Handbook of Research on Computational Science and Engineering:

Theory and Practice

Joanna Leng
Consultant, UK

Wes Sharrock
University of Manchester, UK

Volume I

ENGINEERING SCIENCE REFERENCE

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data

Handbook of research on computational science and engineering: theory and practice / Joanna Leng and Wes Sharrock, editors.
p. cm.
Summary: “This book offers a timely introduction to the possibilities in computational science and engineering to advance the ongoing research and applications leading to the discovery of new resources and cutting edge developments”-- Provided by publisher.
Includes bibliographical references and index.
ISBN 978-1-61350-116-0 (hardcover) -- ISBN 978-1-61350-117-7 (ebook) -- ISBN 978-1-61350-118-4 (print & perpetual access) 1. Science--Data processing. 2. Engineering mathematics--Data processing. 3. Numerical analysis--Data processing. 4. Mathematical models. 5. Computer simulation. I. Leng, Joanna, 1965- II. Sharrock, W. W. (Wes W.)
Q183.9.H36 2012
501’.13--dc23
2011032075

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter 9
Opportunities and Challenges in Porting a Parallel Code from a Tightly-Coupled System to the Distributed EU Grid, Enabling Grids for E-sciencE

Fumie Costen
University of Manchester, UK

Akos Balasko
Hungarian Academy of Sciences, Hungary

ABSTRACT

Any large scale computation, either in the science or arts, requires high performance computing (HPC) facilities. This computational environment may change over time. Thus the source code of a computation needs to be ported. The change in the computational architecture or system can make the porting of code between various HPC facilities challenging. This chapter introduces an example of an engineering application which runs on a HPC facility and the porting from a local computing facility to Enabling Grids for E-sciencE (EGEE) is described in detail.

The computational architecture of Enabling Grids for E-sciencE is introduced as it made our code porting very challenging, and the discussion presented is directly applicable to EGEE users. The final solution to the code poring problem is proposed, and its performance is examined. The solution to this problem be generally faced in the other large scale computation and so is applicable to users of other HPC facilities. This chapter gives a hint to those who have difficulties in applications with heavy data Input/Output (I/O) under the computational environment whose weak point is the data I/O.

DOI: 10.4018/978-1-61350-116-0.ch009

INTRODUCTION

Research on distributed and parallel systems is one of the most important areas in computer science. This area is based on the exploitation of large computational and data storing capabilities. While the main components *i.e.*, processors and hard drives in a single computer are becoming smaller but with larger storage capacity and higher processing performance, distributed systems can integrate these individual resources into one large, heterogeneous, dynamic system that allows users to benefit from the possible improved performance. These systems are called grid.

The main goal of a well-maintained grid is to provide large scale resources connected via the Internet to researchers in the natural sciences and engineering who have applications with high demands for compute resources, or storing more data than a single machine can accommodate. Certainly, these applications must be parallelised to fully exploit the resource capabilities, and make them run faster in grid systems.

Researchers are nowadays surrounded with a variety of grid computing facilities. Some are more suitable to one's application than others but the cost and the performance of each HPC facility is also different. Furthermore, the cost, the performance and the suitability are always changing over time. Therefore researchers have to be prepared for the change in the computational facility and have to be able to adjust to the new computational environment.

This chapter shares the authors' experience of a significant change to the computational environment used in the daily research activities and provides some hints to those who may face the similar situation.

The authors' experience is based on the Enabling Grids for E-sciencE (EGEE) project.

The EGEE project-family, founded by the European Commission, started on April 2004. It has provided academic and industrial researchers the means to have access to large computing resources. It is focused on developing and maintaining a robust and powerful grid network and components, and to attract new users from industry by standardized training and dissemination events.

A new grid-middleware, called gLite was developed during this project. Its aim was to organize and connect the components of the large and international grid system. The last project of this family (EGEE-III) was ended on April 30th 2010. The new project was created to continue the development of distributed systems internationally in Europe and is called European Grid Initiative(EGI). In this project all of the old organizational-ideas have been reformed. EGI manages the collaborative work of NGIs (National Grid Initiatives) that are created to support the national grid-community and maintain the related grid-services.

Another but no less important project, founded by the European Commission is EMI (European Middleware Initiative). This project aims at integrating the three major European grid middleware systems (ARC, gLite, Unicore) into a unified middleware distribution (UMD) in order to support the co-operation of researchers in the same research field but with different grid-middlewares.

The section on Computation in Electromagnetics discusses the motivation of our research and introduces the core part of the equations necessary to understand the nature of our computation. Furthermore the section talks about the computational environment we used before we faced a significant change. The section on EGEE computational facility introduces the computational architecture of Enabling Grids for E-sciencE(EGEE), which is significantly different from our initial architectures. The section on the Adaptation of our code to EGEE describes the problems which we faced and presents the solutions. The section on Future research direction gives some insight and suggestion for the improvement of the computational algorithms as well as the algorithms which could be applied to the data I/O problems.

COMPUTATION IN ELECTROMAGNETICS

The Necessity of Computation in Electromagnetics

One of our research activities in the University of Manchester is the study of the biomedical problems associated with human-body by means of numerical modelling and simulations. Our research goals include the development of the next generation of technology of broadband electromagnetics for bioengineering modelling and simulation for health care technologies. For example, in the films, some scenes involve the defibrillation of a fainted person. Defibrillation with the electric shock to the torso is usually successful in the films. However, in reality, the success rate is not as high as the one you see in the films. This is because the location, the shape, the excitation waveform of the electrodes is not optimum for the person to whom the electrical shock is applied. The optimum method to apply the electric shocks depends on the age, sex, size and body-shape of the person. The currently exercised therapy used to stimulate the heart does not have the known focus points of stimulation. This is mainly due to the fact that nobody knows the relationship between the precise location/number/waveform/phase of the electrodes and the stimulation focus points. In spite of a long clinical experience and detailed studies, the fundamental understanding of the mechanisms responsible for defibrillation is not fully known.

One way to increase this knowledge is to use computer simulations. Computer modelling allows us to perform experiments that are impossible physically and/or ethically to carry out with animals. The knowledge gained from the numerical simulations will be able to:

1. Replace the currently exercised defibrillation procedure with a flexible and more effective technique;
2. And expand the application of the heart defibrillation beyond the currently exercised area and improve the efficiency of these therapies in general.

The numerical simulation involves the propagation of ElectroMagnetic(EM) waves. In order to develop the simulation tool, the Maxwell's equations have to be solved. They can be numerically solved either in the frequency or time domain.

We need to perform numerical simulation of EM wave propagation from various electrodes around the torso to the heart in the time domain to reveal the unknown relationship mentioned above and provide knowledge on the best way to excite particular parts of the heart, aiming to increase the success rate of defibrillation.

For a comprehensive study the computer simulation should be able to handle both arbitrarily-complex and very fine geometry; as well as a wide frequency range and frequency dependent materials in time domain. The most suitable method currently available is the Finite Difference Time Domain(FDTD) method (Taflove and Hagness, 2005) unlike methods such as the Method of Moments (MoM), the Finite Element Method (FEM) (Margetts et al., 2004), the Geometrical Theory of Diffraction (GTD) and the Physical Theory of Diffraction (PTD). The detail required for the numerical modelling of our application is too complicated to be handled by GTD. In particular, broadband system analysis requires the examination of waveform distortion in the time domain during propagation in a wide range of dispersive media. Methods such as MoM and FEM mainly work in the frequency domain, requiring repetition of simulations, sweeping the frequency of interest to construct a single waveform in the time domain. Unlike MoM and FEM, both FDTD and Frequency Dependent (FD) - FDTD (Luebbers et al., 1991) works in the time domain and is capable of explicitly computing macroscopic transient electromagnetic interactions with general 3D geometries. Furthermore in FD-FDTD, the medium parameters such as permittivity and conductivity vary with frequency. It is important for the broadband simulations to have the capability to handle the frequency dependent materials. FD-FDTD is the simplest method among a variety of techniques to produce the time domain signal in the frequency dependent media. Thus, FD-FDTD is the most suitable for the numerical simulation of the wideband wave propagation in the human body. This chapter handles the standard explicit FD-FDTD method for the large scale computation.

Nature of the Computation of the FD-FDTD Methods

The computation of the EM wave propagation in a human body requires the following procedure:

1. Initialisation and data reading of the segmented human body
2. Setting the time loop counter to zero
3. Incrementation of the time loop counter
4. Computation of the electric field ***E***, magnetic field ***H***, and electric flux density ***D*** (Costen and Bérenger, 2009; Rouf et al., 2009a)
5. Simulation of human body either using a soft source or a hard source (Costen et al., 2009)
6. Output of the electric and magnetic field at this time step
7. Go back to the procedure step 3 unless the time loop counter is above the maximum time steps

The procedure 4 above involes the following computation:

The magnetic (***H***) field has 3 components: H_x, H_y, and H_z. For example the computation of H_x is:

$$H_x^n(i,j,k) = \frac{\mu(i,j,k)H_x^{n-1}(i,j,k)}{\mu(i,j,k)} - \frac{\Delta t}{\mu(i,j,k)}\left[\frac{E_z^n(i,j,k) - E_z^n(i,j-1,k)}{\Delta y} - \frac{E_y^n(i,j,k) - E_y^n(i,j,k-1)}{\Delta z}\right] \quad (1)$$

where Δy are Δz are the spatial discretisation in the *y* and *z* directions, respectively and $\mu\ (i,j,k)$ is the permeability at the FDTD grid (*i*,*j*,*k*). Δt is the temporal discretisation. The upper-script of *n* means $n\Delta t$. As (1) shows, $H_x^n(i,j,k)$ is calculated using the four neibouring electric ***E*** field values of $E_z^n(i,j,k)$, $E_z^n(i,j-1,k)$, $E_y^n(i,j,k)$, and $E_y^n(i,j,k-1)$. These four ***E*** values surround $H_x^n(i,j,k)$ on *x*=*i* plane. The computation of the rest of the ***H*** components are calculated in the similar manner to (1); $H_y^n(i,j,k)$ and $H_z^n(i,j,k)$ are calculated using four ***E*** values which surround $H_y^n(i,j,k)$ and $H_z^n(i,j,k)$ on *y*=*j* plane and on *z*=*k* plane, respectively. The upper limit of Δt is governed by the the Courant Friedrichs Lewy (CFL) condition (Taflove and Hagness, 2005) and The CFL stability condition is written as in (2).

$$v\Delta t \leq \left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} + \frac{1}{\Delta z^2}\right)^{-\frac{1}{2}} \quad (2)$$

where v is the highest propagation speed of the signal in the medium. Using the newly computed ***H*** in (1), the electric flux density ***D*** is calculated as in (3). (see Exhibit 1.)

The four ***H*** field values of $H_z^n(i,j+1,k)$, $H_z^n(i,j,k)$, $H_y^n(i,j,k+1)$, and $H_y^n(i,j,k)$ surrounds $D_x^{n+1}(i,j,k)$ and these values on *x*=*i* plane are used to compute $D_x^{n+1}(i,j,k)$. Similarly $D_y^{n+1}(i,j,k)$ and $D_z^{n+1}(i,j,k)$ are calculated using four ***H*** values which surround $D_y^{n+1}(i,j,k)$ and $D_z^{n+1}(i,j,k)$ on *y*=*j* plane and on *z*=*k* plane, respectively. Using the newly calculated ***D*** field values in (3), ***E*** is

Exhibit 1.

$$D_x^{n+1}(i,j,k) = \Delta t\left[\frac{H_z^n(i,j+1,k)-H_z^n(i,j,k)}{\Delta y}-\frac{H_y^n(i,j,k+1)-H_y^n(i,j,k)}{\Delta z}\right]+D_x^n(i,j,k) \tag{3}$$

Exhibit 2.

$$\begin{aligned}E_x^{n+1}(i,j,k) = {}& \frac{-\sigma_{(i,j,k)}(\Delta t)^2+4\varepsilon_0\varepsilon_{\infty(i,j,k)}\tau_{\mathrm{D}(i,j,k)}+2(\varepsilon_0\varepsilon_{\mathrm{S}(i,j,k)}+\sigma_{(i,j,k)}\tau_{\mathrm{D}(i,j,k)})\Delta t}{2\varepsilon_0\varepsilon_{\infty(i,j,k)}\tau_{\mathrm{D}(i,j,k)}+2(\varepsilon_0\varepsilon_{\mathrm{S}(i,j,k)}+\sigma_{(i,j,k)}\tau_{\mathrm{D}(i,j,k)})\Delta t+\sigma_{(i,j,k)}(\Delta t)^2}E_x^n(i,j,k)\\
&-\frac{2\varepsilon_0\varepsilon_{\infty(i,j,k)}\tau_{\mathrm{D}(i,j,k)}}{2\varepsilon_0\varepsilon_{\infty(i,j,k)}\tau_{\mathrm{D}(i,j,k)}+2(\varepsilon_0\varepsilon_{\mathrm{S}(i,j,k)}+\sigma_{(i,j,k)}\tau_{\mathrm{D}(i,j,k)}\Delta t+\sigma_{(i,j,k)}(\Delta t)^2}E_x^{n-1}(i,j,k)\\
&+\frac{2(\Delta t+\tau_{\mathrm{D}(i,j,k)})}{2\varepsilon_0\varepsilon_{\infty(i,j,k)}\tau_{\mathrm{D}(i,j,k)}+2(\varepsilon_0\varepsilon_{\mathrm{S}(i,j,k)}+\sigma_{(i,j,k)}\tau_{\mathrm{D}(i,j,k)}\Delta t+\sigma_{(i,j,k)}(\Delta t)^2}D_x^{n+1}(i,j,k)\\
&-\frac{2(\Delta t+4\tau_{\mathrm{D}(i,j,k)})}{2\varepsilon_0\varepsilon_{\infty(i,j,k)}\tau_{\mathrm{D}(i,j,k)}+2(\varepsilon_0\varepsilon_{\mathrm{S}(i,j,k)}+\sigma_{(i,j,k)}\tau_{\mathrm{D}(i,j,k)}\Delta t+\sigma_{(i,j,k)}(\Delta t)^2}D_x^n(i,j,k)\\
&+\frac{2\tau_{\mathrm{D}(i,j,k)}}{2\varepsilon_0\varepsilon_{\infty(i,j,k)}\tau_{\mathrm{D}(i,j,k)}+2(\varepsilon_0\varepsilon_{\mathrm{S}(i,j,k)}+\sigma_{(i,j,k)}\tau_{\mathrm{D}(i,j,k)}\Delta t+\sigma_{(i,j,k)}(\Delta t)^2}D_x^{n-1}(i,j,k)\end{aligned} \tag{4}$$

calculated as in (4). (see Exhibit 2) where σ , ϵ_∞ , ϵ_S , ϵ_0 , τ_D are conductivity, optical relative permittivity, static relative permittivity, permittivity in a vacuum, and the relaxation time, respectively. These values are the frequency dependent parameters based on Debye model (Debye, 1929) and the function of the Cartesian coordinate (i,j,k) in the FDTD space. They also change depending on the kind of the tissue. (4) is also used for the computation of $E_y^{n+1}(i,j,k)$ and $E_z^{n+1}(i,j,k)$ by changing x to y or z. The E field values are computed using the D values at the same place of (i,j,k).

Theoretically each voxel can have a different tissue number and each tissue has its own σ , ϵ_∞ , ϵ_S , and τ_D . The tissue number of each voxel and Debye parameters for each tissue have to be loaded into the in-house FDTD software at the initialisation step of the computation.

When the explicit FDTD code is parallelised and the FDTD space is divided into several subspaces in *z* direction using the Message Passing Interface(MPI), the computation of both ***D*** and ***H*** field values on the border of each subspace in the FDTD space requires the field values from the adjacent FDTD subspaces. The field values of D_x , D_y , H_x , H_y on the border are calculated using the field values on the *x=i* plane and the *y=j* plane. Therefore the communication between cores (*i.e.*, between subspaces) is limited mainly to H_x , H_y , E_x , and E_y on the border planes. Thus the main computation is localised well and the explicit FDTD method is inherently highly parallelisable on the distributed memory architectures and suitable for grid computing. Even when the latency between the cores is low, the parallelisation gains the computational efficiency relative to the serial code because the communication between cores is limited to a certain level. The high computational efficiency is achieved when each FDTD subspace is sufficiently large and the area of the border plane between subspaces is small. When the shape of the FDTD space is a rectangular parallelepiped, the longest side should be in the *z* direction for high computational efficiency.

One of the main data input for the computation is that of a digital human phantom. The digital human phantom is generally produced in the following procedure:

1. An entire human body is MRI-scanned. Each scan shows the cross section of the human body orthogonal to the direction of the backbone. Our research group has access to digital human phantoms in which the distance between the MRI scan is either 1 mm or 2 mm. This means a human body is scanned from the head to the feet every 1 mm or 2 mm.
2. The MRI scanned image is segmented using knowledge of medical doctors. The segmentation involves the identification of a tissue (such as bone, fat, and muscle) at each pixel in an MRI image. The size of each pixel is either 1 mm × 1 mm or 2 mm × 2 mm in our digital human phantoms. In this way, each pixel has a tissue number for example 10 for bone and 11 for heart muscle. At this stage the MRI scanned image is replaced with a stream of integers without the Cartesian coordinate. This stream of integers is in a file that has a file name that gives the height of each MRI scan phantom from the ground.
3. In some practical biomedical applications, the spatial resolution of the digital human phantom has to be 0.1 ~ 0.3 mm voxels. In such cases, we have to re-sample the original digital human phantom, applying smoothing techniques in order to produce the digital human phantom with the required spatial resolution.

The data for a 64cm×32cm×175cm-sized human torso with 0.3 mm spatial resolution has the tissue numbers for more than $1.3 \cdot 10^{10}$ voxels. Each voxel has one of 70 tissue numbers in integer. Each core reads the digital human phantom data.

Each tissue has 4 parameters of σ, ϵ_S, ϵ_∞, and τ_D for frequency dependent characteristics for the one-pole Debye modelling as shown in (4). The data on the frequency dependent characteristics of each tissue starts in the following format:

1. “Cerebellum” 0.617740 107.79900 23.3220 2.0759E-011
2. “Cerebrospinal Fluid” 2.013500 103.55500 22.1030 1.2639E-011
3. “Cornea” 0.969300 79.49400 22.6600 1.6209E-011

Here the last 4 numerical values are σ [S/m], ϵ_S, ϵ_∞, and τ_D (seconds), respectively. These values are computed through data fitting (Wuren et al., 2007) to the real measurement made by the US Air Force (Gabriel et al., 1996).

Each core needs to read the data file on these media parameters for 70 tissues in order to set the FDTD computational environment. The data of the digital human phantom is not trivial in size. Our in-house FDTD code assumes that this large digital human phantom data is in one directory which is accessible from each core (in our case via Network File System (NFS) mount) without having the copy of the same data at each core for data loading.

The numerical procedure step 6 in the above list produces data files which have the 6 field values of E_x, E_y, E_z, H_x, H_y, and H_z at each voxel together with the Cartesian coordinate of the voxel. These values are all stored in one file. This means more than 10^{11} field values in real (also termed floats or decimal values) are produced at each time step. Each core produces the data for all the voxels in its own FDTD subspace for which the core is responsible. These data files can be produced either in the directory where the job is submitted under the home directory or at any local space available at each node and specified with the input data. This demonstrates that this in-house explicit FDTD software is highly data I/O intensive.

Computational Environment

The initial code development was carried out using a very small local cluster in our office with 8 cores in 5 nodes. The 8 cores were composed of 2 single core AMD Athlon 64 4000+ and 3 dual core 4200+. Each node held 4 GB of memory and a Gigabit Ethernet with a Gigabit switch were used to connect these 5 nodes. In the case of the dual core machine, the maximum amount of the computational memory for each core was 2 GB. This memory availability sets the practical upper limit of the FDTD space that could be allocated to each core to achieve the reasonable load balancing. Thus almost the same amount of the FDTD space, which represents less than 200^3 grid points, is allocated to each core and so avoid the possibility of touching the swap space. Each node has its own local storage and all nodes share the home directory mounted by NFS. One of the machines in the cluster plays the role of the NFS server as well as the computational node.

When data output is produced under the home directory, the machine which serves as the NFS server has direct access to the hard disk whilst the rest of the machines in the cluster produce the data file under the home directory through the network data transfer. Therefore the data production from these computers, which are the NFS clients, takes more time than that from the NFS server. Thus a load imbalance occurs and the overall speed of the computation becomes significantly slower than the case when all cores produce the output data at the local hard disk, not under the home directory. The data post-processing can be carried out either with the data in one directory or with the data distributed across the cluster. The data post-processing takes more time when the data are distributed across the machines in our local cluster. A parallel job can be submitted from any of the machines in the cluster. The job submission is performed in the home directory where all the necessary input files exist.

The performance of our code was evaluated in our local cluster prior to the code porting to the HPC facility. Our analysis of the elapsed time in our code revealed that most of the elapsed time was spent on the output data file production, not on the computation. The same problem is also experienced by others in the FDTD research community (Maaskant et al., 2006) and in other scientific numerical simulations (Ross et al., 2001).

Although it is desirable to apply a high-level I/O library on top of the Message Passing Interface (MPI) - I/O interface such as ROMIO(Coti et al., 2009) configured for the Parallel File Systems (PFS) or the Parallel Virtual File System (PVFS) (Carns et al., 2000), which provides a high performance I/O infrastructure (Li et al., 2002), changing the file system of our local cluster was not possible as other researchers in the group were constantly using it for their numerical experiments.

An alternative method to speed-up of the data production is data compression. We tried Hierarchical Data Format (HDF5) as a replacement for the ASCII format which we were using. Installation of the HDF5 in our local cluster took a significant amount of time and effort even with the help from our technician. Furthermore, in order to understand the merit of HDF5, we minimised the influence of the network. This is achieved by letting one machine in the cluster handle all the data I/O. Thus, at each time step, the machine, which is in charge of the data I/O for the cluster, should produce the same number of data files as the number of cores that are in the cluster. The data was stored as a 1D array of the 6 elements for 3D space in HDF5 files. The data is chunked and shuffled to enable zlib compression. It turned out that the HDF5 showed high efficiency when more than 150 files with a size of less than 6KB, were produced using more than 2^4 cores (Abalenkovs et al., 2008).

After the performance evaluation of the code in our local cluster, we had traditionally utilised the HPC facility run by the University of Manchester. There was no need to re-code for this HPC facility. The supercomputer had 25 nodes with 16 GB of memory per node. There was no practical limit in the disk space. The job submission, storage of the input and output data were all performed under one directory in the system e.g. a home directory or a scratch space. Unfortunately the University's HPC facility had a low throughput (sometimes we had to wait for more than 2 days for a job to start running in the system). Therefore the National Grid Service (NGS) (www.ngs.ac.uk) was used for the medium sized jobs. When we used the NGS facility, the facility had about 20 nodes with 2 cores and 4 GB of memory per node in one site, although the NGS facility has improved significantly since then. The usage of the system was almost the same as the University HPC facility. The NGS did not set any practical restriction in the disk space. The file transfer from the local cluster to the NGS facility was sufficient for code porting, similar to the supercomputer in Manchester. Theis facility was just right fit to our medium-sized computation which involved the massive data I/O.

The newly developed function in our code, the capability to handle the HDF5 was not used on Manchester's supercomputer and on the NGS. In Manchester's supercomputer, the different version of the HDF was installed and there was no compatibility to the one we were using. On the NGS, HDF5 was not installed. Therefore the previous program, which required each core to produce their own data file either in ASCII or in binary, was used for the numerical experiments.

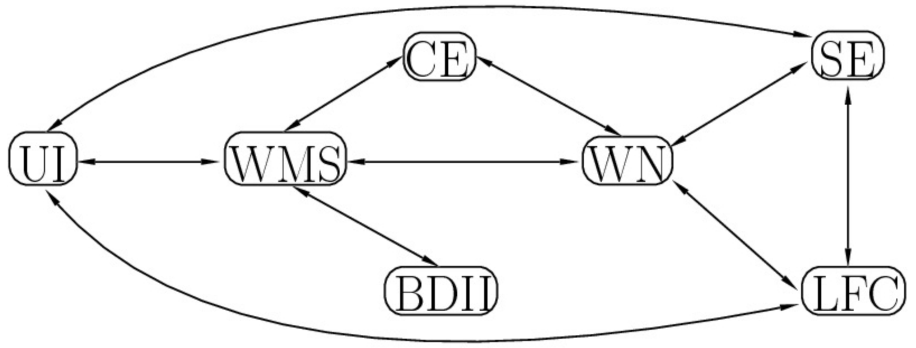
However, these two facilities were to start charging for the CPU time and disk space. Unless a steady stream of funding was obtained for these facilities, they could not be used in a stable and consistent manner. Therefore there was an imminent need to move onto another high performance facility before charging was introduced. We decided to move to Enabling Grids for E-sciencE (EGEE). The code porting took more than one year. The reason for this difficulty in porting the code is explained by understanding the computational architecture of EGEE presented in the next section on EGEE computational facility.

EGEE COMPUTATIONAL FACILITY

Introduction of EGEE

Activity of many grid-projects founded by the European Union are centred on the production of powerful and user-friendly packages, called grid middle-ware, that aggregates distributed machines and clusters turning them into one grid system. The most common grid middle-ware, called g-Lite€□ (founded within the EGEE I-II-III projects (Jones, 2005)), aggregates more than 60000 CPU from different countries in Europe (Laure et al., 2006). Another important middle-ware is called the Advanced Resource Connector (ARC) (Ellert, 2007), which is used by the researchers of the Nordic countries, and Switzerland. The Uniform Interface to Computing Resources (Unicore) (Streite, 2009) is developed by Jülich Supercomputing Centre, Germany, with national foundation. There are more middle-ware that can provide large grid systems. In the next projects the European Union will integrate them into one common and well-designed middle-ware product that will be able to provide as much computational and data capacity as possible within Europe for the scientists for many different research areas.

Figure 1. The connection between each element in the EGEE. UI, WMS, BDII, CE, WN, SE, LFC stand for User Interface, Workload Management System, Berkeley Database Information Index, Computing Element, Worker Node, Storage Element, Logical File Catalogue, respectively.



Scientists in any research areas could use a general grid system. However some organisations that provide the grid the computational resources would like to serve specific areas only. Thus the grid system could be separated to a set of communities called Virtual Organisations (VO). Any of the resources can be added to any VO, but all VO must have dedicated machines as their core grid components. For instance, the Biomed VO is available for biological applications to help research into bio-informatics; however any resource within this VO can be a member of any other VO. Users must have a membership to at least at one of the VOs to be able to submit applications to these resources.

In distributed systems security is more important than computational performance. In a general grid system, every user has to have a digital certificate signed by a regional Certificate Authority (CA), which means that CA vouches for the user's identity. This certificate can allow its owner to become a member of one or several Virtual Organisations and therefore makes a grid system safe from the illegal penetration. On the other hand, users must hide their digital certificates to prevent their identity from being stolen. To reduce the possibility of damage to a grid system being caused by stealing certificates, the digital certificate is valid only for one year. The second safety feature developed for certificates is the creation of a proxy. In other words the users do not use their certificate directly to access to the grid, but instead they use their certificate to create a temporary certificate called a proxy, and submit their applications using that proxy. The proxy is valid for one week. In the gLite middle-ware the users' certificates must be stored in a dedicated component of the grid-system, called the MyProxy Server, which generates the proxies.

Architecture of EGEE (Christodoulopoulos et al., 2008)

Figure 1 shows the connection of the major elements of the EGEE which are explained later in the chapter.

User Interface

The term "User Interface" (UI) is defined as an end-point for the users, where they can use grid-services via a set of user-level command-line tools.

It provides the main functionalities such as checking the status of the available grid-component machines and grid-resources related to the VO, the file-transfer between a user interface machine and grid storage services, job submission, checking a job's state and downloading output files. Depending on the different VO, UI can be official. When a user gets a membership to a VO community, the account of the user will be generated on the UI.

As all users have their own account in the UI, they work in their own user-space. The limited space can be a bottleneck for real and data-intensive algorithms, because the basic usage of EGEE assumes that the source code and the input files are all in the UI at the job submission stage.

Computing Element and Worker Node

From the viewpoint of job execution, the most important component of the grid is the Computing Element(CE). The CE can be defined as an end-point of a set of execution resources, *i.e.*, the Worker Nodes (WNs). The CE provides various execution queues. These are generally First-In-First-Out queues that can be used to place a wide range of jobs.

WNs are the machines,where the real jobs, such as shell scripts, Fortran, or java codes, are executed. The worker nodes are members of a distributed and shared memory cluster using the shared file-system. WNs can execute massively parallel applications such as MPI jobs. A job runs on a generated user-space (created and managed by the middle-ware) with a really strict disk space limitation (1-200 Mb). This means that one job, if it does not use remote storage services supported by the grid system in runtime, can not generate large output files that are greater than local disk space.

Workload Management System

In the gLite middle-ware every job-submission is sent to a machine called a Workload Management System (WMS), which can be used as a broker service. Generally a user does not know which resource executes his or her job prior to the job submission. A user can request that the WMS makes an automated decision, which allocates the job to the best resource based on the job's requirements. To save the WMS server from the overloading, arriving jobs are placed in a queue, and always the first job is processed and sent to its selected resource.

Storage Element

One of the advantages of grid systems is the large data-storage capability. Machines, where the storage component of the gLite middle-ware has been installed, are called Storage Elements (SE). Although the accessibility of the SE strongly depends on the network, theoretically they are accessible from every WN in every CE.

Logical File Catalogue

We usually prefer sorting our remote files (files that can be stored in several storage elements) in a logical structure, similar to a normal file-system *i.e.,* in a directory structure. The Logical File Catalogue (LFC) service provides this function. LCG can be used to show or manipulate remote files in a logical order.

Berkeley Database Information Index

The Berkeley Database Information Index (BDII) was developed to store and provide information about each available resource in the distributed system. It is used by the end-users to explore the load of the resources, and by the WMS to get the availability of a claimed resource, or make a decision which resource fits for the user's requirement.

Job Submission Procedure

The job submission requires users to follow the procedure given here:

1. Login to the official UI (generally via ssh) of the VO.
2. Upload the certificate to the MyProxy server to thus create a temporary Certificate called a proxy.
3. Copy the necessary input files and executables to the UI machine ready to submit a job.
4. Create a Job Description Language (JDL) file to describe the executable files, the input files, command line arguments, and any restrictions for the resource where the job should be executed.

After this user's procedure, the following procedure is taken by the drig and its resource broker to run a job:

1. All the files described in the JDL file are transferred to the WMS server.
2. The WMS gets information by querying BDII server to be able to determine which CE should be used.
3. The WMS creates log entries to allow status-checking and debugging of the system.
4. The WMS matches the user's requirements with the available resources and sends the job to the possible CE.
5. The CE transfers the job to the Worker Nodes (WN).
6. The WN accesses the files in SE directly using an API or LFC server.
7. When a job generates a remote output file in the SE, a command-line provided by gLite can be used to download files (not a directory) to anywhere.

Adaptation of Our Code to EGEE

Disk Restriction to Run a Job in WNs

Each WN has a very limited amount of disk space. As described in the section on the nature of the computation of the FD-FDTD methods, each core reads a file with a size of 1GB or more to make up the data for the entire digital human phantom. This file size easily exceeds the upper-limit of each WN's disk space. Therefore the one large file (one digital human phantom) was separated in the *z* direction, *i.e.*, the direction of the human height. For example, if a 175 cm-height digital human phantom is meshed every 1mm, then there are 1750 files to read at the beginning of the FDTD computation. Since the parallelisation is implemented in the *z* direction, each core reads the *z*= a constant number of slices which the core needs. For example, if there are 10 cores, then each core reads 175 files.

Solution Using SE

The large output data can not be left on the Worker Nodes in the CE. The obvious and immediate solution to use the EGEE was to make use of the SE. We implemented several shell scripts to move the output data file from each Worker Node in the CE to the SE on-the-fly. At the end of each data production step, every WN transfers its output to the SE and deletes the output file in WNs as soon as this file copy is completed. With this approach, there is a high possibility that many of the WNs will try to access the same harddrive in the SE at the same time. After the completion of the job, all output files were transferred to the local cluster in Manchester for data post-processing.

Unfortunately this approach did not work perfectly. Ideally all files produced in the CE should have been transferred to the SE. However, 20-30% of the output files were left in CE and were not copied into the SE. Intensive investigation revealed that there were some bugs in the lcg-cp command itself. This means that we should not rely on the Unix command lcg-cp for the file transfer and that we should have an alternative storage space for the transfer of the output data from the WNs. Our local cluster in Manchester had disk space enough to accommodate the massive data produced on the EGEE. Since the Unix command lines which the EGEE provides does not include the communication between the WNs in the CE and the computers outside the EGEE, we had to create our own mechanism for this data transfer.

Solution Without SE

After a long period of trial and error, the final solution to our main problem of how to handle the large data I/O was to use the direct communication between the WNs and our local cluster. This section describes the final solution in detail by showing the shell scripts which can be adapted to any similar situation and is independently to the programming language e.g., C or Fortran.

We first created the RSA private and public key on the UI machine by typing 'ssh-keygen -t rsa' in order to perform the file access or the file transfer from the UI machine to the local cluster in Manchester without any passwords. The public key (id_rsa.pub) on the UI machine is copied into '$HOME/.ssh/authorized_keys' on the local machine in Manchester. The private key should be cent to the CE together with the job. Hence the WNs can access our local cluster in Manchester with this key. To make this method work, the file 'known_hosts'has to be placed in the directory '$HOME/.ssh/' on each WN. To achieve this To achieve this, our Fortran code calls the following shell script immediately after the MPI initialisation:

```
#!/bin/sh
mkdir $HOME/.ssh/
cp 'pwd'/known_hosts $HOME/.ssh/
known_hosts
chmod 400./id_rsa
```

All the participating WNs call this shell script. This shell script makes the secure copy possible between the WNs and our local cluster using the Unix command 'scp'.

Every time a large output data file is produced at each WN the Fortran code calls the shell script presented in Appendix A to transfer the data file from the WNs to our local cluster in Manchester.

This shell script is named copy.sh and called just after each data is produced as follows:

```
open(unit=70+rank, file=dirfile,
access="sequential", &
form="formatted",status="new",iostat
=err)
do k=zs, zf
do i=0, nx+1
do j=0, ny+1
write(70+rank,"(3I6, 1E11.3)") i, j,
k+zi, &
Hx(i,j,k),Hy(i,j,k),Hz(i,j,k),Ex(i,j,
k),Ey(i,j,k),Ez(i,j,k)
end do
end do
end do
cmd = "chmod 775./copy2SE.sh;./
copy2SE.sh " // dirfile
call system(cmd)
close(unit=70+rank)
```

Here, the variable 'rank' is the number of each core in the MPI-world. It is important that the shell script 'copy.sh' has to be called before the file port closure at 'close(unit=70+rank)'.

The line 'sleep 5' in 'copy.sh' needs special care; if this value is too small and with a heavily loaded network or long-distance file transfer, some files may not be transferred.

An almost identical shell script presented in Appendix B is called just before reading the input data in order to transfer the input data from our local cluster in Manchester to the WNs. The difference between the two shell-scripts presented in Appendix A and Appendix B is that the shell-script, which copies files from the local cluster to the WNs, does not include 'rm $1' because the input files stored in our local cluster in Manchester were not to be deleted. After the data was successfully loaded from one file at each WN, the data file in the WNs is swiftly deleted by calling the Unix command from the Fortran code.

The JDL file job.jdl for this job reads:

```
JobType = "MPICH";
NodeNumber = 30;
Executable = "mpi-start-wrapper-f90.
sh";
Arguments = "fdtd-3d OPENMPI";
StdOutput = "mpi-test.out";
StdError = "mpi-test.err";
InputSandbox = {"all.tar.gz","mpi-
start-wrapper-f90.sh"};
OutputSandbox = {"mpi-test.err","mpi-
test.out"};
Environment = {"LFC_HOST=lfc-biomed.
in2p3.fr","LCG_CATALOG_TYPE=lfc"};
RetryCount = 10;
Requirements =
Member("MPI-START", other.GlueHostAp-
plicationSoftwareRunTimeEnvironment)
&& Member("OPENMPI", other.Glue-
HostApplicationSoftwareRunTimeEnvi-
ronment)
&& other.GlueCEInfoHostname=="grid10.
lal.in2p3.fr"
```

Here, the compressed file 'all.tar.gz' includes the Fortran code, the Makefile for the Fortran code, the two shell scripts mentioned above, the files called 'id_rsa', ' known_hosts', and a shell script used to compile the Fortran code with the Makefile. When this JDL file is submitted via the command 'glite-wms-job-submit -a -c glite_wms.conf job.jdl', the file 'mpi-start-wrapper-f90.sh' and the file 'all.tar.gz' are placed on each WN. Then the shell script mpi-start-wrapper-f90.sh on each WN acts by:

1. Un-zips and then un-tars the file 'all.tar.gz';
2. Runs a shell script to compile the source code;
3. And finally runs the MPI job using 30 cores.

Arguments suggests the arguments for the shell script 'mpi-start-wrapper-f90.sh' and the standard output message and the error messages of the job are written into the output files s 'mpi-test.out' and 'mpi-test.err'. They can be downloaded at the UI after the job is completed using the Unix command:

```
'glite-wms-job-get-output' provided
by gLite.
```

In this example the JDL file, the CE is specific to one site. At this site, there are 80 cores and each core has 2GB of memory. The level of the memory availability is the same as the local cluster in Manchester. However, at another site (gridgate.cs.tcd.ie), there are 768 cores but each core has 0.5 GB of memory. As mentioned in the section on Computation in Electromagnetics, this in-house software achieves the high computational efficiency when the each FDTD subspace is large enough to cover the cost of the communication between cores. When we use 'gridgate.cs.tcd.ie' instead of 'grid10.lal.in2p3.fr' as a CE for the same amount of the total FDTD space (*i.e.*, the same amount of total memory usage), we need 4 times as many cores as required at 'grid10.lal.in2p3.fr', leading to the lower throughput than the site at 'grid10.lal.in2p3.fr' and the computational efficiency is less than the site at 'grid10.lal.in2p3.

fr' due to the increase of the ratio of border area/(FDTD subspace).

The result from this data input/output approach is very promising because:

1. All (100%) of the output/input files are transferred between the local cluster and the EGEE site.
2. The overall elapsed time for a job to complete including the data transfer is significantly shorter than the approach using the SE and LFC.

Although this approach was developed to run our program on the EGEE, it is entirely applicable to any other case where each core produces a significant amount of data at a separate disk space and the disk space is limited at each core.

FUTURE RESEARCH DIRECTION

In order to perform further speed-up of the computation for the practical use, the computation method of the Maxwell equations and the method to handle the data I/O has to be improved.

Improvement of the Computational Algorithm

The explicit scheme is a simple algorithm and highly parallelisable, ideal for a grid computing environment. However, since the temporal discretisation cannot be set independently of the spatial discretisation, one has to run the many FDTD iterations for the fine spatial sampling. This is an inefficiency of the FDTD computation that can be overcome by modifying the algorithm. One of the ways to do so is the development of implicit methods by removing the CFL stability condition. These implicit schemes can set an arbitrary Δt independently of Δs for a stable computation. The Crank-Nicolson implicit method (Crank and Nicolson, 1947) was proposed for this purpose. However, the Crank-Nicolson implicit method includes solving a huge sparse matrix. Therefore at the time when it was invented the ordinary computers were not able to run the Crank-Nicolson code with ease.

In order to solve the Crank-Nicolson implicit method without handling the sparse matrix, a mathematical error was deliberately put into the original Crank-Nicolson based Maxwell's equations and the method successfully removed the sparse matrix(Peaceman and Rachford Jr., 1955) from the algorithm. Instead it handles the tri-diagonal matrix (Douglas, 1955). This alternating direction implicit technique was applied to the FDTD method (Zheng et al., 1999; Costen and Thiry, 2004) and became the standard implicit method. However, the ADI-FDTD method is difficult to implement serially and furthermore in a parallel manner due to the heavy communication between the cores. Therefore there are many other implicit schemes that are currently studied(Rouf et al., 2009b).

In reality, the increase of Δt escalates the numerical error. Therefore in the case of the implicit scheme, the practical upper limit of Δt is set by relation to an acceptable level of numerical noise. This means the implicit scheme itself is not going to achieve a dramatic improvement in computational speed. The implicit scheme shines when the FDTD space needs to be meshed very finely (but usually not the entire FDTD space needs to be meshed); a very localised space needs to be meshed finely; otherwise a coarse meshing is acceptable. In this case, the subgridding techniques (Bérenger, 2006; Costen and Bérenger, 2009) can be used to reduce the total number of voxels. The reduction of the total voxel numbers can also be achieved by using a very efficient and powerful boundary condition (Bérenger, 2007). Usually the boundary condition is one of the most computationally expensive places in the whole computation. Thus utilisation of the most computationally efficient boundary condition is of primary importance.

Improvement on the Data Input and Output

As mentioned, the digital human phantom was sliced in the direction of its height. As you can imagine, the data gradually changes slice by slice in *z* direction. The difference between the slices can be significantly smaller than the size of the original data. Thus, one way to improve the data input is a reduction in the data algorithmically. The information which gives the difference from the neibouring slice can be used rather than the raw digital human phantom. Although this method requires the computation to reproduce the digital human phantom in the FDTD code, the computation cost in time for this will be still significantly smaller than the data transfer cost saving in time. In order to reduce the amount of the data output, high data compression techniques such as HDF can be used as mentioned in the section on Computational environment. However, the implementation of HDF in the source code is not straightforward and HDF is not always available. Therefore a method, which uses the general machine-independent tool such as very general Linux/Unix commands, should be invented.

CONCLUSION

Many researchers who make use of high performance computing facilities face the situation where their computational architecture is changed either suddenly or gradually over time. This can be intentional or out of their control. In either case, this is the nature of computational research because the computational cost, the computational performance, and the funding for their research activities changes all the time. To be able to survive under any circumstances the researchers, who implement their algorithms in software, have to pay attention to the possible re-use of their software, the ease of reading the software, the computational efficiency, and the portability.

This chapter first introduced the nature of our computation by presenting the core equations. It is highly parallelisable, suitable for grid computing apart from the need for the high data I/O and large computational memory per core.

EGEE was chosen to be our next computational facility because it is available to our research group free of charge. However, EGEE, as it is, was not suitable for the application which has a high demand on data I/O.

This problem and the solution to our code porting to a new and novel computational architecture of EGEE are the main points of this chapter. The same solution can be taken by people who are suffering from the computational architecture with weak data I/O.

Finally some suggestions on the performance improvement were proposed from the viewpoint of the computational algorithm and the data I/O.

ACKNOWLEDGMENT

The authors wish to thank Dr. Matthieu Bonilla at EADS Nucletudes, France for the constructive technical discussions for the parallel code production and performance improvement, Pierre Girard, Yannick Legre, Carles Loomis, Stephen Childs for the practical advice on the usage of EGEE.

REFERENCES

Abalenkovs, M., Costen, F., Lucas, C., & Brown, A. (2008). *Data format selection for an I/O-intensive large-scale FDTD*. In IEEE International Symposium on Antennas and Propagations USNC/CNC/URSI Radio Science.

Bérenger, J.-P. (2006). A Huygens subgridding for the FDTD method. *IEEE Transaction on Antennas and Propagation, 54*.

Bérenger, J.-P. (2007). On the Huygens absorbing boundary conditions for electromagnetics. *Journal of Computational Physics*, *226*, 354–378. doi:10.1016/j.jcp.2007.04.008

Carns, P., Ligon, W. B., III, Ross, R. B., & Thakur, R. (2000). *PVFS: A parallel file system for linux clusters*. In 4th Annual Linux Showcase Conference (pp. 317–328).

Christodoulopoulos, K., Gkamas, V., & Varvarigos, E. A. (2008). Statistical analysis and modeling of jobs in a grid environment. *Journal of Grid Computing*, *6*, 77–102. doi:10.1007/s10723-007-9089-1

Costen, F., & Bérenger, J.-P. (2009). *Extension of the FDTD Huygens subgridding to frequency dependent media*. Annals of Telecommunications.

Costen, F., Bérenger, J.-P., & Brown, A. (2009). Comparison of FDTD hard source with fdtd soft source and accuracy assessment in debye media. *Transaction on Antennas and Propagation*, *57*, 2014–2022. doi:10.1109/TAP.2009.2021882

Costen, F., & Thiry, A. (2004). Alternative formulation of three dimensional frequency dependent ADI-FDTD method. *IEICE Electronics Express*, *1*, 528–533. doi:10.1587/elex.1.528

Coti, C., Herault, T., & Cappello, F. (2009). *MPI applications on grids: A topology aware approach*. (LNCS 5704, pp. 466–477). Berlin/Heidelberg, Germany: Springer.

Crank, J., & Nicolson, P. (1947). A practical method for numerical evaluation of solutions of partial differential equations of the heat conduction type. *Cambridge Philosophical Society*, *43*, 50–67. doi:10.1017/S0305004100023197

Debye, P. (1929). *Polar molecules*. New York, NY: Dover.

Douglas, J. Jr. (1955). On the numerical integration of Uxx+ Uyy= Utt by implicit methods. *Journal of the Society for Industrial and Applied Mathematics*, *3*, 42–65.

Ellert, M. (2007). Advanced resource connector middleware for lightweight computational grids. *Future Generation Computer Systems*, *23*, 219–240. doi:10.1016/j.future.2006.05.008

Gabriel, S., Lau, R. W., & Gabriel, C. (1996). The dielectric properties of biological tissues: Iii. Parametric models for the dielectric spectrum of tissues. *Physics in Medicine and Biology*, *41*, 2271–2293. doi:10.1088/0031-9155/41/11/003

Jones, B. (2005). *An overview of the EGEE project* (LNCS 3664, pp. 1–8). Berlin/Heidelberg, Germany: Springer.

Laure, E., Gr, C., Fisher, S., Frohner, A., Kunszt, P., & Krenek, A. (2006). Programming the grid with gLite. *Computational Methods in Science and Technology*, *12*(1), 33–45.

Li, J., Liao, W., Choudhary, A., & Taylor, V. (2002). *I/O analysis and optimization for an AMR cosmology application*. In IEEE International Conference on Cluster Computing (pp. 119–126).

Luebbers, R. J., Hunsberger, F., & Kunz, K. S. (1991). A frequency-dependent finite difference time domain formulation for transient propagation in plasma. *IEEE Transactions on Antennas and Propagation*, *39*, 29–34. doi:10.1109/8.64431

Maaskant, R., Ivashina, M., Mittra, R., Yu, W., & Huang, N. (2006). *Parallel FDTD modeling of a focal plane array with vivaldi elements on the highly parallel LOFAR BlueGene/L Supercomputer*. In IEEE International Symposium on Antennas and Propagation.

Margetts, L., Pettipher, M. A., & Smith, I. M. (2004). *Parafem - performance of a suite of finite element analysis codes on the cray x1*. In 46th International Cray User Group Conference.

Peaceman, D. W., & Rachford, H. H. Jr. (1955). The numerical solution of parabolic and elliptic differential equations. *Journal of the Society for Industrial and Applied Mathematics*, *3*, 28–41. doi:10.1137/0103003

Ross, R., Nurmi, D., Cheng, A., & Zingale, M. (2001). A case study in application I/O on Linux clusters. In *Proceedings of the 2001 ACM Conference on Supercomputing*.

Rouf, H., Costen, F., & Garcia, S. (2009a). 3D crank-nicolson finite difference time domain method for dispersive media. *IET Electronics Letters*, *45*, 961–962. doi:10.1049/el.2009.1940

Rouf, H., Costen, F., Garcia, S., & Fujino, S. (2009b). On the solution of 3-D frequency dependent crank-nicolson FDTD scheme. *Journal of Electromagnetic Waves and Applications*, *23*, 2163–2175. doi:10.1163/156939309790109261

Streite, A. (2009). *UNICORE: Getting to the heart of Grid technologies*. British Publishers Ltd.

Taflove, A., & Hagness, S. (2005). *Computational electromagnetics. The finite-difference time-domain method*. Boston, MA: Artech House.

Wuren, T., Takai, T., Fujii, M., & Sakagami, I. (2007). Effective 2-debye-pole FDTD model of electromagnetic interaction between whole human body and UWB radiation. *IEEE Microwave Wireless Components Letters*, *17*, 483–485. doi:10.1109/LMWC.2007.899295

Zheng, F., Chen, Z., & Zhang, J. (1999). A finite difference time domain method without the courant stability conditions. *IEEE Microwave Guided Letters*, *9*, 441–443. doi:10.1109/75.808026

ADDITIONAL READING

Andreetto, P., Andreozzi, S., Ghiselli, A., Marzolla, M., Venturi, V., & Zangrando, L. (2010). Standards-based job management in grid systems. *Journal of Grid Computing*, *8*, 19–45. doi:10.1007/s10723-010-9146-z

Bérenger, J.-P. (2006). *A Huygens Subgridding for the FDTD method* (*Vol. 54*). IEEE Transaction on Antennas and Propagation.

Bérenger, J.-P. (2007). On the Huygens absorbing boundary conditions for electromagnetics. *Journal of Computational Physics*, *226*, 354–378. doi:10.1016/j.jcp.2007.04.008

Brooke, J. M., Marsh, J., Pettifer, S., Sastry, L. S., 2006. The importance of locality in the visualization of large data sets. Concurrency and Computation: Practice and Experience.

Carns, P., Ligon, W. B., III, Ross, R. B., & Thakur, R. 2000. PVFS: A parallel file system for linux clusters. In: 4th Annual Linux Showcase Conference. pp. 317–328.

Carver, G., Roy, K., & Stringfellow, N. 2005. Using single sided communications to aid load balancing. In: SGI User Group.

Costen, F., & B'erenger, J.-P. (2009). *Extension of the FDTD Huygens subgridding to frequency dependent media*. Annals Telecommunication.

Coveney, P. (1833). 2005. Scientific grid computing. *Philosophical Trans. Royal Soc. A*, *363*, 1707–1713. doi:10.1098/rsta.2005.1632

Foster, I., 2002. What is the grid? a three point checklist. Technical report, GRIDToday.

Gagliardi, F., Jones, B., Grey, F., Bgin, M.-E., Heikkurinen, M., 2005. Building an infrastructure for scientific grid computing: status and goals of the EGEE project. Philosophical Transactions of the Royal Society. A 363 (1833),pp. 1729–1742.

Germain-Renaud, C., Loomis, C., Mo'scicki, J., & Texier, R. (2008). Scheduling for responsive grids. *Journal of Grid Computing*, *6*, 15–27. doi:10.1007/s10723-007-9086-4

Guiffaut, C., & Mahdjoubi, K. (2001). A parallel FDTD algorithm using the MPI library. *IEEE Antennas and Propagation Magazine*, *43*, 94–103. doi:10.1109/74.924608

Guy, A. Schiavone, G. A., Codreanu, I., Palaniappan, R., Wahid, P., 2000. FDTD speedups obtained in distributed computing on a Linux workstation cluster. In: IEEE International Symposium on Antennas and Propagations USNC/CNC/URSI Radio Science Meeting.

Jimenez-Peris, R., Patino-Martinez, M., & Kemme, B. (2007). Enterprise grids: Challenges ahead. *Journal of Grid Computing*, *5*, 283–294. doi:10.1007/s10723-007-9071-y

Jones, B. 2005. An overview of the EGEE project. Vol. 3664. Springer Berlin / Heidelberg, pp. 1–8.

Kenn, E., Coghlan, B., Tsouloupas, G., Dikaiakos, M., Walsh, J., Childs, S., et al. 2005. Heterogeneous Grid Computing: Issues and Early Benchmarks. Vol. 3516. Springer Berlin / Heidelberg, pp. 870–874.

Kong, J. A. (1975). *Theory of Electromagnetic Waves*. John Wiley.

Lingrand, D., Montagnat, J., Martyniak, J., & Colling, D. 2009. Analyzing the EGEE Production Grid Workload: Application to Jobs Submission Optimization. Vol. 5798. Springer Berlin / Heidelberg, pp. 37–58.

Montagnat, J., Glatard, T., Plasencia, I., Castej'on, F., Pennec, X., & Taffoni, G. (2008). Workflow-based data parallel applications on the egee production grid infrastructure. *Journal of Grid Computing*, *6*, 369–383. doi:10.1007/s10723-008-9108-x

Pacitti, E., Valduriez, P., & Mattoso, M. (2007). Grid data management: Open problems and new issues. *Journal of Grid Computing*, *5*(3), 273–281. doi:10.1007/s10723-007-9081-9

Pickles, S. M., Blake, R. J., Boghosian, B. M., Brooke, J. M., Chin, J., Clarke, P. E. L., et al. 2004. The teragyroid experiment. In: Workshop on Case Studies on Grid Applications.

Rouf, H., Costen, F., Garcia, S., & Fujino, S. (2009). On the solution of 3-D frequency dependent crank-nicolson fdtd scheme. *Journal of Electromagnetic Waves and Applications*, *23*, 2163–2175. doi:10.1163/156939309790109261

Taflove, A., & Hagness, S. (2005). *Computational Electromagnetics. The finite-difference Time-domain method*. Boston, MA: Artech House.

Varadajaran, V., & Mittra, R. (1994). Finite-difference time domain analysis using distributed computing. *IEEE Microwave Guided Wave Letters.*, *4*, 144–145. doi:10.1109/75.289515

Vazquez-Poletti, J.-L., Huedo, E., Montero, R., & Llorente, I. 2006. Execution of a Bioinformatics Application in a Joint IRISGrid/EGEETestbed. Vol. 3911. Springer Berlin / Heidelberg, pp. 831–838.

Walsh, J., Coghlan, B., & Childs, S. 2010. An Introduction to Grid Computing Using EGEE. Vol. 791. Springer Berlin /Heidelberg.

Wang, J., Fujiwara, O., Watanabe, S., & Yamanaka, Y. (2004). Computation with a parallel FDTD system of human-body effect on electromagnetic absorption for portable telephones. *IEEE Transactions on Microwave Theory and Techniques*, *52*, 53–58. doi:10.1109/TMTT.2003.821232

Weedon, W. H., & Rappaport, C. M. (1997). A general method for FDTD modeling of wave propagation in arbitrary frequency dispersive media. *IEEE Transactions on Antennas and Propagation*, *45*, 401–410. doi:10.1109/8.558655

Yu, W., Liu, Y., Su, T., Hunag, N.-T., & Mittra, R. (2006). A robust parallel conformal finite-difference time-domain processing package using the mpi library. *IEEE Antennas and Propagation Magazine*, *47*, 39–49.

KEY TERMS AND DEFINITIONS

Code Porting: Usually a software/code is developed with a specific computational environment in mind. Therefore when a code written in one computational environment is going to be run in another computational environment, some part of the code has to be modified. The activity to make a code that runs on one machine usable in the other machine is called code porting.

Computational Electromagnetics: Research on the electromagnetic wave propagation using the computational facility

Enabling Grids for E-sciencE(EGEE): A grid-project that provides large computational resources connected via the Internet and founded by European Union. Applications must be parallelized in order to benefit from EGEE.

Finite Difference Time Domain (FDTD) Method: One of the most simple and powerful method to solve Maxwell equations for the numerical simulation of the ElectroMagnetic(EM) wave propagation. Maxwell equations are temporally and spatially discretised. The basic equations are repeatedly executed at each FDTD iteration. The outcome of the FDTD simulation is the signal signature(waveform) in time domain.

Grid Computing: Computations using a computational grid facility that consists of many computational or data-store resources. The software program that contains the computation has to take into account the fact that there are many computational cores in many nodes.

Job Management: Scheduling and managing computational requests to a particular computing element. In cases where there are more than one user in a grid computing facility, their computational requests (jobs) are submitted to one location. These jobs are ranked in priority and placed at the individual computational nodes.

Message Passing Interface: A programming language-independent specification that provides a multimode communication protocol. Data Input/Output: activity to read/load data from data files and to produce data files of the computation result.

Numerical Methods: The study on how to solve equations accurately and efficiently from the viewpoint of the computation.

Parallel Code: When there are many cores in many nodes in a computational environment, a code which uses a single core for computation can be modified so that a big task in the code can be divided into many little tasks and each little task is handled by one node and many nodes work for this single and big task at the same time. The code modified in this way is called parallel code which can make use of more than one machines in a single run and run on more than one machine simultaneously.

Security: Some information on the users and/or the administrators in the computational facility has to be kept secret. Security addresses the methods available that keep the computational facility, including the private data within the facility safe.

APPENDIX A

A shell script to transfer a data file from a worker node to a computer outside EGEE:

```
#!/bin/sh
./VariableSetting.sh
REMOTE_FOLDER=”$USER”@$MACHINE
FILENAME=$1
CUT_FILENAME=’echo $1 | cut –d/ -f2’
LOCAL_FOLDER=’pwd’
       COUNT=1
       while [ $COUNT –lt 10 ];
       do
sleep 5
scp -vC -P $PORT –I ‘pwd’/id_rsa
$LOCAL_FOLDER/$FILENAME $REMOTE_FOLDER:$REMOTE_PATH$CUT_FILENAME
                    SUCCESS=$?
                    if [ SUCCESS –EQ 0 ];
                    then
                             COUNT=10
                    else
                             COUNT=’ expr $COUNT + 1 ‘
                       fi
        echo $COUNT
        done
rm $1
```

APPENDIX B

A shell script to transfer a data file from a computer outside EGEE to a worker node:

```
#1/bin/sh

./VariableSetting.sh

REMOTE_FOLDER=”$USER”@$MACHINE”

FILENAME=$1

CUT_FILENAME=’ECHO $1 | cut –d/ -f2’

LOCAL_FOLDER=’pwd’

COUNT=1

while [ $COUNT –lt 10 ];

do

sleep 0.01

scp –C –P $PORT –I ‘pwd’/id_rsa

$REMOTE_FOLDER:$REMOTE_PATH$CUT_FILENAME $LOCAL_FOLDER/$FILENAME

SUCCESS=$?

if [ $SUCCESS –eq 0 ];

              then

              COUNT=10

              else

              COUNT=’ expr $COUNT + 1 ‘

              fi

done
```