A MapReduce and MPI Programming Model for Distributed Large Scale 3D Mesh Processing

Da Qi Ren[1], Zane Wei[2] and Dennis D. Giannacopoulos[3], *Senior Member, IEEE*

[1]The US R&D Center, Huawei Technologies, Santa Clara, CA 95050 USA, daqi.ren@huawei.com
[2]The US R&D Center, Huawei Technologies, Santa Clara, CA 95050 USA, weizhulin@huawei.com
[3]Department of ECE, McGill University, Montreal H3A0E9 Canada, dennis.giannacopoulos@mcgill.ca

Developing a high performance platform for large-scale, high-intensity data processing is a priority for researching cost-effective parallel finite element methods (FEM). This paper introduces an efficient MapReduce-MPI based strategy for parallel 3D finite element mesh processing, demonstrates the potential benefits of this approach for optimally utilizing system resources. Preliminary experimental results show that the new platform improves speedup over a range of problem sizes and different machine numbers. In detail, this paper includes the design of scalable Hadoop algorithms for 3D FEM mesh processing; experimental evaluation of these algorithms on computer clusters; and discussions on the benefits and challenges of developing 3D FEM algorithms using the MapReduce-MPI model.

***Index Terms*—Mesh; MapReduce; MPI; Finite Element Method.**

I. Introduction

PARALLEL mesh processing can be beneficial for simulating complex electromagnetic problems required for 3D finite element solutions. However, the high volume of initial mesh data sets can significantly degrade system performance and diminish the benefits. Moreover, the costs of mesh processing are highly dependent on the underlying parallel algorithm as well as the system architecture. Studying a workable design for high performance large-scale mesh data processing is important for the performance of FEM simulations.

MapReduce has been used in multiple big data application domains to efficiently process huge data sets. Hadoop (when MapReduce is implemented) incorporates Hadoop Distributed File System (HDFS) has superb capabilities that can work with intensive data via a distributed file system. This enables MapReduce the possibility to be used for processing scientific HPC over intensive data sets. On the other side, scientific and engineering applications are usually both data- and computation-intensive; and require multiple cycles and chained tasks. Hadoop cannot fully address problems with iterative structures by itself. These limitations are inefficient for scientific computations directly wrapped in MapReduce tasks because iterative computations cannot be rapidly distributed over a HDFS cluster in the same way that it can over a traditional HPC cluster. [1]

MPI programs are designed to distribute data to different nodes, compute asynchronously in parallel, and then collect the results back. MPI has no data locality. MapReduce with HDFS duplicates data so that computation in local storage can occur in a way that streams off the disk straight to the

TABLE 1: ALGORITHM III
HADOOP MANAGER ON MASTER NODE

MapReduce MPI master node Description: enables the master node to perform MPI task management; locates data nodes; assigns task functions to find the key; and updates mesh information between data nodes.

```
1:  for each node n in nodelist do {
2:        MPI _INIT; to initialize MPI functions
3:        }
4:  for  each subdomain, do{
5:            Read the local data file;
6:            Compute Key for each iteration; as the key input
7:  }
8:  for  Node i in MPI task pool  on master node, do{
9:           if status = ready, do  {
10:          update task tracker and start to send task}
10:  }
11:  for  Node i update adjunct list , do{
12:           update adjunct edge and point list ;
13:  }
```

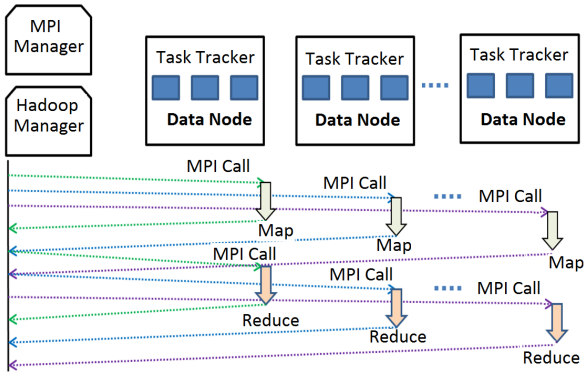
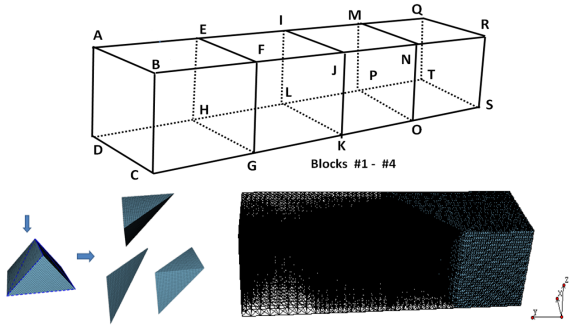


Fig.1 Hadoop MPI programming model: MPI manages hadoop tasks; the tasks are distributed among the group of MPI communicators.



Blocks	A	B	C	D
Sub-domain	ACDH,ABCH ABEH,BCGH BEGH,BEGF	EGHL,EFGL EFIL,FGKL FIKL,FIKJ	IKLP,IFKP IFMP,JKOP JMOP,JMON	MOPT,MFOT MFQT,NKST NQST,NQSR

Fig.2 3D HTO FEM mesh refinement on a rectangular cavity: domain decomposition; workload distribution; and the final refinement results.

processor. MPI delivers a network-bound performance, MapReduce exploits local storage to avoid network bottlenecks when working with big data – it reduces network use and maximizes efficiency. For the above reasons, adding MPI capabilities to the MapReduce framework becomes a priority.

Efforts to integrate MPI into MapReduce framework have been made in various formats by researchers. Ye et al. [2] launched MPI tasks on a Hadoop cluster via the Hadoop streaming interface using a modified OpenMPI, then compared the performance of Gradient Boosted Decision Trees (GBDT) algorithms with MPI-based and MapReduce-based implementation. In [3], T. Hoefler and al. discussed common strategies for implementing MapReduce runtime, and proposed an optimized implementation scenario of MapReduce on top of MPI.

II. MapReduce MPI Programming Model

The programming paradigm in Fig.1 enables MPI to control Hadoop function calls and task distributions. MPI gives precise control over the memory and format of the data allocated by each processor during a MapReduce operation. Fig.1 illustrates the data flow of a typical MapReduce computation where the input files to a Hadoop program are fetched from the Master Node. Compute nodes executing instances of the `map()` function produce intermediate key-value pairs that are stored on local disks. On the other side, compute nodes executing instances of the `reduce()` function use remote procedure calls (RPCs) to copy data to their local disks, and store output back on the HDFS. An HDFS server, map function, and reduce function may be executed on the same or different nodes. [4] [5]

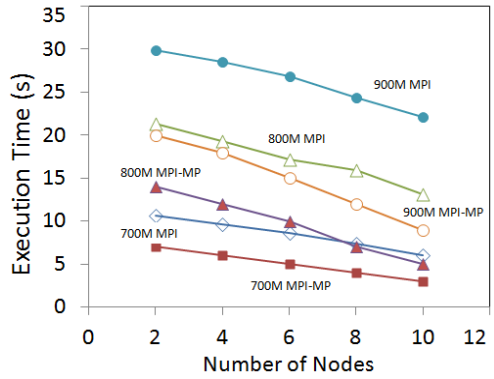


Fig.3 Comparison of HTO mesh refinement execution time between using the MapReduce MPI (MPI-MP) and regular MPI, on the same system platform. The numbers of mesh element executed are 600M, 700M, and 800M.

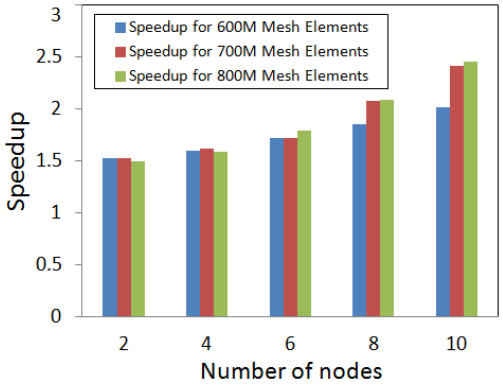


Fig.4 Comparison of performance speedup between using the MapReduce MPI (MPI-MP) and regular MPI, on the same system platform. The numbers of mesh element executed are 600M. 700M. and 800M.

III. Results

Our approach is demonstrated by using MapReduce-MPI method with Hierarchical Tetrahedral and Octahedral (HTO) mesh performing on a rectangular resonant cavity, as shown in Fig.2. The cavity was initially discretized into four smaller rectangular blocks; each of these blocks was subdivided into six tetrahedra. The resulting 24-tetrahedral mesh sub-domains are distributed in the parallel system, and executed on 2 to 10 slave nodes. The algorithm specified in Tab.1 has been implemented. It enables the master node to control edge and vertex load, update the shared edge and vertex, prepare and deliver key values, and manage MPI. Performance measurements are performed on the real symmetric cluster system. Each node has the following configuration:

CPU	Dual Xeon E5-2440	**Memory Size**	64GB RAM
DISK	SATA 1TB x 8	**Inter-node Speed**	10 GE
OS	SLES 11 SP1 (3.0.13-0.27)		
MPI	MPICH2	**Hadoop Version**	Hadoop-0.20.2

Results indicate that MapReduce-MPI provides a combined storage, processing and analysis system that can respond to the growth in data volume and application diversity. When 600M, 700M and 800M HTO tetrahedral elements are produced on 2 to 10 slave nodes, the average speedup of 1.5 to 2.4 is achieved comparing with regular distributed computing with MPI. The results are illustrated in Figs.3-4.

IV. Conclusion

A MapReduce-MPI model has been designed and implemented to enhance the performance of mesh processing for 3D FEM electromagnetics. Specifically MPI capabilities are added into MapReduce framework in order to manage the distributed processing on shared or distributed memories and storages. Detailed algorithms including domain decomposition, data allocation pattern and distributed computing methods have been introduced. The experimental results have been measured on real systems, the performance advantage of using the MapReduce-MPI model has been validated.

References

[1] K. Shvachko, Hairong Kuang, S. Radia, R. Chansler, "The Hadoop Distributed File System", The 26th IEEE Symposium on Mass Storage Systems and Technologies (MSST). Incline Village, NV, 2010.

[2] T. Hoefler, A. Lumsdaine, and J. Dongarra, *"Towards Efficient MapReduce Using MPI"*, Proceedings of the 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface. pp. 240 – 249, Springer-Verlag, Berlin, Heidelberg. ISBN: 978-3-642-03769-6, 2009.

[3] J. Ye, J.-H. Chow, J. Chen, and Z. Zheng, "Stochastic gradient boosted distributed decision trees", CIKM'09. pp. 2061-2064, Hong Kong, China, 2009.

[4] MapReduce-MPI library, http://mapreduce.sandia.gov/, accessed Sept 10, 2013.

[5] Java binding in OpenMPI, http://www.open-mpi.org/faq/?category=java accessed Sept 10, 2013.