HIGH PERFORMANCE COMPUTING FOR THE FINITE DIFFERENCE TIME DOMAIN METHOD WITH THE HUYGENS ABSORBING BOUNDARY CONDITION

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Engineering and Physical Sciences

2014

By
Hanan Almeer
School of Electrical and Electronic Engineering

Contents

Abstract **11**

Declaration **12**

Copyright **13**

Acknowledgements **14**

1 Finite Difference Time Domain Method **15**

1.1 The FDTD Method 15

1.1.1 Yee's FDTD Method 15

1.1.2 Frequency Dependent Media 19

1.1.3 Features of the FDTD Method 19

2 Absorbing Boundary Conditions **21**

2.1 ABC Based on One-Way Wave Equation 22

2.1.1 Numerical formulation from one-way wave equation 22

2.1.2 Mur's ABC 23

2.2 Complex Frequency-Shifted Perfectly Matched Layer ABCs 24

3 The Huygens Absorbing Boundary Condition **25**

3.1 The Huygens Surface 25

3.2 The Huygens Surface for Implementing an Incident Wave 26

3.2.1 The 1D case 28

3.2.2 The Trivial Implementation of the Huygens Surface 30

3.2.3 Huygens Surface with an Object 31

3.2.4 Huygens Surface in 3D 32

3.2.5 The Direction of the Incident wave 34

3.3 Total-Field/Scattered-Field Technique 37

3.3.1 Benefits of the Connecting Surface in TF/SF technique . . 38
3.3.2 TF/SF 1D Formulation 39
3.4 Huygens Absorbing Boundary Conditions 41
3.4.1 Principle of ABC based on Huygens surface 42
3.4.2 HABC in FDTD space 42
3.4.3 3D HABC in FDTD space 44
3.4.4 Numerical Experiments 50
3.5 The Stretched-Mesh HABC 53
3.5.1 Numerical Experiments 55

4 High Performance Computing 68
4.1 Parallel Computing with GPUs 69
4.2 Parallel Computing with Shared Memory 72

5 Parallelisation on Graphics Processing Units 75
5.1 General Purpose GPU Computing with CUDA 75
5.2 Existing Implementations of the FDTD Method on GPU Hardware 79
5.2.1 First approach 79
5.2.2 Second approach 80
5.2.3 Third approach 81
5.3 The implementation of the FDTD method on GPU hardware . . . 83
5.4 Implementations of 3D HABC on GPU hardware 85
5.4.1 First implementation 85
5.4.2 Second implementation 85
5.5 Implementations of 3D CFS-PML on GPU hardware 90
5.6 Numerical experiments 90
5.6.1 Simulations Settings 90
5.6.2 Simulations Results 93
5.6.3 Conclusion 98

6 Parallelisation on Shared Memory Architecture 100
6.1 Threads in Shared Memory Architecture 100
6.2 Strong Scaling and Weak Scaling 101
6.3 Speedup Ratio and Parallel Efficiency 104
6.4 Numerical Experiments 106
6.4.1 Environmental Settings 107

6.4.2 Experimental Results 107
6.4.3 Conclusion 109

7 Practical Application with Human Body 112
7.1 Numerical Modeling of the Human Body 113
7.2 Numerical experiments 116
7.3 Conclusion 117

8 Conclusion and Future Works 120
8.1 Conclusion 120
8.2 Suggested Future Works 121

A List of Publications 122

Bibliography 123

List of Tables

5.1 Specifications of Tesla Fermi C2075 GPUs. 95
5.2 Specifications of Tesla Kepler K20Xm GPUs. 98

6.1 The variation of the size of the FDTD space and the corresponding memory usage when the size of the FDTD space allocated to each thread is set to 500^3 in weak scaling. 107

7.1 Parameters of the human tissues. Part(1). 114
7.2 Parameters of the human tissues. Part(2). 115

List of Figures

3.1 An incident wave extends upon a certain length L in space, from its front to its back with duration of L/c in time at a given point [1]. 27

3.2 The use of Huygens surface to generate the wave [1]. 27

3.3 A source of field exterior to a closed surface [1]. 28

3.4 The zero thickness current is spread upon a width of one FDTD cell Δx in the FDTD domain [1]. 29

3.5 Introducing sources using the Huygens surface in 1D FDTD space [1]. 29

3.6 The use of Huygens surface to generate the wave when an object is present [1]. 31

3.7 Setting the direction of the wave generated by the Huygens surface with the following three angles: ϕ,θ and ψ. 35

3.8 The direction of the wave generated by the Huygens surface when the angles ϕ,θ and ψ equal to 0,0 and 180 respectively. 35

3.9 The direction of the wave generated by the Huygens surface when the angles ϕ,θ and ψ equal to 90,0 and 180 respectively. 36

3.10 The direction of the wave generated by the Huygens surface when the angles ϕ,θ and ψ equal to $0, 90$ and 180 respectively. 36

3.11 Total-field/Scattered-field zoning [2]. 38

3.12 Total-field/Scattered-field component locations, 1D [2]. 39

3.13 1D implementation of Huygens surface in FDTD grid [3]. 42

3.14 The object, Huygens excitation and the HABC in 3D FDTD space. The HABC is placed 3 cells away from the object. 51

3.15 Observation at the corner of the plate. 1 ns Gaussian pulse Huygens excitation. $\Delta t = 18$ps. 51

3.16 Observation at the corner of the plate. 1 ns Unit-Step Huygens excitaion. $\Delta t = 18$ps. 52

3.17 The combination of HABC placed close to an object with a large exterior domain ended with a PEC condition [4]. 53
3.18 HABC and the stretch mesh [4]. 54
3.19 The SM-HABCA is tested with a 280-cell cube of fat covered with 10 cells layer of skin. The Debye parameters for fat are $\sigma = 0.03710634$ (S/m), $\epsilon_{\mathrm{S}} = 5.53071189$, $\epsilon_{\infty} = 3.99812841$ and $\tau_{\mathrm{D}} = 23.6329289$ (ps) and the Debye parameters for skin are $\sigma = 0.54073572$ (S/m), $\epsilon_{\mathrm{S}} = 47.93014336$, $\epsilon_{\infty} = 29.85054779$ and $\tau_{\mathrm{D}} = 43.6257593$ (ps). 56
3.20 A cube with dispersive materials is tested with Huygens excitation and SM-HABC. 57
3.21 Electric field at the center point of a 300-cell cube composed of skin and fat. 58
3.22 A cube with dispersive materials. 2D-plane is observed when y is constant. 59
3.23 A cube with dispersive materials. The colored image of the observed 2D-plane when y is constant at t = $500\Delta t$. 60
3.24 A cube with dispersive materials. The colored image of the observed 2D-plane when y is constant at t = $700\Delta t$. 61
3.25 A cube with dispersive materials. The colored image of the observed 2D-plane when y is constant at t = $850\Delta t$. 62
3.26 A cube with dispersive materials. The colored image of the observed 2D-plane when y is constant at t = $950\Delta t$. 63
3.27 A cube with dispersive materials. The colored image of the observed 2D-plane when y is constant at t = $1050\Delta t$. 64
3.28 A cube with dispersive materials. The colored image of the observed 2D-plane when y is constant at t = $1150\Delta t$. 65
3.29 A cube with dispersive materials. The colored image of the observed 2D-plane when y is constant at t = $1250\Delta t$. 66
3.30 Frequency reflection coefficient with the HABC and CFS-PML ABCs. 67

4.1 Shared memory architecture. 69
4.2 Distributed memory architecture. 69
4.3 Hybrid distributed-shared memory architecture [5]. 70
4.4 Parallel computing with GPUs [6]. 71

4.5 GPU computing with OpenACC directives [6]. 72
4.6 Threads access to local and shared data in shared memory architecture [7]. 73
4.7 Threads in parallel computing with shared memory architecture [7]. 73

5.1 The Tesla unified graphics and computing GPU architecture [8]. . 76
5.2 CUDA grid organisation in Tesla [9]. 77
5.3 CUDA device memory model [9]. 78
5.4 The 3D FDTD space decomposition in the first approach [10]. . . 80
5.5 The 2D decomposition in the first and second approaches when using blocks and threads on a $k =$ constant plane. [10]. 81
5.6 The 3D FDTD space decomposition in the second approach [10]. 82
5.7 The 3D FDTD space decomposition in the third approach [10]. . 82
5.8 Patterns of memory access for multiple threads in a block [10]. . 84
5.9 The Huygens surfaces of the HABC are divided into 3 groups. . . 87
5.10 The Huygens surfaces of the HABC are divided into 3 groups. . . 89
5.11 The Huygens surfaces of the HABC are divided into 3 groups. . . 89
5.12 The CFS-PML regions to calculate $\boldsymbol{E}_x$ and $\boldsymbol{H}_x$ with ncell= 2. . . 91
5.13 The CFS-PML regions to calculate $\boldsymbol{E}_y$ and $\boldsymbol{H}_y$ with ncell= 2. . . 92
5.14 The CFS-PML regions to calculate $\boldsymbol{E}_z$ and $\boldsymbol{H}_z$ with ncell= 2. . . 93
5.15 The simulations setting of the GPGPU implementations. 94
5.16 Performance of the two GPGPU implementations of the HABC and the CFS-PML on Tesla Fermi C2075 GPUs. The arrays of electric and magnetic fields are moved from GPUs to CPUs at each time step for observation. 96
5.17 Performance of the two GPGPU implementations of the HABC and the CFS-PML on Tesla Fermi C2075 GPUs. No data is moved from GPUs to CPUs for observation (no data output). 97
5.18 Performance of the first GPGPU implementation of the HABC and the CFS-PML on Tesla Kepler GPUs. 99

6.1 Threads in parallel computing. 101
6.2 The size of the computations allocated to each thread as the number of threads increases in strong scaling. 103

6.3 The variation of the computational time when the number of threads increases in strong scaling. 104
6.4 The size of the computations allocated to each thread as the number of threads increases in weak scaling. 105
6.5 The variation of the computational time when the number of threads increases in weak scaling. 106
6.6 The variation of the size of the FDTD space and the corresponding memory usage when the size of the FDTD space allocated to each thread is set to 500^3 in weak scaling. 108
6.7 The speedup ratio of the OpenMP codes in strong scaling. The fixed size of the FDTD space is 1000^3 and the memory usage is 100GB. 109
6.8 The parallel efficiency of the OpenMP codes in strong scaling. The fixed size of the FDTD space is 1000^3 and the memory usage is 100GB. 110
6.9 The weak scaling efficiency of the OpenMP codes. The FDTD space vary from 500^3 to 1000^3. The size of the FDTD space is set to 500^3 for each thread. 111
6.10 Performance of the HABC and CFS-PML implementations on shared memory architecture. 111

7.1 The Huygens excitation planes are placed around the human body and the direction of the wave is from the front to the back of the human body. 116
7.2 A snapshots of a 2D colored plane observations of the FDTD domain that has a human body at the center surrounded with Huygens excitation planes in the SM-HABC experiments. y is constant and the snapshots are at $t = 250\Delta t$, $t = 650\Delta t$ and $t = 800\Delta t$. The snapshots show the FDTD domain including the HABC and the stretched cells which are out the Huygens excitation planes. . 118

7.3 A snapshots of a 2D colored plane observations of the FDTD domain that has a human body at the center surrounded with Huygens excitation planes in the CFS-PML experiments. y is constant and the snapshots are at $t = 250\Delta t$, $t = 650\Delta t$ and $t = 800\Delta t$. The snapshots illustrate the FDTD domain starting from the Huygens excitation planes and what is inside the Huygens excitation planes. The CFS-PML cells do not appear in the snapshots. . . . 119

Abstract

The Finite-Difference Time-Domain (FDTD) is the most widely used method for solving Maxwell's equations in the time domain. Since the FDTD domains are usually open regions, an Absorbing Boundary Condition (ABC) is needed to absorb the outgoing waves and simulate the extension to infinity. The most popular and effective ABCs is the Complex Frequency- Shifted Perfectly Matched Layer (CFS-PML) ABCs. The CFS-PML ABCs absorbs almost all of the outgoing waves, but the implementation of the CFS-PML ABCs is complicated and more computational resources such as memory and CPU time are required.

In this thesis, a new ABC called Huygens Absorbing Boundary Condition (HABC), which is simpler to implement than the CFS-PML, is presented. The accuracy of the HABC is studied and compared with that of the CFS-PML. A combination of the HABC and the Stretch Mesh (SM) is introduced. The SM-HABC is tested with an object with dispersive materials.

For practical applications, the FDTD method with the HABC codes are parallelised on the Graphics Processing Units (GPUs) using the Compute Unified Device Architecture programming model (CUDA) in this thesis. Two implementations of the HABC on GPUs are presented. The performance of the two implementations are studied and compared with the implementation of the CFS-PML on GPUs. In addition, the FDTD with the HABC codes are parallelized on the shared memory architecture using Open Multi-Processing (OpenMP). The OpenMP code of the HABC is scaled and the results are compared with the scaled OpenMP code of the CFS-PML.

Finally, Huygens excitation is used in this thesis to heat up the human body as an application of hyperthermia which is a cancer treatment. The SM-HABC is also used in human body simulations. A comparison between the use of the SM-HABC and the CFS-PML in human body simulations is introduced.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

Foremost, I would like to express my heartfelt gratitude to my supervisor Dr Fumie Costen for her guidance, time, support and feedback throughout my research. Her advice have been invaluable in all aspects. She is not only supervisor, but dear friend. Thank you Fumie!

It is my great pleasure to thank Jean-Pierre Bérenger for his kind explanations and advices on many occasions of this research.

Thanks to RIKEN Integrated Cluster of Clusters and Kyushu University, Japan for the access to their computational resources to perform the numerical experiments of this project.

A very special thanks goes out to Ahmed Aloumi for his warm support to me for the whole time. Thanks to Noura and Abdullah Aloumi. And above all, I am very grateful to Saeed Almeer and Khawlah AlEssa for their care, encouragement and support throughout my entire life.

Chapter 1

Finite Difference Time Domain Method

The Finite Difference Time Domain (FDTD) [2] is the most widely used method to solve Maxwell's equation in time domain [11]. The method owns its success to the power and simplicity it provides [12]. This chapter briefly introduces the FDTD method.

1.1 The FDTD Method

1.1.1 Yee's FDTD Method

Yee introduced the FDTD method in 1966 [13]. The idea of Yee's FDTD method is approximating the following Maxwell's curl equations in time domain using the central difference approximations:

$$\frac{\partial \boldsymbol{B}}{\partial t} = -\nabla \times \boldsymbol{E} \tag{1.1}$$

$$\frac{\partial \boldsymbol{D}}{\partial t} = \nabla \times \boldsymbol{H} - \boldsymbol{J}. \tag{1.2}$$

Where $\boldsymbol{E}$, $\boldsymbol{H}$, $\boldsymbol{D}$, $\boldsymbol{B}$, $\boldsymbol{J}$, and t are the electric field, the magnetic field, the electric flux density, the magnetic flux density, the conduction current density, and the time, respectively. In each time step, the approximated Maxwell's curl equations are used to first calculate the electric field $\boldsymbol{E}$ and then the magnetic field $\boldsymbol{H}$. The calculations of $\boldsymbol{E}$ and $\boldsymbol{H}$ are repeated throughout all time steps. Also, Yee has divided the FDTD space into small cells. The calculation of $\boldsymbol{E}$ and $\boldsymbol{H}$ depends on the relation between the neighbouring cells.

Considering linear, isotropic and nondispersive materials, the following relationships are true:

$$\boldsymbol{D} = \epsilon \boldsymbol{E} \tag{1.3}$$

$$\boldsymbol{B} = \mu \boldsymbol{H}. \tag{1.4}$$

Where ϵ and μ are the electrical permittivity and the magnetic permeability respectively. For the source-free medium where $\boldsymbol{J}$ is equal to $\mathbf{0}$, the use of equations (1.3) and (1.4) in (1.1) and (1.2) yields the following partial differential equations:

$$\frac{\partial H_x}{\partial t} = \frac{1}{\mu}\left[\frac{\partial E_y}{\partial z} - \frac{\partial E_z}{\partial y}\right] \tag{1.5}$$

$$\frac{\partial H_y}{\partial t} = \frac{1}{\mu}\left[\frac{\partial E_z}{\partial x} - \frac{\partial E_x}{\partial z}\right] \tag{1.6}$$

$$\frac{\partial H_z}{\partial t} = \frac{1}{\mu}\left[\frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x}\right] \tag{1.7}$$

$$\frac{\partial E_x}{\partial t} = \frac{1}{\epsilon}\left[\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z}\right] \tag{1.8}$$

$$\frac{\partial E_y}{\partial t} = \frac{1}{\epsilon}\left[\frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x}\right] \tag{1.9}$$

$$\frac{\partial E_z}{\partial t} = \frac{1}{\epsilon}\left[\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y}\right] \tag{1.10}$$

Applying central finite difference approximations for the time and space derivatives when the separation between $\boldsymbol{E}$ and $\boldsymbol{H}$ is equal to half-time step gives the

Yee FDTD equations:

$$\begin{aligned} H_x^{n+\frac{1}{2}}{}_{(i,j+\frac{1}{2},k+\frac{1}{2})} &= H_x^{n-\frac{1}{2}}{}_{(i,j+\frac{1}{2},k+\frac{1}{2})} \\ +\frac{\Delta t}{\mu(i,j+\frac{1}{2},k+\frac{1}{2})\Delta z}&\left[E_y^n{}_{(i,j+\frac{1}{2},k+1)} - E_y^n{}_{(i,j+\frac{1}{2},k)}\right] \\ +\frac{\Delta t}{\mu(i,j+\frac{1}{2},k+\frac{1}{2})\Delta y}&\left[E_z^n{}_{(i,j,k+\frac{1}{2})} - E_z^n{}_{(i,j+1,k+\frac{1}{2})}\right] \end{aligned} \tag{1.11}$$

$$\begin{aligned} H_y^{n+\frac{1}{2}}{}_{(i+\frac{1}{2},j,k+\frac{1}{2})} &= H_y^{n-\frac{1}{2}}{}_{(i+\frac{1}{2},j,k+\frac{1}{2})} \\ +\frac{\Delta t}{\mu(i+\frac{1}{2},j,k+\frac{1}{2})\Delta x}&\left[E_z^n{}_{(i+1,j,k+\frac{1}{2})} - E_z^n{}_{(i,j,k+\frac{1}{2})}\right] \\ +\frac{\Delta t}{\mu(i+\frac{1}{2},j,k+\frac{1}{2})\Delta z}&\left[E_x^n{}_{(i+\frac{1}{2},j,k)} - E_x^n{}_{(i+\frac{1}{2},j,k+1)}\right] \end{aligned} \tag{1.12}$$

$$\begin{aligned} H_z^{n+\frac{1}{2}}{}_{(i+\frac{1}{2},j+\frac{1}{2},k)} &= H_z^{n-\frac{1}{2}}{}_{(i+\frac{1}{2},j+\frac{1}{2},k)} \\ +\frac{\Delta t}{\mu(i+\frac{1}{2},j+\frac{1}{2},k)\Delta y}&\left[E_x^n{}_{(i+\frac{1}{2},j+1,k)} - E_x^n{}_{(i+\frac{1}{2},j,k)}\right] \\ +\frac{\Delta t}{\mu(i+\frac{1}{2},j+\frac{1}{2},k)\Delta x}&\left[E_y^n{}_{(i,j+\frac{1}{2},k)} - E_y^n{}_{(i+1,j+\frac{1}{2},k)}\right] \end{aligned} \tag{1.13}$$

$$\begin{aligned} E_x^{n+1}{}_{(i+\frac{1}{2},j,k)} &= E_x^n{}_{(i+\frac{1}{2},j,k)} \\ +\frac{\Delta t}{\epsilon(i+\frac{1}{2},j,k)\Delta y}&\left[H_z^{n+\frac{1}{2}}{}_{(i+\frac{1}{2},j+\frac{1}{2},k)} - H_z^{n+\frac{1}{2}}{}_{(i+\frac{1}{2},j-\frac{1}{2},k)}\right] \\ +\frac{\Delta t}{\epsilon(i+\frac{1}{2},j,k)\Delta z}&\left[H_y^{n+\frac{1}{2}}{}_{(i+\frac{1}{2},j,k-\frac{1}{2})} - H_y^{n+\frac{1}{2}}{}_{(i+\frac{1}{2},j,k+\frac{1}{2})}\right] \end{aligned} \tag{1.14}$$

$$\begin{aligned} E_y^{n+1}{}_{(i,j+\frac{1}{2},k)} &= E_y^n{}_{(i,j+\frac{1}{2},k)} \\ +\frac{\Delta t}{\epsilon(i,j+\frac{1}{2},k)\Delta z}&\left[H_x^{n+\frac{1}{2}}{}_{(i,j+\frac{1}{2},k+\frac{1}{2})} - H_x^{n+\frac{1}{2}}{}_{(i,j+\frac{1}{2},k-\frac{1}{2})}\right] \\ +\frac{\Delta t}{\epsilon(i,j+\frac{1}{2},k)\Delta x}&\left[H_z^{n+\frac{1}{2}}{}_{(i-\frac{1}{2},j+\frac{1}{2},k)} - H_z^{n+\frac{1}{2}}{}_{(i+\frac{1}{2},j+\frac{1}{2},k)}\right] \end{aligned} \tag{1.15}$$

$$\begin{aligned} E_z^{n+1}{}_{(i,j,k+\frac{1}{2})} &= E_z^{n}{}_{(i,j,k+\frac{1}{2})} \\ &+ \frac{\Delta t}{\epsilon(i,j,k+\frac{1}{2})\Delta x}\left[H_y^{n+\frac{1}{2}}{}_{(i+\frac{1}{2},j,k+\frac{1}{2})} - H_y^{n+\frac{1}{2}}{}_{(i-\frac{1}{2},j,k+\frac{1}{2})}\right] \\ &+ \frac{\Delta t}{\epsilon(i,j,k+\frac{1}{2})\Delta y}\left[H_x^{n+\frac{1}{2}}{}_{(i,j-\frac{1}{2},k+\frac{1}{2})} - H_x^{n+\frac{1}{2}}{}_{(i,j+\frac{1}{2},k+\frac{1}{2})}\right] \end{aligned} \tag{1.16}$$

where, Δt, Δx, Δy and Δz are the temporal discretisation, spatial discretisation in x, spatial discretisation in y, and spatial discretisation in z, respectively.

Yee FDTD equations contain non-integer coordinates which cant be implemented in the computational systems. Thus, the equations are modified so that the non-integer coordinates are transformed to integers:

$$\begin{aligned} H_x^{n+1}{}_{(i,j,k)} = H_x^{n}{}_{(i,j,k)} &+ \frac{\Delta t}{\mu(i,j,k)\Delta z}\left[E_y^{n}{}_{(i,j,k+1)} - E_y^{n}{}_{(i,j,k)}\right] \\ &- \frac{\Delta t}{\mu(i,j,k)\Delta y}\left[E_z^{n}{}_{(i,j+1,k)} - E_z^{n}{}_{(i,j,k)}\right] \end{aligned} \tag{1.17}$$

$$\begin{aligned} H_y^{n+1}{}_{(i,j,k)} = H_y^{n}{}_{(i,j,k)} &+ \frac{\Delta t}{\mu(i,j,k)\Delta x}\left[E_z^{n}{}_{(i+1,j,k)} - E_z^{n}{}_{(i,j,k)}\right] \\ &- \frac{\Delta t}{\mu(i,j,k)\Delta z}\left[E_x^{n}{}_{(i,j,k+1)} - E_x^{n}{}_{(i,j,k)}\right] \end{aligned} \tag{1.18}$$

$$\begin{aligned} H_z^{n+1}{}_{(i,j,k)} = H_z^{n}{}_{(i,j,k)} &+ \frac{\Delta t}{\mu(i,j,k)\Delta y}\left[E_x^{n}{}_{(i,j+1,k)} - E_x^{n}{}_{(i,j,k)}\right] \\ &- \frac{\Delta t}{\mu(i,j,k)\Delta x}\left[E_y^{n}{}_{(i+1,j,k)} - E_y^{n}{}_{(i,j,k)}\right] \end{aligned} \tag{1.19}$$

$$\begin{aligned} E_x^{n+1}{}_{(i,j,k)} = E_x^{n}{}_{(i,j,k)} &+ \frac{\Delta t}{\epsilon(i,j,k)\Delta y}\left[H_z^{n}{}_{(i,j,k)} - H_z^{n}{}_{(i,j-1,k)}\right] \\ &- \frac{\Delta t}{\epsilon(i,j,k)\Delta z}\left[H_y^{n}{}_{(i,j,k)} - H_y^{n}{}_{(i,j,k-1)}\right] \end{aligned} \tag{1.20}$$

$$\begin{aligned} E_y^{n+1}{}_{(i,j,k)} = E_y^{n}{}_{(i,j,k)} &+ \frac{\Delta t}{\epsilon(i,j,k)\Delta z}\left[H_x^{n}{}_{(i,j,k)} - H_x^{n}{}_{(i,j,k-1)}\right] \\ &- \frac{\Delta t}{\epsilon(i,j,k)\Delta x}\left[H_z^{n}{}_{(i,j,k)} - H_z^{n}{}_{(i-1,j,k)}\right] \end{aligned} \tag{1.21}$$

$$\begin{aligned} E_z^{n+1}{}_{(i,j,k)} = E_z^{n}{}_{(i,j,k)} &+ \frac{\Delta t}{\epsilon(i,j,k)\Delta x}\left[H_y^n{}_{(i,j,k)} - H_y^n{}_{(i-1,j,k)}\right] \\ &- \frac{\Delta t}{\epsilon(i,j,k)\Delta y}\left[H_x^n{}_{(i,j,k)} - H_x^n{}_{(i,j-1,k)}\right] \end{aligned} \tag{1.22}$$

(1.17), (1.18), (1.19), (1.20), (1.21) and (1.22) are the equations of the FDTD method.

1.1.2 Frequency Dependent Media

The media parameters in the FDTD equations (1.17), (1.18), (1.19), (1.20), (1.21) and (1.22) are frequency-independent. However, in practical simulations such as the electromagnetic wave propagation in the human body, the media parameters depend on the frequency. There are three main models to represent frequency dispersive materials: Debye model, Lorentz-Drude model and Cole-Cole model. Debye model is simple in implementation so that it is the most widely used model for the FDTD method. In this thesis, the human tissues are assumed as Debye media where the relative permittivity reads:

$$\epsilon_r = \epsilon_\infty + \frac{\epsilon_\mathrm{S} - \epsilon_\infty}{1 + \jmath\omega\tau_\mathrm{D}} - \jmath\frac{\sigma}{\omega\epsilon_0}. \tag{1.23}$$

where ϵ_S, ϵ_∞, τ_D and σ are the static relative permittivity, the relative permittivity at infinite frequency, the relaxation time and the conductivity, respectively.

In the Debye media, the FDTD method first solve the Maxwell equations for $\boldsymbol{D}$ and $\boldsymbol{H}$ fields and then obtain the $\boldsymbol{E}$ field from $\boldsymbol{D}$ using Equation 1.23.

1.1.3 Features of the FDTD Method

There are several numerical Computational Electromagnetics (CEM) methods such as Method of Moments (MoM) [14], Transmission Line Matrix (TLM) [15] method, Finite Element Method (FEM) [16] and FDTD method. The FEM, TLM and FDTD methods are based on differential equations while MoM is an integral method. The integral methods are accurate but the capability of those methods become difficult when dealing with complex materials. The FDTD and TLM methods solve Maxwell's equations in time domain. On the other hand, FEM and MoM are used in frequency domain. Frequency domain methods give solutions for a specific frequency and the simulations has to be repeated to provide solutions for a range of frequencies. Thus, for wide-band applications, the time domain methods such as the FDTD and TLM are used. The FDTD and TLM methods are similar, but the FDTD method requires less storage space than the TLM method.

1.1.3.1 Strengths of The FDTD Method

Some of the many strengths of the FDTD method are:

- The FDTD method is relatively simple, straightforward and easy to implement on computers [17].
- The FDTD method is accurate and robust [2].
- The FDTD method has the capability to specify any material at all the grids of the FDTD space.
- The FDTD method is suitable for parallel computing which is useful (faster) for practical applications.

1.1.3.2 Weaknesses of The FDTD Method

The FDTD is the most commonly used method in electromagnetic modeling [2]. However, it has some weaknesses such as:

- The FDTD method cannot accurately model the curves unless the spatial sampling is extremely small. That is because the FDTD method has rectangular grids.
- Memory requirements. The FDTD computations require large amount of data since $\boldsymbol{E}$ and $\boldsymbol{H}$ are stored for each grid point in the FDTD space.

One of the ways to reduce the memory requirement is the application of a good absorbing boundary condition so that the distance between the objects in the FDTD space and the artificial boundary can be minimized. This thesis aims for the development of such a strong and efficient absorbing boundary condition.

Chapter 2

Absorbing Boundary Conditions

The FDTD domains are usually open regions. Since no computer can store infinite data, the FDTD domain needs an Absorbing Boundary Condition (ABC) [2]. When the FDTD domain is bounded, the ABC is used to simulate the extension to infinity. In practical problems, the ABC surrounds the FDTD domain, which contains the structure of interest. In order to have acceptable results, the ABC should absorb the reflection of the outgoing waves to an acceptable level [2].

The use of the ABC is essential in the FDTD practical simulations. That is why different implementations of the ABCs have been introduced. Mur's ABC [18] is one of the oldest and popular ABCs. It is a simple and successful ABC [2]. The implementation of the Mur's ABC is straightforward and requires (relatively) less computational resources. But on the other hand, the accuracy of the Mur's ABC's solutions can be improved [2].

During 1970s and 1980s, an ABC implementation was originated by Higdon [19][20][2]. Higdon's technique terminates the outgoing waves using a series of linear differential operators. Those operators are designed to absorb the wave that is propagating in a specific direction. If the order of the Higdon's operators is n then the waves are (almost) perfectly absorbed at n incident angles [21]. Higher orders Higdon's ABCs are more complex in implementation [21].

In 1994 Jean-Pierre Bérenger introduced the perfectly matched layer (PML) [22] ABC. The PML terminates the FDTD space with an absorbing material medium that has a thickness of few cells [2]. The PML ABC has the capability to absorb the plane waves that have different angles and frequencies. All the waves are matched at the boundary, which makes the PML a highly effective ABC [2]. The success of Bérenger's PML led to propose different versions such as Un-split PML (UPML) [23], stretched coordinate PML [24] [25], and time convolution PML [26][2].

2.1 ABC Based on One-Way Wave Equation

This section briefly introduces Mur's first-order ABC, which is based on the one-way wave equation.

2.1.1 Numerical formulation from one-way wave equation

The one-way wave equation is a partial-differential equation that permits the wave to propagate in a certain direction [2]. It can be applied at the boundary of the FDTD space to numerically absorb impinging outgoing waves [2].

In [11], Maxwell's equations for an isotropic non dispersive homogeneous lossless linear media (1.1) was transformed as follows:

$$\nabla \times \nabla \times \boldsymbol{E} = -\nabla \times \frac{\partial \boldsymbol{B}}{\partial t}$$

using (1.4) yields:

$$\begin{aligned}\nabla \times \nabla \times \boldsymbol{E} &= -\nabla \times \frac{\partial \mu \boldsymbol{H}}{\partial t}\\ &= -\mu \frac{\partial \nabla \times \boldsymbol{H}}{\partial t}\end{aligned}$$

using (1.2):

$$\begin{aligned}\nabla \times \nabla \times \boldsymbol{E} &= -\mu \frac{\partial \frac{\partial \boldsymbol{D}}{\partial t}}{\partial t}\\ &= -\mu \frac{\partial^2 \boldsymbol{D}}{\partial t^2}\end{aligned}$$

substituting by (1.3) gives:

$$\nabla \times \nabla \times \boldsymbol{E} = -\mu\epsilon \frac{\partial^2 \boldsymbol{E}}{\partial t^2}. \tag{2.1}$$

Using the identity:

$$\nabla \times \nabla \times \boldsymbol{E} = \nabla(\nabla \bullet \boldsymbol{E}) - (\nabla \bullet \nabla)\boldsymbol{E}$$

and assuming that the boundary is source-free ($\nabla \bullet \boldsymbol{E} = 0$), (2.1) can be re-written as [27]:

$$\begin{aligned}
-(\nabla \bullet \nabla)\boldsymbol{E} + \mu\epsilon\frac{\partial^2 \boldsymbol{E}}{\partial t^2} &= 0 \\
(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} - \mu\epsilon\frac{\partial^2}{\partial t^2})\boldsymbol{E} &= 0 \\
(\frac{\partial^2}{\partial x^2} - \mu\epsilon\frac{\partial^2}{\partial t^2}(1 - \frac{\frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}}{\mu\epsilon\frac{\partial^2}{\partial t^2}}))\boldsymbol{E} &= 0. \qquad (2.2)
\end{aligned}$$

2.1.2 Mur's ABC

Mur's ABC applies the central differencing on the pseudo-operator (2.2) in time and spatial domains [2]. This approximation gives:

$$\frac{1}{\Delta x}\left[E_{x=1}^{n+\frac{1}{2}} - E_{x=0}^{n+\frac{1}{2}}\right] = \frac{\sqrt{\mu\epsilon}}{\Delta t}\left[E_{x=\frac{1}{2}}^{n+1} - E_{x=\frac{1}{2}}^{n}\right]. \qquad (2.3)$$

(2.3) is second-order accurate [11]. However, equation (2.3) does not include the field at the boundary. Also, the field at time $(n+1)\Delta t$ is missing. Thus, Mur represented $E_{x=0}^{(n+\frac{1}{2})}$ and $E_{x=\frac{1}{2}}^{n}$ by calculating the average of the two adjacent fields in time and spacial domains [11]. This yields:

$$E_{x=0}^{n+\frac{1}{2}} = \frac{1}{2}\left[E_{x=0}^{n+1} + E_{x=0}^{n}\right] \qquad (2.4)$$

$$E_{x=\frac{1}{2}}^{n} = \frac{1}{2}\left[E_{x=1}^{n} + E_{x=0}^{n}\right] \qquad (2.5)$$

Using (2.4) and (2.5) in equation (2.3) yields:

$$E_{x=0}^{n+1} = E_{x=1}^{n} + \frac{\Delta t - \Delta x\sqrt{\mu\epsilon}}{\Delta t + \Delta x\sqrt{\mu\epsilon}}\left[E_{x=1}^{n+1} - E_{x=0}^{n}\right]. \qquad (2.6)$$

Similarly, using the same approach at the other side of the boundary obtains:

$$E_{x=x_{max-1}}^{n+1} = E_{x=x_{max}}^{n} + \frac{\Delta t - \Delta x\sqrt{\mu\epsilon}}{\Delta t + \Delta x\sqrt{\mu\epsilon}}\left[E_{x=x_{max}}^{n+1} - E_{x=x_{max-1}}^{n}\right] \qquad (2.7)$$

(2.6) and (2.7) are Mur's first-order boundary conditions.

The same approach is used to obtain Mur's boundary condition for the 3D-FDTD space [11].

2.2 Complex Frequency-Shifted Perfectly Matched Layer ABCs

Kuzuoglu and Mittra introduced the CFS-PML [28]. The stretching coefficient of the normal PML is replaced with:

$$s_x = 1 + \frac{\sigma_x}{\alpha_x + j\omega\epsilon_0} \tag{2.8}$$

where σ_x is the PML conductivity and α_x is an additional parameter homogeneous to a conductivity. In [26], Roden and Gedney used the following general form of 2.8 to improve the absorption capabilities of the CFS-PML:

$$s_x = \kappa_x + \frac{\sigma_x}{\alpha_x + j\omega\epsilon_0} \tag{2.9}$$

where κ_x is real and ≥ 1.
As its name suggests, the CFS-PML almost perfectly absorb the waves, and that makes it the most famous and effective ABCs nowadays [2]. However, the implementation of the CFS-PML is complicated and needs more computational resources such as memory and CPU time. This project handles a new boundary condition, called the Huygens absorbing boundary condition (HABC), which is proved to be comparable to the CFS-PML in accuracy and to be simpler than the CFS-PML in implementation. The HABC is introduced in the Chapter 3. The numerical results of the HABC experiments in this thesis are compared to the CFS-PML results for accuracy and computational requirement.

Chapter 3

The Huygens Absorbing Boundary Condition

The Huygens principle states that each point on a wave front acts like a new source of waves [29]. In electromagnetics, the equivalence theorem uses the Huygens principle to precisely define those equivalent source current densities [29]. Based on the equivalence theorem, Section 3.1 demonstrates the idea of defining the source current densities on a specific surface which is used in this chapter to implement an incident wave and an Absorbing Boundary Condition (ABC).

3.1 The Huygens Surface

According to the equivalence theorem in electromagnetics, the field obtained inside a given part of space by sources placed outside this part can be re-obtained by introducing the electric and magnetic current densities along the surface splitting these two parts as shown in the equations below:

$$\vec{\boldsymbol{J}}_s = \vec{\boldsymbol{u}} \times \vec{\boldsymbol{H}}_i \tag{3.1}$$

$$\vec{\boldsymbol{K}}_s = \vec{\boldsymbol{u}} \times \vec{\boldsymbol{E}}_i \tag{3.2}$$

where:

- $\boldsymbol{u}$ is the unit vector normal to the surface and opposite to the source direction.

- $\boldsymbol{E}_i$ is the electric field that would exists upon the surface if the source were present.
- $\boldsymbol{H}_i$ is the magnetic field that would exists upon the surface if the source were present.
- $\boldsymbol{J}_s$ is the electric current density.
- $\boldsymbol{K}_s$ is the magnetic current density.

This surface, that is used to produce the equivalent currents to only source-free part, is called the Huygens surface [29][3] [2].

3.2 The Huygens Surface for Implementing an Incident Wave

In wave-structure interaction problems, the incident plane wave strikes an object. As shown in Figure 3.1 ,the incident wave extends upon a certain length L in space, from its front to its back with duration of L/c in time at a given point where c is the speed of light [1].

The wave has to be set in a finite computational domain. Setting the wave as an initial condition is a simple way to introduce a wave in the FDTD domain. This is done by setting the wave at time 0 before it strikes the object [1]. This produces correct results for a certain time. However, the cut of the plane wave on the boundary generates spurious field. This field propagates toward the object and when it reaches the object the results become wrong [1]. Since the domain in computational electromagnetics should be as small as possible, the time in which the results are correct is short. This issue can be solved by using the Huygens surface to generate the wave [1].

The Huygens surface is used to create incident wave within the surface while no field exists outside the surface as shown in Figure 3.2 [1]. Consider a source of field outside a closed surface as illustrated in Figure 3.3. The use of the equivalence theorem removes the source and enforces the sources 3.1 and 3.2 on the closed surface, where $\boldsymbol{E}_i$ and $\boldsymbol{H}_i$ are equal to the incident field on the surface or in other words the field if the sources were present [1]. As a result, the field within the closed surface is equal to the field created by the original source while no field exists outside the closed surface [1].

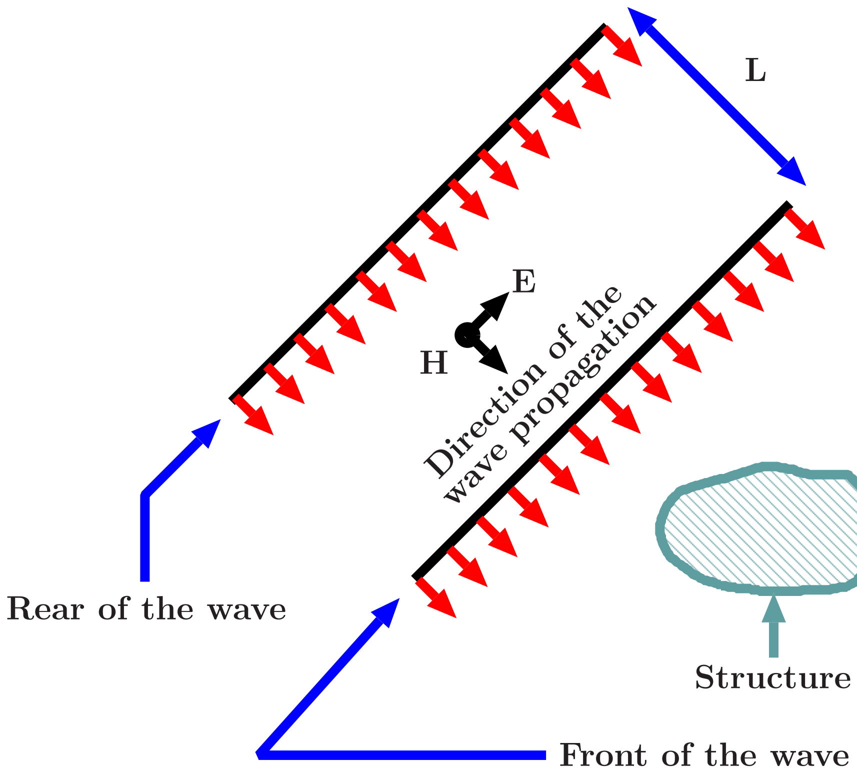


Figure 3.1: An incident wave extends upon a certain length L in space, from its front to its back with duration of L/c in time at a given point [1].

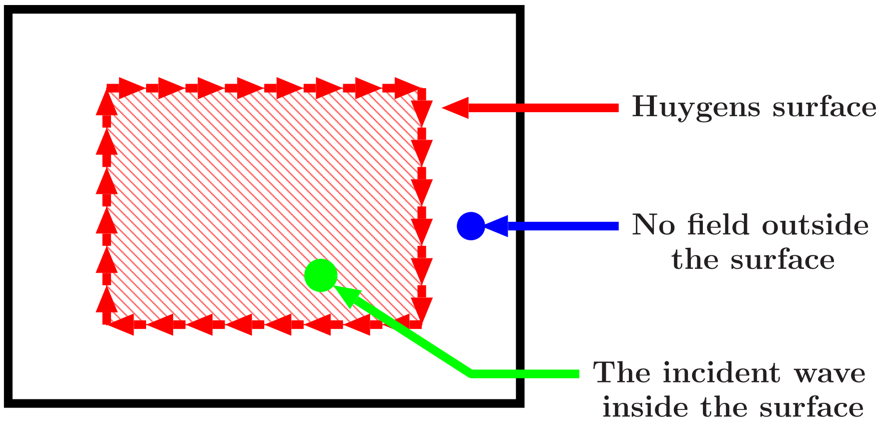

Huygens surface
No field outside the surface
The incident wave inside the surface

Figure 3.2: The use of Huygens surface to generate the wave [1].

The currents $\boldsymbol{J}_s$ and $\boldsymbol{K}_s$ are sheets zero in thickness and expressed in A/m and V/m respectively. Those currents are parallel to the Huygens surface. The magnitudes of $\boldsymbol{J}_s$ and $\boldsymbol{K}_s$ are equal to the components $\boldsymbol{E}$ and $\boldsymbol{H}$ parallel to the Huygens surface [1]. As shown in Figure 3.4, in the FDTD domain, the zero

Source
(antenna)
Closed surface
with no source
inside it

Figure 3.3: A source of field exterior to a closed surface [1].

thickness current is spread upon a width of one FDTD cell Δx:

$$\boldsymbol{J}_y = \frac{\boldsymbol{J}_{Sy}}{\Delta x} = -\frac{\boldsymbol{H}_{y,inc}}{\Delta x}. \tag{3.3}$$

Where $\boldsymbol{J}_{Sy}$ and $\boldsymbol{H}_{y,inc}$ are the y components of $\boldsymbol{J}_s$ and $\boldsymbol{H}_{inc}$ respectively. Thus, the FDTD current $\boldsymbol{J}_y\Delta x$ is equal to the desired current $\boldsymbol{J}_{Sy}$ [1].

3.2.1 The 1D case

Figure 3.5 shows the 1D FDTD grid with the Huygens surface. Introducing the sources in the 1D FDTD space:

$$\frac{\partial E_y}{\partial t} = \frac{1}{\epsilon}\frac{\partial H_z}{\partial x} - J, \tag{3.4}$$

$$\begin{aligned} E_y^{n+1}(IS) &= E_y^n(IS) \\ -\frac{\Delta t}{\epsilon}\frac{H_z^{n+\frac{1}{2}}(IS+\frac{1}{2}) - H_z^{n+\frac{1}{2}}(IS-\frac{1}{2})}{\Delta x} &- (-1)\frac{\Delta t}{\epsilon}\frac{J_z^{n+\frac{1}{2}}}{\Delta x}. \end{aligned} \tag{3.5}$$

Choosing $J_{Sz}^{n+\frac{1}{2}}$ at $(IS+\frac{1}{4})$:

$$\begin{aligned} E_y^{n+1}(IS) &= E_y^n(IS) \\ -\frac{\Delta t}{\epsilon}\frac{\left[H_z^{n+\frac{1}{2}}(IS+\frac{1}{2}) - H_{z,inc}^{n+\frac{1}{2}}(IS-\frac{1}{4})\right] - H_z^{n+\frac{1}{2}}(IS-\frac{1}{2})}{\Delta x}&. \end{aligned} \tag{3.6}$$

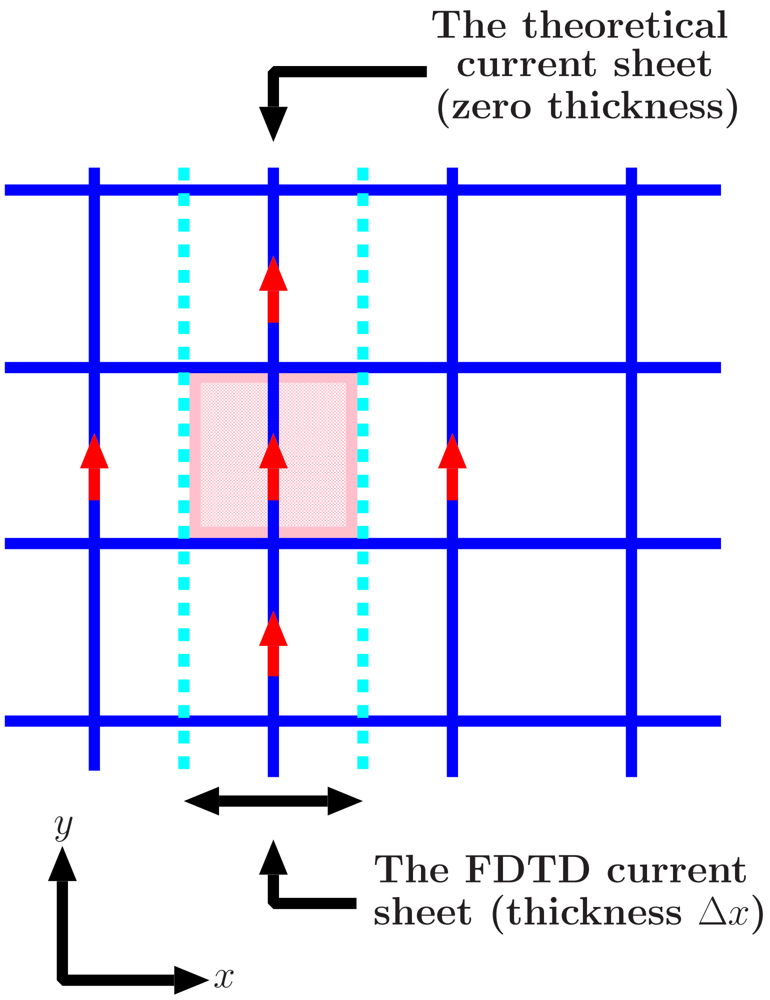

The theoretical current sheet (zero thickness)
The FDTD current sheet (thickness Δx)
y
x

Figure 3.4: The zero thickness current is spread upon a width of one FDTD cell Δx in the FDTD domain [1].

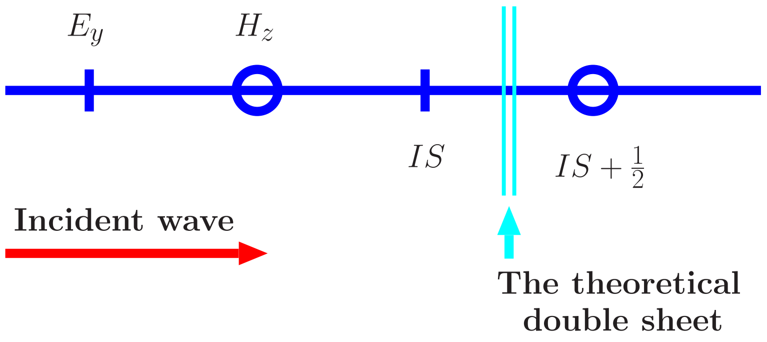


Figure 3.5: Introducing sources using the Huygens surface in 1D FDTD space [1].

Similarly:

$$H_z^{n+\frac{1}{2}}(IS+\frac{1}{2}) = H_z^{n-\frac{1}{2}}(IS+\frac{1}{2}) \\ -\frac{\Delta t}{\mu}\frac{E_y^n(IS+1)-\left[E_y^n(IS)+E_{y,inc}^n(IS+\frac{1}{4})\right]}{\Delta x} \tag{3.7}$$

Those equations are the same as the old equations, but a correction is added at one node. The corrected equations are also applied on the other side of the Huygens surface with just an opposite sign for the corrective terms. The sign is reversed because the unit vector normal to the other side of the surface is oriented in opposite direction [1].

3.2.2 The Trivial Implementation of the Huygens Surface

The incident wave can be implemented in a small closed area of space without the equivalence theorem. For the FDTD method, this implementation is rather trivial [1]. In Figure 3.5, assume at time n the field is equal to zero to the left of the Huygens surface, and equal to the incident field to right of the Huygens surface. $E_y(IS)$ is computed as:

$$E_y^{n+1}(IS) = E_y^n(IS) - \frac{\Delta t}{\epsilon}\frac{H_z^{n+\frac{1}{2}}(IS+\frac{1}{2}) - H_z^{n+\frac{1}{2}}(IS-\frac{1}{2})}{\Delta x} \tag{3.8}$$

However, Equation 3.8 is not correct because $H_z(IS+\frac{1}{2})$ should be zero to let $E_y(IS)$ remain zero. This issue can be solved by simply subtracting the error from the FDTD value $H_z(IS+\frac{1}{2})$. The error is the incident field at this node, which is known by hypothesis [1]. This yield:

$$E_y^{n+1}(IS) = E_y^n(IS) \\ -\frac{\Delta t}{\epsilon}\frac{\left[H_z^{n+\frac{1}{2}}(IS+\frac{1}{2}) - H_{z,inc}^{n+\frac{1}{2}}(IS+\frac{1}{2})\right] - H_z^{n+\frac{1}{2}}(IS-\frac{1}{2})}{\Delta x}. \tag{3.9}$$

For the H_z at node $IS+\frac{1}{2}$, it is not correct to use the regular formula because $E_z(IS)$ is equal to zero. $E_z(IS)$ should be equal to the incident field to obtain a correct H_z. Thus, the incident field should be added to $E_z(IS)$ at this node.

This yield:

$$H_z^{n+\frac{1}{2}}(IS+\frac{1}{2}) = H_z^{n-\frac{1}{2}}(IS+\frac{1}{2}) - \frac{\Delta t}{\mu}\frac{E_y^n(IS+1) - \left[E_y^n(IS) + E_{y,inc}^n(IS)\right]}{\Delta x}. \tag{3.10}$$

The only difference between the Equation 3.6 from the equivalence theorem and the Equation 3.9 from the trivial reasoning is the location of the incident field. The location of the incident field is $IS + \frac{1}{4}$ in case of the equivalence theorem and $IS + \frac{1}{2}$ in the trivial reasoning. The location $IS + \frac{1}{2}$ is better since it is consistent with the FDTD discretisation [1]. Although the difference between the two incident field locations is small, it is visible in actual implementation [1]. Similarly, for H_Z, the equations yield by the equivalence theorem and the trivial reasoning only differ in the locations of the incident E_Z. The location of the incident E_Z is at $IS + \frac{1}{4}$ in equivalence theorem and at IS, which is better and consistent with the FDTD discretisation, in trivial reasoning [1].

3.2.3 Huygens Surface with an Object

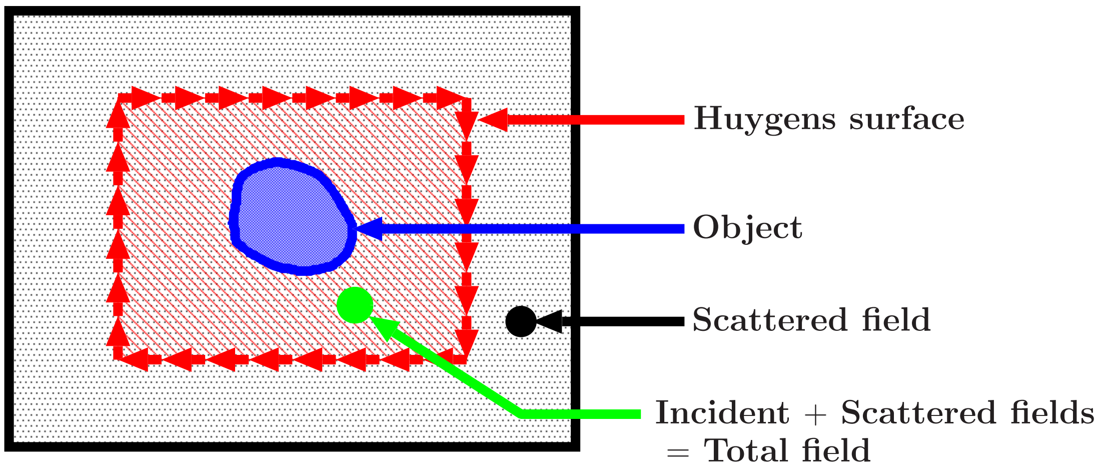


Figure 3.6: The use of Huygens surface to generate the wave when an object is present [1].

When the FDTD domain contains an object as shown in Figure 3.6, a scattered field is created. The scattered field is defined as the difference between the actual (total) field and the field in the absence of object (the incident field) [1]. Outside the Huygens surface, there is only the scattered field while the total field exists inside the Huygens surface. In other words, The Huygens surface separates the total field region and the scattered field region [1].

3.2.4 Huygens Surface in 3D

6 Huygens surfaces are used to excite the 3D FDTD space. These surfaces are placed at:

- $i = ihuyg1$, j from $j = jhuyg1$ to $j = jhuyg2$ and k from $k = khuyg1$ to $k = khuyg2$.
- $i = ihuyg2$, j from $j = jhuyg1$ to $j = jhuyg2$ and k from $k = khuyg1$ to $k = khuyg2$.
- i from $i = ihuyg1$ to $i = ihuyg2$, $j = jhuyg1$ and k from $k = khuyg1$ to $k = khuyg2$.
- i from $i = ihuyg1$ to $i = ihuyg2$, $j = jhuyg2$ and k from $k = khuyg1$ to $k = khuyg2$.
- i from $i = ihuyg1$ to $i = ihuyg2$, j from $j = jhuyg1$ to $j = jhuyg2$ and $k = khuyg1$.
- i from $i = ihuyg1$ to $i = ihuyg2$, j from $j = jhuyg1$ to $j = jhuyg2$ and $k = khuyg2$.

Then following equations [30] are applied on the Huygens surfaces:

$$E_x^{n+1}(i,j,k) = E_x^n(i,j,k) + \frac{\Delta t}{\epsilon(i,j,k)} \times \tag{3.11}$$

$$\left[\frac{H_z^{n+1}(i,j,k) + \widetilde{H}_z^{\tilde{n}+1}(i,j,k) - \left(H_z^{n+1}(i,j-1,k) + \widetilde{H}_z^{\tilde{n}+1}(i,j-1,k)\right)}{\frac{\Delta y_{(j-1)} + \Delta y_{(j)}}{2}}\right.$$

$$\left.-\frac{H_y^{n+1}(i,j,k) + \widetilde{H}_y^{\tilde{n}+1}(i,j,k) - \left(H_y^{n+1}(i,j,k-1) + \widetilde{H}_y^{\tilde{n}+1}(i,j,k-1)\right)}{\Delta z_{(k)}}\right]$$

$$[ihuyg1 \le i \le ihuyg2 - 1$$

$$j = jhuyg1, j = jhuyg2$$

$$k = khuyg1, k = khuyg2].$$

$$\begin{aligned} E_y^{n+1}{}_{(i,j,k)} &= E_y^{n}{}_{(i,j,k)} + \frac{\Delta t}{\epsilon(i,j,k)} \times \\ &\Bigg[\frac{H_x^{n+1}{}_{(i,j,k)} + \widetilde{H}_x^{\tilde{n}+1}{}_{(i,j,k)} - \left(H_x^{n+1}{}_{(i,j,k-1)} + \widetilde{H}_x^{\tilde{n}+1}{}_{(i,j,k-1)}\right)}{\Delta z_{(k)}} \\ &- \frac{H_z^{n+1}{}_{(i,j,k)} + \widetilde{H}_z^{\tilde{n}+1}{}_{(i,j,k)} - \left(H_z^{n+1}{}_{(i-1,j,k)} + \widetilde{H}_z^{\tilde{n}+1}{}_{(i-1,j,k)}\right)}{\frac{\Delta x_{(i-1)}+\Delta x_{(i)}}{2}}\Bigg] \\ &[i = ihuyg1, i = ihuyg2 \\ &jhuyg1 \leq j \leq jhuyg2 - 1 \\ &k = khuyg1, k = khuyg2]. \end{aligned} \tag{3.12}$$

$$\begin{aligned} E_z^{n+1}{}_{(i,j,k)} &= E_z^{n}{}_{(i,j,k)} + \frac{\Delta t}{\epsilon(i,j,k)} \times \\ &+\frac{\Delta t}{\epsilon(i,j,k)}\Bigg[\frac{H_y^{n+1}{}_{(i,j,k)} + \widetilde{H}_y^{\tilde{n}+1}{}_{(i,j,k)} - \left(H_y^{n+1}{}_{(i-1,j,k)} + \widetilde{H}_y^{\tilde{n}+1}{}_{(i-1,j,k)}\right)}{\frac{\Delta x_{(i-1)}+\Delta x_{(i)}}{2}} \\ &- \frac{H_x^{n+1}{}_{(i,j,k)} + \widetilde{H}_x^{\tilde{n}+1}{}_{(i,j,k)} - \left(H_x^{n+1}{}_{(i,j-1,k)} + \widetilde{H}_x^{\tilde{n}+1}{}_{(i,j-1,k)}\right)}{\frac{\Delta y_{(j-1)}+\Delta y_{(j)}}{2}}\Bigg] \\ &[i = ihuyg1, i = ihuyg2 \\ &j = jhuyg1, j = jhuyg2 \\ &khuyg1 \leq k \leq khuyg2 - 1]. \end{aligned} \tag{3.13}$$

$$\begin{aligned} H_x^{n+1}{}_{(i,j,k)} &= H_x^{n}{}_{(i,j,k)} \\ &+\frac{\Delta t}{\mu}\Bigg[\frac{E_y^{n}{}_{(i,j,k+1)} - \widetilde{E}_y^{\tilde{n}}{}_{(i,j,k+1)} - \left(E_y^{n}{}_{(i,j,k)} - \widetilde{E}_y^{\tilde{n}}{}_{(i,j,k)}\right)}{\frac{\Delta z_{(k-1)}+\Delta z_{(k)}}{2}} \\ &- \frac{E_z^{n}{}_{(i,j+1,k)} - \widetilde{E}_z^{\tilde{n}}{}_{(i,j+1,k)} - \left(E_z^{n}{}_{(i,j,k)} - \widetilde{E}_z^{\tilde{n}}{}_{(i,j,k)}\right)}{\Delta y_{(j)}}\Bigg] \\ &[ihuyg1 \leq i \leq ihuyg2 \\ &j = jhuyg1, j = jhuyg2 - 1 \\ &k = khuyg1, k = khuyg2 - 1]. \end{aligned} \tag{3.14}$$

$$
\begin{aligned}
H_y^{n+1}{}_{(i,j,k)} &= H_y^n{}_{(i,j,k)} \\
&+\frac{\Delta t}{\mu}\Bigg[\frac{E_z^n{}_{(i+1,j,k)} - \widetilde{E_z^{\tilde{n}}}{}_{(i+1,j,k)} - \big(E_z^n{}_{(i,j,k)} - \widetilde{E_z^{\tilde{n}}}{}_{(i,j,k)}\big)}{\Delta x_{(i)}} \\
&-\frac{E_x^n{}_{(i,j,k+1)} - \widetilde{E_x^{\tilde{n}}}{}_{(i,j,k+1)} - \big(E_x^n{}_{(i,j,k)} - \widetilde{E_x^{\tilde{n}}}{}_{(i,j,k)}\big)}{\frac{\Delta z_{(k-1)}+\Delta z_{(k)}}{2}}\Bigg] \\
&[i = ihuyg1, i = ihuyg2-1 \\
&jhuyg1 \le j \le jhuyg2 \\
&k = khuyg1, k = khuyg2-1].
\end{aligned}
\tag{3.15}
$$

$$
\begin{aligned}
H_z^{n+1}{}_{(i,j,k)} &= H_z^n{}_{(i,j,k)} \\
&+\frac{\Delta t}{\mu}\Bigg[\frac{E_x^n{}_{(i,j+1,k)} - \widetilde{E_x^{\tilde{n}}}{}_{(i,j+1,k)} - \big(E_x^n{}_{(i,j,k)} - \widetilde{E_x^{\tilde{n}}}{}_{(i,j,k)}\big)}{\Delta y_{(j)}} \\
&-\frac{E_y^n{}_{(i+1,j,k)} - \widetilde{E_y^{\tilde{n}}}{}_{(i+1,j,k)} - \big(E_y^n{}_{(i,j,k)} - \widetilde{E_y^{\tilde{n}}}{}_{(i,j,k)}\big)}{\Delta x_{(i)}}\Bigg] \\
&[i = ihuyg1, i = ihuyg2-1 \\
&j = jhuyg1, j = jhuyg2-1 \\
&khuyg1 \le k \le kuyg2].
\end{aligned}
\tag{3.16}
$$

The values for $\widetilde{E^{\tilde{n}}}{}_{(i,j,k)}$ and $\widetilde{H^{\tilde{n}+1}}{}_{(i,j,k)}$ are as follows when J is set to the incident wave:

$$\widetilde{E^{\tilde{n}}}{}_{(i,j,k)} = E^{n-1}{}_{(i,j,k)} \pm J. \tag{3.17}$$

$$\widetilde{H^{\tilde{n}+1}}{}_{(i,j,k)} = H^{n}{}_{(i,j,k)} \pm J. \tag{3.18}$$

3.2.5 The Direction of the Incident wave

The direction of the incident wave generated by the Huygens surfaces is determined by the three angles: ϕ,θ and ψ. As shown in Figure 3.7, ϕ is the angle from the x axis, θ is the angle from the xy plane and ψ is the angle of the field itself. For instance, if the direction of the wave generated by Huygens surface

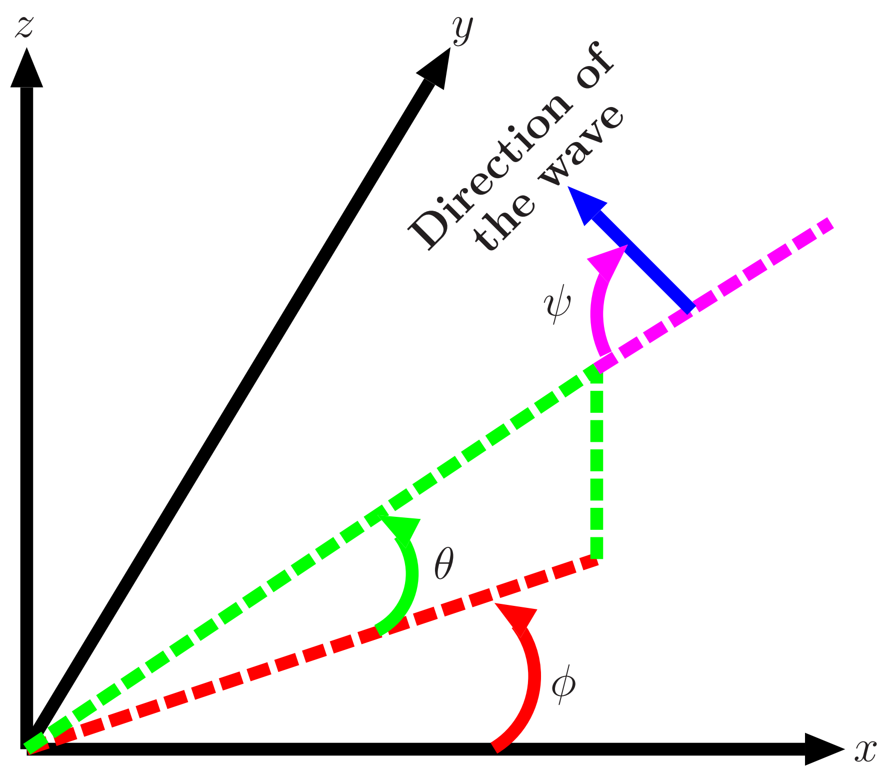


Figure 3.7: Setting the direction of the wave generated by the Huygens surface with the following three angles: ϕ,θ and ψ.

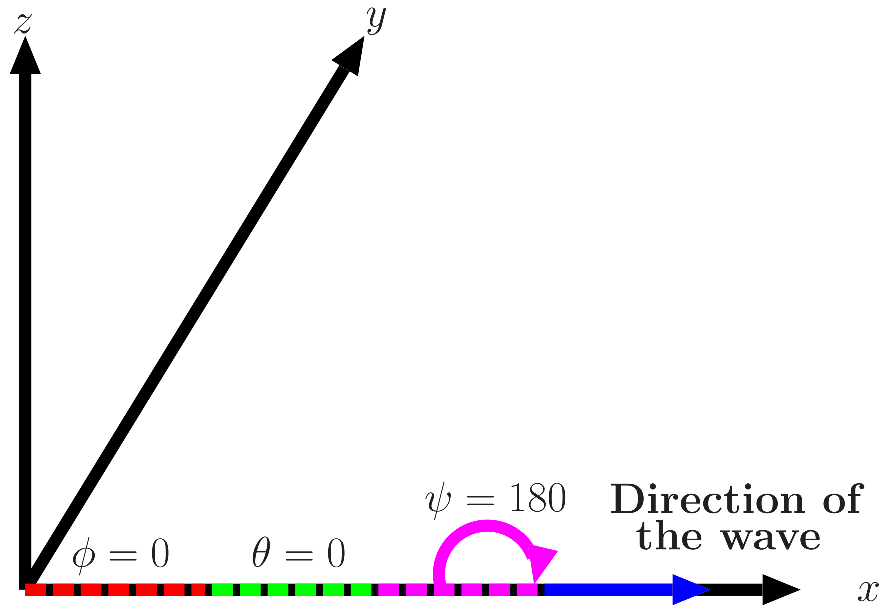


Figure 3.8: The direction of the wave generated by the Huygens surface when the angles ϕ,θ and ψ equal to 0,0 and 180 respectively.

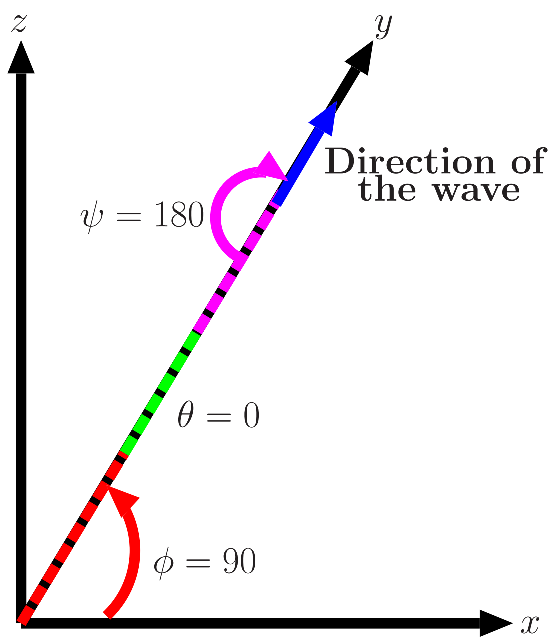


Figure 3.9: The direction of the wave generated by the Huygens surface when the angles ϕ,θ and ψ equal to 90,0 and 180 respectively.

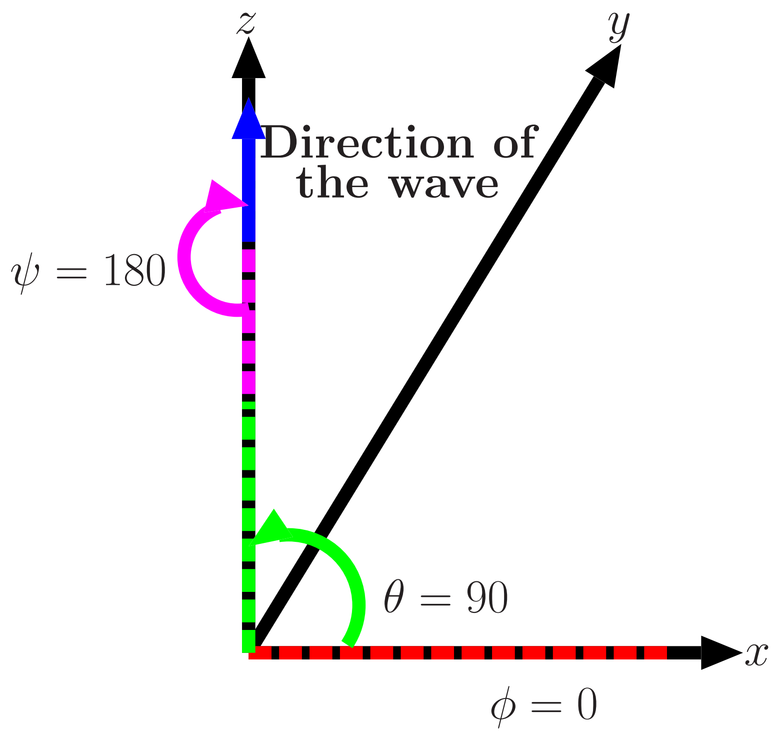


Figure 3.10: The direction of the wave generated by the Huygens surface when the angles ϕ,θ and ψ equal to $0, 90$ and 180 respectively.

is to the positive x axis, then the angles ϕ,θ and ψ equal to 0, 0 and 180 as shown in Figure 3.8. To get the opposite direction, negative x axis, ψ is set to 0 instead of 180. In Figure 3.9, the angles ϕ,θ and ψ are equal to 90, 0 and 180 so that the direction of the wave is towards the positive y axis. Similarly, when ϕ,θ and ψ equal to 0,90 and 180 respectively, the direction of the wave in the FDTD domain is upward as shown in Figure 3.10. Different settings of the angels may give the same direction for the wave. For example, the same direction of the wave is produced when ϕ,θ and ψ equal to 0,0 and 180, and when ϕ,θ and ψ equal to 0,90 and 270.

3.3 Total-Field/Scattered-Field Technique

Huygens ABC is based on total-field/scattered-field (TF/SF) concept which is a special case of Huygens (equivalence) principle, where two staggered surfaces are used, one for the equivalent magnetic currents and the other for the equivalent electric currents. The TF/SF formulation [2] assumes the following decomposition for the physical total electric field $\boldsymbol{E}_{tot}$ and physical total magnetic field $\boldsymbol{H}_{tot}$:

$$\boldsymbol{E}_{tot} = \boldsymbol{E}_{inc} + \boldsymbol{E}_{scat} \tag{3.19}$$

$$\boldsymbol{H}_{tot} = \boldsymbol{H}_{inc} + \boldsymbol{H}_{scat} \tag{3.20}$$

where:

- $\boldsymbol{E}_{inc}$ and $\boldsymbol{H}_{inc}$ are the values of the incident-wave fields. The incident fields are the fields that would exist when there are no materials within the space (only vacuum). $\boldsymbol{E}_{inc}$ and $\boldsymbol{H}_{inc}$ are assumed to be known at all grid points of the FDTD space [2].

- $\boldsymbol{E}_{scat}$ and $\boldsymbol{H}_{scat}$ are the values of scattered-wave fields. The scattered fields are generated when the incident wave interact with any material within the space. $\boldsymbol{E}_{scat}$ and $\boldsymbol{H}_{scat}$ are assumed to be initially unknown [2].

TF/SF technique divides the FDTD domain into TF and SF regions as shown in Figure 3.11 [2]. The TF region contains the incident wave fields and scattered wave fields. Both of those fields are used in the FDTD calculations. On the

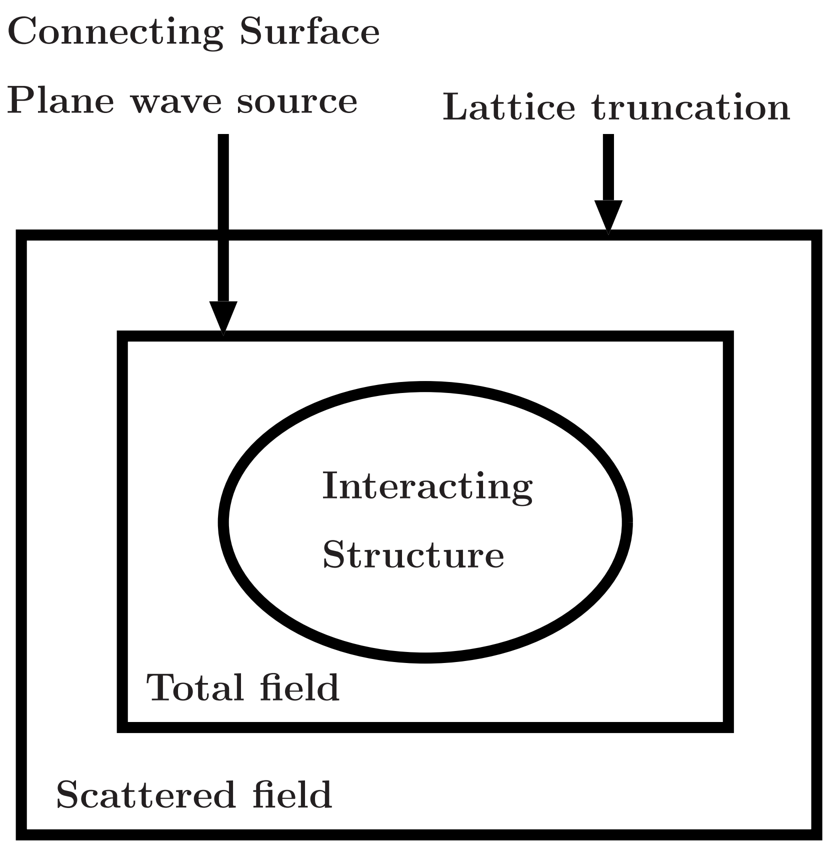


Figure 3.11: Total-field/Scattered-field zoning [2].

other hand, in the SF region, there is no incident wave, which means that only scattered wave fields are used in the computations.

The fields of the two distinct regions are assumed to be stored in computer memory [2]. The surface separating the TF and SF regions is physical and holds the equivalent magnetic and electric currents used to create such a separation. This surface connects the two regions and generates the incident wave [2]. The outer grid surrounding the SF region is actually terminating the calculation and acting as an ABC [2].

3.3.1 Benefits of the Connecting Surface in TF/SF technique

The use of the connecting surface in the TF/SF technique provides the following benefits:

- Ability to generate arbitrary incident wave. The use of the connecting surface gives the user the option to define the time waveform and duration, angle of incidence and angle of polarization for the generated incident wave. This surface well-defines the SF region because it confines the incident wave to TF region and it is transparent to outgoing waves [2].

- The connecting surface is Easy to implement. When TF/SF technique is used, the connected surface has to be programmed. This is done by simply calculating the incident field along the surface.

- Implementing an absorbing boundary condition. Based on the concept of the connecting surface, Section 3.4 illustrates an implementation of an absorbing boundary condition.

3.3.2 TF/SF 1D Formulation

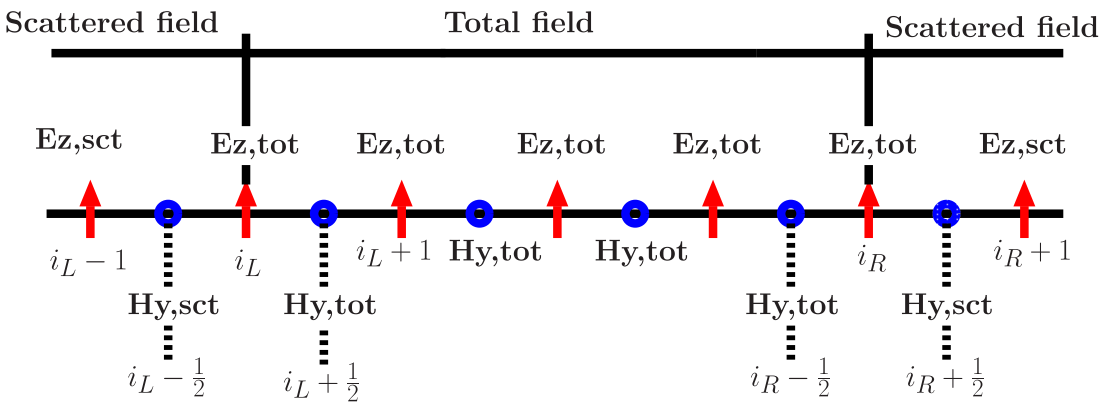


Figure 3.12: Total-field/Scattered-field component locations, 1D [2].

Figure 3.12 shows a 1D x-directed cut through the 2D surface showed in Figure 3.11. Consider the free-space 1D case, the finite-difference formula for E_z is expressed as follows [2]:

$$E_z^{n+1}(i) = E_z^n(i) + \frac{\Delta t}{\epsilon_0 \Delta x}\left[H_y^{n+\frac{1}{2}}(i+\tfrac{1}{2}) - H_y^{n+\frac{1}{2}}(i-\tfrac{1}{2})\right]. \tag{3.21}$$

(3.21) is used to express all total and scattered field components [2].

3.3.2.1 Left side computation with $x = i_L$

Applying (3.21) at the point i_L on left connecting surface shown in Figure 3.12 yields:

$$E^{n+1}_{z,tot}(i_L) = E^{n}_{z,tot}(i_L) + \frac{\Delta t}{\epsilon_0 \Delta x}\left[H^{n+\frac{1}{2}}_{y,tot}(i_L + \tfrac{1}{2}) - H^{n+\frac{1}{2}}_{y,sct}(i_L - \tfrac{1}{2})\right]. \tag{3.22}$$

In (3.22), the TF component is supposed to be calculated using only TF region components. However, both TF and SF magnetic field components are used to calculate $E^{n+1}_{z,tot}(i_L)$. This makes the equation inconsistent because TF region component is based on the difference of TF and SF region components [2]. To resolve this issue, one more term is added to balance the equation (3.22):

$$E^{n+1}_{z,tot}(i_L) = E^{n}_{z,tot}(i_L) + \frac{\Delta t}{\epsilon_0 \Delta x}\left[H^{n+\frac{1}{2}}_{y,tot}(i_L + \tfrac{1}{2}) - \left(H^{n+\frac{1}{2}}_{y,sct}(i_L - \tfrac{1}{2}) + H^{n+\frac{1}{2}}_{y,inc}(i_L - \tfrac{1}{2})\right)\right] \tag{3.23}$$

based on

$$H^{n+\frac{1}{2}}_{y,sct}(i_L - \tfrac{1}{2}) + H^{n+\frac{1}{2}}_{y,inc}(i_L - \tfrac{1}{2}) = H^{n+\frac{1}{2}}_{y,tot}(i_L - \tfrac{1}{2}). \tag{3.24}$$

Just like the electric field, according to (3.21), the magnetic field at the point $i_L - \frac{1}{2}$ on left connecting surface is calculated as follows:

$$H^{n+\frac{1}{2}}_{y,sct}(i_L - \tfrac{1}{2}) = H^{n-\frac{1}{2}}_{y,sct}(i_L - \tfrac{1}{2}) + \frac{\Delta t}{\mu_0 \Delta x}\left[E^{n}_{z,tot}(i_L) - E^{n}_{z,sct}(i_L - 1)\right] \tag{3.25}$$

(3.25) is inconsistent because only SF components should be used in calculating the SF component. But this equation uses the difference of TF and SF components in calculating $H^{n+\frac{1}{2}}_{y,sct}(i_L - \frac{1}{2})$. Thus, to balance the equation, one term is added [2]:

$$H^{n+\frac{1}{2}}_{y,sct}(i_L - \tfrac{1}{2}) = H^{n-\frac{1}{2}}_{y,sct}(i_L - \tfrac{1}{2}) + \frac{\Delta t}{\mu_0 \Delta x}\left[\left(E^{n}_{z,tot}(i_L) - E^{n}_{z,inc}(i_L)\right) - E^{n}_{z,sct}(i_L - 1)\right] \tag{3.26}$$

based on

$$E^{n}_{z,tot}(i_L) - E^{n}_{z,inc}(i_L) = E^{n}_{z,sct}(i_L). \tag{3.27}$$

3.3.2.2 Right side computation with $x = i_R$

Similarly, at the point i_R on the connecting surface at the right in Figure 3.12, the electric field is calculated as follows:

$$E_{z,tot}^{n+1}(i_R) = E_{z,tot}^{n}(i_R) + \frac{\Delta t}{\epsilon_0 \Delta x}\left[H_{y,sct}^{n+\frac{1}{2}}(i_R + \tfrac{1}{2}) + H_{y,inc}^{n+\frac{1}{2}}(i_R + \tfrac{1}{2}) - H_{y,tot}^{n+\frac{1}{2}}(i_R - \tfrac{1}{2})\right] \tag{3.28}$$

based on

$$H_{y,sct}^{n+\frac{1}{2}}(i_R + \tfrac{1}{2}) + H_{y,inc}^{n+\frac{1}{2}}(i_R + \tfrac{1}{2}) = H_{y,tot}^{n+\frac{1}{2}}(i_R + \tfrac{1}{2}). \tag{3.29}$$

Just like the electric field, the magnetic field at the point $i_R + \frac{1}{2}$ on the connecting surface at the right in Figure 3.12 is calculated as [2]:

$$H_{y,sct}^{n+\frac{1}{2}}(i_R + \tfrac{1}{2}) = H_{y,sct}^{n-\frac{1}{2}}(i_R + \tfrac{1}{2}) + \frac{\Delta t}{\mu_0 \Delta x}\left[E_{z,sct}^{n}(i_R + 1) - E_{z,tot}^{n}(i_R) + E_{z,inc}^{n}(i_R)\right] \tag{3.30}$$

based on

$$-E_{z,tot}^{n}(i_R) + E_{z,inc}^{n}(i_R) = -E_{z,sct}^{n}(i_R). \tag{3.31}$$

3.4 Huygens Absorbing Boundary Conditions

There are two novel ABCs used in numerical electromagnetics with FDTD method. These ABCs are the Multiple Absorbing Surfaces (MAS) condition [31] and the re-Radiating Boundary Condition (rRBC) [32] [3]. Both MAS and rRBC are based on the same idea of canceling the outgoing field by radiating a field equal in magnitude and opposite in direction. However, the value of the outgoing field on the surface where the equivalent field is introduced cannot accurately be known [3]. This surface is called Huygens surface [33][3] as mentioned in Section 3.1. Those ABCs need to estimate the values of the outgoing waves that would exist on the Huygens surface. This estimation is done by an operator. Since the cancellation is based on estimation, which is not perfect, it results a certain amount of reflection. To resolve this issue Jean-Pierre Bérenger introduced the HABC. The HABC is based on the same concept of canceling the outgoing waves by introducing the opposite field on the Huygens surface, but it is more general than MAS and rRBC [3]. An important feature of the HABC is that it can be placed close to the object in the FDTD domain. This gives the HABC the ability to be easily combined with other ABCs as shown later in this Section.

3.4.1 Principle of ABC based on Huygens surface

The FDTD implementations require an ABC that surrounds the structure of interest and absorbs the outgoing waves. This can be done by using the Huygens surface introduced in Section 3.1. The Huygens surface can cancel the outgoing waves by generating fields equal in magnitude and opposite in direction. On the Huygens surface, the values of electric and magnetic current densities $\boldsymbol{J}_s$ and $\boldsymbol{K}_s$ from (3.1) and (3.2) are set to $\boldsymbol{E}_i$ and $\boldsymbol{H}_i$ which are equal to the outgoing field and opposite in direction [3]. $\boldsymbol{E}_i$ and $\boldsymbol{H}_i$ are actually the fields that would exist at the location of the Huygens surface. If $\boldsymbol{E}_i$ and $\boldsymbol{H}_i$ are the exact values, the Huygens surface will perfectly absorbs the outgoing waves. But, those fields are estimated by Higdon operator [19][20] using the neighbouring FDTD cells of the Huygens surface [3]. This ABC, which is implemented based on the Huygens surface, is called the Huygens Absorbing Boundary Condition (HABC).

3.4.2 HABC in FDTD space

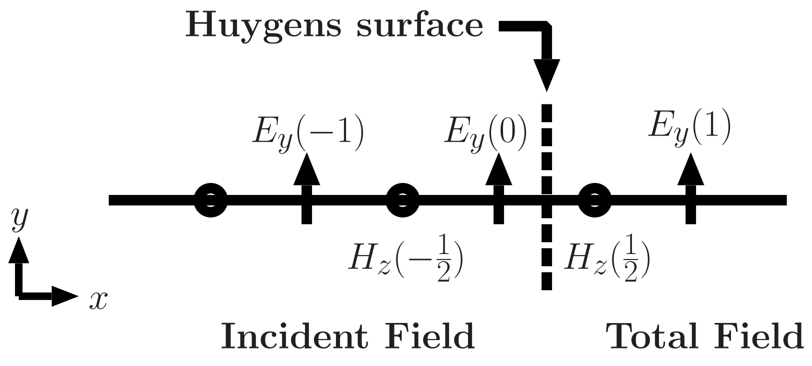


Figure 3.13: 1D implementation of Huygens surface in FDTD grid [3].

Consider the use of the Huygens surface in 1D FDTD space [3] as shown in Figure 3.13. The Huygens surface is placed between the nodes $E_y(0)$ and $H_z(\frac{1}{2})$ while the incident wave is assumed to be propagating in $+x$ direction. At the node $E_y(0)$, using (3.1), the equivalent current at the time $n+\frac{1}{2}$ is given by:

$$J_{S_y}^{n+\frac{1}{2}}(0) = -\widetilde{H}_z^{n+\frac{1}{2}}(\frac{1}{2}). \tag{3.32}$$

Similarly, at the node $H_z(\frac{1}{2})$, the equivalent current at the time n when applying (3.2) is:

$$K^n_{S_z}(\frac{1}{2}) = -\widetilde{E}^n_y(0). \tag{3.33}$$

$\widetilde{E}_y$ and $\widetilde{H}_z$ in equations (3.32) and (3.33) are the estimated values of the incident fields. The following elementary operator is used to estimate the incident fields:

$$P_e = K(-\Delta x)Z(-\Delta t). \tag{3.34}$$

Where Δx and Δt are the space and time steps of the FDTD grid. This estimations reads:

$$\widetilde{H}^{n+\frac{1}{2}}_z(\frac{1}{2}) = H^{n-\frac{1}{2}}_z(-\frac{1}{2}) \tag{3.35}$$

and

$$\widetilde{E}^n_y(0) = E^{n-1}_y(-1). \tag{3.36}$$

The following FDTD equations at the nodes $E_y(0)$ and $H_z(\frac{1}{2})$ are not consistent according to TF/SF technique in Section 3.3.

$$E^{n+1}_y(0) = E^n_y(0) - \frac{\Delta t}{\epsilon_0 \Delta x}\left[H^{n+\frac{1}{2}}_z(\frac{1}{2}) - H^{n+\frac{1}{2}}_z(-\frac{1}{2})\right] \tag{3.37}$$

$$H^{n+\frac{1}{2}}_z(\frac{1}{2}) = H^{n-\frac{1}{2}}_z(\frac{1}{2}) - \frac{\Delta t}{\mu_0 \Delta x}\left[E^n_y(1) - E^n_y(0)\right]. \tag{3.38}$$

Substituting (3.35) and (3.36) into (3.32) and (3.33) and introducing the results into the FDTD equations 3.37 and 3.38 yield the equation:

$$E^{n+1}_y(0) = E^n_y(0) - \frac{\Delta t}{\epsilon_0 \Delta x}\left[H^{n+\frac{1}{2}}_z(\frac{1}{2}) - H^{n+\frac{1}{2}}_z(-\frac{1}{2})\right] - \frac{\Delta t}{\epsilon_0 \Delta x}H^{n-\frac{1}{2}}_z(-\frac{1}{2}) \tag{3.39}$$

at the node $E_y(0)$, and the equation:

$$H^{n+\frac{1}{2}}_z(\frac{1}{2}) = H^{n-\frac{1}{2}}_z(\frac{1}{2}) - \frac{\Delta t}{\mu_0 \Delta x}\left[E^n_y(1) - E^n_y(0)\right] - \frac{\Delta t}{\mu_0 \Delta x}E^{n-1}_y(-1) \tag{3.40}$$

at the node $H_z(\frac{1}{2})$, respectively [3]. (3.39) and (3.40) are now consistent based on the TF/SF technique and can be applied for the Huygens surface in the FDTD domain.

3.4.3 3D HABC in FDTD space

In 3D FDTD space, the HABC is implemented as 6 Huygens surfaces [30] that surround the computational domain. The planes are placed at $i = i_{\text{tlmin}}$, $i = i_{\text{tlmax}}$, $j = j_{\text{tlmin}}$, $j = j_{\text{tlmax}}$, $k = k_{\text{tlmin}}$, and $k = k_{\text{tlmax}}$. We assume the incident waves propagate from around the centre to those 6 planes.

Instead of the standard FDTD equations, the following equations are applied when $\widetilde{E}^{\tilde{n}}(i,j,k)$ and $\widetilde{H}^{\tilde{n}+1}(i,j,k)$ have values on the HABC [30].

$$E_x^{n+1}(i,j,k) = E_x^n(i,j,k) + \frac{\Delta t}{\epsilon(i,j,k)} \times \tag{3.41}$$

$$\left[\frac{H_z^{n+1}(i,j,k) + \widetilde{H}_z^{\tilde{n}+1}(i,j,k) - \left(H_z^{n+1}(i,j-1,k) + \widetilde{H}_z^{\tilde{n}+1}(i,j-1,k)\right)}{\frac{\Delta y_{(j-1)}+\Delta y_{(j)}}{2}} - \frac{H_y^{n+1}(i,j,k) + \widetilde{H}_y^{\tilde{n}+1}(i,j,k) - \left(H_y^{n+1}(i,j,k-1) + \widetilde{H}_y^{\tilde{n}+1}(i,j,k-1)\right)}{\Delta z_{(k)}}\right]$$

$$[i_{min} \le i \le i_{max} - 1, \quad j = j_{\text{tlmin}}, j = j_{\text{tlmax}} \quad k = k_{\text{tlmin}}, k = k_{\text{tlmax}}]$$

$$E_y^{n+1}(i,j,k) = E_y^n(i,j,k) + \frac{\Delta t}{\epsilon(i,j,k)} \times \tag{3.42}$$

$$\left[\frac{H_x^{n+1}(i,j,k) + \widetilde{H}_x^{\tilde{n}+1}(i,j,k) - \left(H_x^{n+1}(i,j,k-1) + \widetilde{H}_x^{\tilde{n}+1}(i,j,k-1)\right)}{\Delta z_{(k)}} - \frac{H_z^{n+1}(i,j,k) + \widetilde{H}_z^{\tilde{n}+1}(i,j,k) - \left(H_z^{n+1}(i-1,j,k) + \widetilde{H}_z^{\tilde{n}+1}(i-1,j,k)\right)}{\frac{\Delta x_{(i-1)}+\Delta x_{(i)}}{2}}\right]$$

$$[i = i_{\text{tlmin}}, i = i_{\text{tlmax}} \quad j_{min} \le j \le j_{max} - 1, \quad k = k_{\text{tlmin}}, k = k_{\text{tlmax}}]$$

$$
\begin{aligned}
E_z^{n+1}{}_{(i,j,k)} = E_z^{n}{}_{(i,j,k)} + \frac{\Delta t}{\epsilon(i,j,k)} \times \qquad (3.43)\\
+\frac{\Delta t}{\epsilon(i,j,k)}\Bigg[\frac{H_y^{n+1}{}_{(i,j,k)} + \widetilde{H}_y^{\widetilde{n}+1}{}_{(i,j,k)} - \big(H_y^{n+1}{}_{(i-1,j,k)} + \widetilde{H}_y^{\widetilde{n}+1}{}_{(i-1,j,k)}\big)}{\frac{\Delta x_{(i-1)}+\Delta x_{(i)}}{2}}\\
-\frac{H_x^{n+1}{}_{(i,j,k)} + \widetilde{H}_x^{\widetilde{n}+1}{}_{(i,j,k)} - \big(H_x^{n+1}{}_{(i,j-1,k)} + \widetilde{H}_x^{\widetilde{n}+1}{}_{(i,j-1,k)}\big)}{\frac{\Delta y_{(j-1)}+\Delta y_{(j)}}{2}}\Bigg]\\
[i = i_{\text{tlmin}}, i = i_{\text{tlmax}}\\
j = j_{\text{tlmin}}, j = j_{\text{tlmax}}\\
k_{min} \le k \le k_{max} - 1].
\end{aligned}
$$

$$
\begin{aligned}
H_x^{n+1}{}_{(i,j,k)} &= H_x^{n}{}_{(i,j,k)} \qquad (3.44)\\
&+\frac{\Delta t}{\mu}\Bigg[\frac{E_y^{n}{}_{(i,j,k+1)} - \widetilde{E}_y^{\widetilde{n}}{}_{(i,j,k+1)} - \big(E_y^{n}{}_{(i,j,k)} - \widetilde{E}_y^{\widetilde{n}}{}_{(i,j,k)}\big)}{\frac{\Delta z_{(k-1)}+\Delta z_{(k)}}{2}}\\
&-\frac{E_z^{n}{}_{(i,j+1,k)} - \widetilde{E}_z^{\widetilde{n}}{}_{(i,j+1,k)} - \big(E_z^{n}{}_{(i,j,k)} - \widetilde{E}_z^{\widetilde{n}}{}_{(i,j,k)}\big)}{\Delta y_{(j)}}\Bigg]\\
&[i_{min} + 1 \le i \le i_{max} - 1,\\
&j + 1 = j_{\text{tlmin}}, j = j_{\text{tlmax}}\\
&k + 1 = k_{\text{tlmin}}, k = k_{\text{tlmax}}].
\end{aligned}
$$

$$
\begin{aligned}
H_y^{n+1}{}_{(i,j,k)} &= H_y^{n}{}_{(i,j,k)} \qquad (3.45)\\
&+\frac{\Delta t}{\mu}\Bigg[\frac{E_z^{n}{}_{(i+1,j,k)} - \widetilde{E}_z^{\widetilde{n}}{}_{(i+1,j,k)} - \big(E_z^{n}{}_{(i,j,k)} - \widetilde{E}_z^{\widetilde{n}}{}_{(i,j,k)}\big)}{\Delta x_{(i)}}\\
&-\frac{E_x^{n}{}_{(i,j,k+1)} - \widetilde{E}_x^{\widetilde{n}}{}_{(i,j,k+1)} - \big(E_x^{n}{}_{(i,j,k)} - \widetilde{E}_x^{\widetilde{n}}{}_{(i,j,k)}\big)}{\frac{\Delta z_{(k-1)}+\Delta z_{(k)}}{2}}\Bigg]\\
&i + 1 = i_{\text{tlmin}}, i = i_{\text{tlmax}}\\
&j_{min} + 1 \le j \le j_{max} - 1,\\
&k + 1 = k_{\text{tlmin}}, k = k_{\text{tlmax}}].
\end{aligned}
$$

$$\begin{aligned}
H_z^{n+1}{}_{(i,j,k)} &= H_z^{n}{}_{(i,j,k)} \\
&+\frac{\Delta t}{\mu}\left[\frac{E_x^{n}{}_{(i,j+1,k)} - \widetilde{E}_x^{\widetilde{n}}{}_{(i,j+1,k)} - \left(E_x^{n}{}_{(i,j,k)} - \widetilde{E}_x^{\widetilde{n}}{}_{(i,j,k)}\right)}{\Delta y_{(j)}}\right. \\
&-\left.\frac{E_y^{n}{}_{(i+1,j,k)} - \widetilde{E}_y^{\widetilde{n}}{}_{(i+1,j,k)} - \left(E_y^{n}{}_{(i,j,k)} - \widetilde{E}_y^{\widetilde{n}}{}_{(i,j,k)}\right)}{\Delta x_{(i)}}\right] \\
&[i+1 = i_{\text{tlmin}}, i = i_{\text{tlmax}} \\
&j+1 = j_{\text{tlmax}}, j = j_{\text{tlmax}} \\
&k_{min}+1 \le k \le k_{max}-1].
\end{aligned} \tag{3.46}$$

The values for $\widetilde{E}^{\widetilde{n}}{}_{(i,j,k)}$ and $\widetilde{H}^{\widetilde{n}+1}{}_{(i,j,k)}$ are as follows.

$$\begin{aligned}
\widetilde{E}_y^{\widetilde{n}}{}_{(i,j,k)} &= E_y^{n-1}{}_{(i\pm1,j,k)} \\
+\frac{\Delta t - \Delta x_{(i)}\sqrt{\mu_0\epsilon_0\epsilon(i\pm1,j,k)}}{\Delta t + \Delta x_{(i)}\sqrt{\mu_0\epsilon_0\epsilon(i,j,k)}}&\left(E_y^{n}{}_{(i\pm1,j,k)} - \widetilde{E}_y^{\widetilde{n}-1}{}_{(i,j,k)}\right) \\
i = i_{\text{tlmin}}\text{for } +\text{sign}, &i_{\text{tlmax}}\text{for } -\text{sign} \\
j_{\text{min}} \le j &\le j_{\text{max}}-1 \\
k_{\text{min}}+1 \le k &\le k_{\text{max}}-1
\end{aligned} \tag{3.47}$$

$$\begin{aligned}
\widetilde{E}_z^{\widetilde{n}}{}_{(i,j,k)} &= E_z^{n-1}{}_{(i\pm1,j,k)} \\
+\frac{\Delta t - \Delta x_{(i)}\sqrt{\mu_0\epsilon_0\epsilon(i\pm1,j,k)}}{\Delta t + \Delta x_{(i)}\sqrt{\mu_0\epsilon_0\epsilon(i,j,k)}}&\left(E_z^{n}{}_{(i\pm1,j,k)} - \widetilde{E}_z^{\widetilde{n}-1}{}_{(i,j,k)}\right) \\
i = i_{\text{tlmin}}\text{for } +\text{sign}, &i_{\text{tlmax}}\text{for } -\text{sign} \\
j_{\text{min}}+1 \le j &\le j_{\text{max}}-1 \\
k_{\text{min}} \le k &\le k_{\text{max}}-1
\end{aligned} \tag{3.48}$$

$$\begin{aligned}\widetilde{E}_x^{\widetilde{n}}{}_{(i,j,k)} &= E_x^{n-1}{}_{(i,j\pm1,k)} \\ +\frac{\Delta t - \Delta y_{(j)}\sqrt{\mu_0\epsilon_0\epsilon_{(i,j\pm1,k)}}}{\Delta t + \Delta y_{(j)}\sqrt{\mu_0\epsilon_0\epsilon_{(i,j,k)}}}&\left(E_x^n{}_{(i,j\pm1,k)} - \widetilde{E}_x^{\widetilde{n}-1}{}_{(i,j,k)}\right) \\ i_{\min} \le i &\le i_{\max}-1 \\ j = j_{\text{tl}\min}\text{for } +\text{sign}, &j_{\text{tl}\max}\text{for } -\text{sign} \\ k_{\min}+1 \le k &\le k_{\max}-1\end{aligned} \tag{3.49}$$

$$\begin{aligned}\widetilde{E}_z^{\widetilde{n}}{}_{(i,j,k)} &= E_z^{n-1}{}_{(i,j\pm1,k)} \\ +\frac{\Delta t - \Delta y_{(j)}\sqrt{\mu_0\epsilon_0\epsilon_{(i,j\pm1,k)}}}{\Delta t + \Delta y_{(j)}\sqrt{\mu_0\epsilon_0\epsilon_{(i,j,k)}}}&\left(E_z^n{}_{(i,j\pm1,k)} - \widetilde{E}_z^{\widetilde{n}-1}{}_{(i,j,k)}\right) \\ i_{\min}+1 \le i &\le i_{\max}-1 \\ j = j_{\text{tl}\min}\text{for } +\text{sign}, &j_{\text{tl}\max}\text{for } -\text{sign} \\ k_{\min} \le k &\le k_{\max}-1\end{aligned} \tag{3.50}$$

$$\begin{aligned}\widetilde{E}_x^{\widetilde{n}}{}_{(i,j,k)} &= E_x^{n-1}{}_{(i,j,k\pm1)} \\ +\frac{\Delta t - \Delta z_{(k)}\sqrt{\mu_0\epsilon_0\epsilon_{(i,j,k\pm1)}}}{\Delta t + \Delta z_{(k)}\sqrt{\mu_0\epsilon_0\epsilon_{(i,j,k)}}}&\left(E_x^n{}_{(i,j,k\pm1)} - \widetilde{E}_x^{\widetilde{n}-1}{}_{(i,j,k)}\right) \\ i_{\min} \le i &\le i_{\max}-1 \\ j_{\min}+1 \le j &\le j_{\max}-1 \\ k = k_{\text{tl}\min}\text{for } +\text{sign}, &k_{\text{tl}\max}\text{for } -\text{sign}\end{aligned} \tag{3.51}$$

$$\begin{aligned}\widetilde{E}_y^{\tilde{n}}{}_{(i,j,k)} &= E_y^{n-1}{}_{(i,j,k\pm1)} \\ +\frac{\Delta t - \Delta z_{(k)}\sqrt{\mu_0\epsilon_0\epsilon_{(i,j,k\pm1)}}}{\Delta t + \Delta z_{(k)}\sqrt{\mu_0\epsilon_0\epsilon_{(i,j,k)}}} &\left(E_y^{n}{}_{(i,j,k\pm1)} - \widetilde{E}_y^{\tilde{n}-1}{}_{(i,j,k)}\right) \\ i_{\min}+1 \le i &\le i_{\max}-1 \\ j_{\min} \le j &\le j_{\max}-1 \\ k = k_{\text{tlmin}}\text{for } +\text{sign}, &k_{\text{tlmax}}\text{for } -\text{sign}\end{aligned} \tag{3.52}$$

$$\begin{aligned}\widetilde{H}_z^{\tilde{n}+1}{}_{(i,j,k)} &= H_z^{n}{}_{(i\pm1,j,k)} \\ +\frac{\Delta t - \Delta x_{(i)}\sqrt{\mu_0\epsilon_0\epsilon_{(i\pm1,j,k)}}}{\Delta t + \Delta x_{(i)}\sqrt{\mu_0\epsilon_0\epsilon_{(i,j,k)}}} &\left(H_z^{n+1}{}_{(i\pm1,j,k)} - \widetilde{H}_z^{\tilde{n}}{}_{(i,j,k)}\right) \\ i = i_{\text{tlmin}}-1 \quad \text{for } +\text{sign}, &i_{\text{tlmax}}\text{for } -\text{sign} \\ j_{\min} \le j &\le j_{\max}-1 \\ k_{\min}+1 \le k &\le k_{\max}-1\end{aligned} \tag{3.53}$$

$$\begin{aligned}\widetilde{H}_y^{\tilde{n}+1}{}_{(i,j,k)} &= H_y^{n}{}_{(i\pm1,j,k)} \\ +\frac{\Delta t - \Delta x_{(i)}\sqrt{\mu_0\epsilon_0\epsilon_{(i\pm1,j,k)}}}{\Delta t + \Delta x_{(i)}\sqrt{\mu_0\epsilon_0\epsilon_{(i,j,k)}}} &\left(H_y^{n+1}{}_{(i\pm1,j,k)} - \widetilde{H}_y^{\tilde{n}}{}_{(i,j,k)}\right) \\ i = i_{\text{tlmin}}-1 \quad \text{for } +\text{sign}, &i_{\text{tlmax}}\text{for } -\text{sign} \\ j_{\min}+1 \le j &\le j_{\max}-1 \\ k_{\min} \le k &\le k_{\max}-1\end{aligned} \tag{3.54}$$

$$\begin{aligned}\widetilde{H}_z^{\widetilde{n}+1}{}_{(i,j,k)} &= H_z^n{}_{(i,j\pm1,k)}\\+\frac{\Delta t-\Delta y_{(j)}\sqrt{\mu_0\epsilon_0\epsilon_{(i,j\pm1,k)}}}{\Delta t+\Delta y_{(j)}\sqrt{\mu_0\epsilon_0\epsilon_{(i,j,k)}}}&\big(H_z^{n+1}{}_{(i,j\pm1,k)}-\widetilde{H}_z^{\widetilde{n}}{}_{(i,j,k)}\big)\\ i_{\min}\le i&\le i_{\max}-1\\ j=j_{\mathrm{tl}_{\min}}-1\quad\text{for }+\text{sign},\,j_{\mathrm{tl}_{\max}}&\text{for }-\text{sign}\\ k_{\min}+1\le k&\le k_{\max}-1\end{aligned}\tag{3.55}$$

$$\begin{aligned}\widetilde{H}_x^{\widetilde{n}+1}{}_{(i,j,k)} &= H_x^n{}_{(i,j\pm1,k)}\\+\frac{\Delta t-\Delta y_{(j)}\sqrt{\mu_0\epsilon_0\epsilon_{(i,j\pm1,k)}}}{\Delta t+\Delta y_{(j)}\sqrt{\mu_0\epsilon_0\epsilon_{(i,j,k)}}}&\big(H_x^{n+1}{}_{(i,j\pm1,k)}-\widetilde{H}_x^{\widetilde{n}}{}_{(i,j,k)}\big)\\ i_{\min}+1\le i&\le i_{\max}-1\\ j=j_{\mathrm{tl}_{\min}}-1\quad\text{for }+\text{sign},\,j_{\mathrm{tl}_{\max}}&\text{for }-\text{sign}\\ k_{\min}\le k&\le k_{\max}-1\end{aligned}\tag{3.56}$$

$$\begin{aligned}\widetilde{H}_y^{\widetilde{n}+1}{}_{(i,j,k)} &= H_y^n{}_{(i,j,k\pm1)}\\+\frac{\Delta t-\Delta z_{(k)}\sqrt{\mu_0\epsilon_0\epsilon_{(i,j,k\pm1)}}}{\Delta t+\Delta z_{(k)}\sqrt{\mu_0\epsilon_0\epsilon_{(i,j,k)}}}&\big(H_y^{n+1}{}_{(i,j,k\pm1)}-\widetilde{H}_y^{\widetilde{n}}{}_{(i,j,k)}\big)\\ i_{\min}\le i&\le i_{\max}-1\\ j_{\min}+1\le j&\le j_{\max}-1\\ k=k_{\mathrm{tl}_{\min}}-1\quad\text{for }+\text{sign},\,k_{\mathrm{tl}_{\max}}&\text{for }-\text{sign}\end{aligned}\tag{3.57}$$

$$\begin{aligned}
\widetilde{H}_x^{\widetilde{n}+1}{}_{(i,j,k)} &= H_x^n{}_{(i,j,k\pm1)} \\
+\frac{\Delta t - \Delta z_{(k)}\sqrt{\mu_0\epsilon_0\epsilon_{(i,j,k\pm1)}}}{\Delta t + \Delta z_{(k)}\sqrt{\mu_0\epsilon_0\epsilon_{(i,j,k)}}} &\left(H_x^{n+1}{}_{(i,j,k\pm1)} - \widetilde{H}_x^{\widetilde{n}}{}_{(i,j,k)}\right) \\
i_{\min}+1 \le i &\le i_{\max}-1 \\
j_{\min} \le j &\le j_{\max}-1 \\
k = k_{\text{tlmin}}-1 \ \text{ for } +\text{sign}, &\, k_{\text{tlmax}}\text{for } -\text{sign}
\end{aligned} \tag{3.58}$$

Equation (3.44) requires equations (3.52) and (3.50), equation (3.45) requires equations (3.48) and (3.51) and finally equation (3.46) requires equations (3.49) and (3.47) [30].

Similarly, equation (3.41) requires equations (3.55) and (3.57), equation (3.42) requires both equations (3.58) and (3.53) and finally equation (3.43) requires equations (3.54) and (3.56) [30].

3.4.4 Numerical Experiments

In this section, the HABC is tested in absorbing the high frequency traveling waves. This is done by increasing the size of the FDTD domain. When the FDTD domain is large, the reflection of the low frequency evanescent fields is negligible.

3.4.4.1 Environmental Settings

Figure 3.14 shows the 3D FDTD space with 100×10 cells plate. The space is excited with Huygens excitation one cell from the plate. Δt is 18ps and Δs is 1 cm. The HABC is used for the termination of the FDTD domain by placing the Huygens surfaces 3 cells away from the plate. E_z is observed at the corner of the plate. d_{PEC} is the number of the FDTD grid points between the object and the outer PEC. The reference is 15 cells CFS-PML placed 100 cells away from the object.

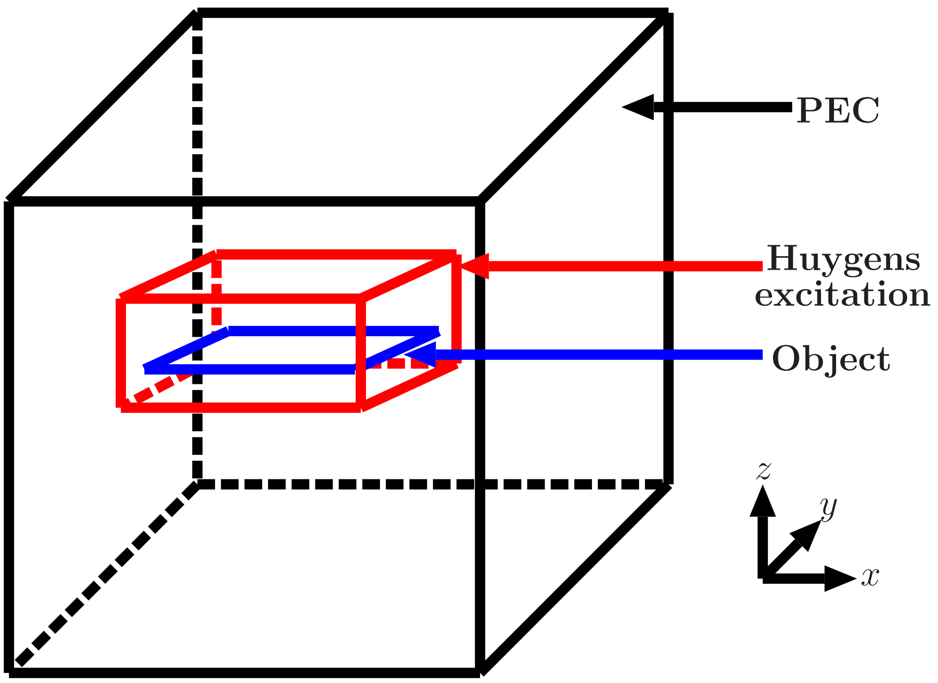


Figure 3.14: The object, Huygens excitation and the HABC in 3D FDTD space. The HABC is placed 3 cells away from the object.

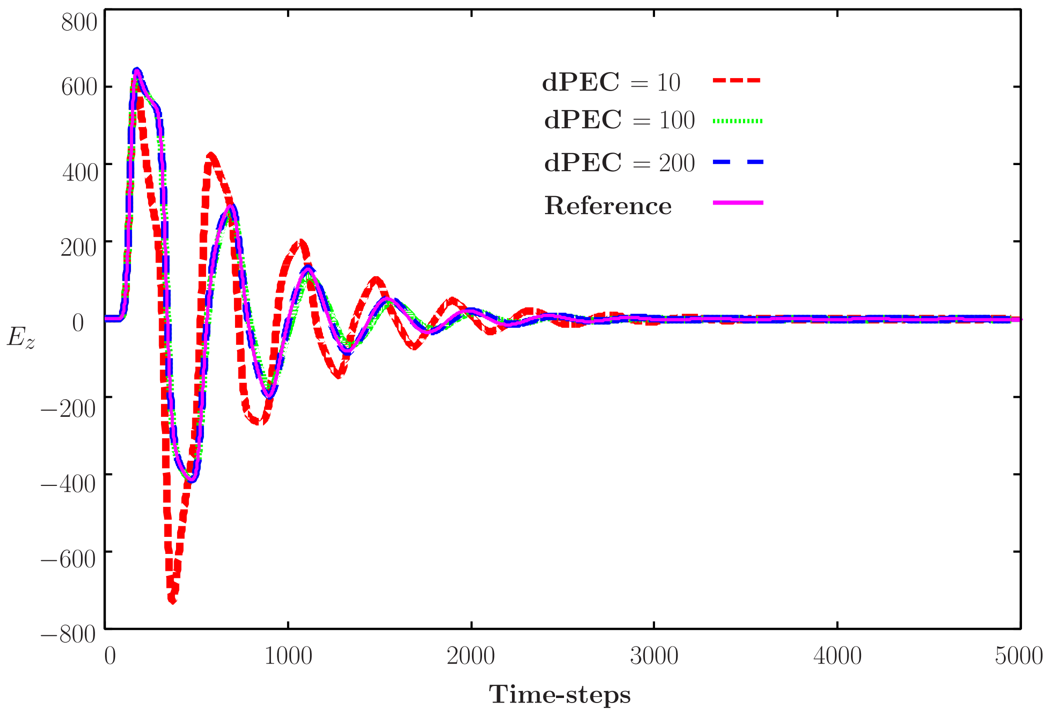


Figure 3.15: Observation at the corner of the plate. 1 ns Gaussian pulse Huygens excitation. $\Delta t = 18$ps.

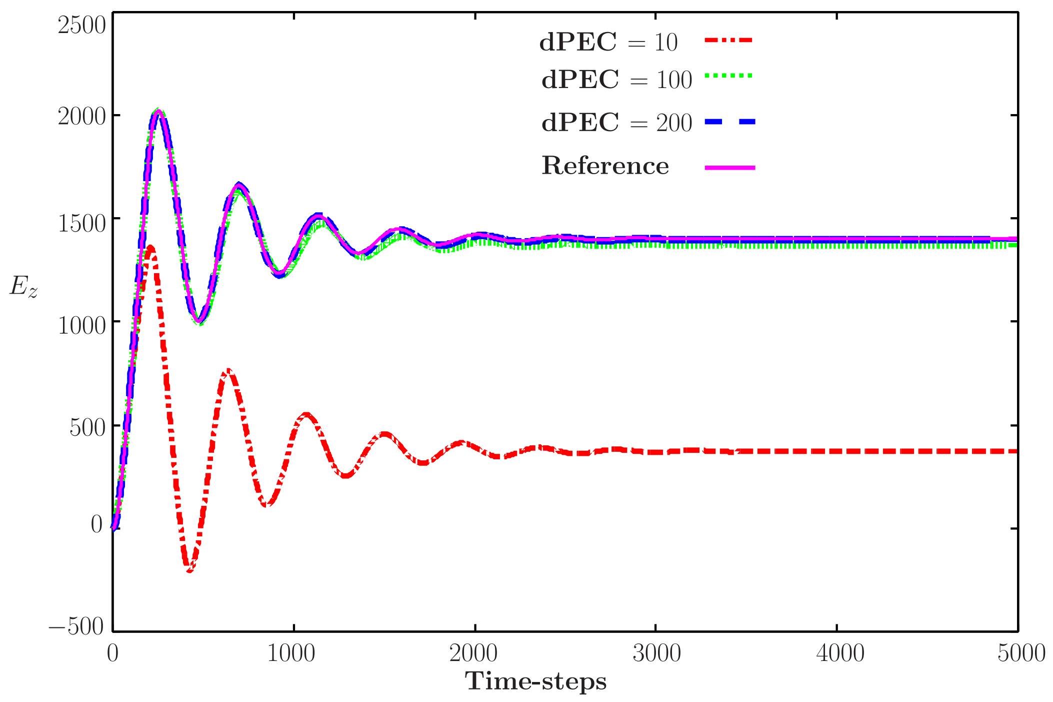


Figure 3.16: Observation at the corner of the plate. 1 ns Unit-Step Huygens excitaion. $\Delta t = 18$ps.

3.4.4.2 Numerical Results

Figure 3.15 shows the results of $d_{PEC} = 10$, $d_{PEC} = 100$, $d_{PEC} = 200$ and the reference in the case of 1ns Gaussian pulse Huygens excitation. The results are superimposed to the reference when d_{PEC} grows. For instance, when $d_{PEC} = 100$, peak values of the HABC and the reference are 643.93 v.s. 643.5 respectively and the difference is $0.4/640 = 0.6/1000$.

1 ns Unit-Step Huygens excitation results are displayed in Figure 3.16, the waveforms represent the results of $d_{PEC} = 10$, $d_{PEC} = 100$, $d_{PEC} = 200$ and the reference. It is clear from Figure 3.16 that the results tend to the reference as d_{PEC} increases. When $d_{PEC} = 100$ cells, the plateau of the HABC is 1373.5 while the plateau of reference is 1403. The difference in the plateau is $\frac{1403-1373.5}{1403} = 1.4\%$. The difference in plateau falls down to 0.35% (the plateau is 1398) in case of $d_{PEC} = 200$ computations.

3.4.4.3 Conclusion

When the FDTD domain is large enough and the reflection of the low frequency evanescent waves is negligible, the HABC is comparable with the CFS-PML in absorbing the high frequency traveling waves. Also, the HABC absorbs the traveling wave even when it is placed only 3 FDTD cells away from the object.

3.5 The Stretched-Mesh HABC

In the Stretched-Mesh HABC (SM-HABC), the HABC, which absorbs the traveling waves, is combined with another ABC called the Stretched Mesh (SM) that is capable to absorb the evanescent waves (SM is not the complex coordinate stretching used in the CPML literature). By stretching the FDTD mesh, the SM decreases the overall number of cells in the FDTD space while keeping the large physical domain. The SM suffered from the strong reflection of high frequencies from the large cells because the ABCs were placed at the outer boundary. However, as shown in Section 3.4.4, the HABC can be placed close to the object instead of the outer boundary. When the HABC is placed in front of the SM, it will absorb the traveling waves and permits only the evanescent waves to get into the SM. Those evanescent waves are then absorbed by the SM because of their natural decrease. Thus, the combination of the HABC and SM provides the ability to absorb both the high frequency traveling waves and low frequency evanescent waves. Figure 3.17 shows the combination of the HABC and the SM in the FDTD domain.

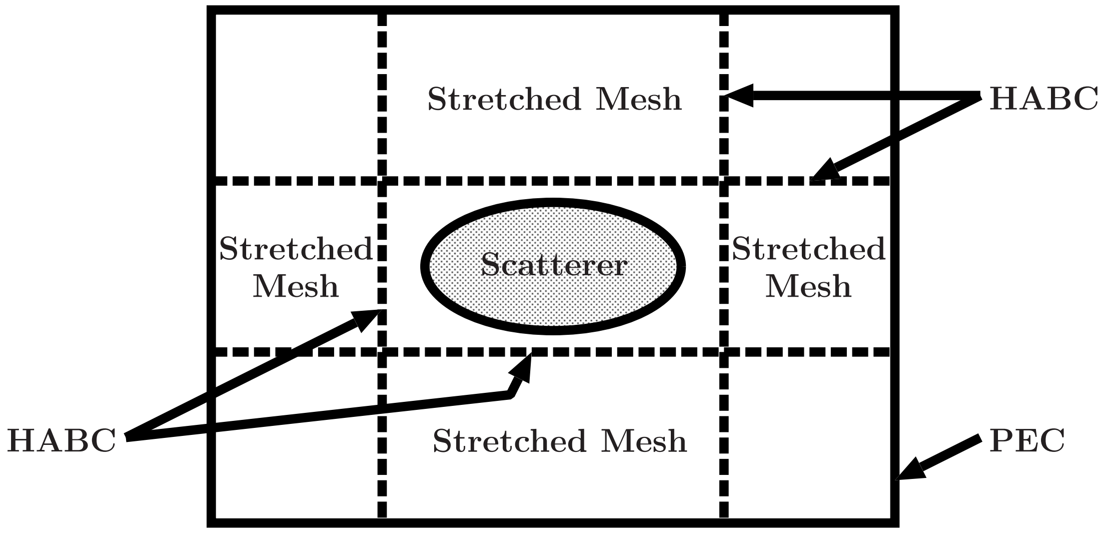


Figure 3.17: The combination of HABC placed close to an object with a large exterior domain ended with a PEC condition [4].

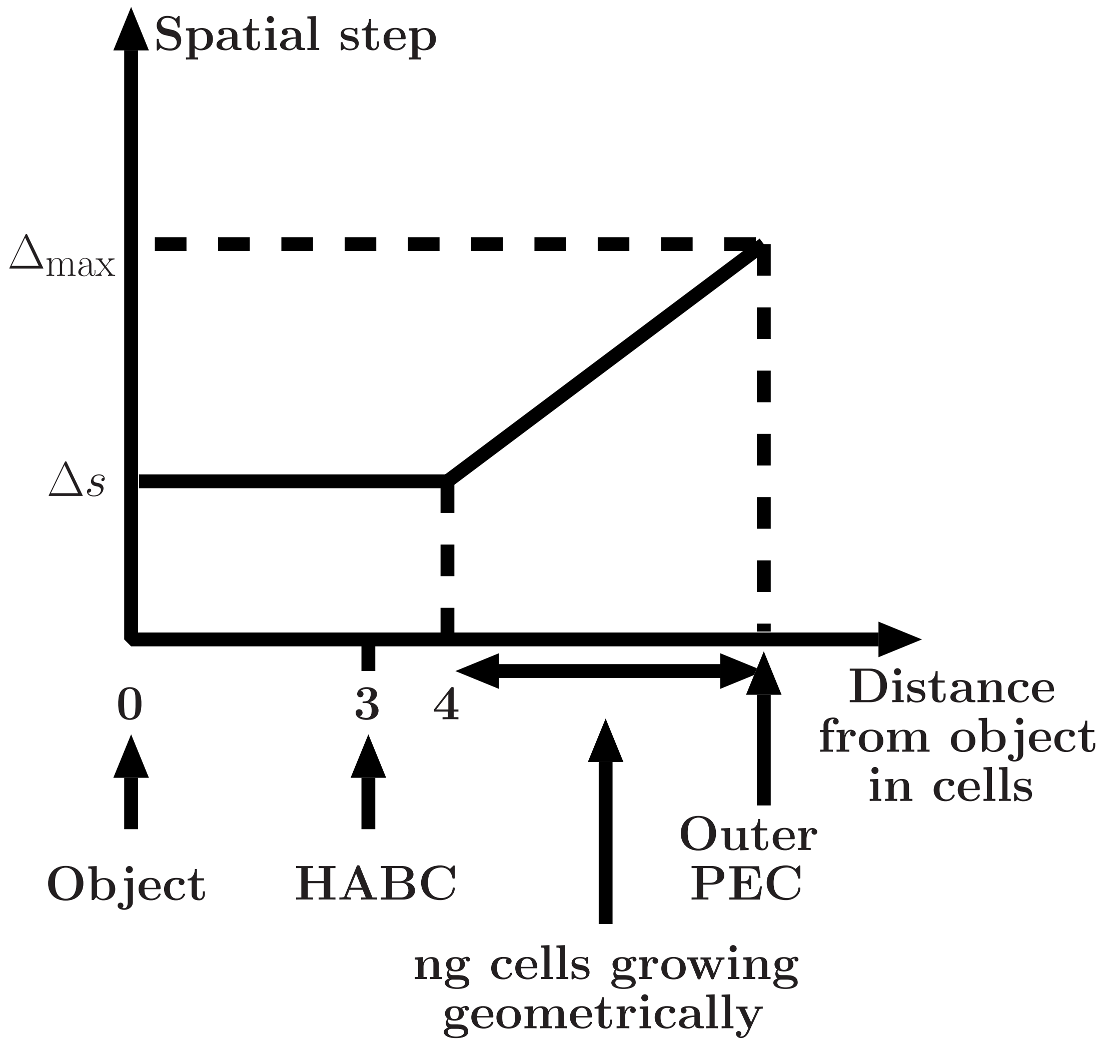


Figure 3.18: HABC and the stretch mesh [4].

The mechanism of applying the SM-HABC in the FDTD domain is shown in Figure 3.18. From the object, the HABC is placed 3 cells away and the SM starts after 4 cells. Those 4 cells that separate the object and the SM are regular non-stretched FDTD cells. Then, starting from the fourth cell from the object, the cells are growing geometrically with ratio g for ng cells up to the maximum cell size (Δ_{max}). The value Δ_{max} is related to the object size w because the characteristic length of the decrease of the evanescent waves is about w [4]. The following formula gives the relation between the number of the stretched cells ng and the corresponding equivalent uniform cells ng_{equ}:

$$ng_{equ} = g\frac{1 - g^{ng}}{1 - g} \tag{3.59}$$

where g is the expansion ratio that satisfies $g = \Delta_{max}^{(1/ng)}$.

3.5.1 Numerical Experiments

In this section, the SM-HABC is tested with the Huygens excitation and an object with dispersive materials. Several choices of ng, the number of stretched cells, are tested to study the ability of small ng in simulating free space.

3.5.1.1 Environmental Settings

In the numerical experiments of SM-HABC, an object with human tissues is tested. As illustrated in Figure 3.19, the outer 10 cells of the 300-cell object are skin, which cover a 280-cell cube of fat. The Debye parameters for skin are $\sigma = 0.54073572$ (S/m), $\epsilon_{\mathrm{S}} = 47.93014336$, $\epsilon_\infty = 29.85054779$ and $\tau_{\mathrm{D}} = 43.6257593$ (ps) and the Debye parameters for fat are $\sigma = 0.03710634$ (S/m), $\epsilon_{\mathrm{S}} = 5.53071189$, $\epsilon_\infty = 3.99812841$ and $\tau_{\mathrm{D}} = 23.6329289$ (ps). $\Delta s = 2$mm and $\Delta t = 3.6$ps, so the size of the 300-cell cube is equal to 60cm. As shown in Figure 3.20, the Huygens excitation surfaces are 1 cell from the object. The incident wave of the Huygens excitation is a Gaussian pulse, which has the following equation:

$$\boldsymbol{E}_{inc}(t) = 100e^{-\left[\frac{t-2.5\cdot10^{-9}}{0.5\cdot10^{-9}}\right]^2}$$

HABC starts 3 cells from the object and SM starts 4 cells from the object. $d_{PEC} = 2w = 600$ and 3,5 and 10 stretched cells are tested in the numerical experiments.

3.5.1.2 Numerical Results

Figure 3.21 shows the observation of the electric field $\boldsymbol{E}$ at the center of the cube. The observed $\boldsymbol{E}$ is parallel to the incident electric field. $d_{PEC} = 2w = 600$ and $ng = 3$,5 and 10. In Figure 3.21, the difference between the results of $ng = 3$ (red curve) and $ng = 5$ (green curve) is bigger than the difference between the results of $ng = 5$ (green curve) and $ng = 10$ (blue curve). However, 2, which is the difference between 3 and 5 is less than 5, which is the difference between 5 and 10. This suggests that the solutions with $ng = 3$,5 and 10 are converging and it is possible to correctly simulate the free space with a small number of cells.

Also, in the case of $ng = 10$ and $g = 1.74$, E_x on 2D-plane is observed when y is constant and x and z start from 1 to the end of the domain as shown

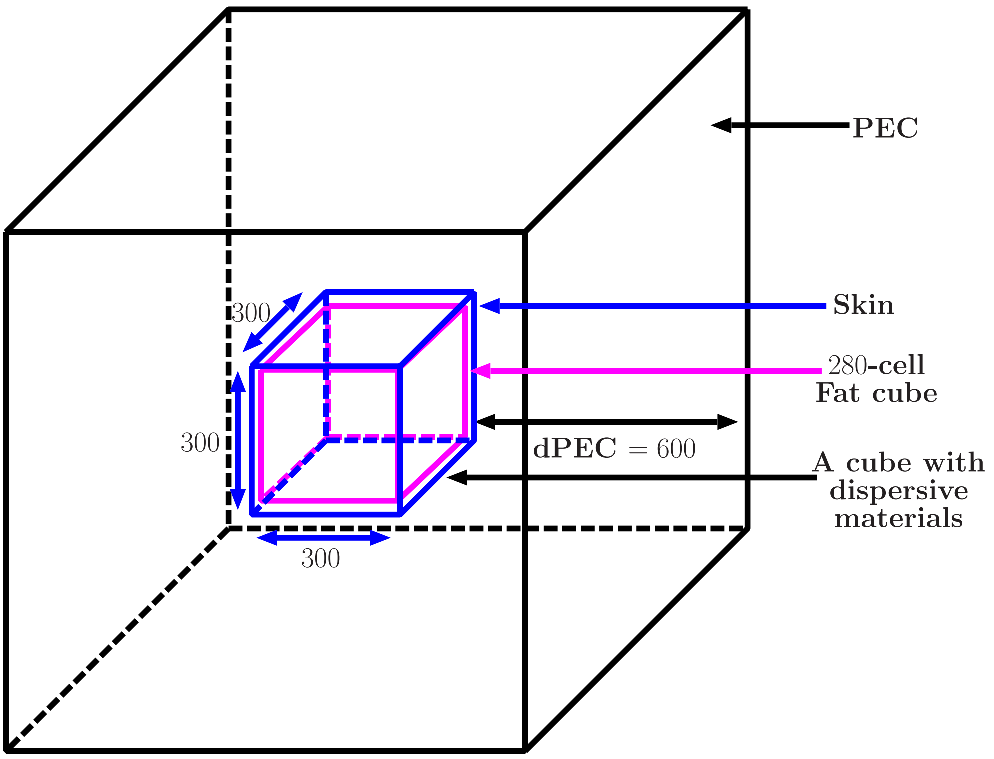


Figure 3.19: The SM-HABCA is tested with a 280-cell cube of fat covered with 10 cells layer of skin. The Debye parameters for fat are $\sigma = 0.03710634$ (S/m), $\epsilon_{\mathrm{S}} = 5.53071189$, $\epsilon_\infty = 3.99812841$ and $\tau_{\mathrm{D}} = 23.6329289$ (ps) and the Debye parameters for skin are $\sigma = 0.54073572$ (S/m), $\epsilon_{\mathrm{S}} = 47.93014336$, $\epsilon_\infty = 29.85054779$ and $\tau_{\mathrm{D}} = 43.6257593$ (ps).

in Figure 3.22. In Figure 3.23, Figure 3.24, Figure 3.25, Figure 3.26, Figure 3.27, Figure 3.28 and Figure 3.29 snapshots of the 2D-plane observations shown at $t = 500\Delta t$, $t = 600\Delta t$, $t = 700\Delta t$, $t = 850\Delta t$, $t = 950\Delta t$, $t = 1050\Delta t$, $t = 1150\Delta t$, $t = 1250\Delta t$ and $t = 1350\Delta t$, respectively. The colored 2D-plane observations show that the wave propagates faster around the cube than inside the cube. Thus, the wave will get into the object not only from the bottom, but also from the other sides of the object.

Figure 3.30 shows the reflection coefficient of the HABC and CFS-PML ABCs.

3.5.1.3 Conclusion

Since the HABC can be placed close to the object in the FDTD domain, it is possible to combine the HABC with other ABCs. In this thesis, the HABC is combined with the SM ABC. This is an effective combination because the HABC

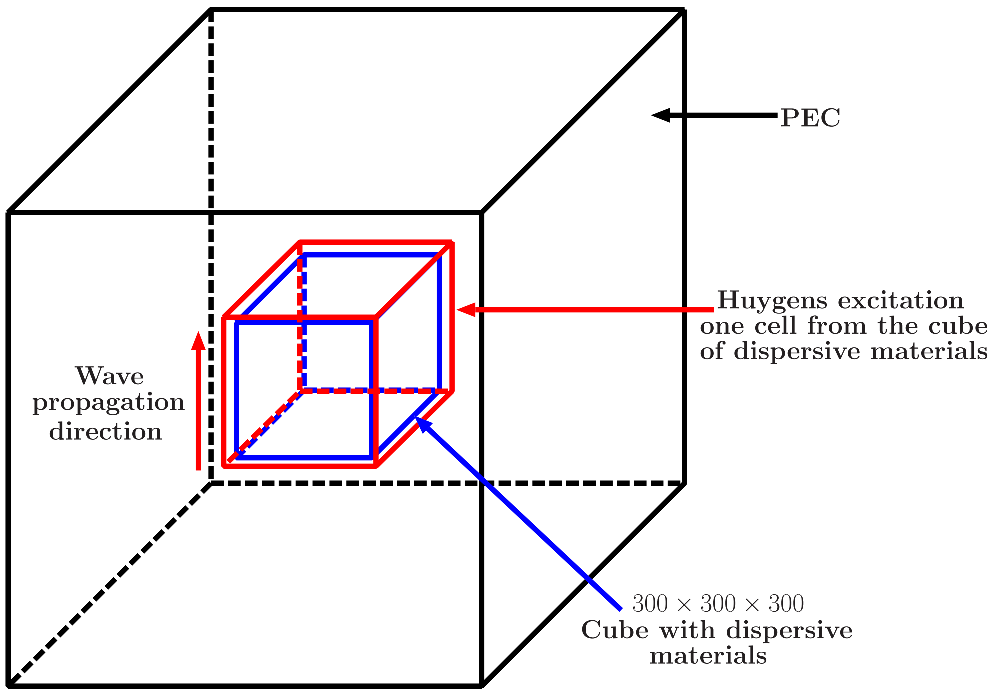

PEC
Huygens excitation one cell from the cube of dispersive materials
Wave propagation direction
$300 \times 300 \times 300$
Cube with dispersive materials

Figure 3.20: A cube with dispersive materials is tested with Huygens excitation and SM-HABC.

absorbs the high frequency traveling waves while the SM ABC absorbs the low frequency evanescent waves. The experiments demonstrate that the SM-HABC can achieve a good level of absorption for both traveling and evanescent waves and suggest that a small number of stretched cells ng can correctly simulate free space when the SM is combined with the HABC.

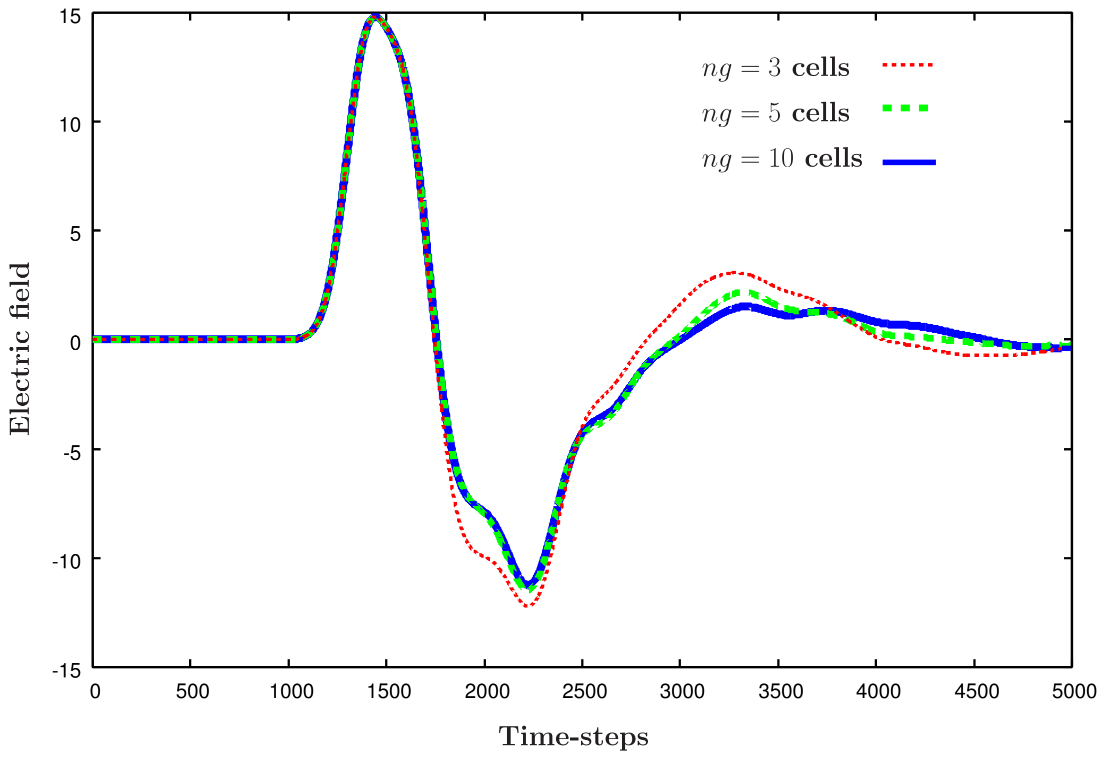


Figure 3.21: Electric field at the center point of a 300-cell cube composed of skin and fat.

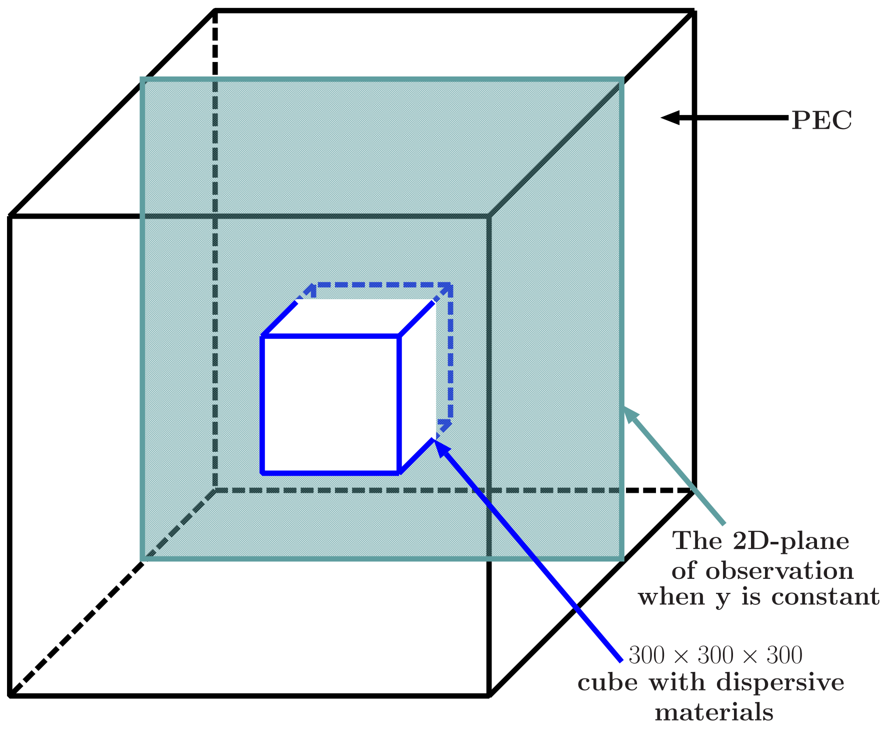

PEC
The 2D-plane of observation when y is constant
300 × 300 × 300 cube with dispersive materials

Figure 3.22: A cube with dispersive materials. 2D-plane is observed when y is constant.

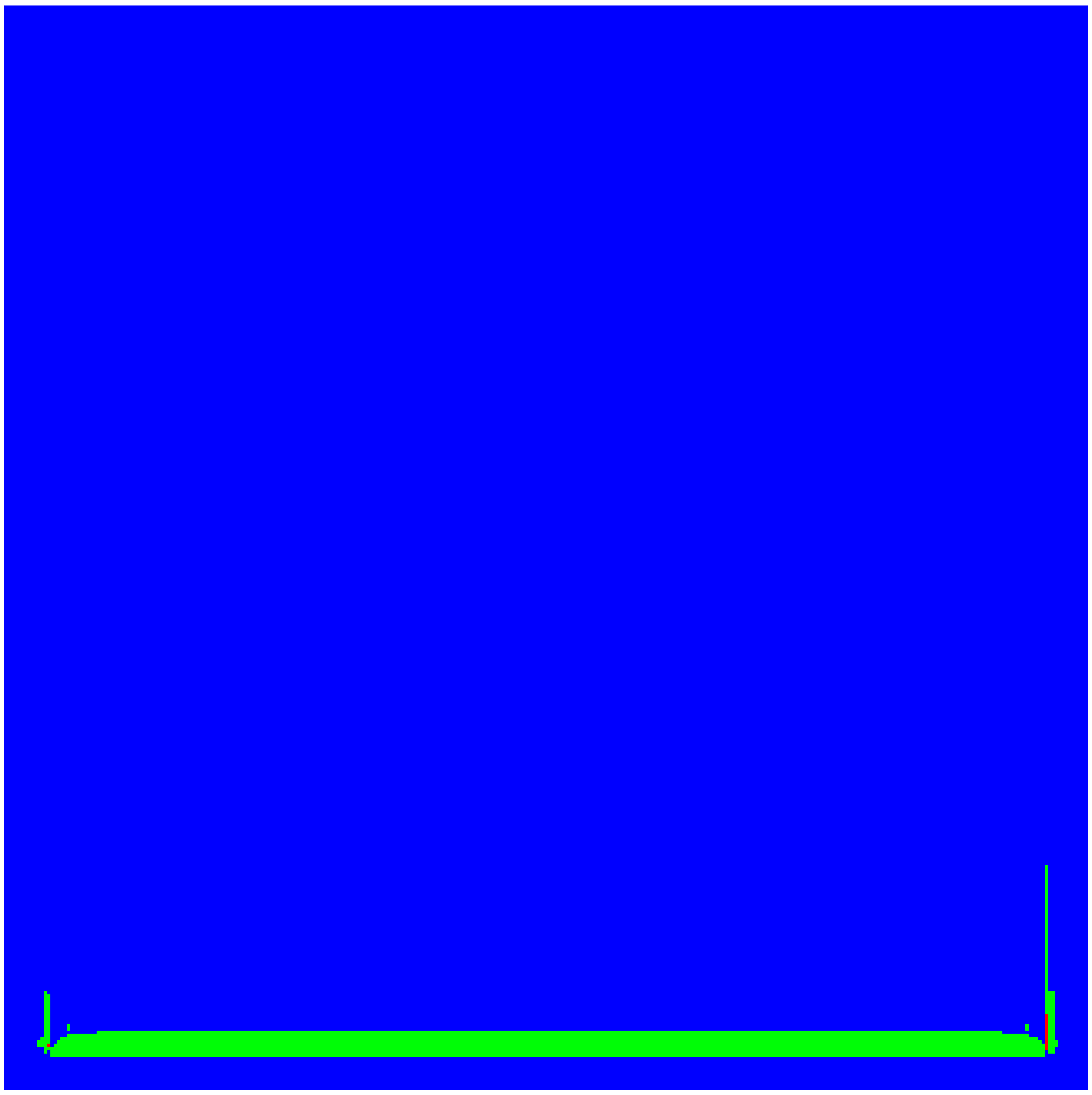


Figure 3.23: A cube with dispersive materials. The colored image of the observed 2D-plane when y is constant at t = $500\Delta t$.

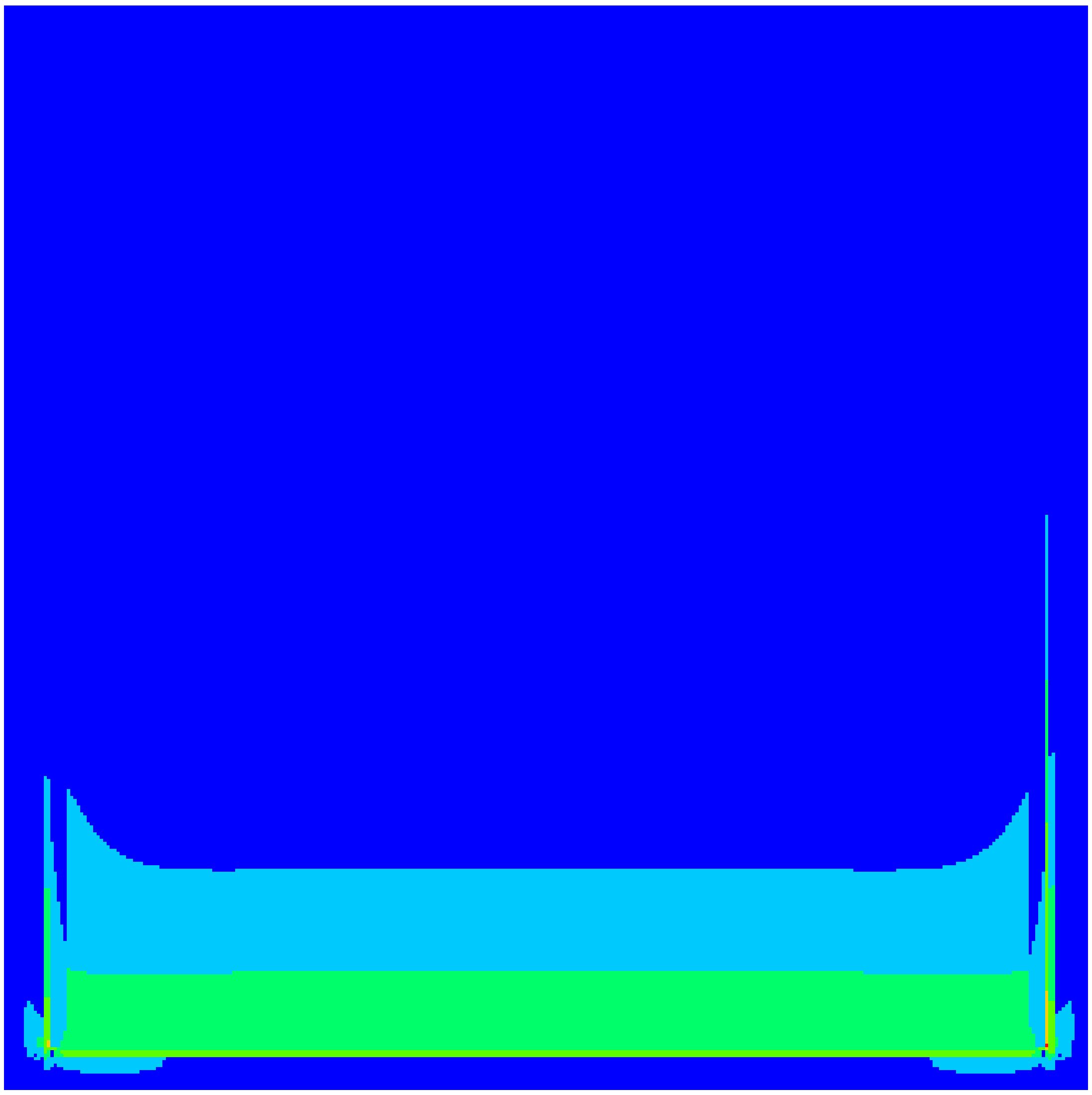


Figure 3.24: A cube with dispersive materials. The colored image of the observed 2D-plane when y is constant at t = $700\Delta t$.

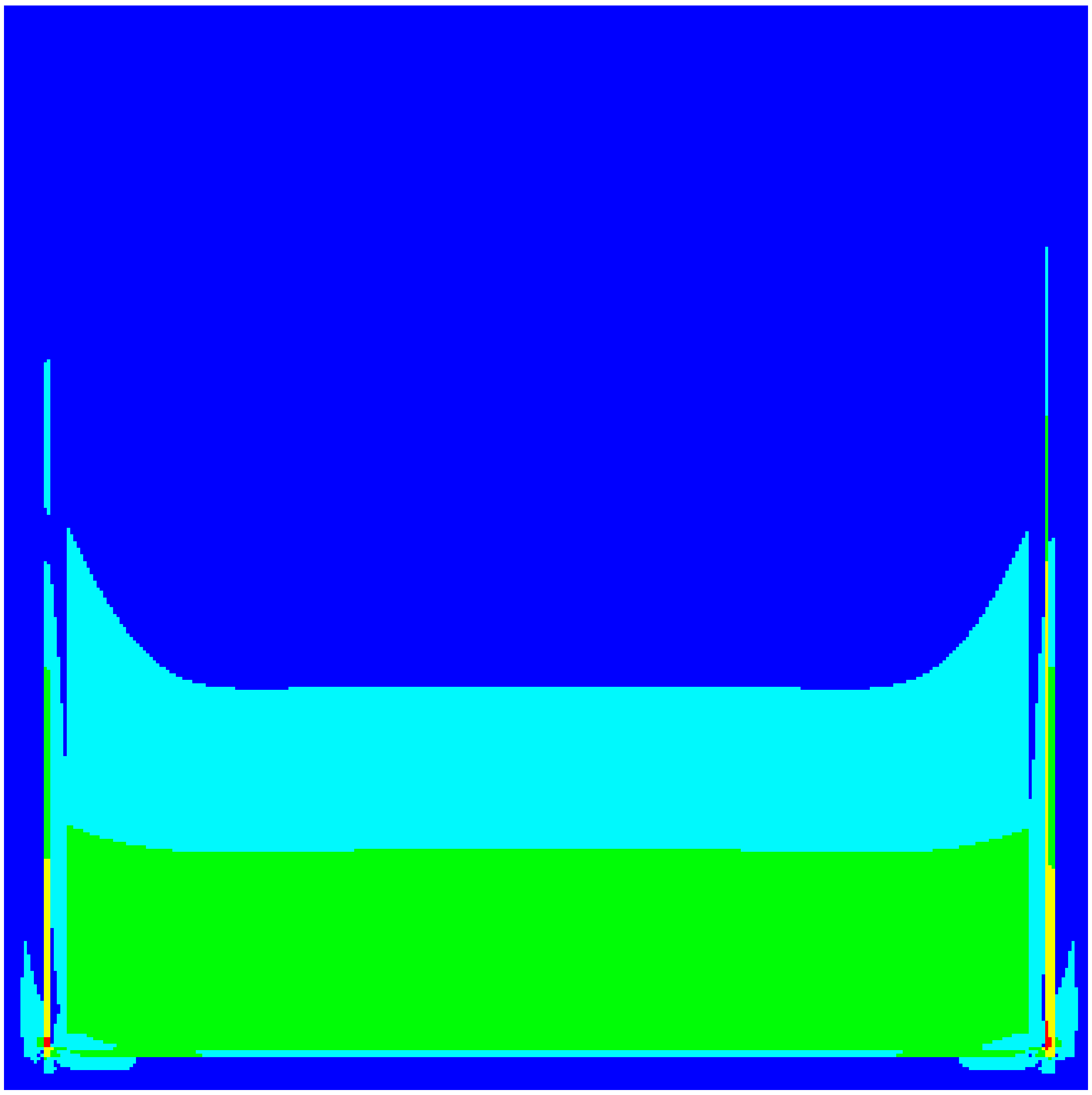


Figure 3.25: A cube with dispersive materials. The colored image of the observed 2D-plane when y is constant at t = $850\Delta t$.

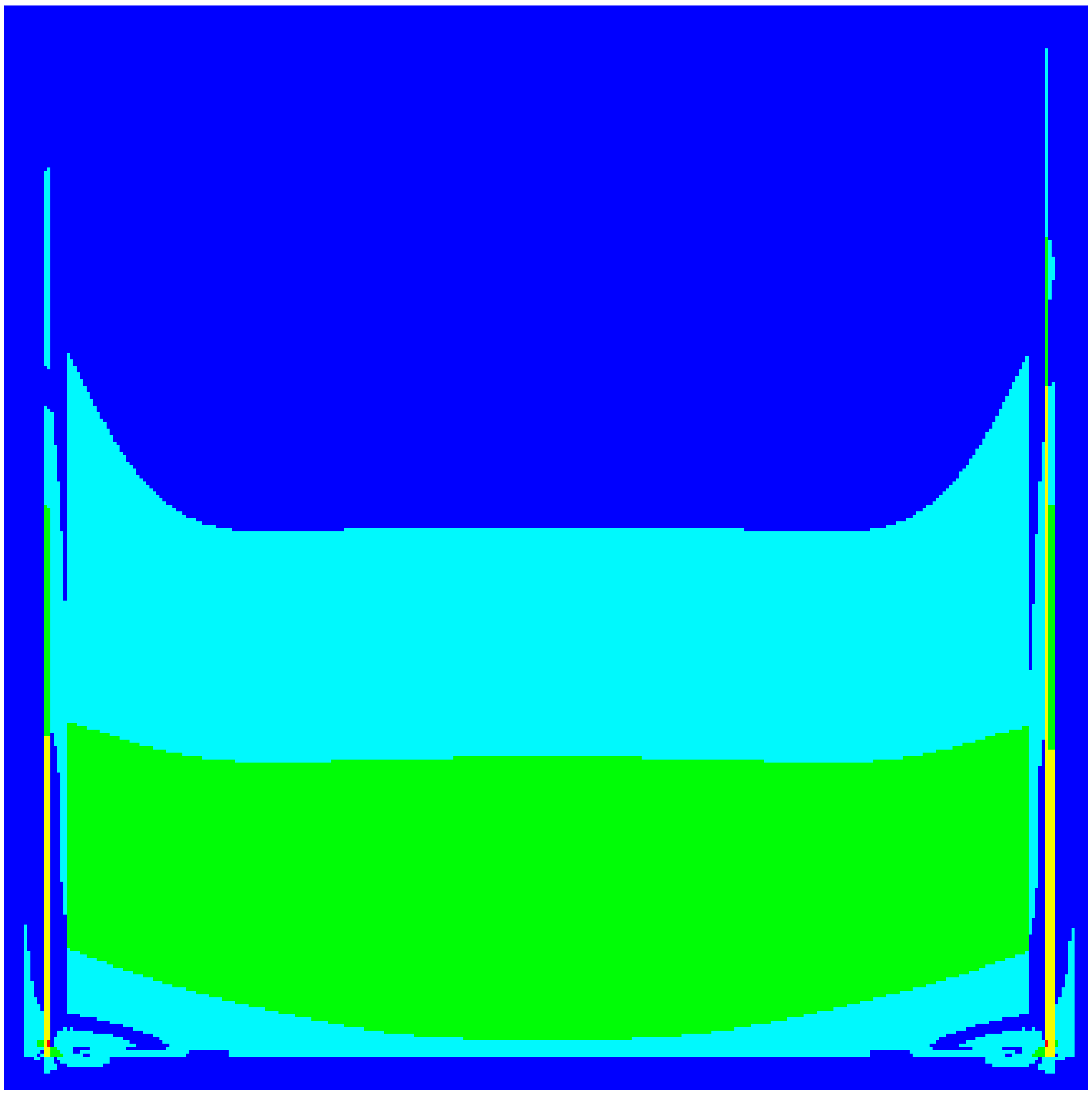


Figure 3.26: A cube with dispersive materials. The colored image of the observed 2D-plane when y is constant at t = $950\Delta t$.

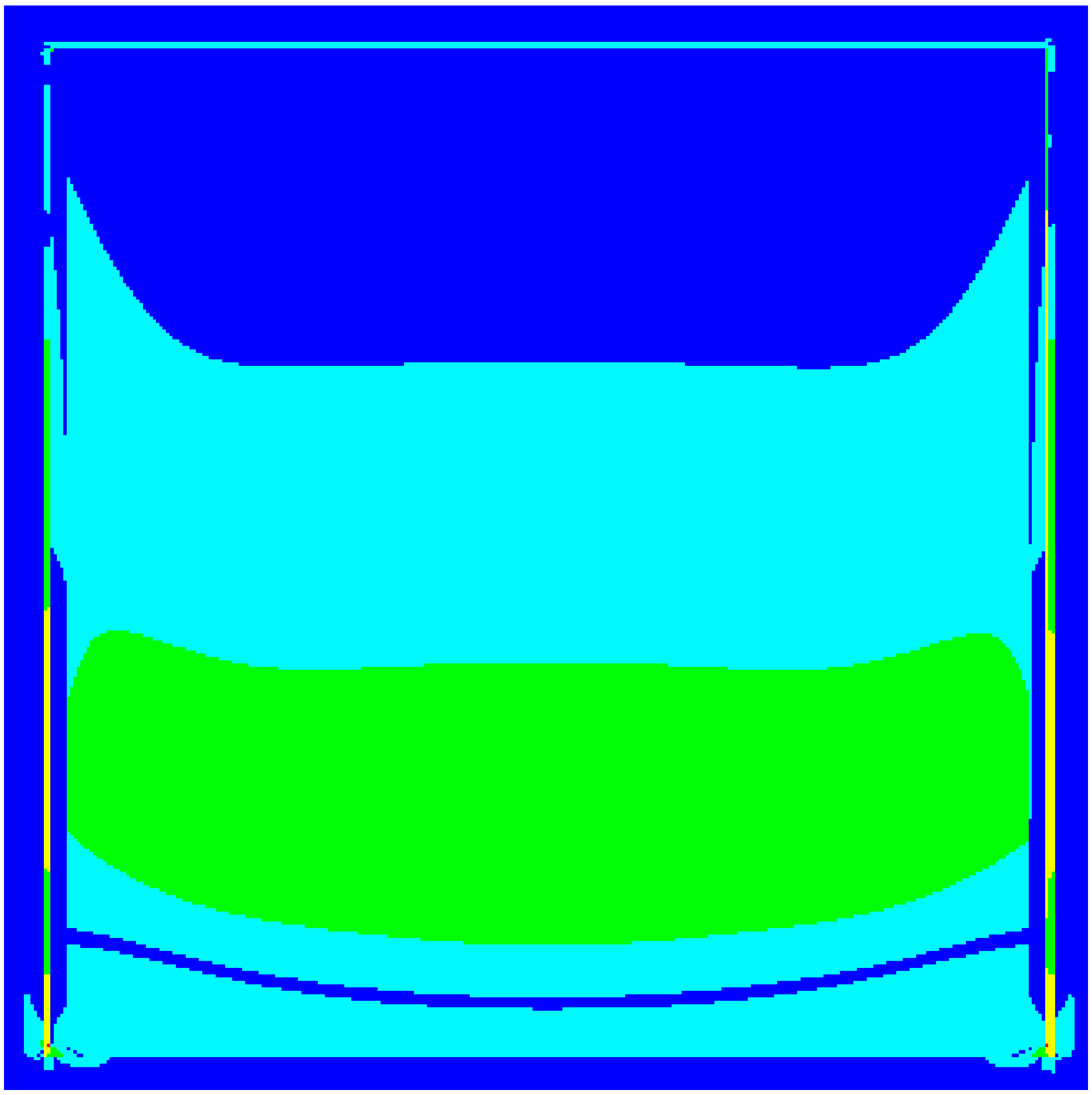


Figure 3.27: A cube with dispersive materials. The colored image of the observed 2D-plane when y is constant at t = $1050\Delta t$.

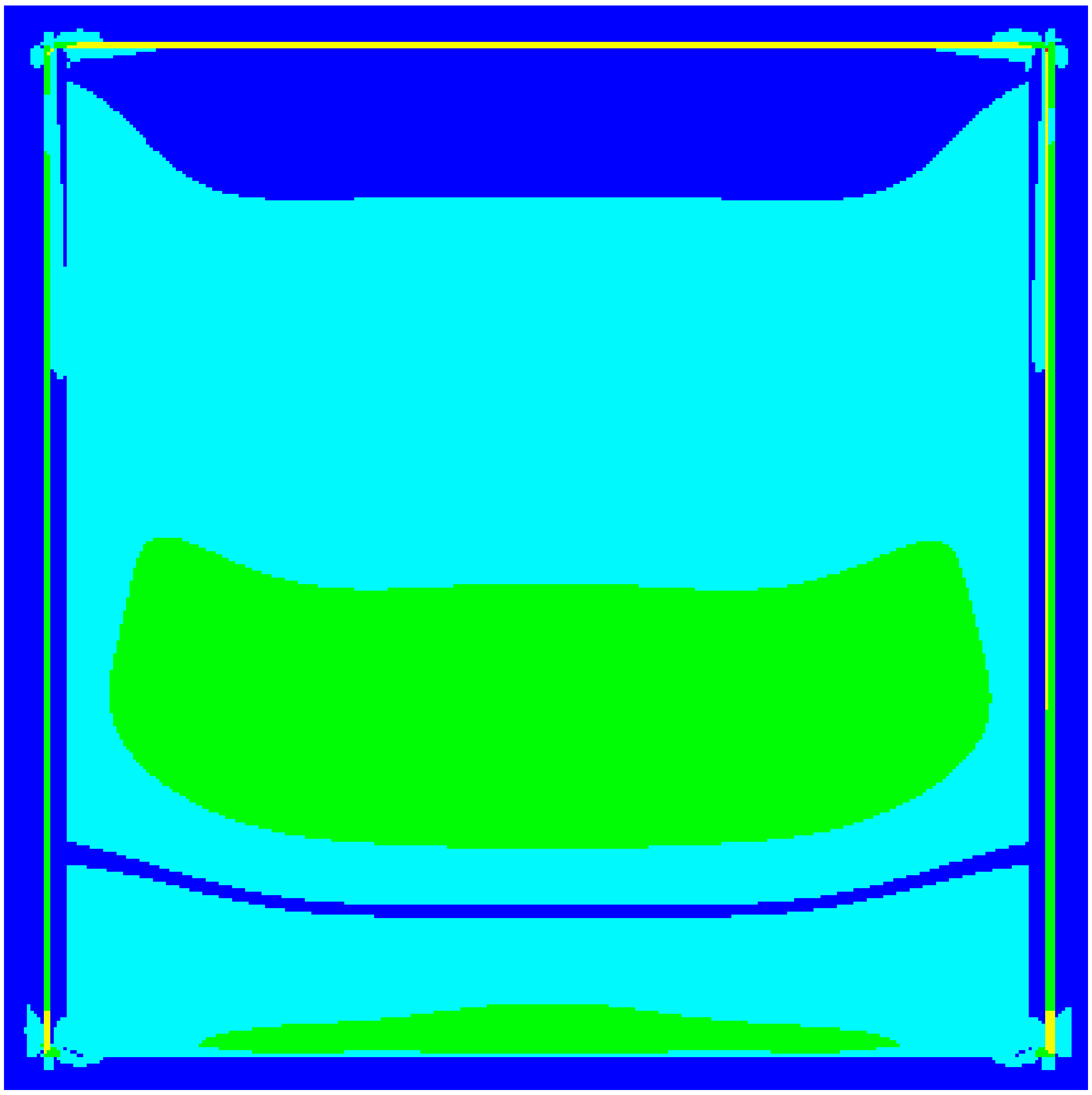


Figure 3.28: A cube with dispersive materials. The colored image of the observed 2D-plane when y is constant at t = $1150\Delta t$.

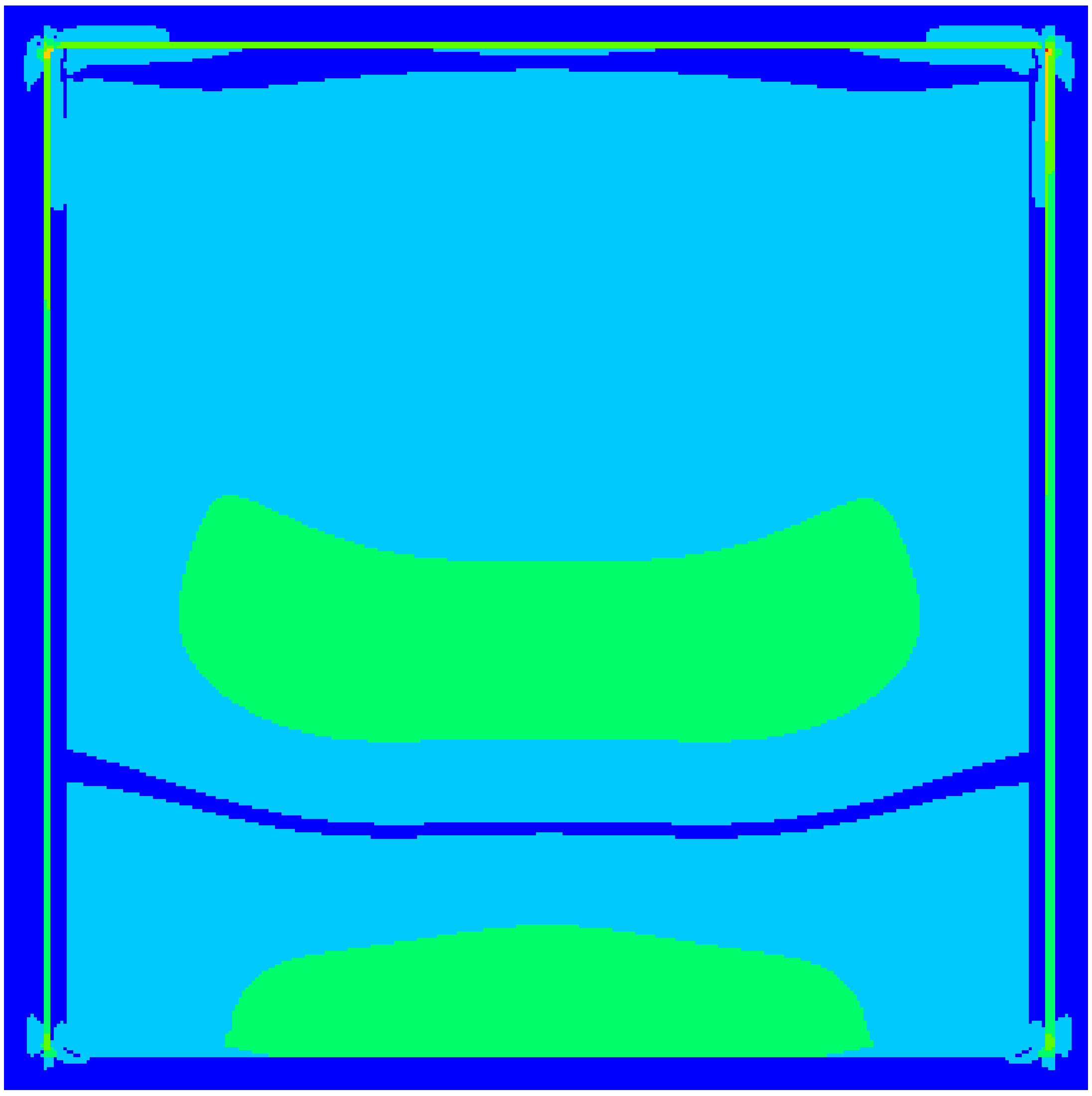


Figure 3.29: A cube with dispersive materials. The colored image of the observed 2D-plane when y is constant at t = $1250\Delta t$.

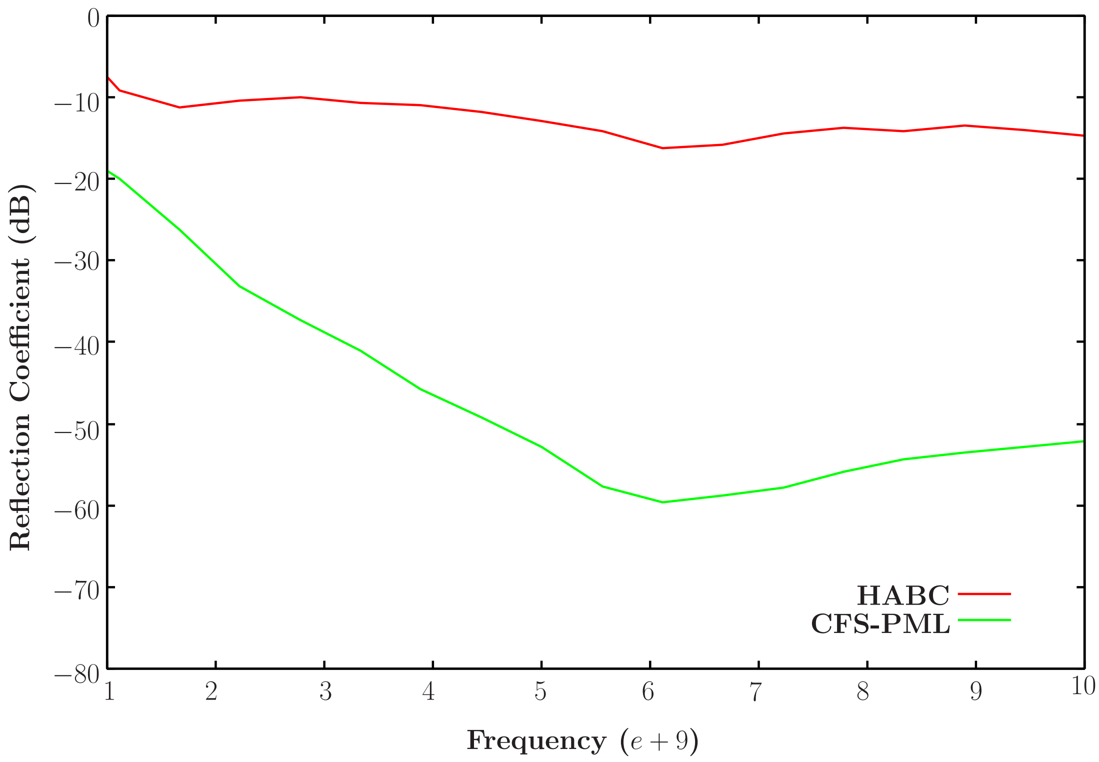


Figure 3.30: Frequency reflection coefficient with the HABC and CFS-PML ABCs.

Chapter 4

High Performance Computing

High performance computing (HPC) is the use of multiple processors to solve significant problems [34]. The simultaneous use of a number of processors in calculating a computational problem is called parallel computing [5]. The computational problem is broken into independent parts to be executed concurrently on different processors. The processors might be of a single computer, a number of computers connected by a network, or a combination of both [5]. HPC is important for practical applications such as the electromagnetic wave propagation in the human body, because one processor can not process a huge amount of data simulation within a realistic time.

In parallel computing, the memory used by multiple processors can have one of the following architectures: shared memory, distributed memory, and hybrid distributed-shared memory [5]. As shown in Figure 4.1, in a shared memory architecture, all processors can access all memory resources as a global memory. Changes in memory by one processor will be visible to all the other processors [5]. Figure 4.2 shows the distributed memory architecture. As the name suggests, there is a local memory for each processor in the distributed memory architecture. Unlike the shared memory architecture, modifications done by one processor to its local memory will not appear in the memory of the other processors. A processor needs to use communication network to access another processor's local memory [5]. Nowadays, the largest and fastest computers in the world use both shared and distributed memory architectures [5]. Such a hybrid distributed-shared memory architecture is presented in Figure 4.3. A node in the hybrid distributed-shared memory architecture contains multiple processors and a shared memory. A communication network is needed to connect the nodes

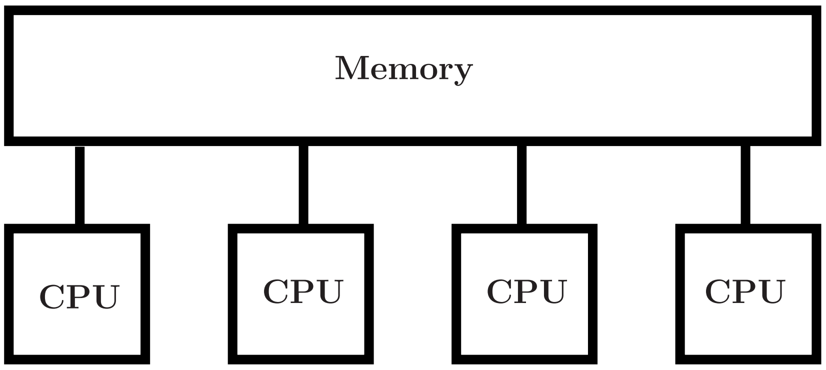


Figure 4.1: Shared memory architecture.

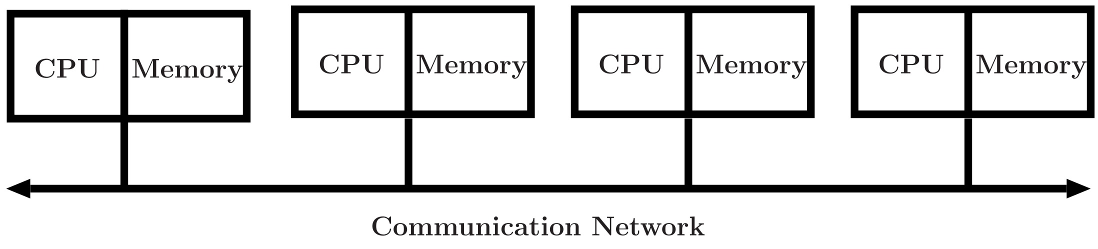


Figure 4.2: Distributed memory architecture.

in the hybrid distributed-shared memory architecture. As shown in Figure 4.3, the graphics processing units (GPU) may be included in the hybrid distributed-shared memory architecture [5]. Introduced by the American global technology company NVIDIA, the GPU is one of the most complex processors to be used together with CPU for accelerating general-purpose scientific and engineering applications [35]. Parallel computing with GPUs has become an industry standard that is used in all HPC supercomputers [36]. For instance, TITAN, the world's fastest open science supercomputer, acquires 90% of its performance from GPUs [37] [38].

4.1 Parallel Computing with GPUs

NVIDIA invented the GPU in 1999 for computer graphics [39]. The availability of programmable pipelines has led to interest in implementing computations for purposes other than graphics on GPUs. Researchers have begun to use GPUs for general-purpose problems but they have had to learn graphics Application Program Interface (API) such as OpenGL to code on GPUs. The use of graphics APIs for general purpose computing led to the concept of General Purpose program-

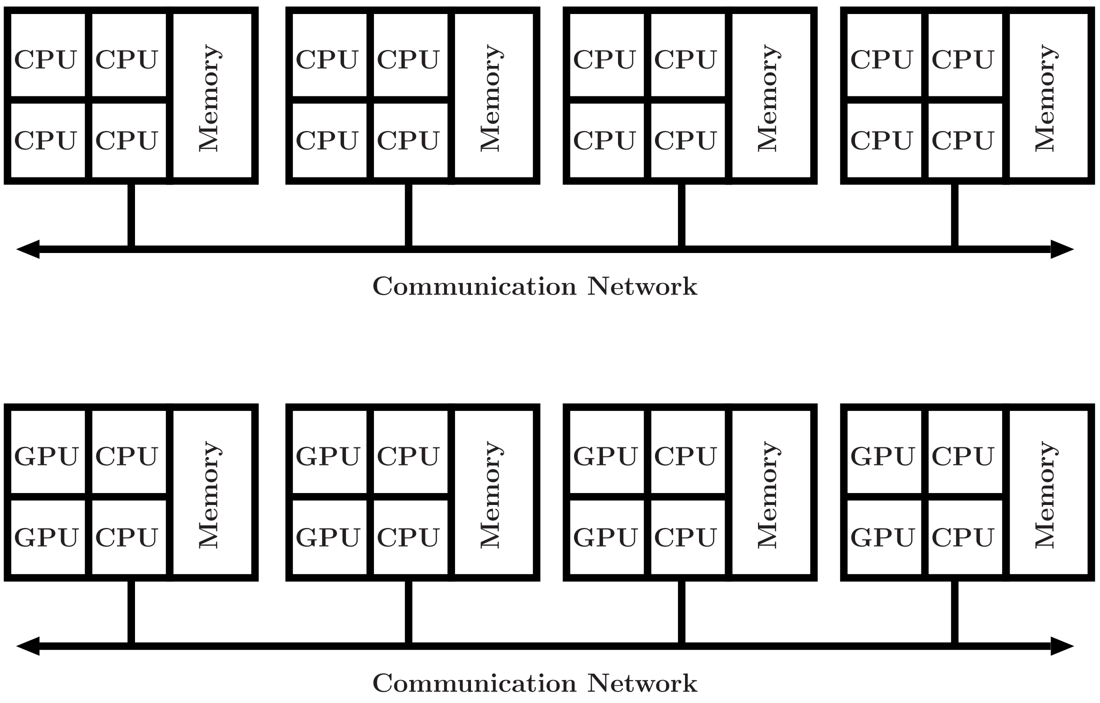


Figure 4.3: Hybrid distributed-shared memory architecture [5].

ming on Graphics Processing Units (GPGPU) [39]. In 2006, NVIDIA introduced the G80 architecture and the Compute Unified Device Architecture (CUDA) [39]. The GPU was then programmed using high level languages since CUDA is an extension of C/C++. The architecture of the GPUs has been improved by including more streaming processors, allocating more space for registers, and increasing the number of threads that could be executed at the same time [39]. Early GPUs only supported single-precision floating-point operations. Since double precision is a primary requirement for scientific calculations, G80 GPUs support double precision but in a limited manner. NVIDIA has introduced a new GPU called Fermi in 2010 [39]. Fermi improved the performance of double-precision floating-point computations. Furthermore, Fermi introduced a true cache hierarchy and increased the size of the shared memory. In 2012, NVIDIA released its latest GPU architecture, named Kepler [40]. Kepler architecture includes a new streaming multiprocessor design that gives a better performance than the streaming multi-processor in Fermi [40]. Furthermore, Kepler has the capability of dynamic parallelism where GPU threads generate new threads without going back to the CPU to update the data [40]. Fermi and Kepler architectures are used in our simulations to examine the performance of the GPGPU implementations of FDTD with the HABC.

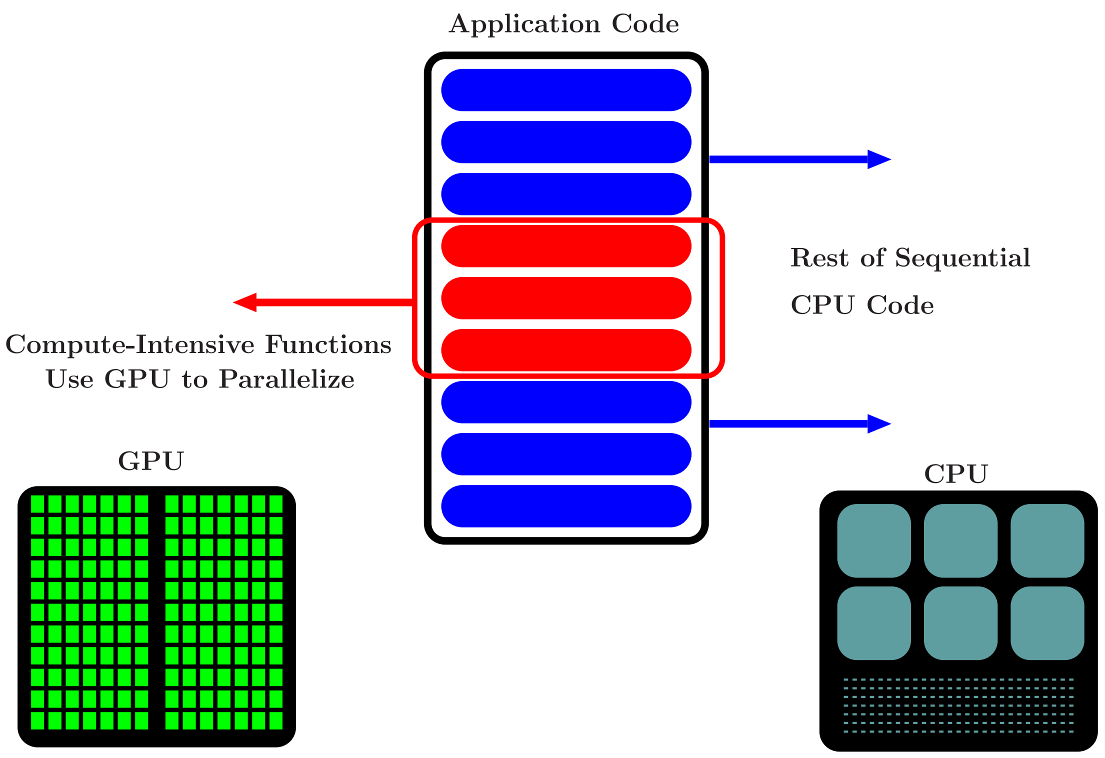
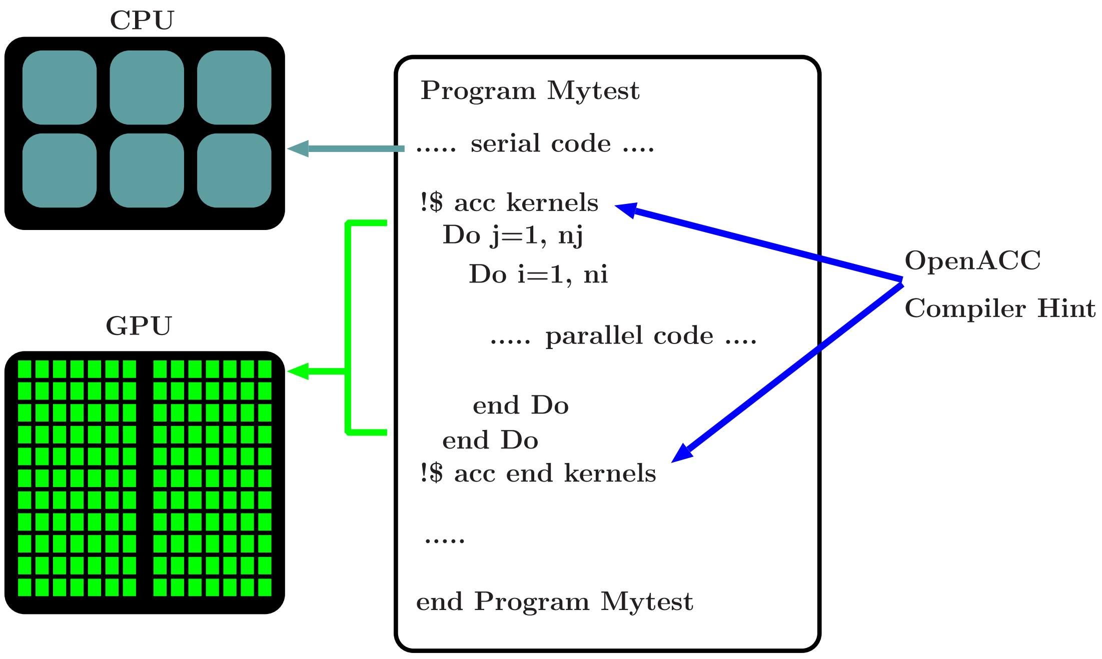

Application Code
Rest of Sequential CPU Code
Compute-Intensive Functions Use GPU to Parallelize
GPU
CPU

Figure 4.4: Parallel computing with GPUs [6].

Figure 4.4 illustrates the main idea of using GPUs along with the CPUs to accelerate an application code [6]. There are three ways to accelerate an application on the GPUs. The first is the use of GPU-accelerated libraries. Using the libraries gives high quality implementations of functions in a wide range of applications [6]. GPU-accelerated libraries are easy to use and do not require a deep knowledge in GPU programming [6]. Those libraries follow the APIs standard, so no big changes will be made to the code so as to accelerate the GPUs. The second way to accelerate the GPUs is GPU computing with OpenACC directives [6]. As shown in Figure 4.5, compiler hints are added to detect which parts of the code should be executed in GPUs. Using OpenACC directives is straightforward and portable across parallel and multi-core processors [6]. Using programming languages for GPU computing is the third way which has the maximum flexibility in accelerating GPUs [6]. There are several programming languages that might be used to code on GPUs such as MATLAB, CUDA Fortran, CUDA C, CUDA C++, PyCUDA and GPU.NET. CUDA Fortran is the key language in HPC for programming GPUs [6]. CUDA Fortran has a familiar syntax, as in using allocate



```
Program Mytest
..... serial code ....
!$ acc kernels
  Do j=1, nj
    Do i=1, ni
      ..... parallel code ....
    end Do
  end Do
!$ acc end kernels
.....
end Program Mytest
```

Figure 4.5: GPU computing with OpenACC directives [6].

and deallocate for the arrays and copying from CPU to GPU or from GPU to CPU with assignment (=) [6]. There are simple language extensions in CUDA Fortran that are used in kernel functions, thread / block IDs, device and data management, and parallel loop directives [6]. We used CUDA Fortran (third way) to program the FDTD with the HABC code on GPUs.

4.2 Parallel Computing with Shared Memory

Recently, a single processor is constructed using independent processing units. The processing units are known as cores and are used to increase the speed of the processor. A processor with multiple cores is called a multi-core processor. In a shared memory architecture, all cores can access any memory location. Accessing the shared main memory by the multi-cores needs to be controlled by the programmer [5]. The speed of accessing the shared memory is the same for all cores but the amount of data transfer is restricted to the maximum memory transfer rate, denoted as bandwidth.

In parallel computing, the parallel unit of the program is split into parts called threads to be executed concurrently on different cores. All threads can access the shared memory address, and the communication between the threads is implicit[7]. The programmer has the ability to explicitly define private data for any thread that is not accessible by the other threads, as illustrated in Figure 4.6.

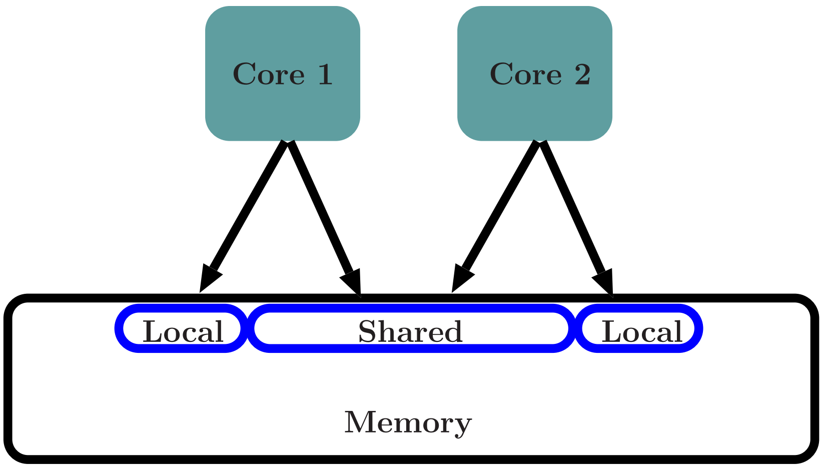


Figure 4.6: Threads access to local and shared data in shared memory architecture [7].

As shown in Figure 4.7, the program runs as a master thread on the processor at the beginning. Then, when the program reaches the parallel part, the master thread spawns multiple threads. For instance, if the processor contains n cores, n threads can run simultaneously which improves the computational speed. The master thread waits unit all threads have finished running and then eliminates them. The program runs the serial part as a master thread after completing the parallel part. This process might be repeated depending on the number of parallel parts in the program.

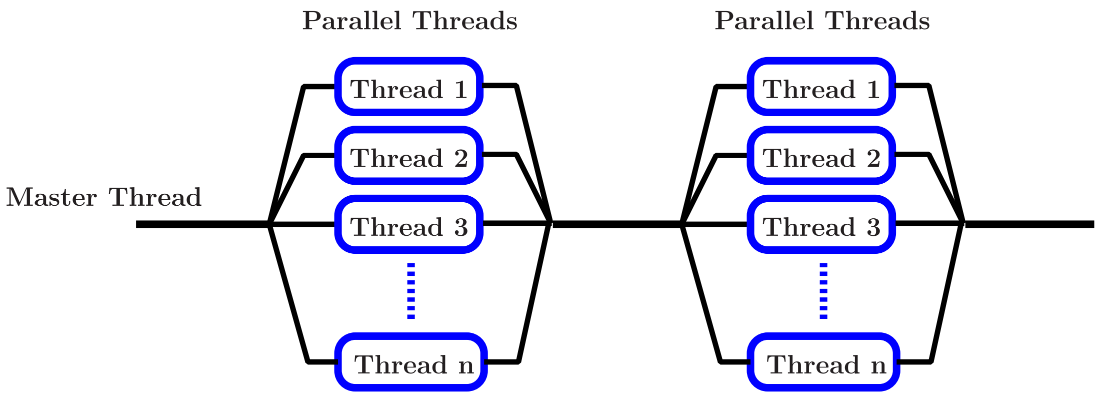


Figure 4.7: Threads in parallel computing with shared memory architecture [7].

Open Multi-Processing (OpenMP) is an API specified for shared memory parallelism [41]. OpenMP was introduced in 1997, so the compilers are relatively advanced [41]. OpenMP is an extension of C/C++ and Fortran, which makes it easy to parallelise an existing code. Most compilers support openMP such as GNU, IBM, Intel, and PGI. OpenMP is the most popular option for multi-threaded, shared memory parallelism [41]. Since using openMP for shared memory has the advantages of standardization, ease of use, and portability [41], Intel intends to produce a new technology with an intensive number of processors based on shared memory architecture. In our application of the FDTD with the SM-HABC on the whole human body, we used openMP to accelerate the code on shared memory architecture.

Chapter 5

Parallelisation on Graphics Processing Units

General Purpose computing on Graphics Processing Units (GPGPU) is becoming popular in scientific computation. The GPU includes many processing units to execute different parts of the code concurrently. To effectively use the GPU in parallel computing, the computational problem should have a high degree of data parallelism [10].

The data in the FDTD method and the HABC is stored as multidimensional arrays. The multidimensional arrays are calculated through tens or hundreds of thousands of time-steps. At each time-step, each item of the multidimensional arrays is calculated independently and requires only one-cell neighbours, which makes the FDTD method with the HABC good applications for parallel computing on GPUs [10].

5.1 General Purpose GPU Computing with CUDA

Tesla architecture is a GPU hardware design invented by NVIDIA. As shown in Figure 5.1, Tesla architecture contains an array of general-purpose, streaming multi-processors [8]. Also, NVIDIA released the Compute Unified Device Architecture programming model (CUDA) to code on Tesla architecture [8] [42] [9]. CUDA is used to indicate which parts of the code should be run on the GPUs. The parallel sections of the program that are executed on the GPUs are called kernels [8].

To make use of the general-purpose, streaming multi-processors in a GPU, the

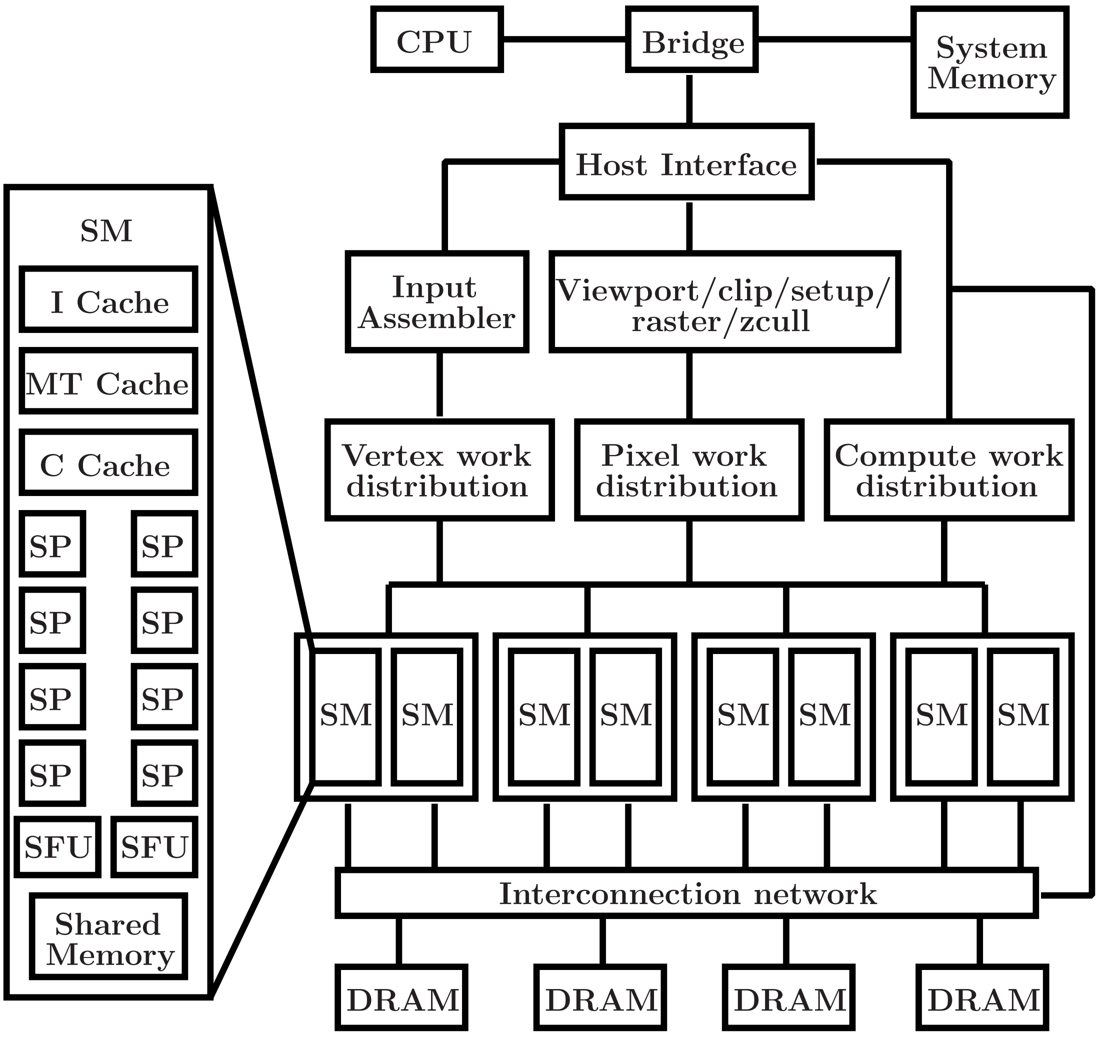


Figure 5.1: The Tesla unified graphics and computing GPU architecture [8].

application should have a massive parallelism [10]. A kernel is used to calculate a grid of threads. As illustrated in Figure 5.2, the grid has one or two dimensions of blocks [43]. Each block is broken into one, two or three dimensions of threads [42]. CUDA allows the programmer to identify each thread based on its indices using the built-in parameters. For instance, the CUDA parameters `blockIdx%x` and `blockIdx%y` indicates which block within a grid contains the thread, and the parameters `threadIdx%x`, `threadIdx%y` and `threadIdx%z` specify the location of the thread with respect to a block [10]. Threads in a single block are grouped into wraps. A wrap usually contains 32 threads and executes on a single Streaming Multiprocessor (labeled SM in Figure 5.1) with each thread runs on different streaming processor core (labeled SP in Figure 5.1). The mechanism of wraps can

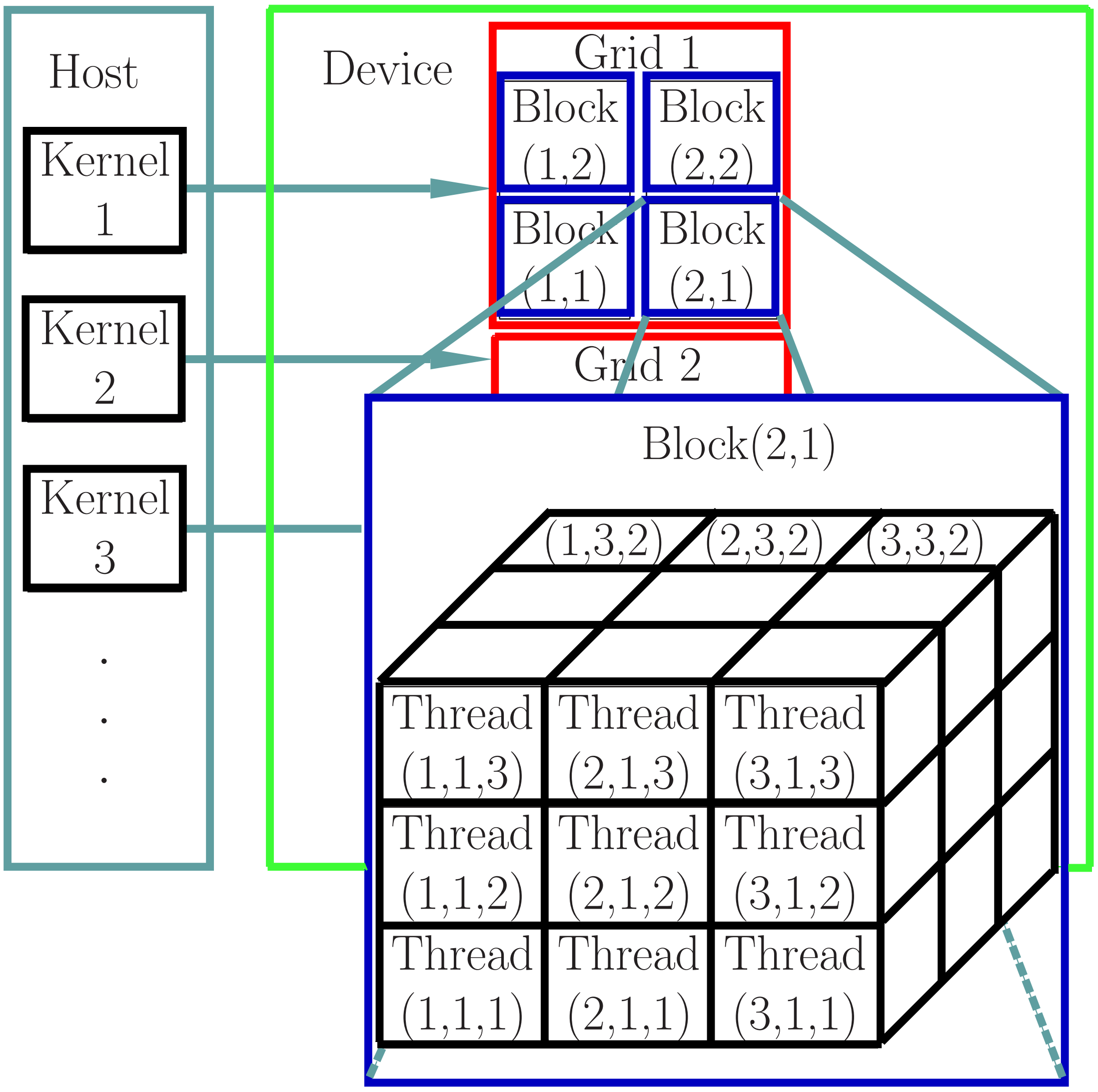


Figure 5.2: CUDA grid organisation in Tesla [9].

improve the performance by hiding latency [9]. This is done by running threads of another wrap while waiting for results of high-latency operation. Thus, the waiting time is hided by the execution of another wrap. Theoretically, there will always be a running wrap [44]. Krik and Huw named it zero-overhead thread scheduling [9].

In CUDA programming system, the term host refers to the CPU, and the term device represents the GPU [9]. Each of the host and device has its own memory space. Figure 5.3 shows CUDA device memory for Tesla architecture. Host memory is implicitly included in the host in Figure 5.3 [9]. The host and the device can read and write to the global memory shown at the bottom of Figure

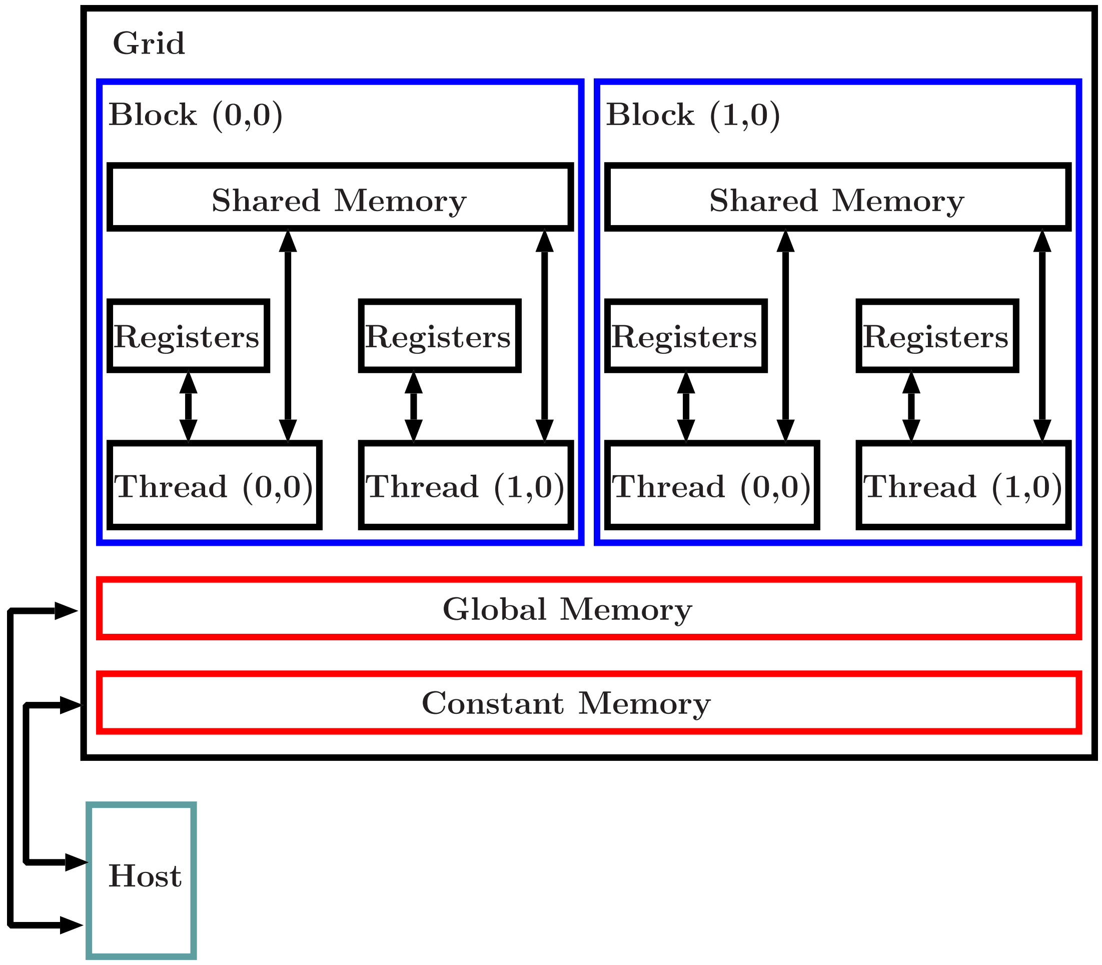


Figure 5.3: CUDA device memory model [9].

5.3. The lifetime of the global memory is the application. When the program starts, the data is transferred from the host memory to the device global memory. And when the program finishes running, the data is moved back from the device global memory to the host memory. All threads in all blocks in a grid can access the large global memory [9]. To improve the performance of global memory access, the "memory coalescing" [9] concept is used. When a group of threads in a single block are concurrently executing and accessing consecutive memory locations, their memory requests can be merged to a single larger memory request. This single larger fetch of consecutive block of memory leads to an efficient use of the memory bandwidth [9] [10].

Like the global memory, the constant memory shown in Figure 5.3 lasts until the end of the application. However, only the host can read and write in constant memory, it is a read-only memory for the device [9]. Host can move data from and to device memory through the global memory and the constant memory as demonstrated by the bidirectional arrows between the host and the device in Figure 5.3 [9].

The shared memory can be accessed by all threads in a single block as illustrated in Figure 5.3. Threads cannot read or write to the shared memory of another block [9]. The shared memory has a lifetime of a kernel. Since the shared memory is an on-chip memory for the device, it is faster to access than the global memory [9]. The register in 5.3 is an on-chip memory assigned to each thread. The register allows a very fast access [9]. Threads can read and write on the registers allocated to them but the host cannot access those registers [44]. The data stored in the register is scalar (not arrays) and private so each thread will have different version of the scalar data [9]. The lifetime of the register is the kernel, which means that the data will not be preserved when the thread ends [9]. Thread's private array variables are stored in the local memory. The space of the local memory is reserved to a thread but stored in the global memory [9]. As for the registers, the local memory has the lifetime of a kernel. However, the speed of accessing the local memory is lower than the speed of accessing the register, since the local memory is located in the global memory, which is an off-chip memory [44].

In CUDA computations, the programmer can explicitly specifies the type of memory used for the data. For instance, the prefixes 'constant', 'shared' and 'device' can be added for a variables in constant memory, shared memory and global memory respectively [9].

5.2 Existing Implementations of the FDTD Method on GPU Hardware

There are three main approaches to map the three dimensions of the FDTD domain to the allocation of blocks and threads [10]. This Section presents those three approaches introduced in [10].

5.2.1 First approach

The first approach splits the FDTD space into 2D planes. Each plane is calculated using separate kernel invocation. The kernel invocations sequentially

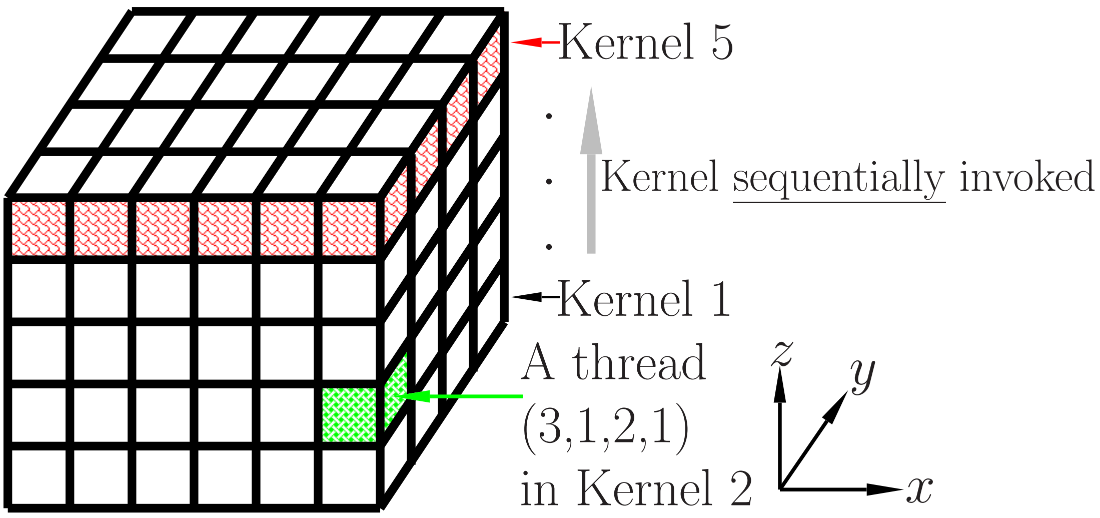


Figure 5.4: The 3D FDTD space decomposition in the first approach [10]. (a, b, c, d) means (`blockIdx%x`,`blockIdx%y`,`threadIdx%x`, `threadIdx%y`)

compute the 2D planes one after the other at each time step. Within a single kernel, the 6 FDTD equations calculate the 2D plane concurrently. The kernel in the first approach consists of 2D blocks with the standard CUDA parameters `blockIdx%x` and `blockIdx%y`. Each block is divided into threads in 2D with the standard CUDA parameters of `threadIdx%x` and `threadIdx%y` [10]. Figure 5.4 illustrates the mechanism of the first approach in implementing a 6 x 4 x 5 dimensional FDTD space on GPUs. A 2D plane from the 3D FDTD domain shown in Figure 5.4 is presented in Figure 5.5 based on the first approach. Each cell of the FDTD space is calculated using a single thread. The coordinates i and j of that thread can be calculated as `blockIdx%x` $\times$ `blockDim%x` $+$ `threadIdx%x` and `blockIdx%y` $\times$ `blockDim%y` $+$ `threadIdx%y` respectively [10]. Although the first approach permit very fine-grained decomposition so that all of the parallelism of a single plane is presented, it does not take the advantage of the fact that all calculations in all planes are independent [10].

5.2.2 Second approach

In the second approach, shown in Figure 5.6, only one kernel invocation is used to calculate the whole 3D FDTD space. The organization of the blocks and threads in the second approach is the same as in the first approach. As shown in Figure 5.5, the blocks and threads perform 2D planes that are mapped to two of the three FDTD dimensions. All the items in the third dimension are calculated in

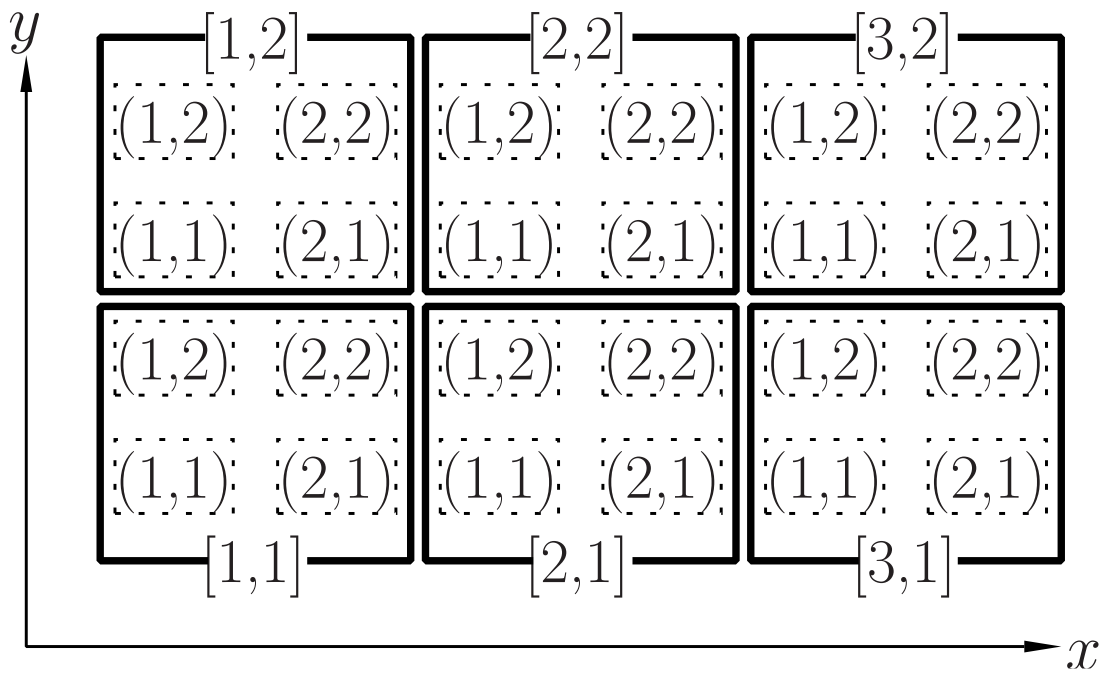


Figure 5.5: The 2D decomposition in the first and second approaches when using blocks and threads on a k = constant plane. [10].
$[a, b]$ and (a, b) mean [`blockIdx%x`,`blockIdx%y`] and (`threadIdx%x`, `threadIdx%y`) respectively.

a `for` loop by a single thread since only one kernel invocation is used in this approach [10]. For instance, in Figure 5.5, the 2D plane is mapped to the x and y dimensions of the FDTD space. Thus, all the items in the z dimension are computed by a single thread.

The second approach uses much less kernel invocations than the first approach and the work done by a single kernel is so heavier in the second approach. However, those differences between the first and the second approaches do not affect the amount of parallelism at any point in time [10].

5.2.3 Third approach

In the third approach demonstrated in Figure 5.7, the whole 3D FDTD domain is calculated using a single kernel invocation [10]. The third approach allocates the blocks in two dimensions that are mapped to two of the three dimensions of the FDTD domain. The threads in each block are allocated in one dimension, which is mapped to the remaining third dimension of the FDTD space [10]. For instance, in Figure 5.7, the x and y dimensions of the FDTD space are mapped

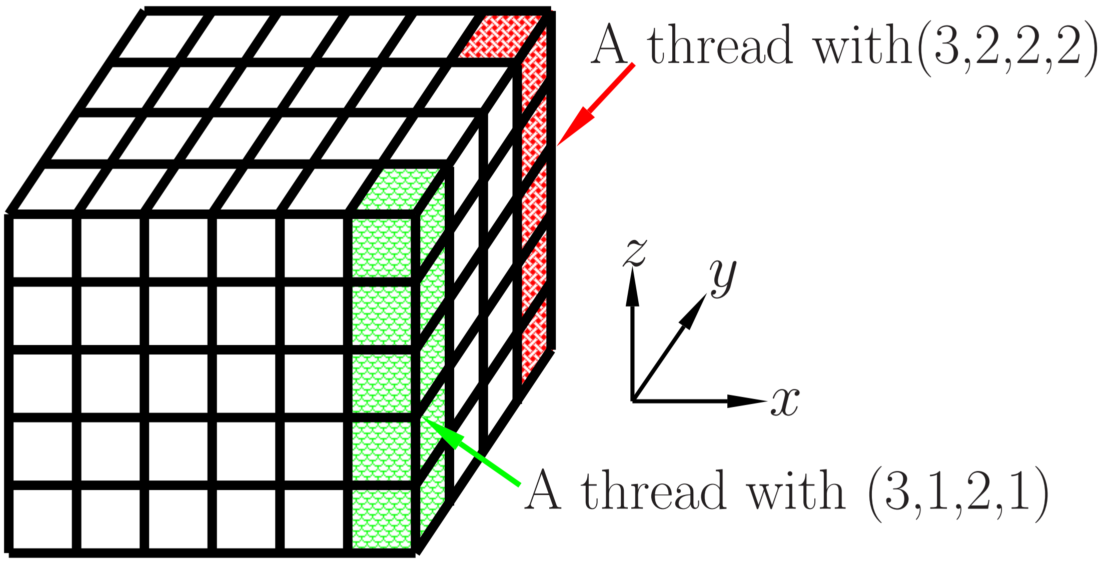


Figure 5.6: The 3D FDTD space decomposition in the second approach [10]. (a, b, c, d) means (`blockIdx%x`, `blockIdx%y`, `threadIdx%x`,`threadIdx%y`).

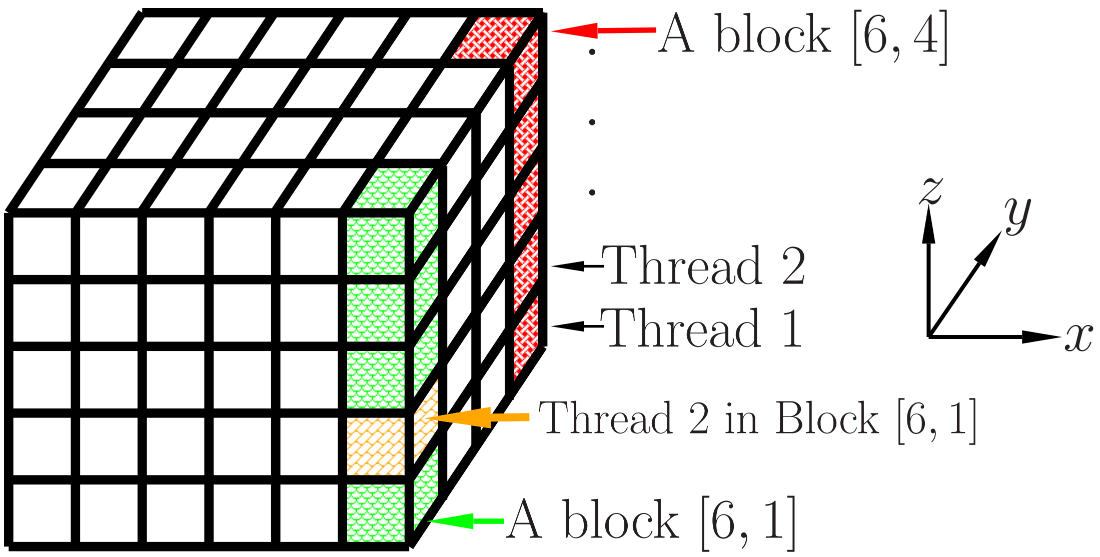


Figure 5.7: The 3D FDTD space decomposition in the third approach [10]. $[a, b]$ means [`blockIdx%x`, `blockIdx%y`].

to the x and y indices of the 2D blocks. And the z dimension of the FDTD space is mapped to the x index of the 1D threads. Thus, a thread is assigned to each cell in the FDTD space, which uses the maximum available parallelism in the FDTD method [10].

5.3 The implementation of the FDTD method on GPU hardware

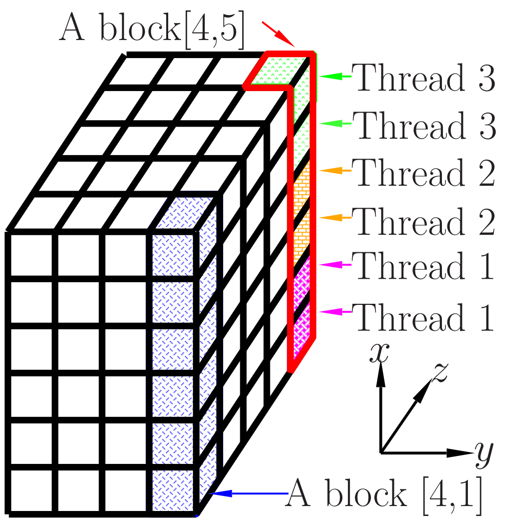
This thesis uses the method introduced in [10] which is based on the third approach but provides more flexibility in terms of the amount of work assigned to each thread.

The builtin data type `dim3` is used to define the dimensions of the blocks and threads. `dim3` contains 3 elements: x, y and z. If one of the elements is not assigned to a value, it will be initialized to 1 [45]. As illustrated in Listing 5.1, two `dim3` variables are used in the kernel invocation. The first `dim3` variable initializes the dimensions of the blocks to the z and y dimensions of the FDTD domain. However, in this method, the dimension of the threads is not assigned to the third dimension of the FDTD domain, x, because each thread calculates one or more cells in the FDTD space. In other words, the threads within a block perform the calculations for the cells in a single dimension as shown in Figure 5.8 [10]. For instance, if N_x, which is the total number of cells in the x dimension, is equal to 6 and the number of threads within a block is 3, each thread will calculate 2 cells in the x dimension. Thus, in Listing 5.1, the second `dim3` variable in the kernel call defines the one dimension of the threads to $THREAD\_COUNT$ so that each thread is responsible for $\frac{N_x}{THREAD\_COUNT}$ cells in the x dimension. The number of active threads is $N_z \times N_y \times THREAD\_COUNT$ [10].

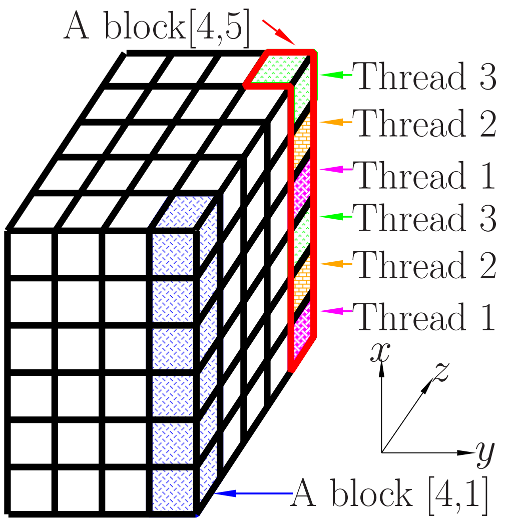
Listing 5.1: Launching kernels with multiple blocks and multiple threads [10].

```
type(dim3) :: dimGrid, dimBlock
dimGrid   = dim3(nk-1,nj-1,1)
dimBlock  = dim3(THREAD_COUNT,1,1)
//...
call ekernel<<<dimGrid ,dimBlock>>>(
Ex_d, Ey_d, Ez_d, Hx_d, Hy_d, Hz_d,
stepsdx, stepsdy, stepsdz,THREAD_COUNT, ni)
//...
call hkernel<<<dimGrid ,dimBlock>>>(
Ex_d, Ey_d, Ez_d, Hx_d, Hy_d, Hz_d,
stmudx, stmudy, stmudz,THREAD_COUNT, ni)
```

In Figure 5.8a, sequential cells are allocated to each thread. In this case, threads of a single block are accessing different locations of the memory concurrently which lead to uncoalesced memory access [10]. On the other hand, when cells are assigned to threads as shown in Figure 5.8b, the access to the memory is coalesced because adjacent memory locations are processed by threads within a block simultaneously [10]. Listing 5.2 shows the coalesced implementation applied in this research.



(a) Uncoalesced memory access



(b) Coalesced memory access

Figure 5.8: Patterns of memory access for multiple threads in a block [10]. $[a, b]$ means [`blockIdx%x`, `blockIdx%y`].

Listing 5.2: Structure of the coalesced kernel implementation [10].

```
k=blockIdx\%x+1
j=blockIdx\%y+1
i=threadIdx\%x+1
Do while(i .le. gridDim\%x)

   //Calculate Ex(i,j,k),
   //Ey(i,j,k) and Ez(i,j,k)
   i=i+THREAD_COUNT

end Do
```

5.4 Implementations of 3D HABC on GPU hardware

The mechanism of implementing the HABC is based on updating the electric fields and the magnetic fields at several regions of the FDTD domain according to a certain calculations. Those regions can be organized in several ways in order to map them to the kernels. Based on the organization, the number of kernel invocations needed for the computation is determined. The dimensions of the HABC parts within the FDTD space are used for the allocation of the blocks and the threads within each kernel. The design of the kernel should optimize the use of the GPUs to speedup the calculations. In this chapter, two different kernel designs based on the characteristics of the HABC are presented. The two implementations of the HABC are integrated with the GPGPU implementation of the FDTD method in Section 5.3. Memory coalescing is used in both of the GPGPU implementations of the HABC to improve the performance of accessing the global memory.

5.4.1 First implementation

The first GPGPU implementation of the 3D HABC is based on the GPGPU implementation of the FDTD presented in Section 5.3. The blocks within the kernel are allocated in two dimensions. The x and y indices of each block are mapped to the y and z dimensions of the FDTD space respectively. Threads within each block are allocated in one dimension, and the x index of the threads is mapped to the x dimension of the FDTD space. This implementation minimizes the number of the kernel invocations. As shown in Listing 5.3, the implementation requires two kernel calls. The first kernel calculates the electric arrays $\boldsymbol{E}_x$, $\boldsymbol{E}_y$ and $\boldsymbol{E}_z$ and the second kernel computes the magnetic arrays $\boldsymbol{H}_x$, $\boldsymbol{H}_y$ and $\boldsymbol{H}_z$. Listing 5.4 presents the code of the kernel that is used to calculate the electric arrays $\boldsymbol{E}_x$, $\boldsymbol{E}_y$ and $\boldsymbol{E}_z$ in the first GPGPU implementation of the 3D HABC.

5.4.2 Second implementation

In the second implementation, the 6 surfaces of HABC are divided into 3 groups as shown in Figure 5.9, Figure 5.10 and Figure 5.11. Since the two planes in each group have the same 2D dimensions, we use one kernel in calculating each

Listing 5.3: Launching kernels with multiple blocks and multiple threads to calculate HABC

```
type(dim3) :: dimGrid, dimBlock
dimGrid  = dim3(nk-1,nj-1,1)
dimBlock = dim3(THREAD_COUNT,1,1)
//...
call habce<<<dimGrid,dimBlock>>>(
Ex_d,Ey_d,Ez_d,Hx_d,Hy_d,Hz_d,ni,itl1,
itl2,jtl1,jtl2,ktl1,ktl2,delta_x,delta_y,
delta_z,delta_t,THREAD_COUNT)
//...
call habch<<<dimGrid,dimBlock>>>(
Ex_d,Ey_d,Ez_d,Hx_d,Hy_d,Hz_d,ni,itl1,
itl2,jtl1,jtl2,ktl1,ktl2,delta_x,delta_y,
delta_z,delta_t,THREAD_COUNT)
```

Listing 5.4: Structure of the single kernel implementation to calculate HABC

```
k=blockIdx\%x
j=blockIdx\%y
i=threadIdx\%x
Do while(i .le. gridDim\%x)
  if((i .eq. itl1) .or. (i .eq. itl2)) then
    //Calculate Ey(i,j,k) and Ez(i,j,k)
  end if
  if((j .eq. jtl1) .or. (j .eq. jtl2)) then
    //Calculate Ex(i,j,k) and Ez(i,j,k)
  end if
  if((k .eq. ktl1) .or. (k .eq. ktl2)) then
    //Calculate Ex(i,j,k) and Ey(i,j,k)
  end if
  i=i+THREAD_COUNT
end Do
```

group. Blocks are allocated in one dimension and the x index of each block is mapped to one of the two dimensions of the surface. Threads within each block are also allocated in one dimension, and the x index of the threads is mapped to the other dimension of the surface. As shown in Figure 5.9, Figure 5.10 and Figure 5.11, the parameters `blockIdx%x` and `threadIdx%x` are assigned to the two dimensions of the planes. For instance, in Figure 5.9 which represents the x direction we have two planes of size $N_y \times N_z$. In this case, the number of blocks is equal to N_y, which is 4, and the number of threads within each block is equal to N_z, which is 5. For the cell in the upper corner shown in Figure 5.9 `blockIdx%x` is equal to 4 since it is the fourth cell in y direction and `threadIdx%x` is equal to 5 because it is the fifth cell in z direction.

Listing 5.5 presents the kernels calls, each kernel is used to calculate two planes as shown in Listing 5.6.

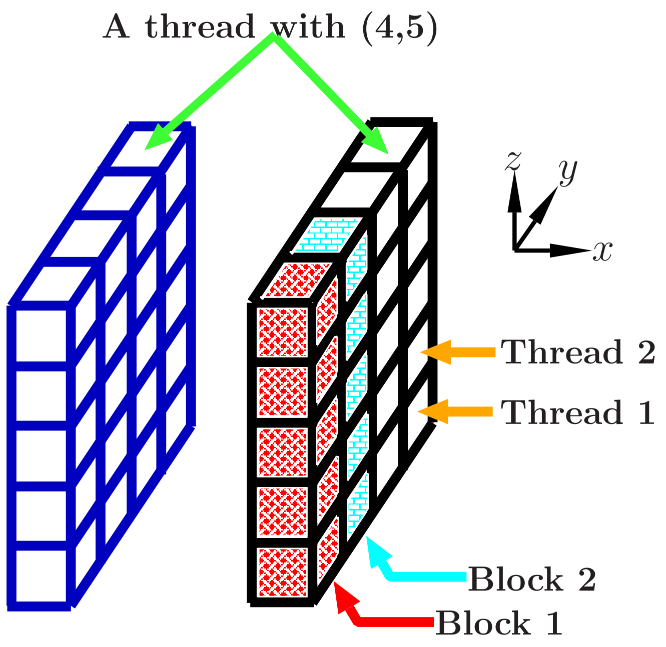


Figure 5.9: The Huygens surfaces of the HABC are divided into 3 groups. The x direction. (a,b) means (`blockIdx%x` , `threadIdx%x`).

Listing 5.5: Launching multiple kernels with multiple blocks and multiple threads to calculate HABC

```
call habcei<<<nk-1,nj-1>>>(
ey_d, ez_d, hy_d, hz_d, iHuyg, iHuyg2,
delta_x, delta_y, delta_z, delta_t)

call habcej<<<nk-1,ni-1>>>(
ex_d, ez_d, hx_d, hz_d, jHuyg1, jHuyg2,
delta_x, delta_y, delta_z, delta_t)

call habcek<<<nj-1,ni-1>>>(
ex_d, ey_d, hx_d, hy_d, kHuyg1, kHuyg2,
delta_x, delta_y, delta_z, delta_t)

//...

call habchi<<<nj-1,nk-1>>>(
ey_d, ez_d, hy_d, hz_d, iHuyg1, iHuyg2,
delta_x, delta_y, delta_z, delta_t)

call habchj<<<ni-1,nk-1>>>(
ex_d, ez_d, hx_d, hz_d, jHuyg1, jHuyg2,
delta_x, delta_y, delta_z, delta_t)

call habchk<<<ni-1,nj-1>>>(
ex_d, ey_d, hx_d, hy_d, kHuyg1, kHuyg2,
delta_x, delta_y, delta_z, delta_t)
```

Listing 5.6: Structure of the multiple kernels implementation to calculate HABC

```
k=blockIdx\%x
j=threadIdx\%x
i=iHuyg1
//Calculate Ey(i,j,k) and Ez(i,j,k)
i=iHuyg2
//Calculate Ey(i,j,k) and Ez(i,j,k)
```

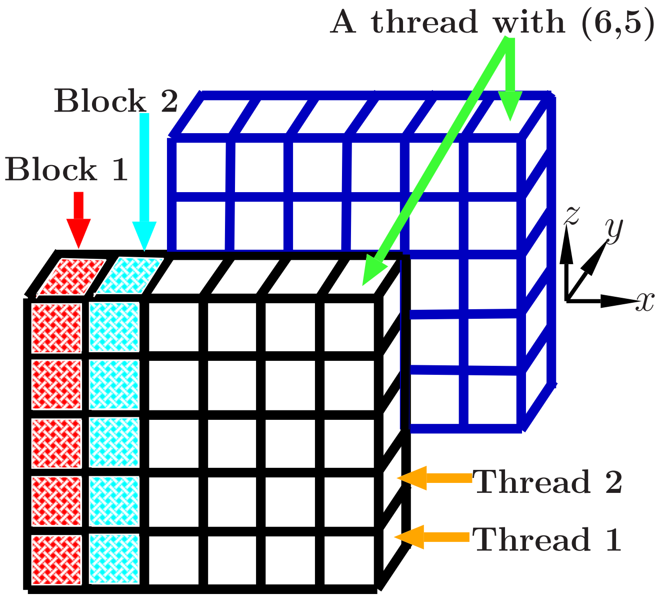


Figure 5.10: The Huygens surfaces of the HABC are divided into 3 groups. The y direction. (a,b) means (`blockIdx%x` , `threadIdx%x`).

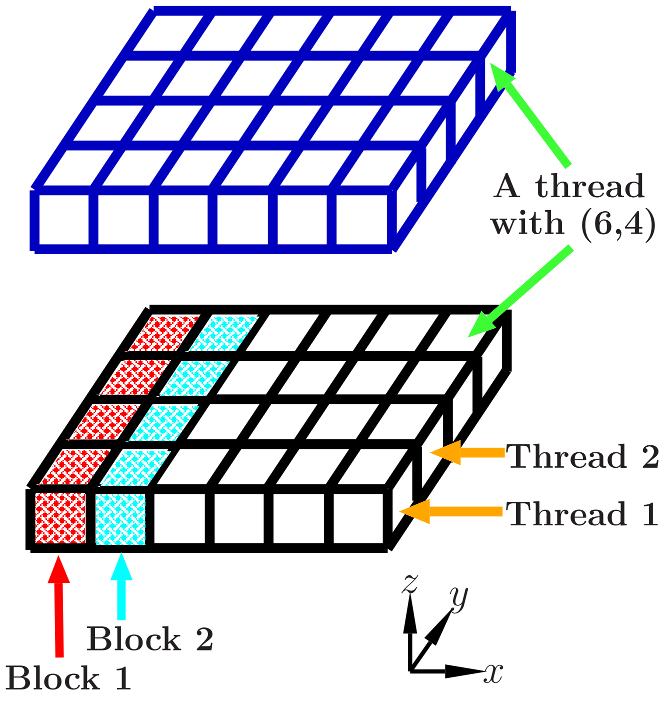


Figure 5.11: The Huygens surfaces of the HABC are divided into 3 groups. The z direction. (a,b) means (`blockIdx%x` , `threadIdx%x`).

5.5 Implementations of 3D CFS-PML on GPU hardware

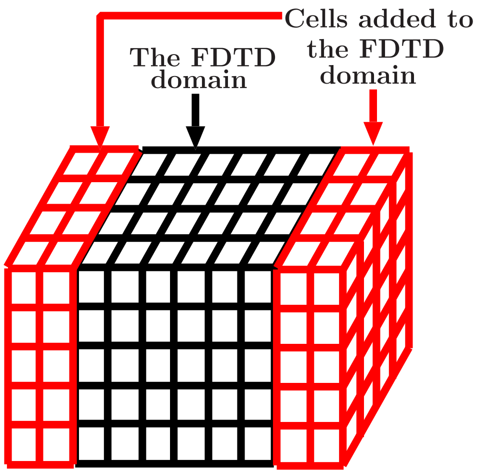
The calculation of the CFS-PML extends the FDTD domain. Assuming that the thickness of the CFS-PML (ncell) is equal to 2, 2 cells will be added in to both sides of each direction to calculate the arrays of the electric and the magnetic fields. However, the organization of the added cells and the calculation of the CFS-PML depend on the calculated field. For instance, Figure 5.12 shows the areas in which the CFS-PML calculates $\boldsymbol{E}_x$ and $\boldsymbol{H}_x$ assuming that ncell is equal to 2 cells. Those areas are computed in different loops using complex equations. Figure 5.13 illustrates the regions required to calculate $\boldsymbol{E}_y$ and $\boldsymbol{H}_y$ and Figure 5.14 demonstrates where the CFS-PML computes $\boldsymbol{E}_z$ and $\boldsymbol{H}_z$ when ncell is equal to 2. The simplest way to implement the CFS-PML on the GPUs is to map the extended 3D domain to a single kernel. The kernel is broken into blocks in 2D, where the x and y indices of the blocks are assigned to two of the three dimensions of the extended domain. Blocks are further divided into threads in one dimension and the x index of the threads is set to the third dimension of the extended domain. The calculation of the CFS-PML equations requires large amount of data, thus memory coalescing is used in the GPGPU implementations of the CFS-PML to improve the performance of accessing the global memory.

5.6 Numerical experiments

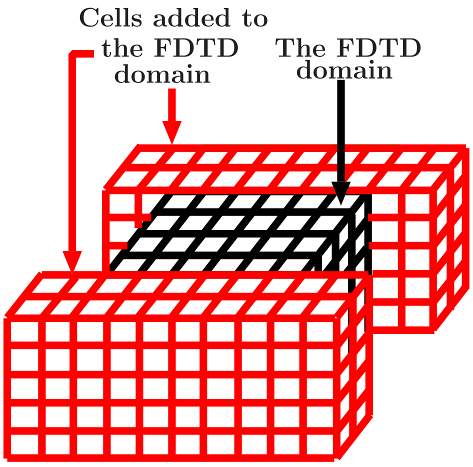
In this section the performance of the GPGPU implementations of the FDTD with the HABC and the CFS-PML is presented. The simulations settings and the computational environments are also illustrated in this section.

5.6.1 Simulations Settings

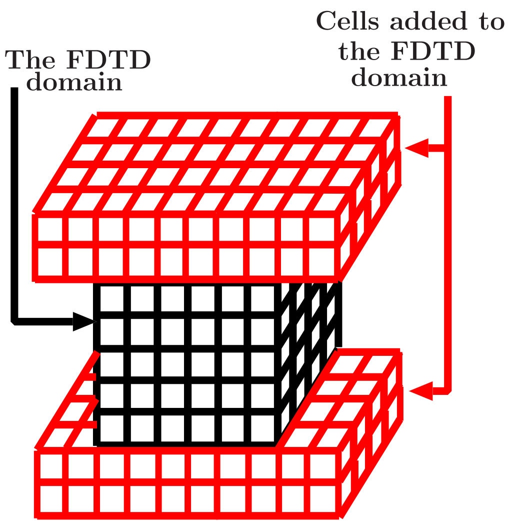
Figure 5.15 shows the environmental settings for the GPGPU numerical experiments. The size of the FDTD space is 256^3 with the spatial step $\Delta x = \Delta y = \Delta z = 1$mm. The metal object is at the center of the FDTD domain. One cell from the metal object is the Huygens excitation. The incident wave of the Huygens



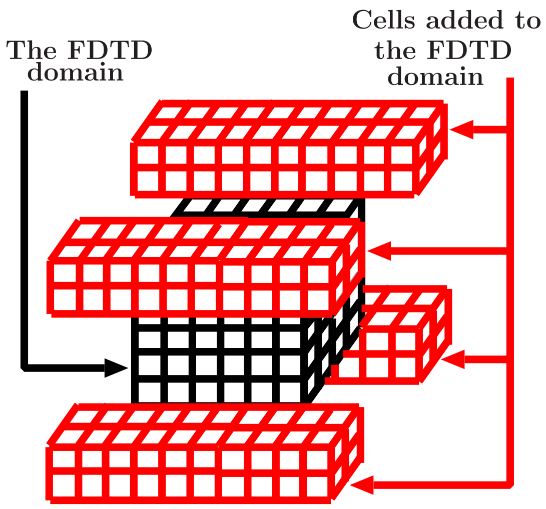
(a) Cells added to the FDTD space at both sides of the x-axis



(b) Cells added to the FDTD space at both sides of the y-axis



(c) Cells added to the FDTD space at both sides of the z-axis



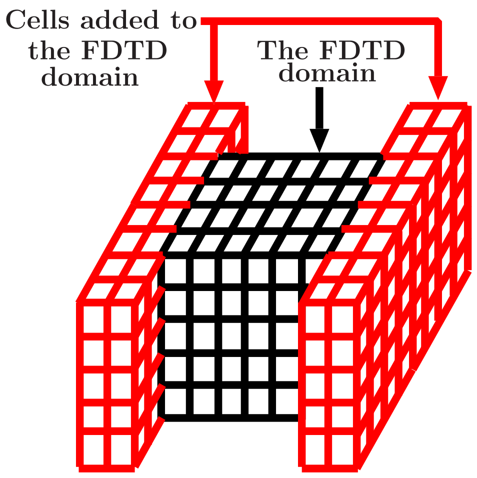
(d) Cells added to the FDTD space at the corners

Figure 5.12: The CFS-PML regions to calculate $\boldsymbol{E}_x$ and $\boldsymbol{H}_x$ with ncell= 2.

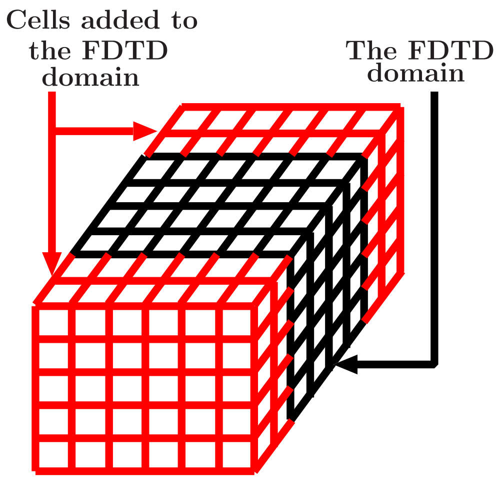
excitation is a Gaussian pulse which has the following equation:

$$\boldsymbol{E}_{inc}(t) = 100e^{-\left[\frac{t-2.5\cdot 10^{-9}}{0.5\cdot 10^{-9}}\right]^2}.$$

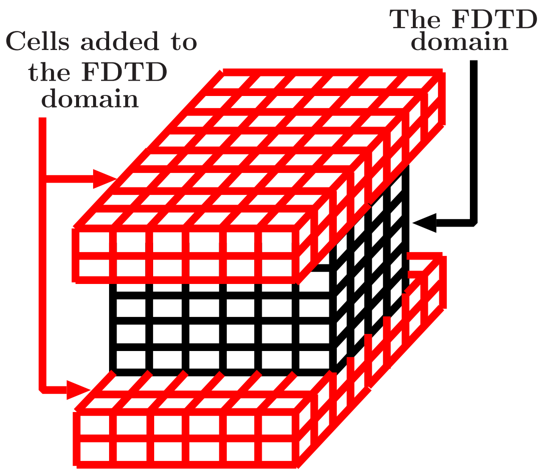
The Huygens excitation is also implemented in CUDA on the GPUs. The HABC is 3 cells away from the metal object and 10 cells to the outer PEC in the HABC



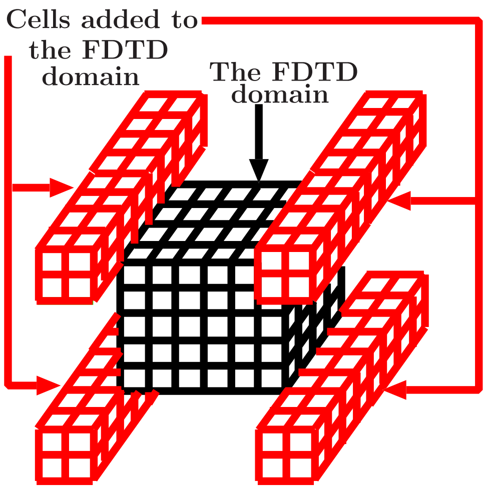
(a) Cells added to the FDTD space at both sides of the x-axis



(b) Cells added to the FDTD space at both sides of the y-axis



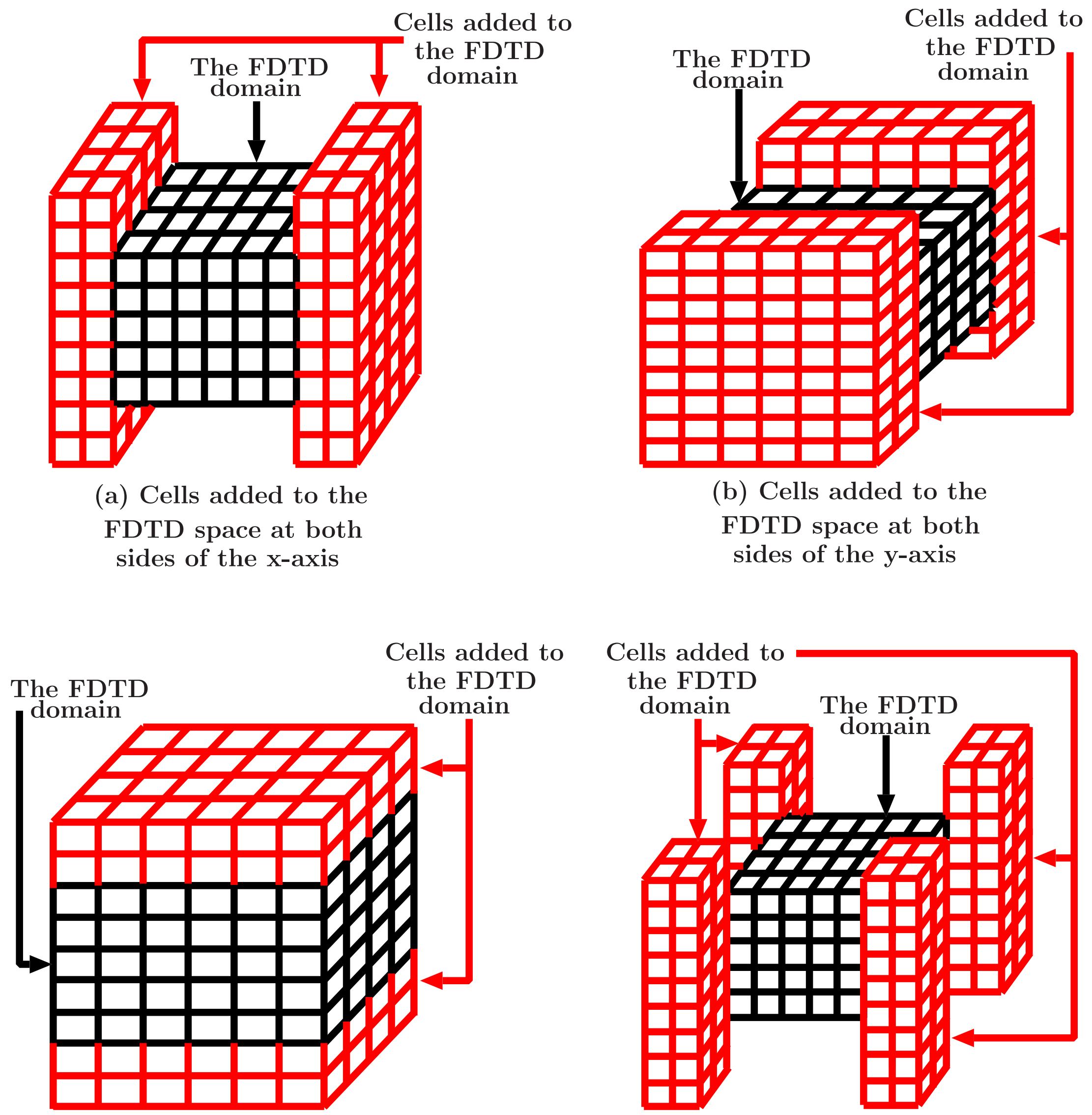
(c) Cells added to the FDTD space at both sides of the z-axis



(d) Cells added to the FDTD space at the corners

Figure 5.13: The CFS-PML regions to calculate $\boldsymbol{E}_y$ and $\boldsymbol{H}_y$ with ncell= 2.

simulations. Similarly, in case of the CFS-PML experiments, the thickness of the PML ncell is equal to 10 cells and it started 3 cells away from the object. Single-precision floating-point computations were used in the simulations.


Cells added to the FDTD domain
The FDTD domain
The FDTD domain
Cells added to the FDTD domain
The FDTD domain
Cells added to the FDTD domain
Cells added to the FDTD domain
The FDTD domain

(a) Cells added to the FDTD space at both sides of the x-axis

(b) Cells added to the FDTD space at both sides of the y-axis

(c) Cells added to the FDTD space at both sides of the z-axis

(d) Cells added to the FDTD space at the corners

Figure 5.14: The CFS-PML regions to calculate $\boldsymbol{E}_z$ and $\boldsymbol{H}_z$ with ncell= 2.

5.6.2 Simulations Results

5.6.2.1 Fermi

In November 2012, the 40th edition of the TOP500 list of the world's fastest supercomputers was released. Japan's K computer developed by Fujitsu Inc. at RIKEN was ranked as the third fastest supercomputer [46]. In RIKEN Integrated Cluster of Clusters (RICC), GPU accelerators are installed on the Multi-purpose Parallel

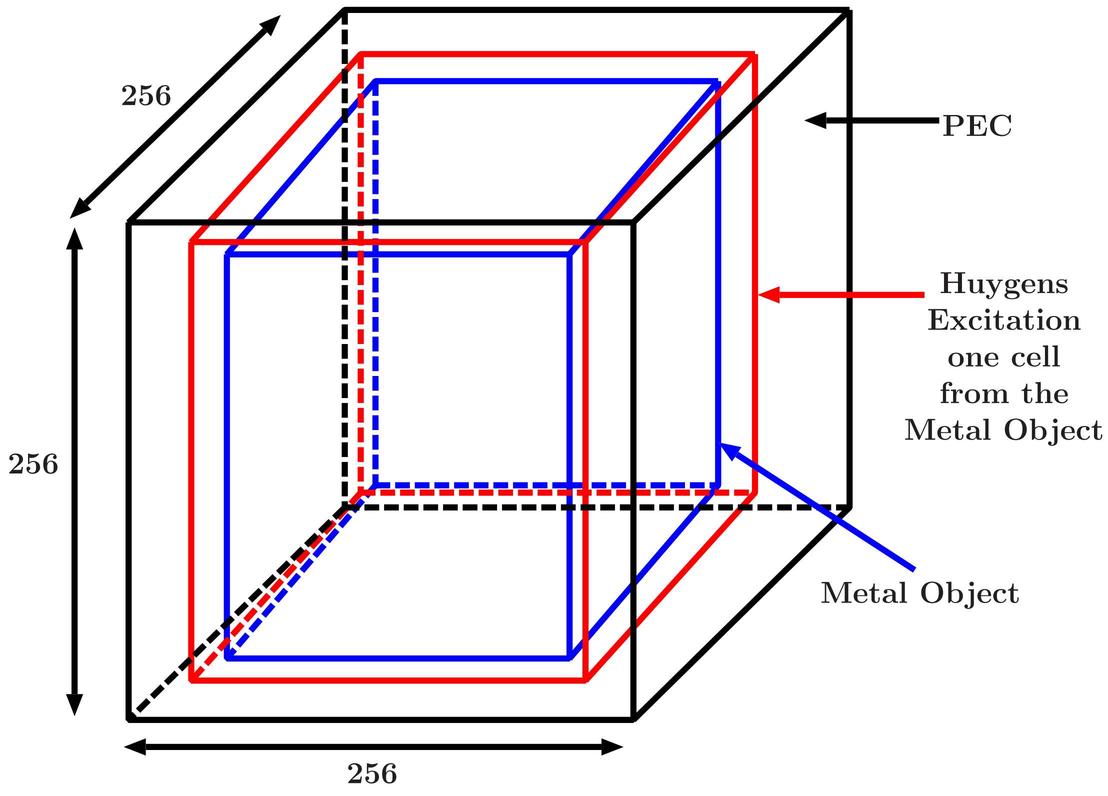


Figure 5.15: The simulations setting of the GPGPU implementations.

Cluster. RICC contains 100 nodes of NVIDIA Tesla C2075 GPUs (Fermi). The specifications of Fermi Tesla C2075 are shown in table 5.1. The CPUs in the Multi-purpose Parallel Cluster are Intel Xeon X5570. Our simulations were run on Fermi GPUs at RICC to analyze the performance.

Figure 5.16 shows the performance of the two GPGPU implementations of the HABC and the GPGPU implementation of the CFS-PML on Fermi GPUs when the 6 FDTD arrays are transferred from GPUs to CPUs at each time step for observation (data output). On the other hand, Figure 5.17 illustrates the performance of the three GPGPU implementations on Fermi GPUs in case of no data output, in other words, no data movement between the GPUs and the CPUs. The curves in Figures 5.16 and 5.17 represent the change in the number of threads per block and the corresponding number of the FDTD cells calculated per second. In Figures 5.16 and 5.17, the red line represents the performance of first GPGPU implementation of the HABC, the green line represents the performance of second GPGPU implementation of the HABC and the blue line represents the GPGPU implementation of the CFS-PML. It is clear from Figures 5.16 and 5.17 that the

	Fermi
GPU name	Tesla C2075
Number of streaming multiprocessors	14
Number of SP cores per SM	32
L2 level cache size per SP (KB)	768
Maximum threads per block	1024
GPU clock (MHz)	575
Clock speed per streaming processor core (MHz)	1150
Type of global memory	GDDR5
Global memory size(GB)	6
Bus width(bit)	384
Global memory bandwidth (GB/s)	144
Peak performance of GPGPU in single precision(GFLOPS)	1030
Peak performance of GPGPU in double precision(GFLOPS)	515

Table 5.1: Specifications of Tesla Fermi C2075 GPUs.

second implementation of the HABC has better performance than the first implementation of the HABC and the implementation of the CFS-PML on GPUs over all selections of the number of threads per block. The first GPGPU implementation of the HABC gives better performance than the GPGPU implementation of the CFS-PML when the number of threads per block is equal to or greater than 32. As shown in Figures 5.16 and 5.17, all of the three GPGPU implementations give their best results when the number of threads is equal to 64. And when the number of threads per block is equal to 64 both of the GPGPU implementations of the HABC give better performance than the GPGPU implementation of the

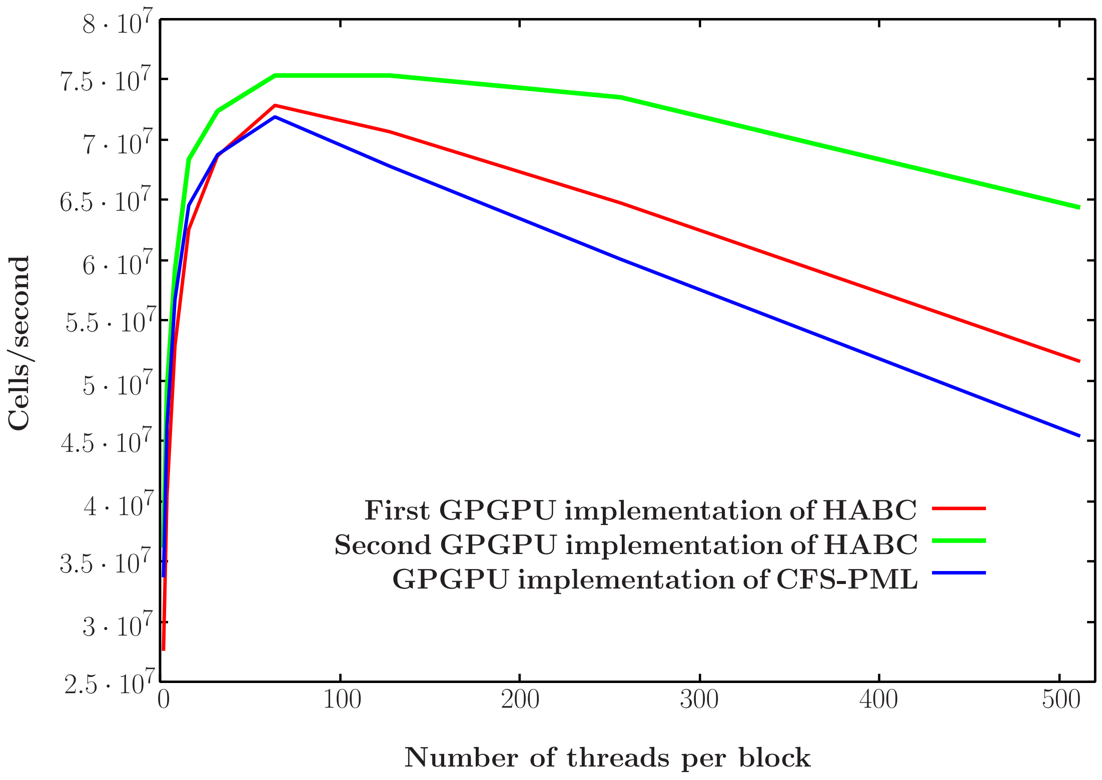


Figure 5.16: Performance of the two GPGPU implementations of the HABC and the CFS-PML on Tesla Fermi C2075 GPUs. The arrays of electric and magnetic fields are moved from GPUs to CPUs at each time step for observation.

CFS-PML. The performance of the three GPGPU implementations improves in case of no data output as illustrated in Figures 5.16 and 5.17.

5.6.2.2 Kepler

The GPUs in the parallel cluster of Kuyushu University are NVIDIA Tesla K20Xm (Kepler). Kepler is the latest Tesla GPU architecture. The specifications of Kepler Tesla K20Xm are shown in table 5.2. The CPUs in Kepler are Intel Xeon E5-2680. Our simulations were run on Kepler GPUs at Kuyushu university HPC to evaluate the performance.

Figure 5.18 presents the performance of the first GPGPU implementation of the HABC and the GPGPU implementation of the CFS-PML on Kepler GPUs. Just like the results of Fermi, the curves represent the change in the number of threads per block and the corresponding number of the FDTD cells calculated per second. The red and blue lines represent the performance of first GPGPU implementation of the HABC and the GPGPU implementation of the CFS-PML

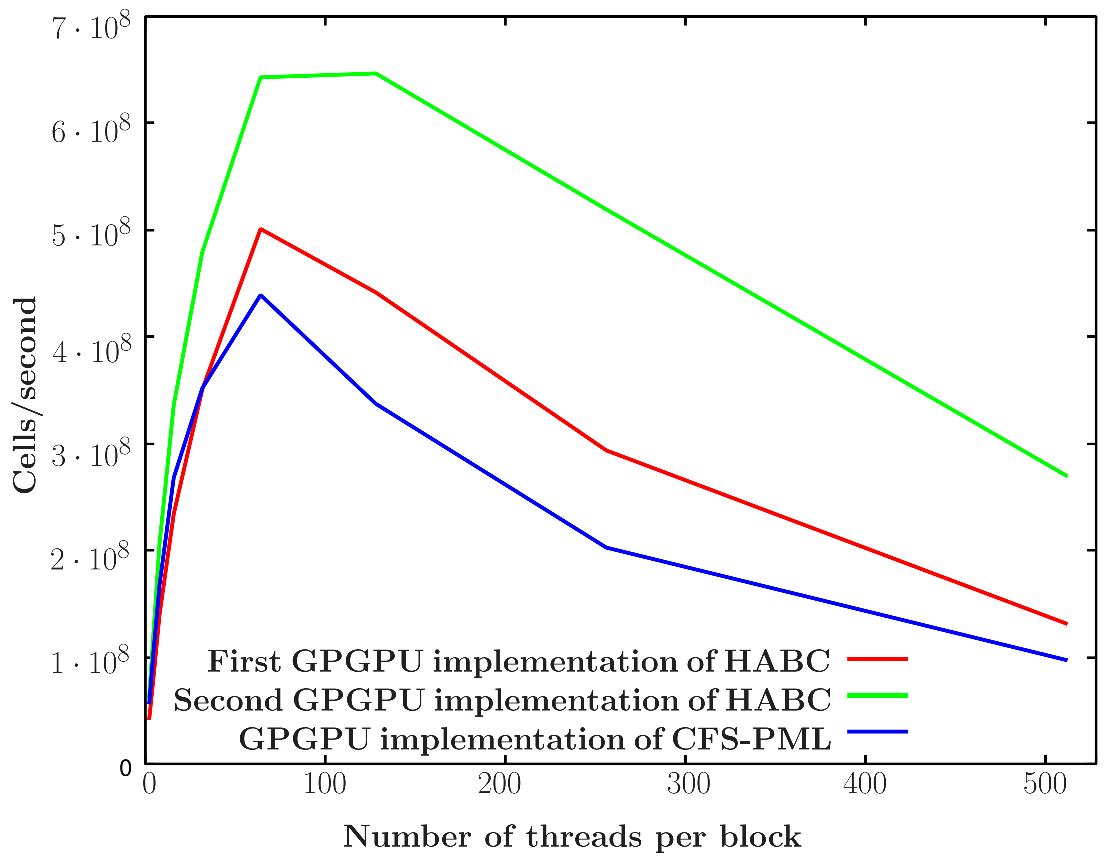


Figure 5.17: Performance of the two GPGPU implementations of the HABC and the CFS-PML on Tesla Fermi C2075 GPUs. No data is moved from GPUs to CPUs for observation (no data output).

respectively. The performance results of the Kepler GPUs agree with the performance results of Fermi GPUs. When the number of threads per block is equal to or greater than 32, the performance of the first GPGPU implementation of the HABC is better than the performance of the GPGPU implementation of the CFS-PML. Also, from Figure 5.18, both of the GPGPU implementations give their best performance when the number of threads is equal to 64 where the performance of the GPGPU implementation of the HABC is better than the GPGPU implementation of the CFS-PML. However, although the computations on the GPUs in Kepler are faster than the computations on the GPUs in Fermi, the performance results of Fermi GPUs is slightly better than the performance results of Kepler GPUs. That is because the communication between the GPUs and the CPUs in Kepler is slower than the communication between the GPUs and the CPUs in Fermi. And since our experiments had outputted the electric field

	Kepler
GPU name	Tesla K20Xm
Number of streaming multiprocessors	14
Number of SP cores per SM	192
L2 level cache size per SP (KB)	1536
Maximum threads per block	1024
GPU clock (MHz)	732
Clock speed per streaming processor core (MHz)	735
Type of global memory	GDDR5
Global memory size(GB)	6
Bus width(bit)	384
Global memory bandwidth (GB/s)	250
Peak performance of GPGPU in single precision(GFLOPS)	3950
Peak performance of GPGPU in double precision(GFLOPS)	1310

Table 5.2: Specifications of Tesla Kepler K20Xm GPUs.

arrays, which required moving the data from the GPUs to the CPUs at each time step, the speed of the communication between the CPUs and the GPUs affected the performance of the simulations.

5.6.3 Conclusion

The new absorbing boundary condition HABC proposed in this thesis is simpler in implementation than the most widely used ABC the CFS-PML. Furthermore, the experiments in this section proofed that the HABC has better performance than the CFS-PML when parallelised on GPUs. Two GPGPU implementations

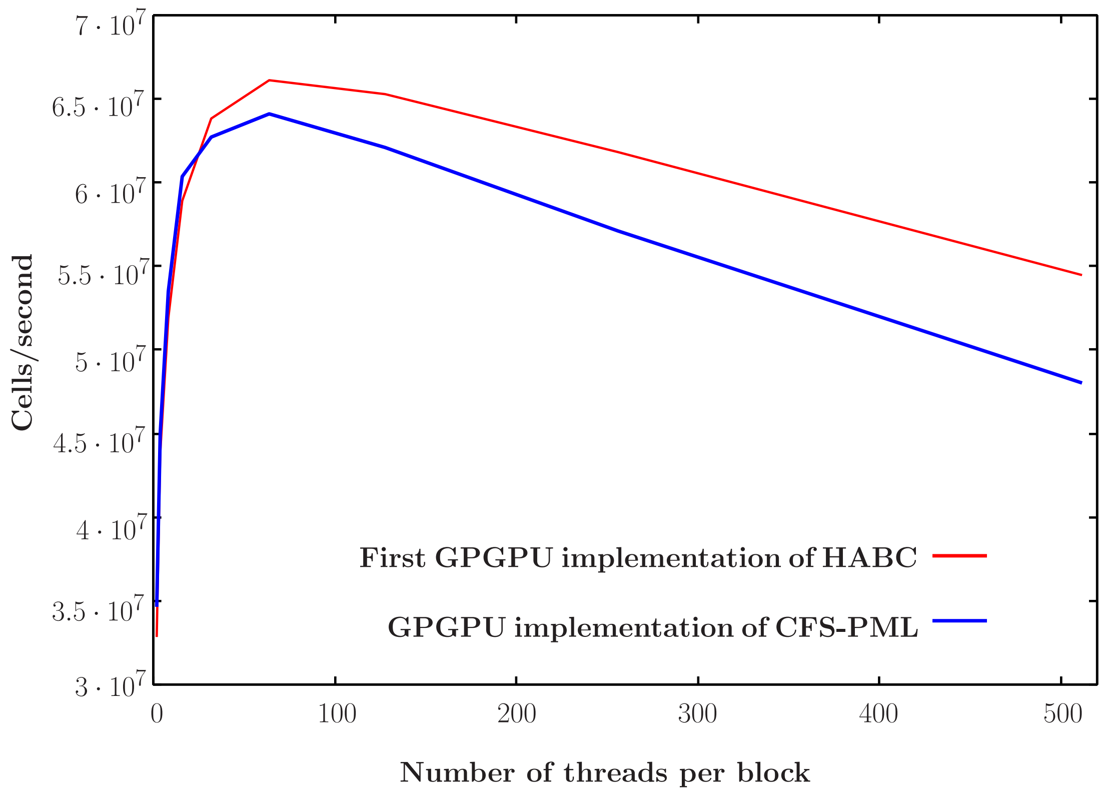


Figure 5.18: Performance of the first GPGPU implementation of the HABC and the CFS-PML on Tesla Kepler GPUs.

of the HABC were introduced, tested and compared with the GPGPU implementation of the CFS-PML. The second GPGPU implementation of the HABC has better performance than the first GPGPU implementation of the HABC and the GPGPU implementation of the CFS-PML. All of the three GPGPU implementations give their best results when the number of threads equals 64. And when the number of threads equals 64, the performance of both GPGPU implementations of the HABC is better than the performance of the GPGPU implementation of the CFS-PML.

Chapter 6

Parallelisation on Shared Memory Architecture

Parallel computing is the use of multiple processors simultaneously to calculate a computational problem [5]. Cores in multi-core processors mentioned in Chapter 4 can have access to their own memory as in distributed memory architecture, or can access one global memory as in shared memory architecture. In shared memory architecture, the simultaneous access to the global main memory by the multiple cores is controlled by the programmer. Open Multi-Processing (OpenMP) is an Application Program Interface (API) specified for shared memory parallelism [41].

OpenMP is the most popular option to parallelize scientific computation on shared memory system [41]. Parallelizing an existing serial code using OpenMP is done by adding special directives. Those directives are responsible for data management and synchronization since the main memory is shared by the multiple cores. As the number of cores increases, the overhead of accessing the main memory also increases. That is because, the speed of accessing the main memory by the multiple cores is restricted by the memory bandwidth mentioned in Chapter 4.

6.1 Threads in Shared Memory Architecture

When the compiler reads the OpenMP directives, it knows that the parallel part of the program is reached. As a result, the master thread swans multiple threads, which are executed concurrently on separate cores. The number of generated threads can be specified in the OpenMP directives. For instance, if the processor contains 8 cores, up to 8 threads can be generated for the parallel computations. When all threads finish running, which is also detected by the OpenMP directives, the master thread terminates the multiple threads and continues executing the rest of the program. This process is illustrated in Figure 6.1. Increasing the

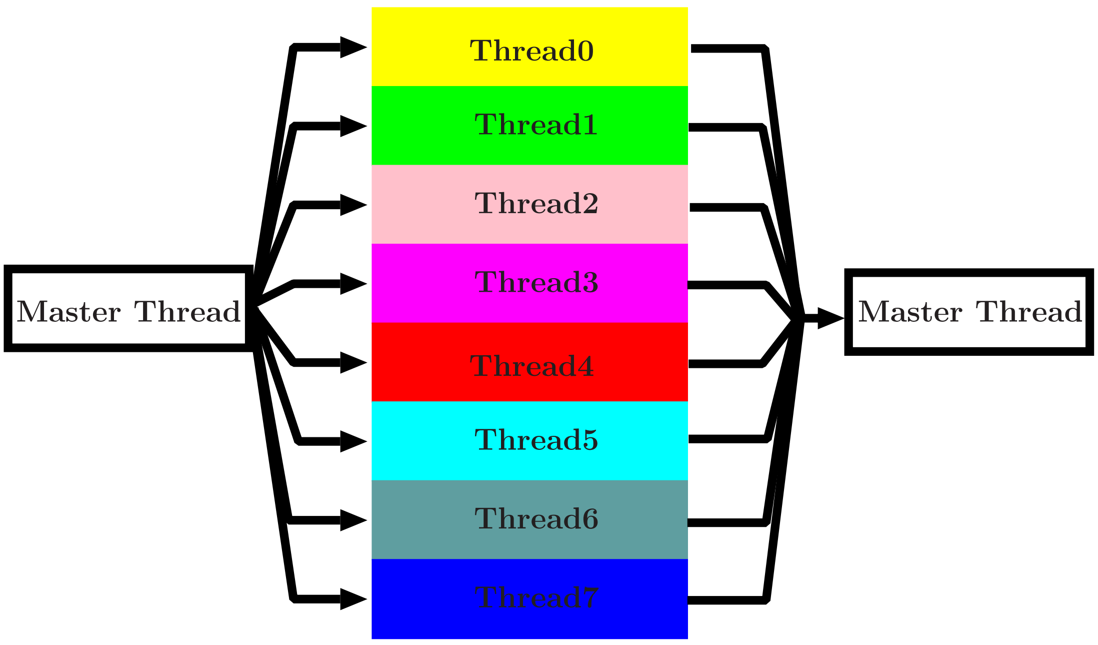


Figure 6.1: Threads in parallel computing.

number of threads lead to speedup the computations. Scaling is used to quantify the speedup and the parallel efficiency when the number of thread increases.

6.2 Strong Scaling and Weak Scaling

The scalability of a parallel program is the evaluation of the execution time relative to the number of processors running that program [47]. Measuring the speedups when the number of the cores increases while the size of the problem is fixed is called strong scaling [47]. However, measuring the speedups while increasing the size of the problem when adding more threads is called the weak scaling.

Amdahl's law [48] states that the speedup ratio of a fixed-size parallel program is:

$$S(N) = \frac{1}{(1-P) + (\frac{P}{N})} \tag{6.1}$$

where S is the speedup ratio, N is the number of the threads executing the parallel program, P is the percentage of the parallel part of the program, and $(1 - P)$ is the percentage of the serial part of the program.

In strong scaling, since the total size of the problem stays fixed as the number of threads increases, the computational size allocated to each thread decreases. Figure 6.2 demonstrates how the size of the computation allocated to each thread decreases when the number of threads increases in strong scaling. Furthermore, when the size of the computational part assigned to each thread is reduced as the number of threads increases, the execution time for each thread will also decreases. Figure 6.3 shows the decrease in the computational time when the number of threads increases in strong scaling.

The speedup in strong scaling depends on the ratio of the parallel part and the serial part of the program. Although strong scaling reduces the size of computations allocated to each thread in the parallel part of the program, the size of the serial part of the program is fixed. This strongly affects the overall speedup of the program. Also, more data transfer is needed as the number of threads increases. The overall speedup in strong scaling is affected by the increase in the data transfer because the speed of the data transfer is relatively slow.

In weak scaling, the size of computation assigned to each thread is fixed. Thus, as the number of threads increases, the overall size of the problem increases. Figure 6.4 demonstrates the size of the computations allocated to each thread when the number of threads increases in weak scaling.

When the number of threads increases in weak scaling, the size of the computation allocated to each thread is fixed. Since the size of the serial part of the program is also fixed, the execution time is expected to be unchanged when more threads are used in the parallel calculations. Theoretically, although the size of the problem is increasing in proportion to the number of threads in weak scaling, the computational time should not be affected as shown in Figure 6.5

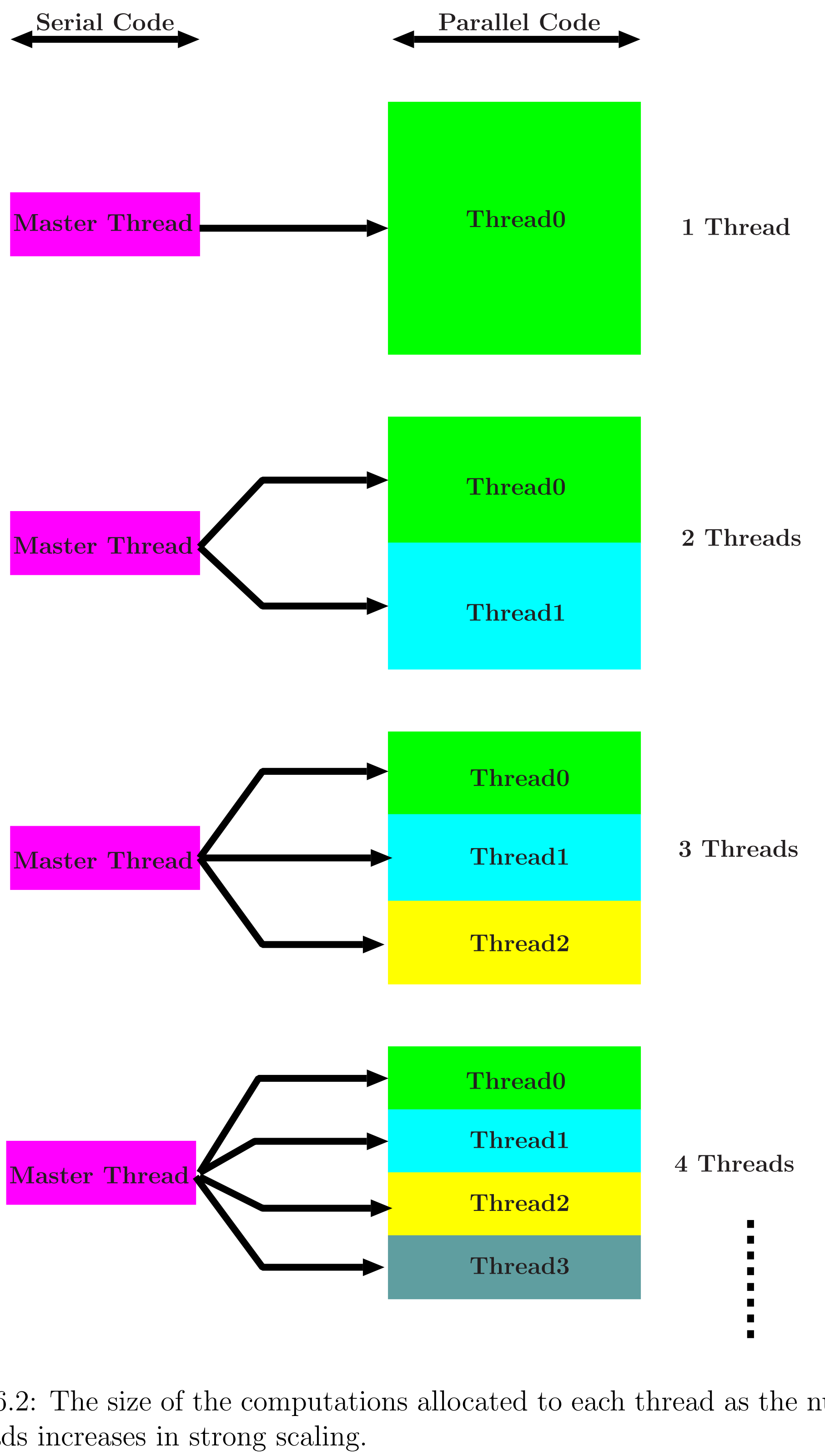


Figure 6.2: The size of the computations allocated to each thread as the number of threads increases in strong scaling.

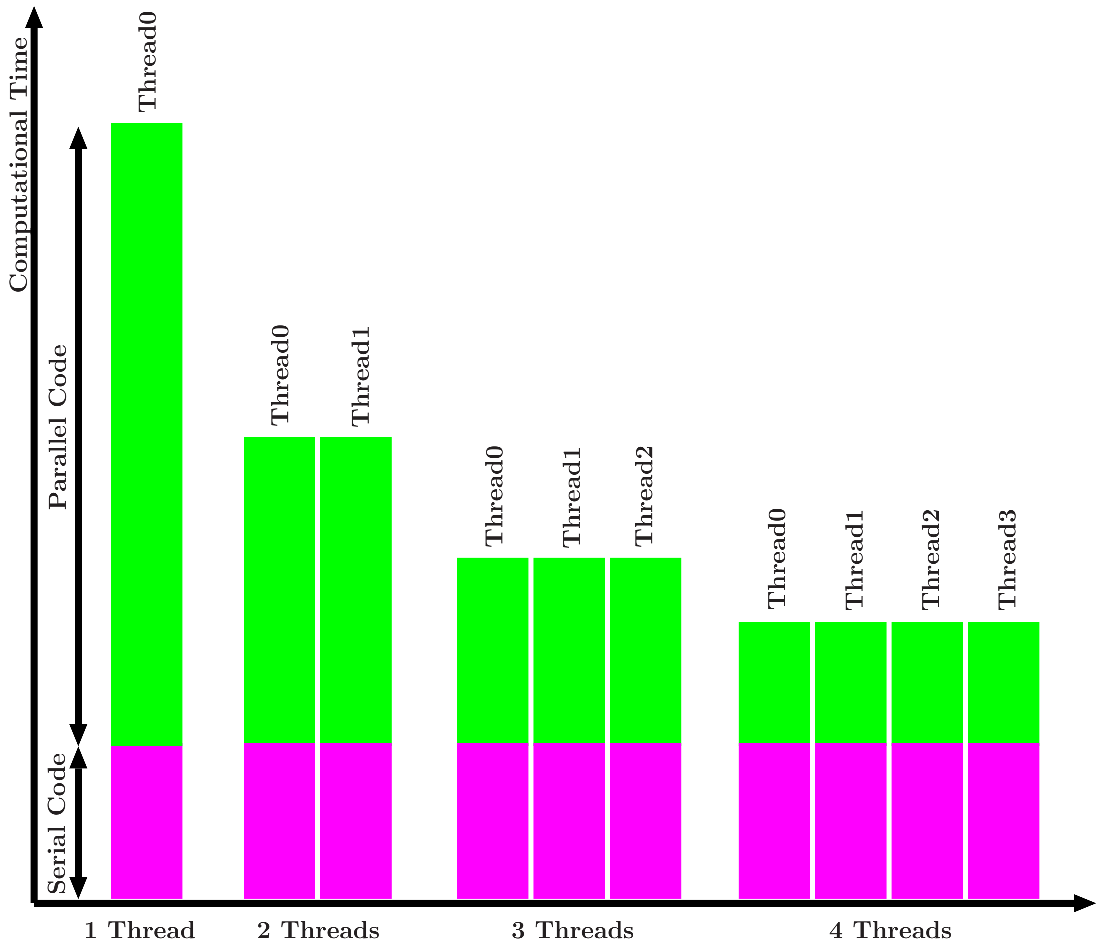


Figure 6.3: The variation of the computational time when the number of threads increases in strong scaling.

6.3 Speedup Ratio and Parallel Efficiency

The speedup ratio for a code is how much faster it runs in parallel relative to the serial execution for the same code [47][49]. If T_1 is the execution time for the serial code and T_N is the execution time for the parallel program with N threads, the speedup ratio is given by

$$S(N) = \frac{T_1}{T_N}. \tag{6.2}$$

Parallel efficiency is how each core is used effectively when the number of threads increases [49]. When the size of a computational problem is fixed (strong

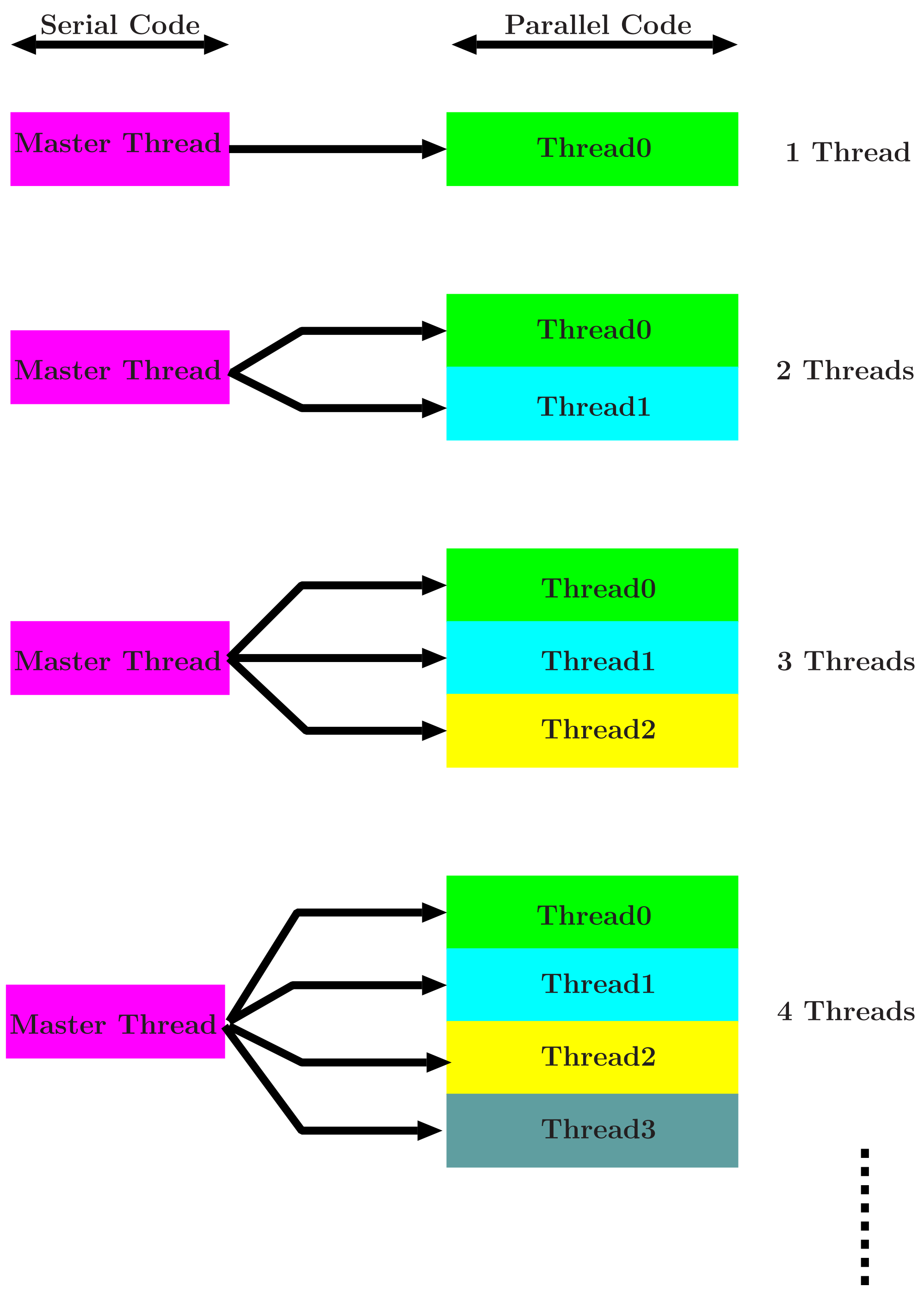


Figure 6.4: The size of the computations allocated to each thread as the number of threads increases in weak scaling.

scaling), the parallel efficiency is calculated as:

$$E(N) = \frac{S(N)}{N} = \frac{T_1}{NT_N}. \tag{6.3}$$

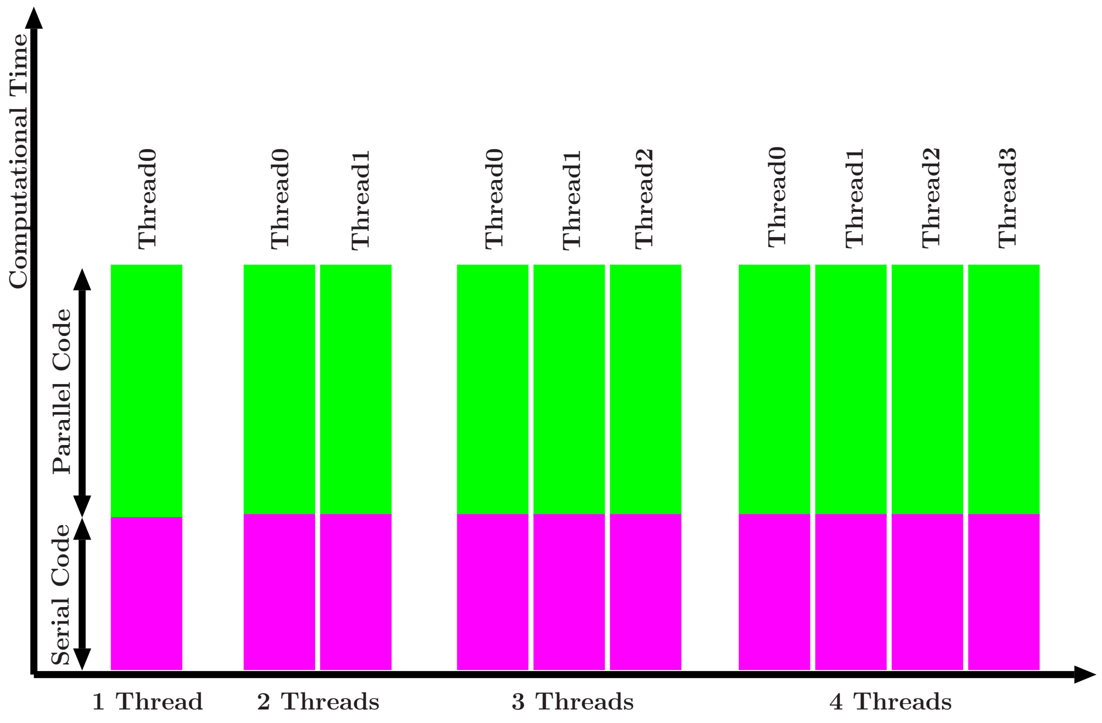


Figure 6.5: The variation of the computational time when the number of threads increases in weak scaling.

In weak scaling, assuming that the time required for the computation should not increase as the number of threads increases, the weak scaling efficiency is defined as [50]:

$$E(N) = \frac{T(1)}{T(N)}. \tag{6.4}$$

The code of the FDTD with the HABC and the CFS-PML is parallelized using OpenMP on shared memory architecture. The parallel efficiency of the HABC for both strong scaling and weak scaling is studied and compared with the CFS-PML.

6.4 Numerical Experiments

The computations of the FDTD method requires huge amount of data. The Large Memory Capacity Server in RIKEN Integrated Cluster of Clusters (RICC) High Performance Computing (HPC) system was used for the experiments. RICC's Large Memory Capacity Server is Altix 450, that consists of 8 cores and 120GB shared main memory. The scalability of the parallelized HABC code with OpenMP on shared memory system is presented in this section.

6.4.1 Environmental Settings

In the HABC and the CFS-PML OpenMP codes, the FDTD space was filled with vacuum. The FDTD space was excited at the center using a soft point source with the Gaussian waveform. The serial and the parallel programs were compiled and run on Altix 450 in RICC.

In strong scaling experiments, the fixed size of the FDTD space was 1000^3 and the memory requirement was 100GB. In case of weak scaling, the size of the FDTD space allocated to each thread was set to 500^3 and the memory requirement per thread is 12.5GB. When the number of threads increases in weak scaling from 1 to 8 the overall size of the parallel program changes from 500^3 to 1000^3. The variation of the size of the FDTD space in weak scaling and the corresponding memory usage are shown in Table 6.1 and Figure 6.6.

Number of Threads	The size of the FDTD space	Memory Usage (GB)
1	500^3	12.5
2	630^3	25
3	721^3	37.5
4	794^3	50.1
5	855^3	62.5
6	909^3	75.1
7	956^3	87.4
8	1000^3	100

Table 6.1: The variation of the size of the FDTD space and the corresponding memory usage when the size of the FDTD space allocated to each thread is set to 500^3 in weak scaling.

6.4.2 Experimental Results

The practical applications of the FDTD method, such as human body simulations require a very large domain and a use of huge amount of data. Thus, in the FDTD

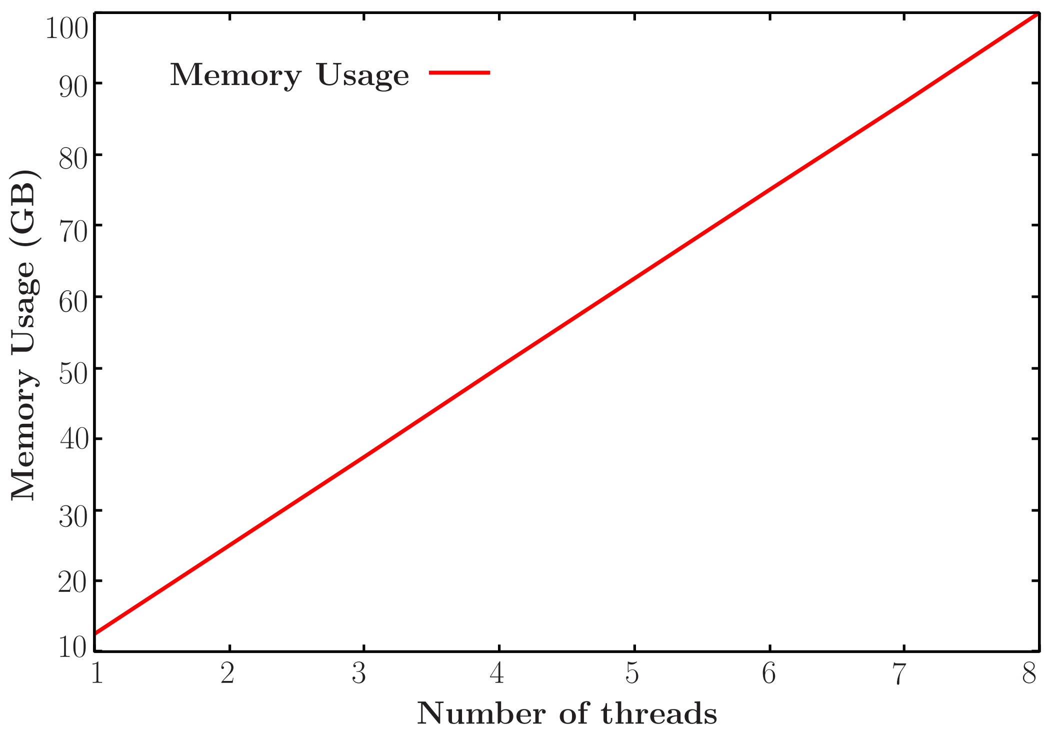


Figure 6.6: The variation of the size of the FDTD space and the corresponding memory usage when the size of the FDTD space allocated to each thread is set to 500^3 in weak scaling.

calculations, the total computational time depends on the speed of data access. When the number of threads increases in the OpenMP code, more threads are sharing the access to the global memory, which causes an overhead to the speed of data access. This overhead affect the overall execution time as the number of threads grows.

In the strong scaling simulations, the size of the FDTD domain is set to 1000^3, which requires large data size of 100GB. As illustrated in Figure 6.7, when the number of threads increases, the speedup ratio of the OpenMP code increases for both HABC and CFS-PML. However, it is clear from the figure that the speedup ratio of the HABC code is better than the CFS-PML OpenMP code.

Figure 6.8 shows the parallel efficiency of the OpenMP codes when the size of the FDTD space is fixed to 1000^3. When the number of threads used for the parallel program increases from 1 to 8, the parallel efficiency of the OpenMP codes decreases because the overhead of accessing the shared main memory by the threads increases. The HABC OpenMP code has better efficiency than CFS-PML OpenMP code over all number of threads.

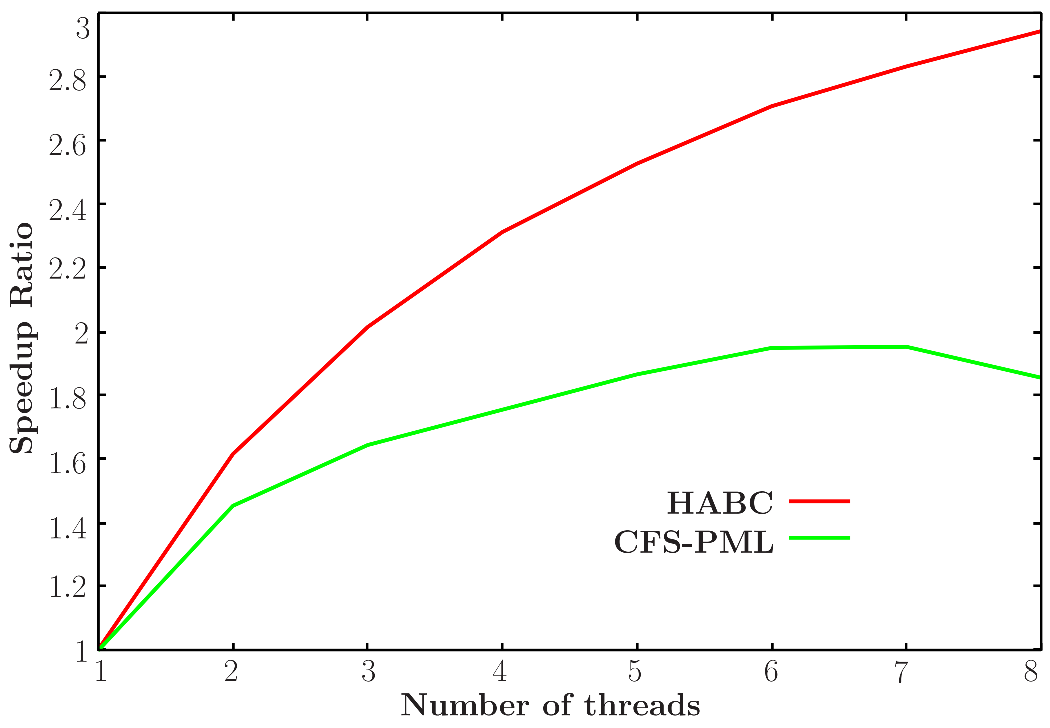


Figure 6.7: The speedup ratio of the OpenMP codes in strong scaling. The fixed size of the FDTD space is 1000^3 and the memory usage is 100GB.

The weak scaling efficiency is presented in Figure 6.9 when the number of threads varies from 1 and 8 and the size of the FDTD space assigned to each thread is 500^3. When the number of threads increases, the size of the FDTD space increases and the corresponding memory usage also increases which lead to the decrease in the weak scaling efficiency of the OpenMP codes. Just like the case of strong scaling, the HABC is more efficient than the CFS-PML over all number of threads in weak scaling.

Figure 6.10 shows the performance of the HABC and CFS-PML implementations on shared memory architecture. The curves in Figure 6.10 represent the change in the number of threads and the corresponding number of the FDTD cells calculated per second. As illustrated in Figure 6.10, the HABC performs better than the CFS-PML on shared memory architecture.

6.4.3 Conclusion

The execution times of the OpenMP codes are compared with the serial time. The FDTD codes with the HABC and the CFS-PML are accelerated when parallelized with OpenMP on shared memory architecture. As the number of threads

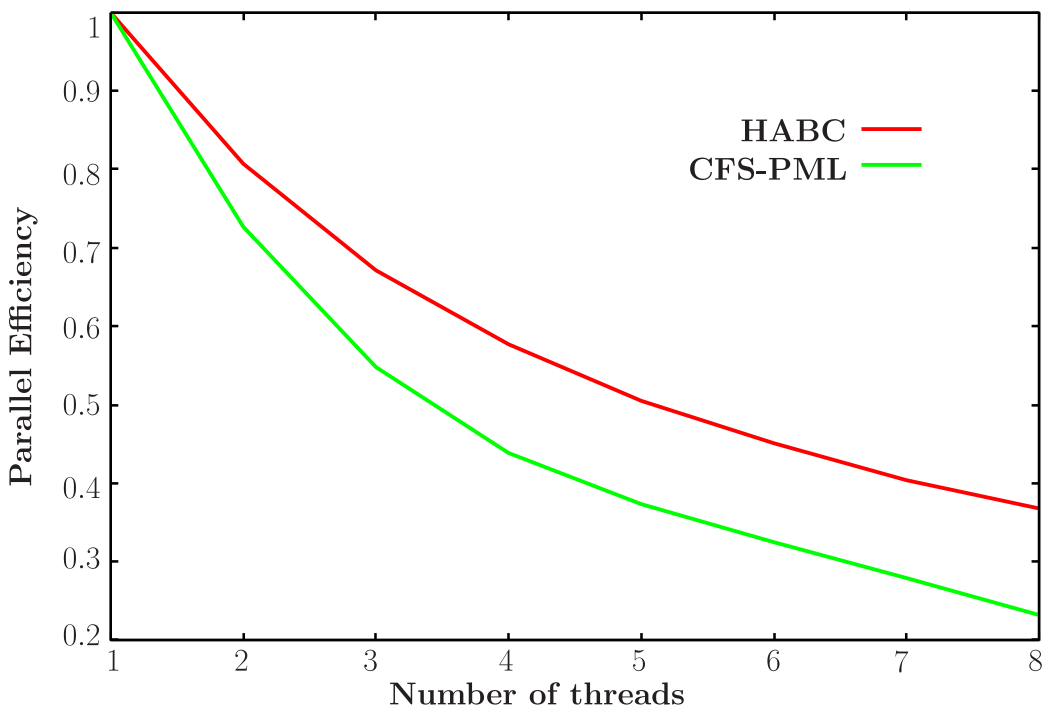


Figure 6.8: The parallel efficiency of the OpenMP codes in strong scaling. The fixed size of the FDTD space is 1000^3 and the memory usage is 100GB.

increases the speedup of the parallel programs increases. In strong scaling, as the number of threads varies from 1 to 8, the speedup ratio of the HABC codes increases from 1 to 3. On the other hand, the increase of the speedup ratio in case of the CFS-PML is less than 2 as the number of threads varies from 1 to 8. Thus, the HABC code scales better than the CFS-PML code. The efficiencies of the HABC OpenMP codes in strong scaling and weak scaling are equivalent. Similarly, the CFS-PML OpenMP codes have the same efficiency in strong scaling and weak scaling. In both strong scaling and weak scaling the HABC is more efficient than the CFS-PML through all number of threads used to parallelize the codes. However, the speed of the memory access highly affects the total computational time in the FDTD calculations, since the FDTD method requires large amount of data. Thus, the efficiency of the OpenMP code decreases when the number of threads increases because more threads are sharing the access to the global memory and more communication time is needed.

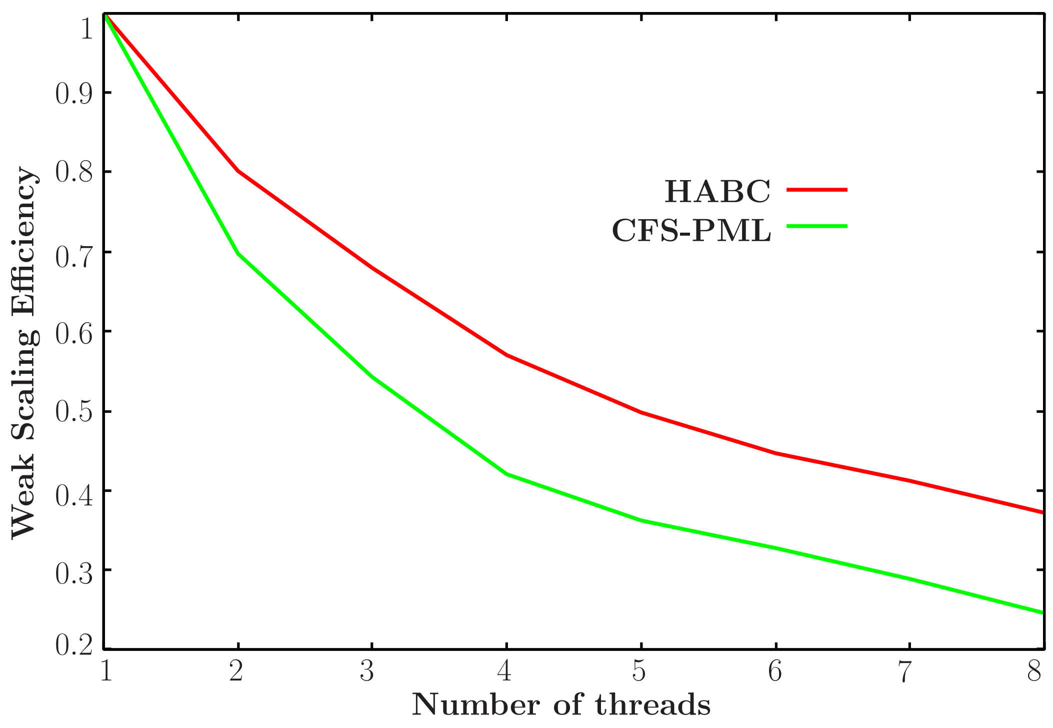


Figure 6.9: The weak scaling efficiency of the OpenMP codes. The FDTD space vary from 500^3 to 1000^3. The size of the FDTD space is set to 500^3 for each thread.

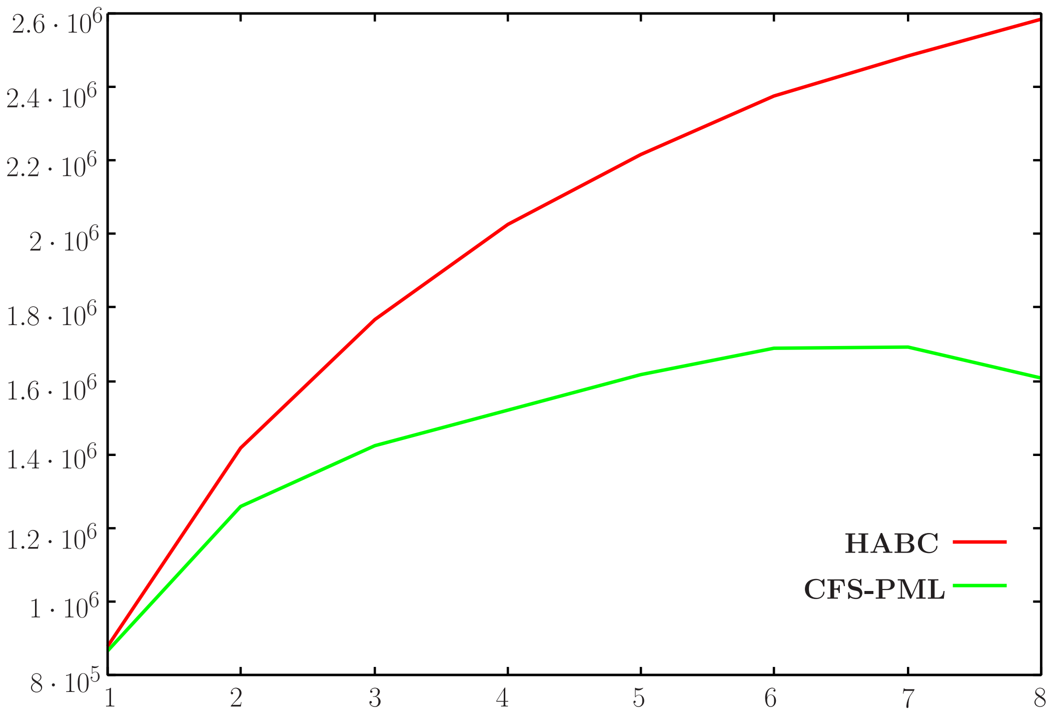


Figure 6.10: Performance of the HABC and CFS-PML implementations on shared memory architecture.

Chapter 7

Practical Application with Human Body

High body temperatures are often caused by illnesses such as fever or heat stroke [51]. However, throughout history, heat has been used as a therapy for the human body. Hyperthermia, which is also called thermal therapy or thermotherapy, is a type of cancer treatment in which body tissue is exposed to high temperatures [52]. Very high temperatures can kill cancer cells outright, but they can also injure or kill normal cells and tissues [51]. Warmer temperatures make changes inside the cells and can render the cells more likely to be affected by other treatments such as radiation therapy or chemotherapy.

Hyperthermia is almost always used in conjunction with other forms of cancer therapy such as radiation therapy and chemotherapy [52]. For instance, the inner part of the tumour masses usually contains oxygen-deprived (hypoxic) cells. The hypoxic cells are sensitive to heat and resist radiation. Thus, when hyperthermia is combined with radiation, the radiation will kill the outer oxygenated cells while hyperthermia will make low oxygen inner cells more susceptible to radiation damage [53]. In 2009, the Berlin European Society of Medical Oncology presented a study on Sarcoma using chemotherapy alone, or chemotherapy plus hyperthermia. The study showed that in a randomised trial, the group also using hyperthermia (almost) doubled their survival times from a mean of 18 months to over 32 months [53].

There are several types of hyperthermia such as local hyperthermia, regional hyperthermia, and whole body hyperthermia [54]. These types can be applied to the human body using different methods. For example, microwaves, magnets, warm water blankets, hot wax, inductive coils and chambers may be used to heat up the human body. This thesis uses Huygens excitation method to heat up the human body with electromagnetic waves as an application of hyperthermia. The Huygens excitation is applied to the human body with the SM-HABC and the results are compared with those of the CFS-PML.

7.1 Numerical Modeling of the Human Body

In this thesis, the FDTD method is used to model the human body. This is done by discretizing the human body and assigning the discretized space (tissues and organs) to its corresponding dielectric parameters. These parameters vary over frequency. The single-pole Debye is used to model the frequency-dependent parameters.

The RIKEN Advanced Science Institute used Magnetic Resonance Imaging (MRI) to scan the human body at every 1mm. They segmented each MRI image using several medics and produced a 1mm^3 voxel of Digital Human Phantom (DHP). The DHP data was provided by the RIKEN Bio-research Infrastructure Construction Team under the non-disclosure agreement between the University of Manchester and RIKEN. Our group is using the DHP data provided by RIKEN. This DHP data is transformed to a set of frequency-dependent media parameters which is used in the Debye FDTD computations. This transformation is based on Tables 7.1 and 7.2 which represent the media parameters of the human tissues. Each voxel in the DHP has an identifying number that is mapped to the corresponding tissue according to Tables 7.1 and 7.2. For instance, if the DHP data gives the identifying number 45 for a voxel, then the corresponding Debye parameters are $\sigma = 0.10401546$ (S/m), $\epsilon_{\mathrm{S}} = 14.16905880$, $\epsilon_{\infty} = 7.36329556$ and $\tau_{\mathrm{D}} = 34.1064989$ (ps) based on Table 7.2. The Debye media parameters in Tables 7.1 and 7.2 were produced by the measurement data provided by the U.S. Air force. The size of the data for the whole human body is $265 \times 490 \times 1682$. In this thesis, the whole human body is placed at the center of the FDTD space with the SM-HABC or the CFS-PML surrounding the human body. The Huygens excitation is applied to the human body in simulations to perform hyperthermia.

Table 7.1: Parameters of the human tissues. Part(1).

Tissue number	Tissue name	σ(S/m)	ϵ_{S}	ϵ_∞	τ_{D}(ps)
0	air	0.00000000	1.00000000	1.00000000	0.00000000
1	cerebral cortex	0.59515458	56.44398499	33.05709076	35.1972652
2	white matter	0.34836939	41.28079033	24.37136650	33.5850722
3	cerebellum	0.82605255	58.15469742	35.19484711	68.2758780
4	midbrain	0.34836939	41.28079033	24.37136650	33.5850722
5	eyeball	1.44514167	67.71049214	10.30803776	8.27143302
6	optic nerve	0.35986477	34.74881363	20.98018456	36.5079009
7	cornea	1.06887341	57.84246635	32.37210846	28.7477611
8	the crystalline lens	0.34613180	37.82515335	18.28276825	21.0985136
9	pituitary gland	0.80907071	60.54281044	27.70847893	17.7068464
10	thalamus	0.59515458	56.44398499	33.05709076	35.1972652
11	hypothalamus	0.59515458	56.44398499	33.05709076	35.1972652
12	pineal gland	0.80907071	60.54281044	27.70847893	17.7068464
13	tongue	0.69344097	56.52260780	28.25727081	20.4540135
14	cerebrospinal fluid	2.14390564	70.39963913	33.14797211	18.1622616
15	right suprarenal body(gland)	0.80907071	60.54281044	27.70847893	17.7068464
16	urinary bladder	0.30010962	19.33148861	9.67463303	20.8416045
18	colon (large intestine)	0.71641898	61.12129974	34.66533279	31.1704239
20	duodenum	0.91899437	66.27923393	31.50330162	18.9756596
21	esophagus	0.91899437	66.27923393	31.50330162	18.9756596
23	gall bladder	1.04225028	60.73483276	28.18050003	15.9106617
24	heart	1.01880646	63.54899597	34.90967941	28.8596560
25	right kidney	0.85720283	67.50839424	39.85999298	60.6079908
26	liver	0.52030158	50.15293502	27.98551559	35.6627956
27	right lung	0.45216662	38.25424575	21.43600845	27.3953464
29	pancreas	0.80907071	60.54281044	27.70847893	17.7068464

Table 7.2: Parameters of the human tissues. Part(2).

Tissue number	Tissue name	σ(S/m)	ϵ_{S}	ϵ_∞	τ_{D}(ps)
30	prostate gland	0.80907071	60.54281044	27.70847893	17.7068464
31	small intestine	1.70134962	65.79007912	39.19070053	45.8777842
32	spleen	0.84441698	62.58494759	37.11508179	42.6902229
33	stomach	0.91899437	66.27923393	31.50330162	18.9756596
37	nasal cavity	0.00000000	1.00000000	1.00000000	0.00000000
38	thyroid gland	0.80898708	60.55047417	27.73882103	17.7273439
39	trachea	0.56548506	43.18475723	22.14025688	22.5001805
45	bone	0.10401546	14.16905880	7.36329556	34.1064989
48	fat	0.03710634	5.53071189	3.99812841	23.6329289
49	muscle	0.74710494	56.93144607	28.00134277	18.6721420
51	skin	0.54073572	47.93014336	29.85054779	43.6257593
55	diaphragm	0.74710494	56.93144607	28.00134277	18.6721420
56	seminal vesicle	0.80907071	60.54281044	27.70847893	17.7068464
57	body of erectile tissue	1.25574398	62.90047836	30.59756851	21.0197831
80	spinal cord	0.35986477	34.74881363	20.98018456	36.5079009
82	paranasal sinus	0.00000000	1.00000000	1.00000000	0.00000000
83	testicle	0.93375176	62.06441498	31.30603409	21.1258597
84	left kidney	0.85720283	67.50839424	39.85999298	60.6079908
85	left lung	0.45216662	38.25424575	21.43600845	27.3953464
86	epiglottis	0.49346393	44.55864907	19.39198685	29.9815103
87	oral cavity	0.00000000	1.00000000	1.00000000	0.00000000
88	external ear	0.00000000	1.00000000	1.00000000	0.00000000
89	salivary gland	0.80907071	60.54281044	27.70847893	17.7068464
90	middle ear	0.05478248	7.31782723	3.96209121	37.7117643
91	inner ear	2.14390564	70.39963913	33.14797211	18.1622616
92	pharynx	0.56548506	43.18475723	22.14025688	22.5001805
94	left suprarenal body(gland)	0.80907071	60.54281044	27.70847893	17.7068464

7.2 Numerical experiments

The FDTD space has a human body at the center. The Huygens excitation planes are placed around the human body to heat it up as an application of hyperthermia. The direction of the wave is from the front of the human body to the back of the human body, as shown in Figure 7.1. The incident wave of the

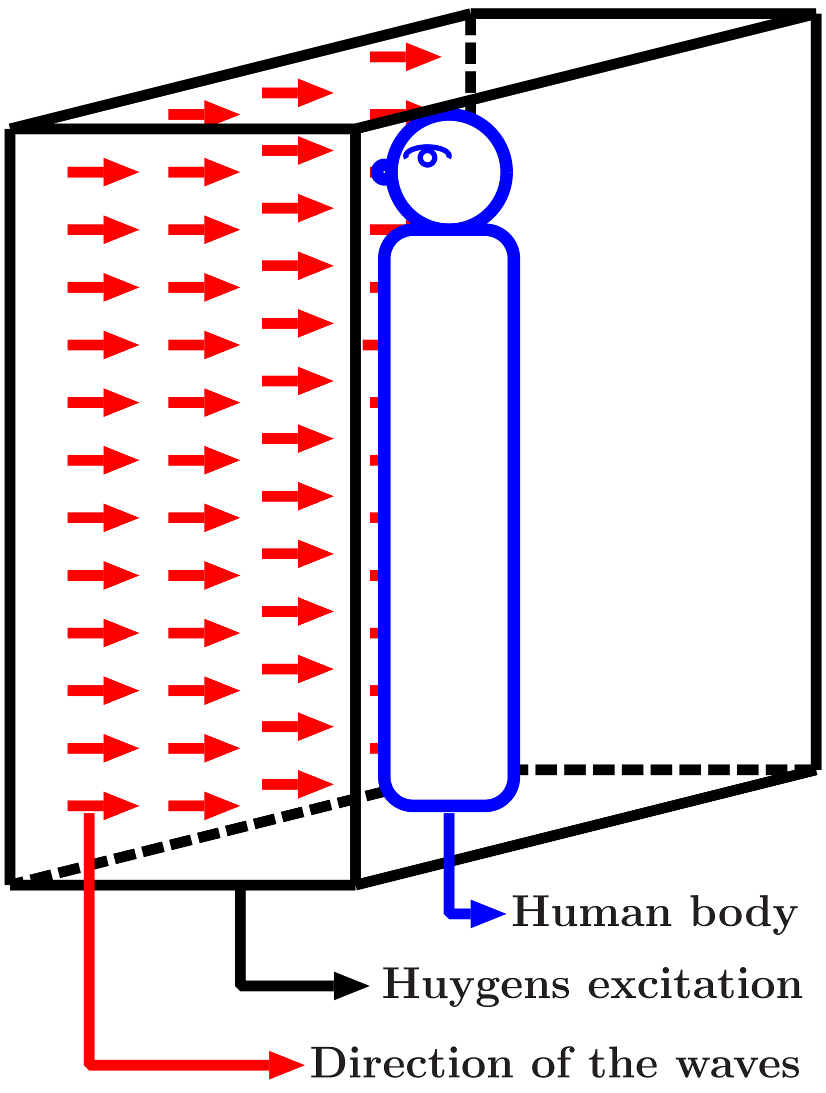


Figure 7.1: The Huygens excitation planes are placed around the human body and the direction of the wave is from the front to the back of the human body.

Huygens excitation is a Gaussian pulse which has the following equation:

$$\boldsymbol{E}_{inc}(t) = 100e^{-\left[\frac{t-2.5\cdot 10^{-9}}{0.5\cdot 10^{-9}}\right]^2}.$$

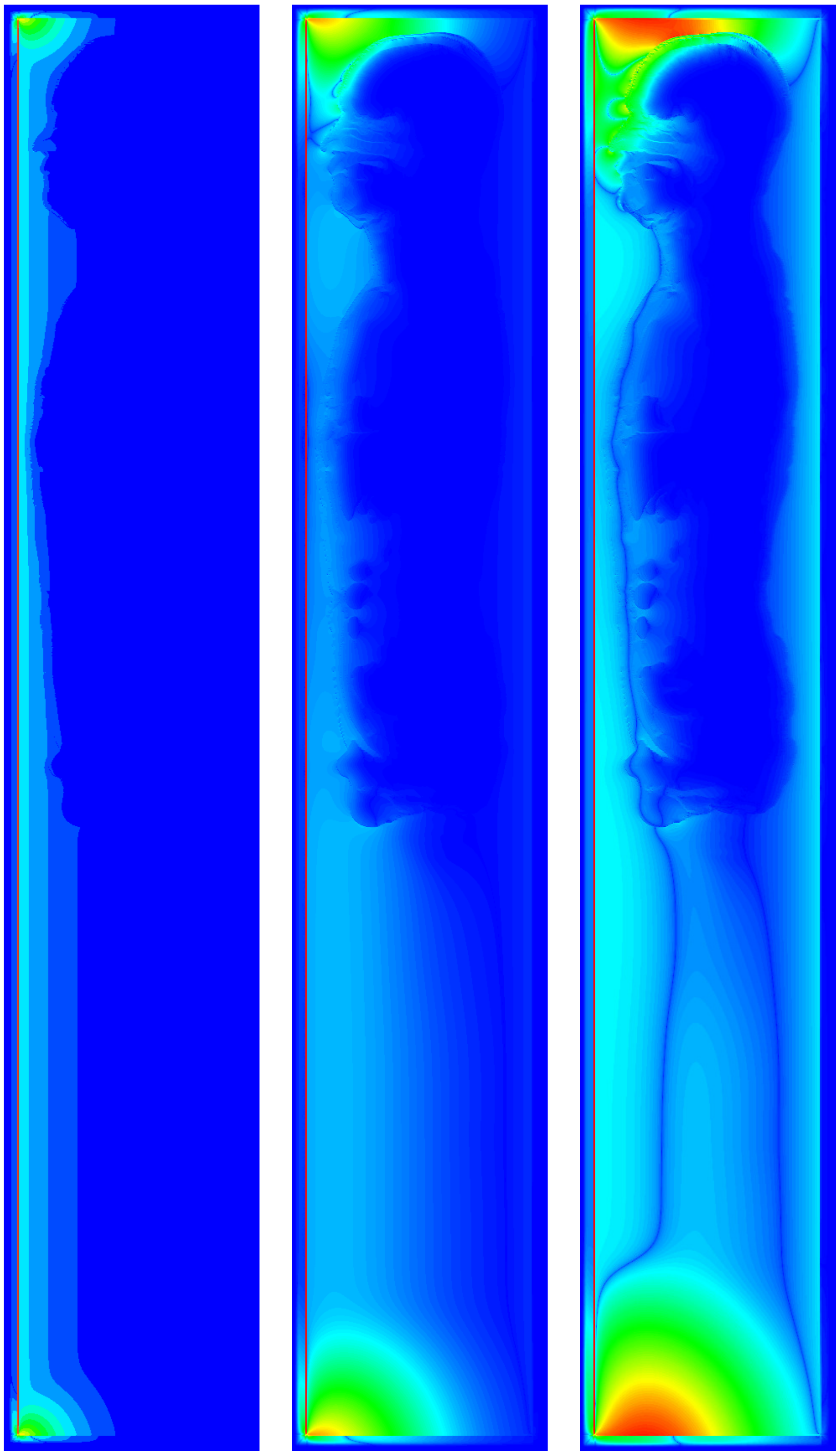
The size of the FDTD domain is $285 \times 510 \times 1707$. In the SM-HABC experiments, from the outer PEC of the FDTD domain, the number of stretched cells (ng) is equal to 16 while the HABC planes are placed 17 cells away. Similarly, in the case of the CFS-PML experiments, the CFS-PML starts 17 cells away from the outer PEC of the FDTD domain, and the thickness of the CFS-PML (ncell) is equal to 17.

Figure 7.2 shows a snapshots of a 2D-colored plane observations of the FDTD domain in the SM-HABC experiments when y is constant. In Figure 7.2, the snapshots show the FDTD domain including the HABC and the stretched cells which are out the Huygens excitation. Figure 7.2(a), Figure 7.2(b), and Figure 7.2(c) show the snapshots at $t = 250\Delta t$, $t = 650\Delta t$ and $t = 800\Delta t$, respectively.

Snapshots of a 2D-colored plane observations of the FDTD domain in the CFS-PML experiments when y is constant are displayed in Figure 7.3. Unlike Figure 7.2, in Figure 7.3 the snapshots illustrate only the Huygens excitation planes and what is inside those planes. The CFS-PML cells do not appear in the snapshots in Figure 7.2. The snapshots at $t = 250\Delta t$, $t = 650\Delta t$ and $t = 800\Delta t$ are presented in Figure 7.3(a), Figure 7.3(b), and Figure 7.3(c), respectively.

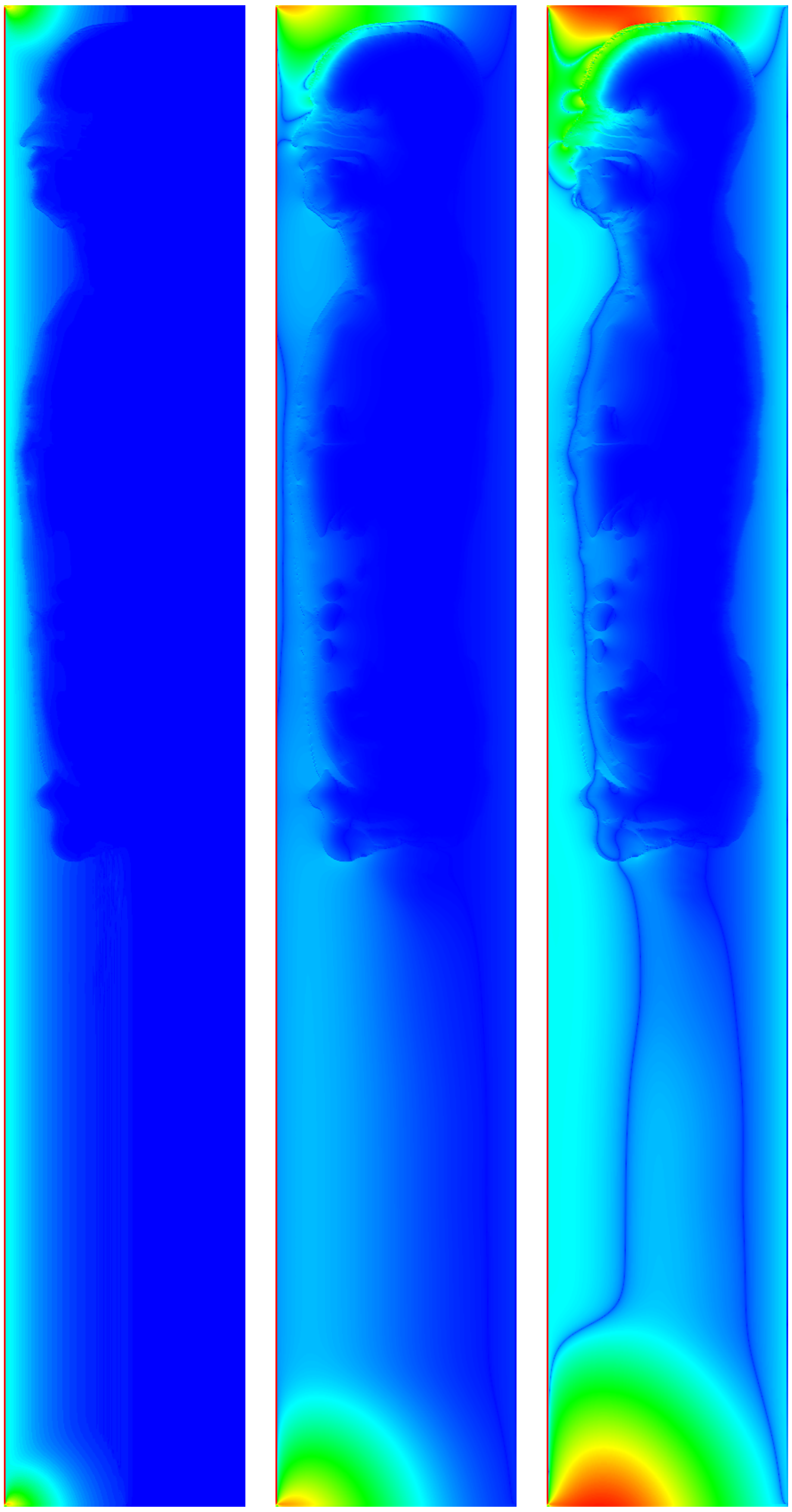
7.3 Conclusion

The Huygens excitation heated up the human body as a method of applying the hyperthermia. The human body was placed at the center of the FDTD space surrounded by the Huygens excitation planes. The SM-HABC was tested with the human body and the Huygens excitation. The results of the SM-HABC experiments ware comparable to the results of the CFS-PML experiments. In addition, for 10 time-steps, the SM-HABC simulations took 154 seconds and the CFS-PML simulations took 283 seconds. Thus, the time required to simulate the whole human body with the SM-HABC is better than the time required to simulate the whole human body with the CFS-PML.



(a) At $t = 250\Delta t$. (b) At $t = 650\Delta t$. (c) At $t = 800\Delta t$.

Figure 7.2: A snapshots of a 2D colored plane observations of the FDTD domain that has a human body at the center surrounded with Huygens excitation planes in the SM-HABC experiments. y is constant and the snapshots are at $t = 250\Delta t$, $t = 650\Delta t$ and $t = 800\Delta t$. The snapshots show the FDTD domain including the HABC and the stretched cells which are out the Huygens excitation planes.



(a) At $t = 250\Delta t$. (b) At $t = 650\Delta t$. (c) At $t = 800\Delta t$.

Figure 7.3: A snapshots of a 2D colored plane observations of the FDTD domain that has a human body at the center surrounded with Huygens excitation planes in the CFS-PML experiments. y is constant and the snapshots are at $t = 250\Delta t$, $t = 650\Delta t$ and $t = 800\Delta t$. The snapshots illustrate the FDTD domain starting from the Huygens excitation planes and what is inside the Huygens excitation planes. The CFS-PML cells do not appear in the snapshots.

Chapter 8

Conclusion and Future Works

8.1 Conclusion

The HABC is a new boundary condition that is simpler to implement than the most widely used ABC the CFS-PML. The HABC is based on the idea of canceling the outgoing wave by introducing the opposite field on the Huygens surface. This thesis showed that the HABC is comparable to the CFS-PML in absorbing the high frequency traveling waves where the domain is large and the reflection of the evanescent waves is negligible. This can be achieved even when the HABC is placed 3 cells away from the object instead of the outer boundary. Since the HABC can be placed close to the structure in the FDTD domain, it is possible to combine it with other ABCs. The combination of the HABC and the SM ABC is effective because the HABC can absorb the high frequency traveling waves and the SM ABC can absorb the low frequency evanescent waves. The SM-HABC can simulate the free space with small number of stretched cells.

For practical applications, the program should be executed within realistic computational time. That is why, in this thesis, the FDTD with the HABC were parallelized on GPUs and shared memory architecture. This thesis introduced two GPGPU implementations of the HABC and compared their performance with the performance of the GPGPU implementation of the CFS-PML. The performance of the GPGPU implementation of the HABC was better than the performance of the GPGPU implementation of the CFS-PML over all selections of the number of threads per block.

The HABC code was scaled on the shared memory architecture in this thesis. The experiments showed that the speedup ratio of the HABC codes is higher than the CFS-PML codes. Also, the efficiencies of the HABC OpenMP codes in strong scaling and weak scaling are equivalent. And similarly, the CFS-PML OpenMP codes have the same efficiency in strong scaling and weak scaling. In both strong scaling and weak scaling the HABC is more efficient than the CFS-PML through all numbers of threads used to parallelize the codes. However, the speed of the memory access highly affects the total computational time in the FDTD calculations, since The FDTD method requires large amount of data.

Finally, in this thesis, the SM-HABC was tested with the human body and the Huygens excitation. The Huygens excitation heated up the human body as a method of applying the hyperthermia. The results of the SM-HABC experiments ware comparable to the results of the CFS-PML experiments. In addition, the time required to simulate the whole human body with the SM-HABC is better than the time required to simulate the whole human body with the CFS-PML.

8.2 Suggested Future Works

The suggested future works for the HABC are:

- Implementing the HABC with higher order operator instead of the first order Higdon's operator to improve the absorption of the high frequency traveling waves.
- Using cascaded HABCs in the FDTD domain, which will improve the performance of the results.
- Implementing the HABC on distributed memory architecture using MPI and study the performance of the implementation.
- Developing GPGPU implementation of the HABC on multiple grids to simulate larger FDTD domains.
- Accelerating the HABC using electromagnetic field simulators such as EMPIRE XCcel [55].

Appendix A

List of Publications

Journals, Submitted

- Jean-Pierre Bérenger, Hanan Almeer and Fumie Costen, "The Stretched-Mesh Huygens Absorbing Boundary Condition (SM-HABC)," Journal of IEEE Transactions on Antennas and Propagation.

Conference Publications, Refereed

- Hanan Almeer, Fumie Costen and Jean-Pierre Bérenger, "Development of a CUDA Implementation of the 3D Huygens Boundary Condition," 2013 Loughborough Antennas & Propagation Conference, 10th & 11th November 2013, Loughborough, UK.
- Hanan Almeer, Fumie Costen, Jean-Pierre Bérenger, Ryutaro Himeno and Hideo Yokota, "Scaling the Huygens Absorbing Boundary Condition Code on Shared Memory Architecture," Ninth International Conference on Computation in Electromagnetics, 31 March – 1 April 2014, London, UK.
- Hanan Almeer, Fumie Costen, Jean-Pierre Bérenger, Ryutaro Himeno and Hideo Yokota, "The Performance of the CUDA Implementations of HABC and CFS-PML ABCs on GPU Hardware," The 8th European Conference on Antennas and Propagation, 6–11 APRIL 2014, Hague, Netherlands.

Bibliography

[1] J.-P. Bérenger. The Huygens surface method for implementing an incident wave in a FDTD computational domain Manual , July 16, 2013. 5, rue des docteurs Thiers, 26400, Crest, France.

[2] A. Taflove and S. Hagness. *Computational Electrodynamics: The Finite–Difference Time–Domain Method, third ed.* Artech House, Boston, MA, 2005.

[3] J.-P. Bérenger. On the huygens absorbing boundary conditions for electromagnetics. *J. Comp. Phys.*, 226:354–378, 2007.

[4] F. Costen J.-P. Bérenger. Application of the Huygens Absorbing Boundary Condition to Wave-Structure Interaction Problems , July, 2010. IEEE AP-S Int. Symp. USNC/ CNC/URSI Radio Science Meeting.

[5] L. Livermore National Laboratory B. Barney. Introduction to Parallel Computing , 2013 (accessed July. 22, 2013). `https://computing.llnl.gov/tutorials/parallel_comp/#Whatis`.

[6] T. Lanfear J. Purches. Approaches to GPU Computing , July 23, 2013. The University of Manchester.

[7] GridAUTH. Advanced Computing Services. , 2013 (accessed July. 24, 2013). `http://linksceem.eu/ls2/images/stories/uploads/OpenMP_-_Joint_HP-SEE_LinkSCEEM-2_and_PRACE_HPC_Summer_Training.pdf`.

[8] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. NVIDIA Tesla: A unified graphics and computing architecture. *IEEE Micro*, 28(2):39 –55, March-April 2008.

[9] D. Kirk and W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach (Applications of GPU Computing Series).* Morgan Kaufmann, 2010.

[10] Matthew Livesey, James Francis Stack, Jr., Fumie Costen, Takeshi Nanri, Norimasa Nakashima, Seiji Fujino . Development of a cuda implementation of the 3d fdtd method. *IEEE Antennas Propag. Magazine*, 2012.

[11] Fumie Costen. *High Speed Computational Modelling in the Application of UWB Signals.* PhD thesis, Kyoto University, Japan, 2005.

[12] S. Gonzalez Garcia, A. Rubio Bretones, B. Garcia Olmedo, and R. Gomez Martin. *Finite Difference Time Domain Methods*, pages 91–132. WIT Press, 2003.

[13] K. S. Yee. Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media. *IEEE Transactions on Antennas and Propagation*, AP-14, 1966.

[14] R. F. Harrington. *Field Computation by Moment Methods.* The McMillan Company, New York, 1968.

[15] P. B. Johns and R. L. Beurle. Numerical solution of two dimensional scattering problems using a transmission-line matrix. *Proceedings of IEE*, 118(12):1203–1208, 1971.

[16] M. N. O. Sadku. A simple introduction to finite element analysis of electromagnetics problems. *IEEE Transactions on Education*, 32(2):85–93, 1989.

[17] Hasan Rouf. *Unconditionally Stable Finite Difference Time Domain Methods for Frequency Dependent Media.* PhD thesis, University of Manchester, UK, 2010.

[18] G. Mur. Absorbing boundary conditions for the finite-difference approximation of the time-domain electromagnetic-field equations. *IEEE Transactions on Electromagnetic Compatibility*, 23:377–382, 1981.

[19] R. L. Higdon. Absorbing boundary conditions for difference approximations to the multi-dimentional wave equation. *Mathematics of Computation*, 47:437–459, 1986.

[20] R. L. Higdon. Numerical absorbing boundary conditions for the wave equation. *Mathematics of Computation*, 49:65–90, 1987.

[21] S. M. Rao and E. K. Miller. *Time Domain Electromagnetics.* Academic Press, 1999.

[22] J.-P. Bérenger. A perfectly matched layer for the absorption of electromagnetic waves. *J. Comp. Phys.*, 114:185–200, 1994.

[23] S. Gedney. An anisotropic perfectly matched layer – absorbing medium for the truncation of FDTD lattices. *IEEE Transactions on Antennas and Propagation*, 44(12):1630–1639, December 1996.

[24] W. Chew and W. Wood. A 3-D perfectly matched medium from modified Maxwell's equations with stretched coordinates. *Microwave and Optical Technology Letters*, 7:599–604, 1994.

[25] C. Rapaport. Perfectly matched absorbing boundary conditions based on anisotropic lossy mapping of space. *IEEE Microwave and Guided Wave Letters*, 5:90–92, 1995.

[26] J. Roden and S. Gedney. Convolution PML (CPML): An efficient FDTD implementation of the CFS-PML for arbitrary medium. *Microwave and Optical Technology Letters*, 27(5):334–339, 2000.

[27] B. Engquist and A. Majda. Absorbing boundary conditions for the numerical simulation of waves. *Mathematics of Computation*, 31:629–651, 1977.

[28] M. Kuzuoglu and R. Mittra. Frequency dependence of the constitutive parameters of causal perfectly matched absorbers. *IEEE Micr. Guid. Wave Let.*, 6(12):447–449, Dec. 1996.

[29] F.W. Smith D.E. Merewether, R. Fisher. On implementing a numeric huygen's source scheme in a finite difference program to illuminate scattering bodies. *IEEE Transactions on Nuclear Science*, NS-27:1829–1833, 1980.

[30] F. Costen J.-P. Bérenger. 3D Huygens ABC . US Patent,.

[31] I. Wayan Sudiarta. An absorbing boundary condition for fdtd truncation using multiple absorbing surfaces. *IEEE Trans. Antennas Propag.*, 51:3268–3275, 2003.

[32] R. E. Diaz and I. Scherbatko. A simple stackable re-radiating boundary condition (rrbc) for fdtd. *IEEE Antennas Propag. Magazine*, 46(1):124–130, 2004.

[33] A. Taflove and S. Hagness. *Computational Electromagnetics. The Finite–Difference Time–Domain Method.* Artech House, Boston, MA, 2000.

[34] The University of Manchester: Research Computing. , 2013 (accessed July. 24, 2013). `http://www.rcs.manchester.ac.uk/services/computational/parallel`.

[35] NVIDIA. WHAT IS GPU COMPUTING? , 2013 (accessed July. 23, 2013). `http://www.nvidia.com/object/what-is-gpu-computing.html`.

[36] NVIDIA UK. WHAT IS GPU COMPUTING? , 2013 (accessed July. 23, 2013). `http://www.nvidia.co.uk/object/gpu-computing-uk.html`.

[37] TOP500: SUPERCOMPUTER SITES , 2013 (accessed July. 23, 2013). `http://www.top500.org/`.

[38] T. Lanfear J. Purches. GPU Computing , July 23, 2013. The University of Manchester.

[39] NVIDIA. NVIDIA's next generation CUDA compute architecture: Fermi. , 2013 (accessed July. 23, 2013). `http://www.nvidia.co.uk/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf`.

[40] NVIDIA. Tesla Kepler GPU Accelerators. , 2013 (accessed July. 23, 2013). `http://www.nvidia.co.uk/content/tesla/pdf/Tesla-KSeries-Overview-LR.pdf`.

[41] The University of Manchester. OpenMP: Getting more from Multicore. , 2013 (accessed July. 24, 2013). `http://www.rcs.manchester.ac.uk/courses`.

[42] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming.* Addison-Wesley Professional, 2010.

[43] NVIDIA. NVIDIA CUDA C Programming Guide 4.1 . Technical report, November 2011. `http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf`.

[44] A. Giraud. Theoretical and numerical analysis of three approaches to the gpgpu application of the explicit fdtd method. Master's thesis, The University of Manchester, 2012.

[45] chooru code: Marginalia on Programming , 2013 (accessed December. 11, 2013). `http://choorucode.com/2011/02/16/cuda-dim3/`.

[46] K computer: RIKEN Advanced Institute for Computational Science. , 2013 (accessed September. 5, 2013). `http://www.aics.riken.jp/en/`.

[47] M. Kim. Scaling Theory and Machine Abstractions. , 2013 (accessed December 10, 2013). `http://www.cs.columbia.edu/~martha/courses/4130/au12/scaling-theory.pdf`.

[48] G. M. Amdahl. Validity of the Single-Processor Approach to Achieving LargeScale Computer Capabilities. *AFIPS Conference Proceedings*, 30:483–485, 1967.

[49] The Computational Science Education Reference Desk (CSERD): Parallel Speedup Tutorial. , 2013 (accessed July 29, 2013). `http://www.shodor.org/cserd/Resources/Tutorials/Speedup/`.

[50] Stack Overflow: Weak vs Strong Scaling Speedup and Efficiency. , 2013 (accessed July 29, 2013). `http://stackoverflow.com/questions/16717556/weak-vs-strong-scaling-speedup-and-efficiency`.

[51] American Cancer Society: Hyperthermia to Treat Cancer. , 2013 (accessed November. 8, 2013). `http://www.cancer.org/treatment/treatmentsandsideeffects/treatmenttypes/hyperthermia`.

[52] National Cancer Institute: Hyperthermia in Cancer Treatment. , 2013 (accessed November. 8, 2013). `http://www.cancer.gov/cancertopics/factsheet/Therapy/hyperthermia`.

[53] Cancer Active: Hyperthermia and the treatment of cancer. , 2013 (accessed November. 8, 2013). `http://www.canceractive.com/cancer-active-page-link.aspx?n=992`.

[54] Stanford Medicine, Stanford Cancer Center: Hyperthermia For Cancer Treatment. , 2013 (accessed November. 8, 2013). `http://cancer.stanford.edu/information/cancerTreatment/methods/hyperthermia.html`.

[55] EMPIRE XCcel. , 2014 (accessed March. 25, 2014). `http://www.empire.de/`.