Architectural evolution of NVIDIA GPUs for High-Performance Computing

Technical Report · February 2015

DOI: 10.13140/RG.2.1.1496.1042

1 author:

Angelo Corana
Italian National Research Council
85 PUBLICATIONS **2,015** CITATIONS

Consiglio Nazionale delle Ricerche
Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni

Technical Report on

Architectural Evolution of NVIDIA GPUs for High-Performance Computing

(IEIIT-CNR-150212)

Angelo Corana

(Decision Support Methods and Models Group)

IEIIT-CNR

Istituto di Elettronica ed Ingegneria
dell'Informazione e delle Telecomunicazioni

e-mail: angelo.corana@ieiit.cnr.it

Genova, February 12th 2015

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

Contents

Document Change Record	
IEIIT-CNR-150212-GE	First Issue date 12-02-2015

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

Abstract

In this report we consider the rapid technological development of Graphics Processing Units (GPUs) in the last decade, which enables, besides the traditional use of GPUs as graphics cards, the so-called GPU computing, i.e. the use of GPUs for non-graphical computationally intensive general-purpose applications. We also provide a brief history of graphics cards and related programming languages. We consider the NVIDIA GPUs, owing their widespread diffusion, high-performance and very rapid evolution. NVIDIA was the first company in the GPU computing market and is currently the leading.

1. Introduction

In recent years there has been a very rapid development of powerful architectures based on graphics cards of small dimension and low cost, originally developed for the market of computer entertainment, but that are also very suitable for High-Performance Computing (HPC).
During years graphics card, to answer to the increasing demand for real-time graphics, has evolved from fixed functions pipelines, i.e. non programmable hardware pipelines to fully programmable highly parallel many-core architectures, suitable for both graphics and non-graphics applications. In 1999 NVIDIA coined the term GPU (Graphics Processing Unit).

The use of GPU for general-purpose applications is called GPU computing, and is suitable for all the classical computationally heavy data-parallel applications, in scientific, engineering, financial fields (e.g. signal and image processing, Computational Fluid Dynamics, Finite Elements, complex simulations, etc.).

Modern GPUs (e.g. NVIDIA GPUs) have a many-core SIMT (Single-Instruction, Multiple-Threads) architecture [1], with a high number of cores grouped into blocks, and memory organized in a hierarchic way (at thread, block and global level). Today's GPUs greatly outperform CPUs both in arithmetic throughput and memory bandwidth. One or more GPUs can fit into a normal PC or workstation to boost performance (personal supercomputing); at large scale some of the most powerful supercomputers in the world rely on GPUs to achieve their outstanding performance.

Also the tools for the development of applications greatly improve and increase during last years. NVIDIA introduced in 2006 the CUDA environment (C/C++ programming language with extensions to manage the GPU, libraries and other development tools like profiler and debugger) [2, 3]. PGI (The Portland Group) developed compilers and software for C and Fortran, fully integrated with CUDA. Apple and The Khronos Group proposed OpenCL (Open Computing Language) [4], aimed at developing applications portable across different multi-vendor GPUs and multi-core CPUs.

Another important manufacturer of GPU cards is AMD, which acquired in 2006 ATI.
AMD sells both discrete GPUs as ATI Radeon and GPU on the same chip of the CPU as AMD Fusion APU (Accelerated Processing Unit).

The evolution of CPUs (e.g. Intel) towards an increasing number of cores (multi-core architectures) suggests that the most widespread and cheap future architectures for high-performance computing will be heterogeneous systems with multi-core CPUs and many-core GPUs [5].

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

An interesting feature of GPU cards is that, with respect to more recent CPUs, they give typically the same performance at 1/10 of cost and at 1/20 of consumed energy, so GPUs are very promising from the point of view of green computing. Moreover, the flops per watt ratio greatly improves from a GPU generation to a subsequent.

The widespread diffusion of heterogeneous multi-core/many-core systems with an increasing number of cores has a great impact on High-Performance Computing, making available low cost, small dimensions and low energy consumption systems, with very high computing power. The Moore law, affirming that the number pf transistors in a chip roughly double every 18-24 months, is now also applicable to the number of cores. CPU/GPU computing systems have a dramatic impact in a high number of technical-scientific applications [5], maintaining in the meantime a general-purpose approach.

2. A Brief History of graphics computing

2.1 Basic graphics operations

Let us consider the basic graphics operations: a 3D scene is represented as 3D coordinates of vertices of polygons; these coordinates are transformed (e.g. translated, rotated, scaled, projected) using basically matrix operations as the scene point of view changes; a rendering phase is needed for various reasons (filtering, smoothing, colouring, shading); the 3D scene is transformed into a 2D geometry and then into a number of fragments/pixels for visualization (rastering); fragments contain data associated with pixels: color, depth, stencil, etc. A typical quite complex scene has roughly 1 M vertices and 6 M pixels and to assure a fluid vision the above operations must be applied at least 25 times per second.

Shading comprises a number of manipulation on image components to improve the final light and color perception; there are basically three kinds of shaders: 1) Vertex shaders that operate on vertices coordinates and attributes; 2) Geometry shaders (introduced in Direct3D 10 and OpenGL 3.2) for a number of operations, e.g. automatic mesh complexity modification, consisting in the automatic generation of extra lines to provide a better approximation of a curve; 3) Pixel (or fragment) shaders, used after rasterization to compute the attributes of a fragment; pixel shaders can also be used to apply textures or special effects to improve the image quality. Currently, shading algorithms are executed on the GPU.

Texture operations involve texture data, a set of texture elements (texels). A texel is the smallest unit of data in a texture map. Texture mapping denotes the process in which a 2D surface (texture map) is wrapped-around a 3D object. Texture memory contains texture data to be used in the rendering phase (from texels to pixels). Texture operations comprise multi-texturing and texture filtering.

Graphics processing has a lot of data parallelism as there are a high number of vertices and fragments and each vertex/fragment is independent. So, graphics processing is very suitable for many-cores architectures; moreover, it is possible to hide memory latency with more computation.

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

2.2 Graphics cards with fixed functions

We can divide the evolution of graphics processors into different phases.
In the first phase, from early '80s to late '90s, graphics card perform in hardware some graphics functions The first cards perform a small number of graphics functions, whereas the others are executed on the CPU; during years, an increasing number of functions is executed on the graphics card, freeing the CPU, until the whole graphics pipeline was implemented in hardware on the card (graphics pipelines with fixed functions).

The term GPU (Graphics Processing Unit) was introduced by NVIDIA in 1999 for GeForce 256 (220 nm process) to denote a single-chip processor with integrated transform, lighting, triangle manipulation and rendering engines, able to process a minimum of 10 M polygons per second; it contained a fixed function 32-bit floating-point vertex transform and lighting processor and a fixed function integer pixel-fragment pipeline, which were programmed with OpenGL and the Microsoft DX7 API. GeForce 256 was the first NVIDIA card to implement the entire graphics pipeline in hardware (fixed function pipelines). Being implement in hardware the pipeline was not flexible and unable to follow the new functions added to graphics APIs [6].

2.3 Programmable GPUs

The next phase in the evolution of GPU hardware, from 2000 to 2003, is characterized by the gradual introduction of some programmable portions in the pipeline; NVIDIA in 2001 released the GeForce 3, with a vertex processor, able to execute small kernels written in low-level shader languages and a configurable 32-bit floating-point fragment pipeline, programmed with DX8 and OpenGL.

In 2002 appeared the first fully programmable graphics cards, the NVIDIA GeForce FX and ATI Radeon 9700. Separate, dedicated hardware were provided for pixel shader and vertex shader processing [7]. ATI coined the term VPU (Visual Processing Unit) in 2002 with the release of the Radeon 9700.

The most used programming languages in these years are OpenGL/Direct 8X with High-Level Shading Languages used to manage shaders by means of specific instructions for vertex/geometry/pixel shaders. The most used High-Level Shading Languages are: HLSL (Microsoft), Cg - C for graphics (NVIDIA, 2002), GLSlang (3D Labs) (or GLSL – OpenGL Shading Language). Although these languages allow the programmer to write GPU programs in a C-like programming language, they remain graphic-oriented and use graphics constructs like vertices, fragments and textures [6].

On the third phase, roughly from 2003 to 2006, more complex graphics programmable processors appear and GPUs, now fully programmable, begin to be used for non-graphics applications, using DirectX 9. GPUs begin to support full floating point and advanced texture processing.

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

In 2004, the GeForce 6 (with a fabrication process of 90 nm) and Radeon X800 were released, and were some of the first cards to use the PCI-express bus to connect with the CPU, with some hardware improvements such as 64-bit double precision, multiple rendering buffers, increased GPU memory and texture accesses. On the software side, early high level GPU languages such as Brook and Sh started to appear for shader programs.

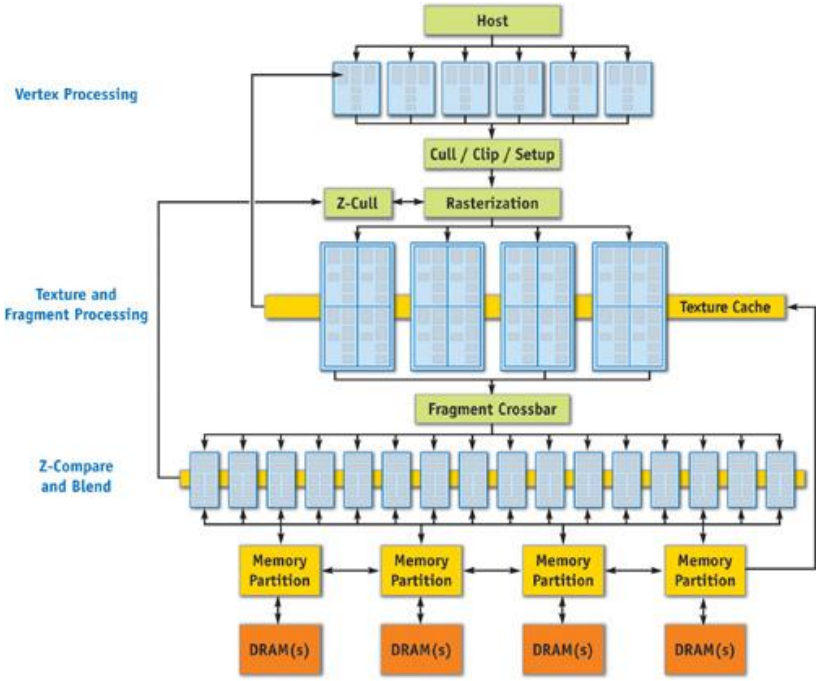


Figure 1. A block diagram of the GeForce 6 Series architecture (from [8]).

When viewed as a graphics pipeline, a GPU contains a programmable vertex engine, a programmable fragment engine, a texture engine, and a Z-compare and blend engine (Fig. 1).
When viewed alternatively as a processor for non-graphics applications, a GPU can be seen as a large number of programmable arithmetic units with high memory bandwidth that can be exploited for general-purpose compute-intensive applications [8].

Vertex and pixel-fragment processors have evolved at different rates: Vertex processors were designed for low-latency, high-precision math operations, whereas pixel-fragment processors were optimized for high-latency, lower-precision texture filtering. Vertex processors have traditionally supported more-complex processing, so they became programmable first. During these years, the two processor types become more and more similar as the result of a need for greater programming generality. However, the increased generality also increased the design complexity, area, and cost of developing two separate processors [1]. So, the next step will be the development of a unified architecture able to execute vertex and pixel-fragment shader programs on the same processor.

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

2.3.1 GPGPU – General-Purpose computation on Graphics Processing Units

The graphics processing unit (GPU) on commodity video cards has evolved into an extremely flexible and powerful processor. The power, flexibility and cheapness of GPUs make them an attractive platform for general-purpose computation, so, roughly from 2002, besides graphics applications GPUs begin to be used for different computationally heavy problems, such as linear algebra, image processing, physics and engineering simulation. The goal is to make the power of the GPU available to users as a computational coprocessor (CPU-GPU computing platform).

By using high-level shading languages such as DirectX, OpenGL and Cg, various data parallel algorithms have been ported to the GPU. Problems such as protein folding, stock options pricing, SQL queries, and MRI reconstruction achieved interesting performance speedups on the GPU. These early efforts that used graphics APIs for general purpose computing were known as GPGPU (General-Purpose computation on GPU). The term GPGPU was coined in 2002, and denotes methods, languages and algorithms to harness GPUs power for general-purpose computation.

GPUs are designed for and driven by video games, as a consequence the programming model is unusual and aimed to computer graphics and the programming environment is much more constrained than a conventional CPU. To access to the GPU computational resources the algorithms have to be expressed in native graphics operations, so efficient applications can be developed, but programming is very difficult and time-consuming.

Underlying architectures are inherently parallel, rapidly evolving (even in basic feature set). Moreover, often not all the architectural features are available to the programmer. So, it is not possible to simply port code written for the CPU, the code has to be rewritten.
Using these early approaches programming GPUs means either: using a graphics standard like OpenGL (which is mostly designed for rendering) or working quite close to the graphics rendering pipeline. In other words, to use a GPU to perform general-purpose processing, the application has to be treated as graphics [6].

While the GPGPU model demonstrated great speedups, it faced several drawbacks. First, it required the programmer to possess intimate knowledge of graphics APIs and GPU architecture. Second, problems had to be expressed in terms of vertex coordinates, textures and shader programs, greatly increasing program complexity. Third, basic programming features such as random reads and writes to memory were not supported, greatly restricting the programming model. Lastly, the lack of double precision support (until recently) make GPUs unsuitable for some scientific applications.

Despite the above limitations, the GPGPU paradigm was successfully adopted in various applicative areas, such as differential equations, linear algebra, signal and image processing, global illumination (algorithms for 3D graphics considering both direct and indirect illumination), geometric computing, databases and data mining.

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

2.3.1.1 Data Streams and Kernels

GPUs are particularly suitable for tasks that are computationally heavy and have a regular computational pattern. Graphics is well suited to the stream programming model and a stream hardware organization. So, the stream programing model was used for various applications, both graphics and general-purpose. Stream programming abstraction is a different way of considering computational problems aimed at developing programming models that match GPU features.

The first GPGPU languages has been developed with the purpose of making programming GPUs easier (of course the degree of simplicity is a relative matter). Examples of such languages are Sh (University of Waterloo), a metaprogramming language built on top of C++ for programmable GPUs, and Brook (Stanford University), used for some years by AMD/ATI (Brook+), before AMD switched to OpenCL.

Using this new languages there is no need to know graphics languages like OpenGL, DirectX, or ATI/NV extensions; they simplify the most common operations and focus on the algorithm, not on the implementation.

Brook is a general purpose streaming language that encourages the data parallel computing and is well suited for applications with high arithmetic intensity. It is based on C with stream extensions and views the GPU as a streaming coprocessor [6]. AMD developed an improved version Brook+.

Arithmetic Intensity is the number of arithmetic operations per word (transferred). GPGPU demands high arithmetic intensity for peak performance. Examples of suitable applications include: solving systems of linear equations, physically-based simulation on lattices, all-pairs shortest paths.

A Data Stream is a collection of a high (typically very high) number of data requiring the same computation (e.g. vertex positions, voxels, Finite-Element Method (FEM) cells, etc.). It is characterized by a high data parallelism. Kernels are functions applied to each element in a input stream to produce elements in a output stream (e.g. Transforms, Partial Differential Equations (PDEs), etc.). Better performance are achieved if there are not dependencies between stream elements and the arithmetic intensity is high. Kernels can be chained together.

The main stream operations available are:

- **Map**: apply a function to every element in a stream
- **Reduce**: use a function to reduce a stream to a smaller stream (often 1 element)
- **Scatter/gather**: indirect read and write
- **Filter**: select a subset of elements in a stream
- **Sort**: order elements in a stream
- **Search**: find a given element, nearest neighbors, etc.

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

2.4 Many-core GPUs

The 2006 is an important date in the GPU evolution: the introduction of NVIDIA's GeForce 8 series GPU has started the next phase in GPU development; the G80 Tesla architecture was the first to have unified, programmable shaders, i.e. a fully programmable unified processor called Streaming Multiprocessor, or SM, that handled vertex, pixel, and geometry computation. With the now unified shader hardware design, the traditional graphics pipeline model is now purely a software abstraction [7]. The G80 architecture with 128 cores (GeForce 8800, Quadro FX 5600, Tesla C870) can be considered the first generation of the new many-core GPUs.

At the software level, DirectX 10 with Shader Model 4.0 unifies the instruction set for shader programming [9].

The other important fact in 2006 was the introduction by NVIDIA of the new programming language CUDA (Compute Unified Device Architecture) [2, 9] (for NVIDIA cards only), first released as Beta version and then as CUDA 1.0. It exposes the GPU as a massively parallel processor (many-core), giving an hardware abstraction mechanism that hides the GPU hardware details from developers [3]. CUDA opened the era of GPU computing. During 2009, ATI released the Stream language (for ATI cards).

Hardware supported multithreading enable the G80 to manage up to thousands of threads concurrently in 128 processing cores. Moreover, the G80 series was the first to employ scalar thread processors instead of vector processors, as in previous GPU architectures.

Each NVIDIA architecture is offered in three versions or product lines: GeForce, of lower cost and more specifically suited for gamed; Quadro, of intermediate cost and suited for graphics professional use; Tesla, of higher cost, more reliable, and specifically suited for HPC marked. Moreover, for each version different models are commercialized, to match the different needs of users. During years, various generations of GPU architectures has been presented, as briefly summarized in the following.

The various generations of NVIDIA GPUs that are presented since 2006 have a many-core SIMT (Single-Instruction, Multiple-Threads) architecture [1, 2], with a high number of cores grouped into blocks, and memory organized in a hierarchic way (at thread, block and global level).

In 2008 NVIDIA presented the second generation of Tesla GPU, the GT200 architecture with 240 cores (GeForce GTX 280, Quadro FX 5800, Tesla T10) and CUDA 2.0.
In 2010 appeared the first Fermi-based architecture GPUs, with three billions of transistors, up to 512 cores and 6 GB of memory equipped first with CUDA 3.0 and then with CUDA 4.0.
At the end of 2012 NVIDIA released the Kepler architecture, with up to 1536 cores.
At the beginning of 2014 has been released the first GPU (GeForce GTX 750Ti) based on the Maxwell architecture (1st generation Maxwell), followed at the end of 2014 by the 2nd generation (GeForce GTX 970, GeForce GTX 980).
During 2016 is expected the release of the first cards based on the Pascal architecture.

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

So, the extremely rapid evolution of GPU hardware thus far has gone from a very specific, single core, fixed function hardware pipeline implementation just for graphics rendering, to a set of highly parallel and programmable cores, developed primarily to answer the increasingly demand of graphical power for videogames (high-definition 3D in real-time), but also very suitable for general purpose computation. Today's GPUs greatly outperform CPUs both in arithmetic throughput and memory bandwidth.

The power of graphics processors increases during years at an impressive rate basically for the following reasons: high arithmetic intensity, i.e. their specialized nature makes it easier to use more transistors for computation; multi-billion dollar video game market and a strong competition among companies drive innovation.
Comparing CPUs and GPUs we can note that in GPUs much more area is dedicated to arithmetic units and less to control units and caches; moreover the architecture of GPUs is complex as number of cores but simple in the organizations.

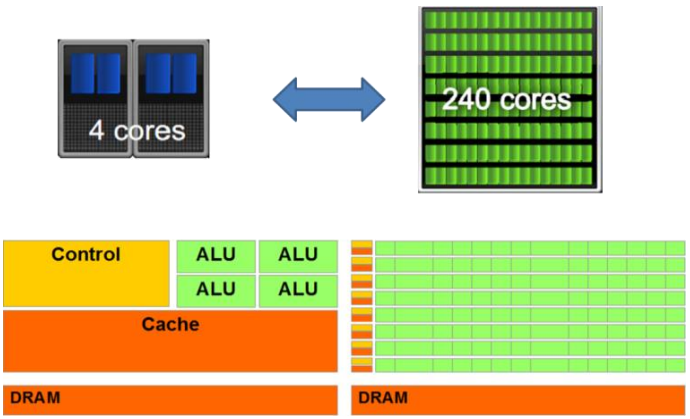


Figure 2. Comparison of CPU and GPU architectures.

With modern many-core GPUs graphics programs, written in shading languages such as Cg or High-Level Shading Language (HLSL), scale transparently over a wide range of thread and processor parallelism. GPU computing programs (see next subsection), written in C or C++ with the CUDA parallel computing model, or using a parallel computing API inspired by CUDA such as Direct-Compute or OpenCL scale transparently over a wide range of parallelism, i.e. number of cores and threads [10].

2.4.1 GPU computing

GPU computing is a new way of using GPUs for general-purpose high-performance applications employing high-level programming languages, very similar to C and Fortran, instead of graphical languages as in the GPGPU era.

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

GPU computing has been enabled by two key technologies introduced in 2006: at the hardware level the development of the new G80 unified graphics and compute architecture (first introduced in GeForce 8800, Quadro FX 5600, and Tesla C870 GPUs) [2], a true many-core parallel architecture; at the software level the release of the first version of CUDA (Compute Unified Device Architecture), a software environment which exposes the GPU as a many-core multi-thread architecture for parallel computing programmable with standard C/C++ languages with some extensions, and more generally with a variety of high level programming languages (besides C/C++, Fortran, Phyton, etc.).

It is worth noting that often also GPU computing is called GPGPU, denoting with GPGPU all uses of GPUs for general-purpose applications, no matter the programming languages and tools used; however it is more clear to name GPGPU the use of GPU for HPC applications using graphic programming languages and GPU computing the use of GPU for HPC using high level and general-purpose languages and programming tools.
As we will see in detail in the second part of the report, from one generation to the next typically increases the number of cores, increases the available memory, improves the flops/watt ratio, and more advanced features are introduced aimed to improve performance simplifying in the meantime the programming.

It is important to point out that, whereas the Tesla product line is specifically devoted to HPC, especially for large servers or supercomputers, often the GeForce line which offers a much better cost/performance ratio, reduced weight and size and lower dissipation, is the most suitable for HPC applications when price, size, weight and power consumption are important, as it is the case for various devices, e.g. medical ultrasound devices, where GPUs can be used to accelerate signal and image processing.

During years, a number of CUDA versions have been released, following the rapid evolution of GPU hardware, in order to include the new architectural features. The CUDA ecosystem [10] includes an increasing number of libraries, debuggers, performance tools, HPC/consumer applications, third-party applications.

Besides CUDA and its ecosystem, some others languages have been developed to program many-core GPUs.

OpenCL (Open Computing Language, Apple e The Khronos Group), first released during 2009, has been proposed as standard for the development of portable applications among a large variety of platforms, in particolar many-core and multi-core architectures. OpenCL is closely related to the CUDA programming model, sharing the key abstractions of threads, thread blocks, grids of thread blocks, barrier synchronization, per block shared memory, global memory, and atomic operations.

PGI (The Portland Group) [11] developed a compiler and related software tools supporting C and Fortran, fully integrated with CUDA.

OpenACC [12, 11], first version released at the end of 2011, is a directive-based standard allowing the specification of regions of a C, C++ or Fortran code that have to be executed on the GPU, yielding good performance and portability both for CPUs and GPUs.

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

DirectCompute is an API that supports general-purpose computing on GPUs on Microsoft's Windows. DirectCompute is part of the Microsoft DirectX collection of APIs, and was initially released with the DirectX 11.

C++ AMP (C++ Accelerated Massive Parallelism) accelerates the execution of a C++ code on a Windows environment by taking advantage of the data-parallel hardware in the GPU. The C++ AMP programming model includes support for multidimensional arrays, indexing, memory transfer, and tiling. It also includes a mathematical function library. C++ AMP language extensions can be use to control how data is moved from the CPU to the GPU and back.

The evolution of CPUs (e.g. Intel) towards an increasing number of cores (multi-core architectures) suggests that in the upcoming years the most common and cheap architectures for high-performance computing will be heterogeneous systems with multi-core CPUs and many-core GPUs [5].

3. Some basic features of the CUDA model

CUDA is the hardware and software architecture [2, 3] that enables NVIDIA GPUs to execute programs written with C, C++, Fortran, OpenCL, DirectCompute, and other languages. CUDA relies on PTX (Parallel Thread eXecution), a low-level parallel thread execution virtual machine and Instruction Set Architecture (ISA), that exposes the GPU as a data-parallel computing device. CUDA allows the use of PTX in a easy way from high level programming languages (e.g. C/C++) [3].

A CUDA program [13] consists of a serial program executing on the CPU (host) which calls one or more parallel kernels executing on the GPU (device) (Fig. 3, 4). A kernel executes in parallel across a set of parallel threads. The programmer or compiler organizes these threads in a hierarchic structure (Fig. 3, 4), i.e. thread blocks and grids of thread blocks. The GPU instantiates a kernel program on a grid of parallel thread blocks. Each thread within a thread block executes an instance of the kernel, and has a thread ID within its thread block, program counter, registers, per-thread private memory, inputs, and output results. A thread block is a set of concurrently executing threads that can cooperate among themselves through barrier synchronization and shared memory. A thread block has a block ID within its grid.

A grid is an array of thread blocks that execute the same kernel, read inputs from global memory, write results to global memory, and synchronize between dependent kernel calls. In the CUDA parallel programming model, each thread has a per-thread private memory space used for register spills, function calls, and C automatic array variables. Each thread block has a per-block shared memory space used for inter-thread communication, data sharing, and result sharing in parallel algorithms. Grids of thread blocks share results in Global Memory space after kernel-wide global synchronization (Fig. 3, 5) At the same level of global memory there are also the constant and texture memories (Fig. 5). Constant memory is mainly used to store read-only constants. Texture memory is used in graphics applications for OpenGL and DirectX rendering pipelines, but it can be also useful in general-purpose programming, for example for linear interpolation.

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

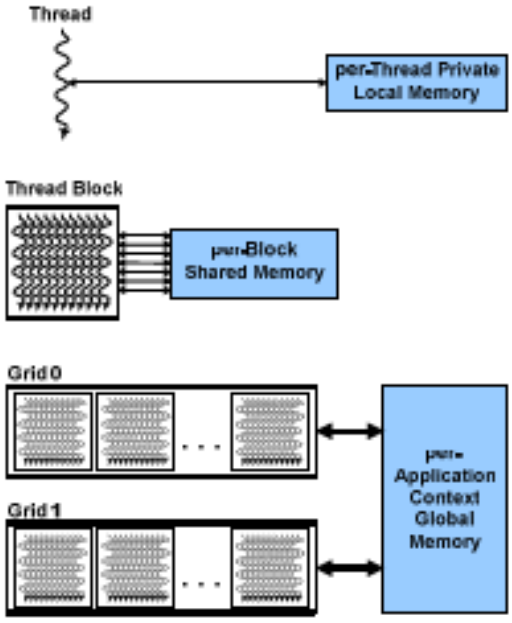


Figure 3. CUDA hierarchy of threads, blocks, and grids, with corresponding per-thread private, per-block shared, and per-application global memory spaces (from [14]).

CUDA's hierarchy of threads maps to a hierarchy of processors on the GPU; a GPU executes one or more kernel grids; a Streaming Multiprocessor (SM) executes one or more thread blocks; and CUDA cores and other execution units in the SM execute threads. The SM executes threads in groups of 32 threads called a warp. While programmers can generally ignore warp execution for functional correctness and think of programming one thread, they can greatly improve performance by having threads in a warp execute the same code path and access memory in nearby addresses.

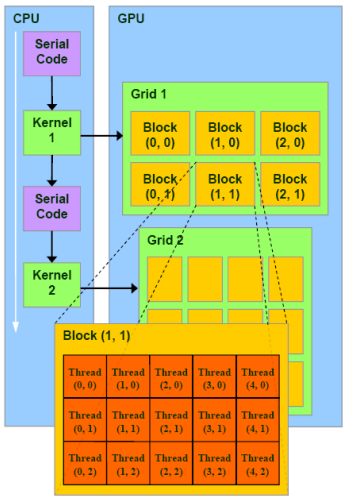


Figure 4. The CPU and GPU: Serial code, Kernels, Grids and Blocks (NVIDIA documentation).

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

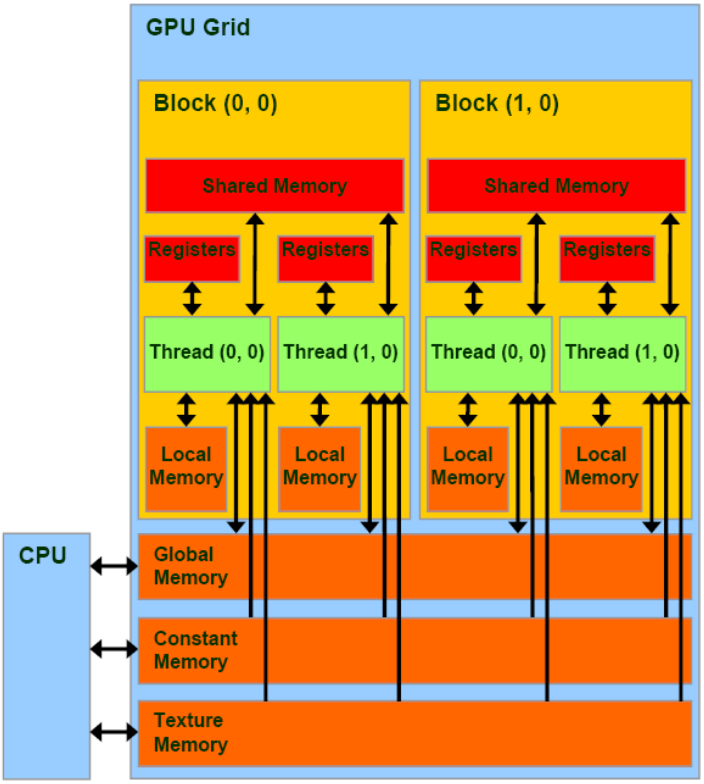


Figure 5. CUDA Memory Model (NVIDIA documentation).

3.1. Compute Capability

The Compute Capability (CC) of a NVIDIA GPU is represented by a version number, sometimes called its SM version. This version number identifies the features supported by the hardware and is used by applications at runtime to determine which hardware features and/or instructions are available on the present device.

The Compute Capability version comprises a major and a minor version number (x.y): devices with the same major revision number are of the same core architecture. The major revision number is for devices based on the Tesla architecture, 2 for devices based on the Fermi architecture, 3 for devices based on the Kepler architecture, and 5 for devices based on the Maxwell architecture. The 4 major version number is currently not used.
The minor revision number corresponds to an incremental improvement to the core architecture, possibly including new features.

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

The table below reports the main technical specifications for the various CCs.

	Compute Capability							
Technical Specifications	**1.1**	**1.2**	**1.3**	**2.x**	**3.0**	**3.5**	**5.0**	**5.2**
Maximum dimensionality of grid of thread blocks	2			3				
Maximum x-dimension of a grid of thread blocks	65535				2^{31}-1			
Maximum y- or z-dimension of a grid of thread blocks	65535							
Maximum dimensionality of thread block	3							
Maximum x- or y-dimension of a block	512			1024				
Maximum z-dimension of a block	64							
Maximum number of threads per block	512			1024				
Warp size	32							
Maximum number of resident blocks per multiprocessor	8				16		32	
Maximum number of resident warps per multiprocessor	24	32		48	64			
Maximum number of resident threads per multiprocessor	768	1024		1536	2048			
Number of 32-bit registers per multiprocessor	8 K	16 K		32 K	64 K			
Maximum number of 32-bit registers per thread	128			63		255		
Maximum amount of shared memory per multiprocessor	16 KB			48 KB			64 KB	96 KB
Maximum amount of shared memory per thread block	16 KB			48 KB				
Number of shared memory banks	16			32				
Amount of local memory per thread	16 KB			512 KB				
Constant memory size	64 KB							
Cache working set per multiprocessor for constant memory	8 KB						10 KB	

4. The G80 Tesla Architecture

The management of three different programmable processors in a single 3D pipeline led to unpredictable bottlenecks and balancing the throughput of each stage was difficult. So, as next step of GPU evolution, a "unified shader architecture" with 128 processing elements distributed among eight

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

shader cores was presented by NVIDIA. Each shader core could be assigned to any shader task, eliminating the need for stage-by-stage balancing and greatly improving overall performance.

NVIDIA's GeForce 8800, with a fabrication process of 90 nm, was the first product of the new GPU line (GeForce 8800, Quadro FX 5600, Tesla C870) named Tesla. Released in November 2006, the G80 based GeForce 8800 introduced several key innovations in the GPU architecture [14]:

- it replaces the separate vertex and pixel pipelines with a single, unified processor that executed vertex, geometry, pixel, and computing programs;
- it utilize a scalar thread processor, with full integer and floating point (IEEE 754-1985 single-precision floating point standard) operations, eliminating the need for programmers to manually manage vector registers;
- it introduces the single-instruction multiple-thread (SIMT) execution model where multiple independent threads execute concurrently using a single instruction; Thread Execution Manager enables thousands of concurrent threads per GPU;
- it introduces shared memory and barrier synchronization for inter-thread communication;
- it supports C, allowing programmers to use the power of the GPU without having to learn a new programming language.

The SM is a unified graphics and computing multiprocessor that executes vertex, geometry, and pixel-fragment shader programs and parallel computing programs. The SM consists of eight streaming processor (SP) cores, two special function units (SFUs), a multithreaded instruction fetch and issue unit (MT Issue), an instruction cache, a read-only constant cache, and a 16-Kbyte read/write shared memory [1]. Two SMs constitute a Texture/Processor Cluster (TPC) (Fig. 6 left).

Each of SM blocks operates in a Single Instruction Multiple Data (SIMD) fashion. All units inside a block execute the same instruction, but they apply it on different data elements. This type of parallelism is called data parallelism. An algorithm should contain enough data parallelism in order to benefit from this type of architecture.

Each core runs at 2x the shader clock. Running execution units at a higher clock rate allows a chip to achieve a given throughput with a lower numbers of execution units, which is essentially an area optimization, but the clocking logic for the faster cores requires more power.

Each SM has on-chip memory of the four following types:

- a high number of registers;
- a shared memory, a on chip memory inside each SM. It has a higher bandwidth and a lower latency than global memory;
- a read-only constant cache that is shared by all the processors and speeds up reads from the constant memory space, which is implemented as a read-only region of device memory;
- a read-only texture cache that is shared by all the processors and speeds up reads from the texture memory space, which is implemented as a read-only region of device memory.

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

The on-chip memory has virtually no latency and is not limited by the bandwidth of the memory bus. Each SM accesses the texture cache via a texture unit that implements the various addressing modes and (bi)linear interpolation.

The global memory is the GPU's off chip memory, which is accessible by all SMs, whit a latency of about 400 to 600 clock cycles.

There is no support for Double Precision and ECC Memory, and L1 and L2 caches are not present.

Each SP is able to run 1 Multiply-Add (MAD, 2 FP OPs) and 1 Multiply (MUL, 1 FP OP) instructions per clock. Special Function Units (SFUs) execute transcendental instructions such as sin, cosine, reciprocal, and square root. Each SFU executes one instruction per thread, per clock. The SM schedules threads in groups of 32 parallel threads called warps.

As the 128 stream processors of the GeForce 8800 GTX are clocked at 1.35 GHz and considering that each SP can execute three floating-point operations per clock, the theoretical peak performance is 518.4 GigaFLOPS [3×1350 MHz×128 SPs = 518.4 GigaFLOPs].

In June 2008, NVIDIA introduced a major revision to the G80 architecture. The second generation unified architecture, named GT200 (first introduced in the GeForce GTX 280, Quadro FX 5800, and Tesla T10 GPUs), manufactured with a process of 65 nm, increased the number of streaming processor cores (also referred to as CUDA cores) from 128 to 240. Each processor register file was doubled in size, allowing a greater number of threads to execute on-chip concurrently. The peak theoretical performance of GT200 is about 1 TeraFLOPS.

To improve memory access efficiency, hardware memory access coalescing was added. DP floating point support was also added to enable the execution of large and precision-critical scientific and high-performance computing applications, although the DP speed was only 1/8 of SP speed. The SM structure is very similar to that of G80, but in this case a TPC consists of three SPs (Fig. 6 right).

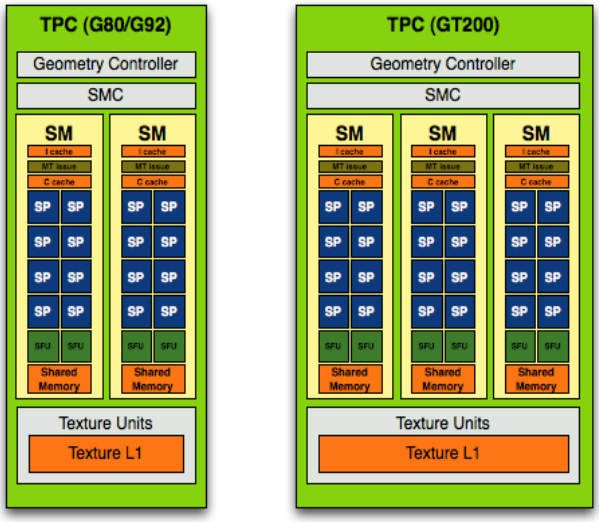


Figure 6. The G80 TPC (Texture/Processor Cluster) (left) vs. the GT200 TPC (right).

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

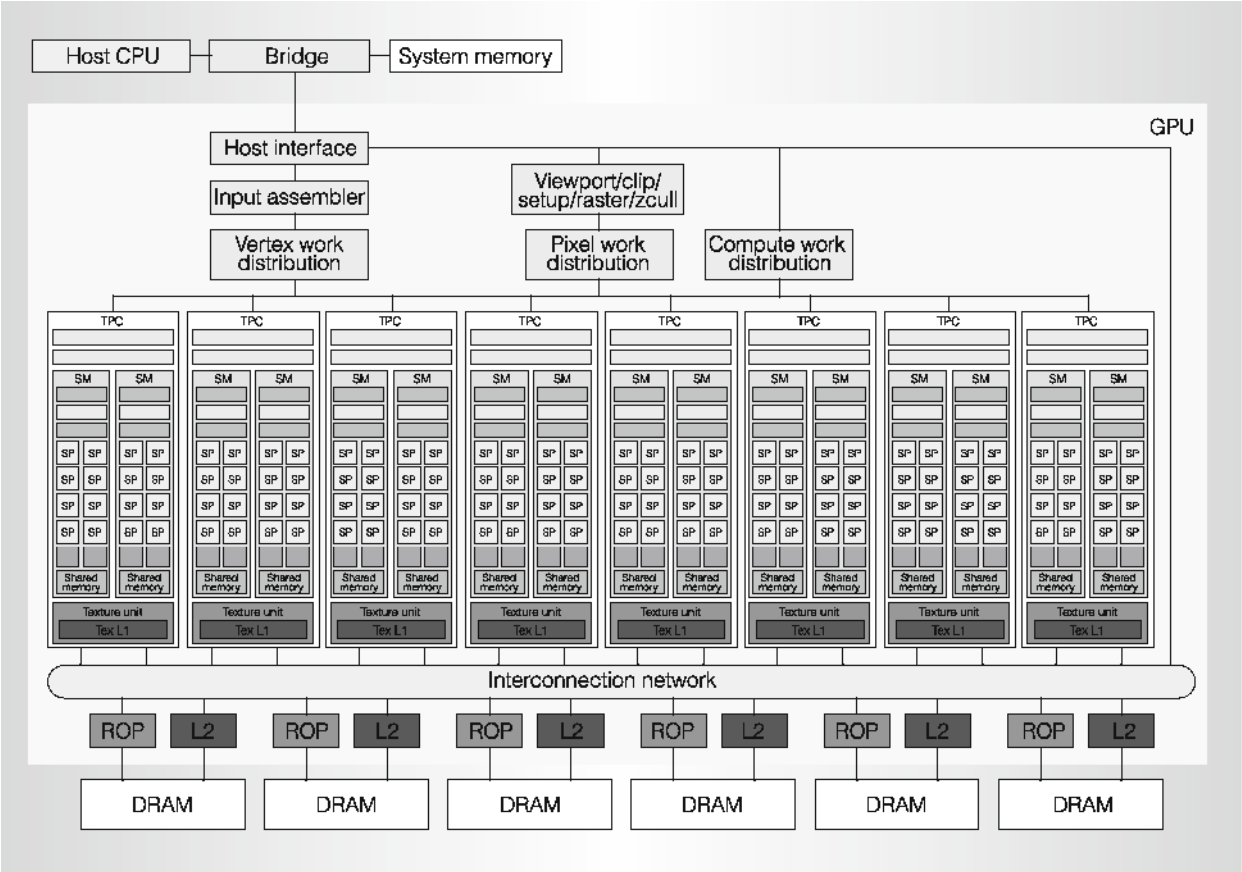


Figure 7. Tesla unified graphics and computing GPU architecture. TPC: texture/processor cluster; SM: streaming multiprocessor; SP: streaming processor; Tex: texture, ROP: raster operation processor (from [1]).

Figure 7 shows a block diagram of a GeForce 8800 GPU with 128 streaming processor (SP) cores organized as 16 SMs in eight independent TPCs.

5. The Fermi Architecture

The Fermi architecture, released in 2010, gives a number of key improvements over previous generations, in part suggested by users experience.

The first Fermi based GPU [14], implemented with 3.0 billion transistors using a production process of 40 nm, features up to 512 SPs (or CUDA cores) organized in 16 SMs of 32 cores each.

Each core (Fig. 8) has a fully pipelined integer arithmetic logic unit (ALU) and floating point unit (FPU) able to execute a floating point or integer instruction per clock for a thread. Prior GPUs used IEEE 754-1985 floating point arithmetic. The Fermi architecture implements the new IEEE 754-2008 floating-point standard, providing the fused multiply-add (FMA) instruction for both single and double

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

precision arithmetic. FMA is more accurate than a multiply-add (MAD) instruction as there is a single final rounding step.

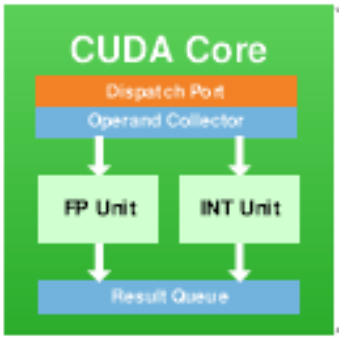


Figure 8. A Fermi core.

Figure 9 shows the Streaming Multiprocessor of the Fermi architecture.

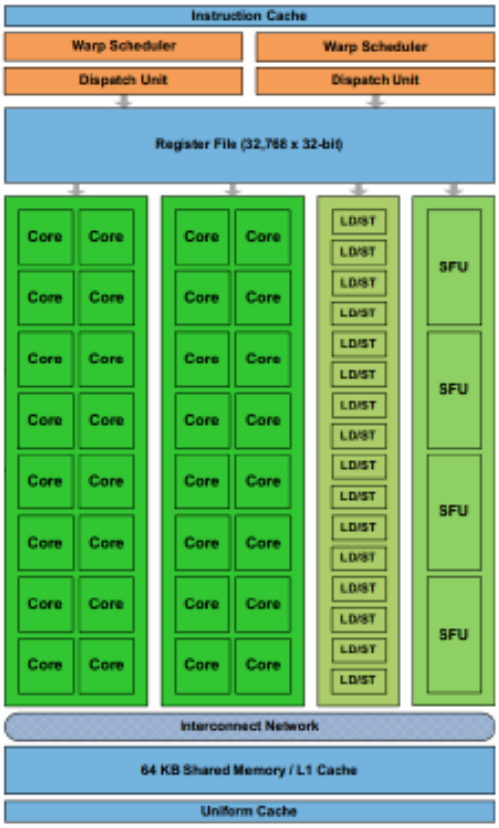


Figure 9. Fermi Streaming Multiprocessor (SM) (from [14]).

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

Each SM has 4 execution blocks:

- 2 computational blocks containing 16 cores (SPs);
- 1 memory instruction blocks with 16 Load/Store units (LD/ST), allowing source and destination addresses to be calculated for sixteen threads per clock;
- a special function block with 4 Special Function Units (SFU).

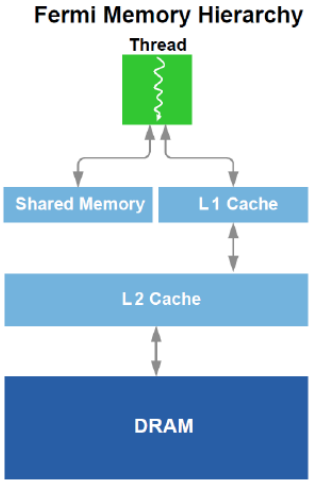
A different number of SMs can be enabled in the various kinds of cards, in order to match the different needs of users.

The Fermi architecture can perform up to 16 double precision FMA operations per SM and per clock, resulting in DP applications performing up to 4.2x faster than GT200.

The SM schedules threads in groups of 32 parallel threads called warps. Each SM features two warp schedulers and two instruction dispatch units, allowing two warps to be issued and executed concurrently [15]. Fermi's dual warp scheduler selects two warps, and issues one instruction from each warp to a group of sixteen cores, sixteen load/store units, or four SFUs. Using this model of dual-issue, Fermi achieves near peak hardware performance.

A true Cache Hierarchy (L1 and L2) is added. Tesla architectures G80 and GT200 have 16 KB of Shared memory per SM. In the Fermi architecture, each SM has 64 KB of on-chip memory that can be configured as 48 KB of Shared memory with 16 KB of L1 cache or as 16 KB of Shared memory with 48 KB of L1 cache, depending on the kind of applications, giving significant performance improvements.

Moreover, Fermi features a 768 KB unified L2 cache, shared across the 16 SMs, that services all load, store, and texture requests. The L2 provides efficient, high speed data sharing across the GPU. Algorithms with non-continuous/irregular pattern of memory accesses greatly benefit from the cache hierarchy. Filter and convolution kernels that require multiple SMs to read the same data also benefit.



Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

The GPU supports up to a total of 6 GB of GDDR5 DRAM global memory with a 384-bit memory interface. A host interface connects the GPU to the CPU via PCI-Express. The GigaThread global scheduler distributes thread blocks to SM thread schedulers [15].

5.1 Main improvements of the Fermi architecture

In the following we list the most important improvements of the Fermi architecture over the previous one Tesla [14].

The total number of cores increased from 128 to 240.

The number of Special Function Units (SFUs) per SM is increased fo four; moreover, the SFU pipeline is decoupled from the dispatch unit, allowing the dispatch unit to issue to other execution units while the SFU is occupied.

The Double Precision Performance increases from 1/8 to 1/2 of Single Precision performance, enabling DP to be routinely used for applications requiring more precision without great loss of performance.

In Tesla GT200, the integer ALU was limited to 24-bit precision for multiply operations; as a result, multi-instruction emulation sequences were required for integer arithmetic. In Fermi, the newly designed integer ALU supports full 32-bit precision for all instructions. The integer ALU is also optimized to efficiently support 64-bit and extended precision operations. Various instructions are supported, including Boolean, shift, move, compare, convert, bit-field extract, bit-reverse insert, and population count.

The Shared Memory size is increased up to 48 KB as many CUDA applications request more than 16 KB of SM shared memory to execute with a good efficiency.

The ECC support is added, particularly useful in GPU computing to safely execute large applications on a high number of GPUs in datacenter installations, and to ensure that data-sensitive applications are protected from memory errors.

Faster context switches between application programs is introduced. Like CPUs, GPUs support multitasking through the use of context switching, where each program receives a time slice of the processor's resources. The Fermi pipeline is optimized to reduce the cost of an application context switch to below 25 microseconds, a significant improvement over last generation GPUs.

Fermi supports concurrent kernel execution, where different kernels of the same application context can execute on the GPU at the same time. Concurrent kernel execution allows programs that execute a number of small kernels to utilize the whole GPU.

Faster graphics and compute interoperation have been introduced.

Faster read-modify-write atomic operations are now available. Thanks to a combination of more atomic units in hardware and the addition of the L2 cache, atomic operations performance is up to 20×

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

faster in Fermi compared to the GT200 generation. Fermi supports atomicAdd, atomicCAS (Compare-And-Swap), and atomicExch (exchange a data with the content of a memory location). Fermi supports the new Parallel Thread eXecution (PTX) 2.0 instruction set. PTX 2.0 introduces several new features over the previous features that greatly improve GPU programmability, accuracy, and performance. These include: full IEEE 32-bit floating point precision, unified address space for all variables and pointers, 64-bit addressing, and new instructions for OpenCL and DirectCompute. Moreover, PTX 2.0 was specifically designed to provide full support for the C++ programming language.

With PTX 2.0, a unified address space unifies all three address spaces (local, shared, and global memory) into a single, continuous address space. A single set of unified load/store instructions operate on this address space. The 40-bit unified address space supports a Terabyte of addressable memory.

In addition to previous support for OpenCL and DirectCompute APIs, Fermi gives hardware support for OpenCL and DirectCompute surface instructions with format conversion, allowing graphics and compute programs to easily operate on the same data.

The table below compares the main features of G80, GT200 and Fermi architectures.

GPU	G80	GT200	Fermi
Transistors	681 million	1.4 billion	3.0 billion
CUDA Cores	128	240	512
Double Precision Floating Point Capability	None	30 FMA ops / clock	256 FMA ops /clock
Single Precision Floating Point Capability	128 MAD ops/clock	240 MAD ops / clock	512 FMA ops /clock
Special Function Units (SFUs) / SM	2	2	4
Warp schedulers (per SM)	1	1	2
Shared Memory (per SM)	16 KB	16 KB	Configurable 48 KB or 16 KB
L1 Cache (per SM)	None	None	Configurable 16 KB or 48 KB
L2 Cache	None	None	768 KB
ECC Memory Support	No	No	Yes
Concurrent Kernels	No	No	Up to 16
Load/Store Address Width	32-bit	32-bit	64-bit

6. The Kepler Architecture

The Kepler GK110 architecture, delivered at the end of 2012, comprises 7.1 billion transistors, more than twice with respect to the Fermi architecture.

A principal design goal for the Kepler architecture was improving power efficiency. The 28nm manufacturing process plays an important role in lowering power consumption, but many GPU

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

architecture modifications were required to further reduce power consumption while maintaining great performance.

Similar to GK104, the cores within the new GK110 use the primary GPU clock rather than the 2x shader clock as in previous architectures. For Kepler, the priority was performance per watt. While many optimizations are beneficial both for area and power, NVIDIA chose to optimize for power even at the expense of some added area cost, with a larger number of processing cores running at the lower, less power consuming GPU clock.

As a result, the Kepler architecture improves both performance and power efficiency, delivering up to 3x the performance per watt of Fermi. Another design goal for the Kepler GK110 was to significantly increase the GPU's delivered double precision performance. Indeed, Kepler GK110 will provide over 1 TFlop of double precision throughput with greater than 80% DGEMM (Double precision GEneral Matrix Multiplication) efficiency versus 60-65% on the prior Fermi architecture.
The Kepler architecture [16] is quite similar to Fermi, but has an important difference. The number of SPs per Streaming Multiprocessor, on Kepler called SMX, increased with a factor 6 from 32 to 192. The clock frequency decreases from roughly 1.5 GHz to about 1 GHz, following the trend of increasing the total power using more cores running at a lower clock frequency.

Key features of the architecture include [16]:

- the new SMX processor architecture:
- an enhanced memory subsystem, offering additional caching capabilities, more bandwidth at each level of the hierarchy, and a fully redesigned and substantially faster DRAM I/O implementation;
- hardware support throughout the design to enable new programming model capabilities.

Kepler GK110 supports the CUDA Compute Capability 3.5. The following table [16] compares some important parameters for Fermi and Kepler GPU architectures:

	FERMI GF100	FERMI GF104	KEPLER GK104	KEPLER GK110
Compute Capability	2.0	2.1	3.0	3.5
Threads / Warp	32	32	32	32
Max Warps / Multiprocessor	48	48	64	64
Max Threads / Multiprocessor	1536	1536	2048	2048
Max Thread Blocks / Multiprocessor	8	8	16	16
32-bit Registers / Multiprocessor	32768	32768	65536	65536
Max Registers / Thread	63	63	63	255
Max Threads / Thread Block	1024	1024	1024	1024
Shared Memory Size Configurations (bytes)	16K 48K	16K 48K	16K 32K 48K	16K 32K 48K
Max X Grid Dimension	2^16-1	2^16-1	2^32-1	2^32-1
Hyper-Q	No	No	No	Yes
Dynamic Parallelism	No	No	No	Yes

Compute Capability of Fermi and Kepler GPUs

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

Each of the Kepler GK110 SMX units feature 192 single-precision CUDA cores, and each core has fully pipelined floating-point and integer arithmetic logic units. Kepler retains the full IEEE 754-2008 compliant single- and double-precision arithmetic introduced in Fermi, including the fused multiply-add (FMA) operation. Kepler GK110's SMX also retains the special function units (SFUs) for fast approximate transcendental operations as in previous-generation GPUs, providing 8x (32) the number of SFUs of the Fermi GF110 SM (Fig. 10).

The SMX warp schedulers also changed. They can now schedule an entire warp, 32 threads, every cycle, while on Fermi a scheduler only scheduled 16 threads. Each SMX features four warp schedulers and eight instruction dispatch units, allowing four warps to be issued and executed concurrently So, 128 threads can be scheduled every cycle, but there are 192 cores. In order to keep all cores busy, warp schedulers have to issue two independent instructions, provided the kernel should supply enough independent instructions to exploit this parallelism.
Unlike Fermi, which did not permit double precision instructions to be paired with other instructions, Kepler GK110 allows double precision instructions to be paired with other instructions. The Kepler scheduler is more simple than the Kermi's one, allowing a saving of energy.

Figure 10. Kepler SMX: 192 Single Precision CUDA cores, 64 Double Precision units, 32 Special Function Units (SFU), and 32 load/store units (LD/ST) (from [16]).

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

The number of registers that can be accessed by a thread has been quadrupled in GK110, allowing each thread access to up to 255 registers. Codes that exhibit high register pressure may achieve substantial speedups as a result of the increased available per-thread register count.

To further improve performance, Kepler implements a new Shuffle instruction, which allows threads within a warp to share data. Shuffle supports arbitrary indexed references and offers a performance advantage over shared memory, as a store-and-load operation is carried out in a single step. Shuffle also can reduce the amount of shared memory needed per thread block. In the case of FFT, which requires data sharing within a warp, a 6% performance gain can be obtained just by using Shuffle.

Throughput of global memory atomic operations on Kepler GK110 is greatly improved compared to the Fermi generation. Atomic operation throughput to a common global memory address is improved by 9x to one operation per clock. Atomic operation throughput to independent global addresses is also significantly accelerated, and logic to handle address conflicts has been made more efficient. Atomic operations can often be processed at rates similar to global load operations. This speed increase makes atomics fast enough to use frequently within kernel inner loops, eliminating the separate reduction passes that were previously required by some algorithms to consolidate results. In addition to atomicAdd, atomicCAS, and atomicExch (which were also supported by Fermi and Kepler GK104), GK110 supports the following: atomicMin, atomicMax, atomicAnd, atomicOr, atomicXor.

Kepler Memory Subsystem

Kepler's memory hierarchy is organized similarly to Fermi. The Kepler architecture supports a unified memory request path for loads and stores, with an L1 cache per SMX multiprocessor. Kepler GK110 also enables compiler-directed use of an additional new cache for read-only data.

In the Kepler GK110 architecture, as in the previous generation Fermi architecture, each SMX has 64 KB of on-chip memory that can be configured as 48 KB of Shared memory with 16 KB of L1 cache, or as 16 KB of Shared memory with 48 KB of L1 cache. In addition, Kepler allows a 32KB / 32KB split between Shared memory and L1 cache. The Shared memory bandwidth for 64b and larger load operations is also doubled compared to the Fermi SM, to 256B per core clock.

In addition to the L1 cache, Kepler introduces a 48KB cache for data that is known to be read-only for the duration of the function. In the Fermi generation, this cache was accessible only by the Texture unit. In Kepler, the cache is directly accessible to the SM for general load operations. Use of the read-only path can be managed automatically by the compiler or explicitly by the programmer.

The Kepler GK110 GPU features 1536 KB of dedicated L2 cache memory, twice the amount of L2 available in the Fermi architecture. The bandwidth of the L2 cache on Kepler has been increased to 2x of the bandwidth per clock available in Fermi.

Like Fermi, Kepler's register files, shared memories, L1 cache, L2 cache and DRAM memory are protected by a Single-Error Correct Double-Error Detect (SECDED) ECC code. The ECC on-vs-off performance delta has been reduced by an average of 66%.

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

The GPU's dedicated hardware Texture units are a valuable resource for compute programs with a need to sample or filter image data. The texture throughput in Kepler is significantly increased compared to Fermi [16] as each SMX unit contains 16 texture filtering units, a 4x increase vs. the Fermi GF110 SM. In addition, Kepler changes the way texture state is managed. In the Fermi generation, for the GPU to reference a texture, it had to be assigned a "slot" in a fixed-size binding table prior to grid launch. The number of slots in that table limits how many unique textures a program can read from at run time. In Kepler, the additional step of using slots is no longer necessary: texture state is now saved as an object in memory and the hardware fetches these state objects on demand. This effectively eliminates any limits on the number of unique textures that can be referenced by a compute program

A full Kepler GK110 implementation includes 15 SMX units, for a total of 2880 cores, and six 64-bit memory controllers (Fig. 11). Different products will use a smaller number of SMXs.

Figure 11. Kepler GK110 block diagram (from [16]).

6.1 Main improvements of the Kepler architecture

The following new features in Kepler GK110 enable increased GPU utilization, simplify parallel program design, and aid in the deployment of GPUs across the spectrum of compute environments ranging from personal workstations to supercomputers. Heterogeneous computing becomes faster, and applicable to a broader set of applications [16].

Dynamic Parallelism

Dynamic Parallelism is a new feature introduced with Kepler GK110 that allows the GPU to generate new work for itself, synchronize on results, and control the scheduling of that work via dedicated, accelerated hardware paths, all without involving the CPU.
With previous GPUs all work was launched from the host CPU, would run to completion, and return results back to the CPU.
In Kepler GK110 any kernel can launch another kernel, and can create the necessary streams, events and manage the dependencies needed to process additional work without the need for host CPU interaction. This architectural innovation makes it easier for developers to create and optimize recursive and data-dependent execution patterns, and allows more of a program to be run directly on GPU. The CPU can then be free for additional tasks, or the system could be configured with a less powerful CPU to carry out the same workload.

The ability of a kernel to launch additional workloads based on intermediate, on-GPU results, allows programmers load-balance work to focus the bulk of their resources on the areas of the problem that either require the most processing power or are most relevant to the solution. One example is dynamically setting up a grid for a numerical simulation which typically requires that grid cells are smaller in regions of greatest change.
With Dynamic Parallelism, the grid resolution can be changed dynamically at runtime in a data-dependent manner. Starting with a coarse grid, the simulation can "zoom in" on areas of interest while avoiding unnecessary calculation in areas with little change.

Hyper-Q

Hyper-Q enables multiple CPU cores to launch work on a single GPU simultaneously, thereby dramatically increasing GPU utilization and at the same time reducing CPU idle times.
Hyper-Q increases the total number of connections (work queues) between the host and the CUDA Work Distributor (CWD) logic in the GPU by allowing 32 simultaneous, hardware-managed connections (compared to the single connection available with Fermi).
Each CUDA stream is managed within its own hardware work queue, inter-stream dependencies are optimized, and operations in one stream will no longer block other streams, enabling streams to be executed concurrently.
Applications that previously encountered false serialization across tasks, thereby limiting achieved GPU utilization, can obtain dramatic performance increase without changing any existing code.
The CUDA Work Distributor (CWD) Unit in Kepler is able to dispatch 32 active grids, which is double the capacity of the Fermi CWD. Also a new Grid Management Unit (GMU) was introduced in

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

Kepler GK110.

Grid Management Unit (GMU)

Enabling Dynamic Parallelism requires an advanced, flexible grid management and dispatch control system. The new GK110 Grid Management Unit (GMU) manages and prioritizes grids to be executed on the GPU. The GMU can pause the dispatch of new grids and queue pending and suspended grids until they are ready to execute, providing the flexibility to enable powerful runtimes, such as Dynamic Parallelism. The GMU ensures that both CPU- and GPU- generated workloads are properly managed and dispatched.

NVIDIA GPUDirect

NVIDIA GPUDirect is a capability that enables GPUs within a single computer, or GPUs in different servers located across a network, to directly exchange data without using the CPU/system memory. The RDMA feature in GPUDirect increases the data throughput and reduces latency.
It also reduces demands on system memory bandwidth and frees the GPU DMA engines for use by other CUDA tasks. Kepler GK110 also supports other GPUDirect features including Peer-to-Peer and GPUDirect for Video.

6.2 NVIDIA GTX Titan Black

In this subsection and in the next we present two GeForce GPUs with Kepler architecture, available in the IEIT-CNR HPC Laboratory.

In February 2014 Nvidia launched the GTX Titan Black [17] into retail channels. GTX Titan Black is at launch time the top card in Nvidia's GeForce product line, the best one available for both gaming and GPU computing.

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

GTX Titan Black is based on the GK110 (Kepler) architecture. Its 15 SMXs, with 192 cores each, are all fully enabled, giving it 2,880 (192x15) CUDA cores and 240 texture (Fig. 12). The memory frequency is 1.75GHz (7GHz effective).

Figure 12. GTX Titan Black utilizes a fully enabled Kepler GK110 GPU with 15 SMXs (from [17]).

GTX Titan Black has a full 6GB of GDDR5, as well as the maximum double precision performance; in non-Titan GK110 products, Nvidia artificially limits the performance of the FP64 CUDA cores to 1/8 of their potential as a way of distinguishing the product lines.

The base clock is 889MHz here, making it the fastest reference GK110 product available, and the boost clock is 980MHz.

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

Table 1 compares four GeForce GTX cards based on Kepler GK110.

	Nvidia GeForce GTX Titan Black 6GB	Nvidia GeForce GTX 780 Ti 3GB	Nvidia GeForce GTX Titan 6GB	Nvidia GeForce GTX 780 3GB
GPU				
Architecture	Kepler	Kepler	Kepler	Kepler
Codename	GK110	GK110	GK110	GK110
Base Clock	889MHz	876MHz	836MHz	836MHz
Boost Clock	980MHz	928MHz	876MHz	900MHz
Stream Processors	2,880	2,880	2,688	2,304
Layout	5 GPCs, 15 SMXs	5 GPCs, 15 SMXs	5 GPCs, 14 SMXs	4 GPCs, 12 SMXs
Rasterisers	5	5	5	4
Tesselation Units	15	15	14	12
Texture Units	240	240	224	194
ROPs	48	48	48	48
F P 6 4 Performance	1/3 FP32	1/24 FP32	1/3 FP32	1/24 FP32
Transistors	7.1 billion	7.1 billion	7.1 billion	7.1 billion
Die Size	561 mm^2	561 mm^2	561 mm^2	561 mm^2
Process	28 nm	28 nm	28 nm	28 nm
Memory				
Amount	6GB GDDR5	3GB GDDR5	6GB GDDR5	3GB GDDR5
Frequency	1.75GHz (7GHz Effective)	1.75GHz (7GHz Effective)	1.5GHz (6GHz Effective)	1.5GHz (6GHz Effective)
Interface	384-bit	384-bit	384-bit	384-bit
Bandwidth	336 GB/sec	336 GB/sec	288 GB/sec	288 GB/sec
Card Specifications				
Power Connectors	1 x6-pin, 1 x8-pin PCI-E	1 x6-pin, 1 x8-pin PCI-E	1 x6-pin, 1 x8-pin PCI-E	1 x6-pin, 1 x8-pin PCI-E
Stock Card Length	267mm	267mm	267mm	267mm
TDP	250W	250W	250W	250W
Typical Price	£785	£500	£770	£385

Table 1. Comparison of Nvidia's consumer GK110 products (from [17]).

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

6.3 NVIDIA Geforce GTX 760M

It is a high-range graphics card for notebooks, based on GK106 Kepler architecture.

The GK106 GPU has 5 blocks of cores (or shader) called SMX, with 192 cores each; in the GTX 760M card are enabled 4 of these blocks, for a total of 768 (192x4) cores.

In next Table are reported some features of the card.

Architettura	Kepler
Cores	768
Core clock	657 MHz
Memory clock	4000 MHz
Memory bus	128 Bit
Memory type	GDDR5
Max. memory size	2048 MB
DirectX	DirectX 11, Shader 5.0
Power dissipation	55 Watt
Transistor	2.54 Billions
Process	28 nm
Announce date	30.05.2013
Interface	PCI Express 3.0 x16

7. The Maxwell Architecture

NVIDIA's first-generation Maxwell architecture (process 28 nm), released at the beginning of 2014, implements a number of architectural enhancements designed to extract even more performance per watt consumed. The first Maxwell-based GPUs, such as the GeForce GTX 750 Ti, are based on the GM107 GPU and designed for use in power-limited environments like notebooks and small form factor PCs [18, 19].

Maxwell introduces an all-new design for the Streaming Multiprocessor (SM) that dramatically improves both performance per watt and performance per area. Improvements to control logic partitioning, workload balancing, clock-gating granularity, scheduling, number of instructions issued per clock cycle, and many other enhancements allow the Maxwell SM (called SMM) to far exceed Kepler SMX efficiency.

The organization of the SM has also changed (Fig. 13). The number of CUDA Cores per SM has been reduced to a power of two (128 cores). Each SMM is now partitioned into four separate processing blocks, each with its own instruction buffer, scheduler and 32 CUDA cores, for a total of 128 cores per SMM. The Kepler approach of having a non-power-of-two number of CUDA cores, with some that are shared, has been eliminated. This new partitioning simplifies the design and scheduling logic, saving area and power, and reduces computation latency.

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

Pairs of processing blocks share four texture filtering units and a texture cache. The compute L1 cache function has now also been combined with the texture cache, and shared memory is a separate unit (similar to the approach used on G80, the first CUDA capable GPU), that is shared across all four blocks.

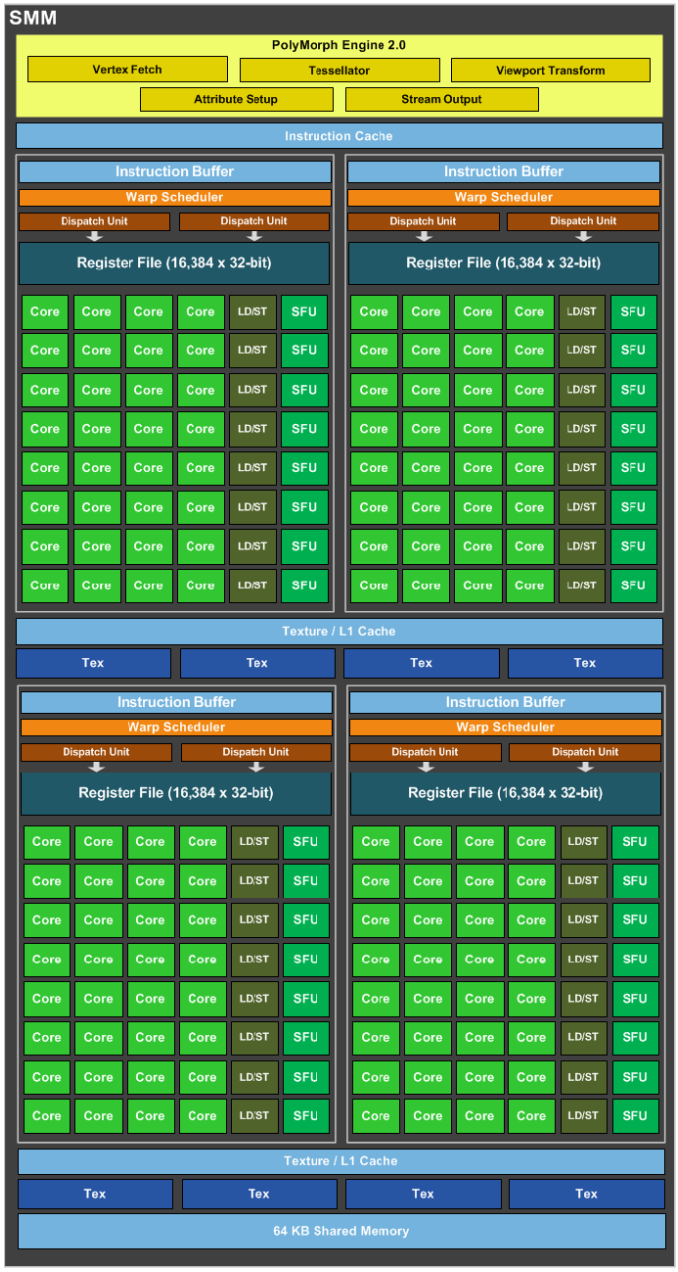


Figure 13. Maxwell SMM Block Diagram (from [19]).

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

Although the total number of cores per SM diminishes from 192 in Kepler to 128 in Maxwell, thanks to Maxwell's improved execution efficiency, performance per SMM is usually within 10% of Kepler performance, and the improved area efficiency of the SMM results in a number of CUDA cores per GPU substantially higher versus comparable Fermi or Kepler chips. The new SMM architecture enabled NVIDIA to increase the number of SMs to five in GM107, compared to two in GK107, with only a 25% increase in die area. GK107 has 1.7 times more CUDA cores and yields 25% more peak texture performance.

The SMM retains the same number of instruction issue slots per clock and reduces arithmetic latencies compared to the Kepler architecture.

As with SMX, each SMM has four warp schedulers, but unlike SMX, all core SMM functional units are assigned to a particular scheduler, with no shared units. The power-of-two number of CUDA Cores per partition simplifies scheduling, as each of SMM's warp schedulers issue to a dedicated set of CUDA Cores equal to the warp width. Each warp scheduler still has the flexibility to dual-issue (such as issuing a math operation to a CUDA Core in the same cycle as a memory operation to a load/store unit), but single-issue is now sufficient to fully utilize all CUDA Cores.

In terms of CUDA compute capability, Maxwell's SM is CC 5.0. SMM is similar in many respects to the Kepler architecture's SMX, with key enhancements aimed to improving efficiency without requiring significant increases in available parallelism per SM from the application. The register file size and the maximum number of concurrent warps in SMM are the same as in SMX (64k 32-bit registers and 64 warps, respectively), as is the maximum number of registers per thread (255). However the maximum number of active thread blocks per multiprocessor has been doubled over SMX to 32, which should result in an automatic occupancy improvement for kernels that use small thread blocks of 64 or fewer threads (assuming available registers and shared memory are not the occupancy limiter).

Another major improvement of SMM is that dependent arithmetic instruction latencies have been significantly reduced. Because occupancy is the same or better on SMM than on SMX, these reduced latencies improve utilization and throughput.

With the changes made in Maxwell's new SMM, the GPU's hardware units are utilized more often, resulting in greater performance and power efficiency. The GeForce GTX 750 Ti delivers over 1.7x more performance than GK107, with a TDP of just 60W.

A significant improvement in SMM is that it provides 64KB of dedicated shared memory per SM, unlike Fermi and Kepler, which partitioned the 64KB of memory between L1 cache and shared memory. The per-thread-block limit remains 48KB on Maxwell, but the increase in total available shared memory can lead to occupancy improvements. Dedicated shared memory is made possible in Maxwell by combining the functionality of the L1 and texture caches into a single unit.

Maxwell provides native shared memory atomic operations for 32-bit integers and native shared memory 32-bit and 64-bit compare-and-swap (CAS), which can be used to implement other atomic functions. In contrast, the Fermi and Kepler architectures implemented shared memory atomics using

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

a lock/update/unlock pattern that could be expensive in the presence of high contention for updates to particular locations in shared memory.

Maxwell architecture maintains the Dynamic Parallelism introduced with Kepler, which allows threads running on the GPU to launch additional kernels onto the same GPU. Moreover, with Maxwell whis feature is made available even in lower-power chips such as GM107.

The GM107 GPU contains five Maxwell Streaming Multiprocessors (SMM), and two 64-bit memory controllers (128-bit total) (Fig. 14). This is the full implementation of the chip, and is the configuration of the GeForce GTX 750 Ti.

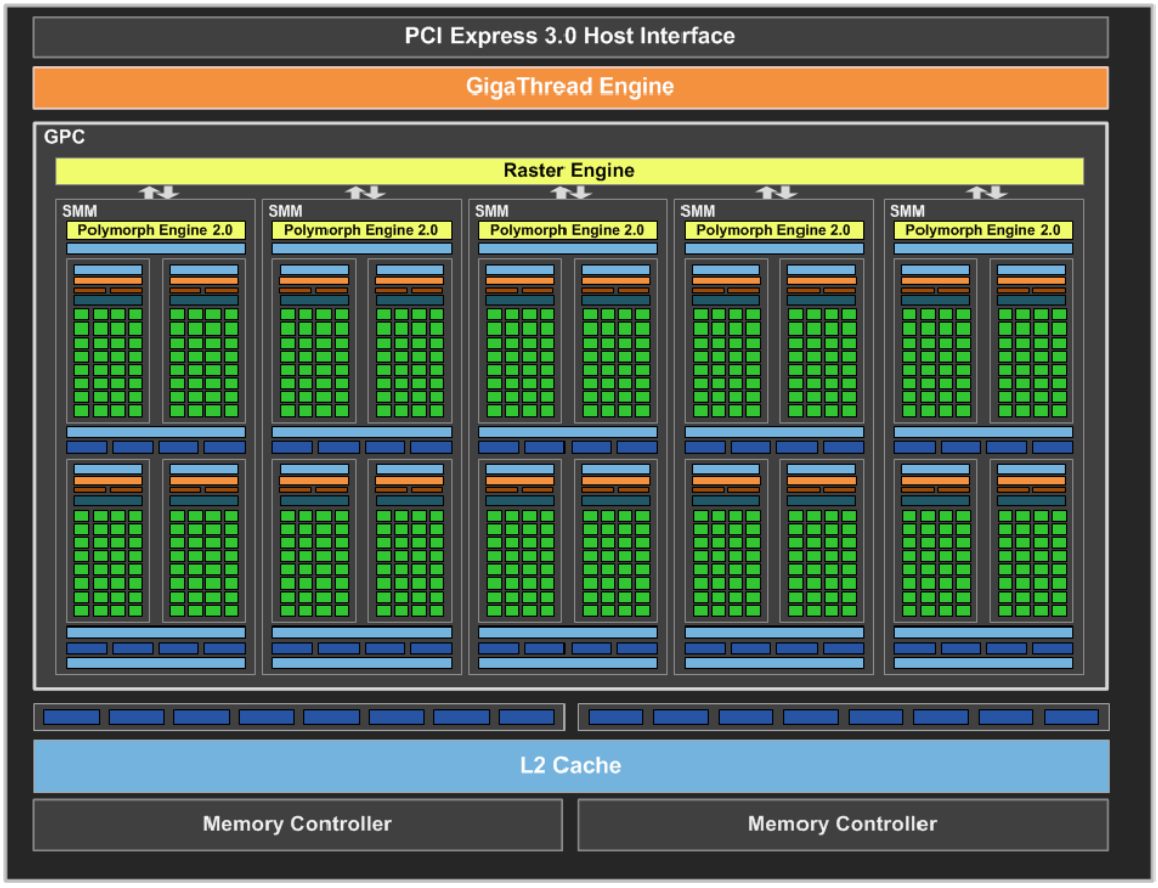


Figure 14. GM107 Full-Chip Block Diagram (from [19]).

Internally, all the units and crossbar structures have been redesigned, data flows optimized, power management significantly improved, and so on. The end result of all of these efforts is that Maxwell is able to deliver 2 times the performance/watt of Kepler, using the same 28nm manufacturing process.

The size of the L2 cache has been greatly increased, 2048KB in GM107 versus 256KB in GK107, providing an additional storage buffer that is shared across the GPU for texture requests, atomic

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

operations or anything else. With more cache located on-chip, fewer requests to the DRAM are needed, thus reducing memory bandwidth demand and ensuring that DRAM bandwidth is not a bottleneck.

The following table provides a high-level comparison of Maxwell vs. prior generation GK107 Kepler GPU.

Table 2. A Comparison of Maxwell GM107 to Kepler GK107

GPU	GK107 (Kepler)	GM107 (Maxwell)
SM	2 SMX	5 SMM
Cores per SM	192	128
CUDA Cores	384 (192x2)	640 (128x5)
Base Clock	1058 MHz	1020 MHz
GPU Boost Clock	N/A	1085 MHz
GFLOP/s	812.5	1305.6
Compute Capability	3.0	5.0
Shared Memory / SM	16KB / 48 KB	64 KB
Register File Size / SM	256 KB	256 KB
Active Blocks / SM	16	32
Memory Clock	5000 MHz	5400 MHz
Memory Bandwidth	80 GB/s	86.4 GB/s
L2 Cache Size	256 KB	2048 KB
TDP	64W	60W
Transistors	1.3 Billion	1.87 Billion
Die Size	118 mm^2	148 mm^2
Manufactoring Process	28 nm	28 nm

The efforts to improve the performance/watt ratio result in a very efficient GPU, particularly suited for situations where the small size and the low power consumption are key factors, such as small form factor PCs, small devices, etc.

7.1 NVIDIA Geforce GTX 750 Ti

In this subsection we present a GeForce GPUs with Maxwell architecture, available in the IEIT-CNR HPC Laboratory. ZOTAC Geforce GTX 750 is a graphics card with last generation NVIDIA Maxwell GPU clocked 1033 MHZ (boost 1111 MHZ) and with 2GB of GDDR5 at 5400 MHZ.

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

The Table below reports the main features.

Model	
Model ZT-70605-10M	
Interface	
Interface PCI Express 3.0 x16 (Compatible with 1.1)	
Chipset	
Chipset Manufacturer	NVIDIA
GPU	GeForce GTX 750 Ti
Process	28 nm
Core clock	1033 MHz (base) 1111 MHz (boost)
Cores per SMM	128
SMM	5
Cores (total)	640 (128x5)
Shader Clock	N/A
Memory	
Memory Clock	5400 MHz
Memory Size	2GB
Memory Interface	128-bit
Memory Type	GDDR5
3D API	
DirectX	DirectX 11.2 (feature level 11_0)
OpenGL	OpenGL 4.4

8. The forthcoming Pascal Architecture

The next-generation family of NVIDIA GPUs named Pascal is planned for 2016 and should be manufactured with a 20 nm technology. Pascal will include three key new features over previous generations: Stacked DRAM, Unified Memory, and NVLink [20].

Stacked DRAM or 3D Memory means to stack memory chips into dense modules with wide interfaces, and put them inside the same package as the GPU. This allows GPUs to get data from memory more quickly, boosting throughput and efficiency, Moreover, in this way it is possible to build more compact GPUs with more power into smaller devices. As a result, bandwidth is several times greater, memory capacity is more than twice and energy efficiency is quadrupled with respect to previous generation.

Unified Memory consists in enabling the CPU to access the GPU's memory, and the GPU to access the CPU's memory, so developers don't have to allocate resources between the two.

NVLink puts a fatter pipe between the CPU and GPU, allowing data to flow at more than 80GB per second, compared to the 16GB per second available now. So, speed of applications is no longer constrained by the speed at which data can be moved between the CPU and GPU.

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

NVIDIA has designed a module to house Pascal GPUs with NVLink. At one-third the size of the standard boards used today, they will put the power of GPUs into more compact form factors than ever before.

9. Conclusions

This report presents the evolution of graphics cards, from early hardware fixed pipelines, to programmable pipelines until modern many-core GPUs, focusing on NVIDIA GPUs, owing their widespread diffusion, high-performance and very rapid evolution. NVIDIA was the first company in the GPU computing market and is currently the leading. It results that evolution is very fast, driven by the growing and more and more sophisticated computer game market.

From early 2000 GPUs began to be exploited also for computationally demanding non graphical applications. In this report we denote with GPGPU (General-Purpose computation on Graphics Processing Units) the use of GPU for HPC applications using graphics programming languages and GPU computing the use of GPU for HPC using high level and general-purpose programming languages and tools (roughly from 2006).

The architectural evolution of modern many-core GPUs is extremely rapid. A new generation, characterized by important improvements, is released roughly every 18-24 months; from a generation to the subsequent typically the number of transistors doubles, the number of cores doubles [10], the clock frequency either remain stable or reduces, the power per flops greatly reduces, the size of memories as well as the size of L1 and L2 caches increases. Moreover, the speed of data transfer between CPU and GPU increases and also increases the integration level of GPU with CPU (for example CPU and GPU memories can be viewed using an uniform addressing mode).

So, many-core GPUs are increasingly well suited for both graphics and non-graphics applications. In particular, the area of GPU computing is more and more important, from notebooks and PCs/workstations to high-level parallel machines. Heterogeneous CPU-GPU architectures are today one of the most powerful, cheap and energy saving high-performance platforms, and surely they will be more and more in the coming years.

It is worth noting that, although presently the dominating approach is the use of discrete CPU and GPU, i.e. CPU and GPU lying in two different dies connected with some high-speed link, another approach proposes the integration of CPU and GPU on the same die (e.g. AMD Fusion APU).

Finally, although this report if focused on the architectural aspects, it is worth noting that also the software tools rapidly improve, making easier the development of complex high-performance application on such kind of machines.

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it

References

[1] E. Lindholm, J. Nickolls, S. Oberman, J. Montrym, NVIDIA Tesla: A Unified Graphics and Computing Architecture, IEEE Micro, v. 28, n. 2, pp. 39-55, 2008.

[2] J. Nickolls, I. Buck, M. Garland, K. Skadron, Scalable Parallel Programming with CUDA, ACM Queue, v. 6, n. 2, pp. 40-53, 2008.

[3] T.R. Halfhill, Parallel Processing with CUDA, Microprocessor Report, January 2008.

[4] B. Gaster et al., Heterogeneous Computing with OpenCL: Revised OpenCL 1, Newnes, 2012.

[5] W. Liu et al., A balanced programming model for emerging heterogeneous multi-core systems, Proceeding HotPar 2010 **(**2nd USENIX conference on Hot topics in parallelism).

[6] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A.E. Lefohn, T.J. Purcell, A Survey of General-Purpose Computation on Graphics Hardware, Computer Graphics Forum 26 (1), pp. 80-113, 2007.

[7] C. McClanahan, History and evolution of GPU architectures, 2010.

[8] GPU Gems 2, M. Pharr, Ed., Addison-Wesley, (Reading, MA), 2005.

[9] J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone, J.C. Phillips, GPU computing, Proceedings of the IEEE 96 (5), pp. 879-899, 2008.

[10] J. Nickolls, W.J. Dally, The GPU computing era, IEEE Micro, pp. 56-69, 2010.

[11] The Portland Group (http://www.pgroup.com).

[12] B. Eagan, G, Civario, R. Miceli, Investigating performance benefits from OpenACC kernel directives, In: Proc. PARCO 2013, pp. 616-625.

[13] CUDA C Programming Guide, PG-02829-001_v6.5, August 2014, NVIDIA.

[14] NVIDIA's Next Generation CUDA Compute Architecture: Fermi, NVIDIA Whitepaper, 2010.

[15] A.R. Brodtkorb, T.R. Hagen, M.L. Saetra, Graphics Processing Unit (GPU) programming strategies and trends in GPU computing, J. Parallel and Distributed Computing, 2012.

[16] NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110, NVIDIA Whitepaper, 2012.

[17] M. Lambert, Nvidia GeForce GTX TITAN Black Review: feat. ZOTAC, February 2014.

[18] M. Harris, 5 Things You Should Know About the New Maxwell GPU Architecture, NVIDIA, 2014.

[19] NVIDIA GeForce GTX 750 (Maxwell technology), NVIDIA Whitepaper, 2014.

[20] NVIDIA Updates GPU Roadmap; Announces Pascal; by S. Gupta on March 25, 2014, NVIDIA Blog.

Torino, Bologna, Genova, Milano, Padova, Pisa
www.ieiit.cnr.it
Headquarters: IEIIT - CNR - c/o Politecnico di Torino - Corso Duca degli Abruzzi, 24 – 10129 Torino – Italia
tel.: +39 011 564 5400/02/03 - fax: +39 011 564 5429 - e-mail: segreteria@ieiit.cnr.it; direttore@ieiit.cnr.it