First step towards Parallel and Adaptive Computation of Maxwell's Equations

Stefan Findeisen

Karlsruhe Institut of Technolgy
Departement of Mathematics
Kaiserstr. 12, 76128 Karlsruhe, Germany
stefan.findeisen@kit.edu

Maxwell's equations describe the behavior of an electromagnetic wave in three spacial dimensions over a certain period $[0,T]$. The wave consists of two fields, the electric $\mathbf{E}$ and the magnetic $\mathbf{H}$, respectively. They can be computed by the linear first–order Maxwell system

$$\mu\partial_t\mathbf{H} + \nabla\times\mathbf{E} = 0, \quad \varepsilon\partial_t\mathbf{E} - \nabla\times\mathbf{H} = 0, \quad \nabla\cdot(\mu\mathbf{H}) = 0, \quad \nabla\cdot(\varepsilon\mathbf{E}) = 0 \tag{1}$$

with permeability μ and permittivity ε. Each field consists of three components. Hence one has to compute six components of the fields, which depend on space and time. For a given problem this can lead to huge linear systems. This is why fast space–time codes are needed to solve the problem in a reasonable computation time.

Together with an initial condition $\mathbf{u}_0$, (1) can be written as

$$M\partial_t\mathbf{u}(t) + A\mathbf{u}(t) = \mathbf{0} \text{ for } t \in [0,T]\,, \qquad \mathbf{u}(0) = \mathbf{u}_0,$$

on a bounded Lipschitz domain $\Omega \subset \mathbb{R}^3$, where M, A, u are given by

$$M := \begin{bmatrix} \mu & 0 \\ 0 & \varepsilon \end{bmatrix}, \quad A := \begin{bmatrix} 0 & \nabla\times \\ -\nabla\times & 0 \end{bmatrix}, \quad \mathbf{u} := \begin{bmatrix} \mathbf{H} \\ \mathbf{E} \end{bmatrix}.$$

Using a discontinous Galerkin discretization with upwind flux for the spacial discretization, we receive the semi–discrete system of equation

$$M_h\partial_t\mathbf{u}_h(t) + A_h\mathbf{u}_h(t) = 0 \text{ for } t \in [0,T]\,, \qquad \mathbf{u}_h(0) = \mathbf{u}_{h,0},$$

where M_h and A_h are the corresponding mass and flux matrices, respectively. In order to avoid a CFL (Courant–Friedrichs–Lewy) condition we use the implicit midpoint rule for time integration, which is of order two. Hence we have to solve a linear system $(M_h + \frac{\tau}{2}A_h)v = b$ in every time step. However an implicit method allows a larger step size τ than an explicit scheme.

In the following we present an adaptive programming model which is able to solve the 2D reduction of Maxwell's equations, e.g., $\mathbf{u} = (\mathbf{H}_1, \mathbf{H}_2, \mathbf{E}_3)$ and $\mathbf{H}_3 \equiv \mathbf{E}_1 \equiv \mathbf{E}_2 \equiv 0$ and $\Omega \subset \mathbb{R}^2$. The discretization of our space–time domain $Q := \Omega \times [0,T]$ is done as follows. First Ω is decomposed into a finite number of open elements (e.g. triangles) $K \subset \Omega$ such that $\bar{\Omega} = \bigcup \bar{K}$. Analogously $[0,T]$ is decomposed into a finite number of N open intervals $I_n = (t_n, t_{n+1}) \subset [0,T]$ for $n = 0, \ldots, N-1$ such that $[0,T] = \bigcup_{n=0}^{N-1} \bar{I}_n$. Now we are able to define space–time cells $K_I := K \times I$ which consists of a spacial element K and a time interval I. Hence Q can be decomposed into a finite number of open space–time elements $K_I \subset Q$ such that $\bar{Q} = \bigcup \bar{K}_I$. The data structure is organized as hash maps. That means that every space–time cell and its components (such as faces, edges, vertices) is identified by its geometric midpoint and stored in a hash map (where the midpoints are used as hash keys). By doing so, it is easy to distribute the space–time cells among the different processes and solve the problem parallel.

So far, our code is p–adaptive and uses different time steps. That means that the order of a polynomial on a space–time cell is adapted and the space–time cells can be refined in time. In a first step we compute our solution $\mathbf{u}_h$ on a slice $S_n := \Omega \times [t_n, t_{n+1}] \subset Q$ with fixed polynomial degree $p = 0$. Then we take the flux over the cell faces as an indicator where the polynomial degree should be increased and cells should be refined in time. To be more precise, the indicator n_{K_I} of a cell K_I is given by the sum of the face indicators η_f:

$$\eta_{K_I}^2 := \sum_{f \in \mathcal{F}_{K_I}} \eta_f^2,$$

$$\eta_f^2 := h_f \left\| (\mathbf{F}^*(\mathbf{u}_h) - \mathbf{F}(\mathbf{u}_h)) \cdot n_{K_I} \right\|_{L_2}^2,$$

where F and F^* are the flux and the numerical flux and h_f, n_f are the area and the outer normal vector of the face f. The polynomial degree is increased, if the indicator of the cell fulfills the inequality

$$\eta_{K_I} > (1-\theta) \max_{K_I \in S_n} \eta_{K_I}$$

for a given parameter θ (e.g. $\theta = 0.9$). After that we recompute the solution with the new distribution of the polynomial degrees and time refined cells. Since our code provides polynomials up to order $p = 4$, four p–adaptive refinements are possible. Although the computation of η_{K_I} is heuristic, it leads to good results in a sense that high polynomial degrees are used in areas where a single wavefront is located. In areas with absence of a wave, lowest polynomial degrees are used.

In the future the described programming model will be the basis for a space–time adaptive code to solve Maxwell's equation in three spacial dimensions. Thus our next steps will be h–adaptivity in space.