A Parallel Implementation of the Correction Function Method for Poisson's Equation With Immersed Surface Charges

David S. Abraham and Dennis D. Giannacopoulos

Department of Electrical and Computer Engineering, McGill University, Montréal, QC H3A 0E9, Canada

In this paper, a novel graphics-processing unit (GPU) implementation of the recently proposed correction function method (CFM) is presented, for the finite-difference solution of Poisson problems with surface-charge distributions. The CFM is a robust and versatile method most notable for its immersed treatment of interface problems of any geometry, to an arbitrary order of accuracy. Given the well-known interface jump conditions associated with the electric scalar potential, the CFM is here shown to be immediately applicable to the computation of electrostatic fields, in the presence of curved surface-charge distributions. Moreover, an in-depth analysis of the CFM algorithm is presented, in which performance bottlenecks are investigated and significant potential for parallelizability is identified. The resulting parallel CFM algorithm is then implemented using NVIDIA's compute unified device architecture GPU language, yielding a significant increase in performance.

Index Terms— **Electrostatics, immersed interface, parallel processing, Poisson.**

I. Introduction

POISSON problems with interface jump conditions are of paramount importance in the study of a wide range of physical phenomena, including electromagnetics. For example, the electrostatic scalar potential ϕ (and/or its derivative) is generally discontinuous across an interface formed by two materials of differing permittivities, or in the presence of infinitesimal source terms, for example, surface-charge distributions. The resulting interface jumps cause significant issues for standard finite-difference-based solvers, particularly in the case of curved interfaces or charge distributions, in which staircasing errors are also of major concern.

As a result, many methods have been devised to address some of these problems. The immersed interface method [1], for example, handles jump discontinuities by altering the finite-difference stencils themselves near the interface. The ghost fluid method [2], on the other hand, adds additional "ghost" nodes near the interface, which serve as smooth extensions of the solution, compensating for jumps. In electrostatics, as well as time-domain electromagnetics, curved dielectric interfaces are often handled via conformal techniques, in which local coordinate transformations [3], or an "effective permittivity" [4], [5], are employed. Unfortunately, these methods are usually limited in terms of order of accuracy, and often place restrictions on the smoothness of the interface.

In contrast, the recently proposed correction function method (CFM) [6] allows for the treatment of Poisson problems with interface jumps to arbitrarily high accuracy, with any interface geometry, free of staircasing errors. The CFM embeds the interface in a regular Cartesian grid, generalizing the notion of discrete ghost nodes to a continuous function. The resulting correction terms can be shown to be equivalent to an additional source term, meaning the CFM only requires a modification of the resulting linear system's right-hand side (RHS), allowing for the use of any standard existing Poisson solver.

In this paper, the well-known interface jump conditions for the electrostatic scalar potential are shown to be immediately compatible with the CFM formulation. Moreover, given the emphasis often placed on efficiency and performance, the main goal of this paper is to analyze the CFM algorithm for bottlenecks and to determine its amenability to parallelization. The resulting algorithm is then implemented in NVIDIA's compute unified device architecture (CUDA) graphics-processing unit (GPU) language, and performance improvements are investigated as a function of problem size.

This paper will focus on the original CFM formulation, in which the domain is assumed to be homogeneous (i.e., no material discontinuities) and rectangular. Discontinuities in ϕ are therefore assumed to be solely due to the action of known surface charge or dipole distributions. While this may seem restrictive at first, a generalization of the CFM capable of handling material discontinuities, arbitrarily shaped domains, and Dirichlet boundaries (on which the associated source distributions are not known) is available and fundamentally based upon the same algorithm [7]. As a result, any parallelization strategies and analysis developed in this paper are immediately applicable to these more complex cases.

II. Correction Function Method

A brief review of the CFM algorithm is now presented. The reader is encouraged to refer to [6] for more details. The CFM is designed to solve interface Poisson problems of the following kind, over a domain Ω, with closed interface Γ:

$$\nabla^2\phi(\vec{x}) = f(\vec{x}) \text{ in } \Omega \tag{1}$$

$$\phi(\vec{x}) = g(\vec{x}) \text{ on } \partial\Omega \tag{2}$$

$$[\phi] = \alpha(\vec{x}) \text{ on } \Gamma \tag{3}$$

$$\left[\frac{\partial\phi}{\partial n}\right] = \beta(\vec{x}) \text{ on } \Gamma \tag{4}$$

in which square brackets denote a jump, that is, $[\phi] = \phi_2 - \phi_1$.

Manuscript received November 20, 2016; revised January 6, 2017; accepted January 18, 2017. Date of publication January 26, 2017; date of current version May 26, 2017. Corresponding author: D. D. Giannacopoulos (e-mail: dennis.giannacopoulos@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.2017.2659702

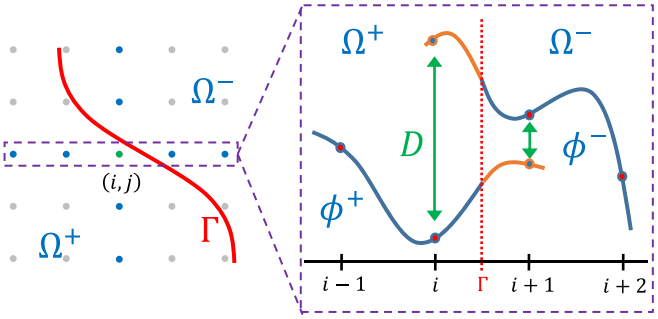

Fig. 1. Smooth extensions of the discontinuous solution across Γ.

In general, finite-difference stencils fail wherever they straddle the interface, given the discontinuous nature of the solution. The CFM circumvents this issue by smoothly extending the solution a short distance across the interface, as depicted in Fig. 1. By defining a correction function (CF), denoted by D, as the difference between the extended solutions on either side of Γ ($D = \phi^+ - \phi^-$), the finite-difference stencils may be augmented as follows, to compensate for the jump:

$$\frac{\partial^2\phi}{\partial x^2} \approx \frac{\phi^+_{i-1,j} - 2\phi^+_{i,j} + \overbrace{\phi^-_{i+1,j} + D_{i+1,j}}^{\phi^+_{i+1,j}}}{\Delta x^2} = f_{i,j} \tag{5}$$

in which the subscripts denote the discrete node number. This may equivalently be expressed as an additional source term

$$\frac{\partial^2\phi}{\partial x^2} \approx \frac{\phi^+_{i-1,j} - 2\phi^+_{i,j} + \phi^-_{i+1,j}}{\Delta x^2} = f_{i,j} - \frac{D_{i+1,j}}{\Delta x^2}. \tag{6}$$

Furthermore, it may be shown the required CFs satisfy the following partial differential equation:

$$\nabla^2 D(\vec{x}) = f_d(\vec{x}) \text{ in } \Omega_{i,j} \tag{7}$$

$$D = \alpha(\vec{x}) \text{ on } \Gamma_{i,j} \tag{8}$$

$$\frac{\partial D}{\partial n} = \beta(\vec{x}) \text{ on } \Gamma_{i,j}. \tag{9}$$

Since the CF is only required near the interface, where the solution is discontinuous, the interface can be covered, or tiled, in a series of square patches. The above defining equation for D can then be solved in a least-squares sense, via a functional minimization, in each patch

$$D_\Omega = \sum_{i=1}^{n} w_i H_i(x, y) \tag{10}$$

$$\min_{w_i} \left\{ l_c^3 \int_{\Omega_{i,j}} [\nabla^2 D_\Omega - f_d(\vec{x})]^2 d\Omega_{i,j} + c_1 \int_{\Gamma_{i,j}} [D_\Omega - \alpha(\vec{x})]^2 d\Gamma + c_2 l_c^2 \int_{\Gamma_{i,j}} \left[\frac{\partial D_\Omega}{\partial n} - \beta(\vec{x}) \right]^2 d\Gamma \right\} \tag{11}$$

where H_i are Hermite polynomials, $l_c \propto \Delta x$ ensures each term scales similarly as the grid is refined, and c_1 and c_2 are penalty coefficients weighting each term equally. The result is a small local matrix equation per patch, $A_\Omega w = b_\Omega$. The overall algorithm is depicted in Fig. 2.

From the above discussion, the method can be seen to be immediately applicable to electrostatics, in which the interface

1. Cover or tile the interface in a series of patches.
2. Solve the local problem $A_\Omega w = b_\Omega$ in each patch for the weights associated with D.
3. Generate the equivalent source terms from each D.
4. Solve the global matrix problem for ϕ.

Fig. 2. Overall steps associated with the CFM algorithm.

jump conditions associated with the $\vec{E}$ and ϕ fields are well-known. Specifically, the appropriate jump conditions for a charged interface in a homogeneous domain are

$$[\phi] = \alpha(\vec{x}_\Gamma) = d(\vec{x}_\Gamma) \tag{12}$$

$$\left[\frac{\partial\phi}{\partial n}\right] = \beta(\vec{x}_\Gamma) = -\frac{\rho_s(\vec{x}_\Gamma)}{\epsilon} \tag{13}$$

in which $d(\vec{x}_\Gamma)$ is a surface dipole density, $\rho_s(\vec{x}_\Gamma)$ is a surface-charge density, and ϵ is the permittivity of the medium [8].

III. Parallelization Strategy

The procedure represented in Fig. 2 offers many avenues for parallelization. Most prominent among them, however, is the solution of all the local matrix equations within each patch (Step 2). Since the number of patches for any given problem could be immense, the resulting number of matrix solving operations required has the potential to be a major bottleneck in computation time (a postulate verified in the Results section). Correspondingly, in this paper, attention focuses only on a parallel GPU implementation of this step of the algorithm, with the remainder on the CPU.

The goal, then, is to parallelize the solution of the CF within each patch. Upon inspection, it is noted that the defining equations for D constitute a local Cauchy problem, in which both the value and normal derivative of the function are prescribed along that piece of the interface contained within each patch. As a result, within each patch, D is uniquely defined by the local interface conditions, $\alpha(\vec{x}_\Gamma)$ and $\beta(\vec{x}_\Gamma)$. Thus, while there may be only a single unique solution for D within the entire domain, each local piece of the solution can be completely independently determined. The calculation of the CFs within each patch therefore constitutes an embarrassingly parallel problem, which is immediately adaptable to the GPU architecture.

The use of a GPU is not without its own potential overhead. For example, in general, the CPU and GPU do not share the same physical memory. As a result, any collaboration between the two will require that data transfers take place. The proposed resulting hybrid CPU–GPU algorithm is detailed in Figs. 3 and 4.

IV. Implementation

The parallelized version of the above algorithm was implemented in NVIDIA's CUDA GPU programming language. The fundamental principle of CUDA lies in the grouping of threads into entities called "warps," with each thread within a warp executing in lock-step with each other, but with the ability to work off of different data sets [9]. This single

(1) Tile the interface in a series of patches on the CPU
(2) Transfer the patch data from the CPU to the GPU
(3) Solve $A_\Omega w = b_\Omega$ in each patch in parallel on the GPU
(4) Transfer all w_i for each patch from GPU to CPU
(5) Compute the equivalent source terms on the CPU
(6) Solve the global matrix system, $A\phi = b$, on the CPU

Fig. 3. Overall steps associated with the hybrid CPU–GPU CFM algorithm.

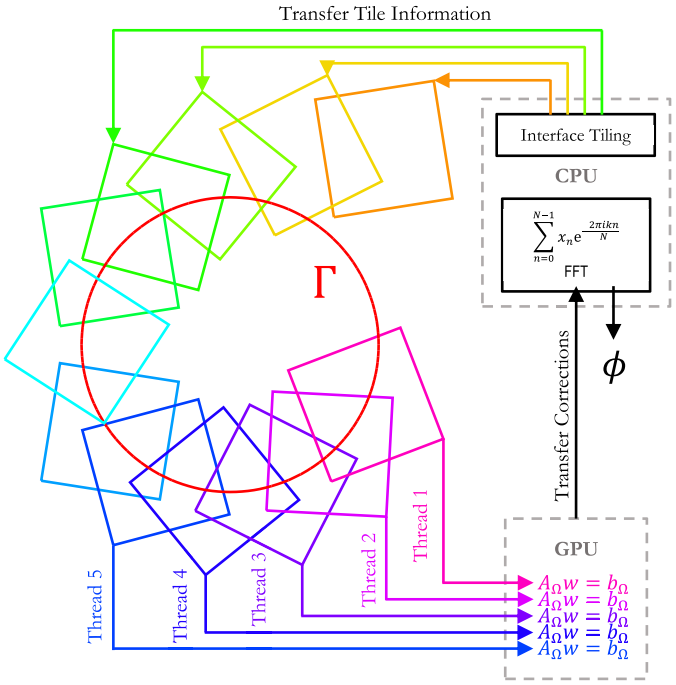

Fig. 4. Pictorial representation of the hybrid CPU–GPU CFM algorithm.

instruction multiple thread (SIMT) system thereby allows for maximum throughput when many identical operations are to be performed, and is ideal for the problem presented above. However, to obtain the best possible results, it is important to consider details of the GPU hardware architecture when writing the main GPU code, or kernel.

For example, should too many resources be required by any one warp, the number of concurrently executing threads will be restricted [10]. The proportion of GPU resources actively being used is known as occupancy, and can be estimated by a number of tools [11]. For the present algorithm, solving for D was divided into multiple kernel calls to reduce register pressure (increasing occupancy) and to take advantage of streams, in which multiple kernels may overlap their execution [9].

Memory-access patterns can also play a pivotal role in GPU performance. Given their SIMT architecture, NVIDIA GPUs are highly optimized to fetch data from main memory in a coalesced fashion, meaning in a single contiguous block [9]. As a result, all data structures used in the present implementation were given particular care to avoid slow noncontiguous memory accesses.

Lastly, as for the implementation of the CPU portions of the algorithm, the only significant optimization was made to the solution of the global matrix system, $A\phi = b$.

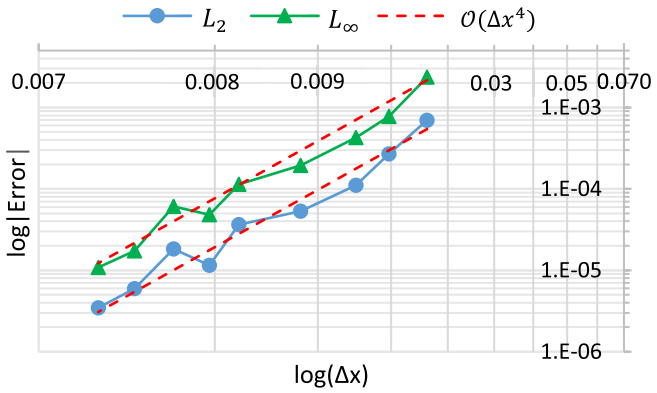

Fig. 5. Error convergence in the L_2 and L_∞ norms.

Since the domain was assumed rectangular, the use of an efficient fast Fourier transform-based Poisson solver was possible [12]. As mentioned earlier, arbitrarily shaped domains can be embedded into larger rectangular ones, with minimal overhead.

V. Results

A test problem was selected to which the above analysis could be applied. The CFM algorithm has the same steps regardless of the complexity of the problem being solved, with only the tiling and number of patches varying between different geometries and interface configurations. The resulting breakdown of computation time and speed improvements would vary slightly depending on these factors; however, the general overall trends and results should remain consistent. For these reasons, a simple 2-D cylindrical coaxial cable was selected for analysis, embedded into a rectangular grounded box, bounding the region $[0, 1] \times [0, 1]$. The cable was centered at $(0.5, 0.5)$, with inner and outer radii of 0.1 and 0.4 m, respectively. The inner and outer conductors had surface charge densities of $\rho_{s1} = 1$ C/m^2 and $\rho_{s2} = -0.25$ C/m^2, respectively.

To determine any speed-up afforded by the use of the CPU–GPU algorithm, control calculations were performed entirely on the CPU. The CPU used in either case was an Intel i7 5820K processor, clocked at 3.3 GHz. The GPU was an NVIDIA GTX960 GPU, clocked at 1.127 GHz, with 1024 CUDA cores. In both cases, computations were performed in single precision.

The GPU algorithm was first validated against the exact solution to ensure accuracy and order of convergence. Fig. 5 demonstrates the error's convergence in the L_2 and L_∞ norms, showing an expected fourth-order trend. In addition, to verify the earlier postulate regarding the bulk of CPU time being spent solving for the CF in each patch, a breakdown of algorithm execution was measured as a function of problem size and is presented in Fig. 6.

From these data, it is clear that the CF computation represents a significant burden in the total solution process, especially for smaller problems. The downward trend in the CF data suggests this process' execution time scales less quickly than that of the remaining steps, yet even for four million variables, it remains important in terms of work.

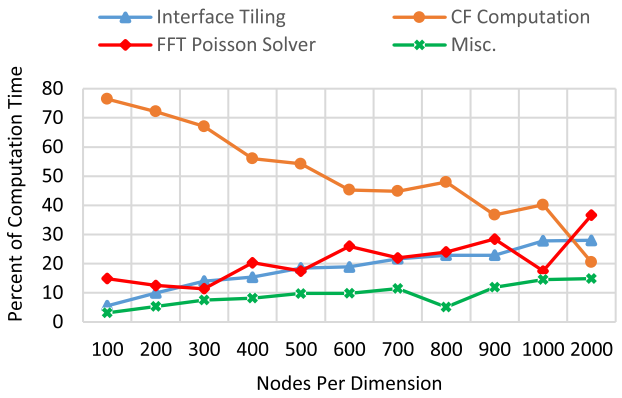

Fig. 6. Breakdown of computation time as a function of problem size.

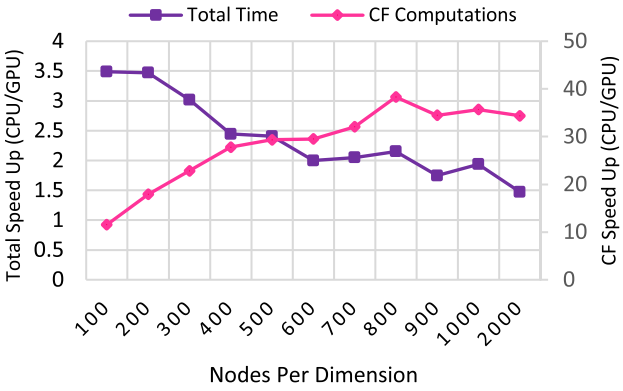

Fig. 7. GPU algorithm performance as a function of problem size.

The metric used to evaluate the performance of the parallelized version is a ratio of CPU-only execution time to the hybrid CPU–GPU execution time. This ratio is plotted as a function of problem size for both the total algorithm time and for the CF computations alone, in Fig. 7. In either case, the data presented include the overhead associated with data transfers to and from the CPU/GPU, and exclude the time required to initialize the CUDA API, since this is a one-time fixed cost incurred by the very first CUDA subroutine call only. As the problem size grows, so too does the acceleration afforded by the GPU. As larger problems result in more work being assigned the GPU, this leads to better use of GPU resources and latency hiding [10], resulting in the GPU performing up to 38 times faster than the CPU-only implementation. Since only the CF computations have been parallelized, the larger the fraction of the total computation time this part occupies, the more pronounced the overall improvement will be. This is highlighted by a peak overall improvement of about 3.5 times occurring for the smallest problem size tested.

VI. Conclusion

In conclusion, a novel GPU implementation of the Correction Function Method for the Poisson equation has been presented. The CFM is a highly versatile and accurate method in which Poisson problems with interface jump conditions are solved to high accuracy, with arbitrary interface geometry, free of staircasing errors. The method is immediately applicable to electrostatics, given the well-known interface conditions associated with the electric scalar potential. Moreover, since the Correction Function itself is defined by a local Cauchy problem, the determination of D within each patch is an embarrassingly parallel problem. By offloading this work to a GPU, the CF calculations were shown to be executed up to 38 times faster, resulting in a total algorithm boost of up to 3.5 times, compared with a traditional serial computation. Though this paper dealt solely with problems without material discontinuities, as well as rectangular domains, it nonetheless has far-reaching applications. As mentioned previously, versions of the CFM that can handle inhomogeneous and irregularly shaped domains have been proposed and are fundamentally based off the same techniques employed here. In addition, a time-domain formulation of the CFM has also recently been proposed, requiring the solution of the CF within each patch, at each time step [13]. Given that the same principle of patch-independence should still apply in the time domain, this paper has the ability to greatly improve the performance of transient CFM simulations. Lastly, while the implementation proposed in this paper used a hybrid CPU–GPU model, in principle, the remaining major steps are also parallelizable, meaning a future implementation should be capable of being entirely run on the GPU, resulting in even better performance.

References

[1] R. J. Leveque and Z. Li, "The immersed interface method for elliptic equations with discontinuous coefficients and singular sources," *SIAM J. Numer. Anal.*, vol. 31, no. 4, pp. 1019–1044, Aug. 1994.

[2] R. P. Fedkiw, T. Aslam, B. Merriman, and S. Osher, "A non-oscillatory Eulerian approach to interfaces in multimaterial flows (the ghost fluid method)," *J. Comput. Phys.*, vol. 152, no. 2, pp. 457–492, Jul. 1999.

[3] Y.-C. Chiang, Y.-P. Chiou, and H.-C. Chang, "Finite-difference frequency-domain analysis of 2-D photonic crystals with curved dielectric interfaces," *J. Lightw. Technol.*, vol. 26, no. 8, pp. 971–976, Apr. 15, 2008.

[4] K.-P. Hwang and A. C. Cangellaris, "Effective permittivities for second-order accurate FDTD equations at dielectric interfaces," *IEEE Microw. Wireless Compon. Lett.*, vol. 11, no. 4, pp. 158–160, Apr. 2001.

[5] M. Fujii, D. Lukashevich, I. Sakagami, and P. Russer, "Convergence of FDTD and wavelet-collocation modeling of curved dielectric interface with the effective dielectric constants obtained by static field analysis," in *Proc. 33rd Eur. Microw. Conf.*, Oct. 2003, pp. 459–462, doi: 10.1109/EUMA.2003.340989.

[6] A. N. Marques, J.-C. Nave, and R. R. Rosales, "A correction function method for poisson problems with interface jump conditions," *J. Comput. Phys.*, vol. 230, no. 20, pp. 7567–7597, Aug. 2011.

[7] A. N. Marques, J.-C. Nave, and R. R. Rosales. May 2016. High order solution of Poisson problems with piecewise constant coefficients and interface jumps. [Online]. Available: https://arxiv.org/abs/1401.8084

[8] J. D. Jackson, *Classical Electrodynamics*, 3rd ed. Hoboken, NJ, USA: Wiley, 1998.

[9] NVIDIA Corp., "CUDA C programming guide," Santa Clara, CA, USA, Tech. Rep. PG-02829-001_v8.0, Sep. 2016.

[10] NVIDIA Corp., "CUDA C best practices guide," Santa Clara, CA, USA, Tech. Rep. DG-05603-001_v8.0, Sep. 2016.

[11] NVIDIA Corp., (2016). CUDA Occupancy Calculator. [Online]. Available: http://developer.download.nvidia.com/compute/cuda/CUDA_Occupancy_calculator.xls

[12] P. N. Swarztrauber, "The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle," *SIAM Rev.*, vol. 19, no. 3, pp. 490–501, Jul. 1997.

[13] D. S. Abraham, J.-C. Nave, and A. N. Marques. Sep. 2016. "A correction function method for the wave equation with interface jump conditions." [Online]. Available: https://arxiv.org/abs/1609.05379