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].

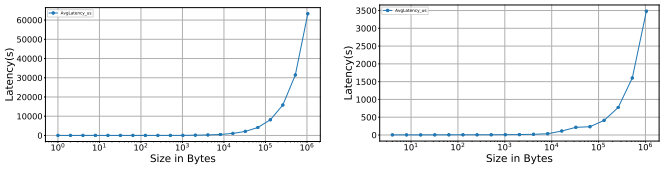
In this paper, we explore two partitioning modes (Uniform 2D-Partitioning and 1D Row Partitioning) and the performance of the different *SpMV* representation based on these partitioning mechanisms on distributed systems. We develop a linear and a polynomial support vector regression (*SVR*) performance 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. SpMV on AVX-512 Architecture (Skylake)

SpMV on the distributed system initially can be divided into two parts,

- MPI Communication: latency to share the vector and gather the results from the MPI process.
- Core SpMV Calculation: latency to calculate part of SpMV for a single MPI process.

A. MPI communication



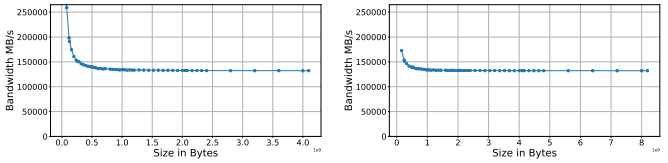
(a) MPI All_GatherV. (b) MPI All_Reduce.

Fig. 1. Latency of the MPI collectives(All_GatherV and All_Reduce) on Skylake.

B. Core SpMV Calculation

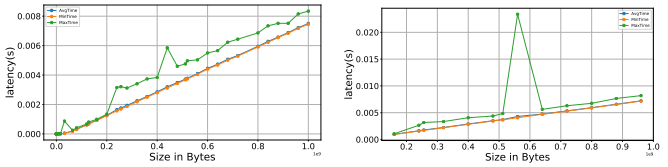
We can represent the basic SpMV by $y = y + Val * x$, which contains two floating point operation(multiply and addition). So, we can say we need to calculate NNZ(number of non zeros) times FMA(fused multiply addition) to perform SpMV. In our experiment we divide the core SpMV calculation into two major parts,

- Run time for FMA: $L_{FMA} \times NNZ$, where L_{FMA} is the latency of a single FMA and NNZ is number of nonzeros.
- Read-write latency: L_{RW}



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

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

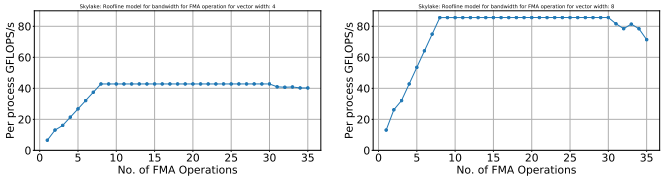


(a) Double Precision. (b) Single Precision.

Fig. 3. single and double precision memory access time on Skylake processor.

1) Memory Access bandwidth: Figure 3 shows, when the size of the data is more than 1GB the bandwidth shows the consistency with the size of the data(132 GB/s).

2) Roofline Model for FMA: The throughput of Skylake and Cascade Lake are same as 2 instructions per cycle and the latency of FMA is 4 cycles for both of them. So there is a potential of pipelinling($2 \times 4 = 8$) to get the optimal results. Now, both of the architecture 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, both of the architecture have 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 informations 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.



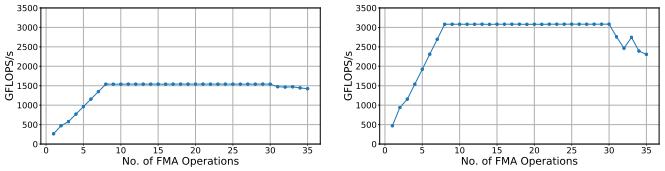
(a) Vector Width: 4 (b) Vector Width: 8

Fig. 4. Skylake: (MPI)Roofline model for bandwidth for FMA operation

We can estimate the theoritical peak performance a single FMA by the following equation,

$$P = Base\_Clock\_Frequency \times Vector\_Width \times \frac{FLOPs}{Instruction}. \quad (1)$$

C. Random 2D-SpMV



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

Fig. 5. Skylake: (Shared memory parallel)Roofline model for bandwidth for FMA operation

TABLE I
SPMV PROPERTY.

Array	Elements Size	Data Type	Access Type	
			CSR	COO
rowA	2×ppn×rpp	Integer	Sequential	Sequential
colA	ppn×nnz	Integer	Sequential	Sequential
valA	ppn×nnz	Floating	Sequential	Sequential
x	ppn×nnz	Floating	Random	Random
y	ppn×rpp	Floating	Sequential	Random

1) CSR:

D. Global 1D-SpMV

1) CSR:

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 II
SPMV ON RANDOM 2D PARTITIONING(SINGLE PRECISION).

Matrices	Nodes	nProcesses	Actual Time	Predicted Time	Error
AS365	4	144	3.285	3.764	14.55%
AS365	5	169	2.832	3.079	8.716%
AS365	7	225	1.949	2.131	9.32%
AS365	8	256	1.738	1.800	3.592%
NLR	4	144	3.674	4.406	19.91%
NLR	5	169	3.184	3.627	13.93%
NLR	7	225	2.416	2.544	5.314%
NLR	8	256	1.960	2.162	10.29%
hugetrace-00010	4	144	9.267	10.787	16.4%
hugetrace-00010	5	169	9.069	9.303	2.582%
hugetrace-00010	7	225	7.102	7.155	0.7472%
hugetrace-00010	8	256	5.629	6.361	13.0%
nlpkkt160	4	144	21.370	44.609	108.7%
nlpkkt160	5	169	18.174	37.239	104.9%
nlpkkt160	7	225	15.164	26.811	76.81%
nlpkkt160	8	256	12.869	23.054	79.14%
nlpkkt200	4	144	54.002	99.510	84.27%
nlpkkt200	5	169	46.353	84.164	81.57%
nlpkkt200	7	225	33.822	62.276	84.13%
nlpkkt200	8	256	28.792	54.319	88.66%
nlpkkt240	4	144	92.395	181.450	96.38%
nlpkkt240	5	169	77.098	154.220	100.0%
nlpkkt240	7	225	63.087	115.240	82.66%
nlpkkt240	8	256	54.864	101.020	84.12%
road_central	4	144	9.509	11.049	16.2%
road_central	5	169	9.636	9.584	0.5457%
road_central	7	225	7.704	7.454	3.24%
road_central	8	256	6.043	6.663	10.25%

TABLE III
SPMV ON GLOBAL 1D-ROW PARTITIONING(SINGLE PRECISION).

Matrices	Nodes	nProcesses	Actual Time	Predicted Time	Error
rhugetrace-00010	4	144	0.835	1.034	23.92%
hugetrace-00010	5	169	0.682	0.768	12.47%
nlpkkt160	4	144	2.728	5.167	89.42%
nlpkkt160	5	169	2.189	4.323	97.54%
nlpkkt160	7	225	1.595	3.201	100.7%
nlpkkt160	8	256	1.427	2.793	95.7%
nlpkkt200	4	144	5.163	10.144	96.47%
nlpkkt200	5	169	4.301	8.629	100.6%
nlpkkt200	7	225	3.112	6.434	106.8%
nlpkkt200	8	256	2.679	5.661	111.3%
nlpkkt240	4	144	8.711	17.648	102.6%
nlpkkt240	5	169	7.389	15.014	103.2%
nlpkkt240	7	225	5.363	11.235	109.5%
nlpkkt240	8	256	4.524	9.871	118.2%
road_central	4	144	0.961	0.976	1.572%
road_central	5	169	0.850	0.769	9.569%