Journal of Microwaves, Optoelectronics and Electromagnetic Applications, Vol. 19, No.2, June 2020
DOI: 10.1590/2179-10742020v19i2851

GPU Finite Element Method Computation Strategy Without Mesh Coloring

Lucas Amorim[1,2], Thiago Goveia[1], Renato Mesquita[2], Igor Baratta[2]
[1]*Federal Center for Technological Education of Minas Gerais — CEFET-MG*
R. Dezenove de Novembro, 121, 35180-008, Timóteo, MG, Brazil,
[2]*Graduate Program in Electrical Engineering, Universidade Federal de Minas Gerais — UFMG*
Av. Antônio Carlos 6627, 31270-901, Belo Horizonte, MG, Brazil.
lucaspa@gmail.com, gouveiat@gmail.com, renato@cpdee.ufmg.br, igorbaratta@gmail.com

***Abstract*— A GPU-adapted Finite Element Method (A-FEM) implementation without using a mesh coloring strategy to avoid race conditions is presented. All the elements are simultaneously processed, and the local data is computed and consolidated into the device memory in a new adapted Coordinate Format data structure (a-COO). The comparison between the proposed solution and two GPU Element-by-Element Finite Element Method (EbE-FEM) approaches shows promising results since it is no longer necessary the mesh coloring step. This new approach is well suited for problems that need re-meshing and consequently would need re-coloring, e.g., adaptive mesh refinement.**

***Index Terms*—** FEM, GPU, linear equations, CUDA.

I. INTRODUCTION

The Finite Element Method (FEM) is one of the most used techniques to approximate the solution of complex engineering problems. In FEM, the computational domain of a linear boundary value problem is discretized with a mesh, and a system of linear equations in the form $Ax = b$ is generated. With the advent of General Purpose Graphics Processing Unit (GP-GPU), several works that harness the massively distributed properties of these devices have been proposed in the literature with the aim of improving computational performance. See for example [1]-[3], [8]-[13], [18]-[19] and references therein.

A redesigned strategy for use in GPUs is the Element-by-Element Finite Element Method (EbE-FEM) [6], [14]. In this method, each element is processed in a device core without the necessity of global stiffness matrix assembling. However, a mesh coloring strategy is required to circumvent the race condition: the mesh elements are computed in independent groups that share the same color, in as many groups as the colors. Group processing needs to be done even within the iterations of the solver until the expected precision is reached. This extra pre-processing step may add an excessive computational burden and lead to under-utilization of parallelism due to group processing.

This work presents an alternative to avoid the race condition without resorting on the mesh-coloring strategy. In order to do so, the global stiffness matrix is dynamically assembled in the device's main memory using a data structure composed of linked lists that are independently and simultaneously

  ISSN 2179-1074

managed, and then distributed atomic operations are responsible for maintaining data consistency.

II. GPUs FOR SCIENTIFIC COMPUTING

With the emergence of multi-core CPUs (Central Processing Unit) and GPUs (Graphics Processing Units), a way has been opened for a type of computation where many calculations are performed simultaneously. It is based on the principle that the big problems can often be divided into smaller parts and then solved simultaneously [6], [14].

GPUs were designed to act as highly parallel coprocessors, initially for graphics processing. With the GPGPU (General Purpose GPU) strengthening, through the use of programming languages (such as OpenCL, OpenMP and CUDA) and libraries (such as CUSPARSE, CUSP and CUBLAS) that allowed the general use programs creation, computational problems cost and potentially parallelizable of diverse areas of science have been rethought for the device [15], [18].

However, the develop parallel computing programs is more difficult than traditional sequential once the concurrence introduces several new potential software errors classes, of which running conditions are the most common. Communication and synchronization between different threads are typically some of the biggest obstacles to successful parallel program performance.

CUDA is used in this work. More than a programming language, it is an API developed by NVIDIA, which allows accommodating in the same source code execution calls to the host (CPU) and device (GPU), simultaneously. The syntax is the same as in languages C and C++, with the creating code fragments possibility for execution in parallel, called kernels, that perform calculations that would be repeated a large number of times [15]. These calculations, in this work context, are those performed on the mesh triangles and, later, on the matrix or vector positions in the solver step.

A. Atomic Functions

In multi-threaded applications that share data, an inconsistency situation may occur due to two or more read-modify-write operations in the same memory space, simultaneously. This is called race condition and can happen in any multi-threaded program if the programmer is not careful. CUDA, however, offers a set of atomic operations that can be used to prevent erroneous results due to this situation, for both shared memory and global memory [16].

Atomic operations in shared memory are generally used to prevent race conditions between different threads within the same thread block, while atomic operations in global memory are used to avoid race conditions between two different threads, regardless of the thread block in which they are. However, in both memory spaces, the feature should be used very carefully as it can generate a parallel processing bottleneck, more evident in global memory than shared memory due to their performance differences.

For example, `atomicAdd()` reads a word at some address in global or shared memory, adds a number to it, and writes the result to the same address. The operation is atomic in the sense that its execution is guaranteed without interference from other threads. In other words, no other thread can

  ISSN 2179-1074

access this address until the add operation completes.

Atomic operations are indispensable in many applications but may make certain approaches unfeasible due to their expensive computational cost [8], [15]. As a general rule, a call to atomic operations should be avoided or delayed as much as possible. Within the scope of the problem addressed in this work, the most common approach is to include a preprocessing step to colorize mesh elements and subdivide computation so that each color does not generate race condition among its elements.

Although it may seem outdated to use race condition again in the solution proposed in this paper, it is important to consider that there has been an important update on GPU architectures with compute capability (c.c.) 6.x or later. From this version, it is possible to operate 64-bit words, and procedures such as `atomicAdd_system()`, which ensures that the instruction is atomic in relation to other CPUs and GPUs in the system, and `atomicAdd_block()`, which ensures that the operation is atomic only with threads in the same thread block, were made available. It is interesting to reassess the strategy using atomic operations on devices with c.c. 6.x or later, mainly for use in solutions where the cost of mesh color pre-processing is prohibitive.

III. FEM OVERVIEW

The FEM is a well-known numerical technique for solving Boundary Value Problems (BVP) by discretizing the domain into a set of subdomains, the so-called finite elements, on which the solution is approximated. In this work, different implementation strategies of the FEM are explored for solving a two-dimensional electrostatic problem: a capacitor composed of two parallel square plates with 2-inch length and 1/64-inch thickness. The plates are positioned 1/32-inch apart and a 48V potential is applied [20]. The electric field distribution is calculated both between the plates and on the space around, as shown in Fig. 1. According to [5], the application of the FEM can be divided into four basic steps, which are briefly described below for the sake of completeness.

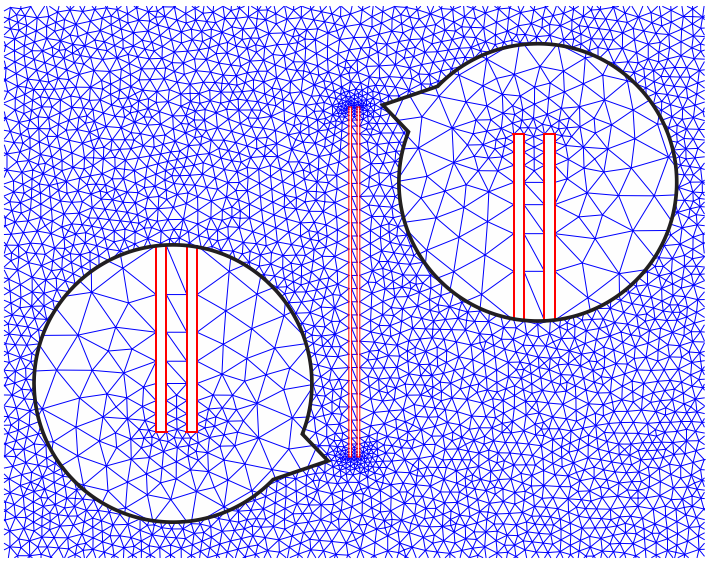

Fig. 1. FEM mesh of the parallel plates capacitor with refinement in the capacitor border.

ISSN 2179-1074

a) **Domain discretization:** the domain discretization consists in obtaining a finite element mesh from the continuous domain Ω. In this step, the shape, quantity, and size of the elements are established. Fig. 1 shows a mesh for the parallel plate capacitor problem, used in this paper as the model problem. In this work, the triangular mesh is generated by the Triangle [17] mesh generator.

b) **Choice of interpolation functions:** also called basis or shape functions, they approximate the BVP solution within each mesh element. First and second degree polynomials are usually chosen because of both the ease of mathematical manipulation and the good approximation obtained.

c) **Formulation of the system of equations:** by applying the weak form of the BVP equation (approximated by the interpolation functions) on each mesh element, a set of systems of linear equations is obtained, as shown in Fig. 2 for first-degree polynomial approximation. After the local linear system of each element is obtained, the global matrix assembling can be performed. This step consists of mapping the coefficients of each element matrix to a single global matrix, which tends to be sparse and with a size proportional to the number of nodes. Fig. 3 shows the mapping process of the matrices presented in Fig. 2, and Fig. 4 shows the data structure used to store this matrix, which is a sparse matrix, in the device memory.

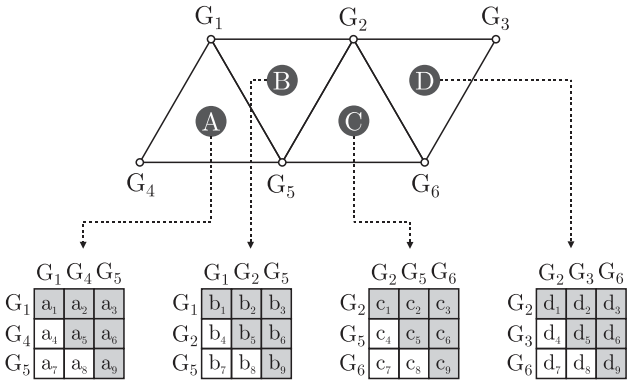

Fig. 2. Discretization domain example and element processing result that composes the mesh.

	G1	G2	G3	G4	G5	G6
G1	a_1+b_1	b_2		a_2	a_3+b_3	
G2	b_4	$b_5+c_1+d_1$	d_2		b_6+c_2	c_3+d_3
G3		d_4	d_5			d_6
G4	a_4			a_5	a_6	
G5	a_7+b_7	b_8+c_4		a_8	$a_9+b_9+c_5$	c_6
G6		c_7+d_7	d_8		c_8	c_9+d_9

Fig. 3. Global stiffness matrix after assembly resulting.

ISSN 2179-1074

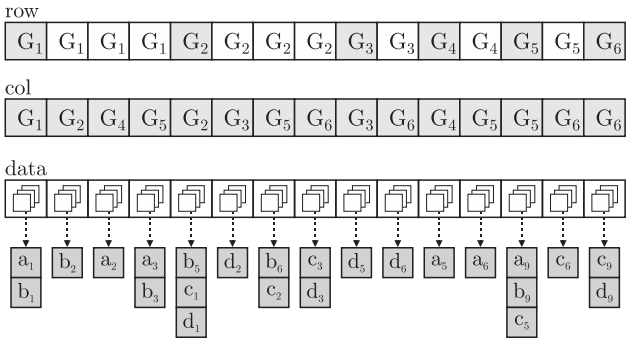

Fig. 4. The adapted Coordinate Format (A-FEM) data structure.

d) **Solution of the system of equations:** once the global system is obtained, the boundary conditions are applied. For the model problem used for illustration in this work, only Dirichlet boundary conditions, that is, the potential in each plate, are considered. The resulting linear system can then be solved by a direct or iterative numerical method. In this work, the Conjugate Gradient (CG) method is adopted since the system is symmetrical and positive definite.

A. EbE-FEM

The EbE-FEM solution consists of solving a finite element system without assembling the global matrix. In this approach, the linear system solution operations are decomposed and applied directly to the element matrices instead of the global matrix. The original objectives of this strategy were mainly to reduce the amount of stored data and arithmetic operations as well as to take advantage of the vector computers parallelism when associated with mesh coloration (grouping of the elements in the sense that adjacent elements receive different colors) [14]. With the popularization of GPGPU, the most current representatives of Single Instruction Multiple Data (SIMD) processors, there has been a resurgence of interest in EbE solvers to explore the parallelism available in these FEM solution architectures [6].

To avoid the race condition, the EbE-FEM combined with the mesh coloring acts on some parallelizable operations (matrix-vector) of the conjugate gradient method. Its implementation consists of anticipating the equation system solution using the element matrices directly for the assignment of the boundary conditions and system solution, without the need to assemble the global coefficient matrix.

Element-level boundary conditions can be assigned to the b^e vectors as an arithmetic boundary value average by the number of elements that share the node [6]. For example, if a node shared by two elements has a boundary-value, each b^e vector receives the contour value equal to half of the original contour value. In the [9], a similar approach based on the weighted average of the initial boundary value by the element matrix coefficients is used.

B. Mesh coloring

The mesh coloring allows the operations of the conjugate gradient method to be parallelized [6]. As

ISSN 2179-1074

the mutual exclusion process, coloration is a strategy adopted to avoid race conditions, which configure a data inconsistent state caused by non-atomic transactions.

For EbE-FEM parallelization, the coloring must occur in such a way that elements that share the same node have distinct colors, which generate an additional step in the basic FEM execution. The coloring process to computational operations parallelizing must be done in such a way that, in addition to using a minimum amount of colors, there must also be a balance between the number of elements in the different colors, so that all threads perform roughly the same amount of operations. In this way, the computational resources distribution is equalized and the execution time is reduced.

IV. FEM ON GPU WITHOUT COLORING

The EbE-FEM is widely used in works that propose to use GPU for significant gains in computational performance [6], [14]. However, solutions based on this technique have a common implementation characteristic that presents itself as a bottleneck: the need to colorize the mesh to avoid the race condition. This is because:

- Requires an additional preprocessing step with significant computational cost;
- Inserts a limit on the number of threads that can be processed simultaneously, which is usually less than the device's threshold (CUDA cores).

With respect to the pre-processing step, it is a set of tasks to be performed for the problem representation through a mesh. The processing time for the most relevant pre-processing steps is shown in Fig. 5. Among these steps, the mesh generation and the identification of the adjacencies, are necessary in almost all approaches. However, the coloring step, necessary for solutions that use parallel computing, can be avoided by adopting another strategy to avoid the race condition.

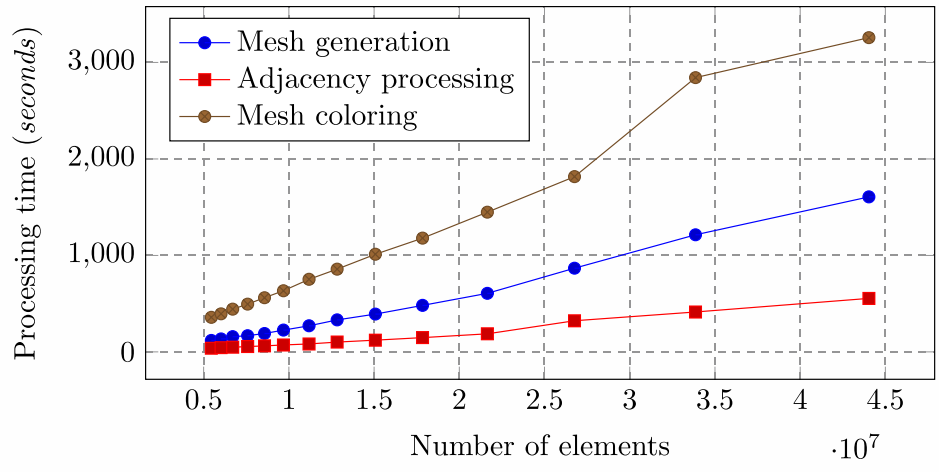

Fig. 5. Pre-processing steps computing time.

The elimination of this step contributes to the global computational cost reduction, as well as the removal of the bottleneck during the process of thread synchronization, originally executed in batches whose elements are grouped by colors. On the other hand, another strategy to circumvent race conditions is needed. In this work, the adopted solution for the problem is the creation of linked lists with values that, at a later step, will be consolidated through atomic operations, as parallel as possible.

The availability of double precision atomics for c. c. 6.x or later devices enabled the implementation of the proposed method [15]. Thus, its coding is more similar to the sequential counterpart, requiring only a few changes in the code, especially the inclusion of the strategy of consolidation of the global stiffness matrix.

A. Solution overview

From the weak form of the Poisson equation and applying the Galerkin method, a system of linear equations for each mesh element is generated. These small matrices compose a bigger linear equations system, that can be solved using a solver. The computational realization, however, can be done quite differently to fit the available hardware characteristics, and thus, achieve performance gain or scale abilities.

Fig. 6 presents a macro-task flow comparison to be accomplished using three implementations:

1. The standard/sequential version of FEM (identified only as FEM in this work), typically used by sequential computing solutions;
2. The proposal of this work, discussed in detail below;
3. The EbE-FEM version, typically used in parallel computing solutions [6], [14].

This work proposal, has a sequence of steps quite similar to FEM. This is due to the fact that the coloring step is not required and that the global stiffness matrix is completely assembled. In contrast, in the EbE-FEM, the global stiffness is never assembled, and all operations are performed at the element level.

Even though proposals the FEM and the proposed approach have similar steps, their implementations are different. Starting from the mesh elements processing, standard FEM calculates the elements' contributions sequentially, which guarantees that there is no race condition during the sum of each local contribution into the global matrix. Despite, the current approach uses as many threads as mesh elements.

B. Matrix assembling

With the individual element matrices values computation, one can start the global matrix assembly process. Since it is common for the same position in the global stiffness matrix to receive data from different threads, a strategy to avoid the parallel-imposed race condition needs to be adopted. This solution also needs to consider the matrix sparsity in order to optimize the occupied memory space. The proposed solution, therefore, unlike EbE-FEM, assembles the global stiffness matrix into the device

  ISSN 2179-1074

main memory using an adapted coordinate format structure, a-COO, depicted in Fig. 7: instead of three vectors (column, row, value), it has a vector of size equal to the number of mesh nodes whose position index identifies the global stiffness matrix row (a-COO-row). Each position is the head of a linked list [8], whose elements contain the a-COO-column and a-COO-value information.

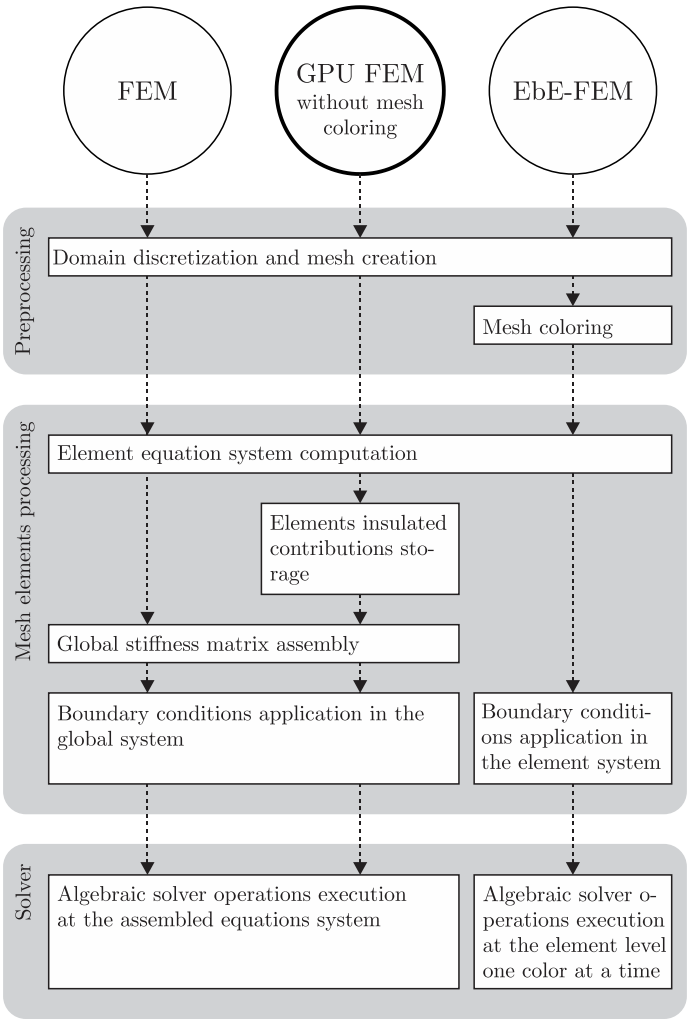

Fig. 6. Steps flowchart comparing the sequential FEM, the GPU-adapted FEM proposed in this work (A-FEM), and the classical EbE-FEM.

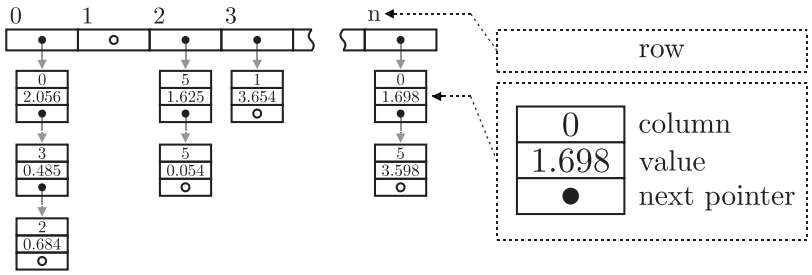

Fig. 7. Adapted Coordinate Format (a-COO): stiffness matrix representation using dynamic GPU memory allocation.

The a-COO is shared by all GPU-threads, each responsible for an element in this assembly step. The values are stored in the proper linked lists without concern for duplicity or ordering.

The next step is the application of the boundary conditions, where each a-COO-row vector position is processed by a GPU-thread, composing the RHS vector and identifying the degrees of freedom (*dof*) that will be eliminated. The linked lists are ordered and consolidated, excluding duplicities and eliminated *dofs*, leaving it ready for the solver step. This step uses the `atomicAdd()` function, available in the CUDA language to avoid inconsistencies [15]. Although typically atomic operations are performance bottlenecks, these, in particular, have a diminished impact by acting on the linked list inside thread scope, that is, on an a-COO-row vector single position.

After recording all the contributions in the consolidated and ordered a-COO structure, the process of converting data to the Compressed Row Format (CSR) used by the solver is started. The a-COO contains information about the size of each a-COO-row linked list position [8], which allows trivial conversion to the compressed format in question. For a brief period, the a-COO structure dynamically created co-exists in the device memory with CSR structure.

C. Solver

The linear equation system, $Ax = b$, resulting from the FEM can be solved iteratively by methods that use the Krylov subspaces to find successive approximate solutions, $A \in R^{n \times n}$ and $x, b \in R^n$ [1], [4-7], [18]. This choice is due to the following properties:

- They take advantage of the efficient memory storage structure of sparse matrices;
- Auto correct if an error is committed, and also for reducing the rounding errors that eventually appear in exact methods;
- Maintain the original structure and matrix sparsity pattern.

During each iteration, an approximate solution increasingly closer is generated, $x_i = x_0 + K_i(r_0, A)$, where i is the iteration counter, x_0 is any initial solution, r_0 is the initial residual given by $r_0 = b - Ax_0$ and $K_i(r_0, A)$ is the n-th Krylov subspace with respect to r_0 and A [7].

The CG method adapted to GPU [1], [10], [12], [18] is used in this work, whose computational cost is smaller and optimized than its EbE-FEM counterpart when the data is processed in the assembled matrix. The solver steps can be decomposed into variables initialization, inner products, vector updates and matrix-vector multiplications, operations with properties that can be exploited under parallel computing environments.

The basic solver's algebraic operations have an optimized implementation for use with CUDA by means of the libraries such as cuBLAS and cuSPARSE, the latter specialized in sparse matrices operations. During the solution process only the trivial operations of addition, subtraction, multiplication, and division, are performed simultaneously in all positions of the input data structures.

Brazilian Microwave and Optoelectronics Society-SBMO *received 20 Aug 2019; for review 22 Aug 2019; accepted 14 Apr 2020*
Brazilian Society of Electromagnetism-SBMag *ISSN 2179-1074*

V. EXPERIMENTAL RESULTS

In this work, two important implementations of the GPU EbE-FEM are selected so that they are analyzed together with the proposed solution. All tested implementations address the same parallel plate capacitor problem (Fig. 1), discretized using a 2D mesh generated in a pre-processing step. This simplified application was chosen for didactic purposes; however, the proposed algorithm can be applied in a straightforward manner to more complex and three-dimensional problems. The implementations are identified as:

- **EbE-v1**, the traditional EbE-FEM technique [2];
- **EbE-v2**, a modified EbE-FEM formulation with increased intra-node parallelism [6];
- **A-FEM**, the proposed solution.

To validate the results, all three solutions generated by GPU (A-FEM, EbE-v1, and EbE-v2) were compared with a benchmark solution generated by a standard FEM on a CPU, and the error L2-norm stayed below 10^{-5} for all cases. Moreover, the L2-norm of the difference between the results generated by the GPU was close to machine precision. This is because the operations are basically the same, and A-FEM basically postpones the consolidation of the global stiffness matrix.

It is worth noting that the division of the tasks in Fig. 8 is not very clear for both EbE-FEM (v1 and v2) methods. It comes from the fact that some operations are shared in the element processing and solution processes, which makes difficult the application of readily available preconditioners.

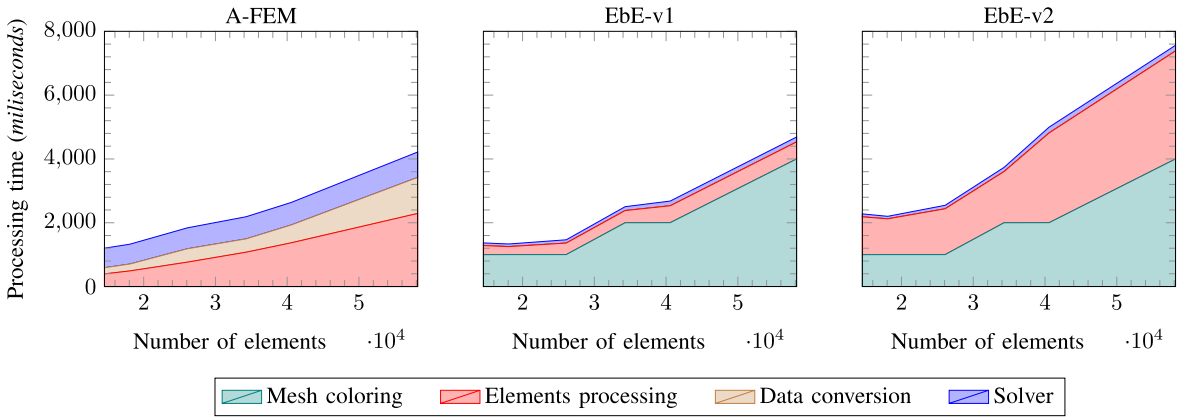

Fig. 8. Analysis of accumulated computing time.

The data conversion step only exists in the proposed solution, A-FEM, where the dynamic data structure a-COO is converted to the vector CSR format. Although the processing time (excluding the pre-processing steps) of the A-FEM appears to be worst, it still presents linear growth and is very close to the EbE-v2, which is a reference in the GPU FEM approach. However, if mesh coloring is considered in the total processing time, the proposed approach becomes more appealing (see Fig. 9).

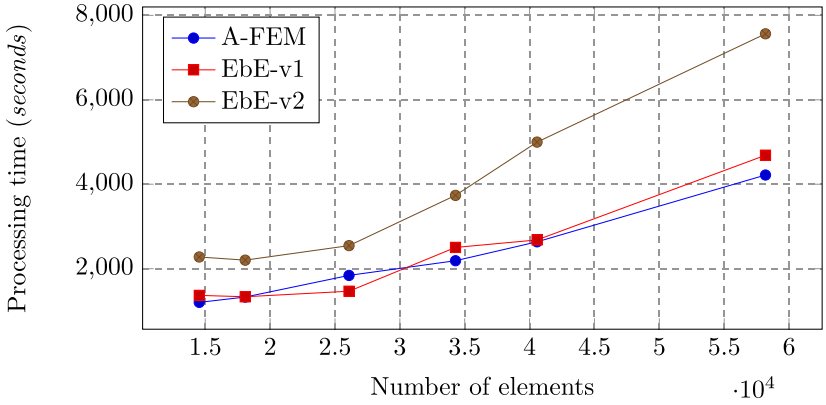

Fig. 9. The total computation time of the steps listed in Fig. 8.

Another critical investigation concerns the memory usage of the device. In all methods, the memory footprint is linearly proportional to the number of triangles in the mesh (see Fig. 10). This behavior is expected for first-order nodal finite elements since the triangles are the elements on which contributions are calculated in the stiffness matrix. However, each method has its average memory demand per element, with EbE-v2 being 79 bytes, EbE-v1 being 151 bytes, and A-FEM being 192 bytes.

In EbE-v2, since the local element arrays for each color are stored and recalculated at each iteration, it is expected that less memory is necessary. On the other hand, some redundant computation is required. In EbE-v1, all contributions of the computed elements are persistent in the device's memory, despite the processed color in the current iteration.

In the A-FEM, atomic sums are avoided by inserting all individual contributions in linked lists for each line in the global matrix. Thus, the consolidation of values sharing the same position is postponed by inserting the value-column tuple in the linked list. When the contribution calculation step is completed, consolidation finally happens in a distributed way, with a GPU thread running through each line of the matrix, and the extra memory is freed. The use of these linked lists explains a larger memory footprint in comparison to EbE-FEM alternatives.

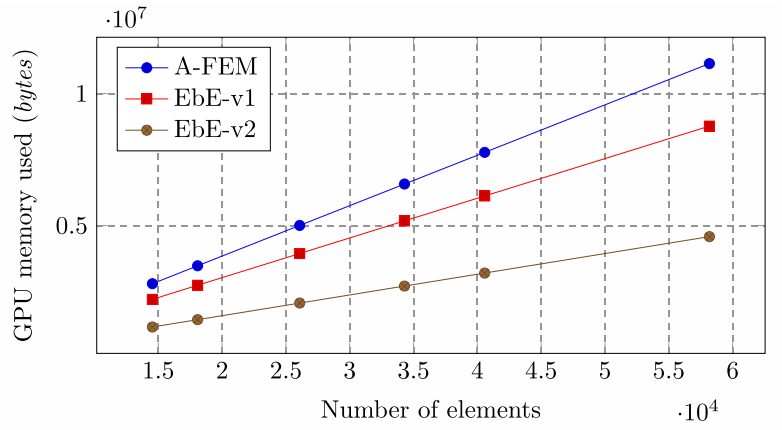

Fig. 10. GPU memory used.

In order to use the proposed algorithm with high-order finite-elements, it would only be necessary to modify the implementation of the local assembly phase; the consolidation strategy remains the same. It is worth noting that the local assembly phase of the finite element method can reach up to 50% of the overall computation run-time with high-order elements [19]. Therefore, the proposed strategy and the EbE-v1 are more attractive than the EbE-v2 since the local element matrices are not recalculated at each iteration.

VI. CONCLUSION

A FEM parallel implementation, without the need for mesh coloring or treating race condition, is presented. This solution is characterized by the simultaneous execution of all elements at the expense of additional memory usage. However, as previously mentioned, the proposed technique is more similar to the FEM than EbE-FEM, and therefore fewer changes need to be made in its coding. Although the global stiffness matrix assembly and solver iterations are performed without color division, distributed atomic operations are necessary, and the computational cost can be high.

As stated before, the main drawback of the proposed solution is that it requires more memory than the EbE-v1 solution; in the numerical experiments, approximately 27% of the memory spent is related to the contributions accumulation process, although the memory is immediately released after the consolidation.

On the other hand, the computational cost of the A-FEM measured by the processing time is slightly lower than the EbE-v1, if pre-processing steps are to be considered. In practice, the computational cost of the atomic operations of A-FEM is similar to the additional mesh coloring stage that needs to be carried out in the EbE-FEM. That is, a significant contribution of the presented work is the parallel algorithm simplification due to the avoidance of the coloring stage and the higher similarity to the standard FEM.

It is important to note that there has been a significant update on GPU architectures with compute capability (c.c.) 6.x or later. Since version 6.x, there is support to atomic addition operating on 64-bit floating-point numbers in global and shared memories, which allowed the development of this work.

REFERENCES

[1] J. Bolz, I. Farmer, E. Grinspun, and P. Schröoder, "Sparse matrix solvers on the GPU: Conjugate gradients and multigrid," *ACM Trans. Graph*., vol. 22, no. 3, pp. 917–924, July 2003.
[2] C. Cecka, A. Lew, and E. Darve, "Introduction to assembly of finite element methods on graphics processors," *IOP Conference Series: Materials Science and Engineering*, vol.10, no. 1, 012009, 2010.
[3] A. Dziekonski, P. Sypek, A. Lamecki, and M. Mrozowski, "Generation of large finite-element matrices on multiple graphics processors," *Int. J. Numer. Meth. Engng*, page n/a, December 2012.
[4] G. H. Golub, V. Loan, and F. Charles, *Matrix computations*, Johns Hopkins University Press, 3rd edition, October 1996.
[5] J. Jin. *The Finite Element Method in Electromagnetics*, John Wiley & sons, 2002.
[6] I. Kiss, S. Gyimothy, Z. Badics, and J. Pavo, "Parallel realization of the Element-by-Element FEM technique by CUDA," *Magnetics, IEEE Transactions on*, vol. 48, no.2, pp. 507–510, February 2012.

Brazilian Microwave and Optoelectronics Society-SBMO *received 20 Aug 2019; for review 22 Aug 2019; accepted 14 Apr 2020*
Brazilian Society of Electromagnetism-SBMag *ISSN 2179-1074*

[7] Y. Saad. *Iterative Methods for Sparse Linear Systems, Second Edition*, Society for Industrial and Applied Mathematics, 2 edition, April 2003.
[8] T. Varga and S. Szénási, "Design and implementation of parallel list data structure using graphics accelerators," *In 2016 IEEE 20th Jubilee International Conference on Intelligent Engineering Systems (INES)*, pp. 315–318, June 2016.
[9] Z. Lin, Y. Chen, X. Zhao, D. Garcia-Donoro, and Y. Zhang, "Parallel Higher-Order Method of Moments with Efficient Out-of-GPU Memory Schemes for Solving Electromagnetic Problems," *ACES JOURNAL*, vol. 32, no.9, pp. 781–788, 2017.
[10] J. Bolz, I. Farmer, E. Grinspun, and P. Schröoder, "Sparse Matrix Solvers on the GPU: Conjugate Gradients and Multigrid," *ACM Trans. Graph*., 2003.
[11] C. Cecka, A. Lew, and E. Darve, "Introduction to assembly of finite element methods on graphics processors*," IOP Conference Series: Materials Science and Engineerin*g, 2010.
[12] N. K. Pikle, S. R. Sathe, and A. Y. Vyavhare, *PGPU-based parallel computing applied in the FEM using the conjugate gradient algorithm: a review*, Sadhana - Academy Proceedings in Engineering Sciences, 2018.
[13] T. Yamaguchi, K. Fujita, T. Ichimura, T. Hori, M. Hori, and L. Wijerathne, "Fast Finite Element Analysis Method Using Multiple GPUs for Crustal Deformation and its Application to Stochastic Inversion Analysis with Geometry Uncertainty," *Procedia Computer Science*, 2017.
[14] Y. Xiuke et al., "Research on preconditioned conjugate gradient method based on EBE-FEM and the application in electromagnetic field analysis," *IEEE Transactions on Magnetics*, vol. 53, no. 6, pp. 1-4, 2017.
[15] J. Sanders, E. Kandrot, *CUDA by example: an introduction to general-purpose GPU programming*, Addison-Wesley Professional, 2010.
[16] NVIDIA Corporation. *CUDA C Programming Guide*. vol. 10.1.243, 2019.
[17] J. R. Shewchuk, *Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator*. *In: Lin M.C., Manocha D. (eds) Applied Computational Geometry Towards Geometric Engineering*. WACG 1996. Lecture Notes in Computer Science, vol 1148. Springer, Berlin, Heidelberg.
[18] L. P. Amorim, R. C. Mesquita, T. D. S. Goveia and B. C. Correa, "Node-to-Node Realization of Meshless Local Petrov Galerkin (MLPG) Fully in GPU," in *IEEE Access*, vol. 7, pp. 151539-151557, 2019. doi: 10.1109/ACCESS.2019.2948134.
[19] R. C. Kirby, M. Knepley, A. Logg, L. R. Scott, "Optimizing the evaluation of finite element matrices," *SIAM Journal on Scientific Computing*, vol. 27, no. 3, pp. 741-758, 2005.
[20] R. L. Boylestad, Electronic devices and circuit theory. Pearson Education India, 2009.

received 20 Aug 2019; for review 22 Aug 2019; accepted 14 Apr 2020

ISSN 2179-1074