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\textsuperscript{a}, Ákos Balaskő\textsuperscript{b}

\textsuperscript{a}School of Electrical and Electronic Engineering, The University of Manchester, M60 1QD, U.K., f.costen@cs.man.ac.uk
\textsuperscript{b}Computer and Automation Research Institute, Hungarian Academy of Sciences, H-1132 Budapest, Víctor Hugo u. 18-22. Room 025., Hungary, balasko@sztaki.hu

Abstract

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

This chapter introduces the computational architecture of Enabling Grids for E-sciencE which made our code porting very challenging and the discussion presented is directly applicable to EGEE users. The final solution to the code porting problem is proposed and its performance is examined. The solution of this problem, which may be generally faced in the other large scale computation, can be applicable to users of any 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.

1. Introduction

Research of distributed and parallel systems is one of the most important areas in computer science. This area is based on the claim of exploitation of the large computational and data storing capabilities. While the main components i.e., processors and hard drives in a single computer are going to be smaller with larger storage and higher processing performance as time goes, distributed systems can integrate these individual resources into one large, heterogeneous, dynamic system to allow 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 resources connected via Internet for the researchers in the natural science and the engineering who have applications with more demanding processing time, or storing data than a single machine can accommodate. Certainly, these applications must be parallelised to 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 would also be different. Furthermore, the cost, the performance and the suitability are always changing over time. Therefore researchers have to be prepared for the change of the computational facility and have to be able to adjust ourselves in the new computational environment.

This chapter shares authors’ experience of the significant change of the computational environment for the daily research activity and provide some hints to those who may face the similar situation to the one presented in this chapter.

The authors’ experience is based on the Enabling Grids for E-Science (EGEE) project. The EGEE project-family, that is founded by the European Commission, started on April 2004. It had provided academic and industrial researchers the means to have access to large computing resources. It focused to develop and maintain a robust and powerful grid network with its components, and to attract new users from industry by standardized trainings 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 30 2010. The new project is created to continue the development of distributed systems
internationally in Europe is called European Grid Initiative(EGI). In the project all of the old organization-idea was 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.

One other but not 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 in different grid-middleware.

Section 2 talks about the motivation of our research and introduce the core part of our equations to
understand the nature of our computation. Furthermore Section 2 talks about the computational environment
before we faced a significant change of the environment. Section 3 introduces the computational architecture
of Enabling Grids for E-sciencE(EGEE), which is significantly different from the architectures which we
handled before adaptation to EGEE. Section 4 describes the problems which we had faced and present the
solutions. Section 5 gives some insight and suggestion for the improvement of the computational algorithms
as well as the algorithms which could be applied for the data I/O problems.

2. Computation in Electromagnetics

2.1. Necessity of computation in electromagnetics

One of our research activities in University of Manchester studies the biomedical problems associated
with human-body stimulation by means of numerical modelling and simulations. Our research aims in-
volve the development of a next generation technology of broadband electromagnetics for bioengineering
modelling and simulation for health care technology. For example, in the films, some scene involves the
defibrillation of a fainted person. The defibrillation with the electric shock on the torso is usually successful
in the films. However, in reality, the successful rate is not as high as the one you watch in the films. This
is because the location, the shape, the excitation waveform of the electrodes are not optimum to the person
for whom the electrical shock is applied. The optimum method to apply the electric shocks depends on the
age, sex, size and shape of the person. The currently exercised therapy 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 the defibrillation is not fully known because of lack of knowledge.

A method 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 simulation will be able to

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

The numerical simulation involves the propagation of the 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 the
knowledge on the way to excite particular parts of the heart, aiming at the increase of the success rate of the
defibrillation.

The computer simulation should be able to handle arbitrarily-complex and very fine geometry, wide
frequency range and frequency dependent materials in time domain for the comprehensive study. 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 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.

2.2. 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. stimulation 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 involves the following computation:

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

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

where \( \Delta y \) and \( \Delta z \) are the spatial discretisation in the \( y \) and \( z \) directions, respectively and \( \mu(i,j,k) \) is the permeability at the FDTD grid \((i, j, k)\). \( \Delta t \) 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 \( E_x^n(i,j,k), E_x^n(i,j-1,k), E_x^n(i,j,k), \) and \( E_x^n(i,j,k-1) \). 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 \( H_x^n(i,j,k) \) and \( H_y^n(i,j,k) \) on \( y = j \) plane and on \( z = k \) plane, respectively. The upper limit of \( \Delta t \) is governed by the the Courant Friedrichs Lewy (CFL) condition (Taflove and Hagness, 2005) and The CFL stability condition is written as in (2).

\[
\nu\Delta t \leq \left( \frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} + \frac{1}{\Delta z^2} \right)^{-\frac{1}{2}}
\]

where \( \nu \) 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).

\[
D_x^{n+1}(i,j,k) = \frac{\Delta t}{\Delta y} \left( \frac{H_x^n(i,j+1,k) - H_x^n(i,j,k)}{\Delta y} \right) - \frac{\Delta t}{\Delta z} \left( \frac{H_x^n(i,j,k+1) - H_x^n(i,j,k)}{\Delta z} \right) + D_x^n(i,j,k)
\]
The four \( H \) field values of \( H_x^{n+1}(i,j,k) \), \( H_y^{n+1}(i,j,k) \), \( H_z^{n+1}(i,j,k) \), and \( H_z^{n+1}(i,j,k+1) \) 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 calculated as in (4).

\[
E_x^{n+1}(i,j,k) = -\sigma_x(i,j,k)(\Delta t)^2 + 4\varepsilon_0\varepsilon_{ro(i,j,k)}\tau_x(i,j,k) + 2(\varepsilon_0\varepsilon_{ro(i,j,k)} + \sigma_x(i,j,k)\tau_x(i,j,k))\Delta t + \sigma_x(i,j,k)(\Delta t)^2E_x^n(i,j,k)
\]

(4)

where \( \sigma_x \), \( \varepsilon_{ro} \), \( \varepsilon_S \), and \( \tau_D \) 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 \( \sigma_x \), \( \varepsilon_{ro} \), \( \varepsilon_S \), and \( \tau_D \). The tissue number of each voxel and Debye parameters for each tissue have to be loaded onto 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 \( D_1, D_2, H_1, H_2 \) on the border are calculated using the field values on \( x = i \) plane and \( y = j \) plane. Therefore the communication between cores (i.e., between subspaces) is limited mainly to \( H_1, H_2, E_1 \), and \( E_2 \) on the border planes. Thus the main computation is well-localised 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 achievable 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 input data for the computation is a digital human phantom. The digital human phantom is generated 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 the 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 the knowledge of medical doctors. The segmentation involves the identification of a tissue (such as bone, fat, muscle) at each pixel in an MRI image. The size of each pixel is either 1 mm \( \times \) 1 mm or 2 mm \( \times \) 2 mm in our digital human phantoms. In this way, each pixel has a tissue number such as 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 whose file name usually shows the height of each MRI scan image 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 voxel. 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 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. Each tissue has 4 parameters of $\sigma$, $\epsilon_S$, $\epsilon_\infty$, and $\tau_D$ 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:

<table>
<thead>
<tr>
<th>No.</th>
<th>Tissue Type</th>
<th>$\sigma$ [S/m]</th>
<th>$\epsilon_S$</th>
<th>$\epsilon_\infty$</th>
<th>$\tau_D$ [seconds]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot;Cerebellum&quot;</td>
<td>0.617740</td>
<td>107.79900</td>
<td>23.3220</td>
<td>2.0759E-011</td>
</tr>
<tr>
<td>2</td>
<td>&quot;Cerebrospinal Fluid&quot;</td>
<td>2.013500</td>
<td>103.55500</td>
<td>22.1030</td>
<td>1.2639E-011</td>
</tr>
<tr>
<td>3</td>
<td>&quot;Cornea&quot;</td>
<td>0.969300</td>
<td>79.49400</td>
<td>22.6600</td>
<td>1.6209E-011</td>
</tr>
</tbody>
</table>

Here the last 4 numerical values are $\sigma$ [S/m], $\epsilon_S$, $\epsilon_\infty$, and $\tau_D$ (seconds), respectively. These values are computed through data fitting (Wuren et al., 2007) to the real measurement by 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 their data loading.

The numerical procedure step 6 above produces the 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 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 at each node specified as a part of the input data. As seen, this in-house explicit FDTD software is highly data I/O intensive.

2.3. Computational environment

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

When the data output is produced under the home directory, the machine which serves as the NFS server has the 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 the 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 is 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 the most of the elapsed time was spent...
on the output data file production, not 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 for their numerical experiments.

Alternative method toward the speed-up of the data production is the data compression. We tried Hierarchical Data Format (HDF5) as a replacement of the ASCII format which we have used. 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, are supposed to produce the same number of data files as the number of cores in the cluster. 3D data-space and 1D array of 6 elements are used for storing the data in HDF5 files. The data is chunked and shuffled to enable the zlib compression. It turned out that the HDF5 shows its high efficiency when more than 150 files, whose size is smaller than 6KB, are 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 of re-coding for the 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 such as 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) had been 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 the code porting, just like the time when we used the supercomputer in Manchester. These 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 made use of in Manchester’s supercomputer and NGS. In Manchester’s supercomputer, the different version of the HDF was installed and there was no compatibility to the one we were using. In NGS, HDF5 was not installed. Therefore the previous program, which ask each core to produce their own data file either in ASCII or in binary, was used for the numerical experiments.

However, these two facilities had been said to start charging for the CPU time and disk space. Unless the steady stream of the funding is obtained for these facilities from somewhere, these facility can not be used in the physically and mentally stable manner. Therefore before the start of charging, there was a need to move onto another high performance facilities. We decided to move to Enabling Grids for E-sciencE (EGEE). The code porting took more than one year. The reason of the difficult code porting is understood by the explanation of the computational architecture of EGEE presented in Section 3.

3. EGEE computational facility

3.1. Introduction of EGEE

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

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

In the distributed systems the security is more important than the 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 be used to become a member of one or several Virtual Organisations. Therefore a grid system is safe from the illegal penetration. On the other hand, users have to hide their digital certificates to prevent the steal of the user’s identity. To prevent the possible damage of the grid system caused by stealing certificates, the digital certificate is valid only for one year. The second safety for certificates is the necessity of proxy generalisation. In other words the users can not use their certificate directly to have an access to the grid, but they have to create a temporary certificate called proxy from it, and submit their applications using the 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 MyProxy Server, from which proxies can be generated.

3.2. Architecture of EGEE (Christodoulopoulos et al., 2008)

Fig. 1 shows the connection of major elements of EGEE which are going to be explained in Section 3.2.1 ~ Section 3.2.6.

![Figure 1: The connection of each element in 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.](image)

3.2.1. 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 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, downloading output files. Depending on the different VO, UI can be official. When a user gets a membership to a VO community, an account of the user will be generated on the UI.

As all users have their own account in the UI, they will work in their own user-space. The limited space can be a bottleneck for the 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.

3.2.2. Computing Element and Worker Node

From the viewpoint of the 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 generally First-In-First-Out queues 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 massive 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 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, the job can not generate real large output files.

3.2.3. Workload Management System

In the gLite middle-ware every job-submission is sent to a machine called Workload Management System (WMS), which can be used as a broker service. Generally a user does not know which resource executes his job prior to the job submission. A user can ask WMS to make an automated decision, which allocate the job to the best resource based on the job’s requirements. To save the WMS server from the overload, arriving jobs will be placed in a queue, and always the first job will be processed and sent to a selected resource.

3.2.4. Storage Element

One of the advantages of grid systems is the large data-storage capability. Machines, where storage component of 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.

3.2.5. Logical File Catalogue

We usually prefer sorting our remote files, that can be stored in several storage elements, in a logical structure, roughly in the same way as a normal file-system like a directory structure. Logical File Catalogue (LFC) service provides this function. LCG can be used to show or manipulate remote files in a logical order.

3.2.6. Berkeley Database Information Index

Berkeley Database Information Index (BDII) is developed to store and provide information about each available resources in distributed systems. 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.

3.3. Job submission procedure

The job submission requires users to take the following procedure:

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

After this user’s procedure, the following procedure is taken to run a job:

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

4.1. Disk restriction to run a job in WNs

Each WN has a very limited amount of disk space. As described in Section 2.2 each core reads a file whose size can be more than 1GB for the entire digital human phantom. This size of the file 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 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 $z = \text{constant}$ slices which the core needs. For example, if there are 10 cores, then each core reads 175 files.

4.2. Solution using SE

The large output data can not be left at the Worker Nodes in the CE. The obvious and immediate solution in EGEE was making use of the SE. We had implemented several shell scripts to move the output data file from each Worker Node in the CE to the SE on-the-fly. At each data production, every WN transfers its output to the SE and delete the output file in WNs as soon as the file copy is completed. With this approach, there is a high possibility that many WNs try to access the same harddrive in the SE. After the completion of the computation, all output files were transferred to the local cluster in Manchester for the 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 not copied into the SE. Intensive investigation revealed that there were some bugs in $\text{lcg-cp}$ command itself. This means that we should not rely on the unix command $\text{lcg-cp}$ for the file transfer and that we should have an alternative storage space for the output data transfer from the WNs. Our local cluster in Manchester had disk space enough to accommodate the massive data produced in EGEE. Since the unix command lines which EGEE provides does not include the communication between the WNs in CE and the computers outside the EGEE, we had to make our own mechanism to carry out this data transfer.

4.3. Solution without SE

After the long trial and error, the final solution to our main problem of the large data I/O turned out to be the direct communication between the WNs and our local cluster. Section 4.3 describes the final solution in detail by showing the shell scripts which are adaptable to any similar situation independently of the main programming language such as C or Fortran.

We first created the RSA private and public key on the UI machine by typing "$\text{ssh-keygen -t rsa}$” 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 ($\text{id_rsa.pub}$) in the UI machine is copied into $\text{HOME/.ssh/authorized_keys}$ in the local machine in Manchester. The private key should be sent to the CE together with the job. Hence the WNs can access our local cluster in Manchester using the key. To make this method work, the file $\text{known_hosts}$ at the UI is first produced by logging in one of the machines in our local cluster in Manchester from the UI. This file $\text{known_hosts}$ has to be placed in the directory of $\text{HOME/.ssh/}$ at each WN. In order to achieve this, immediately after the MPI initialisation in the code, our Fortran code calls the following shell script:

```bash
#!/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 $\text{scp}$.

Every time when 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 production 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
end do

Here, the variable $\text{rank}$ 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 a special care; if this value is too small with the heavy network or
long-distance file transfer, some files may not be transferred.

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
these 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 $\text{rm }$ $\text{\$1}$ because the input files stored in our local cluster
in Manchester should not be deleted. After the successful data loading 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 = "{\text{LFC\_HOST=\text{lfc-biomed.in2p3.fr}},\text{LCG\_CATALOG\_TYPE=\text{lfc}}}";
RetryCount = 10;
Requirements =
  Member("MPI\_START", other.GlueHostApplicationSoftwareRunTimeEnvironment) &
  Member("OPENMPI", other.GlueHostApplicationSoftwareRunTimeEnvironment) &
  other.GlueCEInfoHostname="grid10.lal.in2p3.fr"
```

Here, $\text{all.tar.gz}$ includes the Fortran code, Makefile for the Fortran code, the two shell scripts mentioned
above, the files called $\text{id\_rsa}$, known$\_hosts$, and a shell script to compile a Fortran code using Makefile.
When this JDL file is submitted via glite-wms-job-submit -a -c glite_wms.conf job.jdl, the
file $\text{mpi-start-wrapper-f90.sh}$ and the file $\text{all.tar.gz}$ are placed at each WN. Then the shell script
$\text{mpi-start-wrapper-f90.sh}$ at each WN

1. first un-tars the file $\text{all.tar.gz}$ and then
2. runs a shell script to compile the source code and finally
3. runs the MPI job using 30 cores.

Arguments suggests the arguments for the shell script $\text{mpi-start-wrapper-f90.sh}$ and the output files
$\text{mpi-test.out}$ and $\text{mpi-test.err}$ are the files of the standard output and the error messages of the job and
they can be downloaded at the UI after the job completion using the Unix command $\text{glite-wms-job-get-output}$
provided by gLite.

In this example JDL file, the CE is specified 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 had 0.5 GB of memory.
As mentioned in Section 2, 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 grid10.1atl.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 grid10.1atl.in2p3.fr, leading to the lower throughput than the site at grid10.1atl.in2p3.fr and the computational efficiency is less than the site at grid10.1atl.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 %) 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 SE and LFC.

Although this approach is produced to run our program in EGEE, this is entirely applicable to any other cases where each core produces significant amount of data at a separate disk space and the disk space is limited at each core.

5. Future Research direction

In order to perform the 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.

5.1. Improvement on the computational algorithm

The explicit scheme is simple in algorithm and highly parallelisable, ideal in the grid computing environment. However, since the temporal discretisation can not be set independently of the spatial discretisation, one has to run the many FDTD iterations for the fine spatial sampling. This inefficiency of the FDTD computation can be overcome by the algorithm modification. One of the ways to do so is the development of the implicit methods by removing the CFL stability condition. These implicit schemes can set the arbitrary $\Delta t$ independently of $\Delta s$ for the stable computation. The Crank-Nicolson implicit method (Crank and Nicolson, 1947) was proposed for this purpose. However, the Crank-Nicolson implicit method includes a huge sparse matrix to solve. Therefore at the time when it was invented the ordinary computers was 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 tridiagonal 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 are currently studied (Rouf et al., 2009b).

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

5.2. Improvement on the data input and output

As mentioned, the digital human phantom was sliced in the human height direction. 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 original data size. Thus, one way to improve the data input is the reduction of the
data algorithmically. The information which shows the difference from the neighbouring slice can be sent 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 Section 2.3. 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.

6. Conclusion

Many researchers making use of the high performance computing facilities face the situation where their computational architecture is changed either suddenly or gradually time to time. This can be intentional or out of their control. In either case, this is the part of the nature of their research area because the computational cost, the computational performance, the funding for their research activity change all the time. In order to be able to survive under any circumstances the researchers, who perform the coding for their algorithm implementation, have to pay attention to the possibility of re-use of the code, easiness to read the code, computational efficiency, and code portability.

This chapter first introduced the nature of our computation handled in the HPC facility by presenting the core equations. It is highly parallelisable, suitable for grid computing apart from the point that the computational algorithm requires the high data I/O and the large computational memory per core.

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

This problem and the solution of 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 are proposed from the viewpoint of the computational algorithm and the data I/O.

Acknowledgement

The authors wish to thank Dr. Matthieu Bonilla at EADS Nucleotides, 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


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

```bash
#!/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 $PORT -i 'pwd'/id_rsa $LOCAL_FOLDER/$FILENAME $REMOTE_FOLDER:$REMOTE_PATH$CUT_FILENAME
    COUNT=$(($COUNT + 1))
do
```

13
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

#!/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
Key terms and definitions

Enabling Grids for E-sciencE: a grid-project that provides large computational resources connected via Internet founded by European Union. Any applications must be parallelized in order to benefit from EGEE.

Job management: scheduling and managing computational requests to a particular computing element. In case when there are more than one users in a grid computing facility, their computational requests (jobs) are submitted at one location. These jobs are ranked in priority and placed at the individual computational nodes.

Finite Difference Time Domain 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 used at each FDTD iteration. The outcome of the FDTD simulation is the signal signature(waveform) in time domain.

Grid computing: computation using computational grid facility that consists of many computational or datastore resources. The code for computation has to take into account the fact that there are many computational cores in many nodes.

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

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 in one machine usable in the other machine is called code porting.

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

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.

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

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

Additional Reading


