Efficient Acceleration of Sparse MPIE/MoM with Graphics Processing Units

Danilo De Donno, Alessandra Esposito, Giuseppina Monti, and Luciano Tarricone
University of Salento
Via per Monteroni - Ecotekne
73100 Lecce, Italy
{danilo.dedonno; alessandra.esposito; giuseppina.monti; luciano.tarricone}@unisalento.it

Abstract— In this paper, graphics processing unit (GPU) is used to accelerate both the impedance matrix calculation and the linear system solution in a Mixed-Potential Integral-Equation (MPIE) formulation of the method of moments (MoM). We consider a three-step process. First, the MoM matrix is computed on the GPU by using a quasi-one-dimensional approximation of the MPIE formulation. Second, a sparse representation of the matrix is obtained by thresholding. Third, a GPU-enabled bi-conjugate gradient (BiCG) method is used to solve the linear system. We validate the accuracy of our approach by analysing a microstrip branch-line coupler and comparing the results provided by the GPU code with a well-known commercial simulator. A performance analysis proves that the developed algorithm, running on a cheap off-the-shelf GPU, is about 25 times faster than the CPU version.

Keywords—bi-conjugate gradient method, graphics processing unit, impedance matrix, method of moments, NVIDIA CUDA

I. INTRODUCTION

A recent trend in High Performance Computing (HPC) is the interest in leveraging the computing power of graphics processing units (GPUs) for accelerating general purpose scientific problems. Indeed, GPUs are low-cost devices specialized for highly parallelized computations and their power is continuously increasing, also thanks to the impulse coming from videogame industry. Moreover, the publication of high-level libraries such as NVIDIA CUDA [1], which simplify software implementation, is promoting such kind of parallelization. GPUs have already been exploited in several fields of computational electromagnetics (CEM), such as FDTD [2] and FEM [3]. The Method of Moments (MoM) is amenable to benefit from this hardware platform as well. Two fundamental tasks deserve to be accelerated to improve MoM performance: the computation of the impedance matrix and the solution of the linear system. In [4], [5] and [6] induced currents on PEC objects are calculated using GPU-accelerated MoM. For the impedance matrix computation, the Electric-Field Integral-Equation (EFIE) approach is adopted, while for the linear system solution, the conjugate gradient (CG) method applied directly to dense matrix is implemented. Authors of [7] describe a GPU acceleration of the MoM: the EFIE formulation and a direct solver, based on dense LU-decomposition, are considered for the impedance matrix assembly and the linear system solution respectively.

In this work, we handle both MoM matrix computation and linear system solution by taking advantage from NVIDIA CUDA programming facilities — note that [4] and [7] use CUDA, while [5] and [6] exploit Brook [8]. Differently from all the other works, we adopt the Mixed-Potential Integral-Equation (MPIE) formulation rather than EFIE. The appeal of MPIE is definitely high because of the scalar nature of its potentials and the lower order of the involved singularities [9]. Concerning the solution of the linear system, we adopt the bi-conjugate gradient (BiCG) solver applied to sparse matrices. BiCG is a generalization of CG suited to handle complex non-Hermitian matrices, which are often encountered in MoM problems. In order to achieve best results in terms of performance and accuracy, our implementation uses the native double-precision representation supported by CUDA.

Details about the adopted MPIE formulation are given in Section II, while Section III describes the CUDA-enabled implementation of both matrix assembly and sparse BiCG solver. Finally, in Section IV the high-efficiency of our MPIE/MoM code is demonstrated on a very cheap GPU, in the framework of the design of branch-line couplers in microstrip technology. The achieved results show an impressive reduction in computational times if compared to a reference CPU implementation.

II. MPIE/MOM FORMULATION

We consider $N$-port planar circuits with infinite transverse dimensions for both the dielectric and ground plane; the metallization thickness is assumed negligible. We adopt the MPIE formulation, solved by considering closed-form Green’s functions in the spatial domain and by using the MoM. The evaluation of the Green’s functions in closed form is performed as [10]:

\[
G^d_{\text{xx}} = G^d_{\text{xx,0}} + G^d_{\text{xx,sw}} + G^d_{\text{xx,ci}} \\
G^q = G^q_{\text{sw}} + G^q_{\text{ci}}
\]

i.e., as the sum of direct terms and quasi-dynamic images ($G^d_{\text{xx,0}}$, $G^d_{\text{xx,sw}}$, $G^d_{\text{xx,ci}}$, $G^q_{\text{sw}}$, $G^q_{\text{ci}}$); the Galerkin’s MoM is used to discretize the relevant equations, by selecting rooftop functions defined over elementary rectangular domains. This way, a linear system is derived from the MPIE:

How to cite this paper:
where the entries $Z_{ij}$ in the impedance matrix are expressed by a fourfold integral. In the implemented MoM this integration has been reduced to a quasi-one-dimensional (1-D) integral by means of an analytical pre-processing which exploits the circular symmetry of $G^d$ and $G^q$. The $Z$-matrix terms can be written as:

$$Z_{xx} = \int r \left[ W_{1x}(r)G_{1x}^d(r) - \frac{1}{\omega^2} W_{2x}(r)G^q(r) \right] dr$$

$$Z_{xy} = \int r \left[ -\frac{1}{\omega^2} W_{3x}(r)G^q(r) \right] dr$$

$$Z_{yx} = \int r \left[ W_{1y}(r)G_{1y}^d(r) - \frac{1}{\omega^2} W_{2y}(r)G^q(r) \right] dr$$

$$Z_{yy} = \int r \left[ W_{1y}(r)G_{1x}^d(r) - \frac{1}{\omega^2} W_{2y}(r)G^q(r) \right] dr$$

The spatial-domain mixed-potential Green’s functions are calculated according to (1), $r$ is the source-test distance and the $W(r)$ terms are calculated as:

$$W(r) \triangleq \int S(r \cos \xi, r \sin \xi) d\xi$$

where $S$ is a bidimensional convolution (see [10] for details).

III. GPU-BASED MOM

A. Impedance matrix computation

The first step in MoM modeling of planar circuits consists in the impedance matrix filling ($Z$ matrix in (2)). We developed a CUDA-based algorithm devoted to parallelize the time-consuming calculation of the integrals reported in (3). To simplify, such operation may be described by the following serial pseudo code:

for each cell
  for each r
    Update_Z_Value

where the inner loop refers to numerical quasi-1-D integration.

The outer loop disappears in the CUDA-enabled implementation as the calculations over the current cells are operated in parallel by CUDA threads. Figure 1 shows the mapping between circuit domain ($M \times N$ current cells) and CUDA threads. The code snippet of Fig. 2 shows how the computation of the integral for the $Z_{xx}$ sub-matrix was performed on GPU (similar steps are required for the other sub-matrices).

Once the impedance matrix was computed, we noted that, in many cases, only a few elements retain an amount of information enough to solve the system in (2) with adequate accuracy. On the basis of this consideration, the initial dense matrix was reduced to a significantly sparse one by neglecting entries smaller than a certain threshold value. The optimum threshold was obtained by iteratively comparing the solution of the sparse linear system with the dense one and by imposing a desired accuracy (see Section IV). We stored the computed sparse matrix using the general-purpose Compressed Row Storage (CRS) format. CRS uses three one-dimensional arrays to store non-zero elements, column indices and pointers to the first element of each row. CRS is well suited for the GPU, where the amount of available memory is limited and memory accesses should be as regular as possible in order to increase efficiency.

B. Iterative solver

Solution of linear systems is a hot topic in GPU computing and several CUDA-based iterative solvers are publicly available. Among them, we recall Iterative CUDA [13], the Concurrent Number Cruncher (CNC) [14], SpeedIT [15], and CUSP library [16] which implement the CG method for sparse matrices with real coefficients. CUSP is a continuously growing project and support for double precision complex matrices is under development.

BiCG basics: Figure 4 resumes the BiCG algorithm in its general form, where $\langle x, y \rangle = (x^*)^T \cdot y$ ($T$ and asterisk denote respectively the transpose matrix and the complex conjugate). Convergence is obtained when a termination condition of the form:

$$\frac{\|r\|_2}{\|r_0\|_2} \leq \varepsilon$$

(5)

is satisfied. Values of $\varepsilon$ commonly used in the literature range from $10^{-6}$ to $10^{-7}$. 
CUDA implementation: we adopted the Jacobs’ form of the BiCG algorithm, which imposes $\bar{r}_0 = r_0^*$. In this case, since our MoM formulation gives rise to complex symmetric matrices, $\bar{r}_i$ and $\bar{p}_i$ are complex conjugate of $r_i$ and $p_i$ respectively. It follows that only one sparse matrix-vector product (SpMV) per iteration is needed. We took into account this time-saving feature while implementing our BiCG solver.

The main loop controlling the iterations is kept on the CPU, whilst the computations inside are performed on the GPU. Four kernels are in charge of the operations carried out in the BiCG main loop (see Table I, where $N$ is the matrix dimension and $\text{nnz}$ is the number of non-zeros).

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Floating Point Operations (FLOPs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpMV</td>
<td>sparse matrix-vector product</td>
<td>$8\text{nnz}$</td>
</tr>
<tr>
<td>dot product</td>
<td>scalar product of two vectors</td>
<td>$8N$</td>
</tr>
<tr>
<td>e. w. product</td>
<td>element-wise product of two vectors</td>
<td>$6N$</td>
</tr>
<tr>
<td>axpy</td>
<td>$ax + y$ (or scalar, $x$ and $y$ vectors)</td>
<td>$8N$</td>
</tr>
</tbody>
</table>

The “SpMV” kernel implements a modified version of the Bell and Garland algorithm [11], also adopted by CUSP. We improved texture memory accesses and optimized CUDA block size and register usage in order to maximize the occupancy of our GPU multiprocessors. The “dot product” kernel is an adaption and generalization of the well known parallel reduction algorithm proposed by Harris et al. in [17]. Such code is appreciated for its efficiency due to the adoption of advanced optimization strategies. Although the “element-wise product” and “axpy” routines are made available by CUDA BLAS library (cuBlas), we implemented such functions from scratch. In this way, we aggregated multiple calls in the same kernel and reduced the overhead due to communication between host and device.

IV. RESULTS

As reference problem, we considered the design of branch-line couplers in microstrip technology, which are four ports devices widely adopted in microwave and millimetre-wave applications like power dividers and combiners. More specifically, the analyzed layout consists of two branch-line couplers connected by means of a 360° microstrip line and operating in the 2.5–3.5 GHz frequency band (see Fig. 5). A preliminary analysis by thresholding demonstrated how the impedance matrix can be reduced to a significantly sparse one (about 5% of non-zero elements) while maintaining a good accuracy of the final solution (error less than 2%).

We started our analysis by comparing numerical results attained by our CUDA-enabled code with those provided by a well-known commercial full-wave simulator based on MoM, i.e. the ADS-Momentum tool [18]. Our GPU-based MoM was compiled by using CUDA 3.2 toolkit and tested on the NVIDIA GeForce GTX 260 GPGPU, featuring 24 streaming multiprocessors. As Fig. 6 shows, a perfect agreement on the scattering parameters is obtained, thus demonstrating the accuracy of our approach. Such results have been achieved by setting the circuit cell dimension to (0.30x0.30) mm$^2$.

Then, execution times of our GPU-enabled MoM were compared to execution times obtained by an equivalent code running on the CPU. It is a single-threaded circuit simulator based on MPIE/MoM, written in C and compiled by GCC 4.3.4 with the “-O3” optimization option enabled. The CPU
results were obtained with a single-core of an Intel Core2 Quad CPU Q9550 @ 2.83 GHz and all calculations were performed by using double precision variables. We conducted several tests on problems containing up to 10000 basis functions, achieving considerable speed-ups over the sequential CPU implementation. Details about execution times of both impedance matrix assembly and linear system solution by BiCG are resumed in Table II. As convergence criterion of the BiCG solver we adopted (5) with ε set to 10^{-6}. Achieved speed-ups are presented in Fig. 7. They are higher when matrix dimension allows for an optimum exploitation of hardware resources.

<table>
<thead>
<tr>
<th>Cell side [mm^2]</th>
<th>Basis Functions</th>
<th>Fill time [s]</th>
<th>Solver time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>CPU</td>
<td>GPU</td>
</tr>
<tr>
<td>0.45x0.45</td>
<td>350</td>
<td>12.45</td>
<td>1.21</td>
</tr>
<tr>
<td>0.35x0.35</td>
<td>520</td>
<td>20.21</td>
<td>1.27</td>
</tr>
<tr>
<td>0.25x0.25</td>
<td>2000</td>
<td>38.12</td>
<td>2.34</td>
</tr>
<tr>
<td>0.15x0.15</td>
<td>10000</td>
<td>80.46</td>
<td>4.37</td>
</tr>
</tbody>
</table>

V. CONCLUSIONS

In this work, we have presented a GPU-based approach for accelerating the two main time-demanding tasks of the MoM: the impedance matrix computation and the linear system solution. Regarding the former, we considered a quasi-one-dimensional approximation of the MPIE formulation for the analysis of planar microstrip circuits. Regarding the latter, the original dense system is first reduced to an equivalent sparse one by thresholding, and then solved via BiCG method. Our GPU-enabled code, based on NVIDIA CUDA, has been tested in the framework of a branch-line coupler analysis. Obtained results have shown a perfect agreement with those provided by a well-known commercial simulator. In terms of execution times, our parallel implementation has achieved considerable speed-up (about 25) if compared to an equivalent CPU single-threaded code.

REFERENCES