HIGH PERFORMANCE COMPUTING PRACTICE

EXPERIMENT - 6

MPI:

1) Addition of N-numbers (double precision numbers). Identify the challenge.

2) Vector dot product

Implement using reduction.

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

The size of all the vectors used for below two programs is 1,00,000.

All the elements of the vectors 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. Addition of N numbers

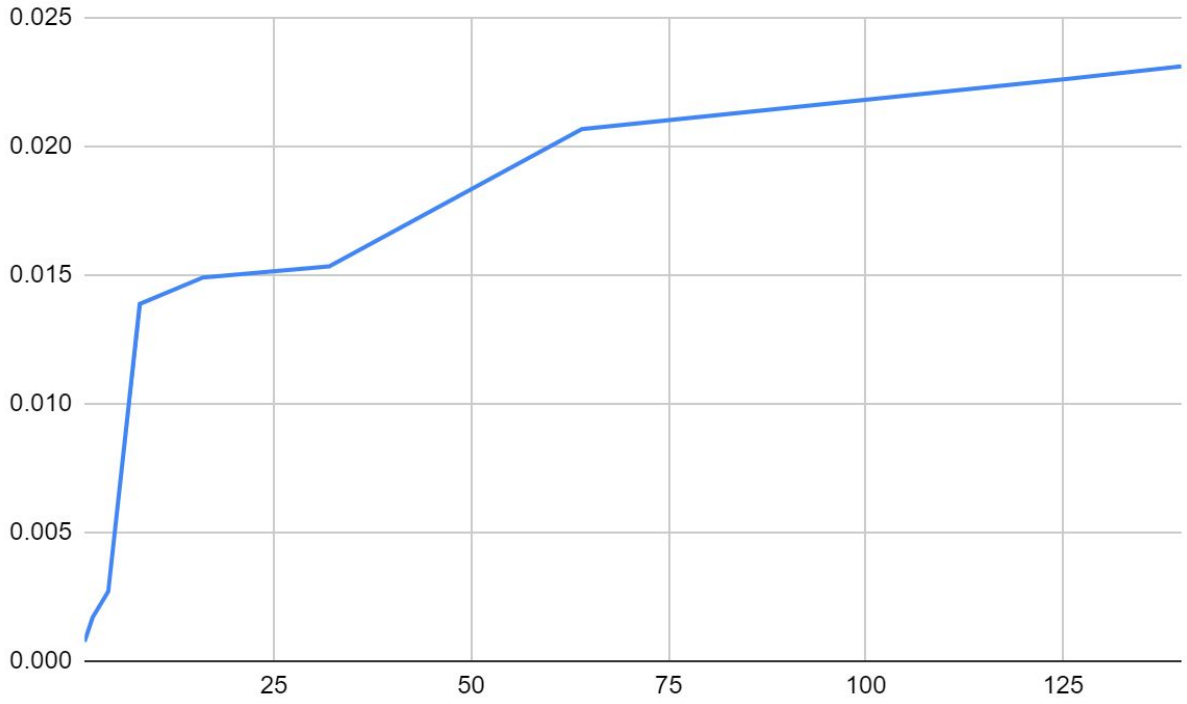
This program contains a vector of size 1,00,000.

Therefore, here, N = 1,00,000.

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	0.000753	1	#DIV/0!
2	0.001701	0.442680776	-2.517928287
4	0.002704	0.2784763314	-3.454625941
8	0.013890	0.05421166307	-19.93853159
16	0.014913	0.05049285858	-20.05843293
32	0.015339	0.04909055349	-19.99537335
64	0.020680	0.03641199226	-26.88353464
126	0.022643	0.03325531069	-29.30294821
140	0.023117	0.03257343081	-29.91353531

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 1. In this case-

Execution time = 0.000753

Speed-up = 1

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 vector a is sent to the workers and each worker computes its assigned part of the result by adding elements in a and storing it in part_sum, which is sent to the master and then added together to form the resultant sum.

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 100000/(no. of worker processes) and the master load is 100000/(no. of processes)+100000%(no. of processes).

Communication:

The master communicates or sends the data to the workers using MPI_Bcast and the workers receive it through MPI_Reduce. Broadcast communication is used here.

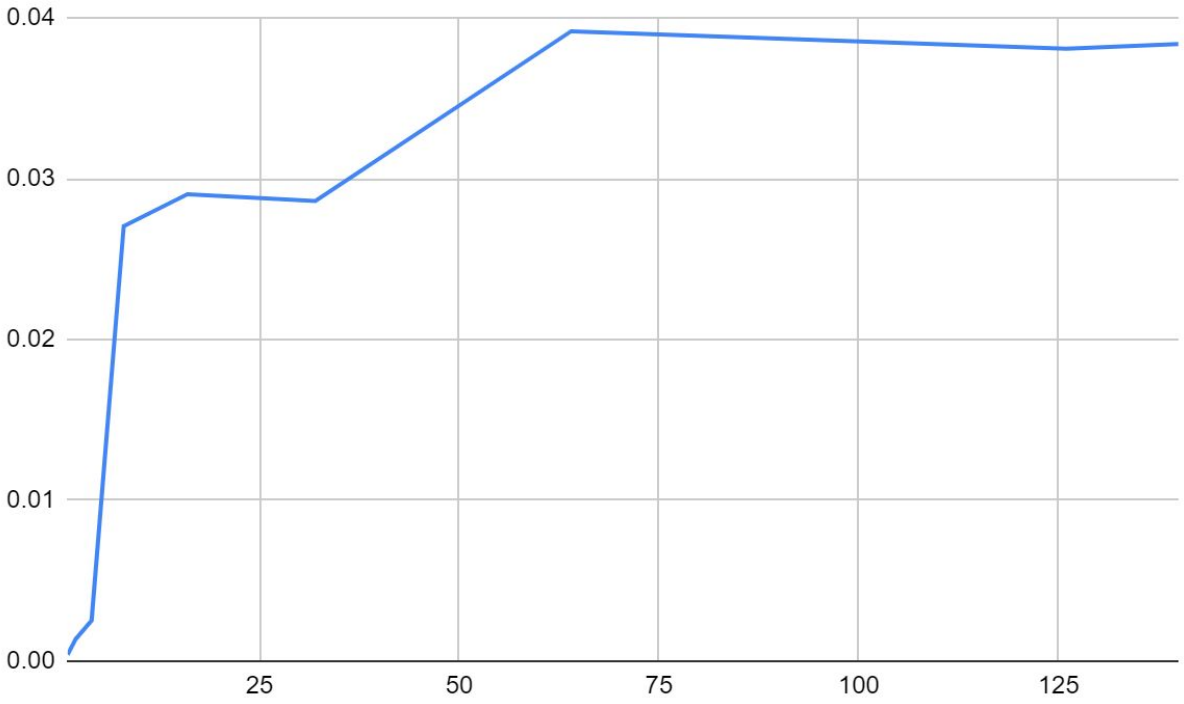
2. Vector dot product

In this program, two vectors, each of size 1,00,000, are used for finding the dot product.

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	0.000406	1	#DIV/0!
2	0.001376	0.2950581395	-4.778325123
4	0.002523	0.1609195402	-6.952380952
8	0.027051	0.01500868729	-75.00351865
16	0.029040	0.01398071625	-75.22889984
32	0.028614	0.0141888586	-71.71905292
64	0.039171	0.0103648107	-96.99585581
126	0.038082	0.01066120477	-93.54041379
140	0.038382	0.01057787505	-94.20987348

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 1. In this case-

Execution time = 0.000406

Speed-up = 1

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 vectors a and b are sent to the workers and each worker computes its assigned part of the result by finding dot product of elements in a and b and storing the sum in part_dot_sum, which is sent to the master and then added together to form the result dot_product.

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 100000/(no. of worker processes) and the master load is 100000/(no. of processes)+100000%(no. of processes).

Communication:

The master communicates or sends the data to the workers using MPI_Bcast and the workers receive it through MPI_Reduce. Broadcast communication is used here.