High Performance Computing

Lab3-OpenMP

Adithya Srinivas ced17i007

Introduction

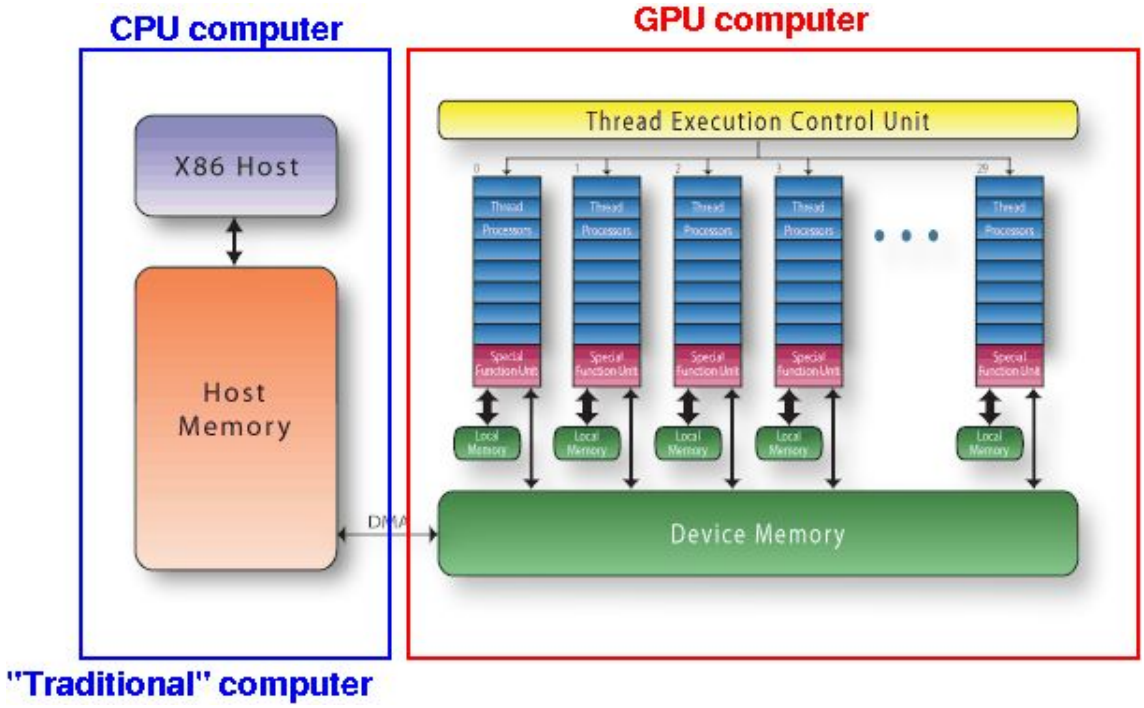
CUDA (Compute Unified Device Architecture) is a parallel computing platform and API model created by Nvidia. It allows software developers to use a CUDA-enabled Graphical processcog unit (GPU) for general purpose processing – an approach termed GPGPU (General-Purpose computing on Graphics Processing Units). The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels.

The CUDA platform is designed to work with programming languages such as C,C++ and Fortran. This accessibility makes it easier for specialists in parallel programming to use GPU resources, in contrast to prior APIs like direct3D and openGL, which required advanced skills in graphics programming.CUDA-powered GPUs also support programming frameworks such as openGCCand openCL and HIP by compiling such code to CUDA. When CUDA was first introduced by Nvidia, the name was an acronym for Compute Unified Device Architecture,but Nvidia subsequently dropped the common use of the acronym.

Architecture :

The CUDA Architecture consists of several components:

- **Parallel compute engines inside NVIDIA GPUs**
- **OS kernel-level support for hardware initialization, configuration, etc.**
- **User-mode driver, which provides a device-level API for developers**
- **PTX instruction set architecture (ISA) for parallel computing kernels and functions**



In this device its is the kernel which does the job.

- The kernel is divided into grids
- The grids are divided into blocks
- The blocks are divided into threads

Question1:

To do Addition of N-number by varying the number of threads from {1, 2, 4,8,16, 32, 64, 128,256,500} and to estimate parallelisation factor.

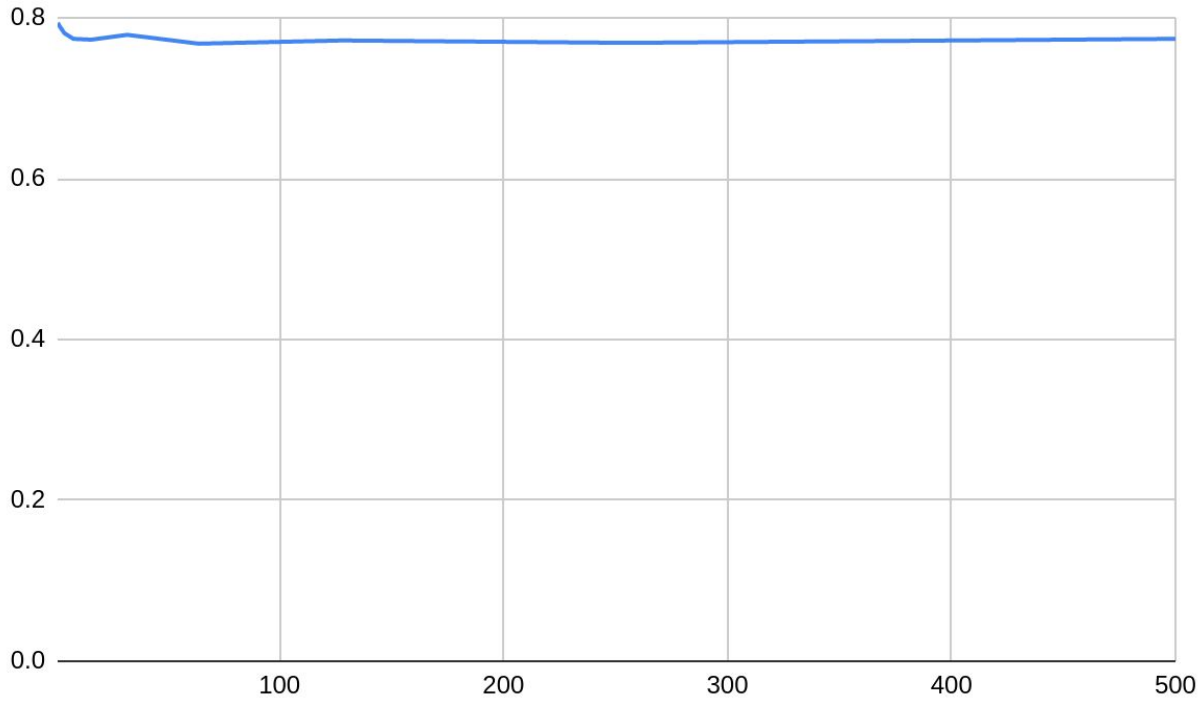
Input:

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)

No of threads	Execution time	speedup
1	0.794	1
2	0.789	1.006337136
4	0.781	1.016645327
8	0.774	1.025839793
16	0.773	1.027166882
32	0.779	1.019255456
64	**0.768**	**1.033854167**
128	0.772	1.028497409
256	0.769	1.032509753
500	0.774	1.025839793



Conclusion:

As we see the time taken is least in case of 64 threads used.so 64 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)

= 0.03326536324

Question2:

To do Dot product with varying the number of threads from {1, 2, 4, 8,16, 32, 64, 128,140} by Reduction method in MPI and to estimate parallelisation factor.

Input:

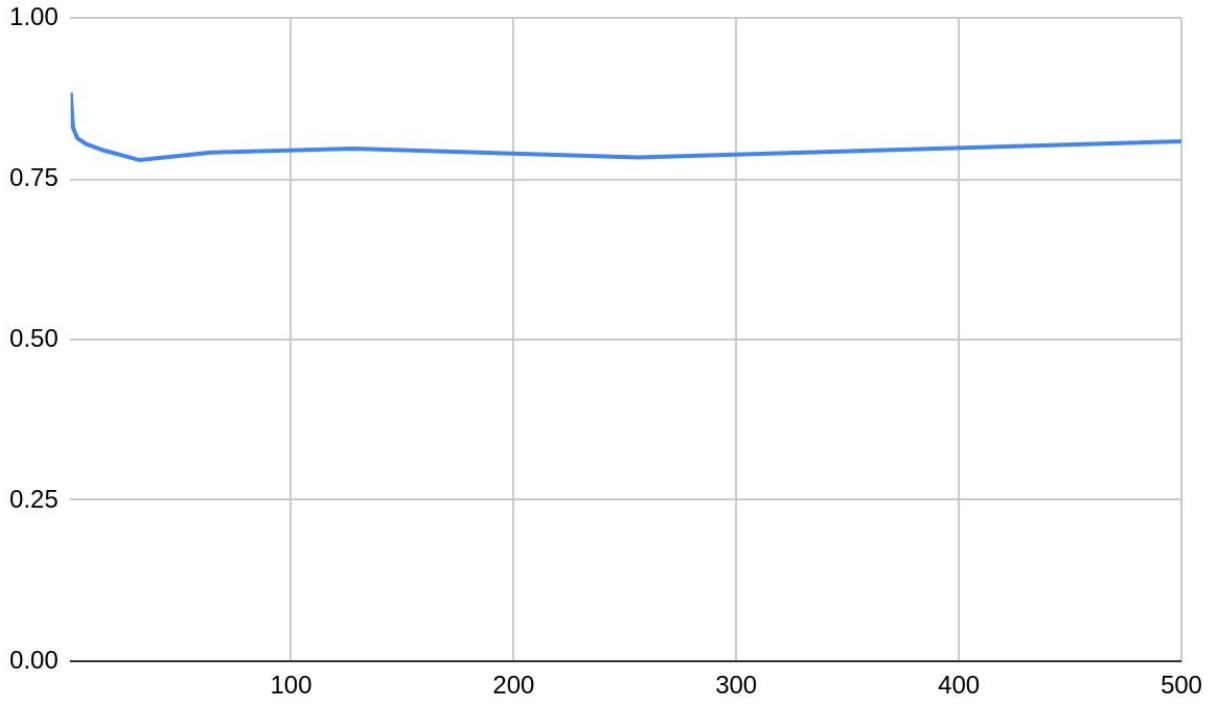
The inputs are two double data type Matrices ‘a’ and ‘b’ which are of very large size and result is stored in ‘sum’.

Output:

So we know that

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

No of threads	Time taken	speedup
1	0.884	
2	0.830	1.065060241
4	0.813	1.087330873
8	0.804	1.099502488
16	0.794	1.113350126
32	0.779	1.13478819
64	0.791	1.117572693
128	0.797	1.109159348
256	0.783	1.12899106
500	0.808	1.094059406



As we see the time taken is least in case of 32 threads used.so ***32 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)

= 0.122609838

.