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

Lots of different performance model for the *SpMV* exist. In particular, Guo and Wang *et.al.* [8] presented a performance model for the general purpose GPU. They provide a linear model that can predict the performance of the matrix based on the matrix strides size and nonzero per row. Guo *et.at.* [9] also provide a *SVM* machine learning 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. [10] 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 $\lceil \frac{matrix\_size}{p} \rceil$ 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.

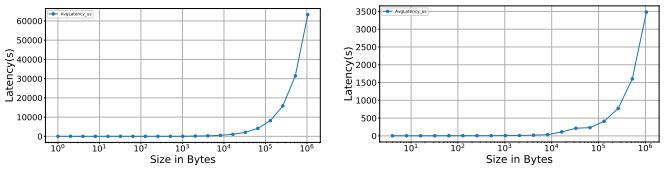
The local matrix stored by the nodes can be represented in a number of formats. We picked the two most popular formats: CSR and COO.

IV. 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



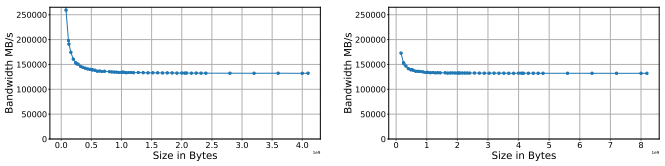
(a) MPI All_GatherV. (b) MPI All_Reduce.

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

B. Core SpMV Calculation

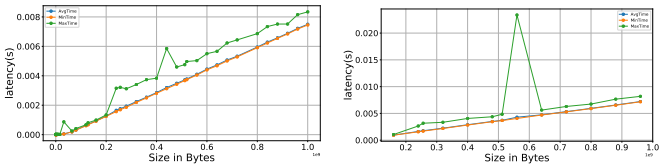
We can represent the basic SpMV by $y = y + Val * x$, which contains two floating point operation(multiply and addition). So, we can say we need to calculate NNZ(number of non zeros) times FMA(fused multiply addition) to perform SpMV. In our experiment we divide the core SpMV calculation into two major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

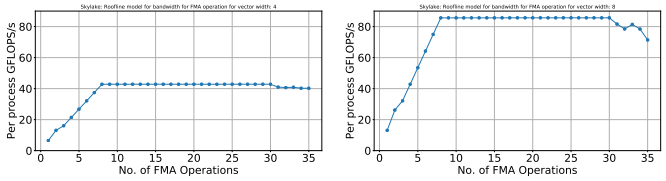


(a) Double Precision. (b) Single Precision.

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

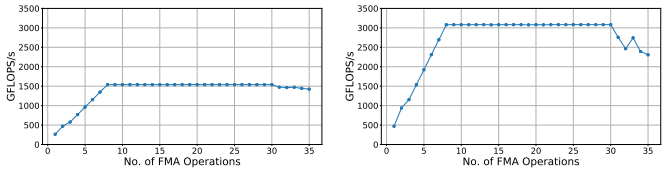
1) Memory Access bandwidth: Figure 3 shows, when the size of the data is more than 1GB the bandwidth shows the consistency with the size of the data(132 GB/s).

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



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

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

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

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

V. SpMV Model from Micro-Benchmark

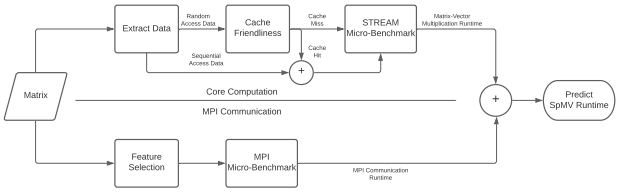


Fig. 6. Structure of the SpMV model from micro-benchmark.

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

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%

A. Random 2D-SpMV

1) CSR:

2) COO:

B. Local 1D-SpMV

1) CSR:

2) COO:

C. Global 1D-SpMV

1) CSR:

2) COO:

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.

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%

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%

[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.

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

[10] 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%