Sreepathy Jayanand

CED17I038

High Performance Computing LAB - 3

Analysis of parallelization fraction

**Processor Specifications:**

* Intel i5 - 5350U
* Base clock speed - 1.80GHz
* Max clock speed - 2.90GHz
* Physical Cores - 2
* Threads / Core - 2
* Logical Cores - 4
* L1 cache size (per core) - 64 KB
* L2 cache size (per core) - 256 KB
* L3 cache size - 3 MB
* Main memory - 8GB

**Question 1A (Reduction Method):**

Input size = 4 \* 1e8

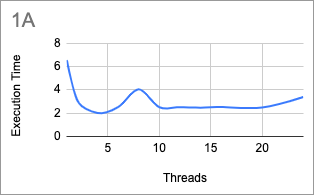
Segmentation fault at 5 \* 1e9

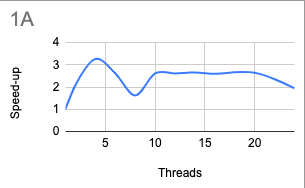
Strategy:

#pragma omp parallel for reduction(+:ans) - This clause is used in 1A. This creates multiple copies of the variable “ans” rather than creating a single variable and all threads accessing that which create a race condition because there is a possibility that 2 threads read the same value parallely rather than reading them serially so that the value updated by one thread should be reflected for the other thread.

Table for 1A:

| No of threads | Execution Time(s) |
| --- | --- |
| 1 | 6.547 |
| 2 | 3.138 |
| 4 | 2.003 |
| 6 | 2.499 |
| 8 | 4.037 |
| 10 | 2.497 |
| 12 | 2.503 |
| 14 | 2.461 |
| 16 | 2.520 |
| 20 | 2.472 |
| 24 | 3.369 |





The most efficient thread size for this program is 4

According to Amdahl’s law :

Speed up = 1/((1-p) + p/N)

Or

Parallelization factor = (N - N/S.U)/(N - 1)

So here N = 4;

Speed Up = 6.547/2.003 = 3.26

PF = (4 - 4/3.26)/3 = .92

**Question 1B (Using critical section):**

Input size = 4 \* 1e8

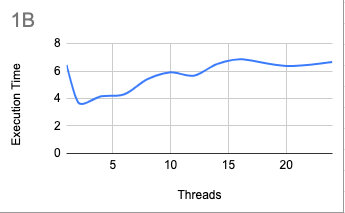
Segmentation fault at 5 \* 1e9

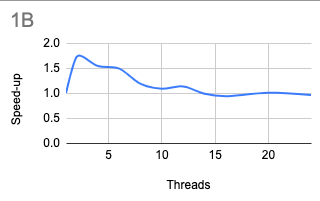
Strategy:

#pragma omp parallel clause creates all the threads and thread gets a work of summing n / num\_of\_threads indices. The #pragma omp critical ensures that no 2 threads update the global variable “ans” simultaneously so that race conditions are taken care of.

Table for 1B:

| No of threads | Execution Time(s) |
| --- | --- |
| 1 | 6.454 |
| 2 | 3.719 |
| 4 | 4.163 |
| 6 | 4.308 |
| 8 | 5.419 |
| 10 | 5.908 |
| 12 | 5.664 |
| 14 | 6.508 |
| 16 | 6.879 |
| 20 | 6.379 |
| 24 | 6.667 |





The most efficient thread size for this program is 2

According to Amdahl’s law :

Speed up = 1/((1-p) + p/N)

Or

Parallelization factor = (N - N/S.U)/(N - 1)

So here N = 2;

Speed Up = 6.454/3.719 = 1.73

PF = (2 - 2/1.73)/1 = .84

**Question 2(Reduction method):**

Input size = 1e8

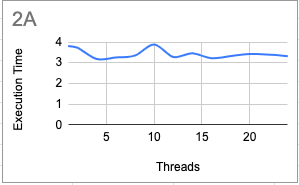
Segmentation fault at 1e9

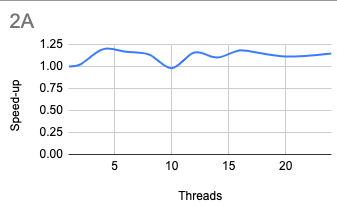
Strategy:

#pragma omp parallel creates all the threads and each thread has the work of multiplying the corresponding elements of both the arrays for n / num\_of\_thread indices and adding that to the “ans” variable to calculate the dot product inside the omp parallel critical clause to ensure that no race condition is happening.

Table Q2:

| No of threads | Execution time(s) |
| --- | --- |
| 1 | 3.813 |
| 2 | 3.725 |
| 4 | 3.179 |
| 6 | 3.259 |
| 8 | 3.350 |
| 10 | 3.883 |
| 12 | 3.282 |
| 14 | 3.458 |
| 16 | 3.217 |
| 20 | 3.422 |
| 24 | 3.320 |





The most efficient thread size for this program is 4

According to Amdahl’s law :

Speed up = 1/((1-p) + p/N)

Or

Parallelization factor = (N - N/S.U)/(N - 1)

So here N = 4;

Speed Up = 3.818/3.179 = 1.20

PF = (4 - 4/1.20)/3 = .22

**Question 2B(Critical section method):**

Input size = 1e8 for both arrays

Segmentation fault at 1e9

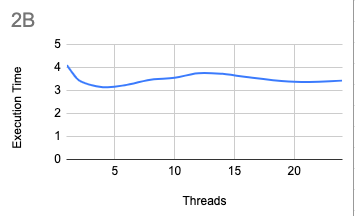
Strategy:

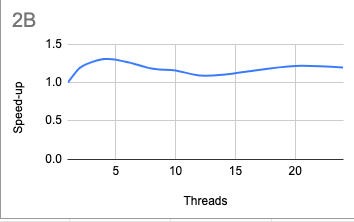
#pragma creates multiple threads and each thread computes the partial sum by adding the multiplied values of n/num\_of\_thread indices. These are done independently.

The addition of the partial sum is done inside the critical section of each of the threads, since we are updating the same variable there is a possibility of a race condition.

Table for 2(Critical section method):

| No of threads | Execution Time(s) |
| --- | --- |
| 1 | 4.083 |
| 2 | 3.430 |
| 4 | 3.126 |
| 6 | 3.299 |
| 8 | 3.456 |
| 10 | 3.535 |
| 12 | 3.739 |
| 14 | 3.708 |
| 16 | 3.572 |
| 20 | 3.360 |
| 24 | 3.414 |





The most efficient thread size for this program is 4

According to Amdahl’s law :

Speed up = 1/((1-p) + p/N)

Or

Parallelization factor = (N - N/S.U)/(N - 1)

So here N = 4;

Speed Up = 4.083/3.126 = 1.306

PF = (4 - 4/1.306)/3 = .43