High Performance Computing

Lab1-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:

The program is to display 'hello world' using multiple threads along with thread ids.

Question2:

To do Vector 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 Vectors(arrays) 'a' and 'b' which are of very large size, say(10,000) and result is stored in 'c'.

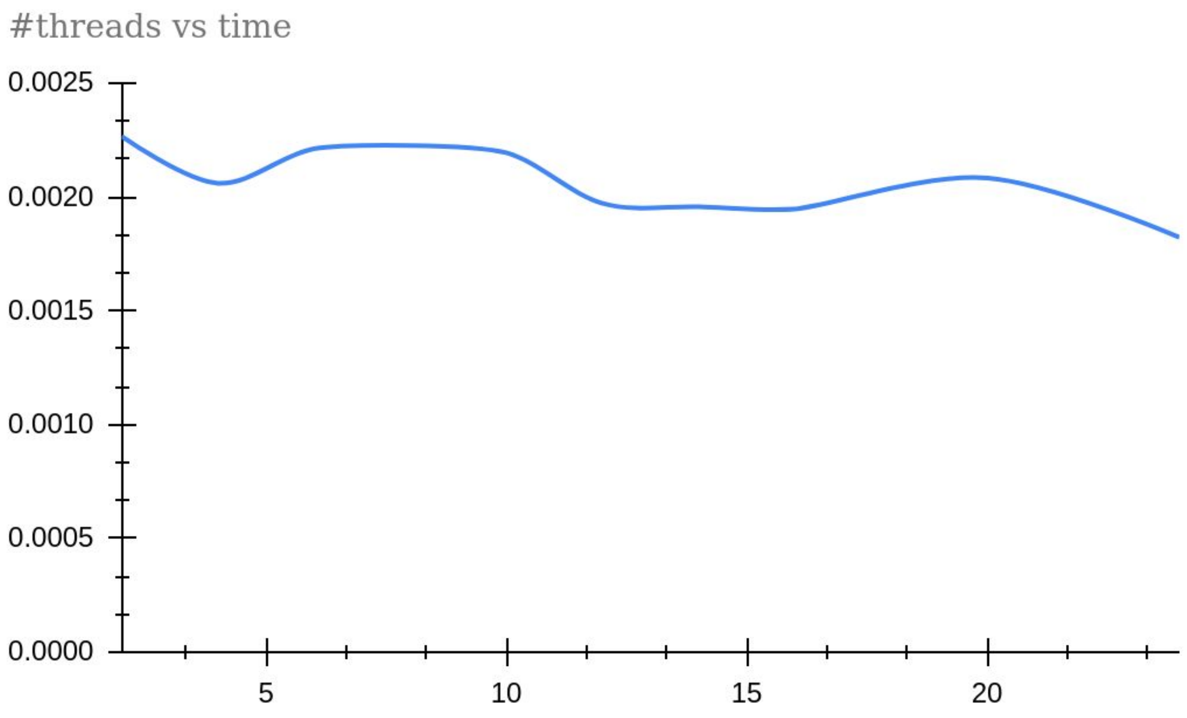
output:

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 0.003778 =T(1)

No of threads	**Execution time**	**Parallelization factor**
2	0.002267	0.799894
4	0.002061	0.605964
6	0.002213	0.497088
10	0.002195	0.465561
12	0.001973	0.521199
14	0.001958	0.518793
16	0.001948	0.518793
20	0.002084	0.471985
24	0.001823	0.467880



Conclusion:

As we see the time taken is least in case of 16 threads used.so **16 threads** is the optimum number of threads for this vector addition.

speedup=T(1)/T(p) = 1.939.

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

Question3:

To do Vector 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 Vectors(arrays) ‘a’ and ‘b’ which are of very large size, say(10,000) and result is stored in ‘c’.

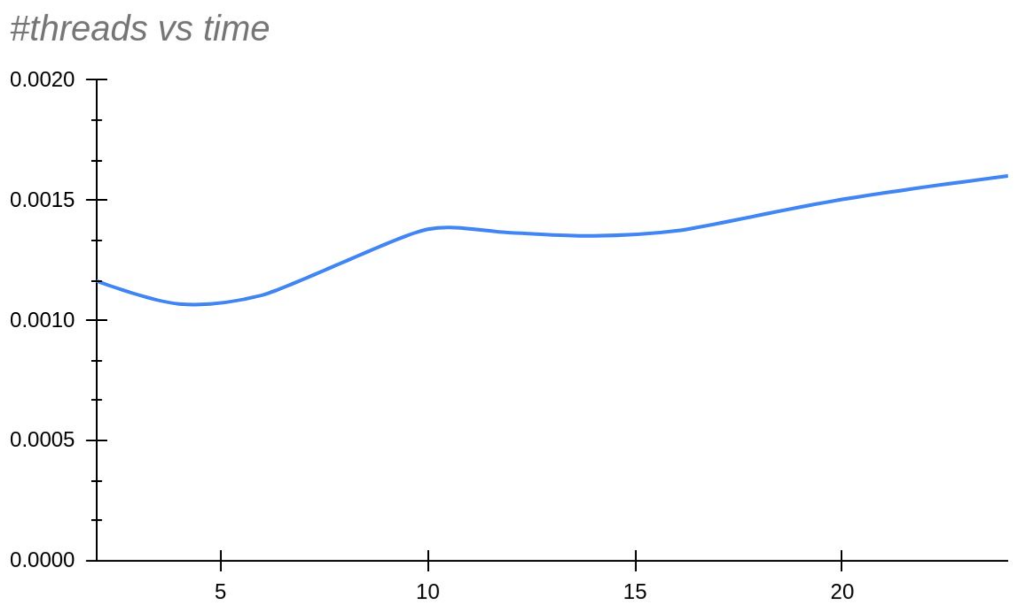
Output:

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:: 0.001625 =T(1)

No of threads	**Time taken**	**Parallelization factor**
2	0.001165	0.566154
4	0.001069	0.456205
6	0.001105	0.384000
10	0.001380	001380
12	0.001365	0.174545
14	0.001352	0.180923
16	0.001373	0.165415
20	0.001504	0.078381
24	0.001602	0.014769



As we see the time taken is least in case of 16 threads used.so ***4 threads*** is the optimum number of threads for this vector multiplication.

speedup=T(1)/T(p) = 1.520

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