ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 3, Issue 11, November 2016

Cuda Programming

[1] A.Naga Mani [2] S.Prathyusha [3] G.Sai Lakshmi
[1][2][3]Department of Information Technology,
Pragati Engineering College
Kakinada, India
[1] mani.anappindi1996@gmail.com [2] samatham.prathyusha95@gmail.com
[3] sailakshmi.grandhi@gmail.com

***Abstract:*-- Cuda is a technology that can make supercomputers personal. The soul of supercomputer is the body of gpu, a gpu is a specially designed processor that helps 3D or 2D graphics from restoring from the microprocessor. Cuda Architecture includes a unified shader pipeline, allowing each and every arithmetic logic unit (ALU) on the chip to be marshaled by a program intending to perform general-purpose computations. Cuda enables effective advancement in computing performance by exploiting the competency of the graphicsprocessing unit(GPU)**

***Keywords*—Parallel computing;host;device; kernel;threads**

I. INTRODUCTION

The word CUDA stands for Compute Unified DeviceArchitecture. Cuda is an application programming interface (API) model created by NVIDIA and it is a parallel computing platform. It helps software programmers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing which is an approach known as GPGPU. The CUDA platform is a software overlay that gives unambiguous access to the GPU's unacknowledged instruction set and parallel computational elements for the execution of computer kernels. The CUDA platform is designed to work with programming languages such as C, C++ and Fortran.CUDA gives developers access to the instruction set and memory of the parallel dataprocessing elements in GPUs.These features of the CUDA Architecture helps to constitute a GPU that would transcend at computating performance and graphical tasks.

II. HISTORY

A. History of CUDA's GPU

The GPU's were described as graphical accelerating supportor which supports only specific fixed function pipelines. Started around late 1990's the hardware became progressively programmable, supreme in NVIDIA's first GPU in 1999.

B. History of CUDA

In 2003, a group of research team managed by Ian Buck unveiled Brook, the first extensively followed programming model extended C programming with data-parallel design. Using these perceptions such as streams, kernels and reduction operators, the Brook compiler and runtime system disclosed the GPU to be a general-purpose processor in a high level programming language. Most eminently, Brook programs were not only simple to write than hand tuned GPU code, they were seven times faster than similar existing code. NVIDIA introduced CUDA in 2006, the world's first solution for general-computing on GPUs. NVIDIA has released a stable version of CUDA in October,2015.

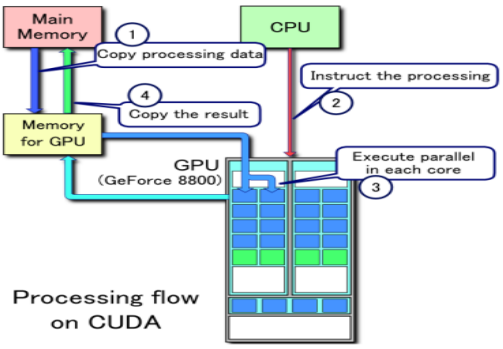
III.CUDA ARCHITECTURE

The CUDA platform is available to software programmers through CUDA-accelerated libraries, compiler directives such as OpenACC, and extensions to industry-standard programming languages including C, C++ and Fortran

Processing Flow of CUDA:

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 3, Issue 11, November 2016



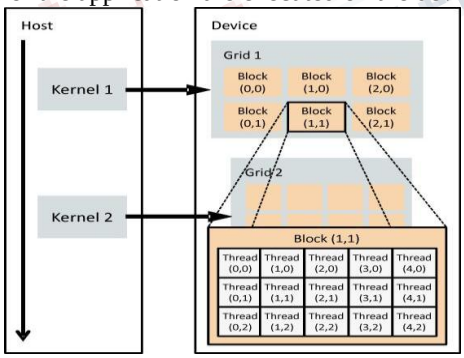
- Copies the data from main memory to GPU memory.
- CPU guides the process to GPU.
- GPU executes parallelly in each core.
- Copies the result from GPU memory to main memory.

IV.PROGRAMMING MODEL

A classic sequence of operations for a CUDA C program is:

- Declares and allocates host and device memory.
- Initializes host data.
- Transfers data from the host to the device.
- Executes one or more kernels.
- Transfers results from the device to the host.

A kernel is executed by a grid of thread blocks. Threads from different blocks cannot cooperate. Parallel portions of the application are executed on the device.



V.CUDA LIBRARY

CUDA's library consists of

A. cuBLAS: BLAS operation
B. cuFFT: FFT opeartion

A. cuBLAS:

Implementation of BLAS (Basic Linear Algebra Subprograms) on top of CUDA driver:

It allows approach to the computational resources of NVIDIA's GPU

The basic design of practicing the CUBLAS library is :

- Create matrix and vector objects in GPU memory space;
- Fill them with data;
- Call the CUBLAS functions;
- Upload the results from GPU memory space back to the host

B.cuFFT

The Fast Fourier Transform (FFT) is a divide and conquer algorithm for conveniently computing discrete Fourier transform of complex or real-valued data sets.

- CUFFT is the CUDA FFT library
- Provides a smooth interface for computing parallel FFT on an NVIDIA GPU

Allows users to leverage the floating-point power and parallelism of the GPU without having to develop a custom, GPU-based FFT implementation

NVIDIA has their own list of GPU-Accelerated Libraries. Perhaps the most well-known are cuBLAS and cuFFT due to the ubiquity of matrix computations and FFTs in scientific programming. More recently, cuDNN has gained significant traction in the machine learning community.

When it comes to pure programmer productivity, three important CUDA libraries are worth mentioning. Firstly, there is Thrust (Thrust - Parallel Algorithms Library) which comes installed with the CUDA toolkit. Thrust can be viewed as a CUDA analog of the C++ Standard Template Library (STL). It provides basic parallel algorithms and data structures in an STL-esque form. This is very useful for prototyping algorithms without having to write lots of device code and CUDA

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 3, Issue 11, November 2016

memory management boilerplate. Thrust also supports some non-GPU backends, as it is really a parallel algorithms library.

Secondly, comes CUB (CUB: Main Page), which stands for CUDA Unbound. CUB was designed specifically for CUDA applications and as a result has slightly more flexibility than Thrust. It's algorithms are aggressively tuned to provide the best performance possible. In addition to providing primitives that can called from the host, CUB also provides block-wide and warp-wide collective primitives that can be called from the device.

VI. CUDA BUILT-IN DEVICE VARIABLES

All _global_ and _device_ functions have access to these variables

Dim3 gridDim:-
Dimensions of the grid in blocks

blockDim:-
Dimensions of the block in threads

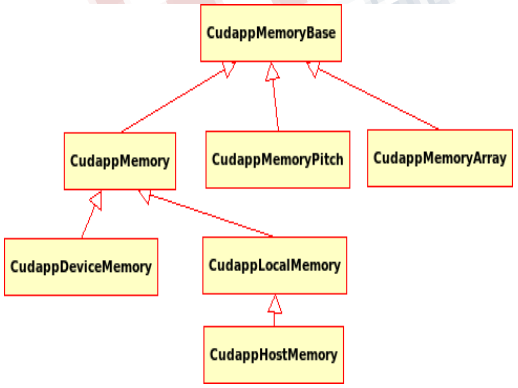
Dim3 blockIdx:-
Block index within the grid

Dim3 threadIdx:-
Thread index within the block.

IX.MEMORY ADMINISTRATION

Host and device memory are independent quantities.

♣DEVICE pointers points to GPU memory



May be passed to/from host code
May not be dereferenced in host code

♣Host pointers point to CPU memory
May be passed to/from device code
May not be dereferenced in device code

♣CUDA's API (application programming interface) for handling device memory
cudaMalloc( ), cudaFree( ), cudaMemcpy( )
Similar to the C equivalents malloc( ), free( ), memcpy( )

VII. ADVANTAGES

CUDA has many advantages over classical general purpose computation on GPUs:

- ♣ Scattered reads
- ♣ Shared memory: - CUDA exposes a fast shared memory region that can be shared amongst threads.
- ♣ Faster downloads and readbacks to and from the GPU
- ♣ Full support for integer and bitwise operations, including integer texture lookups

VIII. DISADVANTAGES

CUDA has several limitations over traditional general purpose computation on GPUs:

- ♣ Threads should be running in groups of at least 32 for best performance, with total number of threads numbering in the thousands.
- ♣ Exception handling is not supported in CUDA code due to performance overhead that would be incurred with many thousands of parallel threads running
- ♣ CUDA-enabled GPUs are only available from NVIDIA.

IX.RESTRICTIONS

Kernels are C functions with some restrictions

- ♣ Can only access GPU memory

- Must have void return type
- No variable number of arguments (“var args”)
- Not recursive
- No static variables

Function arguments automatically copied from CPU to GPU memory

X.FUTURE WORK

- Accelerated rendering of 3D graphics
- Accelerated interconversion of video file formats
- Some of the industrial applications are:
  - Bio Informatics
  - Computational Chemistry
  - Data Science
  - Computational Finance
  - Numerical Analytics

REFERENCES

[1] Abi-Chahla, Fedy (June 18, 2008). "Nvidia's CUDA: The End of the CPU?". Tom's Hardware. Retrieved May 17, 2015.. (references)

[2] Giorgos Vasiliadis; Spiros Antonatos; Michalis Polychronakis; Evangelos P. Markatos; Sotiris Ioannidis (September 2008). "Gnort: High Performance Network Intrusion Detection Using Graphics Processors". Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection (RAID).

[3] "CUDA-Enabled Products" CUDA Zone. Nvidia Corporation. Retrieved 2008-11-03.

[4] Nvidia CUDA Software Development Kit (CUDA SDK) – Release Notes Version 2.0 for MAC OS X.

[5] Manavski, Svetlin A.; Giorgio Valle (2008). "CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment"