Enhancement of Higher Order FDTD Method Using OpenCL, CUDA, and MPI on Single and Multiple CPUs/GPUs

Alec Weiss*[1], Sanjay DMello[1], Ashik Akbar Basha[1], Atef Z. Elsherbeni[2], Melinda J. Piket-May[1], and Mohammed F. Hadi[1,2,3]

[1] Electrical, Computer and Energy Engineering, University of Colorado at Boulder

[2] Electrical Engineering and Computer Science, Colorado School of Mines

[3] Electrical Engineering, Kuwait University

The high order finite difference time domain method (FDTD) such as the FV24 has for many years been an accepted way to increase electromagnetic simulation accuracy over the standard second order in space and time FDTD method. Due to the requirement of extra processing power for the FV24 method, techniques for increasing the processing speed of the FV24 algorithm have been tested and compared. These methods parallelize the FV24 code to decrease simulation time by splitting the processing between multiple central processing units (CPUs) or graphics processing units (GPUs).

The implementation of a multiple CPUs with shared memory was completed through the use of OpenMP for FORTRAN. OpenMP allowed for very easy division of code loops over many processors. This code was then run and timed on The University of Colorado at Boulder's JANUS supercomputer. Work was also extended to multiple CPUs with distributed memory using MPI directives and was also tested on the Janus supercomputer. The use of a multi-CPU approach proved that the high order algorithm can be easily sped up on most of todays systems which contain multiple cores to make it a very viable FDTD solver for electrically large systems.

Multiple GPU approaches were also taken and results compared on up to four GPUs at the Colorado School of Mines. Due to the parallel nature of GPUs, they provide a very large increase in computational capacity for FDTD simulations. Tests were run on both Nvidia Kepler (K10, k40, Titan-Z) and Maxwell (Titan-X) architectures. These GPUs are all affordable scientific computing GPUs readily available on the market today. Codes were created and compiled for these platforms using CUDA FORTRAN and OpenCL. OpenCL was found to be about 2.8% slower on the same hardware. While OpenCL did not perform quite as well as CUDA Fortran, it is a more viable option for many users as it is able to run on other manufacturer hardware in addition to those from Nvidia.