ARTICLE IN PRESS

J. Parallel Distrib. Comput. ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

On optimization of finite-difference time-domain (FDTD) computation on heterogeneous and GPU clusters

Ramtin Shams *, Parastoo Sadeghi

College of Engineering and Computer Science (CECS), The Australian National University, Canberra ACT 0200, Australia

ARTICLE INFO

Article history:
Received 14 May 2010
Received in revised form
28 September 2010
Accepted 14 October 2010
Available online xxxx

Keywords:
Finite-difference time-domain (FDTD)
Heterogeneous computing
Parallel processing
Graphics Processing Unit (GPU)
Optimization

ABSTRACT

A model for the computational cost of the finite-difference time-domain (FDTD) method irrespective of implementation details or the application domain is given. The model is used to formalize the problem of optimal distribution of computational load to an arbitrary set of resources across a heterogeneous cluster. We show that the problem can be formulated as a minimax optimization problem and derive analytic lower bounds for the computational cost. The work provides insight into optimal design of FDTD parallel software. Our formulation of the load distribution problem takes simultaneously into account the computational and communication costs. We demonstrate that significant performance gains, as much as 75%, can be achieved by proper load distribution.

1. Introduction

The finite-difference time-domain (FDTD) method, since its introduction by Yee [31], has been widely used to obtain numerical solutions of Maxwell's equations for a broad range of problems. The applications of FDTD in electrodynamics include antenna and radar design, electronic and photonic circuit design, microwave tomography, cellular and wireless network simulation, mobile phone safety studies, and many more [29]. The method is not limited to electrodynamics and can be used to solve other spatiotemporal partial differential equations such as those occurring in acoustics (e.g. see [25]). The explicit nature of FDTD formulation, its simplicity, accuracy and robustness, together with a well established theoretical framework have contributed to a seemingly unending popularity of the method.

Realistic FDTD simulations involve fine discretization of the spatial domain as well as the temporal domain. It is not uncommon to use spatial grids of 10^9 and spatiotemporal grids of 10^{13} or more cells. As a result, FDTD simulations require significant computational resources both in terms of memory and execution. It is inevitable that many realistic simulations exceed the memory limitations of a single computer and have to be divided across a cluster of computers. The additional incentive to distribute FDTD computation is to minimize execution time where resources are available. This, however, introduces the additional cost of intercommunication between compute nodes in order to execute FDTD simulations on the entire computational domain of the problem.

Parallelization and acceleration of FDTD has been an active area in recent years. In particular, there have been several examples of FDTD acceleration on field programmable gate array (FPGA) hardware and graphics processing units (GPUs). A list of recent contributions in this area is given later in Section 2.2.

Traditionally, high performance computer clusters have been built from identical compute resources. When the communication links connecting the compute resources also have the same latency and bandwidth and all nodes run the same software, the cluster is said to be *homogeneous* [20]. With the introduction of accelerator technologies and their rise in popularity for general purpose computing, this is no longer the case. The current generation of accelerators require a computer host and hence by definition create hybrid nodes when clustered. A notable example of one such cluster is IBM's Roadrunner supercomputer which comprises 13,824 Opteron cores and 116,640 Cell processor cores. According to technical staff at Los Alamos the applications are typically designed for execution on Cell processors and except for trivial house keeping and data transfer to and from Cell processors, the Opteron cores remain idle most of the time.[1] This represents a significant computing capacity that remains under-utilized.

A heterogeneous cluster is one that is not homogeneous and may comprise of computational resources with differing

* Corresponding author.
E-mail address: ramtin.shams@anu.edu.au (R. Shams).

[1] According to discussions at the Path to Petascale Workshop, April 2009, National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign.

0743-7315/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2010.10.011

technology, execution capability, memory size, and speed. A heterogeneous cluster may contain accelerators (e.g. GPUs, Cell processors, FPGA boards), desktop computers, and server computers. While heterogeneous resources are commonly found on any modern network, they are rarely used as heterogeneous clusters particularly across technology boundaries (e.g. x86 and PowerPC). The motivation for heterogeneous computing is to maximize use of available resources, whether across an organization's network or on purpose-built heterogeneous clusters, towards solving larger scientific problems. It comes as no surprise that the subject has attracted a lot of attention from the research community (see [8] for a recent survey).

The two main impediments in effective use of heterogeneous clusters are (a) the additional effort involved in development of heterogeneous applications and (b) the need to design an optimal load distribution scheme across heterogeneous resources. In this work, we assume the reader is sufficiently motivated to tackle the former problem and focus on the latter in the context of FDTD parallelization where we look at the problem of optimal distribution of FDTD computation across a heterogeneous cluster.

1.1. Contributions

This work provides insight into the optimal design of a heterogeneous FDTD application by

(1) modeling the cost of FDTD computation on a heterogeneous cluster (Section 3.1);
(2) formalizing the load distribution problem as a minimax optimization problem (Section 3.2);
(3) deriving analytic lower bounds for the execution cost of FDTD on a heterogeneous cluster (Section 3.4.1); and
(4) solving the problem by taking simultaneously into account the cost of computation and data communication (Section 3.5).[2]

We would like to clarify from the outset that this work is not a software development effort or a specific parallel implementation of FDTD. It is an implementation-agnostic analysis of the inherent computational limitations of FDTD on a most general class of computational clusters (i.e. heterogeneous clusters). It also proposes a near optimal load distribution algorithm for FDTD implementation for heterogeneous clusters in general and for homogeneous clusters as a subset. The only assumption that we make regarding computational resources is that their performance, from the FDTD application's perspective, does not change over time. This means that the problem is being investigated under the so-called 'constant performance model' for a heterogeneous cluster [20] as opposed to a 'non-constant performance model' where the resources may be shared by multiple programs and constant performance cannot be guaranteed over time.

We note that a myriad of parallel implementations of FDTD on homogeneous clusters, in the form of commercial packages, open source libraries, and scholarly research exist, that can benefit from this work. These parallelization efforts cover a wide range of hardware from supercomputers and general purpose programmable CPUs to GPU, application-specific integrated circuit (ASIC) and FPGA clusters. We would particularly like to emphasize the lack of FDTD software that can efficiently run across such technology boundaries in a true heterogeneous fashion.

2. Concepts

2.1. An overview of the FDTD method

In this section, we provide a brief overview of the FDTD method based on 3D Maxwell electromagnetic equations. The extent we delve into the subject is to enable the reader to appreciate the computational model of FDTD presented in Section 3.1. A careful treatment of the subject is outside the scope of this paper and the reader is referred to [29] for detailed discussions.

Maxwell's curl equations for linear, isotropic, lossy and non-dispersive media are given by

$$-\mu\frac{\partial \mathbf{H}}{\partial t} = \nabla\times\mathbf{E} + \sigma_m\mathbf{H} + \mathbf{M}, \tag{1}$$

$$-\epsilon\frac{\partial \mathbf{E}}{\partial t} = -\nabla\times\mathbf{H} + \sigma_e\mathbf{E} + \mathbf{J}, \tag{2}$$

where $\mathbf{H}$ is the magnetic field, $\mathbf{E}$ is the electric field, μ is the magnetic permeability, ϵ is the electric permittivity, $\mathbf{M}$ is the equivalent magnetic current density, $\mathbf{J}$ is the electric current density, σ_m is the equivalent magnetic conductivity, and σ_e is the electric conductivity. For the purpose of FDTD simulations $\mathbf{H}$ and $\mathbf{E}$ fields are unknown and all other quantities are given at each point in space. The equations embody 6 partial differential equations, for example the derivative of the x-component of the electric field with respect to time is given by

$$-\epsilon\frac{\partial E_x}{\partial t} = \frac{\partial H_y}{\partial z} - \frac{\partial H_z}{\partial y} + \sigma_e E_x + J_x. \tag{3}$$

The equations are discretized in space and time to derive an explicit solution for the next time step. Due to dependence of $\mathbf{E}$ and $\mathbf{H}$ components, it is best to interleave values of $\mathbf{E}$ and $\mathbf{H}$ in time with $\Delta t/2$ time difference between them. For example, $\mathbf{H}$ can be computed at $n\Delta t$ and $\mathbf{E}$ computed with a half interval shift at $n\Delta t + \Delta t/2$. Similarly, the $\mathbf{E}$ and $\mathbf{H}$ components are *staggered* in space according to an arrangement known as the Yee cell [31,29]. For convenience we show a function $u(i\Delta x, j\Delta y, k\Delta z, n\Delta t)$ with $u|^n_{i,j,k}$. Using this notation, (3) can be discretized on a cuboid grid of $(\Delta x, \Delta y, \Delta z)$ using second order accurate central differences as

$$\begin{aligned}
&-\epsilon|_{i,j,k}\frac{E_x|^{n+\frac{1}{2}}_{i,j+\frac{1}{2},k+\frac{1}{2}} - E_x|^{n-\frac{1}{2}}_{i,j+\frac{1}{2},k+\frac{1}{2}}}{\Delta t}\\
&\quad\approx \frac{H_y|^n_{i,j+\frac{1}{2},k+1} - H_y|^n_{i,j+\frac{1}{2},k}}{\Delta z} - \frac{H_z|^n_{i,j+1,k+\frac{1}{2}} - H_z|^n_{i,j,k+\frac{1}{2}}}{\Delta y}\\
&\qquad + \sigma_e|_{i,j,k}E_x|^n_{i,j+\frac{1}{2},k+\frac{1}{2}} + J_x|^n_{i,j,k},
\end{aligned} \tag{4}$$

since E_x is not computed at $n\Delta t$, it can be approximated by

$$E_x|^n_{i,j+\frac{1}{2},k+\frac{1}{2}} \approx \frac{E_x|^{n+\frac{1}{2}}_{i,j+\frac{1}{2},k+\frac{1}{2}} + E_x|^{n-\frac{1}{2}}_{i,j+\frac{1}{2},k+\frac{1}{2}}}{2}, \tag{5}$$

and we have

$$\begin{aligned}
E_x|^{n+\frac{1}{2}}_{i,j+\frac{1}{2},k+\frac{1}{2}} \approx{}& \left(\frac{\epsilon|_{i,j,k}/\Delta t - \sigma_e|_{i,j,k}/2}{\epsilon|_{i,j,k}/\Delta t + \sigma_e|_{i,j,k}/2}\right)E_x|^{n-\frac{1}{2}}_{i,j+\frac{1}{2},k+\frac{1}{2}}\\
&- \left(\frac{1}{\epsilon|_{i,j,k}/\Delta t + \sigma_e|_{i,j,k}/2}\right)\\
&\times\left[\frac{H_y|^n_{i,j+\frac{1}{2},k+1} - H_y|^n_{i,j+\frac{1}{2},k}}{\Delta z}\right.\\
&\left. - \frac{H_z|^n_{i,j+1,k+\frac{1}{2}} - H_z|^n_{i,j,k+\frac{1}{2}}}{\Delta y} + J_x|^n_{i,j,k}\right].
\end{aligned} \tag{6}$$

Based on (6), E_x at each grid point and at time $n\Delta t + \Delta t/2$ can be computed from values of $\mathbf{E}$ and $\mathbf{H}$ at previous times. Similar equations can be derived for other components of $\mathbf{E}$ and $\mathbf{H}$ fields. These equations allow the field values to be computed explicitly at an arbitrary time index by marching through all previous time

[2] Existing work in the area, such as the canonical matrix multiplication example studied in [20], determine the partition sizes based on the computational cost first and only then try to minimize the communication cost.

Please cite this article in press as: R. Shams, P. Sadeghi, On optimization of finite-difference time-domain (FDTD) computation on heterogeneous and GPU clusters, J. Parallel Distrib. Comput. (2010), doi:10.1016/j.jpdc.2010.10.011

indices. We note that the rather unusual notation involving half indices such as $j + \frac{1}{2}$ are due to the arrangement of the field components in the Yee cell. We refer the reader to [29] if a detailed explanation of the notation is desired.

The discrete grid discussed above, also known as the computational domain, represents a finite, discrete and bounded model of some real domain of interest. If the simulation is sufficiently long the wavefronts reach the boundaries of the computational domain. With no information about the medium outside the boundary, the wavefronts cannot naturally progress and are reflected back into the computational domain. In effect, the boundary of the computational domain acts as a reflective barrier, which in most circumstances is undesirable and a source of error and clutter. A significant body of work has been dedicated to designing absorbing boundary conditions (ABCs) to address this problem [29]. Commonly used ABC's include Mur's ABC [23], the perfectly matched layer (PML) [3] and its variants such as the uniaxial PML (UPML) [27,10] and the convolutional PML (CPML) [26].

Without getting into the details of different types of ABCs and their characteristics, we note that implementation of the boundary conditions involves additional computation at one or several layers of cells on the border of the computational domain. This makes the computation of boundary cells more expensive than regular cells.

As previously noted, use of FDTD is not limited to the solution of Maxwell's equations or to two or three dimensions. In the following sections, a general d-dimensional ($d > 1$) Cartesian computational domain is assumed.

2.2. Parallelization of FDTD

From the discussion of the previous section it should be obvious that the FDTD method naturally lends itself to parallelization. At each time-step updating a cell, for example with (6), requires values of the field components of a given cell and its neighboring cells in the previous time step. In a more general setting where higher order discretization or non-linear wave equations are involved one may require field values from several previous time steps. Regardless of the complexity of the wave equations, FDTD ensures that each cell update is independent of its neighbors in the current time-step. This allows for FDTD computations to be well suited to parallelization.

Several parallel implementations of FDTD have appeared in the literature. The efforts cover a wide range of parallel architecture and hardware including symmetric multiprocess (SMP) clusters, FPGA hardware, GPUs, and distributed shared memory (DSM) systems. A non-exhaustive summary (post 2000) is given in Table 1 which serves to demonstrate the degree of interest in this problem.

SMP clusters typically use a combination of OpenMP [24] for parallelization on a single node and the message passing interface (MPI) [11] for parallelization across multiple nodes. There has been more interest in using GPUs for acceleration of FDTD in recent years. FDTD is memory intensive and standard caching mechanisms on the CPUs are not well suited to FDTD memory access patterns. The latest generation of GPUs, on the other hand, are specifically suited to this task as they give programmers control over loading data into a small but efficient shared memory that can significantly boost memory access. In addition, bandwidths of device memory on a GPU may exceed 100 GB/s which is at least an order of magnitude higher than standard host memory.

3. Method

3.1. Modeling computational cost of FDTD on a heterogeneous cluster

Consider a hyper-rectangular FDTD computational domain Ω such as the 2-rectangle shown in Fig. 1 partitioned into a number

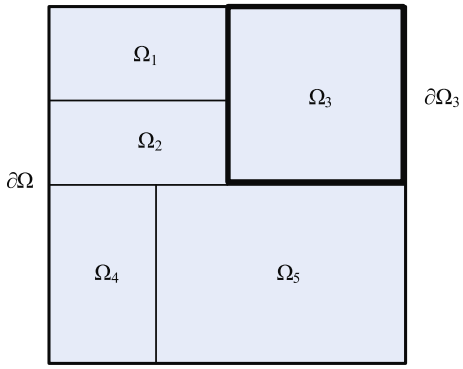


Fig. 1. A 2D computational domain divided into 5 partitions. The boundary of the third partition is shown with a thicker border for emphasis.

of non-overlapping hyper-rectangular sub-domains each denoted by Ω_i and a boundary given by $\partial\Omega_i$. The sub-domains are non-overlapping (except on the boundary), i.e.

$$\bigcup_i \Omega_i = \Omega, \qquad \Omega_i \cap \Omega_j = \emptyset \quad \text{for } i \neq j. \tag{7}$$

Each partition is mapped to a computational resource such as a multi-core CPU, a GPU or a Cell processor and the number of partitions equals the number of available computational resources. We denote the set of all such partitions with n elements by Γ_n.

For each partition the cost of a single time-marching step is broken down into three components

(1) cost of updating regular cells which is proportional to the size of partition $|\Omega_i|$,
(2) additional cost of updating cells in the inner boundary of the partition with neighboring cells due to the need to load information from neighboring partitions through an interconnection between associated resources which depends on the size of the common boundary between a given partition and its neighboring partitions $|\partial\Omega_i \cap \partial\Omega_j|$, and
(3) additional cost of updating cells in the outer boundary of the partition with the boundary of the domain (typically on an absorbing layer) which is proportional to the size of the common boundary between the partition and the computational domain $|\partial\Omega_i \cap \partial\Omega|$.

Ignoring the additional cost of handling source cells (that typically comprise only a small number of cells), the cost of the time-marching algorithm (in terms of execution time) for the ith partition can be written as

$$t_i = \alpha_i|\Omega_i| + \sum_{j \neq i} \beta_{ij}|\partial\Omega_i \cap \partial\Omega_j| + \gamma_i|\partial\Omega_i \cap \partial\Omega|, \tag{8}$$

where $|\Omega_i|$ is the size of the partition, $|\partial\Omega_i|$ is the size of the partition boundary, and α_i, β_{ij} and γ_i are constants of proportionality that relate size of partitions and boundaries to the respective cost of execution. Note that these constants are determined by the computational capability of each resource i and the throughput of the interconnect between resources i, j and are independent of the domain partitioning.

The first term in (8) captures the cost of executing one time-marching step on the interior of the partition where all the information to compute in the next time instance for a given cell resides within the computational resource i, the second term represents the additional cost associated with the transfer of data from resource j to resource i to enable computation of the next time step for boundary cells, and the third term represents the additional cost of computing absorbing boundary conditions on the boundary of the domain.

Please cite this article in press as: R. Shams, P. Sadeghi, On optimization of finite-difference time-domain (FDTD) computation on heterogeneous and GPU clusters, J. Parallel Distrib. Comput. (2010), doi:10.1016/j.jpdc.2010.10.011

Table 1
A sample list of parallel FDTD contributions in the literature.

Application	ABC[a]	Platform	Perf[b]	Group	Year
3D FDTD	PML	Cray T3E (16 CPUs @ 300 MHz), MPI	8.8	Guiffaut [12]	2001
2D FDTD	Mur	Custom Hardware @ 100 MHz	6.3	Kawaguchi [16]	2002
2D FDTD	Mur	Firebird FPGA board (@ 70 MHz)	13.8	Chen [5]	2004
3D FDTD	PML	Xilinx Virtex-II 8000 FPGA	30	Durbano [9]	2004
2D FDTD	Mur[c]	GeForce FX 5800 Ultra, OpenGL	82	Krakiwsky [18]	2004
3D FDTD	PML	IBM RS/6000 SP (8 nodes/ 128 CPUs @ 375 MHz),OpenMP/MPI	36.1	Hughes [13]	2005
3D FDTD	None	GTX 8800 (16 MP/ 128 cores), OpenGL/Cg	–	Adams [1]	2007
3D FDTD	Mur	GTX 280 (30 MP/ 240 cores), CUDA[d]	420	Stefanski [28]	2009
3D ADI-FDTD	Mur	GTX 280 (30 MP/ 240 cores), CUDA	40	Stefanski [28]	2009
3D FDTD	PEC	GTX 280 (30 MP/ 240 cores), CUDA	–	Liuge [21]	2009
2D FDTD	None	GTX 280 (30 MP/ 240 cores), CUDA	1795	Takada [30]	2009

[a] Absorbing Boundary Condition.
[b] Performance in MCells/s.
[c] A combination of 1st order Mur and periodic boundary conditions is used.
[d] Compute Unified Device Architecture [7].

3.2. Problem formulation

All resources need to complete their computation of the current time step before they can proceed to the next step. In other words, a barrier synchronization primitive is required at the end of each time step iteration. This requires faster resources to wait for slower resources to complete their task and hence the cost of a single iteration is given by

$$t_m = \max_i \left\{ \alpha_i |\Omega_i| + \sum_{j \neq i} \beta_{ij} |\partial\Omega_i \cap \partial\Omega_j| + \gamma_i |\partial\Omega_i \cap \partial\Omega| \right\}. \quad (9)$$

We are now ready to formalize the problem: we seek the optimal distribution of load to a heterogeneous cluster that minimizes (9). This is a minimax problem over disjoint n-element partitions of the computational domain Ω

$$t_{\text{opt}} = \min_{\Gamma_n} \left\{ \max_i \left\{ \alpha_i |\Omega_i| + \sum_{j \neq i} \beta_{ij} |\partial\Omega_i \cap \partial\Omega_j| + \gamma_i |\partial\Omega_i \cap \partial\Omega| \right\} \right\}. \quad (10)$$

Finding the optimal partitions based on (10) is far from trivial. This is because, the geometry and position of the partitions need to be known in order to compute the overlap between neighboring partitions and between partitions and the exterior of the domain. However, as shown in the following sections, it is possible to find analytic lower bounds for t_{opt} (under certain conditions) that are independent of the partitioning scheme and hence provide insight into achievable performance levels without the need to directly solve for (10). The problem needs to be relaxed in order to achieve these goals.

3.3. Relaxing the problem

Let $\beta_i = \min_j\{\beta_{ij}\}$ for $j \neq i$; by replacing β_{ij} with β_i in (10) and using

$$\sum_{j \neq i} |\partial\Omega_i \cap \partial\Omega_j| = |\partial\Omega_i| - |\partial\Omega_i \cap \partial\Omega|, \quad (11)$$

we have

$$t_{\text{opt}} \geq \min_{\Gamma_n} \left\{ \max_i \{\alpha_i |\Omega_i| + \beta_i |\partial\Omega_i| + (\gamma_i - \beta_i) |\partial\Omega_i \cap \partial\Omega|\} \right\}. \quad (12)$$

Assuming the third term (involving $|\partial\Omega_i \cap \partial\Omega|$) in the above equation is non-negative ($\gamma_i \geq \beta_i$ for all i), we can write

$$t_{\text{opt}} \geq \min_{\Gamma_n} \left\{ \max_i \{\alpha_i |\Omega_i| + \beta_i |\partial\Omega_i|\} \right\}. \quad (13)$$

This is not an unreasonable assumption, given that computing boundary conditions is typically a more expensive task. Finding a solution for the right hand side of (13) provides a lower-bound for t_{opt}. This is also equivalent to solving a special case of (10) where the normalized cost of data transfer between a resource and all other resources is the same (i.e. $\beta_{ij} = \beta_i$) and the normalized cost of computing boundary conditions is assumed to be the same as the cost of transferring boundary information between the resources (i.e. $\gamma_i = \beta_i$). We also relax the partitioning condition (7) such that we only require the sum of partition sizes to be equal to the size of the domain. This means that we ignore the need to properly pack the partitions in the given domain, at least for now. We denote this 'reduced' optimization problem by:

$$t_{\text{opt}} = \min_{\Gamma_n} \left\{ \max_i \{\alpha_i |\Omega_i| + \beta_i |\partial\Omega_i|\} \right\}, \quad \sum_i |\Omega_i| = |\Omega|. \quad (14)$$

Lemma. *If $\mathbf{\Omega} = \{\Omega_1, \ldots, \Omega_n\}$ is a solution of (14) so is $\tilde{\mathbf{\Omega}} = \{\tilde{\Omega}_1, \ldots, \tilde{\Omega}_n\}$ where $|\Omega_i| = |\tilde{\Omega}_i|$ and the partitions of $\tilde{\mathbf{\Omega}}$ are hyper-cubes.*

Proof. Out of all hyper-rectangles of a given size, the hyper-cube has the least boundary size, hence $|\partial\tilde{\Omega}_i| \leq |\partial\Omega_i|$ and $\alpha_i|\tilde{\Omega}_i| + \beta_i|\partial\tilde{\Omega}_i| \leq \alpha_i|\Omega_i| + \beta_i|\partial\Omega_i|$. So the cost of computation for no resource under $\tilde{\mathbf{\Omega}}$ is higher than under $\mathbf{\Omega}$ and $\tilde{\mathbf{\Omega}}$ must be a solution as well. □

For a d-dimensional hyper-cube we have $|\partial\Omega_i| = 2d|\Omega_i|^{\left(1-\frac{1}{d}\right)}$ and we can now focus our search on finding a hyper-cubic solution by solving

$$t_{\text{opt}} = \min_{\Gamma_n} \left\{ \max_i \left\{ |\Omega_i| \left(\alpha_i + 2d\beta_i |\Omega_i|^{-\frac{1}{d}} \right) \right\} \right\}, \quad \sum_i |\Omega_i| = |\Omega|. \quad (15)$$

3.4. Lower bounds for the minimax load distribution problem

In this section we derive analytic lower bounds for (15).

3.4.1. Bound 1

Let us denote the execution time of a given partition by $t_m = \max_i\{t_i\}$

$$\frac{1}{\alpha_i} t_m \geq |\Omega_i| \left(1 + 2d \frac{\beta_i}{\alpha_i} |\Omega_i|^{-\frac{1}{d}} \right) \quad (16)$$

$$\sum_i \frac{1}{\alpha_i} t_m \geq \sum_i |\Omega_i| + 2d \sum_i \frac{\beta_i}{\alpha_i} |\Omega_i|^{1-\frac{1}{d}} \quad (17)$$

Please cite this article in press as: R. Shams, P. Sadeghi, On optimization of finite-difference time-domain (FDTD) computation on heterogeneous and GPU clusters, J. Parallel Distrib. Comput. (2010), doi:10.1016/j.jpdc.2010.10.011

$$t_m \geq \left(\sum_i \frac{1}{\alpha_i}\right)^{-1} \left(|\Omega| + 2d \sum_i \frac{\beta_i}{\alpha_i} |\Omega_i|^{1-\frac{1}{d}}\right) \tag{18}$$

$$t_{\text{opt}} = \min_{\Gamma_n}\{t_m\} \geq \min_{\Gamma_n} \left\{ \left(\sum_i \frac{1}{\alpha_i}\right)^{-1} \times \left(|\Omega| + 2d \sum_i \frac{\beta_i}{\alpha_i} |\Omega_i|^{1-\frac{1}{d}}\right)\right\}. \tag{19}$$

Minimizing the right hand side of (19) gives a lower bound for t_{opt}. This requires minimizing the term involving $|\Omega_i|^{1-\frac{1}{d}}$. Let q be the index of the partition with the smallest ratio of the normalized transfer cost to the normalized computation cost (i.e. $q = \text{argmin}_i\, \beta_i/\alpha_i$)

$$\sum_i \frac{\beta_i}{\alpha_i} |\Omega_i|^{1-\frac{1}{d}} \geq \frac{\beta_q}{\alpha_q} \sum_i |\Omega_i|^{1-\frac{1}{d}}, \tag{20}$$

$$\geq \frac{\beta_q}{\alpha_q} \left(\sum_i |\Omega_i|\right)^{1-\frac{1}{d}} = \frac{\beta_q}{\alpha_q} |\Omega|^{1-\frac{1}{d}}, \tag{21}$$

where we use $x^p + y^p \geq (x+y)^p$ for $x, y \geq 0$ and $0 \leq p \leq 1$ to derive (21). Therefore,

$$t_{\text{opt}} \geq \left(\sum_i \frac{1}{\alpha_i}\right)^{-1} \left(1 + 2d \frac{\beta_q}{\alpha_q} |\Omega|^{-\frac{1}{d}}\right) |\Omega|. \tag{22}$$

The right hand side of (22) gives a lower bound for t_{opt}. A solution close to this lower bound can be found when the additional cost of computing boundary cells is significantly lower than the cost of time-marching regular cells ($\beta_i/\alpha_i \ll 1$) or ideally when $\beta_i = 0$ in which case t_{opt} from (8) is given by

$$t_{\text{opt}} = \left(\sum_i \frac{1}{\alpha_i}\right)^{-1} |\Omega|. \tag{23}$$

We argue that this is possible when the partition sizes are given by

$$|\Omega_j| = \frac{1}{\alpha_j} \left(\sum_i \frac{1}{\alpha_i}\right)^{-1} \qquad |\Omega| = \frac{t_{\text{opt}}}{\alpha_j}. \tag{24}$$

The proof is by contradiction, first consider for some j, we have $|\Omega_j| > t_{\text{opt}}/\alpha_j$, this results in $t_j > t_{\text{opt}}$, which violates the condition that t_{opt} is the maximum cost of execution of any partition for a given partitioning scheme. Conversely, consider that for some j, we have $|\Omega_j| < t_{\text{opt}}/\alpha_j$. We have already established that $|\Omega_i| \leq t_{\text{opt}}/\alpha_i$ for $i \neq j$. Summing up inequalities for all i we have

$$|\Omega_j| + \sum_{i \neq j} |\Omega_i| < \frac{t_{\text{opt}}}{\alpha_j} + \sum_{i \neq j} \frac{t_{\text{opt}}}{\alpha_i}. \tag{25}$$

Using (23) and noting that $\sum_i |\Omega_i| = |\Omega|$, both sides of the above inequality reduce to $|\Omega|$ which cannot be and the proof is complete. □

According to (24), where the computational cost associated with boundary conditions and data transfers are low, the optimal partitioning scheme is one that ensures all computational resources take the same amount of time to complete one iteration of the algorithm. This is consistent with the desire to ensure computational resources will not be idle when possible.

3.4.2. Bound 2

The bound given in the previous section is tight where $\alpha_i \gg \beta_i$ and becomes less tight as the cost of data transfers increases. We derive a tighter bound under such conditions in this section.

Assuming that $|\Omega_i| > 1$ and using (15), we can write

$$t_{\text{opt}} \geq \min_{\Gamma_n} \left\{ \max_i \left\{ |\Omega_i|^{1-\frac{1}{d}} (\alpha_i + 2d\beta_i) \right\} \right\}, \tag{26}$$

where we replaced $\alpha_i|\Omega_i|$ with the smaller term $\alpha_i|\Omega_i|^{1-\frac{1}{d}}$.

In a manner similar to the proof given in the previous section, it can be shown that the right hand side of (26) is minimized when

$$|\Omega_i|^{1-\frac{1}{d}} (\alpha_i + 2d\beta_i) = |\Omega_j|^{1-\frac{1}{d}} (\alpha_j + 2d\beta_j), \tag{27}$$

for all i and j and $|\Omega_i|$ is given by

$$|\Omega_j| = |\Omega| \left[(\alpha_j + 2d\beta_j)^{\frac{d}{d-1}} \sum_i (\alpha_i + 2d\beta_i)^{-\frac{d}{d-1}} \right]^{-1}. \tag{28}$$

And a new lower bound is given by

$$t_{\text{opt}} \geq \left[|\Omega| \left(\sum_i (\alpha_i + 2d\beta_i)^{-\frac{d}{d-1}} \right)^{-1} \right]^{1-\frac{1}{d}}. \tag{29}$$

The tightness of the bound improves as the number of dimensions increases. We also note that the bound given in (29) is loose when the cost of data transfer is not significant compared to the cost of computations but improves as data transfer becomes the bottleneck. This trend is opposite to that of the bound derived in the previous section. Therefore, it makes sense to combine the two bounds and use their maximum as the lower-bound.

3.5. Numerical optimization

The objective function given in (15) can be solved using constrained numerical optimization methods. The objective function

$$f(|\Omega_1|, \ldots, |\Omega_1|) = \max_i \left\{ |\Omega_i| \left(\alpha_i + 2d\beta_i |\Omega_i|^{-\frac{1}{d}}\right)\right\}, \quad \sum_i |\Omega_i| = |\Omega| \tag{30}$$

is nonlinear and non-convex with linear constraints or alternatively one can parameterize in partition dimension size $x_i = |\Omega_i|^{\frac{1}{d}}$ where the cost function will be convex in x_i but subject to nonlinear constraints. Either way standard convex optimization methods cannot be used. We use a nonlinear constrained optimization method based on *sequential quadratic programming* (SQP) and the Quasi-Newton algorithm to solve the problem [4].[3]

As usual initialization close to the global minimum is an important element for improving the success of the optimization algorithm. We initialize the algorithm with an initial guess in accordance with (24) or (26). We use the equation that corresponds to the tighter of the two bounds. In practice, for a range of experiments, the optimization converges quickly (typically in less than 100 iterations) and given the simplicity of the cost function the computations are most efficient. We defer further discussion on the experiments to Section 4.

3.6. Load distribution algorithm

Up to now, we have discussed methods to determine the size and dimensions of partitions subject to the relaxed constraint that the sum of partitions equals the size of the computational domain. Once the solution of the relaxed problem is found, the partitions need to be fit into the computational domain under the constraint that they must cover its entire volume. There are several reasons that an exact fit may not be possible. In practice, dimensions

[3] An implementation of an active-set method based on SQP and the Quasi-Newton algorithm is given by MATLAB Optimization Toolbox function '*fmincon*'.

Please cite this article in press as: R. Shams, P. Sadeghi, On optimization of finite-difference time-domain (FDTD) computation on heterogeneous and GPU clusters, J. Parallel Distrib. Comput. (2010), doi:10.1016/j.jpdc.2010.10.011

of the computational domain and its partitions belong to the set of positive integers. This inherently means that the optimal[4] partitioning size and dimensions cannot be exactly met except for carefully engineered dimensions.

We also note that even where optimal partition sizes and dimensions are feasible, partitioning of the computational domain to an exact set of partitions may not be possible (e.g. try partitioning a square into two squares). Intuitively, as the number of partitions increase these limitations become less of an issue and a close to optimal partitioning can be achieved.

The problem can be slightly reformulated to ensure that real-valued solutions exist. The condition for the shape of partitions is relaxed by only requiring the sum of boundaries of the partitions to be minimal. This relaxed problem is known as geometric partitioning and has been well-studied in the context of partitioning a 2-rectangle to a set of rectangles with given areas [6,14,2,19]. The problem is NP-complete [2]. However, polynomial solutions for heuristics such as the recursive bisection algorithm [6], the *XY* and tile partitioning [14], and more recently the column-based partitioning [2] and the grid-based partitioning [19] have been proposed. The column-based and grid-based algorithms solve a more constrained version of the problem but have a theoretical advantage, in that they find the optimal solution for their constrained problems. The column-based algorithm, the less constrained of the two, makes the additional assumption that the rectangular partitions are organized into a number of column partitions. In practice, except for pathologically constructed examples, all these methods provide satisfactory results.

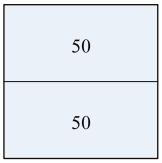
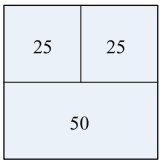
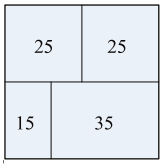
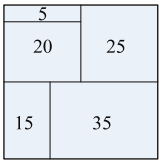
Here, we use a variation of the recursive bisection algorithm. The intuition is that the algorithm should be faithful to the original partition sizes and shapes (which are hyper-cubes) to the extent possible while maintaining the size of bisections balanced in each iteration. We will refer to the method as the *balanced* partitioning algorithm. The motivations behind using the balanced partitioning algorithm are (a) ease of use with FDTD domains: the algorithm can be used for an arbitrary dimensional hyper-rectangle; (b) efficient execution: the complexity of the algorithm is typically $\mathcal{O}(n(\log n)^2)$ where n is the number of partitions. In comparison, the column-based partitioning has a complexity of $\mathcal{O}(n^3)$ and the more constrained grid-based partitioning has a complexity of $\mathcal{O}\left(n^{\frac{3}{2}}\right)$ [20].

Balanced Partitioning Algorithm: Let Ω be a computational domain to be distributed to n computational resources

(1) Measure normalized computational cost α_i and transfer cost β_i of each resource.
(2) Compute the set of optimal partition sizes $S = \{|\Omega_1|, |\Omega_2|, \ldots, |\Omega_n|\}$ using a properly initialized nonlinear optimization algorithm.
(3) Partition S into two sets S_1 and S_2 such that the difference between the sum of elements of S_1 and S_2 is minimal.
(4) Partition Ω along the largest dimension into two partitions whose size is given by the sum of elements of S_1 and S_2. Adjust any integer round-off errors introduced as a result.
(5) Replace S with S_1 and S_2 and Ω with newly created partitions and continue the steps 3–5 until S_1 and S_2 cannot be further partitioned.

The intuition to partition the domain along its largest dimension is to maintain the lowest possible aspect ratio (the ratio of the largest dimension to the smallest dimension). This is an attempt to make the partitions closer to hyper-cubes as the partitioning algorithm progresses.

Table 2
Partitioning a sample computational domain.

Partitions	Balanced partitions	Adjusted sums	Domain
{4, 17, 22, 26, 31}	{4, 22, 26}, {17, 31}	50, 50	
{4, 22, 26}	{4, 22}, {26}	25, 25	
{17, 31}	{17}, {31}	15, 35	
{4, 22}	{4}, {22}	5, 20	

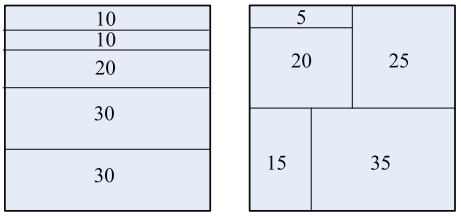
The third step of the algorithm is known as the *number partitioning problem*. The problem is whether a set of numbers can be partitioned into two halves of equal sum or more generally finding two partitions that minimize the maximum partition sum. The number partitioning problem is NP-complete [22], however, there are simple heuristic algorithms that can, in many instances, solve the problem optimally or near optimally in less than $\mathcal{O}(n^2)$ time. The best known heuristic algorithm is the *differencing* algorithm and is given in [15]. Briefly, the differencing algorithm reduces the size of the set by one in each iteration by replacing the two largest numbers with their absolute difference. This is equivalent to deciding that the two largest numbers will go into different sets without actually committing which set receives which number at this time. The forward leg of the algorithm terminates when the set is reduced to a single number. The last number standing will represent the discrepancy of the two sets (the absolute difference of their sums). The algorithm then backtracks and at each step replaces one difference number with its components in such a way that discrepancy of the two sets remains constant. For a detailed discussion and an example of the algorithm refer to [15].

The number partitioning algorithm has a complexity of $\mathcal{O}(n \log n)$ [17]. The balanced partitioning algorithm typically halves the number of partitions in each round resulting in typical complexity of $\mathcal{O}(n(\log n)^2)$. In an absolute worst case scenario the number of iterative partitions is n and the complexity of the algorithm is bounded by $\mathcal{O}(n^2 \log n)$.

An example of partitioning a 2D computational domain of 10×10 cells across 5 resources where the partition sizes are given as $S = \{4, 17, 22, 26, 31\}$ is given in Table 2. Using the balanced partitioning algorithm results in slight adjustments to partition sizes at the end; with final partition sizes being {5, 15, 20, 25, 35} as shown in the last row of the table.

[4] In this section, the use of term 'optimal' refers to the results obtained from the numerical optimization algorithm. We realize that given the nonlinear nature of the problem strict optimality of the optimization algorithm is not guaranteed.

Please cite this article in press as: R. Shams, P. Sadeghi, On optimization of finite-difference time-domain (FDTD) computation on heterogeneous and GPU clusters, J. Parallel Distrib. Comput. (2010), doi:10.1016/j.jpdc.2010.10.011



(a) Stripe. (b) Balanced.

Fig. 2. Comparison of the stripe and balanced partitioning algorithms. The balanced partitioning results in less discrepancy compared with the desired partition sizes.

We compare the performance of the balanced partitioning algorithm with the baseline *stripe* partitioning algorithm where the computational domain is simply partitioned along a single axis. For the stripe algorithm, we choose partition sizes proportional to the resource's performance (i.e. in accordance to (24)). This is similar to what is typically used in FDTD parallelization on homogeneous clusters today. Fig. 2 shows one simple stripe partitioning of the previous example. The first thing to notice is that the discrepancy between achievable partitions and desired partitions is higher. This is the result of larger round-off errors due to the integral dimensions of the computational domain, which in turn translates into an even less optimum distribution of the computational load.

4. Results

In this section, we present a number of simulation results for heterogeneous and homogeneous clusters and compare the performance of the balanced and stripe partitioning methods in respect to the derived bounds. We find it more intuitive to show the results in terms of achievable throughput rather than the computation time. The throughput is defined as the ratio of the number of cells to the processing time (i.e. computation or transfer time). This has the added advantage of having the results normalized to the size of the computational domain. The measurements will be given as reciprocals of α_i and β_i in mega cells per second (MCells/s). We also note that in the context of throughput we will be talking about upper bounds which are inversely related to computation time lower bounds.

As will be demonstrated in this section, the results support our claim that the heuristic partitioning algorithm chosen for our implementation performs reasonably well. The blue curve in the following graphs (optimal partitioning curve) represents the absolute upper bound for the performance of any partitioning algorithm and as demonstrated by the experiments the balanced partitioning algorithm is very close to this upper bound.

We also compared the performance of the balanced partitioning method with the column-based method in a number of experiments. For these experiments, we were limited to 2D domains as the column-based method (in its current form) is suitable for 2D partitioning. We found that the column-based algorithm performed only marginally better than balanced partitioning. It did not result in any significant improvement in throughput (in fact the increase in throughput was only between 0.14% and 0.65%). For this reason and due to our interest in higher dimensional FDTD, in the following experiments, we will only show the results for 3D domains and use the balanced partitioning method that can be easily used in any number of dimensions.

Example 1 (*Homogeneous GPU Cluster*)**.**

- *Domain*: $500 \times 500 \times 500$ cells.
- *Cluster*: homogeneous, 2–8 NVIDIA GT200 GPUs on a single host (certain motherboards allow up to 8 GT200 GPUs to be installed), 3D FDTD performance on a GT200 GPU $\alpha_i^{-1} = 493$ MCells/s.
- *Link*: PCI-E x16, nominal bandwidth: 8 GB/s, actual bandwidth measured at 2.4 GB/s and 4.3 GB/s on two different motherboards, combined throughput is $\sum_i \beta_i^{-1} = 50$ MCells/s and 93 MCells/s respectively.
- *Notes*: The throughput measurements were performed with host memory allocated as standard page-able memory. The throughput can be improved by using page-locked (pinned) memory. However, pinned memory is a scarce resource and not suitable for typically large memory demands of FDTD applications. The total bandwidth is constant and as we increase the number of GPUs the data transfer rate per GPU decreases.

In Fig. 3, we show the performance of the cluster as a function of the number of GPUs on two different hosts with different actual PCI-E bandwidths. A number of observations can be made from Fig. 3: (a) the scalability of the cluster improves with increased throughput; (b) the performance of the balanced partitioning is close to the optimal solution; (c) the balanced partitioning method outperforms the stripe method by up to 47% for the slower host and up to 38% for the faster host; (d) as the number of GPUs increases, the transfer rate per GPU decreases and the advantage of better partitioning is more emphatically demonstrated; (e) the stripe method exhibits poor scalability and the performance plateaus with 5 or 6 GPUS whereas the balanced partitioning continues to scale.

Example 2 (*Heterogeneous GPU Cluster*)**.**

- *Domain*: $500 \times 500 \times 500$ cells.
- *Cluster*: heterogeneous, (a) 1–4 NVIDIA GT200 GPUs, 3D FDTD performance measured at $\alpha_i^{-1} = 493$ MCells/s, (b) 1–4 NVIDIA GT80 GPUs, 3D FDTD performance measured at $\alpha_i^{-1} = 140$ MCells/s.
- **Link**: PCI-E x16, nominal bandwidth: 8 GB/s, actual bandwidth measured at 2.4 GB/s and 4.3 GB/s on two different motherboards, combined throughput is $\sum_i \beta_i^{-1} = 50$ MCells/s and 93 MCells/s respectively.

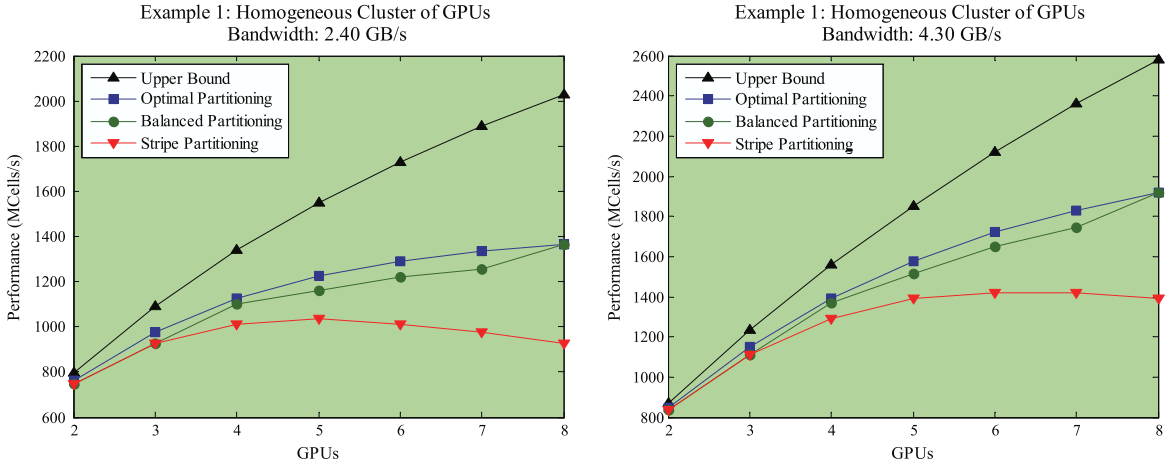
In this example, we look at a heterogeneous cluster of GPUs installed in a single host. The cluster comprises equal numbers of GT80 and GT200 GPUs. The results depicted in Fig. 4 demonstrate the superior performance and scalability of the balanced partitioning for a heterogeneous cluster. The results are more or less consistent with earlier observations in the previous example.

Example 3 (*Homogeneous CPU Cluster*)**.**

- *Domain*: $1500 \times 1500 \times 1500$ cells.
- *Cluster*: homogeneous, 4–32 nodes each with Quad-core Intel Core i7 2.67 GHz CPUs, 3D FDTD performance measured at $\alpha_i^{-1} = 72.8$ MCells/s.
- *Link*: (a) Gigabit Ethernet, actual bandwidth: 0.1 GB/s, $\sum_i \beta_i^{-1} = 2.1$ MCells/s, (b) 10 Gigabit Ethernet throughput, actual bandwidth: 0.6 GB/s, $\sum_i \beta_i^{-1} = 12.5$ MCells/s.
- *Note*: OpenMP is used to parallelize the FDTD code on each node.

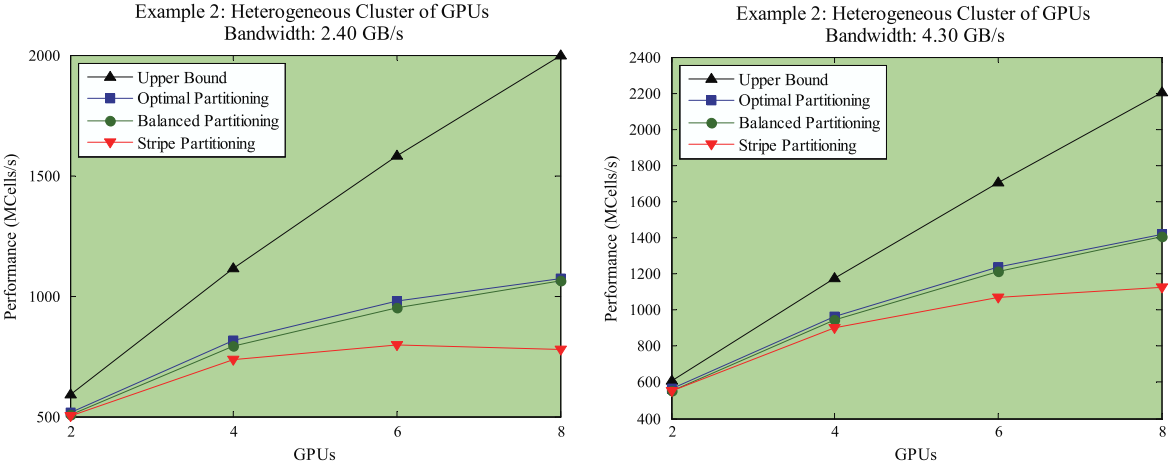
In this example, a larger computational domain is distributed to a cluster of PCs. We compare the scalability and performance of the cluster over a Gigabit and 10 Gigabit network. The simulations predict that for a Gigabit network the cluster saturates with 8 nodes when balanced partitioning is used. Using the stripe partitioning method saturates the cluster with only 4 nodes. Fig. 5(a) also demonstrates that using the balanced partitioning method results in more than 31% improvement in peak performance compared to the stripe method. In Fig. 5(b) the network bandwidth is increased by a factor of 6. This has a significant impact on the performance of the cluster. The balanced partitioning method scales up to 32 nodes now and achieves a peak performance of 669 MCells/s compared to 180 MCells/s on the Gigabit network. Also note that the peak performance of the balanced partitioning is almost 76% higher than the peak performance of the stripe partitioning method.

Please cite this article in press as: R. Shams, P. Sadeghi, On optimization of finite-difference time-domain (FDTD) computation on heterogeneous and GPU clusters, J. Parallel Distrib. Comput. (2010), doi:10.1016/j.jpdc.2010.10.011



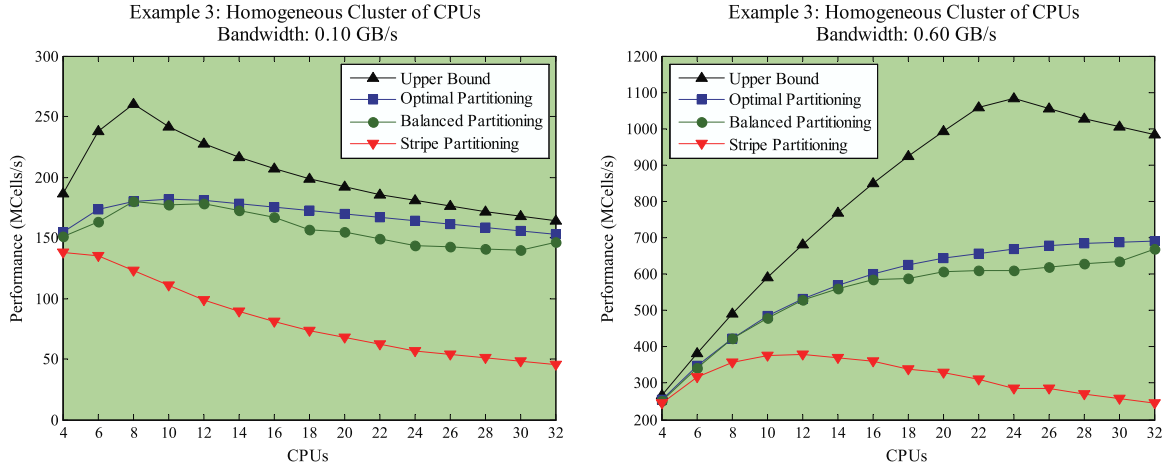
(a) Bandwidth: 2.4 GB/s. (b) Bandwidth: 4.3 GB/s.

Fig. 3. Comparison of balanced and stripe partitioning algorithms. The balanced partitioning results in up to 47% improvement in performance and is close to the performance of the optimal solution.



(a) Bandwidth: 2.4 GB/s. (b) Bandwidth: 4.3 GB/s.

Fig. 4. Comparison of balanced and stripe partitioning algorithms on a heterogeneous cluster. The balanced partitioning results in up to 34% improvement in performance and is close to the performance of the optimal solution.

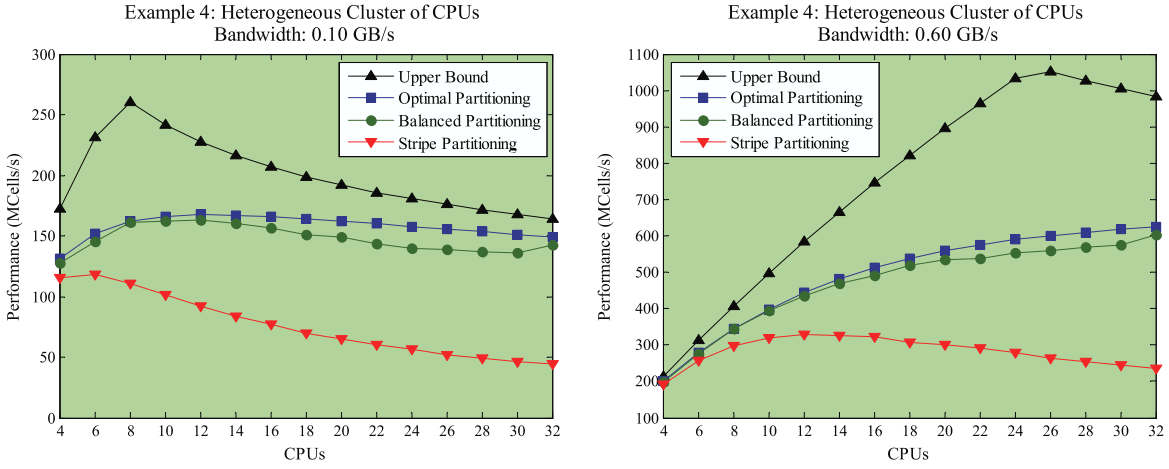


(a) Bandwidth: 0.1 GB/s. (b) Bandwidth: 0.6 GB/s.

Fig. 5. Comparison of balanced and stripe partitioning algorithms on a homogeneous cluster of PCs and for different network bandwidths. The network bandwidth is the main bottleneck. Increasing network bandwidth improves the scalability of the cluster.

The advantage of PC clusters over GPU clusters is their larger memory size. This makes simulation of larger computational domains possible, albeit at the cost of lower performance. As shown in previous examples, a cluster of GPUs on a single host

Please cite this article in press as: R. Shams, P. Sadeghi, On optimization of finite-difference time-domain (FDTD) computation on heterogeneous and GPU clusters, J. Parallel Distrib. Comput. (2010), doi:10.1016/j.jpdc.2010.10.011



(a) Bandwidth: 0.1 GB/s. (b) Bandwidth: 0.6 GB/s.

Fig. 6. Comparison of balanced and stripe partitioning algorithms on a heterogeneous cluster of PCs and for different network bandwidths. The network bandwidth is the main bottleneck. Increasing network bandwidth improves the scalability of the cluster.

exceeds a performance level of 1000 MCells/s. Perhaps to solve the dilemma, one can create a cluster of multi-GPU nodes to address both memory capacity and performance problems. However, such a cluster will hardly scale unless one is prepared to invest in higher bandwidth technologies such as quad data rate (QDR) InfiniBand.

Example 4 (*Heterogeneous CPU Cluster*).

- *Domain*: $1500 \times 1500 \times 1500$ cells.
- *Cluster*: heterogeneous, (a) 2–16 nodes each with Quad-core Intel Core i7 2.67 GHz CPUs, 3D FDTD performance measured at $\alpha_i^{-1} = 72.8$ MCells/s, (b) 2–16 node each with Quad-core Intel Core Duo 2.66 GHz CPUs, 3D FDTD performance measured at $\alpha_i^{-1} = 39.1$ MCells/s.
- *Link*: (a) Gigabit Ethernet, actual bandwidth: 0.1 GB/s, $\sum_i \beta_i^{-1} = 2.1$ MCells/s, (b) 10 Gigabit Ethernet throughput, actual bandwidth: 0.6 GB/s, $\sum_i \beta_i^{-1} = 12.5$ MCells/s.

In our last example, we look at results from a heterogeneous cluster of up to 32 nodes. The cluster comprises an equal number of quad-core Core i7 and quad-core Core Duo nodes. Despite the disparity in performance of the nodes, the balanced partitioning achieves reasonable scalability with the faster network.

We conclude with an implementation note on the use of MPI for message exchange between cluster nodes. Following an FDTD iteration, each node needs to send the field values associated with its boundary cells to the neighboring nodes. The send process can be blocking or non-blocking (non-blocking is preferred). It also needs to receive boundary field values from neighboring nodes. The next iteration cannot start without completing the receive and hence there is an implicit synchronization at this stage regardless of whether one uses blocking or non-blocking receive. Now consider the case where a node has sent its data in a non-blocking manner and has already received all the data it needs. Even so, the node cannot start the next iteration as it will overwrite data in its send buffer. The safe course of action is to ensure that the send operation is acknowledged by all the recipients before overwriting send buffers. Obviously, there are ways around this limitation at the cost of increased memory use which will result in an increase in the effective bandwidth. However, this may complicate the implementation not to mention the fact that typical FDTD problems are usually memory-bound and thrive to minimize memory usage.

5. Discussion

For ease of reference, the main results of the paper are summarized here:

- The problem of distribution of FDTD load to a heterogeneous cluster can be formulated as a minimax problem over n-element partitions of the computational domain Ω

$$t_{\text{opt}} = \min_{\Gamma_n} \left\{ \max_i \left\{ \alpha_i |\Omega_i| + \sum_{j \neq i} \beta_{ij} |\partial\Omega_i \cap \partial\Omega_j| + \gamma_i |\partial\Omega_i \cap \partial\Omega| \right\} \right\}.$$

- A simpler problem is formulated by relaxing the conditions of the original problem as

$$t_{\text{opt}} = \min_{\Gamma_n} \left\{ \max_i \{\alpha_i |\Omega_i| + \beta_i |\partial\Omega_i|\} \right\}, \qquad \sum_i |\Omega_i| = |\Omega|.$$

- Two lower bounds can be found for the relaxed problem. The combination of which gives the following bound

$$t_{\text{opt}} \geq \max \left\{ \left(\sum_i \frac{1}{\alpha_i} \right)^{-1} \left(1 + 2d \frac{\beta_q}{\alpha_q} |\Omega|^{-\frac{1}{d}} \right) |\Omega|, \left[|\Omega| \left(\sum_i (\alpha_i + 2d\beta_i)^{-\frac{d}{d-1}} \right)^{-1} \right]^{1-\frac{1}{d}} \right\}.$$

The results set an upper bound on achievable performance improvements that can be used to predict the extent to which parallelization is practically beneficial. In a dynamic cluster where resources may become available during the lifetime of computations one may have to decide if it is beneficial to repartition the problem to utilize the newly available resources. Redistribution of the problem may incur significant traffic and an algorithm may not redistribute the problem until such time that enough computational resources are available to justify the overhead or may even determine that redistribution of the problem is detrimental to the overall performance. One can simply use the bounds or better sill estimate the performance of the redistributed configuration before making such decisions.

The experiments that we performed were based on the basic implementation of FDTD in the context of Maxwell equations. It should be noted that our computational FDTD model, analytical derivations, and partitioning method are equally valid in other contexts such as FDTD for acoustic equations and also for more complex *stencils* where higher degree discretization of parameters is applied.

Please cite this article in press as: R. Shams, P. Sadeghi, On optimization of finite-difference time-domain (FDTD) computation on heterogeneous and GPU clusters, J. Parallel Distrib. Comput. (2010), doi:10.1016/j.jpdc.2010.10.011

The results show that significant performance gains can be achieved by proper distribution of load across a heterogeneous cluster. We used a heuristic algorithm for partitioning the computational domain and showed by experiment that the algorithm achieves performance levels close to ideal partition sizes obtained by the numerical optimization algorithm. The algorithm is simple and efficient and given the results there does not seem to be any significant room for improvement by switching to other partitioning algorithms. Optimal load distribution also improves scalability which means that more computational resources can be efficiently utilized. The burden of optimizing load distribution as described in this paper is negligible compared to the effort of parallelizing an FDTD code. A properly parallelized FDTD code should in principle be able to run with non-equal partitions and should benefit from the method presented in this paper with minimal effort. As an additional benefit existing FDTD applications can efficiently run on heterogeneous clusters of similar technology. Fully heterogeneous applications that can run across technology boundaries will be a natural extension.

One interesting observation is that the balanced partitioning algorithm is usually closer to the absolute upper bound of partitioning when the number of nodes is a power of 2 (e.g. see the performance of the algorithm in Figs. 5 and 6 for 8 and 32 nodes). This is perhaps due to the fact that the algorithm halves the domain in each iteration and therefore does a better job of partitioning when the number of partitions is a power of 2.

The question remains whether the numerical optimization finds the global minimum of (15). Based on the experiments, we believe the numerical optimization results are optimal or very close to optimal. This is a claim that can be more comfortably asserted if one is able to derive tighter bounds or indeed prove the optimality of the solution.

Acknowledgments

This work was supported in part by the Australian Research Council (ARC) Discovery Project DP1093149 and in part by the ARC/Microsoft Linkage Project LP100100588. The views expressed herein are those of the authors and are not necessarily those of the funding organizations.

References

[1] S. Adams, J. Payne, R. Boppana, Finite difference time domain (FDTD) simulations using graphics processors, in: High Performance Computing Modernization Program Users Group Conference, 2007, pp. 334–338.
[2] O. Beaumont, V. Boudet, F. Rastello, Y. Robert, Matrix multiplication on heterogeneous platforms, IEEE Transactions on Parallel and Distributed Systems 12 (10) (2001) 1033–1051.
[3] J. Berenger, A perfectly matched layer for the absorption of electromagnetic waves, Journal of Computational Physics 114 (2) (1994) 185–200.
[4] J.F. Bonnans, J.C. Gilbert, C. Lemaréchal, C.A. Sagastizábal, Numerical Optimization: Theoretical and Practical Aspects, second ed., Springer, 2006.
[5] W. Chen, P. Kosmas, M. Leeser, C. Rappaport, An FPGA implementation of the two-dimensional finite-difference time-domain (FDTD) algorithm, in: Proc. International Symposium on Field Programmable Gate Arrays, 2004, pp. 213–222.
[6] P.E. Crandall, M.J. Quinn, Block data decomposition for data-parallel programming on a heterogeneous workstation network, in: Proc. Int. Symp. on High Performance Distributed Computing, 1993, pp. 42–49.
[7] Compute Unified Device Architecture (CUDA) Programming Guide, version 2.2. NVIDIA, 2009. http://developer.nvidia.com/object/cuda.html.
[8] J. Dongarra, A. Lastovetsky, An overview of heterogeneous high performance and grid computing, in: B.D. Martino, J. Dongarra, A. Hoisie, L. Yang, H. Zima (Eds.), Engineering the Grid, American Scientific Publishers, 2006.
[9] J.P. Durbano, J.R. Humphrey, F.E. Ortiz, P.F. Curt, D.W. Prather, M.S. Mirotznik, Hardware acceleration of the 3D finite-difference time-domain method, in: Proc. IEEE Antennas and Propagation Society Int. Symposium, vol. 1, 2004, pp. 77–80.
[10] S.D. Gedney, An anisotropic perfectly matched layer-absorbing medium for the truncation of FDTD lattices, IEEE Transactions on Antennas and Propagation 44 (12) (1996) 1630–1639.
[11] W. Gropp, E. Lusk, A. Skjellum, Using MPI: Portable Parallel Programming with the Message Passing Interface, second ed., MIT Press, Cambridge, MA, USA, 1999.
[12] C. Guiffaut, K. Mahdjoubi, A parallel FDTD algorithm using the MPI library, IEEE Antennas and Propagation Magazine 43 (2) (2001) 94–103.
[13] M.C. Hughes, M.A. Stuchly, Hybrid parallel finite difference time domain simulation of nanoscale optical phenomena, in: Int. Conf. on Wireless Communications and Applied Computational Electromagnetics, 2005, pp. 132–135.
[14] M. Kaddoura, S. Ranka, A. Wang, Array decompositions for nonuniform computational environments, Journal of Parallel and Distributed Computing 36 (2) (1996) 91–105.
[15] N. Karmarker, R.M. Karp, The differencing method of set partitioning, Technical Report, University of California at Berkeley, Berkeley, CA, USA, 1983.
[16] H. Kawaguchi, K. Takahara, D. Yamauchi, Design study of ultrahigh-speed microwave simulator engine, IEEE Transactions on Magnetics 38 (2) (2002) 689–692.
[17] R. Korf, A complete anytime algorithm for number partitioning, Artificial Intelligence 106 (2) (1998) 181–203.
[18] S.E. Krakiwsky, L.E. Turner, M.M. Okoniewski, Acceleration of finite-difference time-domain (FDTD) using graphics processor units (GPU), in: IEEE Int. Microwave Symposium, vol. 2, 2004, pp. 1033–1036.
[19] A. Lastovetsky, On grid-based matrix partitioning for heterogeneous processors, in: Proc. Int. Symp. on Parallel and Distributed Computing, ISPDC, 2007, pp. 383–390.
[20] A.L. Lastovetsky, J. Dongarra, High Performance Heterogeneous Computing, Wiley, Hoboken, New Jersey, USA, 2009.
[21] D. Liuge, L. Kang, K. Fanmin, Parallel 3D finite difference time domain simulations on graphics processors with CUDA, in: Int. Conf. on Computational Intelligence and Software Engineering, December 2009, pp. 1–4.
[22] S. Mertens, The easiest hard problem: number partitioning, in: A. Percus, G. Istrate, C. Moore (Eds.), Computational Complexity and Statistical Physics, Oxford University Press, New York, 2006, pp. 125–139.
[23] G. Mur, Absorbing boundary conditions for the finite-difference approximation of the time-domain electromagnetic-field equations, IEEE Transactions on Electromagnetic Compatibility 23 (4) (1981) 377–382.
[24] OpenMP Application Programming Interface, version 3.0. 2009. OpenMP: http://openmp.org/wp/openmp-specifications/.
[25] G.F. Pinton, J. Dahl, S. Rosenzweig, G.E. Trahey, A heterogeneous nonlinear attenuating full-wave model of ultrasound, IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control 56 (3) (2009) 474–488.
[26] J.A. Roden, S.D. Gedney, Convolutional PML (CPML): an efficient FDTD implementation of the CFS-PML for arbitrary media, Microwave and Optical Technology Letters 27 (5) (2000) 334–339.
[27] Z.S. Sacks, D.M. Kingsland, R. Lee, J.F. Lee, A perfectly matched anisotropic absorber for use as an absorbing boundary condition, IEEE Transactions on Antennas and Propagation 43 (12) (1995) 1460–1463.
[28] T.P. Stefanski, T.D. Drysdale, Acceleration of the 3D ADI-FDTD method using graphics processor units, in: IEEE Int. Microwave Symposium, 2009, pp. 241–244.
[29] A. Taflove, S.C. Hagness, Computational Electrodynamics: The Finite Difference Time Domain Method, third ed., Artech House Inc., Norwood, MA, USA, 2005.
[30] N. Takada, T. Shimobaba, N. Masuda, T. Ito, High-speed FDTD simulation algorithm for GPU with compute unified device architecture, in: Proc. IEEE Antennas and Propagation Society Int. Symposium, 2009, pp. 1–4.
[31] K. Yee, Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media, IEEE Transactions on Antennas and Propagation 14 (1966) 302–307.

Dr. Ramtin Shams (ramtin.shams@anu.edu.au) is a Fellow (senior lecturer) at the School of Engineering at the Australian National University (ANU). He received his B.E. and M.E. degrees in electrical engineering from Sharif University of Technology, Tehran, and his Ph.D. in biomedical engineering from ANU. He received an Australian Postdoctoral (APD) Fellowship in 2009 and was the recipient of a Fulbright scholarship in 2008. He has more than ten years of industry experience in the ICT sector and worked as the CTO of GPayments Pty. Ltd between 2001 to 2007. His research interests include medical image analysis, high performance computing, and wireless communications.

Dr. Parastoo Sadeghi (parastoo.sadeghi@anu.edu.au) is a Fellow (senior lecturer) at the Research School of Information Sciences and Engineering at ANU. She received her B.E. and M.E. degrees in electrical engineering from Sharif University of Technology, Tehran, and her Ph.D. degree in electrical engineering from The University of New South Wales in Sydney, in 2006. In 2003 and 2005, she received two IEEE Region 10 Paper Awards for her research in the information theory of time-varying fading channels. Her research interests include applications of signal processing, information theory, and high performance computing in telecommunications and medical image analysis.

Please cite this article in press as: R. Shams, P. Sadeghi, On optimization of finite-difference time-domain (FDTD) computation on heterogeneous and GPU clusters, J. Parallel Distrib. Comput. (2010), doi:10.1016/j.jpdc.2010.10.011