Electrostatic discharge simulation using a GPU-accelerated DGTD solver targeting modern graphics hardware

Matteo Cicuttin[1], Peter Binde[2] and Christophe Geuzaine[1]
[1] University of Liege, Montefiore Institute B28 B-4000 Belgium, matteo.cicuttin@uliege.be
[2] Dr. Binde Ingenieure Design & Engineering

We discuss an implementation on recent GPU hardware of the Time Domain Discontinuous Galerkin method for the Maxwell equations. We assess the performance of the method on the simulation of an electrostatic discharge test.

***Index Terms*—Electrostatic discharge, Discontinuous Galerkin, GPU**

I. Introduction

In the last decade supercomputer designs saw an always increasing adoption of GPU accelerators, and this trend is unlikely to change in the foreseeable future.

Because of their impressive computing power and low cost per Gflop, GPUs immediately attracted the scientific computing community. Their usage in computational electromagnetics (CEM) can be traced back to [1] and [2], targeting FDTD and Discontinuous Galerkin (DG) respectively. Recently, renewed interest in CEM on GPU is found for example in [3] for the Discrete Geometric Approach (DGA) method and in [4], [5] for FDTD.

Early GPUs required careful design of algorithms and data structures in order to avoid significant performance penalities, as studied in [2]. In this work we revisit [2] and we assess the performance of DG methods on contemporary GPUs. Our findings are tested with our own code on the simulation of an electrostatic discharge (ESD) test. We observe that recent technical improvements in GPU hardware allow huge simplifications of the algorithms, while retaining the speedups typical of GPUs.

II. Problem setting

Let $t \in \mathbb{R}, t \geqslant 0$ be the time and $\boldsymbol{x} \in \mathbb{R}^3$ be the position vector. We consider the time-domain Maxwell equations

$$\epsilon \frac{\partial \boldsymbol{e}(\boldsymbol{x},t)}{\partial t} = \nabla \times \boldsymbol{h}(\boldsymbol{x},t) - \sigma \boldsymbol{e}(\boldsymbol{x},t) - \boldsymbol{j}_s(\boldsymbol{x},t), \quad (1)$$

$$\mu \frac{\partial \boldsymbol{h}(\boldsymbol{x},t)}{\partial t} = -\nabla \times \boldsymbol{e}(\boldsymbol{x},t), \quad (2)$$

where $\boldsymbol{e}$ and $\boldsymbol{h}$ are the electric and magnetic field vectors, μ, ϵ and σ are the magnetic permeability, the electric permittivity and the conducibility respectively, and $\boldsymbol{j}_s$ are the volumetric current sources.

Spatial discretization of equations (1) and (2) is obtained using a DG approach [2], [6], [7], whereas for temporal integration a second-order Yee leapfrog scheme [7] is used. Our implementation builds on Gmsh [8] and supports arbitrary polynomial order. Differently from [2] it runs in double precision, supports curved elements and integration is performed using full Gauss quadratures, giving some flexibility in the handling of materials.

The semi-discrete DG formulation yields the equations

$$\boldsymbol{E}^{n+1} = \boldsymbol{E}^n + \Delta t\, \mathcal{L}_E^h(\boldsymbol{H}^{n+1/2}, \boldsymbol{E}^n), \quad (3)$$

$$\boldsymbol{H}^{n+3/2} = \boldsymbol{H}^{n+1/2} + \Delta t\, \mathcal{L}_H^h(\boldsymbol{E}^{n+1}, \boldsymbol{H}^{n+1/2}), \quad (4)$$

where Δt is the timestep size, n is the timestep number and $\mathcal{L}_E^h, \mathcal{L}_H^h$ are the standard DG operators which include element contributions, inter-element numerical fluxes and sources; for space reasons we refer the reader to [7, Sec. V-A.1] for their precise definition, including the discussion of sources and boundary conditions.

III. Implementation

The DG discretization of (1) and (2) results, because of integration by parts, in a sum of volumetric contributions and interface contributions known as numerical fluxes [7]. In turn, an application of either $\mathcal{L}_E^h$ or $\mathcal{L}_H^h$ results in two separate compute paths, as detailed in Figure 1.

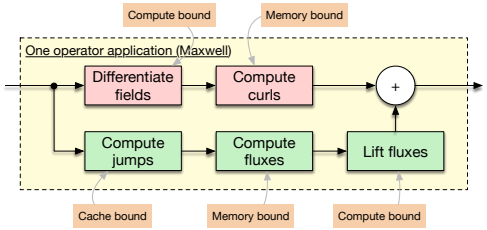

Fig. 1. Computations involved in one application of the DG operators $\mathcal{L}_E^h$ or $\mathcal{L}_H^h$. Red boxes (upper path) represent volumetric operations, whereas green boxes (lower path) represent boundary operations.

The volumetric path, marked in red, is entirely element-local and consists in a series of matrix-vector multiplications to compute the field spatial derivatives, followed by a vector subtraction that yields the field

curls. Locality and arithmetic intensity on this path are ideal and full GPU utilization is easily achieved. Most of the optimizations introduced in [2] are not necessary on recent hardware (CUDA compute capability $\geqslant 3.5$).

The flux path, marked in green, begins with the computation of inter-element jumps: this operation is entirely non-local and has low arithmetic intensity and, at least in principle, could be very problematic on GPU. For this reason in [2] this part has been the subject of a careful analisys, which led to a clever element-reordering algorithm. On recent hardware this non-locality is not problematic anymore, especially at high polynomial orders. The rest of the flux path exhibits mostly uniform memory access and good arithmetic intensity.

The results of the volumetric and flux paths are finally added, and time integration is performed. This phase is mostly memory-bound and limited by the memory bandwidth available on the GPU.

Volumetric field sources are applied by adding the appropriate terms on the volumetric path, whereas interface sources (e.g. plane-wave condition) are applied by modifying the flux terms computed in the second step of the flux path [7]. In both cases the sources for timestep $n + 1$ are evaluated asynchronously on the CPU during timestep n and uploaded to the GPU with an asynchronous copy. This upload requires careful implementation; the details will be given in the full paper.

A goal of our implementation is to be compatible with both NVidia and AMD hardware. As we do not consider the AMD toolchain sufficiently mature yet, we chose to do our implementation in native CUDA and rely on the Hipify tool to generate AMD code on the fly at compile-time. OpenACC and Kokkos were also considered, but they do not provide sufficient control over the GPU hardware. The code resulting from our approach has very good performance on AMD hardware (Table I).

IV. Numerical results

Our solver was used to simulate an ESD test on the device depicted in Figure 2 using the techniques introduced in [9]. Our hardware includes NVidia K20X (2012) and V100 GPUs (2017), AMD MI100 GPU (2020), Xeon Gold 6126 (2017) and Ryzen 5 3600X (2019) CPUs. Detailed results will be given in the full paper, here we report the performance of the solver in DoFs/s in Table I. The NVidia profiler allowed to determine that the computations depicted in Figure 1 fully exploit the hardware capabilities. In addition, we observed that on our code the V100 GPU is much more energy efficient than the MI100: at order 3 the V100 power consumption is of about 140W, whereas the MI100's is of about 260W.

TABLE I
Performance in DoFs/s of the solver. Speedup is between fastest CPU (Ryzen) and fastest GPU (MI100).

Order	Xeon	Ryzen	K20X	V100	MI100	Speedup
1	2.32e7	4.65e7	3.43e8	1.24e9	1.50e9	32.3x
2	1.41e7	3.07e7	4.01e8	1.44e9	1.83e9	59.6x
3	1.05e7	3.55e7	3.80e8	1.38e9	1.84e9	51.8x

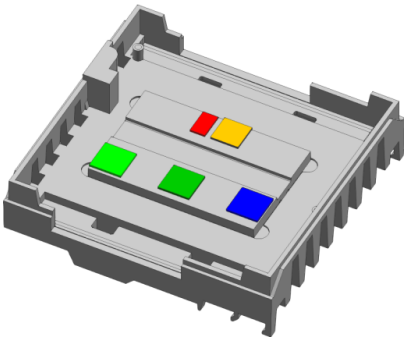

Fig. 2. We simulate an electrostatic discharge applied to the bottom-left corner of a case/heatsink (gray) containing a PCB (not shown) on which some integrated circuits are present (colored boxes).

V. Conclusion

Our solver is already capable of handling efficiently real world problems, however our goal is to make it scalable on huge problems. Future development is focused in investigating multi-GPU and distributed memory parallelism capabilities.

References

[1] Takada, N., Shimobaba, T., Masuda, N., Ito, T., "High-speed FDTD simulation algorithm for GPU with compute unified device architecture". *2009 IEEE Antennas and Propagation Society International Symposium*

[2] A. Klöckner, T. Warburton, J. Bridge, J. S. Hesthaven, "Nodal discontinuous Galerkin methods on Graphics processors," *J. Comp. Phys. 228, 2009.*

[3] M. Cicuttin, L. Codecasa, B. Kapidani, R. Specogna, F. Trevisan, "GPU accelerated time domain DGA method for wave propagation problems on tetrahedral grids,". *IEEE Trans. Magn., Volume 54, Issue 3, March 2018*

[4] C. Warren, A. Giannopoulos, A. Gray, I. Giannakis, A. Patterson, L. Wetter, A. Hamrah, "A CUDA-based GPU engine for gprMax: Open source FDTD electromagnetic simulation software", *Computer Physics Communications Vol. 237, April 2019*

[5] R. Calatayud, E. Navarro-Modesto, E. A. Navarro-Camba, N. T. Sangary, "Nvidia CUDA parallel processing of large FDTD meshes in a desktop computer: FDTD - matlab on GPU", *EATIS '20: Proceedings of the 10th Euro-American Conference on Telematics and Information Systems, Nov. 2020*

[6] J. S. Hesthaven, T. Warburton, "Nodal Discontinuous Galerkin Methods", Springer-Verlag New York, 2008

[7] L. Diaz Angulo, J. Alvarez, M. Fernández Pantoja, S. Gonzales Garcia, "Discontinuous Galerkin Time Domain Methods in Computational Electrodynamics: State of the Art", *Forum for Electromagnetic Research Methods and Application Technologies (FERMAT).* Aug. 2015

[8] C. Geuzaine, J.-F. Remacle. "Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities." *Int. J. Numer. Methods Eng. 79(11), pp. 1309-1331, 2009*

[9] M. Angeli, E. Cardelli, "Numerical modeling of electromagnetic fields generated by electrostatic discharges", *IEEE Trans. Magn., Volume 33, Issue 2, March 1997*