**Research and Study on GPU**

Zhiwei Qiao

George Washington University

G30502862

[jarvisqiao@gwu.edu](mailto:jarvisqiao@gwu.edu)

**Abstract**

The graphics processing unit (GPU) has become a significant and integral part of modern's mainstream computing systems. GPU’s performance and capabilities have increased dramatically in recent years. So that the modern GPU is not only a powerful graphics engine but also a highly programmable processor which can substantially surpass CPU in some specific fields. This paper demonstrates the general purpose computation, the programming model and the environment of GPU. Besides, the paper provides the introduction of the evolution process of GPU architecture in detail. At the end of the essay, the further developments, the research hotspots and the latest techniques of the GPU are presented as well.

**1. Introduction**

A graphics processing unit (GPU), also occasionally called visual processing unit (VPU), is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display. GPUs are used in embedded systems, mobile phones, personal computers, workstations, and game consoles. Modern GPUs are very efficient at manipulating computer graphics and image processing, and their highly parallel structure makes them more effective than general-purpose CPUs for algorithms where processing of large blocks of data is done in parallel. In a personal computer, a GPU can be present on a video card, or it can be on the motherboard or in certain CPUs, on the CPU die. The term GPU was popularized by Nvidia in 1999, who marketed the GeForce 256 as "the world's first 'GPU', or Graphics Processing Unit, a single-chip processor with integrated transform, lighting, triangle setup/clipping, and rendering engines that are capable of processing a minimum of 10 million polygons per second".



**Figure 1. A kind of GPU**

**2. Development**

In 1985, the Commodore Amiga featured a GPU advanced for a personal computer at the time. It supported line draw, area fill, and included a type of stream processor called a blitter which accelerated the movement, manipulation and combination of multiple arbitrary bitmaps. Also included was a coprocessor with its own primitive instruction set capable of directly invoking a sequence of graphics operations without CPU intervention. Prior to this and for quite some time after, many other personal computer systems instead used their main, general-purpose CPU to handle almost every aspect of drawing the display, short of generating the final video signal. In 1987, the IBM 8514 graphics system was released as one of the first video cards for IBM PC compatibles to implement fixed-function 2D primitives in electronic hardware. In 1988, the first dedicated polygonal 3D graphics boards were introduced in arcades with the Namco System 21and Taito Air System.

Throughout the 1990s, 2D GUI acceleration continued to evolve. As manufacturing capabilities improved, so did the level of integration of graphics chips. Additional application programming interfaces (APIs) arrived for a variety of tasks, such as Microsoft's WinG graphics library for Windows 3.x, and their later DirectDraw interface for hardware acceleration of 2D games within Windows 95 and later. In the early- and mid-1990s, CPU-assisted real-time 3D graphics were becoming increasingly common in arcade, computer and console games, which led to an increasing public demand for hardware-accelerated 3D graphics.

In the PC world, initially, performance 3D graphics were possible only with discrete boards dedicated to accelerating 3D functions and lacking 2D GUI acceleration entirely. However, as manufacturing technology continued to progress, video, 2D GUI acceleration and 3D functionality were all integrated into one chip. OpenGL appeared in the early '90s as a professional graphics API, but originally suffered from performance issues which allowed the Glide API to step in and become a dominant force on the PC in the late '90s. Over time, a parity emerged between features offered in hardware and those offered in OpenGL. DirectX became popular among Windows game developers during the late 90s. Unlike OpenGL, Microsoft insisted on providing strict one-to-one support of hardware. The approach made DirectX less popular as a standalone graphics API initially, since many GPUs provided their own specific features, which existing OpenGL applications were already able to benefit from, leaving DirectX often one generation behind. Over time, Microsoft began to work more closely with hardware developers, and started to target the releases of DirectX to coincide with those of the supporting graphics hardware. Direct3D 5.0 was the first version of the burgeoning API to gain widespread adoption in the gaming market, and it competed directly with many more-hardware-specific, often proprietary graphics libraries, while OpenGL maintained a strong following. The Nvidia GeForce 256 was the first consumer-level card released on the market with hardware-accelerated T&L, while professional 3D cards already had this capability. Hardware transform and lighting, both already existing features of OpenGL, came to consumer-level hardware in the '90s and set the precedent for later pixel shader and vertex shader units which were far more flexible and programmable.

**3. GPU computing**

As we enter the era of GPU computing, demanding applications with substantial parallelism increasingly use the massively parallel computing capabilities of GPUs to achieve superior performance and efficiency. Today GPU computing enables applications that we previously thought infeasible because of long execution times.

With the GPU's rapid evolution from a configurable graphics processor to a programmable parallel processor, the ubiquitous GPU in every PC, laptop, desktop, and workstation is a many-core multithreaded multiprocessor that excels at both graphics and computing applications. Today's GPUs use hundreds of parallel processor cores executing tens of thousands of parallel threads to rapidly solve large problems having substantial inherent parallelism. They're now the most pervasive massively parallel processing platform ever available, as well as the most costeffective.

Modern GPUs use most of their transistors to do calculations related to 3D computer graphics. They were initially used to accelerate the memory-intensive work of texture mapping and rendering polygons, later adding units to accelerate geometric calculations such as the rotation and translation of vertices into different coordinate systems. Recent developments in GPUs include support for programmable shaders which can manipulate vertices and textures with many of the same operations supported by CPUs, oversampling and interpolation techniques to reduce aliasing, and very high-precision color spaces. Because most of these computations involve matrix and vector operations, engineers and scientists have increasingly studied the use of GPUs for non-graphical calculations.

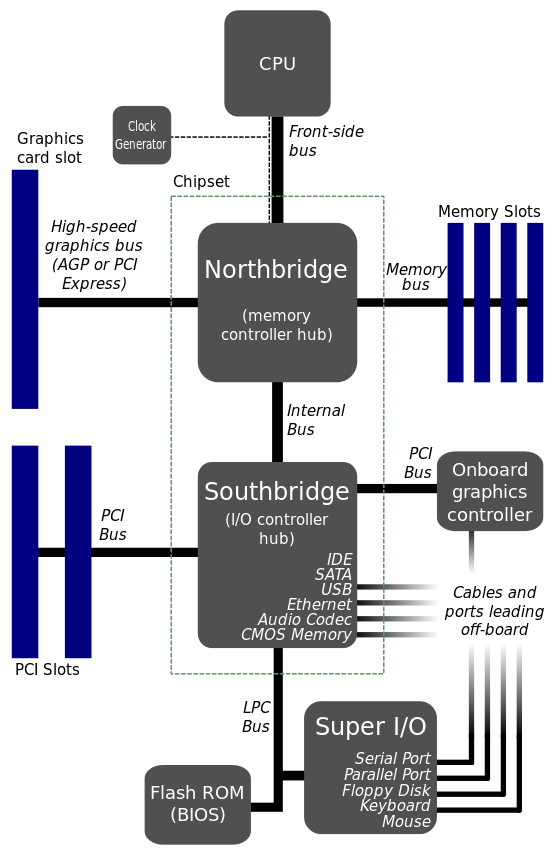
In addition to the 3D hardware, today's GPUs include basic 2D acceleration and framebuffer capabilities, usually with a VGA compatibility mode.

Most GPUs made since 1995 support the YUV color space and hardware overlays, important for digital video playback, and many GPUs made since 2000 also support MPEG primitives such as motion compensation and iDCT. This process of hardware accelerated video decoding, where portions of the video decoding process and video post-processing are offloaded to the GPU hardware, is commonly referred to as "GPU accelerated video decoding", "GPU assisted video decoding", "GPU hardware accelerated video decoding" or "GPU hardware assisted video decoding".

**3.1 GPU Technology Development**

The demand for faster and higher-definition graphics continues to drive the development of increasingly parallel GPUs. GPUs first used floating-point arithmetic to calculate 3D geometry and vertices, then applied it to pixel lighting and color values to handle high-dynamic-range scenes and to simplify programming. They implemented accurate floating-point rounding to eliminate frame-varying artifacts on moving polygon edges that would otherwise sparkle at real-time frame rates.

As programmable shaders emerged, GPUs became more flexible and programmable. Developing the Cg language for programming GPUs provided a scalable parallel programming model for the programmable floating-point vertex and pixel-fragment processors of GeForce FX, GeForce 6800, and subsequent GPUs. A Cg program resembles a C program for a single thread that draws a single vertex or single pixel. The multithreaded GPU created independent threads that executed a shader program to draw every vertex and pixel fragment. In addition to rendering real-time graphics, programmers also used Cg to compute physical simulations and other general-purpose GPU (GPGPU) computations. Early GPGPU computing programs achieved high performance, but were difficult to write because programmers had to express nongraphics computations with a graphics API such as OpenGL.



**Figure 2. Layout of GPU**

**3.2 The GPU Programming Model**

The programmable units of the GPU follow a single-program multiple-data (SPMD) programming model. For efficiency, the GPU processes many elements (vertices or fragments) in parallel using the same program. Each element is independent from the other elements, and in the base programming model, elements cannot communicate with each other. All GPU programs must be structured in this way: many parallel elements, each processed in parallel by a single program.

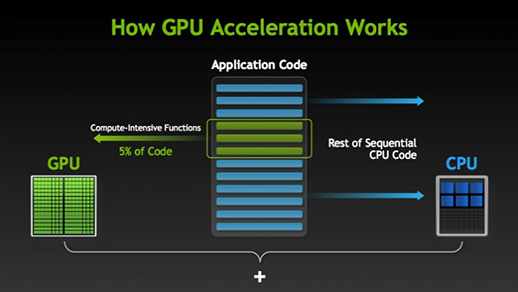
Each element can operate on 32-bit integer or floating-point data with a reasonably complete general-purpose instruction set. Elements can read data from a shared global memory and, with the newest GPUs, also write back to arbitrary locations in shared global memory.

This programming model is well suited to straight-line programs, as many elements can be processed in lockstep running the exact same code. Code written in this manner is single instruction, multiple data (SIMD). As shader programs have become more complex, programmers prefer to allow different elements to take different paths through the same program, leading to the more general SPMD model. One of the benefits of the GPU is its large fraction of resources devoted to computation. Allowing a different execution path for each element requires a substantial amount of control hardware. Instead, today's GPUs support arbitrary control flow per thread but impose a penalty for incoherent branching. GPU vendors have largely adopted this approach. Elements are grouped together into blocks, and blocks are processed in parallel. If elements branch in different directions within a block, the hardware computes both sides of the branch for all elements in the block. The size of the block is known as the “branch granularity” and has been decreasing with recent GPU generations—today, it is on the order of 16 elements. In writing GPU programs, then, branches are permitted but not free. Programmers who structure their code such that blocks have coherent branches will make the best use of the hardware.

**3.3 GPU accelerated computing**

What is GPU accelerated computing? GPU-accelerated computing is the use of a GPU together with a CPU to accelerate scientific, analytics, engineering, consumer, and enterprise applications. Pioneered in 2007 by NVIDIA, GPU accelerators now power energy-efficient datacenters in government labs, universities, enterprises, and small-and-medium businesses around the world. GPUs are accelerating applications in platforms ranging from cars, to mobile phones and tablets, to drones and robots.

How GPUs accelerate applications? GPU-accelerated computing offers unprecedented application performance by offloading compute-intensive portions of the application to the GPU, while the remainder of the code still runs on the CPU. From a user's perspective, applications simply run significantly faster. The specific progression is shown in figure 3.

****

**Figure 3. How GPU acceleration works**

**3.4 CPU versus GPU**

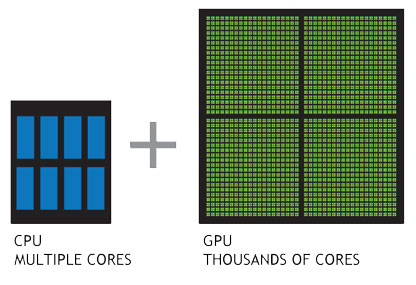
A simple way to understand the difference between a CPU and GPU is to compare how they process tasks. A CPU consists of a few cores optimized for sequential serial processing while a GPU has a massively parallel architecture consisting of thousands of smaller, more efficient cores designed for handling multiple tasks simultaneously. In a word, GPUs have thousands of cores to process parallel workloads efficiently. The differences between CPU and GPU are shown in figure 4.

There are three basic approaches to adding GPU acceleration to applications:

1. Dropping in GPU-optimized libraries.

2. Adding compiler "hints" to auto-parallelize your code.

3. Using extensions to standard languages like C and FORTRAN.

****

**Figure 4. CPU vs GPU**

**4. GPU forms**

**4.1 Dedicated graphics cards**

The GPUs of the most powerful class typically interface with the motherboard by means of an expansion slot such as PCI Express (PCIe) or Accelerated Graphics Port (AGP) and can usually be replaced or upgraded with relative ease, assuming the motherboard is capable of supporting the upgrade. A few graphics cards still use Peripheral Component Interconnect (PCI) slots, but their bandwidth is so limited that they are generally used only when a PCIe or AGP slot is not available.

A dedicated GPU is not necessarily removable, nor does it necessarily interface with the motherboard in a standard fashion. The term "dedicated" refers to the fact that dedicated graphics cards have RAM that is dedicated to the card's use, not to the fact that most dedicated GPUs are removable. Dedicated GPUs for portable computers are most commonly interfaced through a non-standard and often proprietary slot due to size and weight constraints. Such ports may still be considered PCIe or AGP in terms of their logical host interface, even if they are not physically interchangeable with their counterparts.

Technologies such as SLI by Nvidia and CrossFire by AMD allow multiple GPUs to draw images simultaneously for a single screen, increasing the processing power available for graphics.

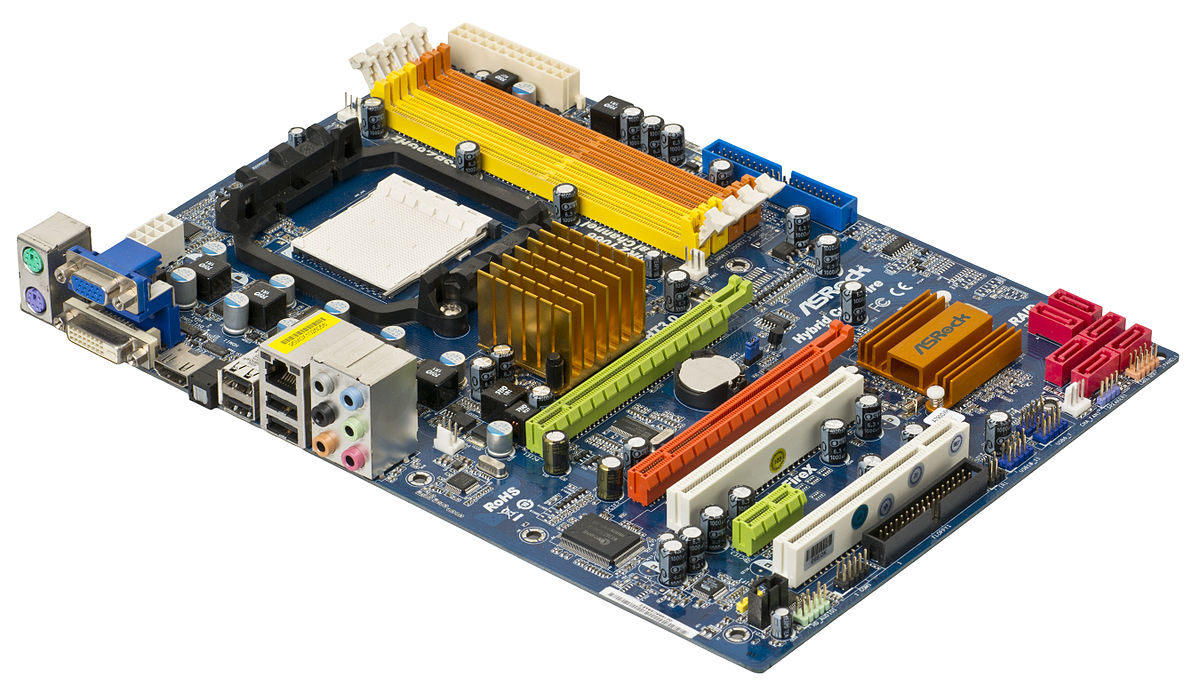


**Figure 5. Dedicated graphics cards made by NVIDIA**

**4.2 Integrated graphics solutions**

Integrated graphics solutions, shared graphics solutions, or integrated graphics processors (IGP) utilize a portion of a computer's system RAM rather than dedicated graphics memory. IGPs can be integrated onto the motherboard as part of the chipset, or within the same die as CPU, like AMD APU or Intel HD Graphics. Some of AMD's IGPs use dedicated sideport memory on certain motherboards. Computers with integrated graphics account for 90% of all PC shipments. These solutions are less costly to implement than dedicated graphics solutions, but tend to be less capable. Historically, integrated solutions were often considered unfit to play 3D games or run graphically intensive programs but could run less intensive programs such as Adobe Flash. Examples of such IGPs would be offerings from SiS and VIA circa 2004. However, modern integrated graphics processors such as AMD Accelerated Processing Unit and Intel HD Graphics are more than capable of handling 2D graphics or low stress 3D graphics.

As a GPU is extremely memory intensive, an integrated solution may find itself competing for the already relatively slow system RAM with the CPU, as it has minimal or no dedicated video memory. IGPs can have up to 29.856 GB/s of memory bandwidth from system RAM, however graphics cards can enjoy up to 264 GB/s of bandwidth between its RAM and GPU core. This bandwidh is what is referred to as the memory bus and can be performance limiting. Older integrated graphics chipsets lacked hardware transform and lighting, but newer ones include it.



**Figure 6. A motherboard with integrated graphics, which has HDMI, VGA and DVI outs**

**4.3 Hybrid solutions**

This newer class of GPUs competes with integrated graphics in the low-end desktop and notebook markets. The most common implementations of this are ATI's HyperMemory and Nvidia's TurboCache.

Hybrid graphics cards are somewhat more expensive than integrated graphics, but much less expensive than dedicated graphics cards. These share memory with the system and have a small dedicated memory cache, to make up for the high latency of the system RAM. Technologies within PCI Express can make this possible. While these solutions are sometimes advertised as having as much as 768MB of RAM, this refers to how much can be shared with the system memory.

4.4 **Stream Processing and General Purpose GPUs (GPGPU)**

It is becoming increasingly common to use a general purpose graphics processing unit as a modified form of stream processor. This concept turns the massive computational power of a modern graphics accelerators shader pipeline into general-purpose computing power, as opposed to being hard wired solely to do graphical operations. In certain applications requiring massive vector operations, this can yield several orders of magnitude higher performance than a conventional CPU. The two largest discrete GPU designers, ATI and Nvidia, are beginning to pursue this approach with an array of applications. In certain circumstances the GPU calculates forty times faster than the conventional CPUs traditionally used by such applications.

GPGPU can be used for many types of embarrassingly parallel tasks including ray tracing. They are generally suited to high-throughput type computations that exhibit data-parallelism to exploit the wide vector width SIMD architecture of the GPU.

Furthermore, GPU-based high performance computers are starting to play a significant role in large-scale modelling. Three of the 10 most powerful supercomputers in the world take advantage of GPU acceleration.

NVIDIA cards support API extensions to the C programming language such as CUDA ("Compute Unified Device Architecture") and OpenCL. CUDA is specifically for NVIDIA GPUs whilst OpenCL is designed to work across a multitude of architectures including GPU, CPU and DSP, using vendor specific SDKs. These technologies allow specified functions such as kernels from a normal C program to run on the GPU's stream processors. This makes C programs capable of taking advantage of a GPU's ability to operate on large matrices in parallel, while still making use of the CPU when appropriate. CUDA is also the first API to allow CPU-based applications to directly access the resources of a GPU for more general purpose computing without the limitations of using a graphics API.

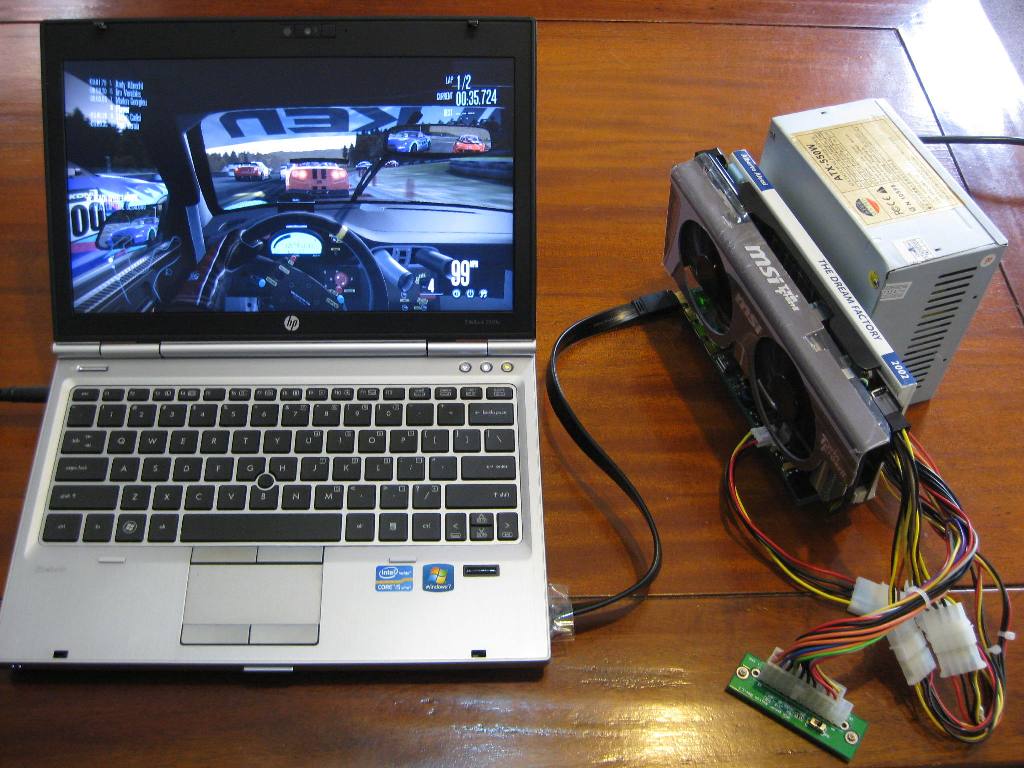
Since 2005 there has been interest in using the performance offered by GPUs for evolutionary computation in general, and for accelerating the fitness evaluation in genetic programming in particular. Most approaches compile linear or tree programs on the host PC and transfer the executable to the GPU to be run. Typically the performance advantage is only obtained by running the single active program simultaneously on many example problems in parallel, using the GPU's SIMD architecture. However, substantial acceleration can also be obtained by not compiling the programs, and instead transferring them to the GPU, to be interpreted there. Acceleration can then be obtained by either interpreting multiple programs simultaneously, simultaneously running multiple example problems, or combinations of both. A modern GPU can readily simultaneously interpret hundreds of thousands of very small programs.

**4.5 External GPU (eGPU)**

An external GPU is a graphics processor located outside of the housing of the computer. External Graphics Processors are often used with laptop computers. Laptops might have a substantial amount of RAM and a sufficiently powerful CPU, but often lack a powerful graphics processor and instead have a less powerful, but energy efficient on-board graphics chip. On-board graphics chips are often not powerful enough for playing the latest games, or for other tasks.

Therefore it is desirable to be able to attach to some external PCIe bus of a notebook. That may be an x1 2.0 5Gbit/s expresscard or mPCIe or wifi port or a 10Gbit/s/16Gbit/s Thunderbolt1/Thunderbolt2 port. Those ports being only available on certain candidate notebook systems.

External GPU's have had little official vendor support. Promising solutions such as Silverstone T004 (aka ASUS XG2) and MSI GUS-II were never released to the general public. MSI's Gamedock promising to deliver a full x16 external PCIe bus to a purpose built compact 13" MSI GS30 notebook. Lenovo and Magma partnering in Sep-2014 to deliver official Thunderbolt eGPU support. This has not stopped enthusiasts from creating their own DIY eGPU solutions.



**Figure 7. A sample of use of external GPU**

**5. Future**

Parallelism is the future of GPU computing. Future microprocessor development efforts will continue to concentrate on adding cores rather than increasing single-thread performance. Similarly, the highly parallel graphics processing unit (GPU) is rapidly gaining maturity as a powerful engine for computationally demanding applications. The GPU's performance and potential offer a great deal of promise for future computing systems, yet the architecture and programming model of the GPU are markedly different than most other commodity single-chip processors.

**5.1 Recent changes**

When it comes to picking a graphics card for your gaming PC, there are just two companies making cards that'll make your games look pretty and smooth: AMD and Nvidia. While AMD and Nvidia have co-existed somewhat peacefully over the past few years, in recent months both companies have become increasingly outspoken about their competitor's products, and neither is backing down. The origins of the GPU owe a lot to the sheer volume of companies trying to take ownership of the growing 3D gaming market back in the late 90s. Indeed, without the likes of 3dfx, Matrox, and ATI rapidly innovating on 3D graphics, the first true GPU--Nvidia's GeForce 256--might never have happened.

Today's GPUs are exponentially more powerful than the likes of the GeForce 256 or Nvidia's competitor at the time, the ATI Radeon DDR. The 64MB of video memory and 200 Mhz clocks speeds of those old cards pale in comparison to the multi-gigabyte, dual-slot monsters at home in current high-end gaming PCs. Additions like programmable shading, tessellation (the ability for a GPU to create geometry on-the-fly), and PhysX have made games look incredibly sophisticated, and with 3D games dominating the market, the demand for ever increasing levels of visual fidelity shows no sign of slowing down. But the GPU makers themselves don't drive all these advances, at least in Nvidia's case.

Real-time graphics that approach the visual fidelity of films might still be a ways away, but Nvidia and AMD are both working on new software technologies to make it happen. One of the most important aspects of creating convincing 3D visuals is implementing realistic lighting. The challenge is not having an artist manually animate realistic-looking lighting, but to simulate how a light reacts with, and bounces off, multiple surfaces in real-time, otherwise known as path tracing. In the film industry, path tracing is done offline, because the calculations required are simply too complex. Epic's Unreal Engine 4, while not true path tracing, did at one point feature a real-time, global illumination engine, called sparse voxel unity global illumination, which produced some impressive results. But, just before the engine launched, the feature was taken out.

Nvidia's most recent attempt at building a dual GPU card, the Titan Z, got off to a rocky start, with the product being delayed after its initial announcement. And when it did finally materialise, its performance--while still impressive--couldn't quite toppel the 295X2. Nvidia claims its priorities in developing the card--aircooling and power consumption--were part of the reason for its delay.

One of the most recent changes in the GPU market has been the rise of Nvidia's GameWorks technology and AMD's Mantle, which both companies are using to push performance forward alongside developing better GPUs. Nvidia's GamesWorks is a set of visual effects, physics effects, and tools designed to give developers more options when it comes to a game's visuals. Effects like HBAO+, TXAA, and FaceWorks are all part of GameWorks. AMD's Mantle is a competitor API to OpenGL and Direct X, allowing console-like low-level access to the CPU and GPU. By removing the some of the software layers of the operating system between the hardware and the software, with the right programming chops developers are able to see an increase in performance. Both technologies are sound in their efforts to push the performance and visual fidelity of 3D games, but there's a philosophical difference in how these technologies are being implemented. AMD claims its Mantle API is open, and free to use, while simultaneously bashing Nvidia's GameWorks for being more proprietary in nature.

**5.2 Multi-GPU technology**

Multi-GPU technology has come a long way in the last few years, to the point where it's now a legitimate use of a second GPU, instead of being a horrible waste of silicon and shaders.

In the before time though, multi-GPU technology was almost forgotten. 3dfx had introduced the original SLI (scan-line interleave) technology in 1998 with its Voodoo 2 cards, which was left on the shelf as Nvidia picked up the pieces of the graphics company. The green goblins did reintroduce it in 2004, though, renamed as the scalable link interface. ATI followed suit the following year and introduced its own multi-GPU tech, CrossFire.

There are still limitations, though, including Nvidia charging a premium for licensing SLI compatibility on motherboards (now that it no longer makes chipsets itself). The key limitations though are on what graphics cards can be used together in a multi-GPU array. ATI's master/slave beginnings aside, it was vital to have the exact same graphics card in a setup to have any chance of it working.

There are still driver issues between certain games and multi-GPU setups, so a single card rig is generally going to be more stable. But with the best single GPU you can afford you can always add in a second card at a later date. And if you're on the AMD side you may even be able to pair it with a faster card as prices drop over time, without the mobo compatibility worries that surround SLI.

**5.3 Challenges for parallel-computing chips**

Modern GPU computing lets application programmers exploit parallelism using new parallel programming languages such as CUDA and OpenCL and a growing set of familiar programming tools, leveraging the substantial investment in parallelism that high-resolution real-time graphics require.

GPUs’ tremendous computing and memory bandwidth have motivated their deployment into a range of high-performance computing systems. An even greater motivator for GPUs in high-performance computing systems is energy efficiency. The potential for future GPU performance increases presents great opportunities for demanding applications, including computational graphics, computer vision, and a wide range of high-performance computing applications.

Scaling the performance and capabilitiesof all parallel-processor chips, including GPUs, is challenging. First, as power supply voltage scaling has diminished, future architectures must become more inherently energy efficient. Second, the road map for memory bandwidth improvements is slowing down and falling further behind the computational capabilities available on die. Third, even after 40 years of research, parallel programming is far from a solved problem. Addressing these challenges will require research innovations that depart from the evolutionary path of conventional architectures and programming systems.

**5.4 The object of future GPU**

Computational requirements are large. Real-time rendering requires billions of pixels per second, and each pixel requires hundreds or more operations. GPUs must deliver an enormous amount of compute performance to satisfy the demand of complex real-time applications.

Parallelism is substantial. Fortunately, the graphics pipeline is well suited for parallelism. Operations on vertices and fragments are well matched to fine-grained closely coupled programmable parallel compute units, which in turn are applicable to many other computational domains.

Throughput is more important than latency. GPU implementations of the graphics pipeline prioritize throughput over latency. The human visual system operates on millisecond time scales, while operations within a modern processor take nanoseconds. This six-order-of-magnitude gap means that the latency of any individual operation is unimportant. As a consequence, the graphics pipeline is quite deep, perhaps hundreds to thousands of cycles, with thousands of primitives in flight at any given time. The pipeline is also feed-forward, removing the penalty of control hazards, further allowing optimal throughput of primitives through the pipeline. This emphasis on throughput is characteristic of applications in other areas as well.

**5.4 A new kind of GPU-ScaleGPU**

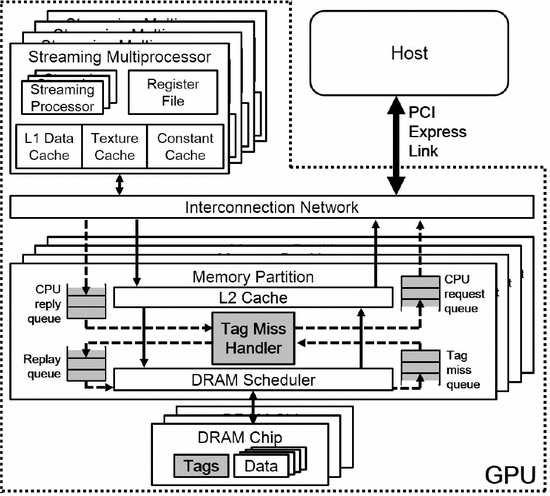
GPUs are becoming popular for general purpose computations due to their high throughput. GPUs achieve orders-of-magnitude speed-up over multi-core CPUs by running thousands of concurrent threads. Although GPUs promise high computational throughput, the programmer-managed GPU memory and slow data transfers through PCI Express make it hard to achieve the optimal throughput. If programmers wish to run GPU codes written and optimized for a large GPU memory on a small GPU memory, they must perform a series of manual code modifications based on the characteristics of the codes and algorithms (e.g., memory access pattern). An example of such application includes GPU-accelerated large scale data analysis [12]. Alternatively, programmers may use the zero-copy scheme [8] to avoid manual code modifications by having GPUs use CPU memory directly. However, the zero-copy scheme incurs prohibitive performance degradation due to redundant long-latency data transfers between CPU and GPU as the GPU memory is disabled. As a result, current GPU architectures and programming models do not provide high-performance memory-unaware GPU programming.

ScaleGPU, a novel GPU architecture which can run applications on any GPU memory size while providing high performance, is invented. To achieve both portability and performance, ScaleGPU uses GPU memory as a cache of CPU memory. First, ScaleGPU achieves portability by providing programmers a view of CPU memory-sized programming space. Second, ScaleGPU achieves high performance by minimizing the amount of long-latency CPU-GPU memory transfer as GPU performs most of the memory accesses on the cached data in GPU memory. Our detailed simulations show that ScaleGPU not only runs GPU applications written for a large GPU memory on a small GPU memory without any programming efforts, but also improves performance significantly. For example, ScaleGPU achieves ~48% speedup for the same size of GPU memory or it consumes ~25% of GPU memory for the same target performance in the case of hotspot application. ScaleGPU outperforms the zero-copy scheme significantly as expected.

Utilizing GPU memory as a cache of CPU memory has two main benefits. First, caching the data improves performance by removing unnecessary memory transfers. Therefore, GPU does not have to forward memory requests directly to CPU as long as the data is present in GPU memory. Instead, GPU can get the data from its local memory without paying the long-latency CPU-GPU communication overhead.

Second, GPU memory only stores the application's current working set which is typically small due to high spatial and temporal localities by the nature of GPU applications. As GPU applications tend to touch only a small portion of memory, the size of GPU memory can be significantly reduced without incurring a performance loss. Therefore, CPU and GPU memories become dynamically balanced by the application's behavior.

ScaleGPU uses GPU memory as a cache of the CPU memory to enable memory-unaware GPU programming. Therefore, ScaleGPU achieves a significant performance improvement, while also reducing the memory requirement significantly.



**Figure 8. ScaleGPU architecture**

**6. Conclusion**

The graphics processing unit has become an essential and integral part of modern's mainstream computer fields. GPU’s performance and capabilities have increased dramatically in recent years. So that the modern GPU is not only a powerful graphics engine but also a highly programmable processor which can substantially surpass CPU in some specific fields. The GPU technology is still in development and the future of GPU is bright.

**References**

[1]. J. D. Owens , D. Luebke , N. Govindaraju , M. Harris , J. Krüger , A. E. Lefohn and T. Purcell

"A survey of general-purpose computation on graphics hardware"

*Comput. Graph. Forum,* vol. 26, no. 1, pp. 80-113, 2007

[2]. D. Blythe

["The Direct3D 10 system"](http://dx.doi.org.proxygw.wrlc.org/10.1145/1141911.1141947)

*ACM Trans. Graph.,* vol. 25, no. 3, pp. 724-734, 2006

[3]. D. Tarditi , S. Puri and J. Oglesby

"Accelerator: Using data-parallelism to program GPUs for general-purpose uses"

*Proc. 12th Int. Conf. Architect. Support Program. Lang. Oper. Syst.,* pp. 325-335,2006

[4]. M. McCool

"Data-parallel programming on the cell BE and the GPU using the RapidMind development platform"

*Proc. GSPx Multicore Applicat. Conf.,* 2006

[5]. B. Bustos , O. Deussen , S. Hiller and D. Keim

"A graphics hardware accelerated algorithm for nearest neighbor search"

*Proc. 6th Int. Conf. Comput. Sci.,* vol. 3994, pp. 196-199, 2006

[6]. M. Kass , A. Lefohn and J. Owens  *Interactive depth of field using simulated diffusion on a GPU*,  2006  [online] Available:

http://www.graphics.pixar.com/DepthOfField/

[7]. J. E. Stone , J. C. Phillips , P. L. Freddolino , D. J. Hardy , L. G. Trabuco and K. Schulten  "Accelerating molecular modeling applications with graphics processors", *J. Comp. Chem.,* vol. 28,  pp.2618 -2640 2007

[8]. M. Pharr  "Interactive rendering in the post-GPU era",  *Proc. Graph. Hardware* 2006,   [online] Available:

http://www.graphicshardware.org/previous/www\_2006/presentations/pharr-keynote-gh06.pdf

[9]. J. Nickolls  "Scalable Parallel Programming with CUDA",  *ACM Queue*,  vol. 6,  no. 2,  pp.40 -53 2008

[10]. *NVIDIA CUDA Programming Guide*,  2009  [online] Available:

http://developer.download.nvidia.com comÏute/cuda/2\_3/toolkit/docs/ NVIDIA\_CUDA\_Programming\_Guide\_2.3.pdf

[11]. C. Boyd  "DirectCompute: Capturing the Teraflop",  *Microsoft Personal Developers Conf.*,  2009  [online] Available:

http://ecn.channel9.msdn. com/o9/pdc09/ppt/CL03.pptx

[12]. J. Montrym and H. Moreton  "The GeForce 6800",  *IEEE Micro*,  vol. 25,  no. 2,  pp.41 -51 2005

[13]. E. Lindholm  "NVIDIA Tesla: A Unified Graphics and Computing Architecture",  *IEEE Micro*,  vol. 28,  no. 2,  pp.39 -55 2008

[14]. M. Garland  "Parallel Computing Experiences with CUDA",  *IEEE Micro*,  vol. 28,  no. 4,  pp.13 -27 2008

[15]. D. B. Kirk and W. W. Hwu  *Programming Massively Parallel Processors: A Hands-on Approach*,  2010 :Morgan Kaufmann

[16]. M. A. Clark  "Solving Lattice QCD Systems of Equations Using Mixed Precision Solvers on GPUs",  *Computer Physics Comm.*,  2009  [online] Available:

http://arxiv.org/abs/0911. 3191v2

[17]. Y. Liu , B. Schmidt and D. L. Maskell  "MSA-CUDA: Multiple Sequence Alignment on Graphics Processing Units with CUDA",  *Proc. 20th IEEE Int\'l Conf. Application-Specific Systems, Architectures and Processors*,  pp.121 -128 2009

[18]. J. H. Huang  "The GPU Computing Tipping Point",  *Proc. IEEE Hot Chips*,  vol. 21,  2009  [online] Available:

http://www.hotchips.org/archives/hc21