High Performance Computing

Lab2-CUDA

Adithya Srinivas ced17i007

Introduction

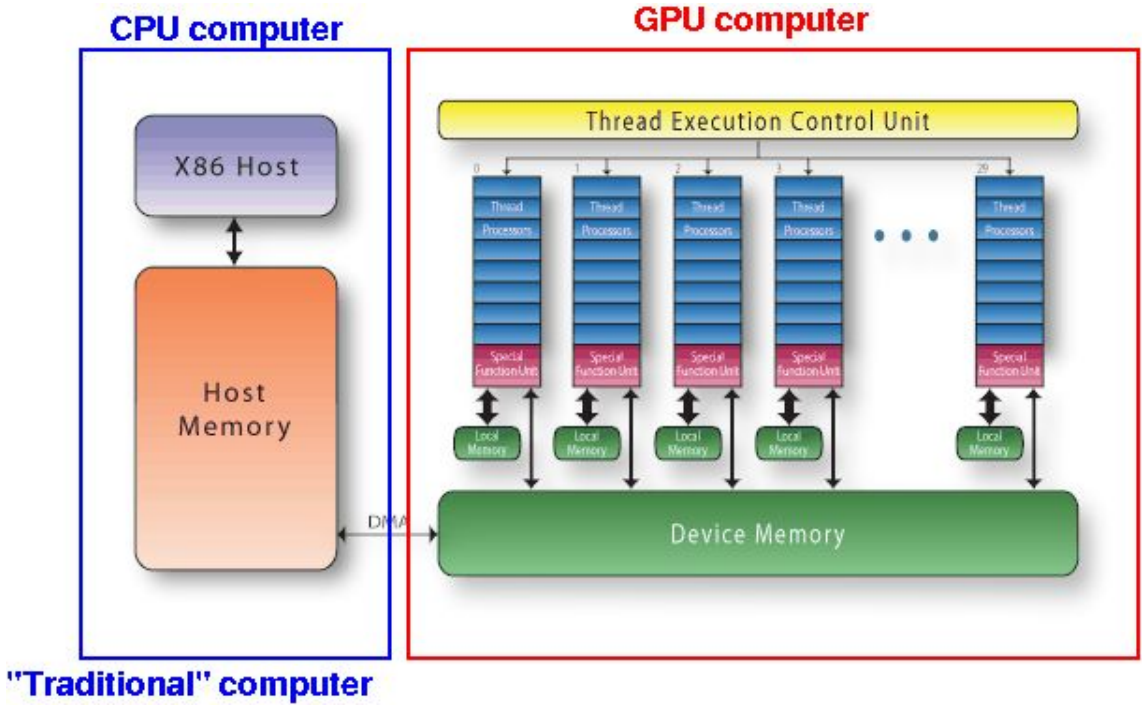
CUDA (Compute Unified Device Architecture) is a parallel computing platform and API model created by Nvidia. It allows software developers to use a CUDA-enabled Graphical processcog unit (GPU) for general purpose processing – an approach termed GPGPU (General-Purpose computing on Graphics Processing 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 Matrix Addition with varying the number of threads from {1, 2, 4,8,16, 32, 64, 128,256,500} and to estimate parallelisation factor.

Input:

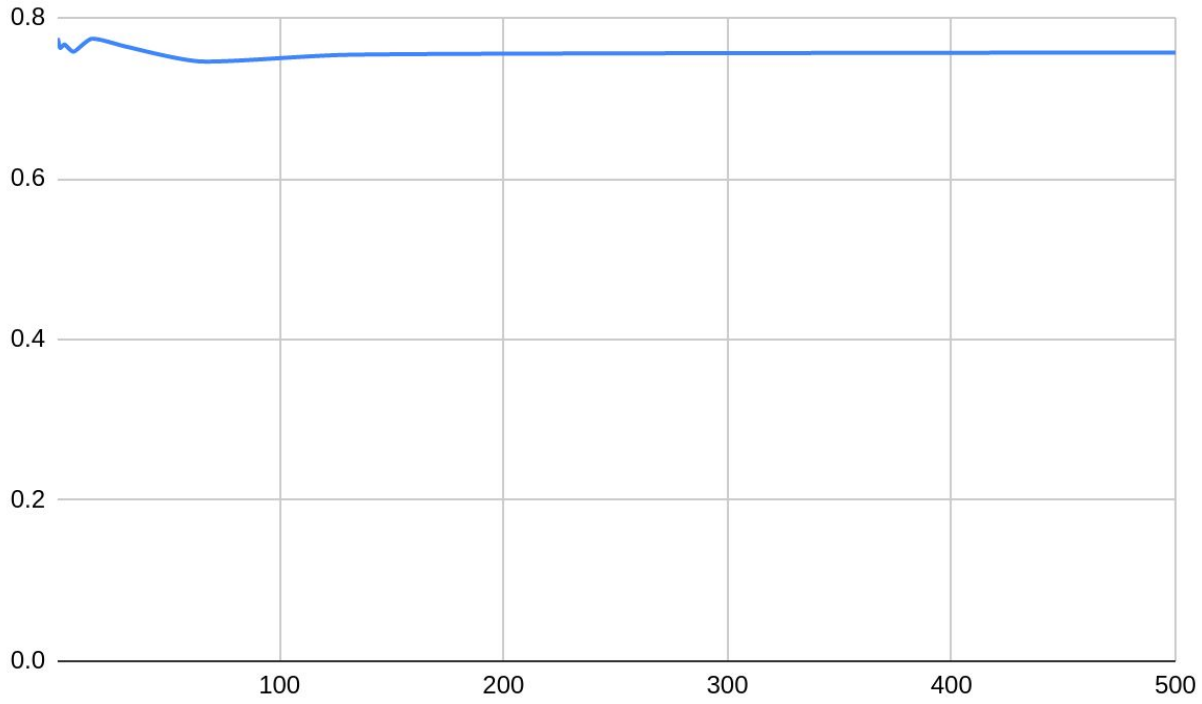
The inputs are two double data type matrices ‘a’ and ‘b’ which are of very large size, and result is stored in ‘c’.

output:

speedup=T(1)/T(p)

The time taken for output when thread count =1 is 0.775 =T(1)

No of threads	**Execution time**	**speedup**
1	0.775	1
2	0.763	1.015727392
4	0.767	1.010430248
8	0.758	1.022427441
16	0.774	1.00129199
32	0.764	1.014397906
64	**0.746**	**1.038873995**
128	0.754	1.027851459
256	0.756	1.025132275
500	0.757	1.023778071



Conclusion:

As we see the time taken is least in case of 64 threads used for this vector addition.

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

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

=0.03801331285

Question2:

To do Matrix multiplication (column major order) with varying the number of threads from {1, 2, 4,8,16, 32, 64, 128,256,500} and to estimate parallelisation factor.

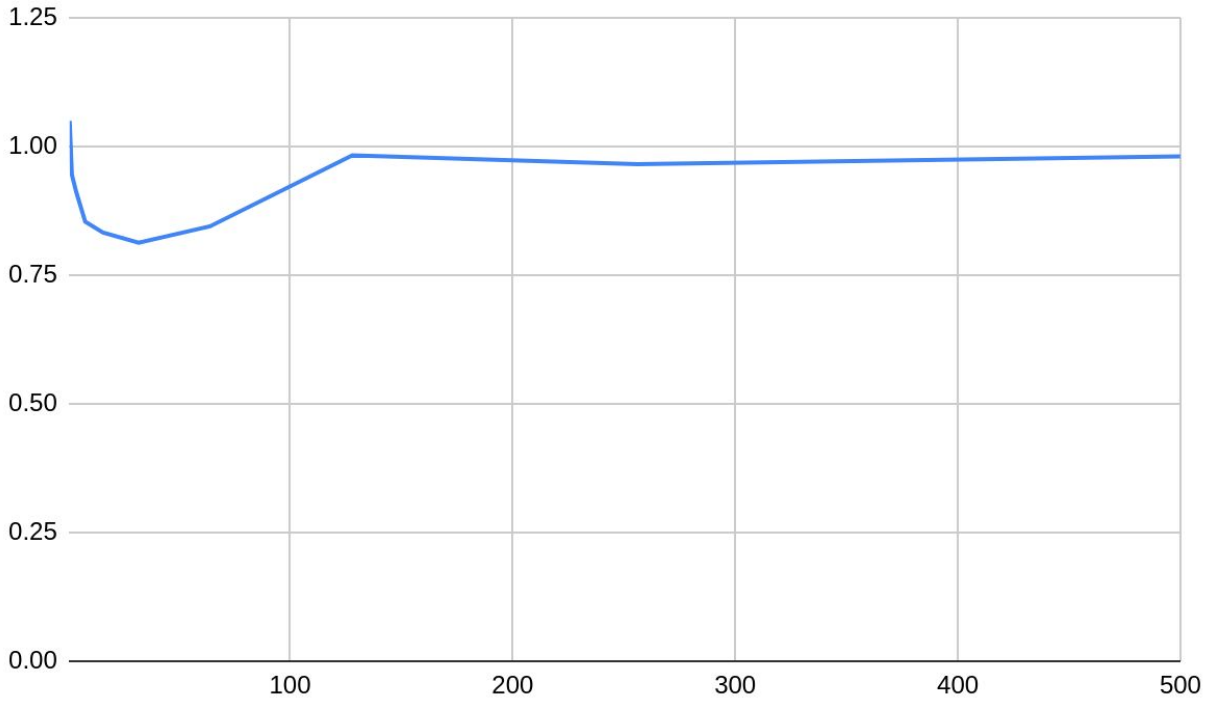
Input:

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

Output:

The time taken for output when thread count =1 is 0the time taken is:: 105 =T(1)

No of threads	**Time taken**	**speedup**
1	1.05	
2	0.945	1.111111111
4	0.911	1.152579583
8	0.854	1.229508197
16	0.833	1.260504202
32	**0.813**	**1.291512915**
64	0.845	1.24260355
128	0.983	1.068158698
256	0.966	1.086956522
500	0.981	1.070336391



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 matrix multiplication.

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

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

=0.2329953917

Question3:

To do Matrix multiplication (Blocking) with varying the number of threads from {{1, 2, 4,8,16, 32, 64, 128,256,500} and to estimate parallelisation factor.

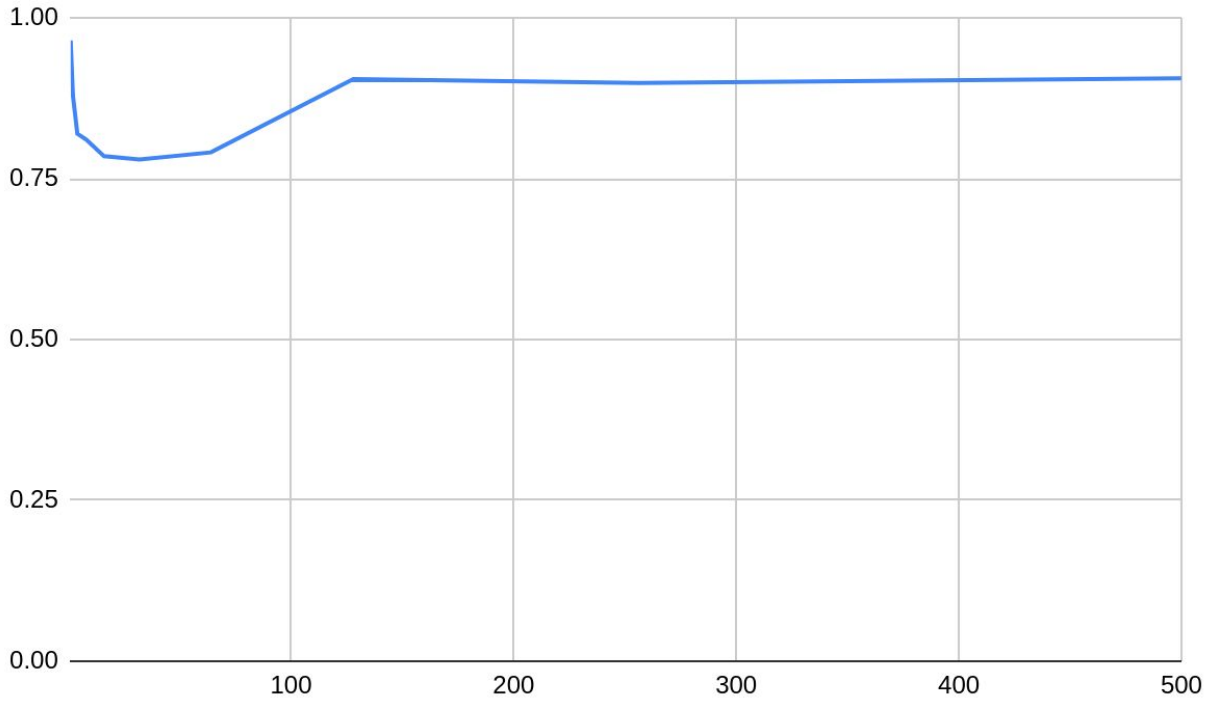
Input:

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

Output:

speedup=T(1)/T(p)

No of threads	**Time taken**	**Speed up**
1	0.965	
2	0.880	1.096590909
4	0.820	1.176829268
8	0.811	1.189889026
16	0.785	1.229299363
32	**0.780**	**1.237179487**
64	0.791	1.219974716
128	0.905	1.066298343
256	0.899	1.073414905
500	0.906	1.065121413



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 Matrix multiplication using blocking.

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

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

=0.19