Parallel Kirchhoff Pre-Stack Depth Migration on Large High Performance Clusters

Chao Li[1(✉)], Yida Wang[1], Changhai Zhao[2], Haihua Yan[1], and Jianlei Zhang[2]

[1] School of Computer Science and Engineering, Beihang University, Beijing 100191, China
casesense@163.com
[2] Research and Development Center, BGP Inc., CNPC, Zhuozhou 072751, Hebei, China

Abstract. Kirchhoff Pre-Stack Depth Migration (KPSDM) is a widely used algorithm for seismic imaging in petroleum industry. To provide higher FLOPS, modern high performance clusters are equipped with more computing nodes and more cores for each node. The evolution style of clusters leads to two problems for upper layer applications such as KPSDM: (1) the increasing disparity of the I/O capacity and computing performance is becoming a bottleneck for higher scalability; (2) the decreasing Mean Time Between Failures (MTBF) limits the availability of the applications. In this paper, we present an optimized parallel implementation of KPSDM to adapt to modern clusters. First, we convert the KPSDM into a clear and simple task-based parallel application by decomposing the computation along two dimensions: the imaging space and seismic data. Then, those tasks are mapped to computing nodes that are organized using a two-level master/worker architecture to reduce the I/O workloads. And each task is further parallelized using multi-cores to fully utilize the computing resources. Finally, fault tolerance and checkpoint are implemented to meet the availability requirement in production environments. Experimental results with practical seismic data show that our parallel implementation of KPSDM can scale smoothly from 51 nodes (816 cores) to 211 nodes (3376 cores) with low I/O workloads on the I/O sub-system and multiple process failures can be tolerated efficiently.

Keywords: High performance computing · Parallel computing · Kirchhoff Pre-Stack depth migration · Seismic imaging · Parallel algorithms

1 Introduction

Seismic exploration is widely used to locate the oil and gas reservoir area and estimate potential reserves in petroleum industry because of its ability to provide the images of earth interior geologic structures. Seismic exploration can be divided into three stages: acquisition, processing and interpretation. Seismic data is collected from fields in the first stage and processed in the second stage using various seismic imaging algorithms such as Kirchhoff Pre-Stack Depth Migration (KPSDM) to translate the unknown

G. Wang et al. (Eds.): ICA3PP 2015, Part III, LNCS 9530, pp. 251–266, 2015.
DOI: 10.1007/978-3-319-27137-8_20

geologic structures into visual images that can be understood and analyzed by geophysicists and geologists in the interpretation stage.

KPSDM is one of the most important and popular depth-domain imaging algorithms [4, 8] for its computation efficiency and support for irregular seismic data as well as target migration. As the survey target is becoming increasingly complex in recent years, e.g., deep sea explorations, the seismic data size of some 3D survey areas can achieve hundreds of Terabytes. The explosive growth of seismic data size leads to the processing time becoming undesirably long.

Clusters with higher FLOPS are deployed to shorten the processing time. This demands the parallel algorithms and applications can scale well so that more nodes can be utilized. However, the increasing scale and performance of the clusters lead to two severe problems that may limit the scalability and availability of the applications. First, since the improvement of I/O capacity cannot compete with the improvement of computing performance [9, 10], the increasing gap between I/O and FLOPS is becoming a hinder for achieving higher scalability. For example, to balance the I/O capacity and computing performance, traditional computing platforms need to provide at least 2 byte/sec for each 1000 float operations [14]. However, the state of art clusters, especially those equipped with GPU accelerators, I/O/flops is usually 1 or 2 orders of magnitude lower than the value recommended for traditional computers. [1, 2] have reported that the I/O sub-system has become the bottleneck of Reverse Time Migration (RTM), a more compute-intensive seismic algorithm. Second, an observation in [15] shows that the Mean Time Between Failures (MTBF) of large clusters consisted by hundreds of computing nodes ranges from 6 to 13 h. Parallel applications deployed on large clusters should tolerate the node failures to ensure the availability.

The parallel implementation of KPSDM proposed in this paper can overcome these problems. First, we convert the KPSDM into a simple task-based parallel application by decomposing data along the dimensions of imaging space and seismic data according to the hardware resource capacity of one computing node. Then, the tasks are scheduled to the computing nodes which are organized by a two-level master/worker architecture. The seismic data is first broadcast by a I/O proxy node to each worker in one process group to reduce the I/O workloads on the parallel file system. Then the data will be buffered by the local disks of computing nodes so that a large amount of remaining tasks can obtain seismic data directly from local disks. Thus, the I/O workloads on the I/O subsystem of clusters can be significantly reduced. Finally, the task-based parallel pattern and two-level master/worker architecture guarantee the independence of computations and processes, which greatly facilitates the implementation of fault tolerance and checkpoint.

Experiments are carried out using practical data on a high performance production cluster. Our implementation of KPSDM can scale smoothly from 51 nodes (816 cores) to 211 nodes (3376 cores). I/O throughput monitoring tests during the run show that our implementation of KPSDM can keep the I/O workloads at a low level, which suggests that higher scalability can be achieved if more computing nodes are added. Failure injection tests show that multiple processes failures can be tolerated and the overhead incurred by those failures is neglectable.

The rest of the paper is organized as follows. Seismic imaging and the theory of KPSDM are introduced in Sect. 2. In Sect. 3, we describe the design and

implementation of parallel KPSDM in details. Section 4 presents the experimental results. We give comparisons to related work in Sect. 5 and conclude the paper in Sect. 6.

2 Background

2.1 Seismic Imaging

Figure 1 describes how the seismic data is collected. First, seismic waves are generated by an artificial seismic source at a shot point (also called source point). During the trip of propagating through the subsurface, those waves will be reflected and refracted by the interfaces between rock stratums. Then the waves will travel back to the surface and be recorded by the geophones deployed on the surface. The term trace refers to the seismic waves recorded by one geophone. A trace is composed of a series of float numbers that represent energy amplitude of the seismic wave. The float number is called sample. Each trace is associated with three key position coordinates: the shot point **S**, the receiver point **R** and the middle point of **S** and **R**. The term gather refers to a collection of traces that have a common attribute. For example, a collection of traces that share a same shot point is called common shot gather (CSG). The term offset refers to the distance between the shot point and receiver point.

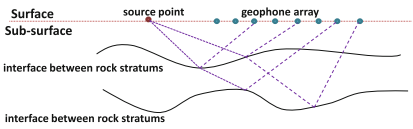

Fig. 1. Seismic data acquisition

2.2 A Brief Introduction of KPSDM

In the processing stage, the survey area of surface is divided by two-dimensional grid as shown in Fig. 2. The dimensions are called inline and crossline respectively. We define the intersection points of inline and crossline as imaging cells. And the points of an imaging cell along the depth direction are defined as imaging points. From the view of surface, the imaging space is a two-dimensional grid consisted by imaging cells. And from the view of subsurface, the imaging space is a three-dimensional grid

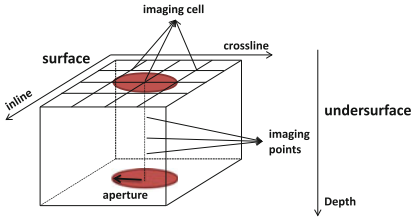

Fig. 2. Seismic data processing

consisted by imaging points of all imaging cells. The KPSDM algorithm aims to calculate the energy value of each imaging point based on the seismic data and travel timetable information of seismic waves.

Each imaging cell can delimit an valid range in a shape of ellipse based on aperture parameters, as shown in Fig. 2. Any trace whose shot point and receiver point located in the ellipse should perform the migration computation to the imaging cell, i.e. to each imaging point of the imaging cell. Let ξ denote the imaging point, $I(\xi)$ denote the imaging result of ξ, T denote the input trace, D denote the valid range. First, we need to get time cost by the seismic wave from the shot point to the imaging point t_s and the time cost from the receiver point to the imaging point t_r. Then, energy amplitude A of imaging point ξ can be calculated with t_r, t_s and discrete energy amplitudes of T. Finally, we add the amplitude A to $I(\xi)$, so the energy contribution of T to imaging point ξ is stacked. The final imaging result of ξ is the summation of contributed amplitudes of all valid traces located in the valid range as shown by Eq. 1.

$$I(\xi) = \sum_{T \in D} f(t_s, t_r, T). \tag{1}$$

2.3 I/O Workload of KPSDM

A KPSDM job can impose huge I/O workloads on the I/O sub-system for several reasons. First, comparing with wave equation-based algorithms such as Reverse Time Migration (RTM), summation operations of integral involve less computations. Second, during the running of KPSDM, travel timetables need to be read from storage system iteratively. For a medium 3D migration job, the data size of travel timetables can achieve dozens or hundreds of Terabytes. While the RTM method has no need of travel timetable data. Third, as the size of travel timetable is large, only a few imaging cells can reside in the memory, KPSDM needs to acquire more seismic traces from I/O sub-system to fully utilize the computation resources.

3 Design and Implementation

For a KPSDM job, before the migration computation starts, there are extra two steps: building travel timetables and pre-processing seismic data. The decomposition of the imaging space and seismic data are done in first two steps. More details on the decomposition are introduced in Sect. 3.1.

3.1 Decomposition

3.1.1 Decomposition of Imaging Space

During the migration computation stage, travel timetable is used for the lookup of t_s and t_r, so the travel timetable should always resides in memory. However, for a medium 3D job, the total size of all travel timetables can achieve dozens of Terabytes.

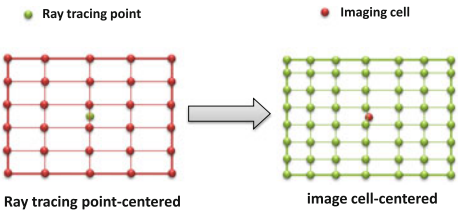

Fig. 3. Ray tracing point-centered travel timetable and imaging cell-centered travel timetable

The decomposition of imaging space is to divide the travel timetable in the principle that the travel timetable of one imaging space can be loaded into memory.

The travel timetable of each ray tracing point (also called shot point) is calculated based on ray tracing theory. Ray tracing points are arranged by a two-dimensional grid on the surface. And ray tracing grid and the imaging cell grid are two independent grids.

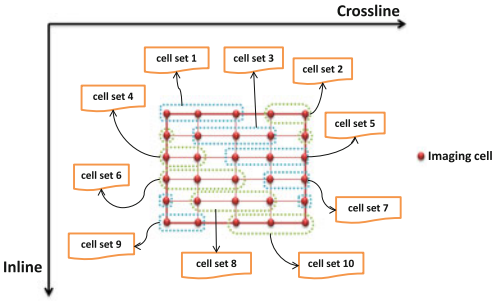

Fig. 4. Decomposition of imaging cells

We propose a data structure called imaging cell-centered travel timetable to organize the travel timetable data. For each ray tracing point, from view of the surface, the travel timetable calculated by ray tracing is a two-dimensional array of which elements are the time cost from the position of ray tracing point to the positions of different imaging cells. While the aim of imaging cell-centered travel timetable is to construct a two-dimensional array of which elements are the time cost from the position of the imaging cell to position of different ray tracing points. Figure 3 contrast these two different organization structures. The benefit of the imaging cell-centered travel timetable structure is that any valid trace for the imaging cell can obtain corresponding t_s and t_r in the time complexity of O(1).

The decomposition of travel timetables is based on the data structure of imaging cell-centered travel timetable. According to the size of memory and the maximum travel timetable size of one imaging cell, the maximum number of imaging cells L can be calculated. Then, total imaging cells are divided into imaging cell sets based on L. Figure 4 illustrates a decomposition example, in which there are 30 imaging cells and the L is 3. The imaging cells are divided into 10 imaging cell sets.

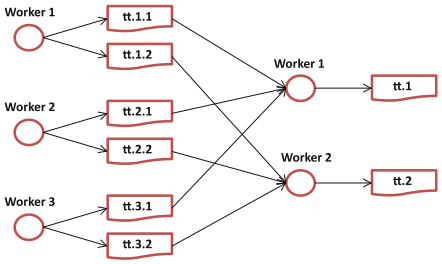

Fig. 5. MapReduce-like travel timetable computation architecture

To accelerate the loading speed of travel timetable, all travel timetables of one imaging cell set are stored in only one file, i.e., each imaging cell set corresponds to a travel timetable file. Note that the travel timetable of each ray tracing point may contain the data that belongs to different imaging cell sets. To reorganize the travel timetable data into files based on imaging cell sets, a MapReduce-like computation pattern is used in the step of travel timetable computation as shown in Fig. 5. In the Map stage, each process creates a file for each imaging cell set. The travel timetable of one ray tracing point will be split into multiple parts by imaging cell set and then these parts will be output to corresponding files. An index will record the relation information of the ray tracing point and the imaging cell set. In addition, we compressed the travel timetable data using zlib and compression ratio can usually achieve $6 \sim 8$. In the Reduce stage, travel timetable files that belong to same imaging set will be merged into one single file. In the example illustrated by Fig. 5, there are 2 imaging sets and each process outputs 2 travel timetable files in the Map stage. And files that share a common imaging set are merged into a single file.

3.1.2 Decomposition of Seismic Data

In our implementation, each migration job can have multiple seismic data files. The input files are first split into sub-files and then reorganized by common offset range. Let F denote the set of input files, F_i denote i-th seismic file. In one migration job, the offset parameter can be set as $[O_1, O_2, O_3]$, in which O_1 is the minimum offset, O_2 is the maximum offset and O_3 is the offset interval. Let O denote the number of offset range, O_i denote i-th offset range. O can be calculated by Eq. 2.

$$O = \frac{(O_2 - O_1)}{O_3} \tag{2}$$

Each seismic file F_i is spilt into O sub-files. Let matrix D denote the sub-files of all input files. D_{ij} is the j-th sub-file of i-th input seismic file. In matrix D, the first dimension is input file and the second dimension is offset range. Sub-files of matrix D that share common offset range are divided into multiple segments according to the available space size of local disks. For example, there are 6 input files and 6 offset range in Fig. 6. All sub-files of offset range 0 and the first two sub-files of offset range 1 are grouped into a seismic data segment. With this decomposition approach, the

seismic data that has same offset range is aggregated. Note that the imaging results are organized by offset range. After all the seismic data of one offset range finishes the migration computations to an imaging cell set, the imaging results of the offset range and the imaging cell set can be output into database.

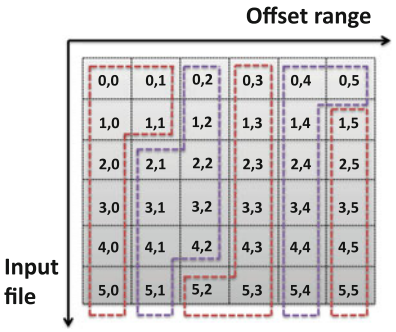

Fig. 6. Decomposition of seismic data

3.1.3 Task-Based Parallel KPSDM

In conclusion, imaging space is divided by the size of memory and seismic data is divided by the available size of local disks. Let N denote the number of imaging cell sets, M denote the number of seismic data segments. The total tasks of one migration job is M*N. The sequential version of KPSDM can be summarized as Algorithm 1. For each seismic data segment, there are N imaging cell sets to be migrated. As to one task, the travel timetable file is always first loaded into memory for the lookup of t_s and t_r during the migration. Then the seismic data is read as gathers for the migration computations. Now, we have convert the KPSDM algorithm into a task-based parallel application.

Algorithm 1. Single node KPSDM

```
1:  for i = 1 to M
2:      for j = 1 to N
3:          load the travel timetable file of j-th imaging cell set
4:          foreach gather in i-th seismic data segment
5:              perform migration computation with gather and travel timetables
```

3.2 Cluster-Level Parallelism

All processes are organized by a two-level master/worker architecture as shown in Fig. 7. Process 0 acts as the master and remaining processes are divided into groups with 4 processes in one group. All the processes in one group are workers and the worker with least rank also acts as submaster.

Figure 8 presents the flowchart of parallel KPSDM. Once the migration computation starts, master builds M task pools. Each pool corresponds to a seismic data

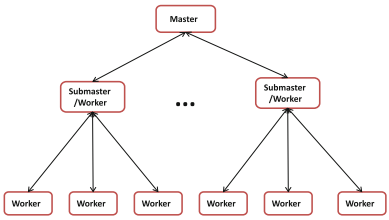

Fig. 7. Two-level master/worker architecture

segment and has N tasks. Then task pools are distributed among different process groups. Specifically, submaster asks master for a task pool and the group are responsible for finishing all the tasks of this pool. The tasks are done in two stages: broadcast stage and autonomy stage. In the broadcast stage, submaster reads gathers of the seismic data segment from the parallel file system and broadcasts the gathers to workers in this group. After the worker receives a gather, it first writes the gather into local disk and then migrates this gather to current imaging cell set. In the autonomy stage, all workers will finish the remaining N-4 tasks and read seismic gathers directly from local disks.

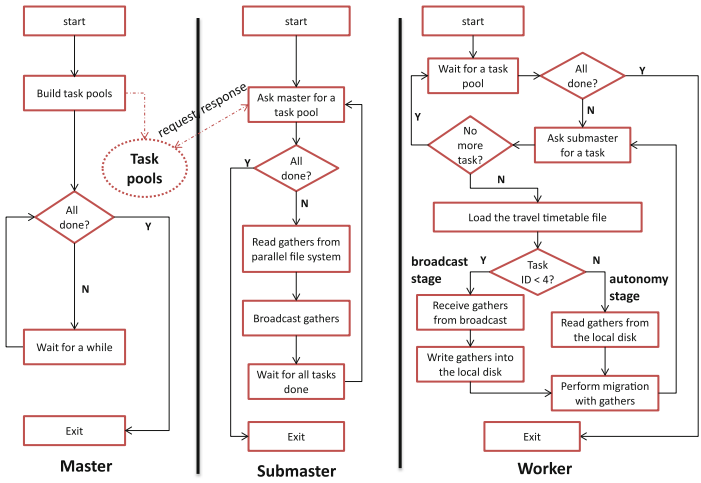

Fig. 8. Flowchart of the parallel KPSDM

The architecture and design can greatly reduce the I/O workloads on the I/O sub-system for several reasons listed as follows.

- Since each group has a submaster acting as the I/O proxy, it is not necessary for all workers to read seismic data from the parallel file system.
- All the seismic data for tasks in autonomy stage can be obtained from local disks, so one seismic data segment needs to be read from the parallel file system for only once and N tasks of the segment can be done.

- By building index information for buffered seismic data on local disks, each task can choose valid seismic data for current imaging cell set in the autonomy stage. Unnecessary I/O workloads caused by invalid seismic data are filtered.
- Local disk buffering mechanism allows part of the buffered seismic data to be cached by the memory. As the seismic data will be read iteratively for different imaging cell sets, memory caching of the local file system can reduce the number of actual disk operations.

Note that in the broadcast stage, performance degrades because of the synchronization side effects and invalid input seismic data. In the autonomy stage, these problems can be avoided. For a medium or large 3D migration job, there are thousands or hundreds of thousands of imaging cell sets. Thus, the overhead caused in the broadcast stage when performing the first four tasks can be ignored. The saved bandwidth of the parallel file systems are conserved for reading travel timetable files.

In addition, a simple work-stealing mechanism are employed to achieve better load balance. When there are no more task pool for a submaster, it first checks current unfinished task pools of other process groups. Submaster will join the group whose task pool has maximum number of unfinished tasks.

3.3 Node-Level Parallelism

Figure 9 illustrates the parallel migration architecture of a single node. After a worker gets a new imaging set from master, the corresponding compressed travel timetable file is first loaded from parallel file system into memory. Then the data structures of imaging cell-centered travel timetable are built based on the index and compressed data of the file. When the building process finishes, each imaging cell corresponds to a three-dimensional grid which enables any valid trace to obtain t_s and t_r in constant time. Then gathers from broadcast or local disks are loaded into a gather queue continuously. Each gather contains 1000 traces in default and all the traces in the gather share a common offset range due to the decomposition design of seismic data. Traces of the gather popped from the gather queue are distributed among multi-threads.

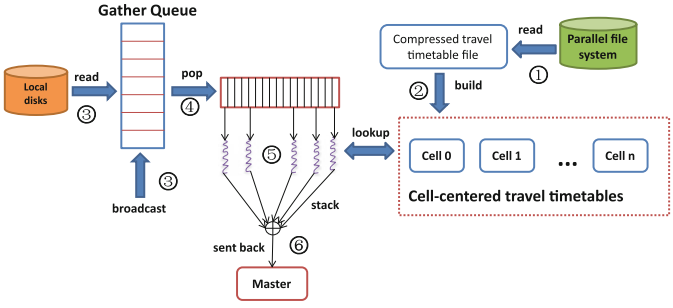

Fig. 9. Parallel architecture of one computing node

Algorithm 2 describes the migration process of one thread. Note that each trace fetched from the gather needs to perform the migration computation for all imaging

cells of current cell set. For one imaging cell C, t_s vector and t_r vector are first obtained from the cell-centered travel timetable of C. Then, imaging result of < T, C > is calculated with t_s vector and t_r vector.

Algorithm 2. Migration process of one thread

1: fetch one trace **T** from the gather **G**
2: foreach imaging cell **C** in current cell set:
3: lookup for $\boldsymbol{t_s}$ vector from cell-centered travel timetable of **C**
4: lookup for $\boldsymbol{t_r}$ vector from cell-centered travel timetable of **C**
5: computing imaging result of <**T**, **C**> with $\boldsymbol{t_s}$ vector and $\boldsymbol{t_r}$ vector

To avoid the writing contention, each thread is equipped with an imaging buffer for storing the imaging results. After all the gathers of one offset range are migrated, the results of all threads are stacked. The final results are then sent back to master. If the whole data of one offset range is contained in only one seismic data segment, master can directly output the imaging results into database. Otherwise, master needs to further stack the imaging results from different groups and then writes the results into database.

3.4 High Availability

We implement fault tolerance at the application layer to guarantee that the migration job can still progress to finish even multiple process failures happen. Specifically, master monitors the health states of all processes. When a process failure happens, master first marks the node and then broadcasts the information of the failed node to every healthy process. And the failed task of will be re-scheduled to a healthy worker. In our implementation, the failed tasks will be re-processed after all normal tasks finish. Processes located in the same group with the failed node will start fault tolerance processing. For example, if submaster is down, a new submaster will be elected. If a worker is down, the collective communication operations will exclude it. In this way, the job can still be done as long as the number of healthy processes is not less than two.

Since the MTTF of one single node is usually longer than the time cost by one migration job, it is unlikely for master to fail. Nevertheless, we implement a checkpoint mechanism at the application layer of KPSDM which records the finished tasks cyclically. In case that master is down, by analyzing the checkpoint, the job can recover from the broken point and unnecessary computations can be avoided.

4 Evaluation and Analysis

All experiments are performed on a cluster called IBM2 which contains 220 computing nodes. Each node is equipped with two 2.6 GHz, 8 cores, 64-bit Intel Xeon E5-2670 processors and 128 GB DRAM. Thus, each node has 16 physical cores. The operating

system of each computing node is Red Hat Enterprise Linux Server release 6.4 (Santiago). The parallel file system deployed on IBM2 is GPFS [3]. The total storage space of the parallel file system is 683 TB.

4.1 Cluster-Level Migration Scalability

Practical seismic data set named Jinyao collected in a field located in western China is used for tests. The field covers an area of 2760 square kilometers and the depth of the survey area is 8.75 km. The number of samples in each trace is 1750. We choose part of the data for the experiments, in which inline ranges from 1686 to 1986 and crossline ranges from 1231 to 2548. And the total size of the test data is 427.58 GB.

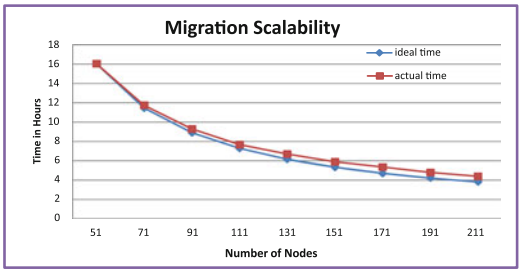

Fig. 10. Cluster-level migration scalability

Figure 10 shows the scalability of our parallel implementation of KPSDM by increasing the number of computing node from 51 to 211. The increment of nodes number between two adjacent tests is 20. The ideal time is calculated proportionally based on the time cost using 51 nodes and the number of nodes in other tests. From Fig. 10, we can conclude that our implementation can achieve high scalability since the actual time cost is nearly close with the ideal time when more computing nodes are added to accelerate the migration computations.

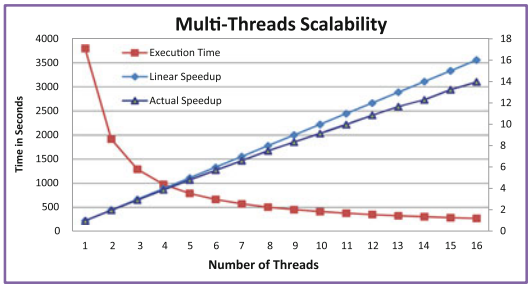

Fig. 11. Node-level migration scalability

4.2 Node-Level Migration Scalability

A small migration job is used for the multi-threads scalability test. The experimental results in Fig. 11 illustrate that our implementation of KPSDM can maintain a near-linear speedup when the number of threads ranges from 1 to 16. Thus, a good scalability for a multi-threads computing node is also achieved.

4.3 Seismic Data Throughput Monitoring

A migration job with Jinyao dataset is used for the tests in this experiment. Figure 12 shows the seismic data throughput of 151 and 211 computing nodes, respectively. In both cases, the throughput only refers to the seismic data that is read from the parallel file system. The average bandwidth during the broadcast stage after the migration starts can achieve about 2.2 GB/s for 211 nodes and 1.5 GB/s for 151 nodes. And the maximum bandwidth of the GPFS of this cluster is about 5 GB/s. Thus, the I/O workloads of reading seismic data is relatively low. When the work stealing mechanism begins to work for better load balance, submaster still needs to read data from the parallel file system as shown in tail of Fig. 12.

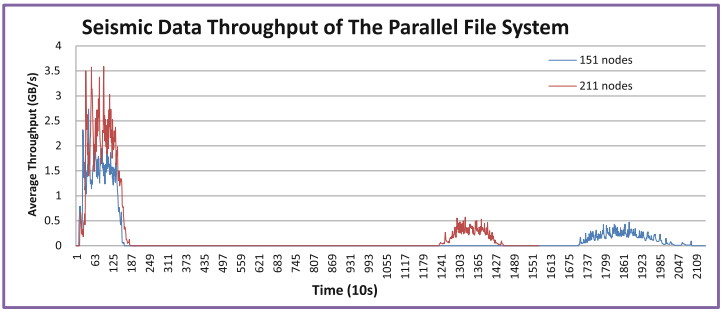

Fig. 12. Seismic data throughput of parallel file system

Figure 13 contrasts the seismic data processing throughput and actual seismic data throughput from local disks. In this test, 211 computing nodes are used for the migration job. In most cases, the actual throughput of local disks is 20 ~ 40 times lower than the seismic data processing throughput. It indicates that the I/O workloads on each local disk is very low. At the beginning, since the seismic data is still not cached by memory, the maximum bandwidth of each local disk can achieve about 15 MB/s. For subsequent tasks, because part of the seismic data is cached by memory of the computing node, the actual bandwidth of each local disk used for reading seismic data is very low.

4.4 Fault Tolerance

A medium migration job using Jinyao dataset is used for this experiment with 33 computing nodes, i.e., 8 process groups. We compare the seismic data processing

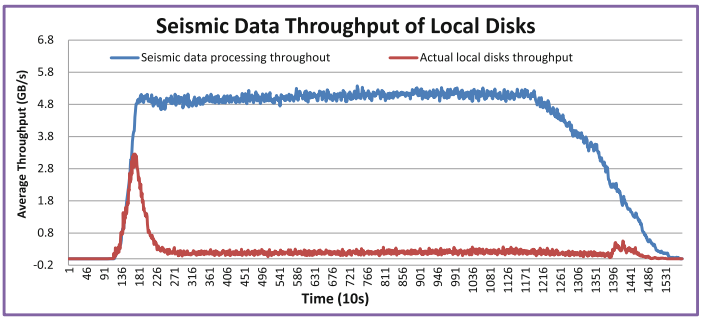

Fig. 13. Seismic data throughput of local disks

throughput of all workers in four tests. In the first test, no process is killed. In the second test, four submaster from four groups are killed. In the third test, four submasters from four groups and four workers from other four groups are killed. In the last test, two workers for each group are killed, i.e. 16 workers are killed. In all the tests, the faults are injected after the KPSDM has been running for about 1500 s. Figure 14 presents the throughput changes when faults happen. First, Fig. 14 shows that multiple process failures can be tolerated. The migration job can still be done even half of the workers fail. Second, Fig. 14 illustrates that the faults almost have no side effects on the healthy processes since the throughput drops proportionally. Thus, our implementation of KPSDM can recover from failures with very low overhead.

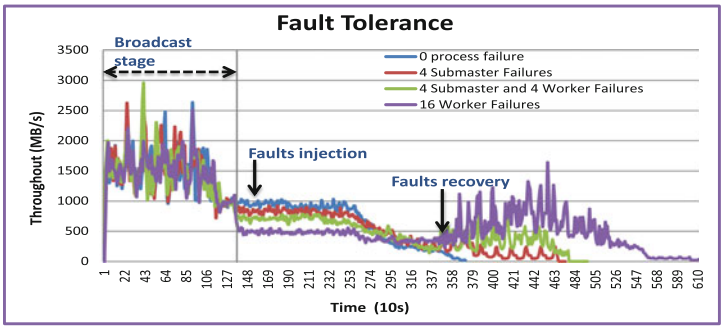

Fig. 14. I/O throughput of KPSDM when multiple process failures happen

5 Related Work

As noted in [11], due to trade secrets, there are only a few literatures on the parallel implementation of KPSDM. One method is to divide the migration computation by inline just as the method proposed in [11] for Kirchhoff Pre-Stack Time Migration (KPSTM) based on the assumption that the memory of all computing nodes can usually hold the travel timetables of all imaging cells of one inline. However, the method cannot scale in problem domain. When the total size of travel timetables of one inline is

larger than the whole memory size of all nodes, the implementation demands adding more nodes to start the migration. In case that no more nodes are available, the density of the travel timetable grid has to be reduced to balance the requirements on memory, which results in imaging precision loss. A decomposition method that divides the imaging cell grid into sizable rectangles is proposed in [6]. We choose a more simple method, i.e., the imaging cell grid is decomposed by treating each imaging cell as an independent unit based on the data structure of cell-centered travel timetable.

In the migration stage, a master/worker architecture is usually used to organize all processes. After all the workers have loaded corresponding travel timetable of an imaging cell set, master begins to read seismic data from the parallel file system and broadcast the data to workers. Although, this design can greatly reduce I/O workloads on the I/O sub-system, it has several shortcomings as follows.

- Poor scalability. For small migration jobs, with the increasing number of computing nodes, the number of imaging cells distributed to each node is decreasing. Master needs to achieve higher read and broadcast bandwidth, which makes the master become the I/O bottleneck. The execution time cannot be shortened even if more nodes are added when master achieves the bandwidth bottleneck.
- Poor performance. Broadcast mechanism makes all workers process the same input seismic data. Note that each computing node progresses at a different pace since the seismic data could be invalid for some nodes when the received data is not in the valid range delimited by current imaging cells. Because of the synchronization side effect caused by broadcast, all other computing nodes need to wait for the slowest node. Thus, a great amount of CPU cycles is wasted.

To avoid these shortages, process groups are proposed in [7]. Nevertheless, the synchronization side effect within one process group still leads to considerable CPU cycles wasted. Another simple way for acquiring the seismic data is that each node reads the seismic data from the parallel file system directly [5, 6, 12]. The main benefit of this method is that all workers are independent from each other and can choose valid seismic data. However, this imposes a huge I/O workloads on the I/O sub-system. In addition, [12] adopts a static seismic data partition method implying that good load balance is hard to achieve.

6 Conclusion

In this paper, we present a highly optimized parallel implementation of KPSDM which achieves four goals:

- **Simple and clear.** We present a simple and clear task-based parallel implementation of KPSDM, which also facilitates the goal of scalability and availability.
- **Low I/O workloads.** The combination of I/O proxy in a process group, local disk buffering, invalid seismic data filtering and memory caching can dramatically reduce the I/O workloads on I/O sub-system of clusters.
- **High scalability.** The design of task-based parallel pattern as well as the balance between I/O capacity and computing performance enables our implementation to

scale out on large high performance clusters. In addition, our implementation of KPSDM also has a near-linear scalability for a multi-cores computing node.

- **High availability.** When the scale of the clusters continues to expand, it becomes failure prone for upper layer applications. The availability of KPSDM is guaranteed by fault tolerance and checkpoint mechanism.

References

1. Perrone, M. P., Zhou, H., Fossum, G., Todd, R.: Practical VTI RTM. In: 72nd EAGE Conference and Exhibition, June 2010
2. Perrone, M., Liu, L. K., Lu, L., Magerlein, K., Kim, C., Fedulova, I., Semenikhin, A.: Reducing data movement costs: scalable seismic imaging on blue gene. In: 2012 IEEE 26th International Parallel & Distributed Processing Symposium (IPDPS), pp. 320–329. IEEE May 2012
3. Schmuck, F.B., Haskin, R.L.: GPFS: A shared-disk file system for large computing clusters. In: Proceedings of FAST 2002, vol. 2, p. 19 January 2002
4. Yilmaz, Ö., Doherty, S.M.: Seismic Data Processing, vol. 2. Society of Exploration Geophysicists, Tulsa (1987)
5. Chang, H., VanDyke, J.P., Solano, M., McMechan, G.A., Epili, D.: 3D pre-stack Kirchhoff depth migration: from prototype to production in a massively parallel processor environment. Geophys. **63**(2), 546–556 (1998)
6. Panetta, J., de Souza Filho, P.R., da Cunha Filho, C.A., da Motta, F.M.R., Pinheiro, S.S., Pedrosa, I., de Albrecht, C.H.: Computational characteristics of production seismic migration and its performance on novel processor architectures. In: 19th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2007. pp. 11–18. IEEE October 2007
7. Li, J., Hei, D., Yan, L.: Partitioning algorithm of 3D pre-stack parallel Kirchhoff depth migration for imaging spaces. In: Eighth International Conference on Grid and Cooperative Computing, GCC 2009. pp. 276–280. IEEE August 2009
8. Bevc, D.: Imaging complex structures with semi-recursive Kirchhoff migration. Geophys. **62**(2), 577–588 (1997)
9. Yin, Y., Byna, S., Song, H., Sun, X.H., Thakur, R.: Boosting application-specific parallel i/o optimization using IOSIG. In: 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 196–203. IEEE May 2012
10. Yu, H., Sahoo, R.K., Howson, C., Almasi, G., Castanos, J.G., Gupta, M., Gropp, W.D.: High performance file i/o for the blue Gene/L supercomputer. In: The Twelfth International Symposium on High-Performance Computer Architecture, pp. 187–196. IEEE February 2006
11. Dai, H.: Parallel processing of pre-stack Kirchhoff time migration on a PC cluster. Comput. Geosci. **31**(7), 891–899 (2005)
12. Sen, V., Sen, M.K., Stoffa, P.L.: PVM based 3-D Kirchhoff depth migration using dynamically computed travel-times: an application in seismic data processing. Parallel Comput. **25**(3), 231–248 (1999)
13. Marr, D.T., Binns, F., Hill, D.L., et al.: Hyper-threading technology architecture and microarchitecture. Intel. Technol. J. **6**(1), 4–15 (2002)

14. Nowak, D.A., Seagar, M.: ASCI tera-scale simulation: requirements and deployments. http://www.ornl.gov/sci/optical/docs/Tutorial19991108Nowak.pdf
15. Heien, E., Kondo, D., Gainaru, A., LaPine, D., Kramer, B., Cappello, F.: Modeling and tolerating heterogeneous failures in large parallel systems. In: 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pp. 1–11. IEEE November 2011