January 2013

Microwave Tomography Using Stochastic Optimization And High Performance Computing

Michael William Holman

MICROWAVE TOMOGRAPHY USING STOCHASTIC OPTIMIZATION AND HIGH PERFORMANCE COMPUTING

by

Michael William Holman
Bachelor of Science, University of North Dakota, 2012

A Thesis
Submitted to the Graduate Faculty

of the

University of North Dakota

In partial fulfillment of the requirements

for the degree of

Master of Science

Grand Forks, North Dakota
August
2013

This thesis, submitted by Michael Holman in partial fulfilment of the requirements for the Degree of Master of Science from the University of North Dakota, has been read by the Faculty Advisory Committee under whom the work has been done, and is hereby approved.

1. Reviewer: Sima Noghanian

2. Reviewer: Travis Desell

3. Reviewer: Reza Fazel-Rezai

This thesis is being submitted by the appointed advisory committee as having met all of the requirements of the Graduate School at the University of North Dakota and is hereby approved.

Wayne E. Swisher, Ph.D.
Interim Dean of the Graduate School

Date

Title	Microwave Tomography Using Stochastic Optimization and High Performance Computing
Department	Electrical Engineering
Degree	Master of Science

In presenting this thesis in partial fulfillment of the requirements for a graduate degree from the University of North Dakota, I agree that the library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by the professor who supervised my thesis work or, in her absence, by the Chairperson of the department or the dean of the Graduate School. It is understood that any copying or publication or other use of this thesis or part thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of North Dakota in any scholarly use which may be made of any material in my thesis.

Michael Holman

July 18, 2013

Abstract

This thesis discusses the application of parallel computing in microwave tomography for detection and imaging of dielectric objects. The main focus is on microwave tomography with the use of a parallelized Finite Difference Time Domain (FDTD) forward solver in conjunction with non-linear stochastic optimization based inverse solvers. Because such solvers require very heavy computation, their investigation has been limited in favour of deterministic inverse solvers that make use of assumptions and approximations of the imaging target. Without the use of linearization assumptions, a non-linear stochastic microwave tomography system is able to resolve targets of arbitrary permittivity contrast profiles while avoiding convergence to local minima of the microwave tomography optimization space. This work is focused on ameliorating this computational load with the use of heavy parallelization. The presented microwave tomography system is capable of modelling complex, heterogeneous, and dispersive media using the Debye model. A detailed explanation of the dispersive FDTD is presented herein. The system uses scattered field data due to multiple excitation angles, frequencies, and observation angles in order to improve target resolution, reduce the ill-posedness of the microwave tomography inverse problem, and improve the accuracy of the complex permittivity profile of the imaging target.

The FDTD forward solver is parallelized with the use of the Common Unified Device Architecture (CUDA) programming model developed by NVIDIA corporation. In the forward solver, the time stepping of the fields are computed on a Graphics Processing Unit (GPU). In addition the inverse solver makes use of the Message Passing Interface (MPI) system to distribute computation across multiple work stations. The FDTD method was chosen due to its ease of parallelization using GPU computing, in addition to its ability to simulate wideband excitation signals during a single forward simulation.

We investigated the use of distributed Particle Swarm Optimization (PSO) and Differential Evolution (DE) methods in the inverse solver for this microwave tomography system. In these optimization algorithms, candidate solutions are farmed out to separate workstations to be evaluated. As fitness evaluations are returned asynchronously, the optimization algorithm updates the population of candidate solutions and gives new candidate solutions to

be evaluated to open workstations. In this manner, we used a total of eight graphics processing units during optimization with minimal downtime.

Presented in this thesis is a microwave tomography algorithm that does not rely on linearization assumptions, capable of imaging a target in a reasonable amount of time for clinical applications. The proposed algorithm was tested using numerical phantoms that with material parameters similar to what one would find in normal or malignant human tissue.

To my parents for their unwavering support and to my girlfriend Cassandra.

Acknowledgements

First and foremost, I would like to thank my thesis advisor, Dr. Sima Noghanian, for encouraging me to enter the graduate program, and for her continued support, direction, and guidance though the course of this research. I also would like to express my appreciation for the guidance and great help I received from my thesis committee members Dr. Travis Desell and Dr. Reza Fazel-Rezai.

I would like to acknowledge the University of North Dakota Graduate School for providing funding for this research and Department of Electrical Engineering and Univeristy of North Dakota Computational Research Center for providing the facilities.

I also would like to thank Dr. Abas Sabouni for his help.

Finally, my thanks goes to my parents, family and friends for all their support and patience.

Contents

List of Figures

List of Tables

Glossary

API - Application Programming Interface; Functions developed to facilitate complex software interactions, usually packaged in a library.

CUDA - Common Unified Device Architecture; A parallel computing platform and programming model invented by NVidia for the purpose of performing general computation on graphics processing units.

DE - Differential Evolution; A evolutionary population based stochastic optimization method.

FDTD - Finite Difference Time Domain; A numerical method for solving Maxwell's equations.

$(FD)^2TD$ - Frequency Dependant Finite Time Domain Method - A modification of FDTD that is able to simulate media whose material parameters change as a function of frequency.

FE - Finite Element Method; A numerical Maxwell's equation (and more generally any differential equation) simulation method.

GA - Genetic Algorithm; A stochastic optimization method inspired by and designed to emulate biological evolution.

GPU - Graphics Processing Unit; A hardware add-on for computers capable of large scale parallelization.

HPC - High Performance Computing; A general term referring to the use of many computing resources in parallel for high computational throughput.

MoM - Method of Moments; A frequency domain numerical Maxwell's equation simulation method.

MPI - Message Passing Interface; A standardized and portable system for passing messages between separate computers used for parallelization.

MRI - Magnetic Resonance Imaing; An imaging modality where the nuclear magnetic resonance is used to obtain high resolution images of biological materials.

MSE - Mean Square Error; An estimate of the error between two sets of data.

MWT - Microwave Tomography; An imaging method that uses microwave radiation to investigate a target and gives an image indicating the map of complex permittivity throughout the target.

PSO - Particle Swarm Optimization; An evolutionary population based stochastic optimization method.

RCS - Radar Cross Section; A far field measurement of the strength the scattered field from an object.

SDK - Software Development Kit; A set of software development tools that aid in creating applications for a specific purpose

1

Introduction

1.1 Electromagnetic Imaging

The use of electromagnetic radiation to for non-invasive imagery them has been a popular topic of research for many decades now. Electromagnetic imaging has a wide variety of potential usage including non-destructive testing of building materials in the realm of civil engineering [1], through-wall imaging [2], and medical imaging which shall be the primary focus of this thesis.

The ability of non-invasively imaging of objects has been an immensely important aspect of modern medicine since the discovery of X-ray imaging. Medical imaging has evolved to use other modalities such as acoustic waves in the case of ultrasound, and nuclear magnetic resonance in the case of Magnetic Resonance Imaging (MRI). Expanding this list to include microwave radiation presents a very worthwhile, yet difficult, challenge. While other imaging modalities can be thought to primarily detect differences in mass density of the target, microwave imaging relies on the differences in the dielectric properties of the target. With regards to medical imaging, this is attractive, due to the relatively large contrast between tissues, including contrast between cancerous tissue and normal tissue.

Current systems designed to image the body via sub-visible frequency radiation include microwave imaging (which includes microwave tomography and radar-based methods), electrical impedance tomography, diffuse optical tomography, MRI, etc. Of these, MRI and electrical impedance tomography have been developed sufficiently enough to be used in actual clinical settings. Of those two, MRI is the most developed technology and has revolutionized the field of medical imaging, and medicine at large. Electrical impedance tomography uses a collection of electrodes which are placed on the surface of an imaging target. Measurements are taken by injecting a current from one electrode and measuring the voltage caused by the resistivity of the target across the two electrodes. Another way of taking measurements is to apply a voltage across two electrodes and mea-

suring the resulting current [3]. This process is repeated across different electrodes until the resistance between all pairs of electrodes are known. From the measurement data, a map of the conductivity of the inside of the imaging target is constructed by a computer. Electrical impedance tomography shows promise in imaging the respiratory system as the conductivity of lung tissue changes based on the level of perfusion. Research into electrical impedance tomography has progressed significantly enough for commercial deployment, whereas a commercial system has been made available by Dräger Medical capable of real-time lung imaging [4]. Diffuse optical imaging uses an infra-red laser as its active element and optical fibres as its sensing elements. During measurement, the laser sends a light signal, usually near infra-red, inside the imaging target [5]. The light ray is scattered by the objects inside the target. Infra-red light is absorbed and scattered mostly by water, haemoglobin, and oxygenated haemoglobin. Because of the differences in the absorption rates of haemoglobin and oxygen haemoglobin, diffuse optical imaging shows potential in functional imaging, where we can see how well biological structures are operating based on their oxygenation levels. Diffuse optical imaging is also being investigated for the purpose of cancer detection [6]. In MRI, a strong magnetic field is applied to the imaging target which causes atoms inside the target to align themselves with the magnetic field. A radio frequency electromagnetic signal is applied to the imaging target and the response is picked up by receivers on the MRI. MRI offers the highest resolution among the previously mentioned imaging methods, with a finer resolution than even X-ray imaging methods [7].

In microwave imaging, both the sensors and active elements of the system are antennas designed to operate within the microwave spectrum. The hardware used for microwave imaging is similar to the hardware used in telecommunications, and thus it is relatively cheap. In microwave tomography (MWT), an imaging target (such as an extremity or other biological structure) is illuminated with microwave radiation. As this radiation penetrates through the imaging target it is refracted and scattered according to the electrical material parameters (permittivity and conductivity) of objects inside the imaging target. Other antennas situated around the imaging target sense this scattered radiation and pass this information to a computer which reconstructs a map of permittivity and conductivity inside the imaging target from the measured scattered radiation. The reconstruction of the material parameters from a known scattered field is a very difficult mathematical problem because it is ill-posed and ill-constrained as described in Chapter 3 of this thesis. Research into functional microwave imaging systems has historically been hamstrung by lack of sufficient computing power necessary to solve the inverse problem. However, advances in computer hardware, such as GPU computing and HPC clusters, offer tremendously increased speed in comparison to serial computer systems.

1.2 Microwave Tomography

As opposed to similar electromagnetic imaging methods such as radar-based imaging, the output of an MWT system offers quantitative information about the electrical material parameters of whatever object is being imaged. In radar-based imaging, the presence of contrast sources within the imaging target is sensed, although the actual material parameters of these contrast sources are unknown. The availability of this quantitative data of the electrical material parameters may aid diagnosis of certain pathologies. In breast cancer diagnosis, for instance, malignant tumours are often classified based on their irregular shapes, whereas benign structures have more regular shapes. In addition to their shapes, malignant tumours also have very different electrical parameters than the surrounding tissue. Microwave tomography also shows potential for diagnosis of cardiac pathologies, as the electrical material properties of tissues change due to events such as ischemia and infarction [8].

MWT is an attractive imaging modality due to the low cost of hardware needed for such systems. The sensors used in MWT are simple microwave antennas which can be manufactured relatively cheaply. Due to the proliferation of cellular telephone technology, microwave hardware is wide spread. The computational hardware may also be inexpensive, as the large market for GPUs, capable of extremely fast parallel computation, has made these devices affordable. Microwave tomography measurement is also comfortable for the patient and uses non-ionizing radiation.

MWT measurement hardware can use either multiple fixed antennas or two antennas that can rotate around the imaging target. Using multiple fixed antennas precludes the need for complicated mechanical systems required to rotate the observation antennas with good accuracy, however, this method introduces the possibility of mutual coupling between the antennas which may be difficult to deal with during image reconstruction. Furthermore, using moving antennas can cause error in the phase of the excitation, as well as errors in the excitation angle of the incident radiation. Phase error is compounded by the use of high frequencies and wideband excitation signals. Given this, the use of moving antennas is unsuitable for the frequencies of radiation our system uses. In [9], two horn antennas were used which were rotated in space about the imaging target which was a thorax of a diseased dog. The measurement stage of this system required 9 hours to complete, where the target was illuminated from 32 angles and measured at 18 observation angles and at 16 different heights for each illumination angle. This yielded 9216 points of data. Measurement methods using multiple antennas are able to take measurements in a very short amount of time in comparison. An MWT system at Dartmouth College [10] uses an array of monopole antennas at fixed locations located around the imaging target. In the previous two examples the MWT enclosure is filled with a matching fluid. This matching fluid has similar electrical characteristics as biological tissue and is used to reduce reflections of the radiation as it enters the imaging target. The MWT systems used at the University of Manitoba use arrays of multiple fixed highly directive and wideband Vivaldi antennas as the illuminating and sensing

elements [11]. These systems also do not use matching fluids, which cuts down on the cost of building the measurement apparatus.

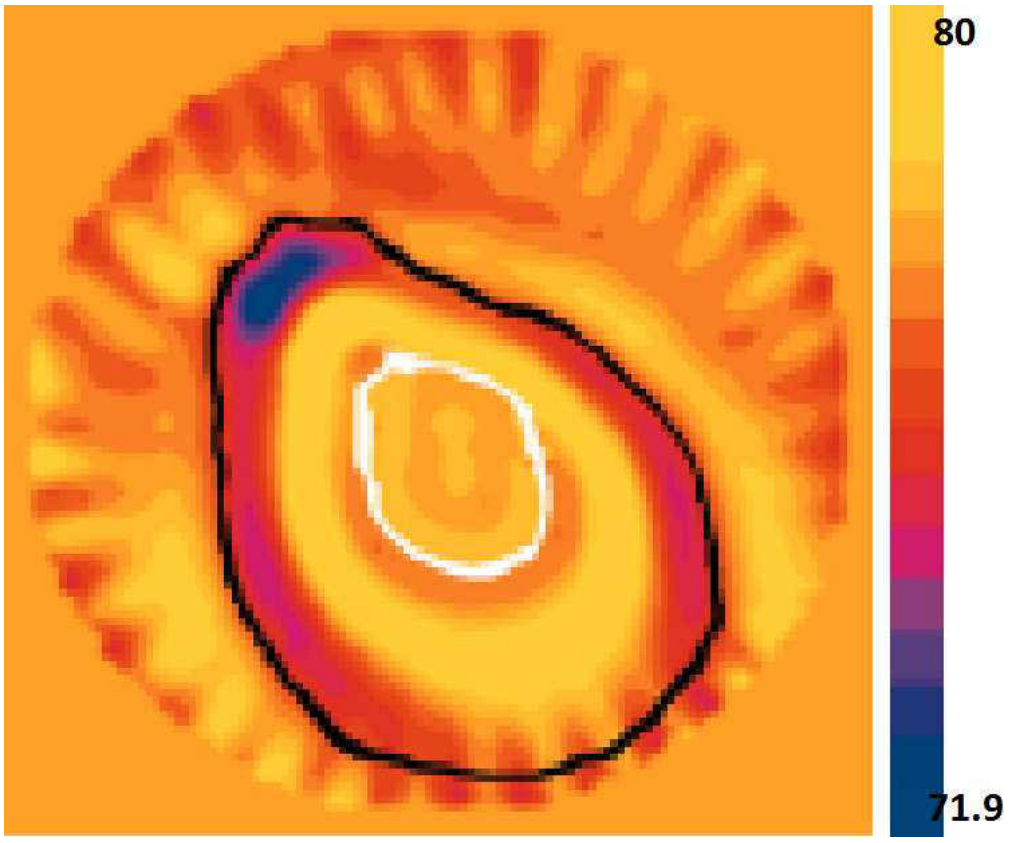


Figure 1.1: A map of permittivity of a dog thorax reconstructed using MWT methods [22].

Much of the research done into MWT is in developing inverse solvers. Because the MWT inverse problem is inherently ill-posed, mathematical methods must be employed to resolve the image. Most MWT methods have used deterministic iterative solvers in conjunction with linearization assumptions and regularizations. These linearization assumptions and regularizations transform the non-linear MWT problem into a linear one and ameliorate the ill-posedness thereof. Linearization techniques include the Born approximation and Rytov approximation. The Born approximation can intuitively be understood as the assumption that the total electric field inside the imaging target during illumination can be modelled as equal to the incident field. This assumption is appropriate for weak scatterers, but with stronger scatterers like highly conductive materials, the total field inside the imaging target can be expected to be very different from the incident field. Tikhonov regularization is used to ameliorate the ill-posedness of the problem [12]. *A-priori* information (such as information about the shape of the imaging

target) is also often used in the inverse solver. The MWT system proposed in [10] uses a Tikhonov regularized inverse solver. In [9], the Newton iterative scheme was used in conjunction with the standard Tikhonov regularization. An MWT image resolved using the Newton iterative scheme with a moving antenna measurement apparatus is shown in Figure 1.1. The use of Born [13] or Rytov [14] linearization allows the MWT inverse problem to be solved without the use of iterative methods which cuts down on the run time [15]. The system proposed in [16] use the Born approximation to solve the MWT inverse problem directly.

By their nature, these methods of solving the MWT inverse problem introduce errors into image reconstruction. The use of regularization methods manifests in a "smoothing out" of the image. This image smoothing means that high levels of contrast (where the permittivity and conductivity of materials changes rapidly with respect to location) cannot be detected, and thus, may reduce the resolution of the reconstruction. The use of linearization methods like the Born approximation can introduce significant modelling error if t he weak scatterer assumption turns out to be incorrect for a given problem. MWT inverse solvers that do not rely on these methods have seen little investigation as such inverse solvers require global optimization methods coupled with full-wave forward solvers which invariably result in long run times of the reconstruction algorithm. One such system, however, has been developed and proposed in [17]. In that system, the reconstruction algorithm used a Genetic Algorithm (GA) in conjunction with a numerical Maxwell's equations simulation program. Image reconstruction took a total of one week for one image.

1.2.1 GPU Computing

In recent years, the speed of Central Processing Units (CPU) has stopped increasing as fundamental properties of the universe limit increasing the frequency of processors. In order to meet increasing demands for computational power, researchers look increasingly towards parallel computation to satisfy the ever increasing computational needs imposed by numerical simulation, medical imaging, computational finance, defence and intelligence, and countless other applications. Multi-core CPUs have become the de-facto standard computer architecture as single core CPUs become more and more obsolete. Parallel computing has taken many forms, including the ubiquitous multi-core CPU, volunteer computing, HPC clusters, and the titular general purpose GPU.

GPUs were originally developed to handle computer graphics processing. The rendering of computer graphics is a very parallelizable task, involving a large amount of instructions per frame rendered. As such, GPUs are inherently parallel devices. The emergence of computer-based entertainment including desktop computer video games has spurred the development of extremely powerful GPUs. Whereas currently available CPUs can have four to eight cores, commercially available GPUs are equipped with thousands of cores. As a result of this extreme parallel architecture, the theoretical performance when measured in floating point operations per second (flops) vastly exceeds

that of serial central processors. Figure 1.2 illustrates the computing power of commercially available GPUs vs. CPUs. The cores of CPUs, however, are much more complex than those of GPUs.

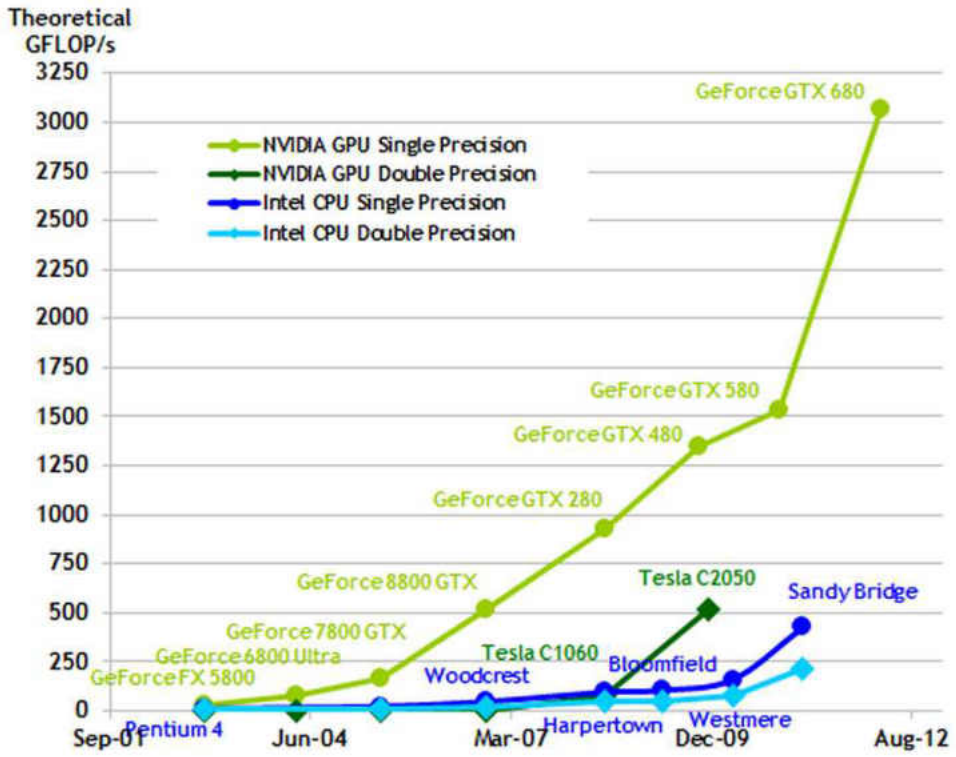


Figure 1.2: Comparison of raw computing power of GPUs vs. CPUs. Sandy bridge and the others are CPU architectures [18].

Given the exceptional computing power of GPUs it is no wonder that researchers have turned their efforts to using GPUs for applications other than computer graphics. The use of GPUs for applications other than computer graphics is known as General Purpose Graphics Processing Unit computing (GPGPU). GPGPU computing is best suited for tasks that are highly parallelizable, namely tasks that involve minimal communication between processors. Parallelizability is classified as either fine-grained parallelization, coarse-grained parallelization, or embarrassingly parallelizable. In fine-grained parallelization, data must be transferred often after a relatively small amount of computation. Coarse-grained parallelization entails less frequent data transfers, wherein data is only transferred after a large amount of computation. For an algorithm to be embarrassingly parallel, there is very little communication between processors. Another definition of embarrassing parallelizability is that data transfer for a computing node is only limited to nearby computing nodes, such that any single computing node needs only to communicate to a few other nodes.

The FDTD method which is used as the forward solver in our MWT system has been described as an embarrassingly parallelizable algorithm [19]. Given that FDTD explicitly models the electric field in space at discrete time steps and uses the differential form of Maxwell's equations to update the fields, the embarrassing parallelizability of the FDTD can intuitively be explained by the principle of locality. The principle of locality states that objects in the universe are only effected by objects in their immediate vicinity, thus the behaviour of the electric field at one location is only dependent on the fields at adjacent locations.

When GPGPU computing first came into use, programmers were restricted to using APIs (Application Programming Interfaces, functions developed to facilitate complex software interactions) developed exclusively for graphics applications such as OpenGL. In order to be used for general computation on GPUs, problems needed to cast in terms of triangles, textures, and other objects commonly used in graphics rendering.

To meet the trend of GPGPU computing and increase the capabilities of GPUs for traditional graphics processing hardware and software developers worked to expand the capabilities of GPUs from being fixed-function devices to fully programmable computing platforms. Nvidia's GeForce 8 series of GPUs are designed for computer video game graphics acceleration and are thus wide spread. The GeForce 8 series of GPUs all come with support for single precision floating point operations. GPGPU computing has become widespread enough to warrant its own market for hardware. The Nvidia Tesla series of GPUs was developed exclusively for general purpose computing. Early versions of the Tesla, in fact, did not even support graphics display, although more recent Tesla GPUs have this ability. The Tesla series most apparently diverges from standard GPUs in that they devote much more chip space to double precision (64-bit) operations, although seemingly this capability comes at the expense of raw computational throughput as shown in Figure 1.2 and increased cost where the Tesla C2050 is capable of around 500 GFLOP/s and the GeForceGTX 480 is capable of more than 2500 GFLOP/s. In addition to double-precision floating point support the Tesla also has increased memory, faster communication from the GPU to the motherboard, and many other features that optimize the Tesla for general purpose computing. Another recent interesting development in graphics processing hardware comes from the company AMD in the form of the AMD Accelerated Computation Unit (ACU) also known as AMD Fusion [20]. The AMD APU combines the CPU and GPU on a single die where both the CPU and GPU have access to a shared memory cache. Although developed for the consumer market, the AMD APU shows quite a bit of potential for GPGPU applications. Whereas in traditional separate CPU/GPU architectures memory must be transferred over buses in a very time consuming manner, the shared memory between CPU and GPU of the AMD APU precludes the need for these memory transfers. In some algorithm the need to transfer memory from CPU to GPU slows down the algorithm, resulting in a "memory bottleneck". In addition to this, AMD APUs support OpenCL, increasing the ease of programming them for general purpose computation. For these reasons, the use of AMD

APU for general purpose computing is a research topic receiving much attention at the time of writing this thesis [21].

1.2.2 High Performance Computing

A more common method of parallel computing is in the form of using multiple CPUs that work in parallel. High Performance Computing (HPC) clusters are used to run parallel programs on homogeneous computing environments. In a computer cluster, multiple similar computers (called nodes) are connected to a Local Area Network (LAN) to communicate with each other. The similarity of the computers aids in the running of synchronous parallel programs. If one were to run a synchronous parallel algorithm using computers of different speeds, the faster computing nodes would need to wait for the slower nodes to catch up, thereby negating their increased speed. However, asynchronous parallel algorithms are able to fully utilize the speed of all computing nodes even with great differences in computing power.

Parallel programs written to be executed on HPC clusters use the Message Passing Interface (MPI) to communicate between nodes of the cluster. MPI is a standardized method of providing the synchronization and communication needed to run parallel algorithms. Most implementations of MPI are designed for C, C++, and Fortran although implementations exist for other languages such as Java. One popular implementation of MPI is MVAPICH which is the implementation used in our system. In addition to being used for computer clusters, MPI is also used to parallelize algorithms on multi-core processors.

The University of North Dakota has a number of computing clusters for use in research. Of these clusters, the Shale cluster is equipped with CUDA enabled GPUs. Four computing nodes have two Tesla C1060 GPUs each, eight GPUs total. Our system is able to fully utilize all eight GPUs of the cluster during image resolution, offering increased speed in addition to the acceleration offered by GPU implementation of the forward solver.

1.3 Motivation

The large amount of contrast between the dielectric properties of malignant, benign, and normal tissue has made the development of a clinical microwave imaging system a high priority among researchers. MWT has been considered for detection of cardiac diseases [22]. The motivation for this is that during physiological events such as myocardial ischemia or infarction, the electrical properties of the tissue change from their normal state [23]. However, the use of MWT for diagnosis of breast cancer has received the greatest amount of attention from researchers. While it is well known that early detection of breast cancer is the biggest factor in successful treatment, current imaging

methods are imperfect, having relatively high false-negative and false-positive rates [24]. The measured ratio in permittivity of malignant to healthy fatty breast tissue has been reported to be on the order of 10:1 [25]. With such a high contrast between healthy and malignant tissue, MWT may have the potential to differentiate malignant tumours and benign tumours based on their electrical material parameters. More recent research indicates that the difference between the permittivity of tumour tissue and normal fibroglandular tissue is on the order of a mere 10% [26].

Current breast cancer detection methods, the most widely used among which is X-Ray mammography, offer limited sensitivity in detection, where the failure to detect malignant tumours using X-ray mammography has been reported to be from 4% up to 34% [27]. In addition to the failure to detect tumours, X-ray mammography and MRI have fairly high false-positive rates [24]. The positive prediction value of X-ray mammography, defined as the ratio of correctly diagnosed cases of malignant breast cancer to the total amount of patients diagnosed with malignant breast cancer is 85.7% [26]. In the case of MRI, however, this value decreases to 73.6%. In other words, as high as 26% of patients who have been diagnosed with malignant tumours can expect their diagnosis to be a false-positive. From these statistics, the need for supplementary diagnostic methods for breast cancer is apparent.

MWT has been proposed as such a method. Because MWT offers poor resolution in comparison to X-ray mammography and MRI, a MWT system in clinical deployment would most likely be used as a supplement to current breast imaging methods such as X-ray mammography. By offering quantitative data about the material parameters of the image, it may be possible to differentiate malignant tumours from benign ones. Currently malignant tumours are differentiated from other structures such as calcifications based on their shapes since they often have highly irregular shapes. The difference in permittivity and conductivity of malignant tumours in comparison to other breast tissue offers another method to classify positive diagnoses.

Other considerations that make MWT an attractive diagnostic method for breast cancer include the fact that the MWT does not use ionizing radiation and is comfortable for the patient. In MWT, usually Ultra-Wide Band (UWB) technology is used. UWB covers frequencies from 1 GHz to 10 GHz and follows FCC's low power regulation [28]. The dosage of this radiation received during MWT measurement pales in comparison to the exposure to Wi-Fi signals, cell phone signals, and other signals emitted from consumer electronics. Because of the non-ionizing nature of microwaves, there is minimum risk of MWT screenings causing further cancer. MWT also requires no breast compression, which is required for X-ray mammography and can cause discomfort. This lack of discomfort may have the effect of encouraging women to have more frequent breast screenings.

As stated in Section 1.2, most MWT systems in literature rely on regularization and/or linearization assumptions in order to deal with the ill-posedness of the inverse

problem. These methods introduce error in the reconstructed images. Because the contrast in permittivity between malignant and benign tissue may be as low as 10%, this error may make MWT using regularization/linearization assumptions infeasible for breast cancer detection. In a clinical setting, error in an image generated by MWT has the potential for serious ramifications such as misdiagnosis. MWT systems in literature that do not use these regularization techniques require inordinate amounts of time for reconstruction, which also precludes the clinical deployment of MWT for breast cancer diagnosis. Such systems require full-wave Maxwell's equations forward solvers and global optimization methods where the forward solver must be called several times per image reconstruction. We have thus set out to create an MWT system that does not rely on linearization or regularization assumptions while keeping reconstruction times more manageable than previous similar systems. Parallel computing is employed in both the inverse solver and forward solver of our system in order to meet the demands for computing power required for such a system. Previous parallel MWT systems have relied on MPI to parallelize both the forward and inverse solvers [29]. The availability of low-cost GPUs has motivated us to implement a GPU parallelized forward solver which does not require expensive HPC clusters.

2

Microwave Tomography Forward Solver

Traditionally, inverse problems are solved with a combination of a forward solver and an iterative optimization technique. In the context of microwave imaging, the forward problem is defined as finding the scattered-field produced by a known scatterer, whereas the inverse problem is finding the complex permittivity profile of the scatterer, with the scattered-field being known. The forward problem is, of course, much easier to solve than the inverse problem. The forward problem is usually solved using numerical algorithms based on Maxwell's equations. Despite the heavy research done into forward problem solvers, the computational load required for numerical Maxwell's equations solvers is high. This high computational cost of forward solvers has led researchers to explore solutions that make no use of numerical forward solvers [30], or that use simplified, non full-wave, forward solvers [31]. Inverse solvers that don't rely on forward solvers invariably use approximations such as the Born approximation or Rytov approximation [32]. These approximations rely on the assumption that, inside of a scatterer, the total electric field can be approximated to be equal to the incident-field. This works well when dealing with weak scatterers, but fails when there is a large amount of contrast within the target. In the context of MWT for breast cancer detection, the use of these approximations may be inappropriate. This is due to the fact that the measured difference in permittivity of malignant tissue to can be as high as 10:1 [25]. Indeed, as mentioned earlier, the high contrast between tumour tissue and background fatty breast tissue is one of the primary motivations for using MWT as an imaging modality for breast cancer detection.

Full-wave solvers such as Method of Moments (MoM), FDTD, or Finite Element (FE) are capable of modelling all electromagnetic phenomena inside of a chosen area of interest. They are thus able to fully model the scattering of an arbitrarily complex imaging target, with error chiefly coming from numerical approximation. This is favourable for the purpose of breast cancer detection, given that breast tissue is highly heterogeneous and filled with strong scatterers. In particular, one phenomena that is hard to

model with direct inverse solving is the existence of secondary scattering. That is where scattered radiation from one source encounters another contrast source and is scattered further. Full wave solvers can model this phenomena accurately. As mentioned before, full-wave solvers have the disadvantage of being computationally intensive. This can be ameliorated with parallelization. The FDTD method, in particular, is very amenable to parallelization, with GPU parallelization in particular. The amenability of FDTD towards GPU acceleration, in addition to the ability of FDTD to simulate wide-band signals in a single forward solve, is the reason that we have chosen FDTD as our forward solver in our proposed MWT system. The following sections will explain the FDTD algorithm we used, the GPU parallelization of the FDTD algorithm, and the FDTD forward solver's integration into the large MWT inverse solver.

Our proposed MWT algorithm uses a stochastic optimization inverse solver which works in conjunction with a full-wave forward solver. In the context of MWT, the forward solver takes a candidate solution given by the inverse solver as input and returns a fitness value. This fitness value is related to how closely the scattered-fields created by the candidate image matches the known measured scattered-fields of one's imaging target which was interrogated during the initial measurement stage. The FDTD method is used in our proposed system to provide the scattered-field of the candidate solution that is proposed by the inverse solver. In our system, the scattered-field is calculated across multiple observation points, for multiple incident angles of illuminating radiation, for and multiple frequencies. Due to the ill-posedness of the inverse problem, inclusion of this extra information aids the convergence of the inverse solver, both in the time it takes to converge on a solution, and the correctness of the solution. Within the larger MWT system, the forward solver is implemented as a function module which takes an array of real numbers corresponding to the proposed image and returns a single real number, the fitness value. This fitness value is the mean squared error of the electric field across all observation points, all illumination angles, and all frequencies. The equation thereof is given below in equation (2.1):

$$error = \sum_{\Phi=\Phi_0}^{\Phi_{max}} \sum_{\Theta=\Theta_0}^{\Theta_{max}} \sum_{f=f_0}^{f_{max}} (E_{meas} - E_{simulated})^2 \tag{2.1}$$

In equation (2.1), Φ is the incident angle of illuminating radiation, Θ is the observation angle, f is the frequency, Φ_{max} is the maximum incident illuminating radiation angle, Θ_{max} is the maximum index of the observation angle, f_{max} is the maximum frequency taken, and *error* is the mean squared error of the radiation pattern of the candidate solution versus the measured radiation pattern. Φ_0 is the minimum incident illuminating radiation angle, Θ_0 is the minimum observation angle and f_0 is the minimum frequency.

2.1 Finite Difference Time Domain Forward Solver

FDTD, as its name would suggest, models the electric and magnetic fields in the time domain. To elaborate, in FDTD, a region of space known as the problem space is discretized into points. The electric and magnetic fields are defined only on these points and are made to rigorously obey Maxwell's equations in differential form:

$$\nabla \times \vec{H} = \frac{\partial \vec{D}}{\partial t} + \sigma \vec{E} \tag{2.2}$$

$$\nabla \times \vec{E} = -\mu \frac{\partial \vec{H}}{\partial t} - \vec{M} \tag{2.3}$$

$$\nabla \cdot \vec{D} = \rho_e \tag{2.4}$$

$$\nabla \cdot \vec{B} = \rho_m \tag{2.5}$$

$$\vec{D} = \epsilon \vec{E} \tag{2.6}$$

$$\vec{B} = \mu \vec{H} \tag{2.7}$$

In equations (2.2) - (2.7) $\vec{H}$ is the magnetic field intensity, $\vec{E}$ is the electric field intensity, $\vec{D}$ is the electric displacement field defined in equation (2.6), $\vec{B}$ is the magnetic displacement field defined in (2.7), ρ_e is electric charge density, ρ_m is the magnetic charge density, $\vec{J}$ is the electric current density, $\vec{M}$ is the magnetic current density, and t is time. Although there is no such thing as magnetic current or magnetic charge in physical reality, the two quantities are often used in solving equivalent electromagnetic problems. Equation (2.2) is Ampere's law, equation (2.3) is Faraday's law, equations (2.4) and (2.5) are Gauss's law for both the electric and magnetic fields. Only (2.2) and (2.3) are explicitly enforced in the FDTD method, as there is the assumption that there are no free electric or magnetic charges in the area of interest. ϵ is permittivity where the permittivity of free space, ϵ_0 is 8.854×10^{-12}. μ is magnetic permeability, where the permeability of free space, μ_0 is $4\pi \times 10^{-7}$. σ is conductivity.

The problem space is discretized according to the scheme put forth by Kane Yee [33] called the Yee cell. In the Yee cell, two sets of points are defined, one for the electric

field and one for the magnetic field. Points of the electric field are distanced from each other in the x, y, and z directions by a constant and finite distance denoted Δx, Δy, and Δz. Recent improvements to the FDTD method have changed this paradigm by allowing for adaptive meshing, and thus non-uniform $\Delta x, \Delta y, \Delta z$ values, but this won't be explored in this thesis [34]. The set of points where the magnetic field is defined is set up in a similar manner, with constant finite distance in the x, y, and z directions between points, however, the points of the magnetic field are displaced from the electric field points by a factor of $\frac{\Delta x}{2}$, $\frac{\Delta y}{2}$, $\frac{\Delta z}{2}$. Figure 2.1 shows a visualization of the distribution of points that comprise the problem space.

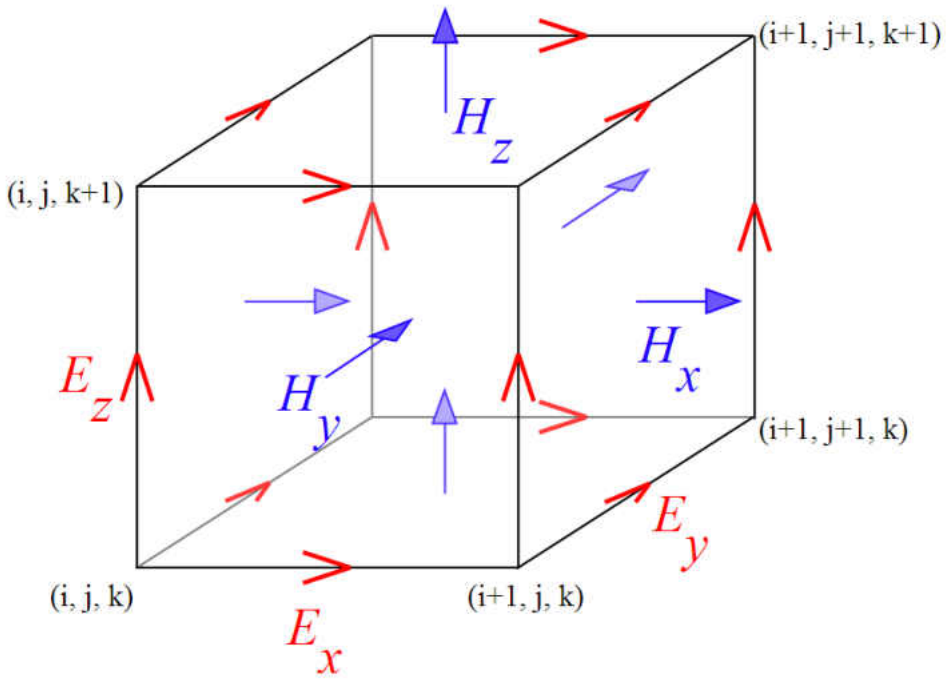


Figure 2.1: One cell of the Yee Grid used in FDTD

In order to apply Maxwell's equations to our discretized problem space, the curl and time derivative operators in equations (2.2), and (2.3) must be transformed from their continuous forms to their discretized forms. In order to do that, we harken back to the definition of the derivative given as follows:

$$\frac{\partial f}{\partial x}(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} \tag{2.8}$$

In the FDTD problem space the field components are always separated by a finite distance so equation (2.8) is altered such that the quantity Δx approaches, not 0, but

a constant. This leads to a straightforward discretization the $\frac{\partial}{\partial t}$ operator, and the curl operator, defined such that $\nabla \times \vec{F} = (\frac{\partial F_z}{\partial y} - \frac{\partial F_y}{\partial z})\hat{x} + (\frac{\partial F_x}{\partial z} - \frac{\partial F_z}{\partial x})\hat{y} + (\frac{\partial F_y}{\partial x} - \frac{\partial F_x}{\partial y})\hat{z}$ is discretized similarly.

Making these substitutions on the curl and time derivative operators in equation (2.2) gives us equation (2.9).

$$\begin{aligned}\frac{E_x^{n+1}(i,j,k) - E_x^n(i,j,k)}{\Delta t} &= \frac{1}{\epsilon(i,j,k)} \frac{H_z^{n+\frac{1}{2}}(i,j,k) - H_z^{n+\frac{1}{2}}(i,j-1,k)}{\Delta y} \\ &- \frac{1}{\epsilon(i,j,k)} \frac{H_y^{n+\frac{1}{2}}(i,j,k) - H_y^{n+\frac{1}{2}}(i,j,k-1)}{\Delta z} \\ &- \frac{\sigma(i,j,k)}{\epsilon(i,j,k)} \frac{E_x^{n+1}(i,j,k) + E_x^n(i,j,k)}{2} \\ \epsilon &= \epsilon_0 \epsilon_r\end{aligned} \tag{2.9}$$

Similar substitutions made on Faraday's Law give us the discretized version thereof. These discretized Maxwell's equations are manipulated further to give explicit updating equations of the electric and magnetic fields. An example of the explicit updating equation for the electric field is shown in equation (2.10):

$$\begin{aligned}E_x^{n+1}(i,j,k) = &\frac{2\epsilon(i,j,k) - \Delta t\sigma(i,j,k)}{2\epsilon(i,j,k) + \Delta t\sigma(i,j,k)} \times E_x^n(i,j,k) \\ &+ \frac{2\Delta t}{(2\epsilon(i,j,k) + \Delta t\sigma(i,j,k))\Delta y} \times (H_z^{n+\frac{1}{2}}(i,j,k) - H_z^{n+\frac{1}{2}}(i,j-1,k)) \\ &- \frac{2\Delta t}{(2\epsilon(i,j,k) + \Delta t\sigma(i,j,k))\Delta z} \times (H_y^{n+\frac{1}{2}}(i,j,k) - H_y^{n+\frac{1}{2}}(i,j,k-1))\end{aligned} \tag{2.10}$$

The simulation space of FDTD can be greatly reduced by assuming a two dimensional case. In a two dimensional FDTD, the fields are assumed to be constant in the z direction. Two dimensional FDTD simulations can be either Transverse Magnetic (TM) or Transverse Electric (TE). In the TM case, the electric field is assumed to be in the z direction with the H field being oriented in the y and x directions (in the plane of the 2D FDTD simulation). Our system uses a 2D TM FDTD as the forward solver.

In equations (2.9) and (2.10), $E_x^n(i,j,k)$ is the x component of the electric field at time step n (corresponding to time $n\Delta t$) at coordinates $i\Delta x, j\Delta y, k\Delta z$. $\epsilon(i,j,k)$ is the absolute permittivity at a point in space. Δt is the difference in time between time steps, which is a constant during the simulation. $\sigma(i,j,k)$ is the electric conductivity

at a point in the problem space, $H_z^{n+\frac{1}{2}}(i,j,k)$ is the z component of the magnetic field at a time step $n+\frac{1}{2}\Delta t$, defined at the coordinate indices (i,j,k). A similar equation exists for updating the magnetic fields of the problem space. Equation (2.11) shows the magnetic field updating equation for a non magnetic medium.

$$\begin{aligned} H_z^{n+\frac{1}{2}}(i,j,k) = H_z^{n-\frac{1}{2}}(i,j,k) \\ + \frac{2\Delta t}{(2\mu_0(i,j,k))\Delta y} \times (E_x^n(i,j,k) - E_x^n(i,j-1,k)) \\ - \frac{2\Delta t}{(2\mu_0(i,j,k))\Delta z} \times (E_y^n(i,j,k) - E_y^n(i,j,k-1)) \end{aligned} \tag{2.11}$$

Unwrapping the discretized Maxwell's equations into the form of equations (2.10) and (2.11) is important, as these equations are used to explicitly update the electric magnetic fields upon each time step of the simulation. Time stepping occurs until the simulation reaches its maximum number of time steps, specified at the start of the simulation. In the course of one time step, the electric field is updated first. When all electric field values are calculated (at $t = n\Delta t$), the magnetic field values are then updated (at $t = (n+\frac{1}{2})\Delta t$). When the simulation is completed, we now have knowledge of the electric and magnetic field values at all points of the problem space, at all time steps. This information can be processed further to give any electromagnetic information typically desired from simulations including scattered matrices, surface current density, impedance information, and as we use in our system, scattered-fields and Radar Cross Section (RCS) plots. Because output information is typically desired in the frequency domain, the Fourier transform is used to extract frequency domain information from the time domain data. The Fourier transform can be evaluated with either the Fast Fourier Transform (FFT), or the Discrete Fourier Transform (DFT). Our system uses the DFT, as its memory requirements are much lower than that of the FFT [35]. Whereas the FFT requires storage of previous samples during simulation, the DFT does not need to store data from previous samples. Instead, the DFT recursively calculates a running sum of complex numbers, calculation thereof requiring only the current complex sum and data from the current time step.

$\Delta x, \Delta y$, and Δz must be chosen in order to satisfy the Nyquist sampling theorem which states that the sampling rate must be twice that of the maximum frequency of a signal for accurate modelling. In practice, the Nyquist rate is exceeded and Δx etc. are chosen such that there are 20 FDTD cells per wavelength of the maximum frequency in the FDTD simulation. Δt is governed by the Courant stability criteria. The Courant stability criteria states that a wave should not travel more than one cell per time step. Mathematically this is represented by equation (2.12).

$$c\Delta t\sqrt{\frac{1}{(\Delta x)^2}+\frac{1}{(\Delta y)^2}+\frac{1}{(\Delta z)^2}}\leq 1 \tag{2.12}$$

2.1.1 Scattered Field Formulation of FDTD

In order to use a far-field source such as a uniform plane-wave as the excitation for an FDTD simulation, equation (2.10) must be modified to include this excitation signal. There are multiple methods of modelling plane-wave excitation in FDTD, notably the total-field/scattered-field formulation and the scattered-field formulation. In the total-field/scattered-field formulation, both the incident plane-wave and scattered-field are modelled within the problem space. In the scattered-field formulation, only the scattered-field is subject to time stepping and the incident-field is calculated analytically. The suitability of these methods defers based on application. Specifically, the pure scattered-field formulation is unsuitable for shielding problems where there exist regions of the problem space where the total-field is near zero [36]. In these regions the scattered-field and incident should be near equal in amplitude but with opposite signs. This unsuitability arises from subtraction error where the scattered and incident-fields may not necessarily cancel out due to numerical error. The fact that the incident-field is calculated analytically whereas the scattered-field is calculated during FDTD time stepping exacerbates this subtraction error given that only the scattered-field is subject to numerical error inherent to the FDTD simulation.

The pure scattered-field formulation, however, shines in its ease of implementation. The total-field/scattered-field formulation divides the problem space into regions which in which calculation of the updating equations differ in that there are two regions which model the total-field in one and the scattered-field in another. The scattered-field formulation is calculated much the same way as described previously in equation (2.10) etc. throughout the whole problem space with some minor alterations. This simplicity is especially important for implementation of FDTD on GPUs as code divergence (the inclusion of control statements such as if statements or case/switch statements) can slow program execution considerably. Given that in our problem (the modelling of human tissue in the microwave band) does not include material of particularly high conductivity, and thus, precludes the possibility of significant subtraction error, the pure scattered-field formulation is used in our system.

The scattered-field formulation as described in [37] relies on the linearity of Maxwell's equations. In the scattered-field formulation the total-field is split up into incident-field and scattered-field components. The incident-field is modelled in equation (2.13) and assuming to be propagating through free space. The total-field is described in equation

(2.14).

$$\nabla \times \vec{H}_{inc} = \epsilon_0 \frac{\partial \vec{E}_{inc}}{\partial t} \tag{2.13}$$

$$\nabla \times \vec{H}_{tot} = \epsilon \frac{\partial \vec{E}_{tot}}{\partial t} + \sigma \vec{E}_{tot} \tag{2.14}$$

Subtracting equation (2.13) from (2.14) gives, with some rearrangement, equation (2.15).

$$\epsilon \frac{\partial \vec{E}_{scat}}{\partial t} + \sigma \vec{E}_{scat} = \nabla \times \vec{H}_{scat} + (\epsilon_0 - \epsilon) \frac{\partial \vec{E}_{inc}}{\partial t} - \sigma \vec{E}_{inc} \tag{2.15}$$

Applying the discrete approximation of the derivative operator (2.8) and rearranging to create the updating equation (2.16) [35].

$$\begin{aligned} E^{n+1}_{x,scat}(i,j,k) = & \frac{2\epsilon(i,j,k) - \Delta t\sigma(i,j,k)}{2\epsilon(i,j,k) + \Delta t\sigma(i,j,k)} \times E^{n}_{x,scat}(i,j,k) \\ & + \frac{2\Delta t}{(2\epsilon(i,j,k) + \Delta t\sigma(i,j,k))\Delta y} \times (H_z^{n+\frac{1}{2}}(i,j,k) - H^{n+\frac{1}{2}}_{z,scat}(i,j-1,k)) \\ & - \frac{2\Delta t}{(2\epsilon(i,j,k) + \Delta t\sigma(i,j,k))\Delta z} \times (H_y^{n+\frac{1}{2}}(i,j,k) - H^{n+\frac{1}{2}}_{y,scat}(i,j,k-1)) \\ & + \frac{2(\epsilon_0 - \epsilon(i,j,k)) - \sigma(i,j,k)\Delta t}{2\epsilon(i,j,k) + \sigma(i,j,k)\Delta t} E^{n+1}_{inc,x} \\ & - \frac{2(\epsilon_0 - \epsilon(i,j,k)) + \sigma(i,j,k)\Delta t}{2\epsilon(i,j,k) + \sigma(i,j,k)\Delta t} E^{n}_{inc,x} \end{aligned} \tag{2.16}$$

Notice that equation (2.16) is nearly identical to (2.10), the standard updating equation, with the addition of the last two terms. This updating equation is applied everywhere in the problem space, a boon for GPU implementation.

The primary purpose of our forward solver is to generate scattered-field information for an object sampled at a high number of observation angles in the far-field of the target across multiple frequencies and resulting from multiple excitation angles. The definition

of RCS is shown in equation (2.17) where r is the distance from the antenna to the target, S_s is the scattered power density seen at a distance r away from the target, and S_i is the incident power density illuminating the target. Figure 2.2 shows a visualization of the RCS information generated by the forward solver for a single frequency and excitation angle. This RCS pattern would then be compared to the measured data using the least mean squared calculation shown in equation (2.1).

$$RCS(\theta) = 4\pi r^2 \frac{S_s}{S_i} \tag{2.17}$$

The RCS data is obtained via a near to far-field transformation [37]. The near-to-far field transformation used in FDTD relies on the surface equivalence theorem originally proposed by Sergei Schelkunoff which is a formal version of Huygen's principle [38]. The surface current equivalence theorem states that for a given source of radiation, one can enclose the radiator in an imaginary surface and calculate equivalent surface currents that would create identical fields outside that imaginary surface. Given those surface currents, the fields on the inside of the imaginary surface will be zero.

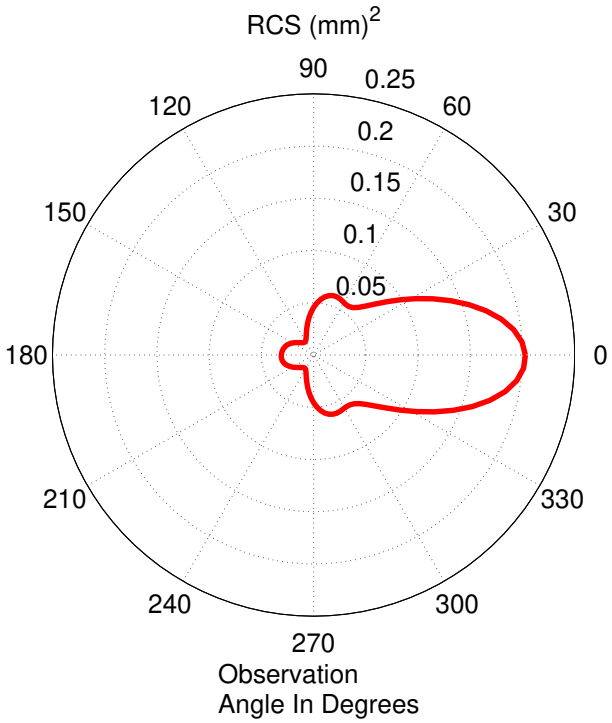


Figure 2.2: An example RCS pattern of a forward scattering dielectric cylinder generated by the FDTD forward solver

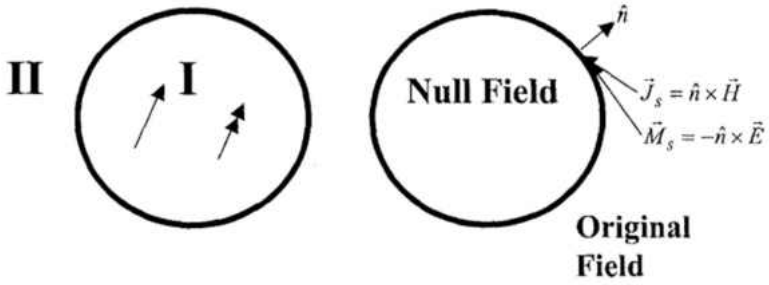


Figure 2.3: A visual demonstration of the field equivalence principle. When equivalent surface currents are placed on the surface of the imaginary boundary, there is a null field condition inside the boundary, but the fields are identical from the original problem outside the boundary [38].

As shown in Figure 2.3, the surface currents that create the equivalent outer fields are calculated by equations (2.22) and (2.23). During post processing these currents are found in the frequency domain via the DFT. Now equipped with the equivalent surface currents, it is possible to find the far-field RCS pattern for a given scatterer. The far-field pattern is obtained by calculating the magnetic and electric vector potentials $\vec{A}$ and $\vec{F}$, respectively. $\vec{A}$ and $\vec{F}$ are calculated by equations (2.18) and (2.19) via numerical integration. With knowledge of these vector potentials it is a simple matter to calculate the electric and magnetic fields in the far-field, as shown in equations (2.20) and (2.21).

$$\vec{A} = \frac{\mu_0 e^{-jkR}}{4\pi R} \int_S \vec{J} e^{-jkr' cos(\Psi)} dS \tag{2.18}$$

$$\vec{F} = \frac{\epsilon_0 e^{-jkR}}{4\pi R} \int_S \vec{M} e^{-jkr' cos(\Psi)} dS \tag{2.19}$$

$$\vec{H} = -j\omega[F + \frac{1}{k^2}\nabla(\nabla \cdot \vec{F})] + \frac{1}{\mu_0}\nabla \times \vec{A} \tag{2.20}$$

$$\vec{E} = -j\omega[A + \frac{1}{k^2}\nabla(\nabla \cdot \vec{A})] + \frac{1}{\epsilon_0}\nabla \times \vec{F} \tag{2.21}$$

$$\vec{J} = \hat{n} \times \vec{H} \tag{2.22}$$

$$\vec{M} = -\hat{n} \times \vec{E} \tag{2.23}$$

2.1.2 Frequency Dependant Finite Difference Time Domain ($(FD)^2$ TD)

Within the microwave band, biological tissue do not have constant permittivity and conductivity across all frequencies. Models such as the Cole-Cole model [39], the Lorentz model [40], and the Debye relaxation model [37] are used to describe the change in the complex permittivity of materials over frequencies. Water is highly dispersive over the band from 1 GHz to 10 GHz, the frequency band used in our system. Because biological tissue contains high amounts of water, they are also highly dispersive over the microwave band. Lazebnik et al. studied the dispersive characteristics of human tissue in [41]. Samples of breast tissue were divided into four groups, varying by adipose tissue content. The groups included tissues with 0-30 % adipose content, 31%-84% adipose tissue, 85%-100% adipose tissue and malignant tissue. The highest amount of dispersion was found to be in tumor tissue, where the dielectric constant dropped from 57 at 1 GHz down to less than 30 at 20 GHz.

In our system the single-pole Debye model is used to characterize the dispersive dielectric material located within breast tissue. The first order Debye relaxation model for dispersive dielectric materials is given in equation (2.24). In equation (2.24) $\hat{\epsilon}$ refers to permittivity values, ω is the angular frequency, ϵ_∞ is the value of relative permittivity of a medium as the frequency approaches infinity, $\Delta\epsilon$ is defined as $\epsilon_s - \epsilon_\infty$ where ϵ_s is the relative permittivity of the material at frequency zero, τ is the relaxation time of the molecules in the dispersive material, and j is the imaginary unit. This equation is in the frequency domain, yet for implementation in FDTD we require all calculations to be performed in the time domain.

$$\hat{\epsilon}(\omega) = \epsilon_\infty + \frac{\Delta\epsilon}{1 + j\omega\tau} \tag{2.24}$$

The inverse Fourier transform allows implementation of equation (2.24) in FDTD. A method of incorporating the Debye model into FDTD using the auxiliary differential equation method is presented in [42]. A direct substitution of equation (2.24) into Ampere's equation (2.2) followed by an inverse Fourier transform gives Ampere's law for a Debye dispersive media shown in equation (2.25).

$$\nabla \times H(t) = \epsilon_0\epsilon_\infty \frac{\partial}{\partial t}E(t) + \sigma E(t) + \epsilon_0 \mathscr{F}^{-1}\{j\omega \frac{\Delta\epsilon}{1 + j\omega\tau}E(\omega)\} \tag{2.25}$$

We define $\epsilon_0 \mathscr{F}^{-1}\{j\omega \frac{\Delta\epsilon}{1+j\omega\tau}\vec{E}(\omega)\}$ as $\vec{J_p(t)}$ which is called the polarizing current. Taking the inverse Fourier transform, we arrive at equation (2.26), which is known as the

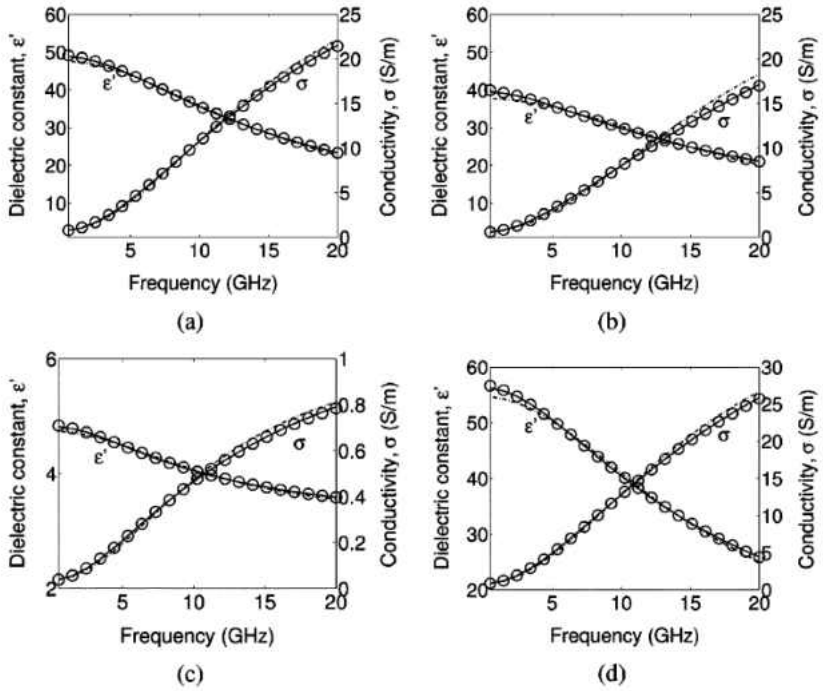


Figure 2.4: A comparison between the permittivities vs. frequency charts of four types of tissue studied by Lazebnik et al. (a) corresponds to 0-30% adipose tissue, (b) corresponds to 31-84 % adipose tissue, (c) corresponds to 85-100% adipose tissue, and (d) corresponds to malignant tissue [41]

.

auxiliary differential equation. $\vec{J_p}$ is defined at every cell of the FDTD problem space, and so adds a small additional memory requirement to the FDTD algorithm.

$$\frac{\partial}{\partial t}\vec{J_p} = \epsilon_0 \Delta\epsilon \frac{\partial}{\partial t}\vec{E} - \frac{1}{\tau}\vec{J_p} \tag{2.26}$$

Discretization of equation (2.26) gives us the updating equation for $\vec{J_p}$ to be performed at every time step. Due to the inverse Fourier transform, there exists a convolution operator involved in the calculation of $\vec{J_p}$. Because straightforward convolutions are unsuited for computational methods due to their large memory requirements, $\vec{J_p}$ is recursively calculated using equation (2.27).

$$J_p^{n+1}(i,j,k) = \frac{\beta_p}{\Delta t}(E^{n+1}(i,j,k) - E^n(i,j,k) + k_p J_p^n(i,j,k) \tag{2.27}$$

$$k = \frac{1 - \frac{\Delta t}{2\tau}}{1 + \frac{\Delta t}{2\tau}} \tag{2.28}$$

$$\beta_p = \frac{\epsilon_0 \Delta\epsilon \frac{\Delta t}{\tau}}{1 + \frac{\Delta t}{2\tau}} \tag{2.29}$$

While the previous analysis is sufficient for implementation of a standard FDTD with near-field excitation, we require the Debye model to be used in conjunction with the scattered-field formulation. An analysis of dispersive materials for use in conjunction with the pure scattered-field formulation is given in [43]. Therein, the updating equations for Lorentz media and Drude media are derived, though the updating equations for Debye media are omitted. Using similar analysis, we arrived at the explicit updating equation for Debye dispersive media for the pure scattered-field FDTD formulation shown in equation (2.30).

$$\begin{aligned} E_{z,scat}^{n+1}(i,j,k) = & \frac{2\epsilon_\infty\epsilon_0 - \sigma\Delta t + \beta_p}{2\epsilon_\infty\epsilon_0 + \sigma\Delta t + \beta_p} E_{z,scat}^n(i,j,k) \\ & + \frac{2\Delta t}{2\epsilon_\infty\epsilon_0 + \sigma\Delta t + \beta_p} \nabla \times H \\ & + \frac{-2(\epsilon_\infty - \epsilon_0) - \sigma\Delta t - \beta_p}{2\epsilon_\infty\epsilon_0 + \sigma\Delta t + \beta_p} E_{inc,z}^{n+1}(i,j,k) \\ & + \frac{2(\epsilon_\infty - \epsilon_0) - \sigma\Delta t + \beta_p}{2\epsilon_\infty\epsilon_0 + \sigma\Delta t + \beta_p} E_{inc,z}^n(i,j,k) \end{aligned} \tag{2.30}$$

where (2.31)

$$\begin{aligned} \nabla \times H = & \frac{1}{\Delta y}(H_x^{n+\frac{1}{2}}(i,j,k) - H_x^{n+\frac{1}{2}}(i,j,k-1)) \\ & - \frac{1}{\Delta x}(H_y^{n+\frac{1}{2}}(i,j,k) - H_y^{n+\frac{1}{2}}(i-1,j,k)) \end{aligned} \tag{2.32}$$

In the explicit updating equation (2.30), β_p is calculated as per equation (2.29). $\vec{J_p}$ is also calculated iteratively during the simulation according to equation (2.27)

2.2 GPU Parallelization of FDTD Forward Solver

As is often the case with numerical simulations of physical phenomena, the FDTD method suffers from incredibly long run times. A single FDTD simulation with a problem space of 600 × 600 cells that runs for 1000 time steps requires a minimum of 360 *million* floating point operations, in addition to an even greater amount of memory operations required. Coupled with this, MWT systems that make use of global stochastic optimization methods without approximations require upwards of 10,000 candidate solutions to

be evaluated by the forward solver [44]. Using only standard computing hardware, such as one would find on a single high end desktop computer, this exorbitant computational load alone would render MWT methods such as this infeasible. It is well known that we cannot expect significant computing power increases from CPUs, even with Moore's law. The obvious solution to this is, rather than relying on a single processor, to use many processors working in parallel. This is known as parallelization.

Graphics processing units, which were originally developed for the fast rendering of graphics as their name would imply, are built based on the principle of parallel processing. The reason for this is that graphics rendering techniques are extremely parallelelizable algorithms. They are parallelizable due to the fact that, although they require a large amount of computation, many parts of the algorithm can be completed without relying on results from other steps of the algorithm. Due to this high degree of specialization, GPUs are suitable only for problems that are very amenable to parallelization. Fortunately, the FDTD method is one such problem. Indeed, the FDTD method has been described as "embarrassingly parallel" in [19] in that during parallel computation a single processor needs only limited information from other processors. This is important, given the fact that a high number of memory accesses during a parallel process can slow down the program significantly.

In order to implement the FDTD forward-solver on GPUs, NVidia's CUDA was used. CUDA was chosen due to its ease of programming. Before the advent of CUDA and modern general purpose GPUs, GPU computing was tremendously difficult to implement, as early GPUs were only capable of specific graphics-based tasks. With CUDA, high level languages such as C, C++, or Fortran are capable of sending instructions to GPUs. This is in contrast to using OpenGL, which is specialized for graphics applications, or assembly languages. Our system in particular uses CUDA in conjunction with C++.

GPU computation using CUDA uses a GPU computing paradigm wherein a function launches a kernel on the GPU. The kernel is made up of a grid containing all processes to be run in parallel embedded in which are multiple blocks. These blocks correspond to separate streaming multiprocessors, each of which is physically isolated from other streaming multiprocessors on the GPU card. Each of these blocks can support up to 512 threads. Each thread is a set of instructions to be executed in serial. All threads within the kernel at large can be considered, from a programming perspective, to run in parallel although in reality only a select number of threads are run concurrently at a time. Figure 2.5 shows this arrangement. This computation hierarchy is not merely an arbitrary grouping, but rather determines what types of memory a process can use. In CUDA there exist multiple types of memory, namely shared memory, global memory, texture memory, register memory, and constant memory. Each of these types of memory are suited for different applications and have unique scopes. Register memory is the fastest type of memory but has very limited storage and, more importantly, is only accessible by a single thread. Global memory is accessible by all threads in the

kernel, even threads in different blocks, but accessing global memory is relatively slow. Shared memory is much faster than global memory, but is only accessible by threads within a single block. Texture memory is a type of constant memory that can store vast arrays of constant float values and is optimized for accesses between nearby threads. Finally, constant memory stores constant values and is optimized for values that need to be accessed by a large amount of threads concurrently. Given these considerations it is important to tailor one's algorithm to the kernel paradigm in order to fully utilize the high computational power of GPUs.

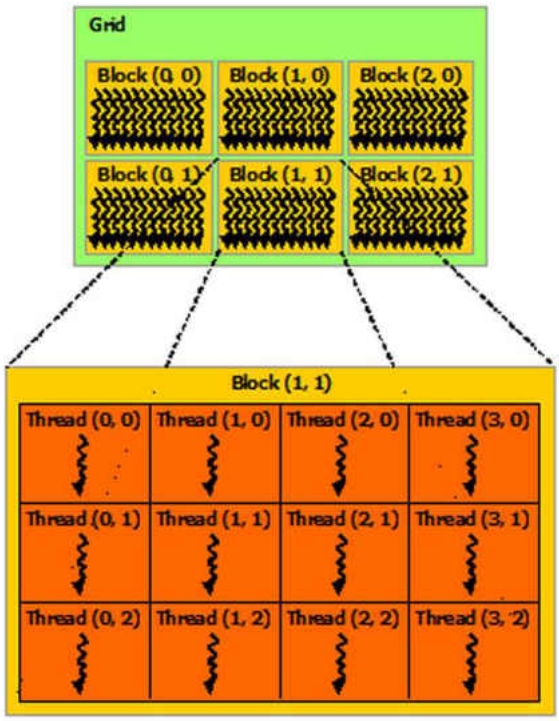


Figure 2.5: Visualization of the parallelization paradigm used in CUDA [18].

The FDTD forward solver used in our system is parallelized by assigning every Yee Cell (shown in Figure 2.1) of the problem space to a single thread on the GPU. Each of these threads is responsible for updating the electric and magnetic fields at one index (x, y) when called upon by the forward solver. The updating of the magnetic and electric fields are separated into two kernels, with one kernel responsible for updating the electric field and one kernel responsible for the magnetic field. During one time step the electric field updating kernel is called and then the magnetic field kernel is called. Other functions are also used that are involved in processing the time domain fields into the RCS data to be passed back to the inverse solver. The separation of the electric and magnetic fields into different kernels is used to ensure that the all electric field values are updated before proceeding to update the magnetic field in order to follow the leapfrogging scheme of

FDTD. Currently there is no way to synchronize threads across different blocks using commands within a kernel, but there exists an implicit synchronization between calls such as copying memory to, or from, the GPU and consecutive kernel calls. Figure 2.6 illustrates the parallelization scheme used in assigning the cells of the problem space to the GPU. In addition to updating the electric and magnetic field values, GPU kernels are also used to initialize constant values and perform post processing, specifically the near to far-field transformation, on the fly, yielding some performance increase.

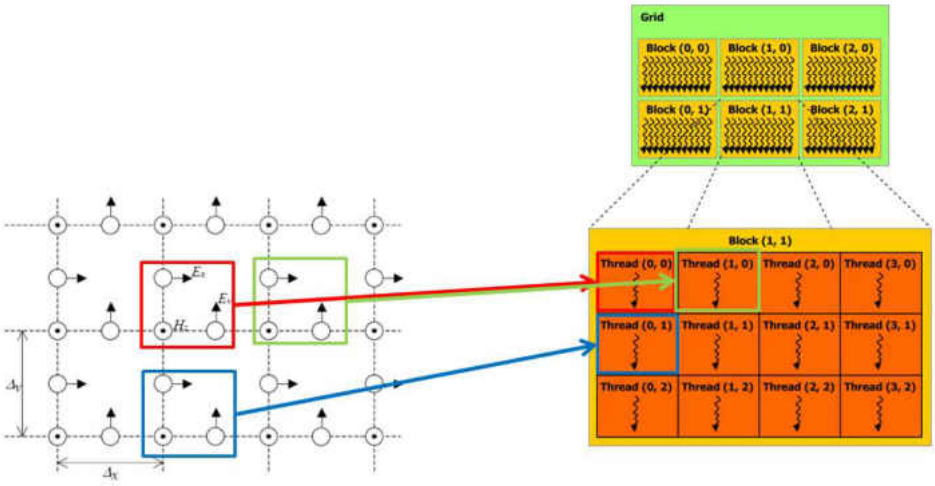


Figure 2.6: The parallelization scheme used to implement the FDTD program on GPU.

2.2.1 FDTD GPU Acceleration Results

A successful forward solver for the purpose of MWT must be both fast and accurate. It is important that GPU implementation of the FDTD algorithm does not introduce any new error when switching from a CPU calculated FDTD to a GPU accelerated FDTD simulation. To test if any error was introduced by GPU implementation, a CPU based FDTD program was developed that was mostly identical to the GPU program. A single point of the problem space was probed at every time step in both simulations and were compared. It was found that both the CPU and GPU programs produced identical field values at all time steps during simulation. While this does not necessarily guarantee that there was no error at any points in the problem space, it does show that there is no significant error introduced solely by GPU implementation, which could arise from non-standard methods of handling floating point numbers.

The absolute accuracy of the system was also assessed. The absolute accuracy of the algorithm is determined by how closely the output data matches what would happen in physical reality. In order to test this, RCS values generated by our GPU accelerated FDTD program were compared with RCS patterns generated by the commercial simula-

tion software FEKO^{TM}[45]. A dielectric cylinder with a relative permittivity of 10 and a radius of 3.15 cm was used as a test case due to its ease of construction in both FEKO and the FDTD forward solver. The cylinder was modelled as being infinitely long in the z direction, to match the fact that our forward solver is a two dimensional FDTD simulation. A TM plane-wave (with the electric field in the z direction) was modelled as the excitation with a frequency of 5 GHz. RCS values were recorded on the azimuthal plane from FEKO and our forward solver and compared by finding the MSE between the RCS at all observation points. The results of this comparison are shown in Figure 2.7. One can see that there is a minute amount of error between the two methods. This error may be attributed to numerical error, given that our forward solver uses only single precision floating point numbers due to the limitations of the GPUs. The least mean squared error was calculated in a similar manner as equation (2.1), except only summed over the number of observation points. The MSE between the two methods was found to be 6.2610×10^{-4}. This error would seem acceptable, as any error introduced by our FDTD forward solver would be vastly overshadowed by experimental error introduced during the measurement stage of MWT in any real clinical deployment of our system.

The most important criteria for our forward solver, and indeed the onus for the development thereof, is the speed at which it can take a candidate image and return its fitness value. A test case of finding the RCS pattern of a dielectric cylinder was used. The problem space contained 1000×1000 cells and marched through 2000 time steps. A serial CPU-computed FDTD algorithm that was otherwise identical to the GPU accelerated FDTD simulation was developed in order to compare the run times of the two programs. The FDTD code was run on an NVidia^{TM}c1060 Tesla GPU and compiled with CUDA 3.0, whereas the CPU program was tested on a 64 bit, 4 core Opteron CPU with 8 cores total equipped with 16 GB of RAM. It was found that the CPU program finished computation in 400 seconds, whereas the GPU accelerated FDTD forward solver completed in a mere 4 seconds, yielding a *100 fold* speed increase. We also compared our algorithm to other parallelization attempts, specifically parallelization using MPI. Guiffaut and Mahdjoubi presented an MPI parallelized FDTD program in [46]. In this paper, an FDTD simulation that used a volume of $150 \times 150 \times 50$ cells with 100 iterations was tested on a computer cluster using 1, 4, 8, and 16 processors. It was found that using 16 processors decreased the run time of the FDTD simulation from 248 seconds to 23 seconds, which is around a 10:1 speed increase. Although the computer hardware used in [46] is quite dated by today's standards, we emphasize the ratios of run times for the serial versus the parallelized simulations in both the MPI parallelized FDTD simulation and the GPU accelerated FDTD simulation.

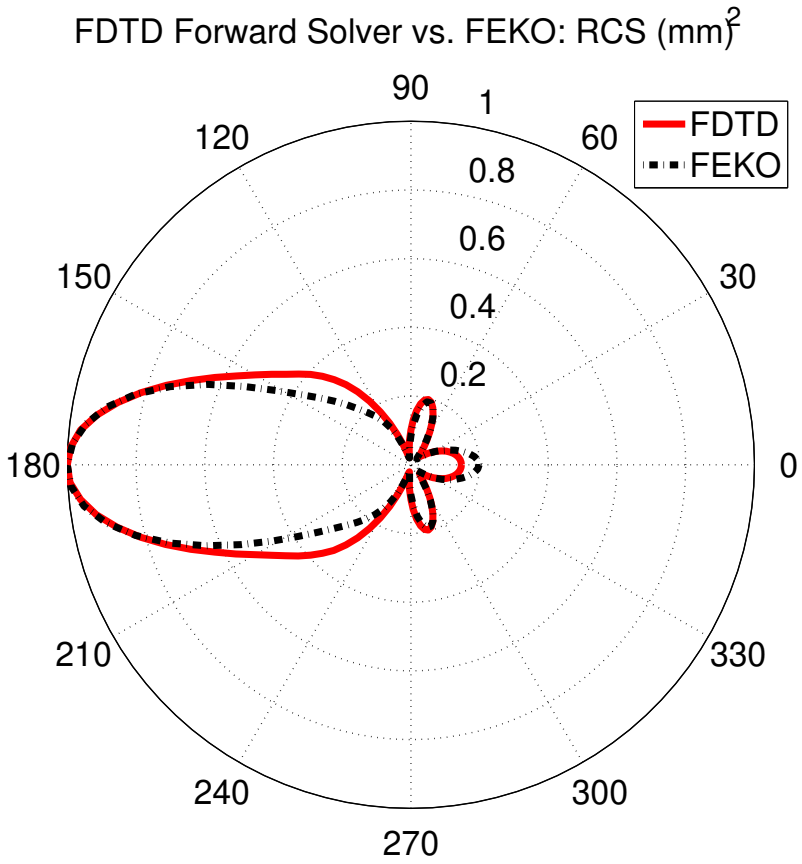


Figure 2.7: Comparison of RCS pattern given from our FDTD simulation and commercial software FEKO^{TM}.

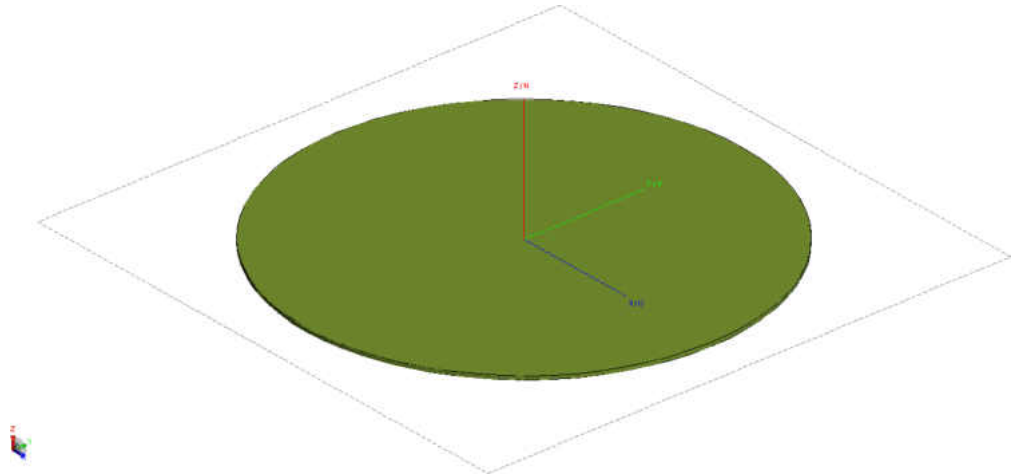


Figure 2.8: Model used to compare the performance of FEKO^{TM}to our forward solver.

3

Microwave Tomography Inverse Solver

The inverse solver of a microwave tomography system extrapolates the arrangement of scattering sources with the scattered field assumed to be known. This problem is under-constrained, and ill-posed. A problem is considered to be well-posed if it satisfies the three following conditions [47]:

1. **The Solution Exists**: For a given input, a solution exists.
2. **The Solution Is Unique**: For a given input, there exists a total of one solution.
3. **The Problem Is Stable**: Small perturbations of the input value create boundedly small variations in the solution.

With regards to the existence condition, in microwave tomography this would indicate that for every set of measurement data, there would exist a map of permittivity such that the MSE as calculated in equation (2.1) between the measured data and the simulated data would be zero. Of course, we can never expect that the error would actually reach zero, due to the finite precision used by computers during calculation, and due to the fact that the forward solver cannot model reality with 100% accuracy. To get around this we call the problem solved when we find the permittivity map that produces the minimum amount of error, rather than when we have found the permittivity map that produces zero error.

In theory Maxwell's equations always have a unique solution [48]. Usually when discussing this uniqueness the forward problem is considered, where the fields are calculated from a set of known boundary conditions and source terms. Based on the uniqueness theorem the inverse problem, in theory, has a unique solution. The uniqueness of the forward problem implies a bijection from the set of all boundary conditions and source terms to the set of all possible field configurations which of course implies a bijection of the inverse problem. In practice, however, we do not have total knowledge of the electromagnetic field at all points in space, especially given that there would be no way

to find the field inside of the imaging target. In reality, we have a finite amount of observation points corresponding to the amount of antennas located around the imaging target. Thus, we cannot guarantee that for a given set of measurement data there is a unique permittivity map. In fact, very different distributions of complex permittivity can create RCS patterns that are very close. This is illustrated in Figures 3.1 and 3.2. In those images, areas coloured in red are regions of high permittivity whereas blue regions are regions of low permittivity.

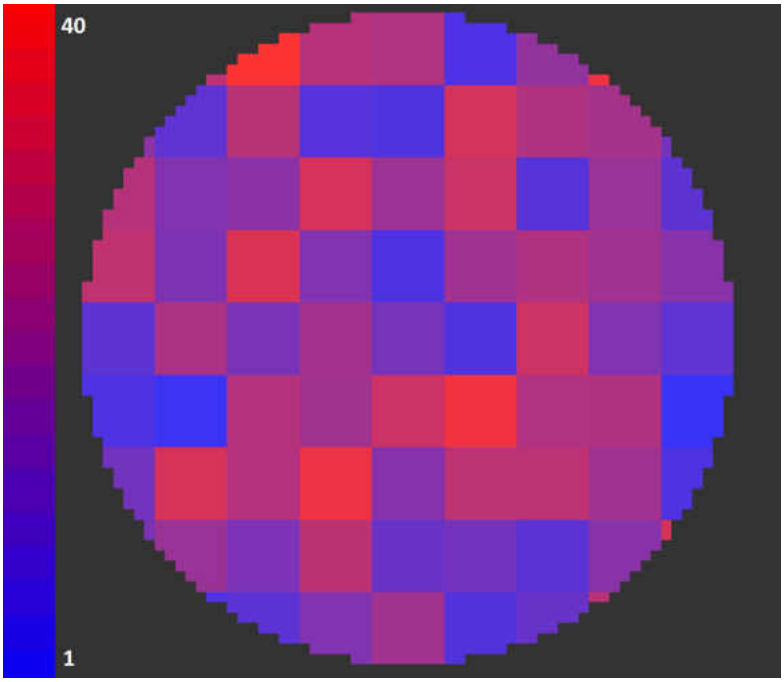


Figure 3.1: A permittivity map that produces an MSE of less than 0.25 from RCS data created by the source image shown in Figure 3.2

The MWT inverse problem may also be unstable. For a small change in the permittivity map of the target, there might be an arbitrarily large change in the scattered-field picked up by the receivers. The stability of the MWT inverse problem using a Newton-Kantorovitch reconstruction method was examined in [49]. In addition to this, the presence of modelling noise and experimental noise can cause instability in the inverse problem. Reconstruction is also highly affected by the arrangement of observation points around the target, in addition to the frequency band used in measurements.

The majority of inverse solvers use iterative deterministic optimization methods to reconstruct images. Such optimization methods include the Newton method [22], contrast source inversion and multiplicative regularized contrast source inversion [50], conjugate gradient method [51], and many more. These optimization methods rely on the gradient, Jacobian, or other derivative values of the inverse problem. In order to deal with the ill-posedness and non-linearity of the MWT inverse problem, inverse solver typically

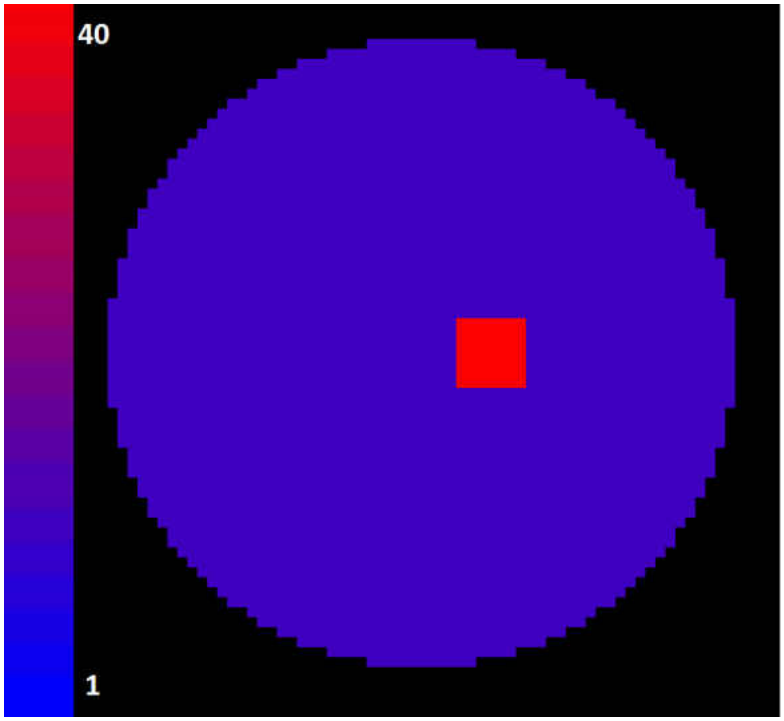


Figure 3.2: Permittivity map used to create the measurement data in simulation to reconstruct Figure 3.1

make use of heavy regularization. This regularization causes distortions to the output image, usually manifesting in smoothing of the permittivity profile. Stochastic optimization methods have also been heavily researched for reconstruction, however these often involve the same regularization assumptions [52]. For the purposes of breast cancer detection, such regularization may hinder the detection of small tumours, as these tumours would be smoothed out during reconstruction. These regularizations are necessary for deterministic optimization methods to prevent convergence upon local minima of the solution space (The solution space being the mapping of the points in Cartesian space of the inputs to the forward solver to their fitness values). Stochastic, a.k.a global, optimization methods do not require such rigorous regularization as they have the ability to escape local minima through randomization or other non-deterministic processes.

Explored in this thesis are the use of Particle Swarm Optimization (PSO), and Differential Evolution (DE). These optimization methods are population based evolutionary algorithms that are designed to operate on non-convex, non-discrete domains. Both DE and PSO perform very well across a wide variety of benchmarks, usually outperforming other population based evolutionary algorithms such as GA in convergence time and convergence to the absolute extrema of a function [53]. PSO has seen much use in the field of microwave imaging. In [54] PSO that has been modified to include hybrid boundary conditions has been used to reconstruct a dielectric target. PSO has been used in

[55] to investigate weakly scattering dielectric objects with complex permittivity profiles. DE has been used in [56] to electromagnetically investigate cylindrical conductive targets.

The populations in both DE and PSO are collections of "individuals" where each individual represents a candidate map of permittivity for the imaging target. The individuals are represented by an array of floating point numbers which dictate the parameters $\Delta\epsilon$, ϵ_∞, and σ as described in equation (2.30) of one region of the image. The shape of the imaging target is assumed to be known, and in simulation is assumed to be circular with a known radius. This radius is able to be changed to accommodate objects of different sizes. The target is split into $n \times n$ regions of constant permittivity. These regions are known as optimization patches as the inverse solver alters the material properties of these patches during optimization. Figure 3.3 shows the layout of these patches with $n = 9$ with a radius of 0.0315 m. Because we know the shape of the imaging target *a priori*, the patches are truncated on the surface of the imaging target by the forward solver. An example of the patch numbering is shown in Figure 3.3. In our encoding scheme there is a total of $n \times n \times 3$ parameters to optimize on. Because the number of optimization parameters increases with the square of the number of patches, convergence time of the inverse solver is highly dependant on the resolution of the image. Equation (3.1) shows how each individual candidate solution is encoded.

$$Individual = \{\epsilon_{\infty,1,1}, \epsilon_{\infty,1,2}, ...\epsilon_{\infty,n,n}, \Delta\epsilon_{1,1}..., \Delta\epsilon_{n,n}, \sigma_{1,1}..., \sigma_{n,n}\} \tag{3.1}$$

For any stochastic optimization method, there is a balance between exploration and exploitation. Exploration is the tendency of an algorithm to more fully explore the optimization space which allows the optimization to escape local minima of the solution space. Exploitation is the ability of the optimization algorithm to converge to a value quickly. Too much exploration will cause an optimization method to never converge, whereas too much exploitation will cause the optimizer to converge to local minima with no chance of escape. Due to the ill-posedness of the unregularized MWT inverse problem, it is important to carefully choose the appropriate input parameters to the optimization methods so that they will be able to explore the problem space enough to not converge to false solutions such as the one shown in Figure 3.1.

3.1 Inverse Solver Implementation

3.1.1 Toolkit for Asynchronous Optimization

Our system uses the Toolkit for Asynchronous Optimization (TAO) as its inverse solver. TAO is a publicly available open-source project which can be found on the code repository Github [57]. TAO was developed to be used as a generic optimization package, suitable for systems that depend on either deterministic optimization methods or global

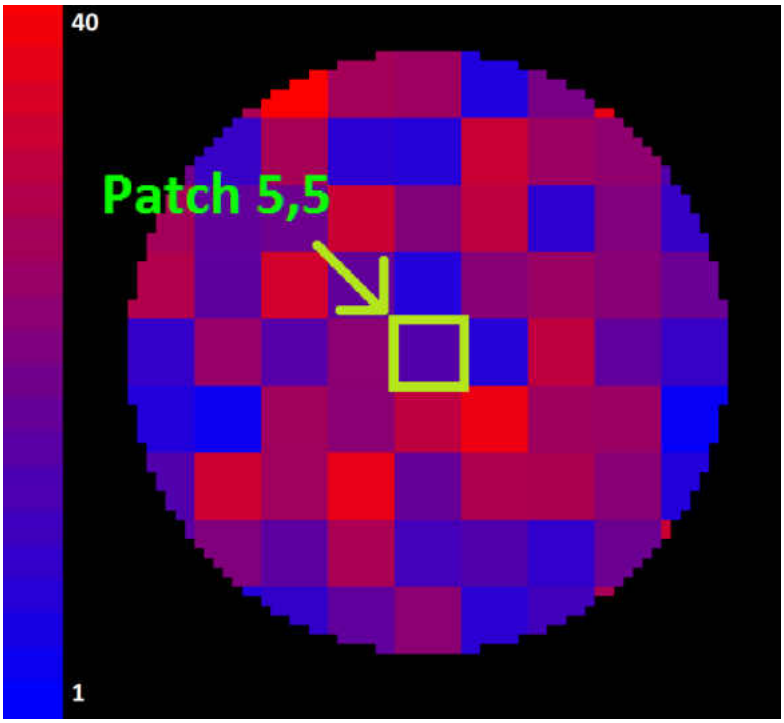


Figure 3.3: Image of an individual solution with 9×9 optimization patches.

optimization algorithms. As opposed to other available global optimization toolkits, TAO is especially suitable for large scale parallelization, whereas it is interoperable with both MPI and CUDA. Currently implemented optimization algorithms include GA, DE, PSO, an asynchronous Newton method, gradient descent, conjugate gradient, and a synchronous Newton method. The DE optimization method can be modified at the user side, namely the method of parent selection, and recombination selection scheme. TAO is implemented in C++ in order to improve performance. Scalability assessments have been performed at the University of North Dakota where it was found that increasing the amount of worker nodes improved performance nearly ideally. Fitness evaluation time during a cancer clustering simulation was found to be 99.99 %, indicating infinitesimal idle times for the worker nodes [58]. Figure 3.4 illustrates the scalability of the TAO framework. Our system uses the PSO and DE optimization methods.

3.1.2 Computational Hardware

The University of North Dakota is equipped with multiple high performance computing clusters. Our system runs on the Shale Linux HPC cluster, given that it is equipped with CUDA compatible GPUs. The Shale cluster includes 4 nodes, each equipped with two NVidia c1060 Tesla GPU cards. During optimization all eight GPUs were used for fitness evaluation. Analysis of one optimization session showed that the master process (the process that affected the optimization algorithm) was idle 99.99% of the time.

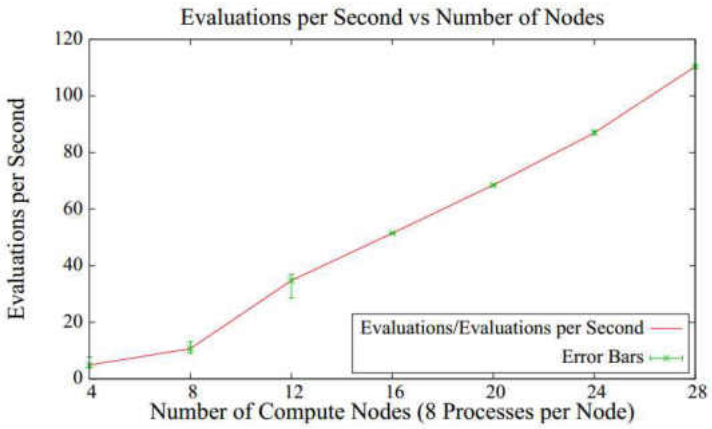


Figure 3.4: A plot of fitness evaluations per second vs. number of computing nodes. The linear behaviour indicates minimal idle time on the worker nodes and high scalability [58].

Given this, it seems likely that a greater number of computing nodes would increase the performance of our system greatly.

3.2 Particle Swarm Optimization

PSO is a stochastic optimization method initially introduced by Kennedy and Eberhart [59]. PSO was originally developed to model the swarming behaviour of animals such as flocks of birds, insects, or fish. In PSO, individuals are considered to be located at points in optimization space. In our system, the particle's would be located within an $n \times n \times 3$ hyperspace. Each particle, in addition to its position vector, has an associated velocity vector. At every iteration of the optimization method, the particles move along their velocity vectors and their velocities are updated so that the particle's move towards both the point associated with greatest fitness that that particle has found and the point associated with the greatest fitness that the swarm as a whole has found. The former is called the "local best" position whereas the latter is called the "global best" point. Figure 3.5 illustrates the movement of a particle through optimization space during updating. Whereas other population based evolutionary algorithms use relatively complex algorithms to update the individuals at each iteration, PSO uses only a simple velocity updating equation (3.2) [60]. Once the velocity is calculated, the position is updated according to equation (3.3) [60].

$$v_i(n+1) = \omega \times v_i(n) + c_1 \times rand() \times (l_i - x_i(t)) + c_2 \times rand() \times (g - x_i(n)) \quad (3.2)$$

$$x_i(n+1) = x_i(n) + v_i(n+1) \quad (3.3)$$

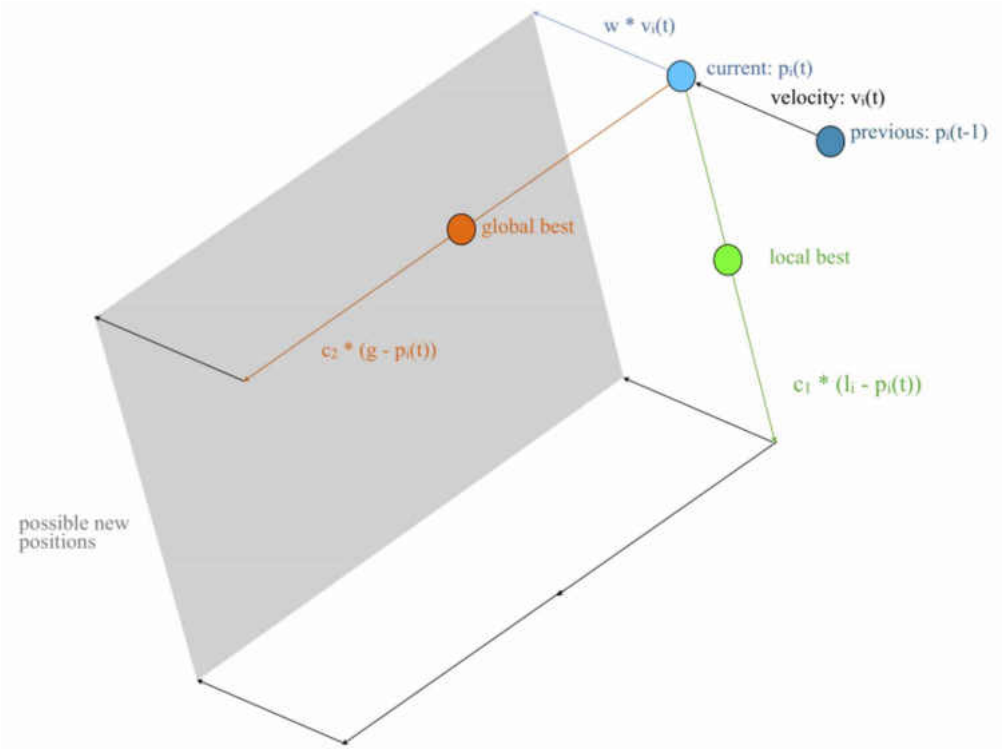


Figure 3.5: Visualization of the velocity and position updating scheme used in PSO [60].

In equations (3.2) and (3.3), $v_i(n+1)$ is the velocity of the i^{th} particle at iteration $n+1$. c_1 and c_2 are the local best weights and global best weights, respectively, as shown in Figure 3.5. $rand()$ is a random number between 0 and 1. l_i is the point with greatest fitness that the i^{th} particle has found, and g is the point of greatest fitness that the whole swarm has found. $x_i(n)$ is the current position of particle i. ω is the inertia, a term which was introduced later by Shi and Eberhart in order to further modulate the exploratory and exploitive behaviour of PSO [61]. For a given PSO session, the user must specify the local best weight c_1, the global best weight c_2, the inertia ω, and the population size. Increasing the local best weight, increasing the inertia, and increasing the population size can be thought of as increasing the exploratory behaviour of the PSO, while increasing the global best weight is associated with increasing the exploitative behaviour of the PSO.

3.2.1 Asynchronous Particle Swarm Optimization

In a parallelized PSO, multiple individuals of a population are evaluated at the same time by a selection of worker nodes. The master node controls the updating of the pop-

ulation and implements one's choice of inverse solver. In the standard PSO, particles are all updated in discrete iterations. For a parallelized PSO algorithm, this synchronism could be problematic, especially in heterogeneous computing environments (where some workstations are markedly faster than other worker nodes). During synchronous updating, all fitness values of the population must be available before the population can be updated and new work orders sent out. Should some worker nodes finish early, they would have to wait until slower worker nodes are finished with calculation.

In the asynchronous PSO worker nodes are not assigned to any particular particle, but rather are given evaluation assignments as they complete calculation. The master node keeps track of the positions and associated best fitness values of the population and sends candidate solutions to the worker nodes in a round robin method such that all particles are eventually updated [60]. In the case that the number of worker nodes is less than that of the number of individuals in the population, the asynchronous PSO performs identically to the standard PSO, although it is much faster. When the number of worker nodes increases beyond the number of individuals of the population, the PSO takes a more exploratory nature. Rather than only calculating a single future position for a given particle, the excess processors calculate additional future positions for a single particle. These positions would not be the same due to the use of randomization in the velocity updating equation (3.2). Future positions that are found to increase the global best fitness or the local best fitness are kept.

3.3 Differential Evolution

DE was developed to be used for optimization on continuous domains by Storn and Price [62]. During a DE optimization, a population is generated randomly throughout the search space and their fitness values are calculated. Individuals are then combined with other individuals according to a parent selection scheme and recombined according to the recombination scheme selected. For a given DE optimization, one can select the parent selection scheme, recombination scheme, number of pairs, parent scaling factor, differential scaling factor, crossover rate, and population size. During optimization, individuals can only increase in fitness. If, during updating, an individuals new position is found to have a lower fitness than it currently has, that position is discarded.

In DE, a parent is selected from the population. The difference between pairs of random individuals is calculated and used to update the position of the individual. The fitness value of this new position is calculated by the forward solver, and if it is greater than the old position, it is used to update that individual. Parent selection schemes include best, random, current-to-best, and current-to-random. Recombination schemes include binomial, exponential, and differential. The number of pairs can be any integer from 1 to the maximum amount of pairs possible in the population, although is usually kept close to 1. In the "best" parent selection scheme, the individual with the best fitness is chosen to be the parent for all other individuals. In the "random" parent

selection scheme, a random individual is chosen to be the primary parent. Equation (3.4) shows the updating algorithm for an individual with random or best parent selection and binomial or exponential recombination selection. Equation (3.5) is the updating equation for a differential optimization using random parent selection with differential recombination. In equation (3.4), $x(l+1)_j$ is the position x of the next iteration $l+1$ along the j^{th} optimization parameter. y is the chosen parent individual which could be either the best individual of the whole population, or a randomly chosen individual. ϕ is the parent scaling factor, which is specified at the start of the simulation. p is the number of pairs which is specified by the user. The $r(l)$ terms represent two randomly chosen individuals of the population which are distinct from each other. $r(0,1)$ is a random number between 0 and 1 and δ is the crossover rate, which is the probability that a given parameter will be updated and is set at the beginning of the optimization. In binomial recombination one or more optimization parameters (in our case an example of this would be the conductivity of one patch) and updates them. In exponential recombination, one parameters is selected at random, and all parameters following that one are updated. There is also a differential recombination scheme wherein two random individuals are chosen and the position of the worse individual is subtracted from the position of the better individual of the pair. Figure 3.6 shows how individuals move through space according to the placement of other individuals of the population.

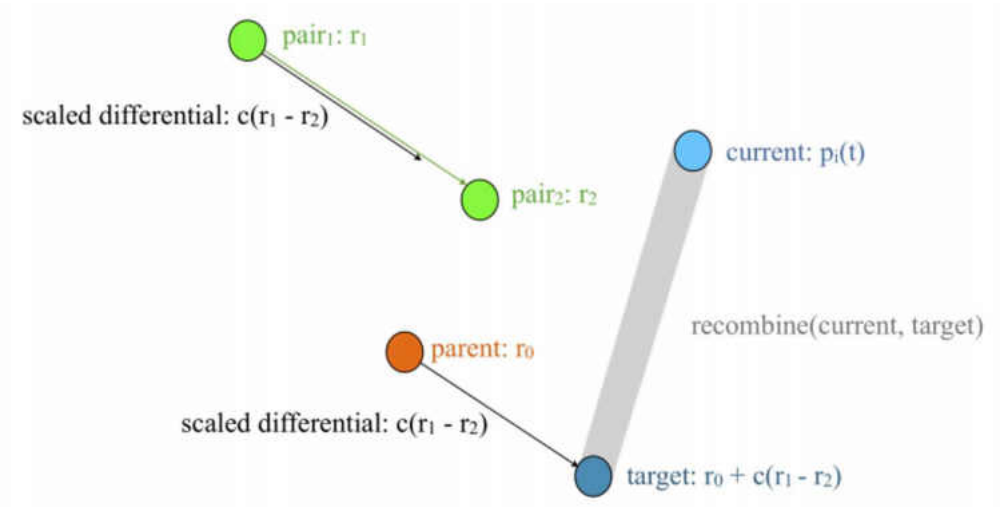


Figure 3.6: Visualization of the position updating scheme used in DE [60].

$$x(1+l)_j = \begin{cases} y(l)_j + \phi\Sigma_{k=1}^{p}[r(l)_j^{1k} - r(l)_j^{2k}] \text{ if r(0,1)} < \delta \\ x(l) \text{ otherwise} \end{cases} \tag{3.4}$$

$$x(l+1)_j = r(l)_j^1 + \phi\Sigma_{k=1}^{p}[r(l)_j^{1k} - r(l)_j^{2k}] \text{where} f(r(l)^{1k}) < f(r(l)^{2k}) \tag{3.5}$$

Using "best" parent selection makes the optimization more exploitative, whereas using random parent selection makes the algorithm more exploratory. Increasing the number of pairs increases the exploratory behaviour of the optimization. It was found that using random parent selection with two pairs and differential recombination performed best across the average of difficult problems which were multi-modal and non separable [60].

For the remainder of this thesis we will use the notation
DE/parent_selection/number_of_pairs/recombination_selection.
Thus, a DE optimization that uses best parent selection, two pairs, with exponential recombination would be named DE/best/2/exp.

3.3.1 Asynchronous Differential Evolution

In standard DE implementations, individuals are updated in an iterative fashion, where the algorithm waits for all individuals to be updated before proceeding to update other individuals. During asynchronous optimization, evaluations of individuals may come back at different times depending on the heterogeneity of the computing environment. The asynchronous DE eschews strict iterative updating in order to fully utilize faster workstations. As soon as an individual is evaluated by a worker node, the master process applies the position updating algorithm and sends out a new work order for that workstation. Due to this, during optimization some individuals would have been updated more times than other individuals, breaking down the discrete iterations. The introduction of this asynchronism introduces additional randomness into the process, given that slower workstations invariably change how the individuals are updated. An analysis of the effect of heterogeneous computing environments on global optimization methods was performed in [63]. It was found that even much slower workstations have positive contributions to the optimization.

4

Microwave Tomography Image Reconstruction Results

Our MWT system was tested entirely in simulation. Instead of a measurement stage involving a physical phantom, a numerical phantom was developed and the measurement data was acquired using an FDTD simulation. The numerical phantoms were assumed to be cylindrical targets with infinite length in the z direction in order to match the two dimensional FDTD used as the forward solver. Multiple numerical phantoms were tested with different resolutions and topologies. The numerical phantoms were constructed similar to those found in Figures 3.1 and 3.2 where homogeneous square patches were assumed truncated at the known radius of the imaging target. Resolution of the phantom can be altered by varying the number of optimization patches located within the target. Figure 4.1 shows a reconstruction target with a total of 9 optimization patches used in the following reconstructions. Each optimization patch has an associated ϵ_∞, $\Delta\epsilon$, and σ as described in Chapter 2.

For initial testing and calibration of the system, measurement data from a totally homogeneous dielectric cylinder was obtained. The numerical phantom had the following Debye parameters: $\epsilon_\infty = 10$, $\Delta\epsilon = 10$ $\sigma = 0$. The target was then reconstructed using a resolution of 21 cm. This resolution corresponds to $3\times$ 3 patches, yielding a total of 27 optimization parameters. ϵ_∞ was assumed to be between 1 and 20, $\Delta\epsilon$ was assumed to be between 0 and 20, and σ was assumed to be between 0 and 1.1 during optimization. Setting correct maximum and minimum bounds cuts down the size of the search space, and thus aids convergence. Figure 4.2a shows the map of both $\Delta\epsilon$ and ϵ_∞ used to create the measurement data. As it turns out, the optimizer had a tendency to underestimate ϵ_∞ values and overestimate values of $\Delta\epsilon$.

For this calibration case, both PSO and DE were tested. The PSO had inertia of 0.8, global best weight of 1.1, and local best weight of 1.8. It was found that inertia values much higher than 0.8 precluded the PSO from converging. The variants of DE

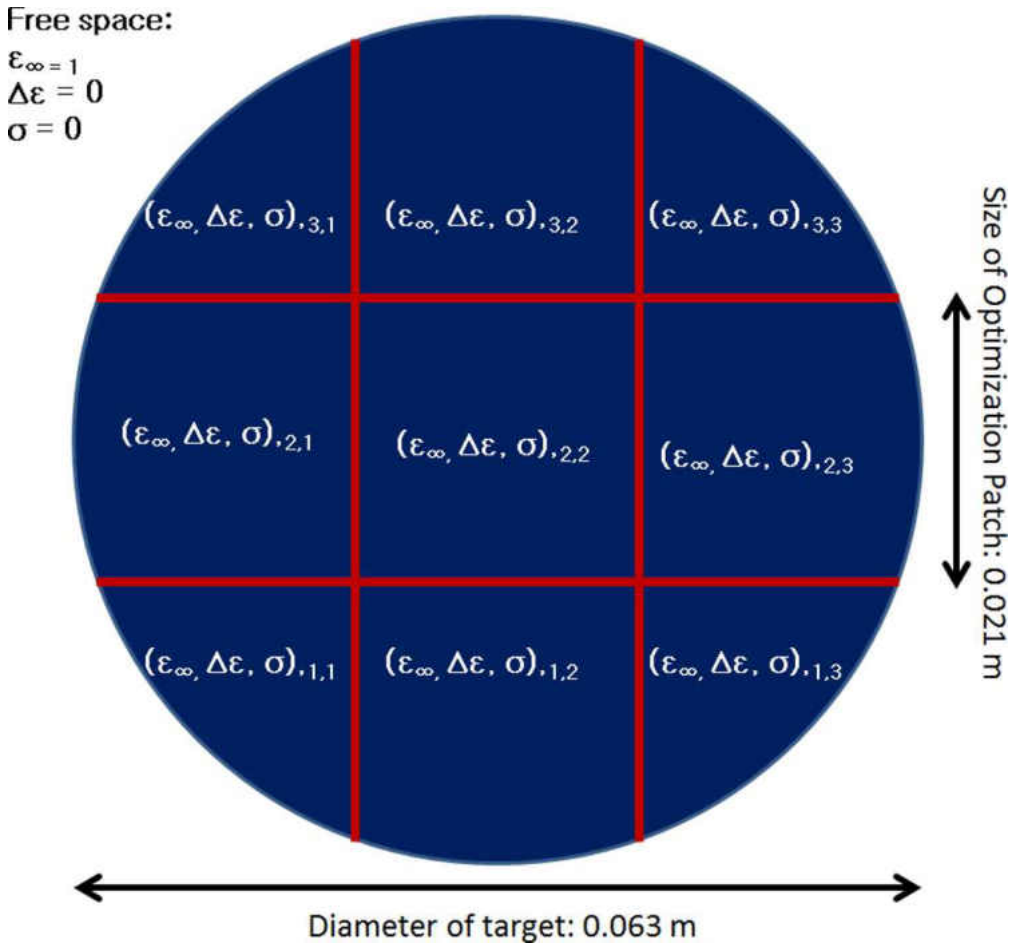


Figure 4.1: Diagram of the reconstruction scheme used.

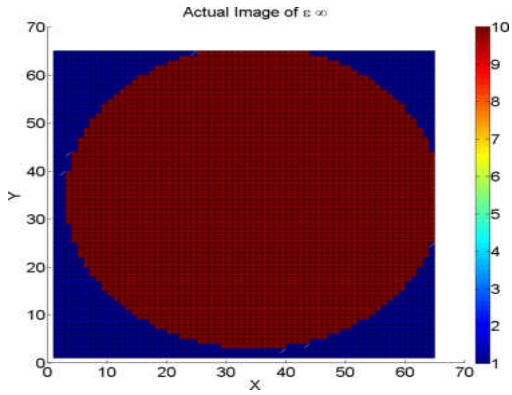
used during this reconstructed included having "best" parent selection with one pair with binary recombination, random parent selection with one pair with binary combination, random parent selection with two pairs with binary recombination, and random parent selection with three pairs and binary recombination. Differential evolution using random parent selection with three pairs and binary recombination had the best results. Figure 4.2c shows a map of the reconstructed map of ϵ_{∞} and Figure 4.2d shows a map of the reconstructed values of $\Delta\epsilon$. σ was also reconstructed correctly. The maximum conductivity reconstructed was 4.195×10^{-11} where the measurement data was formed with a target with zero conductivity everywhere. The MSE (equation (2.1)) in the scattered field produced by most fit individual was found to be 1.4×10^{-4}. This global best individual was the 69,049th evaluation. The optimization took 201,050 seconds, or approximately 56 hours, to reach this fitness value. It was found that a single fitness evaluation for one worker node took an average of 24 seconds. This evaluation time was more or less the same across all work stations. Furthermore, there was very little time that the worker nodes were idle. During this optimization, the maximum amount

of idle time for one of the workers was 0.758 seconds. The time that that worker node spent probing was 20418 seconds, yielding more than 99.99% up time. Table 4.2 shows a comparison between the two optimization methods used for this reconstruction. Figures 4.3 and 4.4 show the best fitness values achieved by the optimizations as a function of fitness evaluations. Table 4.1 shows a quantitative comparison between the true image and the reconstructed images for both PSO and DE. The image accuracy is quantified using a value called the Peak Signal-to-Noise Ratio (PSNR). It is defined in equation (4.1) where MAX is the maximum value that a pixel can take. In most image applications, MAX is defined based on how the pixels are stored, be they stored in an 8-bit byte or double precision float. For the purpose of our calculations MAX is defined based upon which material parameter is being imaged. For $\Delta\epsilon$, MAX is defined to be 40, given that it is the maximum value of $\Delta\epsilon$ one can expect to be found in breast tissue. For ϵ_∞ MAX is defined to be 7 and for σ reconstructions MAX is defined as 0.79. These values are taken from the measurements done by Lazebnik and all in [41]. These values can be seen in Table 4.7. m and n are pixel indices of the two images being compared whereas M and N are the width and length in pixels. Generally PSNR values of 30 or more are considered good, and the higher the PSNR value the better the image is.

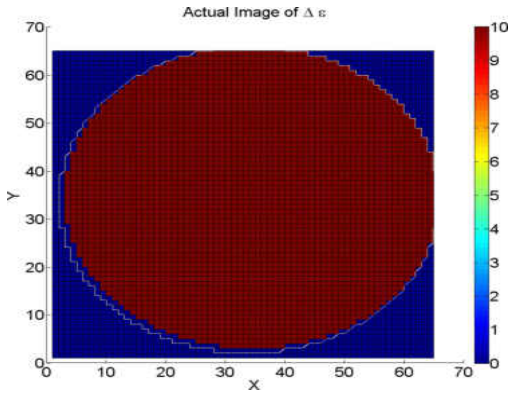
$$PSNR = 10log_{10}(\frac{MAX^2}{MSE}) \tag{4.1}$$

where

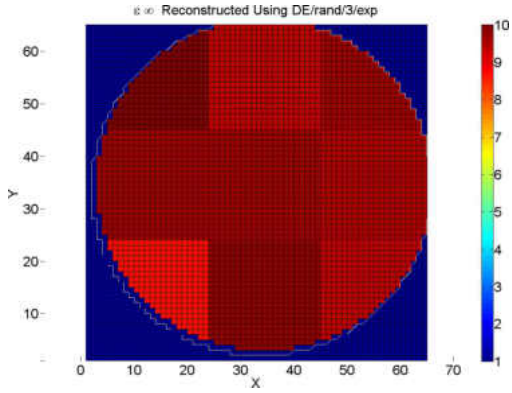
$$MSE = \frac{\sum_{m=1}^{M}\sum_{n=1}^{N}(Pixel_{true}(m,n) - Pixel_{reconstructed}(m,n)^2}{M \times N} \tag{4.2}$$



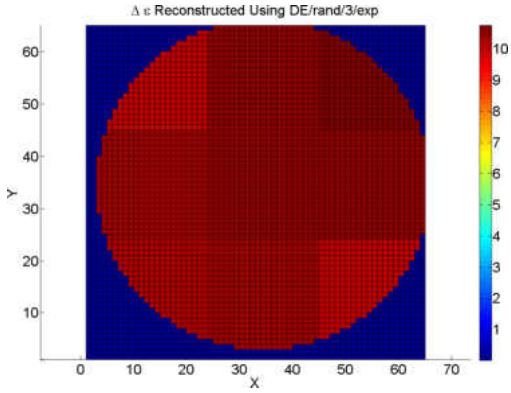
(a) Actual image of ϵ_∞ used in reconstruction of homogeneous imaging target.



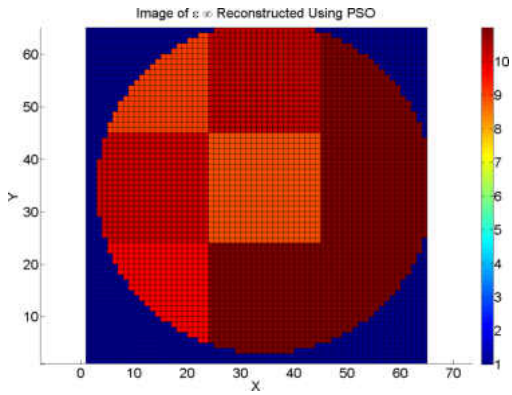
(b) Actual image of $\Delta\epsilon$ used in reconstruction of homogeneous imaging target.



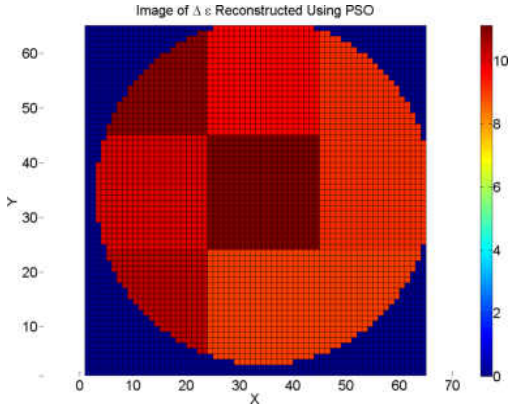
(c) Map of ϵ_∞ for the image reconstructed using DE/rand/3/exp.



(d) Map of $\Delta\epsilon$ for the image reconstructed using DE/rand/3/exp.



(e) Map of ϵ_∞ for image reconstructed using PSO.



(f) Map of $\Delta\epsilon$ for image reconstructed using PSO.

Figure 4.2: Reconstruction of homogeneous target using DE/rand/3/exp and PSO

Table 4.1: Peak Signal-to-Noise Ratio of Reconstructed Images from Figure 4.2

Optimization	PSNR in $\Delta\epsilon$	PSNR in ϵ_∞
DE/rand/3/exp	40.750 dB	23.420 dB
PSO	35.327 dB	20.207 dB

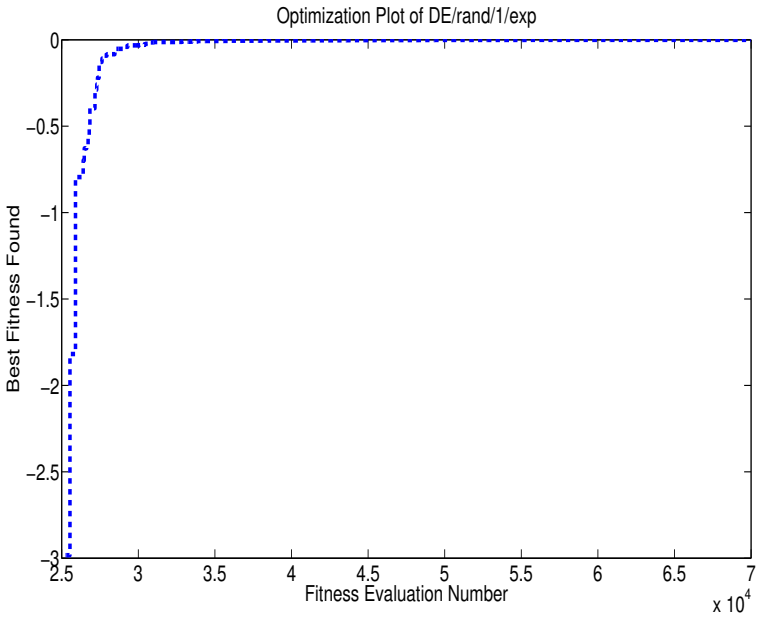


Figure 4.3: Fitness plot of the DE/rand/3/exp optimization used to reconstruct Figure 4.2d.

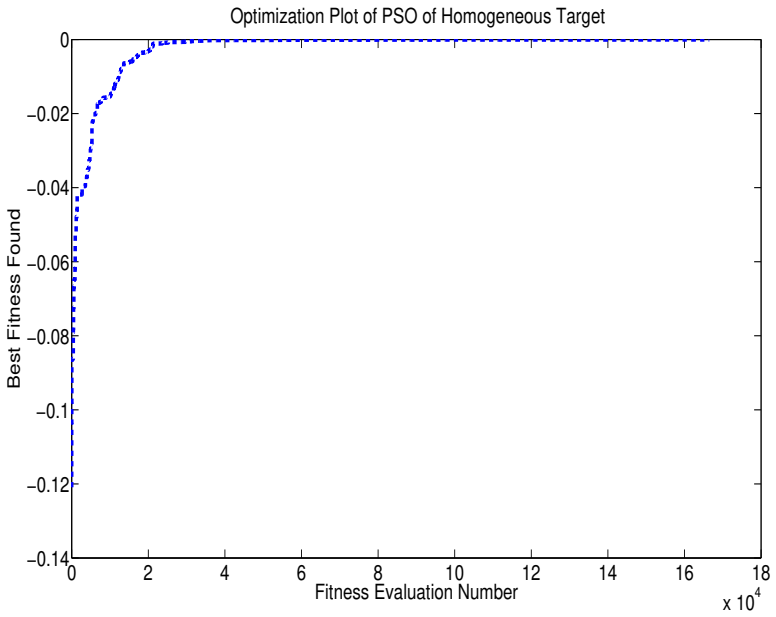


Figure 4.4: Fitness plot of the PSO used to reconstruct Figure 4.2f.

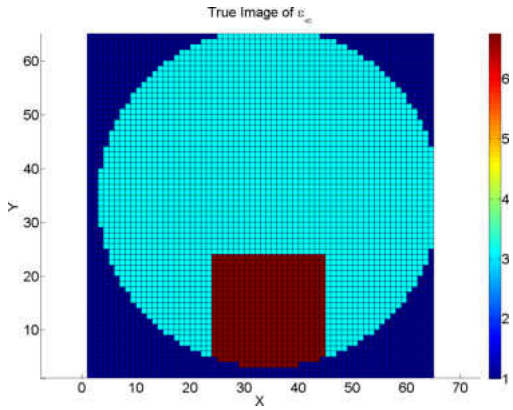
Table 4.2: Comparison of Optimization Methods Used to Reconstruct Figure 4.2d

Optimization	Convergence Time	Final Fitness Value
DE/rand/3/exp	1,657,176 seconds	-0.00014
PSO	4,031,040 seconds	-1.23e-5

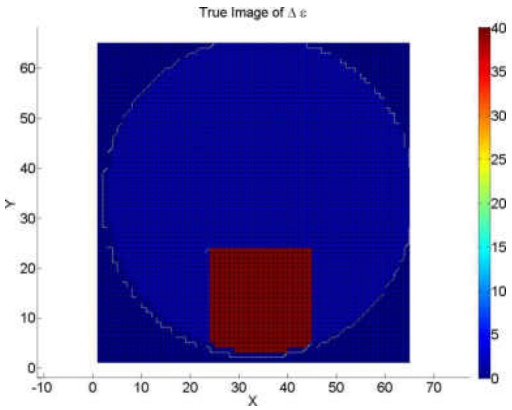
4.1 Complex Target Reconstruction

The reconstruction of a homogeneous target presented above is a relatively easy inverse problem to solve. Increasing the complexity of the target makes the reconstruction thereof more difficult, impeding both the accuracy of the image and the run time of the reconstruction. To test the capability of our system to resolve heterogeneous targets, a numerical phantom was created to emulate a breast containing only fatty tissue and one (unrealistically large) tumour. The regions of the target corresponding to fatty tissue had ϵ_∞ set to 3.14 and $\Delta\epsilon$ set to 1.61, whereas, the tumour had ϵ_∞ set to 6.75 with $\Delta\epsilon$ set to 40. The relaxation time τ was assumed to be 13 ps for all types of tissues. σ was assumed to be 0 for this reconstruction. These material parameters were chosen to model breast tissue (albeit with zero conductivity) that was measured in [41].

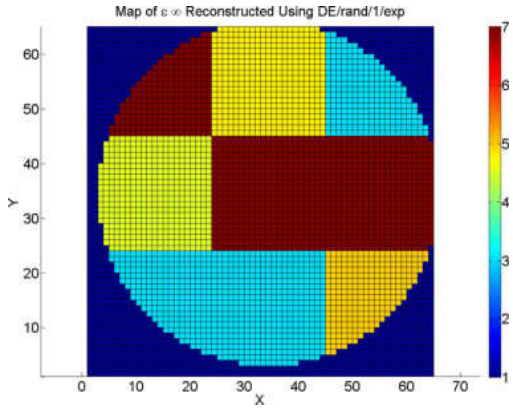
The inverse solver used for reconstruction was chosen as DE with one pair and an exponential recombination given its performance in reconstructing Figure 4.2c and Figure 4.2d. At convergence, the forward solver converged to an image with an MSE in the scattered field of -0.49. This took a total of 59,166 fitness evaluations, which corresponds to a runtime of around 50 hours. The reconstructed images are shown in Figures 4.5c and 4.5d. Figures 4.6 and 4.7 show the best fitness values achieved by each of these optimization methods during reconstruction as a function of the number of fitness evaluations. Table 4.3 shows the PSNR comparison between the true images and the reconstructed images.



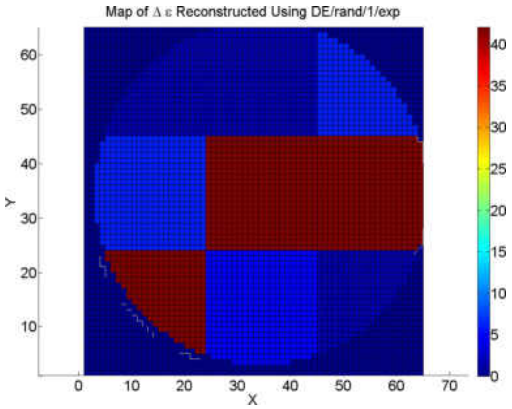
(a) Actual image of ϵ_∞ used to generate measurement data for reconstruction.



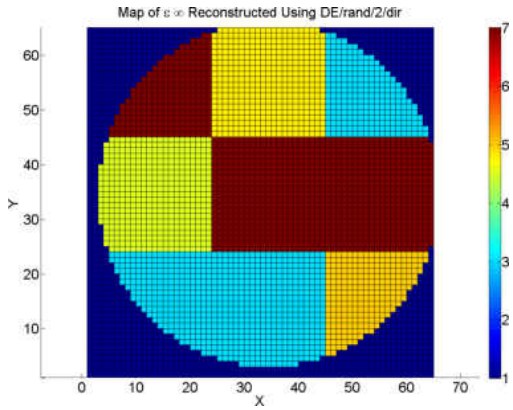
(b) Actual image of $\Delta\epsilon$ used to generate measurement data for reconstruction.



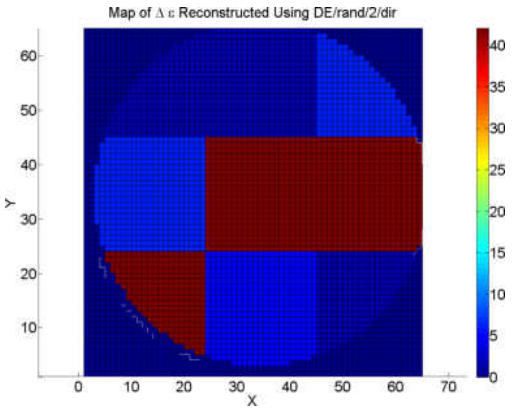
(c) Map of ϵ_∞ Reconstructed Using DE/rand/1/exp.



(d) Map of $\Delta\epsilon$ Reconstructed Using DE/rand/1/exp.



(e) Map of ϵ_∞ Reconstructed Using DE/rand/2/dir.



(f) Map of $\Delta\epsilon$ Reconstructed Using DE/rand/2/dir.

Figure 4.5: Reconstruction of inhomogeneous target using DE/rand/1/exp and DE/rand/2/dir

Table 4.3: Peak Signal-to-Noise Ratio of Reconstructed Images from Figure 4.5

Optimization	PSNR in $\Delta\epsilon$	PSNR in ϵ_∞
DE/rand/1/exp	6.612 dB	9.438 dB
DE/rand/2/dir	6.612 dB	9.438 dB

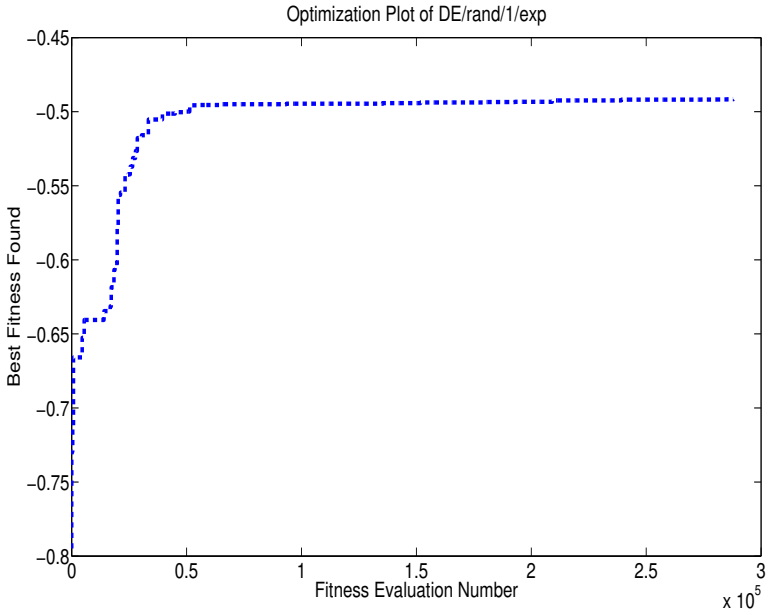


Figure 4.6: Fitness plot of the DE/rand/1/exp optimization used to reconstruct Figure 4.5d and Figure 4.5c.

As can be seen visually, the inverse solver did not converge to the correct image. An optimization plot that shows the level of fitness achieved by the inverse solver as the optimization ran is shown in Figure 4.6. From the plot, it is shown that the inverse solver is able to rapidly increase the fitness during early stages of optimization. After 14,000 fitness evaluations, however, the fitness values stay around the same value. From this it seems likely that the inverse solver has converged upon a local minima of the MWT inverse problem space. Although the simulation was only run for 50 hours, it seemed unlikely that it would escape this local minimum given that the fitness evaluations stayed the same level for so many candidates.

It was reported in [60] that the DE parameters that provided the best performance, generally, for hard optimization problems was a DE optimization with random parent selection, two pairs, and directional recombination. Such an optimization was run using our forward solver. An image was reconstructed with $3 \times 3 \times 3$ optimization patches

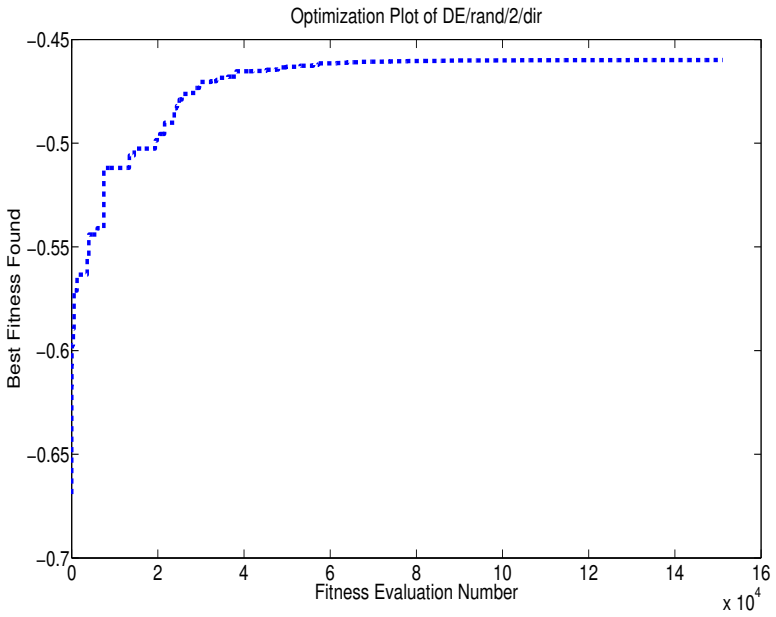


Figure 4.7: Fitness plot of the DE/rand/2/dir optimization used to reconstruct Figures 4.5e and 4.5f.

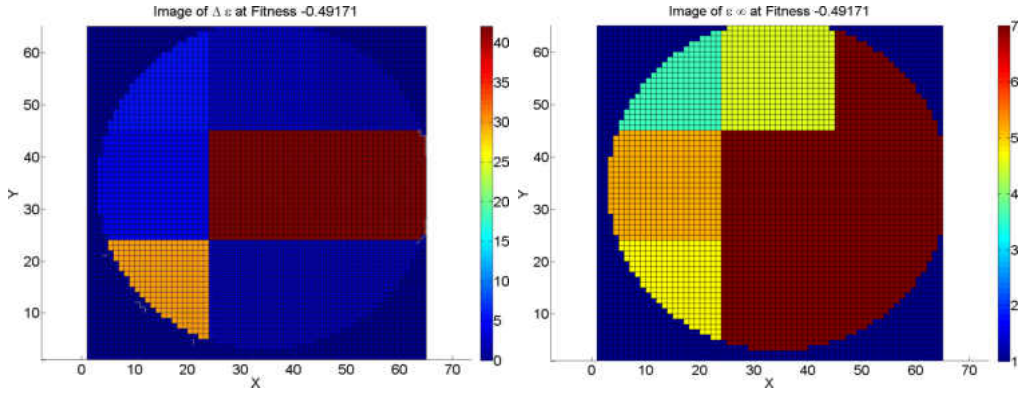
for 217,729 fitness evaluations. The reconstructed image is shown in Figures 4.5e and 4.5f. Much like the DE/rand/1/exp (where this notation is described in section 3.3) optimization, this reconstruction seemingly became stuck in a local minima of the MWT inverse problem space. After 217,447 evaluations, a fitness of -0.4866 was achieved which is very similar to that found by the DE/rand/1/exp simulation (Figure 4.5d). Interestingly, a qualitative inspection of Figures 4.5f and shows that the two different optimization methods converged to extremely similar images. The forward solver was checked by inputting the correct image which was known beforehand to see if it gives an error less than the error found with the reconstructed image. The error in scattered field of the true image as evaluated by our forward solver was found to be 1.21×10^{-21}, which indicates that the true image is, in all likelihood, the global minimum of the MWT inverse problem. Table 4.4 shows a comparison between the two DE optimizations used for this reconstruction.

Table 4.4: Comparison of Optimization Methods Used to Reconstruct Figure 4.5b

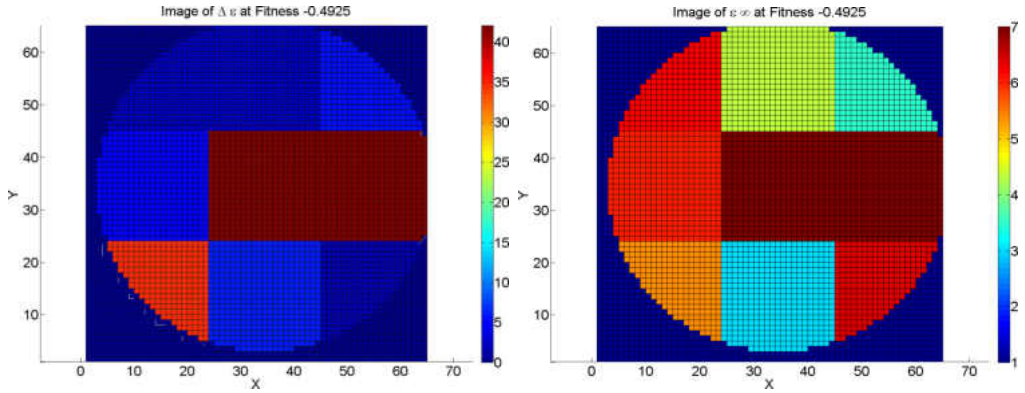
Optimization	Convergence Time	Final Fitness
DE/rand/1/exp	7397632 seconds	-0.491485
DE/rand/2/dir	5218728 seconds	-4.866

For analysis of the relationship between the PSNR and the fitness value of images, we present some of the candidate images that were found created during the DE/rand/1/exp

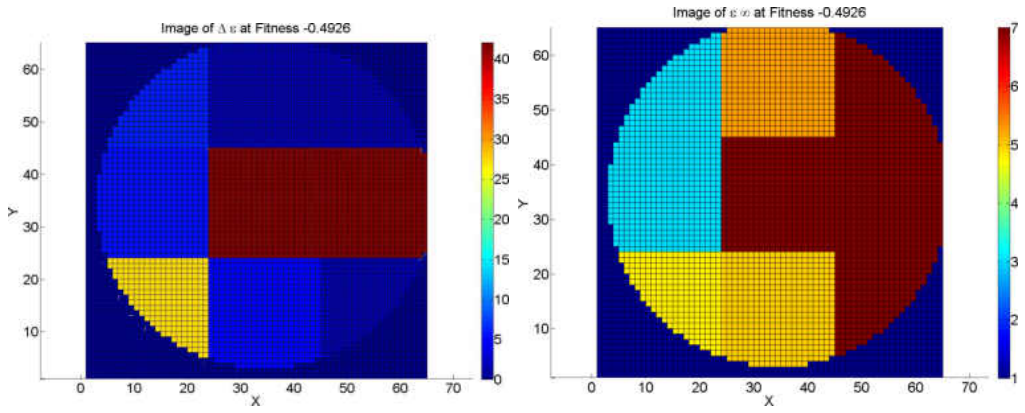
optimization used to reconstruct Figures 4.5c and 4.5d. These candidate images were the global best images at some point during the optimization. This analysis also demonstrates the ill-posedness of the MWT inverse problem in that to very different images can have similar fitness values. Figure 4.8 and Table 4.5 show this data. From the data we can conclude that for candidate images with relatively small (where more negative numbers are defined to be smaller than less negative numbers) fitness values, fitness is not strongly correlated with the PSNR of the associated image. An analysis of candidate images with relatively large fitness values is done in Section 4.3.



(a) Image of ϵ_∞ with fitness of -0.49171. **(b)** Image of $\Delta\epsilon$ with fitness of -0.49171.



(c) Image of ϵ_∞ with fitness of -0.4925. **(d)** Image of $\Delta\epsilon$ with fitness of -0.4925.



(e) Image of ϵ_∞ with fitness of -0.4926. **(f)** Image of $\Delta\epsilon$ with fitness of -0.4926.

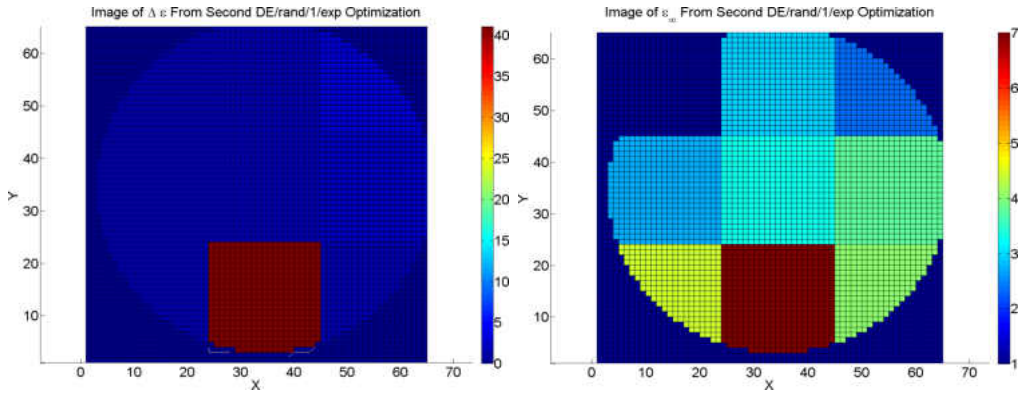
Figure 4.8: Global Best Images Found During Optimization of Figure 4.5.

Because of randomization inherent in optimization techniques, different optimiza-

Table 4.5: Peak Signal-to-Noise Ratio of Reconstructed Images from Figure 4.8

Associated Fitness	PSNR in $\Delta\epsilon$	PSNR in ϵ_∞
-0.49171	6.471 dB	7.962 dB
-0.49251	6.314 dB	9.678 dB
-0.49262	6.112 dB	9.019 dB

tions with the same input parameters may converge on different images. A second DE/rand/1/exp optimization with measurement data from the image shown in Figure 4.5b and Figure 4.5a was run to investigate this. The results of this reconstruction are shown in Figure 4.9. The PSNR values of these images are given in Table 4.6.



(a) Image of ϵ_∞ from second DE/rand/1/exp reconstruction.

(b) Image of $\Delta\epsilon$ from second DE/rand/1/exp reconstruction.

Figure 4.9: Results of second DE/rand/1/exp Reconstruction of Figure 4.5b

Table 4.6: Peak Signal-to-Noise Ratio of Reconstructed Images from Figure 4.8

Associated Fitness	PSNR in $\Delta\epsilon$	PSNR in ϵ_∞
-0.04650775	38.573 dB	19.826 dB

4.2 Conductivity Imaging

The reconstruction static conductivity remains a difficult problem for current deterministic optimization based MWT systems [29]. In order to test the proposed system's capability to correctly image the static conductivity of a target, a numerical phantom was created with non-zero conductivity with Debye dispersion. Although the Debye

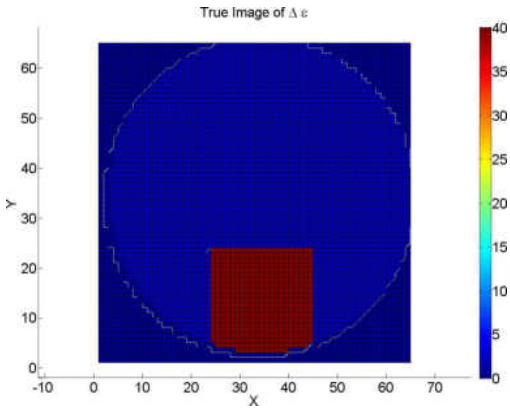
model does account for some part of the imaginary component of complex permittivity (and thus to lossiness of the material), real materials can also have non-zero values of σ. With non-zero values of σ, $\Delta\epsilon$, and ϵ_∞ the expression for the total complex permittivity takes the form of equation (4.3).

$$\epsilon = \epsilon_0\{\epsilon_\infty + \frac{\Delta\epsilon}{1 + j\omega\tau}\} + j\frac{\sigma}{\omega} \tag{4.3}$$

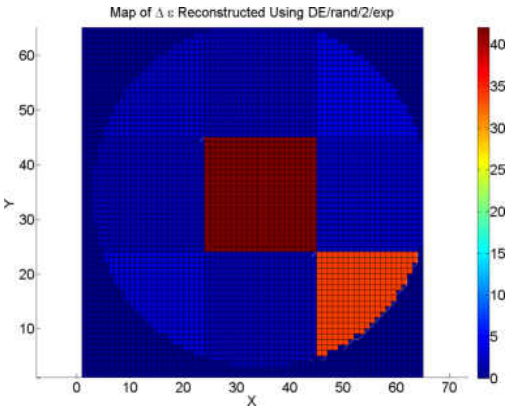
A numerical phantom was created to match the material properties measured in breast tissue by Lazebnik et al. [41]. Two types of materials were modelled, fatty tissue and malignant tissue. The material parameters of these tissues are given in table 4.7. For our reconstruction, the phantom was composed mostly of fatty tissue with one optimization patch being composed of malignant tissue. The results of this reconstruction are shown in Figures 4.10a through 4.10f. The optimization plot of the reconstruction is shown in Figure 4.11. The final fitness reached by this reconstruction attempt was -0.316968. The optimization took a total of 4,914,768 to reach this fitness. Table 4.8 shows the PSNR comparison between the true imaging target and the reconstructed image.

Table 4.7: Material Parameters of Tissue Used for Reconstruction of Target with Non-Zero Conductivity

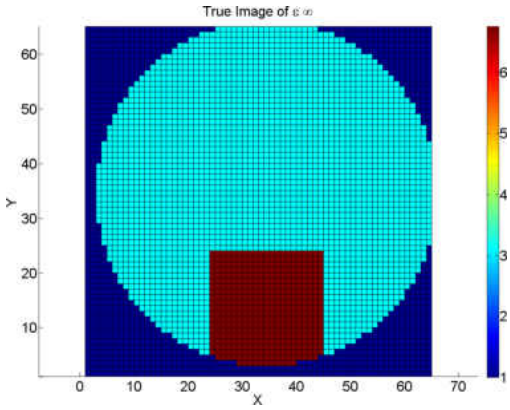
	ϵ_∞	$\Delta\epsilon$	σ (S/m)
Fatty Tissue	3.14	1.60	0.036
Malignant Tissue	6.75	40.0	0.790



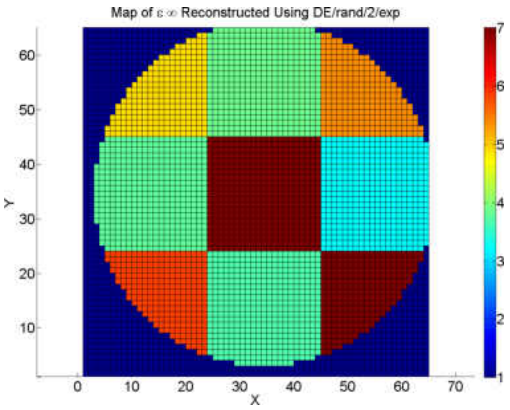
(a) Actual image of $\Delta\epsilon$



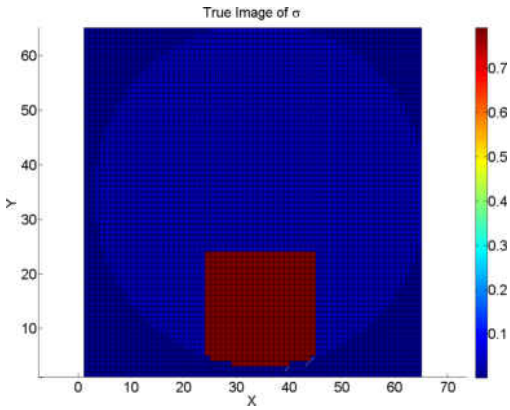
(b) $\Delta\epsilon$ Reconstructed using DE/rand/2/dir.



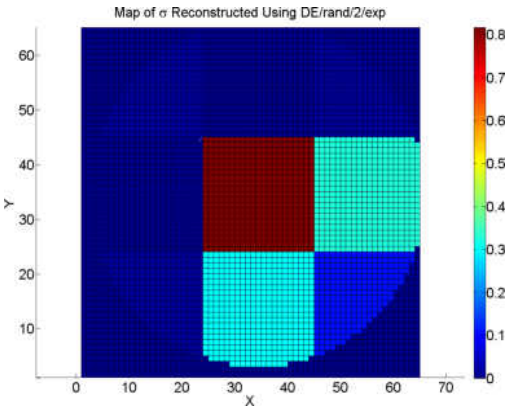
(c) Actual image of ϵ_∞



(d) Map of ϵ_∞ Reconstructed Using DE/rand/2/dir.



(e) Actual image of σ



(f) Map of σ Reconstructed Using DE/rand/2/dir.

Figure 4.10: Reconstruction Inhomogeneous Target With Non-Zero Conductivity

Table 4.8: Peak Signal-to-Noise Ratio of Reconstructed Images from Figure 4.10

PSNR in $\Delta\epsilon$	PSNR in ϵ_∞	PSNR in σ
6.245 dB	10.559 dB	8.378

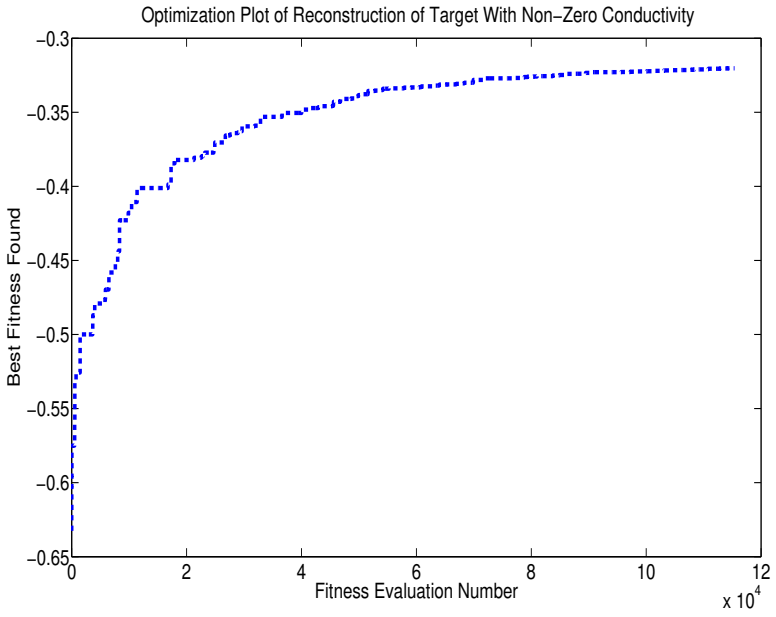
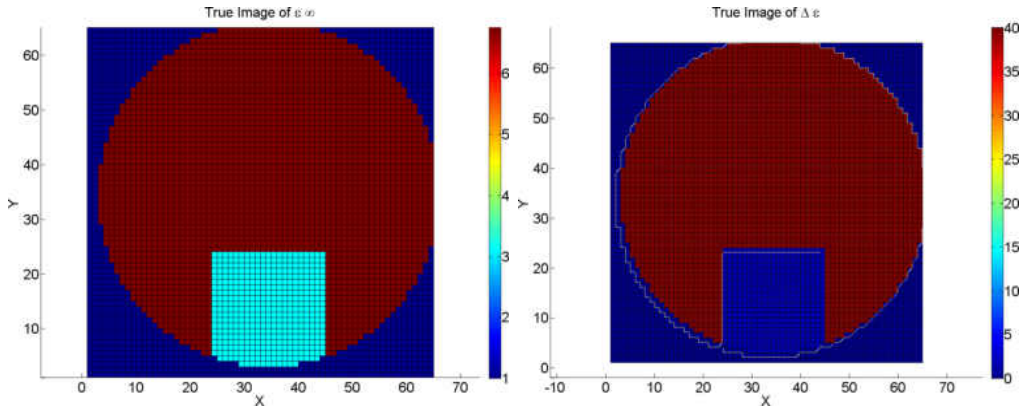


Figure 4.11: Fitness plot of the optimization used to reconstruct Figure 4.10.

4.3 Non-Biological Imaging

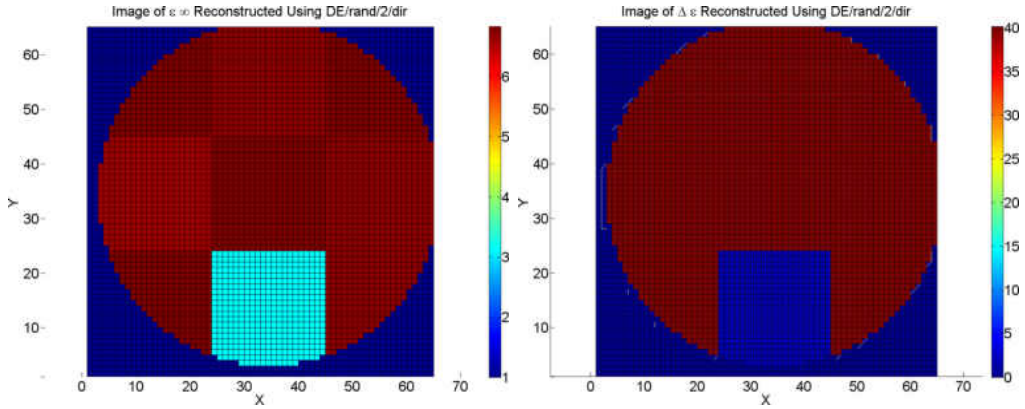
While the numerical phantoms presented previously in this chapter have been modelled to emulate malignant tumours embedded within normal breast tissue, MWT has many more applications than medical imaging. One proposed use of MWT is in the field of non-destructive testing for engineering materials such as high voltage insulators [64]. In such applications one attempts to detect voids in otherwise strongly scattering media. This is in contrast to the previous examples wherein we attempted to detect strongly scattering tumours embedded within weakly scattering fatty tissue.

For testing of such a scenario, a numerical phantom was made with background medium of ϵ_∞ of 6.75 and $\Delta\epsilon$ of 40. The target had a single discontinuity with ϵ_∞ of 3.14 and $\Delta\epsilon$ of 1.61. Figure 4.12 shows the numerical phantom and reconstructed image.



(a) Actual image of ϵ_∞ used to generate measurement data for reconstruction.

(b) Actual image of $\Delta\epsilon$ used to generate measurement data for reconstruction.



(c) Map of ϵ_∞ Reconstructed Using DE/rand/1/exp.

(d) Map of $\Delta\epsilon$ Reconstructed Using DE/rand/1/exp.

Figure 4.12: Reconstruction of inhomogeneous target using DE/rand/2/dir of non-biological numerical phantom

As can be seen in Figure 4.12, $\Delta\epsilon$ was reconstructed nearly exactly while ϵ_∞ was constructed with minimal error. The void was located successfully at the bottom center of the image. The reconstruction process took a total of 69818 fitness evaluations to find this image, which corresponds to 51 hours of computation. An optimization plot of the reconstruction is shown in Figure 4.13. The PSNR comparison between the true and reconstructed image is shown in Table 4.9.

Table 4.9: Peak Signal-to-Noise Ratio of Reconstructed Images from Figure 4.12

PSNR in $\Delta\epsilon$	PSNR in ϵ_∞
57.676 dB	42.553 dB

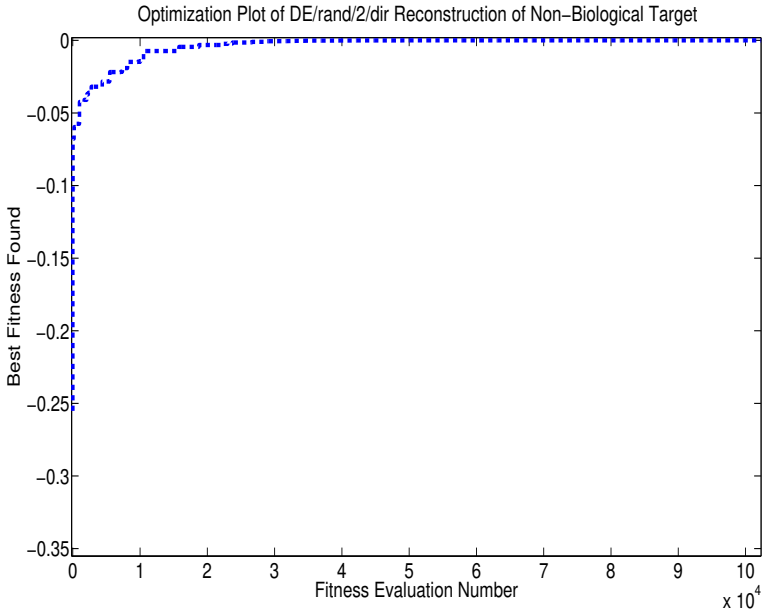
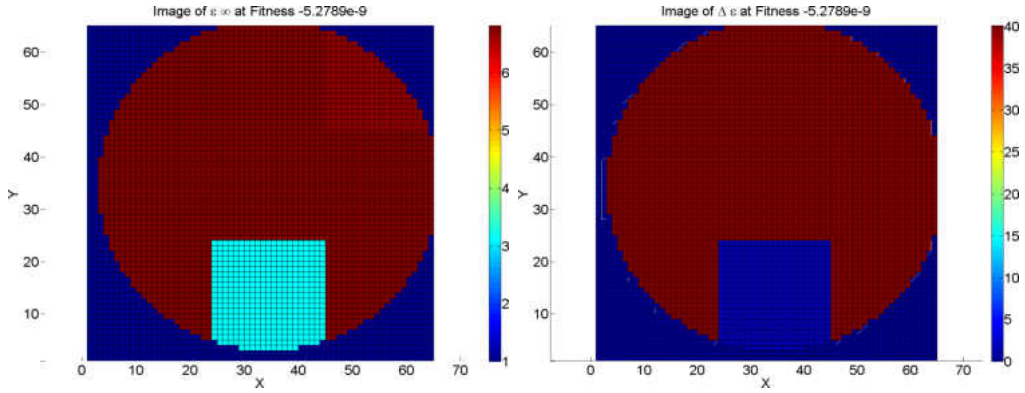
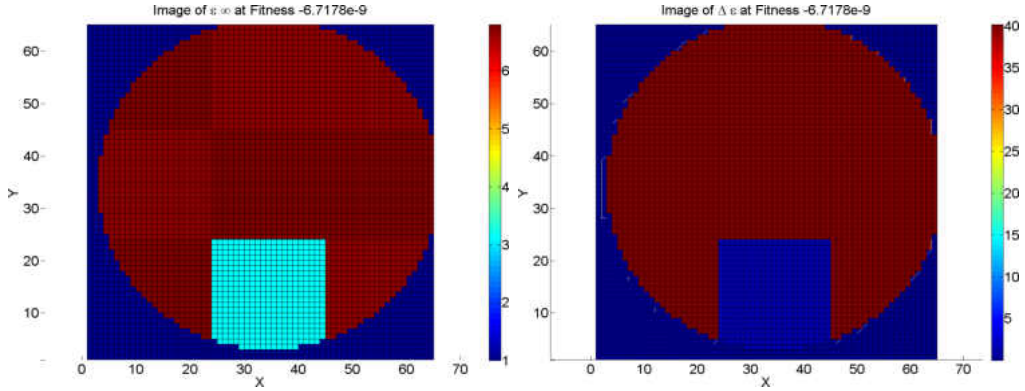


Figure 4.13: Fitness plot of the optimization used to reconstruct Figure 4.12.

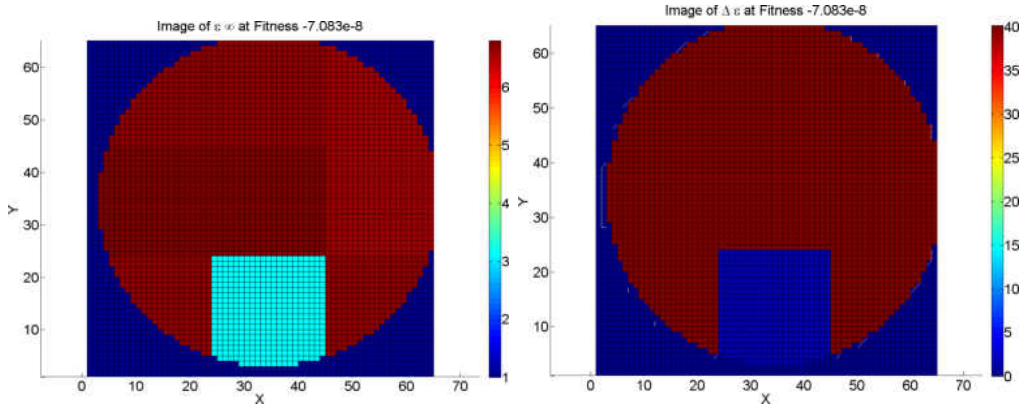
To analyse the relationship between the PSNR of candidate images and their fitness values when the optimization has found candidate images with high fitness (with MSE near zero), we present candidate images that were found during the optimization that yielded Figure 4.12. Their PSNR values and associated fitness values are given in Table 4.10 and the candidate images themselves are presented in Figure 4.14. It seems that for candidate images with high fitness values, the fitness is strongly correlated with the accuracy of the images measured by their PSNR values.



(a) Image of ϵ_∞ with fitness of -5.279e-9. **(b)** Image of $\Delta\epsilon$ with fitness of -5.279e-9.



(c) Image of ϵ_∞ with fitness of -6.7178e-9. **(d)** Image of $\Delta\epsilon$ with fitness of -6.7178e-9.



(e) Image of ϵ_∞ with fitness of -7.083e-8. **(f)** Image of $\Delta\epsilon$ with fitness of -7.083e-8.

Figure 4.14: Global Best Images Found During Optimization of Figure 4.12.

Table 4.10: Peak Signal-to-Noise Ratio of Reconstructed Images from Figure 4.12

Associated Fitness	PSNR in $\Delta\epsilon$	PSNR in ϵ_∞
-5.279e-9	62.524 dB	48.607 dB
-6.7178e-9	56.795 dB	41.497 dB
-7.083e-9	55.259 dB	38.919 dB

5

Conclusion and Future Work

In this thesis we have presented an MWT system which does not rely on either regularization nor linearization assumptions to solve the inverse problem. Stochastic optimization methods are used as the inverse solver for their ability to avoid convergence to local minima of the MWT problem space through randomization. The proposed system takes measurements of the scattered field of an unknown target as input and returns the distribution of permittivity inside a cross section of the target. The system presented herein is capable of modelling dispersive media (specifically dispersive media that can be accurately described with the Debye model), such as biological tissue. The system was tested with a resolution of 2.1 cm. In order to ameliorate long run-times necessary for our method, a heavy parallelization is used in both the inverse and forward solvers.

In order to resolve images, the proposed system uses an $(FD)^2TD$ forward solver to find the scattered field created by a proposed image and a stochastic optimization method based inverse solver to search for the optimal image. The inverse solver begins by generating a population of random images and sending them to the forward solver for evaluation. The forward solver generates the scattered field that would be formed from each of these random images and checks how closely they match the scattered field that was measured from the true image. As fitness evaluations are returned to the inverse solver, the inverse solver proposes new images according to its updating scheme. This continues until the inverse solver has converged upon an image.

The $(FD)^2TD$ forward solver is implemented using CUDA such that the most computationally intensive process of the $(FD)^2TD$, the time-stepping, is performed in parallel on a GPU. The forward solver operates in two dimensions in the TM case (where the electric field is oriented perpendicular to the plane that the simulation is in). The GPU implementation of the forward solver cuts the computation time required for a fitness evaluation approximately one hundred fold, and thus, cuts the time required for image resolution heavily.

The inverse solver used in the proposed system is parallelized in the sense that it

can farm out candidate image to multiple workstations to be evaluated in parallel. The stochastic inverse solver operates asynchronously, wherein candidate images are generated and farmed out as soon as a workstation is free. This aids in keeping the idle time of the work stations as low as possible, as well as increasing the robustness of the system to worker unreliability and improving the scalability of the system. During the experiments presented in Chapter 4, our system used a total of eight GPUs. These GPUs were distributed across four worker nodes, each having two Tesla c1060 GPUs. Our system can be easily scaled to use more GPUs, which would further decrease the time required for image resolution. Stochastic optimization methods considered in our system include PSO and DE. Similar systems have made use of the genetic algorithm [29]. We chose to use PSO and DE because they have been observed to outperform the standard genetic algorithm across a wide variety of problems [60].

Our system has shown to be capable of correctly reconstructing a simple homogeneous dispersive imaging target. The introduction of heterogeneity in the imaging target drastically hampers reconstruction of the correct image, where the inverse solver would be trapped inside of a local minimum of the MWT problem space corresponding to an incorrect image. In order to image more heterogeneous targets, the inverse solver ought to be altered to include more exploratory behaviour to escape local minima.

The author's main contributions to this project were as follows:

- Development of GPU accelerated FDTD forward solver.
- Implementation of two dimensional parallelization within GPU accelerated FDTD forward solver.
- Derivation and implementation of scattered-field Debye model dispersion $(FD)^2TD$ using polarization current model.
- Integration of $(FD)^2TD$ forward solver with TAO library to create a full MWT inverse solver.

From the simulations presented herein, we have found that for the limited cases we have explored DE seems to offer superior accuracy and is more resistant to convergence on local minima of the MWT inverse problem space than PSO. For a full analysis of PSO vs. DE for the purpose of non-linearized MWT inverse solver, multiple PSO reconstructions ought to be executed with different parameters of local best weight, global best weight, and inertia. We found that DE optimizations with random parent selection and exponential or exponential and directional, recombination with 400 or more individuals were able to correctly reconstruct some images. Scalability analysis shows

that all eight GPU worker nodes were evaluating fitness values for 99.9%, meaning that they spent very little time idle.

5.1 Future Work

While the proposed system shows promise as an imaging modality for dielectric and conductive objects, significant improvements are necessary to image objects with good reliability, accuracy and acceptable run-time. Further research is necessary in order to create an economical and effective MWT system.

- Run-time can be decreased with the use of advanced computer hardware. Given the large amount of memory transferred between the CPU and GPU in our algorithm, implementation of the forward solver on AMD APU architecture may offer significant speed increases.

- The stochastic optimization methods investigated herein would often converge on local minima for some problems. Investigation into other optimization methods such as hybrid stochastic/deterministic optimization methods may improve convergence time and reliability. Optimizations we used may also be altered to increase exploratory behaviour to avoid convergence to local minima.

- The FD^{2}TD forward solver can be easily altered to use Lorentz or Drude dispersion models for materials that can be more accurately described by these models.

- Develop and implement proper stopping criterion for the stochastic optimization methods.

- Given the scalability of our system, increasing the amount of GPUs available for computation should drastically decrease the run-time required per image. Integration of our system with grid computing [60] may make MWT systems based on non-linearized and non-regularized stochastic inverse solvers attractive despite the high amount of computational power necessary for such systems.

- The optimization scheme used in the image reconstructions herein suffers severely from curse of dimensionality, although it has the ability to theoretically reconstruct an object with an arbitrary permittivity profile. As the resolution of our system increases, the number of variables to optimize on increases by the square of the reciprocal of the resolution. *A priori* information may be used to decrease the search space drastically. In the context of MWT as an imaging modality for breast cancer detection, the number of suspect objects contained within the breast would

be known from a mammography scan. This information can be used for imaging resolution in order to drastically reduce the amount of acceptable candidate images.

- All results presented in this thesis were created purely in simulation. Future research should integrate our system into a hardware apparatus. Measurement values would then be received from real-world data, rather than those generated in simulation.

- The transition from two-dimensional imaging to three-dimensional imaging would be fairly straightforward to implement, however it would present a drastic increase in run-time and may hamper convergence to the correct image. For transition to three-dimensional imaging, the robustness of the inverse solver as well as its speed must be increased .

- The efficacy of MWT as an imaging modality for breast cancer detection (or other usage) ought to be assessed upon development of a reliable, practical, and robust MWT system. In the realm of breast cancer detection, MWT is attractive in its potential to differentiate malignant structures from healthy tissue based on electrical material parameters, rather than the shape of the structure as is detected by mammography. This modality of differentiation may increase the specificity of breast cancer screening. Research thereof needs to involve human trials.

References

[1] S. Zhong, Y. Yan, and Y. Shen, "Non-destructive testing of gfrp materials by fourier-domain infrared optical coherence tomography," in *Automatic Control and Artificial Intelligence (ACAI 2012), International Conference on*, 2012, pp. 1407–1410. 1

[2] L. Ma, Z. zhao Zhang, and X. Tan, "Two-step imaging method and resolution analysis for uwb through wall imaging," in *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on*, 2008, pp. 1–5. 1

[3] R. Halter, A. Hartov, and K. Paulsen, "A broadband high-frequency electrical impedance tomography system for breast imaging," *Biomedical Engineering, IEEE Transactions on*, vol. 55, no. 2, pp. 650–659, 2008. 2

[4] *Dräger PulmoVista 500*. 2

[5] C.-W. Sun, "Development of diffuse optical imaging systems for clinical applications," in *Communications and Photonics Conference and Exhibition (ACP), 2010 Asia*, 2010, pp. 42–43. 2

[6] M. Xu, M. Alrubaiee, S. K. Gayen, and R. Alfano, "Optical diffuse imaging of an ex vivo model cancerous human breast using independent component analysis," *Selected Topics in Quantum Electronics, IEEE Journal of*, vol. 14, no. 1, pp. 43–49, 2008. 2

[7] S.-C. Lee, K. Kim, J. Kim, S. Lee, J. Han Yi, S. Woo Kim, K.-S. Ha, and C. Cheong, "One Micrometer Resolution NMR Microscopy," *Journal of Magnetic Resonance*, vol. 150, pp. 207–213, Jun. 2001. 2

[8] S. Semenov, V. Posukh, A. Bulyshev, T. Williams, P. Clark, Y. Sizov, A. Souvorov, and B. Voinov, "Development of microwave tomography for functional cardiac imaging," in *Biomedical Imaging: Nano to Macro, 2004. IEEE International Symposium on*, 2004, pp. 1351–1353 Vol. 2. 3

[9] S. Semenov, A. Bulyshev, A. Souvorov, A. Nazarov, Y. Sizov, R. Svenson, V. Posukh, A. Pavlovsky, P. Repin, and G. Tatsis, "Three-dimensional microwave tomography: experimental imaging of phantoms and biological objects," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 48, no. 6, pp. 1071–1074, 2000. 3, 5

[10] A. Golnabi, P. Meaney, and K. Paulsen, "Tomographic microwave imaging with incorporated prior spatial information," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 61, no. 5, pp. 2129–2136, 2013. 3, 5

[11] M. Ostadrahimi, P. Mojabi, S. Noghanian, J. LoVetri, and L. Shafai, "A multiprobe-per-collector modulated scatterer technique for microwave tomography," *Antennas and Wireless Propagation Letters, IEEE*, vol. 10, pp. 1445–1448, 2011. 4

[12] K. Arunachalam, L. Udpa, and S. Udpa, "A computational investigation of microwave breast imaging using deformable reflector," *Biomedical Engineering, IEEE Transactions on*, vol. 55, no. 2, pp. 554–562, 2008. 4

[13] S. Semenov, R. Svenson, A. Bulyshev, A. Souvorov, A. Nazarov, Y. Sizov, A. Pavlovsky, V. Borisov, B. Voinov, G. Simonova, A. Starostin, V. Posukh, G. Tatsis, and V. Baranov, "Three-dimensional microwave tomography: experimental prototype of the system and vector born reconstruction method," *Biomedical Engineering, IEEE Transactions on*, vol. 46, no. 8, pp. 937–946, 1999. 5

[14] S. Semenov, A. Bulyshev, A. Souvorov, R. Svenson, Y. Sizov, V. Vorisov, V. Posukh, I. Kozlov, A. Nazarov, and G. Tatsis, "Microwave tomography: theoretical and experimental investigation of the iteration reconstruction algorithm," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 46, no. 2, pp. 133–141, 1998. 5

[15] I. Catapano, L. Crocco, R. Persico, M. Pieraccini, and F. Soldovieri, "Linear and nonlinear microwave tomography approaches for subsurface prospecting: Validation on real data," *Antennas and Wireless Propagation Letters, IEEE*, vol. 5, no. 1, pp. 49–53, 2006. 5

[16] F. Soldovieri, O. Lopera, and S. Lambot, "Combination of advanced inversion techniques for an accurate target localization via gpr for demining applications," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 49, no. 1, pp. 451–461, 2011. 5

[17] A. Ashtari, S. Noghanian, A. Sabouni, J. Aronsson, G. Thomas, and S. Pistorius, "Using a priori information for regularization in breast microwave image reconstruction," *Biomedical Engineering, IEEE Transactions on*, vol. 57, no. 9, pp. 2197–2208, 2010. 5

[18] N. Corporation, "Cuda tookit documentation," http://docs.nvidia.com/cuda/cuda-c-programming-guide/, 2012. 6, 25

[19] W. Yu, *Parallel Finite-Difference Time-Domain Method*, ser. Artech House Electromagnetic Analysis Series. Artech House, Incorporated, 2006. [Online]. Available: http://books.google.com/books?id=_FyqQgAACAAJ 7, 24

[20] A. Branover, D. Foley, and M. Steinman, "Amd fusion apu: Llano," *Micro, IEEE*, vol. 32, no. 2, pp. 28–37, 2012. 7

[21] M. Daga and M. Nutter, "Exploiting coarse-grained parallelism in b+ tree searches on an apu," in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, 2012, pp. 240–247. 8

[22] S. Semenov, R. Svenson, A. Bulyshev, A. Souvorov, A. Nazarov, Y. Sizov, V. Posukh, A. Pavlovsky, P. Repin, A. Starostin, B. Voinov, M. Taran, G. Tatsis, and V. Baranov, "Three-dimensional microwave tomography: initial experimental imaging of animals," *Biomedical Engineering, IEEE Transactions on*, vol. 49, no. 1, pp. 55–63, 2002. 4, 8, 31

[23] S. Semenov, R. Svenson, K. Dezern, M. Quinn, M. Thompson, and G. Tatsis, "Myocardial ischemia and infarction can be detected by microwave spectroscopy. i. experimental evidence," in *Engineering in Medicine and Biology Society, 1996. Bridging Disciplines for Biomedicine. Proceedings of the 18th Annual International Conference of the IEEE*, vol. 4, 1996, pp. 1361–1362 vol.4. 8

[24] W. Berg, L. Gutierrez, M. NessAiver, W. Carter, M. Bhargavan, R. Lewis, and O. Ioffe, "Diagnostic accuracy of mammography, clinical examination, us, and mr imaging in preoperative assessment of breast cancer." *Radiology*, vol. 233, no. 3, pp. 830–49, 2004. 9

[25] M. Lazebnik, D. Popovic, L. McCartney, C. B. Watkins, M. J. Lindstrom, J. Harter, S. Sewall, T. Ogilvie, A. Magliocco, T. M. Breslin, W. Temple, D. Mew, J. H. Booske, M. Okoniewski, and S. C. Hagness, "A large-scale study of the ultrawideband microwave dielectric properties of normal, benign and malignant breast tissues obtained from cancer surgeries," *Physics in Medicine and Biology*, vol. 52, no. 20, p. 6093, 2007. [Online]. Available: http://stacks.iop.org/0031-9155/52/i=20/a=002 9, 11

[26] A. Hassan and M. El-Shenawee, "Review of electromagnetic techniques for breast cancer detection," *Biomedical Engineering, IEEE Reviews in*, vol. 4, pp. 103–118, 2011. 9

[27] A. J. P. Huynh and S. Daye, "The false-negative mamogram," RadioGraphics, pp. 1137-1154, vol. 18, 1998. 9

[28] "Understanding the fcc regulations for low-power, non-licensed transmitters," Office of Engineering and Technology Federal Communications Comission, Tech. Rep., 1993. 9

[29] A. Sabouni, "Ultra-wideband (uwb) microwave tomography for heterogeneous and dispersive media," Ph.D. dissertation, University of Manitoba, Winnipeg, Manitoba, Canada, February 10 2011. 10, 52, 61

[30] L. Li, H. Zheng, and F. Li, "Two-dimensional contrast source inversion method with phaseless data: Tm case," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 47, no. 6, pp. 1719–1736, 2009. 11

[31] P. Meaney, T. Grzegorczyk, E. Attardo, and K. Paulsen, "Ultrafast 3d microwave tomography utilizing the direct dipole approximation," in *Electromagnetics in Advanced Applications (ICEAA), 2012 International Conference on*, 2012, pp. 838–839. 11

[32] M. Hashemi and M. El-Shenawee, "A comparative study of different tomography methods for breast cancer application," in *Region 5 Technical Conference, 2007 IEEE*, 2007, pp. 21–24. 11

[33] K. Yee, "Numerical solution of initial boundary value problems involving maxwell's equations in isotropic media," *Antennas and Propagation, IEEE Transactions on*, vol. 14, no. 3, pp. 302–307, 1966. 13

[34] M. Bettencourt, C. Lenyk, and T. Fleming, "Analysis of adaptive mesh refinement methods for fdtd particle-in-cell techniques for em simulations," in *Plasma Science, 2004. ICOPS 2004. IEEE Conference Record - Abstracts. The 31st IEEE International Conference on*, 2004, pp. 337–. 14

[35] A. Elsherbeni and V. Demir, *The Finite Difference Time Domain Method for Electromagnetics: With MATLAB Simulations*. SciTech Publishing, Incorporated, 2009. [Online]. Available: http://books.google.ca/books?id=3KMKOwAACAAJ 16, 18

[36] K. Kunz and R. Luebbers, *The Finite Difference Time Domain Methods for Electromagnetics*. Taylor & Francis, 1993. [Online]. Available: http://books.google.com/books?id=00on9fRvJqIC 17

[37] A. Taflove and S. Hagness, *Computational Electrodynamics: The Finite-Difference Time- Domain Method*, ser. The Artech House antenna and propagation library. Artech House, Incorporated, 2005. [Online]. Available: http://books.google.com/books?id=n2ViQgAACAAJ 17, 19, 21

[38] S. Rengarajan and Y. Rahmat-Samii, "The field equivalence principle: illustration of the establishment of the non-intuitive null fields," *Antennas and Propagation Magazine, IEEE*, vol. 42, no. 4, pp. 122–128, 2000. 19, 20

[39] K. S. Cole and R. H. Cole, "Dispersion and absorption in dielectrics i. alternating current characteristics," *The Journal of Chemical Physics*, vol. 9, no. 4, pp. 341–351, 1941. [Online]. Available: http://link.aip.org/link/?JCP/9/341/1 21

[40] K. Oughstun and N. Cartwright, "On the lorentz-lorenz formula and the lorentz model of dielectric dispersion," *Opt. Express*, vol. 11, no. 13, pp. 1541–1546, Jun 2003. [Online]. Available: http://www.opticsexpress.org/abstract.cfm?URI=oe-11-13-1541 21

[41] M. Lazebnik, M. Okoniewski, J. Booske, and S. Hagness, "Highly accurate debye models for normal and malignant breast tissue dielectric properties at microwave frequencies," *Microwave and Wireless Components Letters, IEEE*, vol. 17, no. 12, pp. 822–824, 2007. 21, 22, 42, 45, 53

[42] M. Okoniewski, M. Mrozowski, and M. Stuchly, "Simple treatment of multi-term dispersion in fdtd," *Microwave and Guided Wave Letters, IEEE*, vol. 7, no. 5, pp. 121–123, 1997. 21

[43] S.-C. Kong, J. Simpson, and V. Backman, "Ade-fdtd scattered-field formulation for dispersive materials," *Microwave and Wireless Components Letters, IEEE*, vol. 18, no. 1, pp. 4–6, 2008. 23

[44] A. Sabouni, A. Ashtari, S. Noghanian, G. Thomas, and S. Pistorius, "Hybrid binary-real ga for microwave breast tomography," in *Antennas and Propagation Society International Symposium, 2008. AP-S 2008. IEEE*, 2008, pp. 1–4. 24

[45] E. S. . S. S. P. L. (EMSS-SA), www.feko.info. 27

[46] C. Guiffaut and K. Mahdjoubi, "A parallel fdtd algorithm using the mpi library," *Antennas and Propagation Magazine, IEEE*, vol. 43, no. 2, pp. 94–103, 2001. 27

[47] J. Hadamard, *Lectures on Cauchy's Problem: In Linear Partial Differential Equations*, ser. Dover phoenix editions. Dover Publications, 2003. [Online]. Available: http://books.google.com/books?id=B25O-x21uqkC 30

[48] C. S. Baird, "The uniqueness of maxwell's equations," university of Massachusetts, Lowell, http://faculty.uml.edu/cbaird/95.657(2012)/Maxwell_Uniqueness.pdf. 30

[49] N. Joachimowicz, J. Mallorqui, J.-C. Bolomey, and A. Broquets, "Convergence and stability assessment of newton-kantorovich reconstruction algorithms for microwave tomography," *Medical Imaging, IEEE Transactions on*, vol. 17, no. 4, pp. 562–570, 1998. 31

[50] B. Omrane, J. Laurin, and Y. Goussard, "Subwavelength-resolution microwave tomography using wire grid models and enhanced regularization techniques," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 54, no. 4, pp. 1438–1450, 2006. 31

[51] C. Pichot, J.-Y. Dauvignac, C. Dourthe, I. Aliferis, and E. Guillanton, "Inversion algorithms and measurement systems for microwave tomography of buried objects," in *Instrumentation and Measurement Technology Conference, 1999. IMTC/99. Proceedings of the 16th IEEE*, vol. 3, 1999, pp. 1570–1575 vol.3. 31

[52] M. Pastorino, "Stochastic optimization methods applied to microwave imaging: A review," *Antennas and Propagation, IEEE Transactions on*, vol. 55, no. 3, pp. 538–548, 2007. 32

[53] J. Vesterstrom and R. Thomsen, "A comparative study of differential," in *Congress on Evolutionary Computation 2004*, vol. 2, 2004, pp. 1980–1987. 32

[54] T. Huang and A. Mohan, "A hybrid boundary condition for robust particle swarm optimization," *Antennas and Wireless Propagation Letters, IEEE*, vol. 4, pp. 112–117, 2005. 32

[55] M. Donelli, G. Franceschini, A. Martini, and A. Massa, "An integrated multiscaling strategy based on a particle swarm algorithm for inverse scattering problems," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 44, no. 2, pp. 298–312, 2006. 33

[56] K. A. Michalski, "Electromagnetic imaging of circular-cylindrical onductors and tunnel using a differential evolution algorithm," *Microwave Optical Technology Letters*, vol. 26, 2000. 33

[57] "Github tao repository," https://github.com/travisdesell/tao. 33

[58] S. N. H. T.-Y. Y. K. z. Travis Desell, Michael Holman, "Tao: A toolkit for asynchronous optimization," unpublished so far. 34, 35

[59] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *IEEE International Conference on Neural Network*. 35

[60] T. Desell, "Asynchronous global optimization for massive-scale computing," Ph.D. dissertation, Rensselaer Polytechnic Institute, December 2009. 35, 36, 37, 38, 39, 48, 61, 62

[61] Y. Shi and R. C. Eberhart, "A modified particle swarm optimizer," in *IEEE World Congress on Computational Intelligence*, 1998, pp. 69–73. 36

[62] R. Storn and K. Price, "Minimizing the real functions of the icec'96 contest by differential evolution," in *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, 1996, pp. 842–844. 37

[63] T. Desell, D. Anderson, M. Magdon-Ismail, H. Newberg, B. Szymanski, and C. Varela, "An analysis of massively distributed evolutionary algorithms," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*, 2010, pp. 1–8. 39

[64] A. Merten, "Non-destructive test methods for hollow-core composite insulators," in *High Voltage Engineering and Application (ICHVE), 2010 International Conference on*, 2010, pp. 536–539. 55

6

Appendix: Source Code For FDTD Forward Solver

This source code can also be found on the code repository Github at

https://github.com/travisdesell/tomography.

6.1 FDTD_GPU.cu

```
//#define GLEW_STATIC
//#pragma comment(lib,"glew32.lib")
//#include <windows.h>
//#include <gl/glew.h>
//#include <glut.h>
#include <complex>
#include <stdio.h>
#include <iostream>
#include <cmath>
#include <stdlib.h>
#include <fstream>
#include <cstdlib>
#include <fstream>
#include <cuda.h>
//#include "stdafx.h"
#include <iomanip>
#include <time.h>
//#include <cuda_gl_interop.h>
#include <cuda_runtime.h>
//#include <cuComplex.h>
#include <vector>
#include <math_functions.h>
//#include "EasyBMP.h"
//#include "EasyBMP_DataStructures.h"
//#include "EasyBMP_VariousBMPutilities.h"

#include "FDTD_common.hxx"

void __cudaCheck(cudaError err, const char* file, const int line);
#define cudaCheck(err) __cudaCheck (err, __FILE__, __LINE__)

void __cudaCheckLastError(const char* errorMessage, const char* file, const int line);
```

```
#define cudaCheckLastError(msg) __cudaCheckLastError (msg, __FILE__, __LINE__)

void __cudaCheck(cudaError err, const char *file, const int line) {
    if (cudaSuccess != err) {
        fprintf(stderr, "%s(%i) : CUDA Runtime API error %d: %s.\n", file, line, (int)err, cudaGetErrorString( err ) );
        exit(-1);
    }
}

void __cudaCheckLastError(const char *errorMessage, const char *file, const int line) {
    cudaError_t err = cudaGetLastError();
    if (cudaSuccess != err) {
        fprintf(stderr, "%s(%i) : getLastCudaError() CUDA error : %s : (%d) %s.\n", file,
        line, errorMessage, (int)err, cudaGetErrorString( err ) );
        exit(-1);
    }
}

//#include <unistd.h>
//const cuComplex jcmpx (0.0, 1.0);
/*static void HandleError( cudaError_t err, const char *file,  int line ) {
  if (err != cudaSuccess) {
  printf( "%s in %s at line %d\n", cudaGetErrorString( err ),  file, line );
  exit( EXIT_FAILURE );
  }
  }*/

/*
__device__ int dgetCell(int x, int y, int size) {
    return x +y*size;
}

int getCell(int x, int y, int size) {
    return x +y*size;
}
*/


struct cuComplex {
    float   r;
    float   i;
    __host__  __device__ cuComplex( float a, float b ) : r(a), i(b)  {}
    __host__ __device__ cuComplex(float a): r(a), i(0) {}
    float magnitude2( void ) { return r * r + i * i; }
    __host__  __device__  cuComplex operator*(const cuComplex& a) {
        return cuComplex(r*a.r - i*a.i, i*a.r + r*a.i);
    }
    __host__ __device__ cuComplex operator*(const float& a){
        return cuComplex(r*a,i*a);
    }

    __host__  __device__  cuComplex operator+(const cuComplex& a) {
        return cuComplex(r+a.r, i+a.i);
    }
    __host__ __device__ cuComplex operator+(const float& a){
        return cuComplex(r+a,i);
    }
    __host__ __device__ void operator+=(const float& f){
        r += f;
    }
    __host__ __device__ void operator+=(const cuComplex& C);
    cuComplex();
};

__host__ __device__ cuComplex operator*(const float &f, const cuComplex &C)
{

    return cuComplex(C.r*f,C.i*f);
}

__host__ __device__ void cuComplex::operator+=(const cuComplex& C)
```

```
{
    r +=C.r;
    i += C.i;
}

__host__ __device__ float cuabs(cuComplex x)
{
    return sqrt(x.i*x.i + x.r*x.r);
}

__host__ __device__ cuComplex cuexp(cuComplex arg)
{
    cuComplex res(0,0);
    float s, c;
    float e = expf(arg.r);
    sincosf(arg.i,&s,&c);
    res.r = c * e;
    res.i = s * e;
    return res;

}

__device__ int isOnNF2FFBound(int x, int y)
{
    if(x==NF2FFdistfromboundary||x==nx-NF2FFdistfromboundary||y==NF2FFdistfromboundary||
    y==ny-NF2FFdistfromboundary)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

__device__ int getxfromthreadIdNF2FF(int index)
{
    int x=0;
    if(index<(nx-2*NF2FFdistfromboundary-2))//yn
    {
        x = index+NF2FFdistfromboundary+1;
    }
    else if(index<(nx-4*NF2FFdistfromboundary+ny-2))//xp
    {
        x = nx-NF2FFdistfromboundary-1;
    }
    else if(index<(2*nx-6*NF2FFdistfromboundary+ny-4))//yp
    {
        x = nx-NF2FFdistfromboundary  - (index-(nx-4*NF2FFdistfromboundary+ny-2))-2;
    }
    else if(index<(2*nx-8*NF2FFdistfromboundary+2*ny-4))
    //xn notice 2*nx-8*NF2FFdistfromboundary+2*ny-4 is the max index term.
    {
        x = NF2FFdistfromboundary;
    }
    return x;
}

__device__ int getyfromthreadIdNF2FF(int index)
{
    int y=0;
    if(index<(nx-2*NF2FFdistfromboundary-2))
    {
        y = NF2FFdistfromboundary;
    }
    else if(index<(nx-4*NF2FFdistfromboundary+ny-2))
    {
        y = (index-(nx-2*NF2FFdistfromboundary-2))+NF2FFdistfromboundary;
    }
    else if(index<(2*nx-6*NF2FFdistfromboundary+ny-4))
    {
```

```
        y = ny-NF2FFdistfromboundary-1;
    }
    else if(index<(2*nx-8*NF2FFdistfromboundary+2*ny-4))
    {
        y = ny-NF2FFdistfromboundary-(index-(2*nx-6*NF2FFdistfromboundary+ny-4))-1;
    }
    return y;
}
__device__ __host__ int isOnxn(int x)
{
    if(x==(NF2FFdistfromboundary))
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

__device__ __host__ int isOnxp(int x)
{
    if(x==(nx-NF2FFdistfromboundary-1))
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

__device__ __host__ int isOnyp(int x,int y)
{
    if(y==(ny-NF2FFdistfromboundary-1)&&!isOnxn(x)&&!isOnxp(x))
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

__device__ __host__ int isOnyn(int x, int y)
{
    if((y==(NF2FFdistfromboundary))&&!isOnxn(x)&&!(isOnxp(x)))
    {
        return 1;
    }
    else
    {
        return 0;
    }
}


__global__ void calculate_JandM(float* f,int* timestep,float*dev_Ez,float*dev_Hy,float*dev_Hx,
cuComplex *cjzxp,cuComplex *cjzyp,cuComplex*cjzxn,cuComplex*cjzyn,cuComplex*cmxyp,cuComplex*cmyxp,
cuComplex*cmxyn,cuComplex*cmyxn)
{
    float freq = *f;
    int index = threadIdx.x+blockIdx.x*blockDim.x;
    // should launch 2*nx-8*NF2FFdistfromboundary+2*ny-4 threads.
    if(index<=size_NF2FF_total)
    {
        const cuComplex j(0.0,1.0);
        int x = getxfromthreadIdNF2FF(index);
        int y = getyfromthreadIdNF2FF(index);

        float Ez;
```

```
        cuComplex pi(PI , 0);
        cuComplex two(2.0,0.0);
        cuComplex negativeone(-1.0,0);
        cuComplex deltatime(dt,0);

        if(isOnyp(x,y))
        {
            Ez = (dev_Ez[dgetCell(x,y+1,nx+1)]+dev_Ez[dgetCell(x,y,nx+1)])/2;
            float Hx = dev_Hx[dgetCell(x,y,nx)];
            cjzyp[index-(nx+ny-4*NF2FFdistfromboundary-2)] +=
             -1*Hx*deltatime*cuexp((float)(-1)*j*
             (float)2*pi*freq*(float)(*timestep)*deltatime);
             //cjzyp and cmxyp have nx - 2*NF2FFBoundary -2 elements

            cmxyp[index-(nx+ny-4*NF2FFdistfromboundary-2)] += -1*Ez*deltatime*
            cuexp((float)-1.0*j*(float)2.0*(float)PI*freq*
            ((float)(*timestep)+0.5)*(float)dt);
        }
        else if(isOnxp(x))//X faces override y faces at their intersections
        {
            Ez = (dev_Ez[dgetCell(x,y,nx+1)]+dev_Ez[dgetCell(x+1,y,nx+1)])/2;
            float Hy = dev_Hy[dgetCell(x,y,nx)];

            cjzxp[index-(nx-2*NF2FFdistfromboundary-2)] +=
             Hy*deltatime*cuexp(-1*j*2*pi*freq*(float)(*timestep)*(float)dt);
             //cjzxp and cmyxp have ny-2*NF2FFBound elements

            cmyxp[index-(nx-2*NF2FFdistfromboundary-2)] +=
             Ez*(float)dt*cuexp((float)(-1)*j*(float)2.0*pi*freq*
             ((float)(*timestep)+0.5)*(float)dt);
             // this is the discrete fourier transform, by the way.
        }
        else if(isOnyn(x,y))
        {
            Ez = (dev_Ez[dgetCell(x,y,nx+1)]+dev_Ez[dgetCell(x,y+1,nx+1)])/2;
            float Hx=dev_Hx[dgetCell(x,y,nx)];

            cjzyn[index] +=
             Hx*(float)dt*cuexp((float)(-1)*j*(float)2.0*(float)PI*
             freq*(float)(*timestep)*(float)dt);
               //cjzyn and cmxyn need to have nx-2*NF2FFbound-2 elements
            cmxyn[index] +=
            Ez*(float)dt*cuexp((float)(-1)*j*(float)2.0*(float)PI*freq*
            ((float)(*timestep)+0.5)*(float)dt);
        }
        else if(isOnxn(x))
        {
            Ez = (dev_Ez[dgetCell(x,y,nx+1)]+dev_Ez[dgetCell(x+1,y,nx+1)])/2;
            cjzxn[index-(2*nx+ny-6*NF2FFdistfromboundary-4)] +=
             -1*dev_Hy[dgetCell(x,y,nx)]*(float)dt*
            cuexp((float)(-1)*j*(float)2.0*(float)PI*freq*
            (float)(*timestep)*(float)dt);
             // cjzxn and cmyxn must have ny-2*NFdistfromboundary elements
            cmyxn[index-(2*nx+ny-6*NF2FFdistfromboundary-4)] +=
            -1*Ez*(float)dt*cuexp(-1.0*j*2.0*(float)PI*
            freq*((float)(*timestep)+0.5)*(float)dt);
        }
    }

}


__host__ __device__ float fwf(float timestep,float x, float y,float Phi_inc,float l)
{

    float ar;
    float ky, kx;//k hat
    sincosf(Phi_inc,&ky,&kx);

    ar = (float)timestep*dt-(float)t0-(1/(float)c0)*(ky*y*dx+kx*x*dy-l);
    //ar = timestep*dt-t0;
```

```
    //return exp(-1*(ar*ar)/(tau*tau));// gaussian pulse  argument is k dot r,
    return exp(-1*ar*ar/(tau*tau));
    //return sin(2*PI*1e9*timestep*dt);
}

__global__ void H_field_update(float*dev_Hy,float*dev_Hx,float*dev_Ez,float*dev_bmx,
float*dev_Psi_hyx,float*dev_amx,float*dev_bmy,float*dev_amy,float*dev_Psi_hxy,float*kex)
{
    float buffer_Hy;
    float buffer_Hx;
    float Chez = (dt/dx)/(mu0);
    int x = threadIdx.x +blockDim.x*blockIdx.x;
    int y = threadIdx.y + blockDim.y*blockIdx.y;

    if(x<nx&&y<nx)
    {
        buffer_Hy = dev_Hy[dgetCell(x,y,nx)]+
        Chez*(dev_Ez[dgetCell(x+1,y,nx+1)]-dev_Ez[dgetCell(x,y,nx+1)]);
        buffer_Hx = dev_Hx[dgetCell(x,y,nx)]-
        Chez*(dev_Ez[dgetCell(x,y+1,nx+1)]-dev_Ez[dgetCell(x,y,nx+1)]);
        if(x<ncells)
        {
          buffer_Hy= dev_Hy[dgetCell(x,y,nx)]+
          Chez*(dev_Ez[dgetCell(x+1,y,nx+1)]-dev_Ez[dgetCell(x,y,nx+1)])/kex[ncells-1-x];
          dev_Psi_hyx[dgetCell(x,y,20)]=dev_bmx[ncells-1-x]*dev_Psi_hyx[dgetCell(x,y,20)]+
          dev_amx[ncells-1-x]*(dev_Ez[dgetCell(x+1,y,nx+1)]-dev_Ez[dgetCell(x,y,nx+1)]);
          buffer_Hy+=Chez*dx*dev_Psi_hyx[dgetCell(x,y,20)] ;
        }
        if(x>=(nx-ncells))
        {
          buffer_Hy=dev_Hy[dgetCell(x,y,nx)]+Chez*(dev_Ez[dgetCell(x+1,y,nx+1)]-
          dev_Ez[dgetCell(x,y,nx+1)])/kex[x-nx+ncells];
          dev_Psi_hyx[dgetCell(x-nx+20,y,2*ncells)]=
          dev_bmx[x-nx+ncells]*dev_Psi_hyx[dgetCell(x-nx+20,y,20)]+
          dev_amx[x-nx+ncells]*(dev_Ez[dgetCell(x+1,y,nx+1)]-dev_Ez[dgetCell(x,y,nx+1)]);
          buffer_Hy+=Chez*dx*dev_Psi_hyx[dgetCell(x-nx+20,y,20)];
        }
        if(y<ncells)
        {
            buffer_Hx=dev_Hx[dgetCell(x,y,nx)]-Chez*(dev_Ez[dgetCell(x,y+1,nx+1)]-
            dev_Ez[dgetCell(x,y,nx+1)])/kex[ncells-1-y];
            dev_Psi_hxy[dgetCell(x,y,nx)]=dev_bmy[ncells-1-y]*dev_Psi_hxy[dgetCell(x,y,nx)]+
            dev_amy[ncells-1-y]*(dev_Ez[dgetCell(x,y+1,nx+1)]-dev_Ez[dgetCell(x,y,nx+1)]);
            buffer_Hx-=Chez*dy*dev_Psi_hxy[dgetCell(x,y,nx)];
        }
        if(y>=(ny-ncells))
        {
          buffer_Hx=dev_Hx[dgetCell(x,y,nx)]-Chez*(dev_Ez[dgetCell(x,y+1,nx+1)]-
          dev_Ez[dgetCell(x,y,nx+1)])/kex[y-ny+ncells];
          dev_Psi_hxy[dgetCell(x,y-ny+20,nx)]=
          dev_bmy[y-ny+ncells]*dev_Psi_hxy[dgetCell(x,y-ny+20,nx)]+
          dev_amy[y-ny+ncells]*(dev_Ez[dgetCell(x,y+1,nx+1)]-dev_Ez[dgetCell(x,y,nx+1)]);
          buffer_Hx-=Chez*dy*dev_Psi_hxy[dgetCell(x,y-nx+20,nx)];
        }
        //__syncthreads();
        if(isnan(buffer_Hx))
        {
            dev_Hx[dgetCell(x,y,nx)] = 0.0;
        }
        else
        {
            dev_Hx[dgetCell(x,y,nx)] = buffer_Hx;
        }

        if(isnan(buffer_Hy)) {
            dev_Hy[dgetCell(x,y,nx)] = 0.0;
        }
        else
        {
            dev_Hy[dgetCell(x,y,nx)] = buffer_Hy;
```

```
        }

        //dev_Hx[dgetCell(x,y,nx)] = buffer_Hx;
        //dev_Hy[dgetCell(x,y,nx)] = buffer_Hy;
    }
}


__global__ void E_field_update(int *i,float*dev_Ez,float*dev_Hy,float*dev_Hx,
float*dev_Psi_ezx,float*dev_aex,float*dev_aey,float*dev_bex,float*dev_bey,
float*dev_Psi_ezy,float*kex,float *Ceze,float*Cezhy,float*Cezhx,float*Cezeip,
float*Cezeic,float*Phi,float *dev_Jp,float *dev_Cezjp,float *dev_Beta_p)
{
    int x=threadIdx.x+blockDim.x*blockIdx.x;
    int y=threadIdx.y+blockDim.y*blockIdx.y;
    //  int offset = x+y*blockDim.x*gridDim.x;
    float buffer_Ez;
    //float Ceh = (dt/dx)/(eps0);
    float Cezj = -dt/eps0;
    float length_offset;
    if(x<=nx&&y<=ny)
    {

        //if(x==0||x==nx||y==0||y==ny)
        if(x==nx||y==ny||x==0||y==0)
        {
            buffer_Ez=0.0;
        }
        else
        {

            buffer_Ez = Ceze[dgetCell(x,y,nx+1)]*dev_Ez[dgetCell(x,y,nx+1)]+
            Cezhy[dgetCell(x,y,nx+1)]*(dev_Hy[dgetCell(x,y,nx)]-dev_Hy[dgetCell(x-1,y,nx)])
             -Cezhx[dgetCell(x,y,nx+1)]*(dev_Hx[dgetCell(x,y,nx)]-dev_Hx[dgetCell(x,y-1,nx)])
             +Cezeic[dgetCell(x,y,nx+1)]*fwf((float)(*i)+0.5,x-nx/2,y-ny/2,*Phi,-breast_radius)
             +Cezeip[dgetCell(x,y,nx+1)]*fwf((float)(*i)-0.5,x-nx/2,y-ny/2,*Phi,-breast_radius)
             -dev_Cezjp[dgetCell(x,y,nx+1)]*dev_Jp[dgetCell(x,y,nx+1)];

            dev_Jp[dgetCell(x,y,nx+1)] = Kp*dev_Jp[dgetCell(x,y,nx+1)] +
            (dev_Beta_p[dgetCell(x,y,nx+1)]/dt)*(buffer_Ez - dev_Ez[dgetCell(x,y,nx+1)]

             + fwf((float)(*i)+0.5,x-nx/2,y-ny/2,*Phi,-breast_radius)
             - fwf((float)(*i)-0.5,x-nx/2,y-ny/2,*Phi,-breast_radius));
             //buffer_Ez is Ezs(n+1) dev_Ez is Ezs(n)



           if(x<=ncells&&x!=0) //This is pml stuff.
            {
            buffer_Ez =  Ceze[dgetCell(x,y,nx+1)]*dev_Ez[dgetCell(x,y,nx+1)]+Cezhy[dgetCell(x,y,nx+1)]*
            (dev_Hy[dgetCell(x,y,nx)]-dev_Hy[dgetCell(x-1,y,nx)])/kex[ncells-x]
            -Cezhx[dgetCell(x,y,nx+1)]*
            (dev_Hx[dgetCell(x,y,nx)]-dev_Hx[dgetCell(x,y-1,nx)])/kex[ncells-x];
            dev_Psi_ezx[dgetCell(x-1,y-1,20)] = dev_bex[ncells-x]*dev_Psi_ezx[dgetCell(x-1,y-1,20)]+
            dev_aex[ncells-x]*(dev_Hy[dgetCell(x,y,nx)]-dev_Hy[dgetCell(x-1,y,nx)]);
            buffer_Ez += Cezhy[dgetCell(x,y,nx+1)]*dx*dev_Psi_ezx[dgetCell(x-1,y-1,2*ncells)];
            }
            if(x>=(nx-ncells)&&x!=nx)
            {
            buffer_Ez =  Ceze[dgetCell(x,y,nx+1)]*dev_Ez[dgetCell(x,y,nx+1)]
            +Cezhy[dgetCell(x,y,nx+1)]*(dev_Hy[dgetCell(x,y,nx)]-dev_Hy[dgetCell(x-1,y,nx)])/kex[x-nx+ncells]
            -Cezhx[dgetCell(x,y,nx+1)]*(dev_Hx[dgetCell(x,y,nx)]-dev_Hx[dgetCell(x,y-1,nx)])/kex[x-nx+ncells];
            dev_Psi_ezx[dgetCell(x-nx+20,y-1,20)]=
            dev_bex[x-nx+ncells]*dev_Psi_ezx[dgetCell(x-nx+20,y-1,20)]+
            dev_aex[x-nx+ncells]*(dev_Hy[dgetCell(x,y,nx)]-dev_Hy[dgetCell(x-1,y,nx)]);
            buffer_Ez+=Cezhy[dgetCell(x,y,nx+1)]*dx*dev_Psi_ezx[dgetCell(x-nx+20,y-1,2*ncells)];
            }
            if(y<=ncells&&y!=0)
            {
                buffer_Ez =  Ceze[dgetCell(x,y,nx+1)]*dev_Ez[dgetCell(x,y,nx+1)]+
                Cezhy[dgetCell(x,y,nx+1)]*(dev_Hy[dgetCell(x,y,nx)]-dev_Hy[dgetCell(x-1,y,nx)])/kex[ncells-y]
```

```
                -Cezhx[dgetCell(x,y,nx+1)]*(dev_Hx[dgetCell(x,y,nx)]-dev_Hx[dgetCell(x,y-1,nx)])/kex[ncells-y];
                dev_Psi_ezy[dgetCell(x-1,y-1,nx)]
                =dev_bey[(ncells-y)]*dev_Psi_ezy[dgetCell(x-1,y-1,nx)]+
                dev_aey[(ncells-y)]*(dev_Hx[dgetCell(x,y,nx)]-dev_Hx[dgetCell(x,y-1,nx)]);
                buffer_Ez-=Cezhx[dgetCell(x,y,nx+1)]*dy*dev_Psi_ezy[dgetCell(x-1,y-1,nx)];
            }
            if(y>=(ny-ncells)&&y!=ny)
            {
                buffer_Ez =  Ceze[dgetCell(x,y,nx+1)]*dev_Ez[dgetCell(x,y,nx+1)]+
                Cezhy[dgetCell(x,y,nx+1)]*(dev_Hy[dgetCell(x,y,nx)]-dev_Hy[dgetCell(x-1,y,nx)])/kex[y-ny+ncells]
                -Cezhx[dgetCell(x,y,nx+1)]*(dev_Hx[dgetCell(x,y,nx)]-dev_Hx[dgetCell(x,y-1,nx)])/kex[y-ny+ncells];
                dev_Psi_ezy[dgetCell(x-1,y-ny+20,nx)]=
                dev_bey[y-ny+ncells]*dev_Psi_ezy[dgetCell(x-1,y-ny+20,nx)]+
                dev_aey[y-ny+ncells]*(dev_Hx[dgetCell(x,y,nx)]-dev_Hx[dgetCell(x,y-1,nx)]);
                buffer_Ez-=Cezhx[dgetCell(x,y,nx+1)]*dy*dev_Psi_ezy[dgetCell(x-1,y-ny+20,nx)];
            }
        }
        // unsigned char green = 128+127*buffer_Ez/0.4;
        /*ptr[offset].x = 0;
          ptr[offset].y = green;
          ptr[offset].z = 0;
          ptr[offset].w = 255;*///OpenGL stuff

        //__syncthreads();
        if(isnan(buffer_Ez)) {
            dev_Ez[dgetCell(x,y,nx+1)] = 0.0;
        }
        else {
            dev_Ez[dgetCell(x,y,nx+1)] = buffer_Ez;
        }
        //dev_Ez[dgetCell(x,y,nx+1)] = buffer_Ez;
    }

}

__global__ void Field_reset(float* Ez, float* Hy, float* Hx, float* Psi_ezy,float* Psi_ezx,
float* Psi_hyx,float* Psi_hxy,cuComplex*cjzyn,cuComplex*cjzxp,
cuComplex*cjzyp,cuComplex*cjzxn,cuComplex*cmxyn,cuComplex*cmyxp,cuComplex*cmxyp,cuComplex*cmyxn,float *dev_Jp)
{
    int x = threadIdx.x + blockIdx.x*blockDim.x;
    int y = threadIdx.y+blockDim.y*blockIdx.y;
    int index = x + y*blockDim.x*gridDim.x;
    if(x<=ncells&&x!=0)
    {
        Psi_ezx[dgetCell(x-1,y-1,20)] =0;
    }
    if(x>=(nx-ncells)&&x!=nx)
    {
        Psi_ezx[dgetCell(x-nx+20,y-1,20)]=0;
    }
    if(y<=ncells&&y!=0)
    {
        Psi_ezy[dgetCell(x-1,y-1,nx)]=0;
    }
    if(y>=(ny-ncells)&&y!=ny)
    {
        Psi_ezy[dgetCell(x-1,y-ny+20,nx)]=0;
    }
    if(x<ncells)
    {

        Psi_hyx[dgetCell(x,y,20)]=0;
    }
    if(x>=(nx-ncells))
    {
        Psi_hyx[dgetCell(x-nx+20,y,2*ncells)]=0.0;
    }
    if(y<ncells)
    {
        Psi_hxy[dgetCell(x,y,nx)]=0.0;
    }
```

```
    if(y>=(ny-ncells))
    {
        Psi_hxy[dgetCell(x,y-ny+20,nx)]=0.0;
    }
    if(x<=nx&&y<=ny)
    {
        Ez[dgetCell(x,y,nx+1)] = 0.0;
        dev_Jp[dgetCell(x,y,nx+1)] = 0.0;
    }
    if(x<nx&&y<ny)
    {
        Hy[dgetCell(x,y,nx)] = 0.0;
        Hx[dgetCell(x,y,nx)] = 0.0;
    }

        if(index<size_cjzy)
        {
            cjzyp[index] = cuComplex(0,0);
            //cjzyp and cmxyp have nx - 2*NF2FFBoundary -2 elements
            cjzyn[index] = cuComplex(0,0);
            cmxyp[index] = cuComplex(0,0);
            cmxyn[index] = cuComplex(0,0);
        }
        if(index<size_cjzx)
        {
            cjzxp[index] = cuComplex(0,0);
            cjzxn[index] = cuComplex(0,0);
            cmyxp[index] = cuComplex(0,0);
            cmyxn[index] = cuComplex(0,0);
        }



}

float calc_radiated_power(cuComplex *cjzxp,cuComplex *cjzyp,cuComplex *cjzxn,
cuComplex *cjzyn,cuComplex *cmxyp,cuComplex *cmyxp,cuComplex *cmxyn,cuComplex *cmyxn)
{
    int indexofleg1 = nx-2*NF2FFdistfromboundary-2;
    int indexofleg2 = nx+ny-4*NF2FFdistfromboundary-2;
    int indexofleg3 = 2*nx+ny-6*NF2FFdistfromboundary-4;
    int maxindex = 2*nx-8*NF2FFdistfromboundary+2*ny-4;
    int index;
    cuComplex cjz(0,0);
    cuComplex power = 0;

    for(index = 0; index<indexofleg1;index++)
    {   cjz = cuComplex(cjzyn[index].r,-1.0*cjzyn[index].i);//conjugation
        //z x x = y dot -y = -1
        power+=-1.0*cjz*cmxyn[index]*dx;
        // the negative one comes from the dot product between JxM and the n hat vector
    }
    for(index = indexofleg1; index<indexofleg2;index++)
    {
        cjz = cuComplex(cjzxp[index-indexofleg1].r,-1.0*cjzxp[index-indexofleg1].i);
        //making the conjugate
        // z cross y = -x dot x = -1
        power+= -1.0*cjz*cmyxp[index-indexofleg1]*dy;//positive x unit normal vector
    }
    for(index = indexofleg2;index<indexofleg3;index++)
    {
        // z cross x = y dot y = 1
        cjz = cuComplex(cjzyp[index-indexofleg2].r,-1.0*cjzyp[index-indexofleg2].i);
        power+= cjz*cmxyp[index-indexofleg2]*dx;//postive y unit normal vector
    }
    for(index = indexofleg3;index<maxindex;index++)
    {
        // z cross y = -x dot -x = 1
        cjz = cuComplex(cjzxn[index-indexofleg3].r,-1.0*cjzxn[index-indexofleg3].i);
        power += cjz*cmyxn[index-indexofleg3]*dy;// negative x hat n vector
```

```
    }
    float realpower = power.r;
    realpower *= 0.5;
    return realpower;
}


__host__ __device__ int getOptimizationCell(int x, int y)
{
    int x_coord,y_coord;
    x_coord = (x-(nx/2-(int)(breast_radius/dx)))/(2*breast_radius/(numpatches*dx));
    y_coord = (y-(ny/2-breast_radius/dy))/(2*breast_radius/(numpatches*dy));
    return x_coord+numpatches*y_coord;//The max return should be, numpatches*numpatches-1, hopefully.
}

void N2FPostProcess (float* D,float* P,float f,cuComplex *cjzxp,

cuComplex *cjzyp,cuComplex *cjzxn,cuComplex *cjzyn,cuComplex *cmxyp,

cuComplex *cmyxp,cuComplex *cmxyn,cuComplex *cmyxn)
{
    int indexofleg1 = nx-2*NF2FFdistfromboundary-2;
    int indexofleg2 = nx+ny-4*NF2FFdistfromboundary-2;
    int indexofleg3 = 2*nx+ny-6*NF2FFdistfromboundary-4;
    int maxindex = 2*nx-8*NF2FFdistfromboundary+2*ny-4;
    int x,y;
cuComplex L(0,0);
cuComplex N(0,0);
    float rhoprime;
    float Psi;
    cuComplex E(0,0);
    int Phi_index;
    cuComplex  Mphi(0,0);
    float Phi;
    float k = 2*PI*f/c0;
    cuComplex  negativeone(-1.0,0.0);
    int index = 0;
    cuComplex jcmpx(0,1);
    //float Prad = calc_radiated_power(cjzxp,cjzyp,cjzxn
    ,cjzyn,cmxyp,cmyxp,cmxyn,cmyxn);
    float Prad = calc_incident_power(f);
    //std::cout<<"Prad = "<<Prad<<std::endl;
    float flx, fly;
for(Phi_index = 0; Phi_index<numberofobservationangles;Phi_index++)
{
    Phi = 2*PI/numberofobservationangles*(float)Phi_index;
    for(index = 0;index<indexofleg1;index++)
    {

        x = CPUgetxfromthreadIdNF2FF(index);
        y = CPUgetyfromthreadIdNF2FF(index);
        flx = (float)x;//float x
        fly = (float)y + 0.5;
        rhoprime = sqrt(pow((dx*((-1.0*(float)nx/2)+1+flx)),2)+
        pow((dy*(-1.0*(float)ny/2+1+fly)),2));
        Psi = atan2(-1*((float)ny/2)+1+fly,-1*((float)nx/2)+1+flx)-Phi;
        N+=-1.0*cjzyn[index]*cuexp(1.0*jcmpx*k*rhoprime*cos(Psi))*dx;
        L+=-1.0*sin(Phi)*cuexp(1.0*jcmpx*k*rhoprime*cos(Psi))*cmxyn[index]*dx;

    }
    for(index = indexofleg1;index<indexofleg2;index++)
    {

        x = CPUgetxfromthreadIdNF2FF(index);
        y = CPUgetyfromthreadIdNF2FF(index);
        flx = (float)x+0.5;
        fly = (float)y;
        rhoprime = sqrt(pow((dx*(((float)nx/2)-1-flx)),2)+
        pow((dy*(((float)ny/2)-1-fly)),2));
        Psi = atan2(-1*((float)ny/2)+1+fly,(-1*((float)nx/2)+1+flx))
        -Phi;
```

```
        N+=-1.0*cjzxp[index-indexofleg1]
        *cuexp(1.0*jcmpx*k*rhoprime*cos(Psi))*dy;
        L+=cos(Phi)*cmyxp[index-indexofleg1]
        *cuexp(1.0*jcmpx*k*rhoprime*cos(Psi))*dy;
        //L_phi = -Lxsin(phi)+Lycos(Phi) here we only have Ly
    }
    for(index=indexofleg2;index<indexofleg3;index++)
    {
        x = CPUgetxfromthreadIdNF2FF(index);
        y = CPUgetyfromthreadIdNF2FF(index);
        flx = (float)x;
        fly = (float)y + 0.5;
        rhoprime = sqrt(pow((dx*(((float)nx/2)-1-flx)),2)
        +pow((dy*(((float)ny/2)-1-fly)),2));
        Psi = atan2((-1*(float)ny/2+1+fly),(-1*((float)nx/2)+1+flx))
        -Phi;
        N+=-1.0*cjzyp[index-indexofleg2]*cuexp(jcmpx*k*rhoprime*cos(Psi))*dx;
        L+=-1.0*sin(Phi)*cmxyp[index-indexofleg2]*cuexp(jcmpx*k*rhoprime*cos(Psi))*dx;
    }
    for(index = indexofleg3;index<maxindex;index++)
    {
        x = CPUgetxfromthreadIdNF2FF(index);
        y = CPUgetyfromthreadIdNF2FF(index);
        flx = (float)x+0.5;
        fly = (float)y;
        rhoprime =
        sqrt(pow(dx*(((float)nx/2)-1-flx),2)+pow((dy*(((float)ny/2)-1-fly)),2));
        Psi = atan2(-1*((float)ny/2)+1+fly,-1*(float)nx/2+1+flx)-Phi;
        N+=-1.0*cjzxn[index-indexofleg3]*cuexp(jcmpx*k*rhoprime*cos(Psi))*dy;
        L+= cos(Phi)*cmyxn[index-indexofleg3]*cuexp(jcmpx*k*rhoprime*cos(Psi))*dy;
    }
    D[Phi_index] =
    (k*k*cuabs(L+(float)eta0*N)*cuabs(L+(float)eta0*N)
    /((float)8*(float)PI*(float)eta0*Prad*33.329));
    E = L+(float)eta0*N;
    P[Phi_index] = atan2(E.i,E.r);
    L = cuComplex(0,0);
    N = cuComplex(0,0);

}

}

__host__ __device__ float del_X(float del_eps, int k)
{
return del_eps*exp(-1*(float)k*dt/relaxation_time)
*(1-exp(-dt/relaxation_time))
*(1-exp(-dt/relaxation_time));

}
//static void draw_func(void){
// glDrawPixels(nx,ny,GL_RGBA,GL_UNSIGNED_BYTE,0);
// glutSwapBuffers;
//}

using namespace std;

__global__ void scattered_parameter_init(float*eps_infinity,
float *sigma_e_z,float*Cezeic,float*Cezeip,float *Cezjp,
 float* dev_Beta_p);

double FDTD_GPU(const vector<double> &arguments) {
//    cout << "calculating FDTD GPU" << endl;

//    cudaSetDevice(0);

vector<float> image;
//This is setting the material parameters of the optimization cells.
for (int lerp = 0; lerp < numpatches*numpatches; lerp++) {
//numpatches is the amount of optimization patches across.
image.push_back((float)arguments.at(lerp));
```

```
/* if(lerp == numpatches/2){
image.push_back(6.75);
  }
else
{
image.push_back(3.14);  //eps_infinity of fat
}*/
}

for (int lerp = numpatches*numpatches; lerp < numpatches*numpatches* 2; lerp++) {
image.push_back((float)arguments.at(lerp));

/* if(lerp == numpatches/2 + numpatches*numpatches){
image.push_back(40);// del_eps

}
else
{
image.push_back(1.61);
}*/

}

for (int lerp = numpatches*numpatches*2 ; lerp<numpatches*numpatches*3;lerp++){
image.push_back((float)arguments.at(lerp));

/*   if(lerp == numpatches/2 + numpatches*numpatches){
image.push_back(0);// sigma

}
else
{
image.push_back(0);
}*/
}



cudaError_t error;
float freq;
int grid_x = int(ceil((float)nx / 22));
int grid_y = int(ceil((float)ny / 22));

dim3 grid(grid_x, grid_y);
dim3 block(22, 22);
float *Ez = (float*)malloc(sizeof(float)*(1+nx)*(1+ny));
float *eps_infinity = (float*)malloc(sizeof(float)*(1+nx)*(1+ny));
float *Cezhy = (float*)malloc(sizeof(float)*(1+nx)*(1+ny));
float *Ceze = (float*)malloc(sizeof(float)*(1+nx)*(1+ny));
//Cezj later if using loop current source
//float *Cezj = (float*)malloc(sizeof(float)*(1+nx)*(1+ny));
// if using loop current source
float *del_eps = (float*)malloc(sizeof(float)*(1+nx)*(1+ny));
float *Beta_p = (float*)malloc(sizeof(float)*(1+nx)*(1+ny));
float radius;//tumor_radius,tumor_radius_2,tumor_radius_3;
float *sigma_e_z = (float*)malloc(sizeof(float)*(1+nx)*(1+ny));


/*******************  Setting Material Parameters************************/

for (int j = 0; j < ny + 1; j++) {
for (int i = 0; i < nx + 1; i++) {
    Ez[getCell(i,j,nx+1)] = (float)0;
    eps_infinity[getCell(i,j,nx+1)] = 1;
    del_eps[getCell(i,j,nx+1)] = 0;
    sigma_e_z[getCell(i,j,nx+1)] = 0;
    radius = sqrt(pow( ((float)i-nx/2)*dx,2) + pow( ((float)j-ny/2)*dy,2));

    //tumor_radius = sqrt(pow( ((float)i - target_x)*dx,2) + pow( ((float)j-target_y)*dy,2));
    if (radius <= breast_radius) {
```

```
        eps_infinity[getCell(i,j,nx+1)] = (float)image.at(getOptimizationCell(i,j));
        //This is the line that should be uncommented if using as forward solver
        del_eps[getCell(i,j,nx+1)] =
         (float)image.at(getOptimizationCell(i,j)+numpatches*numpatches);
        sigma_e_z[getCell(i,j,nx+1)] =
        (float)image.at(getOptimizationCell(i,j)+numpatches*numpatches*2);

        //eps_infinity[getCell(i,j,nx+1)] = 10;
        //if(tumor_radius <= tumor_nx+1)//delete this if using as forward solver
        //{
        // eps_infinity[getCell(i,j,nx+1)] = 60;
        //}
    }
    Beta_p[getCell(i,j,nx+1)] =
    eps0*del_eps[getCell(i,j,nx+1)]*dt
    /(relaxation_time*(1+dt/(2*relaxation_time)));
    Cezhy[getCell(i,j,nx+1)] =
    (2*dt/dx)/(2*eps_infinity[getCell(i,j,nx+1)]*eps0
    + sigma_e_z[getCell(i,j,nx+1)]*dt + Beta_p[getCell(i,j,nx+1)]);
    Ceze[getCell(i,j,nx+1)] =
    (2*eps_infinity[getCell(i,j,nx+1)]*eps0 - sigma_e_z[getCell(i,j,nx+1)]*dt
    + Beta_p[getCell(i,j,nx+1)])
    /(2*eps_infinity[getCell(i,j,nx+1)]*eps0 + sigma_e_z[getCell(i,j,nx+1)]*dt
     + Beta_p[getCell(i,j,nx+1)]);
}
}

/*****************************************************************/

/*********************************Setting up PML layer ************************/
float *sigma_e_pml = (float*)malloc(sizeof(float)*ncells);
float *sigma_m_pml = (float*)malloc(sizeof(float)*ncells);

//initialize
float sigma_max = (npml+1)/(150*PI*dx);
float rho;
for (int i = 0; i < ncells; i++) {
rho = ((float)i+0.25)/ncells;
sigma_e_pml[i] = sigma_max*sigma_factor*pow(rho,npml);

rho = ((float)i+0.75)/ncells;
sigma_m_pml[i] = (mu0/eps0)*sigma_max*sigma_factor*pow(rho,npml);

/*
cout<<"sigma_e_pml = "<<sigma_e_pml[i]<<endl;
cout<<"sigma_m_pml "<<sigma_m_pml[i]<<endl;
*/
}

float *kex = (float*)malloc(sizeof(float)*ncells);
float *kmx = (float*)malloc(sizeof(float)*ncells);
float *aex = (float*)malloc(sizeof(float)*ncells);
float *bex = (float*)malloc(sizeof(float)*ncells);
float *amx = (float*)malloc(sizeof(float)*ncells);
float *bmx = (float*)malloc(sizeof(float)*ncells);
float *alpha_e = (float*)malloc(sizeof(float)*ncells);
float *alpha_m = (float*)malloc(sizeof(float)*ncells);


//Initialize kex and kmx (formerly k_e_init and k_m_init)
//And alpha_e and alpha_m, and aex, bex, kex, amx, bmx, kmx
for (int i = 0; i < ncells; i++) {
rho = ((float)i+0.25)/ncells;
kex[i]=pow(rho,npml)*(kmax-1)+1;
alpha_e[i]=alpha_min+(alpha_max-alpha_min)*rho;

rho = ((float)i+0.75)/ncells;
kmx[i]=pow(rho,npml)*(kmax-1)+1;
alpha_m[i]=(mu0/eps0)*(alpha_min+(alpha_max-alpha_min)*rho);

bex[i]=exp(-1*(dt/eps0)*(sigma_e_pml[i]/kex[i]+alpha_e[i]));
```

```
aex[i]=((bex[i]-1)*sigma_e_pml[i])
/(dx*(sigma_e_pml[i]*kex[i]+alpha_e[i]*kex[i]*kex[i]));

float argument = -1*(dt/mu0)*((sigma_m_pml[i]/kmx[i])+alpha_m[i]);
bmx[i]=exp(argument);
amx[i]=(bmx[i]-1)*sigma_m_pml[i]
/(dx*(sigma_m_pml[i]*kmx[i]+alpha_m[i]*kmx[i]*kmx[i]));

/*
cout<<"kex["<<i<<"]= "<<kex[i]<<endl;
cout<<"kmx["<<i<<"]= "<<kmx[i]<<endl;
cout<<"aex["<<i<<"]= "<<aex[i]<<endl;
cout<<"amx["<<i<<"]= "<<amx[i]<<endl;
cout<<"bex["<<i<<"]= "<<bex[i]<<endl;
cout<<"bmx["<<i<<"]= "<<bmx[i]<<endl;
cout<<"alpha_e = "<<alpha_e[i]<<endl;
cout<<"alpha_m = "<<alpha_m[i]<<endl;
cout << endl;
*/
}

float *Psi_ezy = (float*)malloc(sizeof(float)*ny*20);
float *Psi_ezx = (float*)malloc(sizeof(float)*nx*20);
float *Psi_hyx = (float*)malloc(sizeof(float)*ny*20);
float *Psi_hxy = (float*)malloc(sizeof(float)*nx*20);

/*
for (int i = 0; i < nx * 20; i++) {
Psi_ezy[i] = 0.0;
Psi_hxy[i] = 0.0;
}

for (int i = 0; i< ny * 20; i++) {
Psi_ezx[i] = 0.0;
Psi_hyx[i] = 0.0;
}
*/



/*********************************************************************/

float *D =
(float*)malloc(sizeof(float)*numberofexcitationangles*
numberofobservationangles*numberoffrequencies)
float *P =
 (float*)malloc(sizeof(float)*numberofexcitationangles*
 numberofobservationangles*numberoffrequencies);

float *Hy = (float*)malloc(sizeof(float)*nx*ny);
float *Hx = (float*)malloc(sizeof(float)*nx*ny);

//This are output values from the device

cuComplex *cjzxp, *cjzyp, *cjzxn, *cjzyn, *cmxyp, *cmyxp, *cmxyn, *cmyxn;
cuComplex *hcjzyp = (cuComplex*)malloc(sizeof(cuComplex)*size_cjzy);
cuComplex *hcjzyn = (cuComplex*)malloc(sizeof(cuComplex)*size_cjzy);
cuComplex *hcjzxp = (cuComplex*)malloc(sizeof(cuComplex)*size_cjzx);
cuComplex *hcjzxn = (cuComplex*)malloc(sizeof(cuComplex)*size_cjzx);
cuComplex *hcmxyn = (cuComplex*)malloc(sizeof(cuComplex)*size_cjzy);
cuComplex *hcmxyp = (cuComplex*)malloc(sizeof(cuComplex)*size_cjzy);
cuComplex *hcmyxp = (cuComplex*)malloc(sizeof(cuComplex)*size_cjzx);
cuComplex *hcmyxn = (cuComplex*)malloc(sizeof(cuComplex)*size_cjzx);

float *dev_freq, *dev_Phi;
float *dev_Ceze,*dev_Cezhy, *dev_bex, *dev_aex,
 *dev_bmx, *dev_amx, *dev_kex, *dev_kmx;
//dev_Cezj if using loop current source
float *dev_Ez, *dev_Hy, *dev_Hx;

float *dev_Psi_ezy, *dev_Psi_ezx, *dev_Psi_hyx, *dev_Psi_hxy;
```

```
float *dev_Cezeic, *dev_Cezeip;
float  *dev_eps_infinity,*dev_sigma_e_z;
float *dev_Beta_p;
float *dev_Cezjp,*dev_Jp;
//dev_Jp is the polarization current term used in the Debye scattering (FD)2TD

cudaCheck( cudaMalloc(&dev_sigma_e_z,sizeof(float)*(nx+1)*(ny+1)) );
cudaCheck( cudaMalloc(&dev_eps_infinity,sizeof(float)*(nx+1)*(ny+1)) );
cudaCheck( cudaMalloc(&dev_Cezeic,sizeof(float)*(nx+1)*(ny+1)) );
cudaCheck( cudaMalloc(&dev_Cezeip,sizeof(float)*(nx+1)*(ny+1)) );
cudaCheck( cudaMalloc(&dev_Beta_p,sizeof(float)*(nx+1)*(ny+1)) );
cudaCheck( cudaMalloc(&dev_Cezjp,sizeof(float)*(nx+1)*(ny+1)) );
cudaCheck( cudaMemcpy(dev_eps_infinity,eps_infinity,sizeof(float)*(nx+1)*(ny+1),cudaMemcpyHostToDevice) );
cudaCheck( cudaMemcpy(dev_sigma_e_z,sigma_e_z,sizeof(float)*(nx+1)*(ny+1),cudaMemcpyHostToDevice) );

cudaCheck( cudaMemcpy(dev_Beta_p,Beta_p,sizeof(float)*(nx+1)*(ny+1),cudaMemcpyHostToDevice) );

scattered_parameter_init<<<grid,block>>>(dev_eps_infinity,dev_sigma_e_z,dev_Cezeic,dev_Cezeip,dev_Cezjp,dev_Beta_p);
cudaCheckLastError("scattered_parameter_init kernel failed");

//float *Cezeic = (float*)malloc((sizeof(float))*(nx+1)*(ny+1));
// float *Cezeip = (float*)malloc((sizeof(float))*(nx+1)*(ny+1));
//cudaMemcpy(Cezeic,dev_Cezeic,sizeof(float)*(nx+1)*(ny+1),cudaMemcpyDeviceToHost);
//cudaMemcpy(Cezeip,dev_Cezeip,sizeof(float)*(nx+1)*(ny+1),cudaMemcpyDeviceToHost);

cudaCheck(cudaMalloc(&dev_Phi,sizeof(float)));
cudaCheck(cudaMalloc(&dev_kex,sizeof(float)*10));
cudaCheck(cudaMalloc(&dev_kmx,sizeof(float)*10));
cudaCheck(cudaMalloc(&dev_Ez,sizeof(float)*(nx+1)*(ny+1)));
cudaCheck(cudaMalloc(&dev_Hy,sizeof(float)*nx*ny));
cudaCheck(cudaMalloc(&dev_Jp,sizeof(float)*(1+nx)*(1+ny)));
cudaCheck(cudaMalloc(&dev_freq ,sizeof(float)));
cudaCheck(cudaMalloc(&dev_Hx,sizeof(float)*nx*ny));
cudaCheck(cudaMalloc(&dev_Psi_ezy,sizeof(float)*20*(nx+1)));
cudaCheck(cudaMalloc(&dev_Psi_ezx,sizeof(float)*20*(ny+1)));
cudaCheck(cudaMalloc(&dev_Psi_hyx,sizeof(float)*20*(ny)));
cudaCheck(cudaMalloc(&dev_Psi_hxy,sizeof(float)*20*(nx)));

cudaCheck(cudaMalloc(&cjzxp,sizeof(cuComplex)*size_cjzx));
cudaCheck(cudaMalloc(&cjzyp,sizeof(cuComplex)*size_cjzy));
cudaCheck(cudaMalloc(&cjzxn,sizeof(cuComplex)*size_cjzx));
cudaCheck(cudaMalloc(&cjzyn,sizeof(cuComplex)*size_cjzy));
cudaCheck(cudaMalloc(&cmxyp,sizeof(cuComplex)*size_cjzy));
cudaCheck(cudaMalloc(&cmxyn,sizeof(cuComplex)*size_cjzy));
cudaCheck(cudaMalloc(&cmyxp,sizeof(cuComplex)*size_cjzx));
cudaCheck(cudaMalloc(&cmyxn,sizeof(cuComplex)*size_cjzx));

cudaCheck(cudaMemcpy(dev_freq,&freq,sizeof(float),cudaMemcpyHostToDevice));

cudaCheck(cudaMalloc(&dev_bex,sizeof(float)*10));
cudaCheck(cudaMalloc(&dev_bmx,sizeof(float)*10));
cudaCheck(cudaMalloc(&dev_amx,sizeof(float)*10));
cudaCheck(cudaMalloc(&dev_aex,sizeof(float)*10));
cudaCheck(cudaMalloc(&dev_Cezhy,sizeof(float)*(nx+1)*(ny+1)));
cudaCheck(cudaMalloc(&dev_Ceze,sizeof(float)*(nx+1)*(ny+1)));

cudaCheck(cudaMemcpy(dev_amx,amx,sizeof(float)*10,cudaMemcpyHostToDevice));
cudaCheck(cudaMemcpy(dev_kex,kex,sizeof(float)*10,cudaMemcpyHostToDevice));
cudaCheck(cudaMemcpy(dev_kmx,kmx,sizeof(float)*10,cudaMemcpyHostToDevice));
cudaCheck(cudaMemcpy(dev_aex,aex,sizeof(float)*10,cudaMemcpyHostToDevice));
cudaCheck(cudaMemcpy(dev_bex,bex,sizeof(float)*10,cudaMemcpyHostToDevice));
cudaCheck(cudaMemcpy(dev_bmx,bmx,sizeof(float)*10,cudaMemcpyHostToDevice));
cudaCheck(cudaMemcpy(dev_amx,amx,sizeof(float)*10,cudaMemcpyHostToDevice));

//cudaMalloc(&dev_Cezj,sizeof(float)*(nx+1)*(ny+1)); if using current source

Field_reset<<<grid,block>>>(dev_Ez, dev_Hy, dev_Hx, dev_Psi_ezy, dev_Psi_ezx,
 dev_Psi_hyx, dev_Psi_hxy,cjzyn,
 cjzxp,cjzyp,cjzxn,cmxyn,cmyxp,cmxyp,cmyxn,dev_Jp);
cudaCheckLastError("Field_reset kernel failed");
```

```
//Field_reset is also good for making all these values zero.


cudaCheck(cudaMemcpy(dev_Cezhy,Cezhy,sizeof(float)*(nx+1)*(ny+1),cudaMemcpyHostToDevice));
cudaCheck(cudaMemcpy(dev_Ceze,Ceze,sizeof(float)*(nx+1)*(ny+1),cudaMemcpyHostToDevice));

int *dev_i;
cudaCheck( cudaMalloc(&dev_i,sizeof(int)) );
float test_Ez;

dim3 gridNF2FF((int)ceil(size_NF2FF_total/512.0));
dim3 blockNF2FF(512);

float test_Ez_2;
float Phi;
// ofstream measurement_data;
// measurement_data.open("Measurement_Phase_tumor.txt");
for(int Phi_index = 0; Phi_index < numberofexcitationangles; Phi_index++) {

Phi = Phi_index*2*PI/numberofexcitationangles;
cudaCheck( cudaMemcpy(dev_Phi,&Phi,sizeof(float),cudaMemcpyHostToDevice) );

for (int i = 0; i < number_of_time_steps; i++) {
cudaCheck( cudaMemcpy(dev_i,&i,sizeof(int),cudaMemcpyHostToDevice) );

H_field_update<<<grid,block>>>(dev_Hy,dev_Hx,dev_Ez,dev_bmx,dev_Psi_hyx
,dev_amx,dev_bmx,dev_amx,dev_Psi_hxy,dev_kmx);
cudaCheckLastError("H_field_updated kernel failed");
E_field_update<<<grid,block>>>(dev_i,dev_Ez,dev_Hy,dev_Hx,dev_Psi_ezx,dev_aex,dev_aex,
dev_bex,dev_bex,dev_Psi_ezy,dev_kex,dev_Ceze,dev_Cezhy,dev_Cezhy,dev_Cezeip
,dev_Cezeic,dev_Phi,dev_Jp,dev_Cezjp,dev_Beta_p);
cudaCheckLastError("E_field_updated kernel failed");
for(int freq_index = 0;freq_index<numberoffrequencies;freq_index++)
{
freq = startfrequency + deltafreq*freq_index;
cudaCheck( cudaMemcpy(dev_freq,&freq,sizeof(float),cudaMemcpyHostToDevice) );
calculate_JandM<<<gridNF2FF,blockNF2FF>>>(dev_freq, dev_i,dev_Ez,
dev_Hy,dev_Hx, cjzxp+size_x_side*freq_index, cjzyp+size_y_side*freq_index,
 cjzxn+size_x_side*freq_index, cjzyn+size_y_side*freq_index,
  cmxyp+size_y_side*freq_index, cmyxp+size_x_side*freq_index,
   cmxyn+size_y_side*freq_index, cmyxn+size_x_side*freq_index);

cudaCheckLastError("calculate_JandM kernel failed"  );
}

}

cudaCheck( cudaMemcpy(hcjzyn,cjzyn,sizeof(cuComplex)*size_cjzy,cudaMemcpyDeviceToHost));
cudaCheck( cudaMemcpy(hcjzxp,cjzxp,sizeof(cuComplex)*size_cjzx,cudaMemcpyDeviceToHost));
cudaCheck( cudaMemcpy(hcjzyp,cjzyp,sizeof(cuComplex)*size_cjzy,cudaMemcpyDeviceToHost));
cudaCheck( cudaMemcpy(hcjzxn,cjzxn,sizeof(cuComplex)*size_cjzx,cudaMemcpyDeviceToHost));
cudaCheck( cudaMemcpy(hcmxyn,cmxyn,sizeof(cuComplex)*size_cjzy,cudaMemcpyDeviceToHost));
cudaCheck( cudaMemcpy(hcmyxp,cmyxp,sizeof(cuComplex)*size_cjzx,cudaMemcpyDeviceToHost));
cudaCheck( cudaMemcpy(hcmxyp,cmxyp,sizeof(cuComplex)*size_cjzy,cudaMemcpyDeviceToHost));
cudaCheck( cudaMemcpy(hcmyxn,cmyxn,sizeof(cuComplex)*size_cjzx,cudaMemcpyDeviceToHost));



for (int freq_index = 0; freq_index < numberoffrequencies; freq_index++)
{
freq = startfrequency + deltafreq*freq_index;

N2FPostProcess(D + Phi_index*numberofobservationangles*numberoffrequencies+freq_index*numberofobservationangles,
 P + Phi_index*numberofobservationangles*numberoffrequencies+freq_index*numberofobservationangles,
  freq, hcjzxp+size_x_side*freq_index , hcjzyp+size_y_side*freq_index, hcjzxn+size_x_side*freq_index ,
   hcjzyn+size_y_side*freq_index, hcmxyp+size_y_side*freq_index, hcmyxp+size_x_side*freq_index,
    hcmxyn+size_y_side*freq_index, hcmyxn+size_x_side*freq_index);

}
//D is a 3-dimensional array. The x axis is observation angles, z axis is Excitation angles, y axis is frequencies.
//each N2FPostProcess  Fills D(:,freq_index,Phi_index) where ":" is, as per matlab notation, all the elements of the x row.
```

```
Field_reset<<<grid,block>>>(dev_Ez, dev_Hy, dev_Hx,
 dev_Psi_ezy, dev_Psi_ezx, dev_Psi_hyx, dev_Psi_hxy,cjzyn,
 cjzxp,cjzyp,cjzxn,cmxyn,cmyxp,cmxyp,cmyxn,dev_Jp);
cudaCheckLastError("Field_reset kernel failed");
}


// for(int index = 0;index < numberofobservationangles * numberoffrequencies * numberofexcitationangles ; index++)
// {
//          measurement_data<<D[index]<<" , ";
// }
// for(int index = 0;index < numberofobservationangles * numberoffrequencies * numberofexcitationangles ; index++)
// {
//           measurement_data<<P[index]<<" , ";
// }
float measurement[] = {/*insert measurement data created by same forward solver here*/};

/*
for (int i = 0; i < numberofexcitationangles*numberofobservationangles; i++) {
cout << "D[" << i << " ]: " << D[i] << endl;
}
*/

float del_Phi = 0;
float fit = 0;
for(int i = 0; i < numberofobservationangles*numberofexcitationangles*numberoffrequencies; i++)
{
fit -= pow(D[i]-measurement[i],2)/(numberofexcitationangles*numberoffrequencies);
del_Phi = P[i]-measurement[i+numberofobservationangles*numberofexcitationangles*numberoffrequencies];
if(del_Phi>PI)
{
    del_Phi -= 2*PI;
}
else if(del_Phi<-1*PI)
{
    del_Phi += 2*PI;
}
fit -= del_Phi*del_Phi/(10*PI*PI*numberofexcitationangles*numberoffrequencies);

// if(abs(D[index]-measurement[index])>0.01)
// cout<<index<<" D = "<<D[index]<<" measurement = "<<measurement[index]<<endl;
}


error = cudaGetLastError();
free(Ceze);
free(Cezhy);
free(Ez);
free(eps_infinity);
free(del_eps);
free(sigma_e_z);
free(Beta_p);
free(Hy);
free(Hx);
free(kex);
free(aex);
free(bex);
free(amx);
free(bmx);
free(alpha_e);
free(alpha_m);
free(sigma_e_pml);
free(sigma_m_pml);
free(Psi_ezy);
free(Psi_ezx);
free(Psi_hyx);
free(Psi_hxy);
free(kmx);
```

```
free(D);
free(P);

free(hcjzxp);
free(hcjzyp);
free(hcjzxn);
free(hcjzyn);
free(hcmxyp);
free(hcmyxp);
free(hcmxyn);
free(hcmyxn);

cudaCheck( cudaFree(dev_Cezeic));
cudaCheck( cudaFree(dev_Cezeip));
cudaCheck( cudaFree(dev_eps_infinity));
cudaCheck( cudaFree(dev_sigma_e_z));
cudaCheck( cudaFree(dev_freq));
cudaCheck( cudaFree(dev_Phi));
cudaCheck( cudaFree(dev_i));
cudaCheck( cudaFree(dev_Beta_p));
cudaCheck( cudaFree(dev_Cezjp));

cudaCheck( cudaFree(cjzxp));
cudaCheck( cudaFree(cjzyp));
cudaCheck( cudaFree(cjzxn));
cudaCheck( cudaFree(cjzyn));
cudaCheck( cudaFree(cmxyp));
cudaCheck( cudaFree(cmyxp));
cudaCheck( cudaFree(cmxyn));
cudaCheck( cudaFree(cmyxn));


cudaCheck( cudaFree(dev_Cezhy));
cudaCheck( cudaFree(dev_Ceze));

cudaCheck( cudaFree(dev_bex));
cudaCheck( cudaFree(dev_aex));
cudaCheck( cudaFree(dev_bmx));
cudaCheck( cudaFree(dev_amx));
cudaCheck( cudaFree(dev_kex));
cudaCheck( cudaFree(dev_kmx));
cudaCheck( cudaFree(dev_Ez));
cudaCheck( cudaFree(dev_Jp));
cudaCheck( cudaFree(dev_Hy));
cudaCheck( cudaFree(dev_Hx));
cudaCheck( cudaFree(dev_Psi_ezy));
cudaCheck( cudaFree(dev_Psi_ezx));
cudaCheck( cudaFree(dev_Psi_hyx));
cudaCheck( cudaFree(dev_Psi_hxy));

cout << "fitness is: " << fit << endl;
return (double)fit;
}

__global__ void scattered_parameter_init(float *eps_infinity,float *sigma_e_z ,float *Cezeic, float *Cezeip, float *Cezjp,float *dev_Beta_p)
{
    int x=threadIdx.x+blockDim.x*blockIdx.x;
    int y=threadIdx.y+blockDim.y*blockIdx.y;
    if(x<(nx+1)&&y<(ny+1))
    {
        Cezeic[dgetCell(x,y,nx+1)] = (-2*eps0*eps_infinity[dgetCell(x,y,nx+1)]
        +2*eps0 - sigma_e_z[dgetCell(x,y,nx+1)]*dx - dev_Beta_p[dgetCell(x,y,nx+1)])
        /(2*eps0*eps_infinity[dgetCell(x,y,nx+1)]
        + sigma_e_z[dgetCell(x,y,nx+1)]*dt + dev_Beta_p[dgetCell(x,y,nx+1)]);
        Cezeip[dgetCell(x,y,nx+1)] = (2*eps0*eps_infinity[dgetCell(x,y,nx+1)] - 2*eps0 - sigma_e_z[dgetCell(x,y,nx+1)]*dt
         + dev_Beta_p[dgetCell(x,y,nx+1)])
         /(2*eps0*eps_infinity[dgetCell(x,y,nx+1)] + sigma_e_z[dgetCell(x,y,nx+1)]*dt + dev_Beta_p[dgetCell(x,y,nx+1)]);
        Cezjp[dgetCell(x,y,nx+1)] = ((1+Kp)*dt)/(2*eps0*eps_infinity[dgetCell(x,y,nx+1)]
        + sigma_e_z[dgetCell(x,y,nx+1)]*dt + dev_Beta_p[dgetCell(x,y,nx+1)]);

    }
```

```
}
```

6.2 FDTD_common.cxx

```
#include <cmath>
#include "FDTD_common.hxx"

int CPUgetxfromthreadIdNF2FF(int index)
{
    int x=0;
    if(index<(nx-2*NF2FFdistfromboundary-2))//yn
    {
        x = index+NF2FFdistfromboundary+1;
    }
    else if(index<(nx-4*NF2FFdistfromboundary+ny-2))//xp
    {
        x = nx-NF2FFdistfromboundary-1;
    }
    else if(index<(2*nx-6*NF2FFdistfromboundary+ny-4))//yp
    {
        x = nx-NF2FFdistfromboundary  - (index-(nx-4*NF2FFdistfromboundary+ny-2))-2;
    }
    else if(index<(2*nx-8*NF2FFdistfromboundary+2*ny-4))
    //xn notice 2*nx-8*NF2FFdistfromboundary+2*ny-4 is the max index term.
    {
        x = NF2FFdistfromboundary;
    }
    return x;
}

int CPUgetyfromthreadIdNF2FF(int index)
{
    int y=0;
    if(index<(nx-2*NF2FFdistfromboundary-2))
    {
        y = NF2FFdistfromboundary;
    }
    else if(index<(nx-4*NF2FFdistfromboundary+ny-2))
    {
        y = (index-(nx-2*NF2FFdistfromboundary-2))+NF2FFdistfromboundary;
    }
    else if(index<(2*nx-6*NF2FFdistfromboundary+ny-4))
    {
        y = ny-NF2FFdistfromboundary-1;
    }
    else if(index<(2*nx-8*NF2FFdistfromboundary+2*ny-4))
    {
        y = ny-NF2FFdistfromboundary-(index-(2*nx-6*NF2FFdistfromboundary+ny-4))-1;
    }
    return y;
}

float calc_incident_power(float freq)
{
    return (0.5/eta0)*pow(tau*sqrt(PI)*exp(-tau*tau*2*PI*freq*2*PI*freq/4),2);
    // just gonna assume gaussian pulse.  This is the fourier transform of the gaussian pulse.
}

float fitness(float* D,int max_index, float* measurement)
{
    float fit = 0;
    for(int i =0;i<max_index;i++)
    {
        fit -= pow((measurement[i]-D[i]),2)/(numberofexcitationangles*pow(measurement[i],2));
    }
```

```
    return fit;
}
```

6.3 FDTD_GPU.hxx

```
#ifndef FDTD_COMMON_H
#define FDTD_COMMON_H
#define PI 3.141592653589793238
#define alpha_max 0.01
#define alpha_min 0.000
#define eps0 8.85418e-12
#define sigma_factor 1.0
#define ncells 10
#define mu0 (PI*4e-7)
#define center_freq (5e9)
#define eta0 (sqrt(mu0/eps0))
#define c0 (1.0/sqrt(mu0*eps0))
#define dt (dx/c0/2)// dx/c0/2
#define domain_size 0.18
#define dx (0.001)
#define NF2FFdistfromboundary ((int)floor((3.2*breast_radius/dx)))
#define source_position 0.5
#define dy (0.001)
#define number_of_time_steps 2000
#define f1x (nx/2 - 150)
#define f2x (nx/2+150)
#define f1y (ny/2)
#define f2y (ny/2)
//#define nx ((int)ceil(domain_size/dx))
//#define ny ((int)ceil(domain_size/dy))
#define nx ((int)ceil(12.7*breast_radius/dx))
#define ny ((int)ceil(12.7*breast_radius/dy))
#define d (10*dx)
#define npml 2
#define kmax 10
#define numberofexcitationangles 4
#define isPW 1
#define isscattering 1
#define sigma_max_pml (3/(200*PI*dx))
#define size_NF2FF_total (2*nx-8*NF2FFdistfromboundary+2*ny-4)
#define startfrequency (1e9)
#define stopfrequency (1e10)
#define deltafreq ((stopfrequency-startfrequency)/(numberoffrequencies-1))
#define numberoffrequencies 10
#define size_cjzy ((nx-2*NF2FFdistfromboundary-2)*(numberoffrequencies))
#define size_cjzx ((ny-2*NF2FFdistfromboundary)*(numberoffrequencies))
#define size_y_side (nx-2*NF2FFdistfromboundary-2)
#define size_x_side (ny-2*NF2FFdistfromboundary)
#define numberofobservationangles  100
#define t0 (sqrt(20.0)*tau) // t0 = sqrt(20)*tau
#define l0 (nx*dx/2-breast_radius)
#define pwidth 10
#define nc 20 // 20 cells per wavelength
#define  fmax  (c0/(nc*dx))
// change if dy is bigger though now they're the same  fmax is the highest frequency this program can handle
#define tau (3.3445267e-11)
// float ta bu = sqrt(2.3)*nc*dx/(PI*c0*1/sqrt(eps_r_MAX));  from a calculation of fmax.
//#define tau (5.288161e-11)
#define target_x (nx/2+15)//105 is breast_radius / dx
#define target_y (ny/2-15)
#define source_x (nx/2)      //(target_x-105-80)
#define source_y (ny/2)
#define breast_radius 0.0315 //87.535 mm  .  Sample size = 1.
#define tumor_size (0.01)
#define relaxation_time (7e-12)
#define Kp ((1-dt/(2*relaxation_time))/(1+dt/(2*relaxation_time)))
 // Parameter for (FD)2TD.  Using Polarization current formulation
```

```
#define numpatches 3
/**
 * I made getCell a macro which should speed things up
 * a bit, and make the code simpler
 */
#define getCell(x,y,size) ((x) + ((y) * (size)))
#define dgetCell(x,y,size) ((x) + ((y) * (size)))

int CPUgetxfromthreadIdNF2FF(int index);

int CPUgetyfromthreadIdNF2FF(int index);

float calc_incident_power(float freq);

float fitness(float* D,int max_index, float* measurement);

#endif
#ifndef FDTD_COMMON_H
#define FDTD_COMMON_H

#define GL_GLEXT_PROTOTYPES
#define PI 3.141592653589793238
#define alpha_max 0.01
#define alpha_min 0.000
#define eps0 8.85418e-12
#define sigma_factor 1.0
#define ncells 10
#define mu0 (PI*4e-7)
#define center_freq (5e9)
#define eta0 (sqrt(mu0/eps0))
#define c0 (1.0/sqrt(mu0*eps0))
#define dt (dx/c0/2)// dx/c0/2
#define domain_size 0.18
#define dx (0.001)
#define NF2FFdistfromboundary ((int)floor((3.2*breast_radius/dx)))
#define source_position 0.5
#define dy (0.001)
#define number_of_time_steps 2000
#define f1x (nx/2 - 150)
#define f2x (nx/2+150)
#define f1y (ny/2)
#define f2y (ny/2)
//#define nx ((int)ceil(domain_size/dx))
//#define ny ((int)ceil(domain_size/dy))
#define nx ((int)ceil(12.7*breast_radius/dx))
#define ny ((int)ceil(12.7*breast_radius/dy))
#define d (10*dx)
#define npml 2
#define kmax 10
#define numberofexcitationangles 4
#define isPW 1
#define isscattering 1
#define HANDLE_ERROR( err ) err
#define sigma_max_pml (3/(200*PI*dx))
#define size_NF2FF_total (2*nx-8*NF2FFdistfromboundary+2*ny-4)
#define startfrequency (1e9)
#define stopfrequency (1e10)
#define deltafreq ((stopfrequency-startfrequency)/(numberoffrequencies-1))
#define numberoffrequencies 10
#define size_cjzy ((nx-2*NF2FFdistfromboundary-2)*(numberoffrequencies))
#define size_cjzx ((ny-2*NF2FFdistfromboundary)*(numberoffrequencies))
#define size_y_side (nx-2*NF2FFdistfromboundary-2)
#define size_x_side (ny-2*NF2FFdistfromboundary)
#define numberofobservationangles  100
#define t0 (sqrt(20.0)*tau) // t0 = sqrt(20)*tau
#define l0 (nx*dx/2-breast_radius)
#define pwidth 10
#define nc 20 // 20 cells per wavelength
#define  fmax  (c0/(nc*dx))
// change if dy is bigger though now they're the same  fmax is the highest frequency this program can handle
#define tau (3.3445267e-11)
```

```
 // float ta bu = sqrt(2.3)*nc*dx/(PI*c0*1/sqrt(eps_r_MAX));  from a calculation of fmax.
//#define tau (5.288161e-11)
#define target_x (nx/2+15)//105 is breast_radius / dx
#define target_y (ny/2-15)
#define source_x (nx/2)      //(target_x-105-80)
#define source_y (ny/2)
#define breast_radius 0.0315 //87.535 mm  .  Sample size = 1.
#define tumor_size (0.01)
#define relaxation_time (7e-12)
#define Kp ((1-dt/(2*relaxation_time))/(1+dt/(2*relaxation_time)))
// Parameter for (FD)2TD.  Using Polarization current formulation
#define numpatches 3
/**
 * I made getCell a macro which should speed things up
 * a bit, and make the code simpler
 */
#define getCell(x,y,size) ((x) + ((y) * (size)))
#define dgetCell(x,y,size) ((x) + ((y) * (size)))

int CPUgetxfromthreadIdNF2FF(int index);

int CPUgetyfromthreadIdNF2FF(int index);

float calc_incident_power(float freq);

float fitness(float* D,int max_index, float* measurement);

#endif
```