HIGH PERFORMANCE COMPUTING PRACTICE

EXPERIMENT - 5

MPI:

1) Matrix addition

2) Matrix multiplication column major order

3) Matrix multiplication block based approach

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

Run experiment for Threads = {1, 2, 4, 8, 16, 32, 64, 126, 140}.

Estimate the parallelization fraction.

Document the report and submit.

Bhaavanaa Thumu - CED17I021

Specifications

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/processors

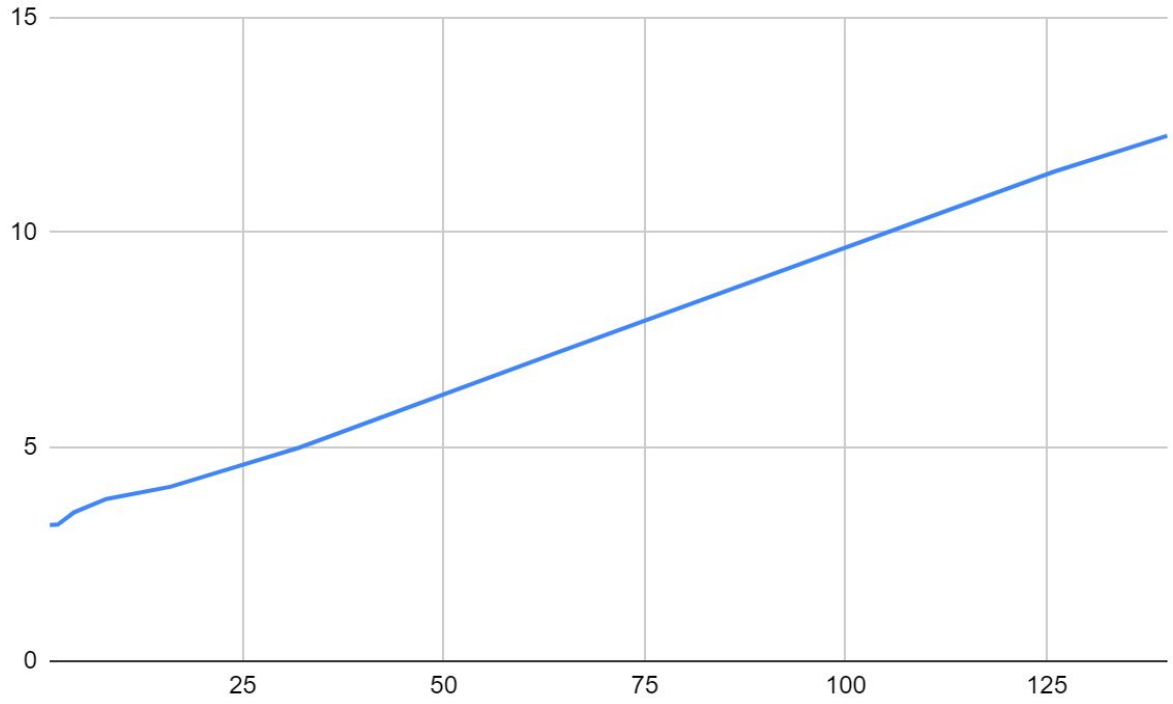
1. Matrix addition

This program contains matrices of size 1000x1000.

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

No. of processors	*Execution time*	*Speed-up*	*Parallel fraction*
1	3.173957	1	#DIV/0!
2	3.184115	0.9968097886	-0.00640084286
4	3.468915	0.9149711077	-0.1239075808
8	3.778498	0.8400049438	-0.2176790675
16	4.065914	0.7806257093	-0.2997585664
32	4.976436	0.6377972107	-0.5862157187
64	7.187900	0.4415694431	-1.284723259
126	11.420791	0.2779104355	-2.61906783
140	12.253410	0.2590264261	-2.881189886

The plot below is the graph between no. of processors and execution time.



x-axis : no. of processors

y-axis : execution time

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

Execution time = 3.173957

Speed-up = 1

Parallel fraction = #DIV/0!

Method for parallel execution:

The master process splits the entire work into sub-parts and gives it to the workers, so that they can work in parallel. So, the matrices a and b are sent to the workers and each worker computes its assigned part of the result by adding elements of a and b and storing it in c1, which is sent to the master and then combined together to form the resultant matrix c.

Load split-up:

The load is equally divided among all the worker nodes. The master process will be doing the left out/ remaining part and has the job of joining all the results obtained from various worker nodes. So, in this case, worker load is 1000x1000/(no. of worker processes) and the master load is 1000x1000/(no. of processes)+1000x1000%(no. of processes).

Communication:

The master communicates or sends the data to the workers using MPI_Send and the workers receive it through MPI_Recv. Point to point communication is used here.

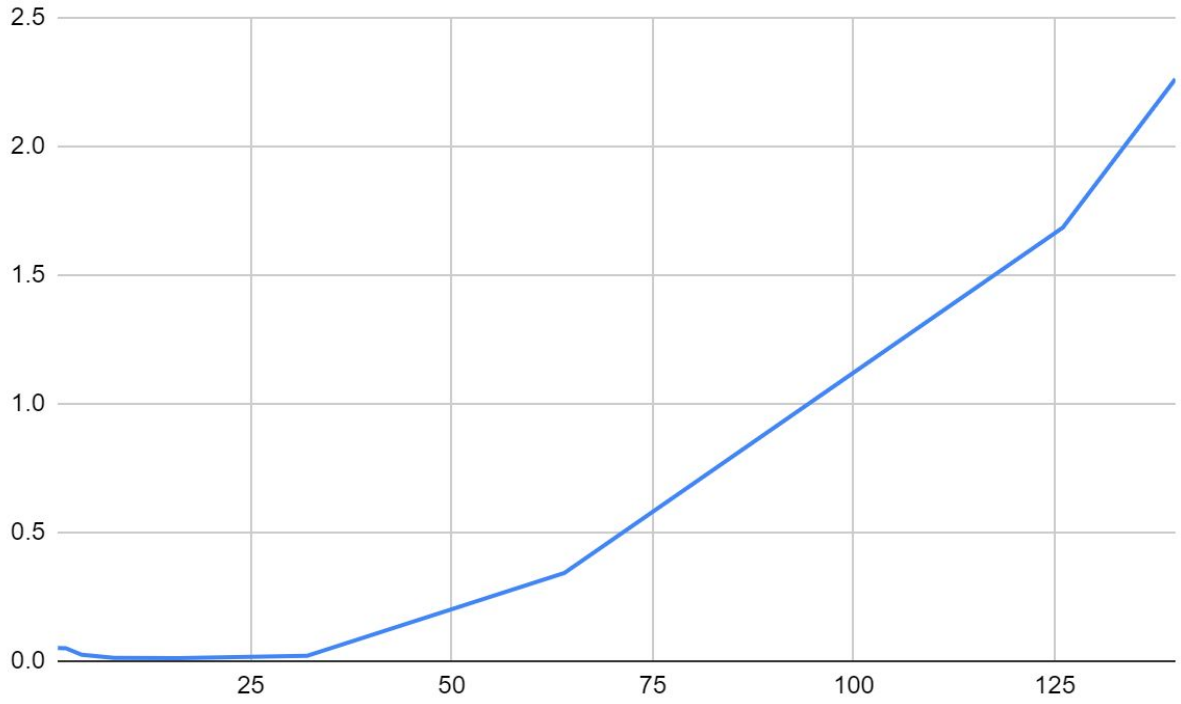
2. Matrix multiplication column major order

This program contains matrices of size 200x200.

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

No. of processors	*Execution time*	*Speed-up*	*Parallel fraction*
1	0.050748	1	#DIV/0!
2	0.050079	1.013358893	0.02636557106
4	0.024455	2.075158454	0.6908121174
8	0.012892	3.936394663	0.8525262079
16	0.011434	4.438341788	0.8263366701
32	0.020559	2.468407996	0.6140702828
64	0.342390	0.1482169456	-5.838086981
126	1.685321	0.03011177099	-32.46728115
140	2.262360	0.02243144327	-43.89380621

The plot below is the graph between no. of processors and execution time.



x-axis : no. of processors

y-axis : execution time

Hence, in this case, the optimal number of processors is 16. In this case-

Execution time = 0.011434

Speed-up = 4.438341788

Parallel fraction = 0.8263366701

Method for parallel execution:

The master process splits the entire work into sub-parts and gives it to the workers, so that they can work in parallel. So, the matrices a and b are sent to the workers and each worker computes its assigned part of the result by multiplying the required elements of a and b and storing it in c1, which is sent to the master and then combined together to form the resultant matrix c.

Load split-up:

The load is equally divided among all the worker nodes. The master process will be doing the left out/ remaining part and has the job of joining all the results obtained from various worker nodes. So, in this case, worker load is 200x200/(no. of worker processes) and the master load is 200x200/(no. of processes)+200x200%(no. of processes).

Communication:

The master communicates or sends the data to the workers using MPI_Send and the workers receive it through MPI_Recv. Point to point communication is used here.

3. Matrix multiplication block based approach

This program contains matrices of size 1000x1000.

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

No. of processors	*Execution time*	*Speed-up*	*Parallel fraction*
1			
2			
4			
8			
16			
32			
64			
126			
140			

The plot below is the graph between no. of processors and execution time.

x-axis : no. of processors

y-axis : execution time

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

Execution time =

Speed-up =

Parallel fraction =

The load is equally divided among all the worker nodes. The master process will be doing the left out/ remaining part and has the job of joining all the results obtained from various worker nodes.