Fast and efficient algorithms for computational electromagnetics on GPU architecture

Paolo Bettini, Tautvydas Maceina, Gabriele Manduchi, and Mauro Passarotto

***Abstract*—Integral formulations, suitable for the numerical solution of quasi-magnetostatic (eddy currents) problems in large and complex 3D domains, require specific post-processing tools to compute the effects of known current density distributions over elementary geometric entities (both mesh elements and field sources). The aim of this paper is to present a fast and robust implementation on a GPU architecture of an accurate algorithm for the computation of magnetic field and vector potential components.**

I. Introduction

INTEGRAL formulations can be more convenient than 3D finite-element-method (FEM) codes for the numerical solution of quasi-magnetostatic (eddy currents) problems in large and complex domains, consisting of many interconnected parts or components (e.g. magnetic confinement fusion devices), since they do not require the discretisation of non-conducting subdomains. A good accuracy is often achieved with a relatively coarse discretization, thus reducing the need of allocated memory and computing time. Moreover, suitable techniques (e.g. the Fast Multiple Method (FMM) [1] or the Adaptive Cross Approximation (ACA) coupled with hierarchical matrix (H -matrix) arithmetics [2]), can be used to overcome the impractical memory and computational time requirements which arise in very large scale models (integral formulations require the storage of dense matrices: the matrix size scales quadratically with the number of degrees of freedom n and its inversion has a computational cost of the order of n^3 for both direct and iterative solvers).

However, by following an integral approach, a specific post-processing tool is needed to evaluate the magnetic flux density and the magnetic vector potential components produced in the 3D space by known current density distributions over elementary geometric entities associated to the mesh elements (uniform polyhedral for 3D, or uniform polygonal sources for 2D) or to the sources themselves (2D axisymmetric massive or filamentary coils, 3D coils modeled by means of uniform polyhedral, polygons or current sticks).

Several analytic expressions for the calculation of the magnetic flux density and the magnetic vector potential produced by elementary geometric entities (bars, bricks, tetrahedrons, prisms with polygonal section and oblique ends, polygons, sticks, arc segments, etc) have been published by many authors [3], [4], [5], [6], [7], [8], [9], [10], [11]. These expressions allow a better accuracy with respect to those achievable by using pure numerical integration schemes, but are generally hard to implement in a fast and efficient way due to some peculiarities (a local coordinate systems is often introduced to perform the integration in a closed-form) or the presence of numerical issues (multiple valued inverse tangent functions) which require a careful programming.

The aim of this paper is to present a fast and robust implementation on a GPU architecture of an accurate expression (closed-form formulas) for the computation of the magnetic field and vector potential components produced by a general polyhedral source, as introduced in [12] and briefly recalled in Section II. Section III provides some details on the GPU implementation, in particular on how to benefit from their SIMD (Single Instruction stream Multiple Data stream) architecture, by programming each thread to compute the contribution to the magnetic field (or magnetic vector potential) of a single elementary source at a single field point [13]. Finally, Section IV presents a critical review of the results for a simple test case together with an overview of pros and cons of GPUs vs CPUs implementations.

II. Uniform current density sources

A general expression of the magnetic flux density and magnetic vector potential produced by a polyhedron with uniform current density has been introduced in [12] by means of a scalar function (here denoted as f_f) which can be expressed in terms of geometric elementary entities used in any solid modelling (nodes, edges, faces, volumes) and the standard incidence matrices (C, G, D) which relate each other in a cell complex (mesh). For the sake of completeness, we recall these expressions, in the following subsections.

A. Magnetic vector potential

The magnetic vector potential $\mathbf{A}$, produced by a uniform current density $\mathbf{J}$ inside a polyhedron v, at the field point $\mathbf{r}$, is given by

$$\boldsymbol{A}(\boldsymbol{r}) = \frac{\mu_0}{4\pi} \int_v \frac{\boldsymbol{J}}{|\boldsymbol{r} - \boldsymbol{r'}|} \, \boldsymbol{d^3 r'} \tag{1}$$

where $\boldsymbol{r'}$ is an arbitrary point of v.

With some algebra (see [12]), (1) can be recast as

$$\boldsymbol{A}(\boldsymbol{r}) = \frac{\mu_0 \boldsymbol{J}}{8\pi} \sum_{S_f \in \partial v} (\boldsymbol{r}_f - \boldsymbol{r}) \cdot \boldsymbol{n}_f f_f(\boldsymbol{r}) \tag{2}$$

Paolo Bettini is with the Università degli Studi di Padova, Padova, Italy, and with the Consorzio RFX, Padova, Italy, (e-mail: paolo.bettini@unipd.it).

Tautvydas Maceina is with the Università degli Studi di Padova, Padova, Italy, (e-mail: tautvydas.maceina@igi.cnr.it).

Gabriele Manduchi is with the Consorzio RFX, Padova, Italy, (e-mail: gabriele.manduchi@igi.cnr.it).

Mauro Passarotto is with the Università degli Studi di Padova, Padova, Italy, (e-mail: mauro.passarotto@studenti.unipd.it).

978-1-5090-2014-0/16/$31.00

where $\boldsymbol{n}_f$ is the outgoing normal unit vector of the planar face $S_f \in \partial v$, $\boldsymbol{r}_f$ denotes an arbitrary point of S_f and $f_f(\boldsymbol{r})$ is a scalar function defined as

$$f_f(\boldsymbol{r}) = \int_{S_f} \frac{1}{|\boldsymbol{r} - \boldsymbol{r}'|}\, \boldsymbol{d^2 r'} \tag{3}$$

where $\boldsymbol{r'}$ is an arbitrary point of S_f.

B. Magnetic flux density

The magnetic flux density $\mathbf{B}$, produced by a uniform current density $\mathbf{J}$ inside a polyhedron v, at the field point $\mathbf{r}$, is given by

$$\boldsymbol{B}(\boldsymbol{r}) = \frac{\mu_0}{4\pi} \int_v \frac{\boldsymbol{J} \times (\boldsymbol{r} - \boldsymbol{r}')}{|\boldsymbol{r} - \boldsymbol{r}'|^3}\, \boldsymbol{d^3 r'} \tag{4}$$

where $\boldsymbol{r}'$ is an arbitrary point of v.

Again, with some algebra (see [12]), (4) can be recast as

$$\boldsymbol{B}(\boldsymbol{r}) = \frac{\mu_0}{4\pi} \sum_{S_f \in \partial v} \boldsymbol{J} \times \boldsymbol{n}_f f_f(\boldsymbol{r}) \tag{5}$$

where $\boldsymbol{n}_f$ is the outgoing normal unit vector of the planar face S_f and $f_f(\boldsymbol{r})$ is the scalar function defined in (3).

C. How to compute f_f

With some algebra (see [12]), (3) can be recast as

$$f_f(\boldsymbol{r}) = f'_f(\boldsymbol{r}) + f''_f(\boldsymbol{r}) \tag{6}$$

The following expressions for the scalar functions f'_f, f''_f are adopted, which are well suited for the implementation on a GPU architecture,

$$f'_f(\boldsymbol{r}) = \sum_{l_e \in \partial S_f} \boldsymbol{n}_f \times (\boldsymbol{r}_e - \boldsymbol{r}) \cdot \boldsymbol{u}_e f_e(\boldsymbol{r}) \tag{7}$$

$$f''_f(\boldsymbol{r}) = (\boldsymbol{r}_f - \boldsymbol{r}) \cdot \boldsymbol{n}_f \Omega_f(\boldsymbol{r}) \tag{8}$$

where ∂S_f is the boundary of the planar face S_f (i.e. a set of oriented edges l_e), $\boldsymbol{n}_f$ is the outgoing normal unit vector of S_f, $\boldsymbol{r}_e$ is an arbitrary point of l_e, $\boldsymbol{u}_e$ is the unit vector of the edge l_e, $\boldsymbol{r}_f$ is an arbitrary point of S_f and $\Omega_f(\boldsymbol{r})$ is the solid angle seen from the calculation point $\boldsymbol{r}$ subtended by S_f.

$f_e(\boldsymbol{r})$ is a scalar function which can be computed using the intrinsic vector form introduced in [4] for a current stick with endpoints $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$,

$$f_e(\boldsymbol{r}) = \ln\left(\frac{|\boldsymbol{r_2} - \boldsymbol{r}| + |\boldsymbol{r_1} - \boldsymbol{r}| + |\boldsymbol{r_2} - \boldsymbol{r_1}|}{|\boldsymbol{r_2} - \boldsymbol{r}| + |\boldsymbol{r_1} - \boldsymbol{r}| - |\boldsymbol{r_2} - \boldsymbol{r_1}|}\right) \tag{9}$$

The solid angle $\Omega_f(\boldsymbol{r})$ can be computed in a very efficient way by invoking the additivity property of solid angles and splitting each face S_f into triangles (e.g. one triangle for each edge $l_e \in S_f$). Then the solid angle $\Omega_T(\boldsymbol{r})$ subtended by the triangular face ($\boldsymbol{r_i}$, $i = 1, 2, 3$ are its vertices) is computed with the intrinsic vector form given in [14], [15]

$$\Omega_T(\boldsymbol{r}) = 2 \arctan\left[\frac{(\boldsymbol{r_1} - \boldsymbol{r}) \cdot (\boldsymbol{r_2} - \boldsymbol{r}) \times (\boldsymbol{r_3} - \boldsymbol{r})}{D}\right] \tag{10}$$

with

$$D = |\boldsymbol{r_1} - \boldsymbol{r}||\boldsymbol{r_2} - \boldsymbol{r}||\boldsymbol{r_3} - \boldsymbol{r}| + |\boldsymbol{r_3} - \boldsymbol{r}|(\boldsymbol{r_1} - \boldsymbol{r}) \cdot (\boldsymbol{r_2} - \boldsymbol{r}) + \\ + |\boldsymbol{r_2} - \boldsymbol{r}|(\boldsymbol{r_1} - \boldsymbol{r}) \cdot (\boldsymbol{r_3} - \boldsymbol{r}) + |\boldsymbol{r_1} - \boldsymbol{r}|(\boldsymbol{r_2} - \boldsymbol{r}) \cdot (\boldsymbol{r_3} - \boldsymbol{r}) \tag{11}$$

III. Implementation on a GPU architecture

Relatively new GPU computation paradigm offers great merits in area of HPC (High Permance Computing). Scientific community is slowly but steadily adopting the it. However not every scientific problem can be solved efficiently on a GPU.

In general, problems of iterative nature, high level of computational complexity and high level of inter-dependence between constituents are less suitable for GPU computing.

Nevertheless many of these problems usually can be re-casted into more GPU-applicable formulation. In order to efficiently map the problem to a GPU architecture one has to find the most independent computation line in the problem and reshape the rest of the problem around it, i.e. one has to expose the "data parallelism" within the problem.

Often the problem has to be broken down into several more primitive parts in order to be efficiently programmed on GPU. In this sense (7)-(10) are very well suited, since they are compact and do not require any programming conditional statements. After these steps are completed one just simply assigns a GPU thread to each of those computation lines.

A. CUDA code development

The code development starts with a Matlab script that serves as prototype for C code, which later is translated into CUDA (Compute Unified Device Architecture). The prototype code consists of 3 nested loops:

1) Loop over sensors (n_s)[1]
2) Loop over volumetric source elements (n_v)[2]
3) Loop over faces of volumetric source elements (n_f)[3]

The magnetic field values (slowest loop) are accumulated in the two higher loops. Summation in GPU may not be a trivial task, if the sum value is shared between threads. Therefore a simple straightforward parallelisation could be implemented only for the first loop.

The pseudo-code for a GPU implementation is

1) Load GPU memory with:
   - sensor values (x, y, z coordinates)
   - current density values (real and imaginary parts of J_x, J_y, J_z components)
   - volumetric source data (faces, edges, nodes)
2) Launch the GPU kernel
   - Each thread loops over volumetric source elements
     - Each thread loops over the faces of volumetric source elements and computes (7)-(10)
3) Copy the acquired field values, according to (2) and (5), from GPU to host

IV. Numerical results

The proposed approach has been applied to a couple of simple test problems, to validate the CUDA implementation.

[1] In a magnetic confinement fusion device, n_s spans from some hundreds (real sensors) to several thousands if we refer to a grid of synthetic sensors.

[2] In a real application it spans from some thousands to hundred thousands mesh or integral source elements.

[3] $n_f = 4$ for tetrahedra, $n_f = 6$ for hexahedra, n_f arbitrary for polyhedra

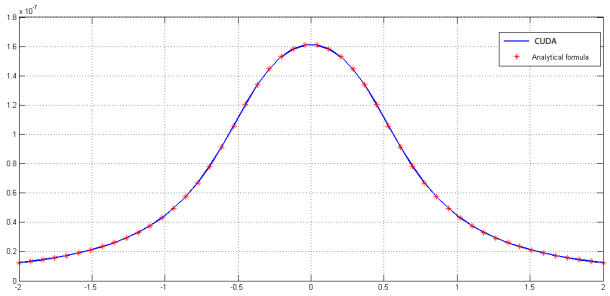

Fig. 1. First problem: a brick-shaped element with a uniform unitary current density. The $\boldsymbol{B}_x$ component computed with CUDA code (solid line) is compared to the exact values (stars).

CUDA $\boldsymbol{B}_x$ $[T]$	Analytical $\boldsymbol{B}_x$ $[T]$
$0.012239788877157 \cdot 10^{-6}$	$0.012239788877179 \cdot 10^{-6}$
$0.015776776206583 \cdot 10^{-6}$	$0.015776776206627 \cdot 10^{-6}$
$0.033089455139092 \cdot 10^{-6}$	$0.033089455139142 \cdot 10^{-6}$
$0.091304199096174 \cdot 10^{-6}$	$0.091304199096172 \cdot 10^{-6}$
$0.160898768631363 \cdot 10^{-6}$	$0.160898768631182 \cdot 10^{-6}$

TABLE I
$\boldsymbol{B}_x$ VALUES COMPUTED AT $x = \{-2, -1, 0, +1, +2\}$ WITH CUDA ROUTINES AND ANALYTICAL FORMULA.

First, a brick-shaped element ($1m \times 1m$ cross section, $100m$ long, centred in $(0, 0, 0)$) with a uniform unitary current density (J_y) is considered. It is a convenient problem since it has a well known analytical solution in terms of magnetic flux density components [10]. The field point $\boldsymbol{r}$ moves on a line through the element and its x coordinate ranges between $r_x = -2$ and $r_x = +2$, while the other coordinates are kept constnt ($r_z = 0.25$, $r_y = 0$, respectivley). As shown in figure 1, a perfect agreement is found between the $\boldsymbol{B}_x$ component computed with CUDA routines and the exact values. Table I summarizes the numerical results for a given number of field points $\boldsymbol{r}$.

Then, the current density induced in a conducting plate (discretised with 8×8 hexahedra) by a uniform vertical magnetic field (B_z=1T, f=50Hz) is computed with a volume integral code [16] and the proposed approach is used to evaluate the magnetic flux density produced on a grid of synthetic sensors placed above the plate, as shown in figure 2.

Nvidia GeForce GTX 480 was used to perform the calculation. GeForce GTX 480 is a graphics card originally designed for gaming purposes built on Fermi architecture and released in 2010. It runs $700MHz$ clock with $1.5GB$ of RAM and $48kB$ of shared memory.

With a simple straightforward parallelisation we get $0.031ms$ of kernel execution and $33.482ms$ for memory transfer from GPU to host. A single-threaded C code ran in about $371ms$. If only kernel execution is measured against single-threaded CPU code, it comprises more than $10000x$ gain in performance[4]. However if memory transfer is included into measurement, then only $\approx$ $10x$ gain in performance is observed comparing to single-threaded CPU code. It is a common nature of GPUs where memory transfer presents a

[4]The gain factor likely increases of an order of magnitude with the most recent and performing GPUs (e.g. Tesla K40 family).

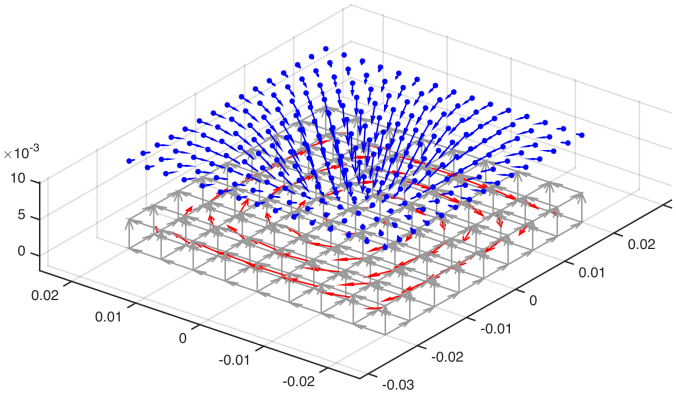

Fig. 2. Eddy currents induced by a uniform vertical magnetic field (B_z=1T, f=50Hz) on a conducting plate discretised with 8×8 hexahedra. Blue dots: synthetic sensors grid (15×15). Blue arrows: Magnetic flux density (a.u.), not to scale.

bottleneck in performance. The kernel and memory transfer scale differently from each other with regard to the increasing number of sensors. Kernel duration time stays constant, while memory transfer scales linearly, as shown in figure 3; we observe the aforementioned manifestation of memory bottleneck, that is inherently present in GPU codes. Moreover, this $\approx$ $10x$ gain would be even more diminished, if one compared against multi-threaded CPU codes.

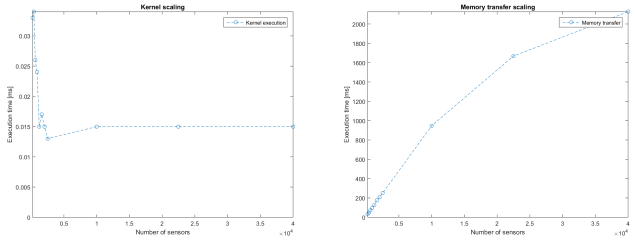

Fig. 3. Scaling of kernel execution (left) and memory transfer right) with respect to the increasing number of sensors.

V. CONCLUSIONS

A programmer should consider several points regarding the code implementation, both in terms of HW resources (GPU/CPU) and SW (platforms/compilers/libraries, etc).

1) GPU codes are good in solving problems of a "data parallel" nature. Most ofently it is up to a programmer to discover this feature in the problem. GPUs offer great computational gains for a trade-off of memory bottleneck and complicated programming process.
2) Single-threaded CPU codes are the simplest and easiest way to construct computation and usually are sufficient for a small caliber problems.
3) Multi-threaded CPU codes can be both good in solving problems of a "data parallel" nature and problems of a "task parallel" nature. Yet a "task parallel" problems are less common in physics and engineering. These codes are relatively easy to implement parallelization into a problem (many efficient libraries are available and

only a modest implementation effort is required to adapt Single-threaded CPU codes to $OpenMP$ directives).

REFERENCES

[1] L. Greengard and V. Rokhlin, A fast algorithm for particle simulations, J. of Comp. Pys., vol. 73, no. 1, pp. 325348, 1987.

[2] W. Hackbusch, A sparse matrix arithmetic based on H-matrices. part i: Introduction to H-matrices, Computing, vol. 62, pp. 89108, 1999.

[3] C. J. Collie, Magnetic fields and potentials of linearly varying currents or magnetization in a plane bounded region, in Proc. Compumag, Oxford, U.K., 1976, vol. 76, pp. 8695.

[4] J. D. Hanson and S. P. Hirshman, Compact expressions for the Biot-Savart fields of a filamentary segment, Phys. Plasmas, 9, 4410-4412 (2002). 11

[5] D. R. Wilton, S. Rao, A. Glisson, D. Schaubert, O. Al-Bundak, and C. Butler, Potential integrals for uniform and linear source distributions on polygonal and polyhedral domains, IEEE Trans. Antennas Propag., vol. AP-32, no. 3, pp. 276281, Mar. 1984.

[6] B. Azzerboni, E. Cardelli, M. Raugi, A. Tellini, and G. Tina, Analytical expressions for magnetic field from finite curved conductors, IEEE Trans. Magn., vol. 27, pp. 750757, Mar. 1991.

[7] G. Aiello, S. Alfonzetti, B. Azzerboni, S. Coco, and G. Tina, Analytical computation of magnetic vector potential from tetrahedral conductors, IEEE Trans. Magn., vol. 28, no. 5, pp. 20452050, Sep. 1992.

[8] S. Pissanetzky and Y. Xiang, Analytical expressions for magnetic field of practical coils, COMPEL, vol. 9, no. 2, pp. 117121, 1990.

[9] I. R. Ciric, Simple analytical expressions for the magnetic field of current coils, IEEE Trans. Magn., vol. 27, no. 1, pp. 669673, Jan. 1991.

[10] L. Urankar, Vector potential and magnetic field of current-carryingfinite arc segment in analytical formPart III: Exact computation for rectangular cross section, IEEE Trans. Magn., vol. MAG-18, pp. 18601867, Nov. 1982.

[11] L. Urankar, Vector potential and magnetic field of current-carrying finite arc segment in analytical formPart V: Polygon cross section, IEEE Trans. Magn., vol. 26, pp. 11711180, May 1990.

[12] M.Fabbri, Magnetic Flux Density and Vector Potential of Uniform Polyhedral Sources, IEEE Transactions on Magnetics, vol 44, no.1, 2008

[13] A. G. Chiariello, A. Formisano, R. Martone, Fast magnetic field computation in fusion technology using GPU technology, Fusion Engineering and Design 88 (2013) 1635 1639

[14] R. Courant and D. Hilbert, Method of Mathematical Physics. New York: Interscience, 1962, vol. 2, p. 246.

[15] A. van Oosterom and J. Strackee, The solid angle of a plane triangle, IEEE Trans. Biomed. Eng., vol. BME-30, no. 2, pp. 125126, Feb. 1983.

[16] P. Bettini, M. Passarotto, R. Specogna, "A volume integral formulation for solving eddy current problems on polyhedral meshes", submitted to CEFC 2016 Conference, 13-16 November, 2016, Miami (USA)