ASSIGNMENT1 - PROGRAMMING WITH OPENMP REPORT

CMT106 – High Performance Computing

1.Introduction

Nowadays, parallelism has become a new trend in software and architecture design, so that we usually consider a new solution for multi-threads applications, and OpenMP is a very good solution which provides parallel application programmer with a model for parallel programming that is portable across architecture from different vendors, and it has been widely accepted as a set of Compiler Directives for multiprocessor programming of shared memory parallel systems . OpenMP supported many programming languages include C, C++, and Fortran; and OpenMP-enabled compilers include Sun Compiler, GNU Compiler, and Intel Compiler. What's more OpenMP also provides a high-level abstract description of parallel algorithms. Therefore, it is necessary to learn and research OpenMP in this era.

In this assignment, I focus on the basic use of openMP, and discuss the performance about using openMP to a picture blur algorithm with diffident threads and chunk sizes. In detail, understand how picture blur algorithm works, generating the test cases, and coding the sequential code blur.c into an OpenMP program, and do some optimizations on code to improve the performance.

2.Experiments environment

The experiments have been conducted on the school provided Linux laptop (latitude 5490). This machine has one CPU(Intel(R) Core CPU i5-8250U @ 1.6Ghz) with 4 cores. And each core has 2 threads therefore totally 8 threads. Each core is equipped with 32KB L1 data cache and 256KB L2 cache. And the operation system is Ubuntu 18.04.3 LTS Linux. The compiler is gcc version 7.4.0(Ubuntu 7.4.0-1ubuntu1~18.04.1).

3. Timing experiment

3.1 description

The OpenMP timing experiment steps are divided in to four categories. Firstly, I measured average running time of serial program of picture blurring algorithm, which is for the further calculation. Secondly, I added OpenMP code to the program to make it parallel. In this experiment, I set the number of threads to 2, 4, 8, 16, and the number of chunk size to 1, 100, 10,000, and 100,000 to observe the program running under different threads and different numbers of chunk size, and running them with the dynamic scheduling and static scheduling separately. Thirdly, I recorded their running time and combined the results of previous serial calculations to calculate their speedup under different conditions and made some graphs. Finally, I got some feature and conclusions from figures.

For calculate the ideal parallel speedup, we suppose the serial program process time is t(serial), and the same parallel program process time is t(parallel), then the speedup is t(serial)/ t(parallel)

3.2 speedup graph (the detailed speedup data can get from timingData.xlsx)

3.2.1 for dynamic scheduling

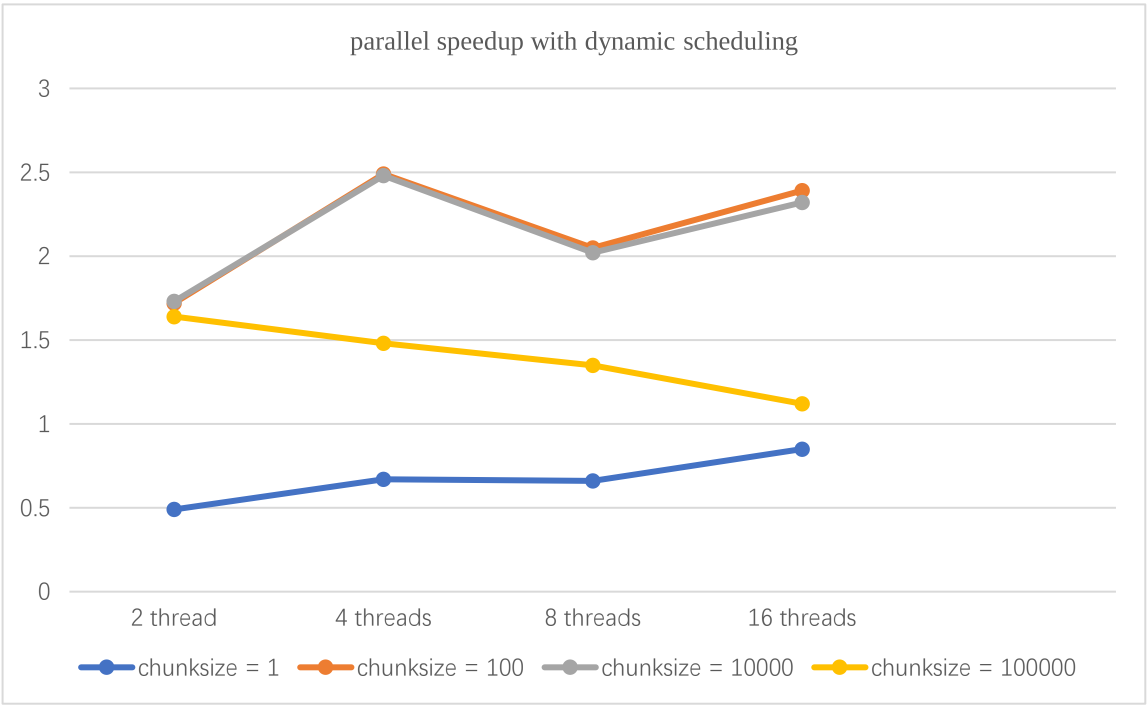

Figure 1 parallel speedup with dynamic scheduling

3.2.2 for static scheduling

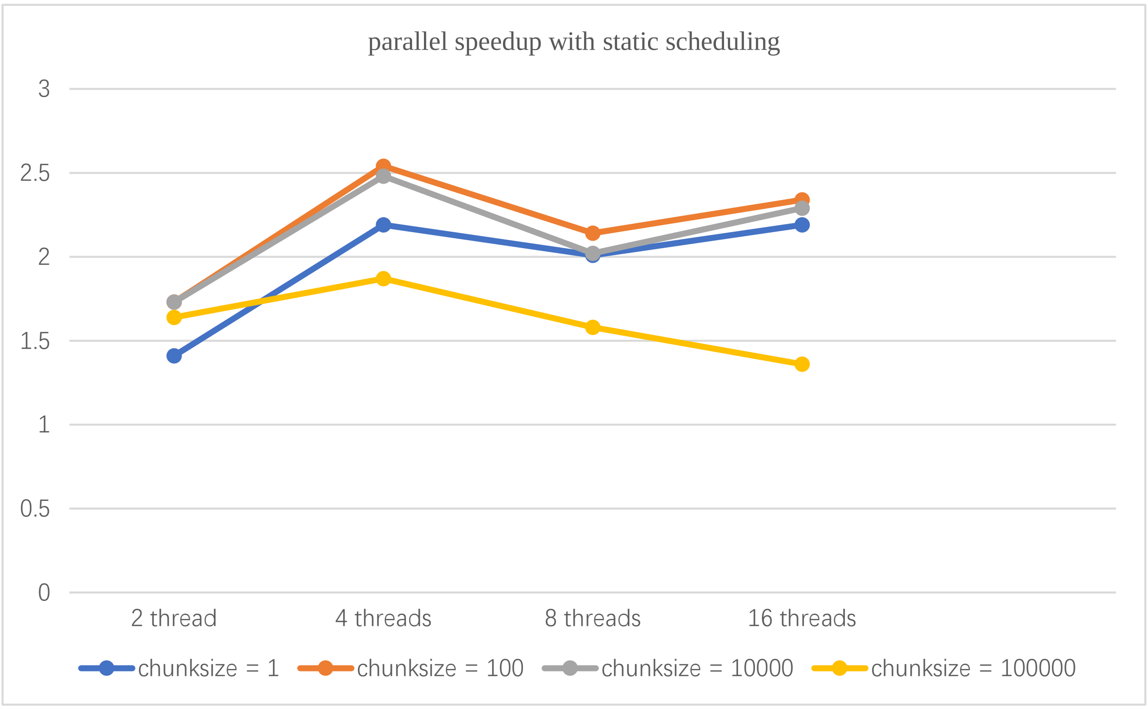

Figure 2 parallel speedup with static scheduling

According to the above figures, we can clearly see that the results can meet the speedup requirements especially when using 2 threads and 4 threads with 100 and 10000 chunk size but not totally fit the ideal parallel speedup especially when chunk size equals 1 and 100000 and when using 8 threads and 16 threads. The different happens maybe the reason that in this experiment, we just using the machine with 4 cores which couldn't give more speedup when we using more than 4 threads. On the other hand, when chunk size equals 1 or 100000, the parallel speedup always around 1. I guess the parallel code cannot give enough speedup because the schedule strategy gives more overhead. When chunk size only equals one, both scheduling algorithms will be executed many times for scheduling the tasks which must reduce the speed. And when chunk size equals 100000, then there must be a potential load imbalance between the threads which need extra time to finish some tasks and finally reduce the speed. Therefore, it's important for us to choose chunk size from a reasonable interval, such as between 100 and 10000. And doing HPC research on multi-core computers is also necessary.

4.Conclusion

This is a very good assignment which helps us totally understand the basic knowledge about OpenMP. Thorough doing the assignment, I consider that there still be a lot of points need me to master, especially the details about 2 different scheduling strategies and how to exploit parallelism to solve different kinds of problems. To this end, my next step is to try to optimize the performance of a parallel program and learn other unknown information about parallel computing for building high efficiency programs in future.