5-2016

Acceleration of DDSCAT Computation by Parallelization on a Supercomputer

Manoj V. Seeram
University of Arkansas, Fayetteville

Introduction

Parallel Programming Hardware

Computing power has been improving on orders of magnitude over the past century as the number of transistors on an integrated circuit has been exponentially rising in line with Moore's Law [4]. However, this technology is approaching the size of individual atoms, the smallest a transistor can be, and industry is searching for new techniques to create more computing power. Multiple alternatives are being researched including optical computers, quantum computers, graphical processing units (GPU), and distributed computing systems, also known as supercomputers. These technologies all capitalize on the power of parallel processing to some degree but along with unique benefits, each of these solutions has unique limitations. Optical computing, for example, can transmit data at a speed comparable to that of light and in parallel by using the diffractive property of electromagnetic waves. However, this energy transfer generates large amounts of heat that need to be removed to maintain data signals and the current cooling units may not be sufficient [5]. Quantum computing is also promising because of a qubit's ability to exist in multiple states and therefore try many solutions to a problem simultaneously. The technology is ideal for solving problems in parallel but the quantum state collapses once the qubit is measured and, since only one of the states can be measured, there is a probability that the solution detected won't be correct [6].

The remaining two technologies are currently feasible solutions that are accessible at most universities or commercially available. While central processing units, CPUs, contain a few cores and more cache memory, GPUs are "composed of hundreds of cores that can handle thousands of threads simultaneously". While each GPU core is less powerful than a CPU core, the GPUs massively parallel capability at a more efficient cost and price outweighs the potential

latency [7]. However, programs written for GPUs can't necessarily be run on regular CPUs, keeping the universal use of the parallelized application limited. Distributed computing, however, exploits a capability similar to the GPU out of multiple interconnected CPU cores. With the resources available for this project, the University of Arkansas High Performance Computing System, a distributed environment, was used to parallelize DDSCAT.

An important limitation in distributed computing is that the time to complete a program is dependent on the time taken by the slowest processor. In other words, the program doesn't move forward until each parallel job has been completed. This issue is encountered when determining how many wavelengths should be submitted to each processor. A special case of Bose-Einstein statistics [14] (also called stars-and-bars) can then be considered to determine the division that will allow for the least number of stars within each bar.

DDSCAT

DDSCAT is a software written in Fortran90 used to calculate "scattering and absorption of light by irregular particles and periodic arrangement(s) of irregular particles" [1] A user is allowed to specify target particle(s), generally in the nanometer scale, that will then be simulated to interact with electromagnetic waves in the visible spectrum, light. The wavelength range of light, specified by the user, is targeted on the nanoparticle(s) on the positive z axis and the resulting distribution and interaction of energy throughout the particle is quantified and mapped. Other programs can be used to plot graphs and 3D visuals of the output data.

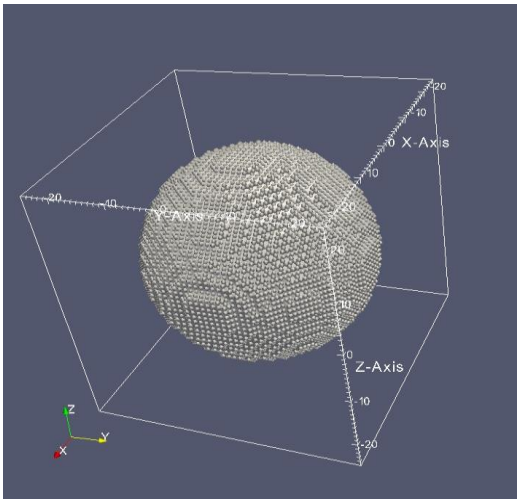

Figure 1 is a 3D model of a spherical nanoparticle generated using MayaVi [1].

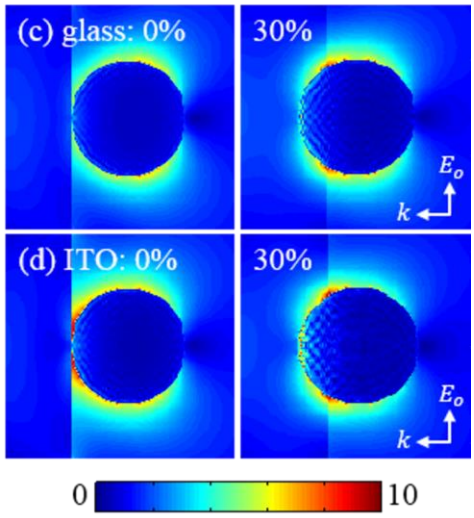

Figure 2 is near-field spectra of a spherical nanoparticle embedded in an Indium Tin Oxide substrate. The graph was generated from DDSCAT output data [15].

DDSCAT is stated to be enabled to exploit the resources in a distributed computing environment with its embedded MPI, Message Passing Interface, or OpenMP, Open Multi-Processing, functionality [1]. The software divides, or parallelizes, the computation by nanoparticle orientation and processes each set of calculations on a different core, a processor or unit that can maintain one thread of instructions at a time. Run times can still take many days or

hours (Figure 4) so whether the software parallelizes the program by other means, namely wavelengths, is to be tested.

Parallel Programming Software

MPI is a library standard "based on the consensus of the MPI Forum" [8] whose interface specifications have been defined for C and Fortran90 language bindings. The standard is useful for efficiently passing data between processors that, by default, are not in communication with each other. This is the crux for parallelization.

OpenMP is a "set of compiler directives and library routines that extend Fortran", C, and C++ [9]. It operates with a shared memory space, unlike MPI, and provides parallelism in this way. The setup is similar to one host storing data with multiple slave processors performing calculations with this data [10].

Testing DDSCAT MPI-enabled

Simulation Methodology

DDSCAT, written in Fortran90, numerically calculates optical cross sections (extinction, absorption, or scattering) of arbitrary subwavelength materials across user-specified incident wavelengths. A minimum of four files are required to run the program.

1. A target file that contains Cartesian positions of each dipole in the nanoparticle
2. A parameter file that describes various simulation parameters (e.g. range of wavelengths, refractive index of the medium)
3. A dielectric function material file for the nanoparticle
4. The DDSCAT executable.

Once the executable is run, it will read the other files within the folder to complete its simulation. Throughout the simulation, DDSCAT generates a file containing data for each wavelength. File files are output by DDSCAT upon completion of the simulation: ‘qtable,’ ‘mtable,’, ‘qtable2,’ and polarizability files are generated. The qtable file contains optical extinction, absorption, and scattering efficiency data of the nanoparticle at each wavelength.

A control simulation was set up using the DDSCAT Shape Generator [11] tool available on the nanoHUB website to determine the improvement that MPI provides to DDSCAT computation time. The simulation was selected to be a gold sphere, a commonly used material in research due to its inert properties, with an 80 nm radius and an Indium Tin Oxide cylindrical substrate with a 160 nm radius. Simulating the substrate allows plasmonic interactions between it and the sphere that would occur in an experiment to be accounted for. Figure 3 shows the graphical interface used to start the tool.

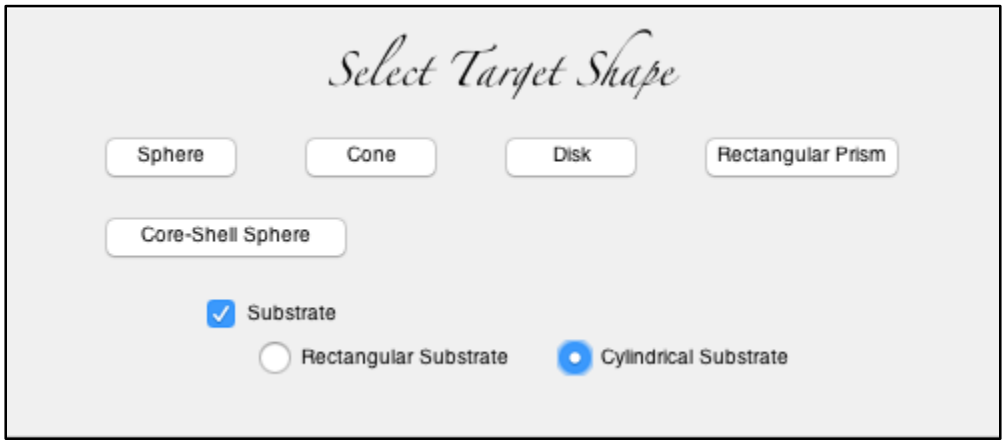

Figure 3: The interface for the nanoHUB tool DDSCAT Shape Generator has multiple screens. This is the first to select the target nanoparticle shape [11].

Parameters to describe the particle and simulation environment were then selected. The individual parameters and the selected values for each are displayed below. The values in Figure 4 were the values used in simulation.

Figure 4 shows the second screen of the nanoHUB tool DDSCAT Shape Generator. Various DDSCAT parameters including shape dimensions and material were selected here [11].

After submitting, the shape generator created two of the four files required to execute a DDSCAT job - the target and parameter file. These files contain the user specified inputs formatted to DDSCAT specifications. Additionally, the tool generated a 3 dimensional plot file of the nanoparticle for visual analysis.

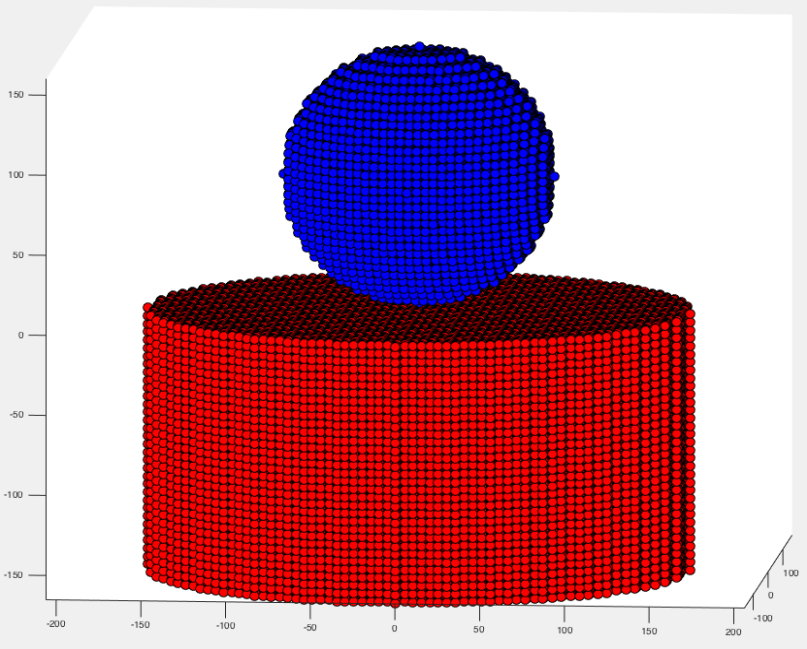

Figure 5 shows the target shape, a nanoparticle atop a substrate, to be submitted as a job. The target is discretized to represent dipoles with blue depicting gold and red depicting Indium Tin Oxide. The cylinder shape was chosen arbitrarily and could be replaced with a rectangular prism.

The dielectric function material file for gold was downloaded separately [12] Using the DDSCAT User Guide, the source code of version 7.3.1 was downloaded and the makefile was edited to enable the software to use MPI functionality. The code was then compiled to create an executable and the four files were finally placed in one folder to be simulated on a supercomputer.

Results & Discussion

The control simulation was uploaded to the University of Arkansas High Performance Computing Resources [13], a Linux environment, and computed with a varying number of processors to verify that enabling MPI parallelization on a singly oriented nanoparticle reduces computation time. The variables changed were range of wavelengths to be simulated and number of processors allocated to each simulation. Each simulation was run 5 times to ensure the precision of results or quantify the run time's level of uncertainty.

The simulation was first run with a fixed number of wavelengths on 1, 2, 4, 8, 16, 32, 64, 128, and 256 processors. Run times are expected to vary more when using fewer processors so more values were taken at smaller resource allocations. When all the cores on a node, similar to a CPU, are not allocated, other users can access the available cores. The node then allocates and deallocates processing power to each core so that the node, rather than individual cores, has the most optimal run time. The node then becomes shared and the run time of one job becomes dependent on another. The fewer data points at larger resource allocations using unshared nodes however are sufficient to approximate the trend between them.

When simulated with 32 and 256 wavelengths, parallelization is expected to decrease run time by nearly half with each data point despite any hardware delays. This pattern is not seen however. The DDSCAT job took an exponentially increasing amount of time to complete between 1 and 32 processors due to a shared node. After this, the trend rose linearly due to communication lag between an increasing number of processors. The spike at 32 processors in the 256 wavelength simulation deviates from the pattern due to 1 of the 5 data points being an outlier due to a shared node. No reduction in computation time is seen.

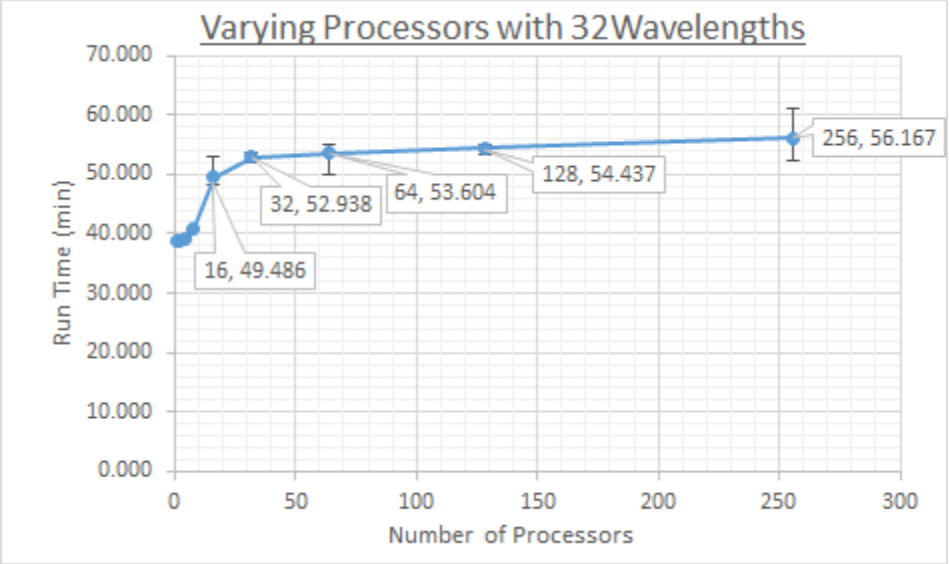

Figure 6: The test simulation was run with 32 wavelengths and an increasing number of processors on the Arkansas High Performance Computing System [13].

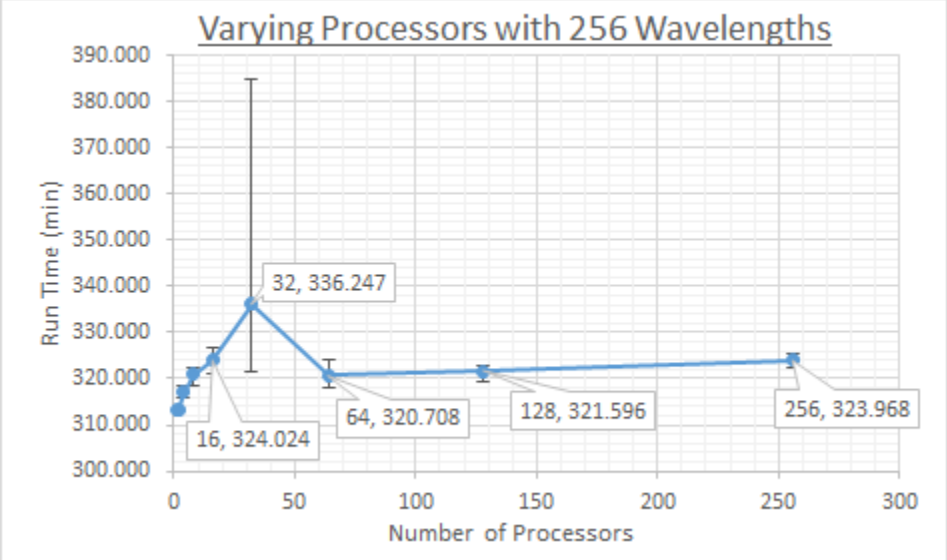

Figure 7: The test simulation was run with 256 wavelengths and an increasing number of processors on the Arkansas High Performance Computing System [13].

Two simulations were run with an increasing number of wavelengths and compared. The first was allocated one processor and the second was allocated as many processors as wavelengths so the ratio between the two variables is always one. If DDSCAT was parallelized by wavelengths, the second graph should be an approximately horizontal with no change in run

time. One wavelength would be computed on one processor at all data points. The first simulation is a control group that shouldn't change whether the software is parallelized or not. The results showed the two simulations to have equivalent graphs. It was determined that MPI enabled DDSCAT provided no parallelization benefit when simulating a target shape of only one orientation.

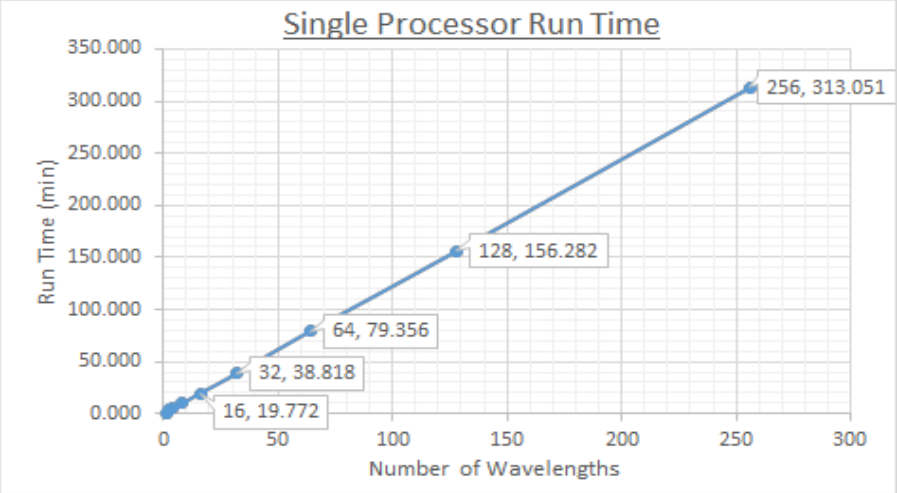

Figure 8: The test simulation was run with a varying number of wavelengths and on a single processor on the Arkansas High Performance Computing System [13].

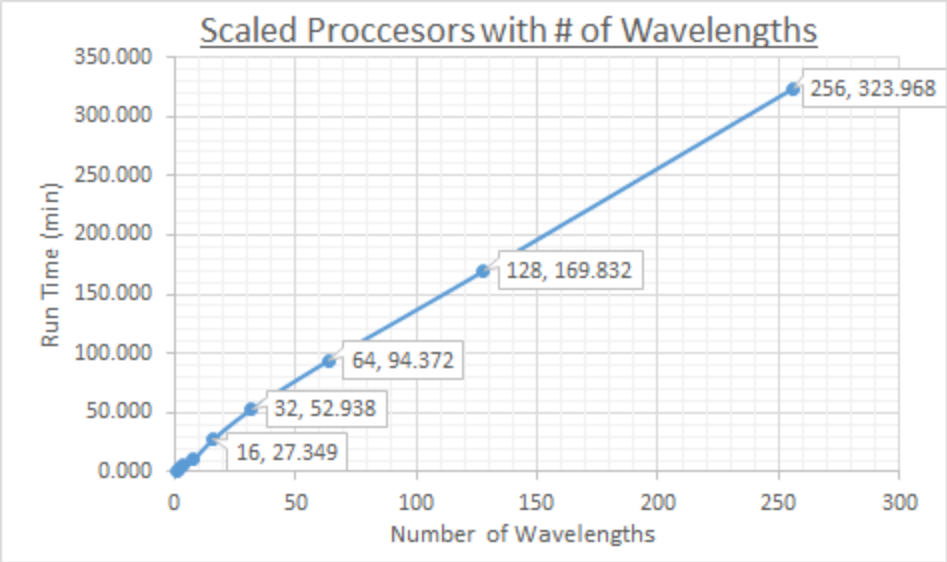

Figure 9: The test simulation was run with a varying number of wavelengths and an equal number of processors on the Arkansas High Performance Computing System [13]. The ratio between wavelengths and processors is always one.

Parallelizing DDSCAT

Programming Methodology (*para_ddscat*)

The parallelizability of a program is dependent on many factors and often sets the performance limit more than the hardware being used. *Serial* problems are calculations that are purely dependent on each other. Each part must be computed only a previous set of instructions have been completed. *Embarrassingly parallel* problems then are calculations that can be done simultaneously with little to no manipulation of the current solution methodology. The problem contains parts that can be computed independently without requiring information from another calculation to move forward. Discrete Fourier Transform, brute force attacks in cryptography, and BLAST searches in bioinformatics are common examples.

Often however, a problem contains parts that are parallel, serial, and a combination of both with some interdependencies. Starting multiple threads from a single thread occurs without issue but when one or multiple threads must wait on another thread's output, synchronization checks must be put in place to allow a thread to know the status of other threads.

DDSCAT was determined to be parallelizable by wavelength computation. The program performs a set of calculations to determine optical efficiency of the target particle at each wavelength. The calculations at each wavelength are independent of each other but are currently computed in serial.

A script titled *para_ddscat* was written to optimize computation time by dividing each wavelength-specific calculation amongst multiple processors and multiple nodes. The script was written in bash because most supercomputers run Linux operating systems. The program is equipped with a minimal interface that prevents users from submitting erroneous simulations to supercomputing resources. It first requires four arguments to run - the titles of the four necessary

files to simulate DDSCAT. It then searches for these files in the current directory or in the path specified by the user. If the files are not found, *para_ddscat* fails the program quickly and returns an error message before the job is submitted to explain which file couldn't be located.

The script then calculates the total number of processors and nodes available to it respectively by calculating the number of lines, and then unique lines, in the environment variable $PBS_NODEFILE. Processors per node, ppn, is then calculated and saved. The parameter file is parsed to initialize the number of wavelengths, x, specified by the user. The submitted job is then divided into j number of smaller jobs that can be submitted in parallel amongst the available processors using the following equation:

$$j = \frac{x}{ppn * n} \tag{1}$$

For example, if 2 nodes with 15 processors each are available to simulate 300 wavelengths, the job will be divided so 10 wavelengths are run on each processor.

This equation doesn't work in all circumstances however. It is often the case that the number of wavelengths to be simulated is not a multiple of the available processors, leaving a remaining number of wavelengths, r, to not be simulated. The solution to this situation must, as before, force each processor to have the least number of wavelengths submitted to it. So, the minimum computation time is achieved by submitting one additional wavelength to $r$processors. The number of processors that must compute additional wavelengths is irrelevant because the maximum number of wavelengths to be computed by any one processor is the only limiting factor.

Subdirectories are then created within the original simulation folder with links to the original material and target files and copies of the parameter file that have j wavelengths to

simulate (links rather than copies of the material and target files were used to save disk space). Figure 1 expresses this directory tree as a flow chart.

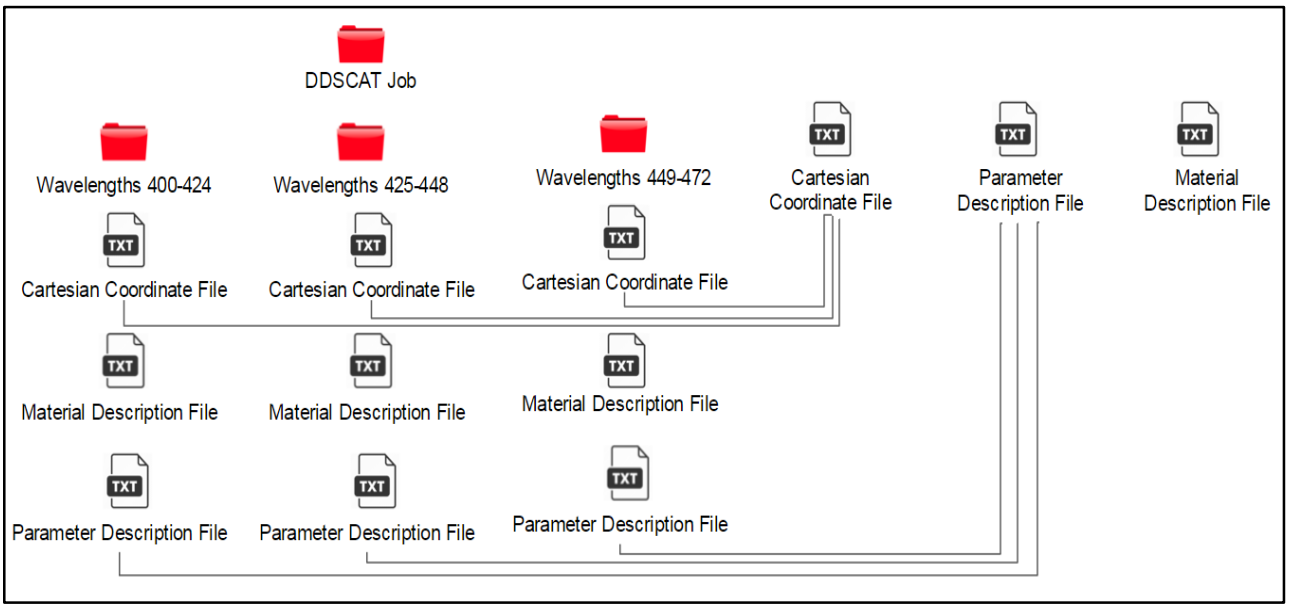

Figure 10 is a graphical representation of the directory layout that *para_ddscat* creates during its function. The user creates the folder, DDSCAT job, and the 4 files on the left, and submits *para_ddscat*. From there, *para_ddscat* divides the submission into smaller simulations and creates n subdirectories for them. It then executes DDSCAT n number of times directing each execution to operate on one subdirectory.

The program then connects via secure shell to each computation node and runs unique DDSCAT simulations that operate on each subdirectory. Each folder represents the simulation occurring on each processor. Once all simulations have finished, the data from each simulation's output files are compiled into one file with all subdirectories deleted. This method is expected to reduce computation time by the number of processors available to it.

.

Results & Discussion

The *para_ddscat* script was uploaded to a simulation folder in the University of Arkansas High Performance Computing Resources [13] and computed with increasing number of processors and an equivalent number of wavelengths so the ratio between the two variables is always one. Results showed that though the number of wavelengths to simulate increased, the run time trend maintained an almost horizontal slope. Slight deviations were due to hardware limitations. The exponential trend before 16 processors is caused by a shared node and the linear slope after 16 is due to communication delay caused by an increasing number of processors.

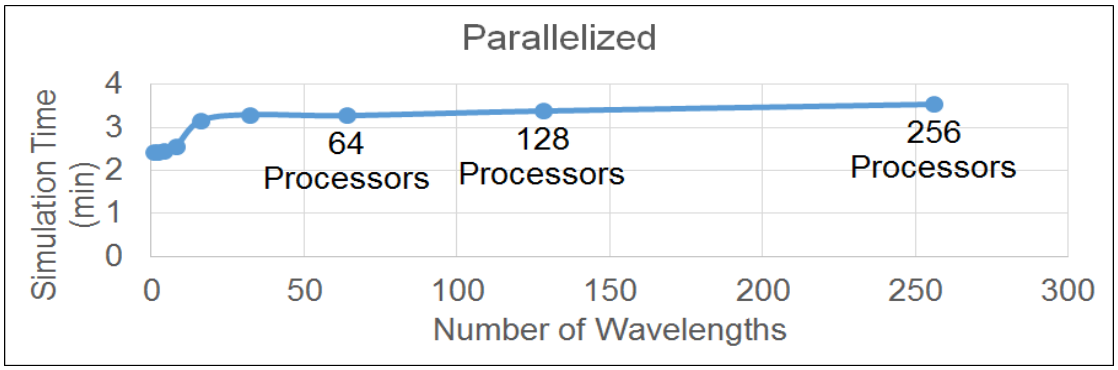

Figure 11: A DDSCAT simulation was submitted using *para_ddscat* to quantify the effects of parallelization. The results show a trend line with an approximately horizontal slope.

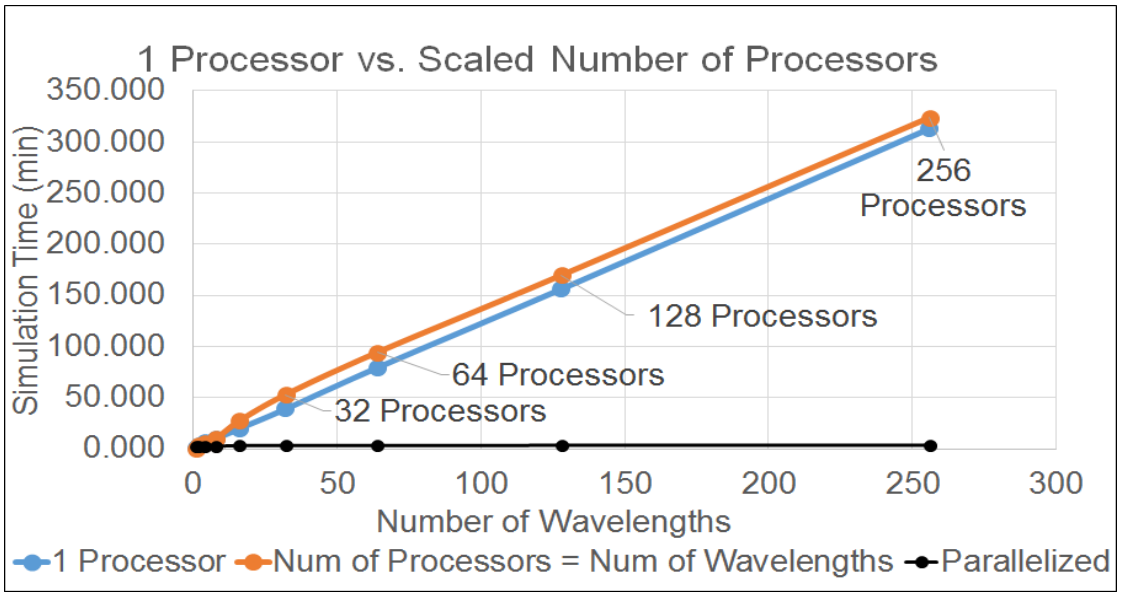

Figure 12 is an overlay of Figures 8, 9, and 11 to show the improvement in run time between the parallelized and serial program.

Conclusion

DDSCAT simulations were run with a singly oriented target shape and with a varying number of processors, and it was proven that the software is parallelized only according to particle orientations. Simulations took hours to complete and enabling MPI provided no process acceleration. It was determined that to improve run times, DDSCAT could be parallelized by another factor, wavelengths. To enable this functionality, a bash script was written to divide wavelength computation amongst multiple processors on a high performance computing system. The results showed a drastic improvement in run time proportional to the number of processors allocated to the simulation. The program is expected to be published on the nanoHUB website along with the DDSCAT Shape Generator [11].

References:

1. Draine, B.T., & Flatau, P.J., "Discrete dipole approximation for scattering calculations", J. Opt. Soc. Am. A, 11, 1491-1499 (1994) (We check references to this paper to track the overall DDSCAT use. On occasion we implement code improvements that way!)
2. Draine, B.T., & Flatau, P.J., "Discrete-dipole approximation for periodic targets: theory and tests", J. Opt. Soc. Am. A, 25, 2593-2703 (2008).
3. P. J. Flatau and B. T. Draine, "Fast near field calculations in the discrete dipole approximation for regular rectilinear grids," Opt. Express 20, 1247-1252 (2012)
4. Graham, T. (2015, July 9). What is Moore's Law? Retrieved May 06, 2016, from http://www.extremetech.com/extreme/210872-extremetech-explains-what-is-moores-law
5. Graham, T. (2016, February 29). Here's why we don't have light-based computing just yet. Retrieved May 06, 2016, from http://www.extremetech.com/extreme/223671-heres-why-we-dont-have-light-based-computing-just-yet
6. Byrne, C. (2015, March 13). The Golden Age Of Quantum Computing Is Upon Us Retrieved May 06, 2016, from http://www.fastcompany.com/3045708/big-tiny-problems-for-quantum-computing
7. Krewell, K. (2009, December 16). What's the Difference Between a CPU and a GPU? | The Official NVIDIA Blog. Retrieved May 06, 2016, from Che, S., Li, J., Sheaffer, J. W., Skadron, K., & Lach, J. (2008). Accelerating Compute-Intensive Applications with GPUs and FPGAs. *2008 Symposium on Application Specific Processors*, 101-107. doi:10.1109/sasp.2008.4570793

8. Barney, B. (2016, May 5). Message Passing Interface (MPI). Retrieved May 06, 2016, from https://computing.llnl.gov/tutorials/mpi/#References Lawrence Livermore National Laboratory
9. Dagum, L. (1997, November 5). What is OpenMP? Retrieved May 06, 2016, from http://www.openmp.org/mp-documents/paper/node3.html
10. Using OpenMP with MPI. (2016, January 4). Retrieved May 06, 2016, from http://www.nersc.gov/users/computational-systems/retired-systems/hopper/running-jobs/using-openmp-with-mpi/ National Energy Research Scientific Computing Center
11. Manoj Seeram; Gregory T. Forcherio; donald keith roper (2016), "Shape Generator for the DDSCAT software," https://nanohub.org/resources/ddscatshapegene. (DOI: 10.4231/D3J960B3D).
12. P. B. Johnson and R. W. Christy. Optical Constants of the Noble Metals, *Phys. Rev. B* **6**, 4370-4379 (1972)
13. Chaffin, D., Pummill, J., & Wolinski, P. (n.d.). AHPCC Arkansas High Performance Computing Center. Retrieved May 06, 2016, from http://hpc.uark.edu/hpc-resources/index.php
14. Blitzen, J. (2016, April 29). *Lecture 2: Story Proofs, Axioms of Probability*. Lecture presented at Statistics 110 Lecture in Harvard University.
15. G.T. Forcherio, P. Blake, M. Seeram, D. DeJarnette, D.K. Roper, “Coupled dipole plasmonics of nanoantennas in discontinuous, complex dielectric environments,” *J. Quant. Spectrosc. Radiat. Transfer* (2015), 166, 93-101.

Acknowledgments

This work would not have been possible without the support and guidance of my mentor, Gregory T. Forcherio, PhD candidate, and my research adviser, Dr. Keith Roper.

This work was supported in part by National Science Foundation grants ARI #0963249, MRI #0959124, EPS #0918970, a grant from the Arkansas Science and Technology Authority, and support from the U of A Office of the Vice Provost for Research and Economic Development. These resources are managed by the Arkansas High Performance Computing Center.

Figures 6-9 were generated by Joseph Hill, PhD candidate, in Dr. Paul Millett's Computation Materials Science Research Group at the University of Arkansas. The simulations to generate the graphs were run on the Millet Research Group supercomputing resources, sponsored by the National Science Foundation, Department of Energy, and the University of Arkansas.