Scalability of Hybrid SpMV on Intel Xeon Phi Knights Landing

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*—SpMV, the product of a sparse matrix and a dense vector, is emblematic of a new class of applications that are memory bandwidth and communication, not flop, driven. Sparsity and randomness in such computations play havoc with performance, especially when strong, instead of weak, scaling is attempted. In this study we develop and evaluate a hybrid implementation for strong scaling of the Compressed Vectorization-oriented sparse Row (CVR) approach to SpMV on a cluster of Intel Xeon Phi Knights Landing (KNL) processors. We show how our hybrid SpMV implementation achieves increased computational performance, yet does not address the dominant communication overhead factor at extreme scale. Issues with workload distribution, data placement, and remote reductions are assessed over a range of matrix characteristics. Our results indicate that as $P \rightarrow \infty$ communication overhead is by far the dominant factor despite improved computational performance.**

***Keywords*-Scalability, KNL, Hybrid SpMV, Communication Overhead, HPC, MPI;**

I. Introduction

Dense linear algebra boasts well documented efficient algorithms and performance models for nearly all modern architectures. In contrast, sparse linear algebra operations remains 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. 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] for finite 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], [6], and is a key function in the GRAPHBLAS spec [7].

An earlier study [8] looked at strong scaling of SpMV in a hybrid Message Passing Interface (MPI) + Multi-threading (OpenMP) environment on a conventional processor multi-node system. This work showed that when strong scaling is attempted (fixed matrix but increasing processor count), SpMV performance *degradation* (not speedup) can be seen for all matrices, often with relatively few processes. While strong scaling increased the performance of the computational part of the code, the impact of network communication among participating MPI processes drastically reduced any overall speedup. Fig. 1 (taken from [8]) illustrates these effects on overall kernel performance for all matrices tested.

Many-core architectures such as graphics processing units (GPUs) and Intel Xeon Phi Knights Landing (KNL) have been shown to exhibit superior performance for sparse kernels due to greater parallelism, lower memory access times, and increased memory bandwidth [9], [10], [11], [12], [13]. A new storage format Compressed Vectorization-oriented sparse Row (CVR) [14] has exhibited superior performance on KNL by addressing the memory locality. Our goal for this study was therefore to explore strong scaling of a hybrid code on KNL using CVR as well as to analyze the impact of communication overhead on performance for extremely sparse matrices, especially considering the difference in networks. Our SpMV Implementation was run on extremely sparse yet relatively small matrices in order to simulate and observe the effects of extreme strong scaling while utilizing more manageable data set and cluster sizes.

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

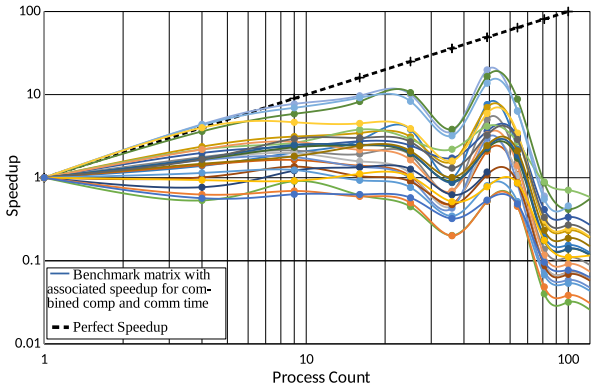


Fig. 1: Observed speedup for 25 Sparse Matrices at varying MPI process counts. As detailed in [8], matrices with higher non-zero per row counts achieve higher performance, while lower non-zero per row count matrices experienced the lowest. The distinct dips and peaks are attributed to various network effects at scale, and were observed for all matrices evaluated.

II. Background

A. The Intel Xeon Phi Processor

Intel Xeon Phi product line exhibited an exploration into intermediate and high band width memory coupled with a many core vector processing unit design. This is not unique to KNL, but is becoming a widespread trend. The Knights Landing is the latest generation Intel Xeon Phi line. KNL utilizes hierarchical memory consisting of double data rate (DDR) memory, as well as multichannel DRAM (MCDRAM). MCDRAM acts as configurable high bandwidth memory (HBM) capable of operating in one of three modes: cache, flat, and hybrid. The MCDRAM consists of 8 units of 2GB each, providing a total of 16GB of HBM. The on board HBM can deliver very high bandwidths of over 450GB/s [9].

In addition to 3D stacked high bandwidth memory, the Xeon Phi product line previewed important architectural features such as AVX-512 floating point units, the on die mesh interconnect, and integration of Omni-Path controllers. Three of the features explored in the Xeon Phi series have already been implemented in traditional Xeon processors, while 3D memory is a likely candidate for inclusion into the Ice Lake generation of Xeons. In light of this, the investigation of sparse problems on KNL allows for behavior analysis on a system which incorporates all of these design features into a single package, and is presently available to researchers.

In this study we evaluated an SPMV implementation developed in an effort to optimize the use of KNLs vector processing units along with minimizing cache miss percentage and associated penalties.

B. SpMV on Many Core Architectures

In previous studies SpMV has been shown to be memory bound on conventional architectures. Most implementations suffered from poor cache hit ratios with the memory traffic for each non-zero being approximately 20 bytes for two floating point operations [15]. This memory bound characteristic severely limits the computational performance of SpMV regardless of a given system's core flop capability. There is a continued push towards greater core counts, for example the Sunway TaihuLight's architecture contains 256 cores per chip. 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 many-core platforms, with the majority exploring the impact of novel matrix compression techniques. For instance ELLPACK and Hybrid [16], [11], originally developed for use with GPUs, have exhibited greatly improved performance over that of the traditional Compressed Sparse Row (CSR) format. Such methods capitalize on the architectural nuances these platforms provide [17], [12]. However these methods focus on single node or shared memory implementations and do not account for strong scaling ability via distributed or hybrid memory environments.

Similarly, previous implementations of SpMV on single-node KNL systems account for the structural mechanisms as well as wide vector instructions intended to increase parallelism and memory access efficiency [18], [19], [14].

While KNL and GPUs share many similarities and are both highly multithreaded, the Compressed Vectorization-Oriented Sparse Row method, discussed in Section III, was written specifically for KNL and the use of AVX-512 extensions. Furthermore in many common implementations, threads within a GPU warp operate over the same instruction queue in lock step, meaning that branching can produce sub-optimal performance among other threads in the warp. In contrast, KNL does not have this constraint as each core supports 4-way simultaneous multithreading (SMT) and is accompanied by 2 vector processing units. This makes it more likely to be an advantage to sparse problems where there is a lot of irregular, unaligned accesses. In this study, it did reduced computation time over that of Xeon servers without it.

C. Workload Imbalance

In general, the effect of matrix structure on performance arises due to work load imbalance among participating computational elements. The benchmark matrices selected (see Section IV) were chosen in an effort to provide a wide array of structural patterns on which to evaluate performance at scale.

In [8] we evaluated the impact of decreased communication overhead as a result of near uniform workload balancing. However one challenge for many-core architecture systems is utilizing the increased per-process compute capacity. Determining optimal load balancing schemes is vital to performance, however determining true optimality is beyond the scope of this paper.

D. Communication Overhead

Communication overhead impact is a complex issue and is rooted in the system hardware, interconnection hardware, as well as communication and user level protocols being used. Both GPUs and KNL experience increased overhead thanks to greater volumes of data required for each device, due to the greater level of concurrency afforded by the hardware. MPI overhead is a well researched area and continues to be of importance as we look towards future HPC systems.

Studies [20], [21] have shown that message latency and effective message bandwidth when using MPI is dependant on many factors such as message size, interconnect type, protocol selection, etc. These studies observed that larger messages can obtain higher network bandwidth utilization due to amortization of their overhead. It is possible that this has increased impact on many-core platforms due to larger messages and reduced process and/or node counts.

Computing paradigms using offloading and onloading work similarly. In Offloading communication is pushed onto the interconnect hardware for completion, allowing the process to continue. Conversely with onloading; communication overhead is handled by a selected number of threads, cores, or sockets in an effort to alleviate the impact of high bandwidth communication on the NIC or interconnect interface [21]. Onloading is often used on KNL systems in which sacrificing

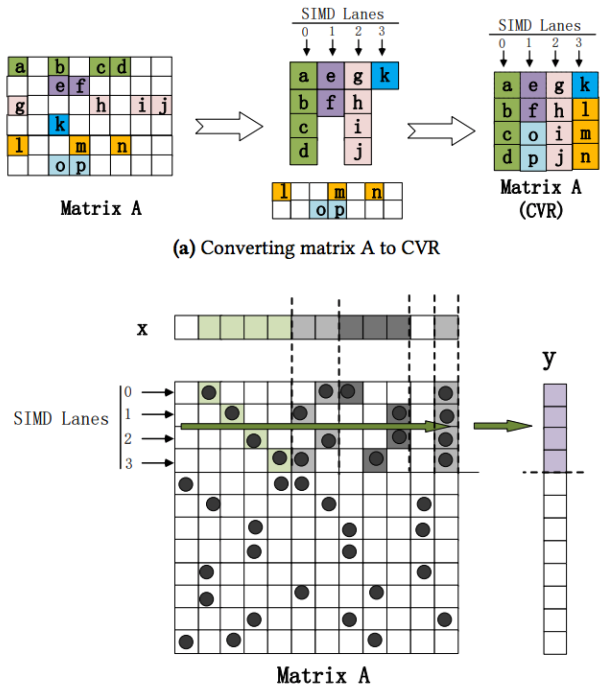


Fig. 2: Conversion of sparse matrix into compressed vectorization-oriented sparse row format. (a) illustrates the packing of contiguous rows into SIMD lanes, while (b) shows the compute pattern of SpMV using a CVR matrix. Conversion visualization obtained from [14].

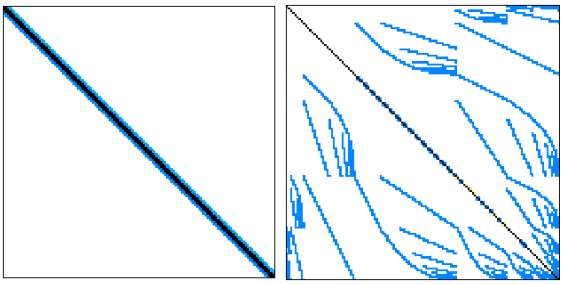
cores for communication handling rather than computation may actually increase overall computational performance [22], [23].

III. Compressed Vectorized sparse Row (CVR)

CVR is a method developed for the KNL to take advantage of the larger instruction width and increased parallelism possible. Specifically designed for use on KNL systems, CVR targets the efficient vectorization of SpMV computation. Similar to a previous study (ELLPACK Sparse Block (ESB) [20], [21]), CVR addresses memory locality issues experienced with SpMV [14] and it is capable of computing over multiple rows concurrently thereby reducing cache miss percentage and its associated performance penalty.

As with most SpMV solvers, the sparse matrix undergoes pre-processing in which it is converted to the desired format for that method. As seen in Fig 2, CVR packs contiguous rows, when possible, into a single column for processing on a single SIMD lane. The number of columns in the CVF format is equivalent to the number of SIMD lanes, 8 in the case of double precision non-zero values. The 8 SIMD lanes permit simultaneous multiply and accumulation to be applied to 8 rows at a time. Besides pre-processing the matrix, CVR must also re-arrange the dense vector to present appropriate vector elements to the correct lanes at the time they are required.

SIMD lanes are packed evenly with the same nnz, performing work stealing when necessary to spread larger rows across multiple lanes. While this process does require the introduction of zero elements for padding, the total volume of padding



(a) atmosmodd (b) parbolic_fem

Fig. 3: Varying Matrix Non-Zero Structure was selected for the benchmark matrices chosen in our experiments. Distribution visualization provided from [24].

is kept to a minimum and facilitates increased performance. We notice then, that this pre-processing step represents a non-trivial amount of time, however the first portion of our study we assume that SpMV is part of a larger solver in which this pre-processing is only performed once .

CVR has been shown to outperform other commercially available and state-of-the-art methods, including MKL and ESB [20], [21]. In light of these findings we selected the CVR SpMV [14] method as the basis for the intra-node processing in our hybrid implementation on a multi-node KNL platform.

IV. Matrix Benchmark Suite

Similar to our previous study we have chosen 25 matrices from the SuiteSparse Matrix Collection[1] [24]. Emphasis was on matrices with either extreme sparsity or irregular patterns. Table I lists their characteristics. Matrices were selected based on their average non-zeros per row nnz_{row}, as well as overall non-zeros. At least two matrices from each nnz range were chosen with similar size or nnz per row but different structure.

Fig. 3 highlights the structure disparity between two of these matrices: *atmosmodd* and *parabolic_fem*. The non-zeros in *atmosmodd* cluster along the main diagonal, whereas *parabolic_fem* has a wider dispersion. We focused on structural differences to evaluate the impact of matrix structure on communication volume and message size across different load balancing methods. Additionally we chose matrices in a quasi logarithmic fashion to ensure a wide spectrum of sparsities.

Not only where these matrices selected due to the rationale outlined previously, but also because these matrices provide a wide range of characteristics while still being small enough to perform rapid file I/O and load balancing for each test. We are aware of much larger matrices which must be evaluated on distributed systems due to their immense size requirements. However evaluating such matrices is often cumbersome due to lengthy matrix read and partitioning times along with increasing cluster allocation sizes. Full scale tests to confirm our findings in this study will be completed but are beyond the scope of this paper.

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

TABLE I: BENCHMARK MATRIX SUITE

matrix	rows	nnz	nnz %	nnz row	Symmetry
atmosmodd	1270432	8814880	5.46E-06	6.93	non-symmetric
parabolic_fem	525825	3674625	1.33E-05	6.98	symmetric
rajat30	643994	6174244	1.49E-05	9.58	non-symmetric
CurlCurl_3	1219574	13544618	9.11E-06	11.10	symmetric
offshore	259789	4242673	6.29E-05	16.33	symmetric
Fem_3D_thermal2	147900	3489300	1.60E-04	23.59	non-symmetric
nlpkkt80	1062400	28192672	2.50E-05	26.53	symmetric
CO	221119	7666057	1.57E-04	34.66	symmetric
gsm_106857	589446	21758924	6.26E-05	36.91	symmetric
msdoor	415863	19173163	1.11E-04	46.10	symmetric
bmw3_2	227632	11288630	2.18E-04	49.59	symmetric
BenElechi1	245874	13150496	2.10E-04	53.48	symmetric
t3dh	79171	4352105	6.94E-04	54.97	symmetric
F2	71505	4294285	8.40E-04	60.05	symmetric
consph	83334	6010480	8.65E-04	72.12	symmetric
SiO2	155331	11283503	4.68E-04	72.64	symmetric
torso1	116158	8516500	6.31E-04	73.31	symmetric
dielFilterV3real	1102824	89306020	7.34E-05	80.97	symmetric
RM07R	381689	37464962	2.57E-04	98.15	non-symmetric
m_t1	97578	9753570	1.02E-03	99.95	symmetric
crankseg_2	63838	14148858	3.47E-03	221.63	symmetric
nd24k	72000	28715634	5.54E-03	398.82	symmetric
TSOPF_RS_b2383	38120	16171169	1.11E-02	424.21	non-symmetric
mouse_gene	45101	28967291	1.42E-02	642.27	symmetric
human_gene1	22283	24669643	4.97E-02	1107.10	symmetric

TABLE II: SYSTEM CHARACTERISTICS

	Voltrino	Prior System
per Socket Characteristics		
Socket Type	Intel Phi 7250	Intel E5–2650v2
Cores per Socket	68	8
Core Clock Rate	1.4 GHz	2.6 GHz
Peak Flops per Core	44.6 GF/s	20.8
Total Main Memory	96 GB	16 GB
Main Memory Type	DDR4	DDR4
Main Memory Channels	6	4
Bandwidth per Channel	19.2 GB/s	14.9 GB/s
Peak Memory B/W	115.2 GB/s	59.7 GB/s
Total Intermediate Memory	16 GB	N/A
Intermediate Memory Type	MCDRAM	None
Peak Memory B/W	490 GB/s	N/A
Single Node Characteristics		
Node Type	Cray XC40 Phi	IBM nx360 M4
Sockets per Node	1	2
Peak Flops per Node	3.04 TF/s	332.8 GF/s
Network Interface	Cray Aries	Mellanox FDR IB
Peak Injection B/W	16 GB/s[2]	8 GB/s
System Characteristics		
MPI Processes per Node	1	2
Total Nodes	24	72
Total Cores	1632	1024
Interconnect Topology	DragonFly	Switched IB (2 lvl)
Peak Flops per Process	3046.4 GF/s	166.4
Peak Memory B/W per Process	115 or 490	59.7
Peak N/W Injection Rate per Process	16	8

Both systems communicate with NICs over PCIe 3.0 from socket.
Voltrino topology appears to have all-to-all among all 16 nodes.
The "Prior System" traverses one switch for up to 64 processes, and three switches for more.

V. Experimental Implementation

A. Hybrid CVR

In order to evaluate the performance impact of communication on distributed SpMV using KNL we developed a hybrid implementation of the CVR method. Our hybrid code disperses application functionality across multiple KNL sockets, effectively performing several sub problems simultaneously. First the sparse matrix is converted to CSR format while being read in from file. We then perform workload balancing, discussed in greater detail in section V-B, to insure uniform non-zero count for all processes. After distributing work allotments to all processes, the temporary CSR format is converted to the packed CVR format as discussed in section III and computation begins. Once SpMV computation has completed we collect localized results back at the global master process to form the single result vector.

Collecting each sub problem's result to the global master process occurs at the end of SpMV CVR computation, and is the inter-process communication we analyze. We used MPI to enable multiple KNL-equipped nodes, thereby greatly increasing total thread concurrency as well as total memory bandwidth at the expense of communication overhead.

We tested our hybrid code on Voltrino, a 24 node KNL cluster at Sandia National Laboratories. Voltrino is based on the Cray XC40 design as used in the Trinity system [10], [25]. For interconnect purposes, Voltrino is effectively an Aries [26], [27] single group in a Dragonfly topology, where a single Aries NIC is shared among 4 nodes.

For comparison to our previous work, only square numbers of MPI processes have been chosen, with only one process being assigned to a single KNL chip on a single KNL equipped node. This allows us to make direct comparisons to the notional process matrix used for workload distribution in [8], yet meant that only a maximum of 16 out of Voltrino's 24 nodes were utilized for the largest runs.

Table II compares the characteristics of the Voltrino cluster to the cluster used in our prior evaluation. On a "per process" basis, Voltrino has about 18X the peak floating point performance, up to 8X the memory bandwidth, and up to 2X the network injection bandwidth.

B. Work Balancing and Distribution

Workload balancing and data distribution for the hybrid CVR method is similar to that of the *balanced* distribution method used in our previous study [8]. Some alterations to that method have been performed to simplify its implementation and increase workload balance uniformity. In this work we elected to keep entire rows contiguous and assign them in their entirety to a single MPI process. Rows are sorted based on their nnz_{row} count. After sorting we perform a greedy bin packing by assigning the next row, and its non-zeros, to the process with the lowest number of currently assigned non-zeros. As such, we are able to achieve a near uniform nnz distribution across processes, despite processes having varying row count assignments. A mapping of these assignments is kept by the global master process to facilitate result collection.

Data is then distributed to the MPI processes in the same fashion as the *balanced* method from our previous study. Once each process has received its work allotment, it performs a local CVR prepossessing on its own data. The CVR preprocessing step reduces the impact of matrix structure by reordering data to improve locality. Additionally our work distribution method performs a 1D row-wise partitioning in

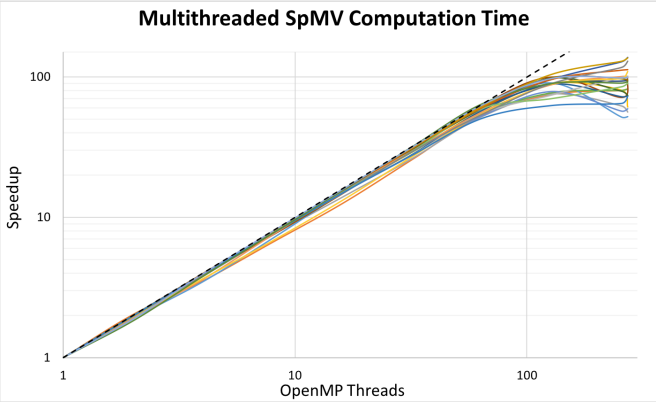


Fig. 4: Observed SpMV computation speedup for single MPI process with varying OpenMP thread counts. Each line represents a different benchmark matrix from the chosen suite discussed in Section IV. Single node multithreaded tests include no communication since OpenMP uses a shared memory model and only one node is in use.

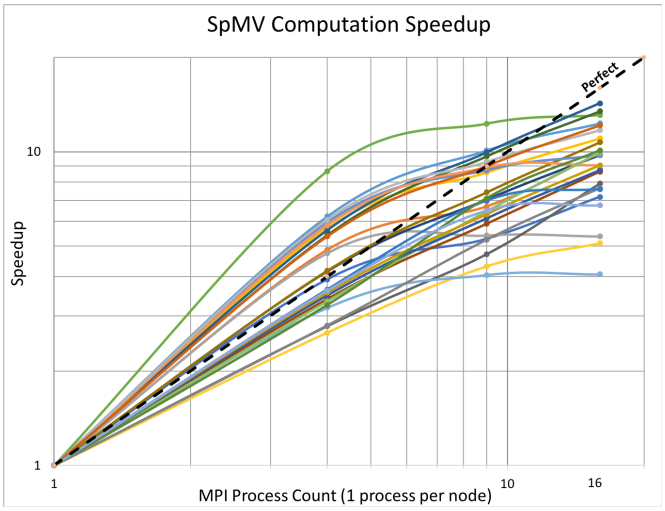


Fig. 5: Observed SpMV computation time for each benchmark matrix with varying MPI process count. We distribute a single MPI process per KNL node such that process count is equal to KNL node count. At 4 MPI processes a group of matrices achieve better than expected speedup. Continued super-linearity is observed for with 9 MPI processes but across fewer matrices. At 16 MPI processes we see that speedup for all matrices has fallen to sub-linear.[3]

which complete rows are assigned to processes based on their nnz in order to create a near uniform nnz per process. This mitigates the impact of matrix structure on computational performance.

All processes then perform localized multi-threaded CVR SpMV, saving results to a local vector. Once all processes are ready, they all participate in an MPI_Gatherv to collect localized row results back to the global master. Similar to our previous study, the variable gather collective was chosen in an effort to minimize total message volume.

VI. Evaluation

A. Multithreaded SpMV Performance

To show the impact that communication from a distributed memory model has on Hybrid SpMV, we first analyzed a single MPI process running on a single node, with increasing numbers of OpenMP threads. Fig. 4 shows that as thread count increases, SpMV computation speedup increases, up until around the maximum concurrent thread limit of 272. As thread counts increase core count, SIMD lane, and cache use increase correspondingly providing additional performance.

For most matrices evaluated, computation time reduction slows or stagnates completely after as we approach 4 threads per core. Therefore given speedup stagnation was seen at around 4 threads per core, subsequent experiments used 1 MPI process per node, and 272 OpenMP threads per process.

B. Hybrid SpMV Performance

We tested our hybrid CVR implementation while recording computation and communication times separately. Timing measurements for SpMV computation include only the time required to perform a hybrid SpMV using the CVR format and do not include data distribution or matrix preprocessing times. However the measured SpMV time does include the preparation time necessary for data gathering, alignment, and populating of SIMD lanes. As seen in Fig. 5, we observed computational speedup for all matrices, with the greatest speedup occurring for between 1 and 9 MPI processes. Additionally we saw that all matrices become sub-linear by P = 16.

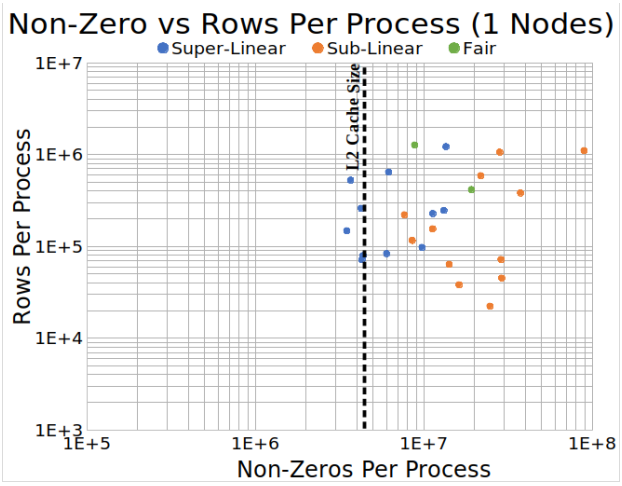
We note that the behavior observed for all matrices is similar due to the CVRs pre-processing and computation. CVR's pre-processing effectively nullifies the impact of non-zero structure during computation, leaving only non-zero and row counts as major contributors to performance behavior.

Approximately 10 out of 25 matrices in our benchmark suite achieved super-linear speedup for the SpMV computation phase. We believe this occurs for several reasons, most significantly of which is improved cache performance as core counts increase [28]. Fig 6 shows the nnz per MPI process as well as the number of rows per process. The vertical dashed line represents the maximum number of 8-byte doubles (approximately 4.4million) which can be stored in L2 cache on a KNL chip. This value was used as the threshold for determining if local problem size will fit entirely in L2 cache.

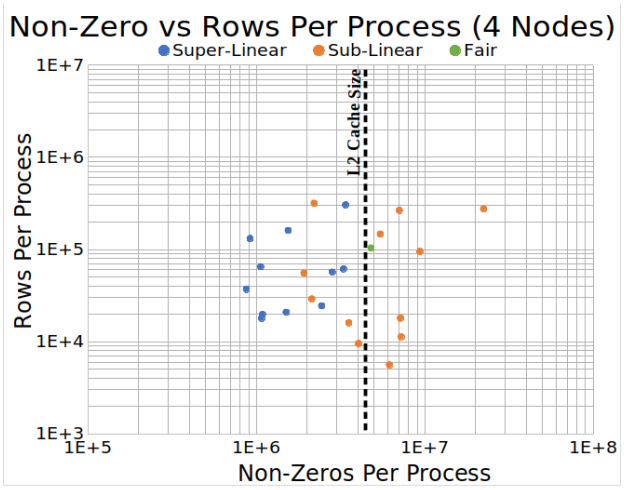
In Fig 6b we can see that all super-linear matrices were beneath the L2 threshold after $P = 4$. These matrices represent the smallest nnz for all matrices tested. Correspondingly their associated local problem size will shrink beneath the cache threshold the fastest. We then saw speedup slow for these matrices until all matrices exhibit sub-linear performance at $P = 16$.

As seen in Fig 7 increasing total nnz causes a decrease in compute time per non-zero. Previous research on SpMV has shown higher non-zero counts equate to greater performance, however our hybrid implementation observes super-linear performance among matrices with the lowest number of non-zeros. This indicates that boost from greater cache efficiency

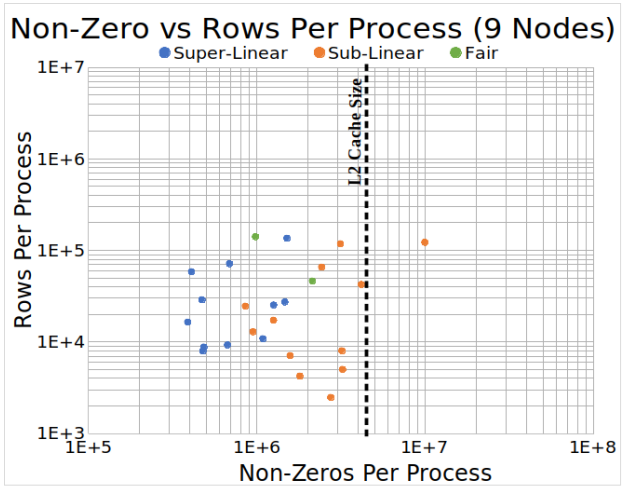
[3]For Fig. 4 and 5 the plotted values are the average fo 10 runs, with a maximum variance between runs of less than 5%. Therefore we opted to exclude error bars on these figures.



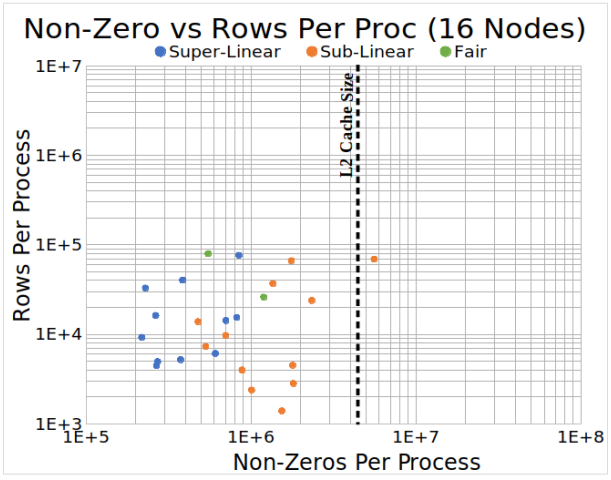
(a) 1 Node



(b) 4 Nodes



(c) 9 Nodes



(d) 16 Nodes

Fig. 6: Non-Zeros Per Process vs Rows Per Process: Each matrix evaluated is represented by a single point, and is colored based on the speedup observed by that matrix at the given process count. The vertical dash line represents the maximum number of 8-byte doubles that can fit into L2 cache across all tiles on a single KNL device. (a) For 1 process we see several matrices fit into L2 cache already, while the remaining matrices fall below this threshold as process count increases (b to d).

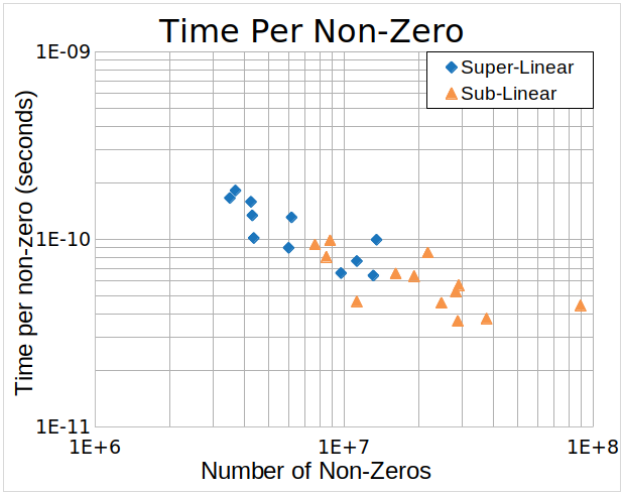


Fig. 7: Computational Time Per Non-Zero: for each matrix evaluated the SpMV computation time vs nnz in the matrix is shown for 1 MPI process (1 node). We can see that those matrices for which superlinear speedup was observed, have the fewest non-zeros as well as the highest time per-non zero.

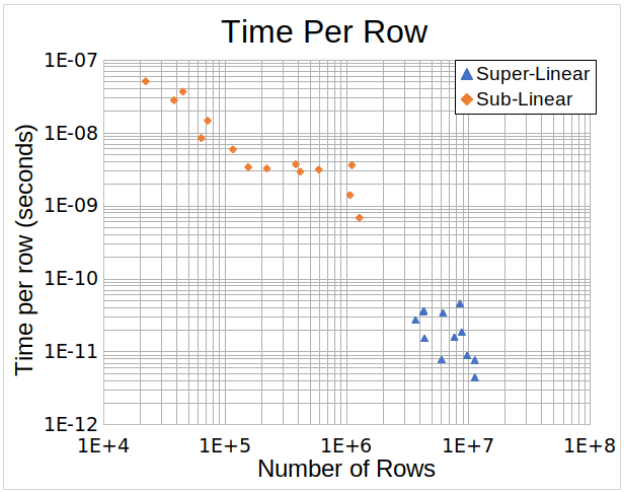


Fig. 8: Computational Time Per Row: The time it takes to compute a row vs total number of rows in the corresponding matrix. Those matrices that achieved super-linear speedup posses more rows, however time per row is between 1 and 2 orders of magnitude lower than matrices that did not achieve super-linearity.

exceeds the decreased time per non-zero of larger matrices.

Additionally, we believe performance of the hybrid code to also be influenced by the number of rows in a sparse matrix. Fig 8 shows the observed computation time per row for each matrix tested. We found that greater row counts achieved the lowest time per row, and that these matrices were also those which achieved super-linear computational speedup. The difference between the two groups approaches 5 orders of magnitude.

Matrices experiencing super-linear speedup possessed a mean nnz, row count, and average nnz_{row} of 3.27E5, 7.27E6, and approximately 36 respectively. It is worth noting however that for all matrices tested, speedup eventually dropped to sub-linear as P continued to increase. This is likely due to increased parallelism overhead, as well as the lack of additional benefit from increased cached efficiency since local problem sizes had already fallen below the L2 threshold.

C. MPI Communication Overhead

As can be seen in Fig. 5, hybrid strong scaling does result in computational speedup of SpMV, however this does not reflect true performance achieved as communication overhead is not accounted for. We assign a single MPI process to each node, thereby making all MPI communication inter-node. Fig. 9 shows communication and computation times, averaged over 10 runs, for all matrices and process counts examined. We saw SpMV computation times decrease as process count increases, however communication time did not follow suit. This is because as process count increases, the number of rows being assigned to an arbitrary process decreases in rough proportionality. Therefore at lower process counts each process must send greater amounts of data to the master process, increasing its transmission time. In contrast, as process counts increase, message size decreases, yet overhead is exacerbated by the larger number of participating processes. From Fig. 9 we clearly see that the cost of communication vastly outweighs the additional computation performance gained.

We observed communication to computation ratios between 200 for $P = 4$ and 2600 for $P = 16$. Reviewing results from the previous study we found that the ratios observed on Voltrino were between 100x and 650x greater than those of conventional Intel Xeon system for the same process counts [8].

Subsequently while the Cray Aries interconnect provided an optimized MPI_Gather, our tests using MPI_Gatherv were 2x-3x slower than on Infiniband used in [8]. Voltrino's interconnect configuration insured that all nodes existed in the same network group and therefore lacked influence from the additional network effects such as switching observed in our earlier study [8].

D. Communication Overhead Impact

It is clear that as soon as a second KNL node is utilized to perform SpMV on extremely sparse matrices, MPI communication becomes the dominant factor in overall runtime.

Interestingly, we found that all matrices exhibited similar scaling behavior regardless of matrix properties. As intended this indicates that our hybrid implementation normalizes distribution and computation to a function of only the total number of non-zeros and rows. Fig. 10 illustrates the immediate impact the addition of MPI communication has on overall performance. As soon as $P > 1$ the addition of communication overhead, which grows increasingly larger than computation time, outweighs performance gains from strong scaling. The degree of performance degradation was anticipated given the communication to computation ratios observed, and are likely to continue as $P \to \infty$.

Fig 11 compares our current findings with those of our previous study. As Fig 11 shows the KNL implementation received approximately a reduction in performance of nearly 2 orders of magnitude when compared to than the Intel Xeon cluster. These results do not illustrate the entire picture however since the Intel Xeon system's computational times are considerably higher than those of Voltrino thereby decreasing the overall impact of communication overhead on overall performance for the process counts shown.

Our hybrid implementation of the KNL CVR method achieves greater SpMV computational performance, however it also experiences greater communication overhead. In Fig. 12 we compare the message latency for *MPI_Gatherv* on both systems. The Voltrino system experienced an order of magnitude larger latency for all message sizes below 4096 bytes, compared to the conventional system. This means that as message sizes decrease due to strong scaling the larger latency experienced by Voltrino leads to overall performance degradation.

Additionally we evaluated the aggregated bandwidth of both systems for 2 through 16 MPI processes and display these results in Fig. 13. Both systems were able to obtain bandwidth saturation, for all process counts tested. We saw that the conventional Xeon system achieved saturation at a message size of approximately 2048 bytes. In comparison Voltrino required a much larger message size of nearly 65536 bytes in order to obtain bandwidth saturation. This indicates that Cray Aries has a higher peak bandwidth than that the Mellanox FDR Infiniband. However, this bandwidth is shared among 4 nodes.

The KNL based Voltrino system and its Cray Aries interconnect possess greater maximum bandwidth but at the cost of much higher per message latency. This means that the interconnect is not able to achieve sufficient link saturation necessary to hide the significantly higher message latency. Combined these two network characteristics account for the overall performance degradation of 2 orders of magnitude observed during our tests. It is important to note that our analysis does not include any additional network effects, therefore continued scaling would likely degrade performance further due to the addition of topological anomalies.

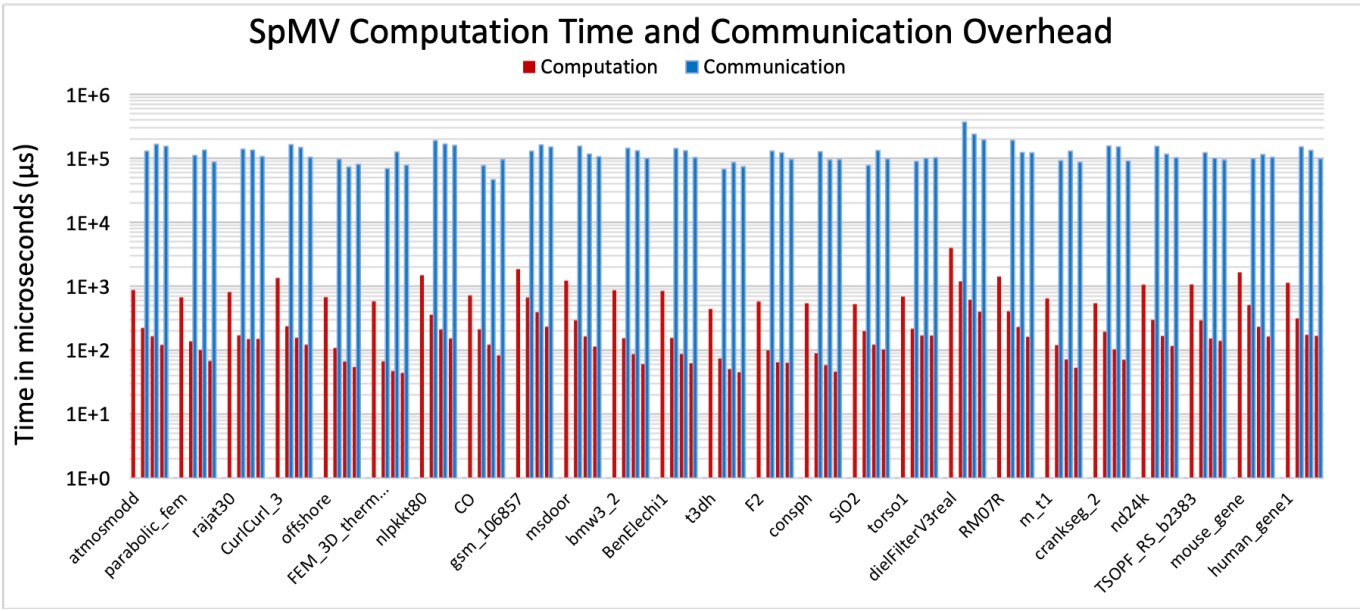


Fig. 9: Observed computation time and communication time from MPI_Gatherv is shown for all process counts (1,4,9, and 16) and benchmark matrix. Benchmark matrices are listed left-to-right in ascending order by avg nnz per row.

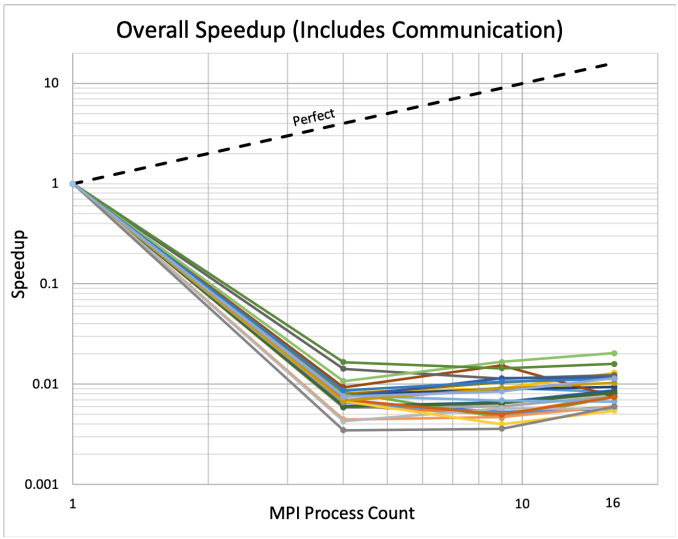


Fig. 10: Combined performance for SpMV computation and communication of result, in terms of speedup. Each line represents a different benchmark matrix for the given number of MPI Processes.

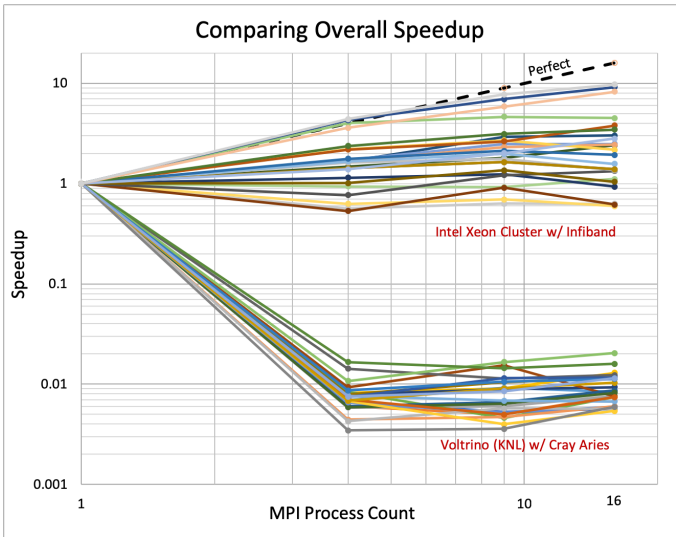


Fig. 11: Overall speedup, incorporating computation and communication of final result, are shown for both the CVR method on Voltrino, and the conventional Intel Xeon system tested in [8]. Overall performance of our hybrid code on Voltrino is on average 2 orders of magnitude lower than those observed on the conventional system.

E. Communication Overhead Amortization

Xie et al [14] analyzed the impact of pre-processing time on SpMV performance measurements. They assumed that for an arbitrary matrix, pre-processing would take time α, while SpMV computation required time β. They then calculated the number of SpMV iterations required to offset the pre-processing time.

Similar to [14] we will use the term *iteration* to refer to one complete SpMV operation on the entirety of a matrix and its dense vector component. In this study we evaluated computation time for only a single iteration of SpMV thereby making *iteration time = SpMV computation time*.

While much of our study is conducted under the assumption that this SpMV code is but a portion of a larger solver, if we visualize that solver to be iterative in nature we can see exactly how poignant communication overhead actually is. Reducing the impact of communication overhead can be accomplished by performing multiple SpMV iterations, such that the performance enhancement provided by our hybrid method outweighs the accompanying overhead. For our implementation the number of SpMV iterations necessary to offset MPI overhead is equivalent to the ratio of communication-to-computation. Therefore we can see that for relatively low

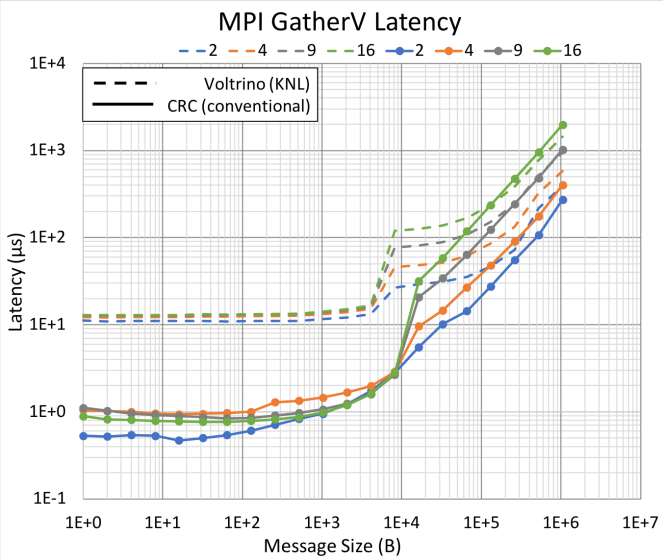


Fig. 12: Average message latency for the variable length gather (MPI_Gatherv) collective for varying MPI process counts are shown. Voltrino system has on average an order of magnitude greater latency for message sizes below 4096 bytes than the conventional system.

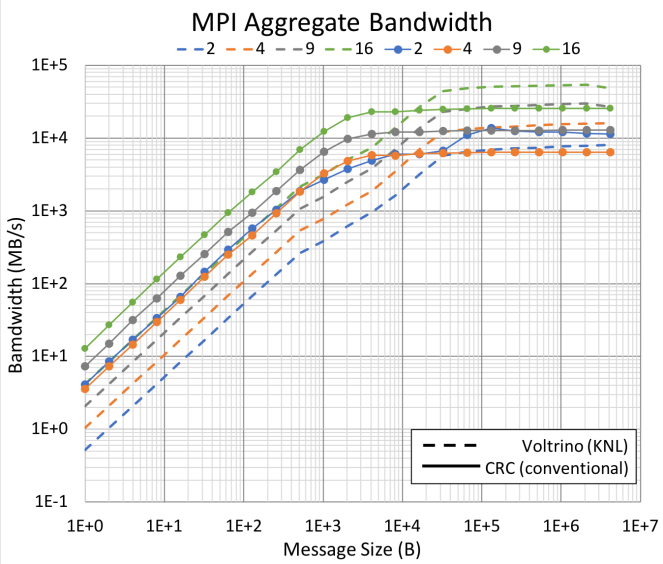


Fig. 13: Average aggregated bandwidth for the MPI process counts shown. The conventional system with its Inifiband FDR interconnect bandwidth saturation begins by a message size of 2048 bytes, compared to approximately 65536 bytes on the Cray Aries interconnect used by the Voltrino system.

values of P potentially thousands of communication-less locally updating iterations are required. Interestingly, the number of SpMV iterations needed for our hybrid implementation to overcome communication overhead is greater than those required by the conventional system due to increased preprocessing requirements.

VII. Future Work

While our work has shown that performance degradation created by communication overhead dominates on KNL clusters for SpMV on extremely sparse matrices, further exploration is needed. Sparse problems will be evaluated on additional architectures such as GPUs and emerging architectures, as will the communication impact these architectures experience in a multi-node environment.

Our tests thus far have required that at least at the beginning, the entirety of a sparse matrix and its accompanying dense vector fit on a single node. We are currently investigating the use of communication avoiding algorithms and the ability to reduce overhead at scale. The issues associated with larger data sets incapable of residing on individual nodes, as well as the ability to handle streaming updates to the sparse matrix and or the dense vector are also of interest.

Furthermore while this study and [8] helped to identify performance behavior of SpMV at the extreme end of strong scaling, we are interested in evaluating much larger matrices that do not initially fit within the memory space of a single node. Evaluating such matrices will allow us to work towards our goal of generating a model for forecasting the point at which hybrid codes stop scaling.

VIII. Conclusions

This study evaluated a hybrid implementation of the CVR SpMV method on KNL systems. We focused primarily on the strong scaling of SpMV computation over extremely sparse matrices with the intent of identifying extreme scaling behavior while utilizing more manageable data set and cluster sizes.

We demonstrated that single process performance does improve as thread counts increase, but flattens out as we approach the total possible concurrent threads for single KNL device. In addition, this paper demonstrates that overall scalability in a message passing environment is overwhelmingly driven by its communication pattern. Our findings suggest SpMV possesses an inability to strong scale beyond some point, even with cutting edge network interconnects or architectures, and that this scaling limit is tightly tied to the performance impact of communication overhead.

Furthermore as we have shown, optimizing SpMV on existing many-core architectures using hybrid design does achieve a computational performance boost, yet does not address the dominant communication overhead factor at scale. Our studies indicates that as $P \rightarrow \infty$ communication overhead is by far the dominant factor and cannot be discarded. For extremely sparse matrices once strong scaling insures that problem size fits within a single node, as was the case with using smaller matrices from the start, we saw a *huge degradation* due to inter-node communication regardless of matrix properties.

CVR and KNL were chosen in an effort to provide an optimal SpMV computation method on an architecture capable of increased parallelism. We expected to see much greater strong scaling using the KNL system than was actually achieved. Comparing this study to our previous hybrid SpMV implementation on a conventional system both implementations share similar scaling behavior.

As the level of on device parallelism is driven higher, so to is message size per device and with it communication overhead.This would suggest a need to shift efforts away from reducing SpMV computation time in favor of eliminating the bulk of communication within sparse applications. Alternative schemes for remote aggregation, or the avoidance of their requirement, will be essential for obtaining superior performance.

IX. 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 acknowledge our appreciation for the use of the Advanced Architecture Test Beds at Sandia National Laboratories, 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," in *Computational Science — ICCS 2002*, P. M. A. Sloot, A. G. Hoekstra, C. J. K. Tan, and J. J. Dongarra, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 632–641.

[3] M. Mosby and K. Matou, "Hierarchically parallel coupled finite strain multiscale solver for modeling heterogeneous layers," *International 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] M. Mosby and K. Matous, "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] P. M. Kogge, N. A. Butcher, and B. A. Page, "Introducing streaming into linear algebra-based sparse graph algorithms," in *accepted for Int. Conf. on High Performance Computing Simulation*, ser. HPCS'19, Jul. 2019.

[7] "Graph blas forum." [Online]. Available: http://graphblas.org/index.php?title=Graph_BLAS_Forum

[8] B. A. Page and P. M. Kogge, "Scalability of hybrid sparse matrix dense vector (spmv) multiplication," in *2018 International Conference on High Performance Computing Simulation (HPCS)*, July 2018, pp. 406–414.

[9] J. Jeffers, J. Reinders, and A. Sodani, *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition 2Nd Edition*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016.

[10] A. Agelastos, M. Rajan, N. Wichmann, P. Li, R. B. S. Domino, E. Draeger, S. Anderson, J. Balma, S. Behling, M. Berry, P. Carrier, M. Davis, K. McMahon, D. Sandness, K. Thomas, S. Warren, and T. Zhu, "Performance on trinity phase 2 (a cray xc40 utilizing intel xeon phi processors) with acceptance-applications and benchmarks," https://www.osti.gov/servlets/purl/1456853, June 2017.

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

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

[13] W. Yang, K. Li, and K. Li, "A hybrid computing method of SpMV on CPUGPU heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 104, pp. 49–60, 2017. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2016.12.023

[14] B. Xie, J. Zhan, X. Liu, W. Gao, Z. Jia, X. He, and L. Zhang, "Cvr: Efficient vectorization of spmv on x86 processors," in *Proc. of the 2018 Int. Symposium on Code Generation and Optimization*, ser. CGO 2018. New York, NY, USA: ACM, 2018, pp. 149–162. [Online]. Available: http://doi.acm.org/10.1145/3168818

[15] V. Marjanović, J. Gracia, and C. W. Glass, "Performance modeling of the hpcg benchmark," in *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*, S. A. Jarvis, S. A. Wright, and S. D. Hammond, Eds. Cham: Springer Int. Publishing, 2015, pp. 172–192.

[16] F. Vzquez, E. M. Garzon, J. Martinez, and J. J Fernndez, "The sparse matrix vector product on gpus," vol. 2, 01 2009.

[17] A. Ashari, N. Sedaghati, J. Eisenlohr, and P. Sadayappan, "An efficient two-dimensional blocking strategy for sparse matrix-vector multiplication on gpus," in *Proc. of the 28th ACM Int. Conf. on Supercomputing*, ser. ICS '14. New York, NY, USA: ACM, 2014, pp. 273–282. [Online]. Available: http://doi.acm.org/10.1145/2597652.2597678

[18] E. Saule, K. Kaya, and Ü. V. Çatalyürek, "Performance evaluation of sparse matrix multiplication kernels on intel xeon phi," *CoRR*, vol. abs/1302.1078, 2013. [Online]. Available: http://arxiv.org/abs/1302.1078

[19] X. Liu, M. Smelyanskiy, E. Chow, and P. Dubey, "Efficient sparse matrix-vector multiplication on x86-based many-core processors," in *Proc. of the 27th Int. ACM Conf. on Int. Conf. on Supercomputing*, ser. ICS '13. New York, NY, USA: ACM, 2013, pp. 273–282. [Online]. Available: http://doi.acm.org/10.1145/2464996.2465013

[20] 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.

[21] M. Rashti, *Improving Message-Passing Performance and Scalability in High-Performance Clusters*, 2011. [Online]. Available: http://qspace.library.queensu.ca/handle/1974/6284

[22] A. Sodani, R. Gramunt, J. Corbal, H. S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y. C. Liu, "Knights landing: Second-generation intel xeon phi product," *IEEE Micro*, vol. 36, no. 2, pp. 34–46, Mar 2016.

[23] N. Butcher, S. L. Olivier, J. Berry, S. D. Hammond, and P. M. Kogge, "Optimizing for knl usage modes when data doesn't fit in mcdram," in *Proceedings of the 47th International Conference on Parallel Processing*, ser. ICPP 2018. New York, NY, USA: ACM, 2018, pp. 37:1–37:10. [Online]. Available: http://doi.acm.org/10.1145/3225058.3225116

[24] 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

[25] A. Agelastos, M. Rajan, N. Wichmann, P. Li, R. B. S. Domino, E. Draeger, S. Anderson, J. Balma, S. Behling, M. Berry, P. Carrier, M. Davis, K. McMahon, D. Sandness, K. Thomas, S. Warren, and T. Zhu, "Performance on trinity phase 2 (a cray xc40 utilizing intel xeon phi processors) with acceptance-applications and benchmarks," https://cug.org/Proc./cug2017_Proc./includes/files/pap138s2-file1.pdf, June 2017.

[26] R. Alverson, E. Froese, L. Kaplan, and D. Roweth, "Cray XC Series Network." [Online]. Available: https://www.cray.com/sites/default/files/resources/CrayXCNetwork.pdf

[27] D. Doerfler, B. Austin, B. Cook, J. Deslippe, K. Kandalla, and P. Mendygral, "Evaluating the networking characteristics of the cray xc40 intel knights landingbased cori supercomputer at nersc," *Concurrency and Computation: Practice and Experience*, vol. 30, Sept. 2017. [Online]. Available: https://doi.org/10.1002/cpe.4297

[28] S. Ristov, R. Prodan, M. Gusev, and K. Skala, "Superlinear speedup in hpc systems: Why and when?" in *2016 Federated Conf. on CS and Information Systems (FedCSIS)*, Sept 2016, pp. 889–898.