HIGH PERFORMANCE COMPUTING PRACTICE

EXPERIMENT - 2

OpenMP:

1) Matrix addition
2) Matrix multiplication column major order, and
3) Matrix multiplication block based approach.

Use input as a larger double number (64-bit).

Run experiment for Threads = {1, 2, 4, 6, 8, 10, 12, 14, 16, 20, 24}.

Estimate the parallelization fraction.

Document the report and submit.

Bhaavanaa Thumu - CED17I021

Specifications

The size of all the matrices used for below three programs is 500x500.

All the elements of the matrices are produced randomly and are large double numbers (64-bit).

Formulae used

Speed-up = T(1)/T(P)

Parallel fraction, f = (1 - T(P)/T(1))/(1 - (1/P))

where,

T(1) - time taken for serial execution

T(P) - time taken for parallel execution

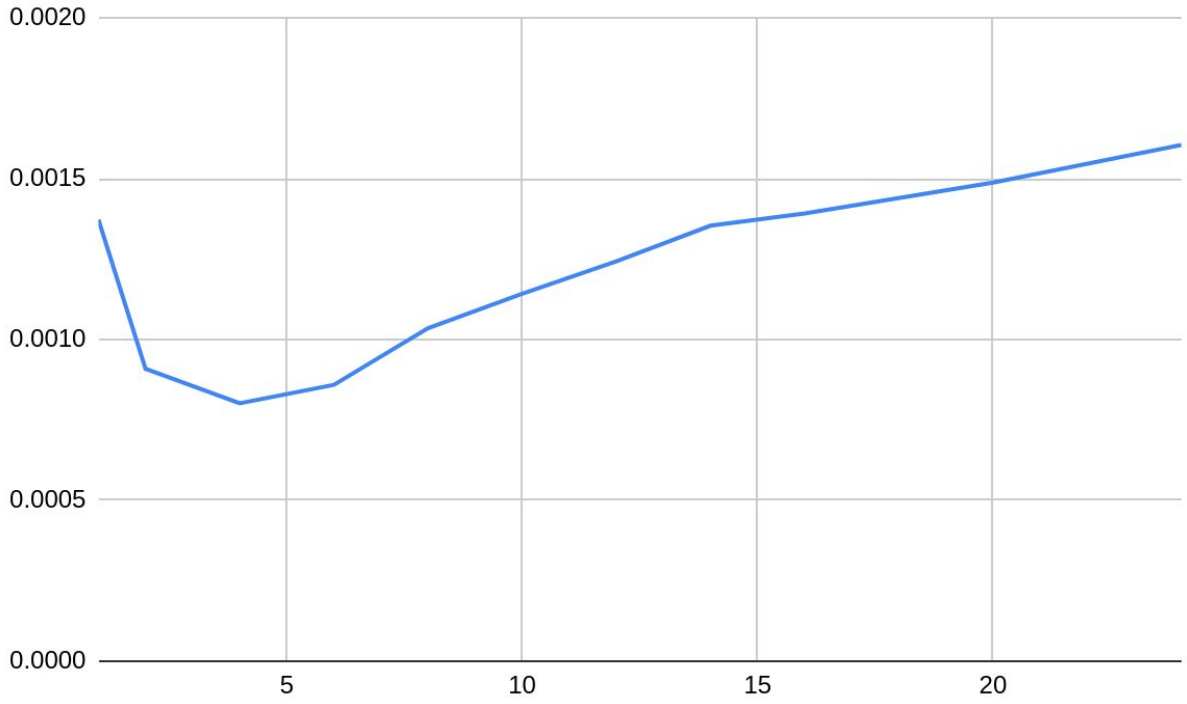
P - number of processes/threads

1. Matrix addition

In this program, two matrices, each of size 500x500, are added.
The number of threads initialized and the execution time in each case is as shown in the table.

No. of threads	*Execution time*	*Speed-up*	*Parallel fraction*
1	0.001373	1	#DIV/0!
2	0.000909	1.510451045	0.6758922068
4	**0.000802**	**1.711970075**	**0.5545035203**
6	0.000859	1.598370198	0.4492352513
8	0.001035	1.326570048	0.281344293
10	0.001142	1.202276708	0.1869385773
12	0.001243	1.10458568	0.1032907369
14	0.001354	1.014032496	0.01490279567
16	0.001392	0.9863505747	-0.01476086429
20	0.001488	0.9227150538	-0.08816651972
24	0.001605	0.8554517134	-0.1763197061

The plot below is the graph between no. of processes/threads and execution time.



x-axis : no. of processes/threads

y-axis : execution time

Hence, in this case, the optimal number of threads is 4. In this case-

Execution time = 0.000802

Speed-up = 1.711970075

Parallel fraction = 0.5545035203

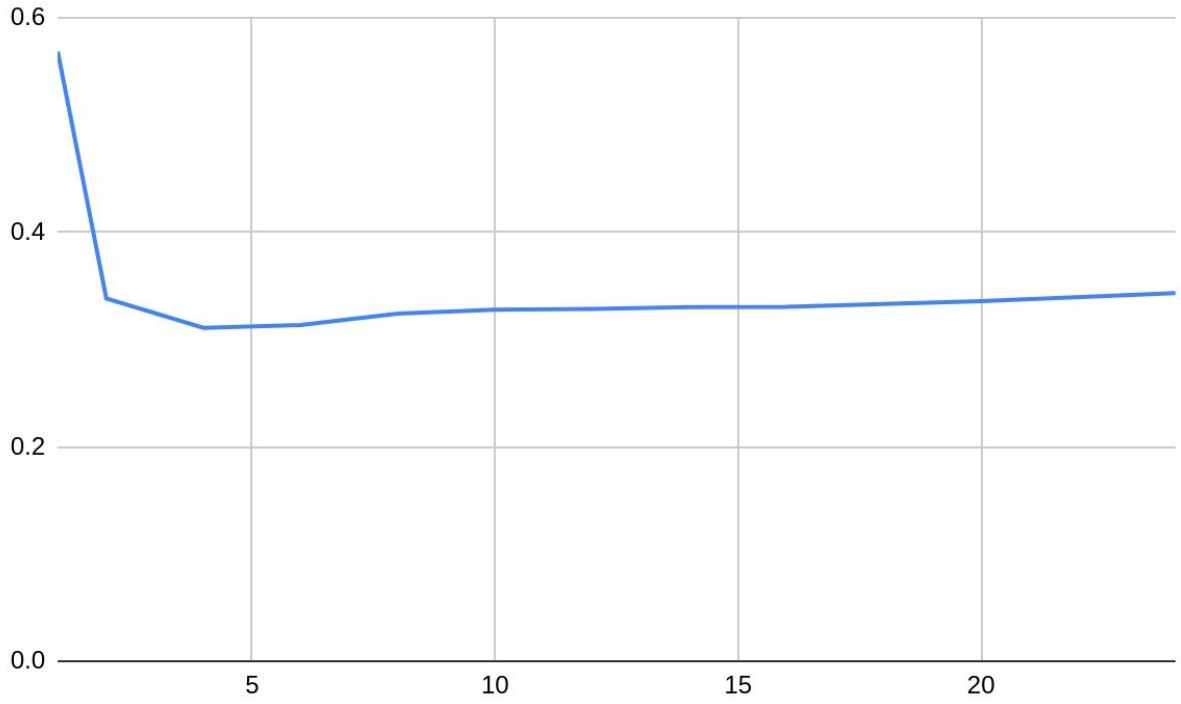
In this case, the loop iterations are being parallelized.

2. Matrix multiplication column major order

In this program, two matrices, each of size 500x500, are multiplied. The number of threads initialized and the execution time in each case is as shown in the table.

No. of threads	*Execution time*	*Speed-up*	*Parallel fraction*
1	0.568707	1	#DIV/0!
2	0.338254	1.681301625	0.8104454491
4	**0.310934**	**1.829028025**	**0.6043486951**
6	0.313480	1.814173153	0.5385416392
8	0.324115	1.754645728	0.4915250107
10	0.327806	1.734888928	0.4706602482
12	0.328532	1.731055118	0.4607101564
14	0.330307	1.721752794	0.4514424151
16	0.330500	1.720747352	0.4467809727
20	0.335737	1.693906242	0.4312090038
24	0.343211	1.657018569	0.4137458725

The plot below is the graph between no. of processes/threads and execution time.



x-axis : no. of processes/threads

y-axis : execution time

Hence, in this case, the optimal number of threads is 4. In this case-

Execution time = 0.310934

Speed-up = 1.829028025

Parallel fraction = 0.6043486951

In this case, the loop iterations are being parallelized.

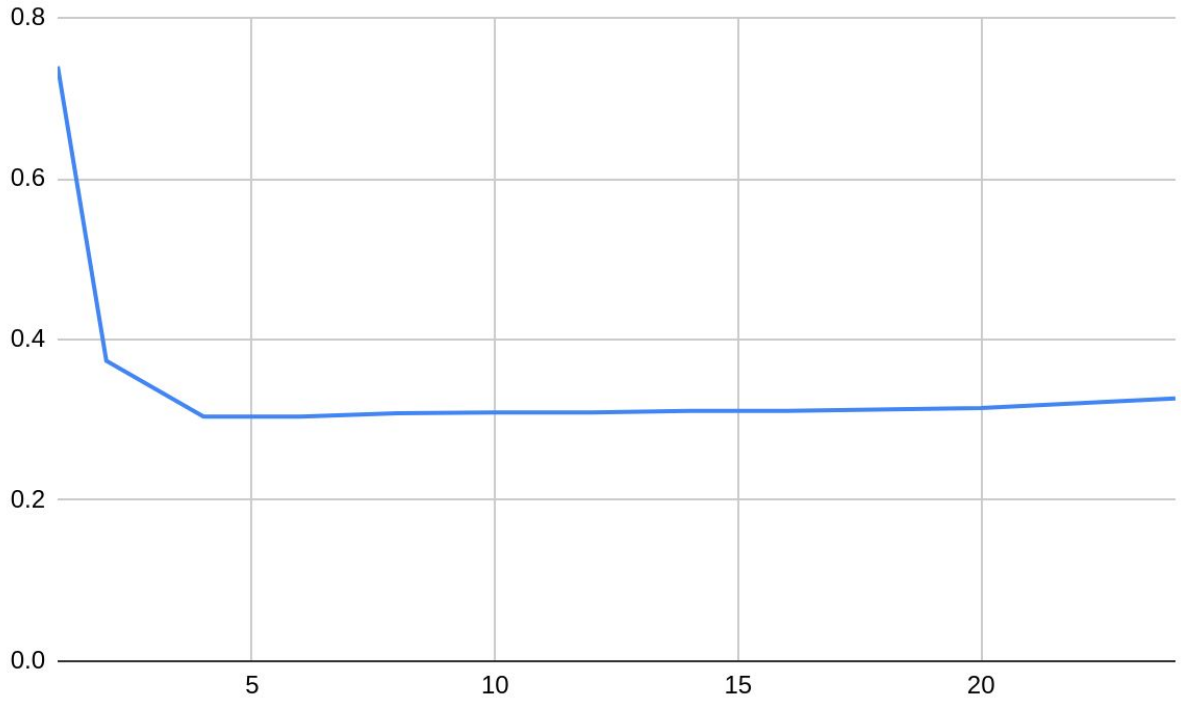
3. Matrix multiplication block based approach

In this program, two matrices, each of size 500x500, are multiplied. The block size B is considered to be 50 in this case.

The number of threads initialized and the execution time in each case is as shown in the table.

No. of threads	*Execution time*	*Speed-up*	*Parallel fraction*
1	0.739750	1	#DIV/0!
2	0.373474	1.980726905	0.9902696857
4	**0.304036**	**2.433100028**	**0.7853355863**
6	0.304275	2.431188892	0.7064143292
8	0.308299	2.399456372	0.6665587795
10	0.309239	2.392162696	0.6466313695
12	0.309352	2.391288888	0.6347077944
14	0.311229	2.376867194	0.623837991
16	0.311263	2.376607563	0.6178476512
20	0.314737	2.350375075	0.6047747283
24	0.326821	2.263471442	0.582470341

The plot below is the graph between no. of processes/threads and execution time.



x-axis : no. of processes/threads

y-axis : execution time

Hence, in this case, the optimal number of threads is 4. In this case-

Execution time = 0.304036

Speed-up = 2.433100028

Parallel fraction = 0.7853355863

In this case, the loop iterations are being parallelized.