C. VILACHÁ, Antonio F. OTERO, J. C. MOREIRA, E. MÍGUEZ

Department of Electrical Engineering of University of Vigo, Spain

Accelerated EMF Evaluation Using a SIMD Algorithm

Abstract. *This article presents a fast Single-Instruction Multiple-Data (SIMD) algorithm that evaluates electromagnetic fields (EMFs). It is based on the Method of Moments (MOM) adapted for execution on an SIMD architecture. The big speed-up obtained with this new implementation enables us to obtain results faster and to simulate more complex and realistic models and keep the computing time within a reasonable range, which give lead to better solutions for current EMF problems. This article gives a brief overview of generic massively parallel processors, taking into consideration their hardware architecture and the new computer languages for managing them. We describe the mathematical foundations of the algorithm in order to explain how the operations are distributed and performed by the GPU. Many cases are simulated to analyze the performance of the method proposed and they are compared with a fully implemented CPU algorithm, as well as with another CPU algorithm that uses the Intel MKL solvers for dense matrices. The differences in performance between floating-point precision numbers and double precision numbers is also studied and how they influence the accuracy of the results. The tests carried out suggest that the acceleration obtained grows with the complexity of the model. As a result, the proposed algorithm's only limitations lie with the hardware features.*

Streszczenie. *Artykuł przedstawia szybki algorytm typu Single-Instruction Multiple-Data (SIMD - pojedyncza instrukcja wiele danych), do obliczania rozkładu pól elektromagnetycznych (EMF). Jest ona oparta na metodzie momentów (MOM) przystosowanych do realizacji w architekturze SIMD. Duże przyśpieszenie uzyskane dzięki tej nowej implementacji pozwala uzyskać wyniki szybciej i dla bardziej złożonych symulacji i realistycznych modeli i utrzymać czas obliczeniowy w rozsądnym zakresie. Artykuł zawiera krótki przegląd procesorów masowo równoległych, biorąc pod uwagę ich architekturę sprzętową i nowe języki programowania do zarządzania nimi. Opisano matematyczne podstawy algorytmu, aby wyjaśnić, w jaki sposób operacje są wykonywane przez procesor graficzny (GPU). Wiele przypadków zostało symulowanych aby przeanalizować działanie proponowanej metody a wyniki porównano ze znanymi algorytmami, jak również z algorytmem, który wykorzystuje Intel MKL Solvers do gęstych matryc. Z przeprowadzonych testów wynika, że uzyskane przyspieszenie rośnie wraz ze złożonością modelu. Jedyne ograniczenia algorytmu zależą od możliwości sprzętowych. (**Przyspieszony algorytm analizy pól elektromagnetycznych za pomocą algorytmu SIMD**)*

Keywords: Algorithms, Electromagnetic analysis, Electromagnetic field, Parallel architectures, Simulation
Słowa kluczowe: Algorytmy numeryczne, Analiza pola elektromagnetycznego, Pole Elektromagnetyczne, Architektury równoległe, Symulacja

Introduction

An ever-increasing demand is being made by today's society on energy from different sources in order to sustain our development. There is no doubt that electrical energy occupies a dominant position among the various energy alternatives in industrial processes and domestic consumption; this is due to its versatility and reliability. In the future there will also be many opportunities to use electrical energy to solve environmental problems. In fact, many car-makers, electric companies and innovative start-ups are introducing electric cars, as well as intelligent appliances, as a means of going "greener". As a result, electrical facilities are rapidly growing in number and complexity.

The use of electric energy is, by its nature, linked to the existence of electromagnetic fields (EMFs) in the areas around the equipment and facilities; consequently, there are various safety and technical reasons for evaluating the electromagnetic fields generated by a facility.

Regarding safety, society is becoming increasingly concerned about the impact of EMFs on health. There are many studies on diseases—and even some genetic mutations which have resulted from exposure to low and high frequency EMFs. Indeed, there are committees such as ICNIRP [1] that provide guidelines which should be considered when designing electrical facilities.

Currently, electric companies are introducing communication technologies in facilities to make electrical networks evolve towards a more intelligent system known as Smart Grids [5]. These devices are specially susceptible to EMF fields, so an analysis of the facilities is required to ensure their electromagnetic compatibility (EMC). Article [9] presents an interesting vision of what the future holds for designers and describes the importance of communication systems. In the design stage, the interaction between the EMFs generated and the ground grid is also very important for guaranteeing the correct functioning of protective relays and for avoiding dangerous voltages on passive parts. In addition to these situations, there are many other technical concerns which currently make an evaluation of EMFs necessary.

Calculating EMFs has been a research topic for many years and its progress has always been closely related to advancements in computer technology [2]. As result of this research, many methods have been published, among them the Method of Moments (MOM), which is one of the most commonly used because of its versatility and robustness. This method has many variants, such as the one used by the authors in [19], and is reviewed in a subsequent section. Currently, more sophisticated techniques such as the Finite Element Method (FEM) and the Finite-Difference Time Domain (FDTD) are attracting more attention and growing in popularity. These methods draw on complicated mathematics that make them difficult to program and difficult to build their system models. For instance, creating an adequate 3D mesh that represents a given facility, is a major challenge in and of itself. In fact, many software companies only focus on developing EMF simulation software based on FEM.

All the previously mentioned methods share a common characteristic: the simulation time depends on the accuracy desired and the size of the simulated system. Moreover, problems are continuously growing in complexity and this implies that their simulations will have much larger computational costs. For these reasons, it is crucial to code algorithms efficiently and use new hardware architectures that speed up some types of calculations. Many papers are being publishing on this subject. In papers [18, 17] the authors obtained a remarkable performance which was 6 times faster at solving an EMF problem on an antenna using the Method of Moments. They used CULA libraries and the CUDA developing framework on an NVIDIA Graphics Processing Unit (GPU). The paper [11] shows a speed-up of almost 34 times using an FDTD algorithm. There are also implementations using FEM methods which obtain a major improvement, such as in [4]. To summarize, many authors have focused on accelerating EMF algorithms by using different known methods adapted to using new hardware capabilities.

This paper presents an implementation of a SIMD based on the Method of Moments. The objective is to improve calculation times by using a GPU. The algorithm is capable of solving EMF problems in low and middle frequency ranges.

Aside from the introduction, this paper is divided into six

additional sections. Firstly, the SIMD architecture and its programming philosophy are explained. Secondly, a brief mathematical approach to the algorithm used is presented. Then, the next section explains how the algorithm is implemented. Some cases are simulated in section five to compare the performance and the accuracy of the algorithm proposed. A real case is presented in section six that demonstrates the algorithm's capabilities, and lastly, the conclusions are given in section seven.

SIMD Architecture

This section presents an overview of the SIMD architectures, which are better known as General Purpose GPU (GPGPU) computation architectures. This computer technology is based on using GPUs to perform computations they were not initially designed for.

A GPU is a hardware component designed to speed up the creation of images to display them on a monitor by accessing the memory differently than a CPU.

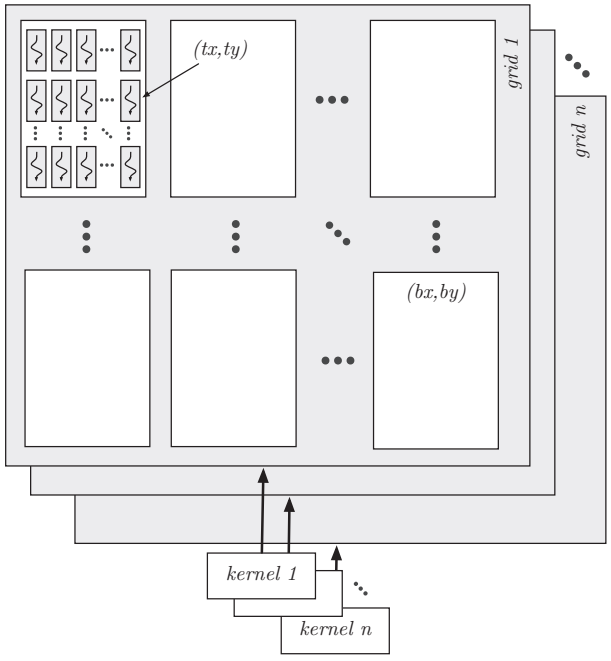

Fig. 1. SIMD architecture.

Modern GPUs perform extraordinarily well managing computer graphics due to their highly parallel architecture. This design also allows them to run algorithms that address large blocks of data faster than high-end CPUs.

Currently, GPUs are connected to the computer motherboard by an interface called PCI Express (PCIe). On the GPU board, there are two kinds of programmable processors, or arithmetic logic units (ALUs) [12]. There are vertex processors that work with meshes, doing mathematical operations and defining the position of the elements on the screen, and there are fragment processors that define colors, light and textures.

Traditionally, 3D computer graphics are created using application programming interfaces, such as OpenGL and DirectX. From the GPGPU perspective, these languages are focused primarily on graphics; their syntax is inadequate for carrying out general-purpose computations. To meet this need for high level languages, HAL and CUDA were developed by the two main GPU vendors, AMD and NVIDIA respectively, to make programming on graphics processors easier. In this article, NVIDIA's CUDA is used. The following paragraphs give an overview of this platform.

CUDA stands for Computer Unified Device Architecture [15, 10]. This technology makes it easier for users to interact with the graphics processor, especially if the user is not an experienced programmer. This architecture uses all the ALUs to perform the general-purpose computations. The CUDA platform complies with IEEE requirements for single precision floating-point arithmetic [20].

According to CUDA syntax, an algorithm is a $kernel$ and is executed by a $grid$ (see Fig. 1). A grid contains blocks organized in rows and columns that can be accessed by two indexes (bx and by). Also, each block contains threads that are again organized in rows and columns and are identified using two indexes[1] (tx and ty). These indexes are commonly used to read and write in the memory. In short, all the threads of a $grid$ execute the same piece of code; that is, a single instruction is run by multiple threads; this is an SIMD. These threads must not be seen as CPU-like threads because they are threads that are light-weight with zero overhead. Additionally, as Fig. 1 shows, several kernels can be executed in a same GPU using synchronization mechanisms.

From the programmer's perspective, the GPU is called $device$ and the motherboard, which includes the RAM and the CPU, is called $host$ (see the Fig. 2). A kernel is called from the $host$. When the $device$ has finished the calculations, it returns the results to the $host$. The user must be careful to copy the data from $host$ to $device$ properly, and vice versa, and must also be meticulous at setting up the number of blocks and threads according the input and output data. Any mistakes in these two steps will have a negative effect on the global performance of the algorithm.

CUDA provides a $local$ memory and three other specific memories that can lower the calculation time, but they must be used correctly [15], because each one is designed for a specific access pattern. The $local$ memory is global and can be read and written from the $host$ and accessed by all threads. The $shared$ memory allows threads of a block to share data among threads. The $constant$ memory is an overall data storage that can be accessed by all threads. It performs well if threads sequentially access a same location of this memory. Lastly, the $texture$ memory is also common to all threads. This memory speeds up the algorithm if threads access a memory location according to their position on the $grid$.

As mentioned previously, a GPU performs better than a CPU executing some types of algorithms, but not all. Therefore it is expected that most applications use both CPUs and GPUs, executing the sequential parts on the CPU and the numeric intensive parts on the GPUs. To do this, CUDA pro-

[1]There is an additional index tz to define 3D thread blocks, however some users do not recommend using this type of block.

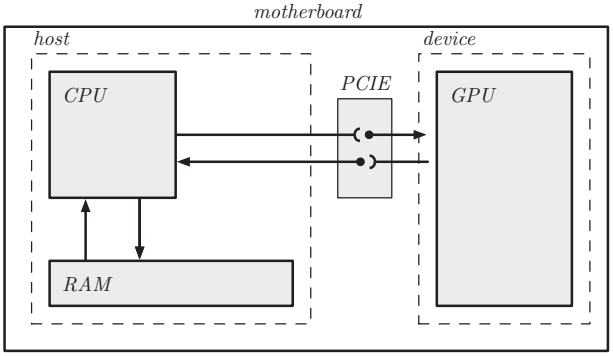

Fig. 2. Hardware diagram.

vides tools to split the execution of a given application between the CPU and the GPU. In these cases, the maximum acceleration obtained is defined by Amdahl's argument [6].

GPGPU computation is experiencing a huge growth in fields such as science, finance and engineering. This experience points toward a future where high level computing may be performed on personal computers.

Mathematical description

This section presents a brief mathematical description of the algorithm to emphasize the mathematic operations that the GPU has to perform. The algorithm draws on modeling a given system as a set of conductors placed in any position or orientation, surrounded by a medium such as air or earth. In order to reduce the model complexity, the following simplifications are applied:

1. Rectilinear conductors. Curves are interpolated by straight conductors.
2. All conductors are considered to have a round cross section. Calculating electrical parameters, such as self inductance, is easier with this simplification.
3. The conductor's length is greater than its diameter.

Consequently, the problem is reduced to obtain the equivalent circuit with r branches and n nodes. Each branch consists of an ohmic resistance and self and mutual inductances. Moreover, between every two conductors there are capacitive and conductive couplings that are represented, concentrating these phenomena in the circuit nodes. As a result, between each node a kind of π model is obtained (see Fig. 3). Two subcircuits are defined to solve the problem: a subcircuit where only the conductive couplings between branches, and branches and earth are considered (Fig. 3a), and another subcircuit that represents the resistance and the self and mutual inductances of the branches (Fig. 3b). By using this superposition technique, the analysis of the circuit is easier than when considering all the effects in a single circuit.

The next step consists of obtaining all the inductive, capacitive and conductive (when buried conductors exist) couplings between all system conductors, as well as the resistive increment due to proximity and skin effects on each conductor. Conductive couplings are only calculated for bare-buried conductors. These three elements are calculated using the following expressions [13, 14]. The total conductor resistance, R_{ca}, can be calculated by using (1).

(1)
$$R_i^{ca} = \frac{1}{r_i}\sqrt{\frac{f\mu\rho_i}{2\pi}}\left(\frac{J_0(q)\dot{J}_2(q) - J_2(q)\dot{J}_0(q)}{\left(\dot{J}_0(q)\right)^2 + \left(\dot{J}_2(q)\right)^2}\right) + R_i^{cc}$$

where, R_i^{cc} is the direct current resistance (this value must consider the environmental temperature); r_i is the conductor radius; f is the system frequency; μ is the magnetic permittivity of the medium; ρ_i is the conductor resistivity for a given temperature; J_α is the Bessel function of the first kind; q is $r\sqrt{2\pi f\mu/\rho_i}$.

The self inductance for a given conductor is calculated by adding the internal (2) and the external inductance (3) .

(2)
$$L_i^{int} = \frac{1}{2\pi r}\sqrt{\frac{\mu\rho}{2\pi f}}\left(\frac{J_0(q)\cdot\dot{J}_0(q) - J_2(q)\dot{J}_2(q)}{\left(\dot{J}_0(q)\right)^2 + \left(\dot{J}_2(q)\right)^2}\right)$$

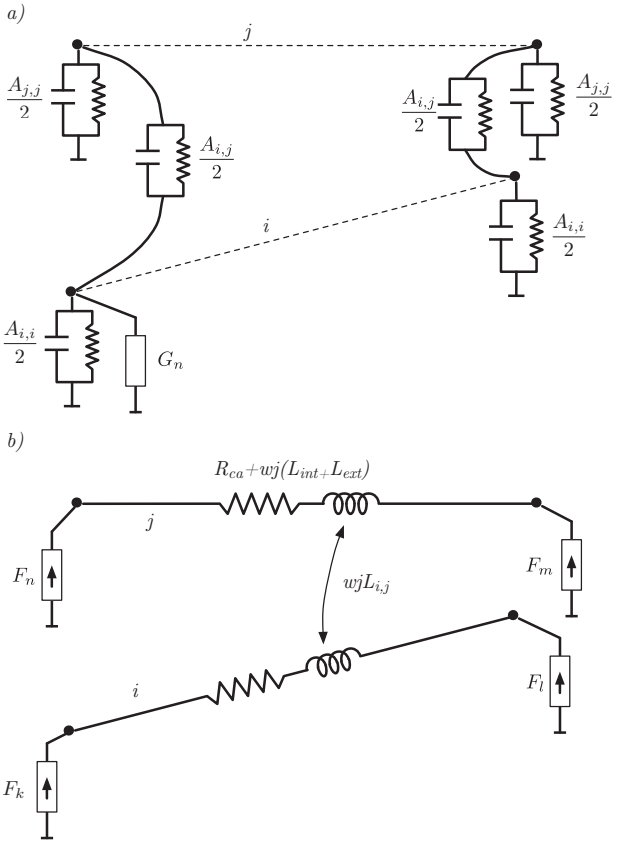

Fig. 3. Electrical Model.

(3)
$$L_i^{ext} = \frac{\mu}{2\pi}l\left(\ln\frac{2l}{r} - 1\right)$$

The mutual inductance is obtained using Neumann's equation (4):

(4)
$$L_{i,j}^m = \frac{\mu}{4\pi}\iint_{l_i,l_j}\frac{dl_i dl_j}{d_{i,j}}$$

where, l_j is the length of the j conductor; $d_{i,j}$ is the distance between conductors i and j.

The next equation (5) allows us to calculate the conductivity coefficients between conductors j and k:

(5)
$$\bar{A}_{j,k} = \frac{1}{\beta l_k}\left(\iint_{l_j,l_k}\frac{dl_j dl_k}{d_{j,k}} + \alpha\iint_{l_j,l_k}\frac{dl_j dl_k}{d_{j,\acute{k}}}\right)$$

where, α is $(\sigma + j\omega(\epsilon - \epsilon_0))/(\sigma + j\omega(\epsilon + \epsilon_0))$; β is $4\pi(\sigma + j\omega\epsilon)$; $d_{j,\acute{k}}$ is the distance between the conductor j and the image of conductor k.

At this stage, the passive elements of the circuit are already defined. The circuit energization is obtained by injecting the nodes with sinusoidal currents [13, 3]. The injected nodal currents are represented by $\bar{F}_j$.

As a consequence of energizing the circuit, a voltage appears between each node and the reference point (typically the earth); they are represented by a column vector $[\bar{V}]$ (see Fig. 3a). From these nodal voltages, the currents flowing from each branch towards the reference point can be obtained using the following equation:

(6)
$$[\bar{J}] = [\bar{A}]^{-1}\cdot[\bar{V}]$$

where, $[\bar{A}]$ is a nodal matrix of impedances, its elements are the $\bar{A}_{i,j}$; $[\bar{J}]$ represents the current leaked to earth from

each node.

Then, the nodal voltages and branch currents are obtained by solving the following linear system that corresponds with Fig. 3b.

$$\left[\bar{F}\right] - \left[\bar{J}\right] = \left[\bar{Y_{mc}}\right] \cdot \left[\bar{V}\right] \tag{7}$$

where, $\left[\bar{F}\right]$ is the vector of external nodal current sources; $\left[\bar{Y}\right]$ is the nodal admittance matrix of magnetic couplings formed by the resistances (1) and the self and mutual inductances, (2), (3) and (4), of the branches. The calculation of matrix $\left[\bar{Y_{mc}}\right]$ is not direct because of the mutual inductances. The existence of mutual impedances makes it necessary to use current-dependent sources which increases the complexity of the calculation. However, matrix $\left[\bar{Y_{mc}}\right]$ can also be obtained from the branch impedance matrix of magnetic couplings $\left[\bar{Z_{mc}}\right]$ by an easy transformation that comes from the relationship between branch and nodal currents according the Kirchhoff's law.

By operating the equations (7) and (6), the following expression (8) is obtained, where the nodal voltages vector, $\left[\bar{V}\right]$, is unknown.

$$\left[\bar{F}\right] = \left(\left[\bar{Y_{mc}}\right] + \left[\bar{A}\right]^{-1}\right) \cdot \left[\bar{V}\right] = \left[\bar{Y}\right] \cdot \left[\bar{V}\right] \tag{8}$$

where, $\left[\bar{Y}\right]$ is the resultant nodal admittance matrix.

After calculating the nodal voltages, the leaked branch currents $\left[\bar{J}\right]$ are obtained from equation (6). The currents that flow along each branch $\left[\bar{I}\right]$ can be easily obtained as the difference between $\left[\bar{F}\right]$ and $\left[\bar{J}\right]$.

With the computed branch currents circulating through the conductors and using the Biot-Savart equation, the magnetic fields can be calculated at any desired point P by superimposing the contribution of every i branch:

$$\overrightarrow{B_P} = \sum_{i=0}^{r} I_i \cdot \frac{\mu_0}{4\pi} \int \frac{\overrightarrow{dl}_i \times \overrightarrow{r}_{iP}}{r_{iP}^2} \tag{9}$$

Now, superimposing the voltage of each n node, the electric field at any desired point P can also be obtained.

$$\overrightarrow{E_P} = \sum_{i=0}^{r} \frac{1}{4\pi\left(\sigma + j\omega\epsilon\right)} \cdot \frac{J_i}{l_i} \int \frac{\overrightarrow{dl}_i \cdot \overrightarrow{r}_{iP}}{r_{iP}^2} \tag{10}$$

where, l_i is the length of the i conductor.

The maximum frequency of applicability of this method is limited by the assumed quasi-stationary approximation of the electromagnetic fields. Taking into account the usual size of the facilities considered (ranging from ten to a hundred meters), the method can be applied to up to tens of kHz.

Algorithm

This section explains how the aforementioned mathematical operations are done on an SIMD architecture. The entire code is programmed in unmanaged C++ language. Inverses of matrices and resolutions of equation systems are performed using commercial libraries called CULA Dense [7], which provide accelerated implementations of the LAPACK and BLAS libraries for dense linear algebra. All matrices and vectors involved in this problem are densely filled because a coupling effect exists between any given conductors. As a result, sparse techniques or simplifications cannot be applied. CULA libraries are proprietary software so how they use the GPU cannot be directly analyzed; however, benchmarking tests show a 10-times improvement compared with specialized CPU libraries such Intel's MKL.

The problem is solved entirely in the GPU; however, the algorithm is divided into ten parts that are called consecutively from the CPU to avoid running out of GPU memory. The CPU acts as coordinator, storing all the intermediate results and sending to the GPU only the necessary data for each part. After the GPU finishes its calculations, the CPU retrieves the results obtained by the GPU and disposes of the data that is no longer needed.

The algorithm's flow chart is shown in Fig. 4. The first step consists of defining the input data:

- System frequency.
- Conductors' edge positions and their connectivity.
- Conductors' resistivity and diameter.
- Characteristics of the medium such as magnetic permeability and electric permittivity.
- Energizing current sources and their connection.
- Loads and the nodes where loads are attached.
- Calculation points.

From these input data, all the data computational structures, such as matrix and variables, are created and then transferred to the GPU.

The equations (4) and (5) can be split into two parts: one part has a term which is only related to the position and size of the conductors and the second part has a term which consists of coefficients that are multiplied by the first term. For the mutual inductances, the first term is basically a double integral that involves the distances between conductors. This term is the same as the first double integral of the conductance equation (5), but the latter requires an additional double integral that involves the distance between each conductor and their images. The integrals cannot be resolved analytically, so a numeric cubature method [16] is used. These operations are done completely in parallel by executing a $kernel$ in the GPU, where each element of the matrices is computed by a dedicated $thread$. When the GPU finishes, it returns two matrixes that contain both integrals, to the CPU.

After solving for the integrals, the $A_{i,j}$ are ultimately obtained by computing the calculated double integrals and coefficients on a dedicated $kernel$ as the equation (5) indicates. More operations are required to create $\left[\bar{Z_{mc}}\right]$: firstly, the $L_{i,j}^{m}$ are calculated by operating the double integrals and the coefficients indicated in (4), then the self inductances are obtained by adding L_i^{int} and L_i^{ext}, according to (2) and (3), and the last step consists of determining R_i^{ca} as (1) describes. These operations are performed by a dedicated $kernel$ as well (see Fig. 4). Defining two different $kernel$ is not an insignificant choice: $\left[\bar{A}\right]$ and $\left[\bar{Z_{mc}}\right]$ have different sizes, the former is a nodal matrix and the latter is a branch matrix, which requires launching grids with different sizes because each element of these matrices are computed by a single $thread$. Splitting the operations is a good practice when transfer times between the CPU and GPU are insignificant regarding the global computation time.

In the step that follows, $\left[\bar{Z_{mc}}\right]$ is converted into a nodal matrix on the GPU via another $kernel$ and then it is inverted by CULA to obtain the nodal admittance matrix $\left[\bar{Y}\right]$, as the Fig. 4 indicates. Then the inverse of $\left[\bar{A}\right]$ is calculated.

The next step consists of obtaining the final admittance matrix $\left[\bar{Y}\right]$. This matrix is the result of adding $\left[\bar{Y_{mc}}\right]$ and the inverse of $\left[\bar{A}\right]$ previously calculated. The adding operation is also done by the GPU.

By solving the system with the final admittance matrix $\left[\bar{Y}\right]$ and the vector of current sources, the nodal voltage vec-

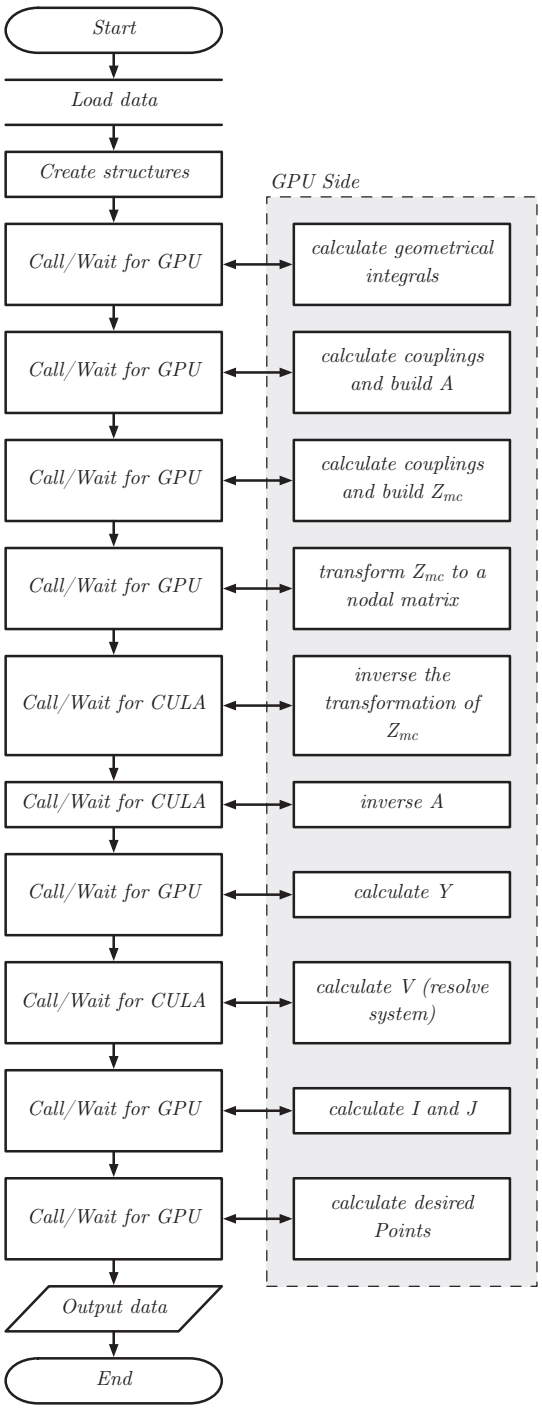

Fig. 4. Algorithm flow diagram.

tor $\left[\bar{V}\right]$ is obtained. This is accomplished using a CULA routine taking into consideration that $\left[\bar{Y}\right]$ is symmetric.

Going back to the development of the equations, the nodal voltage vector allows calculating the branch currents $\left[\bar{I}\right]$ and the leaked branch currents $\left[\bar{J}\right]$. As the Fig. 4 shows, these calculations are carried out by the GPU as well.

Lastly, the branch currents vector and the vector of branch voltages are sent to the GPU to calculate the electric and magnetic fields on the desired solution points. These points are stored in a file to plot figures and analyze the results.

Two additional variants were programmed to be executed entirely on the CPU to measure the performance and accuracy of the algorithm presented. The first variant uses the Intel's MKL library to inverse the two matrices and solve the system of equations. The rest of the calculations are executed sequentially. The second variant is programmed entirely by the authors and runs sequentially on the CPU. Both variants are coded in managed C++ language to obtain the best performance.

Simulation cases and performance

In this section the performance and the accuracy of the proposed GPGPU algorithm is analyzed. The hardware used is an Intel microprocessor i5 3.30 GHz with 4 MB Cache mounted on a motherboard with an Intel X58 chipset and 8 GB of DDR2 RAM. The computer has two GPUs: an NVIDIA GTX 285 that processes the graphics information for output to the display and an NVIDIA CUDA C2050 where the simulation is performed. The performance is measured by simulating five cases with different sizes. The simplest case is comprised of only two conductors, and the electric and magnetic fields are calculated on 200 points. The second case consists of 10 conductors and calculates the fields on 1,000 points. The third case defines a system of 400 conductors and calculates the electromagnetic field on 40,000 points. Even more complex is a fourth case, made up of 1,800 conductors and with 180,000 calculation points. And lastly, the most complex case is composed of 5,000 conductors and 500,000 calculation points.

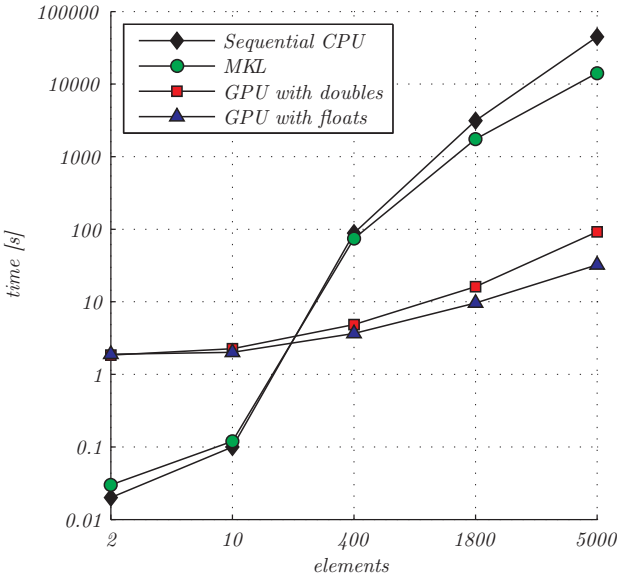

Fig. 5. Simulation times for implementations and different problem sizes.

These cases are simulated with four different implementations to compare their accuracy and calculation times. The first case is a sequential algorithm that is run entirely on the CPU, the second case is also run on the CPU but uses Intel's MKL to inverse the matrices and resolve the system of equations; cases three and four are the algorithm proposed in this paper executed in double-precision floating-point format ($double$) and then again executed in single-precision floating-point format ($float$), respectively. The results are shown in the Fig. 5 and Fig. 6.

The sequential implementation solves the simplest case in only 0.03 seconds, but the calculation time increases significantly as the cases become more complex; more than 12 hours are needed to solve the case of 5,000 conductors. The CPU implementation using the MKL libraries needs more time at the beginning than the simplest case, because adapting the data structures to be processed by the libraries takes some tens of milliseconds. As the number of elements in-

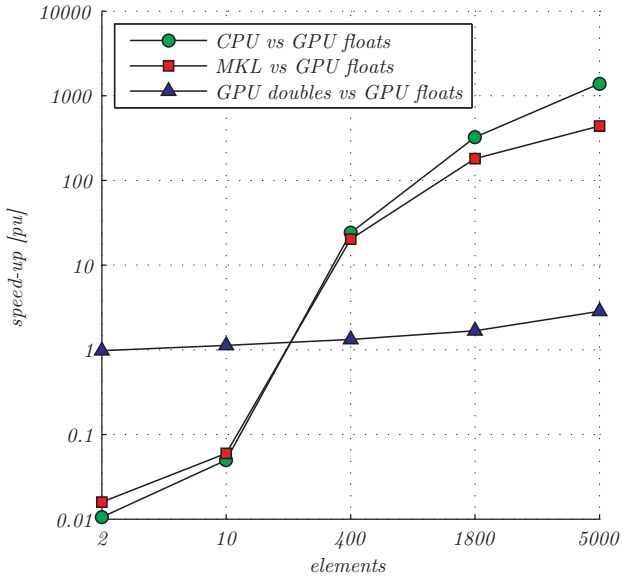

Fig. 6. Speedup.

crease, the MKL routines start to provide a significant improvement; in the most complex case the calculation is 3 times faster than the sequential algorithm (Fig. 5). Nonetheless, four hours is still a long time.

The proposed method executed either in double-precision or in float-precision floating-point format needs much more time (almost 50 times more) to execute the simplest case. This is because the data are transferred back and forth between the CPU and the GPU many times, and in small cases like the previous one, transfer times (around hundreds of milliseconds per transference) are more significant than the calculations themselves—like in the algorithm using Intel's MKL, adapting the data structures to the GPU also consumes time but it is insignificant when compared to the transfer times. For the 400-element case, the GPU provides an acceleration of almost 30 times faster than either of the two CPU algorithms. The acceleration grows with the complexity of the case, as Fig. 6 shows. The GPU solves the most complex case 450 times faster than the CPU-MKL algorithm, using float-precision numbers; and it is 1,300 times faster than the sequential CPU algorithm, which is an enormous improvement. In terms of time, this means reaching a solution in 35 seconds rather than 4 hours or 12 hours, respectively. Additionally, Fig. 6 demonstrates that the GPGPU computations are more effective the larger the problem is because the GPU can use all its microprocessors and take advantage of its architecture.

The difference between using float-precision and double-precision floating-point numbers is quite remarkable: the acceleration is about 4 times in the 5,000-element case. The better performance in single-precision is due to two reasons. Firstly, GPUs provide more computational power ($flops$) by using single-precision numbers instead of double-precision. Particularly, the NVIDIA C2050 has a performance of 515 $Gflops$ in double precision and 1.03 $Tflops$ in single precision. And secondly, less memory and transfer volume is required by using floats, which reduce transfer and allocation times.

By analyzing each part of the algorithm, the calculation of the double integrals and the calculation of the electromagnetic field on the desired points are the parts that take better advantage of the GPU. In both cases, the respective resultant matrix elements are calculated independently by a dedicated thread, which by doing basic arithmetic operations arrives at the final result. This computational schema is the best one to take advantage of SIMD architectures and obtain huge speed-ups. CULA routines also provide a speed-up of three times when compared to MKL.

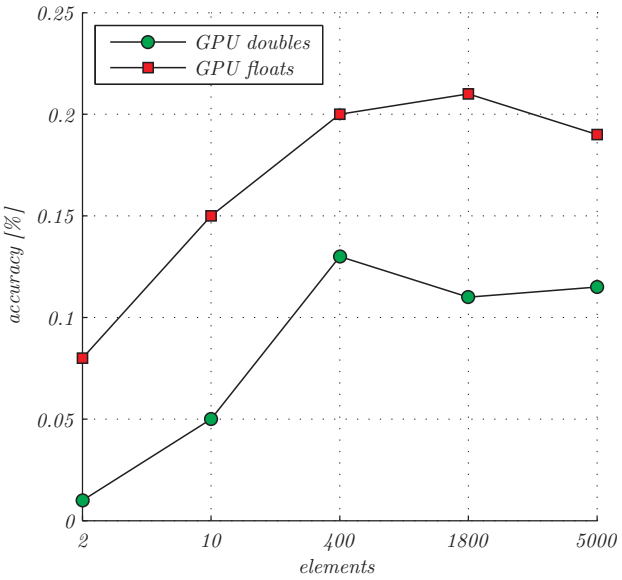

Fig. 7. Algorithm accuracy.

Fig. 7 shows an accuracy analysis of the proposed method for the different cases. The implementation that Intel's MKL utilizes is used as reference to make this comparison. By using double-precision format, the differences with the reference case are very subtle, near 0.13%. On the other hand, if the algorithm is running on single-precision format, some variations of 0.22% begin to appear, but they are not significant. By comparing the matrices obtained in each step of the algorithm, the authors conclude that these errors are made by CULA libraries during the inversion of the matrices and they are very dependent on the conditioning of the impedance matrix: slight variations in the value of the matrix elements cause significant variations in the final inverse. Fig. 7 also shows that although initially the error grows with the complexity of the cases, it stabilizes at same point after the case of 400 elements.

In short, the obtained accuracy is very good and makes the model applicable to any case. Improving matrix conditioning to reduce errors using techniques to create a more robust algorithm could be an interesting topic for research in the future.

Practical application

This section presents a practical application of the proposed algorithm, which corresponds with a previous paper presented by the authors [19].

In this paper, the authors introduced a simulation of the magnetic field generated by an outdoor substation. The results were validated by magnetic field measurements taken at the facility. The algorithm used to perform this simulation was a sequential implementation of the Method of Moments on the CPU. This is the same one being used as a reference for comparing the GPGPU algorithm currently proposed.

Despite calculating only the magnetic field in this simulation, more than 45 minutes was needed to compute the 1,800 elements and calculate the magnetic field on 200,000 points. When the simulation was performed again, this time using single-precision floating-point numbers when executing

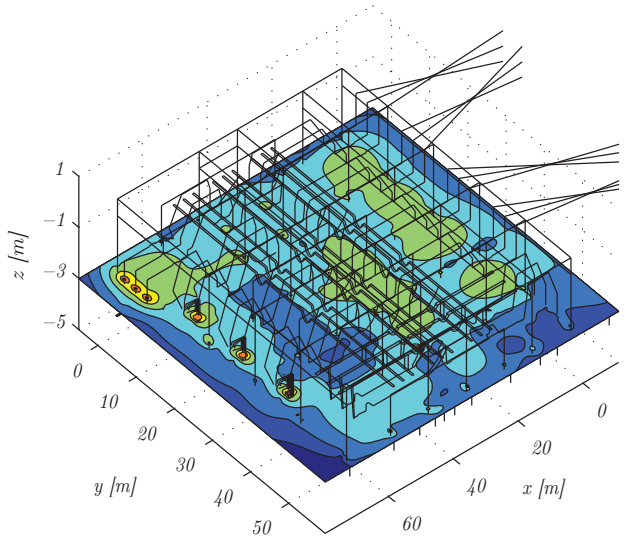

Fig. 8. Results of the simulation.

the case with the GPU, the simulation time was about 9 seconds. The results varied only 0.17% with respect to those which were obtained in the previous paper. Fig. 8 shows a surface plot with the new results.

Prior to developing this GPU algorithm, minor changes in the configuration of the substation, such as varying the load and the connectivity of the busbars, took 45 minutes to calculate. Now, the new algorithm requires only 9 seconds to calculate a variant, which is a major reduction in research time.

Conclusions

This article presents a new way to code the traditional Method of Moments. It is able to accelerate the calculating speed to more than 1,300 times when compared to a sequential implementation on a CPU, and to more than 450 times when compared to a CPU algorithm using Intel's MKL. The authors believe the acceleration speeds obtained bring many benefits.

Firstly, the presented acceleration constitutes a significant help for designing facilities such as substations. Performing complex electromagnetic simulations usually takes a lot of time and most engineers refrain from evaluating them if the designs meet public health guidelines for EMFs. With this proposed algorithm, however, engineers will quickly see the impact their designs will have on electromagnetic fields on a facility's workers [19] and on the neighborhood in the vicinity of the facility.

Secondly, more complex systems can be simulated. In the future, electrical facilities will require an even larger number of electronic devices to better manage the supply of electricity. These electronic devices are very sensitive to electromagnetic fields and require very accurate EMF analyses, meaning more detailed models will be needed. As the previous benchmarking shows, the algorithm presented in this paper can calculate huge models very rapidly and accurately.

Using GPUs to perform computations is a huge breakthrough in the simulation field. It means that some common problems in the electrical engineering field are being reformulated for adaptation to this new architecture; this includes power flows, optimal power flows (OPFs) and transient stability analysis [8]—they may even be addressed in a more accurate manner. Moreover, GPU devices are inexpensive and compact compared to computer clusters and calculation centers, and as a result, their presence will grow in professional and research settings.

C.Vilachá acknowledges to the Spanish Ministerio de Educación its support for his thesis and research activity by the FPU 2010 training program.

REFERENCES

[1] IEEE Standard for Safety Levels With Respect to Human Exposure to Electromagnetic Fields, 0-3 kHz. *IEEE Std C95.6-2002*, 2002.
[2] H.-D. Brüns, C. Schuster, and H. Singer. Numerical electromagnetic field analysis for EMC problems. *IEEE Trans. Electromagn. Compat.*, 49(2):253–262, 2007.
[3] J. Cidrás, A.F. Otero, and C. Garrido. Nodal frequency analysis of grounding systems considering the soil ionization effect. *IEEE Trans. Power Del.*, 15(1):103–107, 2000.
[4] N. Godel, N. Nunn, T. Warburton, and M. Clemens. Scalability of higher-order discontinuous galerkin FEM computations for solving electromagnetic wave propagation problems on GPU clusters. *IEEE Trans. Magn.*, 46(8):3469–3472, 2010.
[5] Vehbi C Gungor, Dilan Sahin, Taskin Kocak, Salih Ergut, Concettina Buccella, Carlo Cecati, and Gerhard P Hancke. Smart Grid Technologies: Communication Technologies and Standards. *IEEE Trans. Ind. Informat.*, 7(4):529–539.
[6] Carl W Hall. *Laws and Models: Science, Engineering, and Technology*. CRC Press, 1 edition, September 1999.
[7] JR Humphrey, DK Price, and KE Spagnoli. CULA: hybrid GPU accelerated linear algebra routines. *EM Photonics*, 2010.
[8] V Jalili-Marandi and V Dinavahi. SIMD-Based Large-Scale Transient Stability Simulation on the Graphics Processing Unit. *Power Systems, IEEE Transactions on*, 25(3):1589–1599, 2010.
[9] M. Kezunovic, Y. Guan, C. Guo, and M. Ghavami. The 21st century substation design: Vision of the future, 2010.
[10] David B Kirk and Wen-mei W Hwu. *Programming Massively Parallel Processors: A Hands-on Approach (Applications of GPU Computing Series)*. Morgan Kaufmann, 1 edition, February 2010.
[11] T Nagaoka and S Watanabe. A GPU-based calculation using the three-dimensional FDTD method for electromagnetic field analysis. In *Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE*, pages 327–330, 2010.
[12] Hubert Nguyen. *GPU Gems 3*. Addison-Wesley Professional, August 2007.
[13] A.F. Otero, J. Cidras, and J.L. del Alamo. Frequency-dependent grounding system calculation by means of a conventional nodal analysis technique. *Power Delivery, IEEE Transactions on*, 14(3):873–878, 1999.
[14] Clayton R Paul. *Analysis of Multiconductor Transmission Lines*. Wiley-IEEE Press, 2 edition, October 2007.
[15] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 1 edition, July 2010.
[16] S L Sobolev and Vladimir L Vaskevich. *The Theory of Cubature Formulas (Mathematics and Its Applications (closed))*. Springer, softcover reprint of hardcover 1st ed. 1997 edition, February 2011.
[17] T. Topa, A. Karwowski, and A. Noga. Using GPU with CUDA to accelerate MoM-based electromagnetic simulation of wire-grid models. *IEEE Antennas Wireless Propag. Lett.*, 10:342–345, 2011.
[18] T. Topa, A. Noga, and A. Karwowski. Adapting MoM with RWG basis functions to GPU technology using CUDA. *IEEE Antennas Wireless Propag. Lett.*, 10:480–483, 2011.
[19] C Vilacha, A.F. Otero, C. Garrido, and J C Moreira. Magnetic-Field Evaluation in the Vicinity of High-Voltage Electric Systems. *Power Delivery, IEEE Transactions on*, (99):1, 2012.
[20] N Whitehead. Precision & Performance: Floating Point and IEEE 754 Compliance for NVIDIA GPUs. *NVidia*, 2011.

***Authors**: M.Sc. C. Vilachá, Ph.D. Antonio F. Otero, Ph.D. J. C. Moreira and Ph.D. E. Míguez. Department of Electrical Engineering, Universidade de Vigo, Lagoas-Marcosende 9, 36310 Vigo, Spain, email: cvilacha@uvigo.es.*