Development of a Parallel Scene Generation Electromagnetic Modeling Tool

Christopher Card
Black River Systems Co,
Utica, New York 13502
card@brsc.com

Abstract:

This paper presents the results to date of an on-going development effort to provide a scalable, portable, parallel scene generation tool that will provide the capability to rapidly generate scenes of radiating and scattering structures in realistically complex electromagnetic environments. The benefit of such a tool is that it will provide users with the capability to solve large problems that cannot be currently solved with existing sequential electromagnetic modeling tools, This tool supports a broad range of users including researchers, algorithm developers, analysts, and system developers. This paper will present the parallelization process highlighting the strategies used and will show the results to date.

The project presented here is the parallelization of WIPL-D, an electromagnetic modeling tool. Through parallelization, the well known and commercially available tool will become faster and possess increased capabilities. This paper will walk you through the parallelization process, providing the strategies used and the results received.

Keywords: WIPL-DP, WIPL-D, Electromagnetic Modeling, Parallel Processing, CHSSI

1. Background

The parallel electromagnetic modeling tool (WIPL-DP) being developed under this effort is leveraging a proven commercially successful sequential electromagnetic modeling and scene generation tool called WIPL-D (Wires, Plates and Dielectrics). WIPL-D has evolved from over ten years of research in numerical electromagnetics. The demand for analysis of composite conducting and dielectric structures necessitated the development of such a tool. WIPL-D uses Method of Moments to solve Maxwell's Equations. It uses a bilinear quadrilateral domain technique, which reduces the number of unknowns for large scale simulation problems. WIPL-D was developed by Dr. Tapan K. Sarkar from Syracuse University and Dr. Branko M. Kolundžija from the University of Belgrade.

This WIPL-DP effort is being funded under the Common High Performance Computing Software Support Initiative (CHSSI). The goal of this initiative is to provide efficient, scalable, portable software codes, algorithms, tools, models and simulations that can run on a variety of DOD High Performance Computing platforms that can be used by scientists and engineers to solve computing problems. For more information on the CHSSI initiative refer to http://www.hpcmo.hpc.mil/Htdocs/CHSSI/. Through the CHSSI initiative, WIPL-D is being parallelized so that it can meet certain pre-determined requirements, namely portability, scalability, and accuracy. For portability, the parallelized code must run on various High Performance Computers located at multiple Department of Defense sites. For scalability, the code running on multiple processors must meet a required speedup level when compared to a single processor run. For accuracy, the parallelized code must have accuracy within a certain percent when compared to the original commercial version of WIPL-D.

This is a three year development effort with four major milestone reviews. We are currently starting the third year of the effort and have two major milestones left to complete. The four major milestone reviews are used to determine if the project is meeting the guidelines set forth by the CHSSI program and, based on this, a decision is made whether or not to continue the out year funding. The past, present, and future years will be discussed in terms of what was or is to be accomplished, along with the CHSSI requirements for that year.

2. WIPL-DP Development First Year

The first year of the effort had two main goals. These goals were to increase the code's portability and to determine the strategies for parallelization. The results were presented at the CHSSI SAT review, along with baseline timings for a selected target demonstration. There were three specific critical test parameters that must be achieved. A speed baseline was to be established for a target demonstration running on a single processor. The code was to be run on one High Performance Computing platform producing valid results. Optimally, the single processor version of the code was to produce an accuracy of 6% error (8% as a minimum) when compared to the commercial version of WIPL-D. The accuracy is determined by comparing the output files from the parallel version to the output files from the commercial version. Matlab scripts were used to calculate the percent error between the values in the corresponding files.

The first goal required providing a parallel version of the code that was more portable, To achieve this, the original WIPL-D FORTRAN version of the code was converted into the C language. The conversion process began with using a FORTRAN conversion tool called f2c which took the original WIPL-D FORTRAN code and converted it to C code. Using this tool saved a great deal of time but did not produce the most efficient C code so some hand tuning was required. To attain higher portability, the dependencies on the f2c libraries were removed, eliminating the need for them to be present or installed on the target machines. First, all of the file and standard input and output functions from f2c were replaced by C's stdio functions. Also, there were data structures in the f2c libraries that were used by the converted code. When they could not be replaced by C data structures, corresponding sections of the f2c library were imported into the new WIPL-D version. Next, the common blocks used by FORTRAN were pulled out and placed into header files. Also performed was some general cleanup of f2c generated white space in the files, providing code that is easier to read and understand. As required by the government, a common header was placed on each file signifying that the code is export controlled.

The second goal was to determine parallelization strategies for WIPL-D. These strategies were defined in a parallelization requirements document. To see which areas of the code would benefit from parallelization, the GNU profiling tool, gprof, was used on several machines running the C version of WIPL-D. The computers that were used were two Linux machines – Huinalu at the Maui High Performance Computing center which contains 933MHz Pentium III processors and a 1.0 GHz Pentium III running RedHat 7.2, a Solaris Sun Blade running Solaris 5.8, and two Windows machines (98 and XP) with Cygwin (a Windows utility for Unix simulation). Various demonstration applications, along with a larger project application, were profiled on these machines. The demonstrations used were the ones that are provided in the WIPL-D commercial version and the larger project application was a simulation of a Mirage aircraft.

The profiling data obtained from the above machines was used to determine candidates for parallelism. A compilation of the high usage functions from the demonstrations and the Mirage project is shown in Table 1 below.

	Demos	Mirage Project
Main	|	|
|→Impedance Matrix Construction (racun_)	|→75%	|→20%
| |→Initial Impedance Calculations (simpi_)	| |→65%	| |→12%
| | ↳Calculate Surface Potential (spint_)	| | ↳40%	| | ↳9%
| |→(sipak_)	| |→3%	| |→3%
| |→(gepak_)	| |→3%	| |→1%
| ↳(sempi_)	| ↳ 3%	| ↳ 1%
|→Solution to System of Equations (smqcc_)	|→20%	|→75%
|→Far Field Calculations (farfi_)*	|→29%	|→2%
↳ Near Field Calculations (nearfi_)*	↳ 11%	↳ 3%
Profiler Overhead	3%	3%

Table 1 – Profiling Data

*The Far Field and Near Field Calculations occur only if the user selects near field and radiation patterns.

The data identifies some good candidates for parallelism, which fall into two main categories. First, there are functions that take a large percentage of time themselves. Second, there are functions that consume a small amount of time themselves, but have many child functions that take a large percentage of time. The desired strategy for the first category of functions would be to internally parallelize the computations inside the function. For the second category the strategy would be to parallelize the calls from the function to its child functions. There is also a sub-category of functions that are not good candidates in themselves. These are functions that take a small amount of time but are called a very large number of times. Since they are too small to be parallelized internally, higher level functions can distribute their calls to these functions. This will be accounted for by the second category described above.

Table 1 shows the call tree that the candidate functions follow. For each function the average processor usage percentage is shown. The percentage of time at each level is the accumulated time of the function plus its descendants. Table 1 shows that the Impedance Matrix Construction could benefit from parallelization by distributing its child function calls. Almost all of its time is taken up by the children processes. The simpi_ and spint_ functions consume a lot of time due to being called many times. Parallelizing them internally would not be very beneficial since their self time per call is very small. The Solution to the Systems of Equations is a function that would greatly benefit from internal parallelization. The amount of self time it takes for each call is very large, which increases with the size of the problem. The Far Field and Near Field Calculations are also functions that can benefit from internal parallelization. They may not always be present though, since they are dependent upon user input. Their percentages of time are based upon when they are used in the project, which is the reason for the total percentage of time being over 100. As can be seen from Table 1, for the smaller projects the Impedance Matrix Construction dominates processor usage, and as the project size increases the Solution to the System of Equations takes over. The parallelization strategies for these candidates will be described in depth in the following sections.

The High Performance Computer that was used to get the baseline timing and accuracy was a Linux Cluster named Huinalu at the Maui High Performance Computing Center (MHPCC). The demonstration application that was chosen was a simulation of a cell phone alongside a dielectric head, as seen in Figure 1. This application was a modified version of Demo-531, a demonstration provided in the commercial version of WIPL-D. Modifications were made to this demonstration to increase the problem size and coverage, allowing for more thorough testing of the software. The demonstration application was run over

two frequencies (0.9 GHz and 2.4 GHz) and contained 3549 unknowns. The baseline timing that was received was 5,793 seconds. The worst-case accuracy was 0.04%, which met the optimum timing objective.

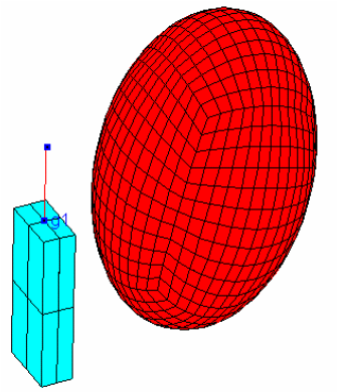

Figure 1 – Cell Phone Alongside a Dielectric Ovoid

3. WIPL-DP Development Second Year

The goal of the second year of this effort was to achieve scaled speedup through parallelization at the frequency level. The results for the second milestone were presented at the CHSSI Alpha Test. The three specific critical test parameters for this milestone were as follows. Optimally, the scaled speedup was to exceed 80% (25% as a minimum) of the single processor C version on 32 nodes (16 nodes as a minimum). The code was to be run on two High Performance Computing platforms producing valid results. The single processor version of the code was to produce an optimal accuracy of 4% (5% minimum) when compared to the commercial version of WIPL-D. The calculation of the speedup is found by taking the execution time on one processor and dividing it by the time it takes to run on multiple processors.

The frequency loop is the main loop of the WIPL-D program and offers a high level of parallelism. The calculations for the iterations of the loop are independent from one another. After acquiring the user inputs, the frequency loop will perform iterations for each frequency present in the simulation. The approach that was chosen was to distribute the number of frequencies among the available processors at run time. When the program arrives at the frequency loop, calculations are performed to determine what frequencies every processor is responsible for. Each processor then proceeds to execute its assigned iterations. To eliminate out of order file output, each processor prints to separate output files, with the frequency number appended to the end of the filename. When the frequency loop has completed the root processor collects all of the output files and assembles them as they would appear if the project was run using a single processor. This method has been implemented and tested with a target demonstration on the target machines identified below. It has had the desired impact by dividing the processing time by the number of frequencies, providing the necessary scaled speedup. The results obtained were above the optimal defined requirements.

The High Performance Computers that were chosen to run on were Huinalu and an IBM SP3 named Tempest at the Maui High Performance Computing Center. The demonstration application that was used was again the cell phone alongside a dielectric ovoid, as described in the previous section and shown in Figure 1. It was run on both machines using 1, 2, 4, 8, 16, and 32 processors, with two frequencies per processor. The frequency range was 0.9 GHz to 2.4 GHz. The results presented at the second milestone (Alpha Test) were above the optimum requirements. The worst case speedup was 88.9% and the worst

case accuracy was 0.08% error, which both met the optimum objectives. A graph of the speedup based on the number of processors can be seen in Figure 2.

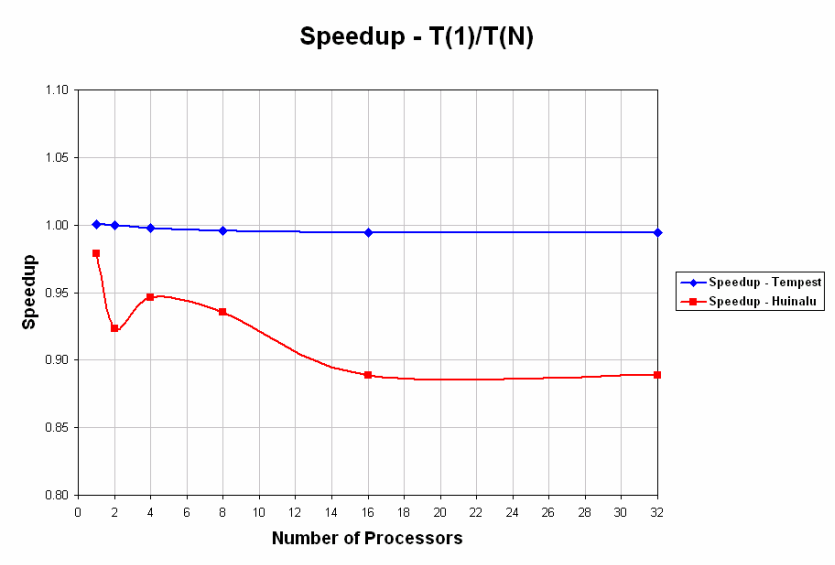

Figure 2 – Speedup Results for Frequency Parallelization

During this second year work was also performed on parallelizing the impedance matrix solution using the Scalapack libraries. The details on this will be presented in the next section.

4. WIPL-DP Development Third Year

Currently we are in the third and final year of the WIPL-DP development effort. The goal for this year is to further increase the performance of WIPL-DP by decreasing run time and adding additional functionality. This will be accomplished by parallelizing the impedance matrix generation and solution. The results will be presented at the Beta Test milestone which is currently scheduled for April 2004. The three specific critical test parameters that must be accomplished are as follows. The scaled speedup is to exceed 80% (25% as a minimum) of the single processor C version on 64 nodes (32 nodes as a minimum). The code is to be run on three High Performance Computing platforms (two as a minimum) producing valid results. The single processor version of the code is to produce an optimal accuracy of 2% (3% minimum) when compared to the commercial version of WIPL-D.

The parallelization of the impedance matrix generation will offer increased speedup, but more importantly it will provide increased functionality. By distributing the matrix, a greater number of unknowns will be able to be used, thus allowing for larger simulations to be solved. The current goal is to reach 100,000 unknowns. The impedance matrix is made up of complex values and has the dimensions of number of unknowns by number of unknowns. Each processor performing the generation will hold a fraction of the complete matrix. The number of processors used will be determined by dividing the number of unknowns for the project by the number of unknowns that can be held in a single processor's memory. A processor can typically address around 12,000 unknowns if it has sufficient memory. The matrix is constructed by looping over the number of elements (number of plates plus number of wires) and performing impedance calculations. There is an outer and inner loop that both cycle over the number of elements. Each iteration of the outer loop determines the columns to be calculated. The inner loop then goes through and performs calculations and accumulates values in the rows of those columns. The outer loop calculations are independent of each other, except for at the end of the inner loop where the accumulations take place. The

outer loop can be distributed, performing all calculations prior to the accumulations. Each processor entering the loop will be assigned iterations to be performed and execute them in parallel. Then results prior to the accumulation will then be saved off, and the accumulation will be done separately afterward. The results will be saved into an array that holds the indexes and the corresponding values to be added to them. To prevent this array from becoming too large, at the end of each iteration every processor will perform the accumulations through communication with the other processors. Each processor determines where an index of the matrix is held and provides the accumulation data to the corresponding processor. Parallelizing this loop can have great benefits since all of the impedance calculations (the child functions of racun_ from Table 1) are contained inside this loop. All of the calculations in the matrix construction would be distributed over the number of processors performing the generation, providing scaled speedup.

The solving of the impedance matrix is the major bottleneck of the program. As seen in Table 1, this portion of the code can take around 75% of the processing time for larger simulations. Through parallelization the solution should take far less time, greatly decreasing the total processing time. Solving the system of equations is currently done using LU Decomposition. The separation of the impedance matrix into a lower and upper triangular matrix takes the bulk of the solution time. In the commercial version solution the order of execution creates dependencies between each iteration of the main loop. Each time through the loop, this implementation will go through and calculate a column for the lower matrix and then compute all of the recalculations for a single row (column number +1) for the upper matrix. This produces dependencies on all previously calculated columns and the previous rows.

In order to parallelize this solution to the system of equations, the section of code that performs the current matrix solution must be replaced with a parallel implementation. To achieve this task the impedance matrix and the vector holding the x^0 terms must be passed into a parallel solution function. The impedance matrix is arranged as shown in Figure 3, where **a** is the real component and **b** is the imaginary component of each impedance value, and n is the number of unknowns.

$$\begin{bmatrix} a_{11} & b_{11} & a_{12} & b_{12} & \dots & a_{1n} & b_{1n} \\ a_{21} & b_{21} & a_{22} & b_{22} & \dots & a_{2n} & b_{2n} \\ \vdots & & & & & & \\ a_{n1} & b_{n1} & a_{n2} & b_{n2} & \dots & a_{nn} & b_{nn} \end{bmatrix}$$

Figure 3 – Impedance Matrix Organization

What you get is a 2n x n array. This array is stored in memory as a linear vector that holds these values in row-by-row order. This can be easily placed in a form that is needed by a parallel implementation and then the solution can be transformed back to the original form for WIPL-D to use.

The parallel complex LU Decomposition function that is currently implemented is contained in the Scalapack library, namely pzgetrf(). It handles the decomposition by distributing blocks of data among processors in a 2-Dimension block-cyclic method, as shown in Figure 4.

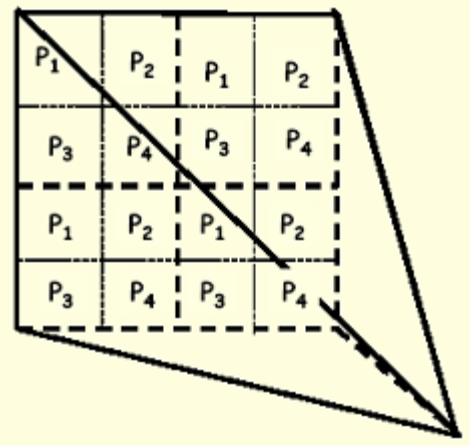

Figure 4 – 2-Dimension Block Cyclic Matrix
Distribution Over 4 Processors

The processors that are used for the solution are arranged in a rectangular grid. For each diagonal process, the corresponding column of processes finds the maximum value in each column, applies row interchanges, and calculates the lower matrix columns. The lower matrix values from the diagonal process are broadcast to the corresponding row of processes for them to calculate the upper matrix rows. All of the newly calculated values and pivot information are broadcast to processes of the remaining sub-matrix. This procedure continues for each process along the diagonal. This distribution of data provides for good load balancing and cuts down on the amount of broadcasting, since the same processor holds multiple sections for each row and column. The optimum number of processors for the solution that is currently being used is the number of unknowns divided by 1,000,000. Optimal processor grid sizes and data block sizes are still being determined.

The Scalapack parallel LU Decomposition function was chosen because Scalapack is widely available and free, providing high portability. It is not the most efficient parallel matrix solver out there though. Alternative parallel solvers may be used in place of the Scalapack version. The code was left general enough so that different parallel matrix solving functions may be substituted in to perform the impedance matrix solution.

Currently the High Performance Computers that have been selected to meet the requirements set forth by CHSSI are Huinalu and Tempest at the Maui High Performance Computing Center and a Compaq SC 40/45 at the Aeronautical Systems Center Major Shared Resource Center (ASC MSRC). Since this work is still ongoing, no real results for the additional parallelization have been received yet. By April of 2004 the results for the target demonstration, along with selected Tri-Service applications, will be available. The target demonstration will be the previously described cell phone alongside a dielectric ovoid with additional modifications that are yet to be determined.

5. Future Developments

The main goal for the final milestone of this development effort will be to move the code to an embedded High Performance Computer. Some additional parallelization will also take place to accomplish this. Again there are three specific critical test parameters. The parallel code must provide scaled speedup that will exceed 80% (25% as a minimum) of the single processor C version on 128 nodes (64 nodes as a minimum). The code is to be run on four High Performance Computing platforms (three as a minimum) producing valid results. The single processor version of the code is to produce an accuracy of 1% when compared to the commercial version of WIPL-D.

The additional parallelization that will be required deals with distributing the Far Field and Near Field calculations. This will be beneficial to simulations that contain near field and radiation patterns with a large number of points. Both the Far Field and Near Field functions contain simple, independent calculations. For the Far Field, the Phi (outer) and Theta (inner) direction loops possess independent iterations and can thus be distributed. For the Near Field, the X, Y, and Z direction loops also contain independent iterations and can be distributed. In both cases there is output file printing intertwined in the loops that must be removed and performed separately.

The High Performance Computers that will be used are the ones described previously along with an embedded machine. The embedded machine that is to be used is yet to be determined.

After all of the parallelization described in this paper has been completed, the profiling tool will again be used to determine if any other areas now present themselves as candidates for parallelism. If any exist they will be looked into and parallelization strategies will be developed as necessary.

6. Conclusion

WIPL-DP will provide users with an electromagnetic simulation modeling tool that can solve larger problems quickly and with increased capability. Applications will be able to be run in a timelier manner and not be constrained by the limits on the number of unknowns present in the commercial WIPL-D. There are a number of Tri-Service applications that can benefit from such a tool. These applications are Foliage Penetration, Synthetic Aperture Radar, Landmine Detection, Re-Entry Vehicles with Chaff, and Antennas on Large Structures.

Upon passing all of the CHSSI reviews with the set-forth requirements met, WIPL-DP will be installed on several Department of Defense High Performance Computers.

