CS343: Introduction to Parallel and Distributed Computing
Laboratory Exercise No 4: Project

Ekow. J. Otoo

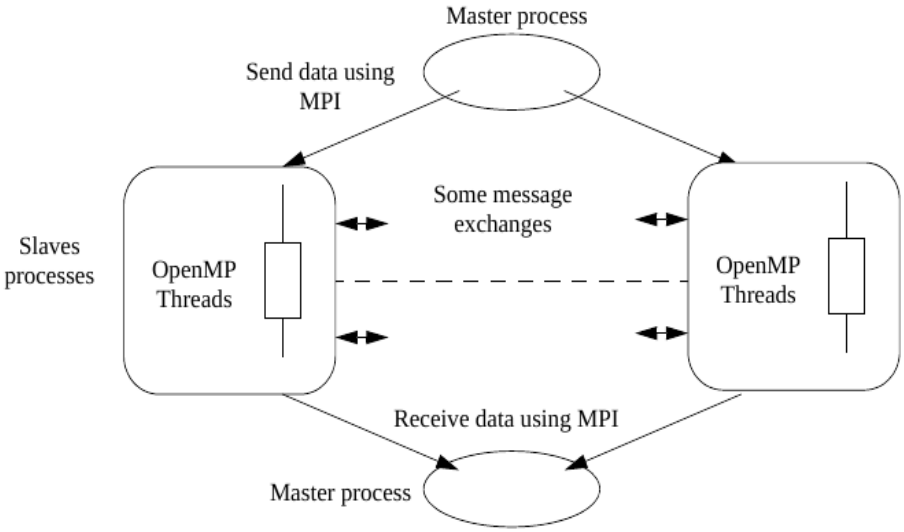
Preamble

You are expected to work in groups of 2. Each member of a group is expected to contribute equal amount of work as the other member of the group. You'll need to work in a virtual environment with one member hosting the code. Same marks will be assigned to every member. You are expected to work with a good understanding of some background skills required; namely:

- OpenMP
- Message Passing Interface (MPI);
- Unified Parallel C (UPC);
- MapReduce Programming Model in Shared Memory
- Use of Bloom Filter - to be explained and encouraged to read the reference.

Note that due to time constraint and unavailability of cluster resources not all particular programming models may have been covered in detail in class. In this case you may have to read about the basics and discuss the details with your instructor or FI. Further explanations, if needed, can be given during office hours or at any reasonable time. The general requirements of most of the project involves comparison of at least one of the hybrid programming model your group chooses with a your developed plan for parallel equi-join computation. The choice of the second approach could be, for example, the use of MapReduce, etc, and conducting some comparative performance analyses. The performance analyses should be illustrated with speed-up and scalability plots.

The project has some research bias. We would like to either reproduce results for a good systems research paper that are especially interesting, and possibly change some of the assumptions, and/or part of the system set-up to develop, implement and evaluate a new system. The basic philosophy of the project is to learn to program in a hybrid environment with MPI, where the programs are cognisant of node failures or link failure. Hybrid parallel programming consists of mixing several parallel programming paradigms in order to benefit from the advantages of using each approach on its own. In general, MPI is used for communication between processes (on different nodes) , and another paradigm (OpenMP, Pthreads, openACC, etc.), is used inside each node. The model may be depicted as shown below.



Brief Problem Description

The original problem is very much simplified given the limitations of the infrastructure available to us; practically we are limited to the use of our laptops. The idea is simply to develop and implement a parallel equi-join of two very large tables on their common join attributes. The two big relational tables $R_1(A, B)$ and $R_2(A, C)$, are maintained as two independent files, R_1 and R_2. These are to be joined on a their common attribute A to produce a relation $R_3(A, B, C)$ that is stored in a file R_3.

The simple idea is for a master process to read the relations into two tables $R_1(A, B)$ and $R_2(A, C)$. The master process then sends R_1 to node-1 (Process-1) and R_2 to node-2 (Process-2). To compute the equi-join, node-1 needs to request the tuples from node-2 that are likely to form a join with tuples of R_1. If node-2 only knows what values are required on node-1, it will then send to node-1, only the relevant tuples. Note that the idea is to reduce as much as possible the information exchanged between the two nodes; node-1 and node-2.

For node-1 and node-2 to know what tuples each node has so that they can eventually compute the equi-join, they use the principle of a Bloom Filter. You can google with the key words "Bloom Filter" or check the details on "https://en.wikipedia.org/wiki/Bloom_filter." The basic idea is to take a long bit-string X[N] of about say $N \geq 10000$ bits and say three distinct hash function $h_1(), h_2(), h_3()$. The bits in the bit-string are initialised to be zeroes. At node-1, the three hash functions are each used to hash the keys $R_1[A]$ to set possibly three distinct positions to 1. For every position hit, the corresponding bit is set to 1. There could be a collision but that is OK. After all the keys of R_1 have been hashed, the bits-ring X is then sent to node-2. Node-2 also repeats the same hashing exercise but this time, if all three hash positions are found to be already set to 1, the corresponding record is recognised as a possible candidate to form a join with a counterpart record of R_1. All such identified records on node-2 are collected and sent to node-1.

The process can be repeated from node-2 to node-1 to get node-1 to be aware of the candidate

equi-join records on node-2. Such bit-string exchanges can be made a number of times between node-1 and node-2 until no more reduction in the number of equi-join records can be achieved. In practice, two rounds of bit-string exchanges are sufficient.

In our project one round of the bit-string exchange is sufficient to understand the idea. The relational join operator is used to relate information from two or more tables. Thus, joins are frequently occurring operations in relational queries. The most frequently occurring group of operator in a relational database is the $S-P-J$ group referred to as the *Select-Project-Join* operations. The *join* being the most expensive. For this reason, efficient join algorithms are critical in determining the performance of a relational database system.

Input Data

Some sample data will be made available for experimentation. You may also generate some artificial data on your own for your tests, or download some large data-sets. Some very large data-sets can also be generated from TPC.org benchmarking suite.

Write-Ups

The write-up of your work should be in a style suitable for submission as a paper to a conference or a journal on information processing. This is typically about $8 \sim 12$ pages in one-half-spacing including references, tables, diagrams and figures. I'll seriously consider a well written project report, after corrections and rewrites, for submission to either a conference or a Journal as a short articles.

Marking and Contribution of Efforts

The same project mark will be given to both members of the group. Please ensure that each group member contributes an almost equal effort towards the project.

Mark Distribution

Component	**Issues Addressed**	**Points**
Programs [60]	I/O and/or Data Generation	20
	Algorithm-1: MPI Native Program	20
	Self Developed Algorithm	20
Report [40]	Abstract + Introduction	5
	Problem Description	5
	High-Level MPI- Algorithm-1.	5
	High-Level Other- Algorithm-2.	5
	Descript. Exp. Environment	5
	Discussion of Results	10
	Conclusion + References	5