01 Aug 2004

Static and Quasi-Dynamic Load Balancing in Parallel FDTD Codes for Signal Integrity, Power Integrity, and Packaging Applications

Sarah A. Seguin

Michael A. Cracraft

James L. Drewniak
Missouri University of Science and Technology, drewniak@mst.edu

Recommended Citation

S. A. Seguin et al., "Static and Quasi-Dynamic Load Balancing in Parallel FDTD Codes for Signal Integrity, Power Integrity, and Packaging Applications," *Proceedings of the IEEE International Symposium on Electromagnetic Compatibility (2004, Santa Clara, CA)*, vol. 1, pp. 107-112, Institute of Electrical and Electronics Engineers (IEEE), Aug 2004.
The definitive version is available at https://doi.org/10.1109/ISEMC.2004.1350006

Static and Quasi-Dynamic Load Balancing in Parallel FDTD Codes for Signal Integrity, Power Integrity, and Packaging Applications

Sarah A. Seguin, Michael A. Cracraft, James L. Drewniak
Department of Electrical and
Computer Engineering
University of Missouri–Rolla
Rolla, MO 65409

Abstract— **The Finite-Difference Time-Domain (FDTD) method is a robust technique for calculating electromagnetic fields, but practical problems, involving complex or large geometries, can require a long time to calculate on any one single-processor computer. One computer with many processors or many single-processor computers can reduce the computation time. However, some FDTD cell types, e.g., PML cells, require more computation time than others. Thus, the size and shape of the individual process allocations can significantly influence the computation time. This paper addresses these *load balancing* issues with *static* and *quasi-dynamic* approaches. The Message-Passing Interface (MPI) library is applied to a three-dimensional (3D) FDTD code. Timing results including *speedup* and *efficiency*, are presented for trials run on a cluster of sixteen processing, nodes and one server node.**

Two examples are shown in this paper, a power bus with 16 decoupling capacitors and a five layer power distribution network. In such models, the problem size and complexity make modeling with a serial code impractical and time consuming for engineering. Models with several million cells take days to run, but proper implementation, including load balancing, can reduce this execution time to hours on a sufficiently powerful cluster.

I. Introduction

Execution time is decreased with parallel computer codes by distributing the problem over many processes. Ideally, the calculation time for a parallel code is proportional to $\frac{1}{N}$, where N is the number of processes. However, communications and poor problem distributions increase calculation time. This increase constitutes a loss of efficiency. For parallel codes the efficiency [1] is defined as

$$E(N) = \frac{T_1}{T_N N}.$$

T_1 is the time required to calculate the problem, using a serial code or the parallel code run on a single process. T_N is the time required to calculate the problem using N processes. A second measure for parallel codes is speedup [1], defined as

$$S(N) = \frac{T_1}{T_N}.$$

Of the two problems decreasing efficiency mentioned above, communications are an essential part of the message passing scheme and can be streamlined but not removed. However, the problem distribution can be improved with proper load balancing, which can increase the efficiency of the code significantly. Assume that each process depends on communications from the other processes during the calculation. Then, the overall calculation time is dominated by the slowest process. If the problem is improperly distributed, some processes will be more stressed than others. These stressed processors hold up the entire calculation. One approach is found in seeking a better load balancing scheme. This paper will describe methods for improving the load balancing for FDTD codes.

The parallel 3D code used in this paper, originally developed in [2], to simulate the models uses the Message-Passing Interface (MPI) for the distribution of processes [3]. MPI is a library of functions written for Fortran, C, and C++, and is not a programming language itself. MPI was designed with portability as the primary consideration. A program written using MPI will run on any system with a working MPI implementation. It provides a medium for programs to exchange data between processes, i.e., functions to arrange processes and functions to send and receive data. Through MPI functions, a process can find out who it is, also known as its rank, and the total number of processes that are present. It is possible to write efficient parallel programs with as few as six MPI functions.

Load balancing concerns how the problem is divided and distributed and can be split into two general categories: static load balancing and dynamic load balancing [4]. In static load balancing, prior to the calculation, some criteria are used to determine how much of the problem each process should be given. In dynamic load balancing, the problem is broken into work units that are sent to each process on request during the calculation.

FDTD is naturally a data-parallel algorithm, and it lends itself well to a static load balancing scheme. In a data-parallel algorithm, the model space is divided among the processes. In this paper, a weighting scheme is used to determine how many cells each process is given. As an example, calculation times of perfectly matched layer (PML) cells are compared with the calculation times of free space cells. The ratio of those times is defined as the cell weight for a PML cell. PML cells are

0-7803-8443-1/04/$20.00 © IEEE.

computationally intensive compared as with free space cells, and processes spatially located at the edge of the model space will have more PML cells than the interior processes. Thus, the edge processes will take longer to calculate than the other processes, slowing the entire calculation down.

Load balancing attempts to alleviate imbalances, which are the major impediment to the efficiency of the parallel algorithm. Static load balancing divides the model at the beginning of the program to minimize differences in the process loads. Then, processes work on their portion of the model space for the duration of the program. Any imbalances in the load balancing will cause delays in the program execution. Dynamic load balancing attemps to break up a problem into work units and distributes these work units as processes request them. This type of load balancing requires more communication between the processes and, thus, delays may be incurred if the server process is mired in multiple requests at once or by the communication speed of the network. Though dynamic load balancing remains the most efficient.

Although FDTD is not suited to a dynamic load balancing approach, a quasi-dynamic load balancing approach is used for the models in this paper that may fix the same problems that a dynamic approach fixes, only more conservatively. The quasi-dynamic load balancing builds on the static load balancing method with PML weighting, but the code allows the process boundaries to shift to minimize the time per time step for all the processes. Basing the process boundaries on performance during the calculation, can not only fix the distribution problems but also can adjust for differences in processes, i.e., processor clock speed or model.

II. Trials and Results

Two examples were simulated using a parallel FDTD algorithm and compared with a serial algorithm. The results for both of the examples were compared, and provide an example of the advantages of the PML weighting. A power bus with 16 decoupling capacitors, shown in Figs. 1 and 2 was simulated above a ground plane with a dielectric in a parallel 3D FDTD code.

In addition, a five layer power bus, described in Fig. 3, shown in Figs. 4 through 8, was also simulated.

Fig. 1 was discussed in [5], it shows the board configuration. The lossy board dielectric in the test device is FR-4, 65 mil thick. The upper and lower planes are both copper, and an array of sixteen global decoupling capacitors connects to the power plane by wires passing through square via holes.

Cell weighting in this paper refers to load balancing by weighting Perfectly-Matched Layer (PML) cells, which take longer to execute than the cells in the interior of the problem space. This may be ascertained from the complexity of the update equations for the PMLs [6]. By weighting the PML cells the problem may be more efficiently divided and distributed to the processing nodes. Load balancing by weighting is employed by assigning a numerical weight to each cell according to its cell type: interior or PML. The weight for an interior electric field and magnetic field calculation is set

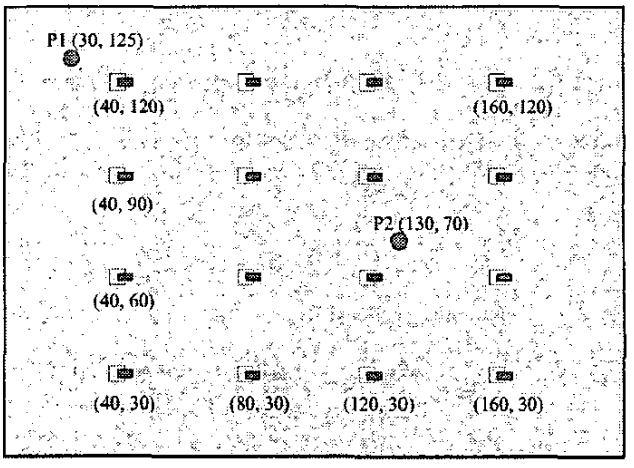


Fig. 1. Top view of the power bus with 16 decoupling capacitors.

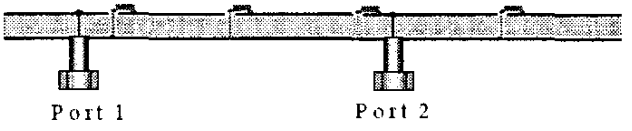
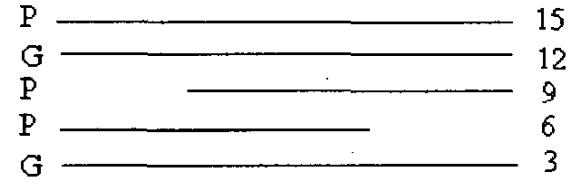


Fig. 2. *Side view of the power bus with 16 decoupling capacitors.*



Layer thickness = 14 mils
Dielectric constant = 4.24, loss tangent = 0.022
dx = dy = 0.5mm, dz = 0.11853mm

Fig. 3. The 5 layer power bus description.

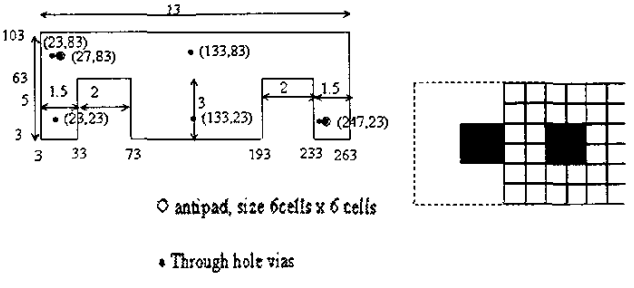


Fig. 4. First layer of the 5 layer power bus.

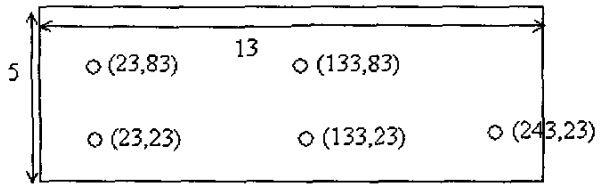


Fig. 5. Second layer of the 5 layer power bus.

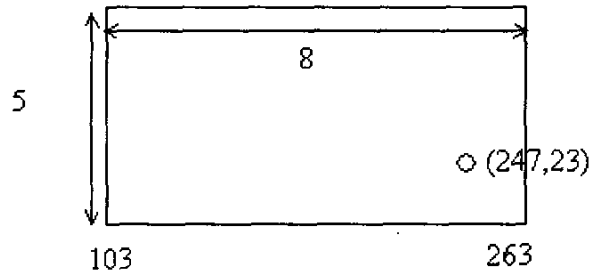


Fig. 6. Third layer of the 5 layer power bus.

to 1. The weights for the electric field PML and the magnetic field PML calculations are then set higher, e.g., 3.5 and 5.3, respecitvely. The break weight is the ideal weight for each process. Ideally, the processes would be equally loaded and there would be no loss in efficiency due to load imbalances. However, it is unlikely that such a breakup is possible.

The objective of this section is to determine how well the PML weighting scheme increases the efficiency of the parallel FDTD program. Two sets of data were computed for both examples. One case with, and one case without the PML weighting were run on two meshes with an increasing number of processors. The two meshes are labeled low density and high density, and are different only in their cell sizes. The cell dimensions of the high density mesh are half that of the low density mesh. Thus, there are eight times the cells in the high density mesh.

The following sections discuss the data collected for each example. Each simulated example was compared and checked against the serial version as well as the measured results. All examples show the average time step, speedup, and efficiency

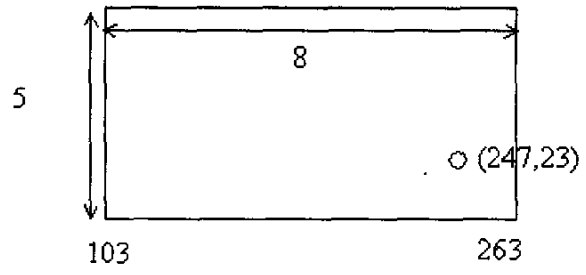


Fig. 7. Fourth layer of the 5 layer power bus.

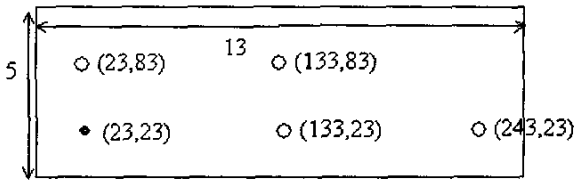


Fig. 8. Fifth layer of the 5 layer power bus.

that were calculated from the results. The number of processes indicated on each graph is the number of worker processes. The total number is the indicated processes plus one, called the server process. In each example, the execution times, the speedup and the efficiency behave approximately as expected when compared with the tehoretical models. Although, the efficiency graph tends to be rather erratic for both examples.

A. Power Bus with 16 Decoupling Capacitors

The measured results and the simulated FDTD results are shown in Figs. 9 and 10. The results for the low and high resolution models are found in Figs. 11 through 13.

Fig. 12 shows the average time steps for the four sets of data collected and their ideal (T_1/N) for the power bus with 16 decoupling capacitors. In particular for the high density model, the unweighted set is asymptotically approaching a time step larger than the weighted version. The weighted set is still decreasing with the ideal even at 16 processor limit of the cluster.

Fig. 13 shows the speedup, which was expressed as the ratio of the single processor execution time to that of N processors. In this figure, the asymptotic limit mentioned for the time steps is even more apparent. With 16 processors the unweighted sets of the two models are already beginning to plateau while the weighted sets are falling away from the linear ideal more gradually.

The efficiency is shown in Fig. 16. As with the speedup, the efficiency is a numerical gage showing how parallelizable the algorithm is. In the plot the weighted data set for both models has the higher efficicency. The efficiency is erratic in appearance, which can be attributed to the setup of the individual cluster, including the machines and networking equipment.

The low density model of the power bus with 16 decoupling capacitors uses 100 by 50 by 20 in the internal region. The PML boundary is 8 cells thick, so there are 100,000 cells in the internal region while there are 175,616 cells in the PML boundaries. The high density model doubles the number cells in each dimension, so there are 200 by 100 by 40 cells in the internal region. Using a 8 cell thick PML boundary, there are 800,000 cells in the internal region and 1,403,136 cells in the PML boundary. Estimate the overall calculation time of a PML cell as $t_{PML} = 8.8t_{FS}$, where t_{FS} is the calculation time for a free-space cell and 8.8 is the addition of the electric and magnetic field weights proposed earlier. Then, the calculation time per time step for each model can

be estimated. The calculation time for the low density model is

$$\begin{aligned} t_{LD} &= (100000)t_{FS} + (175616)t_{PML} \\ &= (100000 + 1545420.8)t_{FS} \\ &= 1645420.8t_{FS} \end{aligned}$$

and the high density model calculation time is

$$\begin{aligned} t_{HD} &= (800000)t_{FS} + (603136)t_{PML} \\ &= (800000 + 5307596.8)t_{FS} \\ &= 6107596.8t_{FS} \end{aligned}$$

The ratio of the two time estimates yields $\frac{t_{HD}}{t_{LD}} = 3.71$. In Fig. 11, the times for one processor yields a ratio of around 2.75, which is less than the estimated ratio but still within a factor of 2 of the estimate.

As more processors are progressively used in the calculations the average time step becomes dominated by communication times between processes and other unparallelizable portions of the program. Therefore, there is a limit to the reduction of calculation time for a particular algorithm. For example, in Fig. 14, the low density models average time step is nearly constant for 12 through 16 processors without using weights. However, the trace for the weighted trials on the same model is still decreasing, which is evidence showing that weighting the PML cells provides for a better balanced problem distribution.

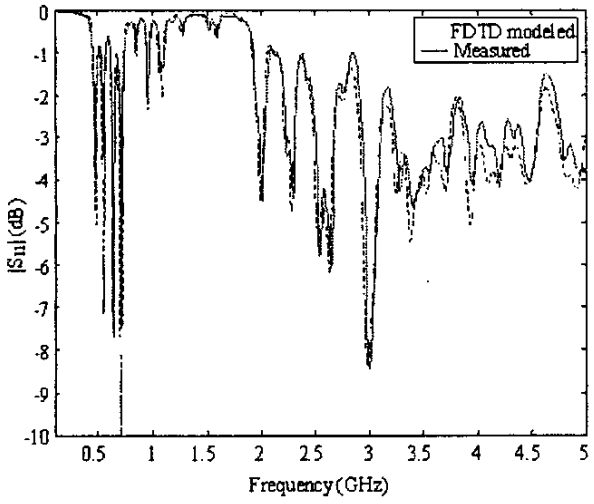


Fig. 9. FDTD modeled and measured $|S_{11}|$ for the power bus with 16 decoupling capacitors.

B. Multi-Layer Power Distribution Network

The results for the multi-layer power bus are similar to that of the power bus with 16 decoupling capacitors, although they are not identical. The timing results are somewhat model dependent, or the results of the two models should be more closely identical. The results for the low and high resolution models are found in Figs. 16 through 18. Fig. 16 shows the

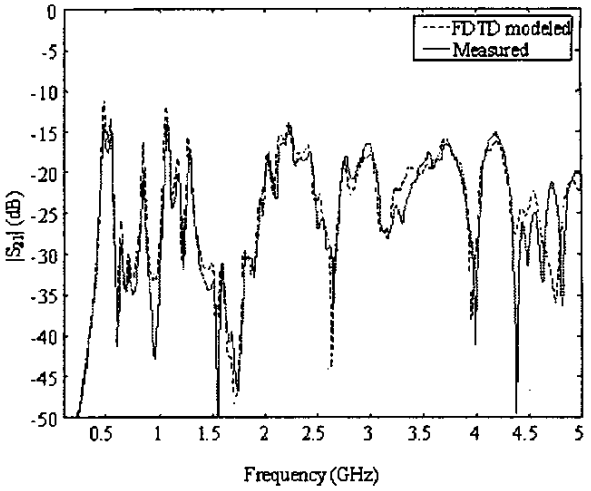


Fig. 10. FDTD Modeled and Measured $|S_{21}|$ for the power bus with 16 decoupling capacitors.

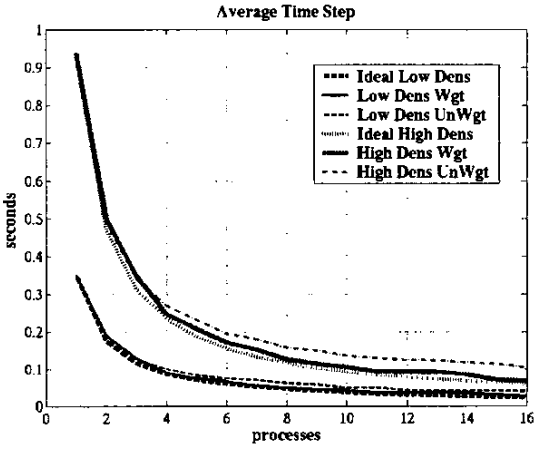


Fig. 11. The average time step for the two sets of data from the high and low resolution models for the power bus with 16 decoupling capacitors.

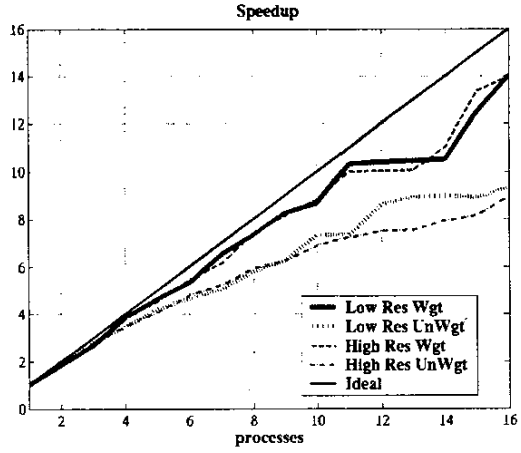


Fig. 12. The speedup for the two sets of data from the two models for the power bus with 16 decoupling capacitors.

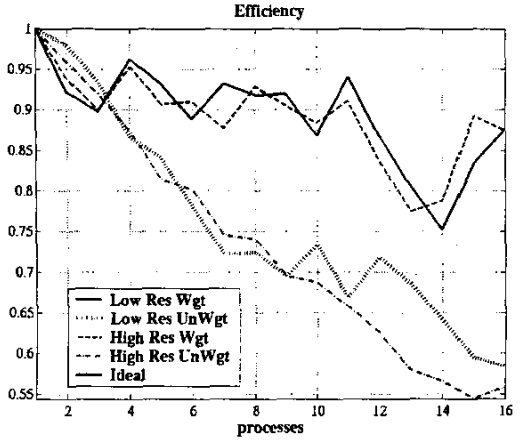


Fig. 13. The efficiency for the two sets of data from the two models for the power bus with 16 decoupling capacitors.

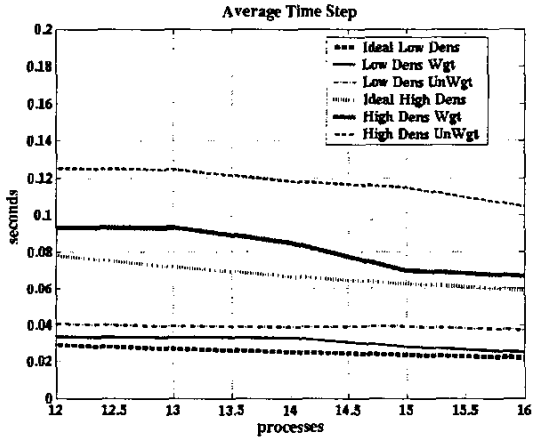


Fig. 14. An enlargement of the average time step for the two sets of data from the high and low resolution models for the power bus with 16 decoupling capacitors.

average time steps for the four sets of data collected and their ideals (T_1/N) for the multi-layer power bus. The difference in the weighted and unweighted cases in the speedup and efficiency curves for the four cases, found in Fig. 17 and Fig. 18, respectively, are similarly apparent for the multi-layer power bus as in that of the 16 layer power bus.

Fig. 18 shows the efficiency for the power bus. Again with this model, the weighted efficiencies are higher than that of the unweighted cases. However, there is little difference in the low and higher resolution models. For a larger model, there will be more cells to work on for each communication. It may be that the high resolution model does not have a significantly larger amount of calculations to make the differences apparent. Again, it is seen in Figure 18, that it is more profitable to use the parallel algorithm with weighted versions. The difference between the cases is more dramatically illustrated by viewing an enlargement of Fig. 16, found in Fig. 19, where only the last 12 to 16 processes are shown.

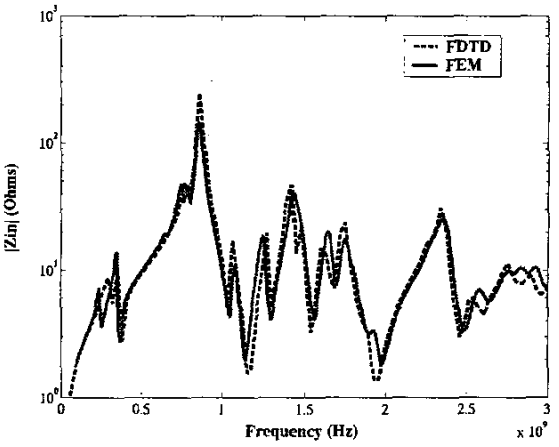


Fig. 15. FDTD and FEM modeled $|Z_{in}|$ for the multi-layer power distribution network.

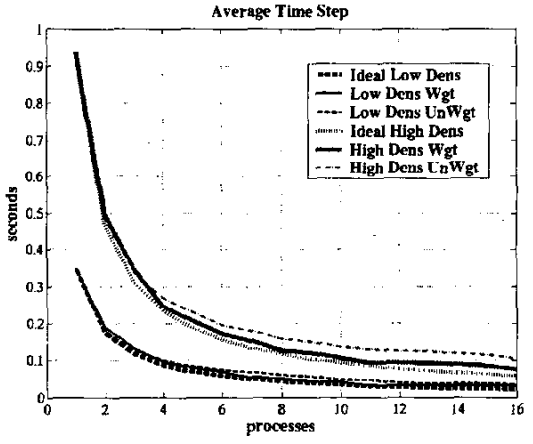


Fig. 16. The average time step for the two sets of data from the high and low resolution models for the multi-layer power distribution network.

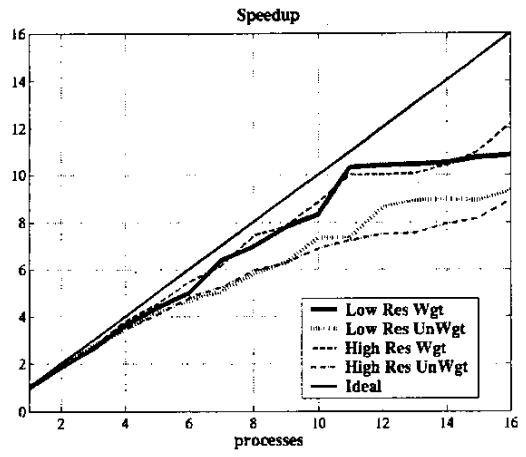


Fig. 17. The speedup for the two sets of data from the two models for the multi-layer power distribution network.

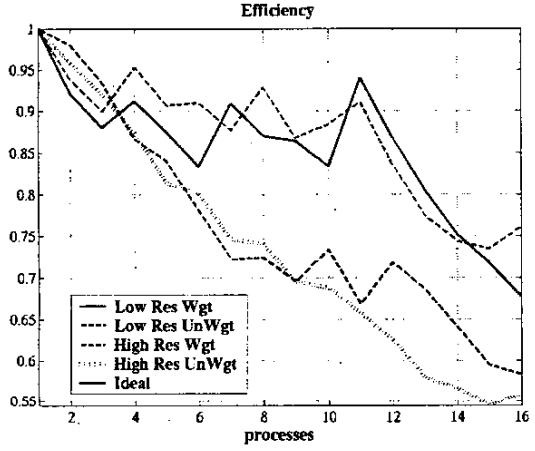


Fig. 18. The efficiency for the two sets of data from the two models for the multi-layer power distribution network.

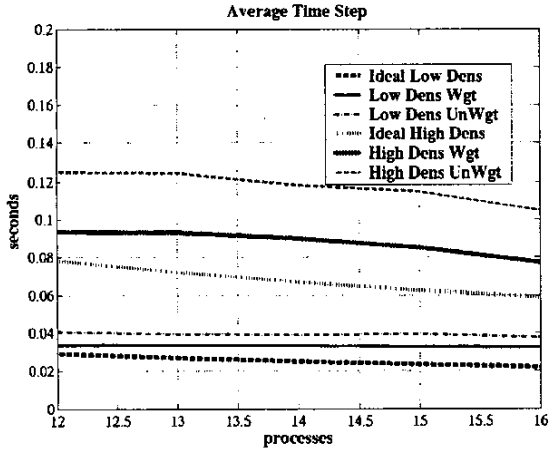


Fig. 19. An enlargement of the average time step for the two sets of data from the high and low resolution models for the multi-layer power distribution network.

III. Summary

Both examples demonstrated that although communications are a defining part of the MPI, they impede the efficiency of the parallel algorithm. Load imbalances are the other major impedance to the efficiency of the parallel algorithm. There are two causes for these load imbalances. First, the data is not equally distributed between the processes or that the data is distributed equally, but part of the problem requires more computation time. Second, the processors used have different capabilities, such as running at different clock speeds or having different speed random access memory (RAM). The load balancing in the parallel code for this report attempted to alleviate these imbalances by implementing a quasi-dynamic load balancing approach that addresses, in part, both the first and second issues.

Adding a weighting scheme for PML cells has its benefits. The weighted versions of the program continually outperformed the unweighted versions with various numbers of processors. When illustrating the significance of the difference in execution times, it may be assumed that there are 16 processes running and, in the high density mesh, the average time steps for the weighted and unweighted cases are approximately 0.5 to 1.4 seconds, respectively. Over 10,000 time steps amounts to a difference of approximately 60 minutes more than what is required for the weighted PML run. The power bus, with 16 decoupling capacitors, and the multi-layer power distribution network are just small examples in comparison to other models that might be considered. In addition to much larger problems, models with vastly different dimensions or mixed scale problems, for example a thin power bus considering the skin depth, are well-suited to a parallel scheme. The high density model only represents mostly simple dielectrics and PEC sheets for both cases.

References

[1] Pacheco, Peter S. *Parallel Programming with MPI.* San Francisco, California: Morgan Kaufmann Publishers, Inc., 1997.

[2] Michael A. Cracraft. *Parallelizing a Finite-Difference Time-Domain Code Using the Message-Passing Interface.* Master's thesis, University of Missouri - Rolla, May 2002.

[3] Gropp, W., E. Lusk, and A. Skjellum. *Using MPI.* Cambridge, Massachusetts: The MIT Press, 1999.

[4] Wilkinson, Barry and Michael Allen. *Parallel Programming: Techniques and Apllications Using Networked Workstations and Parallel Computers.* Upper Saddle River, New Jersey: Prentice Hall, Inc., 1999.

[5] X. Ye, M. Y. Koledintseva, M. Li, and J. L. Drewniak. *DC power-bus design using FDTD modeling with dispersive media and surface mount technology components.* IEEE Transactions on Electromagnetic Compatibility, vol. 43, no. 4, pp. 579-587, Nov. 2001.

[6] Taflove, Allen and Susan Hagness. *Computational Electrodynamics: The Finite-Difference Time-Domain Method–2nd Edition.* Boston, Massachusetts: Artech House, 2000.