A Parallel, Finite-Difference Time Domain Maxwell Propagator : A Star-P Implementation

Alejandro W. Rodriguez

In this manuscript, we discuss the development of a parallel Maxwell solver in the time domain. In particular, Maxwell's equations are discretized in a uniform Yee grid using a finite-difference time-domain discretization approach. The resulting discretized equations are then parallelized in *Star-P*, by partitioning the evolution matrices in space. A *data-parallel*, *hybridization* approach is discussed in the context of this problem, and these parallel results are benchmarked for speed and performance against serial versions of the same code. Finally, the method is checked against standard results in the literature.

I. INTRODUCTION

Many scientific collaborations require large numbers of computers to run in parallel. In fact, to a large extent, the future of many of the interdisciplinary topics of current interest rests entirely on the available supercomputing power. In this paper, we implement a two dimensional parallel finite-difference time-domain electromagnetic solver and benchmark it against it's serial counterpart. In particular, we demonstrate some of the advantages of this parallelized solution by applying it to the study of new interesting and computationally expensive systems.

The basis for solving scientific problems numerically involves discretizing a given equation of motion and solving and/or iterating the resulting matrix equation. While this is generally a feasible approach to solving simple one-dimensional problems, it becomes prohibitive once extended to higher dimensions, or when a high degree of precision is critical to the accuracy of the results. This is particularly true of problems in electromangnetism. Previous approaches to predicting electromagnetic phenomena involved a combination of analytical (perturbative) and numerical methods. The idea then was to approximate the exact equations of motion via a suitable perturbative method, and then to iterate through the subsequent (serial) approximations until one reached the desired degree of accuracy. There are, however, whole classes of problems in electromagnetism to which these semi-analytic methods cannot be applied, and it is precisely these kinds of problems that supercomputers are posed to solve.

To date, there are many pre-existing parallel electromagnetic solvers with substantially more flexibility than what has been produced here. Nevertheless, the practice acquired in developing a parallel code is of great learning value to an amateur like me, and will undoubtedly allow me to play a larger, more direct, and independent role in this field. In the future, the skills I have learned in this class will be applied to the study of higher dimensionsional electromagnetism, to problems involving imaginary-time evolution, higher-dimensional quasicrystalline structures and finally, thermal effects. It is expected that the additional degrees of freedom will dramatically increase the computational complexity, opening up the possibility for increased supercomputing hackery.

This paper is organized into three different sections: Section II describes the theoretical and numerical basis of the code, including analyses of the efficiency and convenience of parallelizing the equations, with benchmarks as a function of resolution and number of processors. These are in turn followed by Sec. III with some results and final remarks.

II. THEORY AND NUMERICS

The following subsections develop the theoretical basis underlying our parallel code, and explain the discretization, and parallelization schemes employed. Because the parallelization procedure that one one can choose to apply to this problem is not unique, alternative frameworks are discussed (and when possible, analyzed) as well. Section II A is a brief description of the equations to be discretized, along with the quantities of interest to us. The discretization procedures and complications thereof are subsequently discussed in Sec. II B.

A. Continuum limit

We begin by writing down Maxwell's equation in the continuum limit. Assuming units of $c = 1$ and neglecting factors of 4π, the most general form of Maxwell's equations will yield:

$$\nabla \times \mathbf{E}(\mathbf{x},t) = -\frac{\partial \mathbf{B}(\mathbf{x},t)}{\partial t} \quad (1)$$

$$\nabla \times \mathbf{H}(\mathbf{x},t) = \frac{\partial \mathbf{D}(\mathbf{x},t)}{\partial t} + \mathbf{J}(\mathbf{x},t), \quad (2)$$

where $\mathbf{D} = \varepsilon_0 \mathbf{E} + \mathbf{P}$ and $\mathbf{H} = \mu_0^{-1}\mathbf{B} - \mathbf{M}$ describe the interaction of the waves with charged matter, mediated via the polarization $\mathbf{P}$ and magnetization $\mathbf{M}$ fields (e.g. in non-magnetic, linear, homogenous media, they are given by $\mathbf{D} = \varepsilon \mathbf{E}$ and $\mathbf{B} = \mu \mathbf{H}$, respectively). Here, $\mathbf{D}$ and $\mathbf{B}$ are subject to constraints of the form: $\nabla \cdot \mathbf{D} = \rho_f$

and $\nabla \cdot \mathbf{B} = 0$. (Note: Because any $\mathbf{D}$ and $\mathbf{H}$ satisfying the last two equations automatically satisfy the local longitudial constraints, one needn't worry about enforcing the divergenceless condition—these are simple statements about charge continuity and can be easily derived from the curl equations above).

In principle, the solution of the above coupled partial differential equations yields all the necessary information to describe light and it's interaction with matter. Our goal is to numerically predict the space-time evolution of the fields, and related expressions thereof: these will mainly focus in the visualization of steady-state field profiles arising from time-dependent sources.

B. Discretized equations

The discretization of the above equations is given by Ref. **?** in thorough detail for the case where P and M are both linear in the fields, and follows a leap-frog approach in time (with center-difference and yee-grid discretizations for the fields—see Fig. 1). By discretization, we simply mean that the continuous degrees of freedom associated with time and space are quantized, e.g. in one dimension, $dx \to \Delta x$, and $dt \to \Delta t$. In general, before performing any numerical calculations, one must check that the discretized equations satisfy a number of numerical criteria for stability and convergence. An analysis of the convergence as a function of resolution is also required. In the case of Maxwell's equations, one can show that, in most standard media, the discretized Maxwell equations are both numerically stable and convergent as a function of resolution, with the possibility of quadratic accuracy for continuous ε. In order to ensure stability however, a courant condition must be enforced (in one dimension, one requires that $\Delta x/\Delta c < 1$).

As turns out, in one and two dimensions, the vectorial equations can be decomposed into solutions with strictly transverse electric (TE, $\mathbf{E} \cdot \hat{\mathbf{z}} = 0$) and magnetic (TM, $\mathbf{B} \cdot \hat{\mathbf{z}} = 0$) fields [ref]. In the TM case ($E_z, B_x, B_y \neq 0$), the discretized equations take the form:

$$\frac{B^{n+1/2}_{y,(i,j)} - B^{n-1/2}_{y,(i,j)}}{\Delta t} = \frac{E^{n}_{z,(i+1/2,j)} - E^{n}_{z,(i-1/2,j)}}{-\Delta x} \tag{3}$$

$$\frac{B^{n+1/2}_{x,(i,j)} - B^{n-1/2}_{x,(i,j)}}{\Delta t} = \frac{E^{n}_{z,(i,j+1/2)} - E^{n}_{z,(i,j-1/2)}}{-\Delta y} \tag{4}$$

$$\frac{D^{n+1}_{z,(i+1/2,j+1/2)} - D^{n}_{z,(i+1/2,j+1/2)}}{\Delta t} = -J^{n}_{i+1/2,j+1/2} + \frac{H^{n+1/2}_{y,(i+1,j)} - H^{n+1/2}_{y,(i,j)}}{\Delta x} + \frac{H^{n+1/2}_{x,(i,j+1)} - H^{n+1/2}_{x,(i,j-1)}}{\Delta y} \tag{5}$$

$$D^{n+1}_{z,(i+1/2,j+1/2)} = \varepsilon_0 E^{n+1}_{z,(i+1/2,j+1/2)} + P^{n+1}_{z,(i+1/2,j+1/2)} \tag{6}$$

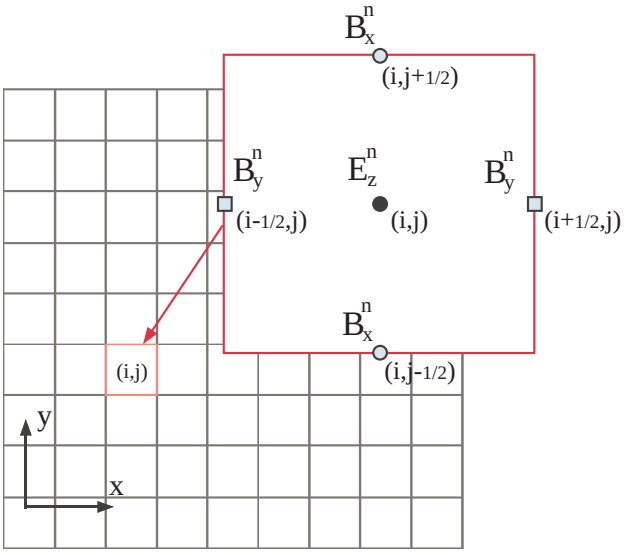

FIG. 1: Schematic illustration of computational cell. The x and y directions are denoted by the integer (i, j) labels, and the fields at each grid point are denoted accordingly. Inset: Illustration of the Yee grid as applied to the electric $\mathbf{E}$ and magnetic $\mathbf{B}$ fields.

$$B^{n+1/2}_{||,(i,j)} = \mu_0 H^{n+1/2}_{||,(i,j)} + M^{n+1/2}_{||,(i,j)} \tag{7}$$

where we have yet to explicitly write the dependence of $\mathbf{P}$ and $\mathbf{M}$ on the fields. The Yee lattice we have chosen places the electric $\mathbf{E}$ and $\mathbf{D}$ fields at half-integer points on the grid $(i + 1/2, j + 1/2)$ and at integer time-steps n. The converse is true for the magentic $\mathbf{B}$ and $\mathbf{H}$ fields. For a schematic illustration, see Fig. 1.

There are a number of posssible boundary conditions that one can enforce. *Metallic* boundary conditions require that $\mathbf{E} \cdot \hat{\mathbf{n}_{||}} = 0$ and $\mathbf{B} \cdot \hat{\mathbf{n}_{\perp}}$, which in two-dimensions (due to the separability of the polarizations) can be enforced by letting $E_z = 0$ or $H_z = 0$ for either the TM or TE polarizations, respectively. Another useful set of boundary conditions are *periodic* boundaries: here, the fields at any given boundary must wrap around the opposite (parallel) boundary. For example, enforcing periodic boundary conditions in the x direction is equivalent to enforcing that the field be given by $E \sim E(x, y)e^{ikx}$, where $k \equiv 2\pi/L_x$ and L_x is the length of the computational cell in the x direction. Finally, one can enforce *perfectly-matched* boundary conditions (PML), which allow waves to attenuate outside the computational cell: this is equivalent to having vaccum at the termination of the cell and can be achieved in practice by applying a coordinate transformation to Maxwell's equations [ref]. We shall not attempt this latter case here.

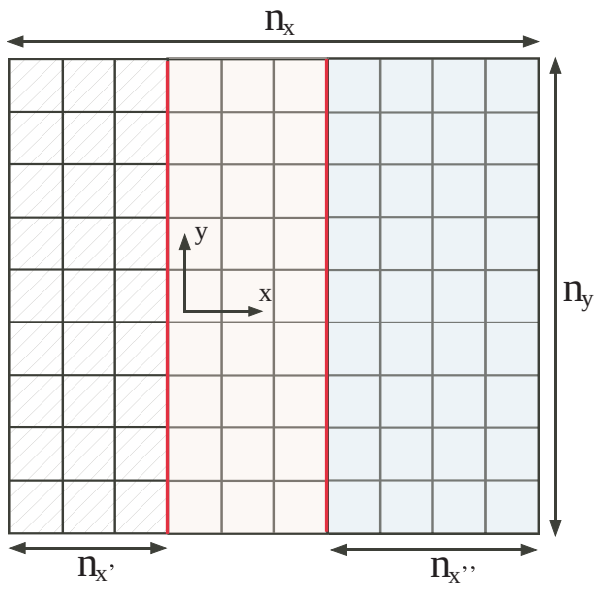

FIG. 2: Schematic illustration of computational cell $(n_x \times n_y)$, parallelized in one direction (x) and distributed over three processors. The dashed/red/blue partitions of size $(n_{x'} \times n_y)/(n_{x'} \times n_y)/(n_{x''} \times n_y)$ each belong to different processors.

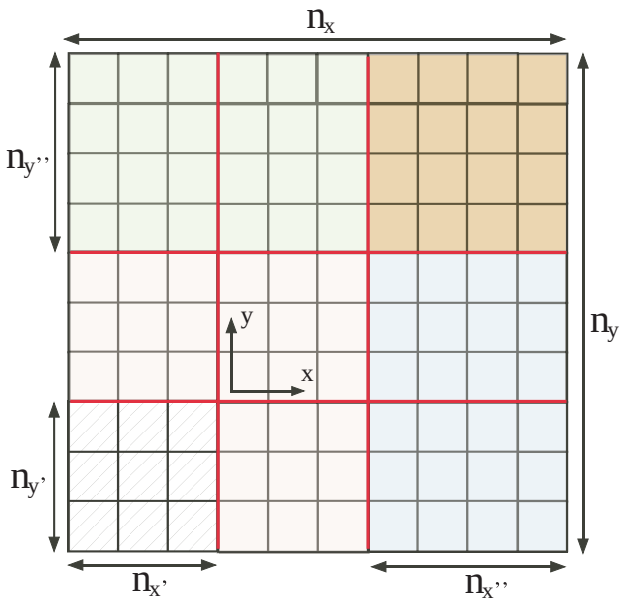

FIG. 3: Schematic illustration of computational cell $(n_x \times n_y)$, parallelized across the x and y directions, and distributed over nine processors. The dashed/red/blue/green/brown partitions each belong to different processors.

C. Parallelizing the domain

When deciding how to parallelize any finite-difference scheme, two very important questions arise: first, how should one distribute the grid over the processors in order to minimize the communication time arising from boundary elements; second, how should one communicate the data over the boundaries? The obvious goal is to reduce the time and/or memory complexity, and this will depend on the number of processors as well as the dimensionality of the problem.

First consider the serial problem. The computational problem is quite simple: each time step (iteration) requires us to multiply a very sparse matrix (the discretized evolution operator), which, upon inspection (see the discretized equations above), yields a time requirement that is linearly proportional with the number of pixels in the computational cell, i.e. of $\sim O(\text{res}^d)$ complexity, for a uniformly spaced grid of dimension d, such as the one under consideration [1]. As a consequence, for relatively large cells, or for three-dimensional problems, the complexity of a single iteration grows quite rapidly, and the numerics become prohibitive. Storage also plays an important factor in the parallelization of our equations. Because we must store six different *complex* fields (inlcuding $\mathbf{P}$ and $\mathbf{M}$), and because these are vector fields in three dimensions, storage in general will go as $\sim O(36\text{res}^d)$. This problem is often a great concern and as we shall see below, an important criteria for parallelization.

Now consider the parallel problem. As before, the time complexity should be linearly proportional to the number of degrees of freedom $N \sim \#\text{pixels} = \text{res}^d$. If we distribute these over n_p processors, the complexity will be reduced by a factor of n_p to yield an $\sim O(\text{res}^d/n_p)$ process. If this were the full story, we'd have an *embarrassingly parallel* problem, but we do not. We have neglected the communication time required due to boundary elements. In general, these will depend on the number of pixels at the boundaries of each processor N_b, which will grow (again neglecting constant factors that depend on the partitioning or discretization scheme) as $N_b \sim \text{res}^{d-1}$. Because each processor must acquire boundary information, the overal communication complexity will be of order $O(\text{res}^{d-1} \times n_p)$. The total time complexity will therefore be given by the following expression:

$$\#\,\text{op. counts} \sim \alpha \frac{\text{res}^d}{n_p} + \beta \text{res}^{d-1} n_p, \tag{8}$$

where the first term is proportional to the volume of the whole computational cell, and the second term is proportional to the "area" of each processor's designated volume. From inspection, we observe that in order to optimize the operation count, one desires a parallel scheme that *minimizes* the surface (communication time) to volume (iteration time) ratio. That is, if we are to gain from the parallelization, we want to make sure that $\beta/\text{res} < \alpha(n_p - 1)/n_p$. For a fixed value of n_p and resolution, this translates into minimizing the value of β. Note that the values of α and β will in general depend on the desired differentiation scheme. For example, in a center-difference discretization scheme, one requires the

values of adjacent grid points. Therefore, $\alpha = 3$ and β will vary depending on how the cell is partitioned.

In one dimension, where our domain is basically a line of pixels, Eq. 8 takes on a simple form. Consider a cell of size N, and n_p processors. If we partition the cell uniformly, then the number of pixels per processor will be given by $[|N/n_p|]$, where $[|\cdots|]$ denotes the "floor" operator (the remaining grid points can be assigned either to the master process or to a designated number of slave processors). As noted before, for a center-difference discretization, each boundary pixel will require an additional grid point, and therefore $\beta = 1$. The overall complexity in this case will grow as (ignoring the additional pixels in the master or last processor):

$$\#\,\text{op. counts} \sim 3\left[\left|\frac{N}{n_p}\right|\right] + n_p, \tag{9}$$

For $1 < n_p \ll N$, one can achieve some gains, but they are small in most cases since N is often not very large. For pratical purposes, this is a relatively unimportant problem to parallelize.

As a first non-trivial example of how the distribution scheme can affect the overall scaling, consider the resulting complexities from uniformly partitioning a two dimensional cell over either one (the largest) or two directions. Figure 2 and Fig. 3 illustrate the two ideas. Like before, the important quantity of interest is the surface "communication" area to volume ratio. Consider a computational cell of size $n_x \times n_y$ distributed over n_p processors via the two different schemes mentioned above. If we partition the data in only one dimension (as depicted by Fig. 2), then the surface area will scale as $\sim n_y \times n_p$, whereas if we partition the data in two dimensions (as depicted by Fig. 3), the latter will scale as $\sim [2([|n_x/\sqrt{n_p}|] + [|n_y/\sqrt{n_p}|]) \times n_p \sim 2(n_x + n_y) \times \sqrt{n_p}$. The distributed volume will be roughly the same for both cases, and will scale as $\sim n_x \times n_y/n_p$. In the end, therefore, the second parallelization scheme (over the two directions) will minimize the communication time by a factor $\sim 4/\sqrt{n_p}$, assuming $n_x = n_y$ for simplicity—in general, $n_x \neq n_y$ will only change the constant factor and not the scaling properties or the convergence of the schemes. The one and two dimensional parallelization schemes will therefore yield dramatically different results depending on the number of processors and size of the problem involved. For completeness, I quote the two scaling complexities:

$$O_{1D} \sim \left(\frac{n^2}{n_p} + nn_p\right) \tag{10}$$

$$O_{2D} \sim \left(\frac{n^2}{n_p} + 4n\sqrt{n_p}\right), \tag{11}$$

where we assume that $n_x \approx n_y \equiv n$ and obviate constant scaling factors.

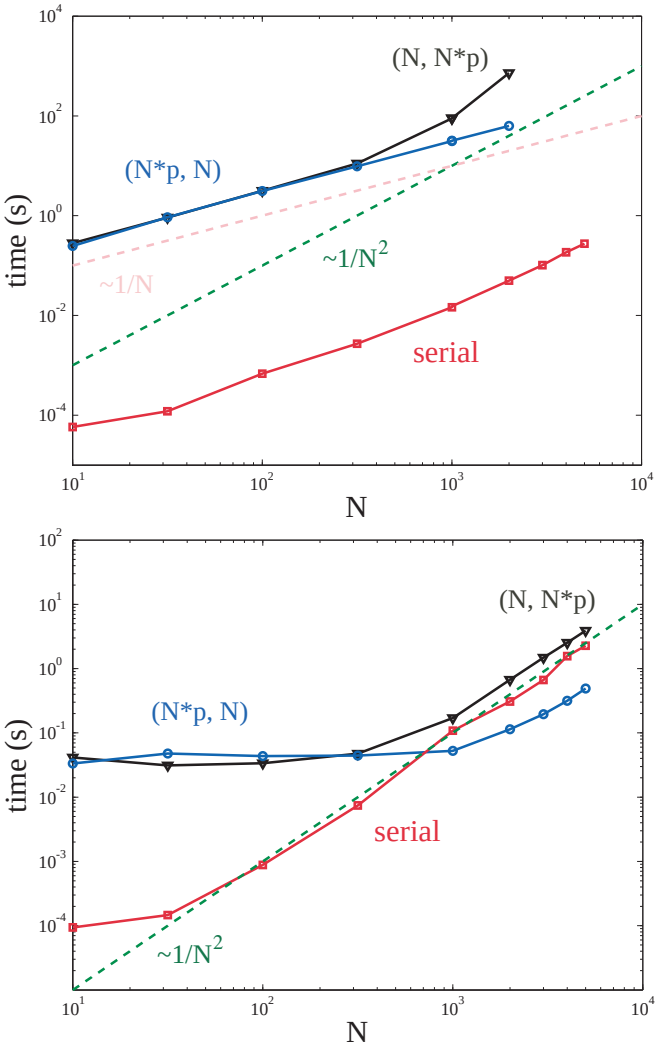

FIG. 4: Log-log plot of temporal complexity (time) in seconds vs. size of matrix N for evaluation of Eq. 12 using both looped (top) and vectorization (bottom) approaches. The implementations were performed in star-p using $n_p = 4$. Linear (pink) and quadratic (green) scalings are also plotted as a reference to the assymptotic dependence.

D. Approach to parallelization: vectorization and hybridization

In considering how to parallelize Maxwell's equations, we must think carefully about which method is best suited for the task, taking into consideration the type of matrix operations required. A general analysis that obviates some of the details of the problem, such as the one performed in the previous section, would suggest that we should partition the domain in two dimensions. However, as we demonstrate below, it is far more beneficial to employ a different discretization procedure, one that takes advantage of the form of Maxwell's equations and which can be easily implemented in star-p.

To begin with, consider two matrices $B \in N \times N$, and $E \in N \times N$, and the following equation for B:

$$B_{i,j} = B_{i,j} + \alpha \left(E_{i,j+1} - E_{i,j} \right), \tag{12}$$

where we note it's resemblance with the discretized Maxwell equations above (the second term is simply a center-difference discretized derivative). When considering which type of parallelization approach to apply, we must take into account that the only communication-intensive operation is the difference equation on the right hand side. Since it only involves differences in the y direction, we would be encouraged to parallelize over the x direction (perpendicular to it). Another important decision to make is whether or not one should use a *data-parallel* or a *task-parallel approach*. The former emphasizes XXX, while the latter emphasizes XXX. This problem is therefore a very likely candidate for a data-parallel method. Two possible implementations of looped and vectorized algorithms are given below:

$$\begin{aligned} &B \in N \times N * p, E \in N \times N * p \\ &for\, k = 1 : (N-1) \\ &\quad B(:,k) = B(:,k) + \alpha \left(E(:,k+1) - E(:,k) \right) \\ &end \end{aligned} \tag{13}$$

$$\begin{aligned} B \in N \times N * p, E \in N \times N * p \\ B(:, 2 : end) = B(:, 2 : end) \\ + \alpha \left(E(:, 2 : end) - E(:, 1 : end - 1) \right) \end{aligned} \tag{14}$$

where we can also parallelize over the x-direction, i.e. $B \in N * p \times N$, and $E \in N * p \times N$. The two previous observations (vectorization and parallelization over the y direction is preferential) can be readily confirmed by Fig. 4, in which we benchmarked both looped (top) and vectorized (bottom) implementations of the algorithm using $n_p = 4$ processors, along with the two possible one-dimensional parallelizations (x and y). As shown, the vectorized x-dimensional parallelization (perpendicular to the direction of cost-operation) is optimal and yields temporal gains for $N \geq 10^3$. The log-log plots demonstrate assymptotic N^2 behaviors (shown also by the green dashed line) for large $N \gg n_p$, as expected. For small N, communication lags $\sim N$ are evident in both the looped and the vectorized versions. All, if not most of these features are expected from our previous analyses.

Based on these benchmarks, we would be inclined to parallelize Maxwell's equations in one dimension, rather than in two dimensions, as is standard in the literature [ref]. However, Maxwell's equations do not involve derivatives over only one direction, but they mix derivatives in multiple directions. For example, while parallelizing the electric field E_z over the direction perpendicular to x would yield temporal gains for Eq. 3, they would ultimately be swamped by the inefficiency (communication) required of Eq. 4. Simply stated, the longest timescale will always be the bottleneck. Similarly, since Eq. 5 involves finite differences both in x and y, a simple one-dimensional discretization would only decrease the numerical efficiency, at least for the parameter range explored above $N \in (10^3, 10^4)$, where communication seems to be a great expense. At this point, we'd like to point out that, a wide range of electromagnetic structures comprise this regime of interest $N >= 10^3$. A general rule for designing parallel codes should, in my opinion, involve achieving maximal temporal and memory improvements (scaling) over the widest range of possible values of N. For small N/n_p, this is often hindered by communication expenses, and it is precisely this small N regime that I shall seek to optimize (note that in doing so, we are also optimizing the large N regime).

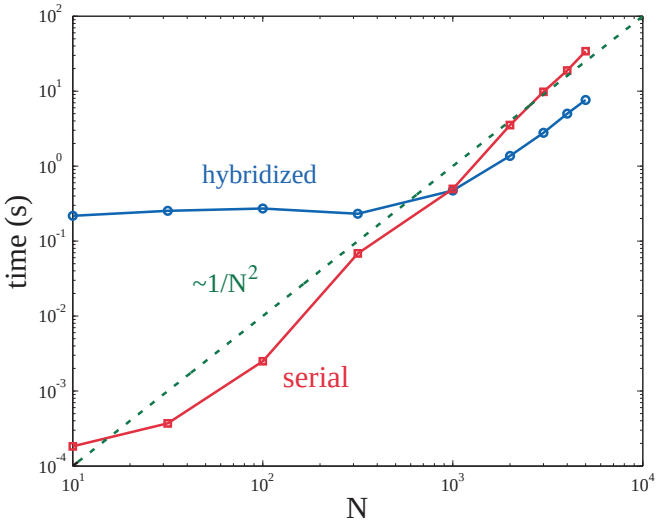

FIG. 5: Log-log plot of temporal complexity (time) in seconds vs. size of matrix N for evaluation of Eq. 12 using the hybridization approach described in the text. The implementations were performed in star-p using $n_p = 4$. The dashed green line is included as a reference to the asymptotic N^2 dependence of the complexity.

Another possible approach to parallelization is the two-dimensional parallelization explored above (see Fig. 3). Unfortunately, star-p does not allow for parallelization over two directions, and we cannot benchmark this case. However, from our previous results, we expect the longest timescale (the communication bottleneck) to dominate and yield suboptimal results.

1. *Hybridization*

We now turn to a more interesting, smarter approach to parallelizing Maxwell's equations. Using the previous results as a guide, and by inspection of our discretized Maxwell equations, we can already guess at a possible improvement. In particular, consider the parallelization

scheme below:

$$B_y \in N \times N * p \sim E_{z,x} \in N \times N * p \tag{15}$$
$$B_x \in N * p \times N \sim E_{z,y} \in N * p \times N \tag{16}$$
$$H_y \in N * p \times N \sim B_y, H_x \in N \times N * p \sim B_x \tag{17}$$
$$D_z \in N * p \times N \sim H_y \in N \times N * p + H_x \in N * p \times N \tag{18}$$
$$E_{z,y} \in N * p \times N \sim \varepsilon^{-1} D_z, E_{z,y} = E_{z,x} \tag{19}$$

where we create two *auxiliary* fields $E_{z,x} \in N \times N * p$ and $E_{z,y} \in N * p \times N$, both of which are parallelized perpendicular to their direction of cost operation (in order to minimize communication cost), along with the corresponding B field. By construction, therefore, Eq. 3 and Eq. 4 have minimal communication cost and should therefore exhibit improvements, as in the example above. This choise already constraints our parallelization for the magnetic fields B, and the corresponding H fields. Fortunately, Eq. 5 involves finite differences over H_x and H_y in directions perpendicular to the parallelization choise for the auxiliary and B fields, and should also exploit the lack of communication between processors. Finally, the only possible communication intensive part is the last assignment operations $D = \varepsilon E$, which can be argued to be negligible compared to the latter four. Here, E refers to the value of the auxiliary electric fields defined above, both of which must be synchronized (equated) at the end of each time step, since ultimately there is only one E field, and not two.

Due to the mixing of parallel directions, I chose to refer to this method as a hybridization approach to parallelization. The resulting benchmarks as a function of N, for $n_p = 4$ (as before) are shown in Fig. 5, demonstrating a rather dramatic increase in parallel performance, thus suggesting that the improvements due to lack of communication in Eq. 3–Eq. 5 are substantial enough to overcome any source of communication from the remaining equations. A particularly encoruaging result is the order-of-magnitude improvement achieved in the $N \sim 10^3$ regime. Extrapolation to larger N suggests the possibility even better > 10 improvements.

E. Pseudocode

In this subsection, I sketch a pseudocode representation of the algorithm used to simulate the numerics (restricted to the case of TM polarization):

(1) Initialize the seven field types $E_{z,x}$, $E_{z,y}$, B_x, B_y, D_z, P_z, J_z to zero. User must specify a resolution (a), a cell size ($N = n_x \times n_y$), dielectric function $\varepsilon(\mathbf{x})$, and current distribution $J_z(\mathbf{x}, t)$—available distribution types are gaussian and continuous wave (cw) sources. This is all performed inside matlab.

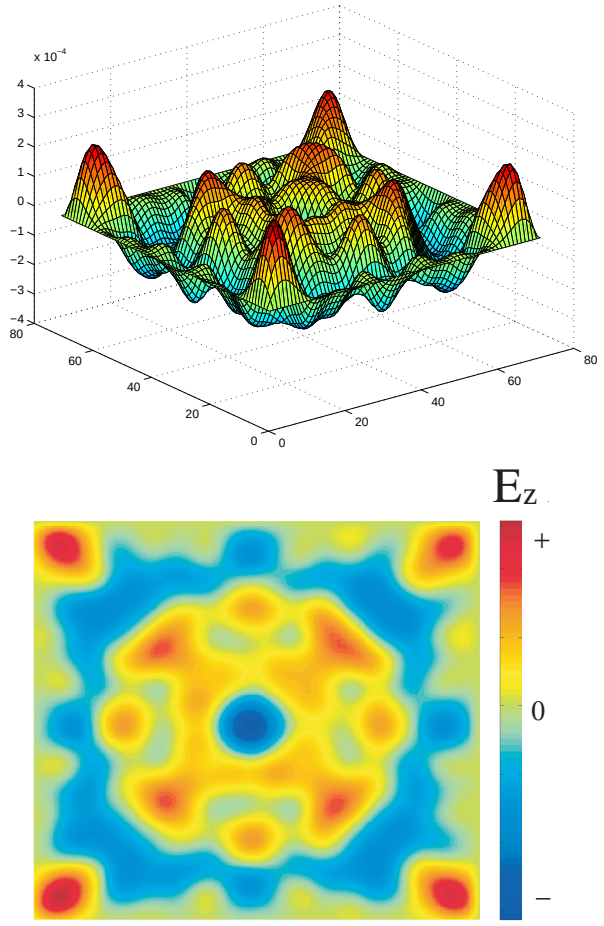

FIG. 6: Three-dimensional (top) and contour (bottom) plots of the steady-state electric field resulting from a source $J_z \sim e^{i\omega t}$. The exact value of ω is not of consequence or important for this manuscript.

(2) Time stepping involves iteration of the discretized equations above over a time $T \sim t \times a$, where t is the user-specified time over which to evolve the equations. The update equations, in the right order are given above.

(3) Return the fields at the specified time. If a fast-fourier transform or a similar operation requiring the field at various times is needed, an option exists to sample the fields over various times.

The actual code will be included along with this manuscript.

III. RESULTS AND REMARKS

Below, we give a sample calculation of the type of field visualization that can be achieved using our code. In this example, an electric field source $J_z \sim e^{i\omega t}$ is excited. The resulting fields are allowed to equilibrate until a steady-state is reached. This steady-state field is plotted as a function of position in Fig. 6 as both 3d (top) and projected contour (bottom) plots, showing the expected (based on symmetry) quadrupole field profile. For these values of resolution ~ 100 and supercell size (computational size), the evolution matrices were no more than $N \sim 200$, resulting in $\sim O(10^3)$ degrees of freedom. Unfortunately, the parallelization gains for such a small problem are quite insignificant, as predicted by our benchmarks in the previous section. In fact, the parallel version was approximately 10 times slower than the serial version, which finished in roughly 10 seconds.

While the previous example was highly suboptimal for parallelization, there are numerous cases of interest whereby our parallelization scheme would be extremely beneficial, e.g. in structures where $N > 10^3$. Such geometries are perhaps the most important and interesting to study in electromagnetism. One such possibility is that of a periodic crystal (with many lattice periods) surrounded by a slab or uniform index structure, requiring a substantially bigger supercell size $L \gg d$, where d is a lattice period. In this case, even a small resolution $res \sim 30$ can yield dramatically large computational cells, on the order or greater than $\sim 10^6$ in two dimensions, and even greater in three dimensions, $\sim 10^9$. Simple calculations performed on a structure such as this one (for $N \sim 10^5$) showed the great power of parallelization, with orders of magnitude improvements in both the time and memory complexities. In particular, for this relatively small supercell $N \sim 10^5$, the memory constraints made this structure impossible to analyze in serial. The parallel version was not only feasible to analyze, but also allowed for quick results (on the order of hours). Such a geometry in serial (ignoring memory constraints) would have required days to analyze.

In conclusion, we have demonstrated an efficient way to parallelize Maxwell's equations using star-p, based on a data-parallel approach. Most general parallel electromagnetic codes do not exploit the possibility of parallelization in one dimension, and the use of auxiliary fields for minimization of communication costs (the hybridization method developed above). While such a method poses other questions regarding it's lack of generality and efficiency in cases where, for example, ε is anisotropic, or where, field quantities such as the total energy are required at each time step, it is essentially quite robust and efficient in the more standard cases of interest. Other possible optimizations that make use of parallelism might involve load balancing the processors for optimal performance. In particular, one could perform a single iteration test case in order to benchmark and identify areas of highly intensive operation (bottle-necks), and use this information in order to re-distribute the sizes and number of processors over the various regions. Such use of non-uniform parallelization could be especially beneficial in cases where Perfectly-Matched-Layer boundary conditions are required, or when Maxwell's equations are discretized in a non-uniform fashion, such as in a finite-element framework.

Acknowledgements

ARW is very grateful to Yee Lok, and Steven G. Johnson for their help and support during the last stages of the project, and to Alan E. for his inspirational lectures throughout the semester.

[1] The constant factor is not of interest to us in this analysis.