Comparison of MapReduce, MPI and OpenMP for All-Pairs Shortest Path Problem

Sol-Ji Kang, Sang-Hun Han, Go-Woo Kim, Keon-Myung Lee
Chungbuk National University, Korea
E-mail : solji9931@gmail.com, likelamb@gmail.com, rlarhdn66@naver.com, kmlee@chungbuk.ac.kr

1. Introduction

The All-Pairs-Shortest-Path problem is to find the shortest path between all pairs of nodes in a graph. Floyd-Warshall algorithm is one of best known algorithms to this problem, which iteratively searches the shortest paths by considering the intermediate nodes one by one.

With the advances of hardware and parallel and distributed computing technology, many of developers have been interested in employing high performance computing technology in their applications. This paper presents a comparative works of MapReduce, MPI, and OpenMP for the All-Pairs-Shortest-Path problem. We implemented Floyd-Warshall algorithm using MapReduce, MPI and OpenMP on a cluster of commodity computers, and run on graphs of different sizes. Then we compared their performance and tried to provide a guideline about which is right tool for the job.

2. Related Works

2.1. MapReduce

MapReduce is a programming paradigm which organizes the job into a pair (or a sequence of pairs) of Map task and Reduce task. It assumes that input comes from file(s) and output is saved into files. Data files consist of records each of which can be treated as key-value pair. Input data is partitioned and processed by Map processes, and their processing results are shaped into key-value pairs and shuffled into Reduce tasks according to key. Map processes are independent each other and thus they can be executed in parallel without collaboration among them. Reduce processes play role of aggregating the values with the same key. MapReduce programs can be run on the Hadoop framework. MapReduce paradigm is a good choice for big data processing because MapReduce handles data record by record without loading whole data into memory and in addition the program is executed in parallel over a cluster of commodity computers. It is very convenient to develop big data handling programs using MapReduce because Hadoop provides everything needed for distributed and parallel processing behind the scene which program do not need to know.

2.2. MPI

MPI (Message Passing Interface) is a message-passing library specifications which defines an extended message-passing model for parallel, distributed programming on distributed computing environment. It is not actually a specific implementation of the parallel programming environment, but several products have been implemented according to it. MPI is such an example which is implement on NFS (network file system). It allows the processes to easily pass messages using libraries without deep understanding the underlying mechanism. It provides various library functions to coordinate message passing in various modes like blocked and unblocked message passing. It can send messages of Giga bytes size between processes. MPI-based programs can be executed on a single computer or a cluster of computers.

2.3. OpenMP

OpenMP (Open Multi-Processing) is a shared-memory multi-processing API which is supported on various platforms like UNIX, LINUX, Windows and various languages like C, C++, Fortran. The programs using OpenMP are compiled into multi-threaded programs, which share the same memory address space and thus the communication between thread can be very efficient. OpenMP is equipped with several structures which make it possible fine-grained control over the threads. It is advantageous for programmer to write multi-threaded programs without serious understanding of multithreading mechanism.

3. Distributed Algorithms for the All-Pairs Shortest Path Problem

To compare the pros and cons of MapReduce, MPI and OpenMP to the All-Pairs shortest path problem, it presents the distributed algorithms for each environment. The algorithms are based on Floyd-Warshall algorithm as shown in Figure 1. In algorithms, $D = (d_{ij})$ denotes the matrix containing the distance d_{ij} between nodes i and j, $\Pi = (\pi_{ij})$ is the matrix containing the information about the shortest paths between nodes, $d_{ij}^{(k)}$ is the distance found so far which is obtained using the node set $\{1, 2, \ldots, k\}$ as intermediate nodes, and $\pi_{ij}^{(k)}$ is the predecessor of node j on the shortest path from node i to j obtained by using the node set $\{1, 2, \ldots, k\}$ as intermediate nodes.

Figure 2 shows the pseudo code for MapReduce algorithm for the All-Pairs shortest path problem, Figure 3 and Figure 4 are the pseudo code for MPI algorithm and OpenMP algorithm, respectively.

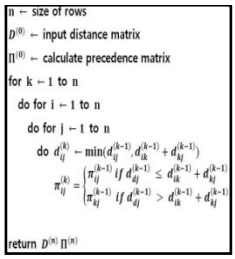

Figure 1. Floyd-Warshall algor

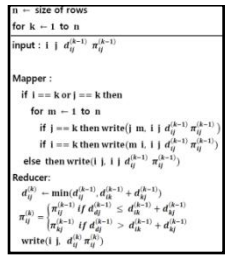

Figure 2. MapReduce algor

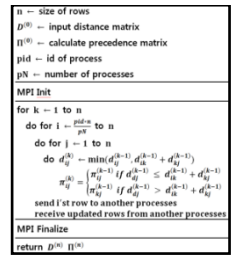

Figure 3. MPI algor

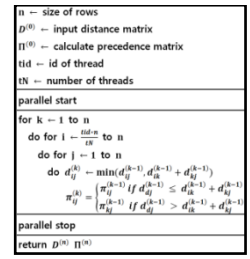

Figure 4. OpenMP algor

4. Experiments

The algorithms were all implemented on the corresponding environment. MapReduce program was executed on a cluster of 5 PCs, and MPI and OpenMP programs were executed on a single PC. The sample graphs were generated at random for a given number of nodes and average number of edges. In the experiments, 3 sample graphs were used of which node size were 10, 100 and 1000. Table 1 shows the execution time obtained in the experiments.

[Table 1] Execution time

Framework / Node size	MapReduce	MPI	OpenMP
10	2m 26s	0.34s	0.1s
100	16m 52s	0.41s	0.25s
1000	4h 4m 39s	24.14s	8.03s

5. Conclusions

As the data size gets big, the developers consider the parallel and/or distributed computing as their execution environment. To see their relative advantages, we have conducted some experiments on such three representative environments for the All-Pairs shortest path problem. If the problem size can be accommodated on single machine, OpenMP can be a choice. In the case of big data, MapReduce is the very choice but it takes considerable time for the problems requiring much iteration, like All-Pairs shortest path problem. MPI allows more flexible control structures than MapReduce, hence MPI is a good choice when a program is needed to be executed in parallel and distributed manner with complicated coordination among processes.

6. Acknowledgement

This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ITRC(Information Technology Research Center) support program(NIPA-2013-H0301-13-4009) supervised by the NIPA(National IT Industry Promotion Agency)

7. References

[1] A. Alexandrov, S. Ewen, et al, "MapReduce and PACT - Comparing Data Parallel Programming Models", In BTW, 2011, pp. 25-44.
[2] G. Jost, H. Jin, et al, "Comparing the OpenMP, MPI, and Hybrid Programming Paradigm on an SMP Cluster", In Proceedings of *EWOMP, 2003*
[3] C. Ranger, R. raghuraman, et al, "Evaluating MapReduce for Multi-core and Multiprocessor Systems", In High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on. IEEE, 2007. pp. 13-24.
[4] S. J. Plimpton, K. D. Devine, "MapReduce in MPI for Large-scale Graph Algorithms", Parallel Computing, 2011. pp. 610-632.
[5] M. Resch, B. Sander, I. Loebich, "A comparison of OpenMP and MPI for the parallel CFD test case", In Proc. of the First European Workshop on OpenMP, 1999. pp. 71-75.