Exploring novel electromagnetic algorithms for efficient PIC simulations

A.T. Sexton, A. Kish, M. Lavell, M.J. Kim, and A.B. Sefkow

The TriForce Center for Multiphysics Modeling, a collaboration between the Departments of Mechanical Engineering, Physics, Computer Science, and the Laboratory for Laser Energetics

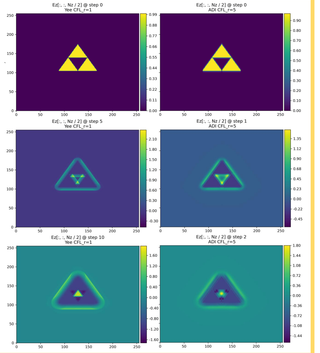
Abstract or summary

- **TriForce is a hybrid fluid-kinetic multiphysics simulation program currently built with the kinetics-based Library for Integrated Numerical Kinetics (LINK).**
- **TriForce uses a combination of Particle-In-Cell (PIC) and radiation Magnetohydrodynamic (rMHD) codes for simulating plasma and fusion physics. Additional goals for LINK are to include a broader variety of fundamental physics simulations and to take advantage of distributed computing and accelerators.**
- **The standard algorithm for electromagnetic simulations require extremely small-time steps and/or high-resolution grids in order to achieve high order accuracy and reduce numerical artifacts that arise from the discrete nature of the explicit algorithms.**
- **Implicit methods are often used to increase performance but are rarely developed with modern accelerator technology in mind.**
- **Long term goal of coupling better algorithms to better hardware design to obtain dramatic increases in simulation performance. Increasing time step size allow for longer running simulations.**

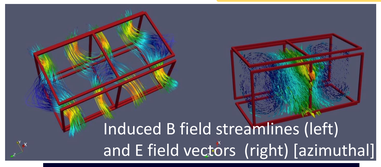
Scientific motivation

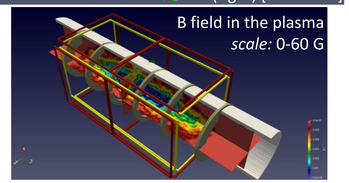
$$CFL = \frac{c\Delta t}{\Delta x} \leq 1 \qquad \Delta t = \frac{CFL \cdot \Delta x}{c} = \frac{1 \cdot 1cm}{3E8\ {}^{m}/_{s}} \approx 33\ ps$$

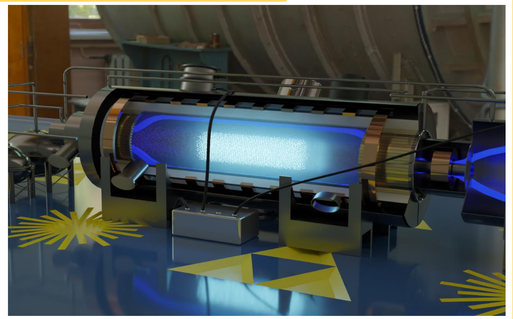
- Electromagnetic solvers are constrained by the Courant-Friedrich-Lewy (CFL) Condition
- Explicit methods are limited to a CFL number $\leq 1/\sqrt{d}$ in order to remain numerically stable where d is the dimensionality of the simulation. This is the upper limit of the propagation speed of information through the mesh
- Implicit methods can exceed this limit while remaining numerically stable. Some algorithms are considered unconditionally stable for any CFL
- Traditional CPU bound methods must solve Maxwell's Equations for every point in the domain one at a time (or up to the number of threads on the CPU)
- These two limits make high resolution solver methods computationally expensive and time consuming, with days or weeks of compute time required to achieve macro-time scales
- Amdahl's Law puts the upper limit of the speed up from parallelization at ~20x. The goal then becomes not only to parallelize and accelerate the code, but also to choose algorithms that trade extra computation for reducing the number of steps to needed to compute



Scientific application: *RF* heating of an FRC plasma



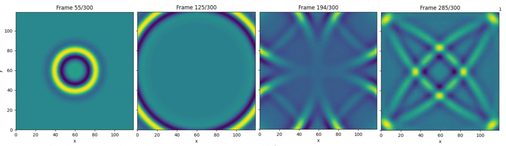




Princeton Field Reversed Configuration (PFRC) Credit: Katie Jarvis (Blender/TriForce LINK)

Yee Algorithm

- Fully explicit method.
- Second order accuracy in time/space
- CFL limited to very small timesteps and small spatial steps
- Low Complexity
- Moderate memory footprint
- High performance (least compute steps per time step)



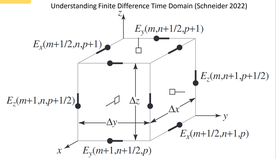
Electric field with unit gaussian pulse source. 2nd order absorbing boundary conditions. Amplitude drops by ~10^4 between second and third image. 128^3 cells.

$$E_x^{n+1} = \frac{1-\frac{\sigma_e\Delta t}{2\epsilon}}{1+\frac{\sigma_e\Delta t}{2\epsilon}}E_x^n + \frac{1}{1+\frac{\sigma_e\Delta t}{2\epsilon}}\left\{\frac{\Delta t}{\epsilon\Delta y}\left(H_z^n|_{i,j,k} - H_z^n|_{i,j-1,k}\right) - \frac{\Delta t}{\epsilon\Delta z}\left(H_y^n|_{i,j,k} - H_y^n|_{i,j,k-1}\right) - \frac{\Delta t}{\epsilon}J_x^{n+1}\right\}$$

Semi-Implicit Algorithms

- Many variants (Alternating Direction Implicit, LOD, BTCS)
- High complexity, High memory footprint
- Low Performance (many compute steps)
- Parallel algorithms for banded matrices currently only work for factors of 2 systems.
- For ADI method, three fields require solving ($N_\alpha \times N_b$) N_c-sized tridiagonal systems

Yee Cube
Understanding Finite Difference Time Domain (Schneider 2022)



$$\frac{-\frac{\Delta t^2}{4\mu\epsilon\Delta z^2}}{(1+\frac{\sigma_e\Delta t}{4\epsilon})(1+\frac{\sigma_m\Delta t}{4\mu})}H_x^{n+\frac{1}{2}}|_{i,j,k+1} + \left(1+\frac{2(\frac{\Delta t^2}{4\mu\epsilon\Delta z^2})}{(1+\frac{\sigma_e\Delta t}{4\epsilon})(1+\frac{\sigma_m\Delta t}{4\mu})}\right)H_x^{n+\frac{1}{2}}|_{i,j,k} - \frac{-\frac{\Delta t^2}{4\mu\epsilon\Delta z^2}}{(1+\frac{\sigma_e\Delta t}{4\epsilon})(1+\frac{\sigma_m\Delta t}{4\mu})}H_x^{n+\frac{1}{2}}|_{i,j,k-1}$$

$$= \frac{1-\frac{\sigma_m\Delta t}{4\mu}}{1+\frac{\sigma_m\Delta t}{4\mu}}H_x^n|_{i,j,k} + \frac{1-\frac{\sigma_e\Delta t}{4\epsilon}}{1+\frac{\sigma_e\Delta t}{4\epsilon}}\frac{\Delta t}{2\mu\Delta z}(E_y^n|_{i,j,k+1} - E_y^n|_{i,j,k}) - \frac{\frac{\Delta t}{2\mu\Delta y}}{1+\frac{\sigma_m\Delta t}{4\mu}}(E_z^n|_{i,j+1,k} - E_z^n|_{i,j,k})$$

$$- \frac{\frac{\Delta t^2}{4\mu\epsilon\Delta z\Delta x}}{(1+\frac{\sigma_e\Delta t}{4\epsilon})(1+\frac{\sigma_m\Delta t}{4\mu})}\left\{(H_z^n|_{i+1,j,k+1} - H_z^n|_{i+1,j,k} - H_z^n|_{i,j,k+1} + H_z^n|_{i,j,k}) - \Delta x(J_y^{n+\frac{1}{2}}|_{i,j,k} - J_y^{n+\frac{1}{2}}|_{i,j,k-1})\right\}$$

Problem Scaling

- Best 3D CPU based methods offer ~100-200 cells per dimension
- GPU accelerated can ~100-1000 cells per dimension
- GPU memory independent of CPU memory. Twice the memory!
- Transfers between CPU-GPU are expensive
- ADI method ~5x slower than Yee (46 operations vs 9 operations)

	Yee	ADI+TDS
Calculations	4 Mult/Div + 5 Add/Sub	6+19 Mult/Div + 11+10 Add/Sub
Memory	22 Arrays (6 Field, 6 Source, 10 Coefficients) ~3 GB (256^3, 64-bit)	34+6 Arrays (6 Field, 3 Auxiliary, 6 Source, 19 Coefficients, 6 Banded Matrices) ~4.5 GB (256^3, 64-bit)

Parallelism

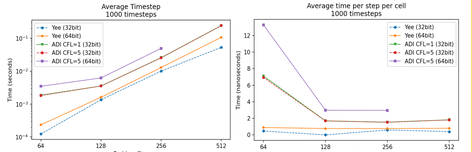
- AMD Threadripper Pro 5995WX (~$9000) - 64 cores / 128 threads
- Nvidia RTX 3090 (~$1500) - 10240 CUDA cores
- Parallelization requires careful management of memory and data structure design to achieve full performance benefits
- Certain algorithms gain little benefit from parallelization or are not parallelizable at all
- Many levels of granularity and hardware specific optimizations.

Challenges

- Complex programming requirements
- Hardware specific and algorithm specific optimizations
- Few analytical solutions for non-trivial problems
- Requires additional, complex code for boundary conditions and sources that have special considerations when applied to different algorithms

Results

- Implementations of Yee and ADI using C++/CUDA
- Tested on Nvidia GV100 GPU
- All work performed on GPU, no memory transfers after initialization
- Time per step grows exponentially with problem size. Time per cell per step remains consistent
- ADI method with CFL ≥ 5 required to outperform Yee method over time



Future Work

- More algorithms (Locally One-Dimension, Backward-time Centered-space,…)
- Timing and accuracy comparisons of algorithms
- Improved boundary conditions (2nd order absorbing, PML's)
- Improved sources, material properties, partial cell aliasing
- Explore algorithm and hardware specific optimizations
- Couple EM solver to particle-based fluid algorithms

Acknowledgments

This material is based upon work supported by the United States Department of Energy ARPA-E under Award No. DE-AR0001272, OFES under Award No. DE-SC0017951, and NNSA under Award No. DE-NA0003856.