White Paper

# Using Graphics Chips for General Purpose Computation

Document Version 0.1
May 12, 2010

Tech Source
An EIZO Group Company
442 Northlake Blvd.
Altamonte Springs, FL 32701
(407) 262-7100

# TABLE OF CONTENTS

# 1. Introduction

As graphics chips become more powerful, new ways of harnessing that power are beginning to take shape. While typical CPUs can process one single stream of commands per core, a modern graphics chip can process hundreds of simple floating point operations at the same time. The benefits of this translate directly into many applications where thousands of complex vector equations are required to be performed every second.

*Compute Cores*

To understand how an application can make use of this architecture, it is necessary to understand how the architecture is laid out. Consider first what a graphics processor is required to do. Every time the computer screen refreshes itself (~60-80 times per second) the graphics chip needs to transform a 3-dimensional image and project it onto a 2-dimensional surface. This translation is done for each pixel on the screen. Given a resolution of 1280x1024, that is over 1.3 million pixels per screen or 104 million pixels per second.

To make this happen as fast as possible, modern graphics chips have introduced an architecture made up of hundreds of compute cores. Each core works independently to carry out the computations required for a single pixel. Working in parallel, these pipelines process hundreds of pixels at the same time.

This same architecture can be used to perform rapid mathematical calculations for a myriad of applications.

# 2 Programming Languages

Programming across multiple cores running simultaneously requires a different mindset and a specialized programming language. Initially, ATI and nVidia worked independently, supporting languages that worked only on their chip. ATI adopted Stanford's Brook extension to C, and Nvidia created Cuda. Developers had to choose which manufacturer they were going to use and stick with it.

Now, however, the mainstream approach is to use an open standard programming language called OpenCL. This language was created by architects from AMD, Apple, IBM, Intel, Nvidia, and Sony, and allows developers the flexibility of choosing a different graphics vendor in the future. In this way, programmers can benefit from the fastest chips on the market each year, and encourage competition between vendors.

Another advantage of OpenCL is its ability to leverage multi-core CPUs together with multi-core GPUs. The benefit from this will scale as the number of CPU cores increase, which is significant as 32-core products become available this year. Applications based solely on Cuda or Brook will not see as much benefit from these multi-core processors as OpenCL programs will.

An exciting new development is that both ATI and Nvidia now offer support for the cl_khr_gl extension. This allows textures, which reside in GPU memory, to be used by both OpenGL and OpenCL, without having to first pass the data over the PCI bus into system memory. Applications that wish to display the product of their numerical processing on the display can greatly benefit from this feature.

# 3 Uses

Many applications can benefit from the processing power of the GPU. Basically, any field where extensive integer or floating point operations must be performed as quickly as possible will benefit from GPGPU technology.

## *3.1 Radar*

Radar interpretation must carry out many floating point operations in order to identify targets, movement, etc. Radar data can be collected much faster than it can be processed, making these computations the primary bottleneck. To improve performance, radar system designers must add more processors (or cores) or increase their clock speeds, both of which work in direct opposition to the need to make these systems smaller and consume less power. GPGPUs offer the ability to increase performance, decrease power consumption, and decrease size.

## *3.2 Scientific computing*

From weather tracking to biological simulations, every aspect of science today requires intense computing power. Being able to multiply the processing power of existing systems means faster results and much greater productivity.

## *3.3 Encryption / Decryption*

One of the most obvious uses of GPGPU programming is encryption and decryption. In addition to performing rapid floating point and integer operations, GPGPUs by nature have built-in XOR logic operations.

Consider the case of full disk encryption, where reads and writes to the hard disk must undergo standard AES encryption. Since hard drives are known to be the performance bottleneck on most systems, any decrease in read/write performance, such as encryption, directly effects the performance of the entire system. Using the GPU to do the encryption not only speeds up the disk accesses, it also frees the CPU from performing these computations, giving the system a double boost in performance.

# 4 Measuring Performance

Each of the major graphics chip vendors uses a unique architecture inside their chip. So, it is difficult to evaluate exactly which chipset will provide the best performance for a given application. Programming skill in this area should also not be underestimated.

Probably the most useful number for most applications is the number of floating point operations (or FLOPs) that can be done each second. Modern chips will be in the hundreds of GigaFLOPS to 1 or 2 TeraFLOPS range.

### *4.1 Running the numbers*

To put the numbers in perspective, consider that a single Opteron 6100 8-core processor, the latest AMD offering, can process 150 GFlops while consuming 80 watts of power. Its competitor, Intel's 6-core 3.33GHz Xeon, consumes 130 watts and processes only about half as many GFlops. Compare that with an embedded GPU like ATI's E4690, which can process nearly 400 GFlops and consumes only 25 watts or ATI's E6760 which consumes 35 watts. Where power use is less of a concern, the new Radeon HD5870 can process 2720 GFlops, but eats up 188 watts - still a big win with 18x the processing capability at little more than twice the power consumption.

| | | | |
|---|---|---|---|
| ATI "Evergreen" 5000 Series | 2.72 TFLOPS | 188 watts | 14.8 GF/watt |
| Nvidia "Fermi" GF100 | 1.03 TFLOPS | | |
| ATI E6760 (condor3xxx) | 575 GFLOPS | 35 watts | 16.5 GF/watt |
| ATI E4690 (condor2xxx) | 384 GFLOPS | 25 watts | 15.4 GF/watt |
| Opteron 6100 (8-core CPU) | 150 GFLOPS | 80 watts | 1.86 GF/watt |
| 3.33GHz Xeon (6-core CPU) | 75 GFLOPS | 130 watts | 0.58 GF/watt |

## 5 Resources

Assuming all this has piqued your interest in OpenCL development and you want to learn more about developing software that takes advantage of your GPU as a fast parallel processor. The first place I would suggest you look is the AMD website, especially their OpenCL Zone at http://developer.amd.com/zones/openclzone/pages/default.aspx. This resource is constantly updated and provides an overview of OpenCL and lots of development help and sample applications.

AMD also provides a good Programming Guide for their Accelerated Parallel Processing (APP) SDK and OpenCL that can be found at http://developer.amd.com/sdks/AMDAPPSDK/assets/AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide.pdf

Another good source is the very good book published by Addison-Wesley titled "OpenCL Programming Guide". This will take you through the OpenCL world in a very straight forward way and introduce you to the idea of parallel software development with a collection of case studies. This book is a must have guide for the new OpenCL developer.

## 6 Conclusion

Using GPUs to perform general purpose mathematical computations provides more

calculations per second using less power. Their smaller size and cheaper cost make them even more attractive. As more programmers and system architects learn how to harness this power, such systems will become ubiquitous.

Helping to expand this market is the introduction of a standardized programming language called OpenCL. Not only does this standard free system designers from boxing themselves into a single source for GPUs, it has the potential to outperform systems written in platform-specific languages such as Cuda and Brook. This is because OpenCL can harness both the power of the GPU and the multiple cores of the CPU, allowing them to work together in a more optimized way. Further, the cl_khr_gl extension provides a mechanism for sharing data between OpenGL and OpenCL, opening up a world of options for post-processing display, without having to move the results across the bus to system memory.