Sparse Matrix-Vector Multiplication for Circuit Simulation

CSE260 (2012 Fall) Course Project Report
Hao Zhuang *, and Jin Wang **
Computer Science and Engineering Department
University of California, San Diego
Email: *hao.zhuang@cs.ucsd.edu, **jiw112@cs.ucsd.edu

Abstract

Sparse Matrix-Vector Multiplication (SpMV) plays an important role in numerical algorithm in circuit simulation. In this report, we utilize Message Passing Interface (MPI) to parallelize the SpMV. In addition, resulting from the circuit simulation matrix formulation, the circuit systems are often represented as unstructured, not evenly-distributed sparse matrices. Therefore, we automatically detect rows' number of non-zero elements and balance the computing workload to distribute tasks among different processes. We observed large speedups to compare with serial SpMV and our first evenly-row partition MPI-SpMV(Algorithm 1).We also design MPI_Reduce method, however, we still use point-to-point method due to the performance based our implementation.

1. Introduction

The sparse matrix-vector multiplication (SpMV) is a very important operation among iterative solvers for linear systems of equations, circuit simulation, eigen-solvers, and PageRank computation for ranking web pages [17]. In circuit simulation, this kind of operation is crucial for time domain iterative simulation, which is involved with solving ordinary differential equations (ODE). In addition, for most advanced ordinary differential equations solving methods, such as the matrix exponential method [3, 4, 7, 8], which are very promising in circuit simulation, requires frequent SpMV usage.

The SpMV is as follow,

$$x = Av$$

where, x is the resulting vector by multiplication with sparse matrix A and vector v.

In this project, our goal is to achieve acceleration in SpMV for circuit simulation via MPI. However, the modified nodal analysis (MNA [13, 14])-based circuit simulator [3, 4, 15] brings much non-ideal characteristic for numerical solver of dynamical system, such as the different component related via current parts and voltage parts, the grounded component nodes. Therefore, the matrices from circuit simulation is not always well and evenly-distributed or symmetry than the methods such as Finite Element methods [18]. In

section 2, we take the workload balancing into consideration, which involves the task partition based on number of non-zeros (*nnz*) in different processes. In section 3, the experiment, we deal with several matrices from circuit simulation at University of Florida (UF) matrix Collection [1] and make our discussion and analysis. Section 4 is our conclusion.

2. Methodology and Implementation

First, we implement the serial SpMV, and parallelize it to different processes with MPI. Then we distribute workload more evenly with the almost same nonzero elements among each other, to different processes.

2.1. Serial version of Sparse Matrix Vector Multiplication

We use CSR data [6] Structure:

Data Structure: SpMV 's Compressed Sparse Row

Sparse matrix storage:

ptr[*i*]: stores the starting position of indices of *i-th* row element, the $i+1$ element stores the number of total non-zeros elements

ind[*j*]: stores the *j-th* element's corresponding column index

val[*j*]: stores the *j-th* nonzero element's value.

Vector storage:

v[*i*]: the vector of sparse matrix vector multiplication

r[*i*]: the result vector of sparse matrix vector multiplication

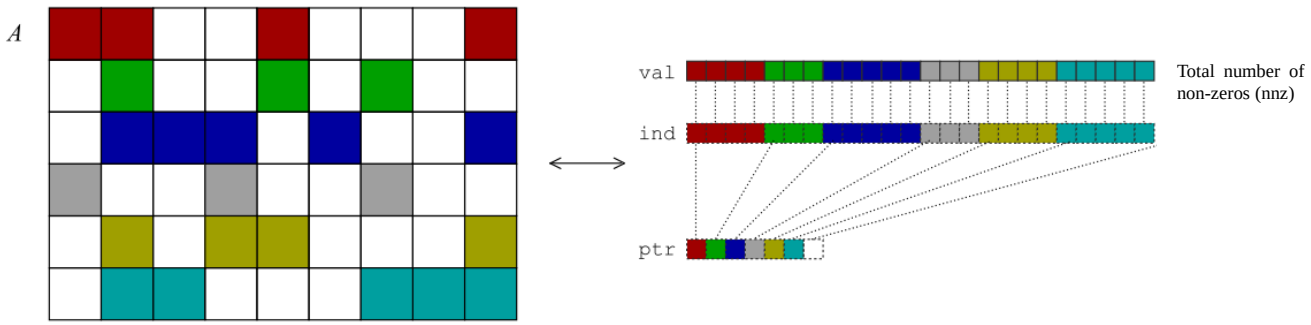

Fig 1.1 CSR data structure for Sparse Matrix A, figure from [18]

The serial version of SpMV is (A is represented in CSR format).

Algorithm 1 x = CSR-based Serial SpMV(A,v)

```
1. for i = 1: n
2.     r[i] = 0;
3.     for k = ptr[i] : ptr[i+1] - 1
4.         col = ind[k];
5.         r[i] += val[k]*v[col];
6.     end for
7. end for
```

2.2. Parallelization via MPI

Following the sparse matrix compression method, eventually three key vectors in contiguous memory locations as follows are generated: the upper ones are for floating number (*val*) and the column index of each corresponding (*val*) value. Here they have the same capacity of *nnz_size*;

val	10.0	-2.0	3.0	9.0	3.0	…	4.0	2.0	-1.0
ind	1	5	1	2	6	…	2	5	6

The bottom one (*ptr* that points to the row) is to store the locations in the *val* vector that starts a row. By convention, we define $ptr(n+1) = nnz\_size + 1$;

ptr	1	3	6	9	13	17	20

In our methodology, the node with *myrank*, which is equivalent to zero, is the root node. The root node has four obligations. (1) To package the buffer need to be transferred to slave nodes with three vectors generating by matrix compression; (2) To dispatch the buffer to each slave node; (3) To compute the rest data in the *val* vector if the amount of *val* cannot be evenly placed on each slave node; (4) To assemble the sub-result from slave nodes.

The whole methodology is detailed described as follows:

1. To dispatch the buffer to each slave node:

   Slave nodes still being called co-working nodes in our project have the size *working_size* of the amount of *process* - 1. We divide *"ptr"* vector by *working_size* evenly to get sub-computing tasks. Each computing task contains the sub-vector "*val*" by calculating the corresponding index range. If it cannot be evenly divided, the root node is in charge of the rest *"val"* multiplication.

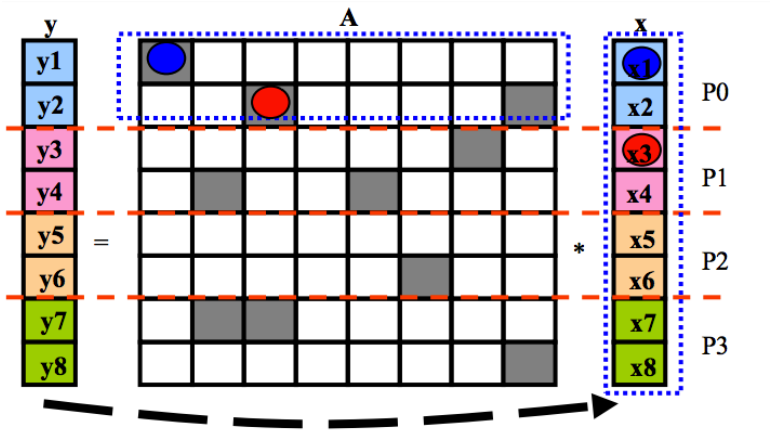

Fig. 2.1. The parallel sparse matrix vector multiplication from [2]

2. To package the passing buffer:

   In order to improve the message sending efficiently, we just send the sub parts of array ind, val, v and their related row numbers, rather than the entire arrays.

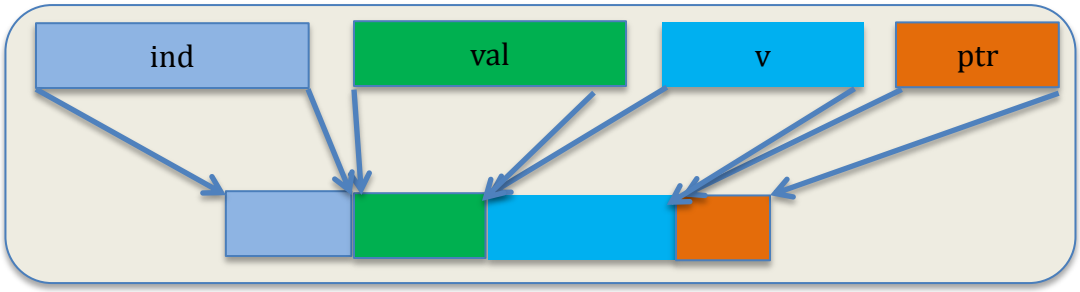

Fig. 2.2. Buffer Format

3. To compute the rest data in the vector *"val"*

   If there are some rest arrays which are not assigned to co-working processes, the root note will perform the multiplication of this part.

4. To assemble the sub result

   In the iteration process, we unpacked the buffer and assign each sub-computing result into the target position that corresponds with row number *I*, both of which were extracted from that received buffer.

The node with the rank from 1 to N works on the sub-computing task separately. One work is to do vector computation as the same as step 3 in the previous (3), and the other one is to package the result and send it to the root.

Algorithm 2 (SpMV-MPI-I): Evenly-row partition for MPI parallelism

In this version of parallel SpMV, the *rows* (*n*) are divided almost evenly by the number of processes (*nprocs*), resulting to $\sim r_{procs}$. Different processes have their own ranks number (*myrank*). The root rank is 0. In Fig.2.1 where in the data within blue box need to be transferred. The T_{myrank} means the process's corresponding part in matrix or

vector T.

Algorithm 2 x = SpMV-MPI-I(A,v)

1. $r_{procs} = n/nproc$
2. if (*myrank* equals 0)
3. for i = 1: *nproc*-1
4. Put the corresponding arrays including (from row i*r_{procs} to (i+1)*r_{procs}-1), indices, value, and whole vector into ***buffer*[] //** based on Fig. 2.2
5. *MPI_Send*(***buffer***) //
6. end for
7. x_{myrank} = CSR-based Serial SpMV(A_{myrank},v_{myrank}) //Algorithm 1
8. else //non-root process
9. *MPI_Recv*(**buffer**) and Get A_{myrank},v_{myrank} from ***buffer***
10. x_{myrank} = CSR-based Serial SpMV(A_{myrank},v_{myrank}) // Algorithm 1
11. *MPI_Send*(x_{myrank}) // including global vector stamping position
12. endif
13. for i = 1: *nproc*-1
14. *MPI_Recv*(x_{myrank}); // Here is first come first server, by using MPI_ANY_SOURCE
15. Assemble the SpMV result vector x_{myrank}, put it to right position based on extra buffer information forming the x
16. end for

We should design the maximum buffers to contain the largest number of non-zeros element resulting from the evenly-row partition. It wastes space and transfer time. Later, we figure out that the nonzero balance is the key characteristic to control work balance.

During this distributed version algorithm, we found the unbalanced workload will harm the efficiency since the unevenly distributed data of circuit simulation. The next step is to adaptively tune the system for the workload balance of each task within process.

Algorithm 3 (SpMV-MPI-II): Evenly-NNZ partition for MPI parallelism

Workload balance based on the number of non-zeros. The distribution of nonzero in sparse matrices from circuit simulation is not often symmetrical, especially in modified nodal analysis. We treat the problem as the irregular decomposition problem. Our approach is very practical for circuit simulation If Divide by block. The vector can be send as a piece, especially for the circuit simulation sparse matrix based on MNA [14].

Algorithm 3 x = SpMV-MPI-II(A,v)

1. $nnz_{proc} = nnz$ / *npocs* // each process should has nnz_{proc} non-zero elements
2. for i = 1 : n
3. $cut[i] = find(row, row+n, i* nnz_{proc})$; // search for the cut place based on $i*nnz_{proc}$

4. end
5. if (*myrank* equals 0)
6. for i = 1: nproc-1
7. Put the corresponding arrays including rows (from row *cut*[i] to *cut*[i+1]-1), indices, value, and whole vector into ***buffer*[]**
8. *MPI_Send*(***buffer***)
9. endfor
10. x_{myrank} = CSR-based Serial SpMV(A_{myrank},v_{myrank}) //Algorithm 1
11. else // not-root process
12. *MPI_Recv*(***buffer***) and Get A_{myrank},v_{myrank} from ***buffer***
13. x_{myrank} = CSR-based Serial SpMV(A_{myrank},v_{myrank}) // Algorithm 1
14. *MPI_Send*(x_{myrank}) // including global vector stamping position
15. end if
16. for I = 1: *nproc*-1
17. *MPI_Recv*(x_{myrank}); // Here is first come first server, by using MPI_ANY_SOURCE
18. Assemble the SpMV result vector x_{myrank}, put it to right position based on extra buffer information forming the x forming the x
19. end for

Algorithm 4 (SpMV-MPI-III): Reduce Strategy for collective communication of MPI parallelism

We think the communication on one process will have ripple structure (Fig). We plug MPI_ANY_SOURCE in our *MPI_Recv* API, which means the program deals with as soon as possible the incoming data, not based on the process ID, but the reaching timing.

Algorithm 4 x = SpMV-MPI-III(A,v)

1. nnz_{proc} = *nnz* / *npocs* // each process should has nnz_{proc} non-zero elements
2. for i = 1 : n
3. *cut*[i] = *find*(*row*, *row*+*n*, i* nnz_{proc}); // search for the cut place based on i*nnz_{proc}
4. end
5. if (*myrank* equals 0)
6. for i = 1: *nproc*-1
7. Put the corresponding arrays including rows (from row *cut*[i] to cut[i+1]-1), indices, value, and whole vector into ***buffer*[]**
8. *MPI_Send*(***buffer***)
9. end for
10. x_{myrank} = CSR-based Serial SpMV(A_{myrank},v_{myrank}) //Algorithm 1
11. else // calculation in root own process
12. *MPI_Recv*(***buffer***) and Get A_{myrank},v_{myrank} from ***buffer***
13. x_{myrank} = CSR-based Serial SpMV(A_{myrank},v_{myrank}) // Algorithm 1
14. x_{myrank} =*FillinWithDummyZero*(x_{myrank})
// to form the large vector for *MPI_reduce* to gather different part of vector

15. end if
16. *MPI_Reduce*/*Allreduce*(x, x_{myrank}); // Assemble the vector x_{myrank}, forming the x

Here is the Ripple message receiving strategy:

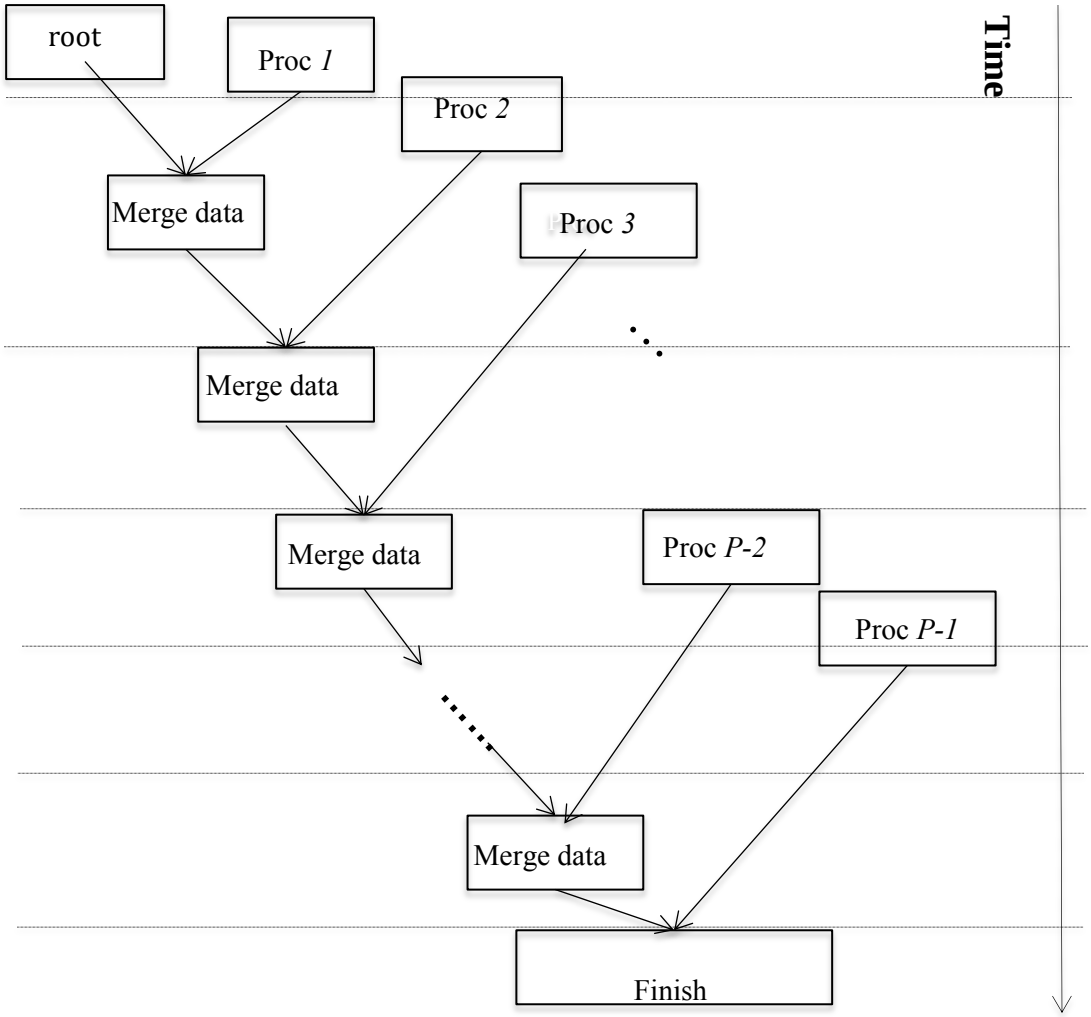

Fig.2.4 Ripple message receive strategy (ordering based on the reaching timing to root)

And utilizing the reduce techniques, which has communication structure like a binary tree (bottom-up). However, our experiment shows that, the *MPI_SUM*, need to deal with global vector. It causes the large data transmission with dummy padding array elements than the ripple structure.

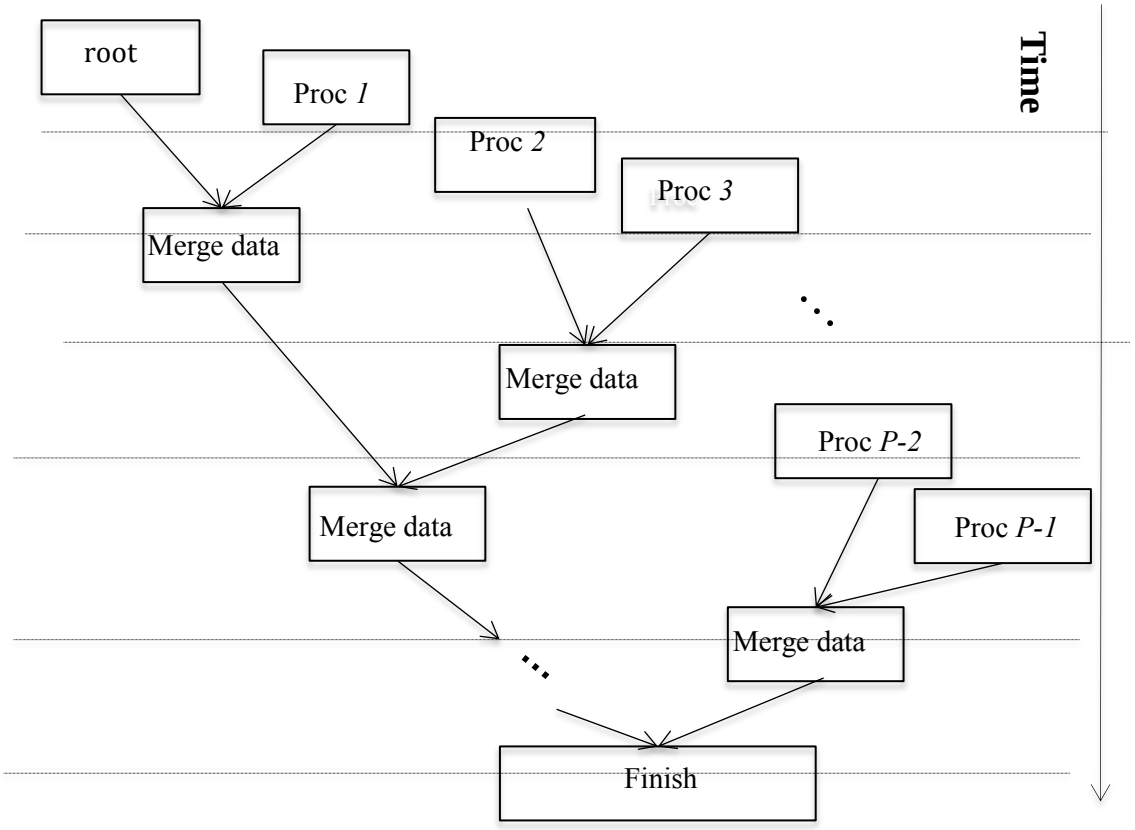

Fig. 2.5. Binary Tree Collective Communication Reduce Model

We also used MPI_Gather, to avoid the redundant dummy space and operation of MPI_SUM in MPI_Reduce. However, the effect is not good compared to the ripple structure.

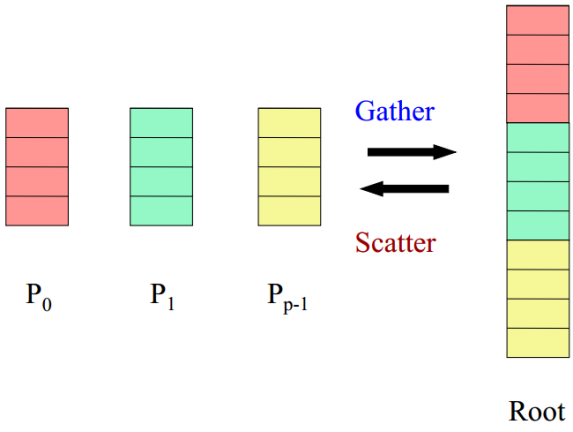

Fig. 2.6. Gather and Scatter from [9]

3. Experiment, Analysis and Discussions

The experiment is carried on UCSD Bang computer servers. We use Circuit Matrices from UFGet [1]. For sparse matrix cases, we the maximum time period calculated via MPI_Wtime API. The matrices benchmarks in UF Sparse Matrix Collection are plotted as following figures:

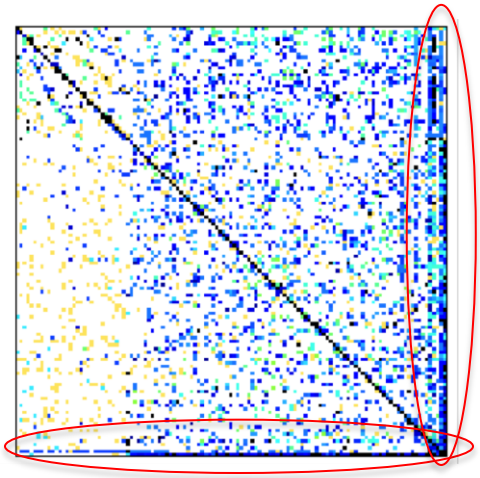

Fig. 3.1. Case A: Sandia's circuit adder_dcop_01 has 11156 nonzero elements. Matrix size 1183*1183 (Red box shows dense zone in sparse matrix) From UF Sparse Matrix Collection [1]

The plot shows the unevenly row partition strategy, for evenly row partition,, which is caused by the relative dense (nonzero number) bottom lines.

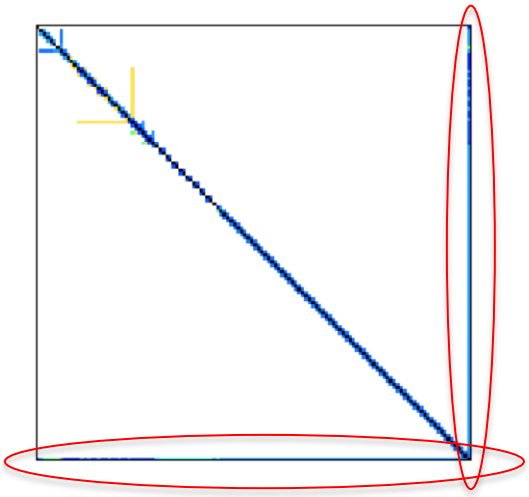

Fig. 3.2. Case B: Bomhof circuit 4 has 307604 nonzero elements. Matrix size is 80209x80209 (Red box shows dense zone in sparse matrix), From UF Sparse Matrix Collection [1]

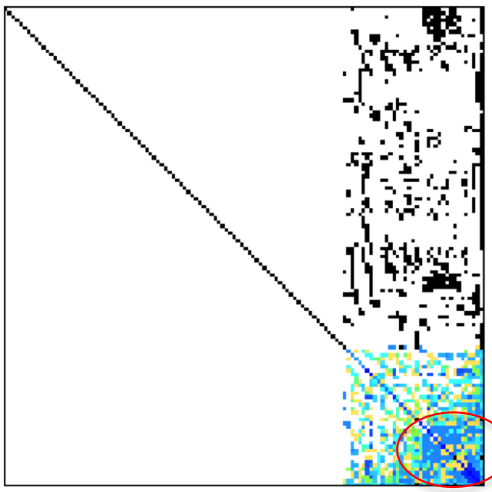

Fig. 3.3. Case C: Bomhof circuit_2 has 21199 nonzero elements. Matrix size is 4510*4510 (Red box shows more dense zone in sparse matrix) from UF Sparse Matrix Collection [1]

Take 8 processes to see the non-zero element distributed in different processes of Case A, B, and C.

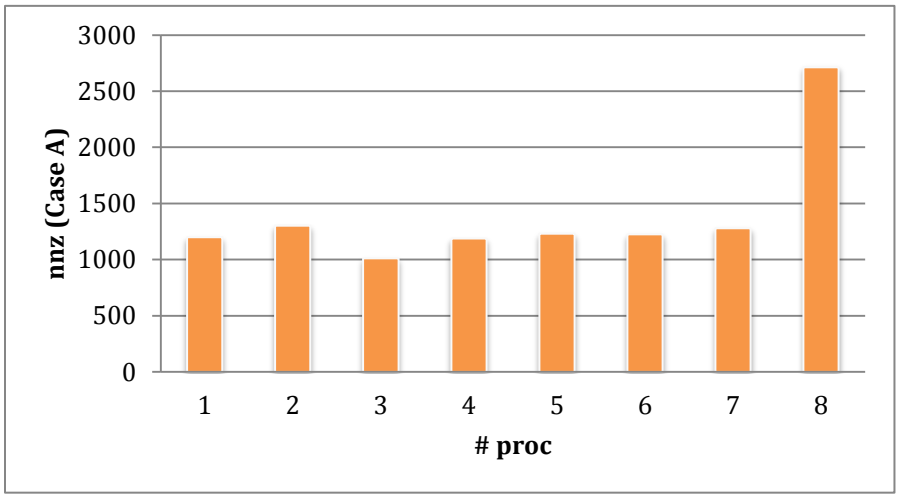

Fig 3. 4 Non-zero element distributions in 8 processes (Case A)

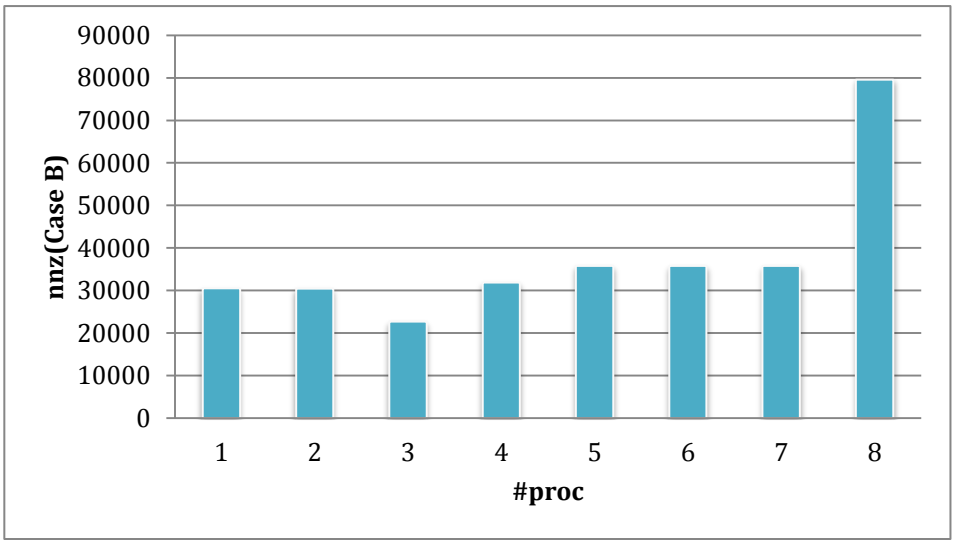

Fig 3. 5 Non-zero element distributions in 8 processes (Case B)

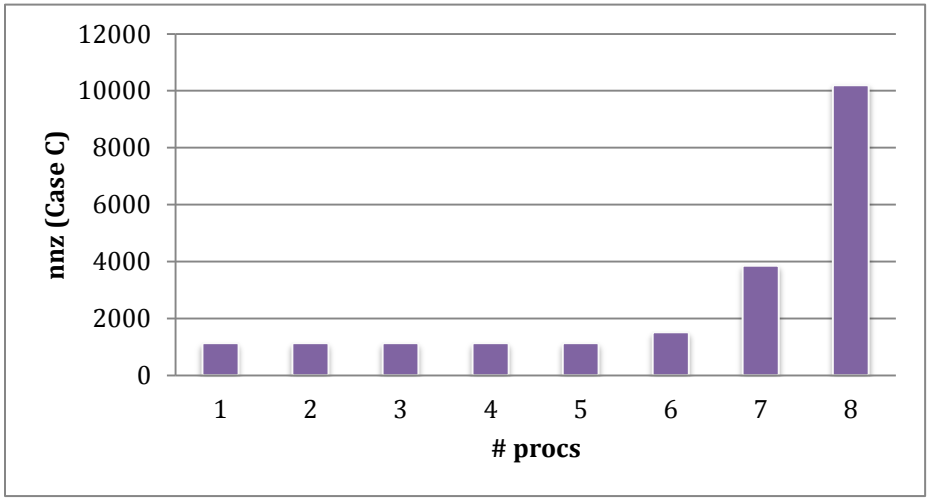

Fig 3.6. Non-zero element distributions in 8 processes (Case C)

We see the huge different between the last node and previous nodes. It is because, in circuit simulation, the last lines usually have coupling information from current branch and voltage nodes [14]. First, we should design the maximum buffers to contain the largest number of non-zero elements resulting from the evenly-row partition. It wastes space and transfer time. We see the very bad work balance (Fig 3.4~3.6). That is the reason the parallelism efficiency is harmed (obvious phenomena is in Table 3.2 and Fig 3.8). Due to the un-symmetry property from circuit simulation, we partition workload based on the number of non-zero elements within continuous combination of rows. We also design our metric to compare the algorithms without Gflops information. The "*normalized efficiency*", which is using the largest MPI runtimes obtain from our all implementations and experiment in one case respectively, and use it as the coefficients to multiply the inverse of all runtimes in the experiment of one certain SpMV case with 50k iterations.

Table 3.1 Runtime v.s. Process Number (Case A)

Case A(procs)	2	4	8	16
Algorithm 2	1.32	0.68	0.42	1.06
Algorithm 3	1.55	0.82	0.43	0.86
Algorithm 4	1.56	0.81	0.41	1.37

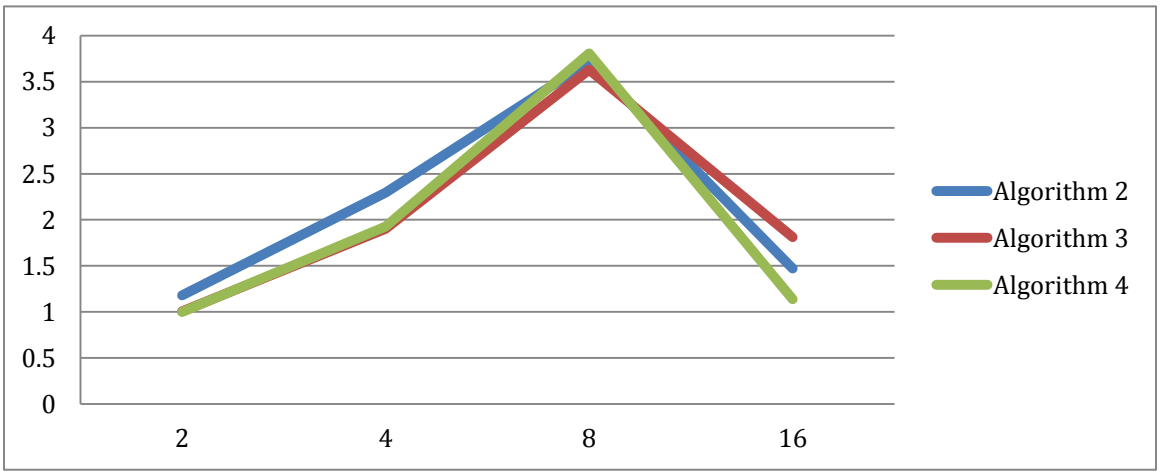

Fig 3. 7 Normalized efficiency of different algorithms in Case A

Table 3.2 Runtime v.s. Process Number (Case B)

Case B (procs)	2	4	8	16	32	64
Algorithm 2	120.14	29.89	22.41	11.88	10.52	13.59
Algorithm 3	44.79	21.93	11.06	5.63	4.09	7.88
Algorithm 4	66.09	35.34	24.21	6.04	4.38	7.90

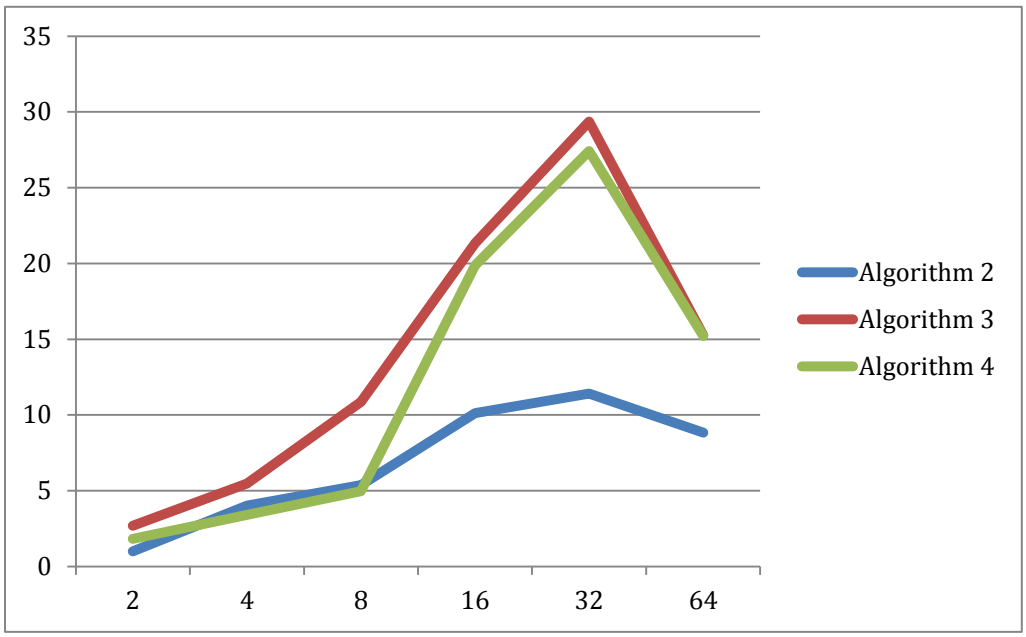

Fig 3.8 Normalized efficiency of different algorithms in Case B

Table 3.3 Runtime v.s. Process Number (Case C)

Case C (procs)	2	4	8	16
Algorithm 2	3.15	1.89	1.88	1.30
Algorithm 3	2.24	1.30	0.65	1.17
Algorithm 4	2.20	1.27	0.66	1.20

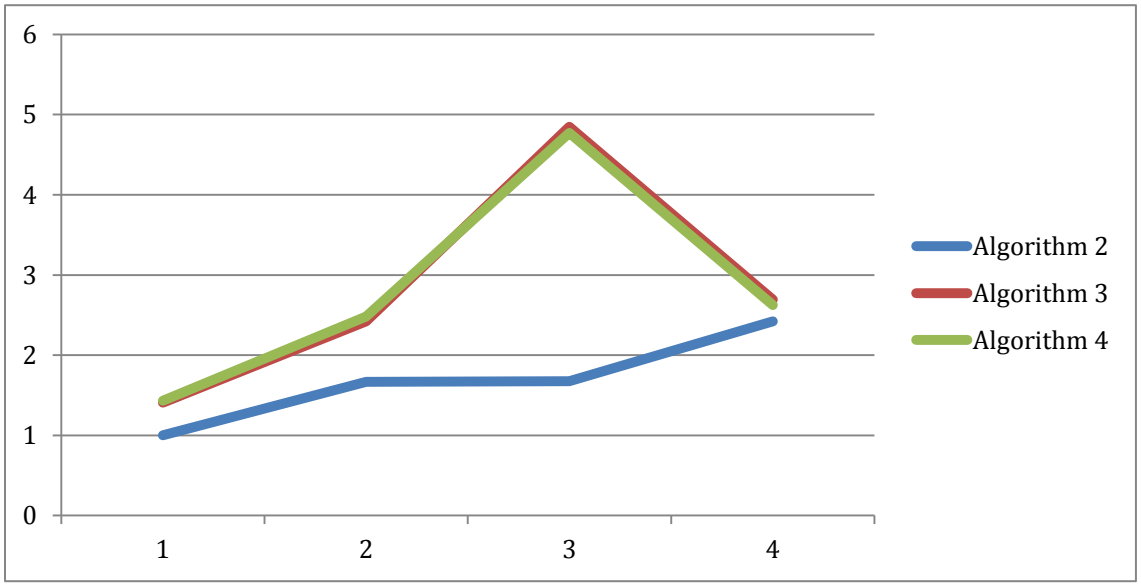

Fig 3.9 Normalized efficiency of different algorithms in Case C

3.1.Efficiency among Algorithm 2, 3, 4

For small case, the buffer wrap-ups will play a relative large portion, which we think it is locality issue, so that the Algorithm 2 has more advantage in here, and the transmission penalty (bigger data transmission compared to other two methods) is compensated (Fig 3.7).

For the large case, such as Case B, the advantage of Algorithm 3 and 4 come out. It can be further accelerated than other two cases, still maintaining the trend till around 32 processes. We also see the algorithm 2 and algorithm 3, 4 approach to each other with the process number increasing, because the matrix is divided into small portion and the non-zeros elements in each process comes to be almost the same or the locality play more importation portion in SpMV.

3.2. Assembling strategy, Ripple or Reduce?

Truly, it is observed that during experiment, the *MPI_Reduce* model smooth the divergence of different process runtime, but the total effect is not better than algorithm 2. Despite reduce model make sense, we should also still take into consideration with the data amount transmission. The dummy elements need to be added for MPI_SUM of forming global resulting vector. So it transfers the whole size of vector, unlike the ripple version design, which only requires the rows per process size of array to feedback. Also ripple has *first come, first assemble* policy, which ease the waiting time to some extent.

It validate our theory about the non-zeros is really the crucial for work load balance. For the small case in A and medium case C, the dummy transmission costs are compensated by the reduce model, however, the large case B, the point to point communication show advantages because it always send small portion of result vector to the root node. We choose the ripple strategy (Fig. 2.2) and submit in our program with the point-to-point communication, which is shown in Algorithm 3.

3.3.Speedups compared to serial version

It achieves large speedups compared to serial version, the large speedups also result from our communication cost in iteration is very low when iteration is large.

- In Case A, it costs 6.69s (Serial SpMV) and 0.43s (Algorithm 2 with 8 processes), we obtain 16X speedups.
- In Case B, it costs 202.90s (Serial SpMV) and 4.09s (Algorithm 2 with 32 processes), we obtain 50X speedups.
- In Case C, it costs 11.64 (Serial SpMV) and 0.65s (Algorithm 2 with 8 processes) we obtain 18X speedups.

4. Conclusion and Future Work

When dealing with sparse matrix vector multiplication in circuit simulation, matrix stamping method in methods like modified nodal analysis, it often results in non-symmetry distribution of matrix element, also with relative dense distribution in bottom rows and right most columns. In this project, we parallelize and accelerate the sparse matrix from circuit simulation. By utilizing the non-zeros numbers at each row, divide them almost evenly can achieve large speedups (Algorithm 3, 4). The point to point communication seems to be better than reduce one, because it doesn't require dummy array to match global result vector.

Through this project, we gain much sense on MPI programming, but still curious about the message passing penalty with the increasing problem size. It is going to benefit our future distributed circuit simulation research. Future works should be dealt with more general and large cases and it may also involve the block and tiling methods on SpMV. Also, this interesting subject needs taking consider much larger cases. The communication-avoiding strategy as well as tradeoff of bookkeeping and vector transfer. Can we employ MPI_Reduce or similar model hierarchically? They are still open questions to us.

References

[1]. UFget and UFgui interfaces to the UF Sparse Matrix Collection, http://www.cise.ufl.edu/research/sparse/mat/UFget.html

[2] S. Lee, and R. Eigenmann. "Adaptive runtime tuning of parallel sparse matrix-vector multiplication on distributed memory systems."Proceedings of the 22nd annual international conference on Supercomputing. ACM, 2008.

[3] S.-H. Weng, Q. Chen, and C.-K. Cheng. "Time-Domain Analysis of Large-Scale Circuits by Matrix Exponential Method With Adaptive Control." Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 31.8 (2012): 1180-1193.

[4] S.-H. Weng, Q. C., N. Wong and C.-K. Cheng, " Circuit Simulation using Matrix Exponential Method for Stiffness Handling and Parallel Processing", in ACM/IEEE Int. Conf. on Computer-Aided Design 2012, to appear.

[5] S. Baden, CSE 260 Lecture notes, CSE Dept., University of California, San Diego, http://cseweb.ucsd.edu/classes/fa12/cse260-b/Lectures/LoadBalancing.html

[6] Y. Saad. Iterative methods for sparse linear systems. Vol. 620. Boston: PWS publishing company, 1996.

[7] Q. Chen, S.-H. Weng, and C.-K. Cheng. "A Practical Regularization Technique for Modified Nodal Analysis in Large-Scale Time-Domain Circuit Simulation."

Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 31.7 (2012): 1031-1040.

[8] M. Hochbruck, C. Lubich, and H. Selhofer. "Exponential integrators for large systems of differential equations." SIAM Journal on Scientific Computing 19.5 (1998): 1552-1574.

[9] S. Baden, CSE 260 Lecture notes, CSE Dept., University of California, San Diego, http://cseweb.ucsd.edu/classes/fa12/cse260-b/Lectures/Lec14.pdf

[10] S. Baden, CSE 260 Lecture notes, CSE Dept., University of California, San Diego, http://cseweb.ucsd.edu/classes/fa12/cse260-b/Lectures/Lec09.pdf

[11] T. Davis, “Direct method for sparse linear systems", Philadelphia, PA: SIAM, 2006

[12] The Message Passing Interface (MPI) standard, http://www.mcs.anl.gov/research/projects/mpi/

[13] G. Hachtel, R. Brayton, and F. Gustavson, "The Sparse Tableau Approach to Network Analysis and Design". IEEE Trans. Circuit Theory, Jan. 1971

[14] C.-W. Ho, A. Ruehli, and P. Brennan. "The modified nodal approach to network analysis." Circuits and Systems, IEEE Transactions on 22.6 (1975): 504-509.

[15] L. Nagel, ``SPICE2: A Computer Program to Simulate Semiconductor Circuits", Technical Report UCB/ERL, 1975.

[16] BeBOP SpMV Benchmark, http://bebop.cs.berkeley.edu/spmvbench/

[17] M. M. Wolf, E. G. Boman, B. A. Hendrickson, ``Optimizing Parallel Sparse Matrix-Vector Multiplication by Corner Partitioning", http://www.sandia.gov/~egboman/papers/PARA08.pdf

[18] S. Williams, L. Oliker, R. Vuduc, J. Shalfa, K. Yelick, and J. Demmel. "Optimization of sparse matrix–vector multiplication on emerging multicore platforms." Parallel Computing 35.3 (2009): 178-194.

Team Self Evaluation Form

If you are working in a team on a project, then submit one copy of this self-evaluation form. The members of each team should discuss how they worked together and what to write for the evaluation.

(1) List the names of your team members:

A: Hao Zhuang

B: Jin Wang

(2) Estimate how much time each team member devoted to this project.

	A(days)	B(days)
meetings	1	1
coding	8	5
writeup	3	2
planning (alone)	5	3
total (including meetings)	17	11

(3) Some Points to answer

(a) What the major responsibilities of each team member were

Hao Zhuang propose the plan for this project, his idea is to do sparse matrix-vector multiplication using distributed system and workload balance methodology. Firstly, He constructs the Compressed Sparse Row data structure for sparse matrix and implements the serial version of sparse matrix-vector multiplication. For parallelism part, he contributes the original MPI model in this project. Then he also implements the parallel model for work balance method based on non-zero element by rows.

Jin Wang implements the MPI version of sparse matrix-vector multiplication; he fully understands the Compressed Sparse Row data structure and makes the efficient distribute d version. On the part how to more fairly distribute the computing task, he proposed seve ral ideas like dynamic programming and so on to inspire Hao Zhuang's idea on current strategy.

(b) Whether or not you met your milestones

Yes, we met our milestones, and to achieve large speedups in SpMV compared to serial version.

(c) The major strengths and weaknesses in how your team worked together

Strength: good communication during the project. After previous homework, we gain much deeper understanding in parallelism algorithm in distributed memory systems and shared memory systems.

Weakness: The analysis of parallelism algorithm in much deeper and solid quantitatively.

(d) The lessons learned from these events

The unbalanced task will slow down the whole performance by the slowest process. We need to smartly partition the task the sparse matrix structure, by doing so, we observed large differences.

(e) Explain any goals that were modified. In particular, discuss any milestones you could not meet along with an explanation.

We didn’t change too much of our milestone.

(f) Anything else that comes to mind.

Time is really short for project, however, this experience would definitely benefit Hao Zhuang’s later circuit simulation research, especially in the aspect from implementation and more parallelism-aware algorithm deduction and desing in the future.