Performance Model of Iterated SpMV for Distributed System.

Anonymous Author(s)

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.

KEYWORDS

SpMV, MPI, Graph Partitionings

ACM Reference Format:

Anonymous Author(s). 2020. Performance Model of Iterated SpMV for Distributed System.. In *Woodstock '21: 50th International Conference on Parallel Processing (ICPP), August 9-12, 2021, Chicago, Illinois, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 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*. Just considering single node efforts leads to stream of storage format and optimizations. **put a bunch of storage format papers**

To perform sparse matrix-vector multiplication(*SpMV*) on distributed systems, the matrix is usually partitioned 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*. 1D and 2D partitioning techniques are theMany partitioning algorithms have been proposed to ensure good load balance and to minimize *MPI communications* [4–6].

A common problem is to choose which of these many approaches to use. Often, picking the right approach to execute an application based on SpMV relies on the rule of thumbs. Ideally when deploying an application, the user should not have to pick a configuration, the systems should be able to pick a good configuration automatically; or at least provide a set of likely good candidate configuration. We posit that building models to predict the performance of different configurations is the only way to make better-informed decisions. These models could also be used to better understand the structure of the performance of SpMV based application across a wide range of graph and computing platform.

In this paper, we study the modeling of performance of iterated SpMV on distributed memory platform using MPI. We are not concerned with achieving the highest performance, but rather modeling accurately the performance of different configurations. We consider two classic representation of sparse matrices, namely CSR and COO representations. And we consider three distributed memory execution model: 1D partitioning with global communication, and with custom communications, and 2d partitioning with collective communications.

We adapt existing linear and polynomial *Support Vector Regression*(*SVR*) [7] model first proposed for GPUs to predict and analyze the run time of *SpMV* on distributed memory system. We also propose a model built from the ground up from micro-benchmarks of the architectures.

We test the quality of the models of the execution of SpMV of seven real world matrices and of randomly generated matrices on MPI clusters up to 8 physical nodes (256 cores). The experiments show that the best configuration of the execution of SpMV depend on the matrix. The performance models usually have less than 10% error in the predicted runtime. That runtime prediction accuracy is enough to predict accurately the configuration will lead to the best performance in most cases.

We conclude by discussing the applicability of the different models in practice.

2 SPARSE MATRIX VECTOR MULTIPLICATION

2.1 Representation formats

There are many representation format designed to store matrices for computing SpMV.

Yet

2.2 Distributed memory exection

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

ICPP '21, August 9-12, 2021, Chicago, Illinois, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

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. [8] 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 [6]. 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 [5] 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.

2.3 Modeling performance

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 [9] *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 [10, 11] 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.

3 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.

3.1 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 1(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.

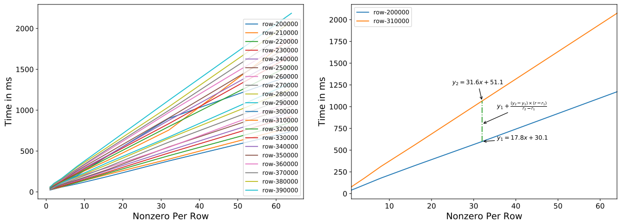
If a test matrix has r row and npr non-zero per row, then our system will find two linear models(L_1, L_2) for row r_1 and r_2 like $r_1 \leq r \leq r_2$. It is important to note that, $L_1(r_1)$ and $L_2(r_2)$ are the two models with immediate smaller and larger rows than r respectively. The following two equations represent the linear model $L_1(r_1)$ and $L2(r_2)$,

$$\begin{aligned} y_1 &= m_1 \times x + c_1 \qquad && \text{for } r_1 \text{ and } x = npr \\ y_2 &= m_2 \times x + c_2 \qquad && \text{for } r_2 \text{ and } x = npr \end{aligned}$$

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 1(b) shows the example for the matrix with 250000 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.



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

(b) Sample equation for the linear model.

Figure 1: Linear relationship between nonzero per row and run time for the SpMV on the uniform 2d-Partitioning.

3.2 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.

3.2.1 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.

3.2.2 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.

3.3 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,

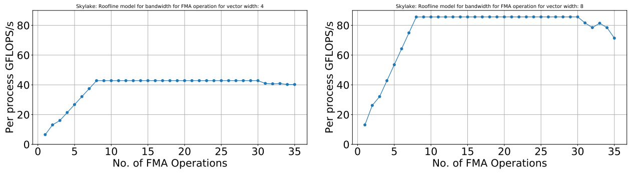
- Core Calculation: performance of the matrix-vector calculation in a MPI node.
- MPI Communication: communication runtime among MPI ranks(processes).

3.3.1 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}

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 2 shows the roofline model of the *FMA* on SkylakeX processor. We can estimate the theoritical peak performance a single FMA by the following equation,



(a) Vector Width: 4

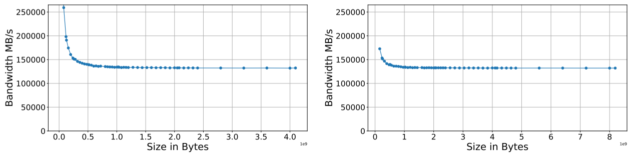
(b) Vector Width: 8

Figure 2: Skylake: (MPI)Roofline model for bandwidth for FMA operation

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

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

Micro-Benchmark for Memory Access Latency. We use the *STREAM* [12] benchmark to find out the bandwidth of the memory access for a selected architecture.



(a) Single Precision.

(b) Double Precision.

Figure 3: Single and double precision memory access bandwidth on Skylake processor.

Figure 3 shows 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. Note that in STREAM, all memory accesses are

Cache Pattern. In SpMV some of the arrays are traverse sequentially and some have irregular pattern. Table 1 shows the memory access property for different matrix representation format. 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.

One could model all irregular accesses as random accesses. However we know that most graphs actually exhbit significant locality [13]. We model the memory accesses by computing a cache friendliness metric which represent the fraction of access that are in cache and the fraction that are in memory.

This cache friendliness is computed for a matrix as follows. We assume the matrix is traversed sequentially and we model the access to the cache assuming the cache has the granularity of a cache line and that the cache replacement policy is LRU. And we assume that the cache is of the size of the L3 cache divided by the number of core on the processor. This model does not accurately capture many properties of the memory subsystem (such as associativity of caches, or cache sharing across multiple cores, the fact that there are multiple levels of caches, and concurrent processes); but it is a simple to compute estimation of how irregular the memory access actually are.

All the sequential data accesses and *hits* of irregular accesses are treated as *Sequential* data accesses. And all cache *miss* in irregular accesses are treated to random data accesses. **then we use memory altency for these?[ans: yes, Bandwidth found from STREAM Benchmark]**

Table 1: 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

3.3.2 *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 4 shows the overall structure of the SPMV model from micro-benchmark.

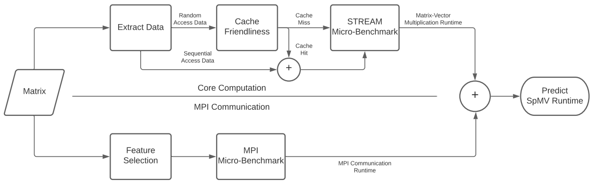


Figure 4: Structure of the SpMV model from micro-benchmark.

4 EXPERIMENTAL SETTINGS

4.1 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.

4.2 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.

5 EXPERIMENTAL RESULTS

5.1 SpMV Model from Micro-Benchmark

5.1.1 Random 2D-SpMV. Table 2 and 3 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 2: Benchmark SpMV Model on Random CSR 2D Partitioning(on Skylake).

Matrices	Nodes	Processes	Time(s)		Error%
			Actual	Predicted	
333SP	4	144	7.2E-03	7.1E-03	1.7%
	5	169	5.7E-03	6.2E-03	9.2%
	7	225	4.7E-03	4.9E-03	3.9%
	8	256	4.2E-03	4.3E-03	1.5%
AS365	4	144	7.4E-03	7.3E-03	1.2%
	5	169	5.8E-03	6.4E-03	9.8%
	7	225	4.7E-03	5.1E-03	8.3%
	8	256	4.4E-03	4.4E-03	1.4%
M6	4	144	6.8E-03	6.6E-03	2.8%
	5	169	5.3E-03	5.8E-03	9.3%
	7	225	4.4E-03	4.6E-03	4.7%
	8	256	4.0E-03	4.0E-03	0.3%
NLR	4	144	8.2E-03	8.2E-03	0.1%
	5	169	6.5E-03	7.2E-03	11.4%
	7	225	5.4E-03	5.7E-03	5.6%
	8	256	4.8E-03	5.0E-03	3.9%
hugetrace-00010	4	144	2.3E-02	2.1E-02	8.2%
	5	169	2.0E-02	1.9E-02	6.7%
	7	225	1.7E-02	1.6E-02	10.1%
	8	256	1.4E-02	1.4E-02	3.4%
road_road	4	144	2.5E-02	2.3E-02	7.4%
	5	169	2.2E-02	2.1E-02	3.5%
	7	225	1.9E-02	1.7E-02	9.4%
	8	256	1.5E-02	1.6E-02	2.4%
road_road	4	144	3.6E-02	3.9E-02	9.0%
	5	169	3.6E-02	3.6E-02	0.1%
	7	225	3.1E-02	3.0E-02	3.1%
	8	256	2.7E-02	2.7E-02	1.7%

Table 3: Benchmark SpMV Model 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_road	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_road	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%

5.1.2 Local 1D-SpMV. Table 4 and 5 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 21.61 and 19.52 respectively.

5.1.3 Local 1D-SpMV. Table **??** and **??** 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.

5.2 Polynomial SVR Model

Table 10 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 10, 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 4: Benchmark SpMV Model on Local CSR 1D-Row Partitioning(on Skylake).

Matrices	Nodes	Processes	Time(s)		Error%
			Actual	Predicted	
333SP	4	144	1.2E-03	1.2E-03	2.3%
	5	169	1.1E-03	1.1E-03	0.2%
	7	225	8.4E-04	8.1E-04	3.8%
	8	256	8.2E-04	7.4E-04	10.7%
AS365	4	144	1.3E-03	1.3E-03	3.9%
	5	169	1.1E-03	1.1E-03	3.4%
	7	225	9.2E-04	8.5E-04	7.5%
	8	256	8.5E-04	7.8E-04	8.5%
M6	4	144	1.3E-03	1.2E-03	4.9%
	5	169	1.1E-03	1.0E-03	2.9%
	7	225	8.7E-04	7.9E-04	8.8%
	8	256	8.3E-04	7.3E-04	12.0%
NLR	4	144	1.4E-03	1.4E-03	2.2%
	5	169	1.3E-03	1.2E-03	5.4%
	7	225	9.9E-04	9.3E-04	6.0%
	8	256	9.5E-04	8.5E-04	10.4%
hugetrace-00010	4	144	2.5E-03	2.7E-03	8.2%
	5	169	2.2E-03	2.4E-03	10.8%
	7	225	1.8E-03	1.9E-03	7.8%
	8	256	1.6E-03	1.7E-03	6.4%
road_road	4	144	2.6E-03	2.9E-03	10.5%
	5	169	2.4E-03	2.6E-03	9.0%
	7	225	1.9E-03	2.1E-03	6.6%
	8	256	1.8E-03	1.9E-03	6.0%
road_road	4	144	4.1E-03	4.1E-03	1.3%
	5	169	3.4E-03	3.7E-03	6.1%
	7	225	2.8E-03	3.0E-03	6.8%
	8	256	2.6E-03	2.8E-03	9.7%

Table 5: Benchmark SpMV Model on Local COO 1D-Row Partitioning(on Skylake).

Matrices	Nodes	Processes	Time(s)		Error%
			Actual	Predicted	
333SP	4	144	1.2E-03	1.2E-03	2.3%
	5	169	1.1E-03	1.1E-03	0.2%
	7	225	8.4E-04	8.1E-04	3.8%
	8	256	8.2E-04	7.4E-04	10.7%
AS365	4	144	1.3E-03	1.3E-03	3.9%
	5	169	1.1E-03	1.1E-03	3.4%
	7	225	9.2E-04	8.5E-04	7.5%
	8	256	8.5E-04	7.8E-04	8.5%
M6	4	144	1.3E-03	1.2E-03	4.9%
	5	169	1.1E-03	1.0E-03	2.9%
	7	225	8.7E-04	7.9E-04	8.8%
	8	256	8.3E-04	7.3E-04	12.0%
NLR	4	144	1.4E-03	1.4E-03	2.2%
	5	169	1.3E-03	1.2E-03	5.4%
	7	225	9.9E-04	9.3E-04	6.0%
	8	256	9.5E-04	8.5E-04	10.4%
hugetrace-00010	4	144	2.5E-03	2.7E-03	8.2%
	5	169	2.2E-03	2.4E-03	10.8%
	7	225	1.8E-03	1.9E-03	7.8%
	8	256	1.6E-03	1.7E-03	6.4%
road_road	4	144	2.6E-03	2.9E-03	10.5%
	5	169	2.4E-03	2.6E-03	9.0%
	7	225	1.9E-03	2.1E-03	6.6%
	8	256	1.8E-03	1.9E-03	6.0%
road_road	4	144	4.1E-03	4.1E-03	1.3%
	5	169	3.4E-03	3.7E-03	6.1%
	7	225	2.8E-03	3.0E-03	6.8%
	8	256	2.6E-03	2.8E-03	9.7%

5.3 Linear Model

Table 8 and 9 show the performance of the linear model on the CSR and COO 2D-Partitioning, where the average errors are recorded 14.02 and 11.53 respectively.

6 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] 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.

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

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

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

[12] John D McCalpin. Stream benchmark. *Link: www. cs. virginia. edu/stream/ref. html# what*, 22, 1995.

[13] Scott Beamer, Krste Asanovic, and David Patterson. Locality exists in graph processing: Workload characterization on an ivy bridge server. In *2015 IEEE International Symposium on Workload Characterization*, pages 56–65, 2015.

Table 8: Linear model performance on CSR 2D-Partitioning(on Skylake).

Matrices	Nodes	Processes	Time(s)		Error%
			Actual	Predicted	
333SP	4	144	7.2E-03	5.9E-03	18.0%
	5	169	5.7E-03	4.6E-03	20.1%
	7	225	4.7E-03	3.9E-03	18.7%
	8	256	4.2E-03	3.7E-03	13.0%
AS365	4	144	7.4E-03	5.7E-03	23.1%
	5	169	5.8E-03	4.7E-03	19.9%
	7	225	4.7E-03	3.9E-03	16.0%
	8	256	4.4E-03	3.5E-03	20.0%
M6	4	144	6.8E-03	5.4E-03	19.9%
	5	169	5.3E-03	4.2E-03	19.9%
	7	225	4.4E-03	3.5E-03	19.4%
	8	256	4.0E-03	3.5E-03	12.0%
NLR	4	144	8.2E-03	7.1E-03	14.0%
	5	169	6.5E-03	5.3E-03	18.4%
	7	225	5.4E-03	4.7E-03	14.0%
	8	256	4.8E-03	4.1E-03	14.4%
hugetrace-00010	4	144	2.3E-02	2.0E-02	12.3%
	5	169	2.0E-02	2.0E-02	0.1%
	7	225	1.7E-02	1.6E-02	5.6%
	8	256	1.4E-02	1.3E-02	3.2%
road_road	4	144	2.5E-02	2.2E-02	11.9%
	5	169	2.2E-02	2.3E-02	3.8%
	7	225	1.9E-02	1.6E-02	17.8%
	8	256	1.5E-02	1.5E-02	4.0%
road_road	4	144	3.6E-02	3.4E-02	6.1%
	5	169	3.6E-02	4.5E-02	24.9%
	7	225	3.1E-02	2.7E-02	13.4%
	8	256	2.7E-02	2.4E-02	8.7%

Table 9: Linear model performance on COO 2D-Partitioning(on Skylake).

Matrices	Nodes	Processes	Time(s)		Error%
			Actual	Predicted	
333SP	4	144	5.4E-03	5.5E-03	2.3%
	5	169	4.2E-03	4.6E-03	10.9%
	7	225	3.5E-03	3.5E-03	0.8%
	8	256	3.4E-03	3.2E-03	3.8%
AS365	4	144	5.5E-03	6.3E-03	13.4%
	5	169	4.3E-03	5.0E-03	17.1%
	7	225	3.6E-03	3.7E-03	2.7%
	8	256	3.6E-03	3.8E-03	4.0%
M6	4	144	5.0E-03	5.7E-03	12.5%
	5	169	3.9E-03	4.2E-03	8.6%
	7	225	3.4E-03	3.3E-03	3.9%
	8	256	3.3E-03	3.1E-03	6.3%
NLR	4	144	6.1E-03	7.3E-03	19.4%
	5	169	4.8E-03	4.8E-03	0.5%
	7	225	4.0E-03	5.2E-03	29.2%
	8	256	3.9E-03	4.0E-03	2.6%
hugetrace-00010	4	144	1.7E-02	2.1E-02	24.4%
	5	169	1.3E-02	1.1E-02	13.8%
	7	225	1.1E-02	1.2E-02	5.7%
	8	256	1.0E-02	9.4E-03	7.8%
road_road	4	144	1.9E-02	2.3E-02	17.6%
	5	169	1.5E-02	1.9E-02	30.3%
	7	225	1.3E-02	1.4E-02	5.2%
	8	256	1.2E-02	1.0E-02	11.0%
road_road	4	144	2.9E-02	2.8E-02	5.3%
	5	169	3.2E-02	3.8E-02	18.3%
	7	225	2.3E-02	1.9E-02	17.2%
	8	256	2.0E-02	2.6E-02	28.2%

Table 10: Dynamic SVR performance model for SpMV on SkylakeX.

Name	Nodes	Processes	Error%					
			CSR L1DR	COO L1DRV	CSR G1DR	COO G1DR	CSR 2DU	COO 2DU
333SP	4	144	15.8	15.5	14.1	8.0	0.7	5.8
	5	169	17.6	18.7	1.4	1.0	17.1	9.7
	7	225	20.3	20.9	5.6	6.1	16.8	8.7
	8	256	24.6	22.6	20.8	16.5	14.4	2.8
AS365	4	144	20.2	19.1	14.0	8.0	0.8	5.9
	5	169	19.7	17.7	0.9	1.0	16.9	9.3
	7	225	23.2	20.0	4.9	6.7	20.9	6.4
	8	256	22.8	18.1	20.9	16.8	13.5	1.9
M6	4	144	20.7	18.8	14.1	9.1	0.0	5.0
	5	169	19.0	20.4	0.3	1.0	19.5	10.1
	7	225	23.9	23.8	4.5	6.1	19.9	4.4
	8	256	25.1	21.1	20.4	16.3	15.3	1.0
NLR	4	144	18.2	15.0	14.0	8.0	2.0	6.4
	5	169	21.2	18.9	1.3	0.9	15.5	7.5
	7	225	22.1	19.8	4.3	6.6	14.7	6.2
	8	256	24.6	22.8	21.3	17.1	13.0	1.0
hugetrace-00010	4	144	0.6	3.1	14.6	6.8	12.3	13.3
	5	169	0.4	2.4	3.2	15.3	7.2	2.0
	7	225	6.0	6.3	11.5	6.8	8.7	1.5
	8	256	8.0	7.7	25.3	21.0	3.4	0.4
road_road	4	144	3.0	6.0	14.8	6.4	8.9	12.3
	5	169	1.9	4.5	4.5	24.9	1.4	3.5
	7	225	9.2	0.6	11.8	6.3	5.8	0.1
	8	256	11.0	3.7	26.3	21.9	4.6	1.4
road_road	4	144	3.9	7.1	14.6	5.6	7.4	1.5
	5	169	5.8	9.4	27.8	2.8	1.8	16.1
	7	225	3.3	3.9	13.7	10.6	0.0	0.3
	8	256	4.2	2.1	26.7	22.0	2.8	2.9

Table 11: Runtime of SpMV and performance of the SVR model(225 MPI process on Skylake).(Time is ms)

Matrices	CSR L1DR		COO L1DR		CSR G1DR		COO G1DR		CSR 2DU		COO 2DU	
	Actual	Pred	Actual	Pred	Actual	Pred	Actual	Pred	Actual	Pred	Actual	Pred
rmat_200M2M	22.41	27.2	40.05	34.23	45.89	40.99	400.43	195.53	7.14	7.14	**4.73**	4.7
rmat_200M3M	27.32	30.56	29.9	34.3	39.68	45.96	243.15	288.49	8.42	9.32	**5.57**	5.76
rmat_400M4M	46.3	51.26	50.79	54.26	92.25	62.47	859.1	440.0	16.4	15.49	**13.91**	10.94
rmat_700M3M	103.86	95.25	149.14	215.75	99.88	90.34	407.89	415.33	25.5	24.12	**24.25**	21.99
rmat_700M4M	117.98	103.63	156.92	157.52	102.77	96.48	588.35	604.14	**30.22**	27.54	31.16	24.05

Table 12: Overall All SpMV model performance on Skylake.

Matrices	Nodes	Prcs	Error													
			Benchmark Model						SVR Model						Linear Model	
			CSR 2DU	COO 2DU	CSR G1DR	COO G1DR	CSR L1DR	COO L1DR	CSR 2DU	COO 2DU	CSR G1DR	COO G1DR	CSR L1DR	COO L1DR	CSR 2DU	COO 2DU
AS365	4	144	1.2%	4.0%	19.2%	34.4%	3.9%	1.2%	0.8%	5.9%	14.0%	8.0%	20.2%	19.1%	23.1%	13.4%
	5	169	9.8%	19.7%	2.7%	26.1%	3.4%	1.9%	16.9%	9.3%	0.9%	1.0%	19.7%	17.7%	19.9%	17.1%
	7	225	8.3%	13.6%	4.8%	24.3%	7.5%	1.0%	20.9%	6.4%	4.9%	6.7%	23.2%	20.0%	16.0%	2.7%
	8	256	1.4%	0.1%	21.2%	31.0%	8.5%	1.1%	13.5%	1.9%	20.9%	16.8%	22.8%	18.1%	20.0%	4.0%
road_road	4	144	7.4%	5.6%	20.2%	27.1%	10.5%	14.3%	8.9%	12.3%	14.8%	6.4%	3.0%	6.0%	11.9%	17.6%
	5	169	3.5%	12.4%	0.6%	1.7%	9.0%	14.3%	1.4%	3.5%	4.5%	24.9%	1.9%	4.5%	3.8%	30.3%
	7	225	9.4%	8.7%	10.3%	16.6%	6.6%	11.4%	5.8%	0.1%	11.8%	6.3%	9.2%	0.6%	17.8%	5.2%
	8	256	2.4%	8.1%	23.4%	27.9%	6.0%	8.4%	4.6%	1.4%	26.3%	21.9%	11.0%	3.7%	4.0%	11.0%
NLR	4	144	0.1%	4.9%	19.1%	34.5%	2.2%	3.7%	2.0%	6.4%	14.0%	8.0%	18.2%	15.0%	14.0%	19.4%
	5	169	11.4%	19.6%	2.3%	26.1%	5.4%	0.1%	15.5%	7.5%	1.3%	0.9%	21.2%	18.9%	18.4%	0.5%
	7	225	5.6%	15.2%	4.1%	24.1%	6.0%	0.6%	14.7%	6.2%	4.3%	6.6%	22.1%	19.8%	14.0%	29.2%
	8	256	3.9%	2.8%	21.3%	31.1%	10.4%	6.4%	13.0%	1.0%	21.3%	17.1%	24.6%	22.8%	14.4%	2.6%
hugetrace-00010	4	144	8.2%	4.2%	20.0%	28.6%	8.2%	9.0%	12.3%	13.3%	14.6%	6.8%	0.6%	3.1%	12.3%	24.4%
	5	169	6.7%	13.7%	0.7%	7.6%	10.8%	11.6%	7.2%	2.0%	3.2%	15.3%	0.4%	2.4%	0.1%	13.8%
	7	225	10.1%	9.5%	10.2%	18.3%	7.8%	9.1%	8.7%	1.5%	11.5%	6.8%	6.0%	6.3%	5.6%	5.7%
	8	256	3.4%	9.2%	22.7%	28.2%	6.4%	7.0%	3.4%	0.4%	25.3%	21.0%	8.0%	7.7%	3.2%	7.8%
333SP	4	144	1.7%	3.7%	19.2%	34.4%	2.3%	2.9%	0.7%	5.8%	14.1%	8.0%	15.8%	15.5%	18.0%	2.3%
	5	169	9.2%	19.6%	2.2%	26.0%	0.2%	0.4%	17.1%	9.7%	1.4%	1.0%	17.6%	18.7%	20.1%	10.9%
	7	225	3.9%	15.5%	5.6%	23.9%	3.8%	1.8%	16.8%	8.7%	5.6%	6.1%	20.3%	20.9%	18.7%	0.8%
	8	256	1.5%	4.3%	21.2%	30.9%	10.7%	6.2%	14.4%	2.8%	20.8%	16.5%	24.6%	22.6%	13.0%	3.8%
M6	4	144	2.8%	3.5%	19.2%	35.2%	4.9%	0.4%	0.0%	5.0%	14.1%	9.1%	20.7%	18.8%	19.9%	12.5%
	5	169	9.3%	18.7%	3.2%	26.1%	2.9%	1.4%	19.5%	10.1%	0.3%	1.0%	19.0%	20.4%	19.9%	8.6%
	7	225	4.7%	9.7%	4.5%	23.9%	8.8%	6.2%	19.9%	4.4%	4.5%	6.1%	23.9%	23.8%	19.4%	3.9%
	8	256	0.3%	0.8%	21.1%	30.9%	12.0%	5.2%	15.3%	1.0%	20.4%	16.3%	25.1%	21.1%	12.0%	6.3%
road_road	4	144	9.0%	3.7%	19.3%	26.1%	1.3%	12.7%	7.4%	1.5%	14.6%	5.6%	3.9%	7.1%	6.1%	5.3%
	5	169	0.1%	12.6%	23.8%	16.0%	6.1%	17.4%	1.8%	16.1%	27.8%	2.8%	5.8%	9.4%	24.9%	18.3%
	7	225	3.1%	5.4%	16.5%	20.1%	6.8%	15.3%	0.0%	0.3%	13.7%	10.6%	3.3%	3.9%	13.4%	17.2%
	8	256	1.7%	7.6%	23.0%	27.4%	9.7%	14.6%	2.8%	2.9%	26.7%	22.0%	4.2%	2.1%	8.7%	28.2%