High Performance Computing

Lab3-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 Addition of N-number implemented using reduction with varying the number of threads from {1, 2, 4, 6, 10, 12, 14, 16, 20, 24} and to estimate parallelisation factor.

Input:

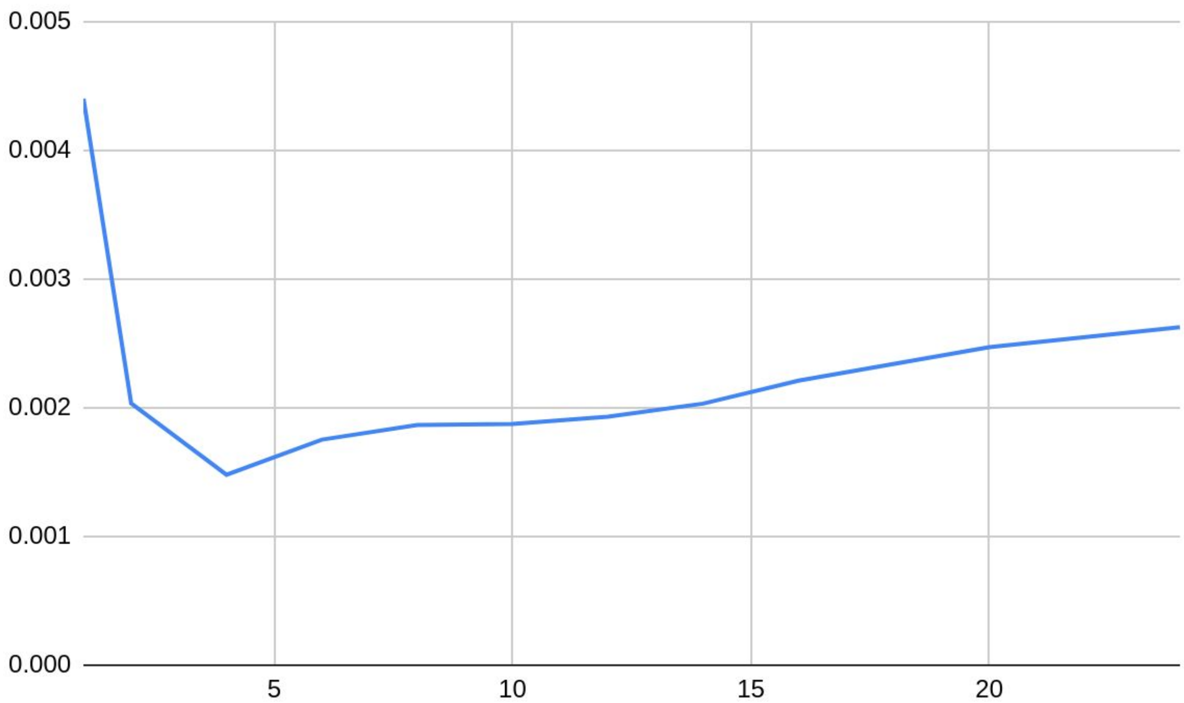
The input is one double data type matrix ‘a’ which is of very large size, and the result is stored in ‘sum’.

output:

So we know that Speed up=T(1)/T(p)

The time taken for output when thread count =1 is 0.004403 =T(1)

No of threads	**Execution time**	**speedup**
1	0.004403	
2	0.002036	2.162573674
4	**0.001480**	**2.975**
6	0.001753	2.511694238
8	0.001866	2.359592712
10	0.001874	2.349519744
12	0.001932	2.278985507
14	0.002033	2.165764879
16	0.002211	1.991406603
20	0.002472	1.781148867
24	0.002628	1.675418569



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 N-number addition implemented using reduction operation .

Parallelization factor=**T(1)/T(p)= 1/((**f/p**)+(1-**f**))**

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

=0.885

Strategy used here is parallelization of each iteration of *for loop* in the N-number addition implemented using reduction operator

Question2:

To do N-number addition implemented using critical section in openmp with varying the number of threads from {1, 2, 4, 6, 8,10, 12, 14, 16, 20, 24} and to estimate parallelisation factor.

Input:

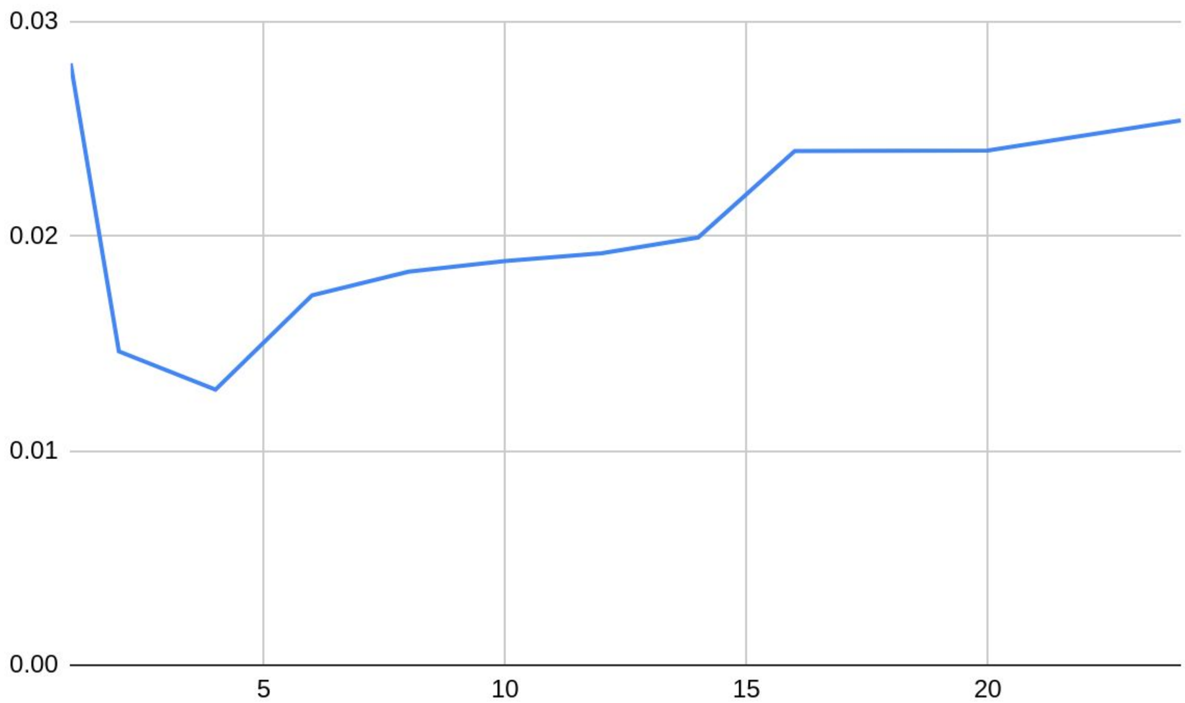
The input is one double data type Matrix ‘a’ which is of very large size and the result is stored in ‘sum’.

Output:

So we know that speedup: T(1)/T(p)

The time taken for output when thread count =1 is 0the time taken is:: 0.004123 =T(1)

No of threads	Time taken	speedup
1	0.028070	
2	0.006328	1.918398032
4	**0.008480**	**2.18477**
6	0.009248	1.627435065
8	0.010854	1.529366895
10	0.011439	1.489677864
12	0.016810	1.461218116
14	0.019946	1.407299709
16	0.023976	1.170754087
20	0.024000	1.169583333
24	0.0254070	1.104813634



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 N-number addition implemented using critical section .

Parallelization factor=**T(1)/T(p)= 1/((**f/p)+**(1-**f**))**

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

=0.930

Strategy used here is parallelization of each iteration of *for loop* in the N-number addition implemented using critical section program.

Question3:

To do Dot product 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 ‘sum’.

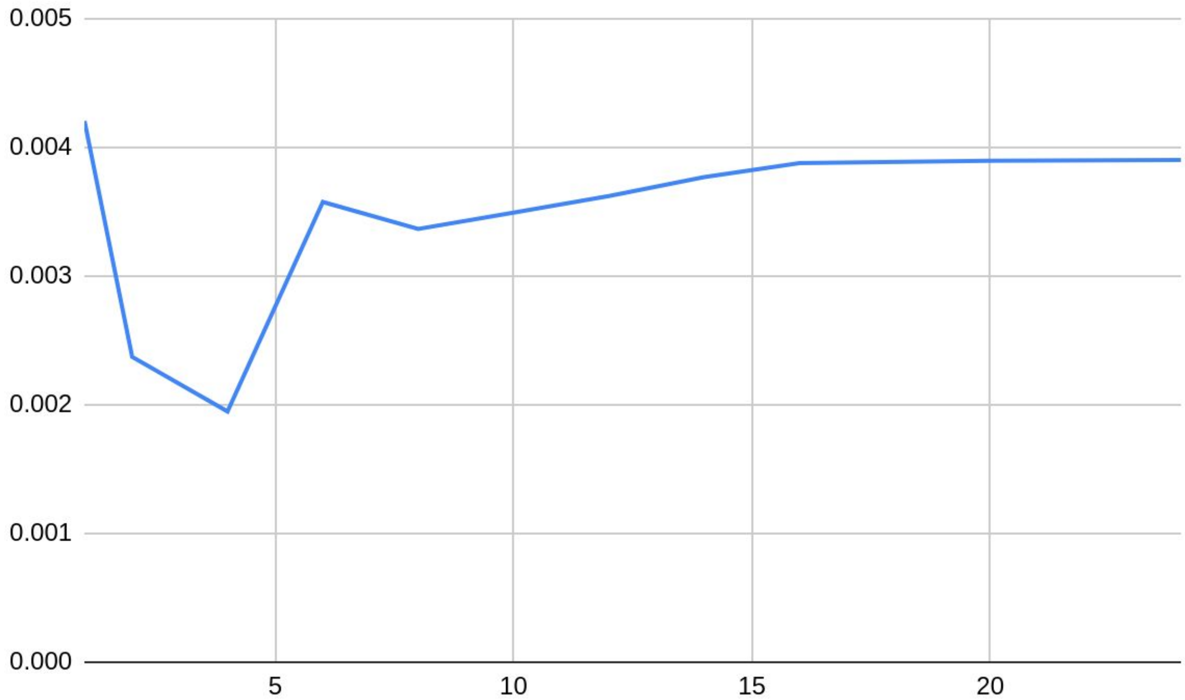
Output:

So we know that

Speed up :T(1)/T(p)

t(1) =0.004789

No of threads	Time taken	speedup
1	0.0042069	
2	0.002373	1.772819216
4	**0.001949**	**2.158491534**
6	0.003577	1.176097288
8	0.003368	1.249079572
10	0.003494	1.204035489
12	0.003623	1.161164781
14	0.00377	1.115888594
16	0.00388	1.084252577
20	0.003897	1.07952271
24	0.003904	1.07758709



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 Dot product.

Parallelization factor =**T(1)/T(p)= 1/((**f/p)+**(1-**f**))**

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

=0.715

Strategy used here is parallelization of each iteration of *for loop* in the Dot product program.

Question4:

To do Dot product using reduction 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 ‘sum’.

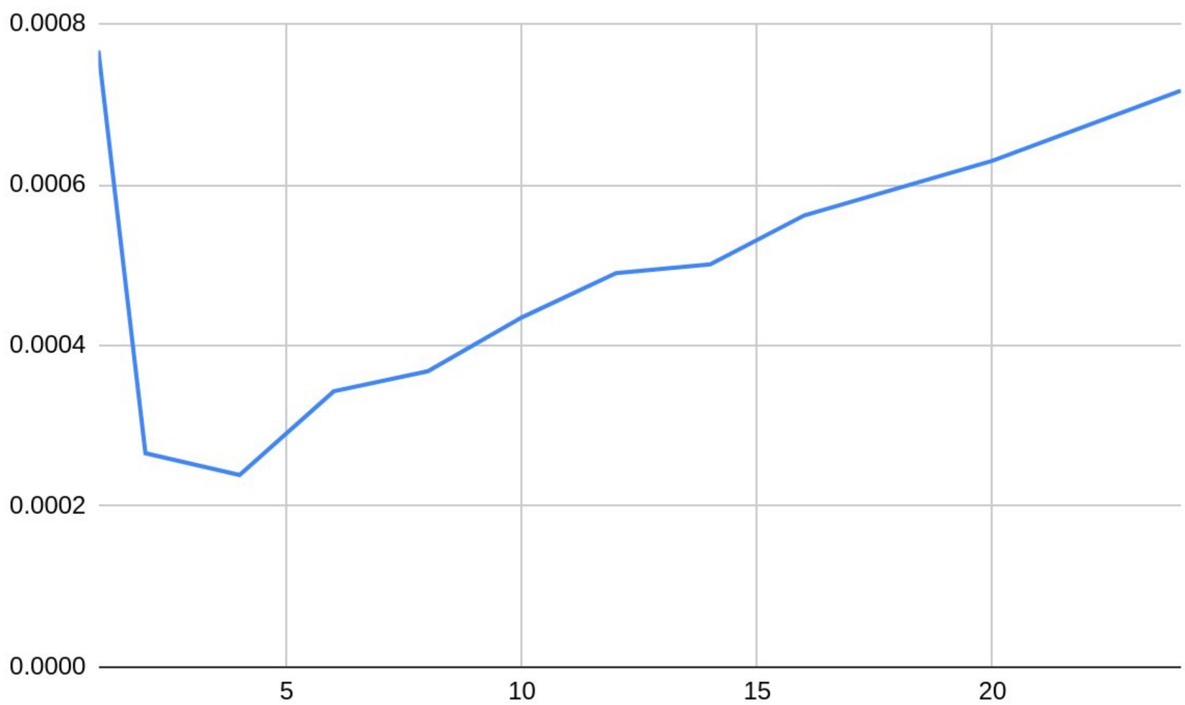
Output:

So we know that

Speed up :T(1)/T(p)

t(1) =0.004789

No of threads	Time taken	speedup
1	0.000767	
2	0.000266	2.883458647
4	**0.000239**	**3.209205021**
6	0.000343	12.83673469
8	0.000368	11.96467391
10	0.000435	1.763218391
12	0.000490	1.565306122
14	0.000501	1.530938124
16	0.000562	1.364768683
20	0.000630	1.217460317
24	0.000717	1.069735007



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 Dot product using reduction method.

Parallelization factor =**T(1)/T(p)= 1/((**f/p)**+(1-**f**))**

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

=0.917

Strategy used here is parallelization of each iteration of *for loop* in the Dot product program using reduction operation.