Alternate Parallel Processing Approach for FEM

David M. Fernández, Maryam Mehri Dehnavi, Warren J. Gross, and Dennis Giannacopoulos

Department of Electrical and Computer Engineering, McGill University, Montreal, QC H3A 2A7, Canada

In this work we present a new alternate way to formulate the finite element method (FEM) for parallel processing based on the solution of single mesh elements called FEM-SES. The key idea is to decouple the solution of a single element from that of the whole mesh, thus exposing parallelism at the element level. Individual element solutions are then superimposed node-wise using a weighted sum over concurrent nodes. A classic 2-D electrostatic problem is used to validate the proposed method obtaining accurate results. Results show that the number of iterations of the proposed FEM-SES method scale sublinearly with the number of unknowns. Two generations of CUDA enabled NVIDIA GPUs were used to implement the FEM-SES method and the execution times were compared to the classic FEM showing important performance benefits.

Index Terms—**Acceleration, finite element method, graphic processing units (GPU), multicore, parallel processing.**

I. Introduction

SOLVING increasingly complex electromagnetic (EM) problems using modern computing resources inevitably requires employing parallel programming paradigms in response to the current trend of advances in microprocessor architecture. The advent of the multicore/manycore processors brings about an important turning point in programming practices; in particular, for EM practitioners and the scientific community this translates to rewriting legacy libraries and applications with new parallel formulations that can efficiently realize the performance benefits offered by these modern computing resources as shown recently in [1], [2]. This work focuses on the finite element method (FEM), a popular numerical simulation technique, and proposes an alternate way for solving the linear systems derived that is well suited for parallel manycore (graphic processing unit-GPU) implementations. The remaining sections describe the new proposed method, sources of parallelism, its advantages and current limitations and the GPU implementations details. Finally the results and conclusions are presented.

II. New FEM Single Element Solution (FEM-SES) Method

The classic FEM formulation can be thought of as a seven step process [3] as shown by the block arrows in Fig. 1: 1) discretization of the domain; 2) definition of boundary conditions (BC); 3) construction of the element stiffness matrices; 4) assembly of the global coefficient matrix imposing BCs; 5) solution of the algebraic system; 6) post-processing of results; and if a required 7) enhancing the solution using mesh refinement and/or changing the basis functions (restarting the whole process). Traditionally, the solution of FEM has been parallelized in three ways: a) partitioning and solving in parallel the derived algebraic system [1], [2], [4]; b) employing domain decomposition techniques [4]–[7]; and c) using multigrid techniques [6], [7]. However, a greater amount of parallelism is sought to take advantage of the aforementioned manycore trend.

Therefore, we propose to decouple the element solution from that of the whole mesh by directly computing on the element stiffness matrices concurrently, going from step three to five in Fig. 1 (shown by the curved arrow). In this new scenario, boundary conditions are applied directly to the element matrices as described later in Subsection II.A. Such disconnected solutions are then averaged node-wise using a weighted sum over all concurrent nodes in an iterative fashion until convergence is achieved. Furthermore, this approach does not require building a global coefficient matrix skipping step four in Fig. 1 and modifying steps three and five encircled in the dashed line. A similar approach is presented in [8] where the solution is computed by nodes as opposed to the element approach proposed here. The mathematical formulation for the proposed decoupled single element solution (called FEM-SES) approach is presented next.

A. Mathematical Formulation

Equations (1)–(3) present the classic FEM variational formulation for a static EM boundary value problem that will be used for simplicity, without loss of generality. Here $F(\varphi)$ represents the functional to minimize, φ the unknown potentials and p the boundary conditions (BC) applied

$$\delta F(\varphi) = 0 \tag{1}$$

$$\varphi = p, \text{ on the boundary } \Gamma \tag{2}$$

$$F(\varphi) = \frac{1}{2}\iint_{\Omega}\left[\alpha_x\left(\frac{\partial\varphi}{\partial x}\right)^2 + \alpha_y\left(\frac{\partial\varphi}{\partial y}\right)^2\right]d\Omega. \tag{3}$$

The functional can then be applied to each element in the discretized domain as shown in (4)–(5), where the superscript e refers to the element index

$$F(\varphi) = \sum_{e=1}^{n} F^e(\varphi^e) \tag{4}$$

$$F^e(\varphi^e) = \frac{1}{2}\iint_{\Omega_e}\left[\alpha_x\left(\frac{\partial\varphi^e}{\partial x}\right)^2 + \alpha_y\left(\frac{\partial\varphi^e}{\partial y}\right)^2\right]d\Omega. \tag{5}$$

Next the local functionals are minimized and BCs are enforced element-wise independently, see (6). This is where the

Manuscript received July 05, 2011; revised October 03, 2011; accepted October 14, 2011. Date of current version January 25, 2012. Corresponding author: D. Fernández (e-mail: david.fernandezbecerra@mail.mcgill.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMAG.2011.2173304

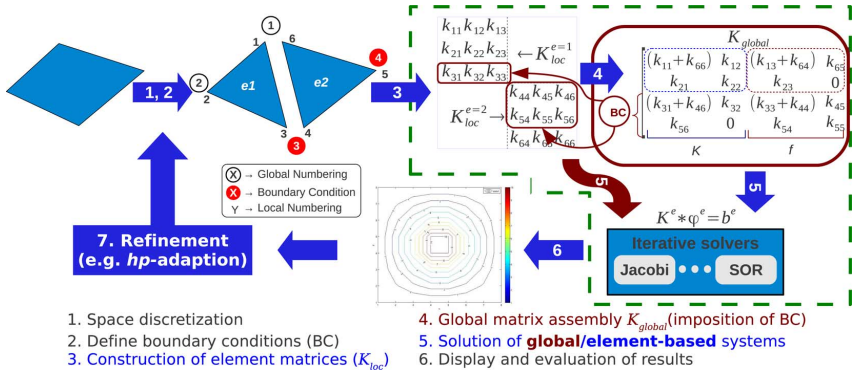

Fig. 1. Steps in the classic finite element method (FEM) and the proposed changes for the FEM-SES method enclosed within the dashed line.

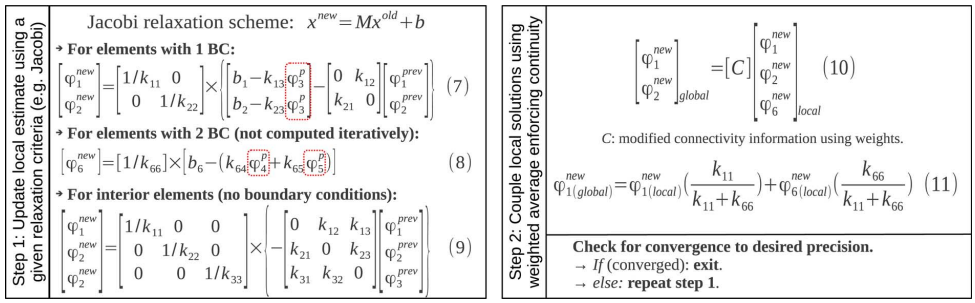

Fig. 2. *2-step iterative relaxation* method: Step-1 shows update rationale for elements with or without BCs and Step-2 shows a weighted average example.

new method departs from the classic FEM

$$\left\{\frac{\partial F^e}{\partial \varphi^e}\right\}_{\text{BC\_reduced}} = \{K^e\}\{\varphi^e\} - \{b^e\} = \{\oslash\}. \quad (6)$$

To obtain the global solution from (6) for each of the elements a *2-step iterative relaxation* approach is proposed. As presented in Fig. 2, the first step updates the local element solutions independently using the current estimate of the unknowns and the reduced local matrices in a relaxation-type of technique (e.g., step 1 in Fig. 2 shows a Jacobi update scheme). The reduced element matrices are obtained from the original element matrices subject to boundary conditions in a matrix modification process. The second step sums the local solutions from overlapping element nodes using a weighted average to compute the global node solutions. The weights (or *weight factors*) are computed using the main diagonal values of the element matrices for overlapping nodes. Fig. 2 shows how the weights are calculated and used to computed the unknown value for global node 1 ($\varphi_{1\ \text{global}}$) based on the corresponding local solutions ($\varphi_{1\ \text{local}}$ and $\varphi_{1\ \text{local}}$). Finally, a convergence check is performed to either exit or repeat the process.

B. Sources of Parallelism, Advantages and Disadvantage

Sources of parallelism identified in the new approach are:

- element stiffness matrices can be built in parallel and preserved in distributed CPU/cores to be computed later;
- elements solutions may be computed in parallel independently of any other element;
- the weighted average can be performed in parallel across nodes taking into account the element connectivity.

Two drawbacks of the proposed *2-step iterative relaxation* method can be identified: a) it will have slow convergence similar to that of Jacobi iterative method, but a great deal of parallelism is obtained in exchange; thus, by exploiting parallelism in each iteration, the new approach reduces the total execution time of the finite element method as demonstrated in the results section; and b) computing the global solution requires a single synchronization per iteration. Among other advantages, the proposed FEM-SES method does not require special numbering (local and global numberings) thus less housekeeping time and effort, no global coefficient matrix is built, uses the same information as the classic FEM, and good scaling is expected considering that the element connectivity is almost constant as the mesh is further refined to better represent the geometry and resolve the solution of the problem.

III. Parallelization of FEM-SES

In this section, techniques to parallelize the FEM-SES are presented. A sequential implementation of the new method is profiled first to determine the dominant computing kernels. The two most important operations in the algorithm are the actual assembly of the element matrices and the *2-step iterative relaxation* method itself, whereas all other operations are considered as pre and post-processing steps. For the largest datasets tested the profiling showed that the *2-step iterative relaxation* method dominates all other operations (assembly, pre and post-processing). Consequently we concentrate on parallelizing the *2-step iterative relaxation*, which correspond to the last two sources of parallelism identified in Section II.B.

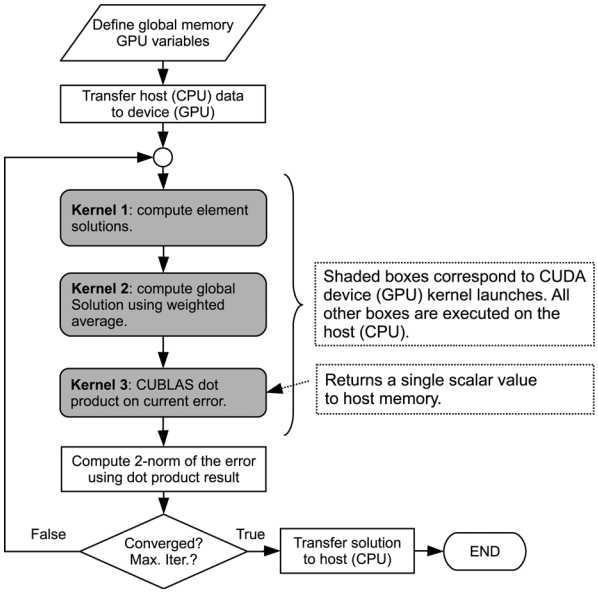

Fig. 3. Workflow to parallelize the *2-step iterative relaxation* method on NVIDIA GPUs.

```
1.__global__ void FEM_coalesed_solver_kernel(int
        num_elements, float *K_dis, float *b_dis, int*
        fem_nodes, float *w, float *phi, float *d_odata) {
2.   // Local memory.
3.   float temp1;
4.   int data_disp, int data_disp3;
5.   // Shared memory.
6.   __shared__ float temp2[256*3];
7.   // Use block and thread IDs to select the element.
8.  data_disp = blockDim.x*blockIdx.x + threadIdx.x;
9.  if (data_disp < num_elements) {
10.     data_disp3 = data_disp * 6;
11.     data_disp = data_disp * 3;
12.     // Load x to shared memory (partially coalesced).
13.     temp2[threadIdx.x*3] = phi[fem_nodes[data_disp]];
14.     temp2[threadIdx.x*3+1] = phi[fem_nodes[data_disp+1]];
15.     temp2[thready*3+2] = phi[fem_nodes[data_disp+2]];
16.     // Coalesced access to all variables.
17.     // Node 1
18.     temp1 = b_dis[data_disp];
19.     temp1 -= K_dis[data_disp3]*temp2[threadIdx.x*3+1];
20.     temp1 -= K_dis[data_disp3+1]*temp2[threadIdx.x*3+2];
21.     d_odata[data_disp] = temp1 * w[data_disp];
22.     // Node 2
23.     temp1 = b_dis[data_disp+1];
24.     temp1 -= K_dis[data_disp3+2]*temp2[threadIdx.x*3];
25.     temp1 -= K_dis[data_disp3+3]*temp2[threadIdx.x*3+2];
26.     d_odata[data_disp+1] = temp1 * w[data_disp+1];
27.     // Node 3
28.     temp1 = b_dis[data_disp+2];
29.     temp1 -= K_dis[data_disp3+4]*temp2[threadIdx.x*3];
30.     temp1 -= K_dis[data_disp3+5]*temp2[threadIdx.x*3+1];
31.     d_odata[data_disp+2] = temp1 * w[data_disp+2];
32.}}
```

Fig. 4. CUDA function used to implement Kernel 1 that computes the single element solutions in FEM-SES.

A. GPU Implementation

To implement the *2-step iterative relaxation* algorithm on the GPU, we used the CUDA 3.2 SDK [9]. The host (CPU-side) function first defines the GPU global memory required to store the element matrices K_e (including their right hand sides b_e), the global unknown vector φ, and the pre-computed weight factors w_e[1]. These data sets are then transferred to global GPU memory. Next, the host function loops over the three device (GPU) kernels (shown as shaded boxes) that parallelize different sections of the *2-step iterative relaxation* until the convergence criteria is satisfied (Fig. 3). It was necessary to create this logical partition in the method due to the limitations in the synchronization mechanisms available in current NVIDIA GPUs. Once convergence is achieved the host function exits the loop and transfers the global solution φ back to host memory. Large data transfers only occur outside the loop minimizing the effects of global memory access latencies. Only single scalar values are transferred inside the loop in Kernel 3.

Kernel 1 computes the solutions of each element in parallel as described in step 1 of Fig. 2. Each thread in the kernel computes the solution of one element and stores it in device global memory. Each block in this kernel consists of 256 threads and the number of blocks in the grid is computed dynamically at runtime depending on the problem size, which equals the number of elements (in the FEM mesh) divided by the block size. Most of the memory accesses are coalesced due to sequential addressing of the K_e, b_e and w_e data sets but non-uniform and indirect access will still be required for the unknown φ-vector. The indirect accesses are one of the main performance limiting factors in this kernel. To minimize the effects of accessing the φ-vector (which is used several times in a thread), it is stored into shared memory. Techniques such as loop unrolling and variable reuse are also used to enhance performance. Fig. 4 presents the code for Kernel 1, which is the dominating computing kernel of the three in FEM-SES.

After the local solutions are obtained, Kernel 2 is called to compute the global node solutions using an average sum per node. Once again, the host function launches a one dimensional grid with 256 threads per block similar to Kernel 1, where each thread gathers the results for one node. The same kernel computes the error between the new approximation of the unknown and the previous one, and stores it in device global memory.

The third kernel partially computes the *2-norm* of the error using the `cublasSdot` function from NVIDIA the CUBLAS [10] library which returns the dot product of this vector to the host function. The host function then computes the square root to obtain the final value of the *2-norm*, and finally convergence is assessed. The GPU kernels are designed to avoid using synchronizations primitives to minimize execution bottlenecks.

IV. Results

A 2-D electrostatic coaxial cable problem (see Fig. 5) was implemented to validate the new method and study its convergence behavior. Tests were conducted on a 2.4 GHz Intel Core2 Quad processor, with 4 GB of DDR2 global memory and running a 64-bit Linux operating system. Two NVIDIA GPUs are used; a 8800GT clocked at 1.5 GHz with 512 MB of global memory and 112 scalar processors (grouped into 14 streaming multiprocessors-SMs), and a GTX480 clocked at 1.4 GHz with 1.5 GB of global memory and 480 scalar processors (grouped into 32 SMs). The C programming language was used to develop the code for both CPU and GPU (GPU version uses CUDA intrinsics). Compilation was done with GCC 4.1.2 and the CUDA SDK 3.2. All computations are carried out in single precision considering that the GPUs used have limited double precision support. Both CPU and GPU codes were compiled with $-O3$ optimizations flag. Additionally, the $-\mathrm{arch} = \mathrm{sm\_20}$ flag was

[1]The subindex e is used to refer to the matrix/vector of an element, otherwise global values are assumed.

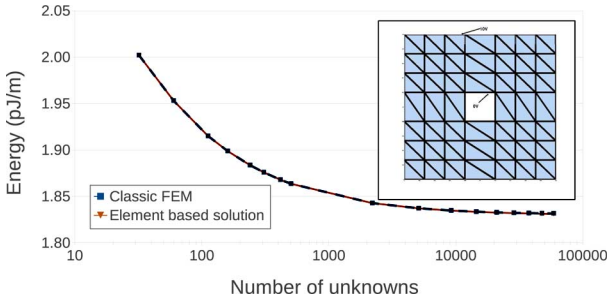

Fig. 5. Energy comparison for classic FEM and proposed element-based formulation. Insert shows a 2-D model of the square coaxial problem.

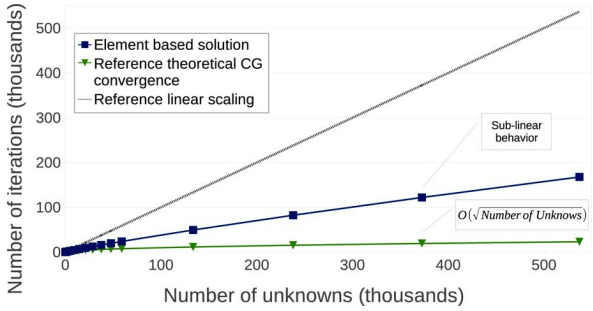

Fig. 6. Iterations scaling with the increase of problem size.

used for the GTX480 to enable FERMI advanced architecture features.

First, sequential implementations for both traditional FEM and the proposed FEM-SES method were done on the CPU. Fig. 5 compares the FEM energy results with those of the new FEM-SES method demonstrating good agreement of the results for different number of unknowns; thus validating the new method. Next, the original mesh was refined to empirically study the convergence scaling of the proposed method. The solid line with out markers in Fig. 6 represents a reference linear scaling (1:1 slope) and the line with square markers shows the iteration count results. These results empirically prove a sublinear iteration scaling of the proposed FEM-SES method as the number of unknowns increases, which is a desirable scaling property of iterative methods. Moreover, this sublinear scaling is obtained even though the condition number increases considerably (from 90 to 9672) as the number of unknowns grows. The convergence rate for the conjugate gradient (CG) method is also shown as a reference for comparison.

Finally, timing results for the reference implementation of the classic FEM (with an efficient CG iterative solver) using hand coded optimizations and up to 4 CPU threads (as explained in [1]) were compared to the best GPU times, which resulted in speedups of up to 14 times and 111 times for the 8800GT (first generation CUDA enabled NVIDIA GPU) and the GTX480 (third generation CUDA enabled NVIDIA GPU-Fermi), respectively, compared to CPU times. The considerable better performance obtained from the GTX480 is due to better support for memory coalescing access, four times more computing resources (scalar processors, local memory and register files) than the 8800GT, and support for L1 cache.

V. Concluding Remarks and Future Work

This work presents two major contributions, a new element-based technique for solving the finite element method (called FEM-SES) well suited for parallel processing, and a methodology for implementing the proposed FEM-SES method to exploit the parallel computing power of modern graphic processors. The goal of designing such a technique is to expose more parallelism in the finite element method compared to traditional approaches. The method was then implemented in two different generations of NVIDIA GPUs obtaining up to 14 times speedup on the 8800GT GPU and 111 times speedup on the GTX480 compared to optimized CPU results for the classic FEM. Future work currently in progress includes exploring ways to accelerate the convergence rate of the FEM-SES method by using preconditioners, alternate relaxation techniques, or multigrid approaches. Also a multi-GPU implementation is currently under development. Other potential future work includes validating the method for non-static problems, and exploring cluster implementations.

References

[1] D. Fernández, D. Giannacopoulos, and W. J. Gross, "Multicore acceleration of CG algorithms using blocked-pipeline-matching techniques," *IEEE Trans. Magn.*, vol. 46, no. 8, pp. 3057–3060, Aug. 2010.

[2] M. M. Dehnavi, D. M. Fernandez, and D. Giannacopoulos, "Enhancing the performance of conjugate gradient solvers on graphic processing units," *IEEE Trans. Magn.*, vol. 47, no. 5, pp. 1162–1165, May 2011.

[3] J.-M. Jin, *The Finite Element Method in Electromagnetics*, 2nd ed. Hoboken, NJ: Wiley, 2002.

[4] T. Itoh *et al., Finite Element Software for Microwave Engineering*. Hoboken, NJ: Wiley, 1996, pp. 385–400.

[5] A. Toselli and O. Widlund, *Domain Decomposition Methods—Algorithms and Theory*. Berlin, Germany: Springer Series in Computational Mathematics, 2005, vol. 34, p. 450.

[6] A. Takei *et al.*, "Full wave analyses of electromagnetic fields with an iterative domain decomposition method," *IEEE Trans. Magn.*, vol. 46, no. 8, pp. 2860–2863, Aug. 2010.

[7] L. Yuanqing and Y. Jiansheng, "A finite element domain decomposition combined with algebraic multigrid method for large-scale electromagnetic field computation," *IEEE Trans. Magn.*, vol. 42, no. 4, pp. 655–658, Apr. 2006.

[8] J. P. A. Bastos and N. Sadowski, "A new method to solve 3-D magnetodynamic problems without assembling an $\mathrm{Ax} = \mathrm{b}$ system," *IEEE Trans. Magn.*, vol. 46, no. 8, pp. 3365–3368, Aug. 2010.

[9] CUDA Programming Guide for CUDA Toolkit 3.2. e-manual. NVIDIA Corporation, Dec. 2010 [Online]. Available: http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/CUDA_C_Programming_Guide.pdf

[10] Cublas users guide. E-Manual. NVIDIA Corporation, Dec. 2010 [Online]. Available: http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/CUBLAS_Library.pdf