An Agent-Based Multidatabase System

XIAOTIAN LI

Term Report

CSS 595: Masters Project

Professor Munehiro Fukuda

Autumn, 2020

1. Introduction

The Mathematical Model of Meaning (MMM) is a multidimensional database system different from the SQL-like database [1]. In the traditional SQL database, it uses SQL language to query the database by compare user-defined conditions. In MMM, it extracts features from a database and explains the database using the data item's feature. For example, in the Longman dictionary, MMM utilizes 871 features words to explain 2115 words. Each feature is translated into a column, and each data item is represented as a row in a matrix. Besides, MMM is also able to realize semantic meaning among data items. It can distinguish the meaning of data items under different contexts, and users can define the different context to customize their query. For instance, the word "buck" has a different meaning in the context of gambling and haunting. In the gambling context, "buck" is slang for a dollar. However, in the hunting context, it means some horned animals. Therefore, MMM is valuable for the applications which require different query context. MMM consists of three steps. Step 1 is a high-dimensional space creation with a correlation matrix, this step includes large-scale matrix multiplication. Step 2 is a user-defined subspace extraction with eigen-decomposition. Step 3 is sorting of data through Euclid distance calculation. All of these 3 steps are computationally intensive; therefore, the Parallelization is necessary for MMM.

In the past quarter, we have been working on parallelizing MMM with the MASS library. The MASS library is an Agent-Based Modeling library that stores data in the distributed places and propagates agents to interact with the data [2]. For the first step of MMM, we were able to parallelize matrix multiplication to get 85% performance improvement over MPI and 95% performance improvement over sequential program. Moreover, since MMM requires extensive file reading on computing nodes, we also implemented parallel I/O to support csv file reading, which generated a satisfying result as well.

2. Matrix Multiplication

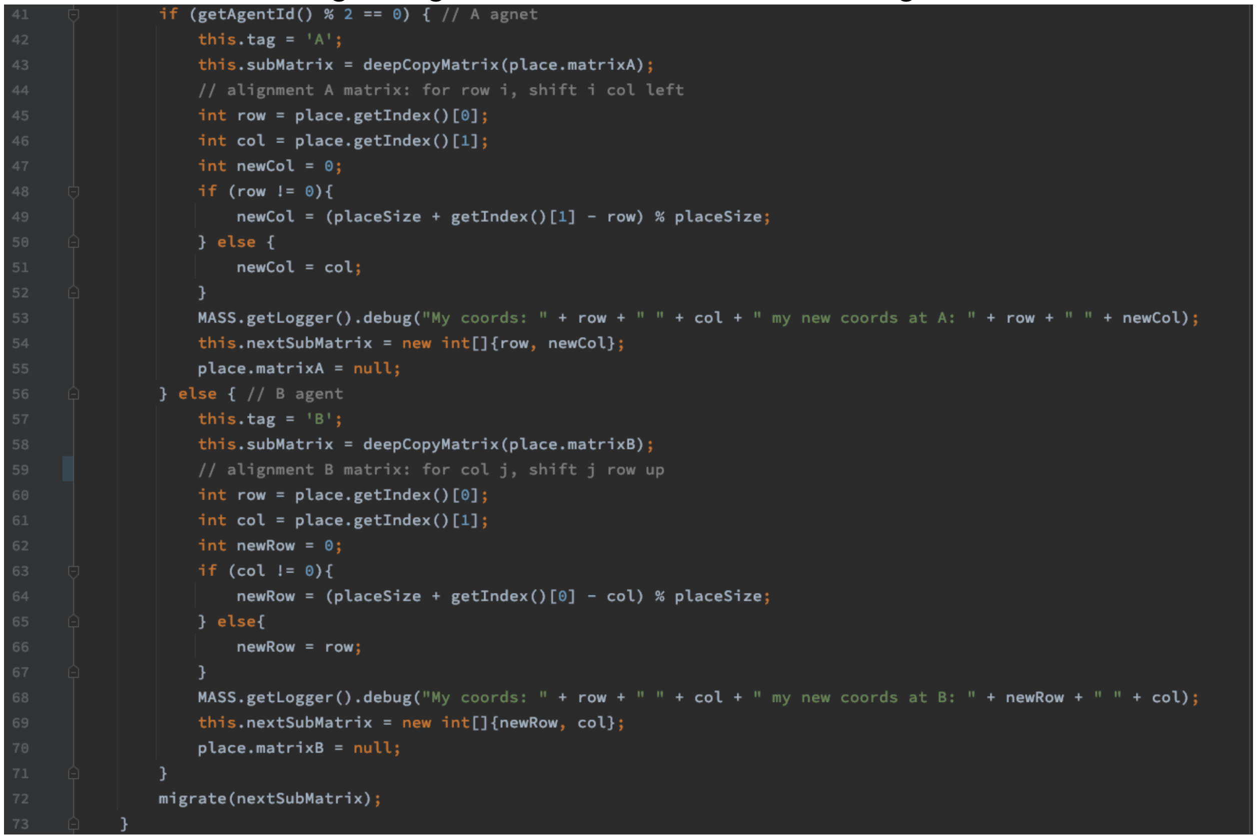
2.1 Cannon's Algorithm with MASS

The matrix multiplication is one of the most popular benchmarks for the parallel framework. In parallel computing, people use Cannon's algorithm to improve the performance. In general, Cannon's algorithm includes two steps. The first step is to align the matrix of A and B so that in each processor $P_{i,j}$, the *j* value of A is same as the *i* value of B. The second step is to move each submatrix A one step left and submatrix B one step up and to perform block multiplication. Finally, the multiplication result is generated by adding up results of each shifting.

In order to implement cannon's algorithm using MASS, we created a *SubMatrix* class and a *Shifter* class to extend *Place* and *Agent* respectively. For the *SubMatrix* class, it provides memory and computing resources. Each place is initialized in the CPU core of a different computing node. Upon initialization, place will read the data from a CSV file and save their corresponding portion into the memory space.

In the Shifter class, it firstly spawns two agents at each place. One agent is responsible for the matrix A, and the other one is responsible for the matrix B. As shown in the Listing 1, In order to make the initial alignment, each agent needs to grab the data from place where it was born and migrate it to the designated places (line 43 and line 58). In line 48 to 52 and line 63 to 67, agent A and agent B are finding the coordinate of destination place. Agent A needs to shift *i* columns left and agent B needs to shift *j* rows up. After initial alignment, each agent does the matrix calculation and saves the temporary result into the place's memory. When finishing the initial alignment, agents will be responsible to exchange data. Compared with the MPI approach, agents move over matrix instead of the conventional approach to shifting data to neighboring elements. MASS provides a more flexible way for data exchange; developers do not need to worry about the potential deadlock, which is likely to happen with MPI. In MPI, *MPI_SEND* and *MPI_RECV* will block the program until the send/receive buffer can be reused. If both ranks send data through a same buffer, the program will deadlock.

Listing 1. Alignment of submatrix in Cannon's Algorithm



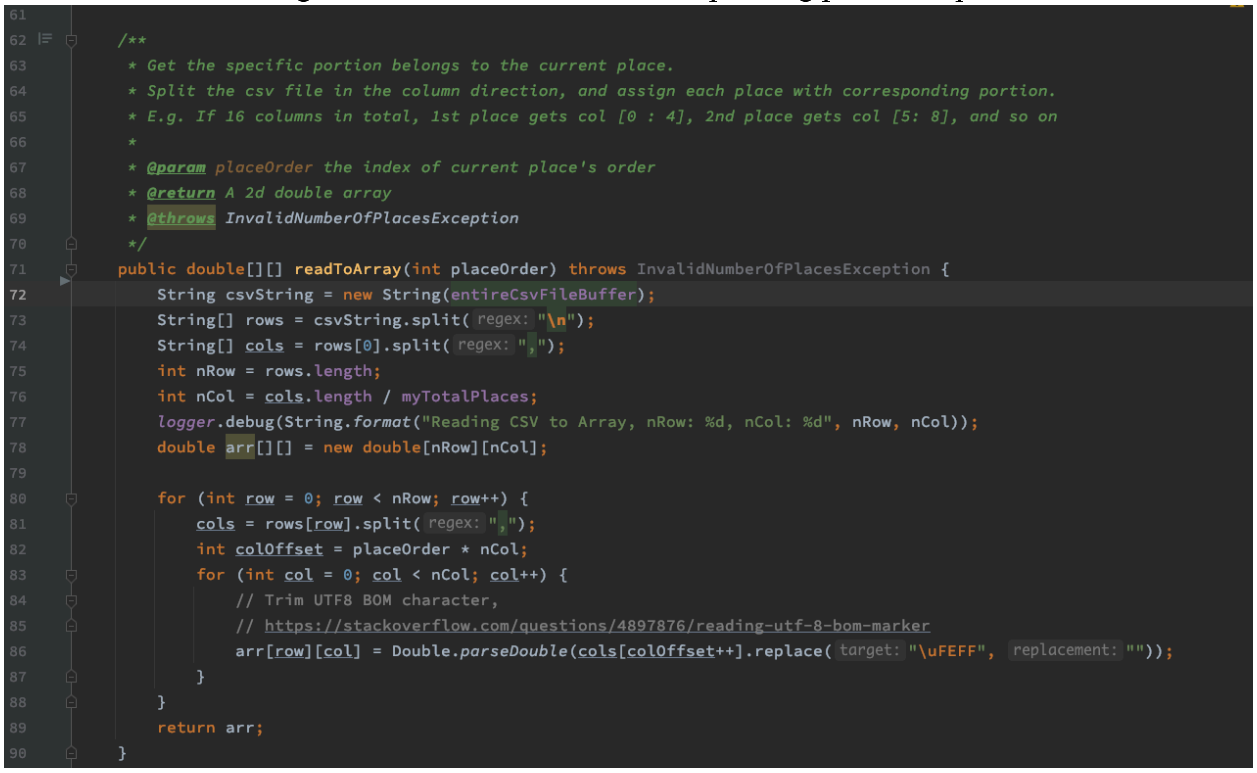
```
41          if (getAgentId() % 2 == 0) { // A agnet
42              this.tag = 'A';
43              this.subMatrix = deepCopyMatrix(place.matrixA);
44              // alignment A matrix: for row i, shift i col left
45              int row = place.getIndex()[0];
46              int col = place.getIndex()[1];
47              int newCol = 0;
48              if (row != 0){
49                  newCol = (placeSize + getIndex()[1] - row) % placeSize;
50              } else {
51                  newCol = col;
52              }
53              MASS.getLogger().debug("My coords: " + row + " " + col + " my new coords at A: " + row + " " + newCol);
54              this.nextSubMatrix = new int[]{row, newCol};
55              place.matrixA = null;
56          } else { // B agent
57              this.tag = 'B';
58              this.subMatrix = deepCopyMatrix(place.matrixB);
59              // alignment B matrix: for col j, shift j row up
60              int row = place.getIndex()[0];
61              int col = place.getIndex()[1];
62              int newRow = 0;
63              if (col != 0){
64                  newRow = (placeSize + getIndex()[0] - col) % placeSize;
65              } else{
66                  newRow = row;
67              }
68              MASS.getLogger().debug("My coords: " + row + " " + col + " my new coords at B: " + newRow + " " + col);
69              this.nextSubMatrix = new int[]{newRow, col};
70              place.matrixB = null;
71          }
72          migrate(nextSubMatrix);
73      }
```

2.2 Parallel I/O for CSV Files

After examining the performance results, we found the most prominent overhead in our implementation is the csv file reading. It takes over 4500 ms to read the 48mb csv file into the memory space. This sequential file reading methodology does not fully use the power of every computing node. Instead, only one computing node is reading the CSV file at one time. Therefore, we decide to add a CSV feature to MASS parallel I/O.

In order to enhance the parallel I/O within the MASS library, each place must be able to open and read the same CSV file in parallel. All the current implementations have been done within the Place class. We implemented this CSV file reading feature on top of the current parallel I/O class. In the current design of parallel I/O, it firstly opens the file by taking the given *filePath* argument, which determines the file's name and whether the file's type is supported. Currently, MASS parallel I/O supports Netcdf, Text and Csv file. If the given file type is not supported, then -1 is returned. Otherwise, this file will be opened and saved into the *fileTable*. The open method is a synchronized method that means only one place will handle each computing node's open task. The file reading is implemented by splitting the file into an equal number of bytes, and each computing node will get a corresponding portion. However, for the csv file, since each data record's character length is different, directly splitting the file will make records lose some digits. To solve this problem, we add paddings on each line to make sure every line will have the same length. Therefore, when we split the file by an equal number of bytes, every computing node will get the same number of line data. As shown in Listing 2, we first retrieve the whole string from the memory buffer and get number of rows and columns. Afterwards, we loop through every line and assign corresponding columns to each place. In our current implementation, each computing node breaks the data vertically and distributes it to each place. This design aligns with Cannon's algorithms' requirement so that each place can save the data as a submatrix and start doing the calculation right away.

Listing 2. Read and distribute a corresponding portion to places



```java
/**
 * Get the specific portion belongs to the current place.
 * Split the csv file in the column direction, and assign each place with corresponding portion.
 * E.g. If 16 columns in total, 1st place gets col [0 : 4], 2nd place gets col [5: 8], and so on
 *
 * @param placeOrder the index of current place's order
 * @return A 2d double array
 * @throws InvalidNumberOfPlacesException
 */
public double[][] readToArray(int placeOrder) throws InvalidNumberOfPlacesException {
    String csvString = new String(entireCsvFileBuffer);
    String[] rows = csvString.split( regex: "\n");
    String[] cols = rows[0].split( regex: ",");
    int nRow = rows.length;
    int nCol = cols.length / myTotalPlaces;
    logger.debug(String.format("Reading CSV to Array, nRow: %d, nCol: %d", nRow, nCol));
    double arr[][] = new double[nRow][nCol];

    for (int row = 0; row < nRow; row++) {
        cols = rows[row].split( regex: ",");
        int colOffset = placeOrder * nCol;
        for (int col = 0; col < nCol; col++) {
            // Trim UTF8 BOM character,
            // https://stackoverflow.com/questions/4897876/reading-utf-8-bom-marker
            arr[row][col] = Double.parseDouble(cols[colOffset++].replace( target: "\uFEFF",  replacement: ""));
        }
    }
    return arr;
}
```

3 Results

We conducted our matrix calculation results on the cssmpiXh clusters using the latest MASS version 1.3.0. We also compared performance among MASS, MPI, and sequential program. Overall, as shown in Table 1, MASS has the fastest file reading performance at 1023ms, which is 2.2 times faster than the sequential program and 4.12 times faster than MPI. When we look at the calculation time, MASS also won the competition. The average computation time on MASS was at 5961ms. On the other hand, MPI and the sequential program took 38791ms and 129816ms, respectively. By comparing the performance between MASS and MPI, we can see that MASS improves around 85%. This is because MASS uses multi-core capability whereas MPI only uses a single core on each machine. In this case, the number of computing nodes of MASS increases from 4 to 16. Therefore, each computing node will be responsible for a smaller submatrix size so that the performance is faster than MPI. Overall, in our experimental results, MASS shows a convincing result at both system I/O time and at calculation time.

Table 1. Performance comparison for Matrix Calculation

No.	MASS Initialization Time	MASS Place Initialization Time	File Reading Time	Calculation Time	Total Time	Type
1	N/A	N/A	2120	115604	117724	Seq
2	N/A	N/A	2396	137150	139546	Seq
3	N/A	N/A	2259	136696	138955	Seq
4	N/A	N/A	4198	38711	42909	MPI
5	N/A	N/A	4122	38910	43032	MPI
6	N/A	N/A	4346	38752	43098	MPI
7	15255	117	1092	5828	22292	MASS
8	15034	134	962	5953	22083	MASS
9	15169	112	1015	6102	22398	MASS

4. Future Research

The next steps for this research are to parallelize the step 2 and step 3 with MASS. For step 2, we would like to use Householder transformation and figure out an approach to apply MASS on. In step 3, the distance calculation is a perfect scenario to utilize agents' mobility to discover the closest agent.

Upon finishing MMM's implementation with MASS, we would like to test the whole system with large-scale datasets. There are two aspects we are looking at the test. Firstly, we want to know the practicability of MASS in resolving real-world problems. MASS has shown significant potential for scientific and biological problems, but it has not been used on many real-world problems. It would be a perfect opportunity for MASS to test its practicability. Moreover, the query accuracy and our system's overall performance is another part we want to test. If the testing results are satisfying, we would like to apply our multidimensional database on problems such as image and video retrieval, semantic search, recommender systems, etc.

5. References

1. “MASS: A parallelizing library for multi-agent spatial simulation.” [Online]. Available: http://depts.washington.edu/dslab/MASS/
2. T. Kitagawa and Y. Kiyoki, "A mathematical model of meaning and its application to multidatabase systems," Proceedings RIDE-IMS `93: Third International Workshop on Research Issues in Data Engineering: Interoperability in Multidatabase Systems, Vienna, Austria, 1993, pp. 130-135, doi: 10.1109/RIDE.1993.281933.