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

ARTICLE INFO

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

ABSTRACT

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/

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