Bulletin of the *Transilvania* University of Braşov
Series I: Engineering Sciences • Vol. 4 (53) No. 2 - 2011

COMPARISON OF SINGLE AND DOUBLE FLOATING POINT PRECISION PERFORMANCE FOR TESLA ARCHITECTURE GPUs

L.M. ITU[1] C. SUCIU[1,2]
F. MOLDOVEANU[1] A. POSTELNICU[3]

Abstract: *The paper compares the single and double floating point precision performance of NVIDIA Tesla GPUs. Double precision is crucial for the accuracy of some applications containing scientific computation. Three representative applications have been chosen in order to compare the performances: matrix multiplication, incompressible Navier-Stokes equations and the steady state heat conduction problem. The expected values of performance decrease lie between two for bandwidth limited applications and eight for compute limited applications. Based on the conducted experiments, the type of each application has been identified and the decreases in performance have confirmed the expected theoretical values.*

Key words: *GPU, floating point precision, Tesla architecture, speed-up.*

1. Introduction

Graphics Processing Unit (GPU) based implementations have introduced an alternative to CPU based solutions. GPUs were initially used only as graphical accelerators in image processing applications. A GPU is a many-core processor, which, given the need of the graphical applications, is designed in order to execute a large number of floating point operations in parallel on hundreds of cores [11]. The transition from graphics applications to general purpose applications has been made possible by the introduction of CUDA (Compute Unified Device Architecture) [3].

When a GPU is programmed through CUDA, it is viewed as a compute device, which is able to run thousands of threads in parallel by launching a kernel (a function, written in C language, which is executed by the threads on the GPU) [13]. The latest GPUs contain several streaming multi-processors, each of them containing eight cores.

Currently all applications which use a GPU in order to accelerate the execution also use the CPU in order to perform auxiliary tasks (like initializations or post-processing) and also to launch the kernels [8]. Until now the GPU is not able to run as a stand-alone device, it needs to be

[1] Dept. of Automatics, *Transilvania* University of Braşov.
[2] Corporate Technology, PSE Siemens Romania.
[3] Dept. of Thermodynamics and Fluid Mechanics, *Transilvania* University of Braşov.

launched by a host thread which also manages the data located in the global memory (it copies initial values to the device and copies the results back at the end of the execution).

The first architecture to support CUDA programming was the G80 architecture. One of the major drawbacks of it was the lack of double precision floating point support. This aspect was then improved in June 2008, when NVIDIA introduced a major revision to the G80 GPUs: the GT200 architecture almost doubled the number of cores (240 instead of 128), memory access coalescing require-ments were alienated and maybe most importantly, double precision floating point support was also added in order to satisfy the need of scientific high-performance computing applications.

The goal of this article is to compare the single and double precision floating point performance of the GT200 Tesla architecture. In various applications containing scientific computation, double precision floating point math is essential in order to obtain accurate results. To perform the comparison, three applications have been chosen: a dense matrix-matrix multiplication, the numerical solution of the incompressible Navier-Stokes equations on a MAC grid and a steady state heat conduction problem. The first one is a common operation in linear algebra and the other two applications are widely spread in the field of computational fluid dynamics. Section two provides a short overview of the CUDA architecture. Section three briefly describes the three applications for which the tests have been conducted. Section four presents the results of the research activity and finally some conclusions will be drawn in chapter five.

2. The CUDA Architecture

CUDA refers to both the hardware and software architecture, which enables NVIDIA GPUs to execute programs written with C, C++, Fortran, OpenCL, DirectCompute, and even other languages. A CUDA program uses the GPU by calling parallel kernels [10]. A kernel then launches thousands of threads into execution (Figure 1). These threads are organized at three levels. First there is the grid of thread blocks, which is a two dimensional arrangement of the blocks. Then every block consists of up to 512 threads which are organized in three dimensions. Finally every group of 32 threads of the same thread block will form a warp. This architecture is called SIMT (Single Instruction Multiple Thread), which enables different threads to execute different instructions at the same time, without loss of performance for different warps (ideally the threads of the same warp should not diverge, otherwise the different execution branches will be serialized).

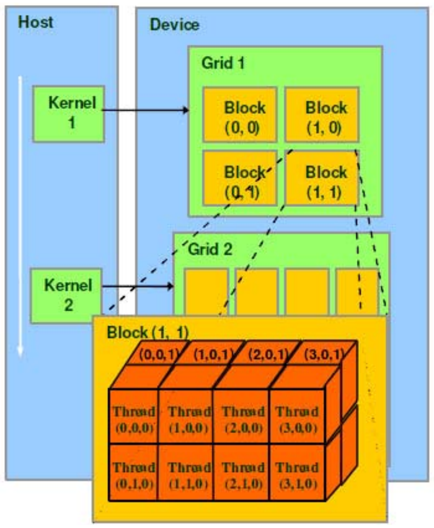

Fig. 1. *CUDA Thread organiyation*

Every thread has access to built-in variables which allow it to identify the specific input data, which has to be manipulated and also where the result should be stored. Further the threads of a thread block can cooperate, by using the shared memory or by waiting for each other through barrier synchronization. Also, each thread has access to a per-thread private memory, which is used for intermediate results, usually called registers (Figure 2 displays the various available memory types). The CPU (the host) has access to the global memory, the constant and the texture memory of the GPU (the device). The three-leveled hierarchy of threads maps straight forward to the hierarchy of processors of the GPU. These are usually called streaming multiprocessors and contain 8 cores. Each CUDA core has both an integer arithmetic logic unit (ALU) and a floating point unit (FPU). Also every streaming multi-processor contains an FPU for double precision. Hence the double precision performance of a CUDA program, when compared to its single precision version, should be up to eight times slower.

In order to obtain optimum performance, there are several guidelines which have to be followed. Some of the most important ones are: minimize transfers of data between CPU and GPU, ensure coalesced (sequential and aligned) access to the global memory, use shared memory to avoid redundant access to global memory, avoid different execution paths for the threads of a warp etc. [9].

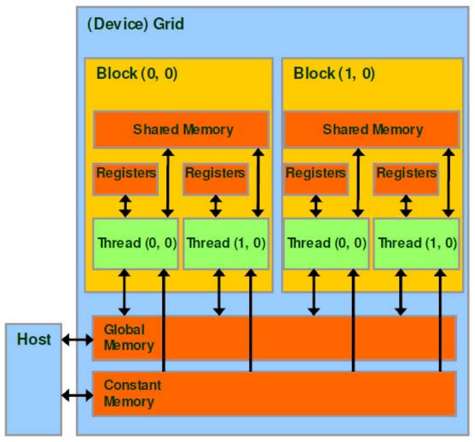

Fig. 2. *CUDA Memories*

3. Test Applications

3.1. Matrix-Matrix Multiplication

Since matrix-matrix multiplication is a common operation, only the idea behind the GPU kernel performing the operation will be discussed (Figure 3). The input matrices as well as the result matrix are divided into blocks (not necessarily square blocks; a very interesting pattern is indicated in [5]). A block of threads computes a block of elements in the results matrix (every thread will usually compute several elements) by calculating the dot product of the corresponding rows and columns of the matrices A and B respectively. In order to limit the wasted global memory bandwidth, data is brought into shared memory in a common effort from the threads of the block. Because the shared memory is a limited resource, data is read in tiles from the global memory and the dot-product is computed in several steps.

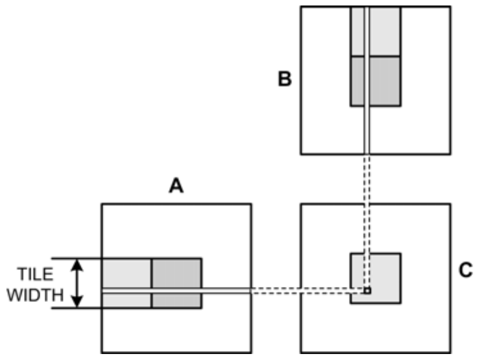

Fig. 3. *Matrix Multiplication with Shared Memory*

3.2. Incompressible Navier-Stokes equations

The incompressible (viscous) Navier-Stokes equations are used in a wide range of applications in fluid dynamics. The further description refers to two dimensional flow problems. Consequently the equations will be composed of a mass conservation equation and two momentum conservation equations, one for each Cartesian velocity component [4], [12]. The dependent variables, which will be determined numerically, are the pressure *p* and the velocity components *u* and *v* in the *x* and *y* directions respectively (for simplicity only the isothermal case will be considered) [6]. The mass conservation equation is:

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0\,. \tag{1}$$

The momentum equations on the two axes are (Eq. 2-3):

$$\frac{\partial u}{\partial t}+\frac{\partial}{\partial x}(u^2+p)+\frac{\partial}{\partial y}(u\cdot v) =\frac{1}{\mathrm{Re}}\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right), \tag{2}$$

$$\frac{\partial v}{\partial t}+\frac{\partial}{\partial x}(u\cdot v)+\frac{\partial}{\partial y}(v^2+p) =\frac{1}{\mathrm{Re}}\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right). \tag{3}$$

The equations above are in the non-dimensional form which has been obtained using the free stream velocity V_∞, the density ρ_∞, the viscosity μ_∞ and a length scale *L*. Consequently, $\mathrm{Re}=\frac{\rho_\infty\cdot V_\infty\cdot L}{\mu_\infty}$. The main difficulty of the incompressible Navier Stokes equations is that there is no time dependent term in the continuity equation, which makes it difficult to solve for the pressure (the momentum equations can be solved for *u* and *v* but the continuity equation does not include *p*). One of the methods to solve this issue, is to include an artificial compressibility term in the continuity equation:

$$\frac{\partial p}{\partial t}+a^2\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)=0\,, \tag{4}$$

where *a* is the speed of sound. The term related to pressure will actually vanish as the scheme progresses to a steady state solution. Consequently, at steady state, the original continuity equation will be recovered.

The solution method is an explicit one [7] and it is implemented on a so called MAC grid (or staggered grid), which generally improves the stability of the incompressible flow (through the stronger coupling between pressure and velocity variables). Figure 4 shows a detail of the grid.

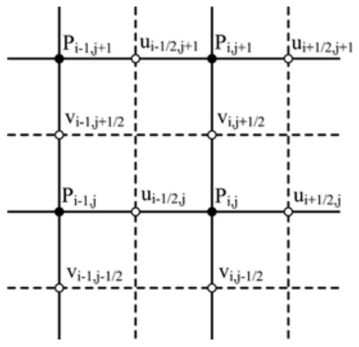

Fig. 4. *The MAC grid*

A primary (solid lines) and a secondary grid (dashed lines) can be distinguished on this MAC grid. Usually the pressure is defined on the primary nodes and the velocities on the secondary grid.

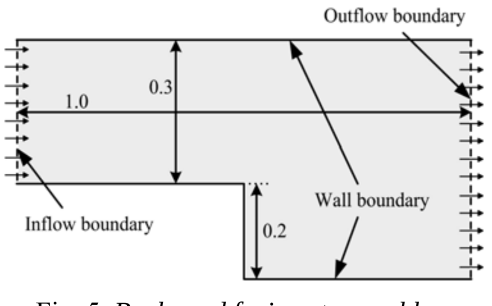

Fig. 5. *Backward facing step problem*

A first order difference in time and a second order central difference in space have been used in the discrete domain equations (2-4). The incompressible Navier-Stokes equations have been solved for a backward facing step problem, which is very popular in benchmarking activities. Figure 5 displays the domain of the flow.

3.3. Steady heat conduction problem

Another application which is common in computational fluid dynamics is the steady state heat conduction problem, or more generally the Laplace equation:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 . \qquad (5)$$

Generally, partial differential equations (PDE) may be divided into: hyperbolic, parabolic and elliptic equations. This classification of PDEs can be obtained by starting from the general form of quasilinear equations (the highest order derivative occurs linearly, i.e. there are no products or exponentials of the highest order derivatives) [1]. This approach of the classification of PDEs is common since all governing equations in fluid dynamics are quasilinear.

Eq. (5) can be easily discretized by using a five point finite difference scheme (Eq. 6):

$$\frac{T_{i+1,j} - 2\cdot T_{i,j} + T_{i-1,j}}{\Delta x^2} + \frac{T_{i,j+1} - 2\cdot T_{i,j} + T_{i,j-1}}{\Delta y^2} = 0. \qquad (6)$$

The numerical scheme which has been obtained is called explicit because it contains a single unknown value [2]. Because the goal of this paper is to perform a comparison of the single and double precision performance of a Tesla GPU, the heat conduction problem will be solved on a simple, rectangular domain as the one in Figure 6. The boundary conditions are known, and the boundaries have been chosen so as to coincide with the grid lines.

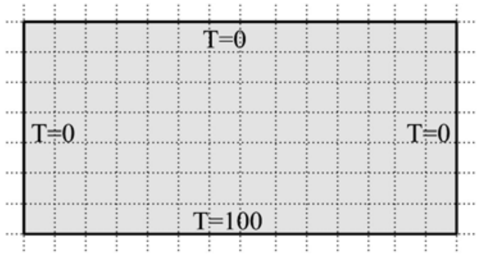

Fig. 6. *Rectangular domain of the problem and boundary conditions*

4. Results

The above described applications have been run for both single and double precision floating point on a NVIDIA GTX260 GPU. In order to deeply understand the results, one has to determine the main limitation factors for the three applications. These can be of three types:

- PCI Express Bus limited: most of the execution time is spent on the memory copies between the host and the device;
- Global memory bandwidth limited: the execution time is determined by the latencies of the global memory read/writes;
- Compute limited: the execution time is determined by the instruction throughput;

Tables 1, 2 and 3 display the summary results of the three types of applications for both single and double floating precision (the results have been recorded with the Visual Profiler).

By analyzing the values in the tables, the first type of limitation can be immediately excluded, since the memory copies operations occupy only a fraction of the total execution time (up to 20% for the matrix multiplication and under 3% for the

Summary table for the matrix-multiplication application (*size 1024x1024*) Table 1

Operation	Nr. of Calls	Execution time [µs]	% of Exec. time	Overall global mem. throughput [GB/s]	Instruction through-put
FLOAT kernel	1	10178.2	79.45	65.86	0.724
Memcopies	3	2633.1	20.55	-	-
DOUBLE kernel	1	32968.2	84.96	41.83	0.403
Memcopies	3	5838.1	15.04	-	-

Summary table for the incompressible Navier-Stokes application (*250 time steps*) Table 2

Operation	Nr. of Calls	Execution time [µs]	% of Exec. time	Overall global mem. throughput [GB/s]	Instruction through-put
FLOAT kernel	500	22908.5	97.63	86.32	1.013
Memcopies	9	556.2	2.37	-	-
DOUBLE kernel	500	150489	99.24	33.90	0.583
Memcopies	9	1078.2	0.76	-	-

Summary table for the steady state heat conduction problem (*250 iterations*) Table 3

Operation	Nr. of Calls	Execution time [µs]	% of Exec. time	Overall global mem. throughput [GB/s]	Instruction through-put
FLOAT kernel	500	39077.8	99.35	110.07	0.694
Memcopies	3	255.8	0.65	-	-
DOUBLE kernel	500	72372.5	98.01	111.29	0.416
Memcopies	3	1472.2	1.99	-	-

other two applications). In order to determine which of the other limitations applies for the three applications, some technical data of the GPU need to be taken into consideration. On the one side, the GTX260 allows an overall global memory throughput of 111.9 GB/s [9] and on the other side, the instruction throughput (ratio of achieved instruction rate to peak single issue instruction rate) can become greater than 1 (in the case of instruction dual-issue coming into play).

Considering now only the float versions of the applications, the steady state heat conduction problem is clearly limited by the global memory bandwidth, since the overall global memory throughput (110.07 GB/sec) is close to the peak value (111.9 GB/sec). The incompressible Navier-Stokes equation is clearly compute limited, since the instruction throughput exceeds the value of 1. The matrix multiplication application lies somewhere in between.

Before displaying the results of the comparison between float and double, expected values of performance decrease can be determined. For bandwidth limited applications the performance should decrease by a factor of two, since the application has to read twice more data for the double precision version than for the single precision version. For compute limited applications the performance should decrease by a factor of eight, since every multiprocessor contains eight single precision floating point units and only one double precision floating point unit. Figures 7, 8 and 9 display the results of the comparison for the three applications.

The measured values confirm the theoretical predictions, which have been based on the limitation of each application.

For the steady state heat conduction problem, the averaged measured execution time increase is of 1.95 and hence very close to the predicted value of 2.

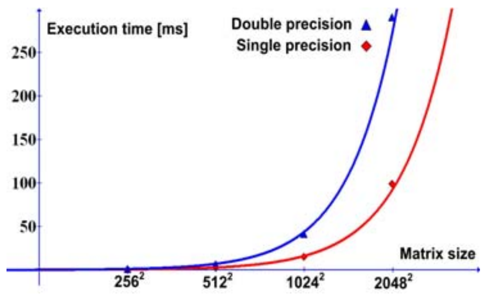

Fig. 7. *Comparison of the execution time for the single and double floating point precision for the matrix multiplication application*

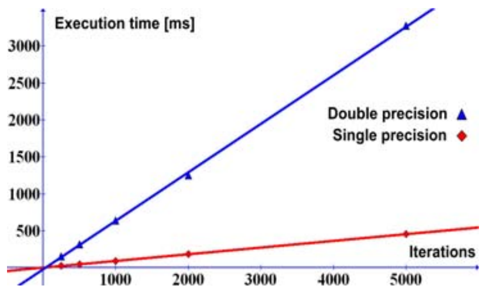

Fig. 8. *Comparison of the execution time for the single and double floating point precision for the incompressible Navier-Stokes application*

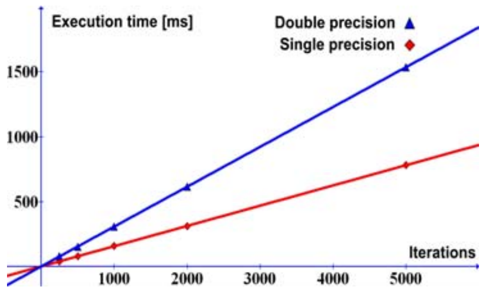

Fig. 9. *Comparison of the execution time for the single and double floating point precision for the steady state heat conduction application*

For the incompressible Navier-Stokes application the averaged measured execution time increase is of 6.98, i.e. also close to the predicted value of 8. For the matrix multiplication problem, which could not be categorized as a certain type of application, the averaged measured execution time increase is of 2.94, i.e. somewhere in between the limiting values of two and eight.

4. Conclusions and Future Work

This paper assesses the performance gap between the double and single precision performance of NVIDIA's Tesla architecture GPUs. Double precision floating point performance is very important for specific applications in order to obtain the desired accuracy for the results. A good example is the conjugate gradients method, with or without preconditioner, which is used to solve large and very large sparse linear systems of equations. Using single precision, the tolerance, i.e. the difference between the current and the actual solution, can not be diminished further than 1e-8. On the other side, using double precision for the floating point operations, values of 1e-20 or even lower for the tolerance, are no problems.

In order to analyze the performance gap, three representative applications have been chosen: matrix multiplication, incompressible Navier-Stokes equations and the steady state heat conduction problem. The second one is compute limited, the third one is bandwidth limited and the first application lies somewhere in between. The nature of each application has been determined through the values reported by the CUDA profiler, regarding execution time, global memory bandwidth and instruction throughput.

Bandwidth limited applications should have their performance decreased by a factor of two, because the number of bytes to be read is doubled (8 instead of 4).

Compute limited applications should have their performance decreased by a factor of eight because there are eight times less double precision than single precision floating point units in each multiprocessor of the GPU. These values are confirmed by the experiments. The execution time of the incompressible Navier-Stokes application increases by a factor of around seven, the execution time of the steady state heat conduction problem increases by a factor of two and the execution time of the matrix multiplication problem increases by a factor of three, confirming its mixed nature.

Future research activity will focus on the new Fermi architecture, which has been implemented inside the newest NVIDIA GPUs. The Fermi architecture should provide double precision performance which represents half of the single precision performance, instead of only an eighth as is the case with the Tesla architecture.

Acknowledgements

This paper is supported by the Sectorial Operational Programme Human Resources Development (SOP HRD), financed from the European Social Fund and by the Romanian Government under the contract number POSDRU/88/1.5/S/76945.

References

1. Blazek, J.: *Computational Fluid Dynamics: Principles and Applications*. London. Elsevier, 2007.
2. Bruaset, A.M., Tveito, A.: *Numerical Solution of Partial Differential Equations on Parallel Computers*. New York. Springer, 2006.
3. Chen, G., Li, G., et al.: *High Performance Computing Via a GPU*. In: 1st International Conference on Information Science and Engineering, Nanjing, China, Dec. 26-28, 2009, p. 238-241.
4. Chung, T.J.: *Computational Fluid Dynamics*. Cambridge. Cambridge University Press, 2002.
5. Cui, X., Chen, Y., et al.: *Improving Performance of Matrix Multiplication and FFT on GPU*. 15th International Conference on Parallel and Distributed Systems, Shenzhen, China, Dec. 8-11, 2009, p. 42-48.
6. Hoffmann, K.A., Chiang, S.T.: *Computational Fluid Dynamics*. Wichita. Engineering Education System, 1998.
7. Jin, Q., Thomas, D.B., et al.: *Exploring Reconfigurable Architectures for Explicit Finite Difference Option Pricing Models*. In: International Conference on Field Programmable Logic and Applications, Prague, Czech Republic, August 31-September 2, 2009, p. 73-78.
8. Kirk, D., Hwu, W.M.: *Programming Massively Parallel Processors: A Hands-on Approach*. London. Elsevier, 2010.
9. NVIDIA Corporation: *CUDA, Compute Unified Device Architecture Best Practices Guide v3.1*. Available at: http://www.nvidia.com. Accessed: 01-12-2010.
10. NVIDIA Corporation: *CUDA, Compute Unified Device Architecture Programming Guide v3.1*. Available at: http://www.nvidia.com. Accessed: 01-12-2010.
11. Owens, J.D., Houston, M., et al.: *GPU Computing*. In: Proc. of the IEEE **96** (2008) No. 5, p. 879-884.
12. Wendt, J.F.: *Computational Fluid Dynamics: An Introduction*. Berlin. Springer, 2009.
13. Zou, C., Xia, C., et al.: *Numerical Parallel Processing Based on GPU with CUDA Architecture*. In: International Conference on Wireless Networks and Information Systems, Shanghai, China, Dec. 28-29, 2009, p. 93-96.