Journal of Natural Gas Science and Engineering 28 (2016) 447–460

Contents lists available at ScienceDirect

Journal of Natural Gas Science and Engineering

journal homepage: www.elsevier.com/locate/jngse

The application of static load balancers in parallel compositional reservoir simulation on distributed memory system

Xuyang Guo [a, *], Yuhe Wang [b], John Killough [a]

[a] Harold Vance Department of Petroleum Engineering, Texas A&M University, 3116 TAMU, College Station, TX 77843-3116, USA
[b] Petroleum Engineering Program, Texas A&M Engineering Building, Education City, PO Box 23874, Doha, Qatar

A R T I C L E I N F O

Article history:
Received 5 September 2015
Received in revised form
13 November 2015
Accepted 17 December 2015
Available online 20 December 2015

Keywords:
Compositional reservoir simulation
High performance computing
Load balance
Graph partitioning
Distributed memory system

A B S T R A C T

Compositional reservoir simulation depicts the complex behaviors of all the components in gaseous, liquid, and oil phases. It helps to understand the dynamic changes in reservoirs. Parallel computing is implemented to speed up simulation in large scale fields. However, there is still many challenges in obtaining efficient and cost-effective parallel reservoir simulation. Load imbalance on processors in the parallel machine is a major problem and it severely affects the performance of parallel implementation in compositional reservoir simulators. This article presents a new approach to the reduction of load imbalance among processors in large scale parallel compositional reservoir simulation. The approach is based on graph partitioning techniques: Metis partitioning and spectral partitioning. These techniques treat the simulation grid, or the mesh, as a graph constituted by vertices and edges, and then partition the graph into smaller domains. Metis and spectral partitioning techniques are advantageous because they take into account the potential computational load of each grid block in the mesh and generate smaller partitions for heavy computational load areas and larger partitions for light computational load areas. In our case, the computational load is represented by transmissibility. After new partitions are generated, each of them is assigned to a processor in the parallel machine and new parallel reservoir simulation can be conducted. Traditionally, the intuitive 2D decomposition is frequently used to partition the simulation grid into small rectangles, and this is a major source of load imbalance. The performance of parallel compositional simulation based on our partitioning techniques is compared with the most commonly used 2D decomposition and it is found that load imbalance in our new simulations is reduced when compared with the traditional 2D decomposition. This study improves the efficiency of compositional simulation and eventually makes it more cost-effective for hydrocarbon simulation on mega-scale reservoir models.

1. Introduction

The application of parallel computing in oil industry has been studied for many years. Lu et al. (2008) introduced a parallel reservoir simulator that can be run on multi-core personal computers. The study showed that normal multi-core computers have the ability of improving simulation efficiency and their parallel implementation does not need large scale parallel machines. Reinders (2012) introduced some processors that can be used to facilitate parallel computing and provided an overview of programming with these processors. Killough and Wheeler (1987) presented the usage of parallel iterative methods to solve linear equations. They proposed a domain decomposition algorithm as preconditioner in parallel simulations and it was proved that their method largely improves computational efficiency. Many efforts have also been put into the understanding of load imbalance. Wang and Killough (2014) managed to mitigate load imbalance by over-decomposing the mesh of a reservoir model. In their study, the mesh was divided into small partitions and the number of the partitions is way larger than the number of available cores in the processors of the parallel machine. By assigning multiple partitions into one core and by introducing a dynamic load balancer, they doubled the speedup and largely reduced load imbalance. Sarje et al. (2015) also reached optimized parallel simulation performance on an unstructured mesh by addressing load imbalance

* Corresponding author.
E-mail addresses: xuyang01@tamu.edu (X. Guo), yuhe.wang@qatar.tamu.edu (Y. Wang), jkillough@tamu.edu (J. Killough).

http://dx.doi.org/10.1016/j.jngse.2015.12.030
1875-5100/© 2015 Elsevier B.V. All rights reserved.

issues. They developed a new way of partitioning the unstructured mesh and it reduces communication between different partitions. Tallent et al. (2010) introduced a method of identifying load imbalance. Although it does not address the issue, it identifies the cores that have the most severe load imbalance. Thus, modifications can be made to the segments that cause load imbalance so as to reduce the imbalance.

Although many efforts have been put into the study of load balance in parallel reservoir simulation, few entertained the possibility of using graph partition as a way of balancing loads. This study was undertaken to incorporate graph partitioning techniques to parallel compositional simulation. Several graph partitioning techniques honoring both reservoir geometry and geological features were used as static load balancers. Their load balancing quality and the resulting parallel load balance performance were examined to evaluate the effectiveness of the implementation. Besides, static partition quality was correlated with load balance from real simulation runs so that one can understand whether a certain graph partitioning technique is capable of relieving load imbalance.

2. Background

Load balancer is widely used in implementation of parallel computing. Generally speaking, it divides the work into small parts so that each of them can be assigned to a core in the parallel machine. There are a variety of load balancers and the choice of load balancer can largely affect the performance of parallel computing implementation. It is ideal if each core is assigned exactly same amount of work load. However, in real applications, after the entire work is divided by load balancers, it is not always easy to evaluate each part's work load. In addition, as the computation proceeds, the work load on a certain part may change tremendously. Another consideration about load balancer is the difficulty of implementation in a parallel system. All of these facts make it hard to choose the best load balancer that help optimize parallel performance.

Popular load balancers consist of static and dynamic balancer. Static load balancer divides the entire work and distributes them to processors before any actual simulation is started while dynamic load balancer divides the work during the simulation process and it is based on the feedback of each processor's real time load distribution. Intuitively, dynamic load balancer is probably the better choice. However, in many cases, it is hard for this method to be implemented. Also, due to its feature of ongoing re-partitioning of the root mesh, the way and the amount of data that need to be exchanged between processors keep changing. This may significantly increase the overheads of the parallel system and largely decrease the overall parallel computing performance. As a result, the benefits of dynamic load balancer can be undermined by its own complexity and overheads. Zhang et al. (1997) compared the performance of static load balancers and dynamic load balancers for a parallel implementation on a heterogeneous system. They found out that, unlike common perception, the quality of parallel implementation based on static load balancers is nearly as good as the one based on dynamic load balancers on small to moderate scale systems. When it comes to large scale systems, static load balancers sometimes even perform better than dynamic load balancers. They identified that the performance of dynamic load balancer is heavily weakened by system overheads. In our study, static load balancer was used based on the following reasons:

1. Static load balancer takes into account physical properties of the reservoir model and can generate partitions based on them.
2. Unlike dynamic load balancer, static load balancer will not introduce huge overheads.
3. Static load balancer is not hard to be implemented into the Message Passing Interface based compositional simulator.
4. The combination of static load balancer and the simulator is able to deal with a variety of reservoir models. If new dataset and geological information are given, static load balancer can easily distribute the entire workload into small parts based on new information.

Graph partition is one way of static load balancing. In order to process the reservoir grid with graph partitioners, the grid is treated as graph. The grid blocks in the mesh are equivalent of vertices in a graph and the connections (faces) between grid blocks are equivalent of edges in a graph. The following form is used to represent a graph G:

$$G = (V, E)$$

here V is vertices and E is edges. A graph can be described solely based on vertices and edges data. In a realistic case (e.g. a reservoir), each vertex (grid block) has properties such as transmissibility and porosity. Such properties can be used as weights so that they are honored in the partitioning process. Thus, three kinds of information are associated with each vertex: vertex location, other vertices that connect to this vertex, and the weight of the vertex.

Metis and spectral methods are the specific graph partitioning techniques that we selected in this article. Both methods require similar inputs from the root graph.

2.1. Metis

Metis is an open access software package developed by University of Minneapolis and it is capable of partitioning unstructured graphs. It can be installed on UNIX system and it provides both standalone software package and library. Vertex location, neighboring vertices, and vertex weights are the inputs to the package. Also, Metis needs the desired number of partitions as input. The partitioner generates new partitions so that the sum of vertex weights in each partition is the same. Besides, the edge cuts, which stand for communication between different partitions, are minimized (Karypis, 2013).

Based on our experiences, Metis partitions a graph with 10,000 vertices in a few seconds and partitions a graph with one million vertices in less than 3 min. This is fast enough for our needs and it largely saves the pre-processing time. Instead of recursive bisection, Metis uses a k-way partitioning which partitions the root graph into k pieces simultaneously. According to Karypis and Kumar (1998), for a graph G = (V,E), Metis algorithm partitions it in O(|E|) time and this is faster than the conventional recursive bisection by a factor of O(log k) where k is the target number of partitions. They pointed out that there are three steps for Metis to partition a graph: coarsening, initial partitioning, and uncoarsening.

In the coarsening step, the original graph $G_0 = (V_0,E_0)$ is coarsened by n times. Vertices of the original graph are distributed into groups of vertices and each group becomes a new vertex in the coarser graph. This coarsening process maintains the connectivity of the initial graph and preserves it in the coarser version of graph. After n times of coarsening, the original graph becomes $G_n = (V_n,E_n)$ and V_n is much smaller than V_0. The connectivity and weights of the original (finest) graph G_0 are largely preserved and transmitted to the coarsest graph G_n. The coarsest graph is small in size and complexity. The partitioning is conducted at this level so that the partitioning process is fast.

In the initial partitioning process, Metis partitions the coarsest graph G_n into the target number of k parts: P_1, P_2,…, and P_k. The partitions obtained at this step are primitive and they will be modified at the later step so that each partition will have similar weights and the communication between partitions can be minimized.

In the uncoarsening step, P_1, P_2,…, and P_k are projected back to the initial graph. During this process, single vertices can be moved to neighboring partitions to balance weights as well as reduce communication (edge cuts). Local refinement algorithm of Kernighan-Lin is automatically applied in Metis and it helps reduce inter-partition communications.

2.2. Spectral partitioning

Spectral partitioning relates the partitioning process to eigenvectors from the matrix constructed by the graph. It was first proposed by Pothen et al. (1990) that the eigenvector of the second smallest eigenvalue of the graph's Laplacian matrix is a good vector to initially bisect the original graph. Spectral partitioning helps to reduce matrix condition numbers related to partitioned graphs and eventually improves solver performance. As mentioned in this article, recursive bisection comes into play when the desired number of partitions is greater than two. The reason of using the second eigenvector (Fiedler vector) of the Laplacian matrix is that it represents the algebraic connectivity of graphs (Fiedler, 1973). Besides, in connected graph, the smallest eigenvalue is zero and its multiplicity is always one. Thus, the second smallest eigenvalue is the first positive eigenvalue. According to Pothen et al. (1990), spectral partitioning can be summarized by four steps:

1. Find the Fiedler vector $v_f = (v_1, v_2, \ldots, v_n)$ of the Laplacian matrix of the original graph with n vertices $G_0 = (V_0, E_0)$. Laplacian matrix L is prescribed as:

$$L_{i,j} = \begin{cases} d_i \ \ if \ \ i = j \\ -1 \ \ if \ \ i \neq j \ \ and \ \ (i,j) \in E_0 \\ 0 \ \ otherwise \end{cases}$$

L is always n by n and it is symmetric. d_i stands for the degree of vertex i, or the number of vertices connected to vertex i.

2. Find the median value of all components in v_f. Put vertices corresponding to components of v_f smaller than the median to V_1 and put the others to V_2. The difference between sizes of V_1 and V_2 should be no more than one.
3. Find the edge separator E'. It contains edges that connect V_1 and V_2. E_1 becomes the set of edges with both ends in V_1 and E_2 becomes the set of edges with both ends in V_2.
4. The two spectral partitions are $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. If the target number of partitions are larger than two, the generated partitions are further bisected using the method in previous steps.

2.3. Parallel simulator and parallel environment

In the article, a parallel compositional reservoir simulator was used (Nexus®). It is based on domain decomposition so that the entire model can be divided into sets of cells. When it is run in parallel mode, computation of multiple sets of cells is started simultaneously and this enables faster simulation, especially for very fine resolution models (Tarman et al., 2011). Sets of cells are mapped to processes and the communication between processes is based on the standard Message Passing Interface. The domain decomposition based modification was incorporated into the serial code so that parallelization could be achieved. Implicit for pressure and explicit for saturation numerical treatment was used in the study (Shiralkar et al., 2005).

The domain decomposition based simulator is able to utilize the multiple nodes simultaneously. The implementation of this memory architecture can improve parallel reservoir simulation performance better than the other memory architecture, the shared memory system (Halliburton, 2015). A parallel machine was used in this study and four nodes of it were actually involved in the simulation. The nodes have same computational capacity so that the only factors that affect computational performance and load imbalance among cores are graph partitioning and the way of assigning partitions to cores. Graph partitioning affects the simulation performance because the way it distributes workload to processes is related to load balance. The way of assigning partitions to cores also affects simulation performance. Although the memory architecture among nodes is distributed memory, the architecture among cores within an individual node is shared memory. This fact implies that if too many tasks, or partitions in our case, are assigned to one node, the memory bandwidth within this node can massively limit the performance of the cores in this node. Consequently, the overall performance is affected. More details will be given in the later sections. In Fig. 1, the memory architecture of the parallel machine is visualized. It is shown that on each node, all cores access the same memory while for each node, it has its own memory and does not share memory with any other nodes' cores.

3. Static load balancer and its implementation

Based on the two proposed graph partitioning methods, it is necessary to conduct a series of works from the adaptation of graph partitioners, to the incorporation of balancer in parallel simulator, to the analysis of load balance based on parallel simulation results (Guo, 2015).

3.1. Selection of weights

Unlike the intuitive and commonly used 2D decomposition, the two proposed partitioners do not assume equal computational load across the reservoir grid. Geological and geometric features are candidates to represent computational load of a certain vertex of the weighted graph when it is processed by either partitioning methods. We used two types of weights in the study: transmissibility and non-idle time based on previous runs.

3.1.1. Transmissibility

Transmissibility is a candidate to indicate local computational load. It is defined by Eqn. (1). A large transmissibility value indicates fast property changes in the grid-block. During simulation, such grid blocks are more likely to have heavier computational load than those with low transmissibility values and it is harder for simulation to converge at such locations.

$$T = \frac{AKK_r}{lB\mu} \tag{1}$$

A sample test was run to substantiate the statement that transmissibility indicates computational load. The sample test was based on the reservoir model that will be discussed later. The test has five transmissibility cases: 1% of original transmissibility, 10% of original transmissibility, original transmissibility, 10 times of original transmissibility, and 100 times of original transmissibility. For the five cases, the only variable is transmissibility. The elapsed time for a 10-day production scenario is shown in Table 1. The table shows that in this model higher transmissibility indicates longer computational time and it is reasonable to use transmissibility as

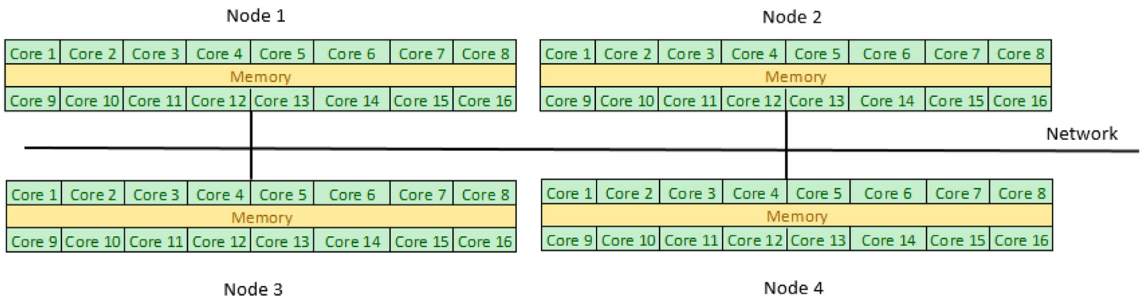

Node 1
Core 1 Core 2 Core 3 Core 4 Core 5 Core 6 Core 7 Core 8
Memory
Core 9 Core 10 Core 11 Core 12 Core 13 Core 14 Core 15 Core 16
Node 2
Network
Node 3
Node 4

Fig. 1. Parallel machine memory architecture.

Table 1
Transmissibility and elapsed time.

Case	$\frac{\text{Transmissibility of the case}}{\text{Original transmissibility}}$	Elapsed time, s
1	0.01	549
2	0.1	604
3	1	609
4	10	723
5	100	1105

weights for load balancers.

3.1.2. Non-idle time

Except for transmissibility, each core's non-idle time of previous 2D decomposition parallel simulations is also a good weighting factor. In parallel environment, there is always idle time for an individual core when it is waiting for results from other cores to synchronize. Such time is the wasted time when the core is not conducting any work. The non-idle time of an individual core is the time when it is conducting computation. It is calculated as shown in Eqn. (2). Before we entertained the possibility of using graph partitioners as load balancers, we already simulated several scenarios on the parallel machine using the 2D decomposition that is solely based on geometry. Although the simulation scenarios are slightly different, the non-idle time distribution among cores does not change much. In these runs, some cores have very long non-idle time while some have very short non-idle time. It is suggested that the partitions (parts of the reservoir mesh) assigned to cores with short non-idle time represent relatively low work load. Load imbalance on such cores can be relieved if the size of these partitions is expanded and more work is assigned to low work load cores. This is realized by using the non-idle time as weights. Since Metis and spectral partitioning have the feature of incorporating weights with grid blocks, graph partitioning results based on non-idle time are also generated using these two partitioners.

$$T_{non-idle} = T_{Elapsed} - T_{Message\ passing} \quad (2)$$

For example, the first partition G_1 is assigned to core 1 and from reservoir simulation results, the non-idle simulation time on core 1 is $T_{Non-idle}$. Using the method mentioned above, all cells within G_1 are assigned the same weight with the value of $T_{Non-idle}$. Cells in all the other partitions can be assigned weights exactly the same.

Since the repartitioning process is based on parallel simulation data acquired from previous runs, it is not very applicable when a totally new reservoir simulation case needs to be studied. One has to run a 2D decomposed parallel simulation in advance to obtain the non-idle time distributions on cores. However, this was included in the study to examine the static load balancers and to test different weighting factors' effects on graph partitioners.

3.2. Non-contiguous subdomains

Spectral method utilizes weighted vertex information to generated partitioned graph. However, it is very likely that spectral method would generate non-contiguous subdomains. Non-contiguous subdomains are the non-neighboring subdomains that belong to the same partition. The areal view on the left in Fig. 2 is an example of non-contiguous subdomains. In this picture, a target number of 2 partitions was prescribed in spectral method. However, the partition results in 4 subdomains: the two maroon domains belong to partition 1 and the two green domains belong to partition 2. The emergence of non-contiguous subdomains increases the number of edge cuts and the number of vertices that are at boundaries with neighboring partitions, which results in the increase of communication between partitions. In the parallel system, such increase lead to augmented overheads and eventually decrease parallel simulation efficiency. To relieve this phenomenon, Kernighan-Lin algorithm was used to modify the partitioning results in our study. Kernighan-Lin algorithm can be used to locally refine the spectral partitioning results and improve overall parallel performance (Hendrickson and Leland, 1995). Fig. 2 is a comparison between spectral method without the algorithm and the one with the algorithm. Since we only consider two dimensional partitions, only the areal view from the top of the grid is shown. It is clear that after the introduction of the algorithm, the partitioning turned out to be cleaner than before. It is worth mentioning that, in our case, when the target number of partitions is large, Kernighan-Lin algorithm does not always cleanly fix up the edge cuts and there are still partitions comprised of more than one discrete domains.

The evidence of the improvement applying Kernighan-Lin algorithm can be seen in Tables 2 and 3. They record the static partition results and results of a test run to evaluate parallel simulation time. In Table 2 total edge cuts and boundary vertices of partitioning with and without Kernighan-Lin are shown. It is seen that application of Kernighan-Lin algorithm reduces the number of edge cuts and boundary vertices. Table 3 shows that the application of Kernighan-Lin increases the speedup of parallel implementation in most cases.

3.3. Evaluation of partitioning results

For each partition method, the actual partitioning quality can be examined based on the performance data after all simulation runs. However, before the simulation, one can evaluate the quality of the partitioning by evaluating weights assigned to partitions. Liu et al. (2015) proposed a way of evaluating partition quality assuming all cells have equal computations. Based on this method, we expand it by taking into account the different weights on cells and define the equations of partitioning quality as follow:

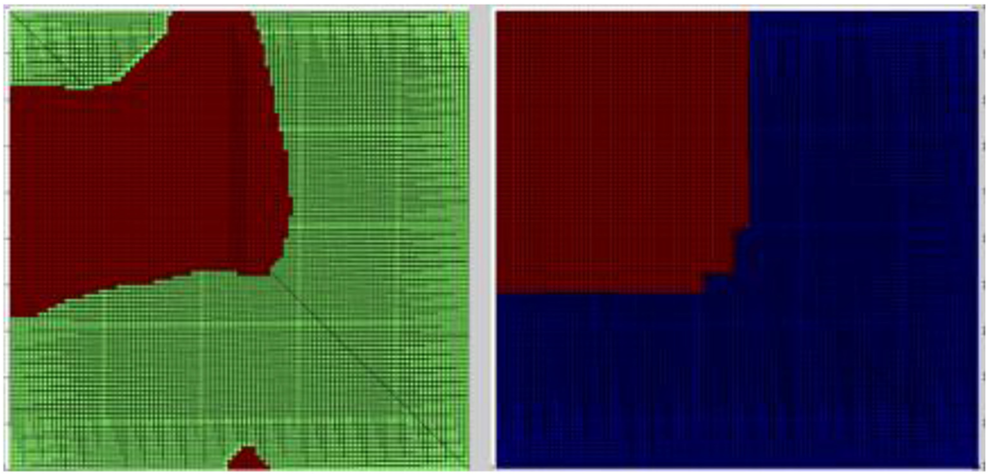


Fig. 2. Partitioning for 2-core parallel simulation without Kernighan-Lin algorithm (left) and with Kernighan-Lin algorithm (right).

Table 2
Kernighan-Lin algorithm and partition performance.

Partition number	Applied Kernighan-Lin?	Edge cuts	Boundary vertices
2	Yes	101	200
	No	507	718
4	Yes	248	491
	No	939	1349
8	Yes	464	908
	No	1516	2205
16	Yes	724	1403
	No	2162	3197
32	Yes	1126	2155
	No	2649	3979

Table 3
Actual simulation performance and Kernighan-Lin algorithm.

Partition number	$\frac{\text{Speedup with K–L}}{\text{Speedup without K–L}}$
2	0.792
4	1.044
8	1.061
16	1.024
32	1.222

$$f_p = \frac{N_p \times \max\limits_{1 \le i \le N_p} |W_i|}{\sum_{i=1}^{N_p} |W_i|} \tag{3}$$

$$W_i = \sum_{j=1}^{N_i} \left| (G_i)_j \times (w_i)_j \right| \tag{4}$$

In these equations, f_p stands for the partitioning quality and it is a value equal or larger than 1. Only when the weights are equally distributed to each of the partitions the value becomes 1. Greater partitioning quality values means greater imbalance. N_p is the number of target partitions. $(G_i)_j$ is the j th cell in the i th graph and $(w_i)_j$ is the j th cell's weight in the i th graph. The value of W_i is the sum of the weights on partition G_i. N_i is the number of cells in partition G_i. These formulas provide a way to examine the partitioning results based on reservoir geometry and reservoir geological data.

3.4. Areal partition vs. 3D partition

In this study, only two dimensional partitioning was used on x-direction and y-direction. On the z-direction there was no further partitioning. Although it is intuitively more reasonable to partition the mesh on all three dimensions to represent vertical variations of geological features, it turns out that this consideration largely infringes the parallel solver performance. When 3D partitioning was used in our simulations, the simulation time was largely increased and much more iterations were needed in each time step.

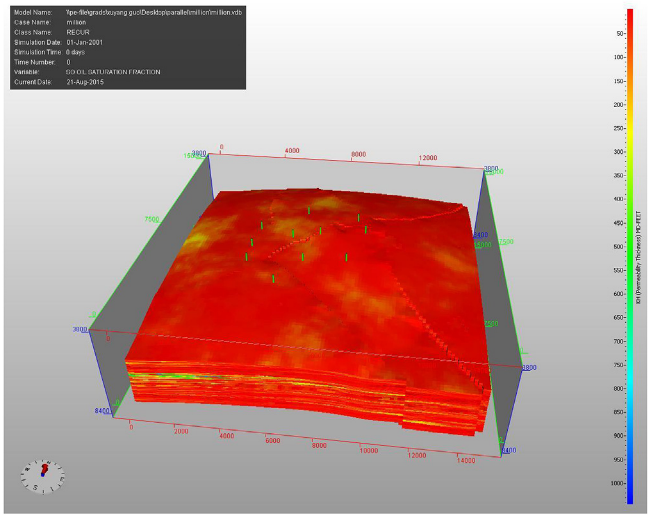
An example is provided to show this effect. For a 32-process simulation, both areal and 3D Metis partitions were generated and incorporated in parallel runs. The results show that during the simulation, the average solver iterations per newton step were 54.20 for 3D case and 23.79 for areal case. In consequence, the 3D case took a much longer total simulation time than areal case and it was not considered an optimum option.

3.5. Example

A large scale synthetic reservoir model was used to examine the selected partition methods as load balancers. In the model, the unstructured grid has one million grid blocks and the dimension is 100 by 100 by 100. Grid blocks do not have uniform sizes. The coordinate system follows the right-hand rule. I-direction numbering is ascending from westernmost to easternmost. J-direction numbering is ascending from northernmost to southernmost. K-direction numbering is from topmost to bottommost. Each grid cell has its own cell ID and it is from 1 to 1,000,000. The reservoir thickness is around 500 feet and it varies with locations. The 100 layers give a very high resolution on vertical direction. The reservoir is 15,000 feet long and wide. As shown in Fig. 3 there are totally 11 producers and they are all fully penetrated through the 100 layers. The reservoir grid represents an anticline structure with hydrocarbons mainly residing at the center and top of the structure. Six major faults traverse the reservoir. There are eight components. Except for water, the other seven components are shown in Table 4.

The simulation time was one year long. The maximum timestep was 1 day and the minimum was 0.001 day. Timestepping method was IMPES and the preconditioner for pressure was ILU. For EOS calculation, Peng-Robinson equation of state was used. The grid decomposition was two-dimensional.

Five specific partition methods were applied to the model:



Model Name: \\pe-file\grads\xuyang.guo\Desktop\parallel\million\million.vdb
Case Name: million
Class Name: RECUR
Simulation Date: 01-Jan-2001
Simulation Time: 0 days
Time Number: 0
Variable: SO OIL SATURATION FRACTION
Current Date: 21-Aug-2015

Fig. 3. Permeability distribution and wells.

partitioned models were put into the simulator and run on the parallel machine. After running all the cases, load distribution results are obtained. Figs. 4–8 show the comparisons of load distribution for the 2-core, 4-core, 8-core, 16-core, and 32-core parallel runs, for all 5 partition methods.

It is noted from Table 5 that whatever the number of partitions, spectral method has the best partition quality while 2D has the worst. 2D decomposition has the worst quality because computational load is not assumed to be equal among cells. Since it only used geometry of the reservoir model and does not use geological properties as input, 2D decomposition does not generate partitions that bear similar loads. Another observation is that as the number of partitions increases, the partition quality value increases. Partitioners can distribute load better when there are less partitions. The maximum partition weight $\max_{1\le i\le N_p} |W_i|$ decides the value of f_p and as partition number increases, it becomes harder to limit the maximum weight to a value that is close enough to the average weight. This makes the quality decrease as partition number increases.

It can be noted from previously mentioned figures that in all five scenarios, 2D decomposition always has the greatest fluctuation of loads. Also, as the number of cores increases, the differences of load among cores have the trend of increasing. The loads are also statistically evaluated and the results are shown in Figs. 9–13. All the relative data are recorded in Table 6.

Table 4
Component properties.

Component	Molar weight	Critical temperature, R	Critical pressure, psia	Critical gas compressibility factor	Acentric factor	Volume shift parameter	Parachor
P1	34.08	671.76	1296.19	0.28358	0.1	−0.115478	97.3714
P2	44.01	547.56	1069.87	0.27404	0.225	−0.0943467	125.743
P3	16.0633	342.87	786.53	0.33879	0.008054	−0.181405	45.8953
P4	53.9334	732.24	571.601	0.27484	0.180321	−0.0618549	154.095
P5	131.895	1107.74	336.801	0.25336	0.459627	−0.241669	392.138
P6	263.455	1390.17	268.456	0.27733	0.748941	−0.125251	752.477
P7	528.436	1935.43	133.059	0.16754	1.15346	−0.022134	1509.82

1. 2D decomposition (as reference)
2. Metis method with transmissibility as weights
3. Spectral method with transmissibility as weights
4. Metis method with non-idle time as weights
5. Spectral method with non-idle time as weights

For each partition method, 5 different target numbers of partitions were generated: 2, 4, 8, 16, and 32. In total, we had 25 kinds of partitioned reservoir model and all of them were put into the parallel machine for simulation. When the partitions were mapped to cores in the parallel machine, each partition was assigned one core so that the target number of partitions is equal to the number of processes involved in that specific parallel run.

In this study, the load was depicted as the ratio of the core's elapsed time excluding communication time (message passing time) to the total elapsed time. On a core, the communication time is what it takes for the core to communicate with all the other cores. As long as it is parallel environment, the communication time is greater than 0 and shorter than the total elapsed time. Consequently, the ratio, or the load, is always below 1 and above 0. In our study, each of the cores that participate in the simulation was calculated a load and we were able to obtain the distribution of loads for all cores. This distribution is a proper way of evaluating load balance and evaluating the efficiency of a certain partition method.

4. Results and discussion

2D, Metis, and spectral methods were used to generate partitioned reservoir models. Table 5 shows the partition quality of the 5 partition methods with Eqn. (3) and Eqn. (4). Then these

The high-to-low ratio of loads is based on the definition by Wang and Killough (2014). According to their definition, the high-to-low ratio is the maximum load to the minimum load. It represents how huge the load differences are. The other figures show the range, interquartile range, sample variance, and average of the load distributions. Range shows the difference between the largest load and the smallest load and it is similar to high-to-low ratio. Interquartile range shows the range between the first quartile and the third quartile of the load distribution's probability distribution. Sample variance reveals the imbalance of all load values and it is a good indicator of load imbalance status. The average of loads stand for the efficiency of processor cores. A larger average load means more efficient usage of the parallel machine.

Based on the statistical data, it is not hard to notice that in most of the cases, the intuitive 2D decomposition does not provide the optimum results and the cases with graph partitioners as load balancer have less load imbalance than the 2D decomposition case.

4.1. High-to-low ratio

The high-to-low load ratios are presented in Fig. 9. Small high-to-low ratios are preferred. The results show that for all load balancing techniques (even including the base case of 2D decomposition), the 8-partition case always has the largest high-to-low ratio. It means that when there are 8 partitions (8 cores) in a parallel simulation, the work load difference between the busiest core

Table 5
Partition quality.

Number of partitions	Partition quality				
	Metis	Spectral	2D	Re-partitioned Metis	Re-partitioned spectral
2	1.007811	1.000068	1.087050	1.006377	1.000401
4	1.009252	1.000163	1.090253	1.029140	1.001422
8	1.018736	1.000379	1.128238	1.029869	1.002172
16	1.029310	1.001208	1.287709	1.029805	1.003317
32	1.028763	1.002901	1.587469	1.030554	1.006127

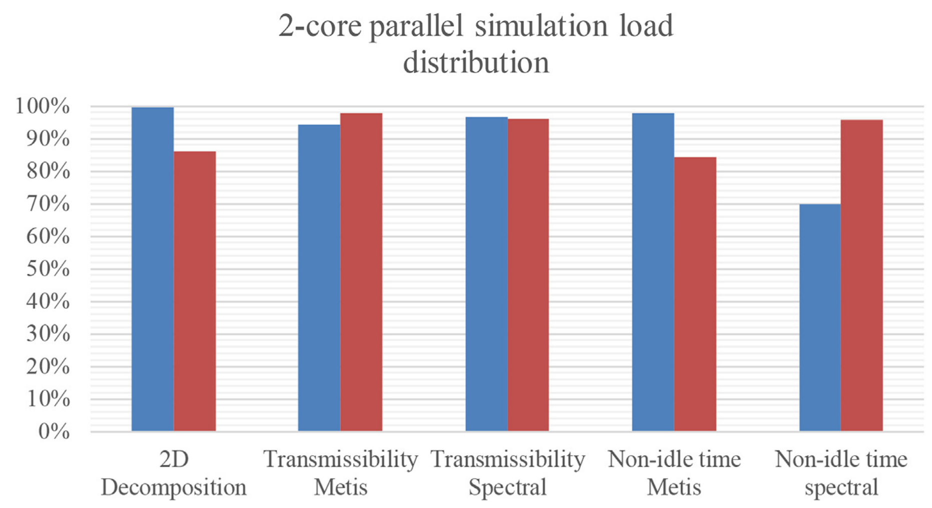


Fig. 4. 2-core parallel simulation load distribution.

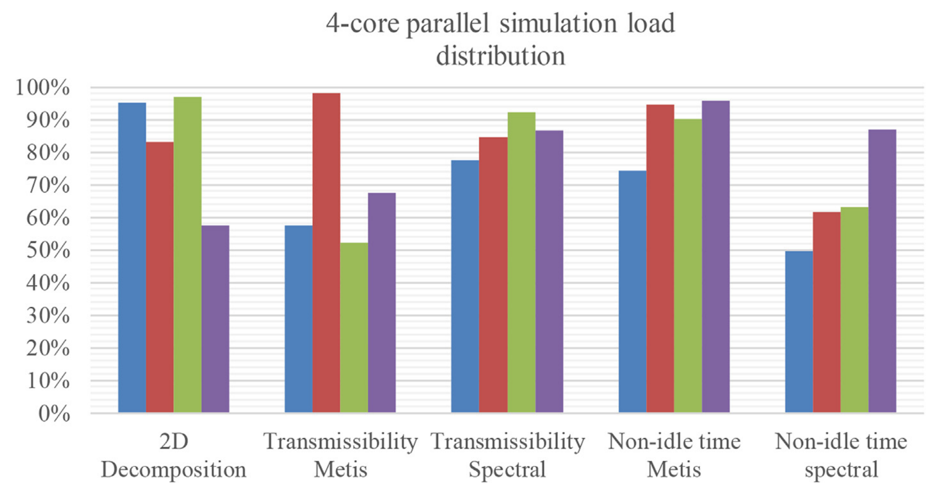


Fig. 5. 4-core parallel simulation load distribution.

and the idlest core will always be huge. However, spectral partitioning with transmissibility as weights always provides the smallest high-to-low ratio except for the 32-core scenario. It means that spectral partitioning is able to decrease the load discrepancy between cores that assume extreme loads. If compared with 2D decomposition, when it is 2 or 4 processes runs, the only significantly inferior partitioner is the spectral partitioning with non-idle time as weights. When the process number is 8 or beyond, Metis with transmissibility, Metis with non-idle time, spectral with transmissibility, and spectral with non-idle time are all better than 2D decomposition and have smaller high-to-low ratios than 2D.

4.2. Range

Fig. 10 shows the range of core load distributions. Small ranges indicate small imbalance. Similar to high-to-low ratio, range also tells the difference between the extremely high load and extremely low load cores. Again, the 8-partition simulation cases have the largest range in most cases. Taking ranges of 2D decomposition cases as reference, spectral is still the optimum load balancer. Spectral with transmissibility as weights has very small ranges whatever the number of cores. Spectral with non-idle time has small ranges when the number of cores is 8, 16, or 32.

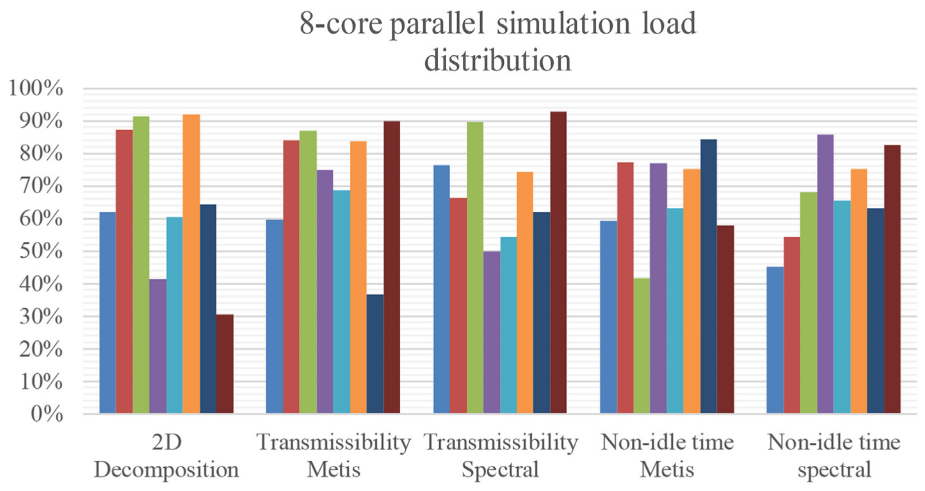


Fig. 6. 8-core parallel simulation load distribution.

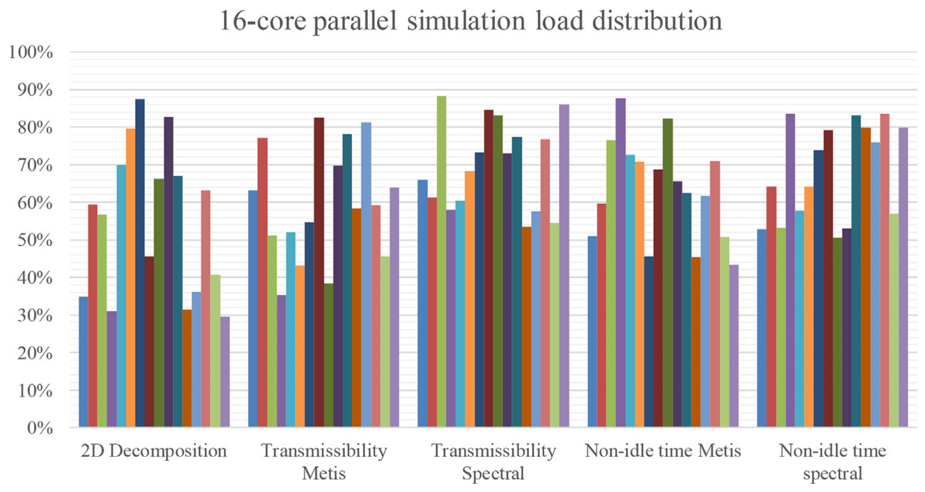


Fig. 7. 16-core parallel simulation load distribution.

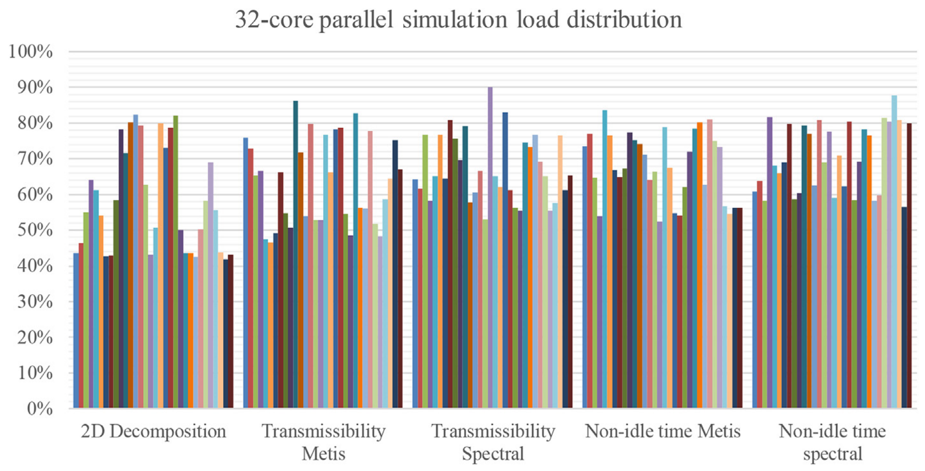


Fig. 8. 32-core parallel simulation load distribution.

4.3. Interquartile range

Fig. 11 shows the interquartile range comparison between different numbers of cores and between different partitioning methods. Interquartile range is used to assess the load dispersion and it is a very robust statistical measurement. The trimmed range

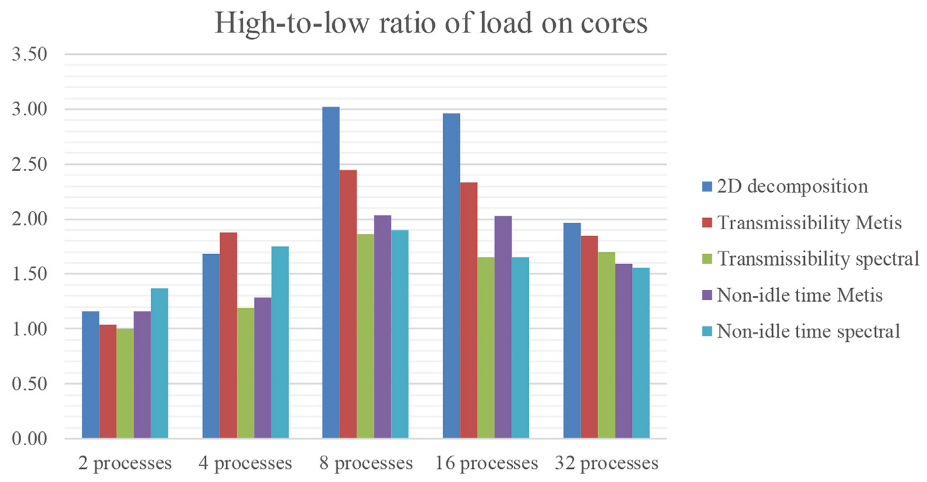


Fig. 9. High-to-low ratios of core loads.

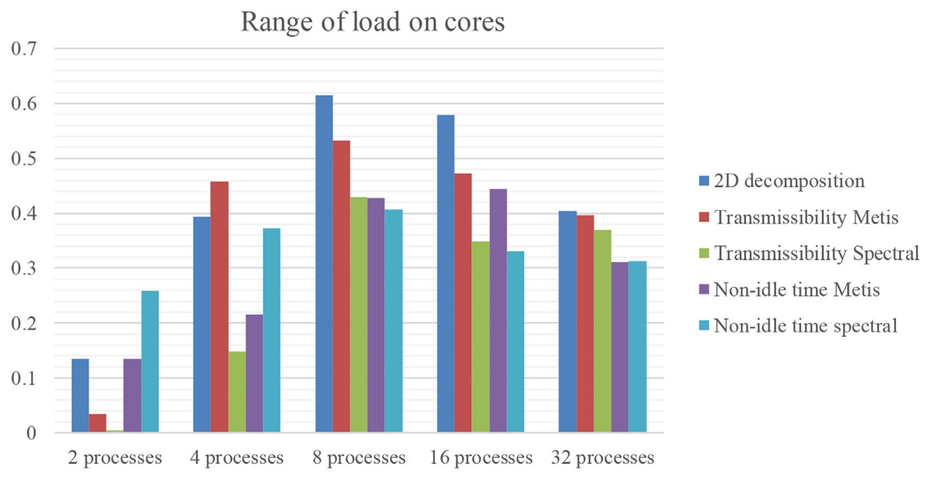


Fig. 10. Range of core loads.

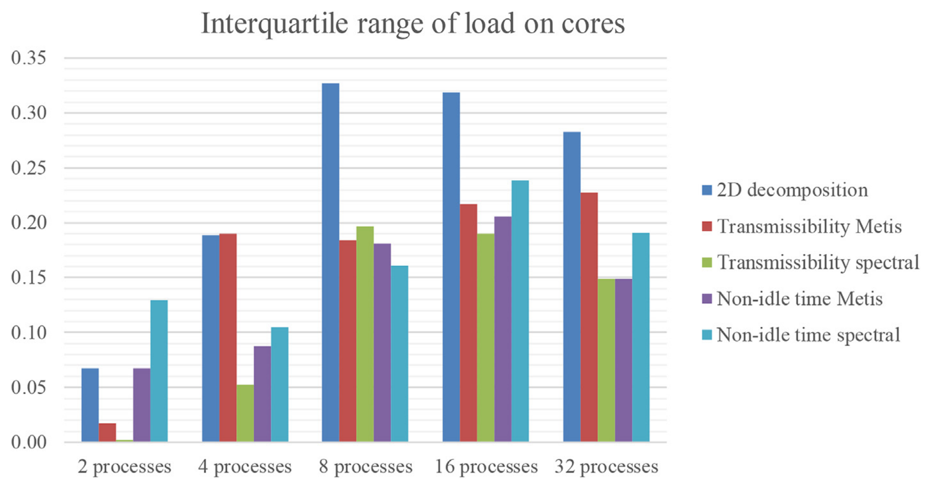


Fig. 11. Interquartile range of load on cores.

represented by interquartile range well depicts the middle fifty percent of the data and it is not affected by outliers. Thus, interquartile range is a good representation of the dispersion of the most important data.

From the results, it is observed that, in most of the cases, 2D decomposed parallel runs has the largest interquartile range or the greatest load imbalance. Both Metis and spectral manage to reduce interquartile ranges of core loads. In general, spectral partitioning

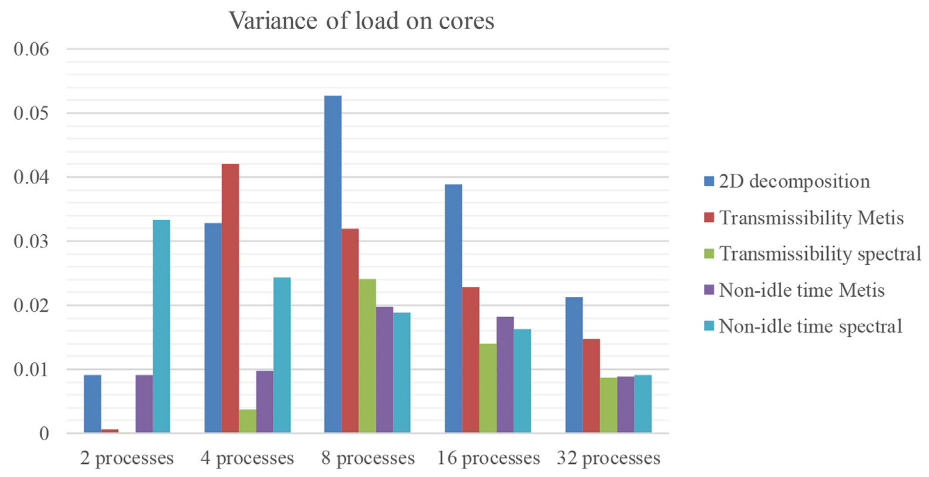


Fig. 12. Sample variance of core loads.

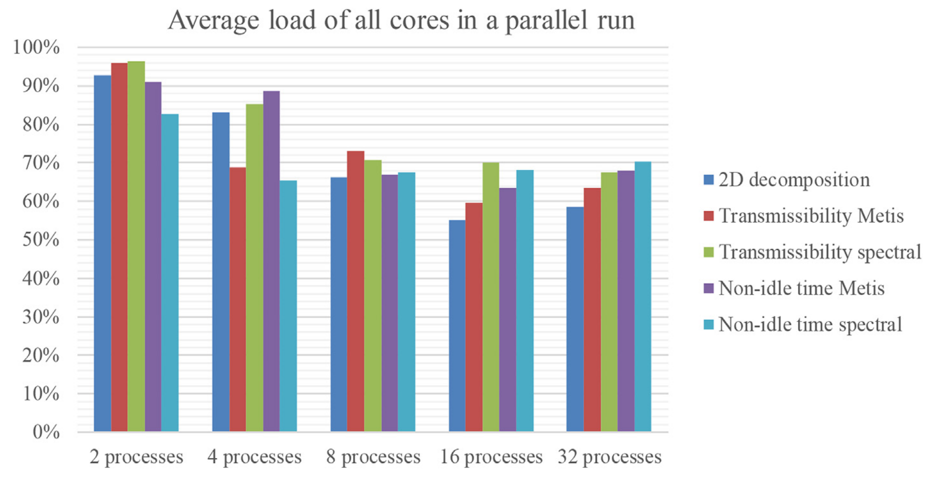


Fig. 13. Average core loads.

with transmissibility as weighting factor gives relatively low interquartile values in all cases and it is the most competent partitioning method that effectively reduces interquartile ranges.

4.4. Sample variance

Fig. 12 is the sample variance of core loads. Since the amount of load data has a limit, instead of population variance, sample variance s_{N-1}^2 is used. It can estimate the differences among all the cores in one parallel compositional simulation. Again, spectral partitioning provides relatively small sample variance values when compared with other partitioning methods and 2D decomposition. Metis partitioning methods also make the load imbalance smaller than 2D decomposition in most cases.

4.5. Average

Averages of core loads are also plotted so that how efficient cores are used can be visualized. The optimum case of efficient core usage has maximized non-idle time and minimized communication time. The average load deceases as number of partitions increases. This is due to the limited scalability of parallel implementation. Limited parallel scalability results in reduced average load and this trend is not affected by the type of load balancer in this study. However, it can be seen from Fig. 13 that at the cases where the processor core numbers are the same, 2D decomposition cases never have the highest average load. Spectral with transmissibility as weights gives relatively high average loads in all five processor number scenarios. Metis with transmissibility as weights, Metis with non-idle time as weights, and spectral with non-idle time as weights provided high average loads in certain scenarios while in several other scenarios, they have lower average loads than 2D decomposition.

Based on the five parameters studied, transmissibility spectral partitioning has the optimum load balancing results. It is the best load balancer for the model studied in this article.

4.6. Partition quality and load balance

Partition quality can be evaluated immediately after graph partitioners generate partitioned reservoir meshes. Eqn. (3) and Eqn. (4) are used to calculate partition quality factor. This factor depicts how well the weighted graph is divided into pieces. If every partition has the same sum of weight, the factor will be one. However, this factor is merely based on static reservoir description and it is not representative when dynamic reservoir description comes into effect.

Load distribution data are available after parallel simulation

Table 6
Load imbalance statistics.

Partition methods	2D	Transmissibility Metis	Transmissibility spectral	Non-idle time Metis	Non-idle time spectral
Number of cores	Range				
2	0.13525	0.03522	0.00415	0.13536	0.25832
4	0.39282	0.45815	0.14881	0.21480	0.37291
8	0.61546	0.53245	0.43005	0.42854	0.40714
16	0.57884	0.47212	0.34857	0.44409	0.33101
32	0.40418	0.39616	0.37002	0.31169	0.31292
Number of cores	Sample variance				
2	0.00915	0.00062	0.00001	0.00916	0.03336
4	0.03288	0.04207	0.00377	0.00977	0.02435
8	0.05275	0.03195	0.02412	0.01976	0.01890
16	0.03891	0.02289	0.01393	0.01822	0.01629
32	0.02133	0.01479	0.00869	0.00888	0.00911
Number of cores	High-to-low ratio				
2	1.15739	1.03739	1.00432	1.16051	1.36988
4	1.68306	1.87588	1.19226	1.28912	1.75141
8	3.02201	2.44943	1.86148	2.03218	1.90186
16	2.96295	2.33744	1.65331	2.02547	1.65539
32	1.96577	1.85143	1.69752	1.59464	1.55401
Number of cores	Interquartile range				
2	0.06763	0.01761	0.00208	0.06768	0.12916
4	0.18836	0.19007	0.05226	0.08720	0.10460
8	0.32673	0.18369	0.19687	0.18126	0.16104
16	0.31881	0.21711	0.18976	0.20544	0.23876
32	0.28267	0.22715	0.14886	0.14920	0.19048
Number of cores	Average				
2	0.92698	0.95944	0.96304	0.91101	0.82754
4	0.83123	0.68855	0.85182	0.88657	0.65358
8	0.66170	0.73030	0.70678	0.66986	0.67500
16	0.55068	0.59590	0.70083	0.63439	0.68204
32	0.58514	0.63572	0.67475	0.67888	0.70413

runs. The load imbalance status can be evaluated by range, variance, average, and high-to-low ratio. At this time, load imbalance and partition quality can be correlated. Thus we can know whether the partition quality factor is a good indicator of load balance status. Partition quality and load balance are correlated in Fig. 14. In the plot, high-to-low ratio stands for load imbalance. The larger the ratio, the stronger the load imbalance. The horizontal axis is partition quality factor minus one ($f_p - 1$). This parameter stands for partition quality and it is a value larger than 0. If the value is 0, graph partitioners distribute weights evenly into partitions. When it is larger than 0, weights distribution gets more uneven when the value gets larger. The data are from partition quality factors of Metis partitioning based on transmissibility and spectral partitioning based on transmissibility. Their corresponding high-to-low ratios are also used based on core load report.

A regression line is generated on the semi-log plot. In general, partition quality factor is positively correlated with high-to-low ratio. This means that in most cases, partition quality can roughly estimate the load balance that will occur in parallel simulation runs. However, there are still exceptions based on the plot. The exceptions can be explained by the fact that partition quality factor only takes information from static reservoir model while load balance status is decided by both static reservoir model data and dynamic simulation. Consequently, partition quality factor only predicts load balance status and it is not always accurate.

In addition, the range of load distribution and interquartile

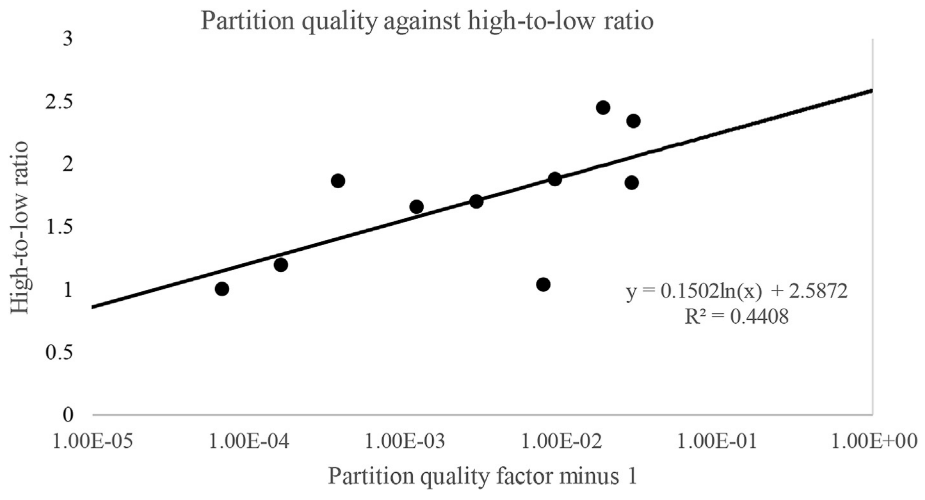


Fig. 14. Partition quality correlated with load balance.

range of load distribution for Metis and spectral are also plotted in Figs. 15 and 16. They both show that the partition quality is positively correlated to range and interquartile range. This is in accordance with the correlation of partition quality with high-to-low ratio. However, there can be occasions when it fails to predict the load imbalance level. Such occasions are represented by outliers in Figs. 14–16.

4.7. Grid-to-process assignment

In our study, graph partitioners largely decides load distribution and load balance status in parallel simulations. However, load balance status can also be affected by the way partitions are assigned to cores. The way partitions are assigned to cores is called grid-to-process assignment here. Once the partitioned mesh is ready, the partition G_i can be assigned to either core i or core j ($j \neq i$). There are four nodes in the machine and core i and core j can be either in the same node or in different nodes. To understand the effects of grid-to-process assignment on load balance, we studied 2D decomposed parallel simulation load balance results with six different assignment patterns. Table 7 shows how many partitions each node takes in the six assignment patterns. Since there are 16 cores in each node, each node can take up to 16 partitions in this set of parallel runs. Table 8 records the load balance statistics of the six patterns. Between the six runs, the only thing that changes is the way partitions are assigned to nodes. All other simulation parameters are the same.

Based on the load balance evaluation in previous sections, it is noted that the assignment pattern has effects on load balance. Each pattern has its own unique core load range, core load sample variance, and average core load. Assignment pattern 1, which assigns all partitions into one node in the parallel machine, has the smallest range and variance and the highest average load. However, this assignment pattern exerts tremendous pressure on the memory bandwidth within node 1. The memory architecture is shared memory within node 1 and the data accessing is largely infringed in this case. In this case, although core loads are large, many of the core processing time is actually wasted on data accessing due to exhausted memory bandwidth. Results show that parallel simulation with assignment pattern 1 takes 38% longer time than the simulation with assignment pattern 6.

Results also suggest that as long as we do not assign all partitions to only one node, the load balance statuses and core performances are nearly the same whatever the assignment pattern is. Assignment pattern 2, 3, 4, 5, and 6 have very close load balance statistics. Also, their simulation execution times are only different by less than 20 s.

Since in all our previous parallel simulations each node was not assigned for more than 8 partitions, the performance was not affected by memory bandwidth and grid-to-process assignment did not largely affect load balance status.

This section shows that the only major factor that affects load distribution and load balance in this study is the static load balancer.

5. Conclusion

The article introduced the application of several static load balancers and their implementation in compositional parallel simulation. The load balancers treated the reservoir grid as weighted and unstructured graph. The intuitive 2D decomposition parallel simulation was used as reference and it is concluded that Metis and spectral partitioning both improve load balance in certain cases. Some conclusions can be drawn as follow:

1. Transmissibility as weights helps reduce load imbalance.
2. Both static load balancers introduced give better load balance than 2D decomposition.
3. The optimum balancer is spectral partitioning with transmissibility as weights.
4. Kernighan-Lin algorithm improves partition quality and increases efficiency of parallel implementation.
5. In general, partition quality solely based on mesh decomposition is positively correlated to load balance from actual parallel runs.
6. Grid-to-process assignment pattern's effect on load distribution can be neglected in the study.

The study shows that certain graph partitioning methods can be used as static load balancer in parallel reservoir simulations and they can help improve the efficiency in the usage of processor cores. Since compositional simulation usually covers complex phase changes and behaviors of components, the proper application of load balancer improves the efficiency of simulation for multi-phase and multi-component models and makes it possible for simulators to process mega-scale compositional reservoir scenarios in a cost-

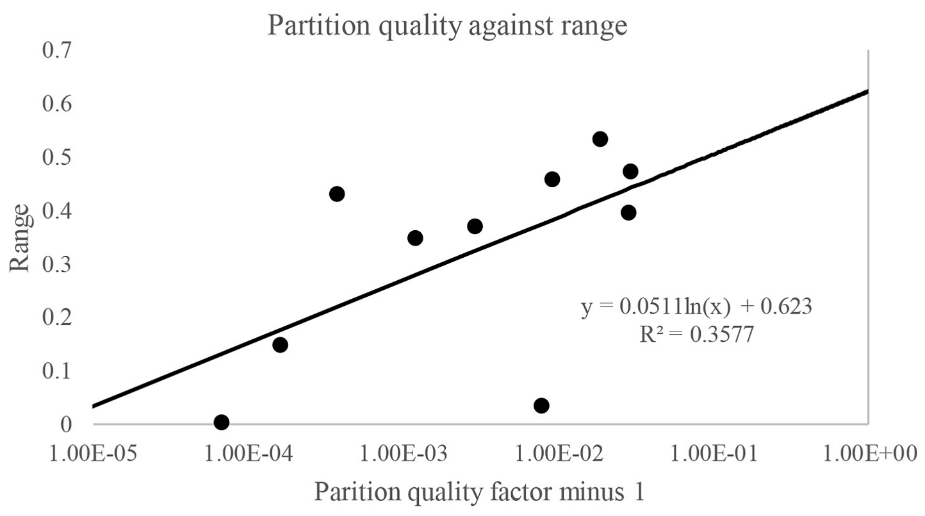


Fig. 15. Partition quality against range.

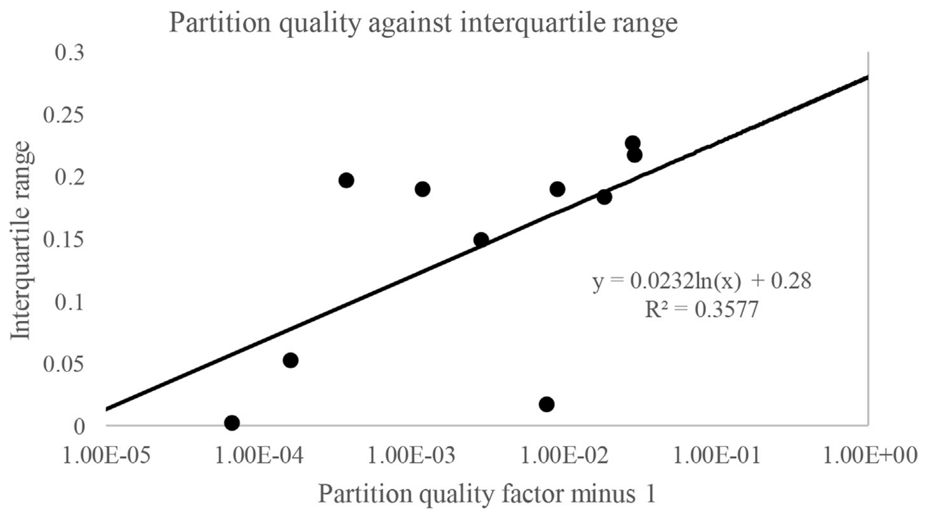


Fig. 16. Partition quality against interquartile range.

Table 7
Grid-to-process assignment patterns.

Assignment pattern	Node 1	Node 2	Node 3	Node 4
1	16	0	0	0
2	15	1	0	0
3	13	3	0	0
4	8	8	0	0
5	5	5	6	0
6	4	4	4	4

Table 8
Statistical comparison of load balance between grid-to-process assignment patterns.

Assignment pattern	Load balance statistics		
	Range	Variance	Average
1	0.442304	0.021057	0.662706
2	0.585042	0.034002	0.585343
3	0.584644	0.03585	0.571008
4	0.563652	0.037019	0.560232
5	0.562314	0.037194	0.554838
6	0.578836	0.038912	0.550683

effective manner.

Acknowledgment

We want to thank Crisman Institute for Petroleum Research of Texas A&M University for partially funding the research.

Nomenclature

A	cross-section area of a cell that is perpendicular to flow direction
K	permeability
K_r	relative permeability
l	length of the cell that is along the flow direction
B	formation volume factor
μ	viscosity
T	transmissibility
IMPES	implicit pressure and explicit saturation
ILU	incomplete LU preconditioner
EOS	equation of state
G	graph
V	vertices
E	Edges
d_i	degree of vertex i
$L_{i,j}$	Laplacian matrix
$T_{non-idle}$	non-idle time on a processor core
$T_{Elapsed}$	total elapsed time on a processor core
$T_{Message\ passing}$	communication time on a processor core during parallel simulation
f_p	partition quality factor
N_p	number of partitions
w_i	the weights of cells in G_i
W_i	the total weights of G_i
G_i	the i th graph of the partitioned initial graph
N_i	the number of cells in G_i
s^2_{N-1}	sample variance

References

Fiedler, Miroslav, 1973. Algebraic connectivity of graphs. Czech. Math. J. 23 (2), 298–305.
Guo, X., 2015. A Study of Load Imbalance for Parallel Reservoir Simulation with Multiple Partitioning Strategies. MS thesis. Texas A&M University, College Station, Texas (August 2015).
Halliburton, 2015. Nexus User Guide.
Hendrickson, B., Leland, R., July 1995. The Chaco User's Guide Version 2.0. Sandia. http://prod.sandia.gov/techlib/access-control.cgi/1995/952344.pdf (accessed 18 February 2015).
Karypis, G., 2013. METIS: a Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-reducing Orderings of Sparse Matrices Version 5.1.0. UMN. http://glaros.dtc.umn.edu/gkhome/metis/metis (accessed 18 February 2015).
Karypis, G., Kumar, V., 1998. Multilevel k-way partitioning scheme for irregular graphs. J. Parallel Distr. Com. 48 (1), 96–129. http://dx.doi.org/10.1006/jpdc.1997.1404.
Killough, J.E., Wheeler, M.F., 1987. Parallel iterative linear equation solvers: an investigation of domain decomposition algorithms for reservoir simulation. In: Presented at SPE Symposium on Reservoir Simulation, 1–4 February, San Antonio, Texas. http://dx.doi.org/10.2118/16021-MS. SPE-16021-MS.
Liu, H., Wang, K., Chen, Z., Jordan, K., Luo, J., Deng, J., 2015. A parallel framework for reservoir simulators on distributed-memory supercomputers. In: SPE/IATMI Asia Pacific Oil & Gas Conference and Exhibition, October, Nusa Dua, Indonesia. SPE-176045-MS.
Lu, P., Beckner, B.L., Shaw, J.S., et al., 2008. Adaptive parallel reservoir simulation. In: Presented at International Petroleum Technology Conference, 3–5 December, Kuala Lumpur, Malaysia. http://dx.doi.org/10.2523/IPTC-12199-MS. IPTC-12199-MS.
Pothen, Alex, Simon, Horst D., Liou, Kang-Pu, 1990. Partitioning sparse matrices with eigenvectors of graphs. SIAM J. Matrix Anal. Appl. 11 (3), 430–452. http://dx.doi.org/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).

Sarje, A., Song, S., Jacobsen, D., et al., 2015. Parallel performance optimizations on unstructured mesh-based simulations. Proc. Comput. Sci. 51 (0), 2016–2025. In: http://dx.doi.org/10.1016/j.procs.2015.05.466.

Shiralkar, G., Fleming, G., Watts, J., et al., 2005. Development and field application of a high performance, unstructured simulator with parallel capability. In: Presented at the SPE Reservoir Symposium, 31 January-2 February, Houston, Texas, USA. http://dx.doi.org/10.2118/93080-MS. SPE-93080-MS.

Tallent, N.R., Adhianto, L., Mellor-Crummey, J.M., 2010. Scalable identification of load imbalance in parallel executions using call path profiles. In: Presented at 2010 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 13– 19 November. http://dx.doi.org/10.1109/SC.2010.47.

Tarman, M., Wang, K., Killough, J., et al., 2011. Automatic decomposition for parallel reservoir simulation. In: Presented at the SPE Reservoir Symposium, 21–23 Feburary, The Woodlands, Texas, USA. http://dx.doi.org/10.2118/141716-MS. SPE-141716-MS.

Wang, Y., 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. http://dx.doi.org/10.2118/163585-PA. SPE-163585-PA.

Zhang, Y., Kameda, H., Hung, S.-L., 1997. Comparison of dynamic and static load-balancing strategies in heterogeneous distributed systems. IEEE Proc. – Comput. Digit. Tech. 144 (2), 100–106. http://dx.doi.org/10.1049/ip-cdt:19970951.