Performance Model of Iterated SpMV for Distributed System.

Md Maruf Hossain
Dept. Computer Science
University of North Carolina at Charlotte
Charlotte, USA
mhossa10@uncc.edu

Erik Saule
Dept. Computer Science
University of North Carolina at Charlotte
Charlotte, USA
esaule@uncc.edu

***Abstract*—Many applications rely on basic sparse linear algebra operations from numerical solvers to graph analysis algorithms. Yet, the performance of these operations is still reasonably unknown. Users and practitioners rely on the rule of thumb understanding of what typically works best for some application domain.**

This paper aims at providing an overall framework for the distributed system to think about the performance of sparse applications. We use the sparse matrix-vector(SpMV) multiplication as the representative of the experiments. We model the performance of multiple SpMV implementations on distributed systems. We model the performance of different modes of execution of SpMV using linear and polynomial regression models for a distributed system. The models enable us to predict how to partition and represent the sparse matrix to optimize the performance of iterated SpMV on a cluster with 225 cores.

***Index Terms*—SpMV, MPI, Graph Partitioning**

I. Introduction

Sparse matrix-vector multiplication(*SpMV*) is one of the fundamental operations in sparse linear algebra. It is critical to solving linear systems and is widely used in engineering and scientific applications [1]–[3]. Distributed memory systems have entered a new age with the popularization of departmental clusters to support these scientific and engineering applications. Many different approaches have been proposed to improve *SpMV* performance on clusters. But choosing the right approach still mostly relies on the rule of thumbs. We posit that building models to predict the performance of different configurations is the only way to make better-informed decisions.

In this paper, we propose a linear and polynomial *Support Vector Regression*(*SVR*) [4] model to predict and analyze the run time of *SpMV* on the distributed system for different ways to execute the operation. Our system will analyze the predicted run time and return the best possible algorithm for the system. The performance of the *SpMV* mainly depends on the size and structure of the matrices and the architecture of the system.

To perform sparse matrix-vector multiplication(*SpMV*) on distributed systems, the most common mechanism is to partition the matrix into multiple parts and perform *SpMV* on each part individually in the different processors. Good partitioning can ensure better load balance and limits the volume of communication between the underlying *MPI process*. Many partitioning algorithms have been proposed to ensure good load balance and to minimize *MPI communications* [5]–[7].

In this paper, we explore two partitioning modes (Uniform 2D-Partitioning and 1D Row Partitioning) and the performance of the different *SpMV* representation based on these partitioning mechanisms on distributed systems. We develop a linear and a polynomial support vector regression (*SVR*) performance models for SpMV operations on the distributed system for these different techniques. The models are accurate enough to predict the best configuration to execute SpMV given a matrix.

II. Related Work

There have been lots of previous work have done on the *SpMV* performance model for the *CPU* and *GPU* architecture. In particular, Guo and Wang [8] *et.al.* have proposed a linear model for the general-purpose *GPU* that can predict the runtime of the *SpMV* based on the strides size and nonzero per row. There are also some other *SpMV* models [9], [10] that exist for the *GPU* architectures. A performance model for *SpMV* on the *GPU* architectures is more common than a single or distributed system of *CPU* architectures. Consistent and parallelisms of the *GPU* performance is the main reason behind this. model for the GPU.

III. SpMV on Distributed Systems

In *2D-Uniform partitioning*, matrices are partitioned into both row-wise and column-wise. There is a choice in the number of column partitions and row partitions. However, for iterated SpMV applications, it is common practice to pick a symmetric partitioning scheme and a number of processors that is a perfect square. Each MPI process handles one of the portions of the matrix. Load imbalance is a common issue in 2D partitioning. Many matrices tend to have a band structure that tends to build very imbalanced partitioning. 2D-Uniform partitioning can balance the number of rows and columns but can have a significant load imbalance in the number of non-zeros in the blocks.

Boman and Devine et al. [11] noted that randomizing the order of the matrix can provide good load balance for SpMV in 2D-Uniform partitioning and is used in scientific applications [3]. We adopt this strategy for our 2D uniform partitioning technique. Each row (and corresponding vector

entry) is assigned to a random process. Since the expected number of rows and non-zeros is uniform for all processes, this method generally achieves a good load balance.

If $P = p^2$ is the number of processes and $matrix\_size$ is the size of the matrix then each processor should contain the same number $\left\lceil \frac{matrix\_size}{p} \right\rceil$ of rows and columns. The last block can be padded with zeros to make blocks of the name size.

With 2D-Uniform partitioning, the execution of the iterated SpMV is done using the classic methods. Each process performs its local multiplication. The values are then reduced within each row onto the diagonal process. These reduced values are then broadcast along with the column processes for the next iteration of the calculation.

In *1D-Row Partitioning*, matrices are split into row-wise only. Balancing the load between the processors and minimizing the communication boils down to solving a K-way graph partitioning problem [7]. We can define the ID-Row(K-way) partitioning for a graph $G = (V, E)$ with $|V| = n$, partition V into k subsets, $V_1, V_2, \ldots, V_k$ such that $V_i \cap V_j = \phi$ for $i \neq j$, $|V_i| = n/k$, and $\cup_i V_i = V$, and the number of edges of E whose incident vertices belong to different subsets is minimized. Each partition will allocate to a different MPI process. We base the 1D row partitioning technique that we will model on this strategy and we choose the METIS graph partitioning tool [6] to provide the precise row partitioning.

In 1D-Row partitioning, one process contains small portions of rows and their corresponding columns. To perform SpMV on the 1D-Row partitioning, a processor can hold either the portion of the **vector elements** corresponding to their row elements or the full vector. We call the strategy where the process holds the entire vector *Global 1D-Row SpMV*(G1DR-SpMV). In this strategy, all MPI processes perform matrix multiplication locally on the part of the matrix that belongs to them. After matrix multiplication, an ALL_Gatherv takes place to share the updated value of the vector.

We call the strategy where a process only holds the portion of the vector that it needs *Local 1D-Row SpMV*(L1DR-SpMV). In this algorithm, initially, a process performs local matrix multiplication on the non zero elements whose column belong to the local vector. For the rest of the non zero elements whose vector elements belong to the other processes, it retrieves the relevant part of the vector using communication tailored to each process using standard MPI point to point communication primitives.

The METIS K-way partitioning technique minimizes the communications performed by the Local 1D-Row SpMV. Therefore if METIS can partition the graph well, one would expect the Local 1D-Row to perform very few communications. However, if the partitioning is not good, the custom message strategy only saves few communications and the Global 1D-Row SpMV would benefit from the effectiveness of MPI Collectives. There is no point in designing a local 2D scheme since the randomized partitioning ensures there is little saving to be expected from customized communications.

The local matrix stored by the nodes can be represented in a number of formats. We picked the two most popular formats: CSR and COO.

IV. Performance Model

We build up three different SpMV models and investigate their performance on different algorithms and storage formats.

1) SpMV Model from Micro-Benchmark.
2) Support Regression Model.
3) Linear Model.

A. SpMV Model from Micro-Benchmark

In this model, we predict the runtime of the SPMV based the performance of couple of micro-benchmark of a selected computer architecture. Initially, SpMV on the distributed system can be divided into two main parts,

- Core Calculation: performance of the matrix-vector calculation in a MPI node.
- MPI Communication: communication runtime among MPI ranks(processes).

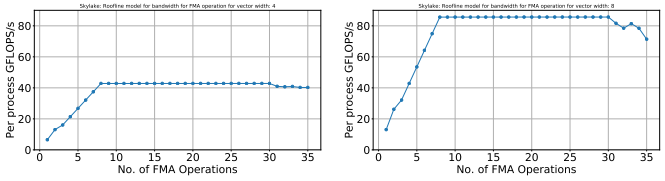
1) Core Calculation: We can represent the basic SpMV by $y = y + Val * x$, which requires two floating point operation(multiply and addition). So, we can say we need to calculate NNZ(number of non-zeros) times FMA(fused multiply addition) to perform SpMV. But, first it needs to load the data and SpMV is bound by the memory bandwidth rather than instruction. So, we separated the *core calculation* into,

- Run time for FMA: $L_{FMA} \times NNZ$, where L_{FMA} is the latency of a single FMA and NNZ is number of non-zeros.
- Memory access latency: L_{RW}

a) Micro Benchmark for FMA: The throughput of SkylakeX is 2 instructions per cycle and the latency of FMA is 4 cycles. So there is a potential of pipelinling($2 \times 4 = 8$) to get the optimal results. Now, SkylakeX has 512-bit register that can give the ability of the vectorization. For 64-bits floating point operation it can give vector width 8 and for 32-bits it can give at max vector width 16. To find the peak performance of the FMA and to avoid the read-write latency we need to setup the benchmark that datasets can be contained in the register. Now, SkylakeX has 32 registers. We can populate the pipeline by using sufficient ampunt of work. By varying the number of fused-multiply-addition calculation, we can find out the performance limitation. From the information of the processors, we can say that 4 cycles required for *FMA* and the throughput of the FMA is 2 instruction per cycle, that means we should at least use $4 \times 2 = 8$ instruction at a time to populate the pipeline. Figure 1 shows the roofline model of the *FMA* on SkylakeX processor. We can estimate the theoritical peak performance a single FMA by the following equation,

$$P = Base\_Clock\_Frequency \times Vector\_Width \times \frac{FLOPs}{Instruction}$$

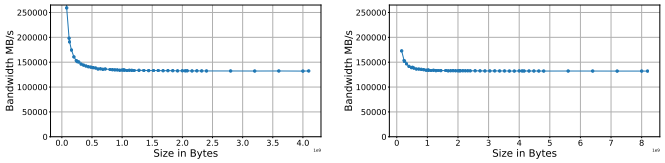
The significance of the *FMA* latency is very few compare to the memory access latency.



(a) Vector Width: 4 (b) Vector Width: 8

Fig. 1. Skylake: (MPI)Roofline model for bandwidth for FMA operation

b) Micro-Benchmark for Memory Access Latency: We explore the *STREAM* benchmark to find out the bandwidth of the memory access for a selected architecture. Figure 2 shows



(a) Single Precision. (b) Double Precision.

Fig. 2. Single and double precision memory access bandwidth on Skylake processor.

the relation between data size and bandwidth of the SkylakeX processor. To pick the right bandwidth for a matrix-vector multiplication, we first calculate the size of the data. Based on the size of the data, we pick the average of the available immediate lower and higher point of the benchmark. Now, memory can be accessed sequentially or random. To calculate the sequential and random access data, we build another micro-benchmark(Cache-Pattern). Next, we will discuss about Cache-Pattern benchmark.

c) Cache-Pattern Micro-Benchmark: In SpMV some array or vector traverse incrementally and some irregular pattern. All the incremental or regular data accesses are treated as *Sequential* data. So, we do not need any extra work to handle them. Now, it is not guaranty that all the irregular access pattern data are fully random access. There is possibility chunks of data in the middle of the array are sequential. The main aim of this benchmark is to separate a irregular access array into sequential and random access data. This benchmark walk through a array and calculate the cache miss and hit based on the *least recently used*(LRU) memory. all the *hits* added to the sequential data and *miss* points are added to the random access data. Table I shows the memory access property for

TABLE I
MEMORY ACCESS PROPERTY FOR 2D-PARTITIONING SPMV MODEL(RPP=ROWS PER PROCESS, NNZ=NON-ZERO ELEMENTS).

Array	Elements Size		Data Type	Access Type	
	CSR	COO		CSR	COO
rowA	2×RPP	NNZ	Integer	Sequential	Sequential
colA	NNZ	NNZ	Integer	Sequential	Sequential
valA	NNZ	NNZ	Floating	Sequential	Sequential
x	NNZ	NNZ	Floating	Irregular	Irregular
y	RPP	NNZ	Floating	Sequential	Irregular

the 2D-Partitioning SpMV model. We can see the access to the vector `x` is irregular for both CSR and COO format. But access to the vector `y` is only irregular for the COO format. But, irregular does not guaranty the fully random access to the memory, that is why we need to separate the `x` for both and `y` for COO format into *hit* and *miss* portions. All the data size is determined for all the MPI ranks of a whole node.

2) Micro-Benchmark for MPI communication: MPI communications mostly depend on the size of the message and network topology. We build a benchmark based on the OSU-MPI-Benchmark. Because of the dynamics of the network topology we build a polynomial model based on the number of nodes, number of MPI ranks, message size, and MPI communication type.

Figure 3 shows the overall structure of the SPMV model from micro-benchmark.

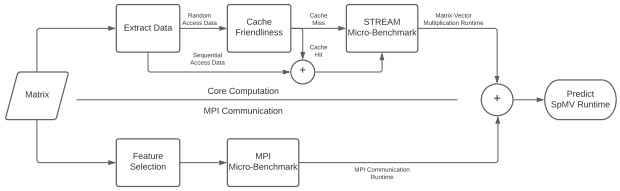


Fig. 3. Structure of the SpMV model from micro-benchmark.

B. Linear Model

We build up a linear model from generated random matrices of the different number of rows. For each matrix with specific rows, we generated matrices with the different number of nonzeros per row $1, 2, 4, 8, 16, \ldots, 128$. Then build a linear regression model for a particular row size matrix against the nonzeros per row. Our aim to find out two near similar generated matrix A and B for given test matrix(M), that the number rows in A and B immediate lower and higher than M respectively form the available matrices of the model. Then we predict the performance of the test matrix M based on these two model matrices.

We realized in early investigations that the linear models would not perform well on 1D partitioning models because the local number of non zero varies significantly. However, in the uniform 2D-Partitioning *SpMV*, every process receives a matrix where non-zero are distributed randomly.

Figure 4(a) shows the linearity of the run time over the non-zero per row for a particular matrix. It confirmed that we should build a regression model that can predict the run time of *SpMV* based on the non zero per row. We build multiple linear models using the different sizes of the random matrix. Each model trains by the different number of non-zeros per row but the same row size.

If a test matrix has r row and npr non-zero per row, then our system will find two linear models(L_1, L_2) for row r_1 and r_2 like $r_1 \leq r \leq r_2$. It is important to note that, $L_1(r_1)$ and $L_2(r_2)$ are the two models with immediate smaller and

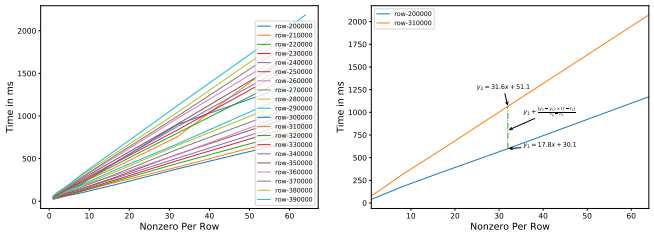
larger rows than r respectively. The following two equations represent the linear model $L_1(r_1)$ and $L2(r_2)$,

$$y_1 = m_1 \times x + c_1 \quad \text{for } r_1 \text{ and } x = npr$$
$$y_2 = m_2 \times x + c_2 \quad \text{for } r_2 \text{ and } x = npr$$

According to our model, we expect the performance(y) of the subject matrix with row r is between y_1 and y_2 ($y_1 \leq y \leq y_2$). Our system finally predict the execution run time of the subject matrix using the following equation,

$$y = y_1 + \frac{(y_2 - y_1)(r - r_1)}{r_2 - r_1}$$

Figure 4(b) shows the example for the matrix with 250000



(a) Nonzer per row vs run time for different matrices.

(b) Sample equation for the linear model.

Fig. 4. Linear relationship between nonzero per row and run time for the SpMV on the uniform 2d-Partitioning.

average rows per process and 32 nonzero per row. Here, possible r_1 and r_2 equations are available for rows 200000 and 310000. The figure reflects the prediction mechanism of our system.

C. Polynomial Support Vector Regression(SVR) Model

We build the polynomial SVR model for all three algorithms (*2D-Uniform partitioning SpMV, G1DR-SpMV, and L1DR-SpMV*). The performance of the SVR models largely depends on the appropriate attributes for the model. Then choose the proper values for the *free parameters* of the SVR model. *Cross-Validation* and *Grid-Search* widely use to do that for the SVR model. The models are built for a particular system, therefore they assume a fixed number of processors and interconnect.

1) Cross-Validation and Grid-Search: Initially, we split the matrices into train and test data set, no test data participate in the training mechanism. Here, we choose the polynomial ("poly") kernel to perform all the machine learning regression. In this kernel, some free variables need to choose. There are no fixed values for these variables, based on the application criteria it can vary. We select $5 - fold$ cross-validation that means we split the training set into 5 different parts, and among 5 parts we choose 4 as a training set and the remaining one as a test set. In the grid search, one need to select a set of variables for the free variable(C, γ), like $C = \{2^{-2}, 2^{-1}, \ldots, 2^4, 2^5\}$ and $\gamma = \{0.01, 0.02, \ldots, 0.2, 0.3\}$ and for each pair of variable need to perform cross-validation and record the score. Then select a pair that gives the best performance in cross-validation. The next step is to train the model using the best variable on the whole train set.

2) Feature selection: We build up polynomial support vector regression(SVR) models for three different SpMV algorithms on the two different graph formats(CSR, COO). The SVR model follows the same mechanism for all multiplication algorithm. The model predicts the average run time for a particular test matrix based on its attributes. So, attributes are the key to a good performance model. Although the local multiplication is the same for every algorithm, the communications are different for the different partitioning modes. The common attributes for all three algorithms are:

1) Average rows per process.
2) Average non zero per process.
3) Average non-zero per row.
4) Density of the matrix.
5) Standard deviation of the non zero per row.

But the communication among the MPI process in the Local L1DR-SpMV (one-to-one) is different than the other two(one-to-all or all-to-all). That is why based on the sparsity of the matrices the communication can vary a lot. For that, we need the following extra attributes for the L1DR-SpMV which can be extracted after partitioning.

1) Average local non-zero elements.
2) Average global non-zero elements.
3) Average inter-process call.
4) Average data transfer.

V. Experimental Settings

A. Hardware Platform and Operating System

All the nodes of the computing cluster come with Intel Xeon Gold 6154 processors (SkylakeX architecture) and 388GB of DDR4 memory. Each node has 36 cores in 2 sockets. Hyper-threading is disabled. The base frequency of each processor is 3.00 GHz. Each processor has 25 MB of L3 Cache. The nodes are connected by EDR Infiniband. The machine uses the `Linux`. To present concise results all experiments are performed on 144, 169, 225 and 256 MPI processes allocated on 4, 5, 7 and 8 nodes respectively. The system support maximum 256 MPI processes.

B. Metrices

We collect over 90 matrices from the *Florida Sparse Matrix* suit and perform train and test on these matrices. We separate test matrices to show the test result and the remaining matrices used for the train purpose. No test matrices participate in the train process, that is how we make the fair test performance. The performance of the machine learning model depend on the range of the training sets. That is why we select the test matrices that way that they are not out of the range of the train data sets.

VI. Experimental Results

A. SpMV Model from Micro-Benchmark

1) Random 2D-SpMV: Table II and III shows the overall performance of the SpMV on the 2D-Partitioning model on CSR and COO storage formats respectively. Average error for

the CSR and COO formats are 7.62 and 9.16 respectively. The error is calculate by,

$$error = \frac{actual\ time - predicted\ time}{actual\ time} \times 100$$

TABLE II
OVERALL SPMV ON RANDOM CSR 2D PARTITIONING(ON SKYLAKE).

Matrices	Nodes	Processes	Time(s)		Error%
			Actual	Predicted	
333SP	4	144	7.2E-03	7.5E-03	4.1%
	5	169	5.7E-03	6.6E-03	15.1%
	7	225	4.7E-03	5.2E-03	8.8%
	8	256	4.2E-03	4.5E-03	6.1%
AS365	4	144	7.4E-03	7.8E-03	4.7%
	5	169	5.8E-03	6.8E-03	15.8%
	7	225	4.7E-03	5.3E-03	13.5%
	8	256	4.4E-03	4.6E-03	6.1%
M6	4	144	6.8E-03	7.0E-03	2.7%
	5	169	5.3E-03	6.1E-03	15.0%
	7	225	4.4E-03	4.8E-03	9.4%
	8	256	4.0E-03	4.2E-03	4.7%
NLR	4	144	8.2E-03	8.7E-03	6.3%
	5	169	6.5E-03	7.6E-03	17.8%
	7	225	5.4E-03	6.0E-03	10.9%
	8	256	4.8E-03	5.2E-03	9.0%
hugetrace-00010	4	144	2.3E-02	2.2E-02	2.9%
	5	169	2.0E-02	2.0E-02	1.6%
	7	225	1.7E-02	1.6E-02	5.6%
	8	256	1.4E-02	1.5E-02	8.6%
road_central	4	144	2.5E-02	2.4E-02	2.5%
	5	169	2.2E-02	2.2E-02	1.4%
	7	225	1.9E-02	1.8E-02	5.2%
	8	256	1.5E-02	1.6E-02	7.2%
road_usa	4	144	3.6E-02	4.1E-02	15.0%
	5	169	3.6E-02	3.8E-02	5.2%
	7	225	3.1E-02	3.1E-02	1.6%
	8	256	2.7E-02	2.8E-02	6.6%

2) Local 1D-SpMV: Table IV and V shows the overall performance of the Local 1D-Partitioning SpMV model on CSR and COO storage formats respectively. Average error for the CSR and COO formats are 7.31 and 8.81 respectively.

3) Local 1D-SpMV: Table VI and VII shows the overall performance of the Global 1D-Partitioning SpMV model on CSR and COO storage formats respectively. Average error for the CSR and COO formats are 13.63 and 25.3 respectively.

B. Polynomial SVR Model

Table VIII shows the performance of the dynamic SVR model for the SpMV and the prediction from the polynomial SVR model for two different graph storage formats on each algorithm. The column *Pred* represents the predicted run time from the SVR model and the *Actual* represents the real run time of the SpMV. A green cell of Table VIII, represents a correct prediction of our models; while a red cell represents that our models predicted the best strategy incorrectly and storage format incorrectly.

The average error for uniform 2D-Partitioning are 9.19 and 5.27 for CSR and COO representation respectively. The model gives 9.86 and 6.55 average error for the G1DR on CSR and COO format respectively. The average error for L1DR is 14.15 for CSR and 13.22 for the COO storage format.

TABLE III
OVERALL SPMV ON RANDOM COO 2D PARTITIONING(ON SKYLAKE).

Matrices	Nodes	Processes	Time(s)		Error%
			Actual	Predicted	
333SP	4	144	5.4E-03	5.6E-03	3.7%
	5	169	4.2E-03	5.0E-03	19.6%
	7	225	3.5E-03	4.0E-03	15.5%
	8	256	3.4E-03	3.5E-03	4.3%
AS365	4	144	5.5E-03	5.8E-03	4.0%
	5	169	4.3E-03	5.1E-03	19.7%
	7	225	3.6E-03	4.1E-03	13.6%
	8	256	3.6E-03	3.6E-03	0.1%
M6	4	144	5.0E-03	5.2E-03	3.5%
	5	169	3.9E-03	4.6E-03	18.7%
	7	225	3.4E-03	3.8E-03	9.7%
	8	256	3.3E-03	3.3E-03	0.8%
NLR	4	144	6.1E-03	6.4E-03	4.9%
	5	169	4.8E-03	5.7E-03	19.6%
	7	225	4.0E-03	4.6E-03	15.2%
	8	256	3.9E-03	4.0E-03	2.8%
hugetrace-00010	4	144	1.7E-02	1.6E-02	4.2%
	5	169	1.3E-02	1.5E-02	13.7%
	7	225	1.1E-02	1.2E-02	9.5%
	8	256	1.0E-02	1.1E-02	9.2%
road_central	4	144	1.9E-02	1.8E-02	5.6%
	5	169	1.5E-02	1.7E-02	12.4%
	7	225	1.3E-02	1.4E-02	8.7%
	8	256	1.2E-02	1.3E-02	8.1%
road_usa	4	144	2.9E-02	3.0E-02	3.7%
	5	169	3.2E-02	2.8E-02	12.6%
	7	225	2.3E-02	2.4E-02	5.4%
	8	256	2.0E-02	2.2E-02	7.6%

VII. CONCLUSION

In this paper, we provide three SpMV models to predict the run time of the SpMV on the *Uniform 2D-Partitioning*, *GK-SpMV* and *LK-SpMV*). The performance models can predict the run time accurately enough to identify the optimal strategy to compute SpMV. Our models consider the matrix structure and partitioning information to predict the best possible communication strategy to perform *SpMV*. These models can be useful for the other graph algorithms that heavily relies on the performance of the SpMV, such as *page rank*. The model may need to be adjusted to predict the correct graph representation format.

REFERENCES

[1] David F Gleich. Pagerank beyond the web. *siam REVIEW*, 57(3):321–363, 2015.

[2] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[3] Tomas Dytrych, Pieter Maris, Kristina D Launey, Jerry P Draayer, James P Vary, Daniel Langr, Erik Saule, MA Caprio, U Catalyurek, and Masha Sosonkina. Efficacy of the su (3) scheme for ab initio large-scale calculations beyond the lightest nuclei. *Computer Physics Communications*, 207:202–210, 2016.

[4] Mariette Awad and Rahul Khanna. Support vector regression. In *Efficient learning machines*, pages 67–80. Springer, 2015.

[5] Mehmet Deveci, Kamer Kaya, Bora Uçar, and Ümit V Çatalyürek. Hypergraph partitioning for multiple communication cost metrics: Model and methods. *JPDC*, 77:69–83, 2015.

[6] George Karypis and Vipin Kumar. Multilevel graph partitioning schemes. In *ICPP (3)*, pages 113–122, 1995.

[7] Kamer Kaya, Bora Uçar, and Ümit V Çatalyürek. Analysis of partitioning models and metrics in parallel sparse matrix-vector multiplication. In *Proc. PPAM*, pages 174–184. Springer, 2013.

TABLE IV
OVERALL SPMV ON LOCAL CSR 1D-ROW PARTITIONING(ON SKYLAKE).

Matrices	Nodes	Processes	Time(s)		Error%
			Actual	Predicted	
333SP	4	144	1.2E-03	1.3E-03	9.8%
	5	169	1.1E-03	1.2E-03	12.6%
	7	225	8.4E-04	9.7E-04	15.6%
	8	256	8.2E-04	8.8E-04	6.6%
AS365	4	144	1.3E-03	1.5E-03	9.1%
	5	169	1.2E-03	1.3E-03	10.9%
	7	225	9.3E-04	1.0E-03	11.5%
	8	256	8.6E-04	9.3E-04	7.9%
M6	4	144	1.2E-03	1.4E-03	12.5%
	5	169	1.1E-03	1.2E-03	15.4%
	7	225	8.9E-04	9.8E-04	10.7%
	8	256	8.1E-04	8.9E-04	9.3%
NLR	4	144	1.4E-03	1.6E-03	9.3%
	5	169	1.3E-03	1.4E-03	8.9%
	7	225	9.8E-04	1.1E-03	12.0%
	8	256	9.6E-04	9.9E-04	3.0%
hugetrace-00010	4	144	2.5E-03	2.7E-03	5.3%
	5	169	2.2E-03	2.4E-03	8.8%
	7	225	1.8E-03	1.9E-03	6.8%
	8	256	1.7E-03	1.7E-03	2.6%
road_central	4	144	2.7E-03	2.6E-03	2.1%
	5	169	2.4E-03	2.3E-03	1.5%
	7	225	1.9E-03	1.9E-03	0.5%
	8	256	1.8E-03	1.7E-03	1.2%
road_usa	4	144	4.1E-03	3.8E-03	7.9%
	5	169	3.4E-03	3.4E-03	0.7%
	7	225	2.8E-03	2.8E-03	1.4%
	8	256	2.6E-03	2.6E-03	0.7%

[8] Ping Guo, Liqiang Wang, and Po Chen. A performance modeling and optimization analysis tool for sparse matrix-vector multiplication on gpus. *IEEE TPDS*, 25(5):1112–1123, 2013.

[9] Israt Nisa, Charles Siegel, Aravind Sukumaran Rajam, Abhinav Vishnu, and P Sadayappan. Effective machine learning based format selection and performance modeling for spmv on gpus. In *Proc. IPDPSW*, pages 1056–1065. IEEE, 2018.

[10] Ping Guo and Changjiang Zhang. Performance prediction for csr-based spmv on gpus using machine learning. In *Proc. ICCC*, pages 1956–1960. IEEE, 2018.

[11] Erik G Boman, Karen D Devine, and Sivasankaran Rajamanickam. Scalable matrix computations on large scale-free graphs using 2d graph partitioning. In *Proc. SuperComputing*, pages 1–12, 2013.

TABLE V
OVERALL SPMV ON LOCAL COO 1D-ROW PARTITIONING(ON SKYLAKE).

Matrices	Nodes	Processes	Time(s)		Error%
			Actual	Predicted	
333SP	4	144	1.2E-03	1.1E-03	8.6%
	5	169	1.1E-03	9.6E-04	11.8%
	7	225	8.9E-04	8.2E-04	8.7%
	8	256	8.2E-04	7.8E-04	5.6%
AS365	4	144	1.4E-03	1.2E-03	11.0%
	5	169	1.2E-03	1.1E-03	7.0%
	7	225	9.3E-04	9.1E-04	2.4%
	8	256	8.5E-04	8.7E-04	2.9%
M6	4	144	1.3E-03	1.2E-03	6.9%
	5	169	1.1E-03	1.0E-03	8.9%
	7	225	9.1E-04	8.7E-04	4.5%
	8	256	8.2E-04	8.5E-04	2.7%
NLR	4	144	1.4E-03	1.3E-03	7.8%
	5	169	1.3E-03	1.1E-03	12.1%
	7	225	1.0E-03	9.6E-04	5.7%
	8	256	9.6E-04	9.2E-04	4.4%
hugetrace-00010	4	144	2.5E-03	2.2E-03	12.3%
	5	169	2.2E-03	2.0E-03	8.4%
	7	225	1.8E-03	1.6E-03	6.3%
	8	256	1.6E-03	1.5E-03	5.1%
road_central	4	144	2.5E-03	2.1E-03	13.0%
	5	169	2.1E-03	1.9E-03	9.5%
	7	225	1.7E-03	1.6E-03	7.2%
	8	256	1.6E-03	1.5E-03	9.2%
road_usa	4	144	3.8E-03	2.9E-03	22.9%
	5	169	3.2E-03	2.6E-03	18.1%
	7	225	2.6E-03	2.2E-03	12.2%
	8	256	2.4E-03	2.1E-03	11.6%

TABLE VI
OVERALL SPMV ON GLOBAL CSR 1D-ROW PARTITIONING(ON SKYLAKE).

Matrices	Nodes	Processes	Time(s)		Error%
			Actual	Predicted	
333SP	4	144	2.6E-02	2.1E-02	19.2%
	5	169	2.0E-02	2.0E-02	2.2%
	7	225	1.9E-02	1.8E-02	5.6%
	8	256	2.4E-02	1.9E-02	21.2%
AS365	4	144	2.7E-02	2.2E-02	19.2%
	5	169	2.1E-02	2.0E-02	2.7%
	7	225	2.0E-02	1.9E-02	4.8%
	8	256	2.4E-02	1.9E-02	21.3%
M6	4	144	2.5E-02	2.0E-02	19.2%
	5	169	1.9E-02	1.8E-02	3.2%
	7	225	1.8E-02	1.7E-02	4.5%
	8	256	2.2E-02	1.8E-02	21.1%
NLR	4	144	2.9E-02	2.4E-02	19.2%
	5	169	2.2E-02	2.2E-02	2.3%
	7	225	2.1E-02	2.1E-02	4.1%
	8	256	2.6E-02	2.1E-02	21.3%
hugetrace-00010	4	144	8.2E-02	6.6E-02	20.0%
	5	169	6.2E-02	6.1E-02	0.7%
	7	225	6.4E-02	5.7E-02	10.2%
	8	256	7.3E-02	5.7E-02	22.7%
road_central	4	144	9.6E-02	7.6E-02	20.2%
	5	169	7.1E-02	7.1E-02	0.5%
	7	225	7.4E-02	6.6E-02	10.3%
	8	256	8.6E-02	6.6E-02	23.4%
road_usa	4	144	1.6E-01	1.3E-01	19.3%
	5	169	9.9E-02	1.2E-01	23.8%
	7	225	9.8E-02	1.1E-01	16.5%
	8	256	1.5E-01	1.1E-01	23.1%

TABLE VII
OVERALL SPMV ON GLOBAL COO 1D-ROW PARTITIONING(ON SKYLAKE).

Matrices	Nodes	Processes	Time(s)		Error%
			Actual	Predicted	
333SP	4	144	3.3E-02	2.1E-02	34.4%
	5	169	2.7E-02	2.0E-02	26.0%
	7	225	2.4E-02	1.9E-02	23.9%
	8	256	2.7E-02	1.9E-02	30.9%
AS365	4	144	3.3E-02	2.2E-02	34.4%
	5	169	2.7E-02	2.0E-02	26.1%
	7	225	2.5E-02	1.9E-02	24.3%
	8	256	2.8E-02	1.9E-02	31.0%
M6	4	144	3.1E-02	2.0E-02	35.2%
	5	169	2.5E-02	1.9E-02	26.1%
	7	225	2.3E-02	1.7E-02	23.9%
	8	256	2.6E-02	1.8E-02	30.9%
NLR	4	144	3.6E-02	2.4E-02	34.5%
	5	169	3.0E-02	2.2E-02	26.1%
	7	225	2.7E-02	2.1E-02	24.1%
	8	256	3.0E-02	2.1E-02	31.1%
hugetrace-00010	4	144	9.3E-02	6.6E-02	28.6%
	5	169	6.7E-02	6.2E-02	7.6%
	7	225	7.0E-02	5.7E-02	18.3%
	8	256	7.9E-02	5.7E-02	28.2%
road_central	4	144	1.1E-01	7.7E-02	27.1%
	5	169	7.1E-02	7.2E-02	1.7%
	7	225	8.0E-02	6.7E-02	16.6%
	8	256	9.1E-02	6.6E-02	27.9%
road_usa	4	144	1.8E-01	1.3E-01	26.1%
	5	169	1.5E-01	1.2E-01	16.0%
	7	225	1.4E-01	1.1E-01	20.1%
	8	256	1.5E-01	1.1E-01	27.4%

TABLE VIII
DYNAMIC SVR PERFORMANCE MODEL FOR SPMV ON SKYLAKEX.

Name	Nodes	Processes	Error%				
			CSR L1DR	COO L1DRV	CSR G1DR	COO G1DR	CSR 2DU
333SP	4	144	15.8	15.5	3.11	2.24	0.47
333SP	5	169	17.6	18.7	11.9	6.55	17.4
333SP	7	225	20.3	20.9	8.65	6.21	16.5
333SP	8	256	24.6	22.6	5.36	3.16	13.2
AS365	4	144	20.2	19.1	2.98	2.15	0.618
AS365	5	169	19.7	17.7	11.5	6.58	17.2
AS365	7	225	23.2	20.0	9.61	5.71	20.6
AS365	8	256	22.8	18.1	5.38	3.35	12.3
M6	4	144	20.7	18.8	3.22	3.5	0.178
M6	5	169	19.0	20.4	10.6	6.44	19.8
M6	7	225	23.9	23.8	9.68	6.04	19.7
M6	8	256	25.1	21.1	5.31	3.31	14.1
NLR	4	144	18.2	15.0	2.85	2.07	1.8
NLR	5	169	21.2	18.9	12.2	6.81	15.9
NLR	7	225	22.1	19.8	10.7	6.14	14.5
NLR	8	256	24.6	22.8	5.28	3.26	11.8
hugetrace-00010	4	144	0.609	3.1	0.732	0.784	12.2
hugetrace-00010	5	169	0.358	2.41	18.7	25.7	6.88
hugetrace-00010	7	225	6.04	6.3	8.29	10.2	8.79
hugetrace-00010	8	256	8.02	7.71	3.33	2.02	2.43
road_central	4	144	2.96	6.02	0.47	0.419	8.8
road_central	5	169	1.93	4.52	20.9	36.4	1.11
road_central	7	225	9.17	0.551	8.87	11.6	5.93
road_central	8	256	11.0	3.71	3.65	2.28	3.58
road_usa	4	144	3.95	7.14	0.19	0.0491	7.42
road_usa	5	169	5.75	9.37	48.1	12.0	2.09
road_usa	7	225	3.27	3.87	41.0	6.56	0.00937
road_usa	8	256	4.22	2.11	3.34	1.79	1.98