A STUDY OF LOAD IMBALANCE FOR PARALLEL RESERVOIR SIMULATION WITH MULTIPLE PARTITIONING STRATEGIES

A Thesis

by

XUYANG GUO

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2015

Major Subject: Petroleum Engineering

ABSTRACT

High performance computing is an option to increase reservoir simulation efficiency. However, highly scalable and efficient parallel application is not always easy to obtain from case to case. Load imbalance caused by mesh partitioning and message passing through connections between partitions are the main reasons that prevent successful parallel implementation. This thesis introduces several mesh partitioning methods that assign relatively similar loads to processes and minimize connections between partitions to a large scale parallel reservoir simulation model. Their effects on enhancing parallel computing performance are discussed. Specifically, the effects are evaluated based on two parameters: parallel overhead and load imbalance status.

The partitioning methods introduced are 2D decomposition, Metis partition, Zoltan partitioning, and spectral partitioning. In the first place, their implementation in the original reservoir model is researched. Then, they are also applied to the same reservoir model with elevated well complexity. In order to increase well complexity, the original model's well geometry and well control constraints are changed. For each partitioning strategy, various subdomain number s are used. They are 2, 4, 8, 16, and 32. Once the mesh is partitioned, the assignment of each subdomain to process is also studied. The fashion of assigning each subdomain's reservoir model computation to a specific process in the cluster affects parallel overhead. When two neighboring subdomains are assigned to two physically neighboring processes in the cluster, the overhead is much smaller than when they are assigned to two non-neighboring processes. Except for the assignment process, load imbalance are examined as well. In the original reservoir model, since the well geometries and well control patterns are not very complex, low load imbalance is obtained for parallel simulation based on the four partitioning methods introduced. The speedups are scalable. When the well model complexity is elevated by introducing horizontal wells and more frequent well control constraints changes, an increased load imbalance can be observed in the parallel reservoir simulation. Thus, the scalability is undermined. In general, this work allows us to better understand the application of various partitioning strategies in terms of load imbalance and parallel overhead.

ACKNOWLEDGEMENTS

I would like to thank my committee chair, Dr. Killough, and my committee members, Dr. Sarin, and Dr. Gildin, for their support and guidance through my study in the graduate school.

I would like to thank my research group members for their help and support in school and in my life. I would also like to thank faculty members and staff in Harold Vance Department of Petroleum Engineering for making the department such a great place.

I would like to thank my friends for their help and for all the good times.

Finally, thanks to my family for their support and love throughout my life.

NOMENCLATURE

S_p	Speedup for a parallel job with p processes, dimensionless
E_p	Efficiency for a parallel job with p processes, dimensionless
T_{Serial}	Serial job simulation time, second
$T_{Parallel}$	Parallel job simulation time, second
CPU	Central processing unit
p	Number of processes, dimensionless
PVT	Pressure-volume-temperature
s	Serial fraction of a code, dimensionless

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

1.1 Problem statement

High scalability and efficiency are the targets of high performance computing. But the existence of overheads and load imbalance prevents us from getting optimum parallel implementation. If overheads and load imbalance are not addressed properly, it will be very hard to obtain high speedups and a high cost of high performance computing will ensue.

A critical step to relieve load imbalance is to select an appropriate partitioning strategy. There are many mesh partitioning methods and graph partitioning methods to our knowledge. Their purposes are to assign relatively similar amount of work to each subdomain and minimize the communications between subdomains. However, less efforts have been put into the comparison of several major partitioning strategies. For a specific computation model, an optimum partitioning approach is able to give the parallel implementation low load imbalance and low overhead (Barney 2015).

Another issue is that many of the partitioning strategies are static and they do not take into account additional information other than grid geometries. In petroleum reservoir simulation model, a grid represents the geometry of the reservoir. Static partitioning strategies take such geometries as inputs and generate various kinds of partitioned meshes. This fashion is expected to provide reduced communications between cores and satisfactory load balance status in case of simply well geometries and well operating constraints. However, as complexity of well data increase, inter-core communications and load variance among cores are expected to increase. As a result, parallel efficiency will not be as good as before. This phenomenon needs to be addressed as well so that we can understand load imbalance better.

1.2 Background

Petroleum reservoir simulation has been playing an important role in the oil industry. It allows us to understand reservoir statically and dynamically. Typically, to represent the reservoir adequately, three phases (oil, gas, and water) of the underground fluids are modeled in simulator. Besides, compositional model is also used to represent multiple components that make up reservoir fluids. In many cases, fine grids are used so that reservoir simulation can be more accurate. Large models often have more than 1 million grid blocks. These considerations largely increase reliability and accuracy of reservoir simulation.

However, computation in reservoir simulation can be very time-consuming, especially in large and complex reservoir models. High costs of simulation are not preferred in the field. As a result, speedup based on parallel computing has been studied by many (Reinders 2012). Lu et al. (2008) introduces an implementation of a parallel reservoir simulator on personal computers based on multi-core CPU. Killough and Wheeler (1987) presents the uses of parallel iterative methods for linear equations in simulation processes.

Parallel computing in reservoir simulation subdivides the reservoir simulation mesh into smaller parts and related properties in each part can be computed simultaneously (Maliassov and Shuttleworth 2009). Some problems occur as such parallel mechanisms are implemented. One of them is load imbalance. This is a field that has not been well addressed as others such as parallel solver. Different loads on processes extend the total computation time and reduce the scalability of parallel implementation. Wang and Killough (2014) proposed a strategy based on over-decomposition of reservoir model mesh to reach satisfactory load balance status. The key to load balance is to assign similar loads to processes while communications between partitions are minimum. Several graph partitioning methods can achieve this by reading and understanding reservoir model geometry. Karypis and Kumar (1998a) introduces a graph partitioning method called Metis. It is able to divide mesh into parts with similar grid-block numbers and a relatively small number of connections. Zoltan is another data management method that simplifies load balancing and data communication (Devine et al. 2002). Spectral partitioning is also a candidate that can partition mesh into similar loads. This method requires the adjacency matrix which contains mesh geometry information and bisects mesh (McSherry, 2001). If more than two partitions are wanted, one can simply repeat the bisection to get more partitions.

Some efforts have been made to understand the load imbalance in parallel implementations. Frachtenberg et al. (2003) talk about methods to mitigate load imbalance when parallel computing is applied to very heterogeneous system where processes tend to have very different loads. Oliker and Biswas (1998) show a dynamic way to balance loads among processors for adaptive grid calculations. Their method is also proved to be effective when there is a large number of processors. Tallent et al. (2010) introduce another way to identify the load imbalance in parallel runs. Load imbalance in both dynamic and static forms can be identified using their results. Sarje et al. (2015) addressed issues in load imbalance and data access manners so that they obtained optimized parallel performance in simulations based on meshes.

It is noticed that many efforts have been put into implementation of successful parallel reservoir simulation. However, there is still a need to understand and compare the performance of multiple major partitioning strategies.

1.3 Objectives

Based on previous review, it can be concluded that there is a great possibility of reducing serial run time by parallel computing. More details about load balance are needed so that we can understand the current major partitioning strategies better. The target of this research consists of the following three points.

(1) Speedups and load imbalance of four mesh partitioning strategies will be compared based on their implementation in the original reservoir model. The four strategies are 2D, Metis, Zoltan, and spectral.

(2) The effect of grid array allocation to processes will be studied.

(3) Mesh partition methods’ ability to reduce load imbalance in a more complex reservoir model will be examined.

(4) This study is about two dimensional partitioning and the results obtained here can be a comparison when we later go to three dimensional partitioning.

1.4 Procedures

Several procedures are conducted to fulfill goals proposed in objective. First, based on the original reservoir model geometry, partitioning methods proposed previously are introduced to partition the grid into a number of subdomains. Second, the reservoir grid are subdivided into multiple partitions (2, 4, 8, 16, and 32) for parallel computing. This fashion are repeated for all partitioning strategies. Third, several subdomain-to-process assignment patterns are tested and the resulting overheads are analyzed. Fourth, load imbalance in multiple time steps during the simulation are shown to sketch the load balance change. Fifth, elevated well specifications are added to the existing model and the load balance changes are studied.

CHAPTER II

PROBLEM IDENTIFICATION

2.1 Reservoir model

A large reservoir model grid is used in this study. There are totally 1 million grid blocks in the model. The grid dimension is 100 by 100 by 100. The coordinate system here follows the right-hand rule. I-direction numbering is ascending from westernmost to easternmost. J-direction is ascending from northernmost to southernmost. J-direction is ascending from top to bottom. The reservoir thickness vary with location and it is around 500 feet. Blocks do not have uniform sizes. **Figure** 1 presents the reservoir model with porosity distribution as shown. Only 850,371 cells are active blocks and the rest are not included in the simulation later on. The reservoir around 15000 feet long from north to south and 15000 feet long from east to west. The reservoir structure is an anticline with oil mainly residing at the top of the structure. Six major faults are observed in the model. Four of them are from northwest to southeast and two of them are from northeast to southwest. The faults are displayed in **Figure 2**. In the original model, there are 11 fully penetrated producers. They are all vertical wells. **Table 1** shows the well locations and their type. The primary reservoir drive mechanism is edge water drive. In hydrocarbon bearing area, initial oil saturation is 77.5%. The simulation time is 365 days with the start date as January 1, 2001. Implicit pressure and explicit saturation method is applied to all grid blocks. The computational precision is 1.11×10^{-16}. The simulator takes into account three phases: water, oil, and gas. 8 components are incorporated in the simulator. Properties of the 7 hydrocarbon components are recorded in **Table 2**.

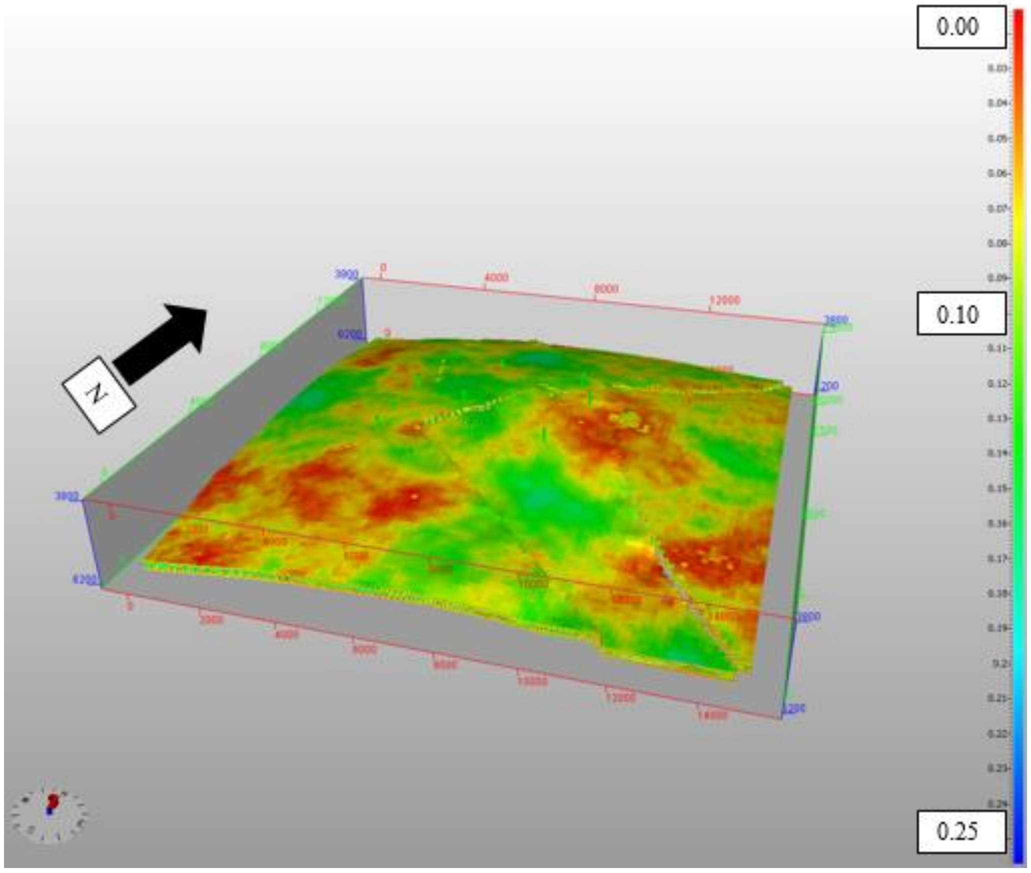


Figure 1. Porosity distribution in the million grid block model

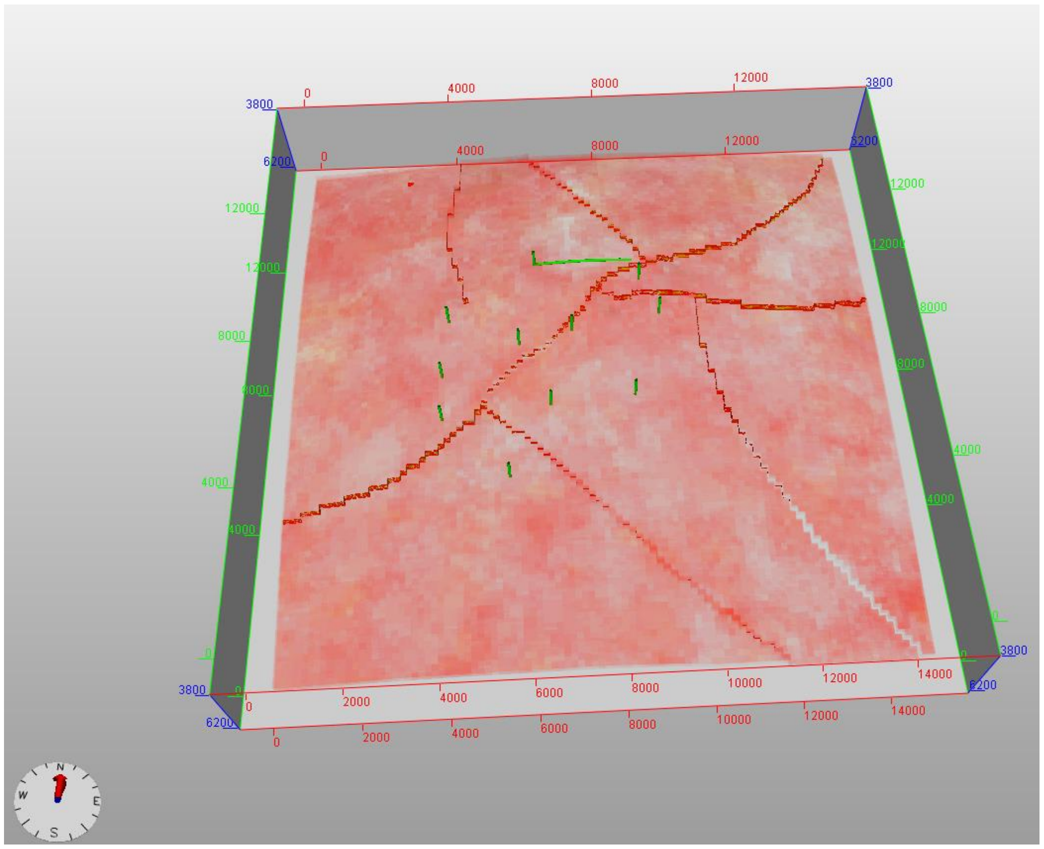


Figure 2. Faults in the million grid block model

Table 1. Well type and locations

	I-durection	J-direction	Type
1NPF0004	37	70	Producer
1BCP0299	58	56	Producer
1FGP0092	27	40	Producer
1WMX0085	26	59	Producer
1DUW0282	60	34	Producer
1JGD0163	26	51	Producer
1BDO0264	48	44	Producer
1NJP0167	63	41	Producer
1WQA0061	39	46	Producer
1YII0191	42	30	Producer
1HTO0146	44	57	Producer

Table 2. Hydrocarbon components properties

COMPONENT	Molecular Weight	Ω_a	Ω_b	Critical Temperature, R	Critical Pressure, psia	Critical Gas Compressibility Factor	Acentric factor	Volume Shift Parameter	Parachor
P1	34.08	0.457236	0.077796	671.76	1296.19	0.28358	0.1	-0.115478	97.3714
P2	44.01	0.457236	0.077796	547.56	1069.87	0.27404	0.225	-0.0943467	125.743
P3	16.0633	0.457236	0.077796	342.87	786.53	0.33879	0.008054	-0.181405	45.8953
P4	53.9334	0.457236	0.077796	732.24	571.601	0.27484	0.180321	-0.0618549	154.095
P5	131.895	0.457236	0.077796	1107.74	336.801	0.25336	0.459627	-0.241669	392.138
P6	263.455	0.457236	0.077796	1390.17	268.456	0.27733	0.748941	-0.125251	752.477
P7	528.436	0.457236	0.077796	1935.43	133.059	0.16754	1.15346	-0.022134	1509.82

2.2 Parallel computing specifications

The simulator is Nexus® version 5000.4.10. The parallel environment of this simulator is based on Message Passing Interface, which enables the work to be simulated on separate cores (Crockett and Devere 2009). This implementation is expected to enhance the performance better than shared memory models (Halliburton 2015). The serial run on the Blackgold cluster took 3258.248 seconds. It is noted that in the serial run, PVT properties calculation took 47.49% of total CPU time as the most time-consuming section.

Table 3 records CPU time and elapsed time of each part of the simulation. Due to licensing limit, up to 32 partitions are available for the cluster to run simultaneously.

Table 3. Serial run time distribution

	CPU Seconds	% Total Run	Elapsed Seconds	% Total Run
INPUT	1.894	0.06	1.893	0.06
OUTPUT	2.266	0.07	2.298	0.07
INITIALIZATION	138.909	4.26	138.828	4.26
PVT PROPERTIES	1547.332	47.49	1546.336	47.48
ROCK PROPERTIES	36.697	1.13	36.733	1.13
EQUATION SETUP	68.287	2.1	68.251	2.1
NETWORK/WELLS	5.845	0.18	5.795	0.18
SOLVER	1123.242	34.47	1122.59	34.47
UPDATE	124.557	3.82	124.594	3.83
MISC.	209.219	6.42	209.26	6.43
Total Run	3258.248	100.00	3256.579	100.00

The Blackgold cluster has 5 nodes. Each node has 32GB RAM. One out of the five nodes has 2 Intel® Xeon® X5672 processors. X5672 has 4 cores and 8 threads. It has 12M cache. Four out of the five nodes have two processors on each node. The processor is Intel® Xeon® E5-2665 with 20 M cache and base frequency of 2.4 GHz. Each E5-2665 processor has 8 cores and 16 threads. Within one node, the parallel architecture is shared memory. Between nodes, the memory architecture is distributed memory. This hybrid of architectures is beneficial for parallel implementation performance enhancement since it does not have strong local overheads on each node.

Table 4. Processor specifications

Processor	Node	Type	Frequency	Cache	Cores	Threads	RAM
1	1	X5672	3.2GHz	12M	4	8	32GB
2	1	X5672	3.2GHz	12M	4	8	32GB
3	2	E5-2665	2.4GHz	20M	8	16	32GB
4	2	E5-2665	2.4GHz	20M	8	16	32GB
5	3	E5-2665	2.4GHz	20M	8	16	32GB
6	3	E5-2665	2.4GHz	20M	8	16	32GB
7	4	E5-2665	2.4GHz	20M	8	16	32GB
8	4	E5-2665	2.4GHz	20M	8	16	32GB
9	5	E5-2665	2.4GHz	20M	8	16	32GB
10	5	E5-2665	2.4GHz	20M	8	16	32GB

CHAPTER III

PARTITIONING STRATEGIES

3.1 Introduction

In order to distribute the work to processes, the reservoir model needs to be divided into subdomains. The decomposition methods discussed here are static, which means that these methods take into consideration the reservoir model mesh before any simulation is actually run. A good partitioning strategy leads to scalable speedup and high parallel computing efficiency. Four methods are introduced here: 2D decomposition, Metis partitioning, spectral partitioning, and Zoltan partitioning. All results presented in this chapter has an orientation of north on the top.

3.2 2D decomposition

2D decomposition is a major decomposition method. It subdivides the mesh into blocks on x-direction and y-direction. In this study, the simulator can utilize this strategy by modifying the grid input file. **Figure 3** shows an example of 2D decomposition in the reservoir model. In this example, the mesh is divided into 16 subdomains. Simulation grid numbers are marked with various colors in the picture. By repeating this pattern, meshes with other partition numbers are also obtained and shown in **Figure 4**, **Figure 5**, **Figure 6**, and **Figure 7.**

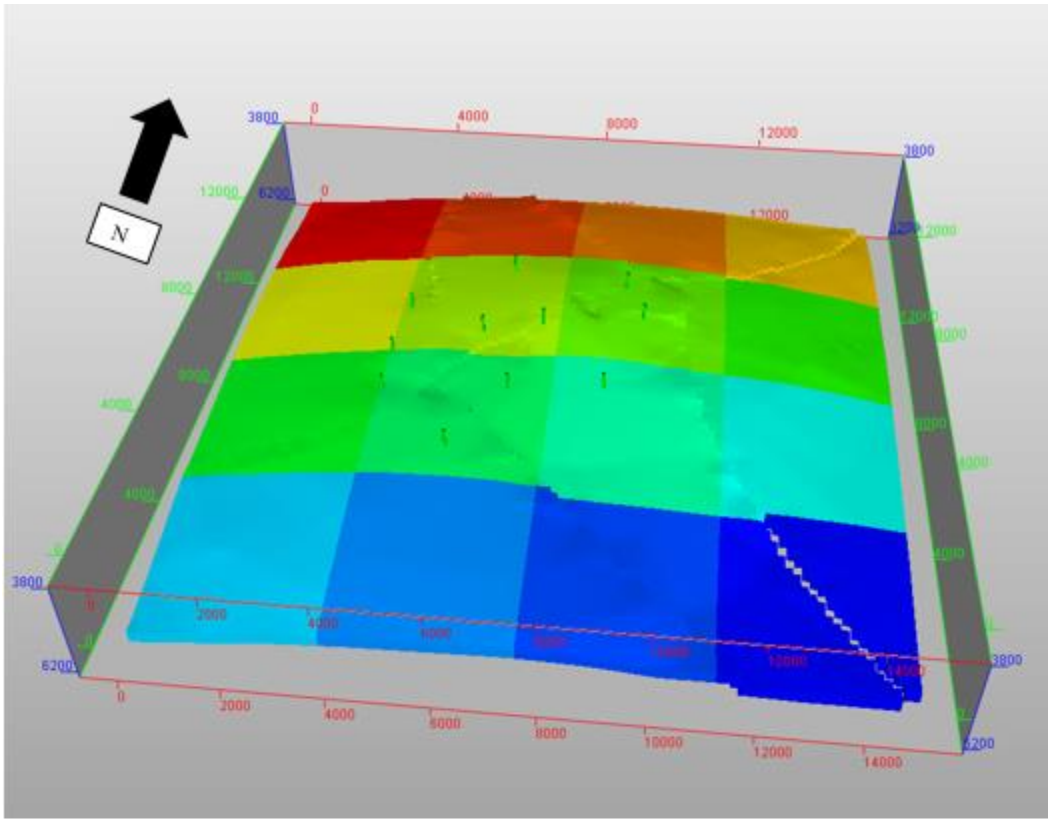


Figure 3. Example of 2D decomposition with 16 partitions

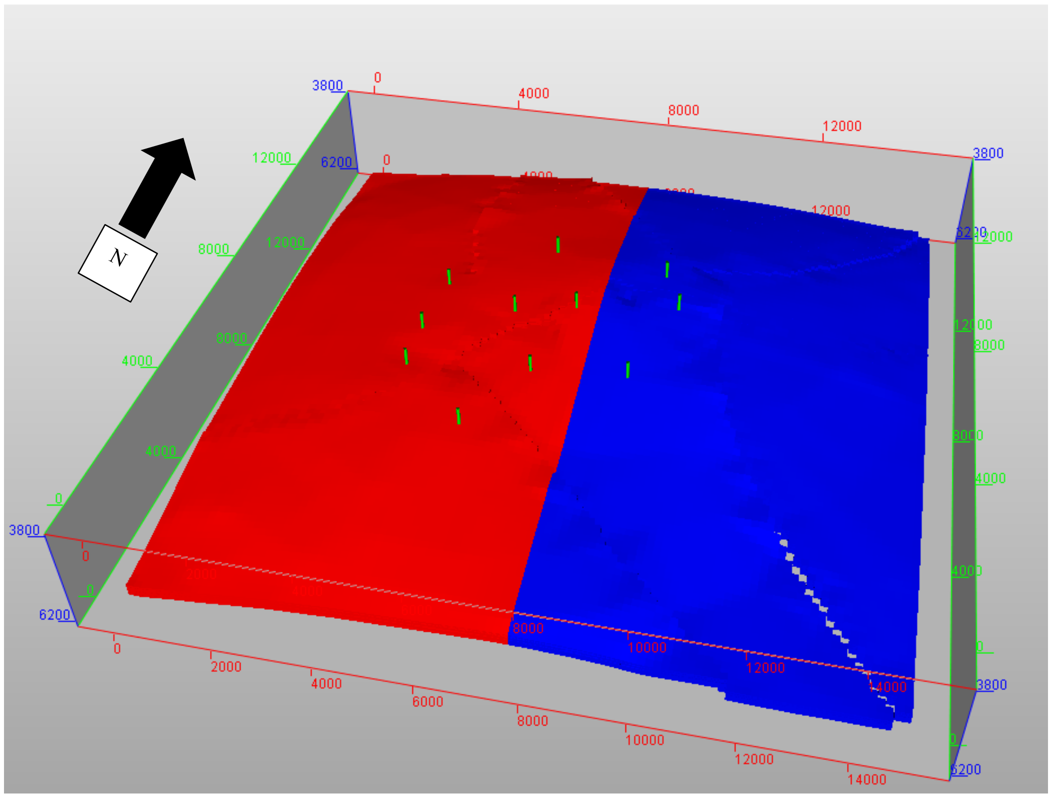


Figure 4. Example of 2D decomposition with 2 partitions

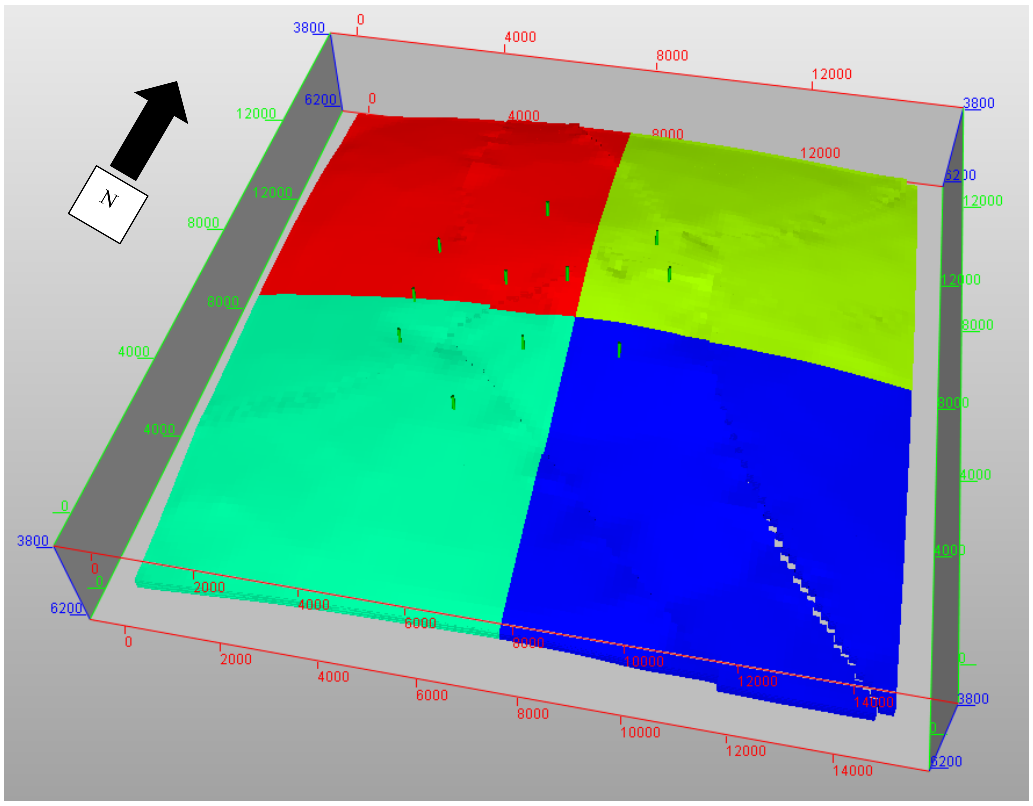


Figure 5. Example of 2D decomposition with 4 partitions

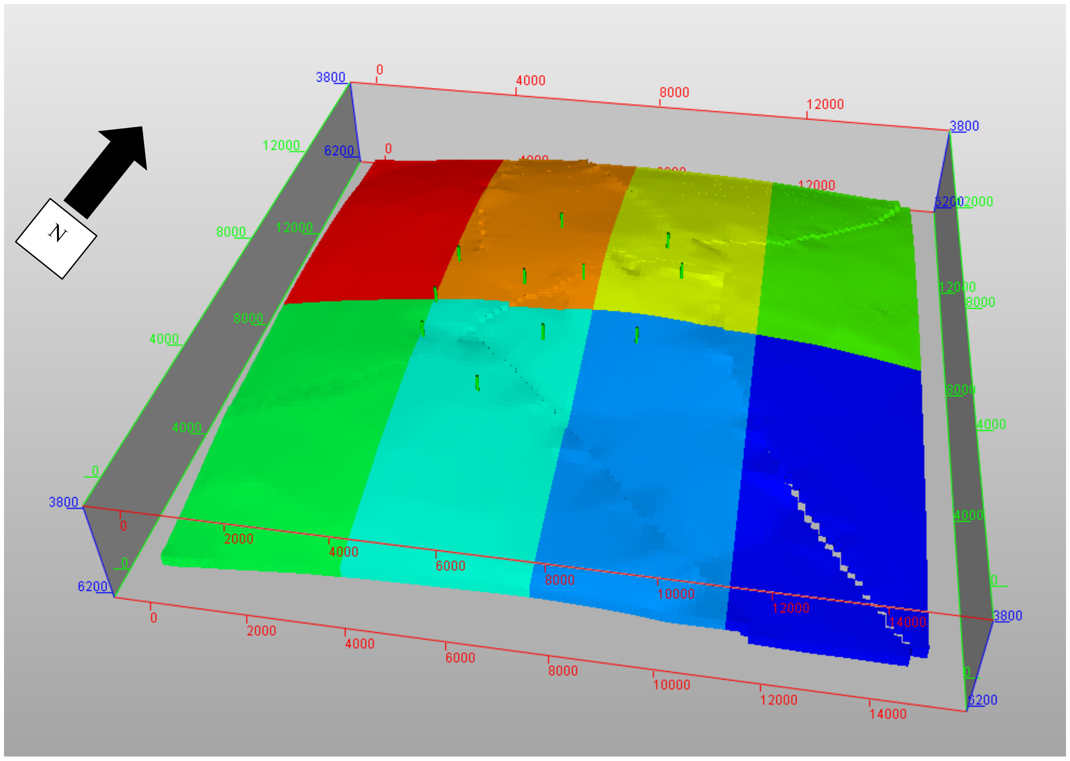


Figure 6. Example of 2D decomposition with 8 partitions

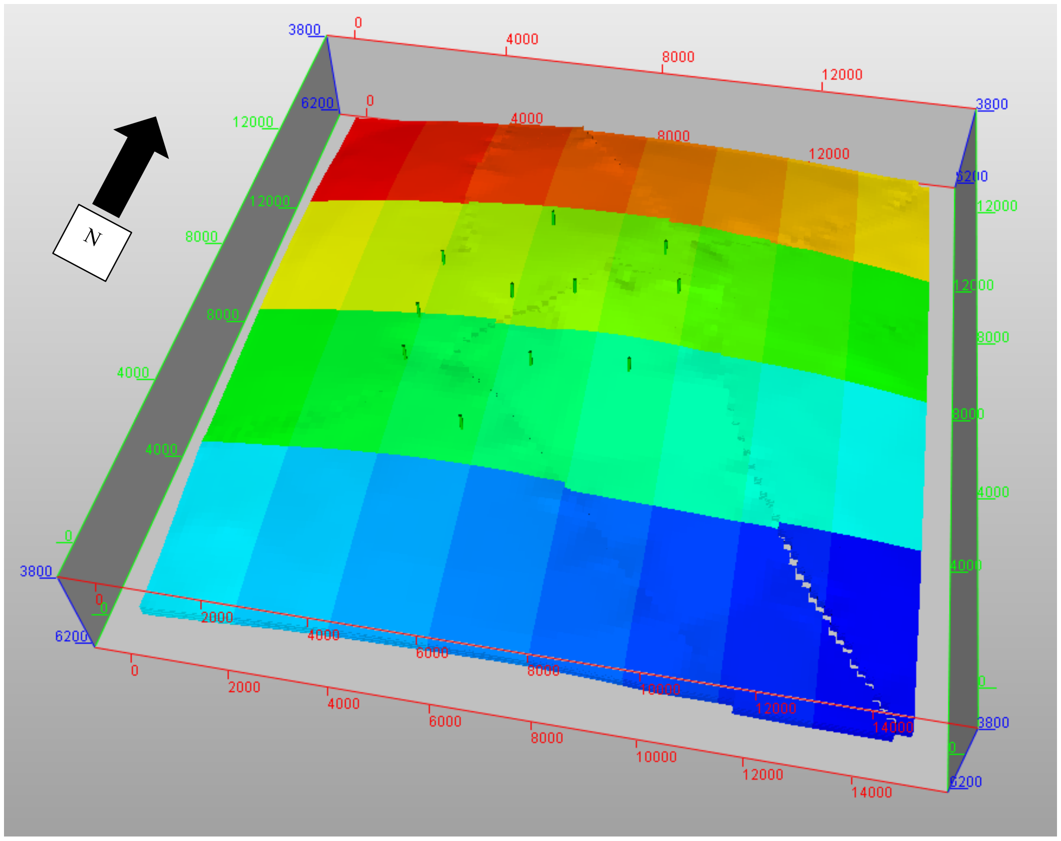


Figure 7. Example of 2D decomposition with 32 partitions

3.3 Metis partitioning

Metis is a graph partitioning algorithm that can be applied to partitioning unstructured graphs and partitioning meshes. Generally speaking, this method has an untraditional algorithm of reading and partitioning meshes. Traditional methods directly read the mesh or graph while Metis pre-processes them first and then partition them. With such improvement, Metis is very fast and efficient to partition grids, especially for large grids like the one in this research (Karypis and Kumar, 1998b). From the practice in the million grid block decomposition case, it usually takes several seconds to partition the mesh with one million blocks. This saves some pre-processing time. In addition, partitions generated by Metis have small numbers of communications in between. Metis graph partition mainly has two routines: multilevel recursive bisection and multilevel k-way partitioning. In this research, Metis k-way graph partitioning feature is used since it can offer minimized communications between partitions and also ensure that partitions are contiguous. These are both preferred to address load imbalance issues.

According to Karypis and Kumar (1998c), the multilevel k-way graph partitioning is a satisfactory partitioning algorithm. Although both multilevel k-way partitioning and multilevel recursive bisection involve graph coarsening, the multilevel k-way partitioning requires only one time coarsening. The multilevel recursive bisection strategy first coarsens a large graph into a new graph with less vertices, then the new graph is partitioned into two. Based on the partition, the coarsened graph is then projected back into the original graph with more vertices. To repeat this bisection strategy, the process is repeated recursively. During this recursive bisection, many coarsening and refining are done. However, this repetitive process is simplified in multilevel k-way graph partitioning. In this partitioning, the original graph only needs to be coarsened for once. Once the coarsened graph is partitioned, a refinement method called Kernighan-Lin algorithm is used to refine the coarsened graph back into the original graph with more vertices. This algorithm involves switching vertices between partitions to reduce the number of edge cuts. It is worth mentioning that this refinement algorithm is also used in the spectral partitioning method and it is proved to be very efficient to reduce edge cuts for spectral partitioning. When switching vertices, the algorithm also makes sure that the number of vertices in each partition is not changed.

Two factors regarding the reservoir model are taken into consideration while applying Metis partitioning. The first is grid geometry. An adjacency matrix is introduced to represent grid geometry. The column or row number of the matrix is equal to the total grid block number. Each grid block’s neighboring blocks are stored in its corresponding row in the adjacency matrix. In Metis, Compressed Row Storage (CSR) is used so that the partitioning speed is largely increased. The second consideration is the transmissibility field. Transmissibility distribution is highly related to reservoir related computation. Since the x-direction transmissibility and y-direction transmissibility distribution are nearly identical, only the first layer’s x-direction transmissibility field is put into Metis as weighting factor. **Figure 8** shows the Metis k-way partitioning result for four subdomains. **Figure 9** shows the x-direction transmissibility field. It is easily noticed that Metis tends to assign smaller size of partitions to areas where transmissibility is high. The reason is that high transmissibility leads to larger property changes. As a result, more iterations are required to reach convergence for each time step in the simulation. To present this trend more clearly, an exaggerated case is generated. In this case, the transmissibility is manually increased to 0.5 RB-CP/DAY-PSI for a square from i=1 to i=30 and j=1 to j=30. Thus, the transmissibility is extremely high for the upper-left section of the model. In other words, this part of the grid has very high weighing factors. **Figure 10** is the partitioning result. As expected, very fine subdomains are partitioned for the upper-left corner while the rest of the gird is assigned very coarse partitions.

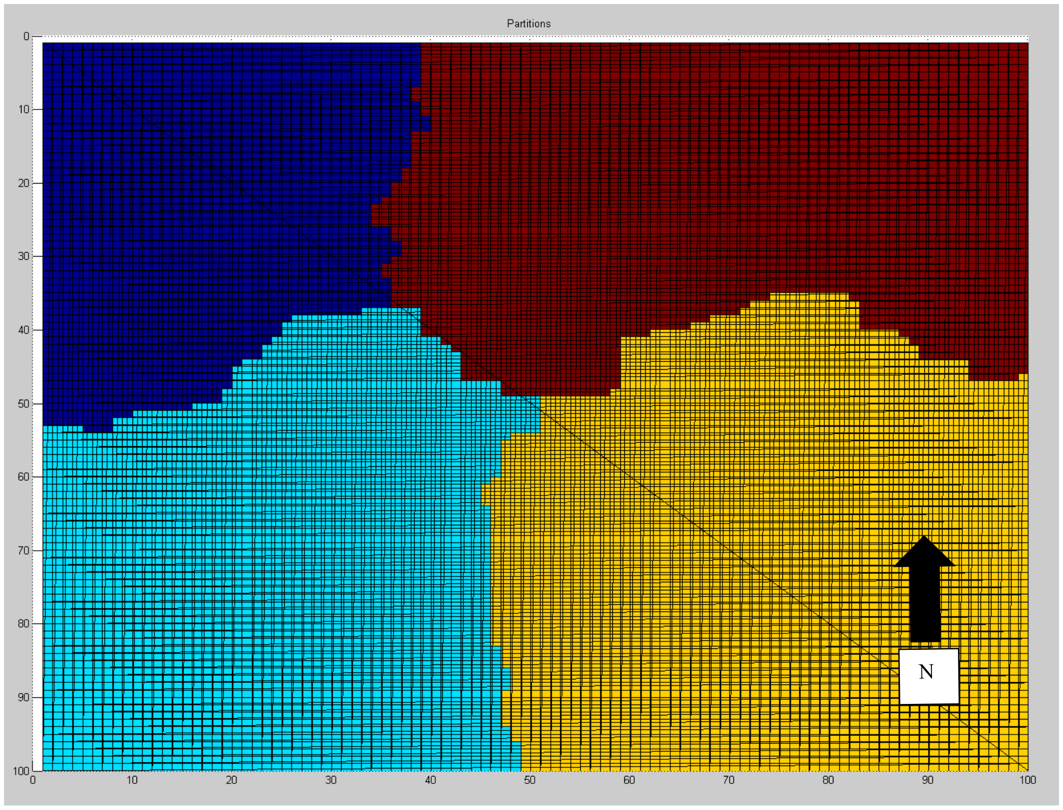


Figure 8. 4 Metis k-way partitions

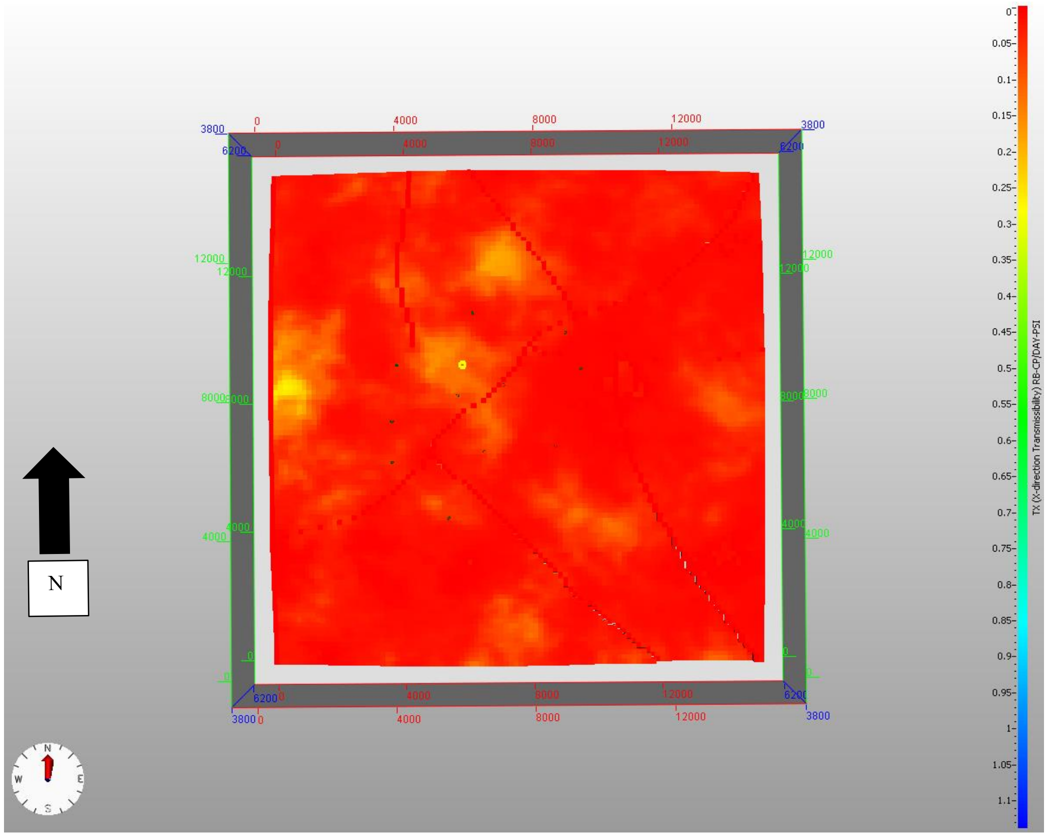


Figure 9. X direction transmissibility field

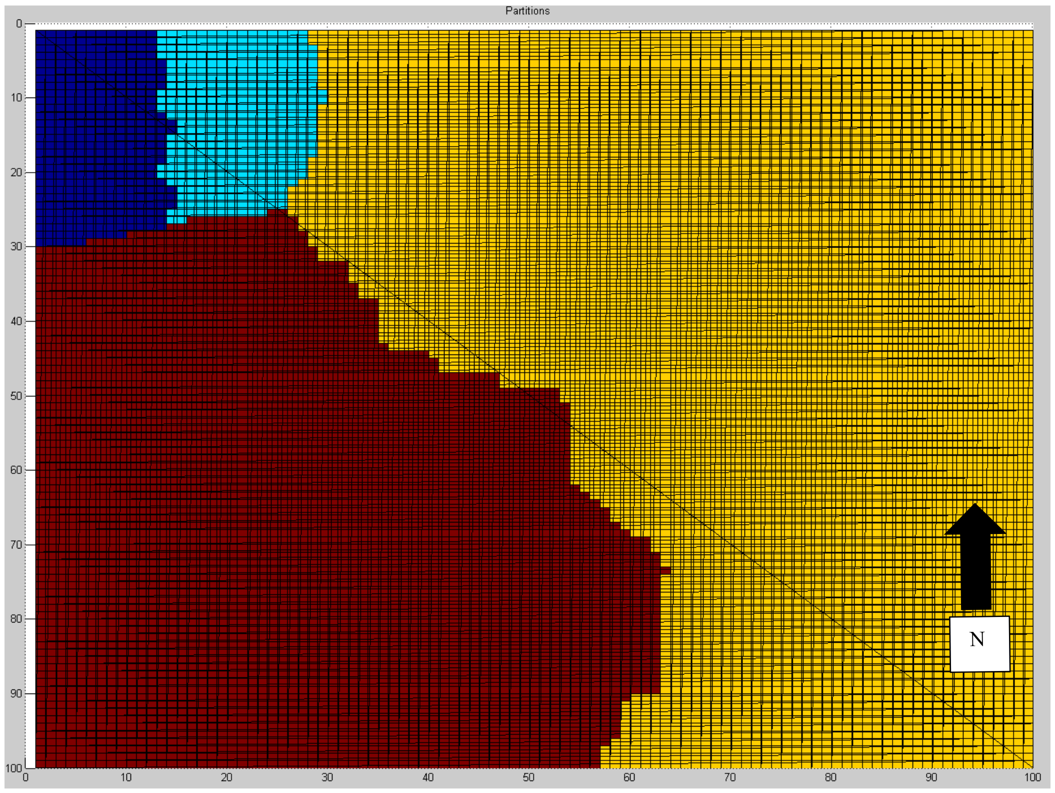


Figure 10. Metis k-way partition result in the extreme case

Figure 11 to Figure 14 show the partitions for this study from 2, 8, 16, to 32. It can be observed that each subdomain has similar amounts of grid blocks. Also, the edge cuts between neighboring partitions are limited.

It is worth mentioning that in the k-way partition for 16 subdomains, the subdomain containing coordinate (50, 5) and the one containing (50, 40) actually belong to the same partition. This assignment has the potential to increase overheads during the parallel run. However, since these two partitions are relatively small and the weights assigned to them are low, the negative effects of this partitioning pattern is expected to be small.

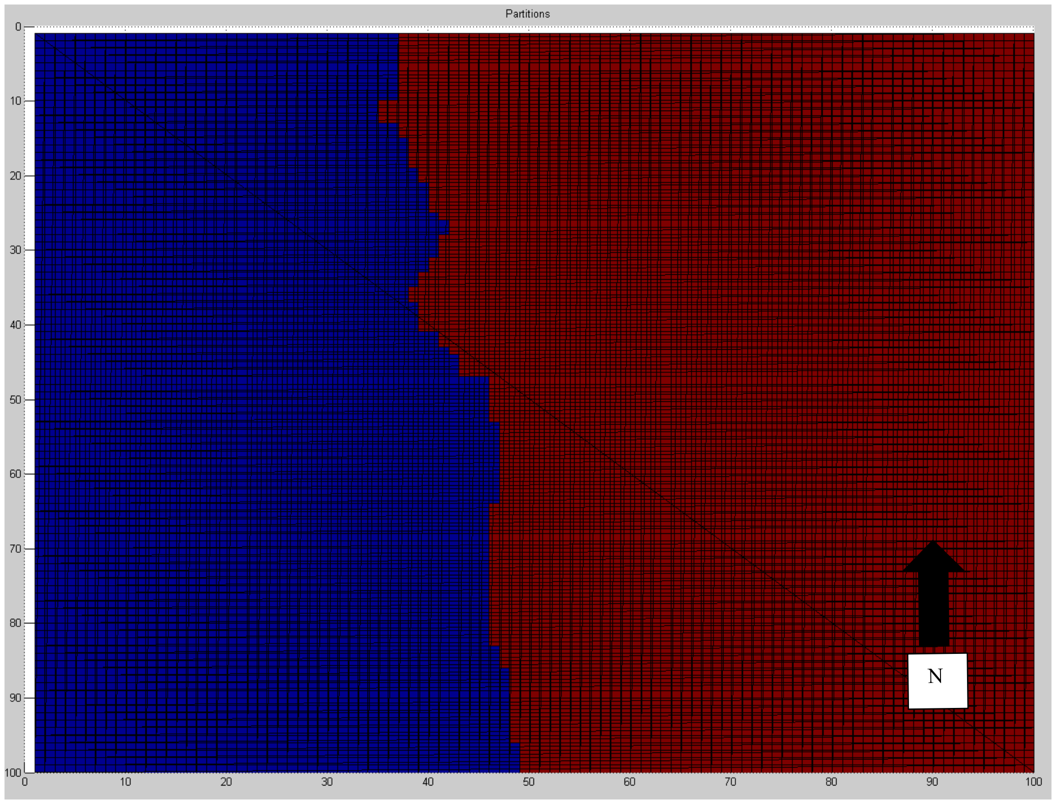


Figure 11. 2 Metis k-way partitions

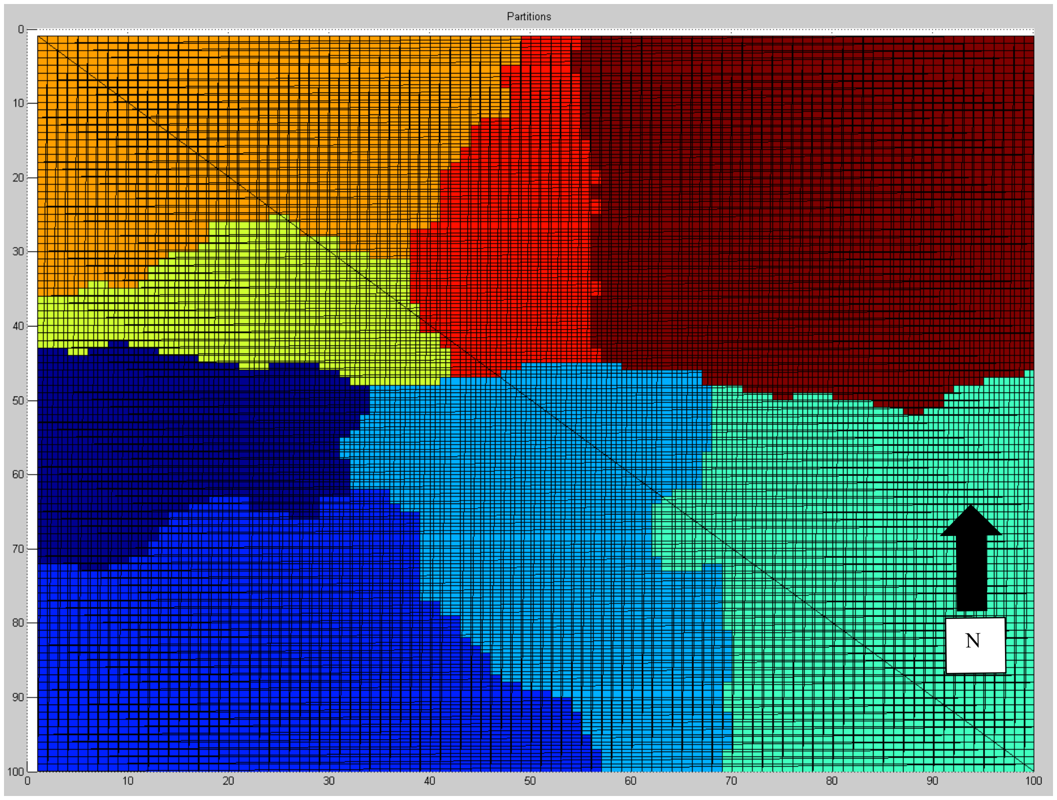


Figure 12. 8 Metis k-way partitions

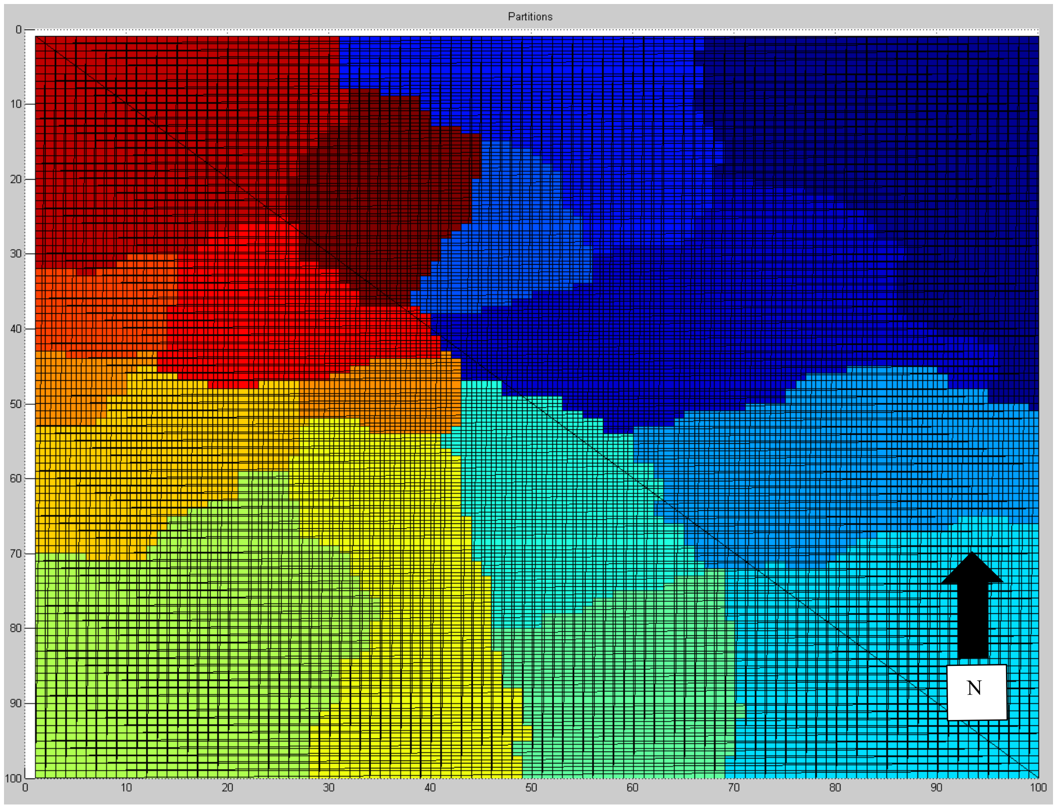


Figure 13. 16 Metis k-way partitions

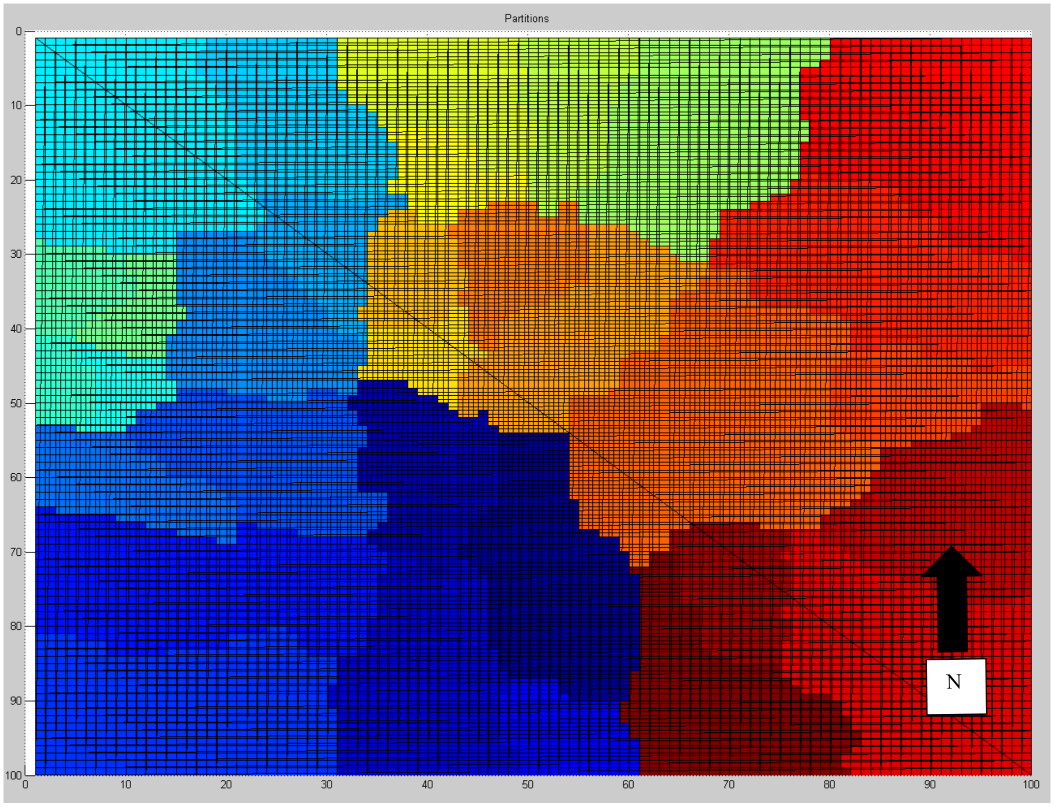


Figure 14. 32 Metis k-way partitions

3.4 Spectral partitioning

Spectral partitioning is based on matrices established from the mesh or graph. An adjacency matrix and a degree matrix is generated based on the graph. They then form a Laplacian matrix, whose second eigenvector (Fiedler vector) divides the graph into two. If more than two partitions are needed, one can repeat this manner as recursive bisection (Fiedler 1975). A description of a spectral bisection procedure is given in the following paragraph.

According to Pothen et al. (1990), the second eigenvector is the eigenvector corresponding to the second smallest eigenvalue. For a graph with n vertices, a median value x_l is calculated based on all the values in the second eigenvector with n components. All vertices are separated into two groups: one group contains vertices with components larger than x_l and the other group contains vertices with components smaller than x_l. In the case that there are components with a value equal to x_l, they can be arbitrarily assigned to either group one or group two. The only restriction is that the difference of numbers of components in the two groups should be at most one. Then an edge separator is calculated and integrated to the two-group separation. In this study, the partitioning package Chaco has an option, the Kernighan-Lin algorithm, to minimize communications between partitions.

Similar to Metis partitioning, weights can be added to vertices and edges of the graph to reflect important areas and less important areas. **Figure 15**, **Figure 16**, **Figure 17**, **Figure 18**, and **Figure 19** show spectral strategies with 2, 4, 8, 16, and 32 partitions with Kernighan and Lin algorithm. **Figure 20** and **Figure 21** show two partition results for 2-process and 4-process without Kernighan and Lin algorithm. The difference brought in by this algorithm is discussed in the following paragraph.

Generally, spectral methods can partition very large meshes into subdomains. However, the details of the subdomains sometimes are not optimum in terms of edge cuts. Also, this method sometimes assign two discrete subdomains to one partition, which will definitely increase the overhead during simulation in a multi-core cluster. For example, **Figure 20** shows two partitions: partition 1 and partition 2. It is very obvious that there are two subdomains for partition 1 and two subdomains for partition 2. Such partitioning fashion significantly increase edge cuts and communications between processes when it is implemented in a parallel job. Similar phenomenon is also observed for a 4-piece partitioning job in **Figure 21**. This kind of result is generated by spectral strategy is because the spectral method primarily emphasizes on load balance among partitions. Edge cuts and communications are not minimized (Hendrickson and Kolda 2000). To find a balance between balanced grid block numbers among partitions and smallest possible communications, a local refinement strategy called Kernighan-Lin algorithm is introduced. It is applied right after a mesh is primitively partitioned by the spectral method (Hendrickson and Leland 1995). With this local refinement, scattered subdomains that belong to the same partition are connected and communications are reduced. This modification can be observed by comparing **Figure 15** and **Figure 20** for 2-partition job, and **Figure 16** and **Figure 21** for 4-partition job.

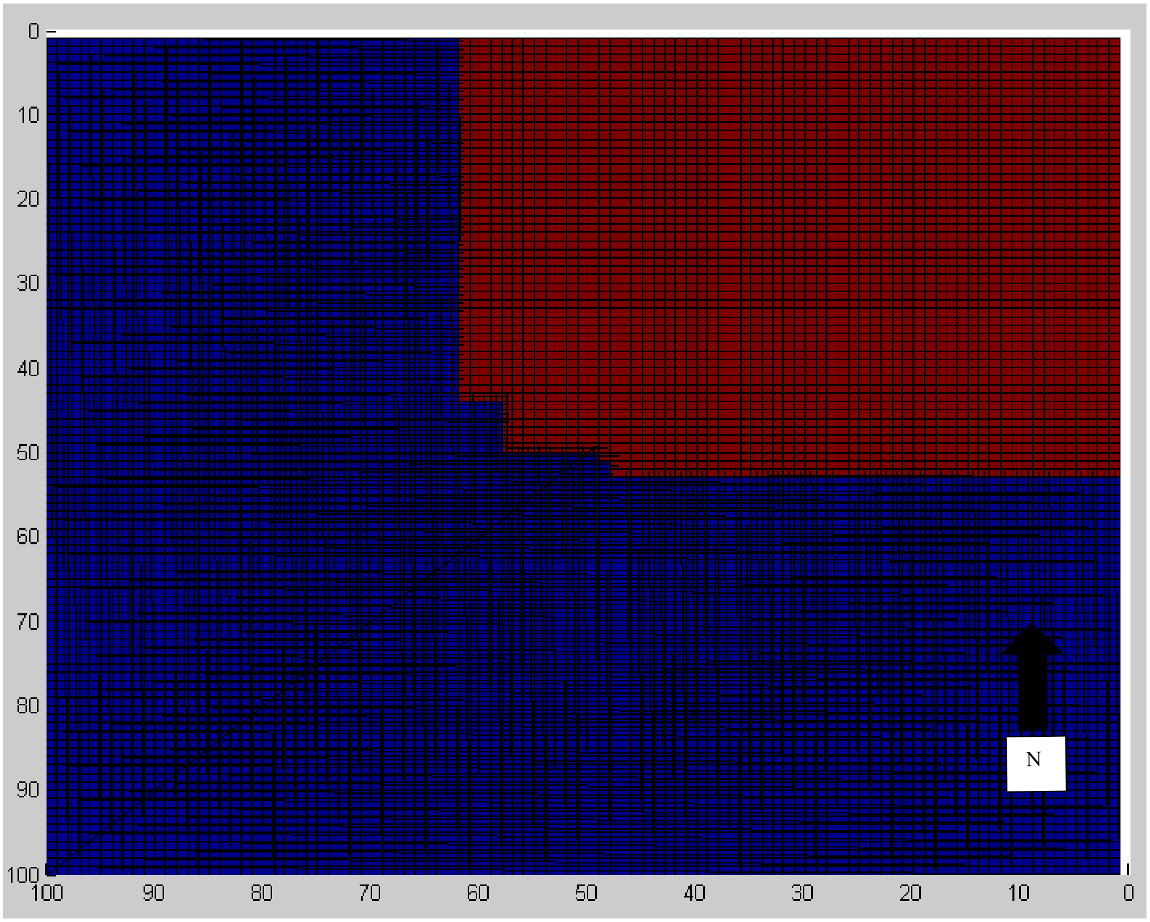


Figure 15. 2 spectral partitions

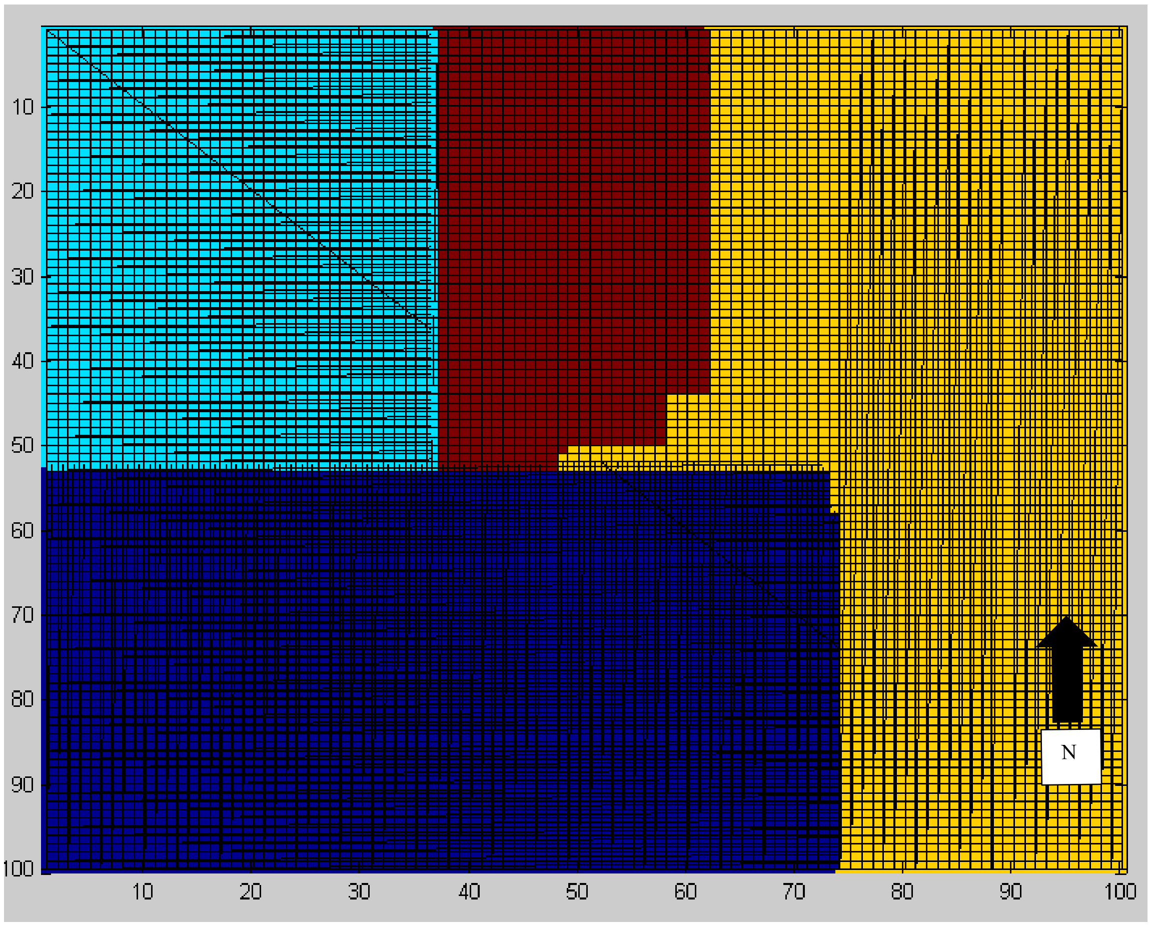


Figure 16. 4 spectral partitions

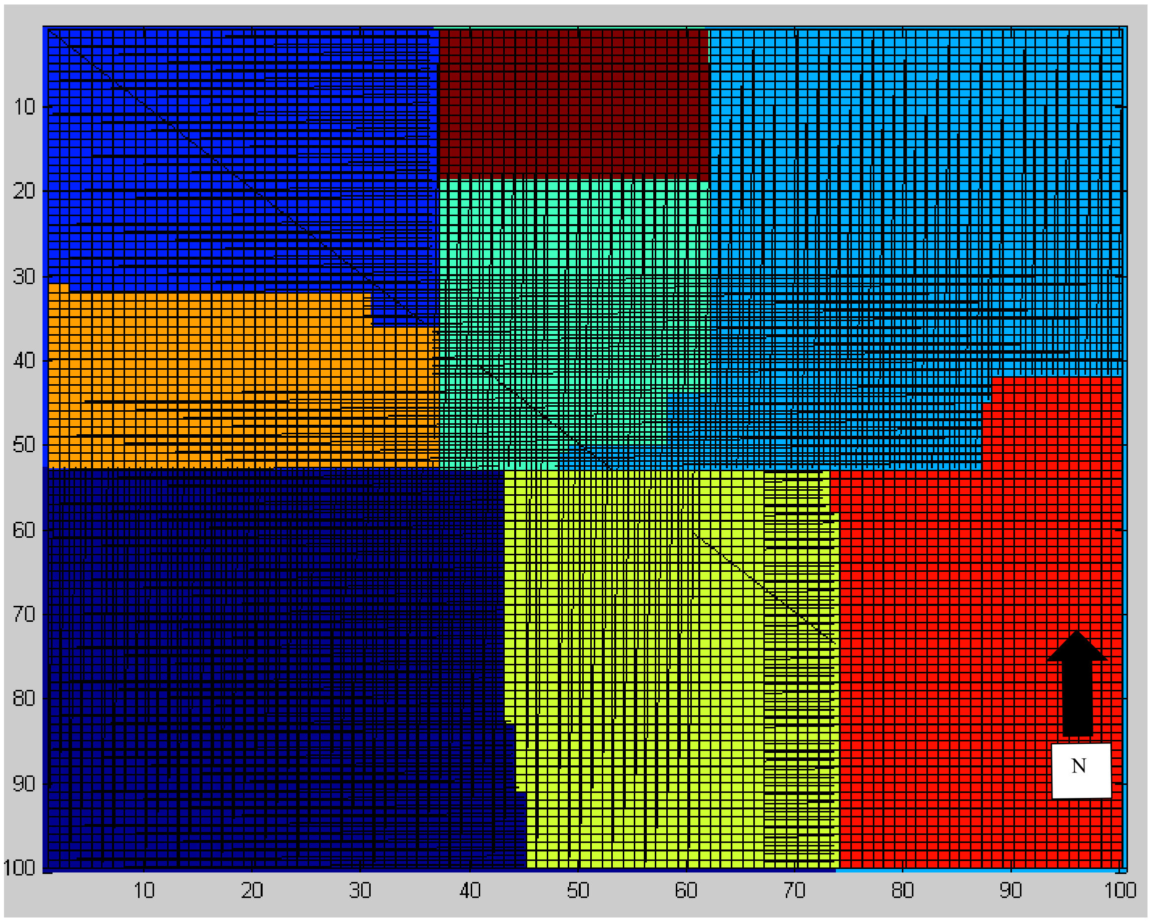


Figure 17. 8 spectral partitions

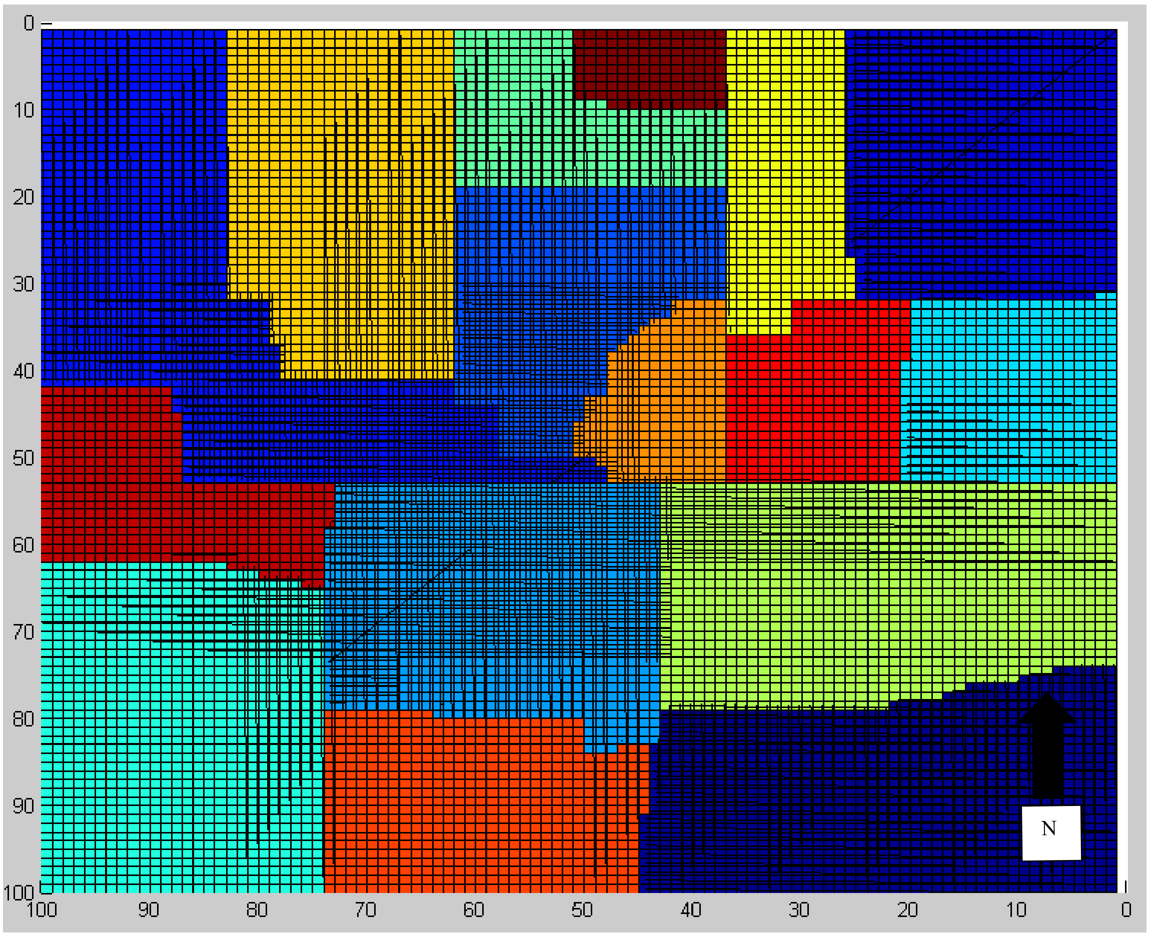


Figure 18. 16 spectral partitions

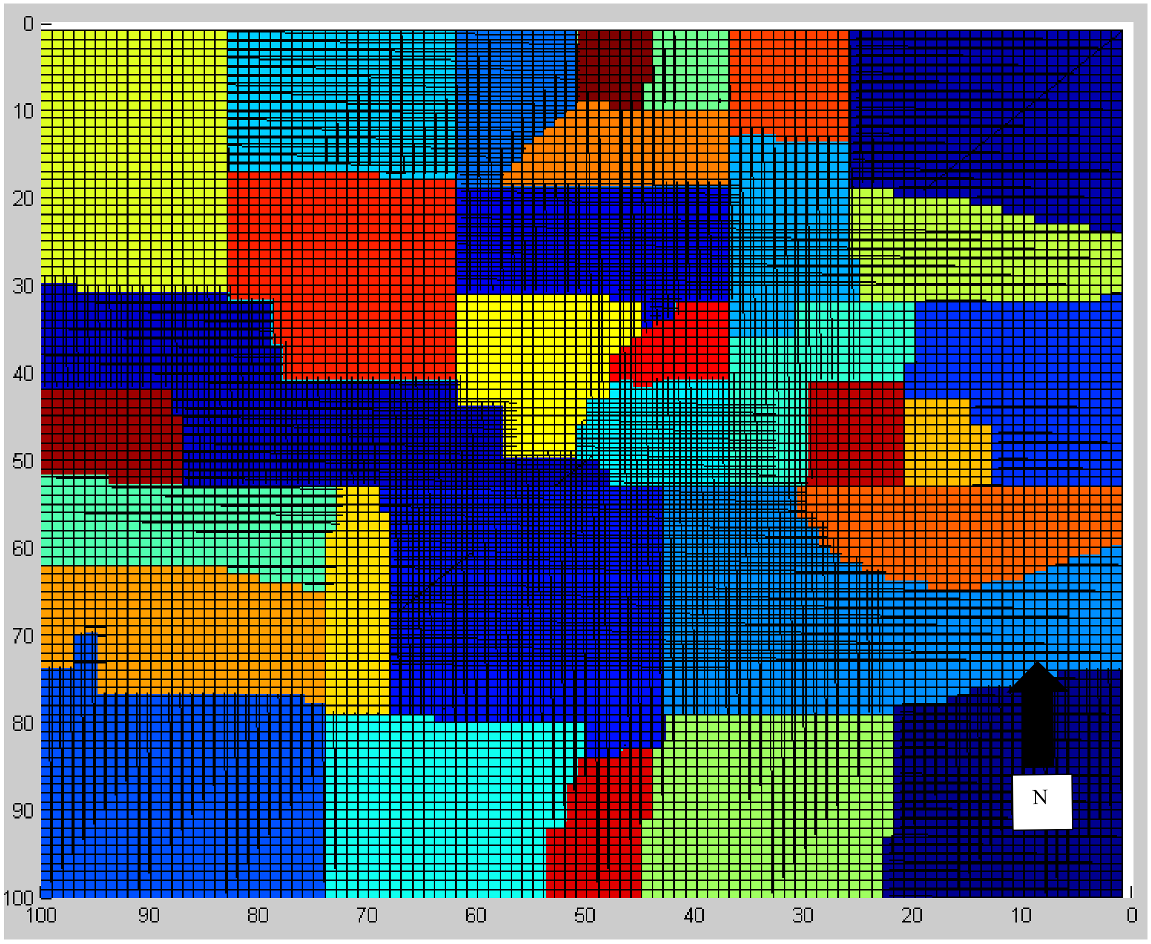


Figure 19. 32 spectral partitions

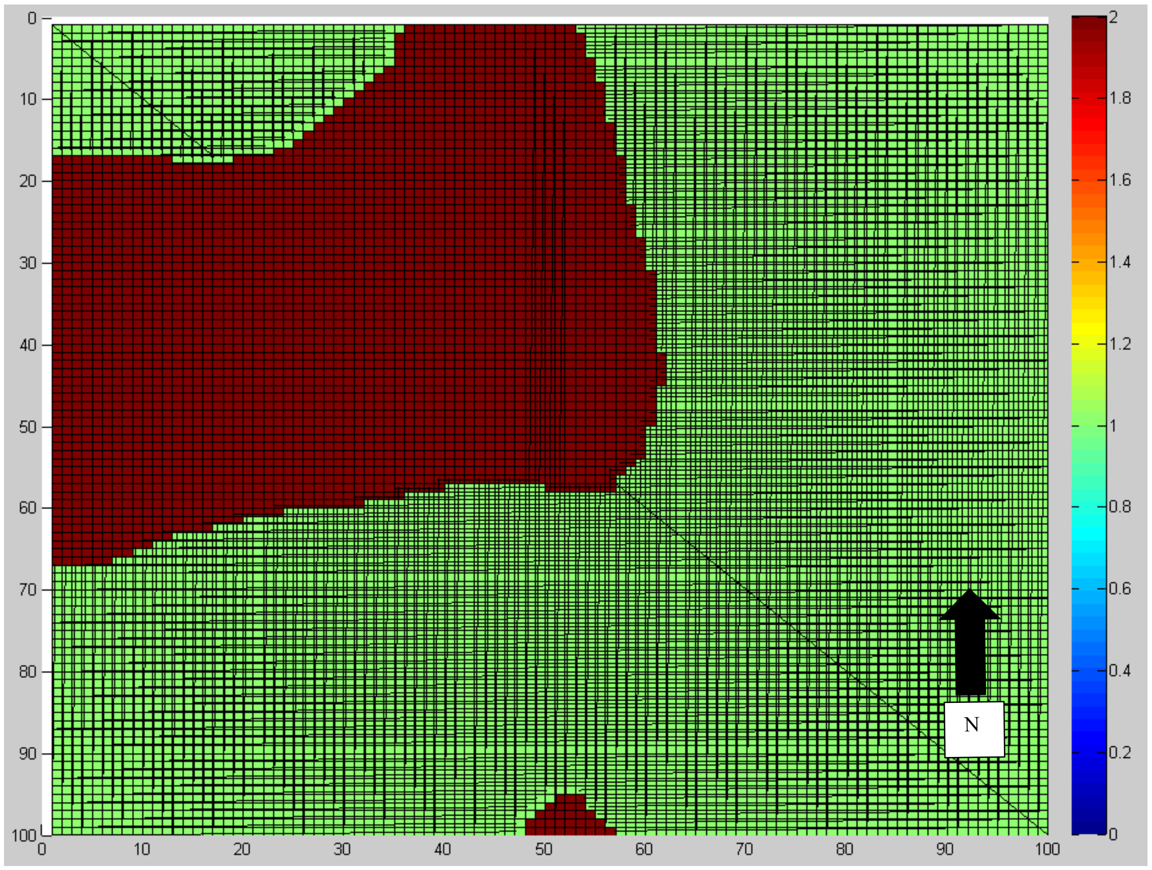


Figure 20. Spectral partitioning for 2 without Kernighan Lin algorithm

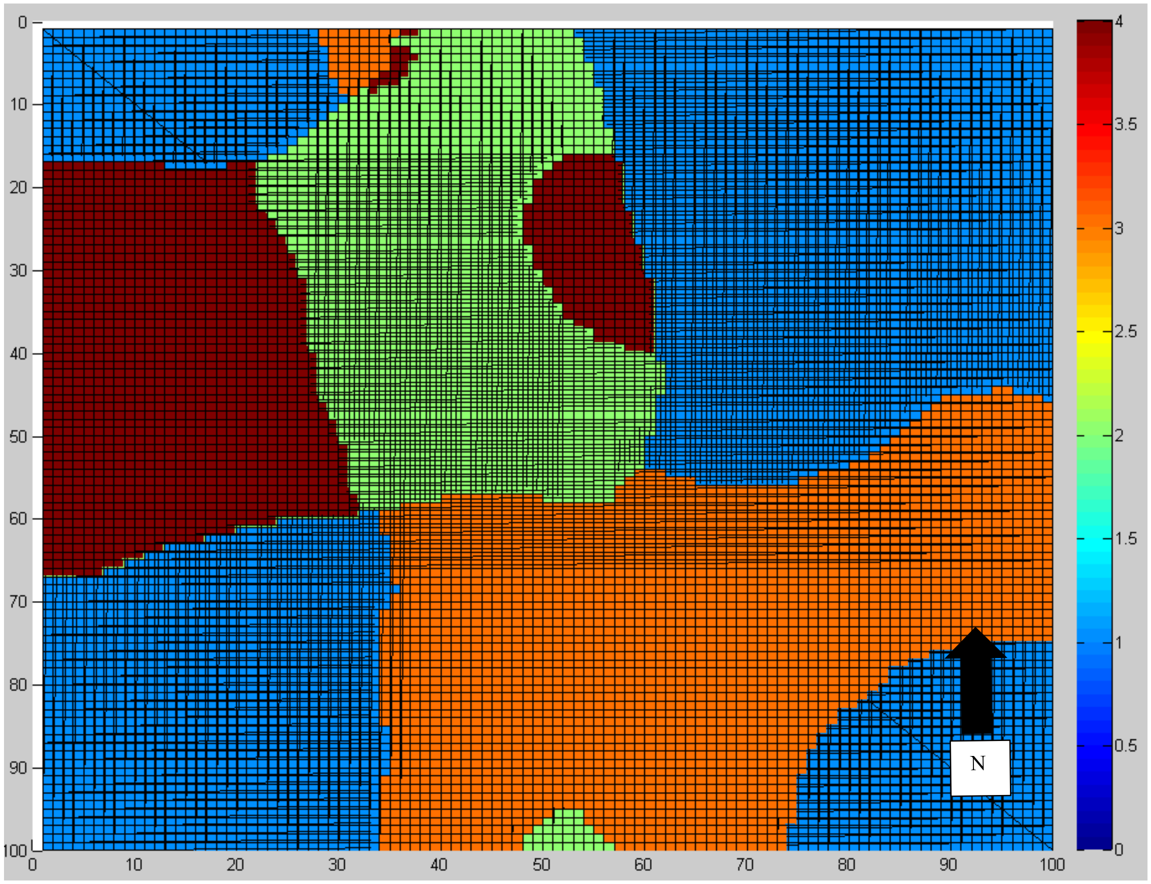


Figure 21. Spectral partitioning for 4 without Kernighan Lin algorithm

3.5 Zoltan partitioning

Zoltan graph partitioner is able to balance grid blocks among subdomains while minimizing edge cuts. This method is already incorporated in the simulator. A keyword "ZOLTAN_OPTIONS" is used to control this partitioner. However, this implementation is not as efficient as other partitioning strategies since it does not utilize all the processes assigned in a parallel run. The following picture **Figure 22** is a Zoltan partition targeted for 32 subdomains. However, with the embedded partitioning package's modification, it only generates totally 7 partitions.

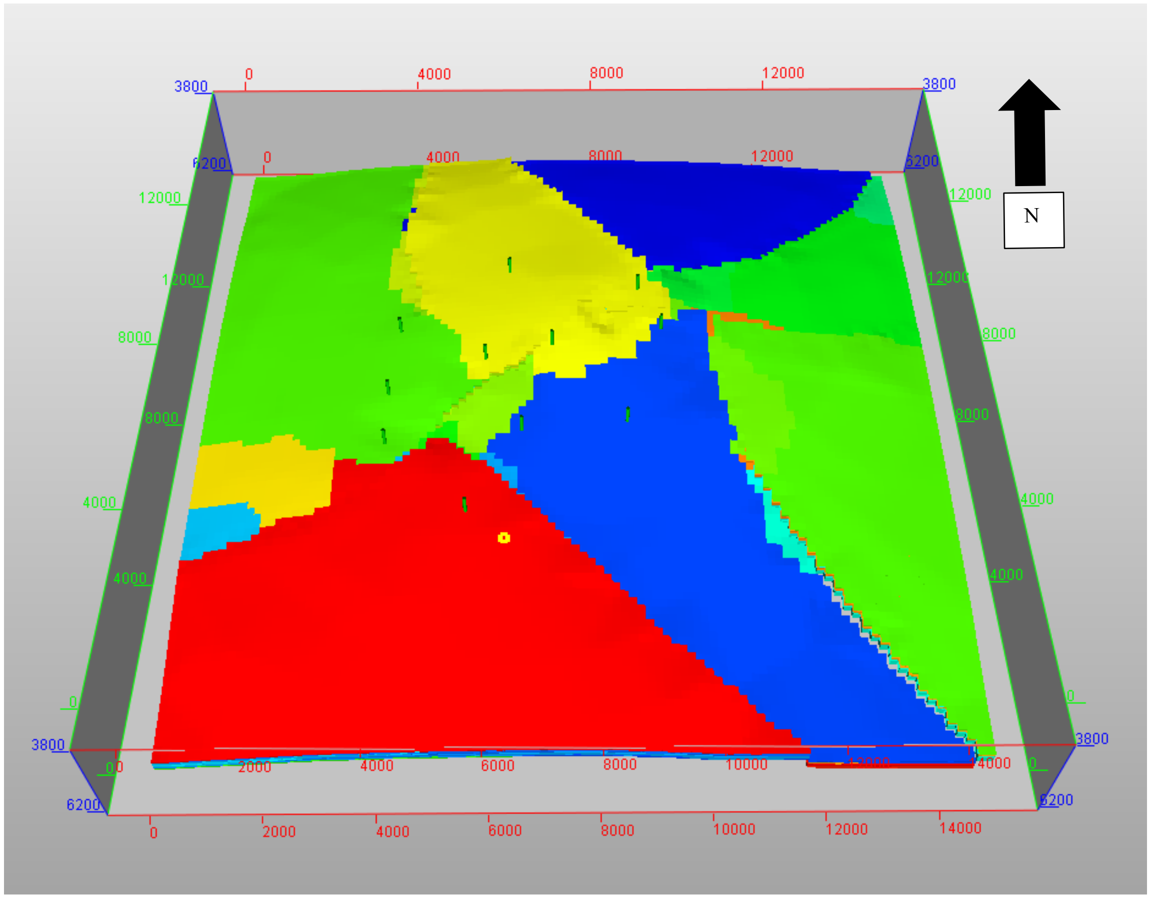


Figure 22. 7 Zoltan partitions

3.6 Conclusions

This chapter discussed several partitioning strategies: 2D decomposition, Metis partitioning, spectral partitioning, and Zoltan partitioning. It is noted that except for the Zoltan partitioner, all the rest can generate partitions up to 32 as needed in this study. Besides, Metis and spectral partitioning results have less communications among partitions than 2D decomposition.

More partitions result in more communications among partitions and this consequently increases the system overheads. A phenomenon is observed for both Metis and spectral partitioning that they both may return partitioning results that assign two non-neighboring subdomains into one partition. This assignment largely increases parallel implementation’s overhead. Fortunately, if the Kernighan-Lin algorithm is used to modify the result, this phenomenon is removed. After the introduction of the algorithm, communications among partitions are reduced while the balance of grid blocks among partitions is still maintained.

Table 5 to **Table 9** have 2D decomposition, Metis partitioning, and spectral partitioning’s partitioning results with each partition’s grid block number. **Table 10** has the edge-cuts of the three partitioning strategies. They show that 2D decomposition has the least edge-cuts while Metis has the highest edge-cuts.

Table 5. Each partition's grid block number, 2 partitions

Partition	2D	Metis	Spectral
1	500000	422600	689300
2	500000	577400	310700

Table 6. Each partition's grid block number, 4 partitions

Partition	2D	Metis	Spectral
1	250000	162000	349900
2	250000	257600	187200
3	250000	316700	339400
4	250000	263700	123500

Table 7. Each partition’s grid block number, 8 partitions

Partition	2D	Metis	Spectral
1	125000	71900	204400
2	125000	157300	113800
3	125000	140500	194200
4	125000	173800	78500
5	125000	56500	145500
6	125000	126200	73400
7	125000	60400	145200
8	125000	213400	45000

Table 8. Each partition’s grid block number, 16 partitions

Partition	2D	Metis	Spectral
1	62500	104200	101800
2	62500	99000	76800
3	62500	64400	114200
4	62500	24700	55700
5	62500	78800	84600
6	62500	102400	41500
7	62500	52400	103600
8	62500	54600	32700
9	62500	104600	102600
10	62500	76600	37000
11	62500	44700	80000
12	62500	21300	22800
13	62500	16400	60900
14	62500	39200	31900
15	62500	79200	41600
16	62500	37500	12300

Table 9. Each partition's grid block number, 32 partitions

Partition	2D	Metis	Spectral	Partition	2D	Metis	Spectral
1	31200	41700	53600	17	31200	49000	48200
2	31200	26500	50700	18	31200	19500	26100
3	31200	43000	60600	19	31200	24500	53600
4	31200	15300	31800	20	31200	11600	23900
5	31200	53300	69800	21	31200	9300	14800
6	31200	55500	34100	22	31200	13100	7400
7	31200	34400	67200	23	31200	33800	36400
8	31200	19200	15800	24	31200	18300	16900
9	31200	31300	74000	25	31200	78000	28600
10	31200	15900	23100	26	31200	19900	13900
11	31200	30300	34300	27	31400	53000	45700
12	31200	54100	14900	28	31400	55900	7900
13	31200	7300	46500	29	31400	63100	14400
14	31200	6400	21100	30	31400	40400	10800
15	31200	9600	27200	31	31400	18200	14400
16	31200	8800	6600	32	31800	39800	5700

Table 10. Edge-cuts summary

Partition	2D	Metis	Spectral
2	100	129	113
4	200	274	239
8	400	532	436
16	600	820	677
32	1002	1249	1032

CHAPTER IV

PARALLEL PERFORMANCE OF THE ORIGINAL MODEL

4.1 Results of 2D decomposition

Enhanced performance is obtained by applying 2D decomposition. To quantitatively present load imbalance, at a certain time, the maximum load is regarded as 100% load and other processes' load are normalized with this maximum value. **Table 11** and **Table 12** show computation time distribution among different sections of a 2-process parallel run and a 16-process parallel run. These two tables are just generated to show the time distribution for subsections of the simulator and these data are not used for later discussions. The load balance and inter-process communication data are recorded in **Table 13**, **Table 14**, **Table 15**, **Table 16**, and **Table 17** for 2-, 4-, 8-, 16-, and 32-process parallel runs respectively. The cases described by Table 11 and Table 12 are run in a different job file other than the one used for cases recorded from Table 13 to Table 16. As a result, the total simulation time on process 2 in Table 11 is not the same as the total time on process 2 in Table 13. Also, the total time on process 16 in Table 12 is not the same as the total time on process 16 in Table 16.

Table 11. Process 2 work distribution for 2-process run

2 Processes	CPU Seconds	% Total Run	Elapsed Seconds	% Total Run
INPUT	1.897	0.09	1.898	0.09
OUTPUT	1.124	0.06	0.956	0.05
INITIALIZATION	71.198	3.56	71.16	3.56
PVT PROPERTIES	667.276	33.38	667.026	33.39
ROCK PROPERTIES	18.079	0.9	18.075	0.9
EQUATION SETUP	35.817	1.79	35.798	1.79
NETWORK/WELLS	4.68	0.23	5.009	0.25
SOLVER	819.481	40.99	819.094	41
UPDATE	77.973	3.9	77.895	3.9
MISC.	301.471	15.08	300.911	15.06
Total Run	1998.996	100.00	1997.822	100.00

Table 12. Process 16 work distribution for 16-process run

16 Processes	CPU Seconds	% Total Run	Elapsed Seconds	% Total Run
INPUT	2.137	0.25	2.421	0.29
OUTPUT	1.2	0.14	1.48	0.18
INITIALIZATION	13.465	1.6	14.469	1.72
PVT PROPERTIES	21.099	2.51	21.235	2.52
ROCK PROPERTIES	2.251	0.27	2.258	0.27
EQUATION SETUP	21.504	2.56	21.215	2.52
NETWORK/WELLS	5.427	0.65	5.988	0.71
SOLVER	454.452	54.1	454.425	53.99
UPDATE	44.892	5.34	44.892	5.33
MISC.	273.529	32.58	273.228	32.46
Total Run	839.956	100	841.611	99.99

Table 13. Message passing and load balance status for the 2-process run

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	0.363	5.580	5.943	2221.893	2215.950	99.73%	428666
2	6.348	342.393	348.741	2222.189	1873.448	84.31%	421705
		Total M-P, s	354.684		Average Load	92.02%	

Table 14. Message passing and load balance status for the 4-process run

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	2.093	59.153	61.246	1205.102	1143.856	94.92%	208400
2	4.533	30.202	34.735	1205.662	1170.927	97.12%	215022
3	6.287	187.476	193.763	1205.651	1011.888	83.93%	211407
4	7.265	484.525	491.790	1205.664	713.874	59.21%	215542
		Total M-P, s	781.534		Average Load	83.79%	

Table 15. Message passing and load balance status for the 8-process run

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	5.602	370.954	376.556	880.489	503.933	57.23%	100918
2	3.980	55.628	59.608	881.116	821.508	93.23%	107482
3	4.258	57.649	61.907	881.116	819.209	92.97%	106742
4	6.872	332.421	339.293	881.121	541.828	61.49%	108280
5	8.386	398.044	406.430	881.119	474.689	53.87%	102703
6	5.322	139.929	145.251	881.121	735.870	83.52%	108704
7	6.990	339.124	346.114	881.120	535.006	60.72%	106969
8	7.869	499.043	506.912	881.122	374.210	42.47%	108573
		Total M-P, s	2242.071		Average Load	68.19%	

Table 16. Message passing and load balance status for the 16-process run

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	4.912	397.545	402.457	767.830	365.373	47.59%	48951
2	6.302	172.944	179.246	768.706	589.460	76.68%	53020
3	6.715	196.048	202.763	768.702	565.939	73.62%	52121
4	7.853	374.724	382.577	768.671	386.094	50.23%	51471
5	7.271	233.260	240.531	768.697	528.166	68.71%	51967
6	4.580	117.394	121.974	768.704	646.730	84.13%	54462
7	4.527	101.007	105.534	768.704	663.170	86.27%	54621
8	6.136	210.703	216.839	768.701	551.862	71.79%	56809
9	7.518	245.783	253.301	768.700	515.399	67.05%	50622
10	4.665	70.688	75.353	768.705	693.352	90.20%	57059
11	6.436	188.324	194.760	768.704	573.944	74.66%	53381
12	6.953	343.887	350.840	768.689	417.849	54.36%	55126
13	7.970	403.097	411.067	768.672	357.605	46.52%	52081
14	7.677	317.091	324.768	768.684	443.916	57.75%	51645
15	7.598	399.404	407.002	768.679	361.677	47.05%	53588
16	8.003	413.227	421.230	768.670	347.440	45.20%	53447
		Total M-P, s	4290.242		Average Load	65.11%	

Table 17. Message passing and load balance status for the 32-process run

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	4.908	423.767	428.675	947.026	518.351	54.73%	25126
2	11.457	376.947	388.404	952.394	563.990	59.22%	25764
3	10.494	225.157	235.651	951.373	715.722	75.23%	26263
4	9.115	190.181	199.296	952.613	753.317	79.08%	29403
5	10.837	235.682	246.519	950.624	704.105	74.07%	25384
6	11.531	268.984	280.515	952.561	672.046	70.55%	24170
7	11.846	373.538	385.384	949.832	564.448	59.43%	24814
8	12.049	423.160	435.209	952.617	517.408	54.31%	24639
9	11.491	387.710	399.201	950.710	551.509	58.01%	26337
10	9.978	160.092	170.070	952.794	782.724	82.15%	27847
11	8.611	153.742	162.353	951.362	789.009	82.93%	27886
12	8.314	148.882	157.196	951.888	794.692	83.49%	28791
13	9.338	180.784	190.122	945.994	755.872	79.90%	25815
14	9.279	144.850	154.129	952.394	798.265	83.82%	26605
15	10.015	174.835	184.850	950.289	765.439	80.55%	27833
16	11.079	389.056	400.135	952.227	552.092	57.98%	26745
17	11.682	375.380	387.062	951.320	564.258	59.31%	25675
18	10.460	165.779	176.239	952.863	776.624	81.50%	27174
19	8.356	114.564	122.920	951.579	828.659	87.08%	29833
20	9.074	122.472	131.546	952.900	821.354	86.20%	29443
21	10.648	223.054	233.702	951.127	717.425	75.43%	26077
22	11.272	252.724	263.996	952.925	688.929	72.30%	25013
23	11.049	330.968	342.017	951.746	609.729	64.06%	26742
24	11.511	399.335	410.846	952.887	542.041	56.88%	26231
25	11.441	405.305	416.746	952.126	535.380	56.23%	27092
26	11.334	387.749	399.083	952.114	553.031	58.08%	27005
27	11.058	317.101	328.159	952.038	623.879	65.53%	27176
28	11.234	335.524	346.758	952.099	605.341	63.58%	26711
29	11.558	401.108	412.666	948.714	536.048	56.50%	25926
30	12.010	408.371	420.381	952.891	532.510	55.88%	25558
31	11.892	408.320	420.212	951.987	531.775	55.86%	26156
32	12.149	415.002	427.151	952.801	525.650	55.17%	25137
		Total M-P, s	9657.193		Average Load	68.28%	

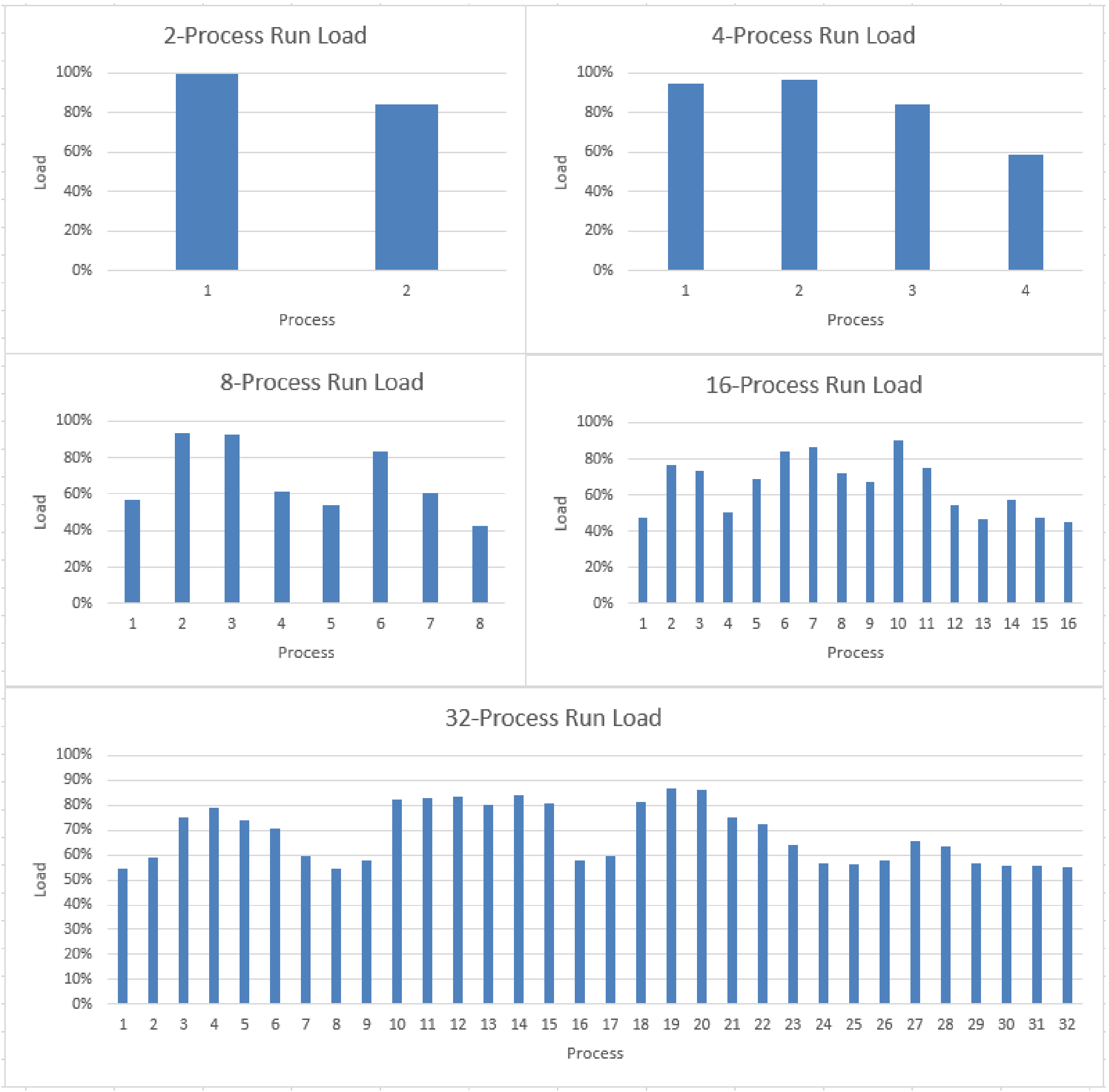


Figure 23. Load charts

There is an obvious trend that the total computation time decreases as process number increases. It is also observed that the parallel jobs with relatively small numbers of processes have a better load balance status than the ones with more processes. Besides, it is easy to note that processes corresponding to partitions far away from partition 1 have lower loads than processes with partitions closer to the first process. The reason is that the first partition is always assigned to the master process (process 1). Communications between the master process and faraway processes are larger and faraway processes need more time for communications and their non-idle time is consequently decreased.

For the 2D decomposition partitioning strategy, all subdomains have the same shape and size. This partitioning fashion reflects an interesting phenomenon that processes corresponding to those partitions in the central area have higher loads. As for those processes corresponding to partitions at the boundary of the reservoir model, their loads are generally lower. For example, in the 32-process parallel case, process 1, process 8, process 9, process 16, process 17, process 24, process 25, and process 32 are the processes at the left boundary and the right boundary. Loads on these processes are the lowest. Inversely, process 4, process 11, process 12, process 13, process 19, process 20 and several other processes adjacent to these processes have very high loads. All these processes are among the central area of the reservoir model. This phenomenon can be explained by the fact that communications required by processes at the center are less than those needed by the processes at the boundaries.

In work distribution tables, it is noticed that the percentage of total run of each section is different from what observed from the original serial run on a single process. The time percentage of PVT properties computation is significantly reduced in parallel mode. This reduction is particularly obvious for the 16-process parallel simulation. In the 16-process parallel job, PVT properties computation only takes about 2% of total runtime, while it takes 47% of total runtime in the serial run. The reason of this reduction is that PVT calculations are local. For each grid block, the calculation of properties related to pressure, volume, and temperature can be conducted in the block without the need of communications with data from neighboring blocks. Thus, parallel computation performance enhancement is especially efficient for PVT computation part. In contrast, the time percentage of solver increases. This is justified by the fact that solver needs data from all subdomains and each process must wait until data from all the other processes are ready.

Message passing data and average load of all processes are also discussed. From the data provided, it is noticed that total message passing time increase as process number increases. The total message passing time here is defined as the sum of all processes' communication time. This time can be larger than the total parallel simulation time since communication time on each process is added and some of communication time are added repetitively. However, this time is only used to reflect the overheads cost of a parallel implementation. Besides, the average load decreases as process number increases. This is related to the increases communications as process number goes up. In fact, communications are related to the way partitions are assigned to processes. The relationship between communications and grid-to-process allocation will be discussed later in details.

The speedup and efficiency of these parallel jobs are also studied. These two concepts are key parameters evaluating a parallel implementation. Equation (4.1) and Equation (4.2) show the computation for speedup and efficiency for a p-process job respectively. **Figure 24** shows the speedup and efficiency's relationship with process numbers. **Figure 25** shows elapsed time's relationship with process numbers. The related data are in **Table 18**. The potential of speedup and efficiency improvement is limited by the Amdahl's Law in Equation (4.3) and (4.4). The serial portion of the code implies the maximum speedup and efficiency a parallel implementation could obtain.

$$S_p = \frac{T_{Serial}}{T_{Parallel}} \tag{4.1}$$

$$E_p = \frac{S_p}{p} \tag{4.2}$$

$$S_p = \frac{p}{sp+(1-s)} \tag{4.3}$$

$$E_p = \frac{1}{sp+(1-s)} \tag{4.4}$$

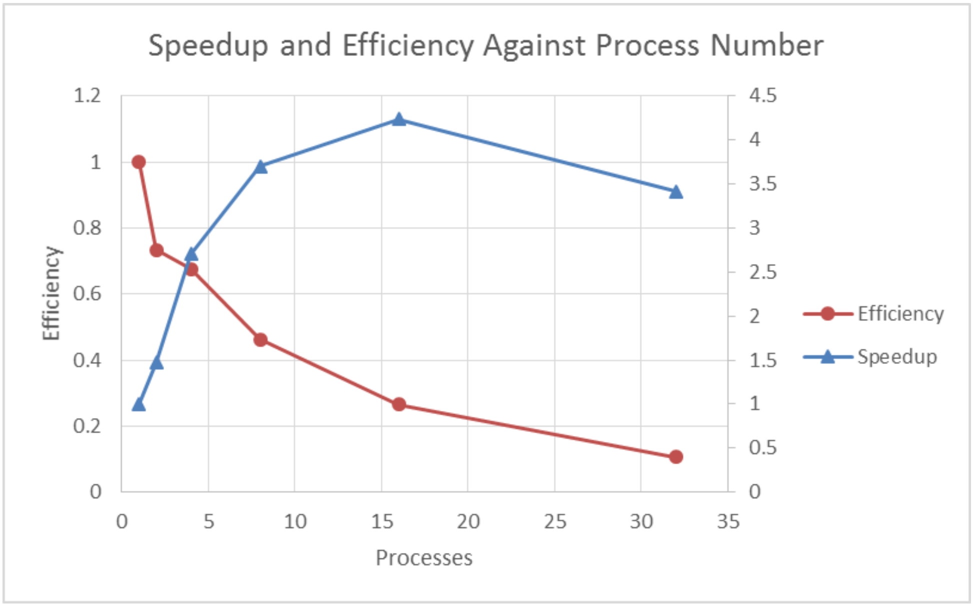


Figure 24. Speedup and efficiency against processes

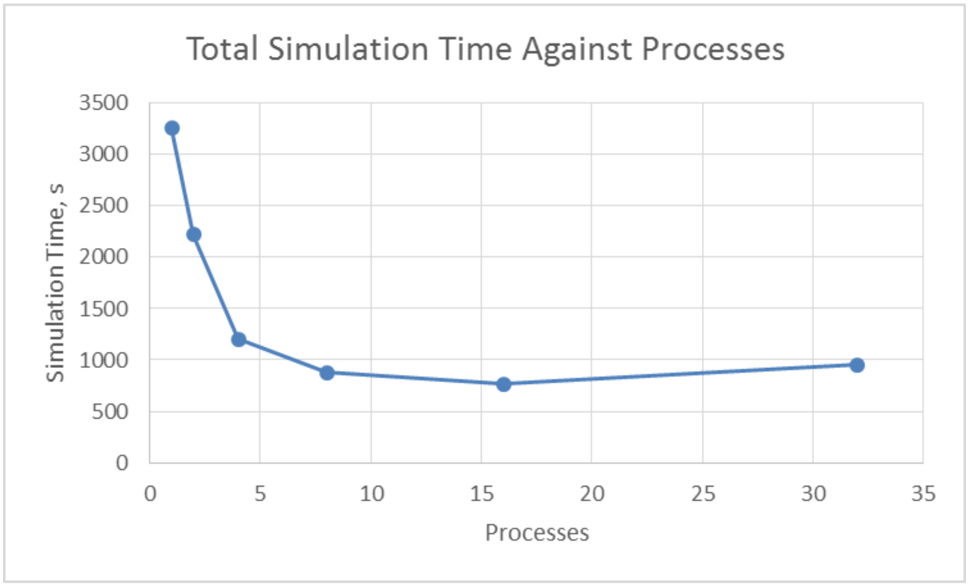


Figure 25. Elapsed time against process number

Table 18. Elapsed time and speedup data

Processes	Simulation Time, s	Efficiency	Speedup
1	3258.248	1	1
2	2221.893	0.733214	1.466429
4	1205.664	0.675613	2.702451
8	881.122	0.46223	3.69784
16	768.705	0.264914	4.238619
32	952.891	0.106854	3.419329

The two figures are very representative of the performance enhancement by 2D decomposition. The parallel job with 16 processes has the greatest speedup of 4.23. It is also noted that the 32-process job is not the faster parallel implementation although it utilizes 32 processes. This is also substantiated by efficiency data: the 32-process job has the lowest efficiency. Although the speedup is not the optimum, the 2-process job has the highest efficiency of 0.73 in all the parallel implementations. Generally speaking, the

8-process implementation is the optimum choice because it has a very satisfactory speedup and an acceptable efficiency. However, if the selection is based primarily on efficiency, which is closely related to hardware costs, one should use 2-process or 4-process parallel job.

It can be concluded for 2D decomposition that a large number of processes does not guarantee a good speedup. In addition, as process number increases, the efficiency always decreases. This fact requires us to find the balance between speedup and efficiency when we are implementing 2D decomposition strategy.

4.2 Results of Metis partitioning

Parallel jobs with Metis partitioning are run for partitions of 2, 4, 8, 16, and 32. Work distributions in terms of computation time and process load distributions are given in tables. **Table 19** and **Table 20** record the 2-process parallel run and the 16-process parallel run's simulator subsection time distributions on a selected node. **Table 21**, **Table 22**, **Table 23**, **Table 24**, and **Table 25** record processes' load distribution for Metis partitioning's all parallel run results. Again, Table 19 and Table 20's process running time is only used to present simulator's subsection time distribution and has nothing to do with the data recorded from Table 21 to Table 25.

Table 19. Process 2's work distribution for 2 Metis partitions

2 processes	CPU Seconds	% Total Run	Elapsed Seconds	% Total Run
INPUT	2.298	0.11	2.298	0.11
OUTPUT	7.759	0.37	7.536	0.36
INITIALIZATION	81.326	3.84	81.277	3.84
PVT PROPERTIES	919.107	43.45	918.587	43.45
ROCK PROPERTIES	21.298	1.01	21.281	1.01
EQUATION SETUP	49.837	2.36	49.67	2.35
NETWORK/WELLS	4.897	0.23	4.838	0.23
SOLVER	818.175	38.68	817.635	38.67
UPDATE	115.49	5.46	115.683	5.47
MISC.	95.179	4.49	95.353	4.51
Total Run	2115.366	100	2114.158	100

Table 20. Process 9's work distribution for 16 Metis partitions

16 processes	CPU Seconds	% Total Run	Elapsed Seconds	% Total Run
INPUT	2.446	0.23	2.466	0.23
OUTPUT	4.556	0.43	4.314	0.4
INITIALIZATION	18.879	1.78	19	1.78
PVT PROPERTIES	134.649	12.7	135.793	12.69
ROCK PROPERTIES	3.772	0.36	3.788	0.35
EQUATION SETUP	25.703	2.42	25.7	2.4
NETWORK/WELLS	6.595	0.62	6.852	0.64
SOLVER	558.695	52.69	563.955	52.72
UPDATE	232.527	21.93	234.648	21.93
MISC.	72.618	6.84	73.28	6.85
Total Run	1060.44	100	1069.796	99.99

Table 21. Load distribution among 2-process Metis partitioning

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	28.152	495.410	523.562	2120.178	1596.616	75.31%	351359
2	3.683	5.119	8.802	2120.518	2111.716	99.58%	499012
		Total M-P, s	532.364		Average Load	87.45%	

Table 22. Load distribution among 4-process Metis partitioning

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	27.563	635.222	662.785	1450.938	788.153	54.32%	130358
2	12.994	197.719	210.713	1451.254	1240.541	85.48%	218462
3	4.443	211.251	215.694	1451.245	1235.551	85.14%	274502
4	9.751	40.261	50.012	1451.249	1401.237	96.55%	227049
		Total M-P, s	1139.204		Average Load	80.37%	

Table 23. Load distribution among 8-process Metis partitioning

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	25.678	544.762	570.440	1162.402	591.962	50.93%	58771
2	15.631	422.993	438.624	1162.879	724.255	62.28%	132192
3	15.308	222.627	237.935	1162.876	924.941	79.54%	123062
4	13.031	483.550	496.581	1162.879	666.298	57.30%	149655
5	30.154	548.086	578.240	1162.877	584.637	50.28%	48194
6	20.794	498.900	519.694	1162.878	643.184	55.31%	101344
7	28.043	496.202	524.245	1162.880	638.635	54.92%	54497
8	3.369	13.448	16.817	1162.881	1146.064	98.55%	182656
		Total M-P, s	3382.576		Average Load	63.64%	

Table 24. Load distribution among 16-process Metis partitioning

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	4.585	484.646	489.231	1033.339	544.108	52.66%	87158
2	4.708	19.250	23.958	1034.141	1010.183	97.68%	87382
3	13.677	383.641	397.318	1034.139	636.821	61.58%	55059
4	19.876	562.792	582.668	1034.139	451.471	43.66%	22779
5	11.222	352.302	363.524	1034.14	670.616	64.85%	68153
6	8.779	563.235	572.014	1034.141	462.127	44.69%	88183
7	15.205	389.849	405.054	1034.136	629.082	60.83%	46263
8	16.308	613.945	630.253	1034.135	403.882	39.06%	46767
9	8.053	445.914	453.967	1034.14	580.173	56.10%	87733
10	11.144	378.503	389.647	1034.143	644.496	62.32%	66297
11	18.197	558.029	576.226	1034.14	457.914	44.28%	35250
12	21.326	662.573	683.899	1034.138	350.239	33.87%	18291
13	22.891	738.487	761.378	1034.143	272.765	26.38%	13404
14	17.769	467.752	485.521	1034.146	548.625	53.05%	33895
15	13.328	598.654	611.982	1034.145	422.163	40.82%	62356
16	18.113	486.642	504.755	1034.142	529.387	51.19%	31401
		Total M-P, s	7931.395		Average Load	52.06%	

Table 25. Load distribution among 32-process Metis partitioning

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	12.699	482.735	495.434	1252.406	756.972	60.44%	36083
2	23.244	467.427	490.671	1259.861	769.190	61.05%	24403
3	20.947	558.810	579.757	1257.874	678.117	53.91%	35711
4	28.375	764.790	793.165	1259.996	466.831	37.05%	13134
5	17.547	468.911	486.458	1257.359	770.901	61.31%	44707
6	17.769	647.346	665.115	1260.043	594.928	47.21%	46410
7	21.874	416.823	438.697	1257.254	818.557	65.11%	29377
8	27.603	695.661	723.264	1260.193	536.929	42.61%	15156
9	22.504	431.778	454.282	1257.167	802.885	63.86%	27303
10	27.430	616.354	643.784	1260.3	616.516	48.92%	13448
11	24.307	598.002	622.309	1256.829	634.520	50.49%	23335
12	19.255	629.533	648.788	1259.276	610.488	48.48%	42802
13	30.791	806.180	836.971	1257.772	420.801	33.46%	5427
14	30.767	821.406	852.173	1259.813	407.640	32.36%	5371
15	30.056	795.328	825.384	1257.304	431.920	34.35%	7896
16	30.184	782.532	812.716	1259.119	446.403	35.45%	7252
17	18.979	428.448	447.427	1259.882	812.455	64.49%	39864
18	26.567	654.869	681.436	1259.486	578.050	45.90%	17263
19	25.406	631.320	656.726	1256.277	599.551	47.72%	20980
20	28.504	684.098	712.602	1260.613	548.011	43.47%	10546
21	29.444	739.428	768.872	1259.318	490.446	38.95%	7883
22	28.153	697.175	725.328	1260.459	535.131	42.46%	11093
23	21.064	453.248	474.312	1255.854	781.542	62.23%	29210
24	25.737	598.467	624.204	1260.324	636.120	50.47%	16816
25	8.234	25.231	33.465	1259.821	1226.356	97.34%	67913
26	26.574	656.842	683.416	1260.177	576.761	45.77%	17707
27	16.010	381.096	397.106	1260.162	863.056	68.49%	48091
28	17.857	649.871	667.728	1259.744	592.016	46.99%	46126
29	16.211	655.613	671.824	1260.103	588.279	46.68%	53906
30	20.949	680.175	701.124	1260.577	559.453	44.38%	35580
31	27.800	765.316	793.116	1260.334	467.218	37.07%	15126
32	22.121	709.055	731.176	1260.673	529.497	42.00%	34452
		Total M-P, s	20138.830		Average Load	50.01%	

Load imbalance and system overheads due to inter-process communications are observed for Metis partitioning parallel implementation. For the simplest parallel implementation of the 2-process case, the total message passing time of the two processes is 532,364 seconds. This value jumps to 20138 seconds for the 32-process parallel run, indicating that the system overheads are much larger for the parallel run with many processes. An explanation to this increase in message passing is that as partition number increases, the edge cuts also increase. The 32-process has much more edge cuts than the 2-process case and it contributes to the increased inter-process communications. Besides, the tables above point out that as processes increase, the average load decreases. The 2-process parallel case and the 4-process parallel case have average load of above 80% while the rest only have average load of 50%-60%.

Load imbalance status and each process's corresponding grid block numbers for the Metis partitioning strategy is depicted in **Figure 26**. In this figure, each process's load and assigned grid block numbers are both presented the same time. The red column stands for the load and the green column stands for the grid block numbers (cells). Unlike the 2D decomposition discussed in the previous section, partitions obtained by Metis do not have equal size. As a result, it is important to show the load and the corresponding grid block number together. Generally speaking, a process with more grid blocks tends to have a larger load. However, this is not the case in every process. The master process does not have a very large load in all 5 cases because the first partition is usually assigned a small number of grid blocks and the master process needs to remain idle to wait for a large amount of information from other processes. There is also a trend that processes with grid blocks at the center of the reservoir model have large loads. For example, in the 32-process parallel implementation, process 13 and process 14 only have less than 10000 active blocks, which is very small compared with other processes, and their loads are around the same level of adjacent processes' loads.

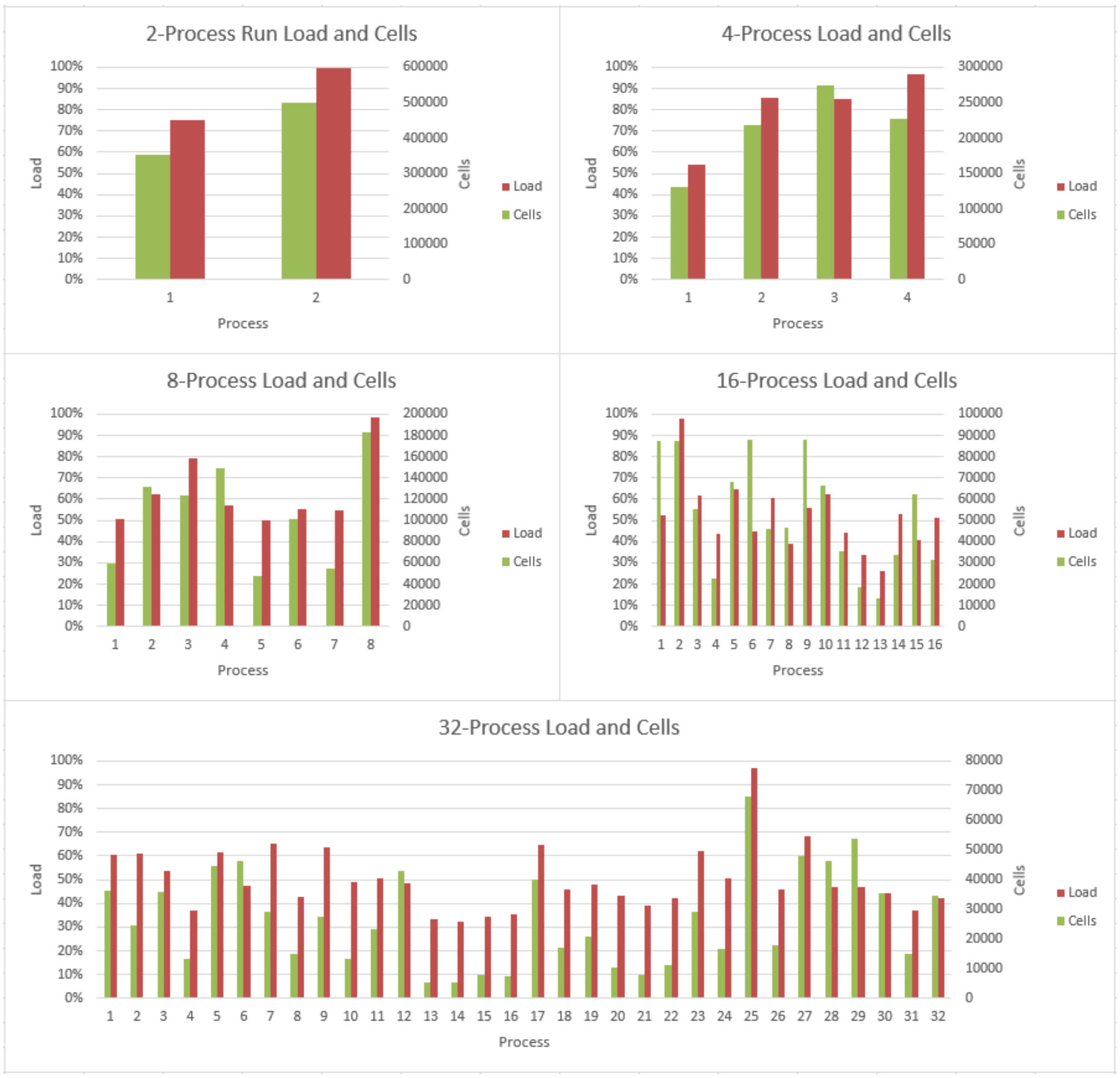


Figure 26. Load imbalance status for Metis partitioning parallel run

An analysis of speedup and efficiency is conducted for Metis partitioning strategy. **Figure 27** presents the speedup and efficiency. **Figure 28** shows the computation time.

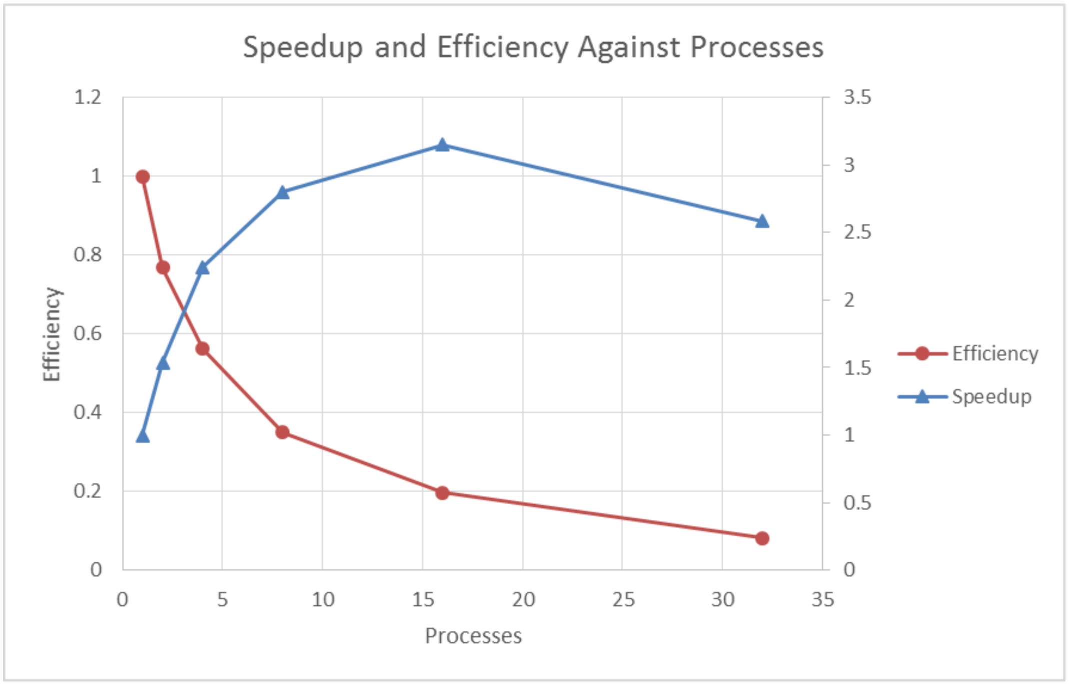


Figure 27. Speedup and efficiency versus processes

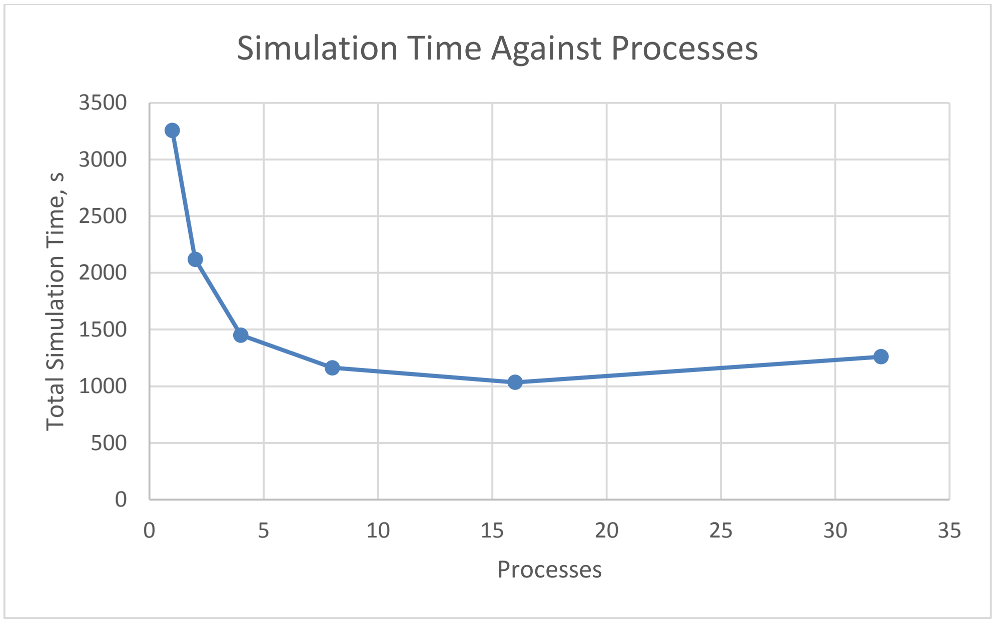


Figure 28. Time versus processes

Results have proved that Metis partitioning has very good scalability in this case. Although the increase of processes to more than 16 did not improve speedup, when processes are less or equal than 16, more processes result in better speedup. Besides, as processes increase, the efficiency decreases. 16 processes give the fastest simulation, but the efficiency related to it is the second lowest. 4-process and 8-process are slower than 16-process parallel job, but they generally have higher efficiency and acceptable speedup.

4.3 Results of spectral partitioning

This section records the parallel job outputs with spectral partitioning. **Table 26** to **Table 30** show the load imbalance status for 2, 4, 8, 16, and 32-process cases.

Table 26. Load distribution among 2-process spectral partitioning

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	0.133	2.634	2.767	2424.721	2421.954	99.89%	590072
2	62.036	800.957	862.993	2425.143	1562.150	64.41%	260299
		Total M-P, s	865.760		Average Load	82.15%	

Table 27. Load distribution among 4-process spectral partitioning

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	0.187	4.158	4.345	1672.355	1668.010	99.74%	301303
2	32.073	641.930	674.003	1672.678	998.675	59.71%	155851
3	11.220	560.397	571.617	1672.679	1101.062	65.83%	288769
4	41.580	707.041	748.621	1672.680	924.059	55.24%	104448
		Total M-P, s	1998.586		Average Load	70.13%	

Table 28. Load distribution among 8-process spectral partitioning

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	0.309	18.989	19.298	1185.553	1166.255	98.37%	174778
2	21.244	526.247	547.491	1186.041	638.550	53.84%	91389
3	6.870	188.381	195.251	1186.048	990.797	83.54%	164850
4	23.053	338.791	361.844	1186.044	824.200	69.49%	69234
5	13.379	268.434	281.813	1186.045	904.232	76.24%	126525
6	24.871	465.175	490.046	1186.048	696.002	58.68%	64462
7	16.488	695.391	711.879	1186.048	474.169	39.98%	123919
8	32.104	744.784	776.888	1186.047	409.159	34.50%	35214
		Total M-P, s	3384.510		Average Load	64.33%	

Table 29. Load distribution among 16-process spectral partitioning

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	3.755	335.181	338.936	962.799	623.863	64.80%	87593
2	12.951	437.306	450.257	963.609	513.352	53.27%	59982
3	5.382	265.628	271.010	963.626	692.616	71.88%	96754
4	13.099	225.458	238.557	963.632	725.075	75.24%	49598
5	8.596	129.501	138.097	963.639	825.542	85.67%	72893
6	17.137	400.281	417.418	963.637	546.219	56.68%	36011
7	8.119	469.122	477.241	963.639	486.398	50.48%	88740
8	19.818	500.916	520.734	963.63	442.896	45.96%	25195
9	4.991	49.278	54.269	963.642	909.373	94.37%	87185
10	18.214	474.588	492.802	963.643	470.841	48.86%	31407
11	10.581	289.570	300.151	963.643	663.492	68.85%	68096
12	20.062	518.355	538.417	963.64	425.223	44.13%	19636
13	14.267	499.636	513.903	963.64	449.737	46.67%	53632
14	17.892	446.532	464.424	963.643	499.219	51.81%	28451
15	18.031	587.058	605.089	963.64	358.551	37.21%	35179
16	22.987	660.536	683.523	963.639	280.116	29.07%	10019
		Total M-P, s	6504.828		Average Load	57.81%	

Table 30. Load distribution among 32-process spectral partitioning

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	9.516	578.772	588.288	1232.565	644.277	52.27%	43400
2	18.397	604.508	622.905	1238.894	615.989	49.72%	39319
3	13.463	290.664	304.127	1233.355	929.228	75.34%	51786
4	20.001	365.931	385.932	1240.001	854.069	68.88%	27992
5	8.951	54.316	63.267	1233.802	1170.535	94.87%	60434
6	20.834	457.488	478.322	1236.552	758.230	61.32%	29138
7	14.104	593.693	607.797	1238.876	631.079	50.94%	57778
8	26.946	700.090	727.036	1240.277	513.241	41.38%	12230
9	8.399	78.183	86.582	1232.611	1146.029	92.98%	62575
10	23.656	489.828	513.484	1235.233	721.749	58.43%	19982
11	22.195	624.747	646.942	1233.999	587.057	47.57%	27506
12	25.665	612.759	638.424	1236.681	598.257	48.38%	12957
13	17.822	562.201	580.023	1236.8	656.777	53.10%	40999
14	23.532	535.502	559.034	1237.888	678.854	54.84%	18265
15	23.699	685.831	709.530	1234.189	524.659	42.51%	23404
16	29.317	792.045	821.362	1235.447	414.085	33.52%	5293
17	16.549	499.276	515.825	1236.761	720.936	58.29%	44193
18	23.932	614.135	638.067	1238.741	600.674	48.49%	20663
19	17.135	658.085	675.220	1238.331	563.111	45.47%	44968
20	22.408	536.261	558.669	1240.933	682.264	54.98%	21606
21	26.860	744.339	771.199	1238.405	467.206	37.73%	12459
22	28.246	717.123	745.369	1240.687	495.318	39.92%	6873
23	21.537	690.097	711.634	1237.749	526.115	42.51%	30962
24	26.480	647.928	674.408	1241.063	566.655	45.66%	12965
25	22.137	532.556	554.693	1237.141	682.448	55.16%	24610
26	27.175	747.299	774.474	1240.614	466.140	37.57%	11425
27	16.931	348.227	365.158	1238.55	873.392	70.52%	40590
28	28.260	720.307	748.567	1240.229	491.662	39.64%	6679
29	26.627	711.925	738.552	1238.724	500.172	40.38%	12633
30	26.835	677.722	704.557	1241.044	536.487	43.23%	10186
31	27.048	759.875	786.923	1237.743	450.820	36.42%	11775
32	29.496	798.342	827.838	1240.931	413.093	33.29%	4726
		Total M-P, s	19124.208		Average Load	51.73%	

From the table above, load imbalance and inter-process data communication can be read. It is noticed that, in all five parallel implementations, the last process always has the lowest load. There are two reasons. The first is that the last spectral partition does not have a very large size. The second reason is that the last process is far away from the master process and the communications take a very long time, which makes the idle time of the last process very large. Another observation is that the total message passing (communications) time increases as processes increase. This is because increased partition number means increased edge cuts. More edge cuts result in more inter-process communications. Besides, there does not exist a clear correlation between a process's grid block number and the same process's load. A process with a large amount of grid blocks is not guaranteed a large load. This is depicted in **Figure 29**. For the 2-process, 4-process, and 8-process spectral parallel implementations, the load on the master process is very large (above 99%).

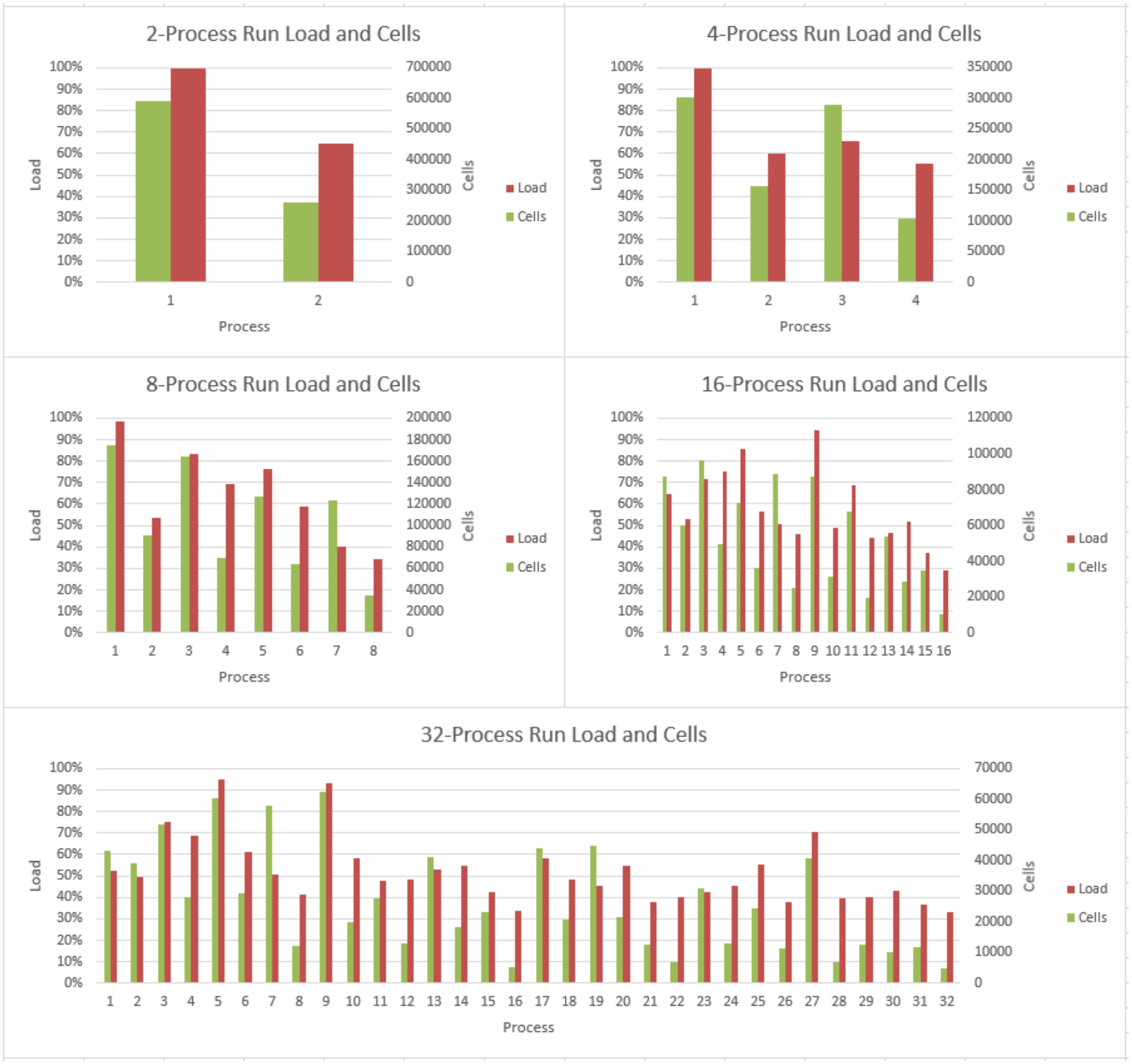


Figure 29. Spectral partitioning load imbalance

Figure 30 and **Figure 31** show spectral partitioning parallel jobs' performance enhancement in terms of speedup, efficiency, and total time. The maximum speedup is achieved by the 16-process parallel run with a speedup of 3.4. Efficiency is decreasing as process number increases. The 32-process parallel implementation is not the fastest because the overheads of a 32-process run are too large.

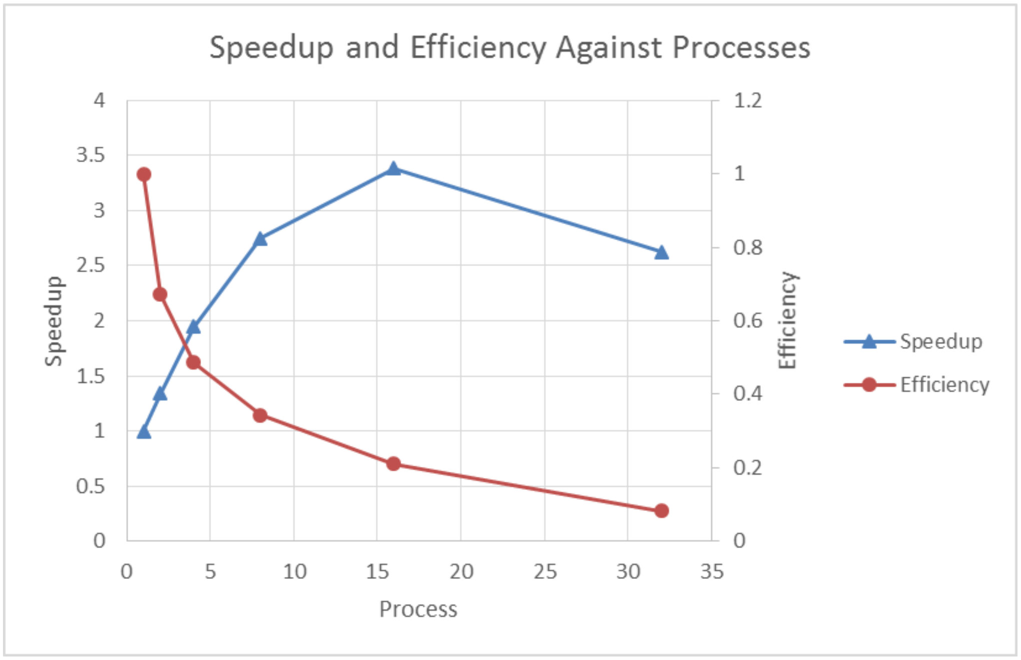


Figure 30. Speedup and efficiency versus processes

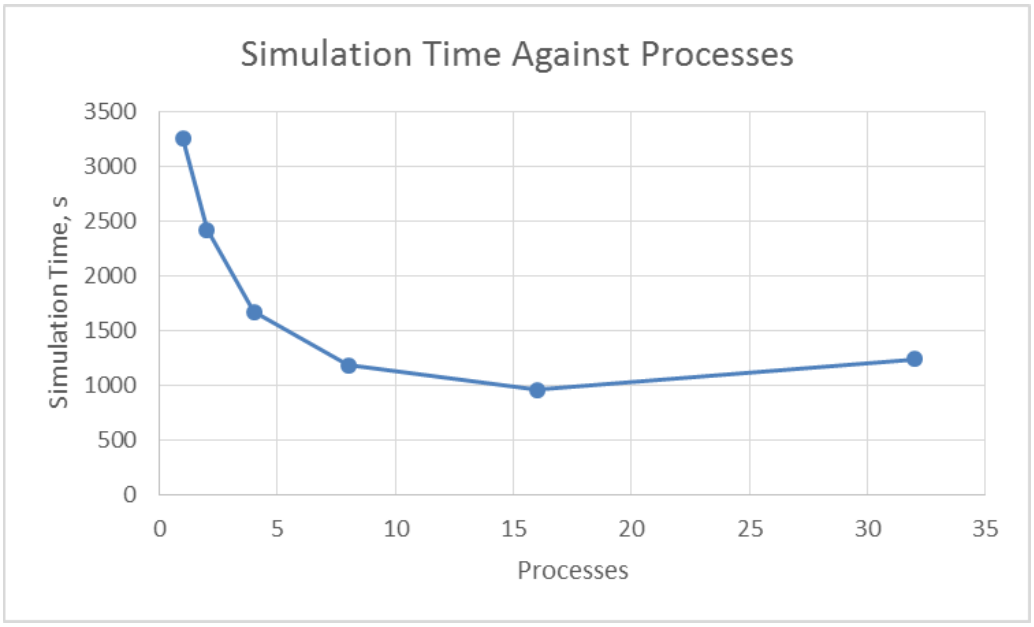


Figure 31. Elapsed time versus processes

4.4 Results of Zoltan partitioning

The implementation the embedded Zoltan partitioner in the simulator is limited due to limited control over its partitioning. First of all, the software arbitrarily assign subdomains to processes and the partitions are solely geometry based. When 32 subdomains are prescribed, Zoltan automatically assigns 7 processes to then and it results in a total simulation time of 2026.794 seconds. When 16 partitions are input as desired partitioning number, Zoltan returns a partitioning result of 5 partitions and the corresponding simulation time is 1501.310 seconds. **Table 31** and **Table 32** show the load imbalance for 5-process and 7-process parallel runs. **Figure 32** is the load imbalance graph. From these results, Zoltan also gives subdomains with similar sizes and the load imbalance is not too strong. However, it does not allow all processes on the parallel machine to be used and a better speedup is not obtainable. In addition, it does not return partition number same as other partitioning methods (2, 4, 8, 16, and 32).

Table 31. 32-subdomain Zoltan partitioning load balance

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	12.149	202.653	214.802	1500.294	1285.492	85.68%	162833
2	18.447	108.084	126.531	1501.306	1374.775	91.57%	150553
3	23.348	457.380	480.728	1036.597	555.869	53.62%	146657
4	8.073	280.403	288.476	1501.307	1212.831	80.79%	222678
5	19.353	429.111	448.464	1501.310	1052.846	70.13%	167650
		Total M-P, s	1559.001		Average Load	76.36%	

Table 32. 16-subdomain Zoltan partitioning load balance

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	10.252	530.626	540.878	2019.714	1478.836	73.22%	130484
2	13.868	24.891	38.759	2024.697	1985.938	98.09%	148970
3	22.683	328.902	351.585	2025.541	1673.956	82.64%	123048
4	28.213	530.505	558.718	2026.703	1467.985	72.43%	110898
5	28.096	631.849	659.945	2024.574	1364.629	67.40%	112465
6	28.157	504.901	533.058	2026.794	1493.736	73.70%	108874
7	24.778	170.307	195.085	2024.388	1829.303	90.36%	115632
		Total M-P, s	2878.028		Average Load	79.69%	

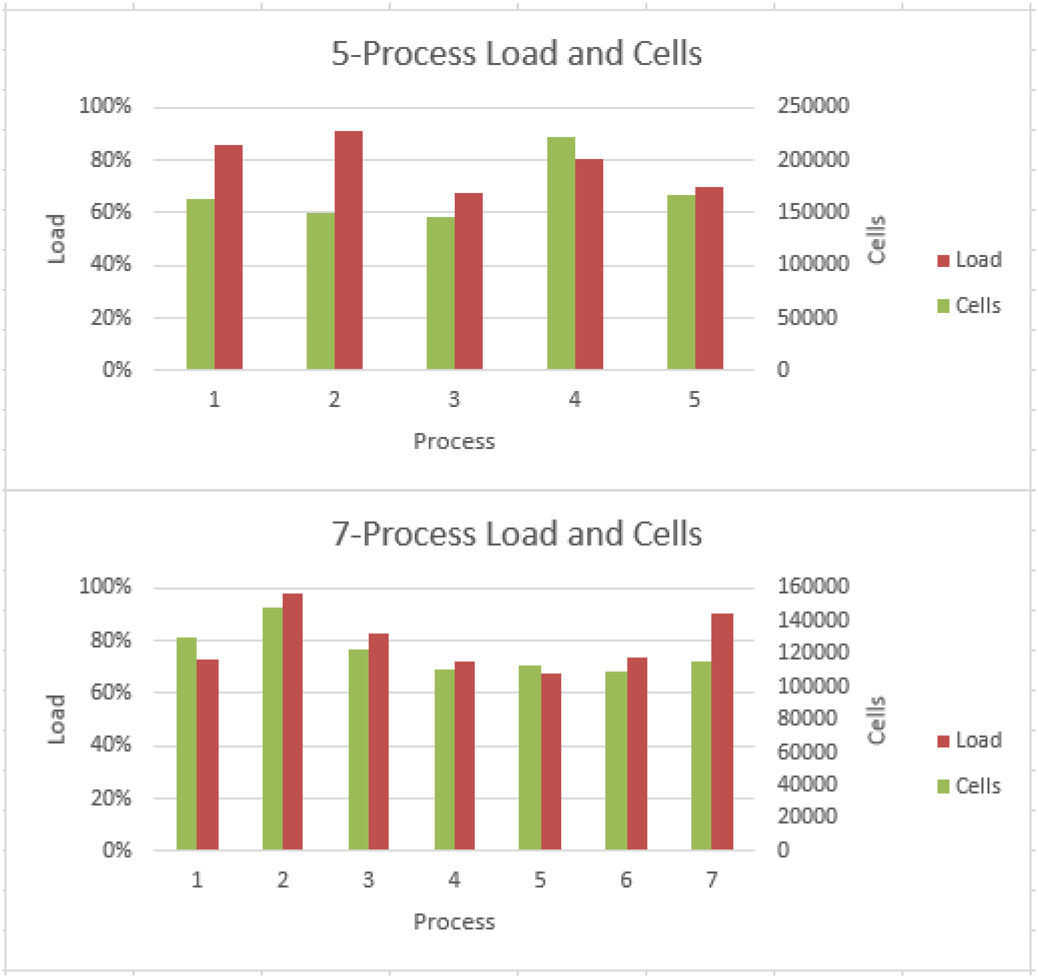


Figure 32. Zoltan load imbalance

4.5 Comparisons

Four graph partitioning strategies are studied. Three of them turn out to be effective partitioning strategies and Zoltan does not allow us to use maximum amount of processes. Based on the results presented in this chapter, the performance of 2D decomposition, spectral partitioning, and Metis partitioning are compared. **Figure 33** shows the comparison of speedup of the three partitioning strategies. **Figure 34** is the comparison of three strategies' efficiency. **Figure 35** is the comparison of simulation time. **Figure 36** is the comparison of the total message passing time.

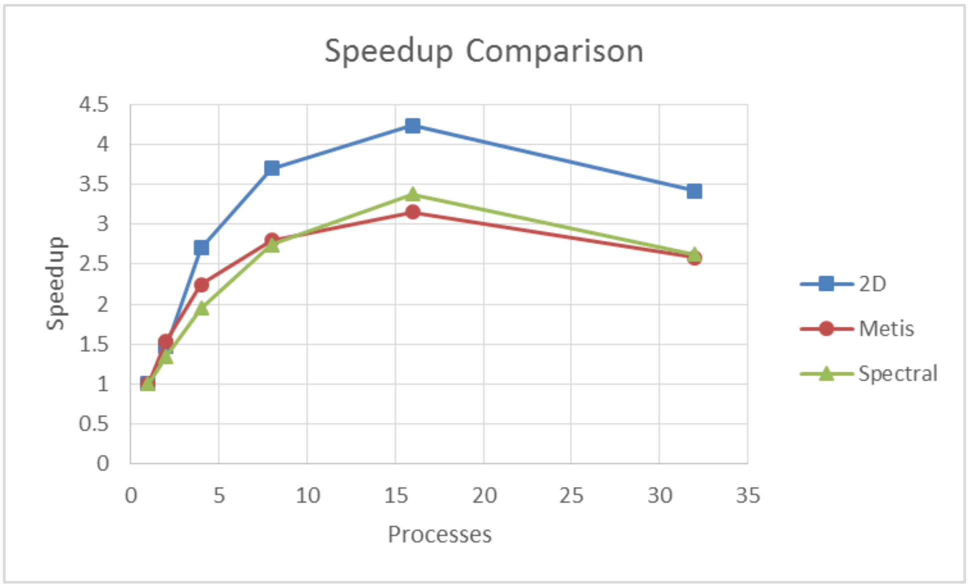


Figure 33. Speedup comparison

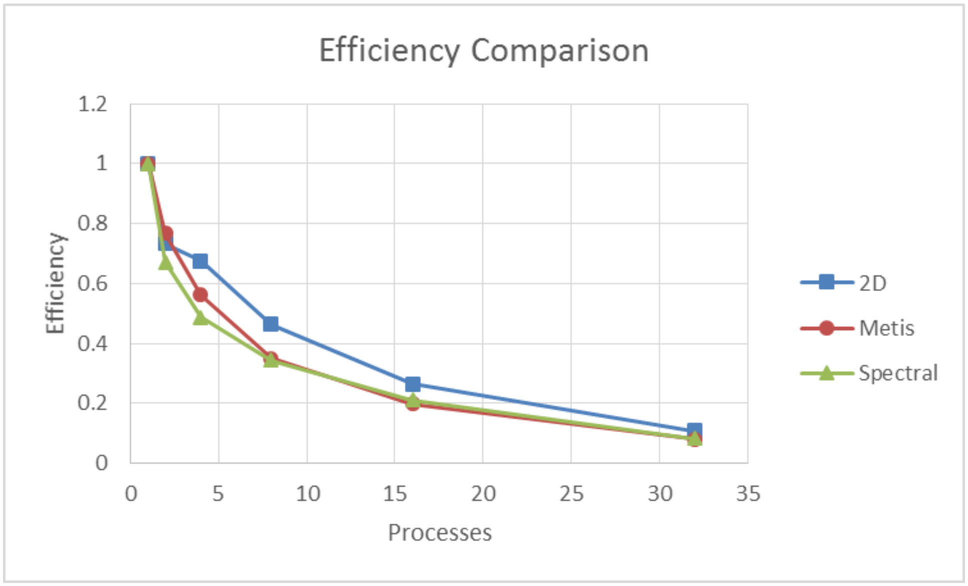


Figure 34. Efficiency comparison

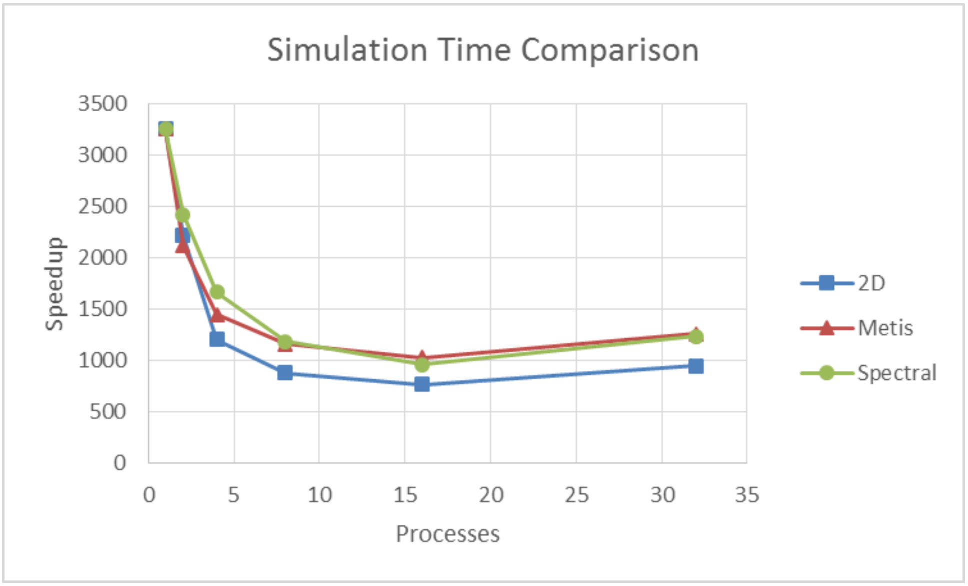


Figure 35. Simulation time comparison

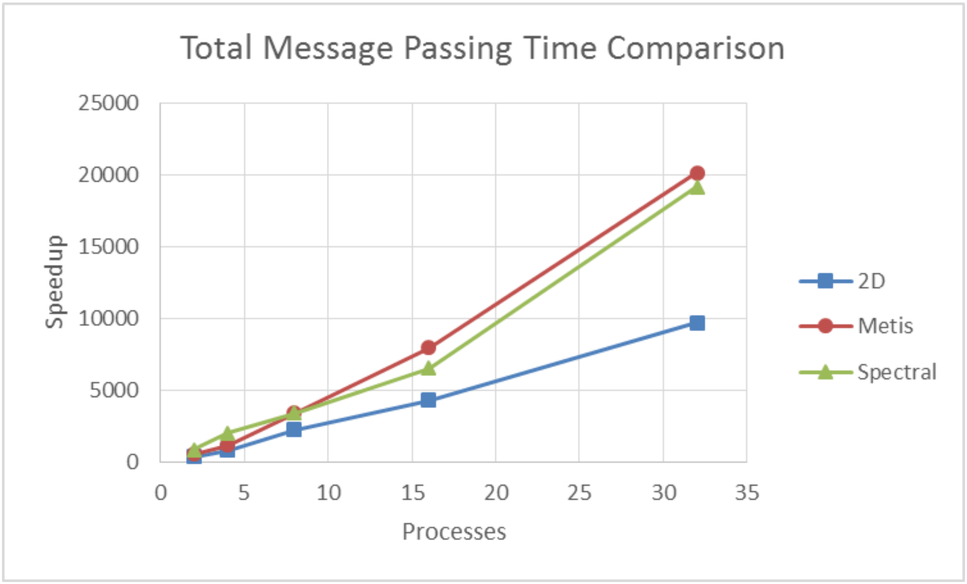


Figure 36. Total message passing time comparison

From these figures, it can be concluded that 2D decomposition gives the fastest parallel performance. The 16-process 2D case reaches a speedup of 4.23 while the other two methods only have the largest speedup of 3.15 and 3.38. 2D decomposition also has the highest efficiency and smallest simulation time. Figure 13 shows that 2D message passing is the smallest. This is especially true for the 32-process cases. For the 32-process cases, the 2D implementation has a total message passing time of 9657 seconds while Metis has 20138 seconds and spectral has 19124 seconds.

Load imbalance comparison is shown in **Figure 37**. In general, 2D decomposition has larger load in processes and the fluctuation is smaller than the other two methods. Also, the range of the load of 2D decomposition is smaller than the other two methods. The 32-process parallel implementation especially presents that 2D decomposition has the largest load on most of the processes.

2D decomposition turns out to be the optimum partitioning methods based on the original reservoir model. Although it does not take into consideration weighting factors, it is capable enough to give a result with relative small load imbalance. For Metis and spectral methods, weighting factors are considered. In this study, the first layer's transmissibility field is used as weighting factors. However, the first layer is not representative of all the 100 layers and this is the reason why weighted spectral and Metis partitioning did not result in a faster parallel implementation performance than the geometric 2D decomposition. Theoretically, an area with a higher transmissibility usually have larger PVT property changes. As a result, more iterations are required to reach convergence, which means that more computation is needed in this area. If weighted Metis and weighted spectral partitioning works well, such areas will be partitioned as subdomains with small size so that a process can simulate for it more efficiently. Weighted Metis and spectral partitioning is also used for a reservoir model with elevated complexity in the next chapter and it managed to reduce the inter-process communications.

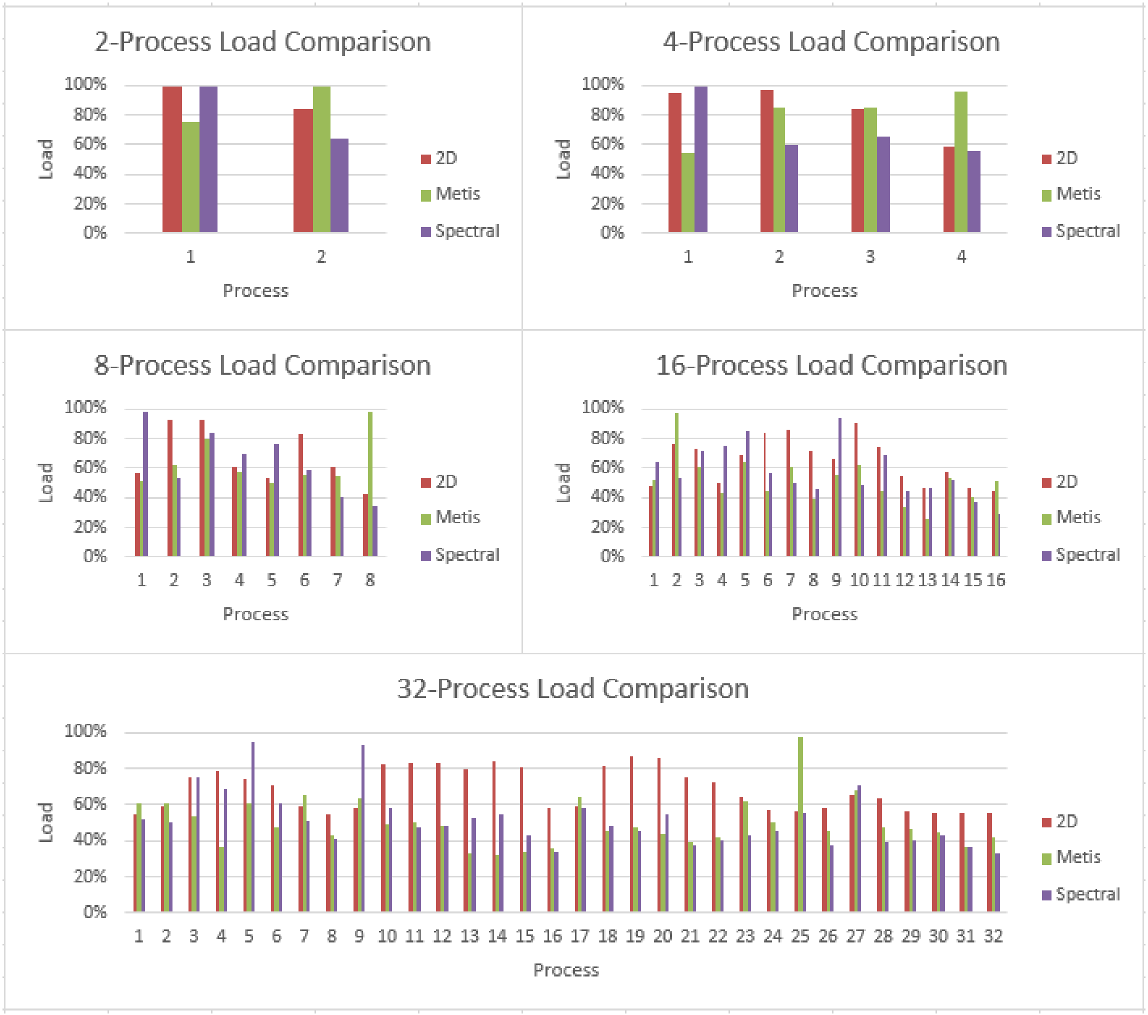


Figure 37. Load imbalance comparison

4.6 Conclusions

This chapter talked about the parallel performance based on several partitioning strategies. In general, all the partitioning strategies can speed up the reservoir simulation and they all run faster than the serial simulation. The scalability is good when process number is under 16 for 2D decomposition, Metis partitioning, and spectral partitioning. When there are more than 8 processes, the scalability is not as good. For cases with 16 processes, the speedup is the largest. However, its corresponding efficiency is very low and the parallel implementation is not preferable if one selects parallel implementation based on efficiency. For cases with 32 processes, the speedup is smaller than 8-process parallel jobs. This is largely due to the communications among processes.

Efficiency of a parallel implementation decreases as process number increases and this is true for all partitioning strategies. When there are more than 8 processes, the efficiency of a single process drops below 50%. The efficiency even drops below 10% when there are 32 processes. All these indicate that one should not go with 32-CPU parallel jobs.

CHAPTER V

ELEVATED RESERVOIR MODEL COMPLEXITY

5.1 Horizontal well

The original model only has vertical wells, whose geometry is relatively simple. Horizontal wells are often introduced to increase hydrocarbon production. It is also widely used in unconventional reservoirs in conjunction with hydraulic fractures. Horizontal wells typical have longer wellbores than vertical wells. The inflow mechanism is also different from vertical wells. In consequence, the complexity of reservoir simulation is increased in cases with reservoir models.

To introduce a new horizontal well, well 1YII0191, originally as a vertical well, is converted to a horizontal well. The horizontal wellbore starts at layer 57 and stretches to the east with a length of 4500 feet. The serial run on the cluster took 3606.828 seconds.

Metis partitioning strategy is applied to the modified model. Since the change of load balance is the priority here, only load balance among processes are studied while work distribution is ignored. **Table 33** to **Table 37** show the load imbalance of the horizontal well model based on Metis partitioning and **Figure 38** show the load imbalance graphically. **Figure 39** is the comparison of load imbalance between the original vertical scenario and the horizontal scenario. **Table 38** records the comparison of simulation time and message passing time (system overheads) between these two scenarios.

Table 33. Load imbalance for 2 Metis partitions

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	28.499	501.275	529.774	2162.151	1632.377	75.50%	351359
2	3.978	5.015	8.993	2162.690	2153.697	99.58%	499012
		Total M-P, s	538.767		Average Load	87.54%	

Table 34. Load imbalance for 4 Metis partitions

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	27.339	641.249	668.588	1482.295	813.707	54.90%	130358
2	12.812	201.242	214.054	1482.639	1268.585	85.56%	218462
3	4.690	214.674	219.364	1482.632	1263.268	85.20%	274502
4	10.470	37.979	48.449	1482.635	1434.186	96.73%	227049
		Total M-P, s	1150.455		Average Load	80.60%	

Table 35. Load imbalance for 8 Metis partitions

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	31.352	638.740	670.092	1317.532	647.440	49.14%	58771
2	20.193	468.046	488.239	1317.522	829.283	62.94%	132192
3	20.153	270.822	290.975	1317.105	1026.130	77.91%	123062
4	17.444	529.104	546.548	1318.706	772.158	58.55%	149655
5	36.389	605.457	641.846	1318.075	676.229	51.30%	48194
6	23.083	537.068	560.151	1318.610	758.459	57.52%	101344
7	32.108	551.543	583.651	1318.664	735.013	55.74%	54497
8	3.723	12.126	15.849	1317.139	1301.290	98.80%	182656
		Total M-P, s	3797.351		Average Load	63.99%	

Table 36. Load imbalance for 16 Metis partitions

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	4.611	484.366	488.977	1056.186	567.209	53.70%	87158
2	4.973	22.367	27.340	1057.285	1029.945	97.41%	87382
3	13.895	385.052	398.947	1057.27	658.323	62.27%	55059
4	20.080	566.243	586.323	1057.242	470.919	44.54%	22779
5	11.446	352.848	364.294	1057.278	692.984	65.54%	68153
6	9.002	567.869	576.871	1057.265	480.394	45.44%	88183
7	15.522	394.111	409.633	1057.252	647.619	61.25%	46263
8	16.542	619.057	635.599	1057.245	421.646	39.88%	46767
9	8.302	450.417	458.719	1057.26	598.541	56.61%	87733
10	11.405	379.972	391.377	1057.26	665.883	62.98%	66297
11	18.445	561.367	579.812	1057.265	477.453	45.16%	35250
12	21.544	666.704	688.248	1057.236	368.988	34.90%	18291
13	23.054	745.696	768.750	1057.218	288.468	27.29%	13404
14	17.983	470.388	488.371	1057.247	568.876	53.81%	33895
15	13.569	605.729	619.298	1057.259	437.961	41.42%	62356
16	18.333	489.092	507.425	1057.247	549.822	52.01%	31401
		Total M-P, s	7989.984		Average Load	52.76%	

Table 37. Load imbalance for 32 Metis partitions

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	12.659	510.762	523.421	1324.459	801.038	60.48%	36083
2	23.146	507.466	530.612	1345.463	814.851	60.56%	24403
3	21.039	599.577	620.616	1345.194	724.578	53.86%	35711
4	28.302	804.116	832.418	1349.721	517.303	38.33%	13134
5	17.517	501.024	518.541	1336.631	818.090	61.21%	44707
6	17.917	679.666	697.583	1336.339	638.756	47.80%	46410
7	21.674	449.970	471.644	1326.412	854.768	64.44%	29377
8	27.569	733.218	760.787	1343.164	582.377	43.36%	15156
9	22.368	467.776	490.144	1330.543	840.399	63.16%	27303
10	27.384	652.405	679.789	1345.465	665.676	49.48%	13448
11	24.045	639.620	663.665	1336.575	672.910	50.35%	23335
12	19.210	668.828	688.038	1343.323	655.285	48.78%	42802
13	30.638	822.130	852.768	1325.92	473.152	35.68%	5427
14	30.805	845.900	876.705	1341.309	464.604	34.64%	5371
15	29.973	823.970	853.943	1339.344	485.401	36.24%	7896
16	29.906	816.056	845.962	1348.027	502.065	37.24%	7252
17	18.672	426.303	444.975	1343.999	899.024	66.89%	39864
18	26.379	654.612	680.991	1348.614	667.623	49.50%	17263
19	25.316	633.824	659.140	1342.655	683.515	50.91%	20980
20	28.308	691.546	719.854	1349.43	629.576	46.65%	10546
21	29.159	747.941	777.100	1346.876	569.776	42.30%	7883
22	28.015	704.222	732.237	1349.891	617.654	45.76%	11093
23	20.786	452.212	472.998	1345.809	872.811	64.85%	29210
24	25.580	600.124	625.704	1348.755	723.051	53.61%	16816
25	8.271	29.000	37.271	1344.826	1307.555	97.23%	67913
26	26.437	651.247	677.684	1344.466	666.782	49.59%	17707
27	15.794	373.871	389.665	1343.292	953.627	70.99%	48091
28	17.660	647.934	665.594	1348.884	683.290	50.66%	46126
29	16.222	652.667	668.889	1345.514	676.625	50.29%	53906
30	20.896	673.750	694.646	1348.388	653.742	48.48%	35580
31	27.661	767.096	794.757	1339.883	545.126	40.68%	15126
32	22.091	706.003	728.094	1347.297	619.203	45.96%	34452
		Total M-P, s	20676.235		Average Load	51.87%	

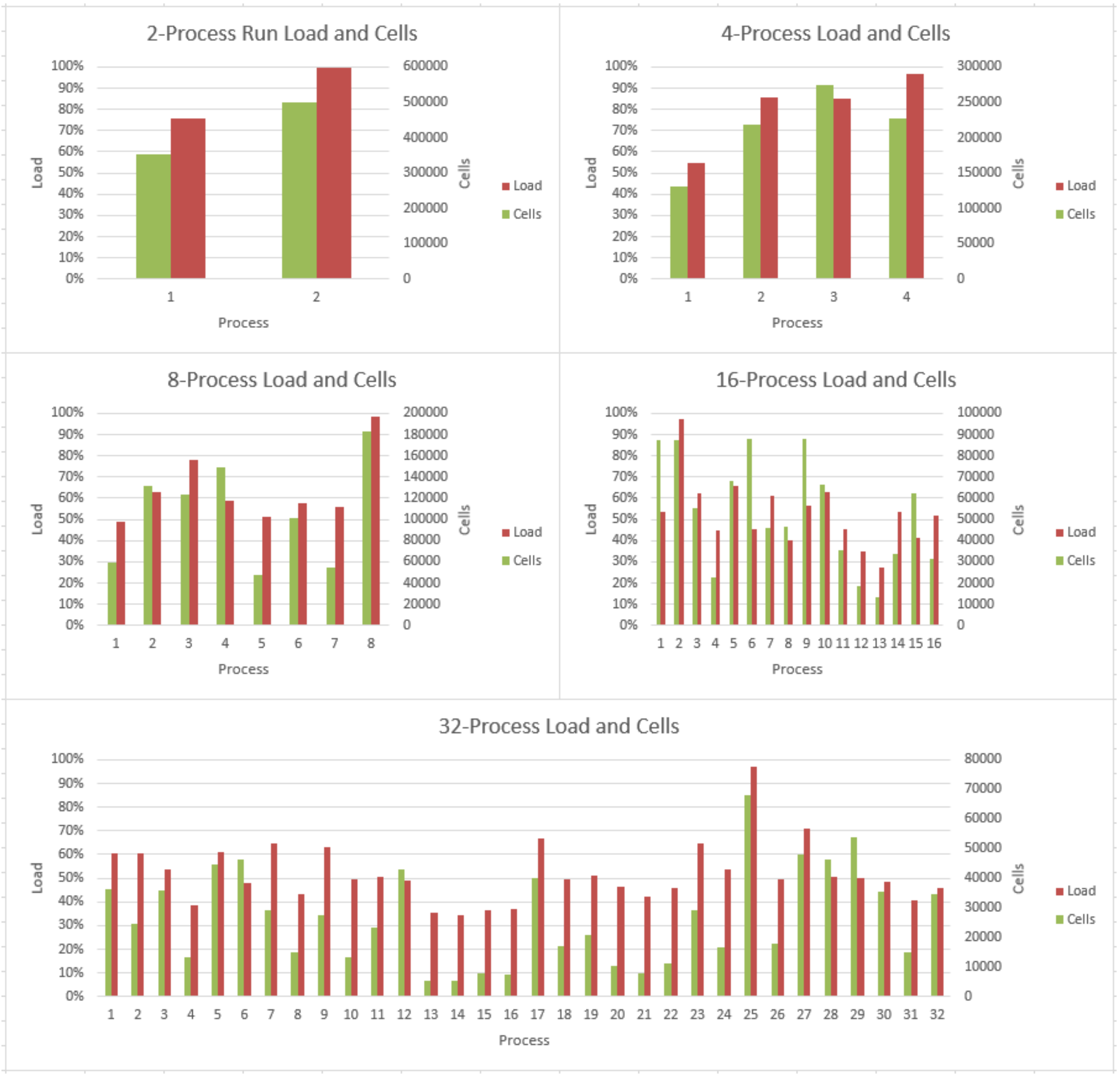


Figure 38. Load imbalance for horizontal well case

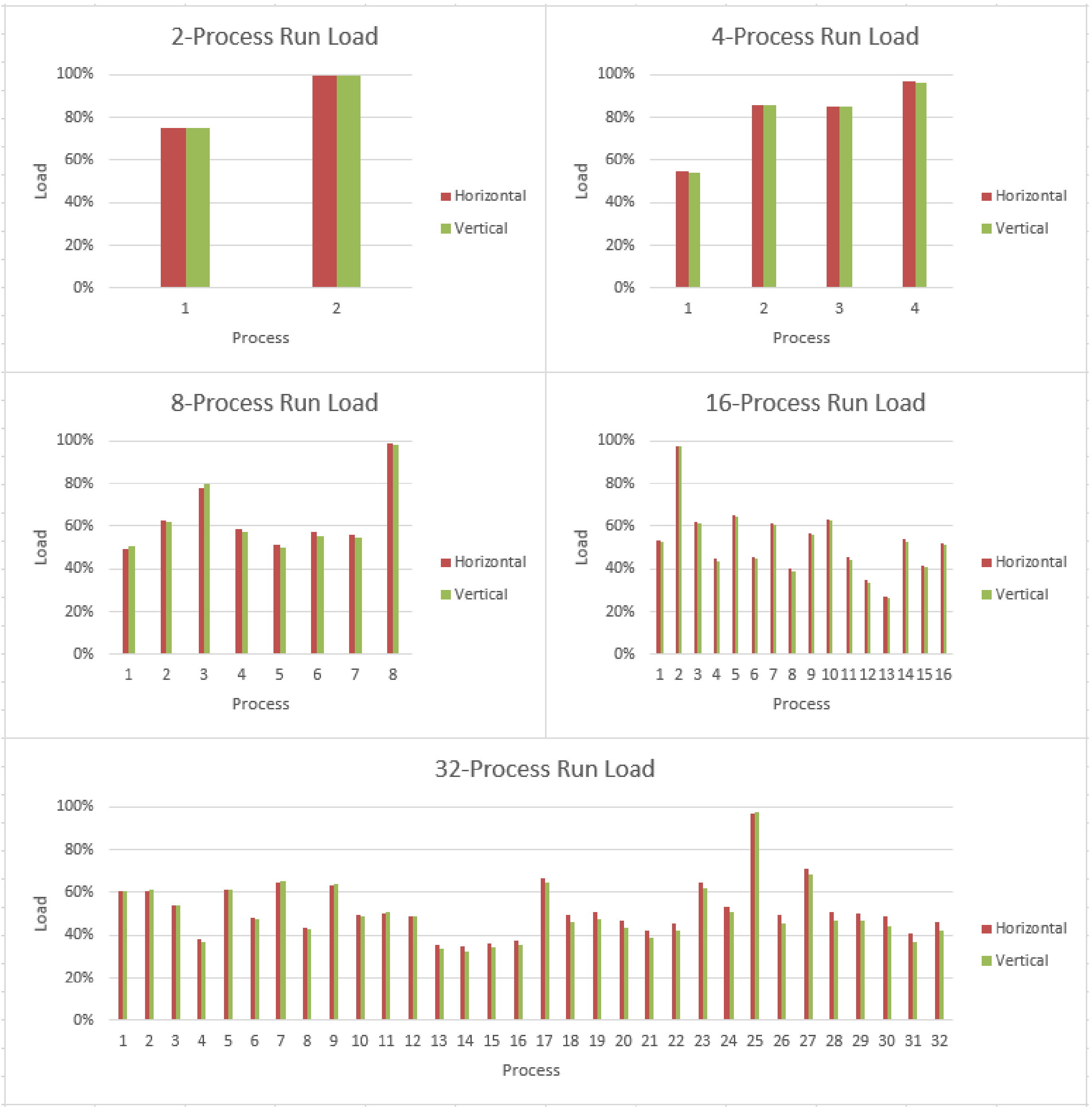


Figure 39. Load comparison between horizontal and vertical cases

Table 38. Simulation time and message passing time

	Simulation Time, s		Message Passing, s	
Processes	Horizontal	Vertical	Horizontal	Vertical
1	3606.828	3258.248	--	--
2	2162.690	2120.518	538.767	532.364
4	1482.639	1451.249	1150.455	1139.204
8	1318.706	1162.881	3797.351	3382.576
16	1057.285	1034.146	7989.984	7931.395
32	1349.891	1260.673	20676.235	20138.830

From these results, it is noted that the load imbalance status is basically the same between the vertical case and the horizontal case. Their difference is nearly negligible. However, the horizontal well case increases the total simulation time, which is true for all 5 parallel implementations. The introduction of horizontal well also increased the communications between processes and the overheads increase. This is substantiated by data provided in Table 38. In this table, it is noticed that for 2-, 4-, 8-, 16-, and 32-process parallel runs, horizontal message passing time is always greater than vertical message passing time. An important reason of the increased overheads is that the horizontal well is across several different partitions and more communications among processes are needed for the horizontal well related computations.

It is concluded from this section that the introduction of horizontal well does not significantly change the load imbalance status in this case. However, it does increase the simulation load and also increase overheads among processes.

5.2 New vertical wells with weighted partitioning strategies

Another scenario with elevated complexity is considered. In this scenario, 10 new wells are added to an area in the reservoir model so that the computation load is increased on purpose in that area. 2D decomposition, weighted spectral partitioning, and weighted Metis partitioning are applied in this scenario to understand their effects on load balancing. For each of the three partitioning strategies, a 4-partition mesh and an 8-partition mesh are generated. In consequence, we have totally 6 cases in this section of study.

The weighting factor here for Metis and spectral is no longer transmissibility. As discussed in the previous chapter, the weighting factor as transmissibility field has it limitation and is not representative of all the 100 layers. If one wants to better represent the reservoir's horizontal transmissibility, a 3D partitioning may be used. By using the 3D partitioning, the reservoir mesh can be portioned vertically and horizontal transmissibility from several different layers can be used as weighting factors. It applied well, it can cancel out the increased overheads caused by increased partitions and ultimately reaches a satisfactory speedup and load balance.

Instead, the location of well is selected to be the weighting factor for Metis and spectral here. A weight is given to each well so that the partitioner can take into account the well's effects on load imbalance. **Figure 40** and **Figure 41** are the weighted partitioning results for 4-partition and 8-partition.

Table 39 to **Table 44** record load imbalance for 2D, Metis, and spectral parallel implementations. **Figure 42** is the load imbalance comparisons of the three partitioning methods.

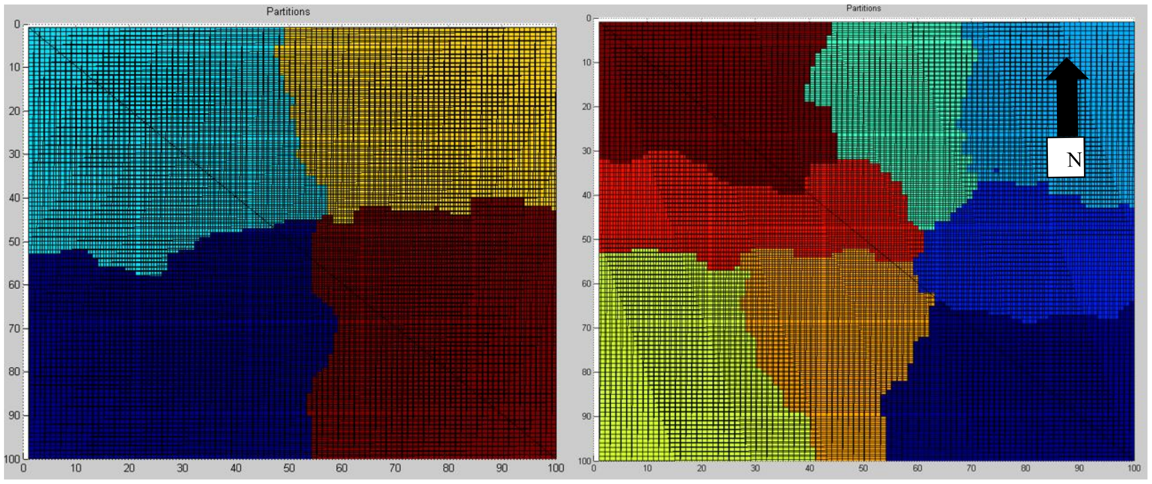


Figure 40. Weighted Metis partitioning

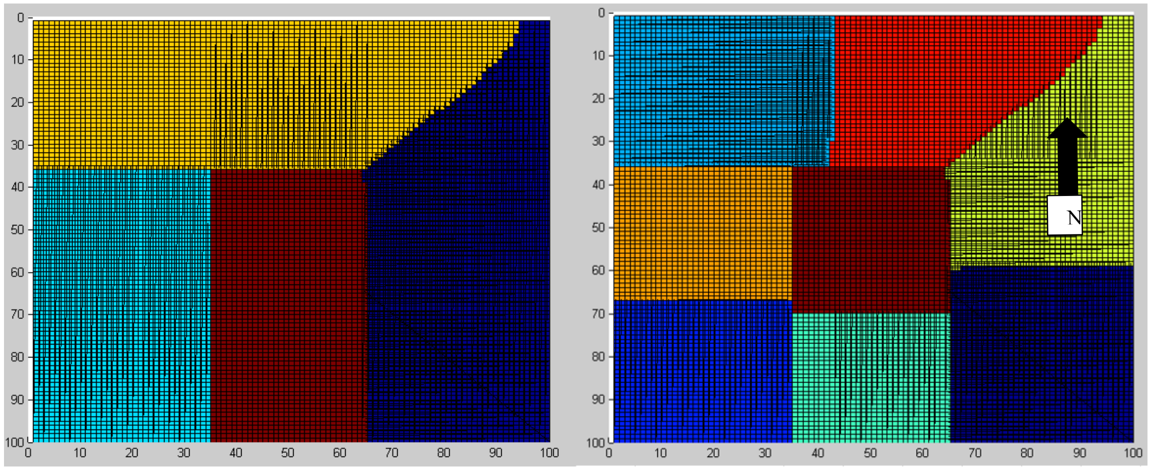


Figure 41. Weighted spectral partitioning

Table 39. 4-process 2D decomposition load imbalance

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	1.997	207.534	209.531	2828.850	2619.319	92.59%	208400
2	4.059	105.532	109.591	2830.061	2720.470	96.13%	215022
3	6.319	415.520	421.839	2830.034	2408.195	85.09%	211407
4	7.404	876.609	884.013	2830.044	1946.031	68.76%	215542
		Total M-P, s	1624.974		Average Load	85.64%	

Table 40. 8-process 2D decomposition load imbalance

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	5.540	629.606	635.146	2142.648	1507.502	70.36%	100918
2	4.112	122.808	126.920	2143.528	2016.608	94.08%	107482
3	4.495	127.917	132.412	2143.536	2011.124	93.82%	106742
4	7.147	495.277	502.424	2143.530	1641.106	76.56%	108280
5	8.656	636.318	644.974	2143.529	1498.555	69.91%	102703
6	5.734	239.763	245.497	2143.532	1898.035	88.55%	108704
7	7.273	579.571	586.844	2143.531	1556.687	72.62%	106969
8	8.159	770.070	778.229	2143.533	1365.304	63.69%	108573
		Total M-P, s	3652.446		Average Load	78.70%	

Table 41. 4-process Metis partitioning load imbalance

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	1.345	244.326	245.671	3078.597	2832.926	92.02%	229036
2	6.185	138.986	145.171	3079.096	2933.925	95.29%	212447
3	13.659	502.227	515.886	3079.093	2563.207	83.25%	176994
4	5.548	606.904	612.452	3079.092	2466.640	80.11%	231894
		Total M-P, s	1519.180		Average Load	87.67%	

Table 42. 8-process Metis partitioning load imbalance

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	1.527	729.235	730.762	2260.895	1530.133	67.68%	131200
2	9.938	312.621	322.559	2262.025	1939.466	85.74%	90582
3	8.089	441.305	449.394	2262.026	1812.632	80.13%	105392
4	9.521	268.304	277.825	2262.031	1984.206	87.72%	87453
5	5.628	410.838	416.466	2262.025	1845.559	81.59%	122605
6	7.005	145.869	152.874	2262.031	2109.157	93.24%	105616
7	8.310	213.220	221.530	2262.033	2040.503	90.21%	93996
8	6.148	296.392	302.540	2262.029	1959.489	86.63%	113527
		Total M-P, s	2873.950		Average Load	84.12%	

Table 43. 4-process spectral partitioning load imbalance

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	1.843	692.880	694.723	3114.931	2420.208	77.70%	258161
2	13.669	368.779	382.448	3115.907	2733.459	87.73%	189926
3	7.024	216.020	223.044	3115.899	2892.855	92.84%	231145
4	15.662	223.377	239.039	3115.906	2876.867	92.33%	171139
		Total M-P, s	1539.254		Average Load	87.65%	

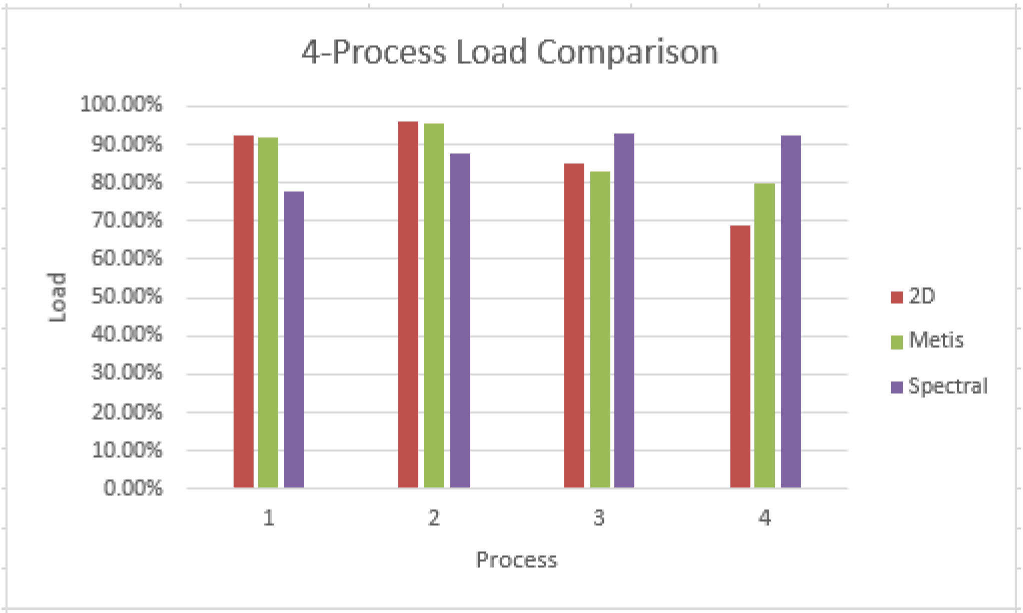
Table 44. 8-process spectral partitioning load imbalance

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	2.849	761.187	764.036	2394.897	1630.861	68.10%	129925
2	10.780	542.959	553.739	2395.685	1841.946	76.89%	97616
3	6.531	353.255	359.786	2395.691	2035.905	84.98%	118437
4	13.257	457.737	470.994	2395.675	1924.681	80.34%	82059
5	5.440	333.189	338.629	2395.674	2057.045	85.86%	128236
6	9.919	250.229	260.148	2395.682	2135.534	89.14%	92310
7	7.281	263.212	270.493	2395.686	2125.193	88.71%	112708
8	9.581	205.286	214.867	2395.679	2180.812	91.03%	89080
		Total M-P, s	3232.692		Average Load	83.13%	

Table 45. Message passing time and average load comparison

Partitioner	Message Passing Time, s		Average Load	
	4-process	8-process	4-process	8-process
2D	1624.974	3652.446	85.64%	78.70%
Metis	1519.18	2873.95	87.67%	84.12%
Spectral	1539.254	3232.692	87.65%	83.13%

Based on these results, it is easy to notice that Metis and spectral decrease the overheads in parallel implementation. Also, Metis and spectral increase the average load. From **Table 45**, Metis has the lowest overheads (message passing time) for both 4-process parallel run and 8-process parallel run. Metis also has the highest average load among all the three partitioners. Comparatively, 2D decomposition does not give the optimum average load or the smallest overheads. The load distribution is better illustrated in **Figure 42**. The comparison column charts also show that Metis and spectral parallel runs have smaller load variations than the 2D decomposition parallel run. The range of 2D decomposition loads is from 68.76% to 96.13%, while the range of Metis loads is 80.11% to 95.29% and the range of spectral loads is 77.70% to 92.33%.



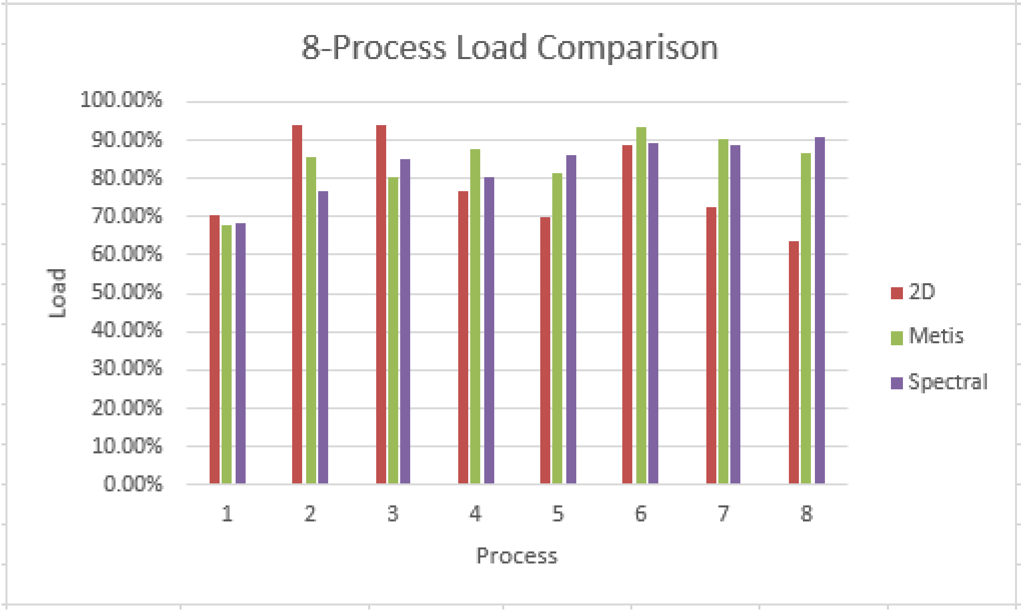


Figure 42. Load imbalance comparison

5.3 Conclusions

This chapter studies two new scenarios: horizontal well and drilling new vertical wells. It is concluded that the introduction of horizontal well both increases the total simulation time and the system overheads. However, horizontal well does not significantly change the load distribution among processes. It is also concluded that Metis and spectral partitioning strategies using well location as weighting factor are capable of reducing overheads and increasing average load on processes. It also decreases the load imbalance among processes.

CHAPTER VI

GRID-TO-PROCESS ASSIGNMENT

6.1 Introduction

After the mesh is partitioned, each partition needs to be assigned to a process so that its corresponding portion of work can be run on this designated process. Intuitively, a partition with small enough size leads to less work load and faster computation. This trend can be justified by looking at the data presented in previous sections. The actually non-idle CPU time is shorter for parallel jobs with more partitions. However, it is also overserved in these cases that the best speedup is not realized in jobs with the most processes (32-process jobs). 8-process and 16-process jobs often result in faster speedups than 32-process jobs. The main reason of the lowered parallel implementation performance is that more partitions bring more communications between processes and it consequently increases system overheads.

From the discussion in previous chapters, the 32-process parallel implementation has the lowest average load and also larger variation in terms of individual process loads. This is largely because of the increased inter-process communications in the 32-process architecture.

Not all parallel jobs have communications. If the computations within a partition does not need data from neighboring or any other partitions, no communications will happen. In this research, PVT related properties are calculated in this fashion. By looking at

Table 3 and **Table 17**, one can notice that the time percentage of PVT property calculations change largely from the serial run to a 32-process parallel run. PVT calculations took 1527.332 seconds in the serial run and it is 47.49% of the total simulation time. After a 32-partition parallel job is used, this time is reduced significantly. Although different processes have slightly different lengths of time, they are close. For example, PVT calculations on process 1 took 16.748 seconds and it is only 2.56% of the total simulation time. These facts show that parallel implementation is capable of bringing in great performance enhancement for PVT property calculations. In other words, effects of communications among processes are irrelevant in this situation.

However, PVT property calculations are just a small part in this simulation. There are many other simulation works that need information communications with other processes. By still looking at the two tables brought up in the previous paragraph, some new information can be found. The time percentage the solver needs actually increases in the 32-process case. The solver took 40.99% of total time in the serial run while it took 62.94% of the time in the 32-process parallel run. This is because during the simulation, each process needs data from other processes so that the solver can be applied to the mesh assigned to the process. It is also noted that in the 32-process parallel run, the total elapsed time on process 1 is 822.747 seconds, which is about 2 times of the same process's actual non-idle CPU time. The negative effect of overhead is especially conspicuous here.

Communications are related to how partitions are assigned to processes. In order to understand this relationship, this chapter studies some scenarios of grid-to-process assignment.

6.2 Overheads of shared-memory and distributed memory

As introduced at the beginning, the cluster has 5 nodes. From its specification, there are totally 72 cores and 144 threads. Within each processor, it has either 4 or 8 cores and these cores are based on shared-memory. Thus, unlike distributed-memory for inter-processor, inner-processor communications are not as time-consuming as those take place across nodes. In the assignment, if equal or less than 8 partitions are assigned to one node, it is guaranteed that the communications among these partitions which belong to the same node are not as much as those which belong to different processes.

Two cases are compared in this section to demonstrate that the using the distributed memory architecture to the maximum degree can help reduce the overheads. Both cases study a 32-process partition and the 8-core processors are used. The first case assigns every 8 partitions into 8 cores which belong to the same processors. In this fashion, totally 4 processors are used for the 32 partitions. **Table 46** has the results. The second case uses totally 6 processors for the 32 partitions. Each of the first 4 processors takes 5 partitions while each of the last 2 processor takes 6 partitions. The second run’s results are in **Table 47**.

Table 46. Case 1 results, 32-process

	Message Passing			CPU Total			
Processor	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	13.836	159.731	173.567	1335.883	1162.316	87.01%	244981
2	38.320	597.716	636.036	1336.381	700.345	52.41%	132834
3	31.546	224.852	256.398	1336.378	1079.980	80.81%	153655
4	4.732	106.938	111.670	1336.376	1224.706	91.64%	318901
		Total M-P, s	1177.671		Average Load	77.97%	

Table 47. Case 2 results, 32-process

	Message Passing			CPU Total			
Processor	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	14.951	394.624	409.575	1198.427	788.852	65.82%	88183
2	22.325	420.933	443.258	1198.977	755.719	63.03%	55059
3	32.756	845.538	878.294	1198.976	320.682	26.75%	87733
4	29.197	545.250	574.447	1198.974	624.527	52.09%	31401
5	15.475	52.284	67.759	1198.975	1131.216	94.35%	87158
6	5.823	566.825	572.648	1198.976	626.328	52.24%	35250
		Total M-P, s	2945.981		Average Load	59.05%	

The results show that the second case runs faster than the first case. However, the first case has smaller overheads than the second case and the average load of case 1 is also larger than case 2. From **Table 25**, the original assigning pattern of the 32 partitions gives a parallel simulation time of 1260.673 seconds and this is between the two cases studied in this section.

The different performances are because of the nature of parallel machine architecture. As mentioned before, the architecture of Blackgold is a hybrid of distributed memory and shared memory. Shared memory can largely reduce the communications if processes are assigned to cores within one node. This feature definitely reduces system overheads. However, shared memory architecture also has its disadvantages. In shared memory architecture, multiple processes usually share the same memory and this may affect the information transition between CPUs and memory. As a result, limited cache would largely restrict the parallel performance. On the contrary, in the distributed memory architecture, each process has its own memory and this memory is not shared with any other processes. Thus, the caching issue that exists in shared memory is no longer a problem in distributed memory architecture. This means that distributed memory architecture can compute faster. This feature of distributed memory is especially useful to achieve a parallel implementation with good scalability. With more nodes used, shared memory's low overheads are slightly sacrificed while the scalability of the entire parallel implementation is improved.

In the results, the 6-processor parallel run turns out to be the fastest while the 4-processor parallel run has the lowest overheads. The original Metis implementation has a much stronger overheads since the grid-to-process assignment is random.

This section proves that shared memory is capable of reducing overheads while distributed memory is capable of increasing parallel implementation's scalability. Finding the balance between this two architectures can reduce the cores needed and increase the efficiency.

6.3 Overheads of non-neighboring processes

In the original grid-to-process assignment, a partition is assigned to a process with the same number. For example, partition 1 is assigned to CPU 1 and partition 2 is assigned to CPU2. This method at least guarantees that some of the neighboring partitions are assigned to CPUs that are neighbors.

This section wants to know what will change in terms of overheads if two neighboring partitions are assigned to non-neighboring cores. Partitions are intentionally assigned to cores that are not neighbors. As a result, the communications between CPUs will take longer time. This effect is especially significant for communications between CPUs that are not in the same node. The 16-partition Metis case is studied in this section. Two assignments are compared here. In the first assignment, 16 partitions are assigned to 6 nodes. 4 out of 6 nodes have 3 partitions on each of them and the rest have 2 partitions on each of them. Neighboring partitions are assigned to non-neighboring cores on purpose. In the second assignment, partitions are randomly assigned to CPUs. This randomness does not guarantee that neighboring partitions are assigned to different nodes. **Table 48** and **Table 49** show how partitions are assigned in the two assignment patterns.

Table 48. Assignment 1 results

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	4.653	482.947	487.600	1012.049	524.449	51.82%	87158
2	8.934	561.162	570.096	1013.024	442.928	43.72%	88183
3	16.467	612.357	628.824	1013.026	384.202	37.93%	46767
4	17.889	433.154	451.043	1013.038	561.995	55.48%	33895
5	23.005	713.908	736.913	1013.031	276.118	27.26%	13404
6	4.886	19.726	24.612	1013.035	988.423	97.57%	87382
7	8.192	399.589	407.781	1013.039	605.258	59.75%	87733
8	19.988	547.947	567.935	1013.031	445.096	43.94%	22779
9	13.788	383.994	397.782	1013.039	615.257	60.73%	55059
10	13.391	556.912	570.303	1013.033	442.730	43.70%	62356
11	18.179	451.363	469.542	1013.044	543.502	53.65%	31401
12	21.448	635.082	656.530	1013.036	356.506	35.19%	18291
13	11.363	358.443	369.806	1013.04	643.234	63.50%	68153
14	11.253	339.109	350.362	1013.034	662.672	65.41%	66297
15	18.313	523.585	541.898	1013.036	471.138	46.51%	35250
16	15.399	390.195	405.594	1013.039	607.445	59.96%	46263
		Total M-P, s	7636.621		Average Load	52.88%	

Table 49. Assignment 2 results

	Message Passing			CPU Total			
Process	Initialization, s	Timesteps, s	M-P Total, s	Elapsed Time, s	Non-idle Time, s	Load	Cells
1	5.137	548.991	554.128	1005.25	451.122	44.88%	88183
2	13.718	368.696	382.414	1006.053	623.639	61.99%	55059
3	8.097	389.655	397.752	1006.048	608.296	60.46%	87733
4	18.120	435.064	453.184	1006.049	552.865	54.95%	31401
5	8.384	473.225	481.609	1006.053	524.444	52.13%	87158
6	18.233	502.941	521.174	1006.054	484.880	48.20%	35250
7	16.384	594.742	611.126	1006.051	394.925	39.25%	46767
8	17.789	413.172	430.961	1006.048	575.087	57.16%	33895
9	4.825	30.271	35.096	1006.051	970.955	96.51%	87382
10	19.941	556.722	576.663	1006.054	429.391	42.68%	22779
11	11.304	362.482	373.786	1006.056	632.270	62.85%	68153
12	11.237	343.736	354.973	1006.053	651.080	64.72%	66297
13	22.961	713.764	736.725	1006.053	269.328	26.77%	13404
14	21.397	638.162	659.559	1006.054	346.495	34.44%	18291
15	13.373	563.406	576.779	1006.055	429.276	42.67%	62356
16	15.398	399.938	415.336	1006.053	590.717	58.72%	46263
		Total M-P, s	7561.265		Average Load	53.02%	

The results show that assignment 1 has a longer simulation time and also a longer message passing time. The reason is that assignment 1 has more communications across processes and this decreases the efficiency of the parallel system.

CHAPTER VII

CONCLUSIONS

Running parallel reservoir simulation jobs on the cluster, this research studies load imbalance and some other issues related to parallel implementation performance. Many variables are analyzed in the study such as partitioning strategies, well geometries, well constraints, and grid-to-process assignment patterns. Some conclusions are drawn as follow.

(1) 2D decomposition can distribute grid blocks evenly into partition. However, it does not take into account of weights and cannot address reservoir models with some degree of heterogeneity.

(2) 2D, Metis, and spectral partitioning are good at distributing grid blocks evenly into subdomains as well as minimizing communications between these subdomains.

(3) Metis and spectral partitioning can incorporate weighting factors when partitioning. Transmissibility field is a good weighting factor. Metis and spectral partitioning give high transmissibility areas finer partitions so that load can be better balanced among partitions.

(4) There is a limitation of using weighting factors in Metis and spectral methods. In this model, planar partitioning is used. However, the vertical variety is very strong and each of the 100 layers presents different transmissibility field. In this study only the transmissibility field from one

layer is used. To better represent transmissibility's vertical diversity, three-dimensional partitioning needs to be applied.

(5) Both Metis and spectral can partition the mesh very fast.

(6) Using well locations as weighting factors in Metis and spectral partitioning can reduce the system overheads.

(7) When partition sizes are similar, processes corresponding to partitions at the center of the model have larger loads than processes assigned with grid blocks at the boundary of the mesh.

(8) Zoltan embedded in the simulator used in the study has very limited flexibility and its parallel implementation cannot utilize all available cores.

(9) The parallel implementation significantly reduces PVT property computation time. The more partitions are used, the less time PVT property computation needs.

(10) Parallel implementation does not reduce solver's time percentage.

(11) Horizontal well increases overheads and simulation time. However, it does not significantly affect the load distribution among cores.

(12) Intentionally assigning neighboring partitions to CPUs that belong to the same node decreases overhead and improve efficiency.

(13) Intentionally assigning neighboring partitions to CPUs in different nodes increases overheads and decrease efficiency.

(14) An optimum implementation of the hybrid of distributed memory architecture and shared memory architecture can reduce the nodes needed and still obtain a satisfactory parallel performance.

(15) There are a few things to do in the future. The first is to go to three dimensional decomposition to honor the vertical variations of reservoir properties. The second is to increase number of nodes involved in the parallel runs to see how efficiency and parallel performance will change. Linear performance's relationship with partitioning strategies is also worth noting.

REFERENCES

Barney, Blaise. 2015. Introduction to Parallel Computing, *LLNL*, https://computing.llnl.gov/tutorials/parallel_comp (accessed 30 March 2015).

Crockett, S. and Devere, S. 2009. Comparing the performance of the Landmark Nexus reservoir simulator and HP servers, Bladenetwork, http://www.bladenetwork.net/userfiles/file/PDFs/09-LM61-04NexusConfigurationWhitePaper.pdf (accessed 10 April 2015).

Devine, K., Boman, E., Heaphy, R. et al. 2002. Zoltan Data Management Services for Parallel Dynamic Applications. Computing in Science & Engineering 4 (2): 90-96. DOI: 10.1109/5992.988653.

Frachtenberg, E., Feitelson, D.G., Petrini, F. et al. 2003. Flexible Coscheduling: Mitigating Load Imbalance and Improving Utilization of Heterogeneous Resources. Presented at Parallel and Distributed Processing Symposium, 22-26 April. DOI: 10.1109/IPDPS.2003.1213191.

Halliburton. 2015. Nexus User Guide.

Hendrickson, B. and Kolda, T.G. 2000. Graph Partitioning Models for Parallel Computing. Parallel Computing 26 (12): 1519-1534. DOI: http://dx.doi.org/10.1016/S0167-8191(00)00048-X.

Hendrickson, B. and Leland R. 1995. The Chaco User's Guide Version 2.0, Sandia, July 2015, http://prod.sandia.gov/techlib/access-control.cgi/1995/952344.pdf (accessed 18 February 2015).

Karypis, G. and Kumar, V. 1998a. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. SIAM Journal on Scientific Computing 20 (1): 359-392. DOI: 10.1137/S1064827595287997.

Karypis, G. and Kumar, V. 1998b. METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version 4.0, UMN, http://glaros.dtc.umn.edu/gkhome/metis/metis (accessed 18 February 2015).

Karypis, G. and Kumar, V. 1998c. Multilevel k-Way Partitioning Scheme for Irregular Graphs. Journal of Parallel and Distributed Computing 48 (1): 96-129. DOI: http://dx.doi.org/10.1006/jpdc.1997.1404.

Killough, J.E. and Wheeler, M.F. 1987. Parallel Iterative Linear Equation Solvers: An Investigation of Domain Decomposition Algorithms for Reservoir Simulation. Presented at SPE Symposium on Reservoir Simulation, 1-4 February, San Antonio, Texas. SPE-16021-MS, DOI: 10.2118/16021-MS.

Lu, P., Beckner, B.L., Shaw, J.S. et al. 2008. Adaptive Parallel Reservoir Simulation. Presented at International Petroleum Technology Conference, 3-5 December, Kuala Lumpur, Malaysia. IPTC-12199-MS, DOI: 10.2523/IPTC-12199-MS.

Maliassov, S. and Shuttleworth, R. 2009. Partitioners for Parallelizing Reservoir Simulations. Presented at SPE Reservoir Simulation Symposium, 2-4 February, The Woodlands, SPE-119130-MS. DOI: 10.2118/119130-MS.

McSherry, F. 2001. Spectral Partitioning of Random Graphs. Presented at 42nd IEEE Symposium on Foundations of Computer Science, 8-11 October. DOI: 10.1109/SFCS.2001.959929.

Oliker, L. and Biswas, R. 1998. Plum: Parallel Load Balancing for Adaptive Unstructured Meshes. Journal of Parallel and Distributed Computing 52 (2): 150-177. DOI: http://dx.doi.org/10.1006/jpdc.1998.1469.

Pothen, Alex, Horst D. Simon, and Kang-Pu Liou. 1990. Partitioning sparse matrices with eigenvectors of graphs. SIAM Journal on Matrix Analysis and Applications 11 (3): 430-452. DOI: 10.1137/0611030.

Reinders, James. 2012. An Overview of Programming for Intel Xeon processors and Intel Xeon Phi Coprocessors, Intel, 15 October 2010, http://download.intel.com/newsroom/kits/xeon/phi/pdfs/overview-programming-intel-xeon-intel-xeon-phi-coprocessors.pdf (accessed 18 February 2015).

Tallent, N.R., Adhianto, L., and Mellor-Crummey, J.M. 2010. Scalable Identification of Load Imbalance in Parallel Executions Using Call Path Profiles. Presented at 2010 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 13-19 November. DOI: 10.1109/SC.2010.47.

Sarje, A., Song, S., Jacobsen, D. et al. 2015. Parallel Performance Optimizations on Unstructured Mesh-Based Simulations. Procedia Computer Science 51 (0): 2016-2025. DOI: http://dx.doi.org/10.1016/j.procs.2015.05.466.

Wang, Y. and Killough, J.E. 2014. A New Approach to Load Balance for Parallel/Compositional Simulation Based on Reservoir-Model Overdecomposition. SPE J. 19 (2): 304-315. SPE-163585-PA. DOI: 10.2118/163585-PA.