COL380
Assignment – 1

FUNCTIONS OF THE OMP LIBRARY USED IN THE ASSIGNMENT :

1. **omp_set_num_threads(threads)** :- Used to set the number of threads which we want
2. **pragma omp parallel for** :- Used to parallelise the for loop by giving it the number of threads specified using omp_set_num_threads() function.
3. **pragma omp barrier** :- used to join all the threads.

FINDINGS FROM THE ASSIGNMENT :

N	Serial_time 1 (in sec)	Serial_time 2	Serial_time 3	Average Serial time (when threads = 1)
1000	0.000077	0.000063	0.000064	0.000068
100000	0.001918	0.001885	0.001998	0.001934
10000000	0.129652	0.132616	0.130149	0.130806

STRATEGY 0 :

N	Threads	Time 1	Time 2	Time 3	Avg. time
1000	2	0.000215	0.000215	0.000218	0.000216
	4	0.000430	0.000330	0.000436	0.000398
	8	0.000538	0.000535	0.000568	0.000547
100000	2	0.001555	0.001638	0.001701	0.001631
	4	0.001724	0.001558	0.001648	0.001643
	8	0.001753	0.001791	0.001827	0.001790
10000000	2	0.072872	0.071482	0.072123	0.072159
	4	0.098625	0.073831	0.071540	0.081332
	8	0.071006	0.070942	0.070407	0.070785

STRATEGY 1 :

N	Threads	Time 1	Time 2	Time 3	Avg. time
1000	2	0.000390	0.000210	0.000232	0.000277

	4	0.000654	0.000634	0.000648	0.000645
	8	0.000915	0.000838	0.000934	0.000896
100000	2	0.001861	0.001779	0.001977	0.001872
	4	0.001867	0.001877	0.001776	0.001840
	8	0.002416	0.002468	0.002558	0.002480
10000000	2	0.105521	0.106306	0.107470	0.101064
	4	0.106925	0.107328	0.107328	0.107194
	8	0.108776	0.107569	0.109020	0.108455

Speedup => T_serial/T_parallel
Efficiency => Speedup/Threads

FINAL TRENDS

N	Threads	Strategy 0		Strategy 1	
		Speedup	Efficiency	Speedup	Efficiency
1000	2	0.314815	0.157407	0.245487	0.122743
	4	0.170854	0.042713	0.105426	0.026356
	8	0.124314	0.031078	0.075892	0.018973
100000	2	1.185775	0.592887	1.033119	0.516559
	4	1.177115	0.294278	1.051086	0.262771
	8	1.080446	0.270112	0.779838	0.194959
10000000	2	1.812747	0.906373	1.294288	0.647144
	4	1.608296	0.402074	1.220227	0.305056
	8	1.847933	0.461983	1.206085	0.301521

SOME OBSERVATIONS MADE :

- Sometimes increasing the threads are also increasing the time taken because of the large overhead over the small computation, i.e., the compution is quite small which is taking just 0.13 seconds in serial version and that too when N = 10000000, so this clearly shows that the total time required by the parallel version is significantly effected by the overhead of the threads.

- The speedup for Strategy 0 is more than that of strategy 1 , it is most probably due to the considerably large amount os load and store operations being done in memory , as we are constinuosly updating the entire array for it.

- The speedup is lower for lower values of N but it is increasing when we increase the value of N, this is because of the fact that the effect of overhead of parallelisation is clearly visible on the much smaller computation when N is less as compared to the comparatively larger computation when N is larger.

- The low values of efficiency and speedup are also a clear indication of the fact that the fraction of the parallelised program is not that good. Which we also concluded above.

- The given program is computationally very less time consuming, hence it is getting effected by the time of the overhead due to which we are not getting that good expected results.

AMDAHL'S LAW :-

Unless the entire serial program is parallelized, the possible speedup is going to be very limited regardless of the number of processors,
i.e.,
Speedup is decided by the sequential/unparallelized version

So,
T_parallel = f * (T_serial / p) + (1-f) * T_serial
S = T_serial / T_parallel = T_serial / (f * (Tserial / p) + (1-f) * Tserial)
So the maximum value of speedup in ideal case when no overhead is there will be when p -> inf. So,

Actual **Speedup ≤ T_serial / (1-f) * T_serial**

IS MY PROGRAM FOLLOWING AMDAHL'S LAW ? :-

Yes, my program is following Amdahl's law.

Since in the serial program the first step is the initialisation of the array which traverses the array once and the second step is calculting the sum sequentially which also traverses the array once.
So, while parallelising the program we have parallelised only the second step of the serial program and since the first and the second step both requires almost equal time, so we can say that while parallelising the given serial program, we have parallelised about 50% of the serial program.
Hence, **f (fraction of the code parallelised) ~ 0.5**
Now, according to the Amdahl's law,

The maximum possible speedup is :- 1 / (1-f) = 1 / (1-0.5) = 2
And the values of speedup which I am getting are all less than 2, hence I concluded that my program is compling with the Amdahl's law.

HOW I COMPILED AND EXECUTED IT :

Compilation: clang -Xpreprocessor -fopenmp 2018CS50413.c -lomp
Execution: -> ./a.out 0 1000 2
-> ./a.out 1 1000 4

SUBMITTED BY :

- Nikita Bhamu
- 2018CS50413