epcc

THE UNIVERSITY of EDINBURGH

Paper of CourseWork1 For

Threaded Computing

Author : Kai Liu (Kevin)

UUN : S1777496

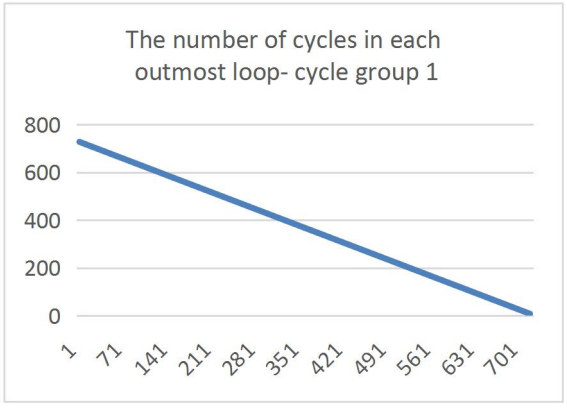
Exam No. : B117780

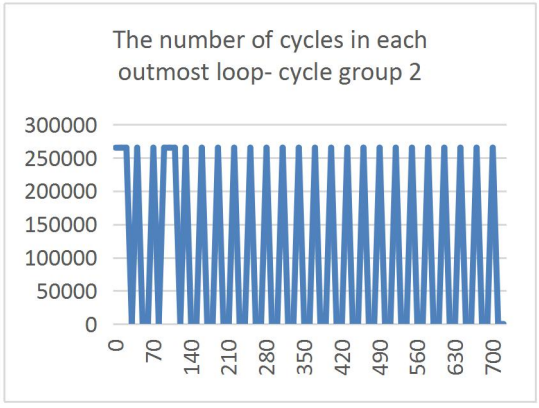
Introduction

OpenMP provide us many useful tools for tuning parallel programs. Schedule clause is an important one. It affords several options which we can choose to arrange the loops into different threads, every option represents one kind of work assignment strategy. It's obviously that no silver bullet in performance tuning, the answer of which kind of schedule is the most suitable for our threaded parallel program is highly related to the program's own characteristics. That's means we should deeply understand the characteristic of different schedule options in OpenMP firstly, then we can get the point and know how to match the schedule options to our program. This project requires us to use different schedule options in the same parallel loops and get the conclusion by analyzing the data we obtained.

Problem Analysis and Program Implementation

Two groups of cycles in a prepared serial program should be parallelized. These two groups have different characteristics. By sampling the outmost loops in each group, we get the chart below, show the difference in a clear way.





The horizontal coordinates of the charts are the serial number of the outmost loop, the vertical coordinates are the total number of the cycles in each outmost loop. It's obviously that the graph of first cycle group is a tilted straight line, means the cycles is linearly reduced when the outmost loops keep on running. The cycles numbers of the

second group are bounced between zero and a big value. Such differences will lead to the different best schedule clause in later experiments.

In this project, we take the number of threads, the type of schedule clause and the chunk size as the key factors that will affect the performance of parallel for cycle groups. Tens of combinations of the three factors should be adopted in our parallelized program and program should be executed, then the output data should be collected and analyzed. A set of shell scripts are developed to do all the work automatically. The script `./submitcreatebin.sh` copy the source code and replace with the value of the factors then compile it into a executable file in ./bin folder. Each combination of schedule clause and chunk size will generate a binary file. All these files will be executed by `./submitexecutebin.sh.` This shell script changes the numbers of threads by modifying pbs file, then submit it into the queue of CIRRUS. All the output data will be collected by cluster and the script saves these data in ./sub folder by changing the default working directory to ./sub folder. All the scripts are easy to deployment and use. All the output data are easy to be processed by Excel due to their predefined .csv format.

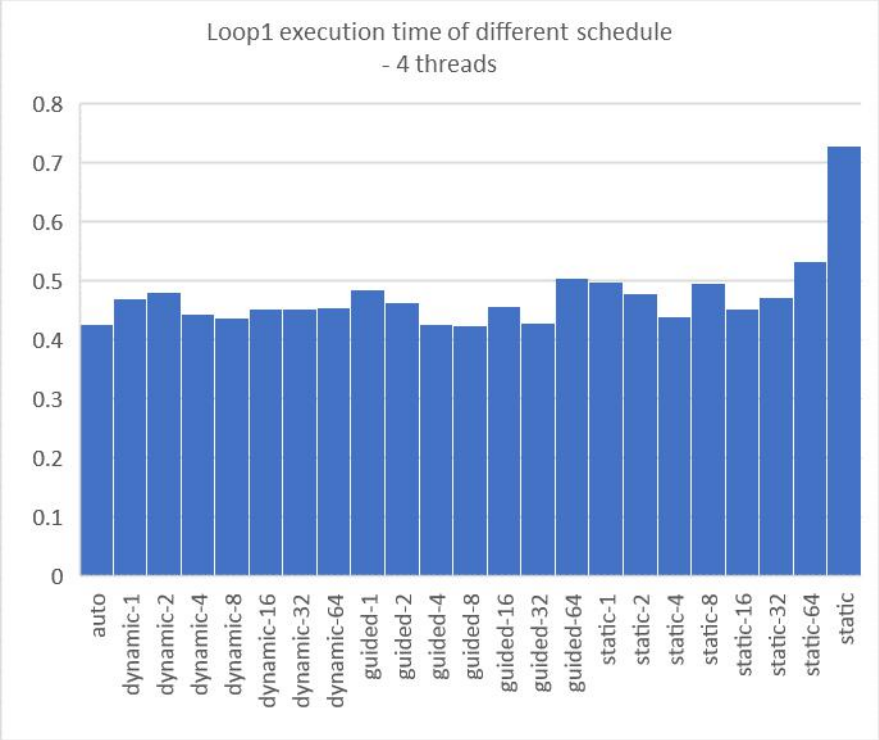
Output Performance Data Analysis

For choosing the best schedule clause for the parallel program and deepening the understanding of the work assignment strategy of schedule clauses, several statistics and comparisons of the output data was performed in this project.

Two kinds of output data will be obtained by running the parallelized program: The first kind is the total execution time of the two loops; The second kind is the distribution of each outmost loop among threads. We can analyze the best schedule strategy by statistic the execution time data and understand the principal of the strategy by the distribution data.

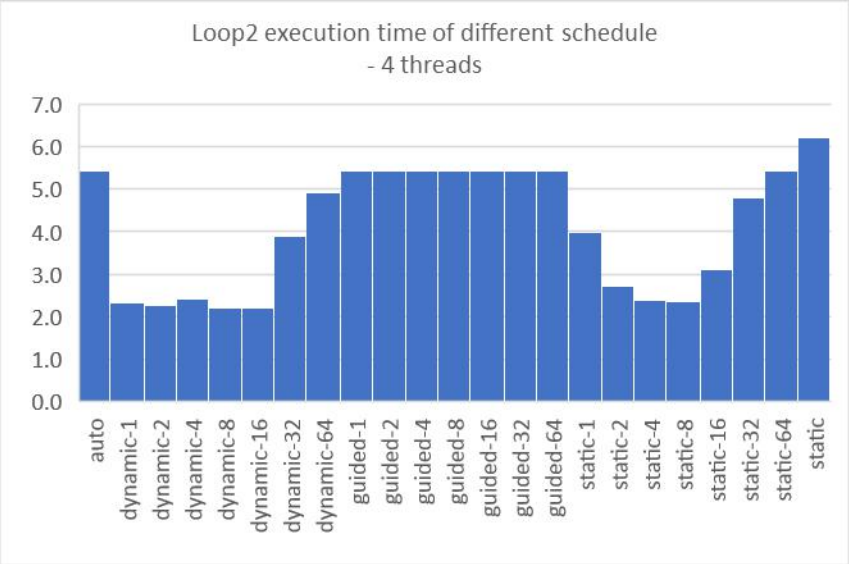
Performance Analysis for 4 Threads Option

All the combinations of schedule clause and chunk size have been set in the program and the program was executed in 4 threads. The execution time was collected and analyzed, showed as the chart below.



This chart shows the amazingly similar execution time from different schedule types **except** STATIC schedule. This phenomenon comes from the workload distribution of loops 1. As mentioned in last chapter, the number of cycles of each outmost loop in loops 1 is linearly reduced. If STATIC schedule is configured, the outmost loops will be divided into 4 parts with same number of loops but without equal workload because of the unbalanced workload in each outmost loop. The rest kinds of the schedule dynamically assign the small pieces of work to the threads, so the actual workload assigned to each thread is approximately equal. The best schedule clause of loops 1 is GUIDED with chunk size 8 when using 4 threads. But maybe the best one will change to some other schedule types because these execution time is too close.

The execution time of loops 2 is showed below.

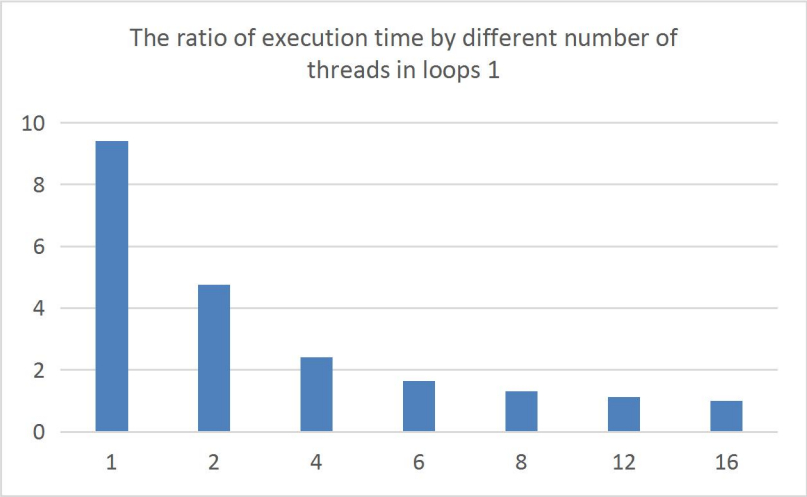


The time distribution of loops 2 is tremendously different from loops 1. The difference comes from the irregular workload assignment from each outmost loop. In this kind of circumstance, the schedule strategy and chunk size will be crucial and will have significantly impaction on the execution time. From this chart, the schedule type with best performance are DYNAMIC and STATIC with proper chunk size (size 8). DYNAMIC part is the best because assigning the loops according to the work status of the threads is truly efficient in unbalanced workload of each loop. The worst part is GUIDED, because this kind of schedule will assign the biggest number of chunks to the first thread when the program starts. Maybe the first part of assigned loops will contain most of the workloads. That's why the GUIDED is the worst no matter what the chunk size is. The STATIC schedule is very interesting. When the chunk size is appropriate, and every chunk just contain the similar number of inner cycles, every thread will get similar workload by STATIC schedule. When the inner cycles of each chunk are totally different because of the inappropriate size of the chunks, the performance will be bad.

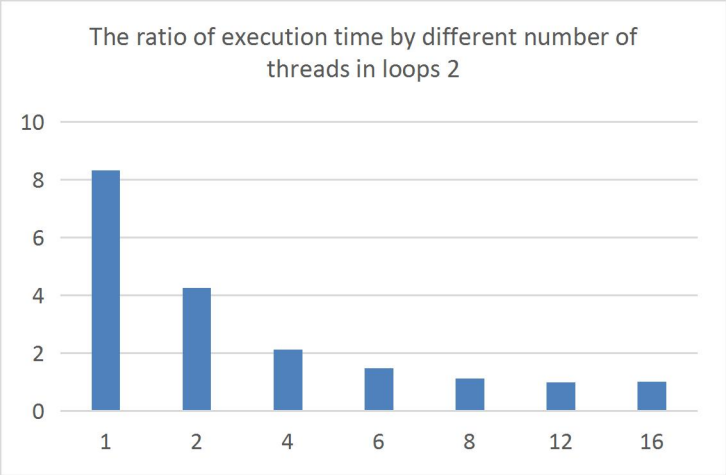
The best schedule of loops 2 is DYNAMIC with block size 8.

Speedup Analysis with Different Number of Threads

The performance will be enhanced by more available threads. In loops 1, we can see the time reduced by more threads, as showed below:



It is obviously that with the growth of the number of threads, the performance is better. When we have only 1 thread, it will take more than 9 times of execution time than 16 threads exist. More threads means greater performance. That also because of the loops we processed have liner reduced workload, we can assignment them equally to more threads.



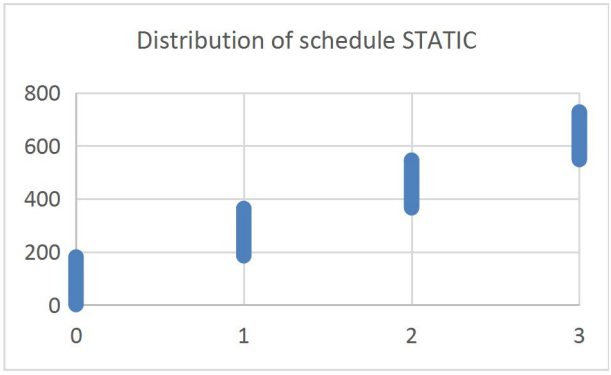
The ratio of execution time of loops 2 is similar to loops 1. All the interpretation for this chart also is the same with loops 1. So, more threads always mean better performance. But we noticed that when the thread number is 16, the performance slightly worse. That gives us a warning that with the increase of thread count, more time will be exhausted in

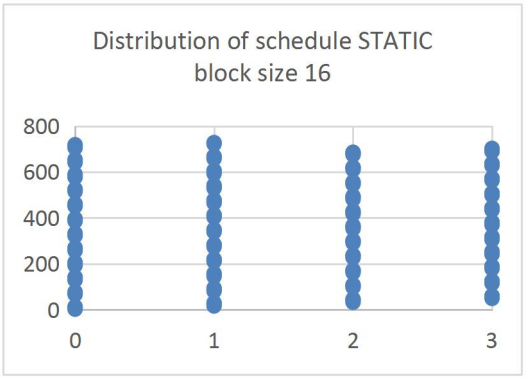
the working assignment, and the unbalanced workload assignment may occur. That remind us that the appreciate number of thread is best, not the more threads are best.

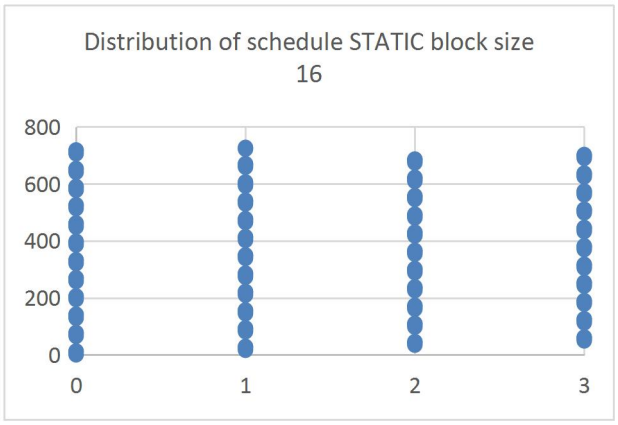
The Work Assignment Strategy Analysis

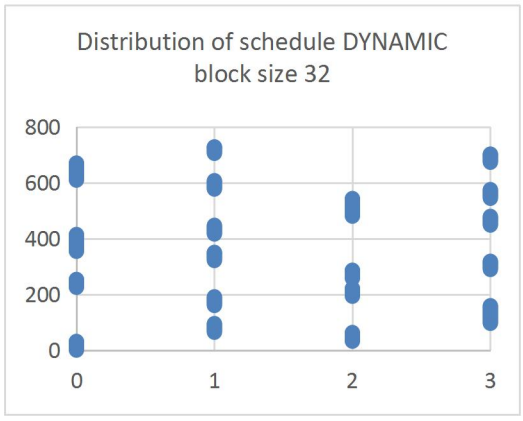
In this project, extra output data was collected to show how the different schedule types assign the loops to different threads. The parallelized program record every outmost loop's sequence number, and this loop was assigned to which thread. After analysis, we can see a clear image about the work assignment.

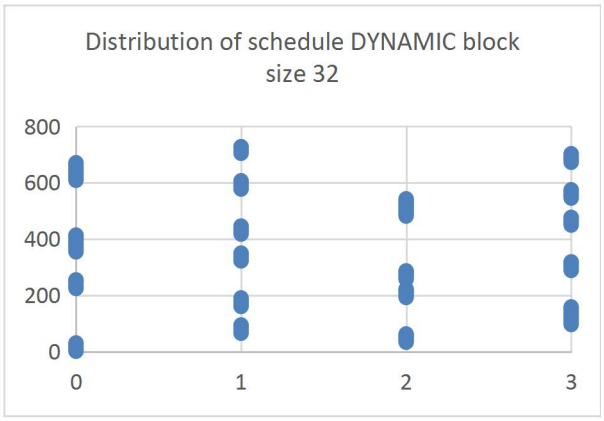
For example, I n loop 1, the work assignment of different schedule type showed below:

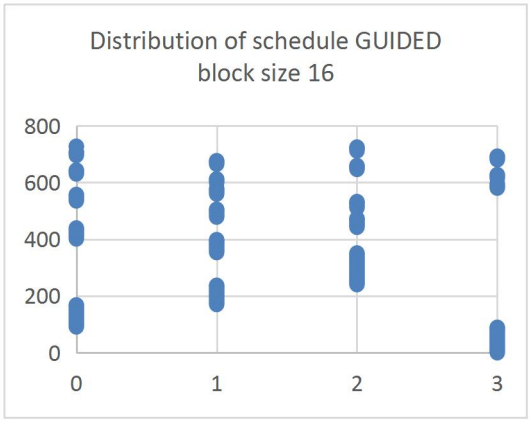


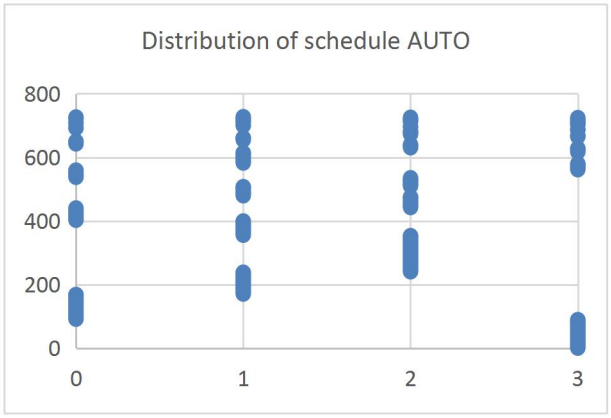












All the charts and data really do great help to understand the backside of the schedule, help us to make the interpretation for the phenomenons.

Conclusion

From this project, we can see how the 3 key factors impact the performance of threaded parallel computing. More threads always mean more performance, but if we can't assign the work equally to the threads, then more threads will take worse performance. In the other aspect, if the workload of the loops is unbalanced, we should use dynamic assignment strategy, but if it is stabled, the static is also good and simple. The proper chunk size will take better performance.