EGRE 531 Multicore and Multithread Programming

Laboratory Number 3

Date: 03/02/18

Luis Barquero

PLEDGE: ______Luis Barquero______________

"On my honor, I have neither given nor received unauthorized aid on this assignment"

Virginia Commonwealth University

<u>Introduction:</u>

The purpose of this lab was to take three example codes and parallelize them using #pragma omp directives depending on what works best for each scenario. As a result, the lab was split into three parts: a Bubblesort program, a Monte Carlo program, and a Two-Works program. For each part, a non-parallelizable execution of the code was performed where the total execution time was calculated, and then the #pragma omp directives were used to properly parallelize the code. The execution time was then recorded for each parallelizable part and compared to each other and the non-parallelizable part.

<u>Lab Content:</u>

Part 1: BubbleSort

The first code provided was a Bubblesort code that would take 500 elements of an array and sort them. It was found that to properly parallelize, only three out of the four loops were able to be used; the outer loop could not be parallelized. Out of the three loops, inner loop worked best when placed under the #pragma omp critical directive, and the next two loops were then modified between static, dynamic, and guided parallelization.

Part 2: Monte Carlo

The second code provided was the Monte Carlo code that would calculate Pi over 10,000,000 iterations.

To properly parallelize, the #pragma omp reduction directive was used before the for loop that would perform the Pi calculations. Since the loop also relied on the variables x and y, a private(x,y) directive was conjoined with the reduction clause. Overall, the directive was as follow:

```
#pragma omp parallel for private(x,y) reduction(+:count)
```

Finally, the value of max iterations was modified to prove that the parallelization did work, and it did speed up the execution time.

Part 3: Two-Works

The third and final code provided was the Two-Works code, which contained two functions labeled smallwork() and bigwork(). Two parallelization methods were used: the first one using the static and dynamic directives before the for loops, and the second using the section directive. For the section directive, the first loop went from 0 < 49, while the second section went from 50 to 100, since the original loop went from 0 to 100.

<u>Test Results:</u>

Part 1:

Overall, there were 7 runs done for Part 1, 1 for non-parallelization and 6 for parallelization.

For the parallelization runs, the following six cases were considered:

A. Static, Dynamic, Critical
B. Dynamic, Static, Critical
C. Static, Guided, Critical
D. Dynamic, Guided, Critical
E. Guided, Static, Critical
F. Guided, Dynamic, Critical

The following is the table with all seven cases, their times, and their respective average times.

Not Parallelizable

	Trials	Time(NP1)
	1	7.746
	2	7.757
	3	7.775
	4	7.698
	5	7.816
	6	7.703
	7	7.921
	8	7.833
	9	7.757
	10	7.71
Average		7.7716

Parallelizable

Static, Dynamic, Critical

Trials	Time(SDC4)
1	8.836
2	8.954
3	8.941
4	8.898
5	8.911
6	8.874
7	9.003
8	8.897
9	8.927
10	8.923
Average	8.9164

Dynamic, Static, Critical

Trials	Time(DSC4)
1	9.264
2	9.355
3	9.407
4	9.301
5	9.247
6	9.246
7	9.245
8	9.253
9	9.316
10	9.41
Average	9.3044

Static, Guided, Critical

Trials	Time(SGC4)
1	8.439
2	8.547
3	8.642
4	8.609
5	8.518
6	8.517
7	8.503
8	8.53
9	8.565
10	8.507
Average	8.5377

Dynamic, Guided, Critical Trials	Time(DGC4)	Guided, Static, Critical Trials	Time(GSC4)	Guided, Dynamic, Critical Trials	Time(GDC4)
1	8.996	1	8.987	1	9.057
2	9.215	2	9.061	2	9.153
3	9.377	3	9.056	3	9.179
4	8.965	4	9.008	4	9.182
5	9.186	5	9.038	5	9.173
6	9.147	6	8.979	6	9.124
7	9.054	7	9.001	7	9.117
8	9.264	8	8.989	8	9.143
9	9.147	9	9.124	9	9.14
10	9.103	10	9.027	10	9.128
Average	9.1454	Average	9.027	Average	9.1396

Table 1 – Table shows the data acquired for the non-parallelizable and parallelizable parts for Part 1.

Unfortunately, it appears that the no matter the combination, the execution time did not improve.

Part 2:

Overall, there were six runs for Part 2: a non-parallelization run, and five parallelization runs: reduction using two threads, reduction using three threads, reduction using eight threads, and two for changing the value of the variable max to check the accuracy of pi. It was found that as max decreased, the time increased, but the accuracy also increased. On the other hand, if max is reduced, the overall execution time is less, but the accuracy will also be less. The following is the table consisting of both non-parallelization and parallelization times with their corresponding pi values.

Non-parallelized

Trial	Time(NP1)	Pi
1	1.525	3.1415962
2	1.52	3.141449
3	1.509	3.141643
4	1.514	3.1417
5	1.577	3.141558
6	1.513	3.141346
7	1.502	3.141546
8	1.537	3.14077
9	1.516	3.141718
10	1.495	3.141542
Average	1.5208	

Parallelized

Trial	Time(R2)	Pi	Trial	Time(R4)	Pi	Trial	Time(R8)	
1	0.85	3.14878	1	0.634	3.140216	1	0.552	3.140464
2	0.842	3.141707	2	0.65	3.140746	2	0.536	3.139812
3	0.869	3.14075	3	0.646	3.14008	3	0.542	3.140292
4	0.911	3.140919	4	0.642	3.140053	4	0.538	3.1405
5	0.831	3.142403	5	0.639	3.14058	5	0.544	3.140644
6	0.858	3.141015	6	0.657	3.140118	6	0.539	3.140433
7	0.891	3.142049	7	0.673	3.140184	7	0.54	3.140633
8	0.883	3.141597	8	0.659	3.140424	8	0.538	3.14046
9	0.897	3.140988	9	0.644	3.140341	9	0.541	3.140187
10	0.838	3.140974	10	0.656	3.140126	10	0.56	3.1406
Average	0.867		Average	0.65			0.543	

Table 2 – Table 2 shows the data acquired Part 2.

Using Table 2, it is evident that the execution does improve. As mentioned above, the accuracy of pi increases as max increases, but that also increases the overall execution time.

Part 3:

For Part 3, there were 13 overall runs of the program: one for the non-parallelization part, and twelve for the parallelization part. For the parallelization part, there were two main modifications (Static, Dynamic; Section, Section) made with four runs for each modification, where the number of threads increases from 2, 3, 4, and 8. The final modification is for the inappropriate modification where the program isn't actually parallelized even though there are #pragma directives. For this lab, the #pragma directive used was the #pragma critical directive used for both for loops. Table 3 shows the data for all thirteen runs and their corresponding.

Table 3 – Table 3 shows the data acquired for Part 3 of the lab.

Non-Parellelizable Trial		Time(NP1)
	1	17.158
	2	17.571
	3	17.357
	4	17.047
	5	17.046
	6	17.174
	7	17.057
	8	17.119
	9	16.988
	10	17.056
Average		17.1573

Parallelizable Trial	Static, Dynamic(2) Time(PSD2)	Static, Dynamic(3) Time(PSD3)	Static, Dynamic(4) Time(PSD4)	Static, Dynamic(8) Time(PSD8)
1	13.369	9.807	7.939	6.287
2	13.718	9.644	7.836	6.06
3	13.352	9.735	7.941	6.124
4	13.261	9.716	7.956	6.038
5	13.299	9.632	8.644	6.096
6	13.52	9.567	7.962	6.05
7	13.354	9.669	7.917	6.086
8	13.395	9.693	8.044	6.121
9	13.346	9.663	7.926	6.253
10	13.303	9.608	7.979	6.054
Average	13.3917	9.6734	8.0144	6.1169

Parallelizable Trial	Section, Section(2) Time(PSECSEC2)	Section, Section(3) Time(PSECSEC3)	Section, Section(4) Time(PSECSEC4)	Section, Section(8) Time(PSECSEC8)
1	4.271	4.271	4.307	4.331
2	4.268	4.28	4.276	4.282
3	4.283	4.265	4.264	4.264
4	4.279	4.354	4.288	4.27
5	4.287	4.284	4.272	4.284
6	4.27	4.275	4.271	4.26
7	4.266	4.274	4.272	4.259
8	4.265	4.351	4.269	4.271
9	4.256	4.273	4.262	4.371
10	4.471	4.273	4.265	4.277
Average	4.2916	4.29	4.2746	4.2869

Parallelizable Trial	Critical (2) Time(PCC2)	Critical (3) Time(PCC3)	Critical (4) Time(PCC4)	Critical (8) Time(PCC8)
1	17.123	17.143	17.221	17.319
2	17.265	17.16	17.128	17.099
3	17.094	17.167	17.15	17.368
4	17.151	17.144	17.131	17.122
5	17.207	17.167	17.142	17.114
6	17.12	17.093	17.14	17.12
7	17.107	17.121	17.132	17.102
8	17.256	17.137	17.122	17.086
9	17.179	17.138	17.143	17.694
10	17.144	17.361	17.133	17.462
Average	17.1646	17.1631	17.1442	17.2486

Using Table 3, it is found that the best parallelization method is to use the section directive, since that provided the best time amongst the rest. However, increasing the number of threads didn't improve the execution time. The first method – Static, Dynamic – came in second, but it worked better because increasing the number of threads did improve the overall execution time. As for the critical run, the times were about the same as the non-parallelized run, which goes to show that the execution time did not improve.

Problems Encountered:

The main problem encountered was understanding were the #pragma directives should go, in terms of improving the execution time but also in term of logically placing them. For example, on the Bubblesort program, the outer loop was parallelized at one point, and it improved the execution time immensely, but that loop should not have been parallelized because it instructed the program to execute the inner loops.