High Performance Computing

COM403P

Week-5

Vector Multiplication

Kadali Paul Babu

CED19I002

CED19I002
Kadali Paul Babu

Objective

Vector Multiplication for given n x n double precision floating point numbers.

Serial Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>

#define SIZE 100000

int main()
{
   double a[SIZE], b[SIZE], c[SIZE], rand_a, rand_b;
   double start, end, exec;

   start = omp_get_wtime();

   for (int i = 0; i < SIZE; i++)
   {
       rand_a = rand();
       rand_b = rand();

       a[i] = i*rand_a;
       b[i] = i*rand_b;

       c[i] = a[i] * b[i];
   }

   end = omp_get_wtime();

   exec = end - start;

   printf("Serial Exec time - %f\n", exec);

   return 0;
}
```

Parallel Code

CED19I002
Kadali Paul Babu

```c
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <time.h>

#define MASTER 0
#define FROM_MASTER 1
#define FROM_WORKER 2

#define SIZE 100

int main()
{
   MPI_Init(NULL, NULL);

   long double a[SIZE][SIZE], b[SIZE][SIZE], c[SIZE][SIZE], rand_a, rand_b;
   double start, end, exec;
   int i,j,k;

   int avgrow, extra;
   int offset, mtype;
   int dest, rows;
   int source;
   MPI_Status status;

   start = MPI_Wtime();

   int taskid;
   MPI_Comm_rank(MPI_COMM_WORLD,&taskid);

   int numtasks;
   MPI_Comm_size(MPI_COMM_WORLD,&numtasks);

   int workers = numtasks -1;

   if(taskid == MASTER)
   {
       for (i = 0; i < SIZE; i++)
       {
           a[i][j] = (i+j)*1.22;
           b[i][j] = (i+j)*1.22;
       }
```

```
//--------------------------------------------------------------------------------------
--------------
       // Master sending Data to Worker tasks

       avgrow = SIZE/workers;
       extra = SIZE%workers;

       offset = 0;
       mtype = FROM_MASTER;

       for(dest = 1; dest <= workers; dest++)
       {
           rows = (dest <= extra)?avgrow+1:avgrow;

           // printf("Sending %d rows to task-%d offset=%d\n",rows,dest,offset);

           MPI_Send(&offset, 1, MPI_INT, dest, mtype, MPI_COMM_WORLD);
           MPI_Send(&rows, 1, MPI_INT, dest, mtype, MPI_COMM_WORLD);
           MPI_Send(&a[offset], rows, MPI_LONG_DOUBLE, dest, mtype, MPI_COMM_WORLD);
           MPI_Send(&b[offset], rows, MPI_LONG_DOUBLE, dest, mtype, MPI_COMM_WORLD);

           offset += rows;
       }


//--------------------------------------------------------------------------------------
--------------
       //Receiving results from Workers

       mtype = FROM_WORKER;
       for(i = 1; i <= workers; i++)
       {
           source = i;

           MPI_Recv(&offset, 1, MPI_INT, source, mtype, MPI_COMM_WORLD, &status);
           MPI_Recv(&rows, 1, MPI_INT, source, mtype, MPI_COMM_WORLD, &status);
           MPI_Recv(&c[offset], rows, MPI_LONG_DOUBLE, source, mtype, MPI_COMM_WORLD,
&status);

           // printf("Received results from task=%d\n", source);
       }
       // printf("Result Matrix\n");
```

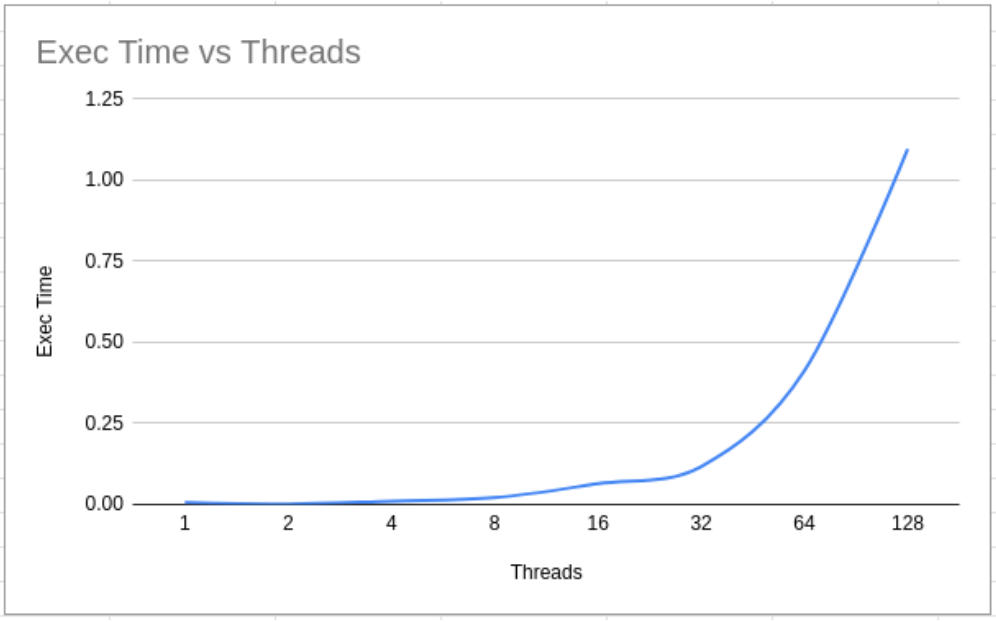
```
        // for(i = 0; i < SIZE;i++)
        // {
        //     // printf("\n");
        //     for(int j = 0; j < SIZE;j++)
        //     {
        //         // printf("%6.2Lf  ",c[i][j]);
        //     }
        // }

        end = MPI_Wtime();

        exec = end - start;
        printf("MPI Exec time - %f\n", exec);
    }


//-------------------------------------------------------------------------------------------
--------------




//-------------------------------------------------------------------------------------------
--------------
    //Worker Task
    if(taskid > MASTER)
    {
        mtype = FROM_MASTER;

        MPI_Recv(&offset, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD,&status);
        MPI_Recv(&rows, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD,&status);
        MPI_Recv(&a, rows, MPI_LONG_DOUBLE, MASTER, mtype, MPI_COMM_WORLD,&status);
        MPI_Recv(&b, rows, MPI_LONG_DOUBLE, MASTER, mtype, MPI_COMM_WORLD,&status);


        for(i = 0; i < rows; i++)
        {
            c[i] = a[i] * b[i];
        }

        mtype = FROM_WORKER;

        MPI_Send(&offset,1, MPI_INT, MASTER,mtype, MPI_COMM_WORLD);
        MPI_Send(&rows,1, MPI_INT, MASTER,mtype, MPI_COMM_WORLD);
```

```
        MPI_Send(&c, rows, MPI_LONG_DOUBLE, dest, mtype, MPI_COMM_WORLD);

    }
    MPI_Finalize();

    return 0;
}
```

Observations

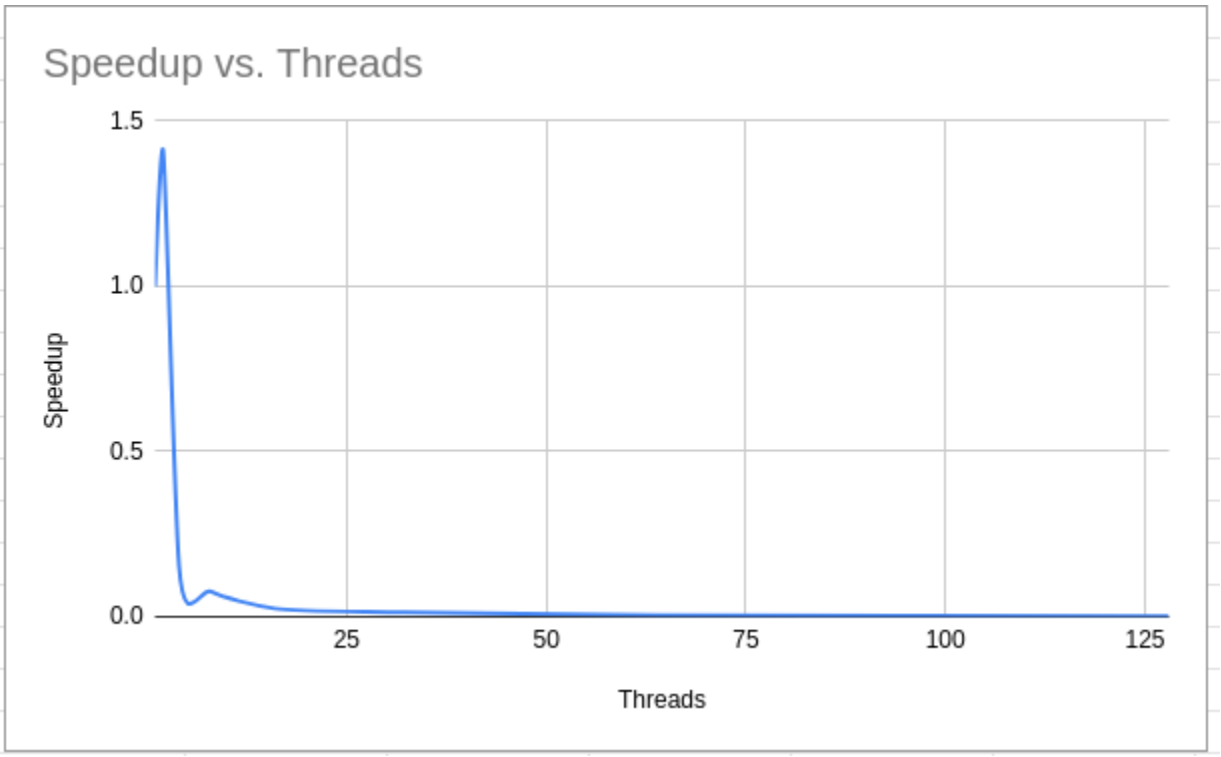
no of processors	Exec Time	Speedup	Parallelization Factor
1	0.005653	1	0
2	0.001123	1.412288513	0.2432744851
4	0.010292	0.1541002721	-3.659520807
8	0.02087	0.07599425012	-4.05296343
16	0.064126	0.02473255778	13.14417823
32	0.117006	0.01355486043	121.2904582
64	0.412533	0.003844540922	1122.806011
128	1.096704	0.001446151377	6674.741908

Threads vs Time



CED19I002
Kadali Paul Babu

Speedups vs Threads



Inferences

Since MPI is a distributed memory architecture, the communication overhead between nodes causes the parallel code to run slower compared to serial code (running in 1 node or only in master)