Threaded Programming Coursework 2

B177252

October 2020

1 Introduction

This paper discusses five different loop scheduling options in OpenMP by performing an experiment on a piece of code which contains two loops parallelised with OpenMP directives. The goal of this experiment is to understand how different schedules work and whether using more threads brings more efficiency by observing and explaining the experimental results.

2 Experimental environment and methods

The experimental environment and methods used in this paper are listed below.

2.1 Experimental environment

- Cirrus
- Intel-Compilers-19
- Linux 4.18.0-147.8.1.el8_1.x86_64
- Bash 4.4
- OpenMP 4.5
- Python 3.8.3
- Matplotlib 3.2.2

2.2 Methods

The main methods used in this paper is control variates method. The variates in this experiment include loops,number of threads, schedule options and chunksize. The first part of this experiment is to study how well different schedules perform on the two loops separately, this paper initiates the number of threads to be a constant and then experiments on different schedules parallelised on loop1 and loop2 respectively. For the latter three schedules, another variate which is chunksize will also be controlled in order to study how different chunksizes affect the execution time. The second part of this experiment is to study the speedup versus number of threads for each loop, this paper controls the schedule to be the optimal option selected in the first part and then change the number of threads to get the experimental results.

The experiment is performed on compute nodes for doing accurate timing. We need to write a SLURM script and use batch processing to access compute node to run the code rather than run it on public nodes. For the first part, this paper initiates the number of threads to be 8 and change the chunksize among 1,2,4,8,16,32,64 for each loop. In order to make it more convenient, this paper replaces the schedule part

in the file loops.c to be "schedule(runtime)". To switch between different schedule options, for example, static,1, we can simply set the environment using "export OMP_SCHEDULE='static,1'" in the SLURM script. For the second part, we set the schedule option to be the optimal one selected in the first part and change number of threads among 1,2,4,6,8,12,16,24,32 by using "export OMP_NUM_THREADS=n" (n is the number of threads) in the SLURM script. We record the execution time for 10 times and calculate the average execution time as the final result in order to avoid occasionality.

After getting all the experimental results, this paper uses python.matplotlib to plot the graphs.

3 Experimental results

As discussed before, this experiment is divided into 2 parts. The tables and graphs of experimental results in two parts are shown in the sections below.

3.1 Static and Auto

The reason why we separate these two options is that the latter three options all need to specify chunksize. The results are as follows.

The table 1 is the record of execution time versus chunksize for loop1

Execution time versus static and Auto Schedule)		
Schedule	loop1	loop2
static	1.81	14.22
auto	0.99	12.36

Table 1: Execution time versus static and auto schedule

3.2 Execution time versus chunksize

The table 2 is the record of execution time versus chunksize for loop1

Execution time(s) versus chunksize(loop1)							
Schedule	1	2	4	8	16	32	64
static	0.99	0.99	1.00	1.00	1.04	1.10	1.23
dynamic	0.99	1.00	0.99	0.97	0.99	0.99	1.06
guided	0.98	0.98	0.98	0.98	0.98	0.98	1.01

Table 2: Execution time versus chunksize for loop1

The table 3 is the record of execution time versus chunksize for loop2

Execution time(s) versus chunksize(loop2)							
Schedule	1	2	4	8	16	32	64
static	5.74	4.54	4.39	3.71	4.41	6.36	11.41
dynamic	4.08	3.67	2.97	2.65	2.92	5.65	10.73
guided	12.35	12.35	12.34	12.34	12.35	12.34	12.33

Table 3: Execution time versus chunksize for loop2

The graph 1 is the record of execution time versus chunksize for loop1

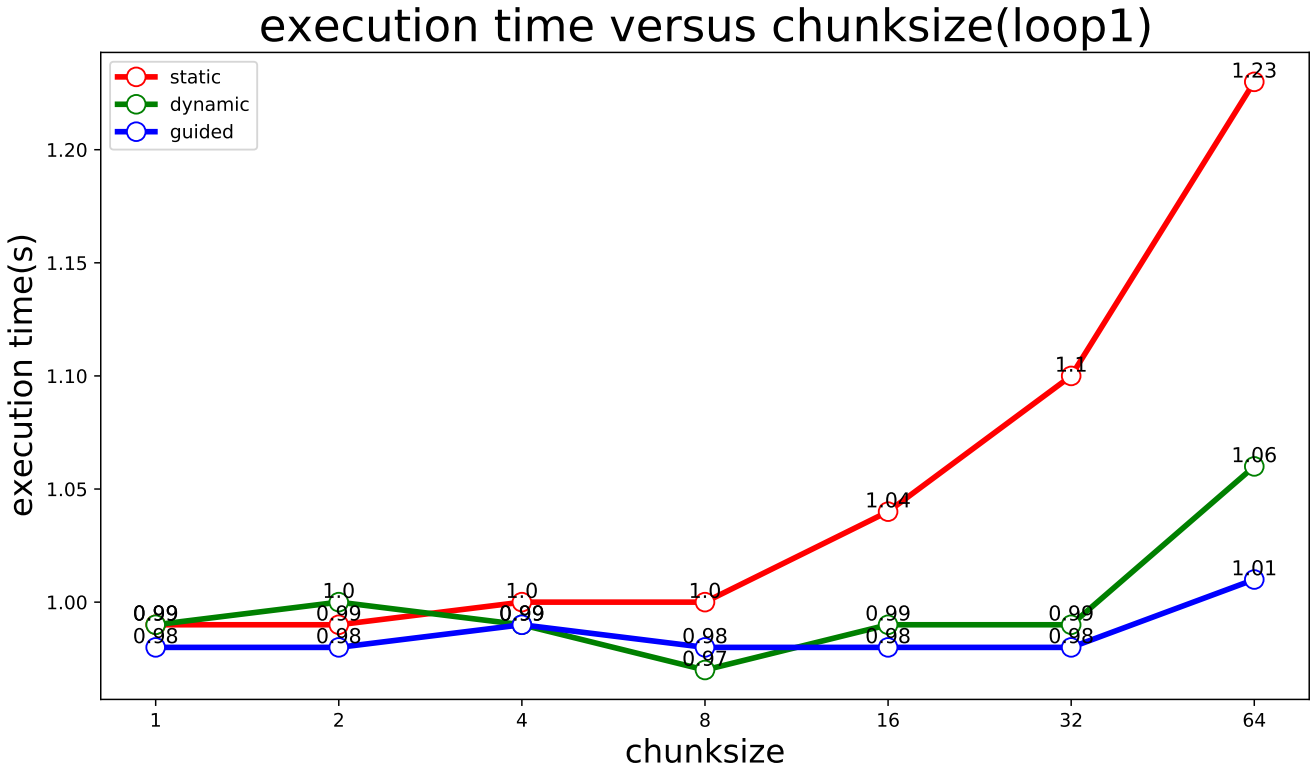

Figure 1: Execution time versus chunksize for loop1

The graph 2 is the record of execution time versus chunksize for loop2

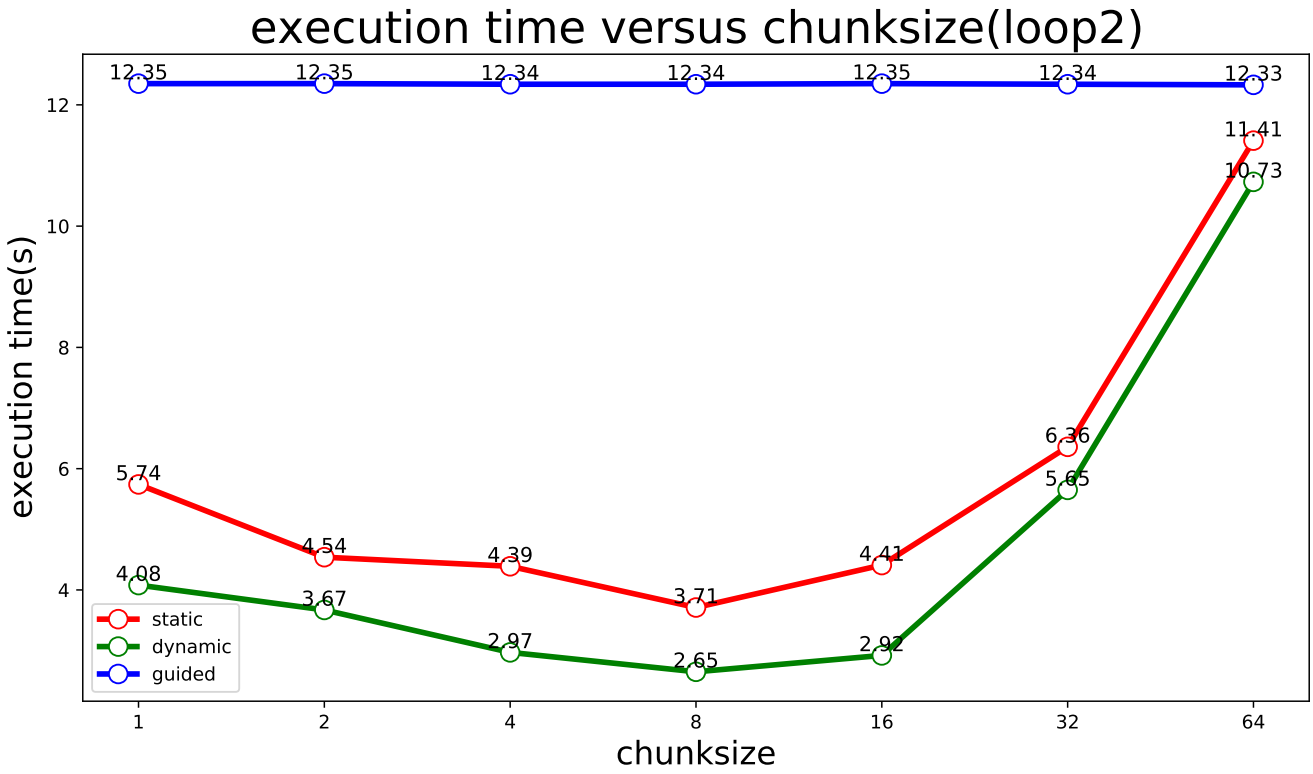

Figure 2: Execution time versus chunksize for loop2

3.3 Speedup versus number of threads

Based on the graphs above, for loop2, it's obvious that (dynamic,8) is the optimal option. However, for loop1, it's not easy to find the optimal option but we can scale down the range of possible optimal options, which are (guided,1), (dynamic,8),(auto) and (guided ,16). We find that (dynamic,8) is slightly better than the others, so we select (dynamic,8) to be the option for both loop1 and loop2. Therefore, in this part, we will control the schedule option to be (dynamic,8) and change the number of threads.

The table 4 is the record of speedup versus number of threads for loop1

Speedup versus number of threads(loop1)									
	1	2	4	6	8	12	16	24	32
speedup	1.00	1.97	3.94	5.89	7.64	11.70	15.44	17.55	17.95

Table 4: Speedup versus number of threads for loop1

The table 5 is the record of speedup versus number of threads for loop2

Speedup versus number of threads(loop2)									
	1	2	4	6	8	12	16	24	32
speedup	1.00	2.00	3.99	5.97	7.88	11.78	14.04	8.65	8.12

Table 5: Speedup versus number of threads for loop2

The graph 3 is the record of Speedup versus number of threads for loop1

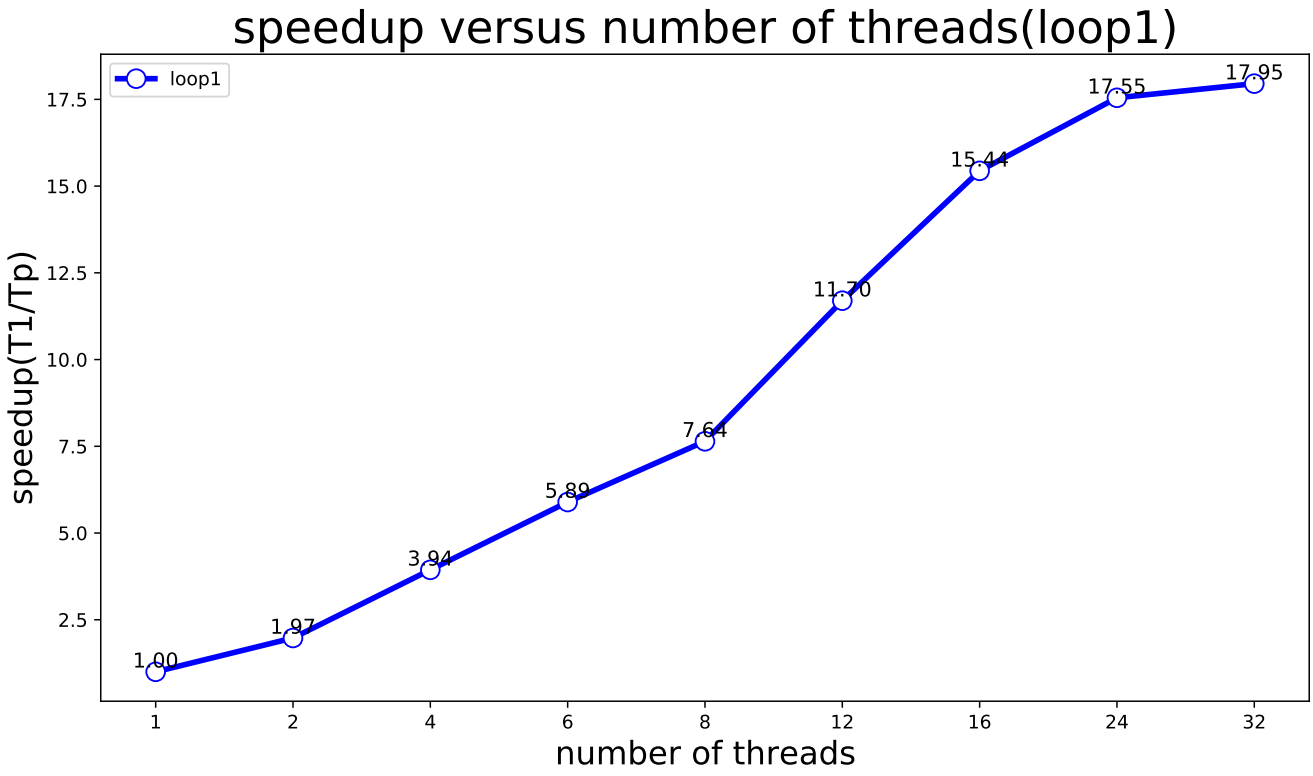

Figure 3: Speedup versus number of threads for loop1

The graph 4 is the record of Speedup versus number of threads for loop2

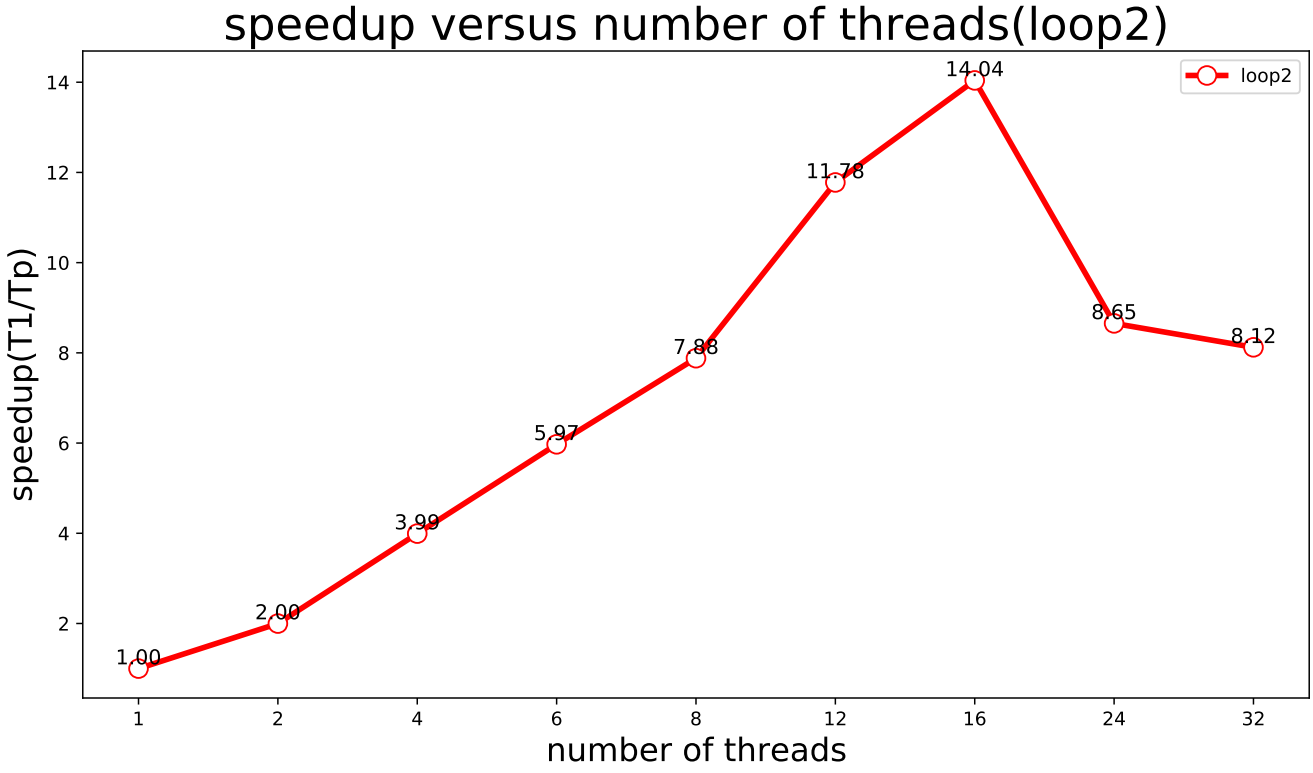

Figure 4: Speedup versus number of threads for loop2

4 Description and explanation of results

4.1 Description

According to the data above, we can see that for loop1, static schedule is relatively worse than the other two especially when chunksize is above 8. The performance of dynamic schedule and guided schedule are basically the same, which makes it difficult to find the best schedule. We finally choose (dynamic,8) to be the optimal option which is approximately 0.97 seconds. For loop2, it's obvious that guided schedule is the worst option with basically the same performance among all the chunksizes. It's also clear that dynamic schedule is better than the other two and the optimal option is (dynamic,8), which is 2.65 seconds. We can also see from the graph that when the chunksize is bigger than 8, the execution time increases dramatically in static schedule and dynamic schedule in loop2. In contrast, guided schedule is more stable no matter how chunksize changes. When it comes to the section Speedup versus number of threads, it turns out that for loop1, the speedup increase as number of threads increase, but for loop2, the speedup increase to the peak when number of threads is 16 and decreases afterwards. Finally, as for static schedule with no chunksize, the performance is bad both on loop1 and loop2. Auto schedule performs well on loop1 but performs badly on loop2.

4.2 Explanation

Here is my explanation for the results. In terms of loop1, we can see from the code of loop1 in figure 5 that this loop is not load-balanced because when i gets larger, there will be less iterations in the inner loop, which means that if we use static schedule, the first thread in the first chunk will undertake the most job and the thread in the last chunk will do the least job, which is load-imbalanced. As a result, the execution time depends heavily on the thread in the first chunk, which makes static schedule a relatively low-efficient option. While for the other two schedules, both of them achieve good performance. As for dynamic schedule, the mechanism is dividing the iterations into chunks of size chunksize and assign them to threads in a first-come-first-served order. If a thread finishes a chunk, it's assigned to the next chunk, which is very flexible. Therefore, dynamic schedule is potentially an efficient option. For guided schedule, it's similar to dynamic schedule. The difference is that the chunks start off large and decrease on an exponential basis, what's more, in guided schedule, the chunksize means the minimum size of the chunks. It works fine because it shares the similar mechanism with dynamic schedule. Although guided schedule might be overhead if the first iteration is extremely expensive, in this case where the load-imbalance is not very large, it can work fine. As for static without chunksize, the iterations are divided into equal chunks and each chunk is assigned to a single thread. As this loop is not load-balanced, the first few chunks take too much workload, therefore, the performance is bad. As to auto schedule, it's free to choose its own assignment of iterations to threads and has good load-balance and low overheads, so the performance is fine.

In terms of loop2, let's review the code in figure 6, we can see that the elements in array jmax are either 1 or N/2(N=1729), which means that this loop is extremely load-imbalanced. If we use guided schedule, the chunk starts very large and then decreases exponentially. In this way, it's easy to get overhead if the first few chunks are extremely expensive. In order to find out why guided schedule in this case is so low-efficient, it's critical that we should know how the values in the array jmax distribute. Therefore, we could print the first 50 values of jmax and it turns out that each of them equals N/2(864) which means that the first 50 iterations are extremely expensive. Therefore, for guided scheduile,the first few chunks will be very expensive and it's easy to get overhead and achieve a terrible performance. However, it's more stable than the other two schedules as chunksize increases because the chunksize in guided schedule specifies the minimum chunksize and its performance is largely dependent on the largest chunk. As to static schedule, it works fine when chunksize is between 1 to 8 because each chunk has the equal chunkszie, which is less load-imbalanced than guided schedule. However, its performance gets increasingly terrible

when chunksize gets larger than 8 because bigger chunksize means that each thread will take much more work. As for dynamic schedule, it works best because it assigns the chunks in a first-come-first-served order and chunksize is equal among all chunks, which is more flexible than static and guided schedule, thus achieving the optimal performance. The reason why it performs badly as chunksize becomes larger than 8 is basically the same as static schedule. As for static schedule without chunksize, the reason is the same as that in loop1. As for auto schedule, the compiler and runtime system take control of the scheduling decision, the reason why it performs bad is probably that the loop is extremely load-imbalanced and an the compiler or runtime system picks a terrible scheduling decison.

Then we come to the speedup versus number of threads. Please note that the schedule option is (dynamic,8) right now. For loop1, the speedup keeps increasing because 32 is not the turning point, which means that for number of threads smaller than 32, the speedup will increase because in this case, more threads could collaborate more efficiently, however, when the number of threads come to a turning point, the speedup will start to decrease, that's because there is not so much work to do, using more threads does not guarantee more efficiency because more threads consume more resources which are limited such as CPU and memory. For loop2, 16 is roughly the turning point, if number of threads get larger, the efficiency will go down.

```
void loop1(void) {
  int i,j;

#pragma omp parallel for default(none) shared(a,b) private(i,j) schedule(runtime)
  for (i=0; i<N; i++){
    for (j=N-1; j>i; j--){
      a[i][j] += cos(b[i][j]);
    }
  }

}
```

Figure 5: Code of loop1

```
void loop2(void) {
  int i,j,k;
  double rN2;

  rN2 = 1.0 / (double) (N*N);

#pragma omp parallel for default(none) shared(b,c,jmax,rN2) private(i,j,k) schedule(runtime)
  for (i=0; i<N; i++){
    for (j=0; j < jmax[i]; j++){
      for (k=0; k<j; k++){
        c[i] += (k+1) * log (b[i][j]) * rN2;
      }
    }
  }

}
```

Figure 6: Code of loop2

5 Conclusion

In conclusion, for static schedule without chunksize, it performs best on load-balanced loops but sometimes terribly on load-imbalanced loops. For static schedule with chunksize, it performs well on load-balanced or mild load-imbalanced loops but perform badly on extremely load-imbalanced loops as the chunksize gets larger. For dynamic schedule, it's more flexible and proves to be useful on load-imbalanced loops but can also performs badly when chunksize is too large. So both static schedule and dynamic schedule depend largely on chunksize in terms of performance. For guided schedule, it's similar to dynamic schedule but if the first few chunks are too expensive, it could cause overhead and perform terribly. However, its performance is slightly affected by chunksize, which makes it a relatively stable option. Lastly, using more threads does not guarantee good performance. More threads more resources, which could lower the efficiency.