Assignment 1: OpenMP

Elijah Kin

September 18, 2024

Problem 1

For this problem, we replaced the code snippet

```
if (dist < min_dist) { min_dist = dist; }
```

with the equivalent

```
min_dist = std::min(min_dist, dist);
```

such that `min_dist` can be made a reduction variable by adding the OpenMP directive

```
#pragma omp parallel for reduction(min : min_dist)
```

above the outer `for` loop. As far as performance, timing the code for problem size 16384 for different numbers of threads, we measure

Threads	1	2	4	8	16	32	64
Time (s)	0.37301	0.27924	0.16289	0.08785	0.04679	0.02501	0.01717

Problem 2

We first replace the code snippet

```
if (A[i * N + j] == 1) {count++}
```

with the below via a ternary expression

```
count += (A[i * N + j] == 1);
```

such that `count` can be made a reduction variable by adding the OpenMP directive

```
#pragma omp parallel for reduction(+ : count)
```

above the outer `for` loop. Additionally, we flatten the two `for` loops into one. Timing the code with problem size 8192 across different numbers of threads, we measure

Threads	1	2	4	8	16	32	64
Time (s)	0.04228	0.02145	0.01675	0.01825	0.02220	0.02176	0.01850

Problem 3

We first rewrote the code snippet

```
if (i % 2 == 1) {result *= 1 / x[i];} else {result *= x[i];}
```

with the below to instead use a ternary expression

```
result *= (i % 2) ? (1 / x[i]) : x[i];
```

Further, then noting that `result` can be made a reduction variable, we added the OpenMP directive

```
#pragma omp parallel for reduction(* : result)
```

Timing the code with problem size 67108864 across different numbers of threads, we measure

Threads	1	2	4	8	16	32	64
Time (s)	0.06922	0.03595	0.03083	0.03074	0.03305	0.02768	0.02909

Problem 4

As specified, we first created a copy `dft_omp` of `dft` and changed line 61 to call `dft_omp`. Within `dft_omp`, we moved the declaration of `theta` to the inner `for` loop (since it is not used outside this loop), and added the OpenMP directive

```
#pragma omp parallel for
```

to the outer for loop. Timing the code for problem size 8192 with different numbers of threads, we find

Threads	1	2	4	8	16	32	64
Time (s)	2.58517	1.93782	1.61645	1.45481	1.43629	1.33427	1.31693

It is expected that the 64 thread version is roughly twice as fast as the single thread version; regardless of the number of threads, the serial function `dft` is being included in the timing.