A ROBUST PARALLEL FRAMEWORK FOR MASSIVE SPATIAL DATA PROCESSING ON HIGH PERFORMANCE CLUSTERS

Xuefeng Guan [a, *]

[a] State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, 129 Luoyu Road, Wuhan 430079, P. R. China –guanxuefeng@whu.edu.cn

Commission IV, WG IV/5

KEY WORDS: Data parallel processing, Split-and-Merge paradigm, Parallel framework, LiDAR

ABSTRACT:

Massive spatial data requires considerable computing power for real-time processing. With the help of the development of multicore technology and computer component cost reduction in recent years, high performance clusters become the only economically viable solution for this requirement. Massive spatial data processing demands heavy I/O operations however, and should be characterized as a data-intensive application. Data-intensive application parallelization strategies are imcompatible with currently available procssing frameworks, which are basically designed for traditional compute-intensive applications. In this paper we introduce a Split-and-Merge paradigm for spatial data processing and also propose a robust parallel framework in a cluster environment to support this paradigm. The Split-and-Merge paradigm efficiently exploits data parallelism for massive data processing. The proposed framework is based on the open-source TORQUE project and hosted on a multicore-enabled Linux cluster. One common LiDAR point cloud algorithm, Delaunay triangulation, was implemented on the proposed framework to evaluate its efficiency and scalability. Experimental results demonstrate that the system provides efficient performance speedup.

1. INTRODUCTION

1.1 Introduction

Spatial datasets in many fields, such as laser scanning, continue to increase with the improvements of data acquisition technologies. The size of LiDAR point clouds has increased from gigabytes to terabytes, even to petabytes, requiring a significant number of computing resources to process them in a short time. This is definitely beyond the capability for a single desktop personal computer (PC).

A practical solution to meet this resource requirement is to design parallel algorithms and run them on a distributed platform. Parallelism can be exploited by decomposing the domain into smaller subsets that can be executed concurrently. Multicore-enabled Central Processing Units (CPU) are becoming ubiquitous from the single desktop PC to clusters (Borkar and Chien, 2011); while the costs to build a powerful computing cluster are getting lower and lower. It is natural and necessary that spatial analysts employ high performance clusters (HPC) to efficiently process massive LiDAR point clouds.

Nowadays data processing algorithms were designed without any consideration in concurrency. For applied scientists, adapting these serial programs into a distributed platform is challenging and error-prone. They usually do not have much knowledge and experience in parallelization for the distributed context. Furthermore, processing massive LiDAR point cloud is inherently different from classical compute-intensive applications. Such applications devote most of their processing time to Input/Ouput (I/O) and manipulation of input data. This type of application should be characterized as a data-intensive application, as opposed to traditional compute-intensive application. Thus, the manipulation of input data must be taken into consideration during decomposition, scheduling, and load-balance.

Such a framework could be helpful and desirable, in which low-level thread/process operation routines are hided and high-level functions/classes are supplied in an application programming interface (API) library. This paper proposes a general parallel framework on a HPC platform to facilitate this transition from a single-core PC to a HPC context. This framework defines a Split-and-Merge programming paradigm for users/programmers. With the help of this paradigm, our framework can automatically parallelize and schedule user's tasks. Finally, we evaluate this robust framework with one typical massive LiDAR point cloud processing example, Delaunay triangulation.

Section 2 presents related work on the research on parallel data processing framework. Section 3 introduces the Split-and-Merge paradigm for the parallel framework. Section 4 respectively describes the detailed implementation of our parallel framework. Section 5 presents the results and discussion of the experiments. Section 6 closes the paper with our conclusions.

2. RELATED WORK

Parallel data processing has been an active research field for many years. Presently, a large body of work on parallel frameworks for data-intensive applications can be found in the literature.

Hawick et al. (2003) have used the grid computing techniques to build an operational infrastructure for data processing and data mining. Dean and Ghemawat (2008) proposed a programming

* Corresponding author. Email: guanxuefeng@whu.edu.cn, Tel: +86 27 68778311, Fax: +86 27 68778969

model, called MapReduce, for processing and generating large datasets. Users' common processing algorithms are abstracted into two functions, Map and Reduce. Programs written in this model are automatically parallelized and executed by the runtime framework on a large cluster of commodity machines.

Tehraniana et al. (2006) presented an architectural model and a system prototype for massive satellite data processing. Their prototype system shows that considerable performance improvement can be gained for the specific science algorithms in real-time.

Jonker described a data and task parallel framework to smoothly program parallel applications for image processing (Jonker and Nicolescu, 2008). In their proposed framework they also introduced a special paradigm for parallel processing, called the distributed bucket processing (DBP) paradigm, which is able to dynamically distribute data sets over processors.

Guan developed a general-purpose parallel raster processing programming library (pRPL) and applied it to speed up a commonly used cellular automaton model (Guan and Clarke, 2010).

Wang et al. (2011) presented a common parallel computing framework based on the message passing interface (MPI) protocol for modeling hydrological river-basin processes. This framework is computationally efficient, and independent of the type of physical models chosen.

All the frameworks target distributed environments and aim to improve the resources usage efficiency. Google's MapReduce, distributed bucket processing, and pRRL provide users with a higher-level abstract model and greatly reduce the implementation burden for programmers. However, Google's MapReduce model is too simple and strict for spatial algorithms. For example, if some spatial algorithms are expressed in the MapReduce model, the Reduce steps must be executed hierarchically, thus conflicting with the original linear and out-of-order execution pattern. As for distributed bucket processing and pRRL, these two frameworks are specifically designed for raster image processing, and does not support vector data processing. Thus, a general parallel framwork should own the efficiency of distributed bucket processing and pRRL, and also support both images and vector data.

3. THE SPLIT-AND-MERGE PARADIGM

3.1 Data locality

Jonker and Nicolescu (2008) distinguished image processing into three levels: image oriented operations (point or local neighborhood), symbolic processing, and knowledge-based processing. In P. P. Jonker's classification, the kernel of point operations focused on a single pixel or feature; while for local neighborhood operations the neighboring elements also participate in current element processing.

A common characteristic between point operations and local neighborhood operations is data locality. Data locality means that the kernel of these algorithms requires no other elements or only the adjacent elements when processing one element. Data locality provides fine-grain data parallelism. It can be illustrated by k-nearest neighbor search in Fig.1.

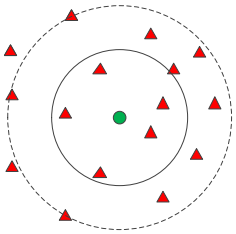

Figure 1. Data locality illustrated with k-nearest neighbor search

This characteristic provides a basis for LiDAR point cloud processing in a Split-and-Merge paradigm. In this paradigm, the entire LiDAR point cloud is first decomposed into many discrete blocks; then these blocks are individually processed by user ' s program *S*; and finally the intermediate results are merged into the actual output by user ' s program *M*.

3.2 The Split-and-Merge Paradigm

In the split-and-merge paradigm, the whole process is abstracted into two types of tasks: Split task *S* and Merge task *M*. Split tasks are mainly controlled by two factors: data decomposition and neighbour definition. All the Merge tasks are connected into a tree.

Regular decomposition usually divides the spatial domain into rows, columns, or blocks, illustrated in Fig.2. Currently the proposed Split-and-Merge paradigm only supports regular domain decomposition. The size of rows, columns, or blocks represents the granularity of parallel tasks. The grid index method is used to index decomposed blocks, which are named in this style, dataname_colnum_rownum.

Figure 2. Three regular domain decomposition methods (rows, columns, blocks)

For different algorithms when processing one element, the requirement for neighboring elements varies. The Split-and-Merge paradigm provides an option for users to specify the neighborhood configuration for each algorithm. The first attribute of neighbor definition is the shape. Two shapes are supported in this paradigm, cross or square. The second attribute is the size of the neighbor, defined by the block number in one direction. The neighbor definition is illustrated in Fig.3.

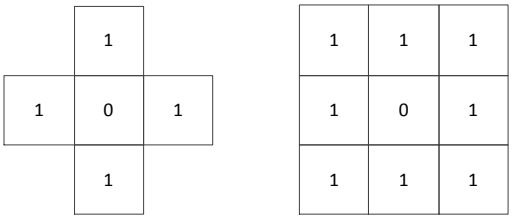

Figure 3. Two types of neighbor definition (left: cross; right: square)

After decomposing a LiDAR point cloud and concurrently processing each block, all the intermediate results are merged

into the final output. Due to the intrinsic diversity of processing algorithms, merge procedures vary. For some algorithms, additional edge operations must be carried out before the merge between two adjacent results; while for other ones, all the intermediate results can be merged in a batch mode without any additional edge operations.

Therefore, the complete execution graph can be divided into two categories according to the decomposition and merge paradigms: two-level n-ary tree, and n-level binary tree patterns, illustrated by Fig.4. In the first type, all the intermediate results are merged in a whole; in the second type two adjacent intermediate results are merged hierarchically.

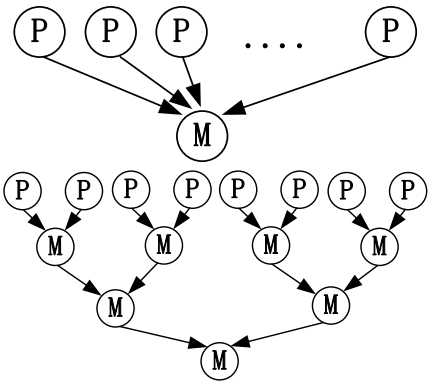

Figure 4. Two types of Split-and-Merge paradigms
(left: two-level n-ary tree; right: n-level binary tree)

For a specific LiDAR algorithm, the execution pattern is defined by its internal procedure. Users/programmers only focus on the actual implementation of two tasks: Split (program S) and Merge (program M). After the implementation of these two programs, the framework will automatically generate a collection of scripts to encapsulate these individual split and merge tasks. Illustrated by interpolating four point blocks into a DEM, the generated task scripts are listed in Fig.5 and Table 1.

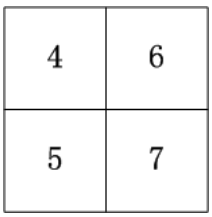

Figure 5. Four point blocks for interpolation

```
idw_interpolate -i abc_0_0.las -o abc_04.dem
idw_interpolate -i abc_1_0.las -o abc_05.dem
idw_interpolate -i abc_0_1.las -o abc_06.dem
idw_interpolate -i abc_0_2.las -o abc_07.dem
dem_merge -i abc_04.dem/ abc_05. dem / abc_06. dem /
abc_07. dem  -o abc_01.dem
```

Table1. A list of automatically generated task scripts

4. THE PARALLEL FRAMEWORK

Our universal parallel framework is built on a standard SMP (Symmetric Multiprocessor) cluster. Each node is connected by the local area network. The open source TORQUE project (Staples, 2006) was customized to provide a basis for our parallel framework.

Fig. 6 illustrates the structure of our parallel framework.The system consists of four components: pbs_server, pbs_sched, database, and pbs_mom. These four components collaborate closely to perform LiDAR point cloud processing.

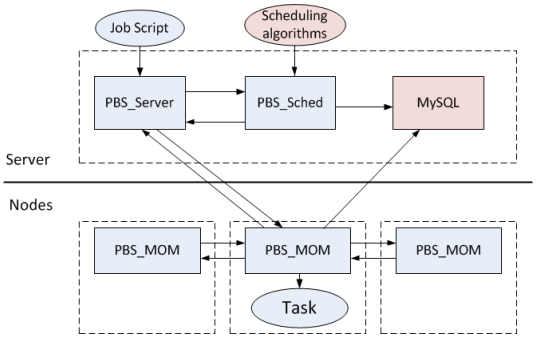

Figure 6. The TORQUE-based parallel framework

The database is the newly designed module for TORQUE. It stores current information for later scheduling, e.g. the distribution of decomposed blocks, the status of job execution, and the state of I/O between nodes. The database is hosted in a MySQL instance. The detailed information about these tables is listed in Table 2, 3 and 4.

Field Name	Type	Length
block_name	vchar	40
node_name	vchar	40
node_type	integer	

Table 2. The structure of tbl_block_distribution

Field Name	Type	Length
task_id	vchar	40
node_name	vchar	40
start_time	datetime	
completion_time	datetime	

Table 3. The structure of tbl_task_execution

Field Name	Type	Length
block_name	vchar	40
node_source	vchar	40
node_dest	vchar	40
start_time	datetime	
completion_time	datetime	

Table 4. The structure of tbl_io_information

The pbs_server is the central module in TORQUE. It can communicate with other modules and accept the user's commands via network protocol. Besides its main functions, such as receiving/creating a batch job, modifying the job, and dispatching the job, a specially designed function was added to extract data dependencies of a batch job. The input data are an important criterion for later scheduling.

The pbs_mom is the daemon which places the job into execution on the node where it resides. One pbs_mom runs on each computing node. The pbs_mom receives a copy of the job from pbs_server, creates a new session, places the job into execution, monitors the status of the running job and reports the status to pbs_server. The modification to the pbs_mom enables it to report the data status to the database after successfully executing a job, including input blocks and output results.

The daemon, pbs_sched, implements the administrator's policy to control which job can be ready, when this job is run and with which resources. The pbs_sched communicates with the pbs_server to determine the availability of jobs in the job queue and the state of various system resources. It also communicates with the database for block information to make a scheduling decision.

5. EXPERIMENTS

5.1 Experimental Environment and Datasets

The experimental environment was a six-node Linux cluster running RedHat Enterprise Linux 5.5. Each node has two Quad-Core Intel Xeon CPUs, 8GB DDR2-667 ECC SDRAM, and 1TB hard disk (7200 rpm, 32- MB cache). In this cluster, one node is configured as the master node, while the other five are the workers.

The LiDAR point cloud of Gilmer County, West Virginia is chosen for our experiments, illustrated in Fig.7. It contains 0.883 billion points and occupies 16.4 GB of external space. The average point space is about 1.4m.

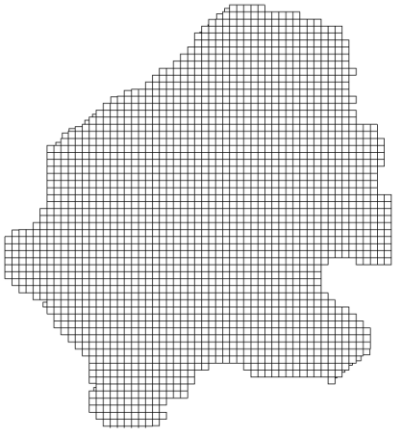

Figure 7. The Gilmer county LiDAR dataset

5.2 Experimental Algorithms

One common LiDAR processing algorithms, Delaunay triangulation (DT), was chosen to demonstrate the proposed Split-and-Merge paradigm. The algorithm was executed on the proposed parallel framework to examine its efficiency and suitability.

The Delaunay triangulation pipeline for our proposed framework is modified from a parallel approach, called ParaStream (Wu et al., 2011). ParaStream integrates traditional D&C methods with streaming computation, and can generate a Delaunay triangulation for billions of LiDAR points on multicore architectures within ten to twenty minutes.

The implementation of Split step in the Split-and-Merge paradigm is to carry out Delaunay triangulation for each decomposed block, erase the finalized triangles from the current triangulation (InnerErase), and output the temporary results. The Merge step in the Split-and-Merge paradigm merges the triangulations of two adjacent blocks and also erases the finalized triangles (InterErase). All these discrete tasks need no neighbor definition. The entire Delaunay triangulation pipeline falls into the type of n-level binary tree.

5.3 Results and Discussion

All the Split and Merge tasks for the algorithm was written in C++ and compiled with linux gcc 4.3. In the experiments, the execution time, speedup, and efficiency were used as the metrics for evaluating the performance of the parallel framework.

The first experiment evaluated the influence of different task granularity on parallel performance. The decomposition size of 1000m was adopted. The detailed test results are listed in Table 5 and shown in Fig. 8.

Processors	DT
1	10380
3	3840
5	3300

Table 5. Execution time (in seconds) with the DT algorithm

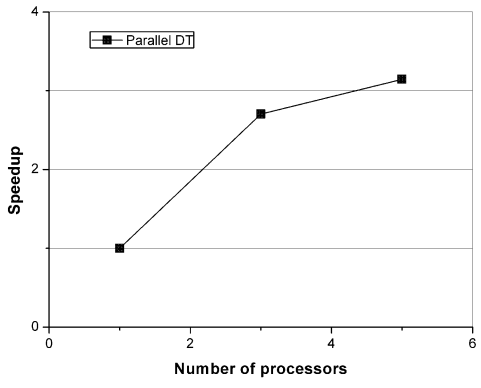

Figure 8. Speedup of parallel DT in this framework

All these experimental results demonstrate that significant speedup and high data-throughput are achieved with this framework. At the same time, with this parallel framework, our

proposed data-aware scheduling algorithm is much more efficient than the traditional FIFO method when a neighbor requirement is present in the user's processing algorithm.

6. CONCLUSION

Parallel computing has been increasingly used to solve data-intensive problems in geospatial science. Inspired by these problems, this paper proposed a universal parallel framework for processing massive LiDAR point clouds in a HPC environment. Within this framework, the user/programmer is supported with a predefined Split-and-Merge programming paradigm. In this paradigm, user/programmers can focus on the simple functional expression of their specific algorithm into two distinct programs, Split and Merge, and leave parallelization and scheduling to the runtime system. This framework automatically and intelligently handles key scheduling decisions for tasks and data. For considering data sharing between task inputs, a specific data-aware scheduling algorithm is proposed to decrease the data communication time. One common LiDAR algorithm, DT, was evaluated to prove the efficiency and suitability of our proposed framework.

ACKNOWLEDGEMENTS

This work is supported by the Natural Science Foundation of China (Grant: 40971211 and 40721001).

REFERENCES

Borkar, S. and Chien, A., 2011. The future of microprocessors. *Communications of the ACM*, 54(5), pp.67-77.

Dean, J. and Ghemawat, S., 2008. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1), pp. 107-113.

Guan, Q. and Clarke, K., 2010. A general-purpose parallel raster processing programming library test application using a geographic cellular automata model. *International Journal of Geographical Information Science*, 24(5), pp.695-722.

Hawick, K., Coddington, P. and James, H., 2003. Distributed Frameworks and Parallel Algorithms for Processing Large-Scale Geographic Data. *Parallel Computing*, 29(10), pp.1297-1333.

Jonker, P., Olk, J., and Nicolescu, C., 2008. Distributed bucket processing: A paradigm embedded in a framework for the parallel processing of pixel sets. *Parallel Computing*, 34(12), pp.735-746.

Staples, G., 2006. TORQUE resource manager. In: Proceedings of the 2006 ACM/IEEE conference on Supercomputing. Tampa, Florida.

Tehranian, S., Zhao, Y., Harvey, T., Swaroop, A. and Mckenzie, K., 2006. A robust framework for real-time distributed processing of satellite data. *Journal of Parallel and Distributed Computing*, 66(3), pp.403-418.

Wang, H., Fu, X., et al., 2011. A common parallel computing framework for modeling hydrological processes of river basins. *Parallel Computing*, 37(6-7), pp. 302-315.

Wu, H., Guan, X. and Gong, J., 2011. ParaStream: A parallel streaming Delaunay triangulation algorithm for LiDAR Points on Multicore Architectures. *Computers & Geosciences*, 37(9), pp.1355-1363.