Towards the Specification of the GPU using Performance Parameters

Cristian Perez, Fabiana Piccoli

Universidad Nacional de San Luis,
Ejército de los Andes 950,
5700 - San Luis - Argentina
e-mail: {*cristian.perez@gridtics.frm.utn.edu.ar, mpiccoli@unsl.edu.ar*}

Abstract.
The characteristics of graphics processing units (GPUs), especially their parallel execution capabilities and fast memory access, render them attractive in many application areas. They promise more than an order of magnitude speedup over conventional processors for some non-graphics computations. The use of GPUs in general-purpose computing is becoming a very accepted alternative. In addition, the CUDA programming model gains acceptance. Each of these arguments make necessary count with tools that allow to evaluate GPUs.

The performance parameters allow to model an architecture to predict the execution time of any application with any parallelism level. Furthermore, they are a useful tool to compare different architectures and determine its advantages and troubles.

This work presents suitable benchmarks to evaluate different performance parameters of GPUs. The presented measurements focus on two issues of GPU performance: computing power and the global memory bandwidth. Their estimation will allow us determine technical characteristics of GPU and, in consequence, the analysis and optimization of the performance of applications that could run on actual or future GPUs.

1 Introdution

The GPU (Graphic Processing Unit) was adopted as a new hardware platform to resolve general-purpose applications. This opens a vast universe of possibilities, especially for the resolution of massively parallel applications.

For many years, the GPU was used only to accelerate the graphics applications, such as video games and interactive 3D. Its good performance in this area joined to its constant and quickly evolution (compared with the general-purpose microprocessors) and a smaller number of instructions [1, 2], has allowed the development of domestic supercomputer model cheaper than a PC. This model can solve certain kind of problems, all of them can be solved applying a massively parallel computing model with the following characteristics: architecture multicore, shared memory and multithreaded support [3].

There are many alternatives of processing on GPU, the most widely used is the NVIDIA card [4]. For its programming was development CUDA (Compute Unified Device Architecture), a C programming kit that includes a data communication model and a threads control model. CUDA provides a GPU-CPU interface [5]. It pretends simplify the programming work through a parallel programming model [6] and facilitate the threads synchronization and the communication between CPU and GPU [7, 8].

Taking advantages of computational power, low cost, continuous development, high memory bandwidth, flexibility and programmability of the GPU; and despite its limitations and difficulties in resolving any type of application, it is feasible to use the GPU computation power to solve non-graphics application. This computing kind is called GPGPU: General-Purpose computation on Graphics Processing Units.

This paper is organized as follows, the next section explains the features of the GPU: architecture and CUDA programming model. Section 3 establishes the importance of performance metrics and details which parameters we considered in this work and how we measure them. Finally we show experimental results, conclusions and future work.

2 Characteristics of GPU

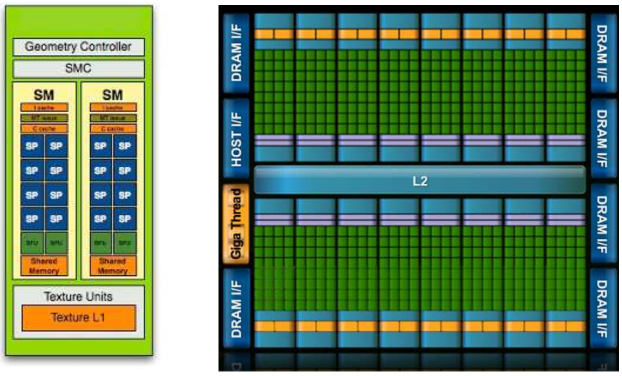
A GPU computing system consists of two basic components, the traditional CPU and one or more GPUs (Streaming Processor Array). The connection between CPU and GPU is by mean of PCI Express bus.

The GPU can be considered as a multicores coprocessor ables to support fine grain parallelism (a lot of threads run in parallel, all of them collaborate in the solution of the same problem) [4, 9, 10]. GPU is different than other parallel architectures because it shows flexibility in the local resources allocation (memory or register) to the threads. In general, a GPU multiprocessor consists of several streams, each of them has multiple processing units, records and on-chip memory. Each stream multiprocessor can run a variable number of threads. The programmer decides how many threads and how they will work. These can be adjusted to achieve improvements in the system performance.

In this section we detail the GPU features, its hardware and the proposed programming model. Although most of the GPU on the market have similar characteristics, in this paper we focus on the architecture and programming model proposed by NVIDIA [2].

2.1 Architecture

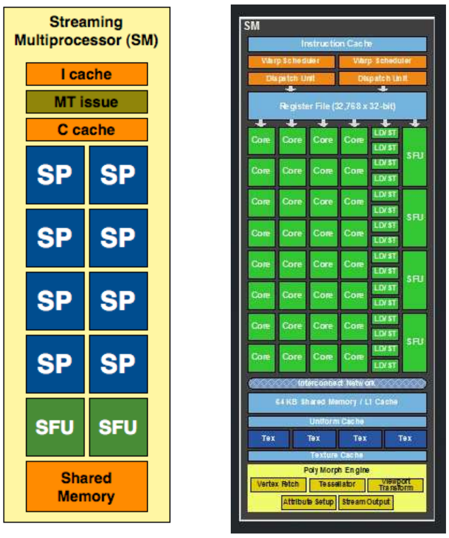
The GPU architecture differs significantly from CPUs in many aspects. The figure 1 shows how architectures of GPU, typical (figure 1(a)) and modern (figure 1(b)) that are formed by a set of units called Texture/Processors Cluster (TPC), each of them is made up of several multiprocessors, one texture unit and some logic control.



(a) Typical (b) Modern

Fig. 1. GPU Architecture

A multiprocessor or SM (Streaming Multiprocessor) consists of eight (thirty two in modern GPU) Scalar Processor (SP), sometimes called *core*, two special function units for transcendental functions such as sine or cosine, a multithreaded instruction unit, and on-chip shared and cache memory. The Figure 2 depicts the two SM architectures.



(a) Typical GPU (b) Modern GPU

Fig. 2. SM Architecture

Each GPU has own RAM memory, global memory. The connection bus between GPU and global memory is usually larger than the corresponding at CPU, it can vary from 64 bits until 768 bits. This property allows to transfer a large amount of information per unit time between both.

2.2 CUDA Programming Model

The GPU computing system consists of a CPU (host) and one or more GPU (massively parallel co-processors). Each co-processor applies the Simple Process-Multiple Data (SPMD) model, all units of computation (thread) running the same code, not necessarily synchronously, over different data. Every thread shares the global memory space [6, 7, 11, 12, 5, 8, 10].

The CUDA programming model proposes a model for GPU programming. It has two main characteristics: the parallel work through concurrent threads and the memory hierarchy. A CUDA program is written in standard C/C++ extended by several keywords and constructs. The user supplies a single source program encompassing both *host* (CPU) and *kernel* (GPU) code. Each CUDA program consists of multiple phases that are executed on either the CPU or the GPU. All phases that exhibit little or no data parallelism are implemented in the CPU. In opposition, if the phases present much data parallelism, they are implemented as kernel functions in the GPU. A kernel function defines the code to be executed by each threads launched in a parallel phase.

There are several restrictions on kernel functions, they cannot: be recursive neither have static variables declarations or a variable number of arguments. The communication between CPU and GPU is through API calls. Kernel code is initiated performing a function call.

Threads in the CUDA model are grouped into thread *blocks*. All threads in a block execute on one SM and communicate among them through the shared memory. Threads in different blocks can communicate through global memory. Besides shared and global memory, the threads have their local variables. Thread blocks form a *grid*. The number of grids, blocks per grid and threads per block are parameters fixed by the programmer. As they can affect the performance of the application, can be adjusted.

3 Performance Parameters

If you want to compare two or more computer systems, you need to compare them through their technical characteristics that can be measured by one or more parameters. Among the most common parameters are: Instructions per second (FLOPS or MIPS), Performance per watt and Memory Bandwidth.

There are many technical characteristics or theoretical parameters of GPU performance completely dependent of the hardware, some of them are:

- Computing power: It is related with the power of processor or set of processors (ALUs) to make logical or arithmetic operations. It is independent of its

subsequent ability to write the results in memory. If this parameter measures integers arithmetic operation is called MIPS, but if it refers to floating-point arithmetic operation is called FLOPS. In certain cases, this ability to perform arithmetic depends on the data type but in the GPU, the processing power of integer (MIPS) and floating point (FLOPS) are the same [11].
- Calculation Delay: This parameter determines the instructions delay. It calculates since the begin of execution instruction until the end of its processing through the architecture pipeline.
- Memory Bandwidth: It sets the amount of writes, reads or read-write operations over a memory type in a certain time. The bandwidth refers to the maximum capacity of the channel used to transfer data between the processor and memory. According to the nature of memory, the number of reads and writes per unit of time may differ substantially.
- Time or Latency of Memory Access: It measures the time between the write or read request and its carrying out.

Most of the time, their estimations are not easily accomplished by software tools (They can be used to establish the potential of some resource: computer, network, program or device). The tests can be quantitative: response time, MIPS, etc. [13], or qualitative: reliability, scalability, in-interoperability, etc. Many times, the performance evaluation is made by stress testing. The next section details the performance indicators considered in this work.

3.1 Estimation of Performance Parameters

Through the performance parameters, we can characterize the architecture and, consequently, predict the performance of applications in different situations and/or programming modes. In the practice, the most of obtained values are lower than those established by the physical environment. This is due to the limitations and delays imposed by the used hardware technology and the applied programming model. Therefore, they depend on how they use the GPU resources, if they employ:

- All the SM and every scalar processor of them (it is called $All - All$).
- All scalar processors of one only SM ($One - All$).
- All SM and one only scalar processor for each of them ($All - One$).
- A single scalar processor of one only SM ($One - One$).

To measure some of the proposed parameters and establish the technical characteristics of GPU, we developed BEN_MyCP 2.0, a benchmark with several applications. In [14] we established the theoretical parameters and in this work, we calculate the practical or real parameters of the GPU.

Considering two technical parameters: the computing power and global memory bandwidth, and each of above four situations, we can determine eight parameters, they are:

- Computing power of $All - All$.

- Computing power of $One - All$.
- Computing power of $All - One$.
- Computing power of $One - One$.
- Global Memory bandwidth of $All - All$.
- Global Memory bandwidth of $One - All$.
- Global Memory bandwidth of $All - One$.
- Global Memory bandwidth of $One - One$.

In this work, we express the first four parameters in MIPS or FLOPS and the last four in GBytes by second. In the next sections we explain the features of applications included in BEN_ MyCP 2.0 and how the parameters estimation is made.

3.2 Computing power

This parameter determines the inherent ability of a processor or processors set to make logical and/or arithmetic operations, regardless to the write of results in a memory of the hierarchy. We can use it to compare with other architectures.

To measure the computing power of GPU, our work is focused particularly on the ability to solve floating-point arithmetic operations: FLOPS (It can be generalized to integer operations, MIPS).

The parallel algorithm must generate a high throughput without the need to access GPU memory using only records. The developed application performs intensive mathematical operations of multiplication and addition on a SP register. To achieve reliable results, we avoid the memory access (any type of memory) until the end when we obtain the final results. The intermediate results are stored temporarily in the internal registers of each SP. A thread is responsible to obtain the result of one operation.

Below we analyze the four parameter related with Calculation Capacity.

Computing power of $All - All$ ($Cp_{All-All}$)

To obtain $Cp_{All-All}$, we must take account several considerations to avoid:

1. Delays related to the administration and allocation of threads to the SP of SM (warps). We must ensure the filling of the pipeline of all the SMs.
2. The underutilization of the SMs. Blocks must have at least 32 threads, quantity of threads managed per warp.

In consequence, we launch many blocks and many threads per block to ensure the maximum parallelism degree and, hence, the use of every SM and all of its SP. In other words, the application uses the maximum available resources of GPU.

By all this characteristics, we estimate the computing power of GPU P, ie

$$Cp_{All-All} = P.$$

Calculation Capacity of One $-$ *All* $(Cp_{One-All})$

In this case, the number of blocks is 1. In consequence, as we must avoid the influence of warps manager, the application has to have the enough parallelism to eliminate this effect. We achieve it by generating a block whose threads number is sufficient larger how to fill the pipeline of SM.

This parameter represents the computing power of a SM. With it and above parameter, it is possible to obtain the SM number of a GPU, ie

$$\|SM\| = \frac{Cp_{All-All}}{Cp_{One-All}}$$

Calculation Capacity of All $-$ *One* $(Cp_{All-One})$

To find this parameter, we have to have many blocks with only one thread. Considering the restriction 1, it is necessary to generate a lot of blocks to eliminate the influence of warps manager.

Through this parameter, we obtain the wastage factor of SM by the novectorial use of hardware. This non-dimensional parameter indicates the number of parallel pipeline. It can be expressed as

$$N_{PPipe} = \frac{Cp_{All-All}}{Cp_{All-One}}$$

In other words, N_{PPipe} can be viewed as missed computing power owing to the using of all SM as scalar processors.

Calculation Capacity One $-$ *One* $(Cp_{One-One})$

For this case, the number of blocks and threads per blocks are equal to 1. From it and the above parameters, we can express the pipeline depth of architecture that is defined by the next expression:

$$L_{Pipe} = \frac{Cp_{All-One} \times Cp_{One-All}}{Cp_{One-One} \times Cp_{All-All}}$$

With these four parameters, we obtain the intrinsic characteristics of GPU hardware such as: its Calculus power(P), the SM number, the number of parallel pipeline (N_{PPipe}), and the pipeline depth (L_{Pipe}). The last three describe the organizational structure of the hardware.

3.3 Memory Transfer Rate

This parameter determines the amount of write, read or read-write that can be made at any position of a memory type in a time instant. Also, the bandwidth refers to the maximum capacity of the channel used for data transfer of bus that connects the processor with the memory. Generally, the values obtained in the practice are less than the established by the physical environment. The most of the time, this is due to the limitations and delays imposed by the hardware technology used.

In CUDA model, as mentioned above, the threads can access different types of memory during execution of a *kernel*. The memory access can be to: private local memory, shared memory or/and the global memory. In this work, we measure the global memory rate of the read-write operations.

Transfer Rate of Read-Write Operation on Global Memory

An operation of read/write involves a global memory access by each of these operations. To measure the speed of read/write operations on global memory of GPU, we implement an algorithm that reads a particular memory location, performs a simple arithmetic operation and, finally, writes the result in the same memory position. These three operations are made in order by a thread on a memory location.

Such as the CUDA/C compiler optimizes the code we have to avoid it. For this particular case and in presence of *read-arithmetic-write* operations, the compiler avoids the first (read) and works directly with the processor registers displaying an unreal speed. To resolve it, we employ large memory blocks as to avoid the use of the registers of the GPU and to force it to read and the write on the global memory.

Below we analyze the four parameter related with Global Memory Bandwidth.

Global Memory Bandwidth of $All - All$ *(*$GMb_{All-All}$*)*

When many blocks and threads per block are launched, we can ensure a continuous data transfer to memory and reduce the high memory latency. Also, if the blocks are enough large (there are many threads by each), each thread makes individual transfers. The set of threads transfers made in parallel would use the maximum bandwidth of the memory channel.

As the continuous transfer to memory, many blocks and many threads per block are ensure, we estimate the max memory bandwidth (B) of GPU, i.e.:

$$GMb_{All-All} = B.$$

Global Memory Bandwidth of $One - All$ $(GMb_{One-All})$

This parameter is obtained when the number of blocks is equal to 1. To guarantee that each thread make transfers individually, it is necessary ensure a block size sufficiently large (many threads). It is the same case that the above parameter.

It indicates the maximum bandwidth of memory that can reach a SM by read-write operations. From this parameter and the above is possible to estimate the necessary number of simultaneous full blocks (max number of threads per block) that occupy the whole bandwidth of memory ($N_{MIN}Bl$), it is

$$N_{MIN}Bl = \frac{GMb_{All-All}}{GMb_{One-All}}$$

Global Memory bandwidth of $All - One$ $(GMb_{All-One})$

To obtain $GMb_{All-One}$, we execute the algorithm on many blocks of a single thread each. Although there are many simultaneous blocks, there is only one thread per block; in consequence it is impossible to exploit the max memory bandwidth.

The max benefit of memory bandwidth of GPU architectures is achieved when the read or/and write operations of parallel threads are made in structured form: with only one memory access we can read or/and write simultaneously every processed values by the large set of threads, it is named memory alignment or memory coalescing. The alignment constraint varies according to the hardware architecture, for older architectures are stricter [15]. Therefore, it is possible to determine the waste factor of the memory bus (W) when read-write no coalescing are done. It is achieved by

$$W_{Memory\ Bus} = \frac{GMb_{All-All}}{GMb_{All-One}}$$

Global Memory bandwidth of $One - One$ $(GMb_{One-One})$

In this case, the number of blocks and number of threads per blocks are equal 1. With it and $GMb_{One-All}$, we can calculate how long the thread is delayed during its processing on GPU, i.e.

$$T_{Thread\ Delay} = \frac{GMb_{All-One}}{GMb_{One-One}}$$

With these four parameters, we can obtain the intrinsic characteristics of GPU global memory such as: its Bandwidth (B), the min number of necessary and simultaneous blocks to occupy the whole bandwidth of memory ($N_{MIN}Bl$), the waste factor of the memory bus ($W_{Memory\ Bus}$), and time that a thread is delayed during its processing on GPU ($T_{Thread\ Delay}$).

4 Experimentals Results

In this section we show and analyze the results for every parameters proposed. The analysis was made for three generations of GeForce GPU: GTS 250, GTX 260 and the GTX 470 whose characteristics are:

	GTS 250	GTX 260	GTX 470
Global Memory	1073020928 bytes	938803200 bytes	1341325312 bytes
SM	16	27	14
SP	128	216	448
Clock rate shader	1836 MHz	1400 MHz	1215 MHz
Clock rate gpu	740 MHz	666 MHz	607 MHz
Clock rate memory	1100 MHz	1150 MHz	1674 MHz
Type of Memory Bus	256 bit DDR3	448 bit DDR3	320 bit GDDR5

Each reported value is the averages of many executions of corresponding algorithm that detailed above.

The next table shows the average values obtained of four parameter of computing power.

	GTS 250	GTX 260	GTX 470
$Cp_{All-All}$	436.344289 GFLOPS	562.329588 GFLOPS	503.849162 GFLOPS
$Cp_{One-All}$	27.308304 GFLOPS	20.882264 GFLOPS	36.189371 GFLOPS
$Cp_{All-One}$	13.608844 GFLOPS	17.550189 GFLOPS	7.032963 GFLOPS
$Cp_{One-One}$	0.170045 GFLOPS	0.108546 GFLOPS	0.062842 GFLOPS

The $Cp_{All-All}$ is closer to the theorical computing power (P), which can be get for GTS250 and GTX260 by

$$P = 2 \times shader\_frecuency \times \|SP\|$$

For GTX 470, we used operations 24-bit MAD without optimization for modern hardware [11], hence the theorical computing power can express as

$$P = shader\_frecuency \times \|SP\|$$

The max computing power ($Cp_{All-All} = P$) for GTS 250, GTX 260 and GTX 470 architectures are achieved when there are at least 64, 128 and 1024 threads per block respectively.

In the case of second parameter, $Cp_{One-All}$, we determined the block dimension minimum to keep busy the pipe. This has to be at least of 160, 224 and 544 for GTS 250, GTX 260 and GTX 470 architectures respectively.

From these four parameters, we can obtain different technical characteristics of GPU. The next table displays three technical features: P, SM number, N_{PPipe} and L_{Pipe}.

	GTS 250	GTX 260	GTX 470
P	436.344289 GFLOPS	562.329588 GFLOPS	503.849162 GFLOPS
$\|SM\|$	16	27	14
N_{PPipe}	32	32	32× 2.25
L_{Pipe}	5	6	8

Regarding of this four GPU features, P is the main performance Indicator of architecture. Instead the others three will allow us to establish which architecture is the best suited to solve a particular application. For example, if the application has a low degree of parallelism, a GPU with fewer resources will be more appropriate. Besides, as in the new architectures not only P increases but N_{PPipe} and L_{Pipe} also, these become inadequate when the applications are not highly parallel.

Below we present the results of parameters related to memory bandwidth. The next table shows the average values obtained of four parameters analyzed for transfer rate of read-write operation on global memory (GMb_{X-X}).

	GTS 250	GTX 260	GTX 470
$GMb_{All-All}$	57.312697 GB/s	104.866219 GB/s	108.394003 GB/s
$GMb_{One-All}$	10.443662 GB/s	8.520936 GB/s	29.362377 GB/s
$GMb_{All-One}$	2.719361 GB/s	3.878613 GB/s	9.640949 GB/s
$GMb_{One-One}$	0.023726 GB/s	0.020135 GB/s	0.098046 GB/s

The first parameter is one magnitude lower than theoretical value expected (B) that could be calculated by the following expression:

$$B = \frac{clock\_frecuency \times Width\_of\_memory\_bus \times 2}{8}$$

where the factor 2 obeys to that the transfer is ddr (Double Data Rate) [16], and 8 is to express the result in bytes.

The max memory bandwidth for read-write operations is achieved when the number of threads per block is greater or equal than 64, 128 and 192 for the GTS 250, GTX 260 and GTX 470 architectures respectively. In the case of second parameter, the max value is obtained if the threads number of block is greater or equal than 512, 512, 640 for of three architectures considered in this work respectively.

Through these four parameters related to memory bandwith of read-write operations, we can obtain different technical characteristics of GPU. The next table displays four technical features: B,$N_{MIN}Bl$, $W_{Memory\ Bus}$, $T_{Thread\ Delay}$.

	250 GTX	GTX 260	GTX 470
B	57.312697 GB/s	104.866219 GB/s	108.394003 GB/s
$N_{MIN}Bl$	5.49	12.3	3.69
$W_{Memory\ Bus}$	21	27	11.24
$T_{Thread\ Delay}$	114.6	192.5	98.37

B, $W_{Memory\ Bus}$ and $T_{Thread\ Delay}$ depend exclusively on of the memory architecture. $N_{MIN}Bl$ also is influenced by the GPU architecture and technology of global memory. If B is bigger then the memory performance is better, the same is valid when $N_{MIN}Bl$, $W_{Memory\ Bus}$ and $T_{Thread\ Delay}$ are lower. In the case of GTX 470 has better performance in any type of applications because it uses global memory kind faster and with lower latencies.

5 Conclusions, Remarks and Future Work

The GPU is a massively parallel architecture, it has a high throughput because its capacity of parallel processing for thousands of threads. With each new generation of GPU, new parallel processing capabilities are incorporated.

In this work we showed how, through a simple methodology, we can determine eight performance parameters of GPU related to computing power and global memory transfer rate. We presented the results of each of them on different generations of GPUs. The first four parameters characterize the architecture of the GPU and the last four correspond to the GPU memory architecture.

Such as the eight parameters offer enough information, we can determine the technical features of the architectures discussed and know their advantages and troubles. All of them are very important and necessary when we want to estimate the performance of any application on GPU.

In the future, we pretend to extend the set of performance parameters and apply the proposed analysis to others GPU manufacturers such as AMD-ATI. Further, it is important verify if presented method can be an universal method to determine features of massively parallel architectures.

6 Acknowledgements

We wish to thank to UNSL and UTN-Mendoza for allowing us the access to their computational resources. This research has been partially supported by UNSL under project PROICO-30310.

References

1. M. Lieberman, J. Sankaranarayanan, and H. Samet. A fast similarity join algorithm using graphics processing units. In *ICDE 2008. IEEE 24th International Conference on Data Engineering 2008*, April 2008.
2. Humphreys G. Luebke, D. How gpus work computer. *EEE Computer*, 40(2), Feb 2007.
3. D. Lloyd, C. Boyd, and N. Govindaraju. Fast computation of general fourier transforms on gpus. In *IEEE International Conference on Multimedia and Expo*, April 2008.
4. NVIDIA. Nvidia geforce 8800 gpu architecture overview. In *NVIDIA*, 2006.
5. M. Joselli, M. Zamith, E. Clua, A. Montenegro, A. Conci, R. Leal-Toledo, L. Valente, B. Feijo, M. Dórnellas, and C. Pozzer. Automatic dynamic task distribution between cpu and gpu for real-time systems. In *11th IEEE International Conference on Computational Science and Engineering*, July 2008.
6. I. Buck. Gpu computing with nvidia cuda. *ACM SIGGRAPH 2007 courses ACM*, 2007. New York, NY, USA.
7. W. Chen and H. Hang. H.264/avc motion estimation implementation on compute unified device architecture (cuda). In IEEE, editor, *IEEE International Conference on Multimedia*, April 2008.
8. D. Luebke. Cuda: Scalable parallel programming for high-performance scientific computing. In *5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro, ISBI 2008*, May 2008.
9. NVIDIA. Nvidia geforce gtx 200 gpu architectural overview. In *NVIDIA*, 2008.
10. S. Ryoo, C. Rodrigues, S. Baghsorkhi, S. Stone, D. Kirk, and W. Hwu. Optimization principles and application performance evaluation of a multithreaded gpu using cuda. In *ACM*. ACM, 2008.
11. NVIDIA. Nvidia cuda compute unified device architecture, programming guide version 2.0. In *NVIDIA*, 2008.
12. Felipe A. Cruz. Introduction to gpu computing. In *PASI- Scientific Computing the Americas*.
13. Demmel J. W. Volkov, V. Benchmarking gpus to tune dense linear algebra. In *ACM/IEEE conference on Supercomputing*, Piscataway, NJ, USA, 2008. IEEE Press.
14. Cristian Perez and Fabiana Piccoli. Estimación de los parámetros de rendimiento de una gpu. In MECOM 2010 CILAMCE, editor, *Mecánica Computacional-High Performance Computing in Computational Mechanics*, volume XXIX, pages 3155–3167, 2010.
15. Wen-mei W. Hwu David B. Kirk. Programming massively parallel processors: A hands-on approach, 2010.
16. JEDEC-Global Standars for the Microelectronics Industry. *DOUBLE DATA RATE (DDR) SDRAM STANDARD (JESD79F)*. JEDEC- Global Standars for the Microelectronics Industry,, http://www.jedec.org/standards-documents/docs/jesd-79f, Febrary 2008.