High Performance Computing

Lab2-OpenMP

Adithya Srinivas ced17i007

Introduction

OpenMP stands forOpenspecifications forMultiProcessing. It has been jointly defined and endorsed by a group of major computer hardware and software vendors. It is a standardized Application ProgramInterface (API) that supports multi-threaded, shared address space parallelism

OpenMP supports thread-based parallelism. It provides an explicit programming model to control the creation, communication and synchronization of multiple threads. OpenMP uses the fork-join model of parallel execution:

- All OpenMP programs begin as a single process: the master thread. The master thread executes sequentially until the first parallel region construct is encountered.
- The master thread then creates a team of parallel threads.
- The statements in the program that are enclosed by the parallel region construct are then executed in parallel among the various team threads.

When the team threads complete the statements in the parallel region construct, they synchronize and terminate, leaving only the master thread.

Question1:

To do Matrix Addition with varying the number of threads from {1, 2, 4, 6, 10, 12, 14, 16, 20, 24} and to estimate parallelisation factor.

Input:

The inputs are two double data type matrices ‘a’ and ‘b’ which are of very large size, and result is stored in ‘c’.

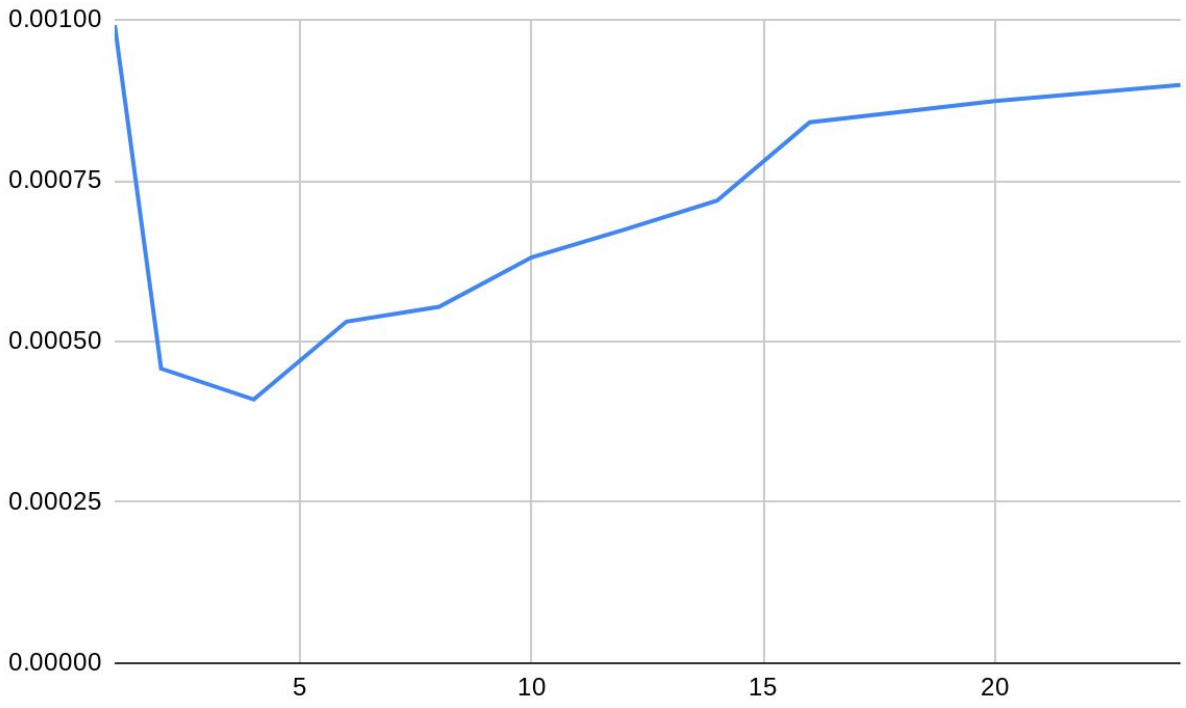
output:

So we know that T(1)/T(p)= 1/((f/p)**+(1-**f**))**

=> f= p/(p-1) * (T(1)-T(p))/T(1)

The time taken for output when thread count =1 is 0.000992 =T(1)

No of threads	**Execution time**	**Parallelization factor**
1	0.000992	
2	0.000458	0.605964
4	**0.000410**	**0.782258**
6	0.000531	0.557661
8	0.000554	0.504608
10	0.000631	0.404346
12	0.000674	0.349707
14	0.000719	0.296371
16	0.000841	0.162366
20	0.000874	0.125212
24	0.000899	0.098878



Conclusion:

As we see the time taken is least in case of 4 threads used.so **4 threads** is the optimum number of threads for this vector addition.

speedup=T(1)/T(p) = 2.419

Strategy used here is parallelization of each iteration of *for loop* in the vector addition program.

Question2:

To do Matrix multiplication (column major order) with varying the number of threads from {1, 2, 4, 6, 8,10, 12, 14, 16, 20, 24} and to estimate parallelisation factor.

Input:

The inputs are two double data type Matrices ‘a’ and ‘b’ which are of very large size and result is stored in ‘c’.

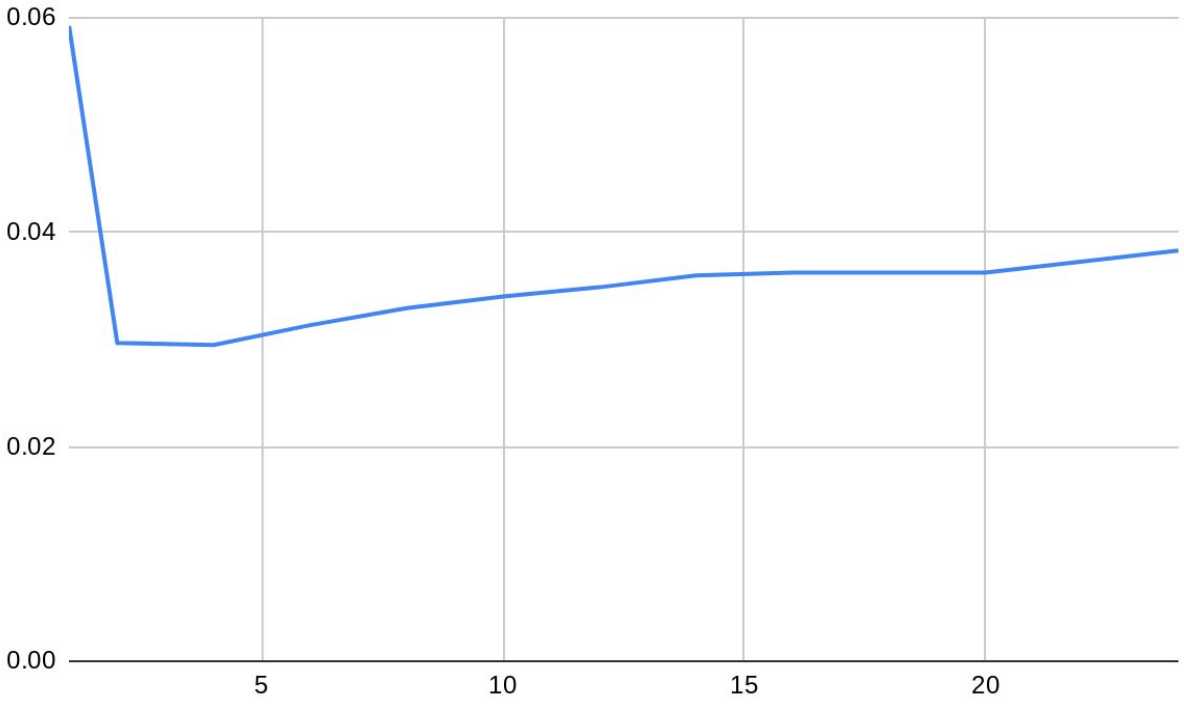
Output:

So we know that T(1)/T(p)= 1/((f/p)+(**1-**f**))**

=> f= p/(p-1) * (T(1)-T(p))/T(1)

The time taken for output when thread count =1 is 0the time taken is:: 0.059255 =T(1)

No of threads	**Time taken**	**Parallelization factor**
1	0.059255	
2	0.029675	0.998397
4	**0.029492**	**0.669716**
6	0.0.031341	0.565299
8	0.032933	0.507675
10	0.034022	0.473153
12	0.034878	0.448791
14	0.035982	0.422972
16	0.036245	0.414210
20	0.037250	0.390906
24	0.038295	0.369105



As we see the time taken is least in case of 4 threads used.so ***4 threads*** is the optimum number of threads for this vector multiplication.

speedup=T(1)/T(p) = 2.009

Strategy used here is parallelization of each iteration of *for loop* in the Matrix multiplication (column major order) program.

Question3:

To do Matrix multiplication (Blocking) with varying the number of threads from {1, 2, 4, 6, 8,10, 12, 14, 16, 20, 24} and to estimate parallelisation factor.

Input:

The inputs are two double data type Matrices ‘a’ and ‘b’ which are of very large size and result is stored in ‘c’.

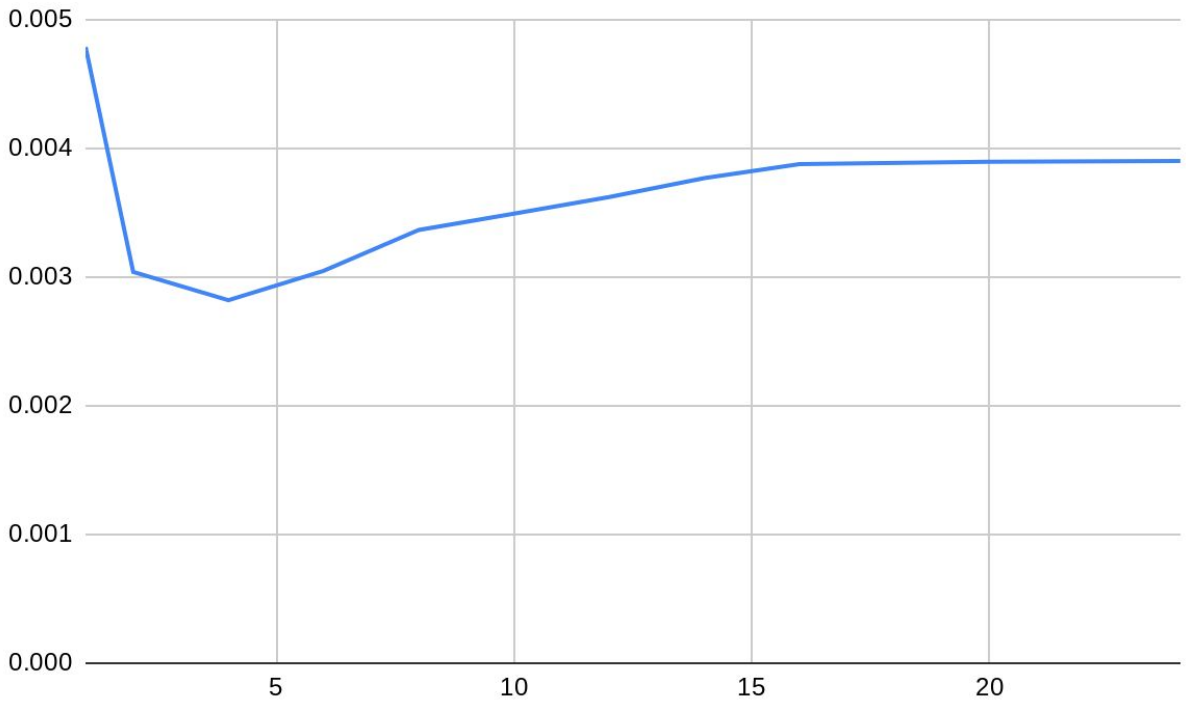
Output:

So we know that T(1)/T(p)= 1/((f/p)+(**1-**f**))**

=> f= p/(p-1) * (T(1)-T(p))/T(1)

t(1) =0.004789

No of threads	**Time taken**	**Parallelization factor**
1	0.004789	
2	0.003041	0.730006
4	**0.002821**	**0.547922**
6	0.003050	0.435749
8	0.003368	0.339110
10	0.003494	0.300457
12	0.003623	0.265609
14	0.003770	0.229147
16	0.003880	0.202464
20	0.003897	0.196063
24	0.003904	0.192833



As we see the time taken is least in case of 4 threads used.so ***4 threads*** is the optimum number of threads for this Matrix multiplication using blocking.

speedup=T(1)/T(p) = 1.697

Strategy used here is parallelization of each iteration of *for loop* in the Matrix multiplication (column major order) program.