Nvidia Fermi

Ahmed Labib

February 25, 2010

Introduction

Nvidia Fermi is the codename of nvidia's new GPU architecture. This architecture was announced by nvidia sometime in the second half of 2009 as a game changing architecture.

Competition & Long Term Strategy

Nvidia is facing tough competition from its two main rivals Intel and AMD. Both these two produce their own CPUs and GPUs while nvidia produces only GPUs. Nvidia has tried to somehow ease itself into a new market, which is the chipset market. Releasing custom nvidia chipsets which also incorporated a low end nvidia GPU which acted as an alternative to Intel's Media Accelerator. These chipsets showed superior performance graphics wise compared to Intel's solution. Several companies included these chipsets in their laptops to provide consumers with a better GPU experience in the low end graphics market. Also several companies included this chipset into what is called the Hybrid SLI architecture. Basically the Hybrid SLI architecture allows a laptop to have two GPUs on board; one low end weak one which drains very little battery power and one high end strong GPU. The Hybrid SLI architecture allows a user to dynamically switch between both based on his preferences. Nvidia also released a chipset for the new Atom processor which is widely used in current netbooks. Intel didn't like this competition and felt threatened by nvidia. Intel therefore didn't give nvidia the right to release chipsets for its new core i architecture and also sold the atom processor with its chipset cheaper than the processor alone. Thus driving nvidia totally out.

With nvidia locked out of the CPU and its chipset market it had only the GPU market to compete in. With the five main markets like seismology, supercomputing, university research workstations, defense and finance; which can represent about 1 billion dollar turnover; nvidia had to find a way to compete better. Nvidia saw a great opportunity in the use of the GPU's large amount of processor cores in general computing application. It saw it as a new and untapped market which is very promising and could allow nvidia to widen its market share and revenues.

Nvidia started to research in the use of GPUs for high performance computing applications such as protein folding, stock options pricing, SQL queries and MRI reconstruction. Nvidia released its G80 based architecture cards in 2006 to address these applications.

This was followed by the GT200 architecture in 2008/2009 which was build on G80's architecture but provided better performance. While these architectures targeted what is called GPGPU or general purpose GPU, they were somehow limited in the sense that they targeted only specific applications and not all applications. The drawbacks of the GPGPU model was that it required the programmer to possess intimate knowledge of graphics APIs and GPU architecture, problems had to be expressed in terms of vertex coordinates, textures and shader programs which greatly increased program complexity, basic programming features such as random reads and writes to memory were not supported which greatly restricted the programming model and finally the lack of double precision support meant that some scientific applications could not be run on the GPU.

Nvidia came around this by introducing two new technologies. The G80 unified graphics and compute architecture and CUDA which is software hardware architecture which allowed the GPU to be programmed with a variety of high level programming languages such as C and C++. Therefore instead of using graphics APIs to model and program problems the programmer can write C programs with CUDA extensions and target a general purpose massively parallel processor. This was of GPU programming is commonly known as "'GPU Computing"'. This allowed for a broader application support and programming language support.

Nvidia took what it has learned from its experience in the G80 and GT200 architectures to build a GPU with strong emphasize on giving a better GPU Computing experience while at the same time giving a superior graphics experience for normal GPU use. Nvidia based on its Fermi architecture on these two goal and regarded them as its long term strategy.

The Fermi architecture

Things needed to be changed

To allow Fermi to truly support "'GPU Computing"' some changes to the architecture had to be done. These changes can be summarized as follows:

- Improve double precision performance: many high performance computing application make use of double precision operations. Nvidia had to increase the the DP operations performance in order to attract these markets.
- ECC (Error Correcting Code): ECC allows users using GPU's for data sensitive computations like medical imaging and financial options are protected against memory errors.
- True Cache Hierarchy: The GPU architectures developed before Fermi didn't contain an L1 cache, instead it contained a shared memory. Some users use algorithms that need true L1 cache.
- More Shared Memory: Some users required more shared memory than the 16KB per SM.

- Faster Context Switching: This allows for faster switches between GPU computing applications and normal graphics applications.
- Faster Atomic Operations: These atomic operations are similar to (read - modify - write).

General Overview of the Fermi Architecture

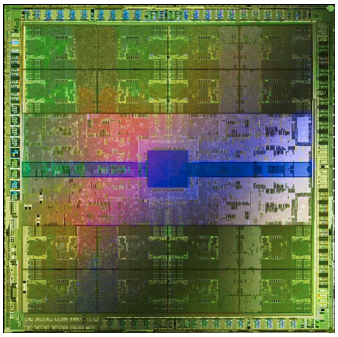

Figure 1: Nvidia Fermi Die

- 3 Billion transistors
- 40nm TSMC
- 384 bit memory interface (64 x 6)
- 512 Shader Cores (CUDA cores)
- 32 CUDA cores per shader cluster
- 16 Shader Clusters
- 1MB L1 Cache (64KB per shader cluster)
- 768KB Unified L2 Cache
- Up to 6GB GDDR5 Memory
- IEEE 754 - 2008 Double precision standard
- Six 64 bit memory controllers
- ECC Support
- 512 FMA in SP
- 256 FMA in DP

3 Billion transistors is a huge number, which when compared with its closest competitor which is just over 2 Billion transistors; shows how big the nvidia Fermi will be. To be able to put this huge number of transistors nvidia had to switch from the 45nm fabrication processes to the 40nm processes. This allowed nvidia to put this huge number of transistors on a dye without compromising with size and flexibility. But this also resulted in a very long delay to ship this chip. Due to relatively new fabrication processes and to the huge number of transistors on each chip, the yield of every wafer turned out to be very smaller, even smaller than expected. This hindered any hopes to mass produce the chip for large scale retail.

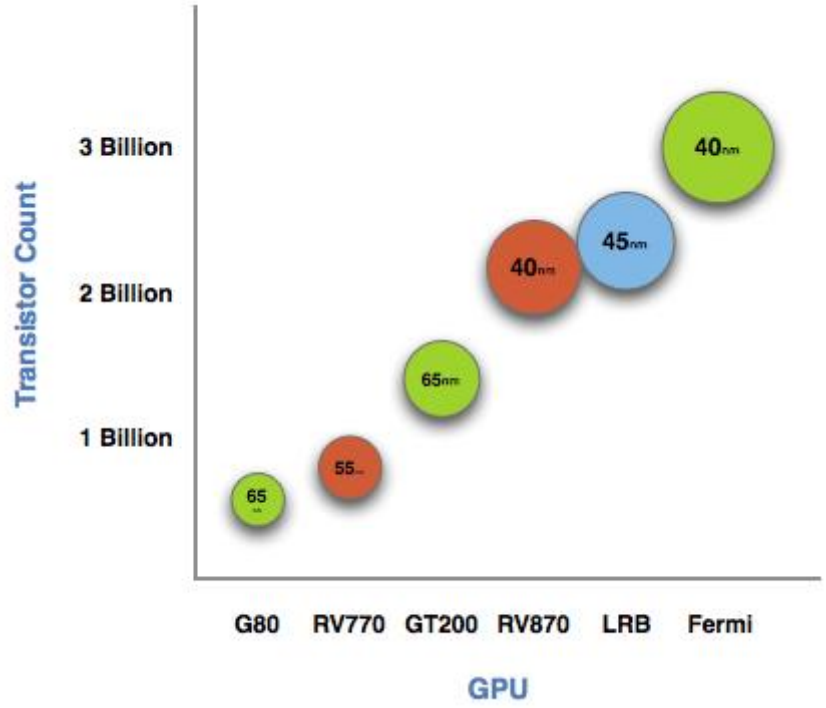

Figure 2: Transistor Count

Figure 3: Fermi Architecture

In Fermi nvidia aimed for a truly scalable architecture. Nvidia grouped every 4 SM (Stream Multiprocessor) into something called Graphics Processing Cluster or GPC. These GPC are in a sense a GPU on its own. Allowing nvidia to scale GPU cards up or down by increasing or decreasing the number of GPCs. Also scalability could be achieved by changing the number of SMs per GPC. Each GPC has its own rasterization engine which serves the 4 SMs that this GPC contains.

The SM Architecture

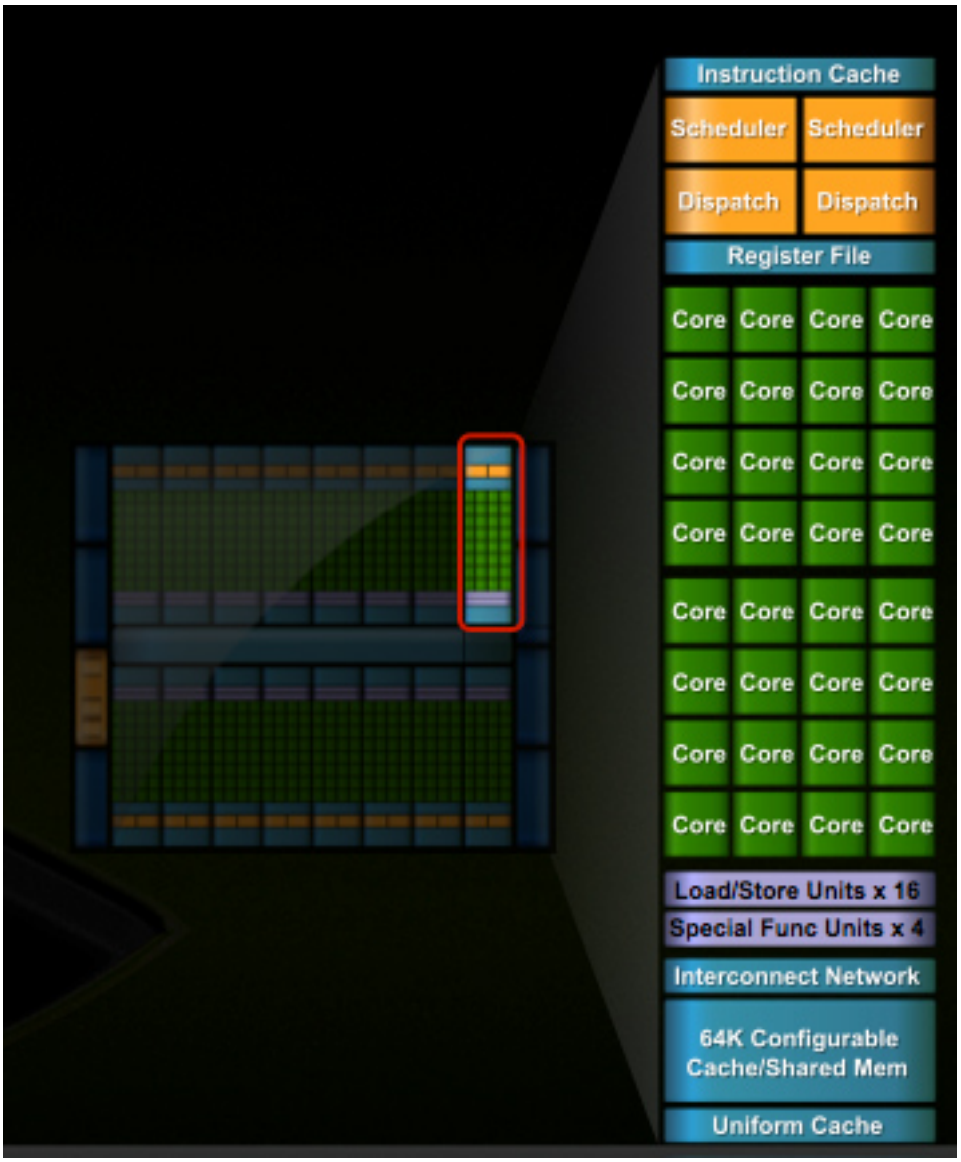

Figure 4: The SM

Each SM contains 32 stream processors or CUDA cores. This is 4x the amount of CUDA cores per SM compared to the previous GT200 architecture. These SM contain the following:

- 32 CUDA cores
- 4 SFU (Special Function Unit)
- 32K (32,768) FP32 registers per SM. Up from 16K in GT200
- 4 Texture Units
- A Polymorph (geometry) engine
- 64K L1 Shared Memory / L1 Cache
- 2 Warp Schedulers
- 2 Dispatch Units
- 16 load / store units

The SM 32 CUDA cores contain a fully pipelined ALU and FLU. These CUDA cores are able to perform 1 integer or floating point instruction per clock per thread in SP mode. There has been a huge improvement in the DP mode. DP instructions are now take only 2 times more than SP ones. This is a huge improvement when compared to 8 times the time in previous architectures. Also instructions can be mixed, for example FP + INT, FP + FP, SFU + FP and more. But if DP instructions are running then nothing else can run.

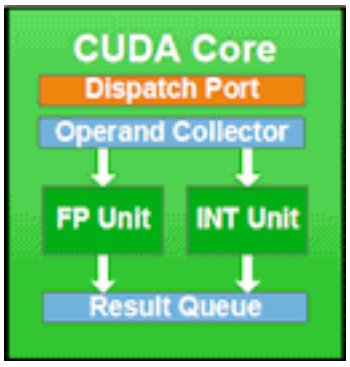

Figure 5: CUDA Core

The Fermi also uses the new IEEE 754 - 2008 Standard for Floating Point Arithmetic instead of the new obsolete IEEE 754 - 1984 one. In previous architectures nvidia used the IEEE 754 1984 standard. In this standard nvidia nvidia handled one of the frequently used sequence of operations which is to multiply two numbers and add the result to a third number with an special instruction called MAD. MAD stands for Multiply-Add instruction which allowed both operations to be performed in a single clock. The MAD instruction performs multiplication with truncation. This was followed by addition with rounding to the nearest even. While this was acceptable for graphics applications, it didn't meet the GPU Computing standards of needing a very accurate results. Therefore

with the adoption of the new standard nvidia introduced a new instruction which is called FMA or Fused Multiply Add which supports both 32 bit single precision and 64 bit double precision floating point numbers. The FMA improves upon MAD in retaining full precision without any truncations or rounding to the nearest even. This allows precise mathematical calculations to be run on the GPU.

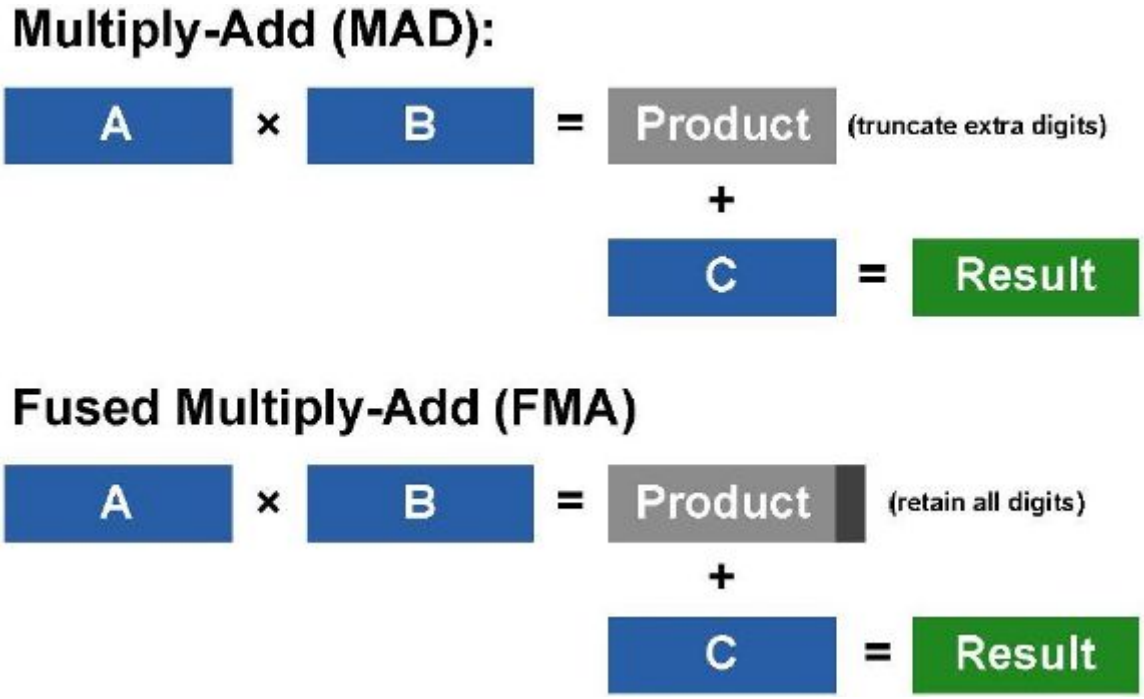

Figure 6: MAD vs FMA

CUDA is a hardware and software blend that allows nvidia GPUs to be programmed with a wide range of programming languages. A CUDA program calls parallel kernels. Each kernel can execute in parallel across a set of parallel threads. The GPU first of all instantiates a kernel to a grid of parallel thread blocks, where each thread within a thread block executes and instance of the kernel.

Thread blocks are groups of concurrently executing threads that can cooperate among themselves through shared memory. Each thread within a thread block has its own per-Thread private local memory. While the thread block has its per-block shared memory. This per-block shared memory helps in inter thread communication, data sharing and result sharing between the different threads inside the same thread block. Also on a grid level, there is a per-Application context global memory. A grid is an array of blocks that execute the same kernel.

This hierarchy allows the GPU to execute one or more kernel grids, a streaming multi-processor (SM) to execute one or more thread blocks and CUDA cores and other execution units in the SM to execute threads. Nvidia groups 32 threads in something called a warp.

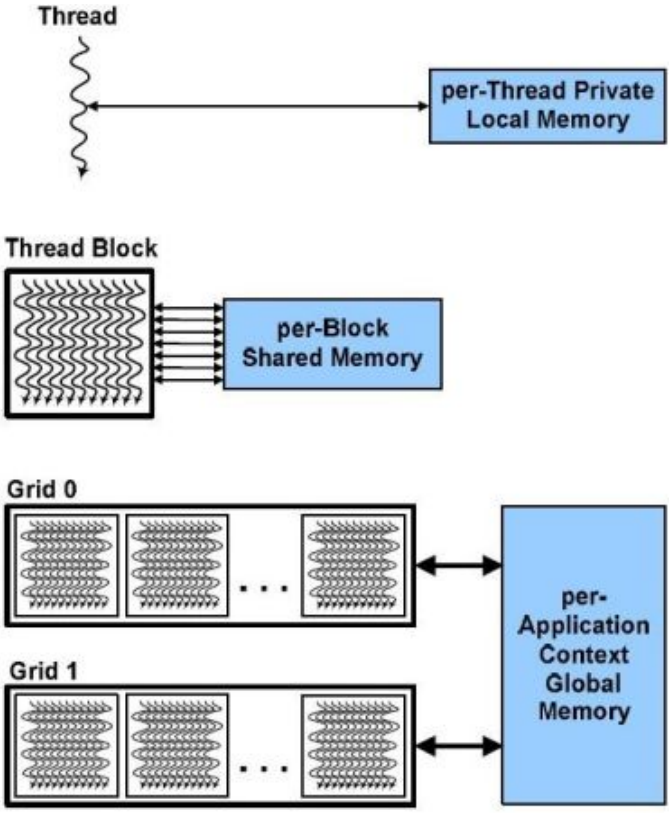

Figure 7: CUDA Hierarchy of threads

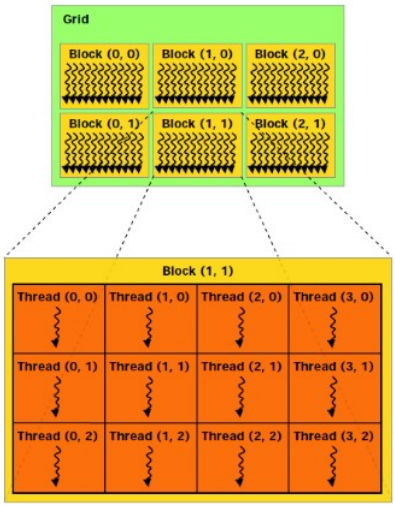

Figure 8: Thread Hierarchy

Each SM has two warp schedulers and two instruction dispatch units. This allows for two warps to be issued and executed in parallel on each SM.

The execution take place as follows. The dual warp schedulers inside each SM choose 2 warps for execution, one instruction from each warp is issued to be executed on a group of 16 cores, 16 load / store units or 4 SFU.

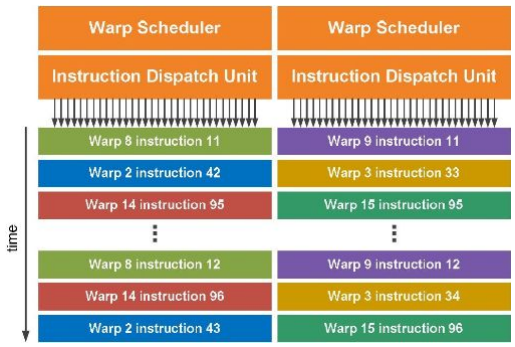

Figure 9: Warp Scheduling

The streaming multiprocessor's special function units (SFU) are used to execute special computations such as sine, cosine, reciprocal and square root. The SFU is decoupled from the dispatch unit. This decoupling means that the dispatch unit can still dispatch instructions for other execution units while the SFU is busy.

One of the biggest selling points of the new Fermi architecture is its implementation of a true cache. As stated earlier, earlier GPU architecture didn't have a true L1 cache. Instead these architecture contained something called "'Shared Memory"'. This was fine for graphics needs, but since nvidia is aiming to improve its GPU computing market share, it needed to implement a true L1 cache as it is often needed by some GPU computing applications. Nvidia included a 64KB configurable shared memory and L1 cache. To be able to handle both the graphics and GPU computing needs at the same time, this 64KB memory allows for the programmer to explicitly state the amount he needs to act as a shared memory and the amount to act as an L1 cache. Current options are for the programmer to have either 16KB L1 cache and 48KB shared memory or vice versa. This allowed the Fermi to keep the support for applications already written that made use of the shared memory while at the same time allowed new application to be written to make use of the new L1 cache.

For a long time there had been a huge gap between the geometry and shader performance. From the Geforce FX to the GT200, shader performance has increased with a factor of 150. But on the other hand the geometry performance only tripled. This was a huge problem that would bottleneck the GPU's performance. This happened due to the fact that the hardware part that handles a key part of the setup engine has not been parallelized. Nvidia's solution was to introduce something called a PolyMorph (geometry) Engine. The Engine facilitates a host of pre-rasterization stages, like vertex and

hull shaders, tessellation and domain and geometry shaders. Each SM contains its own dedicated polymorph engine which will allow to overcome any bottlenecks by parallelizing the different units inside the PolyMorph Engine.

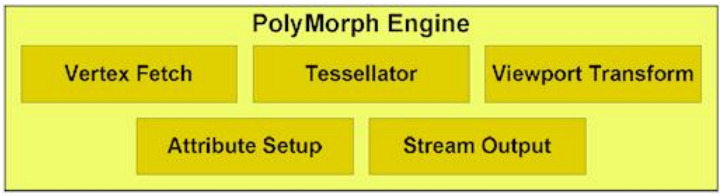

Figure 10: PolyMorph Engine

The SM also contains 4 separate texturing units. These units are responsible for rotate and resize a bitmap to be placed onto an arbitrary plane of a given 3D object as a texture.

Fermi Memory Hierarchy

In addition to the configurable 64 KB memory contained in each SM. The Fermi contains a unified L2 cache and DRAM. The size of the L2 cache is 768 KB.The 768KB unified L2 cache services all load, store and texture requests. The Fermi also contains 6 memory controllers. This large number of memory controllers allows the Fermi to support up to 6GB of GDDR5 memory. There can be several memory configurations supporting 1.5GB, 3GB or 6GB according to the field it will run in. It is important to mention that all types of memory from registers, to cache to DRAM memory are ECC protected.

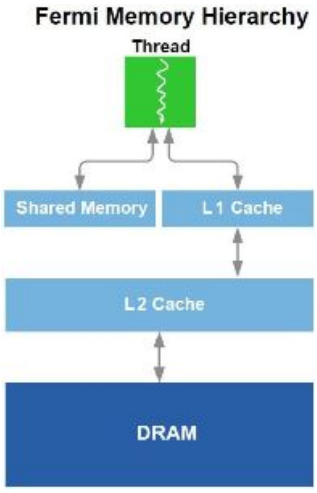

Figure 11: Fermi Memory Hierarchy

The Unified Address Space

Fermi unified the address space between the three different memory spaces (thread private local, block shared and global). In the previous architecture the load and store operations were specific for each type of memory space. This posed a problem for GPU

computing applications, as it made the task of the programmer more complex if not impossible to manage these different types of memory spaces, each with its own type of instruction. In the unified address space, Fermi puts all of the three different addresses into a single and continuous address space. Therefore Fermi unified the instruction to access all these types of memory spaces for a better experience. The unified address space uses a 40 bit addressing thus allowing for a Terabyte of memory to be addressed with the support of 64 bit addressing if needed.

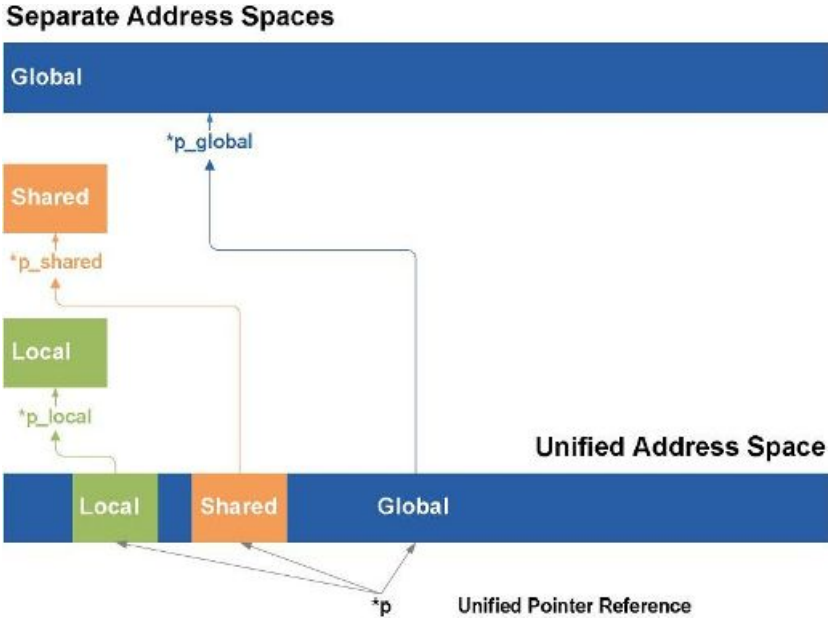

Figure 12: Unified Address Space

The GigaThread Scheduler

The nvidia Fermi architecture makes use of two thread schedulers. The scope of each scheduler differs from the other. At the chip level there is global scheduler which schedules thread blocks to various SMs. This global scheduler is called the GigaThread Thread Scheduler. At a lower level and inside an SM there are two warp schedulers which schedule individual threads inside the warp / thread block. The GigaThread Scheduler handles a huge number of threads in real time and also offers other improvements like faster context switching between GPU computing applications and graphics applications, concurrent kernel execution and improved thread block scheduling.

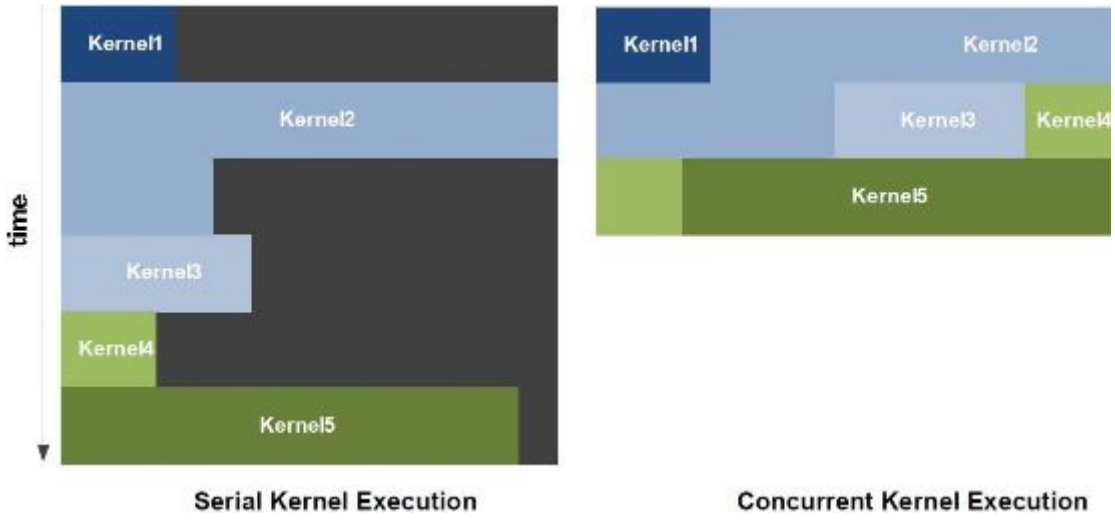

Figure 13: Concurrent Thread Execution

ROPs

ROP stands for raster operators. The raster operator is the last step of the graphics pipeline which writes the textured / shaded pixels to the frame buffer. ROP are supposed to handle several chores towards the end of the graphics pipeline. Chores like anti-aliasing, Z and colour compression and ofcourse the writing of the pixels to the output buffer. Fermi contains 48 ROPs which are placed in a circle surrounding the L2 cache.

Figure 14: ROPs

Nvidia Nexus

Nvidia Nexus is a development environment which was designed by nvidia to facilitate programming massively parallel CUDA C, OpenCL and DirectCompute applications for the nvidia Fermi cards. The environment is designed to be part of Microsoft Visual Studio IDE. Nexus allows for writing and debugging GPU source code in an easy way similar to the one used to develop normal CPU applications. It also allows to develop co-processing applications which make use of both the CPU and the GPU.

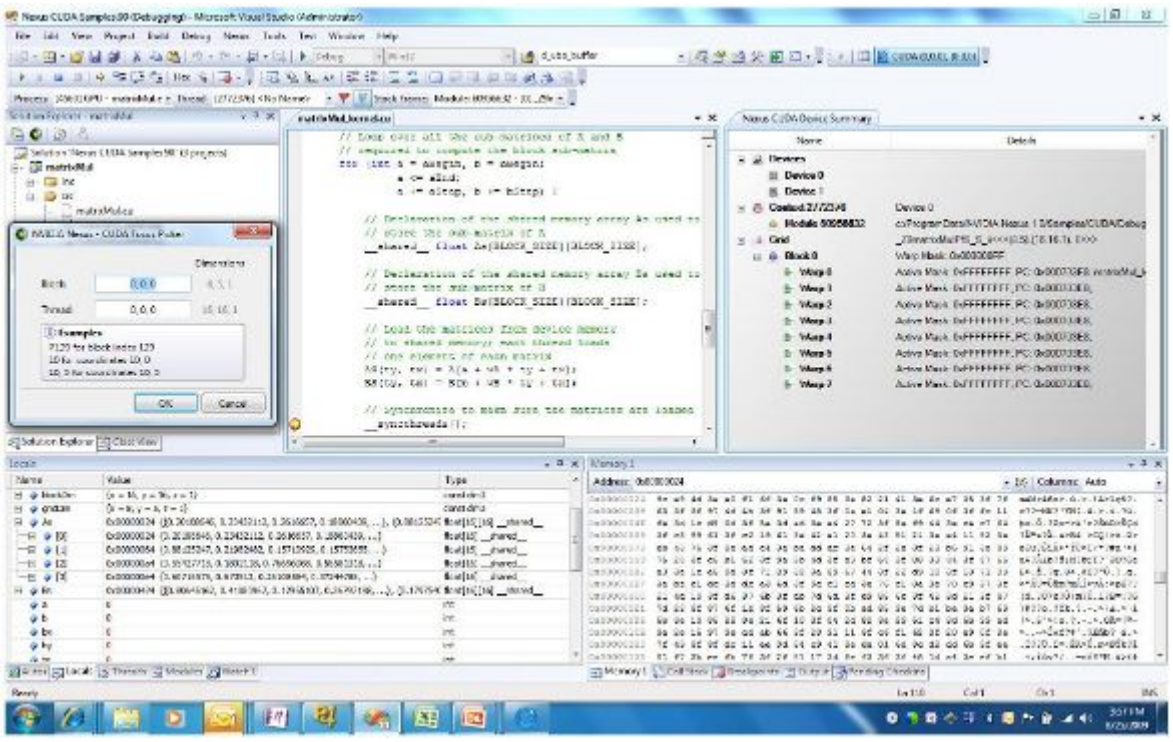

Figure 15: Nvidia Nexus inside Visual Studio

References

[1] www.brightsideofnews.com

[2] www.xbitlabs.com

[3] beyond3d.com

[4] www.techreport.com

[5] www.semiaccurate.com

[6] www.hardocp.com

[7] www.gpureview.com

[8] www.behardware.com

[9] www.nvidia.com