ISSN 1870-4069

A Parallel Implementation on CUDA for Solving 2D Poisson's Equation

Jorge Clouthier-Lopez[1], Ricardo Barrón Fernández[2], David Alberto Salas de León[3]

[1,2] Instituto Politécnico Nacional, Centro de Investigación en Computación, Mexico
[3] Universidad Nacional Autónoma de México, Instituto de Ciencias del Mar y Limnología, Mexico

clouthier@gmail.com[1], rbarron@cic.ipn.mx[2], david.alberto.salas.de.leon@gmail.com[3]

Abstract. A parallel iterative Finite Difference (FD) method for solving Poisson's equation on CUDA is implemented. The aim of this paper is to give a detail explanation about the parallel solution of a Partial Differential Equation (PDE). To examine the performance of the implemented iterative algorithm, a number of experiments were tested. The performance shows the benefit of using the implemented approach on GPU devices in terms of execution time.

Keywords: numerical schemes, partial differential equations, GPU, CUDA.

1 Introduction

In parallel computing many calculations are carried out simultaneously in one or in different hardware environments. In this paper, we focus on the implementation of a parallel numerical scheme over GPU NVIDIA devices, as well as on the detailed explanation on the process of discretization and parallelization of a PDE. Parallel computing is applied to solve large problems over High Performance Computing (HPC). These large problems are mainly present in the research and forecasting of weather, climate (*e.g.* simulations of extreme events where atmosphere-ocean interactions have a great relevance, [1] and [2]), planetary sciences, and astronomy; as well as in engineering (*e.g.* computer, civil, and mechanical), where the reduction in computation time and the improvement in speedup is of fundamental importance.

More and more research has focused on solving numerical models applying different numerical schemes over HPC involving GPUs. With the latest advances on GPU cards, new numerical models on self or hybrid hardware have arisen, *e.g.*, MPI and CUDA combined, *e.g.* [3]. This of fundamental importance since large scale problems, represented by sets of PDEs and their parametrizations, *e.g.* source and sink terms that cannot be resolved directly [4], require high resolutions to reproduce many characteristics, that today are not possible. These large scale problems need substantial computation time and memory to run on ordinary computers. This means that powerful capabilities are required to obtain the numerical simulations in a reasonable period time.

FD schemes have been employed in many numerical models that are applied in different branches of geophysics. The content of this paper is based on the fundamentals that are being acquired to develop and implement a geophysical model with applications on oceanography and climate for the Ph.D. research of the first author. This model will be solved applying numerical hybrid methods based on its variational formulation.

The Poisson's equation is present in many geophysical models. In the solution of this equation, a numerical method is defined for a rectangular domain. As a result, a linear system of equations is found. Then, the system of equations is implemented on a CUDA parallel algorithm as a numerical linear algebra problem. Finally, the CUDA program is executed and the results are analyzed.

Numerical schemes based on FD are classical and straightforward ways to solve numerically both PDEs and systems of PDEs. It is well known that FD schemes are applied to solve systems of PDEs with many nodes on their structured domains [5]. The resulting algebraic systems are solved using iterative methods due to the fact that direct methods have disadvantages to calculate inverse matrices [6]. To reach higher accuracy and faster convergence rate, modifications have been made over iterative implicit methods, such as Jacobi, Gauss-Seidel and SOR, to have parallel algorithms, *e.g.* [7]. Moreover, pre-conditioning has also been applied for ill conditioned matrices that have large condition numbers.

In this paper we use the worth and simplicity that FD schemes have to solve the 2D Poisson's equation in parallel as a numerical modeling problem using CUDA C. A two-colored domain decomposition is applied in the FD discretization. The discrete equation is solved applying the Gauss-Seidel iterative method.

This paper is organized as follows. In Section 2, both the modeling problem and the 2D domain decomposition method are introduced. In Section 3, the parallel iterative finite difference algorithm and its implementation are presented. In Section 4, a brief overview about CUDA and the hardware architecture is provided. In Section 5, the numerical experiments and discussions are given.

2 Modeling Problem and Discretization

The 2D Poisson's equation is solved in the rectangular region **Ω**=[0≤x≤m]X[0≤y≤m]. This PDE is written as:

$$\frac{\partial^2\varphi}{\partial x^2}+\frac{\partial^2\varphi}{\partial y^2}=f(x,y), \tag{2.1}$$

with boundary conditions:

$$\varphi(x,y)=g(x,y), \qquad (x,y)\,\epsilon\,\partial\Omega. \tag{2.2}$$

The domain was divided uniformly into a number of shares with mesh grid size h, where

$$x_i=ih, \text{ and } y_j=jh, \quad i,j=1,2,3,\dots,n \tag{2.3}$$

with

Assuming a homogenous mesh, see Figure 1, each second derivative in (2.1) is approximated with centered FD according to the following expressions:

$$\frac{\partial^2\varphi}{\partial x^2} \approx \frac{\varphi(x+h,y)-2\varphi(x,y)+\varphi(x-h,y)}{h^2}, \tag{2.4}$$

$$\frac{\partial^2\varphi}{\partial y^2} \approx \frac{\varphi(x,y+h)-2\varphi(x,y)+\varphi(x,y+h)}{h^2}. \tag{2.5}$$

Substituting (2.4) and (2.5) into (2.1) we have

$$\begin{gathered}\varphi(x+h,y)+\varphi(x-h,y)+\varphi(x,y+h)+\\ +\varphi(x,y-h)-4\varphi(x,y)=h^2f(x,y).\end{gathered} \tag{2.6}$$

Writing (2.6) with subscripts

$$\begin{gathered}\varphi_{i+1,j}+\varphi_{i-1,j}+\varphi_{i,j+1}+\\ +\varphi_{i,j-1}-4\varphi_{i,j}=h^2f_{i,j}\ .\end{gathered} \tag{2.7}$$

Expression (2.7) is the discretized Poisson's equation. This expression will result in an algebraic system with n^2 equations of the form

$$AX=b. \tag{2.8}$$

where b is a known n^2-vector (its elements are boundary conditions in (2.2)), X is a n^2-vector to be determined (its elements are the function $\varphi(x,y)$ evaluated on the n^2 nodes of the mesh), and A is a n^2x n^2 matrix. A has the following form

$$A=\begin{bmatrix} C & E\\ F & D\end{bmatrix}, \tag{2.9}$$

with

$$C=D \text{ and } E=F^T,$$

where

$$C_{k,l}=-4 \text{ if } k=l,$$

and

$$C_{k,l}=0 \text{ if } k\neq l.$$

In order to arrive at the system (2.8), the first $n^2/2$ rows are obtained applying (2.7) on all nodes where $i+j$ is even and the following $n^2/2$ rows are obtained applying (2.7) on all nodes where $i+j$ is odd.

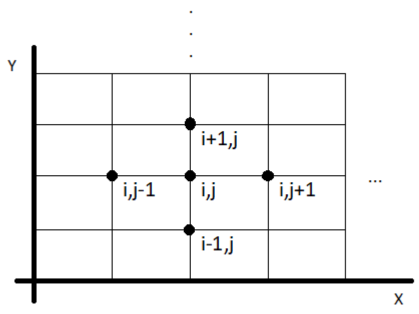

Fig. 1. The homogeneous mesh and the dependency of node i,j on the nearest ones.

3 The Iterative Algorithm

3.1 Sequential Gauss-Seidel

Before going directly to the parallel Gauss-Seidel (GS) method, we must first discuss the sequential one. The GS is an improvement of the Jacobi algorithm. It can be written in matrix and iterative notation. GS in the first form is expressed as

$$X^{k+1} = -(D+L)^{-1}UX^{k} + (D+L)^{-1}b, \tag{3.1}$$

where L, D, and U are the lower, diagonal, and upper-triangular parts of A, respectively.

In the second form GS is written as

$$X_i{}^{l} = \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{i,j}X_j{}^{l} - \sum_{j>i} a_{i,j}X_j{}^{k}\right), \tag{3.2}$$

where $l = k + 1$.

In (3.2) GS corrects the ith component of the vector $X_i{}^{k}$, where $i = 1, 2, \cdots n^2$. The approximation solution is updated immediately after the new component is determined.

3.2 Parallel Gauss-Seidel

In (2.8) A is a diagonal-dominant matrix, meaning that convergence will occur for the algorithm in (3.2). In applying (3.2) directly to (2.8), the $X_i{}^{k+1}$ for the first $n^2/2$ rows of the system of equations are calculated at the same time. Then the $X_i{}^{k+1}$ for the following $n^2/2$ rows are calculated also at the same time. The reason is that the $X_i{}^{k+1}$ for the last $n^2/2$ rows depend on the previous $X_i{}^{k+1}$, that were calculated in the first $n^2/2$ rows of the system of equations. This procedure is repeated many times until a desirable tolerance or approximation is reached. If the mesh ordering was not applied, the parallel procedure would not be possible.

Kernel Pseudocode

```
1: Define indices for the corresponding block and thread
2: Assign the index value to the corresponding thread
3: Select the number and order of the rows of $A$ to work with
4: Determine index_i and index_f according to the balance of rows in the grid
5: for (from index_i to index_f)
6:       Assign indices $i$ and $j$ to each node in the corresponding assigned
         rows of A, where $i+j$ is even, in order to not calculate zeroed
         elements
7:       Calculate $\varphi_{i,j}$ for the assigned rows, where $i=j$
8:       Calculate the local error
9: end
10: synchronize
11: for (from index_i to index_f)
```

```
12:         Assign indices $i$ and $j$ to each node in the corresponding assigned
            rows of A, where $i+j$ is odd, in order to not calculate zero elements
13:         Calculate $\varphi_{i,j}$ for the assigned rows, where $i=j$
14:         Calculate the local error
15: end
```

The CUDA kernel, which is executed in the device, is called from the host (CPU) many times until the desirable tolerance is reached.

4 CUDA Programming and Hardware Architecture

CUDA is a general purpose parallel computing architecture that exploits the parallel compute structure in NVIDIA GPUs to solve many complex computational problems. CUDA C extends C by defining C functions that are executed in the device (in the GPUs). Each function, also called a kernel function, is mapped to all the threads on the device. All the threads in a GPU make up a grid. The GRID is divided into blocks. Threads within a block can communicate with each other and synchronize together, while threads from different blocks cannot share information between them. This means that a kernel function is executed from all the threads in a grid. In other words, kernel functions are copied to all threads to be executed simultaneously [8].

GPUs were originally designed for graphics rendering. Due to their unique hardware architecture, they have become a powerful and suitable tool for general purpose computing. Each thread reads data in different memory locations when executing a kernel function and has its own registers and local memory. Each block has the same shared memory of its own and all threads in a grid can access the data in global memory. Additionally, there are five kinds of memory: register, shared, local, global, and constant [8].

5 Results

In this section, we evaluate experimentally the performance of the implemented algorithm on CUDA C. In the experiments we used 10 different node densities or problem sizes (36, 64, 100, 400, 1 600, 3 600, 6 400, 10 000, 40 000, and 160 00) for the mesh of the domain.

First, considering m=1, we solve (2.1) for four different cases where the numerical solutions in the Figures 2 to 5 are obtained considering a problem size of 3 600. This means that a higher value for the problem size would increase the resolution, while a lower value would present a poor resolution in the depiction of the solution of the PDE, as well a low quality of the numerical solution. Then we present the performance of the implemented algorithm.

The first three cases are solved taking into account homogeneous boundary conditions $(x = 0\ y = 0,\ \ (x, y) \in \partial\Omega)$ and the last case is solved with nonhomogeneous boundary conditions. For each case a different $f(x, y)$ in (2.1) is used.

i) First case

$$f(x, y) = \cos(2xy). \tag{5.1}$$

In Figure 2 the graph of the solution of (2.1) in the selected domain is presented.

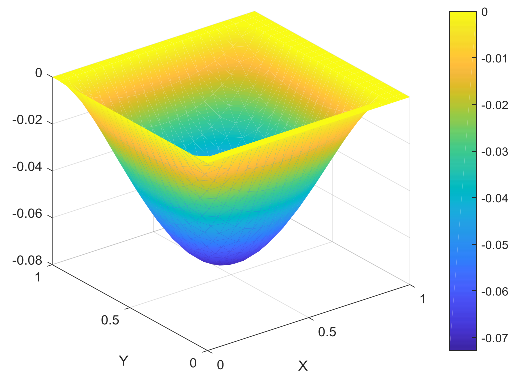

Fig. 2. Solution of Poisson's equation with $f(x,y) = \cos(2xy)$ and homogeneous boundary conditions.

ii) Second case

$$f(x,y) = \cos(2\pi x). \tag{5.2}$$

The solution for (2.1) for this case is presented in Figure 3.

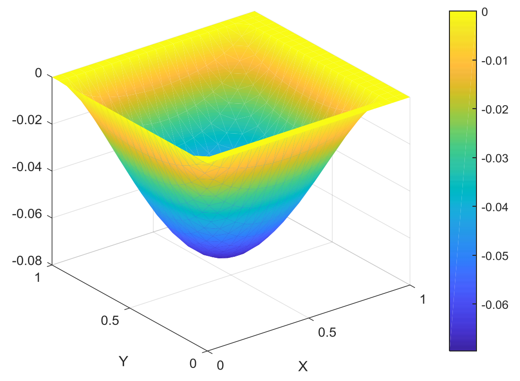

Fig. 3. Solution of Poisson's equation with $f(x,y) = \cos(2\pi x)$ and homogeneous boundary conditions.

iii) Third case

$$f(x,y) = 0. \tag{5.3}$$

The solution of (2.1) is presented in the following figure:

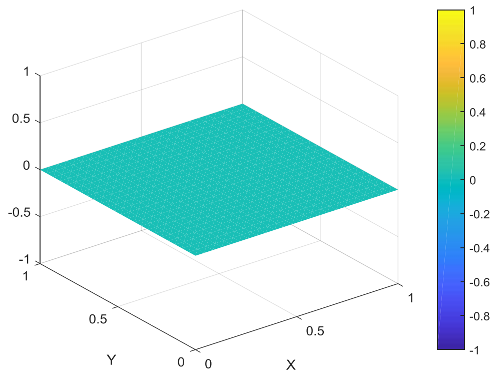

Fig. 4. Solution of Poisson's equation with $f(x,y) = 0$ and homogeneous boundary conditions.

iv) Fourth case

$$f(x,y) = 2x^2 + 2y^2 + 6x, \tag{5.4}$$

with the following boundary conditions:

$$x^3 + x^2y^2 \ (x,y) \in \partial\Omega.$$

The graph of the solution, in the selected domain, for this case is presented in the following figure:

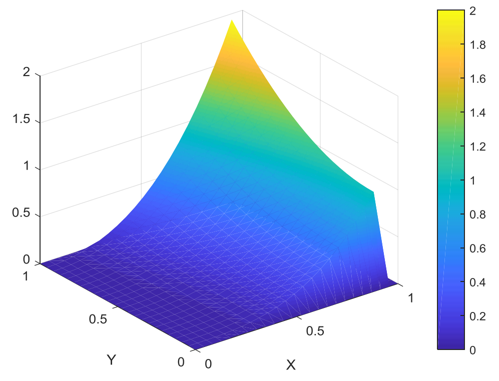

Fig. 5. Solution of Poisson's equation with $f(x,y) = 2x^2 + 2y^2 + 6x$ and nonhomogeneous boundary conditions equal to $x^3 + x^2y^2$.

In Figure 6 the execution time, in seconds, of the parallel algorithm for different problem sizes or node densities is presented. The sizes are obtained using 4, 9, 64, 100, and 200 threads. The results show that the parallel algorithm performs better as the domain resolution increases. In order to calculate the execution time, the fourth case is used. The reason is that it considers non-homogeneous boundary conditions.

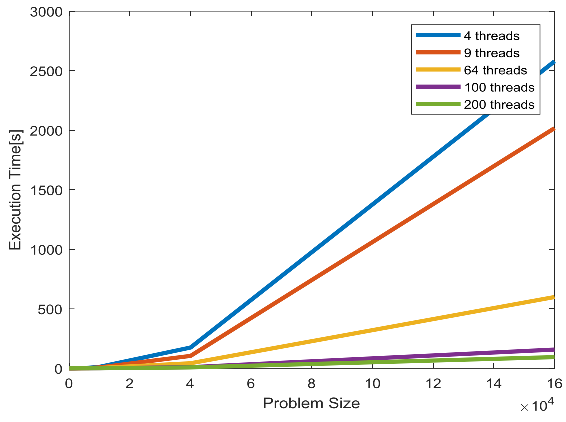

Fig. 6. The relationship between the execution time and the problem size.

Besides the execution time, the speedup also is calculated, see Figure 7. It is calculated selecting four different problem sizes: 3600, 10 000, 40 000 and 160 000 nodes. It can be seen that a better performance is obtained when large problems are considered.

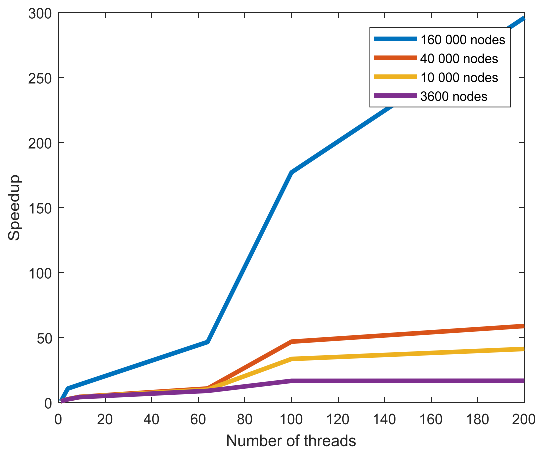

Fig. 7. The speedup for 3 600, 10 000, 40 000 and 160 000 nodes.

6 Conclusions

In this paper, we have implemented and analyzed a parallel implementation of the Gauss-Seidel algorithm with two-colored grid ordering to solve the 2D Poisson's equation on a GPU device using CUDA. The implemented parallel algorithm presents a good performance when the number of nodes of the mesh (discrete domain of the PDE) increases. This means that the behavior of the resulting parallel scheme is acceptable and good when the spatial resolution of the discrete domain is high.

The parallel algorithm takes advantage of the iterations to solve the linear system that results as a consequence of the discretization of the PDE.

Acknowledgements. The first and second author would like to thank to Instituto Politécnico Nacional for the support provided for the development of this work through the project SIP: 20181698

References

1. Skamarock, W., Klemp, J., Dudhia, J., Gill, D., Barker, D., Duda, M., Huang, X. Y., Wang, W., Powers, J.: A description of the Advanced Research WRF Version 3, NCAR Technical Note TN-475+STR, http.//www.mmm.ucar.edu/wrf/ (2008)
2. Regional Oceanic Modeling Syste: [Online]. Available: https://www.myroms.org/ (2018)
3. Salgueiro, D. V., Silvestre, N., Conde, D. A. C., Ferreira, R. M. L.: Implementation and experimental benchmark of a two-layer CPU+ GPU hydrodynamics model. Geophysical Research Abstracts, 20, EGU2018-18718 (2018)
4. Solano-Quinde, L., Gualan-Saavedra, R., Zuñiga-Prieto, M.: Multi-GPU implementation of the Horizontal Diffusion method of the Weather Research and Forecast Model. Proceedings of the 7th International Workshop on Programming Models and Applications for Multicores and Manycores, pp 98-103, Barcelona, Spain (2016)
5. Vázquez-Báez, V., M., Rubio-Arellano, A. B., García-Toral, D., Rodríguez-Mora, I: Model and Solution of Darcy's Law: Homogeneous and Inhomogeneous Media. arXiv:1802.00890 [physics.geo-ph].
6. Al-Towaiq, M H.: Parallel Implementation of the Gauss-Seidel Algorithm on k-Ary n-Cube Machine. Applied Mathematics, 4, pp 177–182 (2013)
7. Olszewski, L.: A Timing Comparison of the Conjugate Gradient and Gauss-Siedel Parallel Algorithms in a One- Dimensional Flow Equation Using PVM. In: Proceedings of the 33rd Annual on Southeast Regional Conference, Clemson, pp. 205–212 (1995)
8. NVIDIA, CUDA C Programming Guide. [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/ (2018)