A Novel FDTD Application Featuring OpenMP-MPI Hybrid Parallelization

Mehmet F. Su
Dept. of Electrical and Computer Engineering
University of New Mexico
Albuquerque, NM 87131
mfatihsu@ece.unm.edu

Ihab El-Kady
Sandia National Laboratories
PO Box 5800, MS 0603
Albuquerque, NM 87185
ielkady@sandia.gov

David A. Bader
Dept. of Electrical and Computer Engineering
University of New Mexico
Albuquerque, NM 87131
dbader@ece.unm.edu

Shawn-Yu Lin
Sandia National Laboratories
PO Box 5800, MS 0603
Albuquerque, NM 87185
sylin@sandia.gov

Abstract

We have developed a high performance hybridized parallel Finite Difference Time Domain (FDTD) algorithm featuring both OpenMP shared memory programming and MPI message passing. Our goal is to effectively model the optical characteristics of a novel light source created by utilizing a new class of materials known as photonic band-gap crystals. Our method is based on the solution of the second order discretized Maxwell's equations in space and time. This novel hybrid parallelization scheme allows us to take advantage of the new generation parallel machines possessing connected SMP nodes. By using parallel computations, we are able to complete a calculation on 24 processors in less than a day, where a serial version would have taken over three weeks. In this paper we present a detailed study of this hybrid scheme on an SGI Origin 2000 distributed shared memory ccNUMA system along with a complete investigation of the advantages versus drawbacks of this method.

Keywords: FDTD, Finite Difference Time Domain, OpenMP, MPI, Maxwell Equations, Photonic Crystals.

1 Introduction

Photonic crystals are materials fabricated with a periodicity in index of refraction and specifically designed to affect and control the properties of light (or photons) in much the same way that semiconductors affect and control the properties of electrons. This provides scientists with a completely new dimension in the ability to control and manipulate the properties of light. The key lies in the concept of a photonic band gap — the optical analogue of the electronic band gap in semiconductors. In effect, it allows us to tailor the properties of photons the way we tailor the properties of electrons. Harnessing the broad potential of photonic crystals promises to have enormous technological implications in telecommunications, optical computing, and optoelectronics, as well as in associated imaging applications. One such application is the use of such materials in the lighting technology, where such crystals were suggested to possess the ability to internally recycle the unwanted thermal losses of usual light sources, such as conventional light bulbs, into useful visible radiation in the form of visible light [7]. Recent work has proven that such suggestions are actually possible, and an observation of the novel emission characteristics of such systems has appeared in several publications [13, 10, 11, 12].

Modeling the behavior of these photonic systems is however a very complicated task given the nature of the metal-photon coupled system. Conventional methods based on real-space transfer matrices [14, 6] and Modal Expansion techniques [9, 8], are only capable of producing passive optical properties of such systems such as transmittance, reflectance, and in some cases absorbance. Such methods however lack the ability to produce the real-time behavior and development of the electromagnetic field vectors in the photonic crystal environment. Furthermore, because such methods can at most provide passive optical properties, they are incapable of estimating the emissivity of such systems, and lack the ability to quantify the anisotropy of the emission process.

In this paper we present a finite difference time domain (FDTD) method based on the solution of the second or-

Proceedings of the 2004 International Conference on Parallel Processing (ICPP'04)
0190-3918/04 $20.00 © 2004 **IEEE**

der discretized Maxwell's equations in space and time, and designed specifically for modeling such complicated photonic systems. Because of the frequency dependent metallic properties and the skin-depth effects that arise from light-metal interactions, a very fine meshing of the system is needed. As a result huge data sets are involved and a sluggish computational performance is observed. To alleviate the problem we introduce a novel hybrid parallelization scheme that employs both OpenMP shared memory programming and MPI message passing. The result is a highly performing algorithm capable of handling such complicated physical problems, and paves the way for more efficient algorithms better suited for the new generation of clusters with SMP nodes. With parallel computations, we are able to reduce the time required for a full scale simulation from over three weeks to about one day.

This paper is organized as follows: The next section gives a brief formulation of Maxwell Equations and a specific FDTD method known as Yee's algorithm. The third section discusses the specifics of OpenMP and MPI parallelization schemes and their applications to FDTD. The fourth section shows some experimental results and discusses performance issues. In the fifth and final section, we conclude.

2 Maxwell Equations and Yee's Algorithm

2.1 Maxwell Equations

Maxwell Equations are a set of coupled vectorial partial differential equations. They govern the world of electromagnetics and optics. The usual compact formulation is:

$$\frac{\partial \vec{B}}{\partial t} = -\nabla \times \vec{E} - \vec{M} \quad (1)$$

$$\frac{\partial \vec{D}}{\partial t} = \nabla \times \vec{H} - \vec{J} \quad (2)$$

$$\nabla \cdot \vec{D} = 0 \quad (3)$$

$$\nabla \cdot \vec{B} = 0 \quad (4)$$

with definitions

$$\vec{D} = \epsilon_r \epsilon_0 \vec{E} \quad (5)$$

$$\vec{B} = \mu_r \mu_0 \vec{H} \quad (6)$$

$$\vec{J} = \vec{J}_{source} + \sigma \vec{E} \quad (7)$$

$$\vec{M} = \vec{M}_{source} + \sigma^* \vec{H} \quad (8)$$

where $\epsilon_r, \mu_r, \sigma, \sigma^*$ are known material parameters, $\vec{J}_{source}, \vec{M}_{source}$ are known properties of electromagnetic sources inside the system and ϵ_0, μ_0 are defined constants. $\vec{E}$ is known as the electric field and $\vec{B}$ is known as the magnetic field.

Maxwell Equations have both space and time derivatives. In the above form, space derivatives are hidden in the $\nabla\times$ (curl) and $\nabla\cdot$ (divergence) operators.

2.2 FDTD revisited: Yee's algorithm

FDTD is a direct integration method which casts partial derivatives to partial differences on a finite mesh. This mesh contains the material parameters mentioned above, and is actually a representation of the scientific problem to be solved. Then the resulting equation is integrated numerically in space and time by the aid of initial conditions and termination conditions (boundary conditions).

Yee's algorithm is a well known method published in 1966 [17]. It gives a second order (in time and space) correct FDTD solution to Maxwell equations. For reasons of brevity and focus, we shall not attempt to give a full derivation of all the difference equations here, but merely cite the resulting equations. The interested reader is referred to [15].

In Yee's algorithm, the electric and magnetic fields are defined on an intertwined double mesh, where electric field components are circulated by four magnetic field components and magnetic field components are circulated by four electric field components. The first partial space derivative of a field u is defined to be (correct to the second order):

$$\frac{\partial u}{\partial x}(i\Delta x, j\Delta y, k\Delta z, n\Delta t) \approx \frac{u^n_{i+1/2,j,k} - u^n_{i-1/2,j,k}}{\Delta x} \quad (9)$$

and the first time derivative is similarly (again, correct to the second order),

$$\frac{\partial u}{\partial t}(i\Delta x, j\Delta y, k\Delta z, n\Delta t) \approx \frac{u^{n+1/2}_{i,j,k} - u^{n-1/2}_{i,j,k}}{\Delta t} \quad (10)$$

For mathematical reasons, it suffices to apply these transformations to only the first two of the Maxwell equations. The other two equations are satisfied inherently by the formulation of the algorithm. Usage of these difference formulas with simple averaging where necessary yields six explicit time stepping equations. The electric fields and the magnetic fields are updated using [5] equations (11) and (12), respectively.

In these equations, $\hat{i}, \hat{j}, \hat{k}$ refer to space indices x, y, z and their cyclic permutations.

$$\vec{E}_{\hat{i}}|_{i,j,k}^{n+1} = \left(\frac{1-\frac{\sigma_{i,j,k}\Delta t}{2\epsilon_{i,j,k}}}{1+\frac{\sigma_{i,j,k}\Delta t}{2\epsilon_{i,j,k}}}\right)\vec{E}_{\hat{i}}|_{i,j,k}^{n} + \left(\frac{\frac{\Delta t}{\epsilon_{i,j,k}}}{1+\frac{\sigma_{i,j,k}\Delta t}{2\epsilon_{i,j,k}}}\right)\left(\frac{\vec{H}_{\hat{k}}|_{i,j+\frac{1}{2},k}^{n} - \vec{H}_{\hat{k}}|_{i,j-\frac{1}{2},k}^{n}}{\Delta\hat{j}} - \frac{\vec{H}_{\hat{j}}|_{i,j,k+\frac{1}{2}}^{n} - \vec{H}_{\hat{j}}|_{i,j,k-\frac{1}{2}}^{n}}{\Delta\hat{k}}\right) \quad (11)$$

$$\vec{H}_{\hat{i}}|_{i,j,k}^{n+\frac{1}{2}} = \left(\frac{1-\frac{\sigma^*_{i,j,k}\Delta t}{2\mu_{i,j,k}}}{1+\frac{\sigma^*_{i,j,k}\Delta t}{2\mu_{i,j,k}}}\right)\vec{H}_{\hat{i}}|_{i,j,k}^{n-\frac{1}{2}} + \left(\frac{\frac{\Delta t}{\mu_{i,j,k}}}{1+\frac{\sigma^*_{i,j,k}\Delta t}{2\mu_{i,j,k}}}\right)\left(\frac{\vec{E}_{\hat{k}}|_{i,j+1,k}^{n} - \vec{E}_{\hat{k}}|_{i,j,k}^{n}}{\Delta\hat{j}} - \frac{\vec{E}_{\hat{j}}|_{i,j,k+1}^{n} - \vec{E}_{\hat{j}}|_{i,j,k}^{n}}{\Delta\hat{k}}\right) \quad (12)$$

2.3 Initial conditions and boundary conditions

The FDTD explicit time stepping requires initial conditions for the field values and boundary conditions at space boundaries. The conventional way to define initial conditions is to initialize all the field values to zero everywhere. It is also possible to save the fields between two consecutive FDTD time steppings and resume such a saved calculation, where saved fields are loaded as an initial condition.

As for boundary conditions, several choices are possible. The electromagnetic energy is reflected at the end of FDTD numerical space since the space is finite. This yields spurious, unwanted energy reflections. The usual way of preventing such reflections is to use Absorbing Boundary Conditions (ABC's) at the space boundaries.

In our application Liao and Uniaxial Perfectly Matched Layer (UPML) boundary conditions [16] were used. Liao ABC is based on a Newton extrapolation polynomial technique. Used at the space boundaries, this extrapolation simulates "endless free space" and absorbs electromagnetic energy, removing reflections. The UPML ABC is a variant of Berenger's Perfectly Matched Layer [3] technique, and it is a slab of artificial material. The properties of the slab are set up so as to absorb all incoming electromagnetic energy. Deeper into a PML absorber slab, the absorbance of the material is increased polynomially or geometrically. We chose to increase the absorbance polynomially.

```
Algorithm: Serial-FDTD
Do one time initialization work;
Initialize fields, apply initial conditions;
for t = 1 to tmax do
    for i, j, k = 1 to imax, jmax, kmax do
        Update electric fields using magnetic fields;
        Update magnetic fields using updated electric
        fields;
        Update fields at boundaries, apply boundary
        conditions;
    end
end
```

Algorithm 1: Serial FDTD algorithm.

3 Parallel FDTD calculations

3.1 FDTD and MPI Parallelization

In this part, details on MPI parallelization for FDTD are given. To make comparisons easier, a serial version of the algorithm is presented first:

The MPI programming paradigm assumes that data structures are not shared, but divided among the multiple processors used. This idea of storage division calls for a domain decomposition technique. Since the field updates at a mesh point take field values from surrounding points, exchanges are required at domain boundaries. Alg. 2 implements domain decomposition using MPI message passing, and is a modification of the preceding serial FDTD algorithm.

The number of required field components to exchange grows with total surface area of the domains, which is proportional to *both* the total size of the mesh *and* the number of processors (or distinct domains) used in calculation. This is an overhead cost that is brought by domain decomposition.

On the other hand, the MPI parallelization is advantageous not only because it allows one to take advantage of a parallel machine to reduce execution time, but it also has the very important features of distributed allocation and distributed storage allowing for the solution of larger problem instances. With MPI parallelized FDTD, the total storage used is nearly evenly divided between the MPI processes. Hence, the per-process storage is low, and it becomes lower as the number of processes increases. This enables one to calculate with larger meshes and even overcome the 32-bit storage limit problem in the parallel systems built using commodity 32-bit system components.

Algorithm: MPI-FDTD
Do one time initialization work;
Initialize fields, apply initial conditions;
for *t = 1* ***to*** *tmax* **do**
  for *i, j, k = 1* ***to*** *imax, jmax, kmax* **do**
    Using MPI message passing, exchange magnetic fields with neighbors;
    Update electric fields using magnetic fields;
    Using MPI message passing, exchange updated electric fields with neighbors;
    Update magnetic fields using updated electric fields;
    Update fields at boundaries, apply boundary conditions;
  end
end

Algorithm 2: MPI parallelized FDTD algorithm.

3.2 FDTD and OpenMP

Many modern large-scale computer systems are networks of connected shared memory SMP nodes, using combined distributed and shared memory approaches. In these systems, message passing in a node is emulated on shared memory and message passing between nodes uses actual messages. In order to efficiently harness the computational power of such systems, one has to use both shared and distributed memory approaches together [1].

Inspecting the field update equations, a simple fact is realized: When the electric fields are being *updated*, the magnetic fields are only *read*, and when the magnetic fields are being *updated*, the electric fields are only *read*. As long as the field update loops are kept separate, these calculations can be completed in a multi-threaded environment. There is no risk of a race condition as long as different threads update different field components. There is also no domain distribution overhead problem as long as the storage is shared and accessible by all the threads.

In our approach, we use the OpenMP shared memory parallel programming standard. The FDTD code maps naturally to the OpenMP paradigm, and has the benefits mentioned above. However, there are several down sides of shared memory FDTD:

- OpenMP lacks support for distributed allocation of shared structures, causing bottlenecks on some systems.
- Since the data set is very large, there are many cache misses, affecting performance and scalability severely.
- Optimization of OpenMP constructs is not trivial and requires extensive experimentation and optimization.

To solve the first issue, we use the distributed allocation and automated data relocation features available on an SGI Origin 2000 [4]. SGI's OpenMP implementation accepts several environment variables which affect how the system allocates data and how data is relocated at run-time to optimize access latency. It is possible to make the system allocate data with a first-touch policy, which distributes data properly if the storage is allocated by all the processors in parallel. Since OpenMP does not have such support, the *first-touch* policy is not useful. Another policy is to allocate each page of storage on different regions of the distributed memory, that is, the *round robin* allocation. That way, access latency is not optimized, but any bottlenecks caused by massive usage of a single portion of the physical memory is prevented. The third policy is the usage of a *predetermined storage map*, which requires a separate run to generate the map first. The allocated pages can be relocated by the system automatically to minimize access latency incurred by two processors, according to the usage statistics by each processor, yielding optimal memory placement at run time. For our case, we find round-robin allocation with automatic relocation best.

To solve the second problem, we employ some methods to reduce the total storage requirement (to contain the number of total cache misses) as well as rearrangement of calculations to make better use of caches and prefetch features. To reduce the storage requirement, our implementation uses a lookup table and stores material properties as short one-byte integers referring to this table. Specifically, 12 double complex parameters are replaced per grid point. This approach reduces the storage requirement by almost half and boosts scalability, as shall be discussed in the next section.

Last but not least, algorithm engineering techniques [2] of explicit timings, educated guesses and trial-and-error are used in the rearrangement of FDTD equations to increase prefetch efficiency and to optimize OpenMP options such as whether it would be more efficient to use *SHARED* or *FIRSTPRIVATE* and whether an SGI extension to OpenMP, the *NEST* construct, should be used. The effects of all these optimizations will be discussed in the next section.

3.3 Hybrid FDTD: Using OpenMP and MPI together

Shared memory machines do not scale up beyond a certain number of processors as overheads become unbearable. To obtain a more general parallel FDTD application, OpenMP and MPI are used together. In this case, the calculations performed in the MPI storage domains go multithreaded. This gives the option to choose how the processors are shared between MPI processes and how many OpenMP threads are used in each MPI process. It also reduces the MPI domain decomposition overhead, since there

will not be as many domains as a pure MPI parallelized FDTD. Where OpenMP is not supported, one can of course use just MPI with only a change in compiler options and no changes in the source code. The hybrid algorithm is:

Algorithm: Hybrid-FDTD
Do one time initialization work;
Using OpenMP multithreading, initialize fields, apply initial conditions;
for *t = 1* ***to*** *tmax* **do**
  for *i, j, k = 1* ***to*** *imax, jmax, kmax* **do**
    Using MPI message passing, exchange magnetic fields with neighbors;
    Using OpenMP multithreading, update electric fields using magnetic fields;
    Using MPI message passing, exchange updated electric fields with neighbors;
    Using OpenMP multithreading, update magnetic fields using updated electric fields;
    Using OpenMP multithreading, update fields at boundaries, apply boundary conditions;
  end
end

Algorithm 3: MPI-OpenMP hybrid parallelized FDTD algorithm.

In the next section, we shall present and discuss the results obtained.

4 Results and Discussion

In this section, we shall concentrate on the performance analysis of the OpenMP parallelized FDTD since MPI parallelized FDTD has been around long enough to reach standard university curricula. Performance tests of MPI-based and hybrid FDTD schemes will be discussed in a future work in order to keep this paper concise and focused on the issues with and optimization of OpenMP enhanced FDTD codes. The FDTD application code analyzed here has two versions: The first version implements OpenMP in a straightforward manner, relying mostly on compiler optimizations. The second (a further optimized version) is derived from the first version and includes changes to improve cache usage and reduce memory footprint, as mentioned earlier. Analysis has been made on an SGI Origin 2000 featuring 350 MHz MIPS R12000 processors.

Figure 1 shows the execution times of the two versions of the code for a sample case. For this sample, the mesh contains 188x188x78 grid points, and the calculation was allowed to run for 3000 time steps. The time scale is in seconds, which means the calculation completes in almost 5 hours when run on a single processor. For uses of physics research, the calculation often has to be repeated with a different set of parameters (such as in a sweep of the frequencies emitted by the sources). As a result, the total time required can easily become several weeks when a single processor is employed.

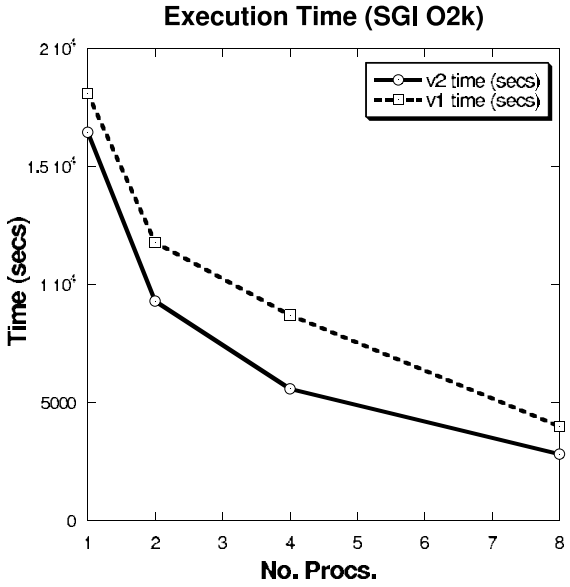

Figure 1. Execution times on the SGI Origin 2000

The speedup curves (Figure 2) obtained from the same system make it easier to see that a straightforward OpenMP parallelization does not quite provide an efficient answer for our needs.

Usage of the profiling facilities [4] available on the SGI Origin as a first probe indicates that the program has a problem of cache and prefetch misses. The average reuse rate for L1 cache lines are 10.4, whereas for L2 cache this rate drops to only 3.4 (see Table 1). To appreciate these results as well as the problem itself, a closer look at the profile statistics is necessary.

Without doubt, the most time consuming parts of such an FDTD calculation are the electromagnetic field updates. Formulated in the complex equations (11) and (12), these updates are performed at every grid point for every time step. The main data for these equations are the fields and the constant factors[1]. At every time step, a complete sweep of the space grid (and this data) is performed. Considering that 32-byte *double complex* variables are used for the fields, a small calculation shows a whopping amount of 200 MB is necessary for electromagnetic field data storage alone. As

[1] The terms with ϵ and μ

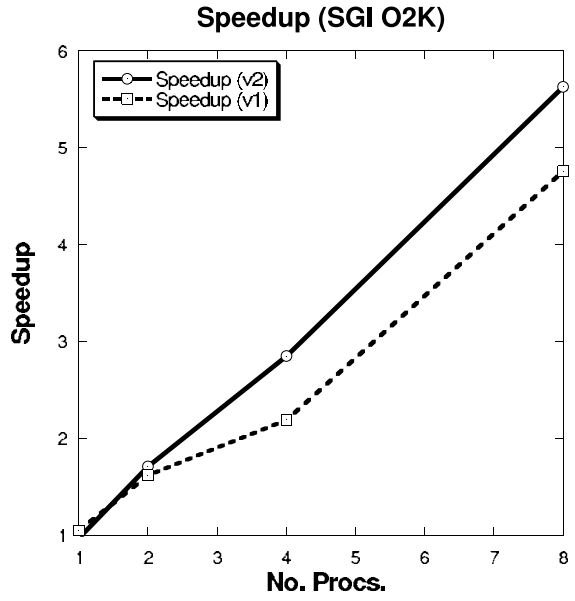

Figure 2. Speedups on the SGI Origin 2000

Categories	Statistics (v1)	Statistics (v2)
Data mispredict /Data scache hits	0.003954	0.007023
Instruction mispredict /Inst. scache hits	0.043838	0.064697
L1 Cache Line Reuse	10.412669	17.648883
L2 Cache Line Reuse	3.375819	4.589698
L1 Data Cache Hit Rate	0.912378	0.946377
L2 Data Cache Hit Rate	0.771471	0.821099
Time accessing memory /Total time	0.638376	0.576530
Memory BW (MB/s, avg per proc)	186.573422	150.096080
Cache misses/cycle (avg)	2.572237	1.557080
Prefetch cache miss rate	0.433175	0.282235

Table 1. Various statistics for the two versions of the application (SGI Origin 2000, 8 processors)

the problem size grows, this exceeds current cache sizes except for perhaps the highest-end systems available.

The situation is even more complicated by the form of the field update equations (11) and (12). A closer look indicates that, 27 floating point numbers have to be loaded for every grid point at every time step per each of the two equations: 9 floating point numbers per the 3 spatial components mapped at every grid point. Moreover, the access patterns are far too complicated to allow for any optimizations, and they are even not at a stride to fit into cache.

Hence, there are two different performance problems: one is about the complicated access patterns and the other is about the high number of cache misses. In fact, the first problem somewhat contributes to the second one, since complicated access patterns with large strides often incur cache misses. Both of these problems indicate that our code is memory bound, and its performance will depend on the performance of the memory subsystem.

Our solution to the first problem mentioned above is to merge all the field updates for one grid point into one big loop. Although the access patterns are still complex when the field updates are taken one spatial component at a time, the updates for all three spatial components per grid point per field access close array elements in aggregate. Profiling the code confirms that the rate of prefetch cache misses is reduced from 43.3% to 28.2% (Table 1).

To improve cache usage is to find ways to *slow down*, if not *prevent*, cache flushing as the calculation goes on and new data is read into the cache. Since the data set does not fit into the cache, prevention is not an option for our case. The other option, to slow down, calls for inventing ways to reduce memory footprint of the data set. Our solution is to implement a lookup table which would keep the few constant factors used in cache. The elements of this table are pointed to by short (1-byte) integers, which replace the old storage of constant factors, which are floating point numbers. In our implementation, 12 constant factors are used per grid point (4 factors per 3 spatial components). By implementing the lookup table, we are able to reduce the overall memory footprint in half, and the cache line reuse rates are improved to 17.7 for L1 cache and 4.6 for L2 cache (Table 1).

The SGI Origin 2000 is a cache coherent, non-uniform memory access (ccNUMA) distributed shared memory platform. Since OpenMP lacks support for distributed allocation of shared storage, an OpenMP parallelized code using default allocation policies allocates storage from a single segment of the distributed memory. This single location bombarded by many processors is bound to become a memory hotspot and a bandwidth bottleneck. Indeed, timings of the second version of the code on 8 processors indicate 4975 seconds for the running time when default allocation policies are used (versus 2828 seconds[2]).

The Origin has facilities to migrate pages of data automatically to a closer position to the processors using them most. To activate this feature, one sets an environment variable named _DSM_MIGRATION. Another environment variable, _DSM_PLACEMENT, affects the policies used to determine the physical location of the pages allocated. The default, named *first touch allocation* causes memory pages become allocated closer to the first processor access-

[2]with round-robin allocation and automated data migration

ing to them. This same policy is the culprit that causes an OpenMP parallelized code to allocate all the shared storage from one single location. An alternative, allocating pages in a *round robin* fashion from all available distributed memory locations, is available and can be activated by setting the aforementioned environment variable. During our runs, we activate both of these features.

5 Conclusions and Future Directions

To sum up; we have designed, implemented and analyzed an FDTD simulation for a challenging physics problem which takes weeks to compute without parallel calculations. We indicated how an OpenMP based parallel implementation could be improved and gave quantitative indications to support our suggestions. The results suggest that OpenMP parallelization can be used together with an MPI domain distribution scheme to get a high performance, hybrid parallelized FDTD application to harness the power of newer parallel systems constructed out of interconnected SMP nodes.

Another lesson that has been learned is, relatively newer technologies in software and programming (such as OpenMP) can shed a new light to the application areas where the methods are well known and beyond a certain age. It may take time to see how an improved technology or technique could help in probable application areas that look unrelated or impossible at first sight.

As for future work, we will be studying the performance characteristics exhibited by this application under several other architectures, so as to expose details on how to get better performance from these machines. As of the preparation date of this document, analysis is under way on Sun UltraSparc and IBM Power based platforms. The intimate experience gained in optimizing the application on the Origin 2000 and other architectures will be useful as case studies and the knowledge will help the designs of future scientific simulations that achieve the highest performance.

6 Acknowledgments

This work was supported in part by: DOE Sandia National Laboratories contract 161449; NSF Grants CAREER ACI-00-93039, ITR ACI-00-81404, DEB-99-10123, ITR EIA-01-21377, Biocomplexity DEB-01-20709, ITR EF/BIO 03-31654; and DARPA Contract NBCH30390004.

References

[1] D. A. Bader and J. JáJá. SIMPLE: A methodology for programming high performance algorithms on clusters of symmetric multiprocessors (SMPs). *Journal of Parallel and Distributed Computing*, 58(1):92–108, 1999.

[2] D. A. Bader, B. M. E. Moret, and P. Sanders. Algorithm engineering for parallel computation. *Experimental Algorithmics, Lecture Notes in Computer Science*, 2547:1–23, 2002.

[3] J. P. Berenger. A perfectly matched layer for the absorption of electromagnetic waves. *J. Computational Physics*, 114:185–200, 1994.

[4] D. Cortesi, J. Fier, J. Wilson, and J. Boney. *Origin 2000 and Onyx2 Performance Tuning and Optimization Guide*. Silicon Graphics Inc., 2001. Document Number 007-3430-003.

[5] I. El-Kady. *Modeling of photonic band gap crystals and applications*. PhD thesis, Iowa State University, Ames, Iowa, 2002.

[6] I. El-Kady, M. M. Sigalas, R. Biswas, K. M. Ho, and C. M. Soukoulis. Metallic photonic crystals at optical wavelengths. *Phys. Rev. B*, 62(23):15299, 2000.

[7] J. G. Fleming, I. El-Kady, S. Y. Lin, R. Biswas, and K. M. Ho. All metallic, absolute photonic band gap three-dimensional photonic-crystals for energy applications. *Nature*, 417:52, 2002.

[8] Z. Y. Li, I. El-Kady, K. M. Ho, S. Y. Lin, and J. G. Fleming. Photonic band gap effect in layer-by-layer metallic photonic crystals. *Journal of Applied Physics*, 93:38, 2003.

[9] L. L. Lin, Z. Y. Li, and K. M. Ho. Lattice symmetry applied in transfer-matrix methods for photonic crystals. *Journal of Applied Physics*, 94:811, 2003.

[10] S. Y. Lin, J. G. Fleming, and I. El-Kady. Experimental observation of photonic-crystal emission near photonic band-edge. *Applied Physics Letters*, 83(4):593, 2003.

[11] S. Y. Lin, J. G. Fleming, and I. El-Kady. Highly efficient light emission at 1.5microns by a 3D tungsten photonic crystal. *Optics Letters*, May 2003. Accepted.

[12] S. Y. Lin, J. G. Fleming, and I. El-Kady. Three-dimensional photonic-crystal emission through thermal excitation. *Optics Letters*, May 2003. Accepted for publication.

[13] S. Y. Lin, J. G. Fleming, Z. Y. Li, I. El-Kady, R. Biswas, and K. M. Ho. Origin of absorption enhancement in a tungsten, three-dimensional photonic crystal. *J. Opt. Soc. Am B.*, 20(7):1538, 2003.

[14] J. B. Pendry. Calculating photonic band structure. *J. Phys. Cond. Mat.*, 8:1085–1108, 1996.

[15] A. Taflove and S. C. Hagness. *Computational Electromagnetics: The Finite Difference Time Domain Method*, chapter 3. Artech House, Boston, MA, second edition, 2000.

[16] A. Taflove and S. C. Hagness. *Computational Electromagnetics: The Finite Difference Time Domain Method*, chapter 7. Artech House, Boston, MA, second edition, 2000.

[17] K. S. Yee. Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media. *IEEE Trans. Antennas and Propagation*, 14:302–307, 1966.