Title:

The Matrix

Abstract:

“The Matrix”(Project’s Name) in CUDA is about the parallelizing of array movements of random integers in certain range which was serial in earlier implementation of the same project.

We have used this to implement an application program similar to that in the movie “Matrix” with help of open-source graphics library. Though the maximum display had its constraints, as 2-D vector of integers of size 64 X 24, this implication is not very efficient in parallel as compared to serial as the results prove. However, when the vector size grows as big as 512 X 512 significant increase in performance is observed. This is simply because for one CPU number of operations is 262144, while for GPU no of operations per block is only 512, while per thread no of operations is only 1! This is where we have utilized the advantage of GPU architecture.

We have passed the CPU generated 64 X 24 (1-D) integer values to the GPU and asked it to shift this complete 1-D vector, dividing it into 64 chunks each chunk or block handles 24 values. In each block there are defined 24 1-D threads which perform 1 shift operation. The result is then copied back to the CPU’s RAM, from where it displayed on to user screen, this cycle continues except that only 1st generation of integers was random and then on every generation is only shifting. That is why when we look at output screen of our project one feels numbers coming down and the bottom number appearing in the next column’s 1st row.

Intro/ Theory:

**CUDA** (Compute Unified Device Architecture) is a parallel computing platform and application Programming Interface(API) model created by Nvidia.It allows software developers and software engineers to use a CUDA-enabled Graphics Processing 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 OpenAcc and 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.

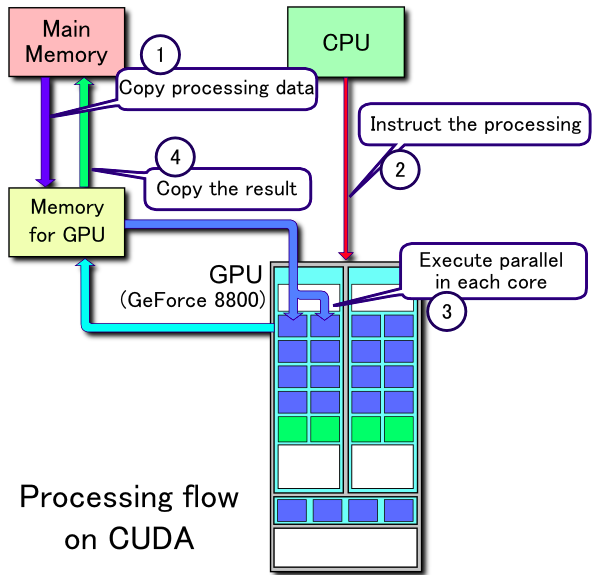
CUDA as we know, is used to take advantage of parallel programming. Operations which can be divided and performed independently (such as addition of 2 large vectors), as need of time should be given to GPU than overloading the CPU. In our project we have taken advantage of this GPU architecture by giving the task of integer shifting by 1 position to GPU than CPU. If the CPU is kept free from this task it can be kept free to do its other work. In the project while pertaining to constraints to libgraph the pixel vector we got was only 640 X 480, of which 10 pixels from x-axis were used to represent an integer and y-axis was given 20 pixels thus the number of integers at max that could be present in 1 row is 64 and in column is 24. We have used a block in GPU to do shifting of each column, thus the number of threads active per block is equal to number of rows i.e 24.

We have used only x-axis or 1-Dimension of CUDA’s architecture for sake of simplicity, this could have been also performed as 2-Dimensional project or even a 3-Dimensional, increasing complexity of computations by multiple times. In this project we haven’t taken into account the time for transferring of data from Host to Device’s Memory or vice-e-versa, as this was dependent all machine dependent.

Objective:

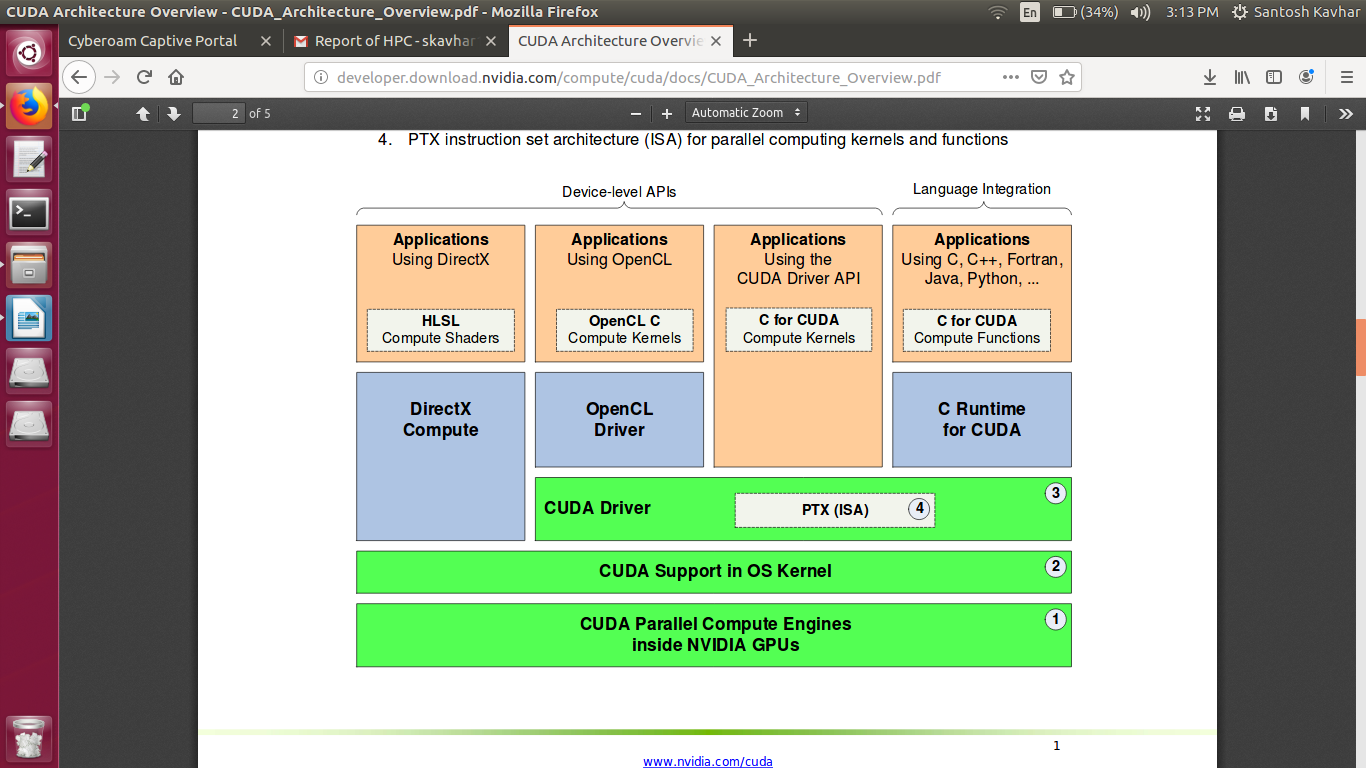
Our main objective was to reduce the burden of CPU when the shifting operations are to be performed. CPU might do dumb job of movements and this would cause high performance degradation. Such tasks are very efficiently handled by the GPU, we have run both , the serial(CPU) program and the parallel(GPU) program, though the performance of CPU is high for less size of movements, but it is not efficient when number of movements increases by higher degrees.

CUDA Flow:



The CUDA Architecture:

The CUDA Architecture consists of several components, in the green boxes below: 1. Parallel compute engines inside NVIDIA GPUs 2. OS kernel-level support for hardware initialization, configuration, etc. 3. User-mode driver, which provides a device-level API for developers 4. PTX instruction set architecture (ISA) for parallel computing kernels and functions



Future Scope:

- The model used in this project can be used for very high scale parallel data shift operations.

- The Arithmetic shift operation such as multiply by 2 or divide by 2 or any base ‘n’ is performed efficiently by any normal computer. But if the number of such multiplications or divisions is very high, this can be performed parallely and thus saving lots of computation efforts.

-Similar type of model can simulate animation actions such as falling of rain in which the rain’s pixel move from top to bottom.

Conclusion:

Thus we can conclude that whenever same type of operation is to be performed on large amount of input, GPU should be a preferred choice. And for less size input the CPU will be a better choice than GPU because CPU may simply outperform GPU with its high clock speed.

This simple implemented project proves that if the calculations to be performed however complex but same type of calculations on similar type of data but in huge amount, GPU is to be used than CPU.