CMPE300 - Analysis of Algorithms 2014 Fall Parallel Programming Project

Atakan Arıkan
2011400243

Explanation of the Project

In this project we were asked to implement a parallel algorithm, using C/C++ with the MPI library. The problem was about a person's sleeping data. We needed to work on the data but since it was too large, we needed to use parallelism in our code in order to gain from time.

Example Data:

recordtime	accelerationX	accelerationY	accelerationZ
1	-0.06387329	-0.02742004	-1.012283
2	-0.06611633	-0.02549744	-1.014832
3	-0.0940094	-0.01487732	-0.9822693
4	-0.03929138	-0.01205444	-1.023972
5	-0.03903198	-0.02276611	-1.015198
6	-0.02810669	-0.02711487	-1.012329
7	-0.02963257	-0.02600098	-1.015549
8	-0.03118896	-0.02496338	-1.010818
9	-0.0296936	-0.02507019	-1.010376
10	-0.03474426	-0.02088928	-1.011963
11	-0.03240967	-0.02444458	-1.005692
12	-0.03179932	-0.02433777	-1.008087
13	-0.02679443	-0.02392578	-1.01152
14	-0.03535461	-0.02268982	-1.009903
15	-0.04534912	-0.01869202	-1.007355

According to this data, we needed to compare the same column values in two adjacent rows and according to their differences, we'd decide which sleeping state this person is in. We have three states:

- **Deep sleep:** Acceleration change <= 0.008 corresponds to deep sleep.
- **Light sleep:** Acceleration change > 0.008 and < 0.03 corresponds to light sleep.
- **Awake:** Acceleration change >= 0.03 corresponds to awaking from sleep or being awake.

If any of the *accelarerationX*, *accelarerationY* and *accelarerationZ* values have a change more than 0.008 than the person should be
in light sleep (>= 0.03 for being awake.)

Apart from the state, we need to count how many movements this person does in his sleep. A movement is counted only if this person goes from deep sleep to light sleep, or awakens from either deep sleep or light sleep.

Explanation of the Algorithm

The algorithm starts running and initializes MPI:

```
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

Now we have our *Message Passing Interface* ready, we can start reading data in master processor, which has a rank value of zero:

```
/* master node */
if(rank==0){
// do stuff
}
```

First of all, extracted the filename from arguments. Then I got the *startingHour* value from arguments as well. After that, I split that string by the delimiter ":" and calculated it's value in terms of seconds:

```
startingHour=argv[argc-1]; // starting time of the sleeping data
/* extract the start hour data */
            size_t pos = 0;
            string token, startHour, startMin;
            string delimiter = ":";
            while ((pos = startingHour.find(delimiter)) != string::npos) {
                   token = startingHour.substr(0, pos);
                   startHour = token;
                   startH = atoi(token.c_str());
                   startingHour.erase(0, pos + delimiter.length());
            }
            startMin = startingHour;
            startM = atoi(startingHour.c_str());
            startSec = startH*3600 + startM*60; // find the value of starting hour in
terms of seconds.
```

After doing that, I read the whole file once just to get the size of the data:

```
/*get the size of the data*/
ifstream myfiletemp(filename);
string line;
if(myfiletemp.is_open()){
      while(getline(myfiletemp, line)){
            arrSize++;
      }
}
```

Then, I initialized my arrays:

```
sleepData = new double*[arrSize];
awakeData = new int[arrSize];
for(int i=0; i<arrSize; i++){
      sleepData[i]= new double[4];
}
```

Awake data has a size of *arrSize* since there can't be a awake data size bigger than *arrSize*. Then, I opened the same file again, and filled my *sleepData* array.

Now we have our data stored in our container, we can start distrubiting it to the worker processors, but for that to happen, we need to calculate the *beginIndex* and *endIndex* values for each processor.

```
        for(int i=1;i<size;i++){ // for each worker processor
            beginIndex = (i-1)*(arrSize/(size-1));
            endIndex = i*(arrSize/(size-1));
            if(i==size-1){ //data for the last processor
                endIndex = arrSize - 1;
            }
            int tempSize = endIndex-beginIndex;
            currentTime = beginIndex;
            .
            .
            .

        }


    for(int m=beginIndex; m<=endIndex; m++){ // m<=endIndex since we want the endIndex
line to be in 2 processors.
        for(int j=0; j<4; j++){
            MPI_Send(&sleepData[m][j],1,MPI_DOUBLE,i,0,MPI_COMM_WORLD);
        }
    }
```

We sent the partial data to each processor. Now it's time to process it.

```
    for(int i=0; i<arrSize; i++){ // where the actual work gets done.
        currentTime=sleepData[i][0];
        diffX=sleepData[i][1]-sleepData[i+1][1]; // difference in X-axis
        diffY=sleepData[i][2]-sleepData[i+1][2]; // difference in Y-axis
        diffZ=sleepData[i][3]-sleepData[i+1][3]; // difference in Z-axis
        if((diffX >= 0.03 || diffX <= -0.03) || (diffY >= 0.03 || diffY <= -0.03) ||
(diffZ >= 0.03 || diffZ <= -0.03)){
            /* we are awake */
            if(currentTime>(lastTempAwake+480)){
                lastTempAwake=currentTime;
                awakeData[awakeNum]=currentTime+1;
                awakeNum++;
            }
            if(currentTime>(lastLight+480) && currentTime>(lastAwake+480) && state <
2){

                movement++;
                lastAwake=currentTime;
                lastTempAwake=currentTime;
                lastLight=currentTime;
                state = 2;
            }
        }else if((diffX > 0.008 || diffX < -0.008) || (diffY > 0.008 || diffY < -0.008)
|| (diffZ > 0.008 || diffZ < -0.008)){
            /* we are in light sleep */
            if(currentTime>(lastLight+480)&& state==0){ // 8 minute rule, and we only
count a move if we were in deep sleep.
                movement++;
                state = 1;
                lastLight=currentTime;
            }
        }else if((diffX <= 0.008 && diffX >= -0.008) || (diffY <= 0.008 && diffY >=
-0.008) || (diffZ <= 0.008 && diffZ >= -0.008)){
            /* we are in deep sleep */
            deepSleepTime++;
            state = 0;
        }
    }
```

This is the loop each processor runs in parallel. Each processor compares the data between rows i and *i+1*. Depending on the difference, determines the state of the person. First they check whether the acceleration difference is enough for the person to be awake, if so, checks if there's an 8 minute difference between last time we encountered such data and current time. Notes the current time so we can use it in *awake at: section.* Also checks which state we were in right before that data is considered. If we were in deep or light sleep, increases the movemet by one. Sets the *lastAwake* value to *currentTime.*

If we get a data that claims we are in light sleep, checks for 8 minute rule and if we came from deep sleep. If both are correct, increments movement and sets *lastLight* value to *currentTime*.

If data tells us we're in deep sleep, we just set the state to deep sleep and increment the amount we spent in deep sleep.

After this for loop, all processors will have their finished data. Now it's time to send the data back to the main processor:

```
/* send finished data back to the master node */
MPI_Send(&movement,1,MPI_INT,0,0,MPI_COMM_WORLD);
MPI_Send(&deepSleepTime,1,MPI_INT,0,0,MPI_COMM_WORLD);
MPI_Send(&awakeNum,1,MPI_INT,0,0,MPI_COMM_WORLD);
for(int j=0; j<awakeNum; j++){
      MPI_Send(&awakeData[j],1,MPI_INT,0,0,MPI_COMM_WORLD);
}
```

Now in master processor, we just need to collect the data:

```
                    /* receive the finished data from worker nodes */
                    MPI_Recv(&tempmovements,1,MPI_INT,i,0,MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
                    movements+=tempmovements;
                    MPI_Recv(&tempdeepSleepCounts,1,MPI_INT,i,0,MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
                    deepSleepCounts+=tempdeepSleepCounts;
                    MPI_Recv(&tempawakeNums,1,MPI_INT,i,0,MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
                    for(int m=0; m<tempawakeNums; m++){

MPI_Recv(&awakeData[m+awakeNums],tempawakeNums,MPI_INT,i,0,MPI_COMM_WORLD,MPI_STATUS_IGN
ORE);
                    }
                    awakeNums+=tempawakeNums;
```

Now all we have to do is to print this data to a file.

Conclusion

I learned a lot about parallel programming. It looked like a hard project at the beginning since I had no idea how to implement a parallel algorithm, but now looking back, it doesn't seem so hard. I enjoyed doing the project.