Noah del Angel

Question 1a) What compute times do you get (in seconds)? List them in a table where the first column lists the number of threads (increasing from top to bottom) and the remaining columns list the compute times for the different schedules (using the same order as in the submission script). Use two digits after the decimal point for the compute times. Highlight the cell showing the lowest compute time in each column.

Threads	static, 1	static, 64	static, 1024	dynamic, 1	dynamic, 64	dynamic, 1024	guided, 1	guided, 64	guided, 1024
14	2.92	2.76	2.75	6.00	2.75	2.74	2.73	2.74	2.73
28	1.51	1.38	1.37	5.78	1.38	1.37	1.37	1.37	1.37
42	0.99	0.93	0.94	5.55	0.93	0.91	0.91	0.93	0.91
56	1.07	0.75	0.72	5.34	0.70	0.69	0.69	0.76	0.69

Question 1b) Which schedule yields the worst parallel performance and why?

The schedule with the worst performance is the dynamic with a chunk size of 1. This is because threads have to come back and wait for the next portion of available chunk.

Question 1c) With 56 threads, what is the main reason for why the dynamic and guided compute times with the two larger chunk sizes are lower than the corresponding static compute times?

This is because dynamic, and schedules provide optimal load balancing, whereas the static chunk size just remained the same as the problem continues.

Question 1d) Explain why the dynamic schedule performs better with larger chunk sizes.

In guided the overhead will be slightly higher per chunk, than in dynamic. Thus, guided will perform worse than dynamic at larger chunk sizes.

Question 1e) How does the default schedule in OpenMP distribute the loop iterations over the threads, and how can it be emulated with a static schedule?

The default schedule in OpenMP is to statically assign loop iterations to available threads. This could be emulated in the static schedule by diving up the loop iterations into chunk sizes.

Question 2a) What compute times do you get (in seconds)? List them in a table where the first column shows the number of threads used (increasing from top to bottom) and the remaining columns show the compute times for the three code versions (in increasing version order). Use two digits after the decimal point for the compute times. Highlight the cell showing the lowest compute time in each column.

Threads	fractal1	fractal2	fractal3
1	27.21	27.09	27.08
12	2.7	4.09	4.09
24	1.38	2.16	2.53
36	1.00	1.48	2.19
48	0.75	1.14	2.57
60	0.75	1.43	2.21

Question 2b) Which version is the fastest (for large thread counts) and why?

The fastest for large thread counts is fractal1. We parallelized the whole for loop, which weakly scales.

Question 2c) The submission script runs one experiment twice. Even though the executable, the program input, and the hardware are the same, the compute times usually differ. Explain why.

Because we don't specify a schedule, omp is scheduling loop iterations statically. As a result runtimes may vary based.

Question 2d) Even though we are using default(none), cx cannot be specified in either a shared() or private() clause in any of the three versions. Why not?

The variable is declared within the parallel section, thus it cannot be shared or private.

Question 2e) Explain why it is safe to share the pic array even though all threads write to it.

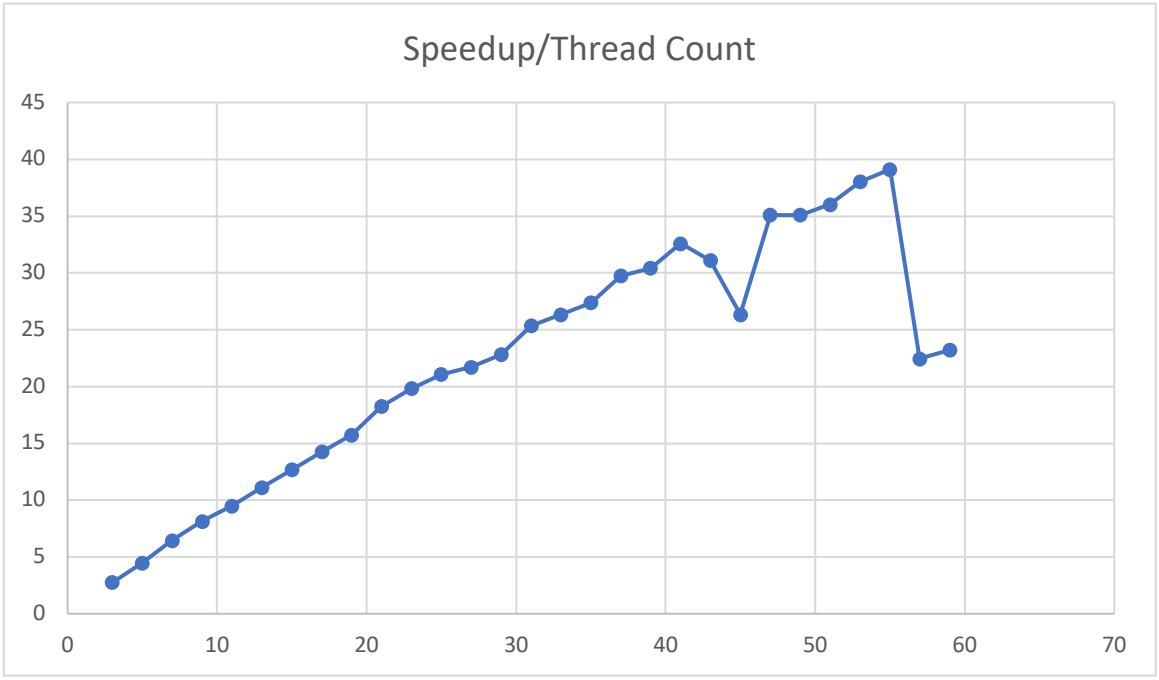
None of the threads work on the same elements, thus it is safe for all the threads to share the array.

Question 3a) What runtimes do you get (in seconds)? List them in a table where the first column shows the number of threads used (increasing from top to bottom) and the second column shows the runtime of the code. Use two digits after the decimal point for the compute times. Highlight the cell showing the lowest compute time.

Threads	Runtime
1	13.69
3	4.97
5	3.05
7	2.12
9	1.68
11	1.44
13	1.23
15	1.08
17	0.96
19	0.87
21	0.75
23	0.69
25	0.65
27	0.63
29	0.60
31	0.54
33	0.52
35	0.50
37	0.46
39	0.45
41	0.42
43	0.44
45	0.52
47	0.39
49	0.39
51	0.38
53	0.36

55	0.35
57	0.61
59	0.59

Question 3b) Plot the speedup (along the y axis) as a function of the thread count (along the x axis) in a line chart.



Question 3c) The speedup curve is smoother between 1 and 27 threads than between 29 and 53 threads. Provide a likely reason for this behavior.

We do not specify a schedule we are using the default static schedule. As a result, chunk size is equal to the number of iterations divided by threads. So, as the thread count increases the chunk decreases. As a result, sometimes the chunk might be too small for the computation.

Question 3d) Explain why the speedup drops sharply for the highest tested thread counts.

There are more threads than cores, and since SMT is disabled, performance drops precipitously.

Question 3e) As discussed, the n-body code requires barriers to work properly. Explain why no "#pragma omp barrier" pragmas are needed in the OpenMP code.

An implied barrier exists at the omp single, which waits for all threads to reach the end of the parallelized code section.

Question 4a) What compute times do you get (in seconds)? Present them in a table for increasing thread counts from top to bottom and use the same order as in the submission script for listing the inputs from left to right. Use three digits after the decimal point for the runtimes. Highlight the cell showing the lowest compute time in each column.

Threads	USA Road	Europe	r4-2e23	Soc-LiveJournal	UK-2002
1	0.711	1.308	0.588	0.298	0.931
14	0.066	0.135	0.079	0.044	0.098
28	0.056	0.096	0.075	0.044	0.085
42	0.043	0.083	0.063	0.060	0.052
56	0.060	0.096	0.073	0.058	0.059

Question 4b) Why is it not possible to specify a schedule for the outermost pragma?

We can only specify schedule for loop pragmas, not on just omp parallel.

Question 4c) Why does the omp.h header file not need to be included?

We only every use the pragma's, as a result we don't need to include the omp.h.

Question 4d) Why would it be incorrect to specify "nowait" for the first loop?

Executions after the first for loop require all elements computed within the for loop to be completed. As a result, further execution must wait for all threads to finish their portion of the for loop.

Question 4e) How and where is the number of threads specified?

The number of threads are specified outside the program execution, in the sub file.