MapReduce Implementation of Prestack Kirchhoff Time Migration (PKTM) on Seismic Data

Nikzad Babaii Rizvandi[1,2], Ali Javadzadeh Boloori[1,2], Najmeh Kamyabpour[3], Albert Y.Zomaya[1]

[1]Centre for Distributed and High Performance Computing, School of Information Technologies, University of Sydney, NSW 2006, Australia
[2] Networked Systems Theme, National ICT Australia (NICTA), Australian Technology Park, Sydney, NSW 1430, Australia
[3] iNEXT-Centre for Innovation in IT Services and Applications, University of Technology, Sydney, NSW 2007, Australia

{nbab6945, ajav4801}@uni.sydney.edu.au
albert.zomaya@sydney.edu.au
najmeh@it.uts.edu.au

***Abstract*- The oil and gas industries have been great consumers of parallel and distributed computing systems, by frequently running technical applications with intensive processing of terabytes of data. By the emergence of cloud computing which gives the opportunity to hire high-throughput computing resources with lower operational costs, such industries have started to adopt their technical applications to be executed on such high-performance commodity systems. In this paper, we first give an overview of forward/inverse Prestack Kirchhoff Time Migration (PKTM) algorithm, as one of the well-known seismic imaging algorithms. Then we will explain our proposed approach to fit this algorithm for running on Google's MapReduce framework. Toward the end, we will analyse the relation between MapReduce-based PKTM completion time and the number of mappers/reducers on pseudo-distributed MapReduce mode.**

Keywords- MapReduce, PKTM, Seismic imaging.

I. INTRODUCTION

The main use of high performance computing in both oil and gas industries for seismic imaging of the earth's subsurface is motivated by the business requirement to make best decisions to drill wells during petroleum exploration and production. Since drilling each oil/gas well in exploration areas costs several tens of millions of dollars, producing high-quality seismic images in a reasonable time can notably decrease the risk of drilling a "dry hole" [2]. Similarly, these images are important as they can improve the position of wells in billion dollar producing oil/gas fields.

The idea behind seismic imaging, or migration, is to place reflected energy at subsurface positions where it originated. Ideally, seismic data have correct reflection magnitude and phase after migration. To have a correct reflection, a good migration algorithm has to correctly estimate the shape of wave propagation in the sub-surface, i.e. how to utilize the wave equation in some form. Generally, some assumptions about the Earth model are made in almost all migration algorithms to simplify numerical operations. Depending on the complexity of migration data, these assumptions may make the final outcome of the migration inaccurate.

The processing of seismic data may be split into two steps [2]. In the first step, signal processing algorithms are applied to normalize the signal over the whole seismic survey and to raise the signal-to-noise ratio, which is initially very low. There are hundreds of mathematical algorithms which geophysical experts can choose a particular one for seismic data. The aim in the second step is to correct the influences of subsurface velocity changes on the wave propagation through the earth by using Pre-Stack Kirchhoff Time Migration (PKTM), one of the most popular imaging approaches in the seismic data processing industry. Either of these steps needs an iterative process to get more clear images of the complicated subsurface structures.

Seismic trace is basically a convolution of source function and earth's response. "Gather" is defined as a group of traces that share common attributes [4]. In most of modern seismic processing systems, seismic algorithms are implemented by processing modules in form of shared libraries. Zhao et al [4] have categorized these processing modules into three classes, based on their data dependency:

- Single-trace modules, which require one input trace at a time and outputs one trace, several traces, or null.
- Gather modules which require one input gather and output one gather, one trace, or null.
- Global modules, which depend on traces from separate gathers or the whole data set. PKTM belongs to this module.

A typical seismic imaging job is computationally very expensive which needs to process terabytes of data and requires Gflop-months of computation prior to being interpreted by the experts. Consequently, parallel computing techniques are used to process the data in order to reduce the time compilation.

Nowadays, cloud computing has received a lot of attention from both research and industry due to the deployment and growth of commercial cloud platforms. Cloud services enable customers to change, or dynamically supply their own IT infrastructures with large choices of computational and storage resources that are accessible anytime. In this paper, we propose a method to develop PKTM seismic imaging algorithm on Google MapReduce as one of the famous distributed data processing frameworks on cloud infrastructure. After investigating how to fit both forward and inverse PKTM seismic algorithms on MapReduce

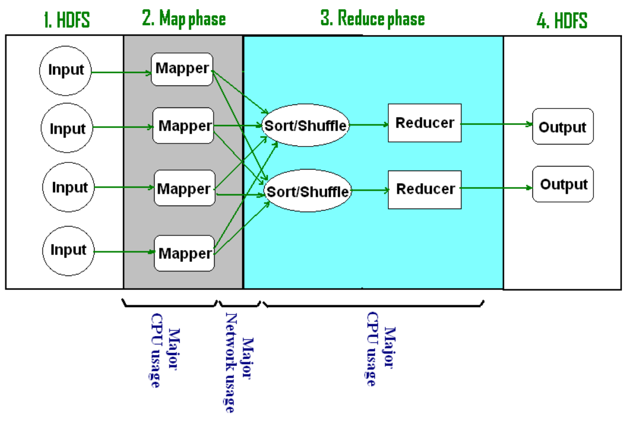
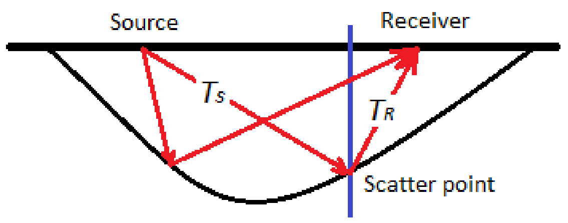
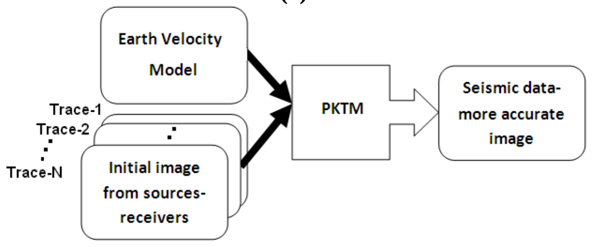


Figure 1.MapReduce workflow [1, 3]

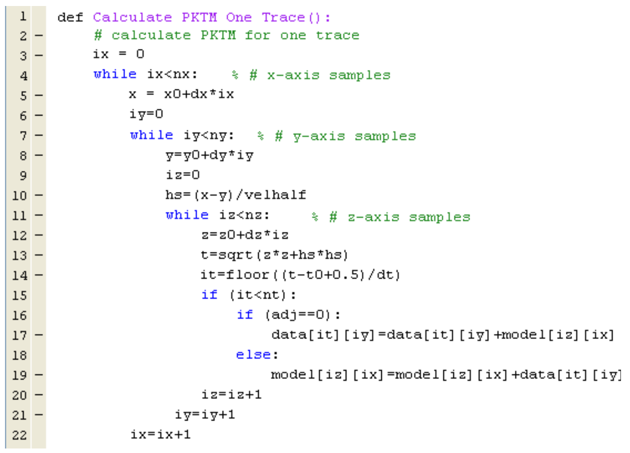


(a)


Earth Velocity Model
Trace-1
Trace-2
Trace-N
Initial image from sources-receivers
PKTM
Seismic data-more accurate image

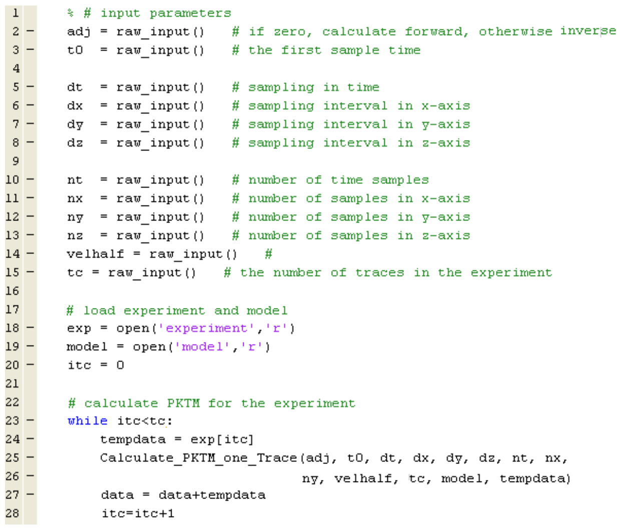
(b)

Figure 2. (a) the relation between source, receiver and scatter point and the related migration curve[2], (b) the big picture of using PKTM in seismic data processing.



```
def Calculate PKTM One Trace():
    # calculate PKTM for one trace
    ix = 0
    while ix<nx:    % # x-axis samples
        x = x0+dx*ix
        iy=0
        while iy<ny:  % # y-axis samples
            y=y0+dy*iy
            iz=0
            hs=(x-y)/velhalf
            while iz<nz:     % # z-axis samples
                z=z0+dz*iz
                t=sqrt(z*z+hs*hs)
                it=floor((t-t0+0.5)/dt)
                if (it<nt):
                    if (adj==0):
                        data[it][iy]=data[it][iy]+model[iz][ix]
                    else:
                        model[iz][ix]=model[iz][ix]+data[it][iy]
                iz=iz+1
             iy=iy+1
        ix=ix+1
```

(a)



```
% # input parameters
adj = raw_input()   # if zero, calculate forward, otherwise inverse
t0  = raw_input()   # the first sample time

dt  = raw_input()   # sampling in time
dx  = raw_input()   # sampling interval in x-axis
dy  = raw_input()   # sampling interval in y-axis
dz  = raw_input()   # sampling interval in z-axis

nt  = raw_input()   # number of time samples
nx  = raw_input()   # number of samples in x-axis
ny  = raw_input()   # number of samples in y-axis
nz  = raw_input()   # number of samples in z-axis
velhalf = raw_input()   #
tc = raw_input()   # the number of traces in the experiment

# load experiment and model
exp = open('experiment','r')
model = open('model','r')
itc = 0

# calculate PKTM for the experiment
while itc<tc:
    tempdata = exp[itc]
    Calculate_PKTM_one_Trace(adj, t0, dt, dx, dy, dz, nt, nx,
                                ny, velhalf, tc, model, tempdata)
    data = data+tempdata
    itc=itc+1
```

(b)

Figure 3. the sequential calculation of PKTM algorithm for an experiment including many traces: (a) calculate forward/inverse PKTM for one trace in the experiment (some parts come from [1]), (b) merge the result of (a) for all traces to form the final image of the experiment. The both algorithms have been written in python.

framework, we implement these algorithms on Apache Hadoop implementation of MapReduce.

II. MAPREDUCE

MapReduce [5-6], introduced by Google in 2004, is a framework for processing large quantities of data on distributed systems. Each computation at this framework consists of two major phases:

- "Map" phase: after copying the input file to the MapReduce file system and split the file into smaller files, the data inside the files are converted into $< zkey, values >$ format (e.g. key can be the line number and values are the data in the lines). These $< key, values >$ pairs enter the mappers, where the first part of processing is applied to them. One of the major advantages of MapReduce is that the mappers are independent. Therefore in theory, it gives a good opportunity for parallelization. However, this parallelization can be bounded because of the data source and/or the numbers of CPUs close that data.
- "Reduce" phase: After finishing the Map phase, a network intensive job starts in order to send the intermediate $< key, values >$ coming from mappers to the reducers. Then, depending on the MapReduce configuration, a sort/shuffle stage may be applied. Subsequently, the map operations with the same key will be presented to the same reducer, at the same time. The result will then be written in output files (typically one output file) in the file system.

MapReduce has been utilized in a quite wide range of applications including multiple sequence alignment in bioinformatics [7], distributed sorting of data, distributed searching/indexing, web linked-graph reversal, web access log statistics [8], and machine learning [9] due to its outstanding fault tolerance capabilities, scalability and ease of use. Generally, MapReduce has been designed for computing on significantly large quantities of data instead of doing complicated computations on a small amount of data [10]. Therefore MapReduce is more appropriate for seismic imaging algorithms where the complexity of algorithms is moderate but the size of data is huge.

III. PRESTACK KIRCHHOFF TIME MIGRATION (PKTM)

Prestack Kirchhoff Time Migration (PKTM) is known as an efficient migration algorithm for processing seismic data due to its I/O flexibility and target-orientation [11]. However, PKTM is computationally expensive, resulting in a long execution time. This problem becomes worse when iterative update of the velocity model is required.

Several sources, distributed in the region, send powerful sounds into the ground to both estimate subsurface conditions and possibly detect high concentrations of corruption [12]. Receivers, which are microphones, hear “echoes” coming back through the ground. Meanwhile, the intensity and time of these echoes are recorded on storage units like computers (Figure 1-a). The data then is converted to images of the geological structure. The outcome of these initial images, however, is not accurate enough in details as sound speed in the ground is not the same for all sound traces. Therefore, a post-processing stage is needed to correct the images by applying the earth velocity model to the initial images and then forming new but more accurate images. Figure 2-b describes this post processing stage where PKTM is the key technique to merge the initial images and the earth velocity model to form new images.

If a point in an input trace (x_i) has the travel time of t, then the assumption in PKTM is that the energy in this point is the superposition of all the underground scatter/reflection points with the same travel time (t)[13]. The main idea behind the migration processing, including PKTM, is to spread the points on an input trace to all possible scatter/reflection points in the 2D space. The PKTM repeats this process for all points on all input traces and adds each resulting contribution into the output image. There are two types of PKTM [4]:

- *Forward PKTM:* In forward PKTM, the velocity model and initial images are used as input. The output is new accurate images produced as:
  $$data[it][iy] = data[it][iy] + model[iz][ix]$$
- *Inverse PKTM:* the same as forward PKTM, inverse PKTM uses both earth velocity model and initial images as input. The output is modification of the earth velocity model as:
  $$model[iz][ix] = model[iz][ix] + data[it][iy]$$

Where
$$\begin{cases} 0 \le it \le nt \\ 0 \le ix \le nx \\ 0 \le iy \le ny \\ 0 \le iz \le nz \end{cases}$$
The parameters nt, nx, ny, and nz represent number of time samples, numbers of samples in x-axis, number of samples in y-axis, and number of samples in z-axis, respectively. Both algorithms are derived from the following circle-hyperbola relation [1]:
$$t^2 = \tau^2 + \frac{x^2}{v^2}$$
In forward PKTM, the migration code is considered “hyperbola-like” since parameter τ is given and t is solved for. The key thought behind PKTM is that each output sample (migrated) is visited only once by each input sample. As each input-output pair is independent from other pairs, the problem becomes more suitable for parallelization.

If $X = \{x_1, x_2, \dots, x_T\}$ and $Y = \{y_1, y_2, \dots, y_T\}$ are assumed as sets of initial (un-migrated) and corrected (migrated) data traces, respectively, and $\mathcal{F}(.)$ is the migration function (here PKTM), then the relation between X and Y is $Y = \mathcal{F}(X)$.

The related output of an initial trace x_i is $y_i = \mathcal{F}(x_i)$ which depends only on the initial trace. This fact can be used by MapReduce to parallelize the seismic imaging algorithm as each initial trace (x_i) is assigned to one mapper in order to migrate it into its migrated output (y_i). Consequently, the final migration image is the sum of all the migrated traces [14]:
$$Y = \sum_i y_i = \sum_i \mathcal{F}(x_i)$$
The algorithm to produce migrated image by PKTM is heavily time-consuming due to the huge number of iterations at runtime. Because of the mathematical complexity of PKTM which also needs some information about geosciences, the python code for sequentially calculating both forward and inverse PKTM in a complete experiment has been described in Figure 3.

IV. MAPREDUCE IMPLEMENTATION OF PKTM

In general, any algorithms that involves doing operations on a large set of data, where the problem can be broken down into smaller independent sub-problems can be described in MapReduce framework. More specifically, Chu et al in [9] explained that machine learning algorithms which can be written in a certain summation form can be parallelized on MapReduce. We follow the same concept by dividing the PKTM algorithm in a summation form.

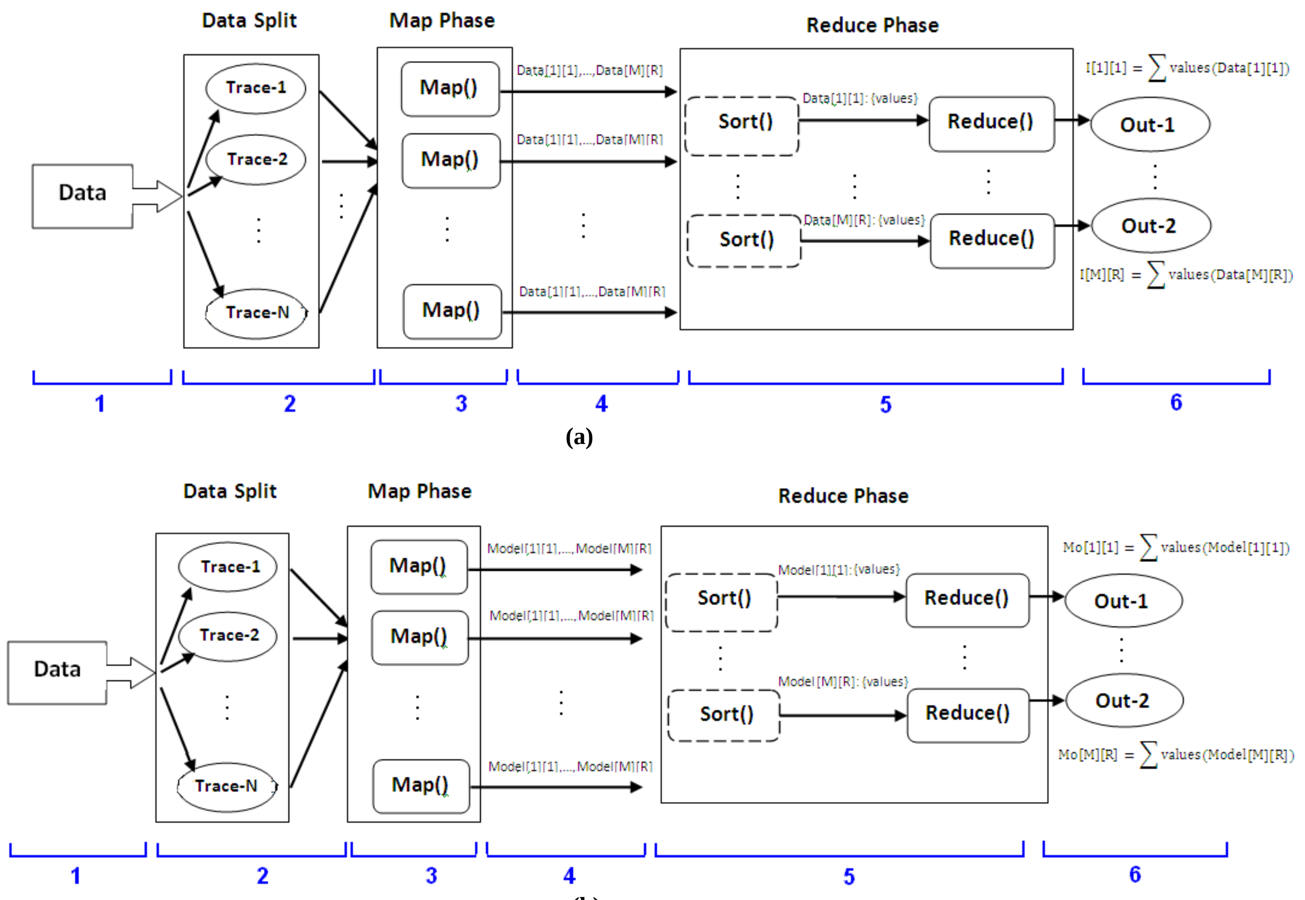


Figure 4. The proposed MapReduce framework to calculate (a) forward PKTM and (b) inverse PKTM for an experiment with N traces

A. Forward PKTM

As mentioned before, in both forward and inverse PKTM, the initial velocity model and data values are utilized as the input of MapReduce framework. The output of forward PKTM is data modification while the output of inverse PKTM is modification of the velocity model. Figure 4-a shows the proposed MapReduce framework to develop Forward PKTM on MapReduce cluster regarding to the algorithm in Figure 3-a as:

- Step-1: For the current experiment, MapReduce framework input consists of a set of data traces (*Trace 1*, *Trace 2*,…, *Trace N*). If there are K sources and L receivers, then the number of obtained traces will be $N = K * L$.
- Step-2: The primary key of the mappers in Map phase is the index of a trace and the value is the sampling values of that trace.
- Step-3: In the mappers, each trace is migrated by using sequential forward PKTM (Figure 3-a). In other words, map function is a forward migration algorithm working on one trace.
- Step-4: The output of a mapper is values for $Data[1][1], \dots, Data[M][R]$. Some of the elements may have zero value.
- Step-5: A set of $Data[i][j]$, $1 \le i \le M$, $1 \le j \le R$ is the output of mappers and it may be repeated several times (with different values) as each receiver is involved in several traces. In sort and group stage, the occurrences of each $Data[i][j]$ among all traces are sorted and grouped. In another word, the output of each mapper is sorted based on the order of the $Data[i][j]$ indexes, e.g. $Data[1][1], Data[1][2], \dots, Data[M][R]$. Also, the values of each element are the value of each occurrence of that element. For example in Figure 3, $Data[1][1]$ is repeated three times with values: $\{value_1\ , value_2, value_3\}$.
- Step-6: In Reduce phase, the values of all occurrences of each data ($Data[i][j]$) are summed and form the final image for the experiment.

B. Inverse PKTM

Figure 4-b represents the proposed MapReduce framework for inverse PKTM. As can be seen, both forward and inverse MapReduce frameworks have the same data flow except that:

- Step-3: In Map phase (step-3), reverse PKTM is applied on the traces (adj=1 in Figure 3-a, line 16) in order to update the velocity model.
- Step-6: The final image is the update of velocity model, not data.

V. EXPERIMENTAL RESULTS

A. *Experimental setting*

Our proposed method has been implemented and evaluated on a pseudo-distributed Map-Reduce framework. In such framework, all five Hadoop daemons (namenode, jobtracker, secondary namenode, datanode and task tracker) are distributed over cores/processors of a single laptop PC. Hadoop writes all files to Hadoop Distributed File System (HDFS), and all services and daemons communicate over local TCP sockets for inter-process communication. In our evaluation, the system runs Hadoop version 0.20.2 which is Apache implementation of Map-Reduce developed in Java. We have performed our experiments on a Dell Latitude E4300 laptop with two processors: Intel Centrino model 2.26GHz, 64-bit; 2 x 2GB memory; 80GB Disk.

For our experiments, we followed the same instructions as [4]. The velocity model is simply built by using paint software such as Microsoft Windows Paint. Refer to input variables in algorithm in Figure 3, a Bitmap file of size 1500*1500 (*nt=1500* and *nz=1500*) is generated with several lines and curves to simplify the creation of velocity model. Also, another empty bitmap file with the same dimensions (*nx=1500* and *ny=1500*) is produced as the initial data image. Moreover, the values of *velhalf*, *dt* and *dx* have been set to 0.1, 8.0 and 4.0, respectively. As mentioned in [4], these values give a reasonable shaped hyperbolas in running forward PKTM, otherwise aliasing happens.

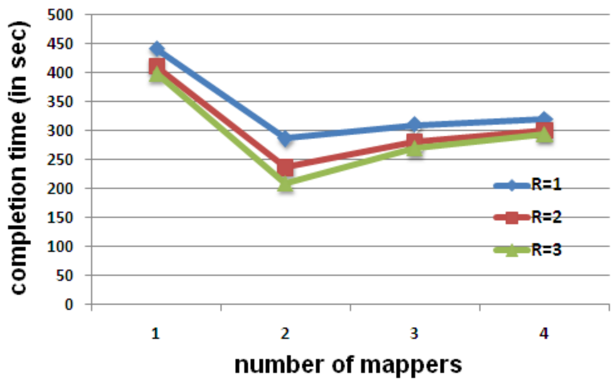
B. *Results*

The MapReduce implementation of PKTM has been developed and evaluated on a single machine with two processors. The performance of the proposed implementation of PKTM can be evaluated based on some MapReduce performance indicators such as resource utilization (CPU usage, network/Memory usage), completion time and energy efficiency. Among these performance indicators, the completion time is very important as seismic imaging algorithms process terabytes of data for several months to finish only just one experiment. Therefore the main issue in such algorithms is to make the completion time as short as possible.

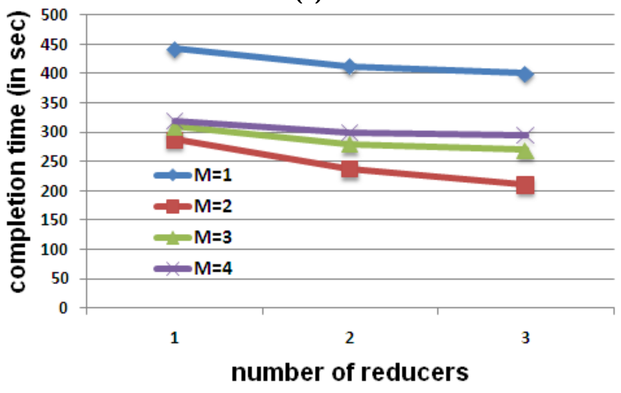
Generally, there is a strong dependency between a MapReduce application (in our experiment: completion time) and the MapReduce configuration parameters such as number of mappers and reducers. Figure 5 shows the dependency between these two configuration parameters and completion time. As can be seen from this figure, total completion time has correlation with the number of both mappers and reducers. When the number of mappers is equal to the number of processor units, the total execution time decreases. By increasing the number of mappers, the total execution time increases slightly. This means the lowest completion time happens when (1) each trace is assigned to one mapper and (2) each processor unit in the system processes only one mapper at each time. When a cluster with K processor units is used, we expect one mapper and reducer on each processor unit give the minimum completion time. In other words, in this K-node cluster the minimum completion time occurs when an experiment is executed with K mappers and K reducers. Figure 6 also shows two examples on the obtained forward PKTM images after applying earth velocity models. The earth velocity models have been used by both sequential and MapReduce implementation of PKTM in figure 3 and 4, respectively.

VI. CONCLUSION

Parallel execution of terabyte amount of data is one of the challenging tasks in both research and industry. In this paper, we described our experience in applying Google's MapReduce framework on forward/inverse Prestack



(a)



(b)

Figure 5. the relation between the completion time in seconds and (a) number of mappers and (b)number of reducers of running PKTM on MapReduce (R: number of reducers, M: number of mappers).

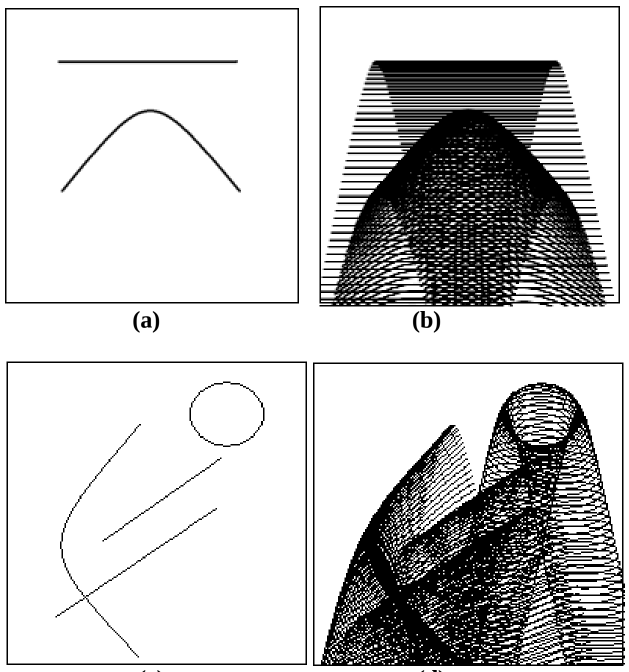


Figure 6. The earth velocity models to test forward PKTM algorithm in both sequential (figure 3) and MapReduce implementation (figure 4): (a,c) earth velocity models draw by MS-Paint software, (b,d) the forward PKTM

Kirchhoff Time Migration (PKTM) as one of the well-known and time-consuming algorithms in seismic imaging. We then gave a detailed performance discussion about the impacts of the number of mappers and reducers on the completion time of running a PKTM algorithm on MapReduce framework.

REFERENCES

[1] B.J.Guillot. (2003, *2D Kirchhoff Migation in Java*. Available: www.bgfax.com/school/kirchhoff_java.pdf

[2] D.Bevc, "Imaging Complex Structure with Smi-Recursive Kirchhoff Migration," *Geophysics*, vol. 62, pp. 577-588, 1997.

[3] N. B. Rizvandi, *et al.*, "Preliminary Results on Modeling CPU Utilization of MapReduce Programs," University of Sydney, TR665, 2010.

[4] H. Yan, *et al.*, "DECF: A Coarse-Grained Data-Parallel Programming Framework for Seismic Processing," presented at the International Conference on Computer Science and Software Engineering, 2008.

[5] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107-113, 2008.

[6] N. B. Rizvandi, *et al.*, "On using Pattern Matching Algorithms in MapReduce Applications," presented at the The 9th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA), Busan, South Korea, 2011.

[7] G. S. Sadasivam and G. Baktavatchalam, "A novel approach to multiple sequence alignment using hadoop data grids," in *the 2010 workshop on Massive Data Analysis on the Cloud*, Raleigh, North Carolina, USA, 2010, pp. 1-7.

[8] J. Ekanayake, *et al.*, "MapReduce for Data Intensive Scientific Analyses," presented at the 2008 4th IEEE International Conference on eScience, 2008.

[9] C.-T. Chu, *et al.*, "Map-Reduce for machine learning on multicore," presented at the 2006 Neural Information Processing Systems Conference (NIPS), Vancouver, Canada, 2006.

[10] *Hadoop Developer Training*. Available: http://www.cloudera.com/wp-content/uploads/2010/01/1-ThinkingAtScale.pdf

[11] S. Etgen, *et al.*, "Seismic Migration Problems and Solutions," *Geophysics*, vol. 66, pp. 1622-1640, 2001.

[12] *Seismic Imaging*. Available: http://www.cpeo.org/techtree/ttdescript/seisim.htm

[13] X. Shi, *et al.*, "A Practical Approach of Curved Ray Prestack Kirchhoff Time Migration on GPGPU," presented at the Proceedings of the 8th International Symposium on Advanced Parallel Processing Technologies, Rapperswil, Switzerland, 2009.

[14] D. Hengchan, "Parallel Processing of Presteak Kirchhoff Time Migration on a PC Cluster," *Computers & Geosciences*, vol. 31, pp. 891-899, 2005.