Randomization of Sparse Matrix by Vector Multiplication

ABHISHEK JAIN, ISMAIL BUSTANY, and PAOLO D'ALBERTO

A sparse matrix by vector multiplication (SpMV) is simplified by the matrix non-zero elements and how we store them. There are many SpMV applications, many matrix storage formats, and thus algorithms. However, there is no optimality without considering the architecture: for example, the CPU is only one among ... many.

By nature, randomization is resilient to counter techniques, thus suitable to avoid worst case scenarios, improve performance on average, and reduce performance variance; however, it does to the best case the same thing it does to the worst case, it can nudge it off. Like preconditioning, randomization is advantageous when the matrix is reused or a constant such as in the power method, Krilov's space, or convolutions for image classifications. Randomization is also an optimization that any architecture may take advantage although in different ways.

We shall present cases where we can improve by 15% performance for general purpose architectures and by 8x for custom architectures.

ACM Reference Format:
Abhishek Jain, Ismail Bustany, and Paolo D'Alberto. 2020. Randomization of Sparse Matrix by Vector Multiplication . 1, 1 (May 2020), 7 pages.

1 INTRODUCTION

B.S. Goes here.

2 BASIC NOTATIONS

Let us start by describing the basic notations so we can clear the obvious (or not). A Sparse-matrix vector multiplication *SpMV* on an (semi) ring based on the operations $(+, *)$ is defined as $\boldsymbol{y} = \mathbb{M}\boldsymbol{x}$ so that $y_i = \sum_j M_{i,j} * y_j$ where $M_{i,j}$=0 are not even represented and stored. Most of the experimental results in Section 9 are based on the classic addition (+) and multiplication (*) in floating point precision using 32 or 64bits (i.e., single and double floating point precision). SpMV based on semi-ring (min,+) is a short path algorithm based on an adjacent matrix of a graph, and using a Boolean algebra we can check if two nodes are connected, which is slightly simpler.

We identify a sparse matrix $\mathbb{M}$ of size $M \times N$ as having $O(M + N)$ non-zero elements, number of non zero *nnz*. Thus the complexity of $\mathbb{M}\boldsymbol{x}$ is $O(M + N) = 2nnz$. Of course, the definition of sparsity may vary. We represent the matrix $\mathbb{M}$ by using the Coordinate *COO* or and the compressed sparse row *CSR*[1] format. The COO represents the non-zero of a matrix by a triplet (i, j, val), very often there are three identical-in-size vectors for the ROW, COLUMN, and VALUE. The COO format takes $3 \times nnz$ space and two consecutive elements in the value array are not bound to be neither in the same row nor column. In fact, we know only that $VALUE[i] = M_{ROW[i],COLUMN[i]}$.

[1] a.k.a. Compressed row storage CRS.

Authors' address: Abhishek Jain; Ismail Bustany; Paolo D'Alberto.

The CSR stores elements in the same row and with increasing column values consecutively. There are three arrays V, COL, and ROW. The ROW is sorted in increasing order, its size is M, and $ROW[i]$ is an index in V and COL describing where row-i starts (i.e., if row i exists). We have that $M_{i,*}$ is stored in $V[ROW[i] : ROW[i+1]]$ and the column are at $COL[ROW[i] : ROW[i+1]]$ and sorted increasingly. The CSR takes $2 \times nnz + M$ space and a row vector of the matrix can be found in $O(1)$.

The computation as $y_i = \sum_j M_{i,j} * x_j$ is a sequence of dot products and the CSR representation is a natural:

$$Index = ROW[i] : ROW[i+1]$$
$$y_i = \sum_{i \in Index} V[i] * x_{COL[i]}$$

The matrix row is contiguous (in memory) and contiguous rows are contiguous. The access of the (dense) vector $\boldsymbol{x}$ could have no pattern. The COO format could use a little preparation: For example, we can sort the array by row and add row information to achieve the same properties of CSR; however transposing a COO matrix is just a swap of the array ROW and COL. Think about matrix multiply. As today, each dot product achieves peak performance if the reads of the vector $\boldsymbol{x}$ are streamlined as much as possible and so the reads of the vector V. If we have multiple cores, each could compute a sub set of the y_i and a clean data load balancing can go a long way. If we have a few functional units, we would like to have a constant stream of independent $*$ and $+$ operations but with data already in registers: that is, data pre-fetch will go a long way especially for $x_{COL[i]}$, which may have an irregular pattern.

3 RANDOMIZATION

We refer to *Randomization* as row or column permutations of the matrix $\mathbb{M}$ (thus a permutation of $\boldsymbol{y}$ and $\boldsymbol{x}$) and we choose these by a pseudo-random process. Why we want to introduce uncertainty? The sparsity of our matrix $\mathbb{M}$ has a pattern representing the nature of the original problem; such a pattern may exploit the wrong computation for an architecture; we could break such a pattern so that the only property left is a uniform distribution (of some sort). We must avoid the worst case and we would opt for an average case instead and we could do this to a class of $\mathbb{M}$. This is the gist.

If we know the matrix $\mathbb{M}$ and we know the architecture, preconditioning must be a better solution. Well, it is. If we run experiments long enough, we choose the best permutations for the architecture, permute $\mathbb{M}$, and go on testing the next. On one end, preconditioning exerts a full understanding of both the matrix (the problem) and how the final solution will be computed (architecture). This is the culminating point of knowing and we must strive to it. On the other end, the simplicity of a random permutation requires no information about the matrix, the vector, and the architecture. Such a simplicity can be exploited directly in HW. We are after an understanding when randomization is just enough: we want to let the hardware do its best with the least effort, or at least with the appearance to be effortless. Also we shall show there are different flavors of random.

Interestingly, this work stems from a sincere surprise about randomization efficacy and its application on custom SpMV. Here, we want to study this problem systematically so that to help future hardware designs. Intuitively, if we can achieve a uniform distribution of the rows of matrix $\mathbb{M}$ we can have provable expectation of its load balancing across multiple cores. If we have a uniform distribution of accesses on $\boldsymbol{x}$ we could exploit column load balancing and exploit better sorting algorithms: in practice the reading of $\boldsymbol{x}_{COL[i]}$ can be reduces to a sorting and we know that different sparsity may require different algorithms. This is a lot to unpack but this translates as better performance of the sequential algorithm without changing the algorithm.

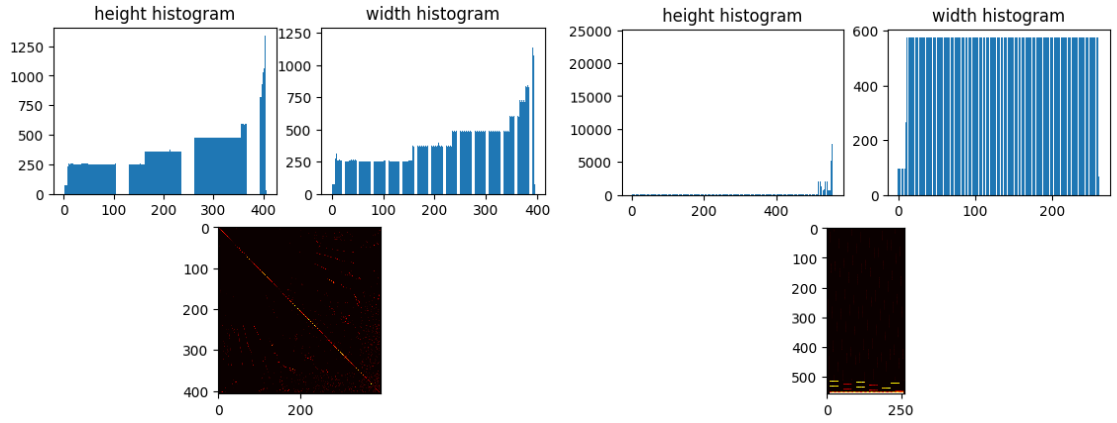

Fig. 1. Left: OPF 3754. Right: LP OSA 07. These are histograms where we represent normalized buckets and counts

We will show that (different) randomness affects architectures and algorithms differently making it a suitable optimization especially when the application and hardware are at odds. We want to show that there is a randomness hierarchy that we can distinguish as global and local; there are simple-to-find cases where the sparsity breaks randomness and the matrix has to be split into components. We want to show that this study uses common tool, open software tools and sometimes naive experiments; however, we can infer properties applicable to proprietary and custom solutions.

4 ENTROPY

Patterns in sparse matrices are often visually pleasing, see Figure 1 where we present the height histogram, the width histograms and a two-dimensional histogram as heat map. We will let someone else using AI picture classification. Intuitively, we would like to express a measure of uniform distribution and here we apply the basics: *Entropy*. Given an histogram $i \in [0, M-1]$ $h_i \in \mathbb{N}$, we define $S = \sum_{i=0}^{M-1} h_i$ and thus we have a probability distribution function $p_i = \frac{h_i}{S}$. The *information* of bin i is defined as $I(i) = -\log_2 p_i$. If we say that the stochastic variable X has PDF p_i than the entropy of X is defined as.

$$H(x) = -\sum_{i=0}^{M-1} p_i \log_2 p_i = \sum_{i=0}^{M-1} p_i I(i) = E[I_x] \tag{1}$$

The maximum entropy is when $\forall i, p_i = p = \frac{1}{M}$; that is, we are observing a uniform distributed event. There is no conceptual difference when the PDF represents a two dimensional distribution. Thus our randomization should aim at higher entropy numbers.

The entropy for matrix LP OSA 07 is 8.41 and for OPF 3754 is 8.39. A single number is satisfying because concise.

5 UNIFORM DISTRIBUTION

We know that we should **not** compare the entropy numbers of two matrices because there entropy does not use any information about the order of the buckets. By construction, the matrices are quite different in sparsity, ins shapes and their entropy numbers are so close. To appreciate their difference, we should compare their distributions by Jensen-Shannon measure (which is a symmetric). Or we could use a representation of a hierarchical 2d-entropy, see Figure

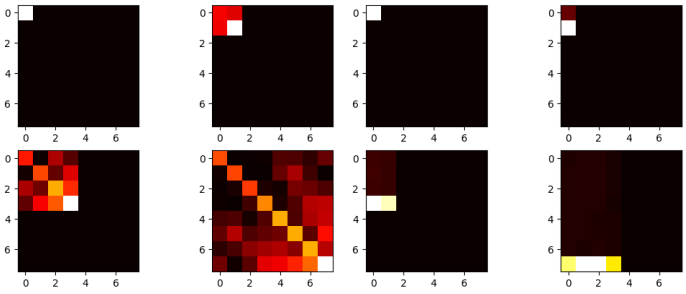

Fig. 2. Hierarchical 2D entropy for OPF 3754 (left) and LP OSA 07 (right).

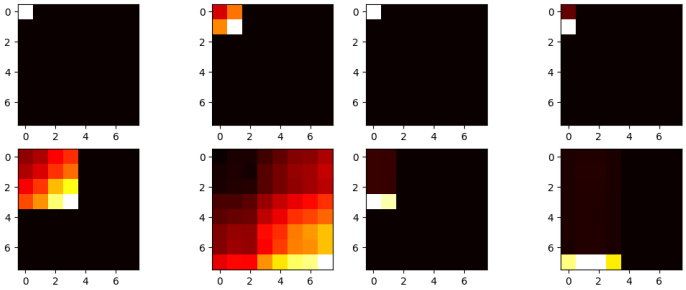

Fig. 3. Hierarchical 2D entropy after row and column random permutation for OPF 3754 (left) and LP OSA 07 (right).

2, where the entropy is split into 2x2, 4x4 and 8x8 (or fewer if the distribution is not square). We have a hierarchical entropy heat maps.

We can see a more granular entropy measure summarizes better the nature of the matrix. In this work, the entropy vector is used mostly for visualization purpose more than for comparison purpose. Of course, we can appreciate how the matrix LP OSA 07 has a few very heavy rows and they are clustered. This matrix will help up in showing how randomization need some tips. Now we apply row and column random permutation once by row and one by column: Figure 3: OPF has now entropy 11.27 and LP 9.26. The numerical difference is significant. The good news is that for entropy, being an expectation, we can use simple techniques like bootstrap to show that the difference is significant or we have shown that Jensen-Shannon can be used and a significance level is available. What we like to see is the the hierarchical entropy heat map is becoming *more* uniform for at least one of the matrix.

In practice, permutation need some help especially for relatively large matrices. As you can see, the permutation affects locally the matrix. Of course, it depends on the implementation of the random permutation (we use numpy for this) but it is reasonable a slightly modified version of the original is still a random selection but unfortunately they seem more likely than they should. We need to compensate or help the randomization so that this current implementation does not get too lazy.

If we are able to identify the row and column that divide high and low density, we could use them as pivot for a shuffle like in a quick-sort algorithm. We could apply a sorting algorithm but its complexity will the same of SpMV. We use a gradients operations to choose the element with maximum steepness, Figure 4 and 5

LP achieves entropy 8.67 and 9.58 and OPF achieves 10.47 and 11.40.

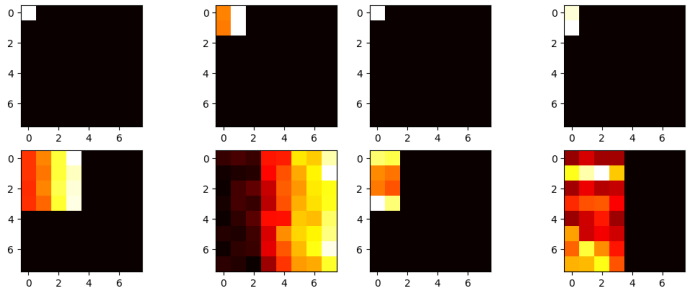

Fig. 4. Hierarchical 2D entropy after height gradient based shuffle and row random permutation for OPF 3754 (left) and LP OSA 07 (right).

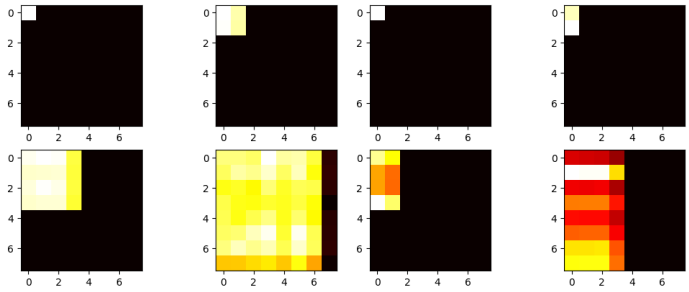

Fig. 5. Hierarchical 2D entropy after height and width gradient shuffle and row and column random permutation for OPF 3754 (left) and LP OSA 07 (right).

If the goal is to achieve a uniform matrix sparsity, it seems that we have the basic tool to compute and to measure such a sparsity. We admit that we do not try to find the best permutation. But our real goal is to create a work bench where randomization can be tested on different architectures and different algorithms.

6 WORKLOAD AND PARALLELISM

What we do not want to have is a partition into lots of smaller computations with very different workloads. Here we may want to indicate cases where we could split the matrix into sparse and dense formats.

7 GUIDING RANDOMIZATION

There is random and there is Random.

8 CALL FOR A DIFFERENT STRATEGY

We want to find out randomization techniques that are suitable for custom hardware but also what are the most common and simple heuristics that can justified for any hardware.

9 EXPERIMENTAL RESULTS

Plots and pots.

Matrix		np		FIJI		ELLES		VEGA				THE
bits	64			64		64		64				Rt
	H	COO	CSR	COO	CSR	COO	CSR	COO	CSR	Cores	CSR	
lp osa 07	8.41	0.41	1.28	*4.27	*10.44	*4.92	*9.21	*7.92	*19.28	1	0.999	1.03
Row-Premute	9.26	0.36	1.34	4.20	9.74	4.68	8.65	7.85	17.25	1	0.902	1.03
Grad-Row-Permute	8.68	*0.91	1.26	4.23	9.88	4.88	8.52	7.85	18.30	12	1.681	1.60
Grad-Row-Colum	9.60	0.48	1.24	4.18	9.08	4.70	7.92	7.75	16.40	15	*1.799	1.58
Row-Colum-Permute	9.26	0.37	*1.35	4.20	9.75	4.70	8.45	7.79	17.23	1	0.925	1.03
OPF 3754	8.39	0.37	*1.45	*4.43	*16.17	*5.74	*9.24	*7.87	*27.12	15	1.573	1.93
Row-Premute	11.27	0.35	1.37	4.24	12.39	5.36	7.95	7.56	20.96	14	1.516	1.93
Grad-Row-Permute	10.47	0.37	0.84	4.38	14.38	5.58	8.91	7.77	24.62	13	1.724	1.43
Grad-Row-Colum	11.40	*0.82	0.84	4.24	11.50	5.36	7.85	7.57	21.01	14	*1.726	1.48
Row-Colum-Permute	11.27	0.36	1.34	4.24	12.35	5.36	8.01	7.62	21.39	14	1.458	1.98

Matrix		np		FIJI			ELLES			VEGA			THE				
	64	64		32		64		32		64		32		64			Rt
	H	COO	CSR	COO	CSR	COO	CSR	COO	CSR	COO	CSR	COO	CSR	COO	CSR	Cores	CSR	
mult dcop 03	9.69	0.38	0.79	5.64	0.00	5.13	10.38	7.10	0.00	5.69	7.54	9.99	0.00	9.05	19.44	15	1.238	2.33
Row-Premute	10.74	0.51	0.79	5.47	0.00	4.99	9.53	6.89	0.00	5.54	6.82	9.81	0.00	8.58	17.30	13	1.227	2.01
Grad-Row-Permute	10.58	0.37	0.76	5.58	12.80	5.08	10.52	7.03	10.91	5.58	7.44	9.62	22.91	8.64	19.74	13	1.618	1.64
Grad-Row-Colum	10.83	0.38	0.76	5.43	12.14	5.03	10.18	6.85	10.27	5.52	7.24	9.58	22.24	8.98	19.01	15	1.588	1.79
Row-Colum-Permute	10.74	0.38	0.79	5.43	0.00	4.97	9.50	6.88	0.00	5.50	6.80	9.64	0.00	8.52	17.22	11	1.235	1.83
mult dcop 01	9.69	0.43	0.79	5.65	0.00	5.14	10.44	7.12	0.00	5.69	7.51	9.73	0.00	8.63	19.66	10	1.224	1.94
Row-Premute	10.74	0.41	0.79	5.42	0.00	4.96	9.36	6.88	0.00	5.52	6.80	9.55	0.00	8.62	17.15	13	1.295	1.94
Grad-Row-Permute	10.58	0.38	0.76	5.56	13.14	5.10	10.56	7.07	10.61	5.62	7.32	9.60	22.93	8.79	19.07	15	1.623	1.76
Grad-Row-Colum	10.84	0.37	0.76	5.48	12.64	4.99	10.36	6.88	10.17	5.54	7.07	9.51	21.78	8.44	17.65	12	1.639	1.57
Row-Colum-Permute	10.74	0.64	0.79	5.47	0.00	4.99	9.47	6.93	0.00	5.50	6.82	9.64	0.00	8.66	17.12	12	1.244	1.91
mult dcop 02	9.69	0.64	0.79	5.63	0.00	5.13	10.25	7.09	0.00	5.70	7.51	9.66	0.00	8.71	19.65	11	1.200	2.06
Row-Premute	10.74	0.42	0.79	5.47	0.00	5.00	9.42	6.92	0.00	5.54	6.82	9.52	0.00	8.68	17.08	12	1.259	1.90
Grad-Row-Permute	10.58	0.37	0.77	5.54	12.29	5.08	10.07	7.01	10.66	5.60	7.34	9.70	22.89	8.68	19.89	12	1.639	1.56
Grad-Row-Colum	10.84	0.51	0.77	5.48	11.93	5.02	9.91	6.84	9.90	5.53	7.02	9.53	21.74	8.54	18.49	11	1.626	1.50
Row-Colum-Permute	10.74	0.36	0.80	5.43	0.00	4.98	9.49	6.90	0.00	5.51	6.79	9.52	0.00	8.58	17.27	13	1.262	1.98
bloweya	7.20	0.38	1.45	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	14	1.766	1.58
Row-Premute	11.03	0.40	1.21	4.35	11.11	4.02	9.50	5.97	10.06	4.88	6.49	7.79	19.39	7.04	17.80	14	1.676	1.54
Grad-Row-Permute	10.30	0.81	0.86	4.50	12.19	4.10	10.19	6.36	12.01	5.07	7.31	7.83	21.88	7.07	18.97	16	1.694	1.53
Grad-Row-Colum	10.88	0.37	0.87	4.34	11.24	3.98	9.64	5.92	10.84	4.84	6.70	7.66	18.87	6.95	17.38	16	1.704	1.51
Row-Colum-Permute	11.03	0.34	1.19	4.35	11.11	4.01	9.33	5.98	9.84	4.85	6.50	7.63	19.89	7.07	16.85	13	1.653	1.53
lp osa 07	8.41	0.41	1.28	4.67	12.81	4.27	10.44	6.41	15.33	4.92	9.21	8.42	20.79	7.92	19.28	1	0.999	1.03
Row-Premute	9.26	0.36	1.34	4.61	11.69	4.20	9.74	6.04	12.49	4.68	8.65	8.37	19.82	7.85	17.25	1	0.902	1.03
Grad-Row-Permute	8.68	0.91	1.26	4.64	12.21	4.23	9.88	6.33	13.36	4.88	8.52	8.34	17.73	7.85	18.30	12	1.681	1.60
Grad-Row-Colum	9.60	0.48	1.24	4.58	10.97	4.18	9.08	6.09	10.88	4.70	7.92	8.29	18.04	7.75	16.40	15	1.799	1.58
Row-Colum-Permute	9.26	0.37	1.35	4.60	11.85	4.20	9.75	6.02	12.47	4.70	8.45	8.37	18.23	7.79	17.23	1	0.925	1.03
ex19	8.23	0.42	1.29	7.07	27.61	6.17	20.84	6.40	15.61	5.87	10.90	12.61	41.53	11.52	37.96	10	2.144	1.51
Row-Premute	11.84	0.42	1.25	6.61	18.87	5.84	14.64	4.55	12.09	5.06	9.07	12.23	31.70	11.13	27.34	13	2.074	1.77
Grad-Row-Permute	10.75	0.39	1.20	6.95	25.29	6.12	18.20	6.24	14.72	5.74	10.71	12.54	40.60	11.45	36.74	9	2.049	1.45
Grad-Row-Colum	11.88	0.38	1.20	6.59	18.55	5.81	14.32	4.51	12.21	5.09	9.00	12.25	32.80	11.05	27.03	8	2.006	1.36
Row-Colum-Permute	11.84	0.37	1.25	6.62	18.76	5.86	14.96	4.52	12.09	5.11	9.05	12.29	31.51	11.15	27.30	12	2.128	1.66
brainpc2	7.48	0.90	1.45	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	15	1.391	2.34
Row-Premute	9.81	0.37	1.20	5.22	12.68	4.78	10.57	6.97	10.92	5.64	7.51	9.15	24.32	8.29	19.78	15	1.389	2.05
Grad-Row-Permute	9.72	0.39	0.71	5.28	13.99	4.82	11.13	7.10	11.65	5.66	7.73	9.30	24.77	8.54	20.49	13	1.718	1.45
Grad-Row-Colum	10.37	0.37	0.71	5.21	13.26	4.76	10.48	6.99	11.51	5.60	7.57	9.18	23.30	8.28	20.49	14	1.685	1.53
Row-Colum-Permute	9.81	0.35	1.18	5.19	12.83	4.76	10.50	6.96	11.28	5.60	7.48	9.21	23.63	8.33	20.01	15	1.402	2.04
shermanACb	8.60	0.38	0.80	4.50	12.65	4.09	10.07	6.10	13.48	4.61	7.74	7.92	20.27	7.29	18.98	11	1.466	1.48
Row-Premute	10.38	0.36	0.81	4.41	11.22	4.03	8.88	5.59	11.14	4.18	7.14	7.90	19.16	7.13	17.87	13	1.366	1.58
Grad-Row-Permute	9.92	0.82	0.68	4.47	12.04	4.08	9.35	5.85	12.72	4.36	7.44	7.87	18.85	7.17	16.06	12	1.424	1.51
Grad-Row-Colum	10.59	0.40	0.68	4.41	11.19	4.04	8.79	5.65	11.51	4.27	7.14	7.80	18.28	7.14	15.39	11	1.423	1.44
Row-Colum-Permute	10.38	0.35	0.81	4.40	11.32	4.03	9.11	5.66	11.28	4.18	7.25	7.81	19.15	7.14	17.07	11	1.337	1.51
cvxqp3	8.65	0.42	1.32	3.86	14.45	3.50	13.05	5.24	18.77	4.26	8.79	6.92	21.71	6.24	20.87	15	1.758	1.36
Row-Premute	11.03	0.38	1.22	3.69	12.32	3.36	11.18	4.68	12.19	3.99	8.27	6.61	20.44	6.03	18.98	12	1.738	1.26
Grad-Row-Permute	11.06	0.37	0.65	3.69	12.38	3.37	10.66	4.71	11.90	4.00	7.96	6.60	21.49	6.06	18.24	13	1.447	1.36
Grad-Row-Colum	11.13	0.37	0.65	3.67	12.00	3.37	10.71	4.71	11.79	4.00	7.92	6.60	20.28	6.05	19.31	13	1.414	1.39
Row-Colum-Permute	11.03	0.39	1.22	3.68	12.21	3.36	11.21	4.70	12.15	4.01	8.21	6.60	20.46	6.01	19.02	13	1.734	1.31
case9	7.38	0.90	1.53	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	16	1.571	1.89
Row-Premute	10.04	0.50	1.37	4.53	12.89	4.14	10.40	6.39	12.58	5.38	7.84	8.02	22.33	7.40	18.79	15	1.573	1.66
Grad-Row-Permute	9.70	0.37	0.92	4.54	13.07	4.17	10.49	6.55	13.96	5.44	8.00	7.99	19.76	7.45	20.33	11	1.438	1.59
Grad-Row-Colum	10.63	0.37	0.91	4.47	11.76	4.10	9.57	6.39	11.91	5.37	7.54	7.95	19.06	7.42	18.47	11	1.433	1.61
Row-Colum-Permute	10.04	0.37	1.29	4.52	12.97	4.12	10.45	6.40	12.85	5.35	7.79	8.02	21.30	7.40	18.84	15	1.571	1.67
TSOPF FS b9 c6	7.38	0.38	1.53	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	16	1.564	1.89
Row-Premute	10.05	0.39	1.42	4.52	12.98	4.13	10.47	6.40	12.85	5.35	7.83	8.03	22.36	7.32	18.84	15	1.589	1.63
Grad-Row-Permute	9.71	0.39	0.92	4.56	13.43	4.15	10.85	6.55	13.97	5.46	7.94	8.02	19.48	7.39	18.69	12	1.600	1.48
Grad-Row-Colum	10.64	0.39	0.92	4.47	12.13	4.10	9.44	6.34	11.90	5.40	7.56	7.94	18.95	7.29	17.69	14	1.588	1.58
Row-Colum-Permute	10.04	0.37	1.32	4.51	12.96	4.12	10.49	6.41	12.50	5.34	7.87	8.01	21.22	7.38	18.94	15	1.547	1.69
OPF 6000	8.80	0.38	1.39	8.09	27.37	7.28	20.88	8.92	14.36	7.45	11.23	13.90	46.28	12.77	45.18	14	1.887	2.29
Row-Premute	11.88	0.38	1.22	7.24	16.93	6.67	13.99	7.94	10.54	6.80	8.92	12.98	29.63	11.65	25.97	13	1.775	2.27
Grad-Row-Permute	11.11	0.37	0.84	7.85	23.65	7.12	18.35	8.79	13.56	7.38	11.04	13.46	39.90	12.41	35.69	12	2.138	1.49
Grad-Row-Colum	12.04	0.37	0.84	7.21	16.98	6.64	14.03	7.84	10.58	6.78	9.02	12.90	29.42	11.85	25.04	13	2.168	1.52
Row-Colum-Permute	11.87	0.35	1.24	7.24	16.85	6.65	13.89	7.88	10.57	6.83	8.95	12.96	31.03	11.91	26.46	16	1.743	2.46
OPF 3754	8.39	0.37	1.45	4.86	19.37	4.43	16.17	6.95	18.86	5.74	9.24	8.57	32.13	7.87	27.12	15	1.573	1.93
Row-Premute	11.27	0.35	1.37	4.65	14.58	4.24	12.39	6.46	13.21	5.36	7.95	8.32	24.16	7.56	20.96	14	1.516	1.93
Grad-Row-Permute	10.47	0.37	0.84	4.80	17.27	4.38	14.38	6.87	17.63	5.58	8.91	8.47	26.02	7.77	24.62	13	1.724	1.43
Grad-Row-Colum	11.40	0.82	0.84	4.58	13.51	4.24	11.50	6.34	12.64	5.36	7.85	8.22	24.32	7.57	21.01	14	1.726	1.48
Row-Colum-Permute	11.27	0.36	1.34	4.65	14.54	4.24	12.35	6.52	12.84	5.36	8.01	8.34	23.92	7.62	21.39	14	1.458	1.98
c-47	8.36	0.38	1.31	6.01	19.25	5.32	13.79	7.78	14.20	6.21	9.30	10.87	32.47	9.75	25.62	16	1.380	2.68
Row-Premute	10.06	0.37	1.27	5.90	16.64	5.22	11.76	7.59	13.60	6.08	8.46	10.66	27.23	9.56	22.03	15	1.350	2.57
Grad-Row-Permute	10.22	0.41	0.90	5.87	15.30	5.23	11.01	7.61	0.00	6.09	7.21	10.58	25.57	9.68	18.52	16	1.873	1.78
Grad-Row-Colum	11.23	0.37	0.89	5.69	11.34	5.11	8.91	7.27	0.00	5.83	6.07	10.39	18.55	9.36	15.29	15	1.870	1.73
Row-Colum-Permute	10.06	0.61	1.30	5.90	16.69	5.24	11.69	7.61	13.72	6.06	8.51	10.69	27.48	9.59	21.85	15	1.358	2.60
mhd4800a	7.13	0.43	1.28	3.38	12.58	3.10	11.91	5.28	16.62	4.70	13.87	6.03	18.42	5.79	18.10	9	1.410	1.31
Row-Premute	10.96	0.38	1.24	3.35	12.38	3.02	10.67	5.16	14.84	4.55	10.83	6.06	20.50	5.56	17.21	9	1.400	1.31
Grad-Row-Permute	9.52	0.89	1.21	3.38	12.54	3.09	11.66	5.27	16.58	4.66	13.38	6.05	20.79	5.72	19.34	9	1.417	1.29
Grad-Row-Colum	10.94	0.43	1.20	3.36	12.31	3.03	10.43	5.18	14.94	4.54	10.49	6.05	20.44	5.60	18.68	11	1.416	1.38
Row-Colum-Permute	10.96	0.81	1.24	3.34	12.31	3.02	10.67	5.19	14.74	4.45	10.90	6.00	17.94	5.58	17.03	11	1.409	1.39
gen4	9.23	0.42	1.35	3.58	10.34	3.31	8.85	5.62	13.46	4.99	10.27	6.42	16.73	6.12	14.96	8	1.491	1.28
Row-Premute	10.25	0.71	1.39	3.58	10.42	3.30	8.86	5.54	13.34	4.94	10.36	6.44	15.38	6.08	14.59	11	1.457	1.43
Grad-Row-Permute	9.94	0.38	1.33	3.58	11.30	3.30	9.72	5.59	15.00	4.98	10.88	6.41	17.09	6.09	14.75	8	1.518	1.26
Grad-Row-Colum	10.86	0.61	1.32	3.56	11.20	3.26	9.17	5.52	14.61	4.85	9.91	6.59	15.20	6.05	14.45	8	1.529	1.25
Row-Colum-Permute	10.25	0.90	1.40	3.58	10.51	3.30	9.04	5.58	13.20	4.99	10.42	6.43	17.44	6.11	14.76	12	1.452	1.46
Maragal 6	9.93	0.37	1.18	11.87	27.58	10.59	21.35	11.69	18.06	10.06	14.05	20.29	49.93	18.66	40.32	1	0.800	1.02
Row-Premute	10.78	0.38	1.32	11.63	22.76	10.36	16.68	11.48	15.29	9.78	11.58	20.35	39.88	18.12	30.88	16	0.899	5.48
Grad-Row-Permute	11.28	0.36	1.05	12.02	20.49	10.64	15.58	11.84	13.66	10.12	10.28	20.77	35.04	18.92	27.27	16	2.488	2.25
Grad-Row-Colum	12.04	0.36	1.07	11.71	15.50	10.41	11.92	11.67	10.51	9.91	7.93	20.50	26.67	18.36	20.52	14	2.444	1.97
Row-Colum-Permute	10.78	0.36	1.40	11.64	22.68	10.35	16.66	11.46	15.29	9.73	11.60	20.46	39.00	18.53	30.84	16	0.882	5.36
aft01	7.81	0.37	1.57	4.06	15.54	3.69	14.40	5.65	20.64	4.76	11.65	7.04	24.77	6.55	24.10	11	1.830	1.29
Row-Premute	11.16	0.37	1.45	3.91	13.87	3.52	11.67	5.24	13.99	4.42	10.17	7.00	22.32	6.43	19.61	11	1.775	1.31
Grad-Row-Permute	10.26	0.37	1.37	4.03	15.04	3.65	13.67	5.54	18.23	4.68	12.11	7.10	21.64	6.56	24.48	11	1.774	1.30
Grad-Row-Colum	11.12	0.39	1.36	3.92	13.75	3.52	11.86	5.22	14.12	4.43	9.97	6.96	21.13	6.47	21.17	10	1.782	1.25
Row-Colum-Permute	11.17	0.46	1.44	3.92	13.81	3.51	11.89	5.25	13.99	4.44	9.78	7.02	23.65	6.44	19.65	11	1.753	1.34
TSOPF RS b39 c7	7.30	0.39	1.61	6.76	27.48	6.00	21.32	8.58	15.39	7.12	10.90	12.35	48.58	11.32	38.27	12	1.755	2.41
Row-Premute	10.54	0.37	1.48	6.55	21.44	5.86	16.85	8.31	13.54	6.91	9.94	12.35	38.92	11.18	30.78	14	1.658	2.20
Grad-Row-Permute	9.64	0.37	1.32	6.74	26.49	5.99	19.67	8.53	15.00	7.04	10.85	12.24	47.38	11.28	36.58	9	2.015	1.42
Grad-Row-Colum	11.22	0.37	1.28	6.50	20.59	5.82	16.35	8.22	13.10	6.90	9.67	11.98	36.17	11.01	29.17	12	2.051	1.65
Row-Colum-Permute	10.54	0.47	1.49	6.55	21.65	5.86	16.81	8.29	13.48	6.93	9.94	12.18	36.43	11.19	30.92	12	1.651	2.03