ISSN:2229-6093

Measurement of real time information using GPU

Pooja Sharma
M. Tech Scholar, Department of Electronics and Communication
AFSET, Faridabad, Haryana, India
E-mail: poojachaturvedi1985@gmail.com

Rajni Billa
M. Tech Scholar, Department of Electronics and Communication
AFSET, Faridabad, Haryana, India
E-mail: rajnibilla@gmail.com

Javed Ashraf
Asst. Prof., Department of Electronics and Communication
AFSET, Faridabad, Haryana, India
E-mail: jashraf.jmi@gmail.com

Abstract

Parallel programming is about performance, for otherwise we'd write a sequential program. A problem is where to find truly parallel hardware that can be dedicated to the task, One answer is graphical processing units (GPUs), which can have hundreds of cores and are found in millions of desktop and laptop computers. GPU computing with Open CL is a new approach to computing where hundreds of on-chip processor cores simultaneously communicate and cooperate to solve complex computing problems, transforming the GPU into a massively parallel processor. The NVIDIA C-compiler for the GPU provides a complete development environment that gives developers the tools they need to solve new problems in computation-intensive applications such as product design, data analysis, technical computing, and game physics. In this paper we are going to analyze the speed of GPU in comparison to CPU.

.

1. Introduction

Microprocessors based on a single central processing unit (CPU), such as those in the Intel_ Pentium_ family and the AMD_Opteron_family, drove rapid performance increases and cost reductions in computer applications for more than two decades. These microprocessors brought giga (billion) floating-point operations per second (GFLOPS) to the desktop and hundreds of GFLOPS to cluster servers. This relentless drive of performance improvement has allowed application software to provide more functionality, have better user interfaces, and generate more useful results [1].

Traditionally, the vast majority of software applications are written as sequential programs, as described by von Neumann [2] in his seminal report. The execution of these programs can be understood by a human sequentially stepping through the code. A sequential program will only run on one of the processor cores, which will not become significantly faster than those in use today. Without performance improvement, application developers will no longer be able to introduce new features and capabilities into their software as new microprocessors are introduced, thus reducing the growth opportunities of the entire computer industry. Rather, the applications software that will continue to enjoy performance improvement with each new generation of microprocessors will be parallel programs, in which multiple threads of execution cooperate to complete the work faster. This new, dramatically escalated incentive for parallel program development has been referred to as the concurrency revolution [3]. The practice of parallel programming is by no means new. The high-performance computing community has been developing parallel programs for decades. These programs run on large-scale, expensive computers. Only a few elite applications can justify the use of these expensive computers, thus limiting the practice of parallel programming to a small number of application developers. Now that all new microprocessors are parallel computers, the number of applications that must be developed as parallel programs has increased dramatically.

2. GPUs as Parallel Computers

Since 2003, the semiconductor industry has settled on two main trajectories for designing microprocessor [4]. The multicore trajectory seeks to maintain the execution speed of sequential programs while moving into multiple cores. The multicores began as two-core processors, with the number of cores approximately doubling with each semiconductor process generation. A current exemplar is the recent Intel_ Core_i7 microprocessor, which has four processor cores, each of which is an out-of-order, multiple instruction issue processor implementing the full x86 instruction set; the microprocessor supports hyper threading with two hardware threads and is designed to maximize the execution speed of sequential programs. In contrast, the many-core trajectory focuses more on the execution throughput of parallel applications. The many-cores began as a large number of much smaller cores, and, once again, the number of cores doubles with each generation.
A current exemplar is the NVIDIA_ GeForce_ GTX 280 graphics processing unit (GPU) with 240 cores,

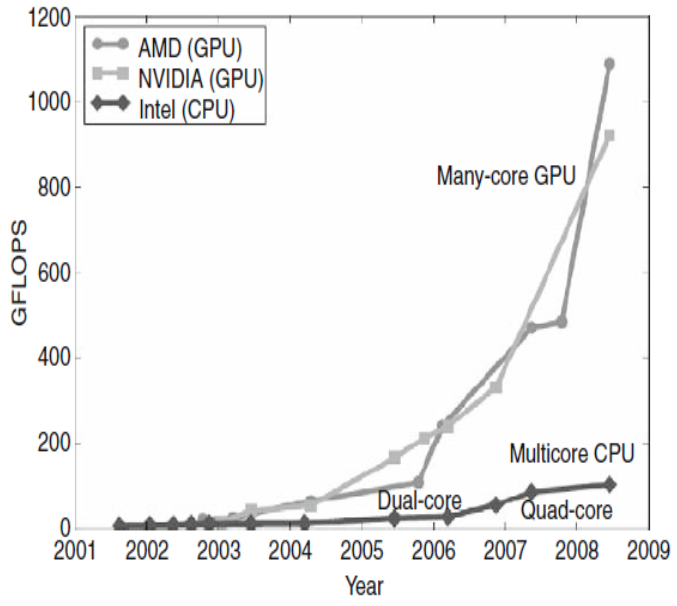


Figure 1. Enlarging performance gap between GPUs and CPUs

each of which is a heavily multithreaded, in-order, single-instruction issue processor that shares its control

and instruction cache with seven other cores. Many-core processors, especially the GPUs, have led the race of floating-point performance since 2003. This phenomenon is illustrated in Figure 1. While the performance improvement of general-purpose microprocessors has slowed significantly, the GPUs have continued to improve relentlessly. As of 2009, the ratio between many-core GPUs and multicore CPUs for peak floating-point calculation throughput is about 10 to 1. These are not necessarily achievable application speeds but are merely the raw speed that the execution resources can potentially support in these chips: 1 teraflops (1000 gigaflops) versus 100 gigaflops in 2009.

2.1. Differences between CPU and GPU

In Comparison to CPUs, modern GPUs contain hundreds of arithmetic units, and their power can be used to accelerate a lot of compute-intensive applications. The existing generation of GPU has a flexible architecture, whereas the architecture of a CPU is bound. Most of the transistors in CPU are dedicated for data caching and controlling, in contrary to GPU where most of transistor power is dedicated to computing.

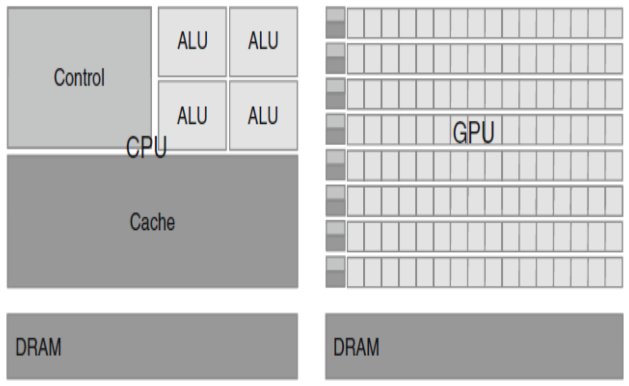


Figure 2. CPUs and GPUs have fundamentally different design philosophies

3. Goals of Parallel Computing

The first goal is to solve a given problem in less time. An investment firm, for example, may need to run a financial portfolio sce- nari risk analysis package on all of its portfolios during after-trading hours. Such

ISSN:2229-6093

an analysis may require 200 hours on a sequential computer; however, the portfolio management process may require that analysis be completed in 4 hours in order to make timely decisions based on the information. Using parallel computing may speed up the analysis and allow it to complete within the required time window. The second goal of using parallel computing is to solve bigger problems within a given amount of time. The third goal of using parallel computing is to achieve better solutions for a given problem and a given amount of time.

3.1. Task Decomposition

Task decomposition reduces an algorithm to functionally independent parts. Tasks may have dependencies on other tasks e.g. If the input of task B is dependent on the output of task A, then task B is dependent on task A.Tasks that don't have dependencies (or whose dependencies are completed) can be executed at any time to achieve parallelism. Task dependency graphs are used to describe the relationship between tasks.

3.2. Data Decomposition

For most scientific and engineering applications, data is decomposed based on the output data. Each output pixel of an image convolution is obtained by applying a filter to a region of input pixels. Each output element of a matrix multiplication is obtained by multiplying a row by a column of the input matrices. Input data decomposition is similar, e.g. a histogram is created by placing each input datum into one of a fixed number of bins.

4. Simulation Environment

To compare the speed of CPU and GPU, entropy of various signals has been calculated. AMD ATI Fire Pro V8800 Graphic Card has been used for Graphic Processing. The Simulation for the speed measurements has been done using Open CL language in Visual Studio using AMD Accelerated Parallel Processing (APP) SDK V 2.4.

4.1. Open CL

Traditional C and C++ only target traditional CPUs. The same holds true for Cray's proprietary Chapel language and the Cray Assembly Language (CAL). Nvidia's CUDA (Compute Unified Device Architecture) can be used to program Nvidia's GPUs, but not CPUs. The answer is OpenCL (Open Computing Language). OpenCL routines[5] can be executed on GPUs and CPUs from major manufacturers like AMD, Nvidia, and Intel, and will even run on Sony's PlayStation 3. OpenCL is *nonproprietary*—it's based on a public standard, and we can freely download all the development tools we need. When we code routines in OpenCL, we don't have to worry about which company designed the processor or how many cores it contains. Our code will compile and execute on AMD's latest Fusion processors, Intel's Core processors, Nvidia's Fermi processors, and IBM's Cell Broadband Engine.

OpenCL™ (**Open C**omputing **L**anguage)[6] is the first truly open and royalty-free programming standard for general-purpose computations on heterogeneous systems. OpenCL™ allows programmers to preserve their expensive source code investment and easily target multi-core CPUs, GPUs, and the new APUs.

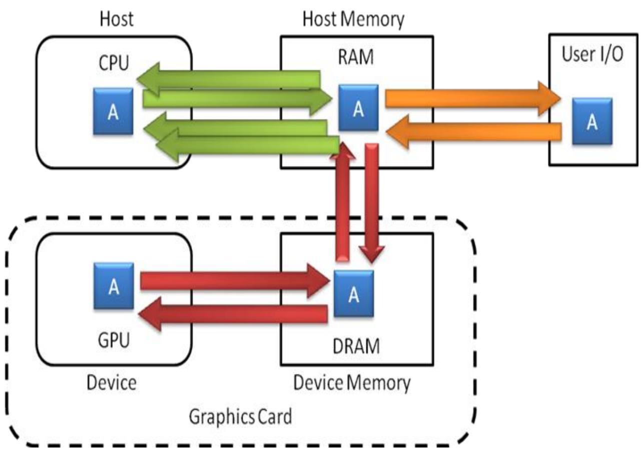


Figure 3. Open Cl Programming model

4.2. AMD Accelerated Parallel Processing (APP) SDK V 2.4

AMD APP technology [7] is a set of advanced hardware and software technologies that enable AMD graphics processing cores (GPU), working in concert with the system's x86 cores (CPU), to accelerate many applications beyond just graphics. This enables better balanced platforms capable of running demanding computing tasks faster than ever, and sets software developers on the path to optimize for AMD Accelerated Processing Units (APUs).

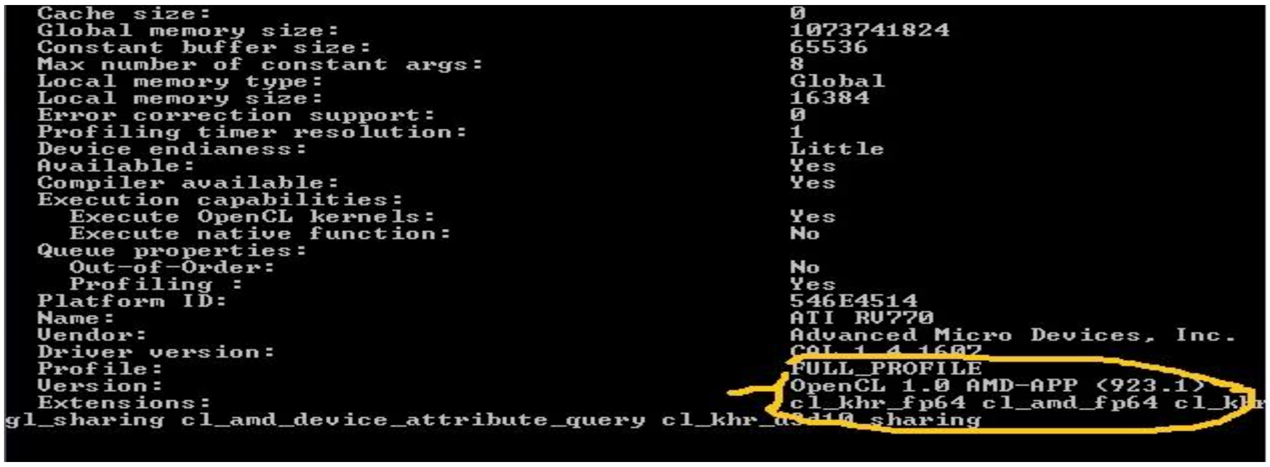
APP technology is a set of advanced hardware & software technologies enabling AMD GPU, working in concert with the system's x86 cores (CPU). It accelerate many applications beyond just graphics. The APP SDK is a complete development platform created by AMD which allow us to quickly and easily develop applications accelerated by AMD APP technology. The SDK allows us to develop our applications in a high-level language, OpenCL™ (Open Computing Language).

ISSN:2229-6093

5. Results

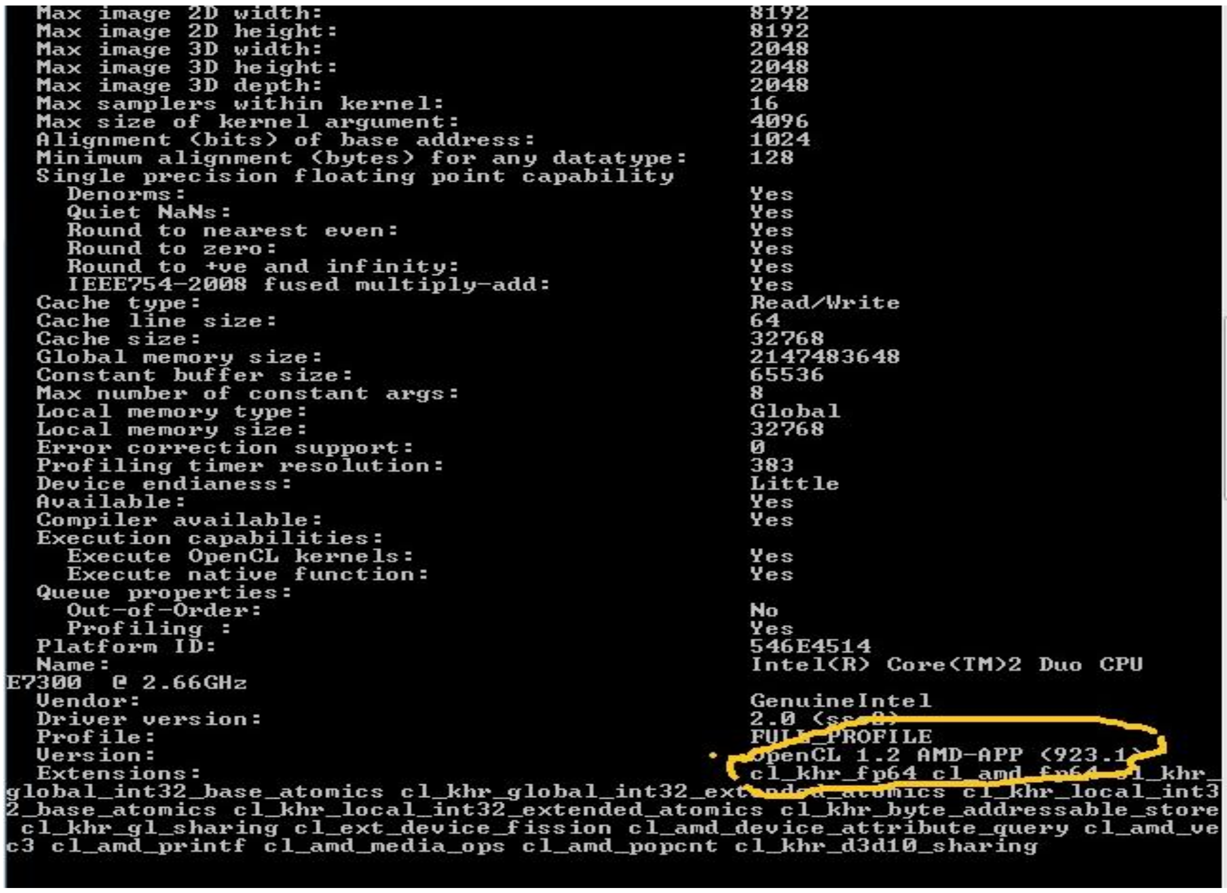
Figure 4 and 5 shows the input information for CPU and GPU respectively. Figure 6 shows the speed up of GPU over CPU over varying no. of users. It is observed that as the no. of users increases the GPU speeds up or in other words that efficiency of GPU increases.

So for larger applications where CPU becomes less promising, GPU becomes more efficient. GPU is efficient for large set of input processing and programmable. It is optimized for throughput. Explicit management of on- chip memory can be done.



```
  Cache size:                                    0
  Global memory size:                            1073741824
  Constant buffer size:                          65536
  Max number of constant args:                   8
  Local memory type:                             Global
  Local memory size:                             16384
  Error correction support:                      0
  Profiling timer resolution:                    1
  Device endianess:                              Little
  Available:                                     Yes
  Compiler available:                            Yes
  Execution capabilities:
    Execute OpenCL kernels:                      Yes
    Execute native function:                     No
  Queue properties:
    Out-of-Order:                                No
    Profiling :                                  Yes
  Platform ID:                                   546E4514
  Name:                                          ATI RV770
  Vendor:                                        Advanced Micro Devices, Inc.
  Driver version:                                CAL 1.4.1607
  Profile:                                       FULL_PROFILE
  Version:                                       OpenCL 1.0 AMD-APP (923.1)
  Extensions:                                    cl_khr_fp64 cl_amd_fp64 cl_khr
gl_sharing cl_amd_device_attribute_query cl_khr_d3d10_sharing
```

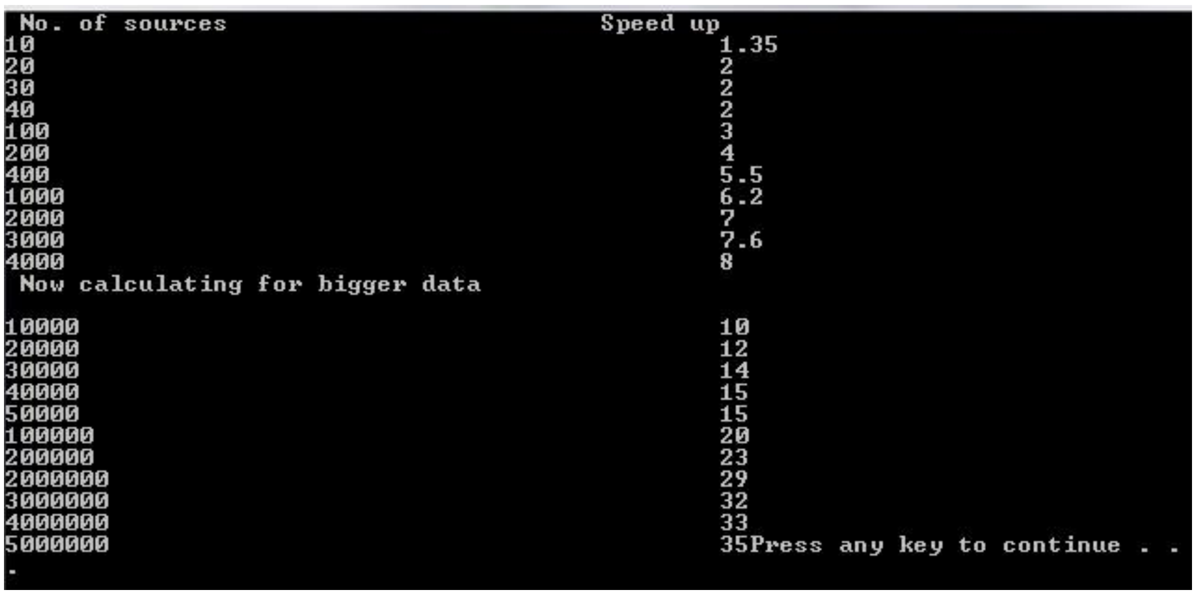
Figure 4. Characteristics of CPU



```
  Max image 2D width:                            8192
  Max image 2D height:                           8192
  Max image 3D width:                            2048
  Max image 3D height:                           2048
  Max image 3D depth:                            2048
  Max samplers within kernel:                    16
  Max size of kernel argument:                   4096
  Alignment (bits) of base address:              1024
  Minimum alignment (bytes) for any datatype:    128
  Single precision floating point capability
    Denorms:                                     Yes
    Quiet NaNs:                                  Yes
    Round to nearest even:                       Yes
    Round to zero:                               Yes
    Round to +ve and infinity:                   Yes
    IEEE754-2008 fused multiply-add:             Yes
  Cache type:                                    Read/Write
  Cache line size:                               64
  Cache size:                                    32768
  Global memory size:                            2147483648
  Constant buffer size:                          65536
  Max number of constant args:                   8
  Local memory type:                             Global
  Local memory size:                             32768
  Error correction support:                      0
  Profiling timer resolution:                    383
  Device endianess:                              Little
  Available:                                     Yes
  Compiler available:                            Yes
  Execution capabilities:
    Execute OpenCL kernels:                      Yes
    Execute native function:                     Yes
  Queue properties:
    Out-of-Order:                                No
    Profiling :                                  Yes
  Platform ID:                                   546E4514
  Name:                                          Intel(R) Core(TM)2 Duo CPU
E7300  @ 2.66GHz
  Vendor:                                        GenuineIntel
  Driver version:                                2.0 (sse2)
  Profile:                                       FULL_PROFILE
  Version:                                       OpenCL 1.2 AMD-APP (923.1)
  Extensions:                                    cl_khr_fp64 cl_amd_fp64 cl_khr_
global_int32_base_atomics cl_khr_global_int32_extended_atomics cl_khr_local_int3
2_base_atomics cl_khr_local_int32_extended_atomics cl_khr_byte_addressable_store
 cl_khr_gl_sharing cl_ext_device_fission cl_amd_device_attribute_query cl_amd_ve
c3 cl_amd_printf cl_amd_media_ops cl_amd_popcnt cl_khr_d3d10_sharing
```

Figure 5. Characteristics of GPU

ISSN:2229-6093



```
 No. of sources                              Speed up
10                                                   1.35
20                                                   2
30                                                   2
40                                                   2
100                                                  3
200                                                  4
400                                                  5.5
1000                                                 6.2
2000                                                 7
3000                                                 7.6
4000                                                 8
 Now calculating for bigger data

10000                                                10
20000                                                12
30000                                                14
40000                                                15
50000                                                15
100000                                               20
200000                                               23
2000000                                              29
3000000                                              32
4000000                                              33
5000000                                              35Press any key to continue . .
.
```

Figure 6. Processing speed up of GPU with increasing no. of sources

6. Conclusions

GPU is efficient over CPU when large number of calculations is to be handled. As the number of sources to be handled parallel increase, the efficiency of GPU is clearer over CPU. The GPU is the only parallel processor that has seen widespread success in many commercial applications. It allows us to experiment with 100s of cores today. It can be used to replace conventional CPUs if its shortcomings like memory management, debugging problem and inefficient pointer chasing are taken care of.

References

[1] Mattson, T. G., Sanders, B. A., & Massingill, B. L., "*Patterns of parallel programming*" Upper Saddle River, NJ: Addison-Wesley, 2004.

[2] Von Neumann J., "*First draft of a report on the EDVAC*", Contract No. W-670-ORD-4926, U.S. Army Ordnance Department and University of Penn- sylvania (reproduced in Goldstine H. H. (Ed.), The computer: From Pascal to von Neumann. Princeton, NJ: Princeton University Press), 1972.

[3] Sutter, H., & Larus, J., "Software and the concurrency revolution", *ACM Queue*, 3(7), 2005, pp. 54–62.

[4] Hwu, W. W., Keutzer, K., & Mattson, T., "The concurrency challenge", *IEEE Design and Test of Computers*, July/August, 2008, pp. 312–320, 2008.

[5] Matthew Scarpino, " *Open CL in action how to accelerate graphics and computations*" , Foundation of Open CL programming, 2011, PP. 1-7.

[6] *Benedict R. Gaster*, *"OpenCL and the AMD APP SDK"*, AMD Developer Central, 2011. Available online: http://developer.amd.com/documentation/articles/pages/OpenCL-and-the-AMD-APP-SDK.aspx.

[7] Aaftab Munshi, "*The OpenCL Specification*", Khronos OpenCL Working Group, Version: 1.0, Document Revision: 29, 2008. Available Online: http://www.khronos.org/registry/cl/specs/opencl-1.0.29.pdf.