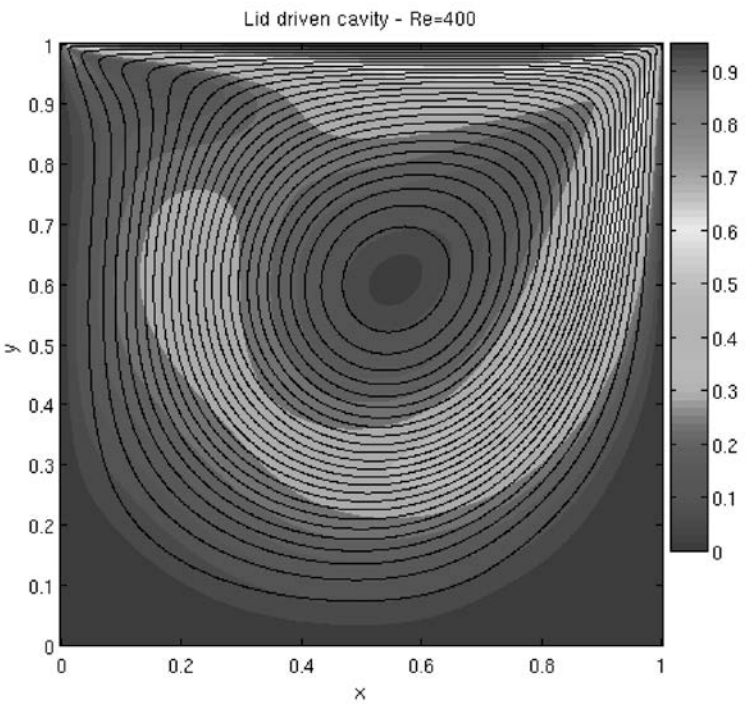
Application of the CUDA technology to the solution of fluid dynamics problems

Ediguer Franco*
Olmedo Arcila*
Santiago Laín*



Resumen

Este trabajo explora el uso de la tecnología CUDA en la solución de problemas relacionados con la dinámica de fluidos. Tres problemas clásicos de diferente nivel de complejidad: convección-difusión en un canal, la cavidad movida por pared y la cavidad movida por diferencia de temperatura, fueron solucionados por el método de las diferencias finitas, usando la CPU (procesador) y la GPU (tarjeta de video) para comparar el desempeño. Algunos aspectos importantes vinculados con la implementación numérica en la GPU son discutidos. Así mismo, los resultados mostraron un importante aumento de la velocidad cundo se usó la GPU.

Palabras clave: GPU, CUDA, dinámica de fluidos, método de las diferencias finitas.

Abstract

This work explores the use of the CUDA technology in the solution of fluid dynamics problems. Three classical problems with different level of complexity: advection-diffusion in a channel, lid driven cavity and thermally driven cavity, were solved using the finite difference method in both CPU and GPU in order to compare the computational

♂ Facultad de Ingeniería, Universidad Autónoma de Occidente. Calle 25 #115-85, Cali, Colombia.
Reception's date: 01/07/2014 • Aceptation's date: 04/06/2014.

performance. Important features related to the GPU implementation are discussed and results show an important increase in the computation speed with the use of the GPU.

Keywords: GPU, CUDA, fluid dynamics, finite difference method.

1I ntroduction

In Computational Fluid Dynamics (CFD), the study of flows requires intensive numerical calculations that, depending on the level of detail of the desired solution, could require an excessive processing time. In order to increase the computational power, scientists have developed computers with many processors working in parallel (computing clusters), but these computers are expensive and most scientist and small institutions have no the financial resources to afford one.

In the lasts years, some industries, particularly the entertainment industry, have required graphics with an increasing level of detail and real-time interactivity. In order to meet the demands, the graphic processor manufacturers have increased parallelism, developing devices with a grid of graphic processors that emulate computing clusters. Current graphic processors can feature thousands of processing cores in a single device, each of them improved for floating-point arithmetic operations.

The Compute Unified Device Architecture (CUDA) is a parallel computing technology developed by Nvidia for graphic processing. This technology includes a programming environment that make accessible to programmers the computing resources of the graphics devices, allowing the development of general purpose programs. Therefore, the graphic processing units (GPUs) can be used by scientist and engineers to increase their computing power with a modest investment of money. As a consequence, the general-purpose computing on graphic processing units (GPGPU) has arisen as an important topic of study.

Many works about methodologies for the implementation of numerical calculations on GPUs and the solution of engineering problems using these devices have been published in the lasts years. Some relevant work are related to the efficient implementation of linear algebra routines (Krüger & Westermann, 2003; Bell & Garland, 2008), solution of large linear systems (Bolz, Farmer, Grinspun & Schröder, 2003; Courtecuisse & Alard, 2009), finite element analysis (Cecka, Lew & Darve, 2011), FFT calculations (Moreland & Angel, 2003), computational fluid dynamics (CFD) calculations (Tölke & Krafczyk, 2008; Frezzotti, Ghiroldi & Gibelli, 2011), among others.

In this work, the use of graphic processor in the solution of fluid dynamics problems is evaluated.

Three classical CFD problems with well-known solution: advection-diffusion in a channel, lid driven cavity and thermally driven cavity, were solved using the finite difference method in both the CPU and the GPU. These problems are modeled by the Navier-Stok s equations and have different level of complexity.

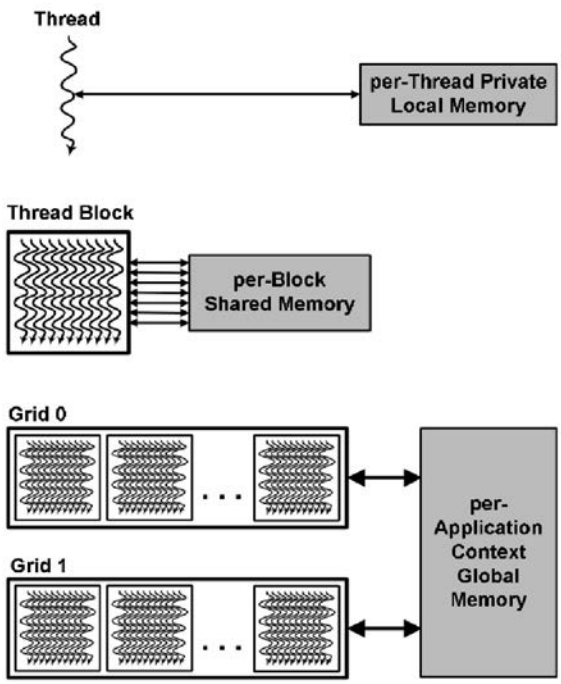
Relevant issues such as the numerical implementation in the GPU, restrictions and performance are discussed. It is important to point out the difference between the GPU and the CPU implementations. The parallel programing and the restricted features in the GPU require a change in the programing paradigm and some operations or algorithms of easy implementation in the CPU could be difficult in the GPU. Furthermore, GPU is highly hardware dependent. The performance results show the graphic device was more than a thousand fold faster than the processor for the simple advection-diffusion problem, but just threefold faster for the others, considerably more complex, problems. However, all GPU developed codes can be further optimized for better performance.

2. CUDA Technology

CUDA is a hardware and software architecture developed by Nvidia to execute programs in parallel (2009). This programs can be both graphic routines or general purpose programs that can be executed in the CUDA enabled graphic devices and written in diverse languages, such as C, C++, Fortran, OpenCL, Python, among others. This technology has had a significant impact in the scientific computing. As an example of this, Matlab, Mathematica, Ansys and other scientific and engineering software are developing interfaces to take advantage of the computational power available in the GPUs.

A CUDA program executes a kernel simultaneously. A kernel is a piece of code susceptible of being executed in parallel (see Table 1). For example, the scalar–matrix multiplication can be performed in parallel because each individual multiplication does not depend on the other ones and, thus, can be executed for a different processor at the same time. Other calculation routines cannot be directly parallelized. Each individual call to a kernel is called a thread, and a thread has an ID, a program counter, registers, per-thread private memory, input, and output results. A block is a set of threads concurrently executed. It shares a per-block memory (shared memory) and can be synchronized. A grid is an array of blocks executing the same kernel and it can read and write data in the global memory and synchronize between dependent kernel calls. Each block has an ID within the grid. Figure 1 shows the CUDA hierarchy of threads, blocks and grids.

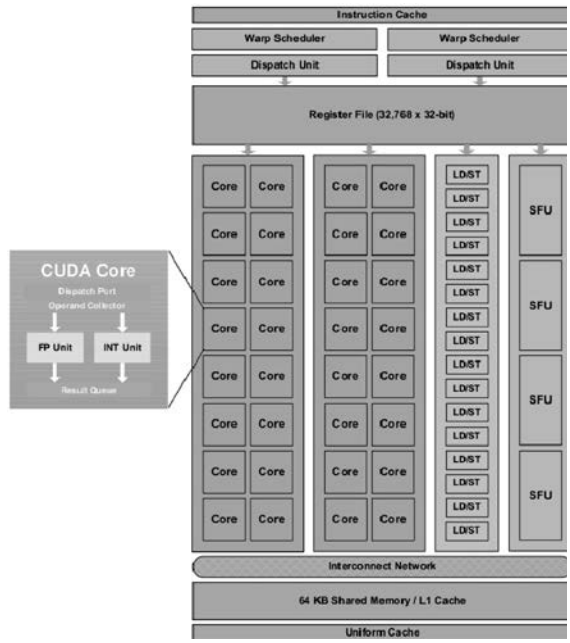
Figure 1 Hierarchy of CUDA threads, blocks and grids



Source: Nvidia (2009).

Figure 2 shows a scheme of the Fermi streaming processor (SM), featuring 32 cores. Each core has an arithmetic logic unit (ALU) and a floating point unit (FPU). The SM has four special function units (SFUs), that executes transcendental instructions such as sin, cosine, reciprocal, and square root, sixteen load/store units (LD/ST) for addressing, registers, L1 cache and scheduler and dispatch units. Fermi architecture implements the new and more accurate IEEE 754-2008 floating-point standard.

Figure 2. Fermi architecture



Source: Nvidia (2009).

Each device has an array of up to 48 streaming processors (up to 1536 CUDA cores), with a common L2 cache, six partitions of a 64-bit DRAM global memory, a PCI Express host interface for communication with the CPU and a global scheduler that distributes the thread blocks in the streaming processors. All these units work together, interconnected by a network like a computing cluster in a single device.

Software is another important issue. NVIDIA has released a development toolkit for the most important operative systems, including an extensive collection of code samples. These software tools have facilitated the understanding of the technology and its features to general purpose programmers and scientists. A CUDA program is compiled by the NVCC (Nvidia's CUDA compiler), available in the development kit. The executable contains the GPU code and CPU routines required to control the operation, i.e., data transfer between CPU and GPU and the kernels scheme of execution.

As an example, Table 1 shows the implementation of a function, that calculates the square of each element in an array, in both ANSI C and CUDA. The CPU code is executed in a serial way, that is, the operations in the for loop are performed in sequence (one after the other). On the other hand, the CUDA kernel can be executed in parallel, using different processing cores to execute the kernel (threads) simultaneously. If the number of operations (N) is less than or equal to the maximum allowable block size and the number of cores is enough, all operations could be performed in a single step. In general, the threads are arranged in grids of blocks that can be executed sequentially, in an order defined automatically by the GPU scheduler.

Table 1 Simple CUDA kernel example

```
// CPU code
void square(float * a, int N)
{
int k
  for(k=0;k<N;k++)
   a[k] = a[k] * a[k];
}
```

```
// CUDA Kernel executed in the GPU
global void square(double * a, int N)
{
  int idx = threadIdx.x;
     a[idx] = a[idx] * a[idx];
}
```

Source: by the author.

A CUDA kernel is defined by the keyword "global" and cannot return a value, therefore, it must be declared "void". It should be noted the lack of a "for" statement. In this case, the thread ID, accessible by means the built in variable "threadIdx", replaces the index loop and the kernel launching statement determines the size block(number of dispatched threads).

CUDA programming is different from the standard programming, because parallelism implies another programing paradigm. Some algorithms can be implemented in GPU almost translating the CPU code and other ones are difficult to parallelize. Moreover, hardware specific issues, specially the memory management, and the restrictions in the computing features lead to different programming techniques. Many information about the architecture and programming can be found in the CUDA documentation and the training resources available in the Nvidia site.

3M ethodology

3.1 Mathematical definition of the problem

The test problems are solutions of the transient incompressible Navier-Stokes equations. These equations are composed by the following set of partial differential equations (Versteeg & Malalasekera, 1995):

$$\nabla \cdot \mathbf{u} = 0 \qquad (1)$$

$$\rho\left[\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right] + \nabla p = \mu\nabla^2\mathbf{u} + S\mathbf{j} \qquad (2)$$

$$\rho\left[\frac{\partial T}{\partial t} + \mathbf{u}\cdot\nabla T\right] + \nabla p = \frac{\mu}{\mathrm{Pr}}\nabla^2 T \qquad (3)$$

where **u** is the velocity vector field, p is the pressure, T is the temperature, t is the time, ρ is the density, μ is the viscosity and Pr is the Prandt number.

Equation (1) is the continuity equation. It establishes the conservation of mass through the domain, restricting the velocity to a divergence free vector field. Equation (2) establishes the conservation of the momentum, where the term $(\mathbf{u}\cdot\nabla)\mathbf{u}$ is the convective acceleration, $\mu\nabla^2\mathbf{u}$ is the viscous forces, ∇p is the pressure gradient and S is a source term. Equation (3) is the energy equation, establishing the conservation of the internal energy in the domain, which, in the incompressible case, depends on the temperature only. In this equation, the term $\mathbf{u}\cdot\nabla T$ model the energy transport due to the fluid movement (advection) and the term $(\mu/\mathrm{Pr})\nabla^2 T$ model the heat conduction into the fluid.

The determination of the pressure scalar field, called pressure-velocity coupling is implemented using the projection method proposed by Chorin

(1968). In this method, the velocity field is calculated avoiding the pressure term, then, the pressure is used to project the velocity onto a space of divergence-free velocity field to get the next update of velocity and pressure. The pressure gradient can be interpreted as Lagrange multipliers, calculated in such a way to ensure that the resulting velocity field does satisfy continuity equation.

This advection-diffusion problem is modeled by the energy equation (3), where the velocity vector field (**u**) is known in the entire domain and the temperature field is unknown. For temperature, Dirichlet boundary condition is established for the hot region and von Neumann (adiabatic wall) elsewhere. The lid driven cavity problem is modeled by equations (1) and (2), where the velocity vector field (**u**) and the pressure scalar field (p) are the unknowns. For velocity, the Dirichlet condition (no-slip condition) is established in the entire boundary. For pressure, von Neumann condition (zero gradient normal to the boundary) is established. In the thermally driven cavity case, the problem is modeled by the three equations: continuity, momentum and energy. In addition to **u** and p, the temperature (T) is also an unknown. The boundary conditions for velocity and pressure are the same of the lid driven cavity problem ones. For temperature, Dirichlet condition is established in the left and right boundary and von Neumann condition (adiabatic wall) in the top and bottom ones.

32 Numerical implementation

The problems are solved by the finite difference method. The $L \times H$ rectangular domain is subdivided in $M \times N$ rectangles, equivalent to $(M+1) \times (N+1)$ nodes. The boundary is composed by two nodes, as required for the fourth order stencils employed [13]. For numerical implementation simplicity, only explicit schemes were used. Explicit schemes allow the evaluation of the derivatives avoiding the implementation of solvers for linear systems, as required by the more stable and precise implicit schemes. This approach required the minimum data storage, however, for numerical stability, the time increment can be very small, taking a long time to reach steady state.

The numerical implementation consists on the approximation of the derivatives finite difference stencils to obtain difference equations. The advection-diffusion problem is modeled by a unique difference equation that can be solved using a single CUDA kernel. In the other problems, the discretization problem lead to a set of difference equations, and they have to be solved simultaneously. In addition, the momentum and energy equations are evaluated term by term in a series of steps called “internal iterations”. Details of this method can be found in the work of Seibold (2008). Therefore, the solution code of these problems is more intricate and they are required four or more CUDA kernels.

33 Performance comparison

The most important concern about the numerical implementation of the solutions was the homogeneity between CPU and GPU codes. This means performing, as far as possible, the same calculations to allow comparison. Initially, each solution was programed in Matlab™ and the results compared to those obtained from literature. This step was carried out in order to validate the implemented solutions and to establish a set of simulation parameters. Then, the solutions were programed in ANSI C, avoiding the use of matrix or any other numerical libraries. Matrix algebra and file access routines were programed to assure the CPU and GPU code make, as possible, the same operations. Finally, the solutions are programmed in CUDA, following the same structure of the ANSI C code, and executed in the GPU. Table 2 shows the specifications of the computing devices.

Table 2. Main characteristics of the hardware

GPU	
Graphic device	NVIDIA GTX-570
Architecture	Fermi
Streaming processors	15
CUDA cores	480
Processor clock (MHz)	1464
Memory (Gb)	1.28
CPU	
Processor	Intel Xeon E5620
Architecture	X86_64
Cores	4
Processor clock (MHz)	2400
RAM Memory (Gb)	8

Source: by the author.

The processing time in both CPU and GPU was measured using the Linux system time library (time.h). This library provides functions to read the system time in resolution of miliseconds. Then, the system time is read at the beginning and at the end of the solution routines and the difference is calculated.

4Results and discussion

Figure 3 shows the temperature distribution and a representation of the velocity vector field for the advection-difussion problem. The domain is a rectangular channel of 0.5x0.1 meters with a fully developed flow of water at 20°C and a hot region at the bottom of the channel at 80°C. The velocity in the center of the channel (maximum velocity) is 1 cm/s. The domain was discretized using 640x128 subdivisions and the termination criterion were steady state (maximum difference of temperature between two successive iteration less than 10^{-8}). In this problem, the fluid next to the hot region is warmed (diffusion), at the same time; the hot fluid is transported downstream (advection). The temperature distribution shows the expected behavior.

Figure 4 show the solution of the lid driven cavity problem in a square domain of 1.0x1.0 meters with the top boundary horizontal velocity fixed to 1 m/s. The physical properties of the fluid were selected in order to have a Reynolds number of 400. The domain was discretized using 512x512 subdivisions and the program was executed 8000 temporal iterations, almost reaching steady state (the maximum difference of velocity magnitude between two successive iteration was less than $2x10^{-6}$). In this problem, the movement of top boundary induces a rotation of the fluid into the cavity, generating a vortex with center in and specific location that depends on Reynolds number. The figure shows the magnitude of the velocity vector field and the streamlines. This results are in agreement with those reported in literature (Ghia, Ghia & Shin, 1982).

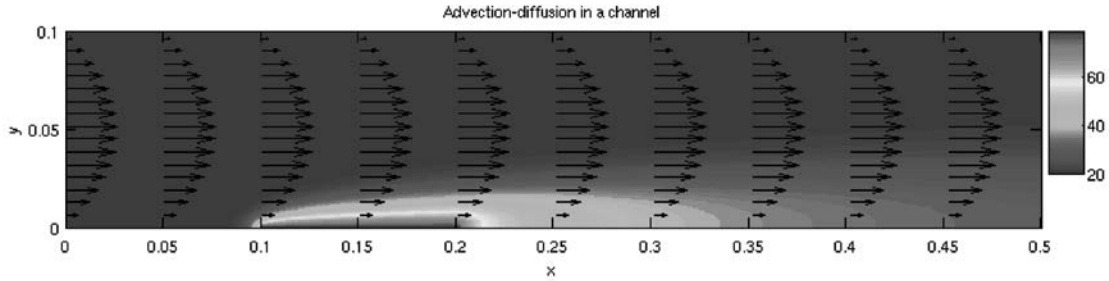
Figure 5 shows the solution of the thermally driven cavity problem in a square cavity of 0.5´0.5 meters. The fluid is air, and the domain size and the temperature difference were chosen in order to have a Rayleigh number of 10^6. The domain was discretized using 512x512 subdivisions and the program was executed 8000 temporal iterations, almost reaching steady state (the maximum difference of velocity magnitude between two successive iteration was less than $5x10^{-6}$). In this problem, the high temperature wall warms the surrounding fluid, generating a density reduction and, consequently, buoyancy. In this work, buoyancy is modeled using the Bousinesq approximation. The figure shows the temperature distribution and the streamlines. These results are in agreement with those reported in literature (Pérez, 1994).

Table 3 shows the comparison of the processing time for the solutions showed in figures 3 to 5. The processing times, in seconds, for CPU and GPU solutions are compared and the speed up (CPU to GPU processing time ratio) calculated. Additional information such as the domain size and the termination criterion are included.

The advection diffusion problem is relatively simple and it can be solved using a single CUDA kernel. In this problem, the speed up is many thousand times. It is consequence of the complete parallelism, that is, each iteration required to evaluate the spatial derivatives can be executed by an individual thread. This is an interesting result; however, the real problems are not so simple.

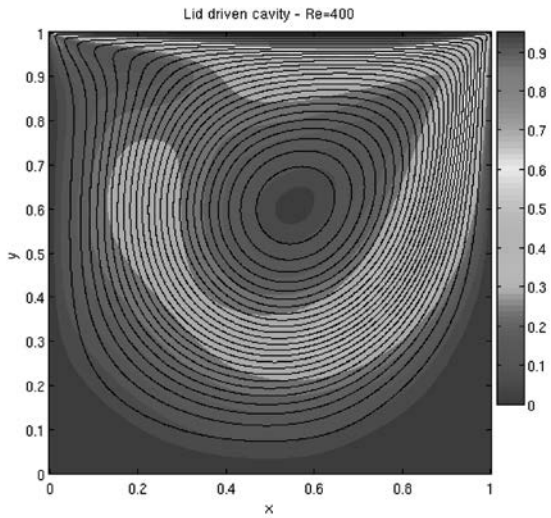
For the lid driven cavity problem, the speed up is just 3.6x, a drastically lower value. In this problem, each temporal step is obtained by the sequential execution of four kernels. The first kernel evaluates the convective terms, the second evaluates the diffusive terms and calculate the divergence of velocity, and the third and fourth kernels solve the pressure-velocity coupling and make the temporal update. For the thermally driven cavity, the speed up is 3.4x, similar to that of the lid driven cavity problem. The solution of this problem was implemented using six CUDA kernels, the same as the previous problem and two additional ones for the evaluation of the advective and diffusive terms of the energy equation. However, the larger number of kernels is not the main cause of the performance reduction. The main of performance reduction is the pressure-velocity coupling, because the finite difference scheme implemented can be partially parallelized.

Figure 3 Advection-diffusion problem between infinite parallel plates: temperature distribution and velocity vector field



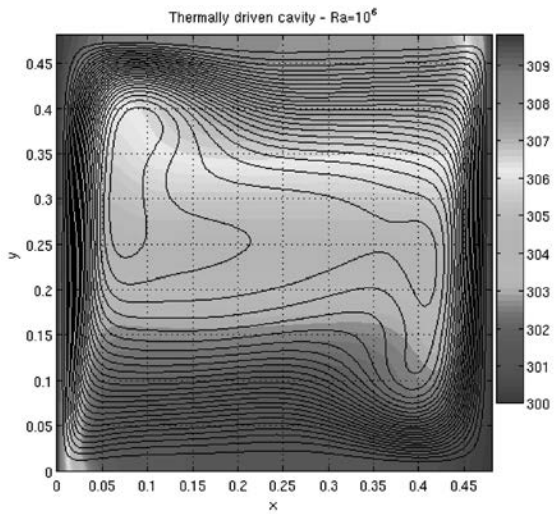
Source: by the author.

Figure 4. Lid driven cavity problem at Reynolds number of 400: magnitude of the velocity vector field and streamlines



Source: by the author.

Figure 5. Thermally driven cavity problem at Rayleigh number of 106: temperature distribution and streamlines



Source: by the author.

Table 3 Main characteristics of the hardware

Problem	Domain size	Termination criterion	CPU time (s)	GPU time (s)	Speed up
Advection-diffusion	640x128	Steady state	28991.0	0.539	~50000x
Lid driven cavity	512x512	8000 temporal iterations	686.5	188.9	3.6x
Thermally driven cavity	512x512	8000 temporal iterations	713.7	205.5	3.4x

Source: by the author.

5C onclusions

In this work, the use of the CUDA technology to the solution of CFD problems was analyzed. Three classical CFD problems with different level of complexity were solved using the CPU and the GPU. The performance analysis showed an important reduction in the computation time with the GPU, however, this reduction is highly dependent of the level of complexity of the problem. In the advection-diffusion problem, where the calculation reduces to the evaluation of a single finite difference stencil and parallelism is complete, the speed up is many thousand fold. However, for the other more complex problems the speed up is less

than fourfold. This is consequence of the greater number of CUDA kernel required and the partially parallelization of the solution scheme used for the pressure-velocity coupling.

The learning of CUDA programming is not easy. Parallelization issues, in addition to the restrictions in the computing features and the memory management of a GPU, make difficult the development of complex numerical algorithms. Nevertheless, with an acceptable expertise on the syntax of the CUDA language and the understanding of memory architecture and the hierarchy of threads, block and grids, it is possible to face the development of elaborated applications. Because the parallelization issues generate most implementation problems, it is highly recommended to write a serial code and then, when application gives correct results, incorporate parallelism. This technique proved valuable in simple implementations, and indispensable in the most complex.

Many code samples can be found in the CUDA Tool Kit and internet. These optimized codes are useful as examples or can be incorporated in the application, it is important to not reinvent the wheel. An example is the determination of the max/min value of a matrix in an efficient way. This k nd of problem is called "reduction", because, in order to exploit parallelism, the original matrix must be sequentially sectioned in smaller ones. This is a difficult programming problem, however, the Nvidia team provides different code samples that can be adapted or directly incorporated to the program.

The fluid dynamic problems selected and the numerical methods employed for the solutions are relatively simple. Real problems such as fluid dynamics in irregular and 3D-domains or finite elements calculations are more complex and the solutions involve sparse matrices algebra, factorization or inversion of large linear systems, submatrices, complicated indexing, etc. These problems bring new challenges. Fortunately, there are software libraries incorporating much of these required functionalities.

The codes implemented in this work are optimization susceptible. Actually, this is a first approaching to the CUDA technology and the performance improvement is an advanced issue yet to be done.

Finally, the CUDA technology has great potential as a scientific and engineering tool, because it brings high performance computing to universities and industries without the financial cost of a computing cluster. Nowadays, Nvidia has released their new GPU architecture called "Kepler" with features and performance superior to the one used in this work and the cost is comparable. An important feature is scalability, because it is possible to incorporate several devices to increment the computational power. On the other hand, the software is in constant improvement, including new capabilities in the language and libraries for specific tasks.

Acknowledgments

This research was supported by the Research and Technological Development Program of Universidad Autónoma de Occidente with the grant INTER11-147.

References

Bell, N. & Garland, M. (2008). *Efficient sp se mtrix-vecto mlu tip ictiion o CUDA.* Technical Report NVR-2008-004. NVIDIA Corporation.

Bolz, J., Farmer, I., Grinspun, E. & Schröder, P. (2003). Sparse matrix solvers on the gpu: conjugate gradients and multigrid. In *ACM SIGGRAPH 2003 Papers* (pp. 917-924). New YorkN Y, USA.

Cecka, C., Lew, A. J. & Darve, E. (2011). Assembly of finite element methods on graphics processors. *Intertib Jo h fo Nmericl Methl inE g e erig* , 85, (5), 640-669.

Chapra, S. & Canale, R. (2010). *Métodos m éricos p a ig in ers* . 6 ed. México: McGrawHill/Interamerica Editores.

Chorin, A. J. (1968). Numerical solution of the navier-stok s equations. *Mth mtics of Comp tio* , 22, 745-762.

Courtecuisse, H. & Alard, J. (2009). Parallel dense gauss-seidel algorithm on many-core processors. In *Proceedings of the 11th IEEE Intertib Cfpereo e o Hig Perfo mce Comp ig C mi ctis* . Seul, Korea.

Frezzotti, G., Ghiroldi, P. & Gibelli, L. (2011). Solving model kinetic equations on gpus. *Comp ers & Flids* , 50, 136-146.

Ghia, U., Ghia K. N. & Shin, C. T. (1982). High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. *Journal of Computational Physics*, 48, 387-411.

Krüger, J. & Westermann, R. (2003). Linear algebra operators for gpu implementation of numerical algorithms. In *ACM SIGGRAPH 2003 Papers* (pp. 908-916). New York, NY, USA.

Moreland, K. & Angel, E. (2003). The fft on a gpu. (2003). In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, HWWS '03* (pp. 112-119). Aire-la-Ville, Switzerland, Switzerland.

Nvidia Corporation. (2009). *Nvidia's next generation cuda compute architecture: Fermi*. Technical Report V1.1. NVIDIA Corporation.

Pérez Caeiras, C. (1994). *Desarrollo de métodos de vólumenes finitos para la resolución de las ecuaciones de Navier-Stokes incompresibles* (Phd. Thesis). Centro Politécnico Superior de la Universidad de Zaragoza, España.

Seibold, B. (2008). *A compact and fast Matlab code solving the incompressible Navier-Stokes equations on rectangular domains*. Massachusetts Institute of Technology. Retrieved from http://math.mit.edu/cse/codes/mit18086_navier-stokes.pdf.

Tölke, J. & Krafczyk, M. (2008). Teraflop computing on a desktop pc with gpus for 3d cfd. *International Journal of Computational Fluid Dynamics*, 22, (7), 443-456.

Versteeg, H. K. & Malalasekera, W. (1995). *An introduction to computational fluid dynamics*. England: Addison Wesley longman Limited.