Parallel Computing with CUDA

Kunal Someskar

Student, VES Institute of Technology, Mumbai, Maharashtra, India

---

Abstract - *With the change from singlecore to multicore processors basically entire, for all intents and purposes all item CPUs are currently parallel processors. Expanding parallelism, as opposed to expanding clock rate, has turned into the essential motor of processor execution development, and this pattern is probably going to proceed. This brings up numerous critical issues about how to beneficially create productive parallel projects that will scale well crosswise over progressively parallel processors.*

Key Words: CPU, GPU, Parallel Computing, Parallel Processors, CUDA, Nvidia.

1. INTRODUCTION

CUDA is a parallel figuring stage and application programming interface (API) display made by Nvidia. It permits programming designers and programming architects to utilize a CUDA-empowered illustrations handling unit (GPU) for broadly useful preparing – an approach named GPGPU (General-Purpose registering on Graphics Processing Units). The CUDA stage is a product layer that gives guide access to the GPU's virtual direction set and parallel computational components, for the execution of register portions.

The CUDA stage is intended to work with programming dialects, for example, C, C++, and Fortran. This openness makes it less demanding for authorities in parallel programming to utilize GPU assets, as opposed to earlier APIs like Direct3D and OpenGL, which required propelled aptitudes in designs programming. Likewise, CUDA bolsters programming structures, for example, OpenACC and OpenCL. When it was first presented by Nvidia, the name CUDA was an acronym for Compute Unified Device Architecture, yet Nvidia in this way dropped the utilization of the acronym.

1.1 Advantages

CUDA has a few focal points over conventional universally useful calculation on GPUs (GPGPU) utilizing designs APIs:

- Scattered reads – code can read from arbitrary addresses in memory
- Unified virtual memory (CUDA 4.0 and above)
- Unified memory (CUDA 6.0 and above)
- Shared memory – CUDA exposes a fast shared memory region that can be shared among threads. This can be used as a user-managed cache, enabling higher bandwidth than is possible using texture lookups.
- Faster downloads and read backs to and from the GPU
- Full support for integer and bitwise operations, including integer texture lookups

1.2 Disadvantages

Regardless of whether for the host PC or the GPU gadget, all CUDA source code is currently handled by C++ structure rules. This was not generally the case. Prior adaptations of CUDA depended on C sentence syntax rules. As with the broader instance of incorporating C code with a C++ compiler, it is in this manner conceivable that old C-style CUDA source code will either neglect to order or won't carry on as initially planned.

- Interoperability with rendering dialects, for example, OpenGL is one-path, with OpenGL approaching enrolled CUDA memory however CUDA not approaching OpenGL memory.
- Replicating amongst host and gadget memory may acquire an execution hit because of framework transport transmission capacity and idleness (this can be halfway mitigated with non-concurring memory exchanges, took care of by the GPU's DMA motor)
- Strings ought to keep running in gatherings of no less than 32 for best execution, with add up to number of strings numbering in the thousands. Branches in the program code don't influence execution altogether, gave that every one of 32 strings takes a similar execution way; the SIMD execution display turns into a critical restriction for any inalienably unique assignment (e.g. navigating a space apportioning information structure amid beam following).
- Dissimilar to OpenCL, CUDA-empowered GPUs are just accessible from Nvidia.
- No emulator or fallback usefulness is accessible for present day amendments.
- Substantial C++ may here and there be hailed and counteract aggregation because of the way the compiler approaches improvement for target GPU gadget limitations.

- C++ run-time compose data (RTTI) and C++-style special case taking care of are just upheld in have code, not in gadget code.

- In single accuracy on original CUDA process ability 1.x gadgets, denormal numbers are unsupported and are rather flushed to zero, and the precisions of the division and square root activities are somewhat lower than IEEE 754-agreeable single exactness math. Gadgets that help process capacity 2.0 or more help denormal numbers, and the division and square root tasks are IEEE 754 agreeable naturally. Be that as it may, clients can acquire the earlier quicker gaming-review math of register capacity 1.x gadgets if wanted by setting compiler banners to debilitate exact divisions and precise square roots, and empower flushing denormal numbers to zero.

2. CUDA Programming Model

The CUDA Programming Model is based on:

- Minimal extension of C and C++ languages.
- Write a serial program that calls parallel kernels.
- Serial portions execute on the host CPU.
- A kernel executes as parallel threads on the GPU device:
- Kernels may be simple functions or full programs. Many threads execute each kernel.

The CUDA parallel programming model stresses two key plan objectives. To start with, it intends to expand a standard successive programming dialect, particularly C/C++, with a moderate arrangement of deliberations for communicating parallelism. Extensively, this gives the software engineer a chance to center around the vital issues of parallelism—how to make effective parallel calculations—as opposed to catching with the mechanics of a new and muddled dialect. Second, it is composed for composing exceedingly versatile parallel code that can keep running crosswise over a huge number of simultaneous strings and several processor centers. This is fundamental in light of the fact that the physical parallelism of current NVIDIA GPUs ranges from eight processor centers and 768 string settings up to 240 processor centers and 30,720 string settings. The CUDA demonstrate normally controls the developer to compose parallel projects that straightforwardly and productively scale over these diverse levels of parallelism.

A CUDA program is sorted out into a have program, comprising of at least one successive strings running on the host CPU, and at least one parallel portions that are reasonable for execution on a parallel preparing gadget like the GPU. A portion executes a scalar successive program on a set of parallel strings. The software engineer sorts out these strings into a matrix of string squares. The strings of a solitary string square are permitted to synchronize with each other by means of obstructions and approach a rapid, per-square shared on-chip memory for interthread correspondence. Strings from various obstructs in a similar network can organize just through tasks in a mutual global memory space obvious to all strings. CUDA requires that string squares be autonomous, implying that a piece must execute effectively regardless of the request in which squares are run, regardless of whether all squares are executed consecutively in subjective request without acquisition. This confinement on the conditions between squares of a bit gives adaptability. It likewise infers that the requirement for global correspondence or synchronization among strings is the fundamental thought in breaking down parallel work into particular pieces.

3. CUDA Grids of Thread Blocks

In CUDA the kernel threads are organized into grids of thread blocks as shown in the figure:

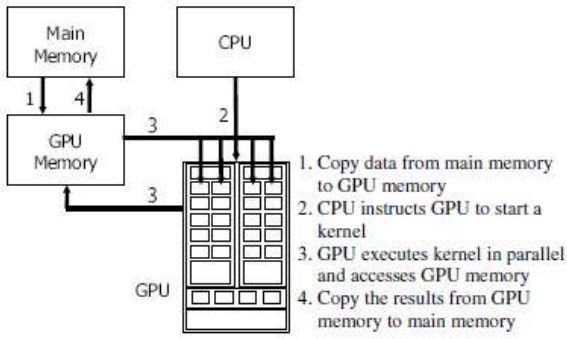

Fig -1: CUDA Threading Model

We can see that each kernel thread is assigned to each Grid in GPU. These Grids have multiple Blocks which contain the threads.

Each Block has a number of threads assigned to it. And there are multiple threads for each block.

Differences between CUDA and CPU threads:

- CUDA threads are extremely lightweight.
- CUDA uses 1000s of threads to achieve efficiency.

4. Application experience with CUDA

Numerous applications comprise of a blend of in a general sense serial control rationale and characteristically parallel calculation. Moreover, these parallel calculations are much of the time information parallel in nature. This straightforwardly coordinates the program display that CUDA embraces, to be specific a successive control string fit for propelling a progression of parallel pieces. The utilization of parallel pieces propelled from a consecutive program

moreover makes it generally simple to parallelize an application's individual parts, instead of requiring a discount modifying of the whole application.

Most Applications are accelerated using CUDA to achieve significant parallel speed-ups. CUDA is used in both academic and industrial products. Its applications are:

- Molecular Dynamics
- Fluid Dynamics
- Seismic Imaging
- Medical Imaging
- Numerical Linear Algebra

4.1 Molecular Dynamics

Molecular dynamic simulations are inherently parallel calculations and are in a perfect world suited to the CUDA programming model. Amid each time step of the reenactment, the program must ascertain the powers following up on all iotas and utilize the subsequent powers to coordinate atom positions and speeds forward to the subsequent stage. The distinctive assignments required in a period step can each be executed in a different piece. Because each atom can be handled freely of the others amid a solitary time step, it is normal to outline small to a solitary string. Therefore, the application normally gives the vast measure of fine-grained parallelism for which the GPU is planned, and a few sub-atomic dynamics10 and sub-atomic demonstrating codes have been effectively accelerated with CUDA.

4.2 Fluid Dynamics

Physical stimulations based on finite element, difference and volume are similar but not usually parallelized as molecular dynamics. However, by embracing a blocking system comparative to those utilized as a part of grid increase and picture handling, calculations of this sort can additionally, be changed into exceptionally parallel calculations.

4.3 Seismic Imaging

The Petroleum Industry uses seismic data of Earth's subsurface need CUDA for constructing images to find oil and gas. A seismic review of a forthcoming area will ordinarily comprise of many a huge number of seismic analyses. Each analyze includes an indiscreet acoustic source that creates a flag that engenders up to several kilometers through the subsurface, reflects off the interfaces between shake layers, and is recorded by a couple thousand weight touchy beneficiaries at the surface. This obtaining procedure produces terabytes of seismic information.

4.4 Medical Imaging

In Medical Field the use of imaging is very useful. Various scans like Ultrasounds and X-rays including 3D scans are popular nowadays. Unlike conventional ultrasound imaging, in which reflected ultrasound is used to form images, inverse-scattering uses ultrasound transmitted through, refracted by, and scattered by tissue or muscle to generate high-resolution speed and attenuation ofsound images.

4.5 Numerical Linear Algebra

Dense matrix-matrix multiplication, especially as given by the BLAS library

GEMM routines, is one of the essential building blocks of numerical linear algebra algorithms. It is likewise a characteristic fit for CUDA what's more, the GPU in light of the fact that it is innately parallel and can normally be shown as a blocked computation.

5. CONCLUSIONS

The Paper has been done in such a way that it explains the concept of CUDA. This paper explains the concept CUDA as well as also specifies its Applications. It gives the broad understanding on how it helps us to do parallel computing.

ACKNOWLEDGEMENT

This acknowledgement is a small effort to express my gratitude to all those who have assisted us during the course of preparing the paper.We are greatly indebted to express our pleasure and sense of gratitude towards our guide and mentor Prof. Meenakshi Garg for their constant support and valuable encouragement.

REFERENCES

1. "Nvidia CUDA Home Page"
2. Abi-Chahla, Fedy (June 18, 2008). "Nvidia's CUDA: The End of the CPU?". Tom's Hardware. Retrieved May 17, 2015.
3. Shimpi, Anand Lal; Wilson, Derek (November 8, 2006). "Nvidia's GeForce 8800 (G80): GPUs Re-architected for DirectX 10". AnandTech. Retrieved May 16, 2015.
4. PARALLEL COMPUTING EXPERIENCES WITH CUDA
5. Michael Garland,Scott Le Grand, John Nickolls
6. NVIDIA