國立陽明交通大學 電機工程學系
Department of Electrical and Engineering

矩陣運算平行化架構設計

專題指導教授:賴伯承 專題學生:大二 何祁恩 coherent17

Matrix multiplication has been widely used in scientific area, such as AI/ML, semiconductor atomic simulation and so on. My project will combine all of the content I had learned in this semester. With comparing the spatial locality of the 6 types of matrix multiplication methods, apply parallel programming with OpenMP on the 6 different methods, and analyze the performance of the program after parallelization.

A. 6 methods

There are 6 types of methods to do matrix multiplication: ijk, jik, jki, kji, kij, ikj. The memory address is continuous in an array, and since C is an row-wise language, if we access the data in the memory which is in the same cache line, then it will be really fast. This is the concept of spatial locality.

The cache not hit rate for these 6 methods is:

$$jki = kji > ijk = jik > kij = ikj$$

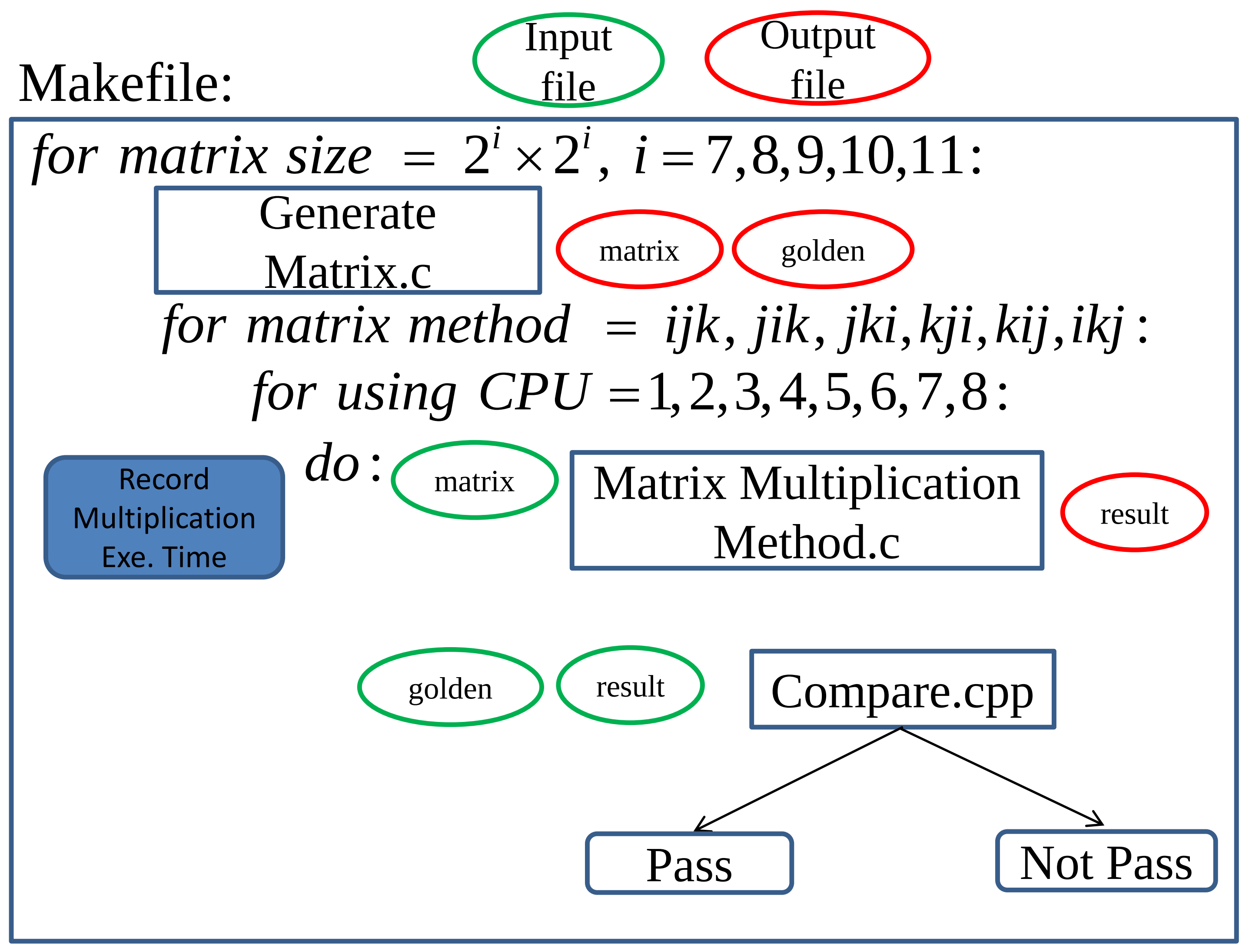
To know more detail, please refer to my note

B. Parallel Programing

Parallel Programming separates the jobs and assign to different CPU to execute. In this project, I use OpenMP to parallel my program. However, the hazard of the parallel programming is data-dependency problem. If there exist data-dependency problem, it must stall all of the jobs, and wait for specific instructions to be done. In kji and kij methods, there exist data-dependency hazard I must to deal with.

C. Procedure

Makefile:



```
for matrix size = $2^i \times 2^i$, $i = 7,8,9,10,11$:
    for matrix method = ijk, jik, jki, kji, kij, ikj :
        for using CPU = 1,2,3,4,5,6,7,8 :
            do :
```

Source code is available on my github account

You can run the program in your own platform through:

```
git clone https://github.com/coherent17/EE-project
cd EE-project/Parallel_Matrix_Multiplication_With_Openmp
make full
```

D. Result Dataset

The execution time for each matrix size, method, number of CPU and data visualization process is also available on my github account.

E. Data Visualization

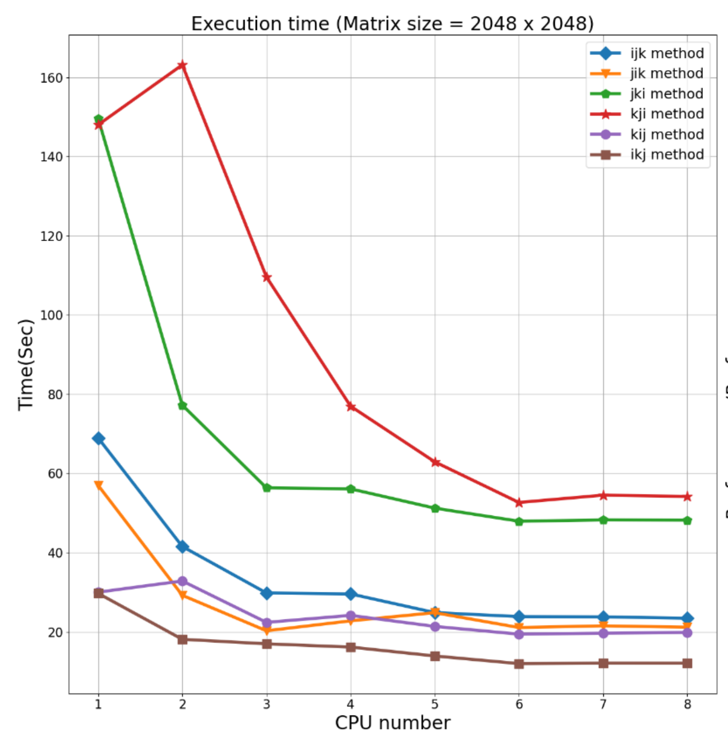


Figure 1.1

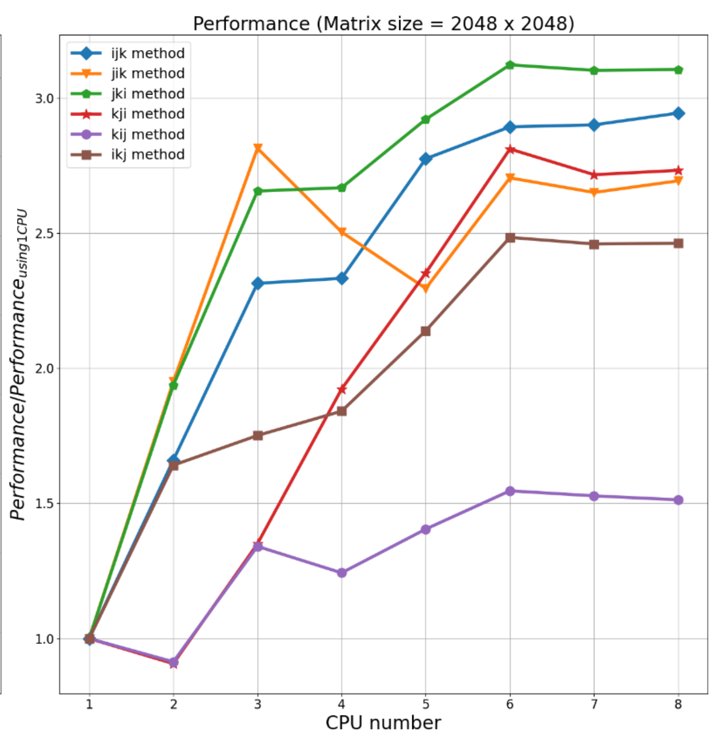


Figure 1.2

In Figure 1.1, shows the execution time of 6 methods. And in Figure 1.2, shows the performance ratio to only use 1 CPU.

(a) $t_{jki}, t_{kji} > t_{ijk}, t_{jik} > t_{kij}, t_{ikj}$

Because of the spatial locality(cache not hit rate)

(b) $t_{kji} > t_{jki}, t_{kij} > t_{ikj}$

In both pair, although their cache not hit rate is the same, but in method kji and kij I need to deal with the data-dependency problem, therefore the time is much longer.

(c) The performance of each methods compare to using only one CPU is visualized in the Figure 1.2. The increasement of the kij method is the least in 6 method. For the other methods, the performance is almost 2.5 times higher before paralleling.

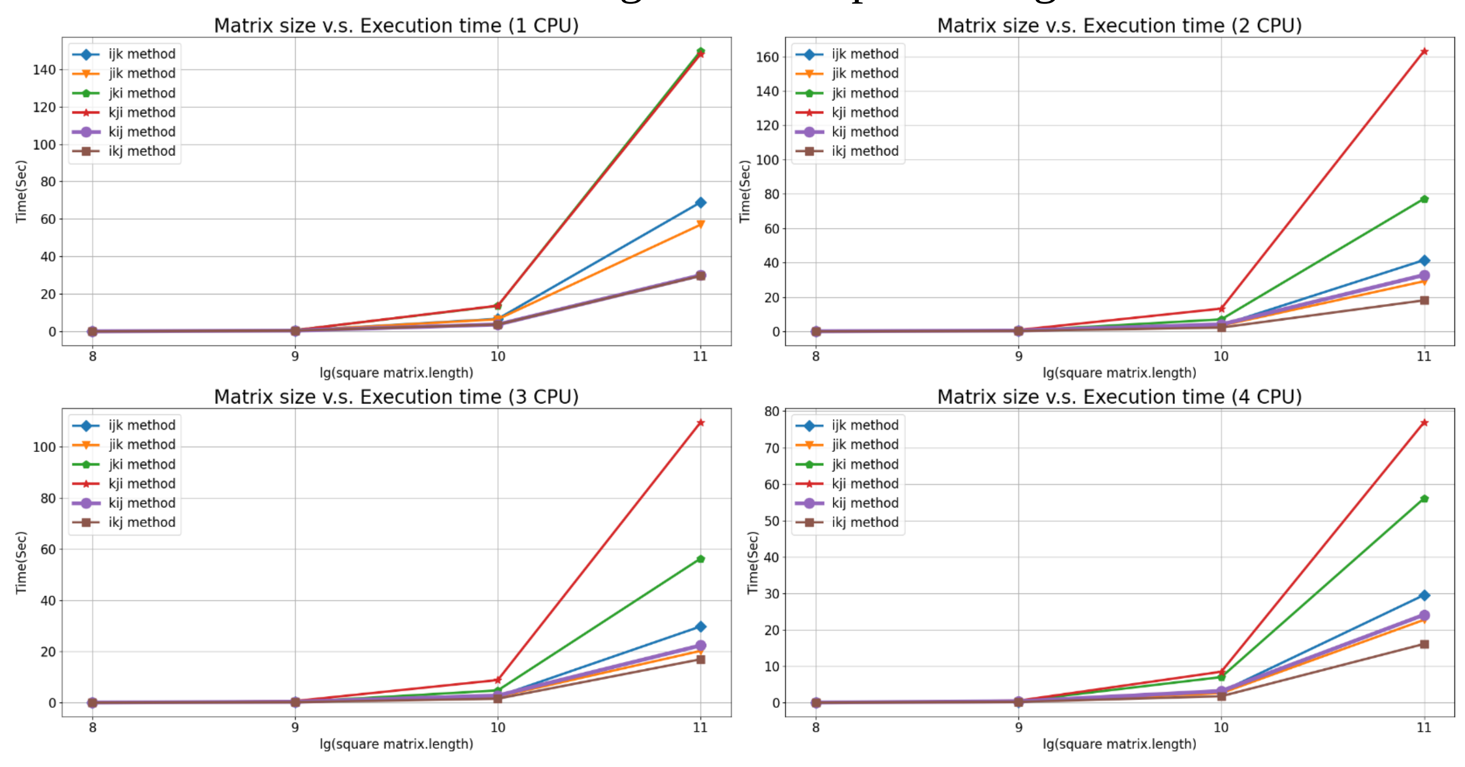


Figure 1.3

In figure 1.3 shows how matrix size affect the execution time of matrix multiplication result. As the matrix size grow, the process take more time to finish the job.

F. Improvement

The matrix multiplication can still improve by prefetching data into the cache. We can use __builtin_prefetch() to enhance the performance. To know more, please refer to my note.