performance measurement run-time infrastructure for periscope, scalasca, tau, and vampir," in *Tools for High Performance Computing 2011*. Springer, 2012, pp. 79–91.

[33] I. Faraji and A. Afsahi, "GPU-Aware intranode MPIAllreduce," in *Proceedings of the 21st European MPI Users' Group Meeting*, ser. EuroMPI/ASIA '14. ACM, 2014, pp. 45:45–45:50.

[34] J. Shun, L. Dhulipala, and G. E. Blelloch, "Smaller and faster: Parallel processing of compressed graphs with ligra+," in *Data Compression Conference (DCC), 2015*. IEEE, 2015, pp. 403–412.