Threaded Programming Part 1 Report

S1929166 Xinyue Sheng

October 18,2019

Contents

1 Introduction

Scheduling strategy directly affects system resource utilization and parallel efficiency. To keep good load balancing and optimize the processor's performance, there are many loop schedules designed to help us assign the workload to each thread efficiently and reasonably.

In this coursework, we will take a C program including two loops as an example to record the execution time based on different loop schedules with a growing chunksize when 4 threads are allocated. Every condition will be executed 10 times and then we will calculate the average execution time to ensure the accuracy. The shorter the execution time is, the better the scheduling. Then, the program with the best scheduling options will be executed again on up to 16 threads to test how the speedup changes with the increasing number of threads. Finally, We will describe the results by making graphs and try to explain the reasons behind these results.

2 Loop1 analysis

In loop1, there are two loops in the parallel region but only the outermost loop's iterations are shared with multiple threads. Before we parallelize this algorithm, the relationship between the loop variable i and the number of inner iterations need to be considered. From Figure 1, we can see that the number of inner iterations falls steadily when the value of i increases. It reflects a top-heavy workload distribution. Therefore, the workload in each chunk must be different to others no matter what size the chunk is. By testing the program execution time under different kinds of loop schedules and selecting the shortest run time, we can find out which schedule can help achieve load balance to the greatest extent.

Figure 2 portraits the execution time of loop1 when the loop schedule is static, dynamic or guided with an increasing chunk size. Values of two schedules static-default and auto without a chunk size are set to the longitudinal axis. As we can see from the graph, the run time of the static-default schedule is extremely long. In contrast, the auto schedule helps the parallel loop reduce the run time to a lower value. As for the other three schedules, we can observe that their execution time all drops to a low value before it increases.

2.1 Static schedule

The execution time of the static schedule is much longer than that of the other two schedules. According to the static schedule, if we distribute the outermost loop equally between each chunk and allocate the chunks in order, the thread taking the front part of chunks always have a heavier workload than the thread executing the latter part. When

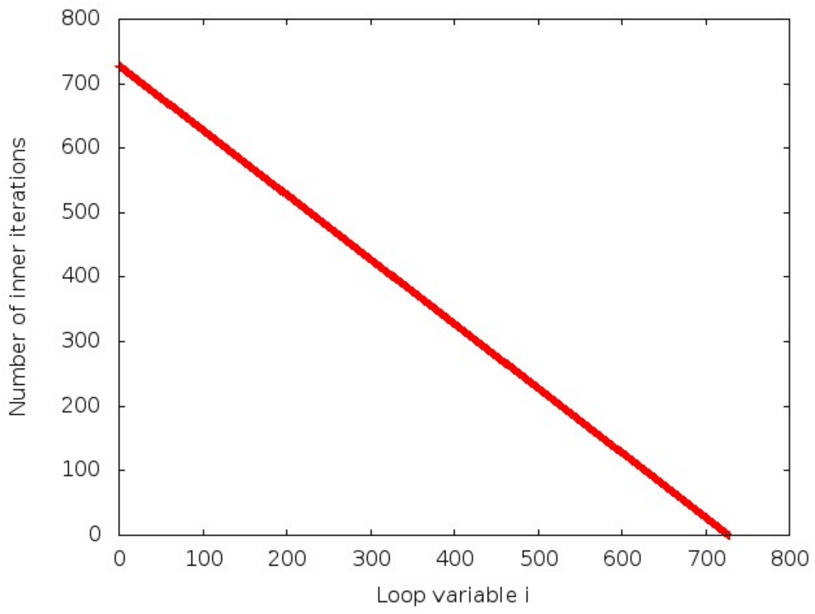

Figure 1: Number of inner iterations against i

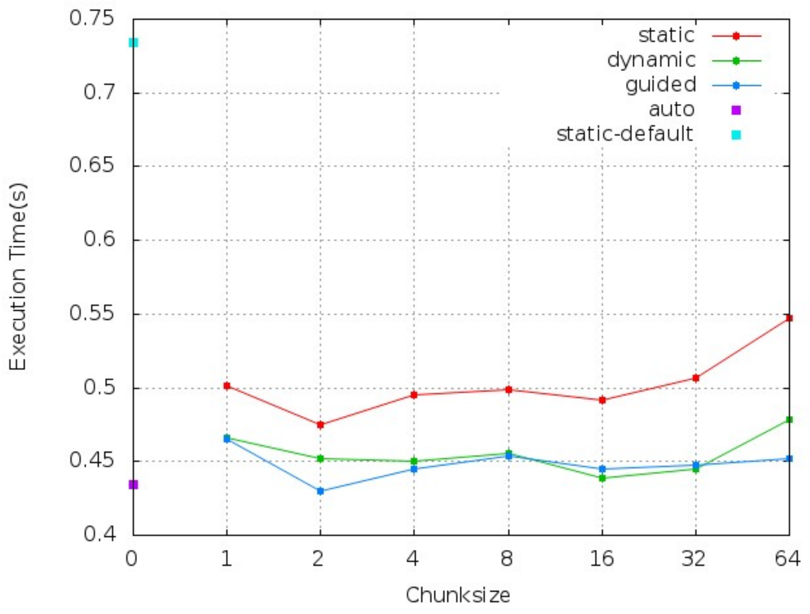

Figure 2: loop1 execution time under different schedules and chunksizes

the size of chunk is small, the difference between each thread's total workload is small, however, with the chunk size increasing, the difference is bigger.

When the chunk size is 2, the parallel loop has the shortest run time for the static schedule. In this case, the total workload and run time of each thread are roughly the same. However, with the chunk size increasing, the run time also rises. This is because the difference of threads' workload is big, and threads with light work have to wait for threads with heavy work to finish, thus lengthening the run time.

2.2 Dynamic schedule

Similar to the static schedule, the dynamic schedule divides parallel iterations into many chunks of the same size, but it allows idle threads to steal work from unallocated chunks. The advantage of this schedule is that it can alleviate and balance too long and too short execution time depended on the different workload of each chunk, but at the same time, it can also bring overheads due to the real-time allocation.

When the chunk size equals 1, the difference in workload on each thread is not obvious. However, when the chunk size increases to 16, the run time reaches the lowest value of this schedule. When the chunk size continues growing, the run time rises as well. It is because when the chunk size is too big, the workload of some threads is too large, resulting in long run time in some threads and high overheads, so the total time is lengthened.

2.3 Guided schedule

In the guided schedule for loop1, chunks start large and become smaller exponentially. It is also very flexible. When the chunk size equals 2, the execution time is the shortest among all schedules and chunk sizes. It means that in this case, there is a good load balancing between each thread and the processor's resource utilization is the highest. Whereas, when the chunk size becomes larger, the run time also increases, albeit with some ups and downs. The main reason is that as the size of the smallest chunk grows, there are more large chunks that contain too much heavy workload. When these big chunks are allocated to some threads, other threads which finish all the rest of the work have to wait for them to complete their work. Therefore, the entire run time is extended.

3 Loop2 analysis

Loop2 is also an irregular nested loop. As we can see from Figure 3, for the array jmax[N], the value which equals to N in the front part is very dense, and as i increases, the number of values which equal 1 grows gradually. The value of the array jmax[N] determines the number of inner iterations.

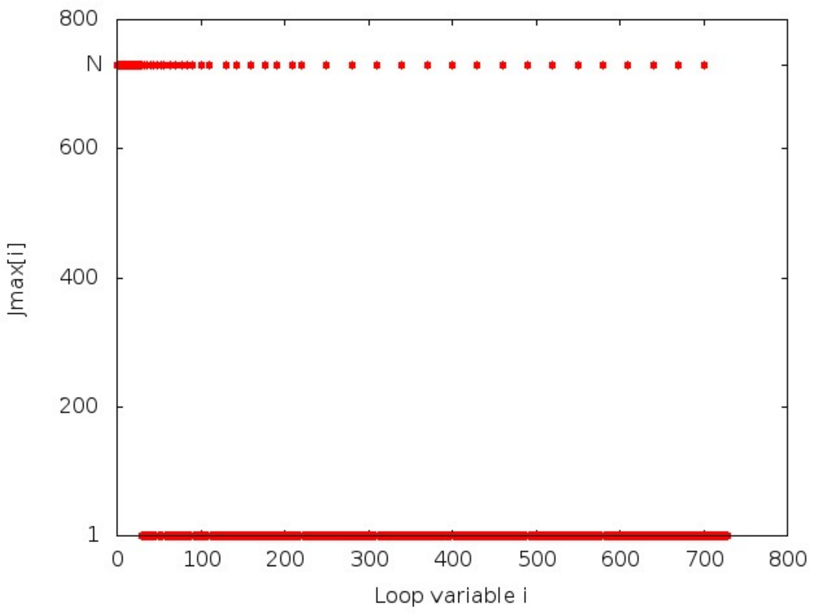

Figure 3: Array jmax[i] against i

In Figure 4, to help us find the law more clearly, the range of i is set to between 0 and 100. The number of inner iterations is the sum of two inner loops' iterations. We can see that the number of iterations jumps back and forth between a large value and 0. In the first 29 outermost iterations, the number of inner iterations keeps an extremely high value 265356, and in the latter part of the outermost loop, the number of 0 iteration corresponding to the increasing variable i goes up greatly. By counting the number of internal iterations, we can know that the front part of iterations takes up almost half of the entire workload.

Figure 5 portraits loop2's execution time under various schedules and chunk sizes. According to this graph, the parallel loop under static schedule without chunk size takes the longest run time, and the auto schedule fails to balance the load properly for this loop. Besides, the run time of guided schedule is stable and much longer than that of other schedules before the chunk size reaching 64. The static schedule and dynamic schedule show similar trends, but the dynamic schedule always collects a shorter time.

3.1 Guided schedule

As we can see from the graph, the guided schedule provides poor performance for this parallelization. The execution time for each chunk size is almost the same, which implicitly indicates that in these tests, the total run time is determined primarily by the time required by the thread executing the largest chunk in the front part of the outermost iteration. Therefore, no matter what the chunk size is, the extreme imbalanced workload allocation will hardly change and result in stable run time.

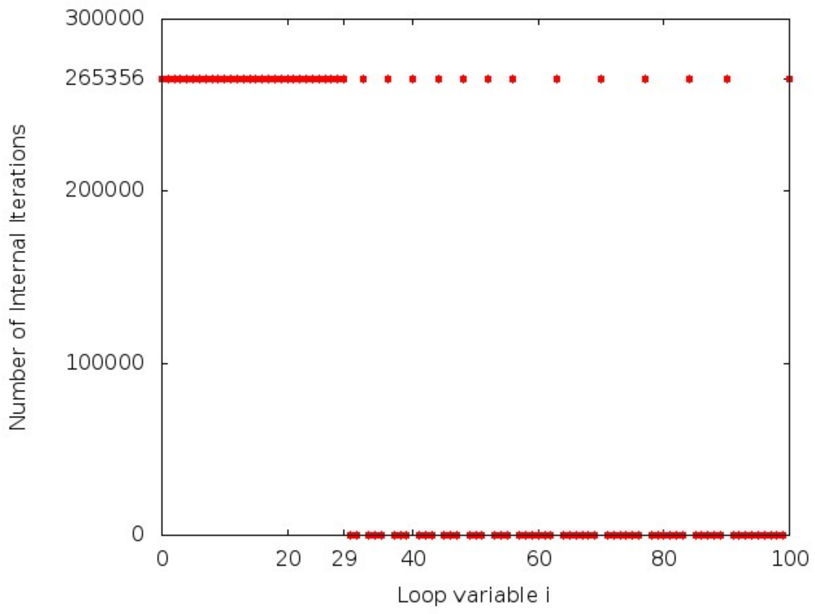

Figure 4: Number of inner iterations against i

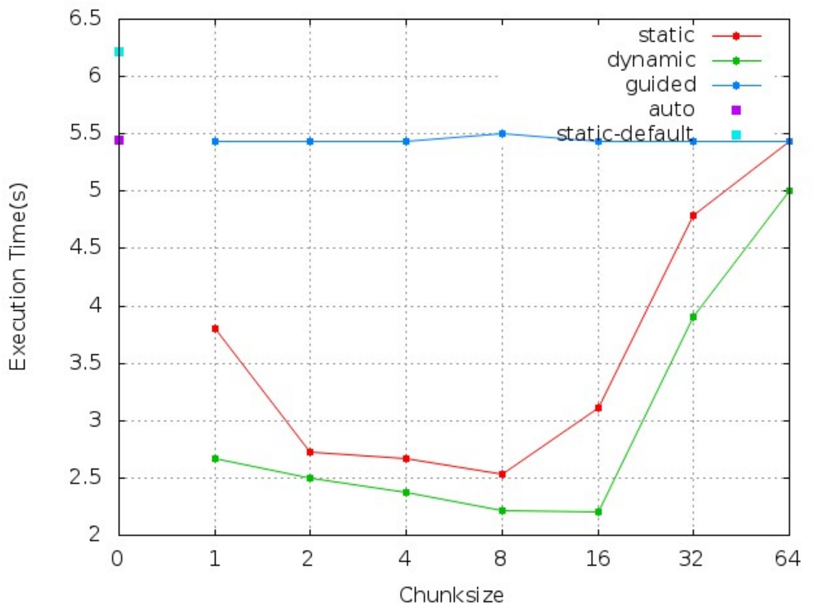

Figure 5: Loop2 execution time under different schedules and chunksizes

3.2 Static schedule

When the chunk size equals 1, each thread corresponds to one iteration for this schedule. In the first 29 iterations, each thread takes about the same amount of time, but after the 30th iteration, some threads need to execute many inner iterations, while some have no workload, so they have to wait for the busy threads completing their work before starting their new work. The unequal workload causes the waste of the processor's resources and greatly reduces the performance.

The execution time reaches the lowest value with chunk size 8 before rising dramatically. When the chunk size is set to 8, the load imbalance for each thread is alleviated to a large extent. Those front 29 outermost iterations are divided into 4 chunks and assigned to each thread almost equally, and at the same time, threads always have similar workloads, thus shrinking the total run time. As the chunk size increases, front chunks' workloads are heavier. The number of parallel iterations drops quickly, followed by the longer run time.

3.3 Dynamic schedule

The dynamic schedule is more flexible in scheduling thread than static schedule, so it is better at handling irregular parallel loops. Although it encounters similar problems when the chunk size is too small or too large, this schedule can preferentially call threads with no workload to execute the next chunk. This helps the processor to balance the load for each thread, optimize parallelism and finally reduce run time.

When the chunk size is 16, the performance of the processor is optimized, the load is relatively balanced, and the run time is the shortest among all schedules and chunk sizes.

4 Speedup analysis

In this part, loop1 and loop2 are tested with their best schedules respectively. This experiment is conducted to find out the law of speedup ratio as the number of threads grows. Figure 6 is the execution time on different number of threads for loop1 and loop2 respectively. From Figure 7 we can observe that before the number of threads is 4, the speedup ratios of loop1 and loop2 are similar. However, when the thread number continues to increase, the loop1's speedup grows steadily, while the loop2's speedup remains stable. The ideal speedup line illustrates the ideal condition when all chunks have the same workload.

For loop1, the speedup trend is close to the ideal trend, although with the increase of threads, the acceleration is lower than the ideal situation because of its top-heavy workload allocation and overheads generated by real-time allocation. When the thread number grows, chunks with different sizes of the workload are simultaneously assigned to

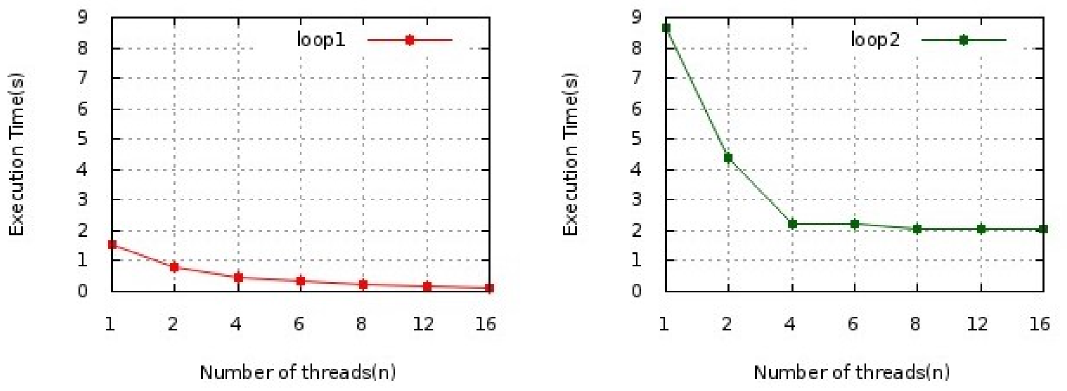

Figure 6: The execution time on different number of threads for loop1 and loop2

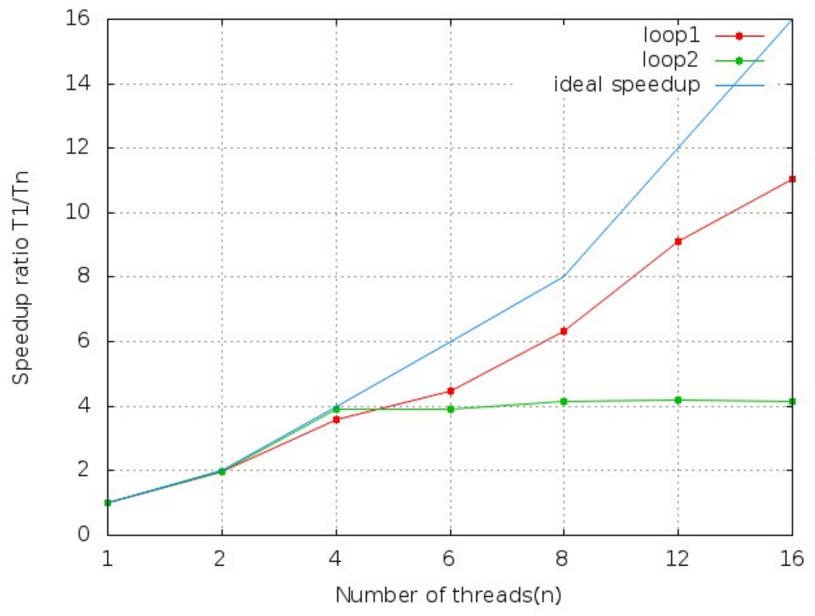

Figure 7: The speedup changes differently on a growing thread number in loop1 and loop2

more threads to execute, thus greatly reducing the run time and increasing the speedup ratio steadily.

For loop2, the speedup ratio reaches 4 when it has 4 threads before remaining this value with the increasing number of threads. This is because the size of the chunks does not change, and there is a limited number of chunks involving heavy workload. The execution time for each chunk cannot be reduced with the change of the number of threads since the iterations in the chunk will be executed sequentially. Therefore, if the number of threads is greater than 4, there will always be some idle threads waiting for other threads to complete their work. As a result, the total run time and the speedup ratio will stall at a certain value and are no longer changed.

5 Conclusions

From the experiment results, it is proved that different kinds of loops may have their optimistic loop scheduling and threads. When the number of threads is 4, loop1 and loop2 using guided,2 and dynamic,16 can obtain the shortest execution time respectively.

When facing a parallel loop program, we need to analyze and do experiments regarding the internal construction of the loop firstly before setting the loop schedule. A good schedule can help the processor to achieve load balance and improve the efficiency of execution.