ISSN No. 0976-5697

Volume 6, No. 8, Nov-Dec 2015

International Journal of Advanced Research in Computer Science

CASE STUDY AND REPORT

Available Online at www.ijarcs.info

Parallelism through Graphics Processor Unit using MATLAB : A Survey

Divya Kundra
Assistant Professor, Deen Dayal Upadhyaya College
Delhi, India

Abstract: While a CPU has handful number of cores, Graphics Processor Unit (GPU) has a large number of cores along with dedicated high speed memory. GPU allows hundreds of threads to run parallel on different cores thus accelerating algorithm by many folds. Many algorithms are speed up by GPU nowadays. Programs can be accelerated by GPU while using MATLAB more easily than by learning low level programming languages like C or Forton. This paper presents a survey study for using GPU through MATLAB. It discusses the high level programming language MATLAB, concept of parallelism, Parallel Computing Toolbox, GPU and the parallelism through GPU with help of MATLAB.

Keywords: Parallelism; Parallel Computing Toolbox, GPU, CPU, MATLAB Distributed Computing Server

I. INTRODUCTION

MATLAB[1] is a high level programming language used for various scientific and engineering calculations that is developed by MathWorks. It provides interactive environment for problem exploration and design, offers various mathematical functions and development tools for improving code and features for integrating program with programs written in other languages

Due to presence of microprocessors having multicore, MATLAB has evolved to provide feature of parallelism. MATLAB supports 3 kinds of parallelism: multithreaded, explicit parallelism and distributed computing. In multithreaded parallelism, some of MATLAB's inbuilt functions implicitly provide multithreading. A single MATLAB process generates multiple instruction streams. CPU cores execute these streams while sharing the same memory. In explicit parallelism, multiple instances of MATLAB run on separate processors often each with its own memory and simultaneously execute the code that externally invokes these instances. In distributed computing, multiple instances of MATLAB run independent computations on each computer, each with its own memory. In explicit parallelism, MATLAB also supports parallelism through Graphics Processor Unit (GPU). Multiple independent threads run parallel on thousands of different cores of GPU to execute the task faster. Use of Parallel Computing Toolbox is done in MATLAB to access the GPU.

A. *Parallelism*

In sequential programming, there is an ordered relationship of execution of instructions where only a single instruction executes at a particular instance of time. The program is executed over a single processor only. Serial implementation done on a single thread provides a base for evaluating comparisons from other models. In contrast to sequential processing, parallel processing lets execution of multiple tasks at the same time [1]. In parallel processing, the instructions are distributed to different processors which work simultaneously in order to complete the task fast. The ease and success of parallelism depends on how much synchronization exists between the divided tasks. Speedup will be maximum when tasks are independent i.e. there is no communication between tasks executing in parallel [2]. Parallel computing is done on multi-core computers.

[1] http://in.mathworks.com/products/matlab/?refresh=true

B. *Parallel Computing Toolbox*

Parallel Computing Toolbox[2] in MATLAB lets to solve computationally intensive and massive problems. It provides several high-level programming constructs that converts applications to take advantage of computers equipped with multicore processors and GPUs. This toolbox lets to use the full processing power of multicore computers. This can be extended to a grid or computer cluster with the help of MATLAB Distributed Computing Server toolbox.

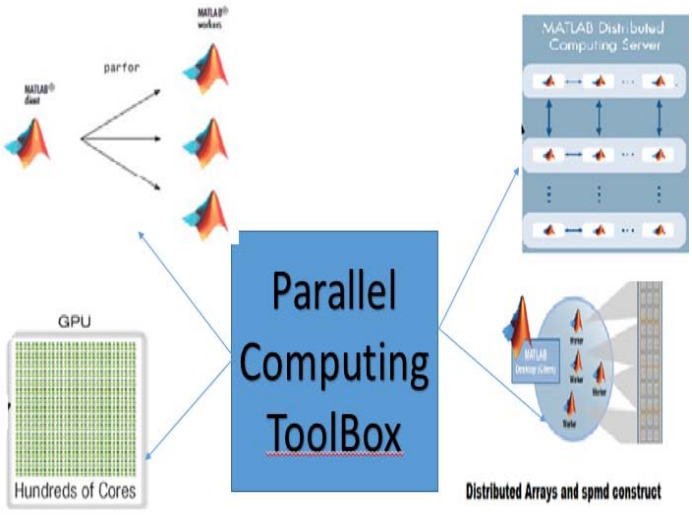

Fig1: Parallel Computing Toolbox Features

The key features provided by this toolbox as depicted in Figure 1 are :

- Parallel **for** loop (**parfor**) for running independent chunks of code on different cores of computer.
- It supports the access and transfer of data to and from NVIDIA GPU.

- Independent tasks can also be run on independent machines, cluster or on cloud by using Parallel Computing Toolbox with MATLAB Distributed Computing Server.
- It supports interactive and batch execution of parallel applications.

[2]http://in.mathworks.com/products/parallel-computing/

- There is also the provision of distributed arrays and single program multiple data (**spmd**) construct for large dataset handling and data parallel algorithms

C. *GPU*

Graphics Processing Unit (GPU) also known as visual processing unit (VPU) is a specialized electronic circuit which is designed to manipulate and alter memory to accelerate creation of images. Structure of a GPU is different from a CPU. While a CPU has a handful number of cores, GPU has a large number of cores along with dedicated high speed memory [4]. GPUs have thousands of cores to process parallel workloads efficiently. Differences between CPU and GPU architecture can be analyzed from Figure 2 [4]. CPU has larger cache with less number of CUs (Control Unit) and ALUs (Arithmetic Logic Unit) and is designed for serial processing [2]. Whereas GPU has more number of ALUs (Arithmetic Logic Unit) and CUs (Control Unit) that helps in parallel computing of large computation intensive problem [4]. GPU accelerated computing is the use of a GPU together with a CPU to accelerate scientific, analytics, engineering, consumer and enterprise applications. In GPU accelerated computing, the computational intensive tasks are offloaded to GPU by the CPU, with the remaining code is executed on CPU only.

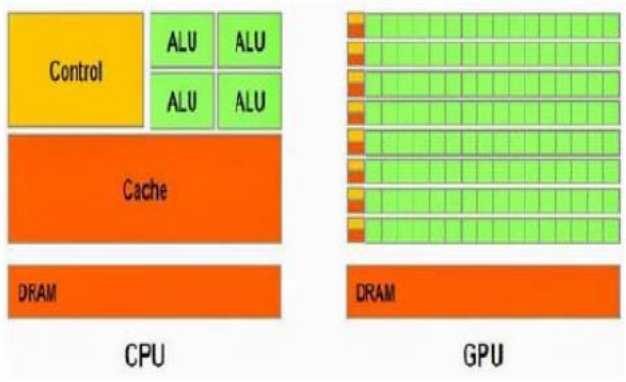

Fig2: Computational Resources of CPU and GPU [4]

The requirements of a program to execute and make use of GPU for better speed performance are that it should be computationally intensive and massively parallel [5]. GPUs perform poor when given a piece of code that involves logical branching. They are meant for doing simple scalar (addition, subtraction, multiplication, division) arithmetic tasks by hundreds of threads running in parallel [5]. Very small problems lack parallelism and thus not fit for running on GPU

While working with GPU, one bottleneck can be transferring the data to and fro from memory as there is a PCI Express (Peripheral Component Interconnect Express) bus through which GPU is connected to CPU, thus memory access is not fast when compared with CPU [6].

D. *Parallelism through GPU using MATLAB*

Programs can be accelerated by GPU while using MATLAB more easily than by using C or Forton. Even the CUDA GPU computing can be accessed using MATLAB without having to learn the intricacies of GPU architecture or low level GPU computing libraries. Parallel Computing Toolbox provides straightforward way to speed up MATLAB code by executing it on a GPU.

The general system requirement to access GPU using MATLAB are:

- CUDA-enabled NVIDIA GPU with compute capability 2.0 or higher. For releases 14a and earlier, compute capability 1.3 is sufficient.
- Latest CUDA driver

If these system requirements are fulfilled, then the MATLAB command **gpuDevice** would return the GPU details that is accessible to MATLAB as shown in Figure 3.

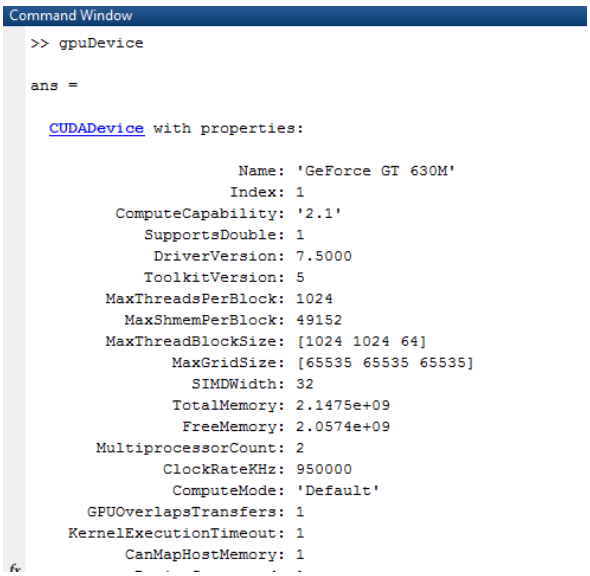

Fig3: **gpuDevice** command in MATLAB

The data type that works with GPU is **gpuArray**. An array is created on GPU using **gpuArray**. In Equation 1 **gpuArray** copies numeric array B to A and returns a **gpuArray** object. The results are brought back to CPU in the MATLAB workspace through gather Equation 2.

A = gpuArray(B) (1)
B = gather(A) (2)

GPUs can be used with MATLAB in following manners[1] :

(a) Calling some of the GPU offloaded functions available in MATLAB like **fft**, **filter** etc.

[1]http://in.mathworks.com/discovery/matlab-gpu.html

Many MATLAB build in functions support datatype that creates array on GPU (**gpuArray**). Whenever any of these functions is called with at least one **gpuArray** as an input argument, the function executes on the GPU and generates a **gpuArray** as the result. Inputs using both **gpuArray** and MATLAB arrays in the same function call can be mixed. MATLAB arrays of type gpuArray are transferred to the GPU for the function execution. Supporting functions include the discrete Fourier transform (**fft**), matrix multiplication (**mtimes**), and left matrix division (**mldivide**).

(b) By performing element wise operations through MATLAB functions like **arrayfun**, **bsxfun** etc.

arrayfun applies the function specified in function handle 'func' to each element of equal sized arrays. Execution of 'func' happens on different cores at the same time. Different threads execute 'func' at the same time on different array elements. The order of execution of function on the elements is not specified, thus tasks should be independent of each other. In Equation 3 function 'func' is applied to corresponding cell elements of equal sized arrays A1, A2,.., An and the respective result is returned in B1,B2,…,Bm. The mechanism of **arrayfun** is shown in Figure 4 where independent threads apply function 'func' on corresponding elements of 2 arrays A1 and A2.

[B1,...,Bm] = arrayfun (func,A1,...,An) (3)

arrayfun is overloaded to accept arguments of **gpuArray** also. So when **gpuArray** type arguments are passed into **arrayfun**, it executes on GPU instead of CPU.

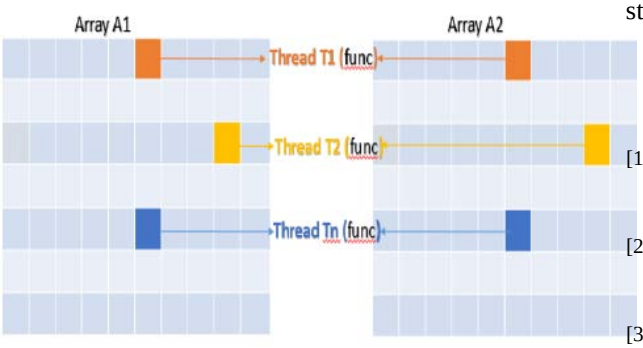

Fig 4: **arrayfun** mechanism

(c) By creating and running the kernal of available CUDA file from MATLAB.

The kernel is represented in MATLAB by a CUDAKernal object, which can operate on MATLAB array or **gpuArray** variables.

The CUDAKernal general workflow can be described as:

1. Use compiled PTX code to create a CUDAKernal object, which contains the GPU executable object.
2. Set properties on the CUDAKernal object to control its execution on the GPU.
3. Call **feval** on the CUDAKernal with the required inputs, to run the kernel on GPU.

An application will only be accelerated by GPU using MATLAB, if it fulfils the following 2 conditions:

- **Computationally intensive**- The problem should involve number of computations such that time spent on computation significantly exceeds time spend on transferring data to and from GPU.

- **Massively Parallel**- The nature of problem should be such that it should be broken into hundreds or thousands of independent units of work.

Thus we see that MATLAB provides a black box for doing parallelization. It internally calls the subroutines written in C, C++ hiding from programmer all details about data parallelism exploration. Writing directly into C, C++ can though be cumbersome for user but it will let user to have a better control over the parallelization and can help in obtaining a better speedup. Also the supported GPU parallelism through MATLAB is suitable only while working on large matrices, for rest of the cases GPU might incur high time than CPU.

II. CONCLUSION

This paper gives the review about the MATLAB supported GPU parallelism. It discusses the high level programming language MATLAB, concept of parallelism, Parallel Computing Toolbox, GPU and parallelism through GPU with help of MATLAB. Any novice user having no prior experience in performing GPU parallelization using MATLAB can make use of this study to get some useful insights about starting with the same.

III. REFERENCES

[1] Vipin Kumar. Introduction to Parallel Computing. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[2] G. S. Almasi and A. Gottlieb. Highly Parallel Computing. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1989

[3] Desmond J. Higham and Nicholas J. Higham, Matlab Guide. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2005

[4] V.H. Naik and C.S. Kusur. Analysis of performance enhancementon graphic processor based heterogeneous architecture: A CUDA and MATLAB experiment. In Parallel Computing Technologies (PARCOMPTECH), 2015 National Conference on, pages 1-5, Feb 2015.

[5] Jung W. Suh and Youngmin Kim. Accelerating MATLAB with GPU Computing: A Primer with Examples. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2013.

[6] Ravi Budruk, Don Anderson, and Ed Solari. PCI Express System Architecture. Pearson Education, 2003.