Report

CSE 6010 ASSIGNMENT 4, Zhiquan Zhang

1.Introduction

This assignment is about different parallel computing methods and the comparison of their efficiency when the size of matrix and the number of threads vary.

My software contains three parts, which are sequential multiplication, static mapping and dynamic mapping. And every part will calculate the multiplication of two matrixes, which are completely made up of float number between 0 and 1. I use the sum of all elements to verify the correctness of my program.

Below are two images of some crucial program.

```
omp_set_num_threads(K);
start = omp_get_wtime();
#pragma omp parallel
{
    int j, nthrds;
    double myC;
    int ID=omp_get_thread_num();
    int num_threads=omp_get_num_threads();
    for(int n=ID; n<SIZE; n+=K){
        for(int i=0; i<SIZE; i++){
            for(j=0, myC=0.0; j<SIZE; j++){
                myC += (A[n][j]*B[j][i]);
            }
            C[n][i] = myC;
            #pragma omp critical
            {
                sum += myC;
            }
        }
    }
}
double sta_map_t = omp_get_wtime()-start;
printf("Static parallel sum is %f\n", sum);
printf("The time is %f\n", sta_map_t);
```

```
omp_set_num_threads(K);
start = omp_get_wtime();
int rank = 0;
#pragma omp parallel
{
    while(rank<SIZE){
        int j;
        double myC;
        int ID=omp_get_thread_num();
        int num_threads=omp_get_num_threads();
        int n;

        //critical part, prevent race for rank
        #pragma omp critical
        {
            n=rank;
            rank++;
        }
        for(int i=0; i<SIZE; i++){
            for(j=0, myC=0.0; j<SIZE; j++){
                myC += (A[n][j]*B[j][i]);
            }
            C[n][i] = myC;
            #pragma omp critical
            {
                sum += myC;
            }
        }
    }
```

2.Verificatin

To verify the correctness of my code. I try to calculate the sum of the elements in the multiplication result. After every method's calculation, I will sum up all the elements of the result matrix. Finally, I will compare the sum of them. It's obvious that three methods can result in the same sum, which prove that my code is correct.

In the following images, the sequential calculation can result in the same sum as those calculated in static mapping and dynamic mapping.

```
Accoding to sequential, sum is 103638657.240000
The time is 8.234553
Static parallel sum is 103638657.240000
The time is 9.087875
Dynamic parallel sum is 103638657.240000
The time is 2.823060
```

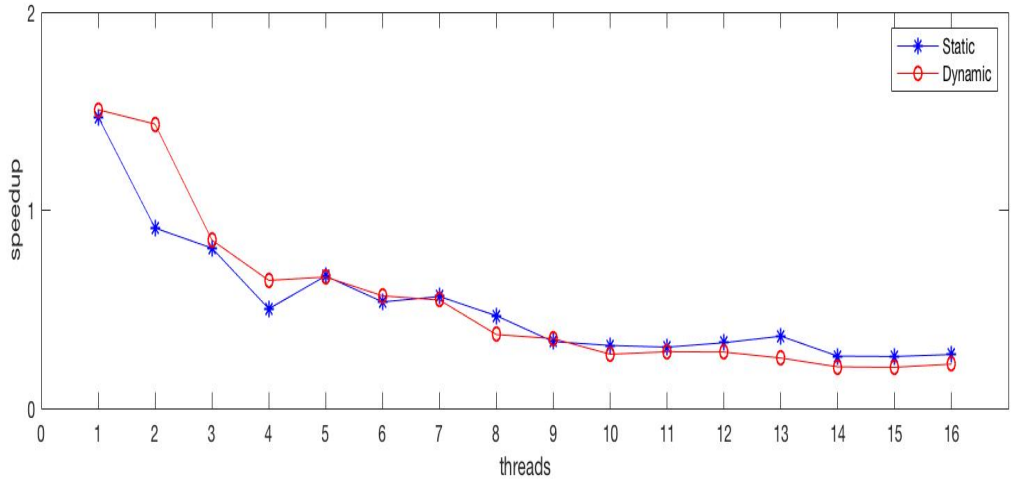
```
Accoding to sequential, sum is 103560356.880000
The time is 9.704462
Static parallel sum is 103560356.879999
The time is 1.757355
Dynamic parallel sum is 103560356.880000
The time is 1.078274
```

3.Experiment

3.1 The experiments on small size matrix

First of all, I do experiments on the matrix of size 50, observe and check the final output of the speedup value of static mapping and dynamic mapping.

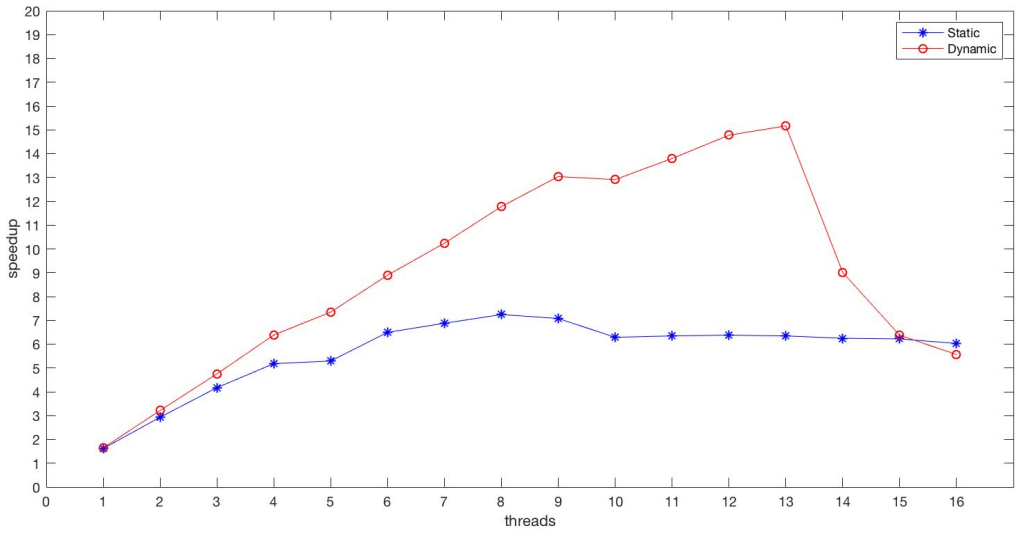
When the thread is 1, the parallel computing seems faster than sequential one. However, with the increasement of the number of the threads, the static and dynamic mapping will be slower. So it's not far to see that when the matrix is small, sequential calculation is faster.



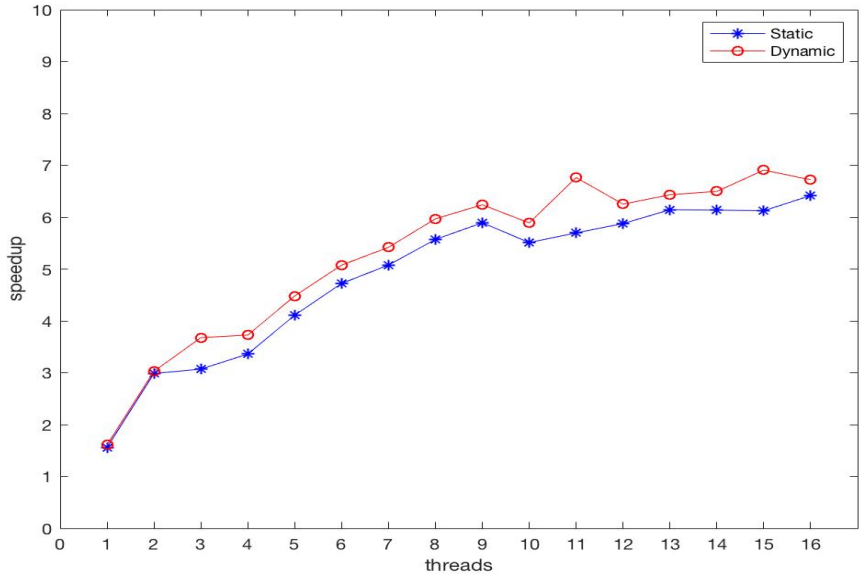
3.2 The experiments on large size matrix

Then I do the experiment of the matrix of size 800, observe and check the final output of the speedup value of static mapping and dynamic mapping.

First of all, it's apparent that the parallel computing is faster than sequential computing so the speedup value is larger than 1. When the threads number increases gradually, the dynamic computing grows faster while the static one is slower. However, after the threads number reaches 13, the dynamic mapping's efficiency begins to slow down. I think this is because when there are too many threads, the allocation will take too much time and thus slow down the speed of calculation. And at this time, the static mapping, in which threads have been arranged before, will be faster.

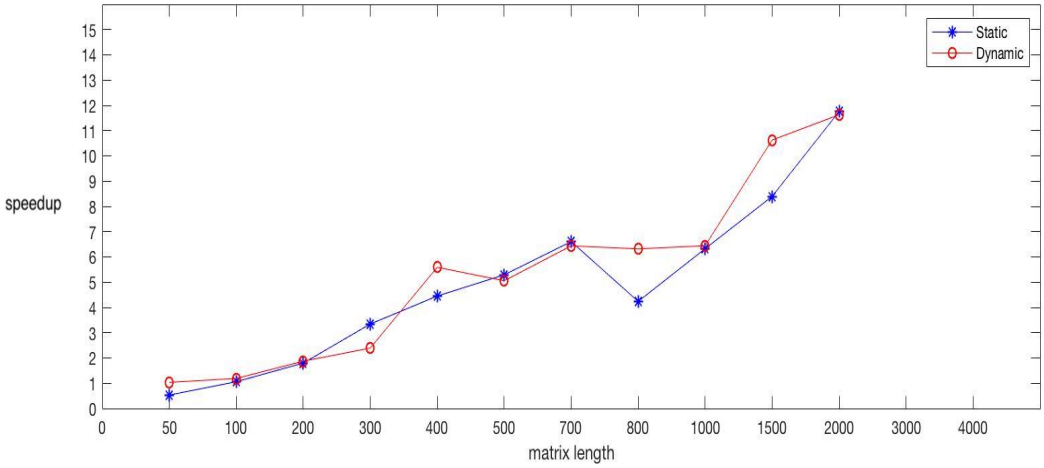


However, when I test the matrix of 1000X1000, it seems that the dynamic mapping is always better than static mapping. I think perhaps the speed sometimes depends on the state of the machine.



3.3 The experiments on different sizes of matrix

When I do experiments on the different length of matrix, I find that when the size gets larger, the advantage of parallel computing over sequential computing is more and more obvious. I think it's because when the size is very large, the multi-threads can make the can deal with more data at one moment, which will make the task much easier.



3.4 Conclusion

To my conclusion, the parallel computing will be better when dealing with large scale of data, especially the data is very big. And the dynamic mapping is generally better than static mapping. However, if the data is not very large, too many threads in dynamic mapping will give rise to the reduction of speed. At this time, static mapping will be more efficient.

4 Research

MPI(Message Passing Interface) is a very popular message passing API, which is widely used for parallel computing. In the paper, the author suggested that matrix multiplication can be implemented through MPI.

MPI's core functions are like:

```
MPI_Send(void* data, int count, MPI_Datatype datatype,int

destination, int tag, MPI_Comm communicator)
```

```
MPI_Recv(void* data, int count, MPI_Datatype datatype, int

source, int tag, MPI_Comm communicator, MPI_Status* status)
```

These two functions can implement the fundamental massage passing from one point to another point. As for broadcast of some message, there is a function like this:

```
int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype,
int root, MPI_Comm comm)
```

Through these functions, we can implement matrix multiplication.

First of all, we need to have a root processor and other worker processor. Assume that we need to calculation the multiplication of matrix A and B. The matrix A is divided into different parts on interval. And then different parts are sent to different work processors. Furthermore, the matrix B is broadcast to all other processors. The root process is also responsible for some calculation.

Workers processors, after receiving their part from A and the complete matrix B, will begin to calculate the multiplication of them separately at the same time. After finishing calculation, every processor will send the result to root process.

The root processor will receive all data from other processor, and the receive function is synchronous, which means that only after the root process successfully receives from one of the processor, it will begin to receive the data from next processor.

Finally, the processor will receive all data from other processor, and return the final result.

Reference:

1.Gropp W D, Gropp W, Lusk E, et al. Using MPI: portable parallel programming with the message-passing interface[M]. MIT press, 1999.