ACCELERATION OF ASYMPTOTIC COMPUTATIONAL ELECTROMAGNETICS PHYSICAL OPTICS – SHOOTING AND BOUNCING RAY (PO-SBR) METHOD USING CUDA

BY

HUAN-TING MENG

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Adviser:

Professor Jianming Jin

ABSTRACT

The objective of this research is to accelerate the Physical Optics - Shooting and Bouncing Ray (PO-SBR), an asymptotic computational electromagnetics (CEM) method, on the recently emerged general purpose graphics processing unit (GPGPU) using NVIDIA's CUDA environment. In modern engineering, simulation programs are used to aid the development of advanced devices, and this is where CEM plays the important role of simulating the propagation of electromagnetic (EM) waves and fields using modern computers. In this thesis, CUDA on NVIDIA's GPU is used to accelerate the PO-SBR method, which greatly reduces the computational time required for various problems.

Starting with the theoretical background, we introduce the PO-SBR method, including the ray tracing and the electromagnetic aspects of the method. Next, we discuss its implementation using the standard CPU C++ language and point out the computationally parallel nature of the method. NVIDIA GPU's hardware architecture is then described to show the portability of the method onto GPU devices. Then, NVIDIA's GPU programming environment, CUDA, is introduced for the implementation of the part of the method to be parallelized. Finally, this thesis presents a novel and flexible method of implementation which fully exploits the hardware architecture of the GPU devices, while at the same time remaining flexible and intelligent enough to be able to optimize itself even on different NVIDIA GPU hardware platforms. The acceleration reaches more than 50 times speedup as compared to the traditional CPU version of the code, and it is believed that a higher speedup can still be achieved with problems of increasing complexity.

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

This chapter provides an overview for the background and the general approach discussed in the thesis. For electrically small problems, where the size of the structures under study is relatively small compared to the wavelength of the fields, full wave methods are good computational electromagnetic (CEM) methods to accurately calculate the field distribution on the structures. However, when the problems are electrically large, for example, the scattering of radar cross section (RCS) of transportation carriers, or the computation of signal strength from the telecommunication towers to the whole city, full wave methods are extremely time-consuming to the extent that the computational process becomes practically impossible. Various analytical methods have been proposed in the past as alternative solutions to the problem [1]-[4]. However, the analytical approaches often rely heavily on the simplicity and the symmetry of the structures, such as open-ended circular cylinders. These approaches are not able to solve structures with more complexity.

To address electrically large problems, asymptotic methods are developed to approximate electromagnetic propagation, using the optical properties of very high frequency EM waves. One of these methods is the Physical Optics - Shooting and Bouncing Ray (PO-SBR) method [5]. It is a method combining the method of Geometric Optics (GO) [6]-[9] and Physical Optics (PO) [9]. In Geometric Optics, electromagnetic propagation is first represented with the reflection, refraction, and divergence of optical rays. The electromagnetic properties of magnitude, direction, and phase are then added on top of the ray traces to mimic the properties of waves. In Physical Optics, incident electromagnetic waves are converted into equivalent surface currents on the scattering surfaces of the structure under study, using the surface equivalence principles. The current is then integrated and re-radiated as electromagnetic waves towards the observation points everywhere in the computational domain.

Combining Geometric Optics and Physical Optics methods, PO-SBR begins the computation by launching numerous rays from the sources, usually antennas or regions of plane waves. The magnitude of the rays is dependent upon the pattern of the source. Each ray then propagates through the computational domain and bounces between the target surfaces under study using the GO method. Next, the electromagnetic fields at each hit point are converted into surface currents using the PO methods and re-radiated towards all observation points. Finally, the fields at each observation point are summed up to represent the final electromagnetic field computed at the corresponding location of the computational domain. Although the optical approximation is not a precise electromagnetic theory compared to the full wave methods, asymptotic methods are able to provide a very fast and reasonable estimation of the field strength in presence of electrically large structures.

It can be seen that the PO-SBR method described above can still be computationally expensive in terms of computation time. For the ray tracing part of the method, modern rendering programs such as Physically Based Rendering Technique (PBRT) [10] are advanced enough with many built-in ray tracing accelerators. Also, since the majority of the PO-SBR applications focus on target structures whose shapes are relatively simple for the ray tracers, the majority of the computational time taken is due to the steps dealing with the electromagnetic aspect of the method, where the fields found at each hit point need to be converted into surface currents, then integrated and re-radiated towards all observation points. Moreover, computing fields from different source frequencies involves going through the entire process separately, and thus further increasing the overall computation time.

To accelerate PO-SBR method, the parallel nature of the method is carefully studied. It can be easily seen that the electromagnetic information carried in a given ray is independent of that carried in another. In other words, the field distribution of the computational domain is merely a summation of the fields resulting from different initial rays. These rays bounce, induce the currents, and re-radiate without interference with each other, and thus are highly parallel in terms of the electromagnetic computation. Traditionally, parallel computation is done through many-core clusters of CPU processors, which are expensive and space-consuming. Alternatively, there have been numerous attempts in the recent past to utilize the graphics processing units (GPU) for their multi-core nature on personal computers [11]. While GPU hardware is originally constructed for graphical purposes, researchers exploit libraries and frameworks designed for graphical purposes for general computation purposes. However, extensive modifications and translations of the original codes need to be developed to fit the graphical libraries and frameworks, thus making the applications of GPGPU difficult. A turning point came recently when NVIDIA released CUDA, an environment specifically designed for GPGPU programming purposes. This has eased the use of GPU for general purpose computation, and GPU parallelization has quickly been adapted for electromagnetic computational methods [12].

Starting with the theoretical background, we introduce the PO-SBR method, including the ray tracing and the electromagnetic aspects of the method in this thesis. To simplify the code to its most basic elements for parallelization, only PEC target structures are considered in this thesis. Next, we discuss its implementation using the standard CPU C++ language and point out the computationally parallel nature of the method. NVIDIA GPU's hardware architecture is then described to show the portability of the method onto GPU devices. Then, NVIDIA's GPU programming environment, CUDA [13], is introduced for the implementation of the part of the method to be parallelized. Finally, this thesis presents a novel and flexible method of implementation which fully exploits the hardware architecture of the GPU devices, while remaining flexible and intelligent enough to be able to optimize itself even on different NVIDIA GPU hardware platforms. The acceleration reaches more than 50 times speedup as compared to the traditional CPU version of the code, and it is believed that a higher speedup can still be achieved with problems of increasing complexity.

CHAPTER 2
PO-SBR THEORY

This chapter describes the PO-SBR method by breaking the method into a few components. Section 2.1 describes the ray tracing aspect of SBR. Section 2.2 describes the electromagnetic aspect of SBR. Section 2.3 describes the Physical Optics. The version of the PO-SBR used by this thesis is detailed by the derivations of equations and the explanation of their meanings and choices, and the overall picture is shown in Figure 2.1. First, SBR traces the hit points and updates the incident electromagnetic fields from the source onto those hit points. Next, PO paints the equivalent surface current using the surface equivalence theory. The re-radiations of the surface currents from all hit points are then collected and summed at the desired observation points. And finally, the source radiates directly towards the observation points using SBR, and the resulting fields are added on top of the summation from the surface currents to become the final field solution at the observation points.

2.1 Shooting and Bouncing Rays - Ray Tracing

This section describes the ray tracing aspect of SBR. The ray tracing aspect of the PO-SBR theory is fairly simple and straightforward. It consists of bundles of rays radiating from sources in the computational domain, whether they are antennas positioned in the space or plane waves propagating towards the target structures.

There are many different methods to launch the rays, and here three common methods are described (consultation with SAIC, August 2010). The first method is the burst method, in which the rays are launched in equal degree spacing throughout the entire antenna radiation pattern or plane wave front, as shown in Figure 2.2. This is the most straightforward way to launch the rays, in that it does not require sophisticated treatments on how to distribute the rays. However, this crude way will most likely generate excess rays in the computational domain that will never be incident upon the target structures, and thus it is a drawback in terms of computational resources.

A more advanced method is the bounding box method, which first encloses the target structure in a volume. Then, rays are generated at equal spacing on the volume and launched towards the volume, as shown in Figure 2.3. This is a more dynamic way of generating the rays, in that it can guarantee that all of the generated rays will be able to be incident at the structure. While being a more advanced method than the previous one, there are still spaces for improvement. For one, the rays are still generated in equal degree spacing without considering the detailed geometry of the structure. That is, some parts of the target structure might be large and flat, where the field information can be easily obtained using sparse rays. On the other hand, some parts of the target structure might be intricate, where dense rays need to be generated to capture the signature of the complexity.

Therefore, the most sophisticated method is the surface mesh method, where the rays are generated according to the triangular facets of the target structure mesh, as shown in

Figure 2.4. This is the most optimal method in terms of accurately capturing the geometry of the target structure, and the field information resulting from the unevenly spaced rays will yield a more precise solution than any of the previous methods.

Notice that the maximum spacing between rays in any of the above methods is not arbitrary. The ray tracing result should provide sufficient initial hit points to correctly mimic the impact of the electromagnetic wave front. The surface mesh method is naturally an efficient method, where only a few rays need to be generated to mimic the impact on large flat surfaces, and more rays are needed to model the impact on curvy surfaces. As for the burst method and the bounding box method, since the rays are generated in equal angles, the angular spacing of the rays needs to be small enough to capture the curvy surfaces, and thus the computational efficiency is sacrificed by the extra rays on the flat surfaces. Since perfect precision is not a concern for the asymptotic methods, the practice in industry is to find convergence to the problem through initial guess and trial and error.

Once the rays have been successfully generated, they will propagate independently through the computational domain until incident upon the target structure surfaces. At this point the rays behave analogously to light. For perfect electric conductor (PEC) surfaces, the rays bounce off with angles equal to that of the incident field, as described with laws of reflection. For dielectric surfaces, the rays refract into the target structure using Snell's law.

After the first hit points, the subsequent reflected and refracted rays will continue to propagate independently within the computational domain until the following. 1. The ray propagates out of the computation environment; 2. The ray has bounced far enough to lose its corresponding electromagnetic intensity; 3. The ray is caught in a cavity and is bouncing back and forth between a set of repeating hit points. At this point, the rays are then terminated to avoid unnecessary tracings, and the electromagnetic information is ready to be added on top of the rays.

For simplicity, the burst method is used in this thesis to launch the rays into the computational domain, with a variable angular spacing radiance, denoted *spacingRad*.

2.2 Shooting and Bouncing Rays - Electromagnetics

This section describes the electromagnetic aspect of SBR on top of the ray tracing. From the previous section, the ray tracing procedure has provided the computational domain with numerous hit points, caused by the emission of rays from the sources. The ray tracing procedure will output the necessary information required for the calculation of the electromagnetic aspect of the field, namely, the ray depth of bounces, the coordinates of the hit points, the normal vector $\hat{n}$ of the respective surface elements, and the propagation constant $\hat{k}$ of the reflected or refracted ray at the hit points. Once the step is complete, the electromagnetic aspect of SBR is ready to be added on top of the ray traces.

Starting from the sources, the field distribution of the sources is represented onto the electromagnetic aspect of the rays with varying magnitudes. If the source is simply a plane wave incident into the computational domain, each ray then carries the same initial magnitude and phase. On the other hand, if the source is an antenna in the computational domain, each ray still carries the same initial phase with the magnitudes varying according to the antenna field directivity, power directivity, or simply according to the far or near field distribution equation. In this thesis, the Hertzian dipole is used as the source, with a field distribution equation of:

$$E_\theta = j\eta \frac{kIdl\sin\theta}{4\pi}\frac{e^{-jkr}}{r}, \tag{1.1}$$

where two kinds of radiation from the source need to be computed. One of them is the direct line-of-sight radiation from the source to the observation points. This radiation represents the direct Geometric Optics contribution to the final field at the observation points. The other is the Physical Optics contribution to the final field at the observation points through the equivalent surface currents on the hit points. These currents and their resulting re-radiating fields can be obtained through the following steps.

The fields at the first hit point of each ray can be obtained from the field distribution of the source and the distance between the source and the first hit points. Since only PEC target structures are considered in this thesis, only the magnetic field $\vec{H}^{inc}$ at the first hit points needs to be calculated for the conversion to surface current. While traveling further onto subsequent hit points, only three aspects of the field change their value: direction, phase, and magnitude. Upon incidence on the first hit points, the fields will bounce away following the path of the respective ray trace, and propagate toward the next hit points, as shown in Figure 2.5. The field at the new hit points can be obtained from the previous hit points from Equation (1.2) and Equation (1.3):

$$\vec{E}^{inc} = \vec{E}^{ref}_{prev} e^{-jkd}, \qquad \vec{E}^{ref} = \hat{n}(2\hat{n} \cdot \vec{E}^{inc}) - \vec{E}^{inc}, \tag{1.2}$$

$$\vec{H}^{inc} = \vec{H}^{ref}_{prev} e^{-jkd}, \qquad \vec{H}^{ref} = \vec{H}^{inc} - \hat{n}(2\hat{n} \cdot \vec{H}^{inc}), \tag{1.3}$$

where d is the distance between the current and previous hit points. First, the incident fields at the current hit points are flipped of their direction, using the laws of reflection. Then, the resulting reflected fields at the current hit points propagate toward the following hit points, found by the ray tracing procedure described in Section 2.1. The fields then become the incident fields for the following hit points, and the procedure repeats until the respective rays die out, which can be either hard fixed to a certain number of bounces, or when the magnitude of the field drops to a certain desired level. At this point, the direction and phase of the field are updated at all hit points.

Notice that the electromagnetic fields of the sources are continuous but the initial rays are discreet. Each ray is responsible for carrying the electromagnetic information of a patch of the radiating sphere of the source as ray tubes. These patches will also follow the path of their respective ray traces and transform into rectangular footprint areas for the painted currents. The patches will first travel the computational domain along with the respective rays until projected upon the target surface mesh, where an initial footprint size can be found. Based on the ray distribution of the burst method, the corresponding patch size of each ray is integrated as a function of ray angle and distance to the hit point, as shown in Equation (1.4):

$$\begin{aligned}\text{Area} &= \iint ds \\ &= \iint r^2 \sin\theta d\theta d\phi \\ &= -\Delta_\phi r^2 \cos\theta \Big|_{\theta_1}^{\theta_2} \\ &= -\Delta_\phi r^2 \left(\cos\theta_2 - \cos\theta_1\right) \\ &= -\Delta_\phi r^2 \left[\cos\left(\theta_0 + \frac{\Delta_\theta}{2}\right) - \cos\left(\theta_0 - \frac{\Delta_\theta}{2}\right)\right]\end{aligned} , \tag{1.4}$$

since

$$\cos a - \cos b = -2\sin\left(\frac{a+b}{2}\right)\sin\left(\frac{a-b}{2}\right), \tag{1.5}$$

$$\text{Area} = -\Delta_\phi r^2 2\sin\theta_0 \sin\left(\frac{\Delta_\theta}{2}\right). \tag{1.6}$$

Thus, the length l of the initial patch size is just the square root of the patch area:

$$l = r_o\sqrt{2 \cdot SpacingRad \cdot \sin\theta \cdot \sin\left(\frac{SpacingRad}{2}\right)}. \tag{1.7}$$

Since electromagnetic waves diverge with distance, the initial footprint size will expand with subsequent bounces to reflect ray tube divergence, as shown in Figure 2.5, where the rate of expansion is proportional to the total distance traveled, d_{total}, as denoted by Equation (1.8):

$$l_2 = \left(\frac{d_{total}}{d_0}\right) l_1, \tag{1.8}$$

where l_1 and l_2 are respectively the current and next patch length, and d_0 is the distance from the source to the initial hit point.

As the footprints expand to represent the area of the eventual painted currents, power conservation is also needed for each incident field at the hit points. To conserve energy, the magnitude of the fields must now be inversely proportional to the total distance traveled, as will be described in the following section. At this point, the magnitude of the field is updated, and the fields have been radiated from the source to all hit points.

2.3 Physical Optics

This section describes the process of Physical Optics after the process of SBR is completed. Once the incident field at each hit point is found through SBR, it is converted into the surface painted current using the surface equivalence principle, with the area equivalent to the projected footprint by the incident ray tube. Equation (1.9) is used to convert incident magnetic fields into surface painted currents on PEC surfaces:

$$\vec{J} = 2\hat{n}\times\vec{H}^{inc}\cdot\frac{-\vec{k}^{inc}\cdot\hat{n}}{|\vec{k}^{inc}\cdot\hat{n}|}. \tag{1.9}$$

Notice that the sign correction at the end of Equation (1.9) is added as an extra guarantee to the normal direction of the corresponding surface mesh, since the normal vector $\hat{n}$ obtained from the ray tracing might not be on the correct side of the mesh. The region of the footprint of the surface currents is represented by vectors $\vec{u}$ and $\vec{v}$ on the corresponding surface mesh, as shown in Figure 2.6, where

$$\vec{v} = l\cdot\hat{v}, \qquad \hat{v} = \frac{\hat{n}\times\vec{k}^{ref}}{\left|\hat{n}\times\vec{k}^{ref}\right|}, \tag{1.10}$$

where $\hat{v}$ is the unit surface vector of one side of the footprint obtained by the normalized cross product of $\hat{n}$ and $\vec{k}^{ref}$, and

$$\vec{u} = \frac{l}{|\hat{n}\cdot\vec{k}^{ref}|}\cdot\hat{u}, \qquad \hat{u} = \hat{v}\times\hat{n}, \tag{1.11}$$

where $\hat{u}$ is the unit surface vector of one side of the footprint obtained by the cross product of $\hat{v}$ and $\hat{n}$. The factor $|\hat{n}\cdot\vec{k}^{ref}|$ is divided from the magnitude of $\bar{u}$ to take into account only the tangential component of the incident wave.

While Equation (1.9) is applicable to derive the surface painted current throughout the entire area of the corresponding footprint, it will only result in correct phasing at the hit point, the center of the footprint, where $\vec{H}^{inc}$ is used. To correct the phasing due to the field approximation, Equation (1.12) is used to adjust the phase of the center current $\vec{J}^{center}$ at the hit point to represent the current everywhere in the corresponding footprint:

$$\vec{J} = \vec{J}^{center} e^{-j\vec{k}^{inc}\cdot\vec{s}}\,, \tag{1.12}$$

where $\vec{k}^{inc}$ is the unit incident wave vector and $\vec{s}$ is the vector from the hit point to current locations on the footprint, as shown in Figure 2.6.

After footprints of painted surface current have been found everywhere on the target structure's surface, they will re-radiate back into the computational domain, as described by physical optics. Specifically, every painted current at every hit point will re-radiate towards every observation point, the locations in the computational domain where the strength of the field is of interest. Then, the fields re-radiated from all hit points toward the same observation point are summed, and become the PO part of the final field at the observation point.

Green's function is introduced to calculate the field at the observation points. In general, if there exists a wave equation in the form of

$$(\nabla^2 + k^2) f(\bar{r}) = s(\bar{r})\,, \tag{1.13}$$

in which $f(\bar{r})$ is the unknown, then it can be solved using Green's theorem where

$$f(\vec{r}) = -\int g(\vec{r},\vec{r}')s(\vec{r}')d\vec{r}' , \tag{1.14}$$

where

$$g(\vec{r},\vec{r}') = \frac{e^{-jkR}}{4\pi R}, \quad R = |\vec{r} - \vec{r}'| \tag{1.15}$$

is a Green's function satisfying

$$(\nabla^2 + k^2) f(\vec{r}) = -\delta(\vec{r} - \vec{r}') . \tag{1.16}$$

For the re-radiation problem, the equation for the fields can be formulated using the wave equation. Start from Maxwell's equations, where

$$\nabla \times \vec{E} = -j\omega\mu\vec{H} , \tag{1.17}$$

$$\nabla \times \vec{H} = \vec{J} + j\omega\varepsilon\vec{E} . \tag{1.18}$$

The equations can be combined to form

$$\begin{aligned} \nabla \times \nabla \times \vec{E} &= -j\omega\mu(\vec{J} + j\omega\varepsilon\vec{E}) \\ &= -j\omega\mu\vec{J} + \omega^2\mu\varepsilon\vec{E} \end{aligned} . \tag{1.19}$$

Using the vector identity of

$$\nabla \times \nabla \times \vec{A} = \nabla(\nabla \cdot \vec{A}) - \nabla^2 \vec{A}, \tag{1.20}$$

Equation (1.19) can be rewritten as

$$\nabla(\nabla \cdot \vec{E}) - \nabla^2 \vec{E} = -jk\eta\vec{J} + k^2\vec{E} , \tag{1.21}$$

then reordered into

$$\nabla^2 \vec{E} + k^2 \vec{E} = jk\eta\vec{J} + \frac{1}{\varepsilon}\nabla\rho . \tag{1.22}$$

For free space propagation, since the patches only contain uniform surface current without charge density source, the second term of Equation (1.22) can be omitted, reducing the equation into

$$\nabla^2 \vec{E} + k^2 \vec{E} = jk\eta \vec{J} \,, \tag{1.23}$$

which is in the form of Equation (1.13). The unknown electric field can then be solved using Equation (1.14) and is written as

$$\vec{E} = -jk\eta \iint_s g(\bar{r}, \bar{r}')\vec{J} ds \,, \tag{1.24}$$

with the current $\vec{J}$ expressed as in Equation (1.12). The distance variable R in the Green's function expressed in Equation (1.15) can be approximated as shown in Figure 2.7, with

$$kR = kd - \vec{k}^{obs} \cdot \vec{s} \,, \tag{1.25}$$

$$R \approx d \,, \tag{1.26}$$

and approximating the Green's function as

$$g = \frac{e^{-jkR}}{4\pi R} \approx \frac{e^{-jkd} \cdot e^{j\vec{k}^{obs} \cdot \vec{s}}}{4\pi d} \,. \tag{1.27}$$

Substituting Equation (1.12) and Equation (1.15) into Equation (1.24) yields

$$\vec{E}^{scat} = -\frac{jk\eta}{4\pi} \frac{e^{-jkd}}{d} \vec{J}^{center} \iint_s e^{j\left(\vec{k}^{obs} - \vec{k}^{inc}\right) \cdot \vec{s}} ds \,. \tag{1.28}$$

Notice that the double integral in Equation (1.28) integrates the current distribution across the whole footprint area. With $\vec{s}$ being the vector from the hit point to current locations on the footprint, it can be split and integrated separately by vectors $\vec{u}$ and $\vec{v}$. Using the integral equation of

$$\int_{-a/2}^{a/2} e^{jkx}dx = \left.\frac{e^{jkx}}{jk}\right|_{-a/2}^{a/2} = a\frac{\sin\left(\frac{ka}{2}\right)}{\left(\frac{ka}{2}\right)}, \tag{1.29}$$

Equation (1.28) can be written as

$$\begin{aligned}\vec{E}^{scat} = -\frac{jk\eta}{4\pi}\frac{e^{-jkd}}{d}\vec{J}^{center}\,\cdot \\ A\cdot\text{sinc}\left[\left(\vec{k}^{obs}-\vec{k}^{inc}\right)\cdot\hat{u}\frac{\Delta u}{2}\right]\cdot\text{sinc}\left[\left(\vec{k}^{obs}-\vec{k}^{inc}\right)\cdot\hat{v}\frac{\Delta v}{2}\right],\end{aligned} \tag{1.30}$$

and this is the equation for the electric field used for re-radiation from all hit points to all observation points. Note that the theoretical re-radiated far fields of the currents are plane waves orthogonal to the traveling direction. Since approximations have been made on the electric field equation, it is desired to take out the erroneous computed field parallel to the traveling direction. Equation (1.30) thus becomes

$$\vec{E}_{final}^{scat} = \vec{E}^{scat} - (\vec{E}^{scat}\cdot\vec{k}^{obs})\vec{k}^{obs}, \tag{1.31}$$

and this is the final re-radiated electric field equation. The final re-radiated magnetic field equation is then simply

$$\vec{H}_{final}^{scat} = \frac{1}{\eta}\vec{k}^{obs}\times\vec{E}_{final}^{scat}. \tag{1.32}$$

2.4 Figures

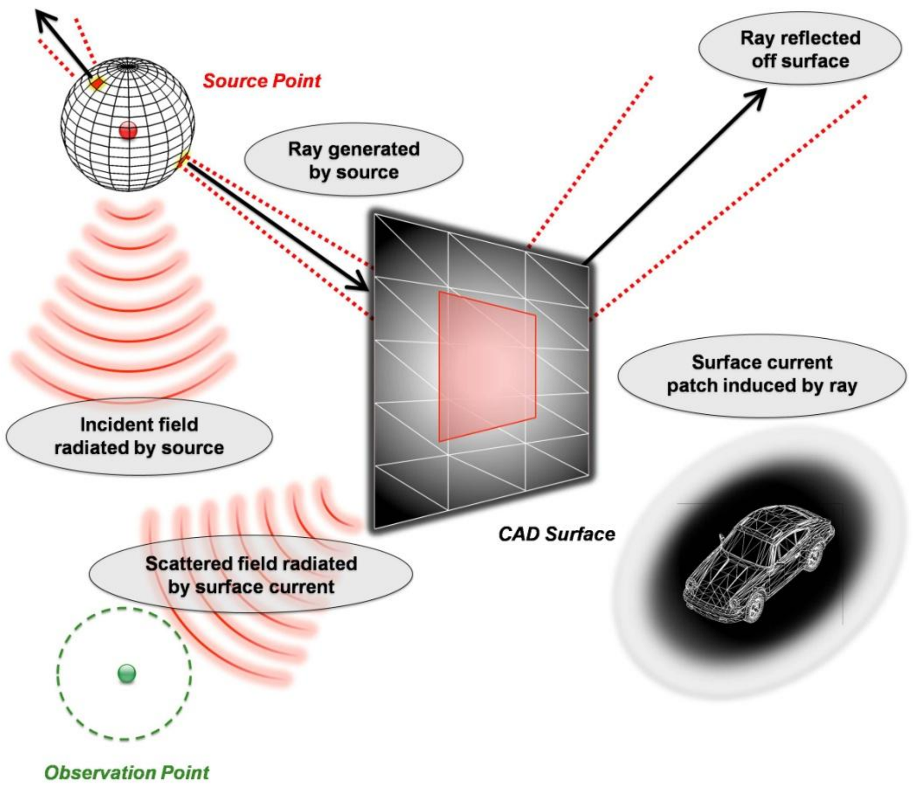

Source Point
Ray generated by source
Ray reflected off surface
Incident field radiated by source
Surface current patch induced by ray
CAD Surface
Scattered field radiated by surface current
Observation Point

Figure 2.1 PO-SBR overall picture [14].

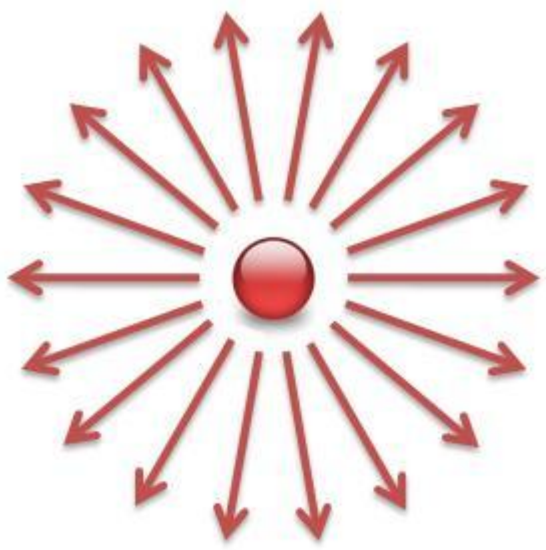


Figure 2.2 Burst ray launching method.

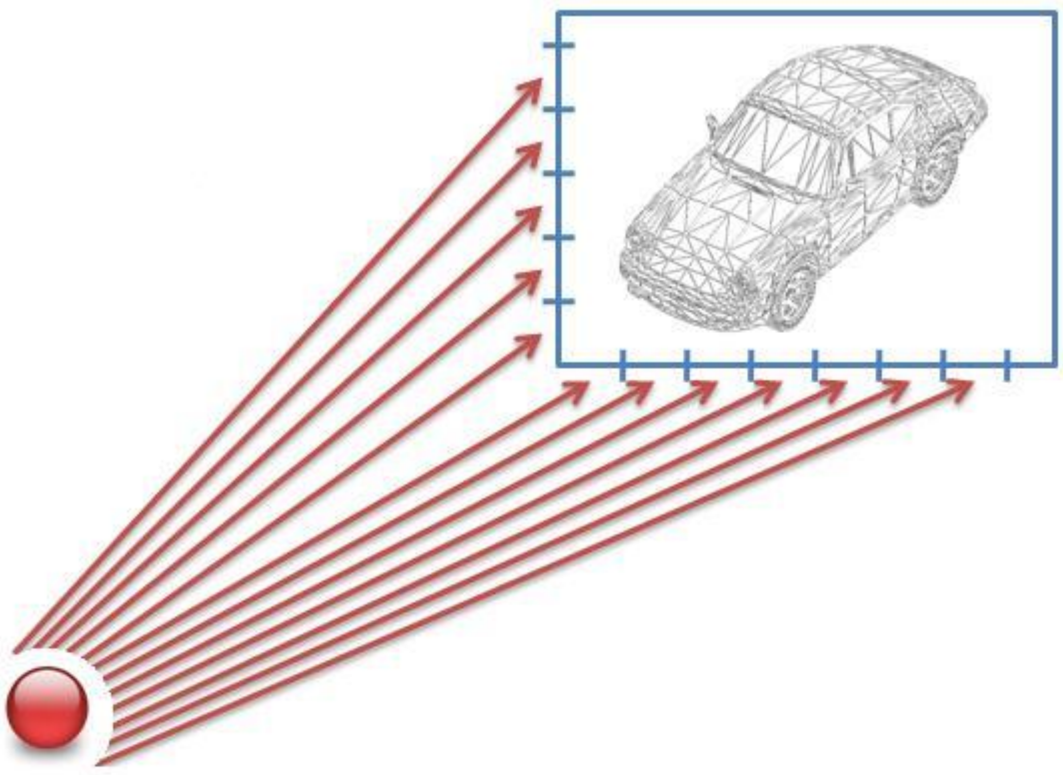


Figure 2.3 Bounding box ray launching method.

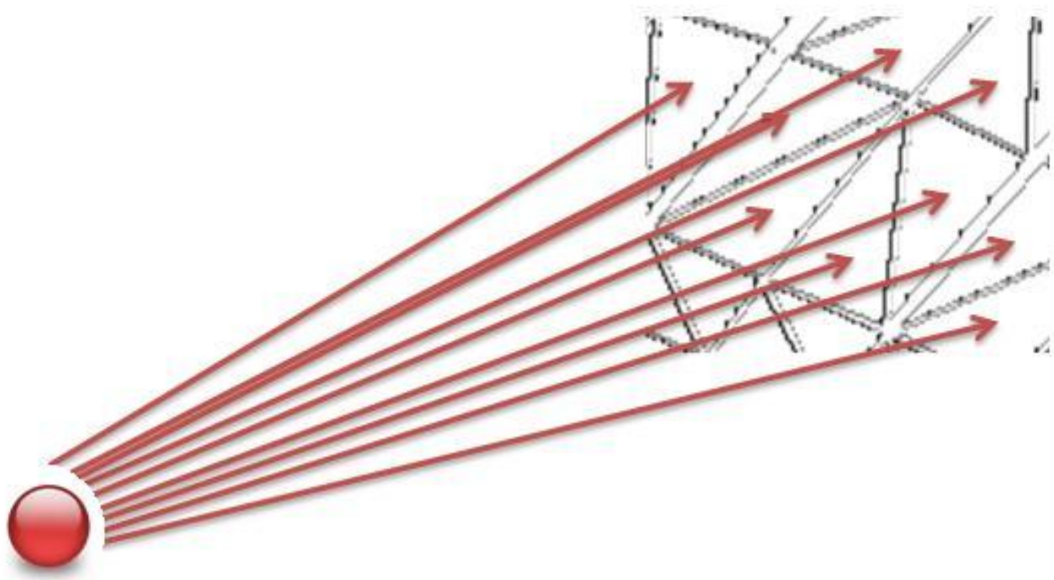


Figure 2.4 Surface mesh ray launching method.

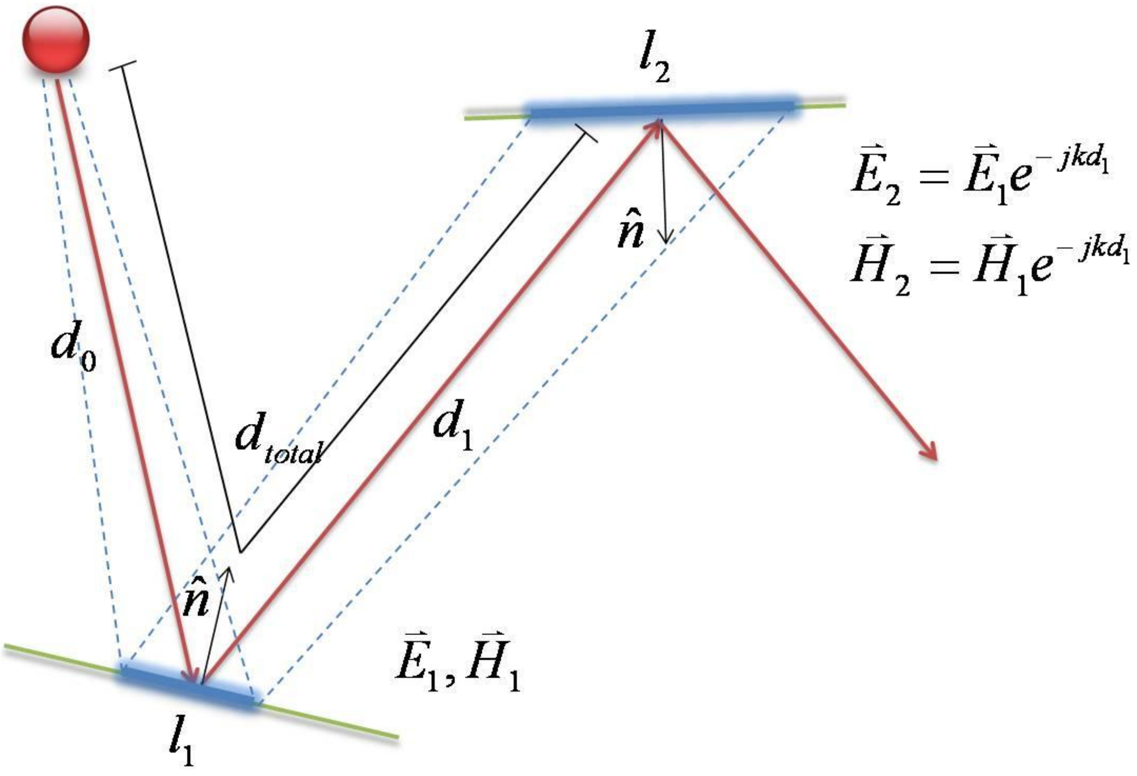


Figure 2.5 Field phase update and surface current expansion.

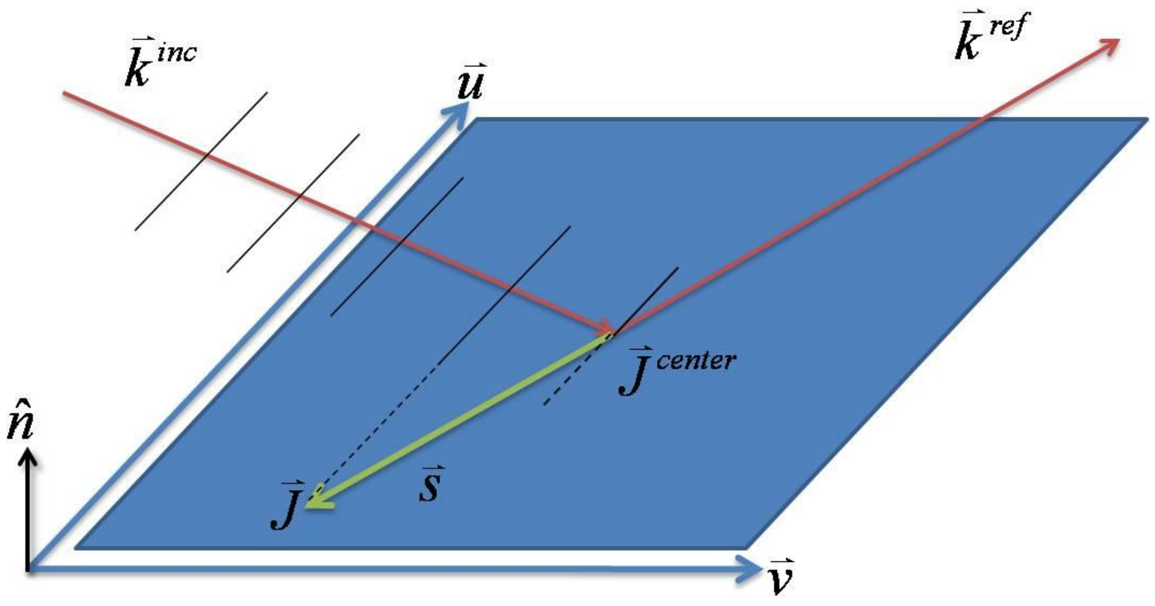


Figure 2.6 Equivalent surface painted current.

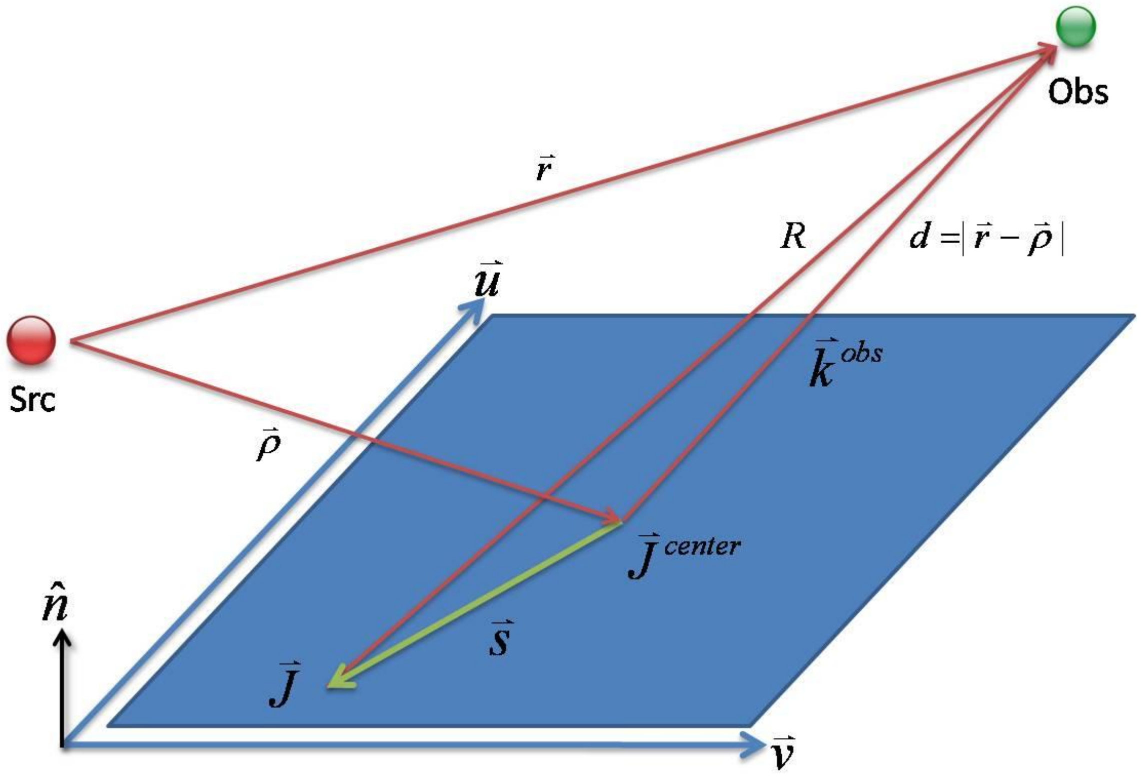


Figure 2.7 Re-radiation of the surface current towards the observation point.

CHAPTER 3
GPU IMPLEMENTATION

This chapter describes the GPU implementation of the PO-SBR. Section 3.1 provides the background and the architecture of GPU. Section 3.2 introduces CUDA as the interface to implement codes on GPU. Section 3.3 lays out the proposed method used for GPU implementation of the PO-SBR.

3.1 NVIDIA's GPU Architecture

This section provides the background and the architecture of the GPU from NVIDIA. In contrast to a modern CPU which has few cores operating at high frequency, a modern GPU has numerous cores operating at a slightly lower frequency. While CPU specializes in serial information processing, GPU operates much faster in massively parallel information processing.

To compare the processing capability of the processors, floating point operations per second (FLOPS) are used, and the FLOPS comparison between Intel's CPUs and NVIDIA's GPUs is shown in Figure 3.1, where the peak number of flops of the GPUs exceeds that of the CPUs, with a trend of an ever increasing difference. Another important factor in the overall processing speed is the memory bandwidth. According to Figure 3.2, it can be seen that the same trend can be found with the memory bandwidth in the CPU-GPU memory bandwidth comparison.

Although GPUs have higher flops and memory bandwidth than those of CPUs, they serve a completely different purpose and thus cannot easily be exchanged or replaced. As stated above, CPUs have very few cores, but with each core operating at a high frequency. This is especially suitable for normal operating system software processing and computation, as most programs and tasks are serial and thus single threaded. CPUs are especially adapted to this, with high frequency processing units and large cache, as shown in Figure 3.3. However, with the increasing popularity of computer graphics and physics simulation, GPU architectures are introduced specifically to target these tasks, where massive information can be processed in parallel using numerous processing units in the GPU, as shown in Figure 3.4, where the size of the cache is shrunk due to the relatively little information carried by each individual processing unit.

The hardware specification of NVIDIA's GPUs is shown in Figure 3.5. Each GPU contains a number of streaming multiprocessors (SM), each of which is a little cluster of several scalar processors (SP) executing in groups. Each SM carries its own shared memory, which is cached upon initial access. Each SM also carries its own constant and texture cache, and is read-only during the GPU processing phase. The memory and the caches are shared among the SPs in the same SM, and are not communicable to other SMs. Each of the SPs carries its own small number of register units, which is not communicable to other SPs. On top of the local memories and processing units, there is a global memory residing outside the GPU chip but still on the GPU card. Its access is slow and not cached, but it has a large capacity and is communicable to all the processing units.

NVIDIA’s Quadro FX5800 is used as the GPU for this thesis, with a Compute Capability of version 1.3. The hardware architecture is determined by the version of the Compute Capability, as shown in Table 3.1. To put the specific hardware architecture of the graphics card in numbers, FX5800 is clocked at 1.3 GHz, with 4 GB of global memory and 64 kB of constant memory. It has 30 SMs, each of which contains 8 SPs, 16 kB of registers, and16 kB of shared memory.

3.2 Compute Unified Device Architecture Environment

This section introduces CUDA as the interface to implement codes for the GPU. While the hardware architecture of the GPU is specified, it is up to the software architecture to control the GPU for parallel processing. The Compute Unified Device Architecture (CUDA) from NVIDIA is a platform which acts as an interface between the regular CPU computer language and the GPU hardware.

On the CUDA level, the GPU computing units are grouped in smaller partitions. The computation process is shown in Figure 3.6 as follows: the serial part of the program is first run on the CPU, denoted as the “host.” Then, the GPU, denoted as the “device,” is initiated to prepare for parallel computation. Each parallel computation, denoted as a kernel, will invoke a grid, which contains a maximum of 2 dimensions of blocks; each block contains a maximum of 3 dimensions of threads, where each thread is an individual processing unit analogous to a single scalar processor, and is capable of carrying out one instruction at a time. With enough threads declared in the kernel and carrying out the same instructions simultaneously, it is not hard to see that parallelism can be achieved through the numerous threads. To access these threads, a set of intrinsic variables are provided for the x, y and z coordinate of each block in the grid and each thread in a block. Various thread arrangement schemes can be achieved by combinations of these variables.

While the threads are used for massive parallel computation, the number of thread declarables is limited. Specifically, only a certain number of blocks are allowed to be declared in a grid, and a certain number of threads in a block. To put these into numbers for Quadro FX5800, the maximum dimension of a grid is 65535×65535 blocks, and the maximum dimension of a block is 512×512×64 threads, with a maximum of 512 threads per block.

Mapping CUDA's software architecture to the device's hardware architecture, one can see that since an individual thread is analogous to a scalar processor, the accessibility of different memory then varies by the different thread groupings, as shown in Figure 3.7. Each thread will be provided with a number of registers, which are only accessible through the respective thread and not the others. Similarly, a block of threads is analogous to a streaming multiprocessor, which is able to utilize its own private portion of the on-chip shared memory. As for the top level grid, all of the blocks and therefore threads are able to access the global memory without any constraints.

A key difference in the analogy between the hardware and the software architecture is how the GPUs actually carry out the computation. In the usual situation, there will be many more threads declared than the number of scalar processors, and many more blocks declared than the number of streaming multiprocessors. As a result, blocks and threads are brought onto the processing units and are computed sequentially. A streaming multiprocessor is able to take in one or more blocks at a time, providing that the total amount of shared memory used for the blocks is less than the total amount of shared memory available for one streaming multiprocessor. The threads are then divided into groups of 32 threads, called warps. The 8 scalar processors in the streaming multiprocessor execute the same instruction sets on one warp at a time, until all the warps have been taken care of.

The general flow of the computation process in CUDA is as follows: once the program reaches the parallel portion, CUDA commands will be used to initiate the device. Next, the data on the host memory which is to be parallel computed are sent to the different memory locations in the device using CUDA malloc and memory copy functions. Then, CUDA will use the various thread arrangement schemes to distribute the threads to the streaming multiprocessors and the scalar processors for parallel computation. The result of the computation is then transferred back from the device memory to the host memory, completing the parallel computation portion of the program.

3.3 PO-SBR Implementation with CUDA

This section lays out the proposed method used for the GPU implementation of the PO-SBR method. To implement the PO-SBR method on the GPU using CUDA, the parallel nature of the method needs to be utilized. From Chapter 2, it can be seen that nearly all steps of the method involve separate rays of Geometric Optics and Physical Optics computed independently. While the ray tracing part of the SBR could be parallelized using NVIDIA's newly developed OptiX, modern ray tracing programs already have many built-in accelerators which dramatically speed up the ray tracing process on the CPU. Moreover, since the electromagnetic process of the method is much more computationally expensive than the ray tracing process, the parallelization of ray tracing is not considered in the scope of this thesis.

In the electromagnetic process of the method, three chunks of wave propagation processes can be identified for parallelization: from the source to all hit points, from all hit points to all observation points, and from the source to all observation points. Multiple frequencies also need to be considered for parallelization.

The PO-SBR computation starts up with using a modified version of the image rendering program Physically Based Rendering Technique (PBRT) for ray tracing through the computation domain. The geometry of the target object is meshed into a facet format file, and is fed into the ray tracer along with the source coordinates. The burst method is used to launch initial rays from the source in equal degree of spacing towards all directions. The ray tracer then returns the ray depth, the hit point coordinates, the normal vector $\hat{n}$, and the propagation constant $\hat{k}$ upon completion. The obtained hit points are stored in the host memory for the later electromagnetic computation.

At this point, CUDA commands are used to initialize the device. The data on the host memory, which is to be parallelly computed, are sent to the different memory locations in the device using CUDA functions. The large amount of data for the hit points are first sent to the global memory. Smaller constant data, such as the number of hit points, the number of observation points, the number of frequencies, spacing between the rays in radiance, and propagation constants, are sent to the cached constant memory for repeating access.

Before invoking the kernels for parallel computation, threads must be allocated in a way that utilizes the hardware and software architectures of the device. First, each block calculates one or more hit or observation points, depending on the amount of shared memory available per streaming multiprocessor. Also, it is known that the equations for computing different frequencies are exactly the same. Therefore, different frequencies on the same hit or observation point should be placed together in the same block, with common information about the points stored in the shared memory for fast cached access. A total of three kernels are invoked: from source to all hit points, from all hit points to all observation points, and from source to all observation points. The design strategy for each kernel is as follows: first, the total number of threads declared is distributed evenly to each streaming multiprocessor. Each streaming multiprocessor will then handle one block at a time. Depending on the available amount of the shared memory per streaming multiprocessor, each block will contain a fixed number of maximum threads per block. It was found that with the amount of shared memory used per thread, the maximum number of threads per block will never exceed 512, which is the intrinsic

CUDA limitation for the size of a block.

3.3.1 Radiation from Source to All Hit Points

Radiation from source to all hit points is the first electromagnetic process of the PO-SBR method. The incident field from the source needs to be computed at every initial ray hit point, and then its phase updated for all subsequent ray bounces in order to find the surface currents. This is when complete parallelization is not possible, since the electromagnetic information at the subsequent hit points depends on the previous hit points. Therefore, a thread is declared for each of the N_{hit} ray hit points, as shown in Figure 3.8. The threads for the initial hit points are computed first; then those for the subsequent hit points are computed sequentially on the GPU using the computed field values at the previous hit points. The following is carried out in this kernel:

- A 2D grid of blocks is allocated (2D is to avoid grid dimensional limit).
- Each block in the grid computes the incident field at L ray hit points for all N_{freq} frequencies.
- Loops over all hit points and all frequencies.
- The exact number of ray hit points per block is based on the amount of shared memory available per block.

- For a single frequency simulation this is around 110 hits per block. More frequencies require more memory and therefore allow fewer ray hits per block.
- Each thread within the block (maximum of 512) computes the incident field for one combination of frequency and hit point.
- The field values for all frequencies and all hit points are stored in device global memory to be used by the next step.

3.3.2 Radiation from All Hit Points to All Observation Points

Radiation from all N_{hit} hit points to all N_{obs} observation points is the Physical Optics part of the PO-SBR method. The induced surface current at each hit point needs to be re-radiated towards every observation point in order to find its contribution towards the total scattered field at the respective observation points. A thread is declared for each observation point, as shown in Figure 3.9. This is the process which takes up the majority of the computation time, since the process loops through all hit points, observation points, and frequencies. Kernels of different hit points are invoked and looped to compute the fields at all observation points due to the surface current of each hit point. The following is carried out in each kernel:

- A 2D grid of blocks is allocated (2D is to avoid grid dimensional limit).

- Each block in the grid computes the scattered field at M observation points for all N_{freq} frequencies.
- Loops over all hit points, all observation points, and all frequencies.
- The exact number of observation points per block is based on the amount of shared memory available per block.
- For a single frequency simulation this is around 290 observation points per block. More frequencies require more memory and therefore allow fewer observation points per block.
- Each thread within the block (maximum of 512) computes the induced current and scattered field for one combination of frequency and observation point.
- This requires the host calling kernels sequentially for each hit point, looping through each hit point to accumulate the field values at all observation points in device global memory to be used by the next step.

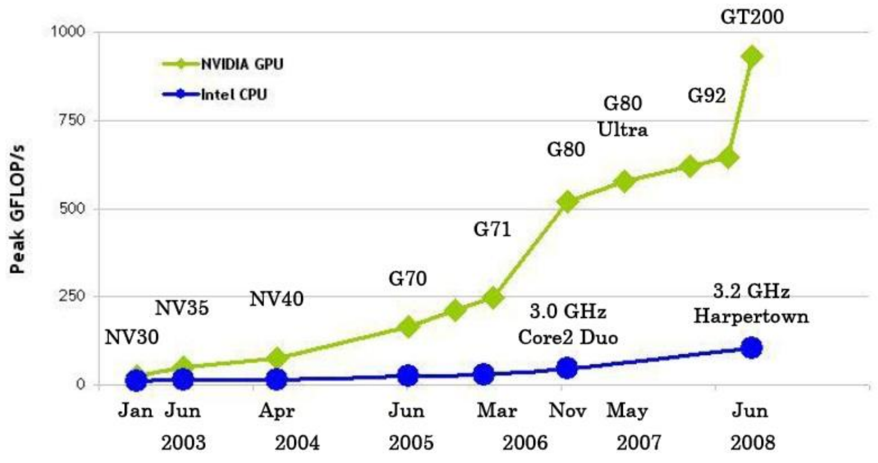
3.3.3 Radiation from Source to All Observation Points

Radiation from source to all observation points is the last electromagnetic process of the PO-SBR method. The incident field from the source needs to be computed at every observation point in order to compute the contribution of the direct line-of-sight field to the total field at the N_{obs} observation points. The equations involved are almost identical to the finding of initial hit points for radiation from source to all hit points. A thread is declared for each observation point, as shown in Figure 3.10. The following is carried out in this kernel:

- A 2D grid of blocks is allocated (2D is to avoid grid dimensional limit).
- Each block in the grid computes the incident field at M observation points for all N_{freq} frequencies.
- Loops over all observation points and all frequencies.
- The exact number of observation points per block is based on the amount of shared memory available per block.
- For a single frequency simulation this is around 290 observation points per block. More frequencies require more memory and therefore allow fewer observation points per block.
- Each thread within the block (maximum of 512) computes the induced current and incident field for one combination of frequency and observation point.
- The field values for all frequencies and all observation points are added to the scattered field values in device global memory before transferring the result back to the host machine for visualization.

Notice that the fields at the observation points are accumulated at the same place in the global memory for the hit point loops and for the line-of-sight field. Only the final accumulated fields at the observation points need to be transferred from the device back to the host. This is especially important for CUDA implementation, since memory transfer speed is a potential bottleneck. Therefore the amount of memory transferred needs to be minimized.

3.4 Figures and Table



GT200 = GeForce GTX 280	G71 = GeForce 7900 GTX	NV35 = GeForce FX 5950 Ultra
G92 = GeForce 9800 GTX	G70 = GeForce 7800 GTX	NV30 = GeForce FX 5800
G80 = GeForce 8800 GTX	NV40 = GeForce 6800 Ultra	

Figure 3.1 Floating point operations per second (FLOPS), CPU vs. GPU.

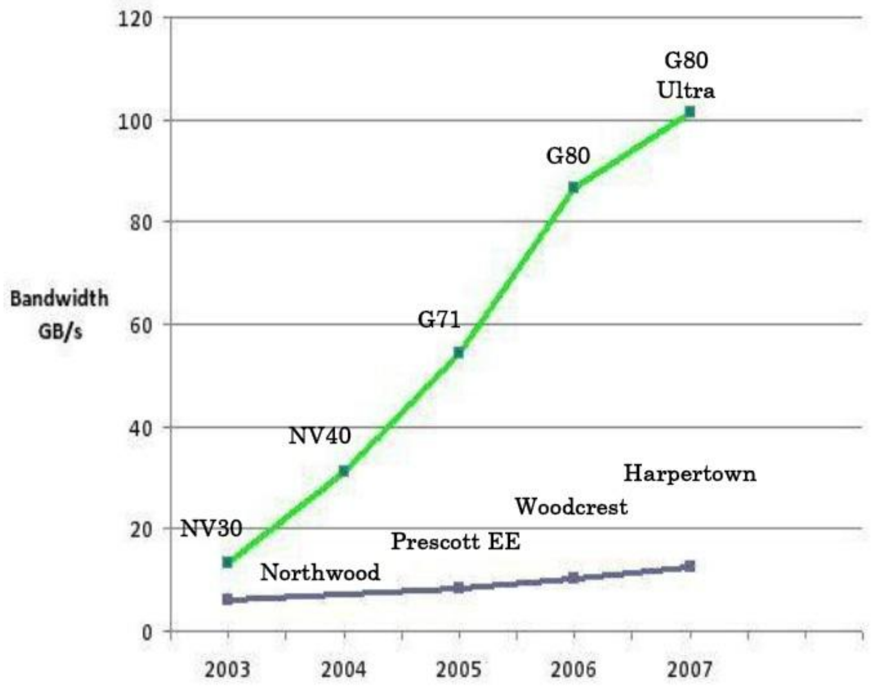


Figure 3.2 Memory bandwidth, CPU vs. GPU.

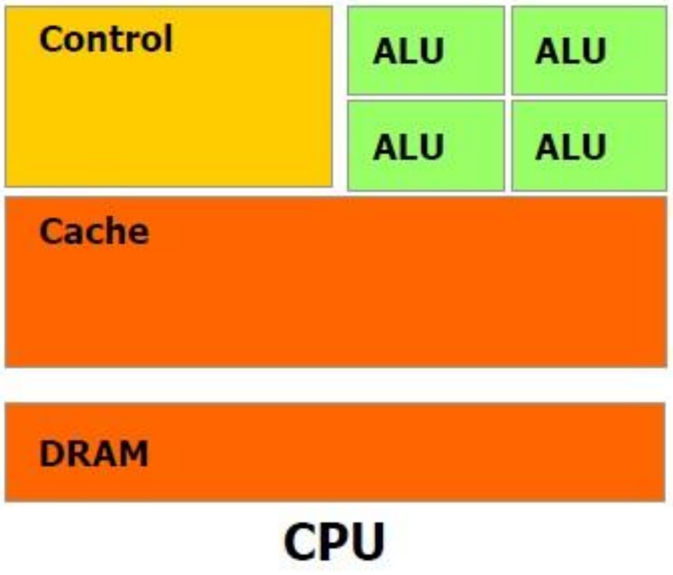


Figure 3.3 CPU hardware architecture.

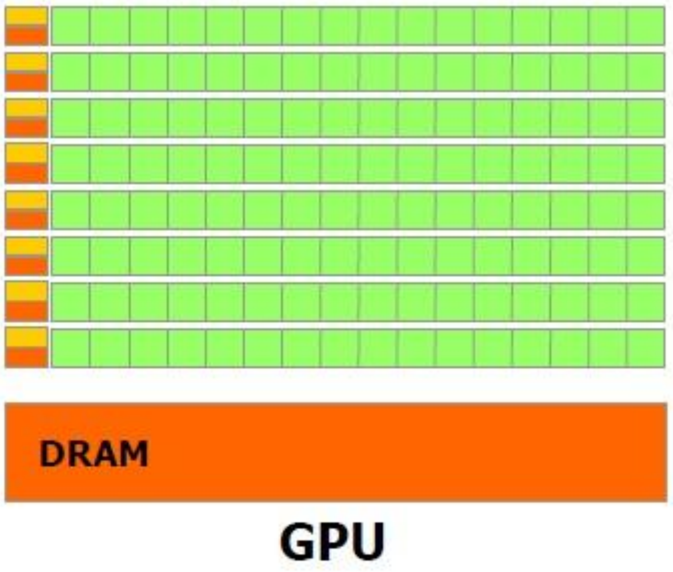


Figure 3.4 GPU hardware architecture.

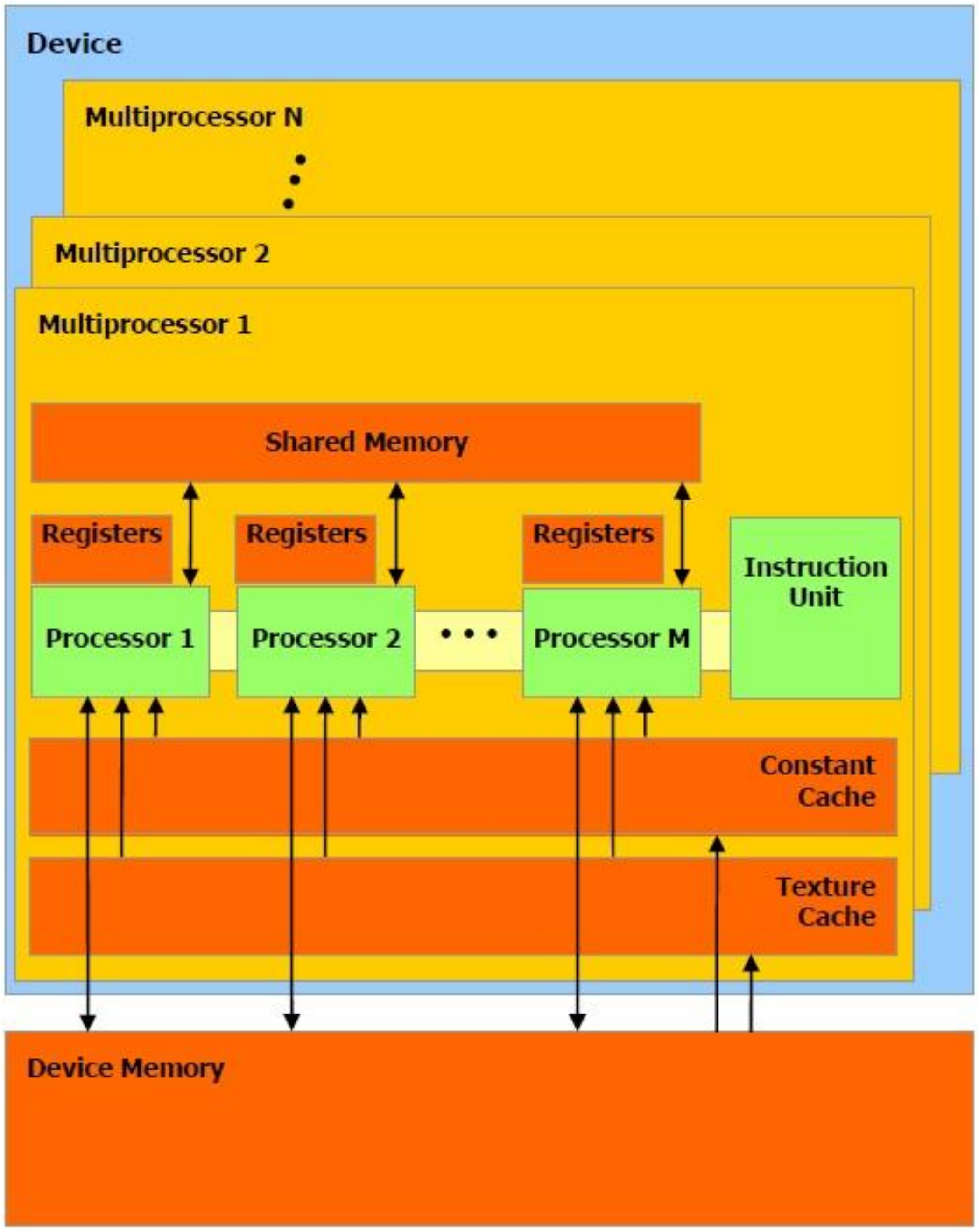


Figure 3.5 Detailed NVIDIA GPU architecture.

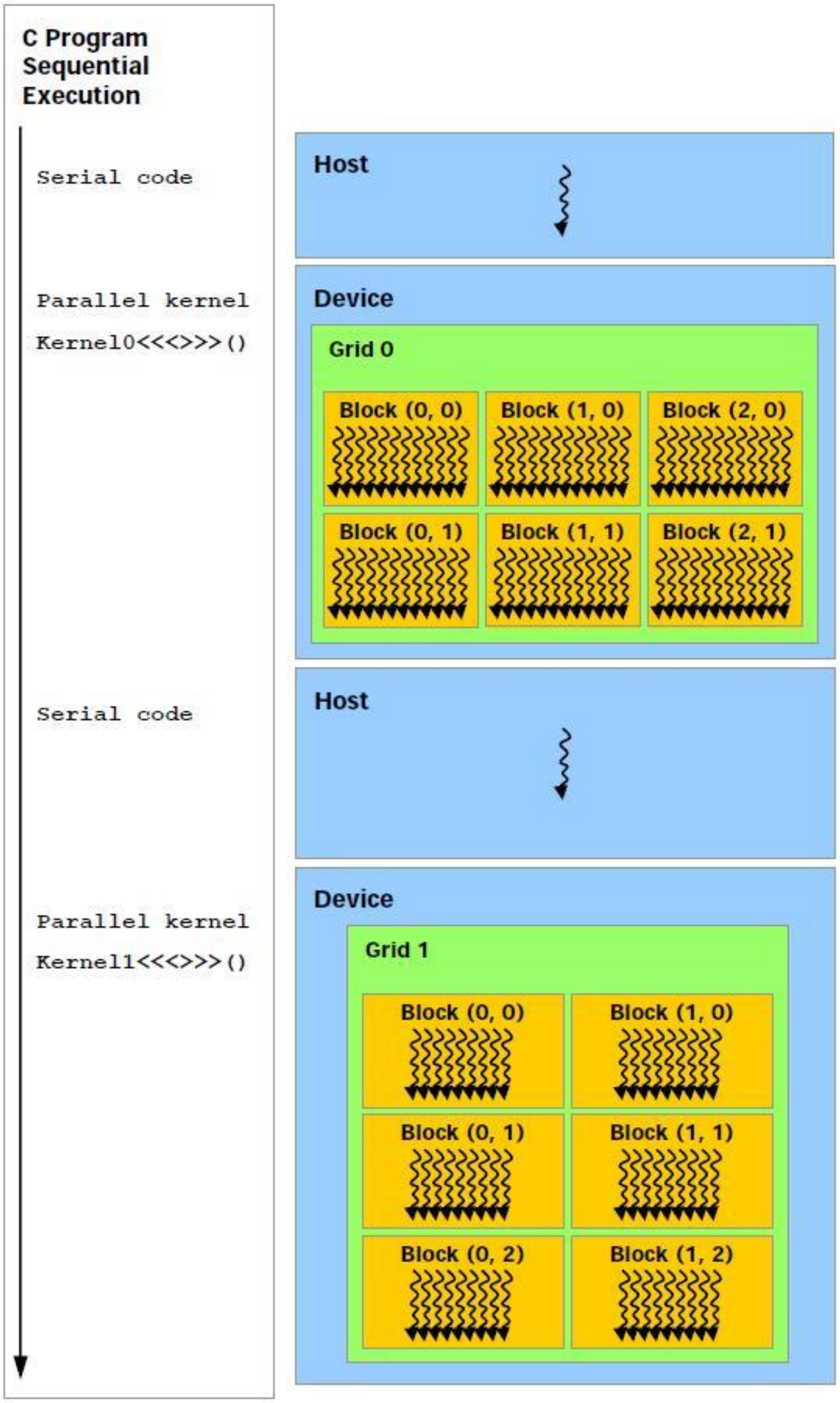


Figure 3.6 The computation process flow chart.

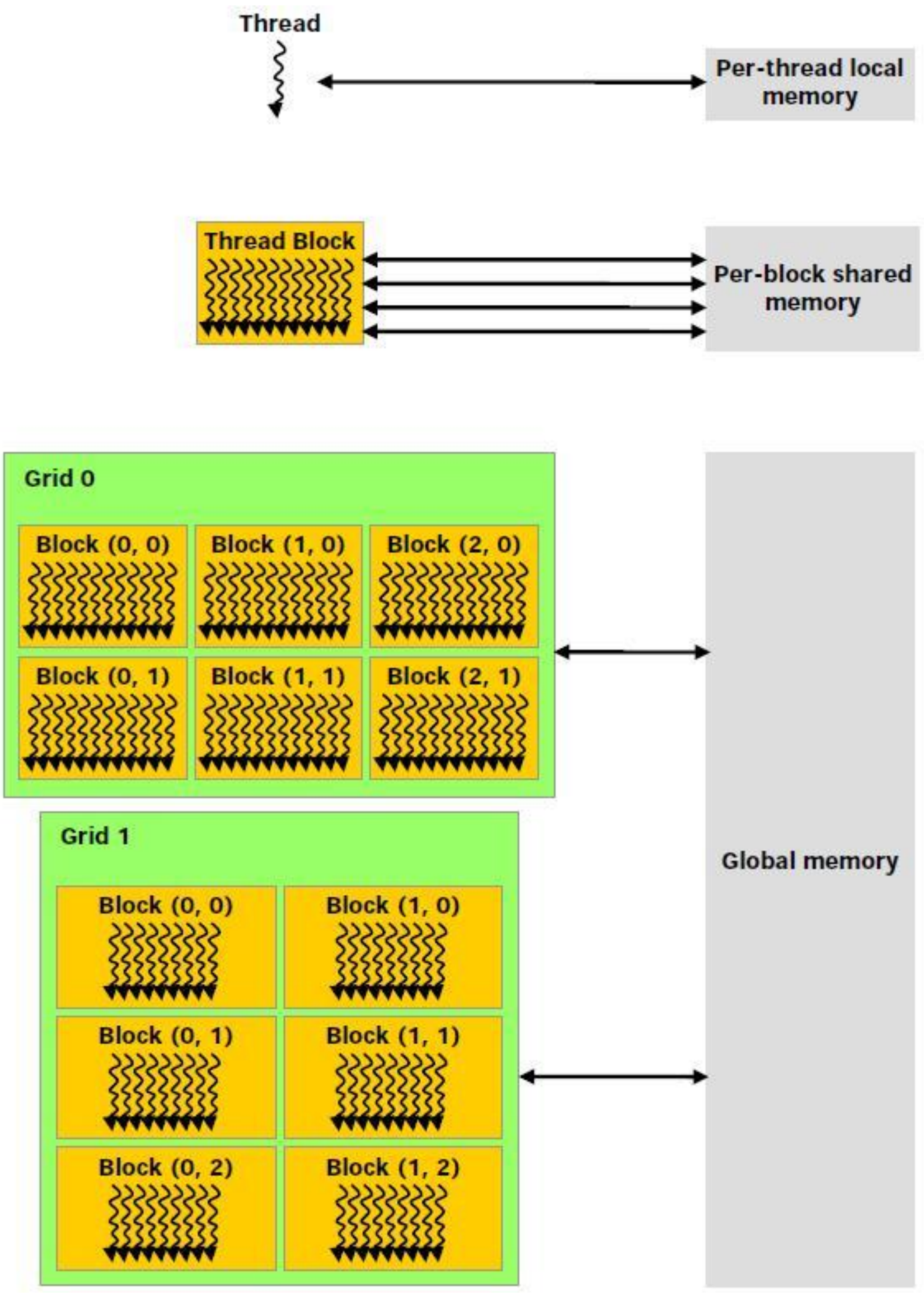


Figure 3.7 Memory accessibility of the thread divisions.

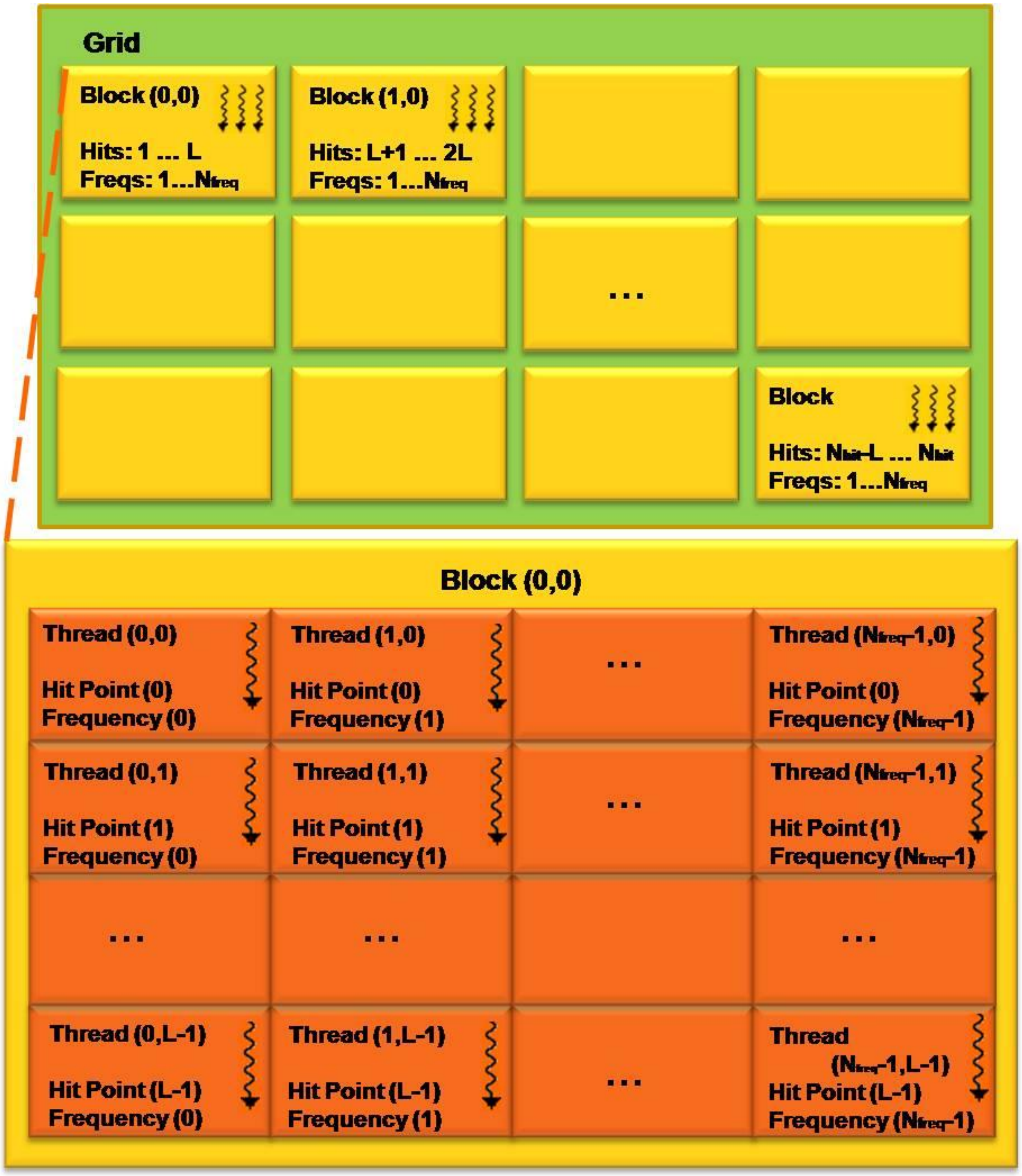


Figure 3.8 Thread declaration of kernel “Radiation from source to all hit points.”

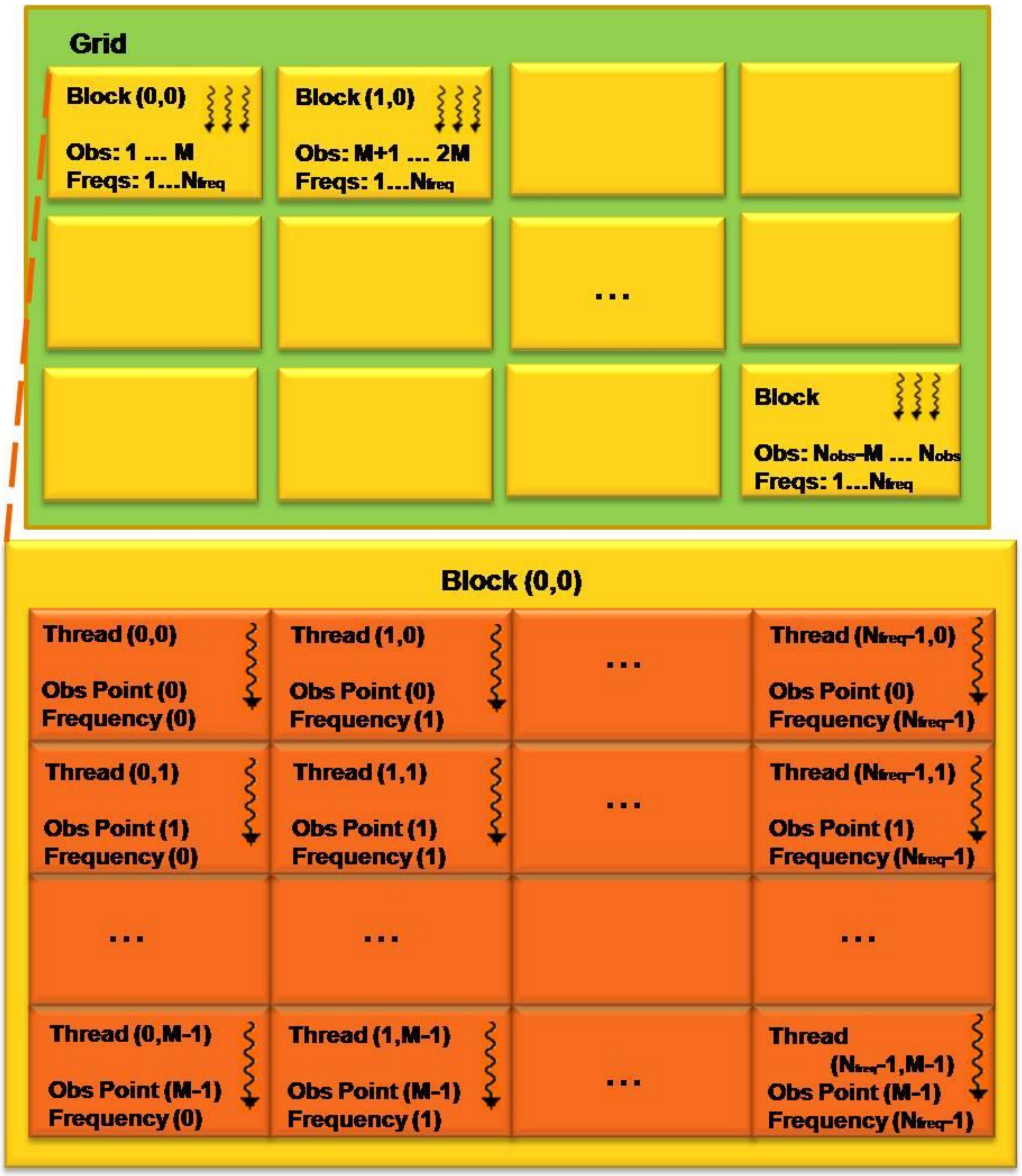


Figure 3.9 Thread declaration of kernel “Radiation from source to all hit points.”

.

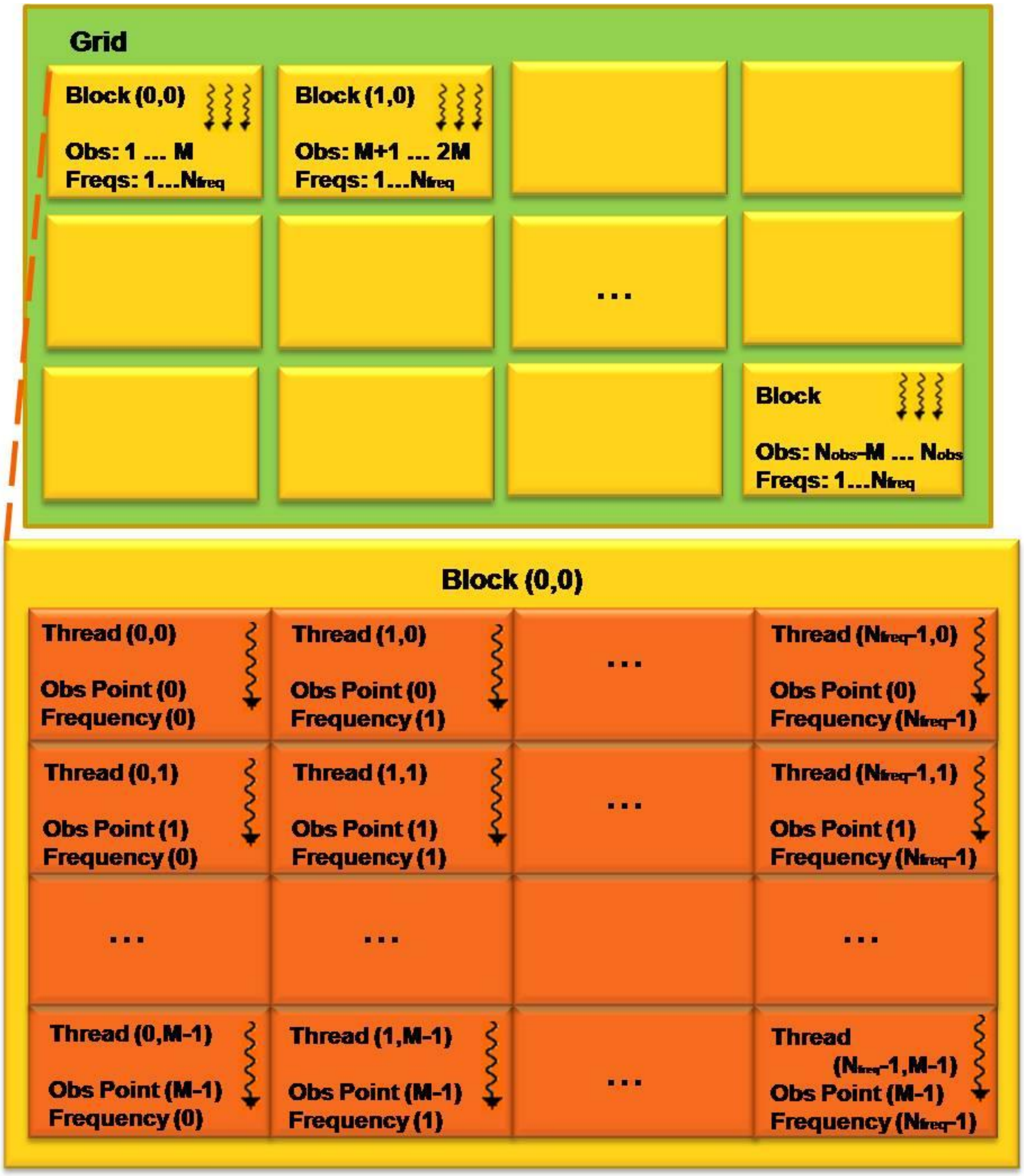
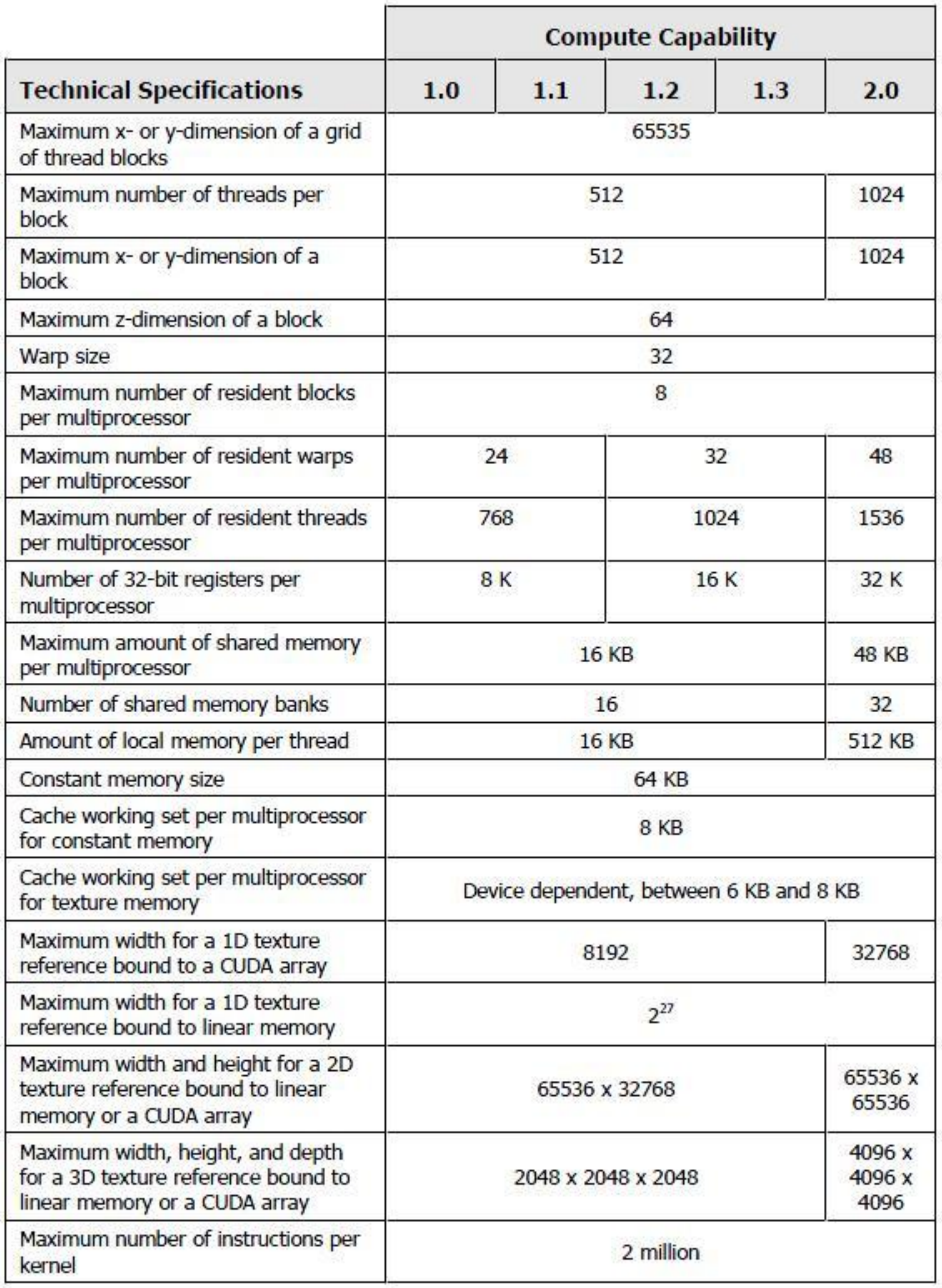


Figure 3.10 Thread declaration of kernel “Radiation from source to all hit points.”

Table 3.1 Compute Capability version and the corresponding hardware specifications.

.



	Compute Capability				
Technical Specifications	**1.0**	**1.1**	**1.2**	**1.3**	**2.0**
Maximum x- or y-dimension of a grid of thread blocks	65535				
Maximum number of threads per block	512				1024
Maximum x- or y-dimension of a block	512				1024
Maximum z-dimension of a block	64				
Warp size	32				
Maximum number of resident blocks per multiprocessor	8				
Maximum number of resident warps per multiprocessor	24		32		48
Maximum number of resident threads per multiprocessor	768		1024		1536
Number of 32-bit registers per multiprocessor	8 K		16 K		32 K
Maximum amount of shared memory per multiprocessor	16 KB				48 KB
Number of shared memory banks	16				32
Amount of local memory per thread	16 KB				512 KB
Constant memory size	64 KB				
Cache working set per multiprocessor for constant memory	8 KB				
Cache working set per multiprocessor for texture memory	Device dependent, between 6 KB and 8 KB				
Maximum width for a 1D texture reference bound to a CUDA array	8192				32768
Maximum width for a 1D texture reference bound to linear memory	2^{27}				
Maximum width and height for a 2D texture reference bound to linear memory or a CUDA array	65536 x 32768				65536 x 65536
Maximum width, height, and depth for a 3D texture reference bound to linear memory or a CUDA array	2048 x 2048 x 2048				4096 x 4096 x 4096
Maximum number of instructions per kernel	2 million				

CHAPTER 4
NUMERICAL RESULT AND COMPARISON ANALYSIS

This chapter provides examples of the original and the accelerated PO-SBR method. Section 4.1 validates the version of the PO-SBR method described in this thesis by comparing the computational result with a known analytical result. Section 4.2 gives an example of the scattering field computation. Section 4.3 gives an example of the radiation pattern computation.

4.1 Result Validation

This section validates the version of PO-SBR described in this thesis. The PO-SBR method is asymptotic and thus lacks precision compared to the full wave methods. Moreover, different approximations and assumptions can be freely made to construct different versions of the PO-SBR method. Since it is difficult for asymptotic methods to obtain a very precise solution of the fields, result validation will be made, based on the similarity rather than the numerical error, of the resulting field strengths between analytical and numerical solutions. To obtain an analytical solution of the field of an antenna in presence of an object, simple geometry is employed.

4.1.1 Hertzian Dipole with Infinite Plate

The PO-SBR method described in this thesis is validated through the comparison of an existing analytical solution, where a hertzian dipole of 10 GHz, located at $\left(0, 0, 0.1\ \text{m}\right)$, is placed on top of an infinite PEC plate on the x-y plane, as shown in Figure 4.1. The field is calculated at a distance of 10 m away from the plate. The magnitude of the $\hat{z}$ component of the electric field is plotted for validation, with an analytical equation of

$$E_\theta = -\frac{j\eta kIl}{2\pi r}\sin\theta\cos(kh\cos\theta)e^{-jkr}. \tag{4.1}$$

Figure 4.2 shows the comparison between the analytical and the numerical radiation pattern of the $\rho = 90^\circ$ plane at $r = 5\ \text{m}$ away from the origin. The resulting analytical field strength of x-y plane at $z = 10\ \text{m}$ is shown in Figure 4.3, and the numerical field strength is shown in Figure 4.4. Figures 4.5 and 4.6 show the analytical and numerical field strength of the $\rho = 0^\circ$ plane. Notice that although the details of the field strength differ slightly between the two plots, the general shape of the field strength remains the same, and thus the PO-SBR method described in this thesis is validated.

4.2 Scattering Field Problems

This section gives an example of the scattering field computation. The scattering field problems are used to obtain the field scattering signatures of target objects. The solution shows the field distribution and distortion of the incident field in presence of an object.

In the following scattering field example, three parameters of the effect of computational speedup are examined and analyzed. The parameters are the number of observation points, the number of hit points, and the number of frequencies.

4.2.1 Scattering Field: 1982 Porsche

Using CUDA for acceleration, the speedup in computation timing of the PO-SBR method is documented and compared. Following is a scattering field problem where a hertzian dipole antenna radiates towards a 1982 Porsche from a distance of 4.5 meters away from the head of the car at 10 GHz, as shown in Figure 4.7. The ray tracing with an angular spacing of $0.1°$ resulted in a total of 18,184 hit points. The scattered field is computed at $z = 0$ m around the car, and is plotted in dB and shown in Figure 4.8 with a varying number of 100, 2,500, 10,000, and 250,000 observation points.

The computational time taken for the ray tracing of PO-SBR on CPU is 4.5 seconds at an angular spacing of $0.10°$ with 18,184 hit points. The computation time and the speedup taken for the electromagnetic aspects of the PO-SBR method on both CPU and GPU are shown in Table 4.1, with a varying number of 100, 2,500, 10,000, and 250,000 observation points. By varying the number of observation points, the looping sizes increase in the processes of "from all hit points to all observation points" and "from source to all observation points." As discussed in Section 3.3, since the majority of the computation time is taken in the process of "from all hit points to all observation points," inspection of the computational timing on this process is sufficient. Since the increasing number of observation points in the process is parallelized in the GPU, it can be seen that as the number of observation points increases, the speedup increases.

Table 4.2 shows the computation time and the speedup taken with a varying number of 7,155, 10,719, 18,184, 36,988, and 112,807 hit points. By varying the number of hit points, looping sizes increase in the processes of “from source to all hit points” and “from all hit points to all observation points.” And again only the process of “from all hit points to all observation points” is inspected. Since the hit points are looped in GPU with separate kernels, no sufficient speedup is found.

Table 4.3 shows the computation time and the speedup taken with a varying number of 1, 2, 3, 4, and 5 frequencies. No clear pattern is shown by varying the number of frequencies. The reason might be caused by the increasing usage of shared memory with an increasing number of frequencies. The increasing amount of shared memory required per thread decreases the number of threads per block, and thus decreases the overall speedup.

4.3 Radiation Pattern Problems

This section gives an example of the radiation pattern computation. The radiation pattern problems are used to map out the field distribution of antenna sources emitted in their residing environment. Usually, antennas are designed with a certain desired standalone radiation pattern. However, antennas are used in different environments and mounted onto a platform, i.e. the antenna tower and car roof, which could alter the radiation pattern of the antennas. The study of radiation pattern using PO-SBR can provide a rough but quick view on how a given structure affects the radiation pattern of the standalone antennas.

In the following radiation pattern example, three parameters of the effect of computational speedup are examined and analyzed. The parameters are the number of observation points, the number of hit points, and the number of frequencies.

4.3.1 Radiation Pattern: 1964 Thunderbird

Using CUDA for acceleration, the speedup in computation time of the PO-SBR method is documented and compared. Figure 4.9 shows a radiation pattern problem where a hertzian dipole antenna is mounted on top of a 1964 Thunderbird at 10 GHz. The ray tracing with an angular spacing of 5° resulted in a total of 28,321 hit points. The radiation pattern is computed at $z = 0$ m around the car, and is plotted in dB and shown in Figure 4.10 with a varying number of 100, 2,500, 10,000, and 250,000 observation points.

The computation time and the speedup taken for the electromagnetic aspects of PO-SBR on both CPU and GPU are shown in Table 4.4 with a varying number of 100, 2,500, 10,000, and 250,000 observation points. It can be seen that the trend of speedup is similar to the above scattered field problem described in Section 4.2. Since the increasing number of observation points in the process is parallelized in the GPU, it can be seen that as the number of observation points increases, the speedup increases.

Table 4.5 shows the computation time and the speedup taken with a varying number of 13,952, 19,699, 28,321, 44,314, and 78,751 hit points. It can be seen again that the increasing number of hit points does not increase the speedup, since different hit points are looped by declaring different kernels.

Table 4.6 shows the computation time and the speedup taken with a varying number of 1, 2, 3, 4, and 5 frequencies. Again no clear pattern is shown by varying the number of frequencies. The reason might be that the increasing amount of shared memory required per thread decreases the number of threads per block, and thus decreases the overall speedup.

4.4 Figures and Tables

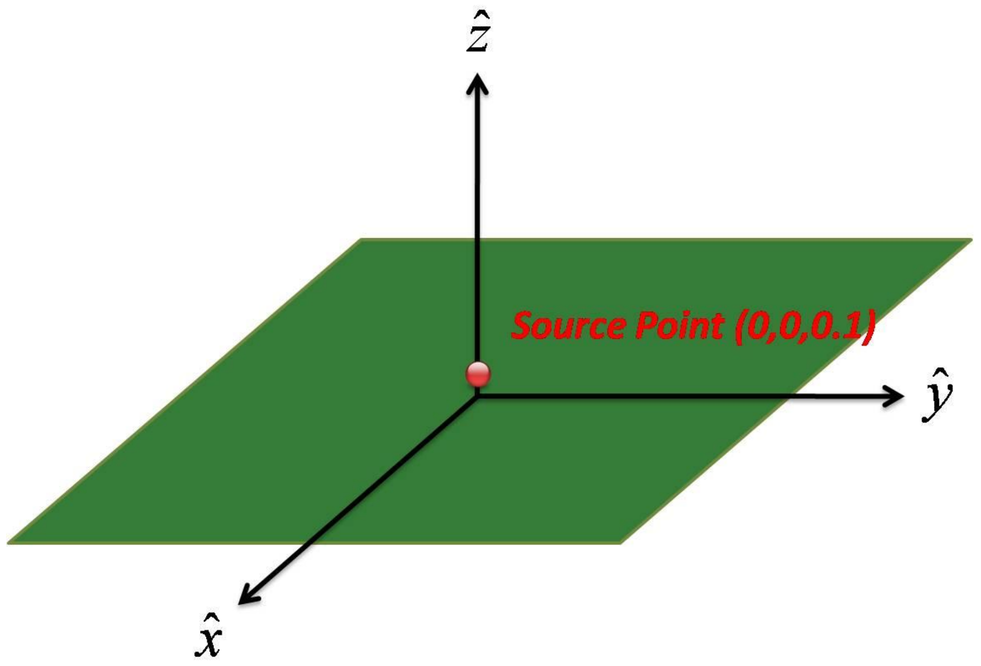


Figure 4.1 Infinite plate with hertzian dipole source located at (0,0,0.1 m).

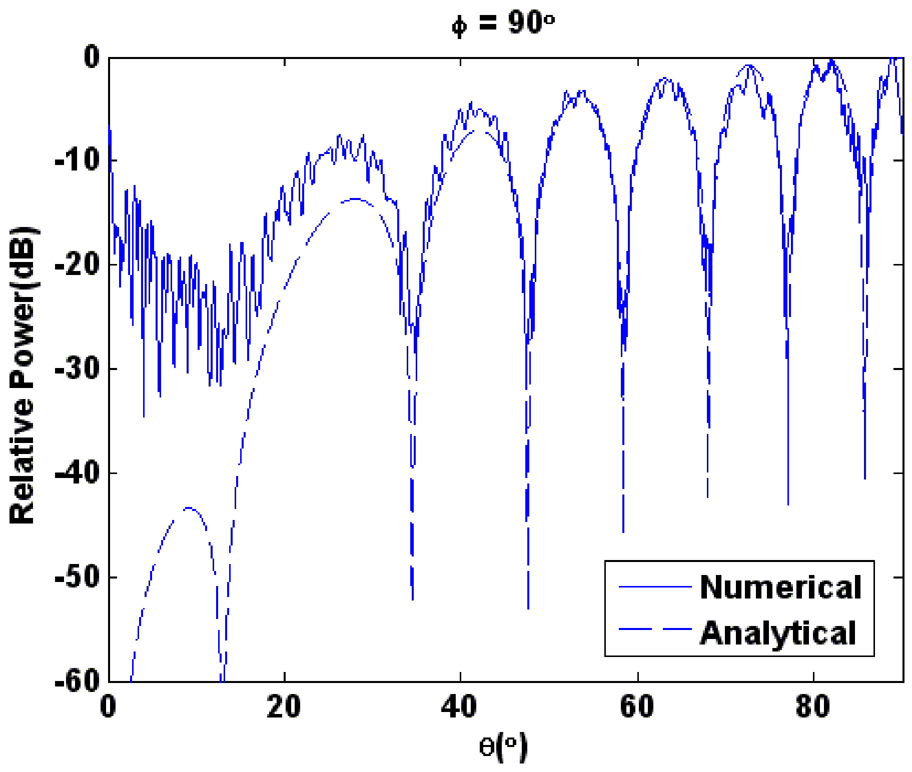


Figure 4.2 Y-Z plane radiation pattern of hertzian dipole in presence of an infinite plate.

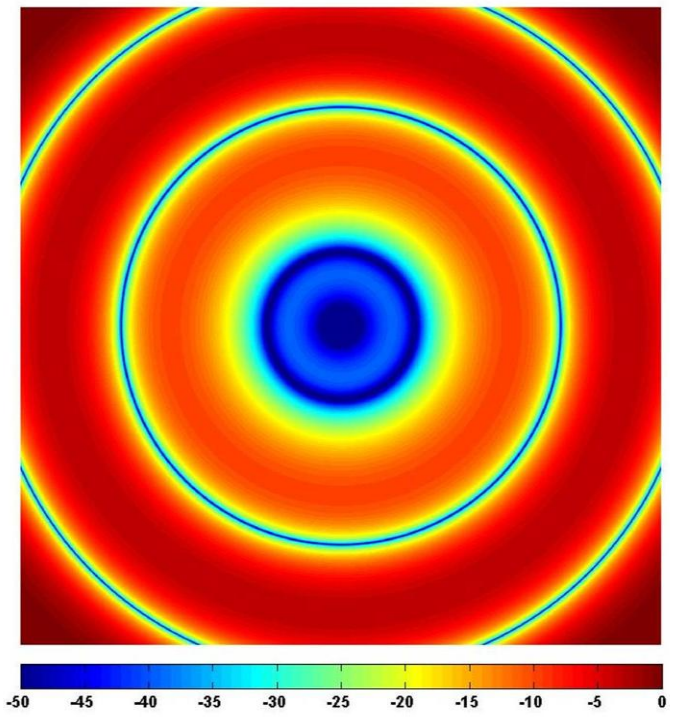


Figure 4.3 Analytical X-Y plane radiation pattern of hertzian dipole in presence of an infinite plate.

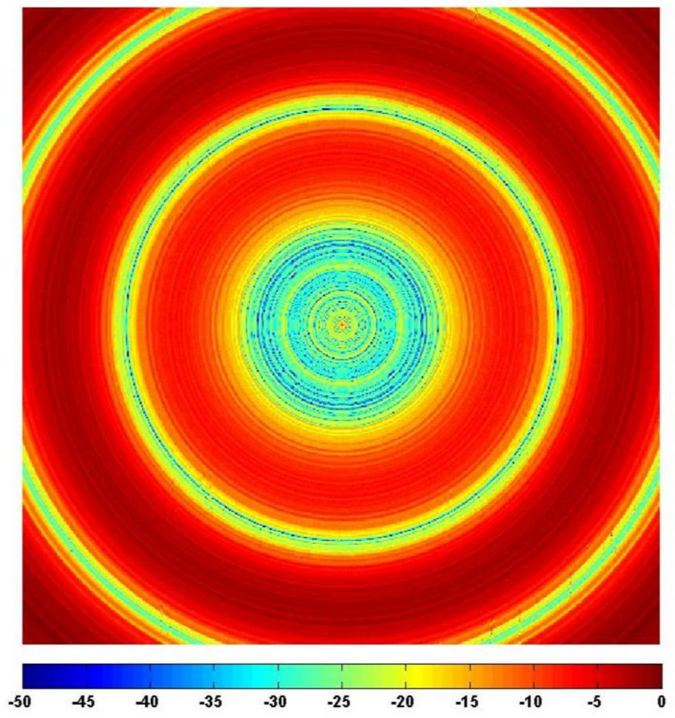


Figure 4.4 Numerical X-Y plane radiation pattern of hertzian dipole in presence of an infinite plate.

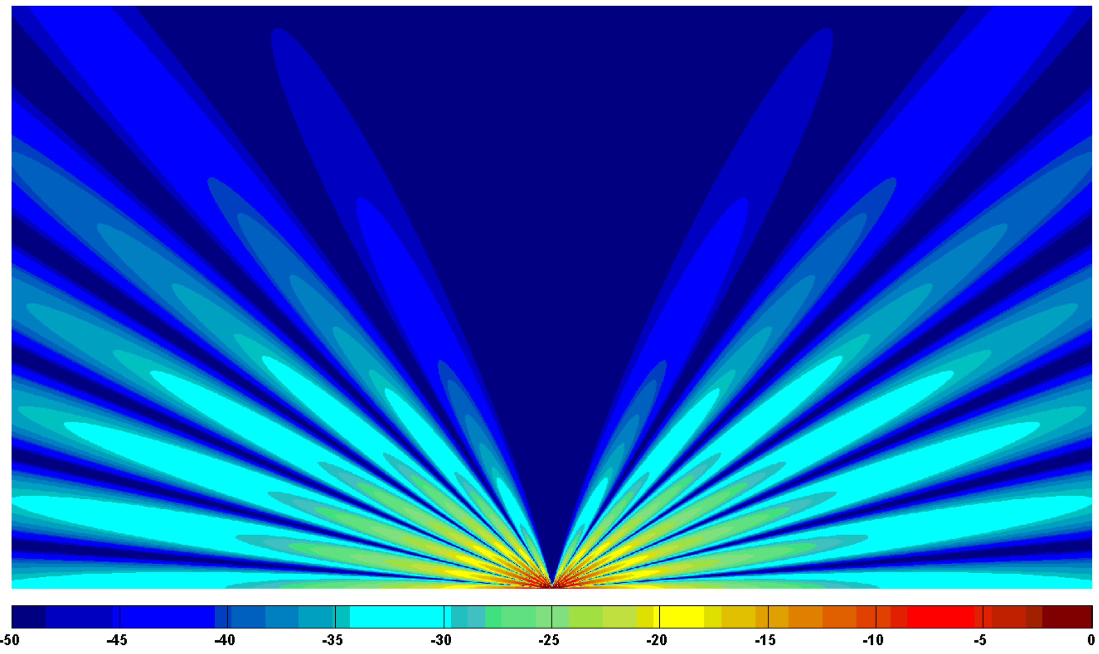


Figure 4.5 Analytical X-Z plane radiation pattern of hertzian dipole in presence of an infinite plate.

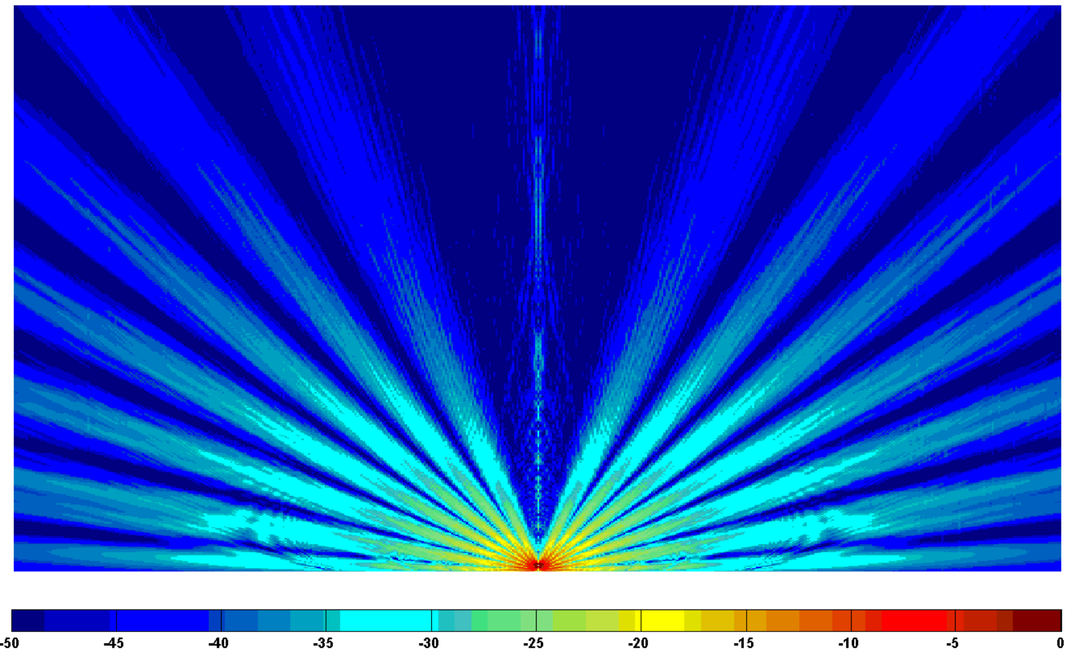


Figure 4.6 Numerical X-Z plane radiation pattern of hertzian dipole in presence of an infinite plate.

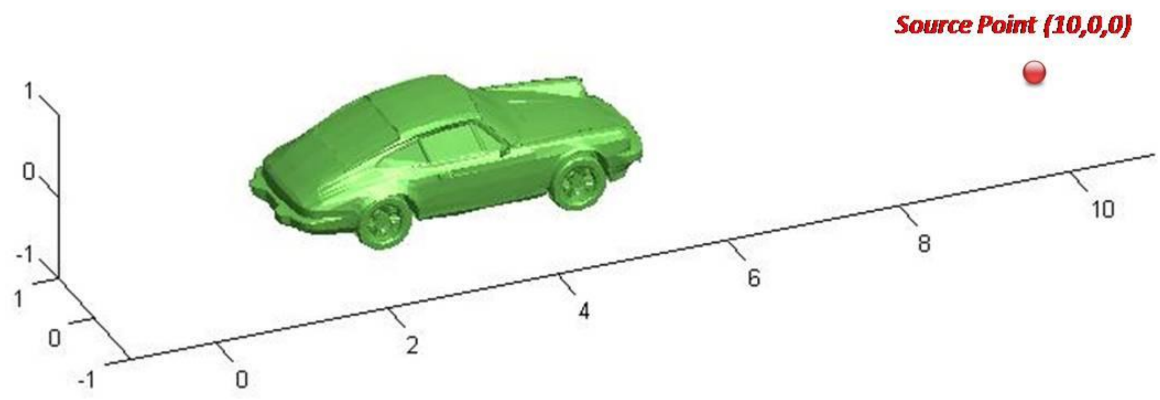
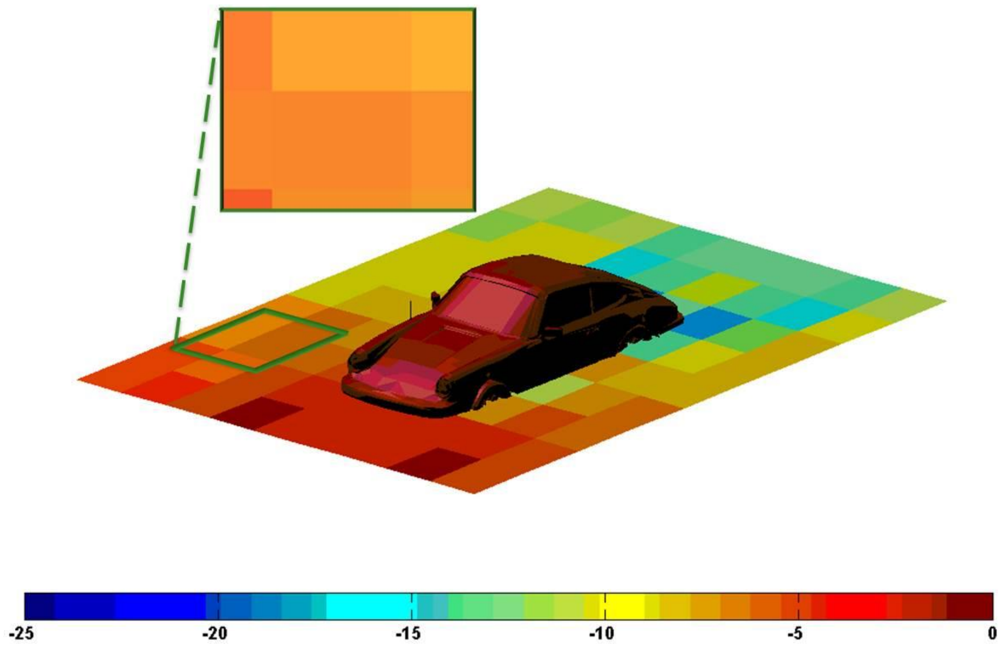
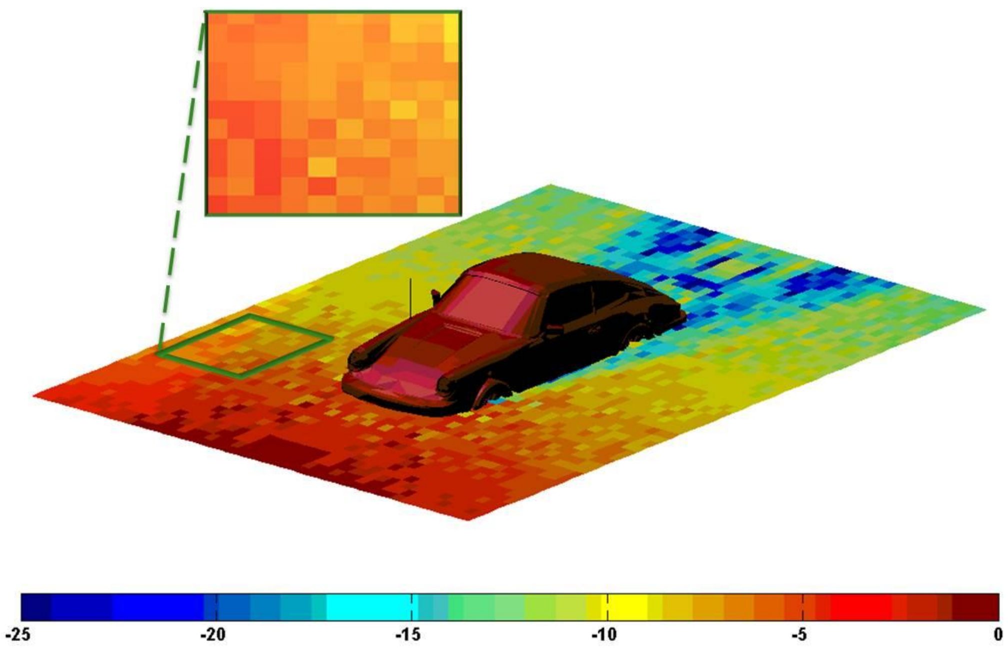


Figure 4.7 1982 Porsche with hertzian dipole source located at (10 m,0,0).

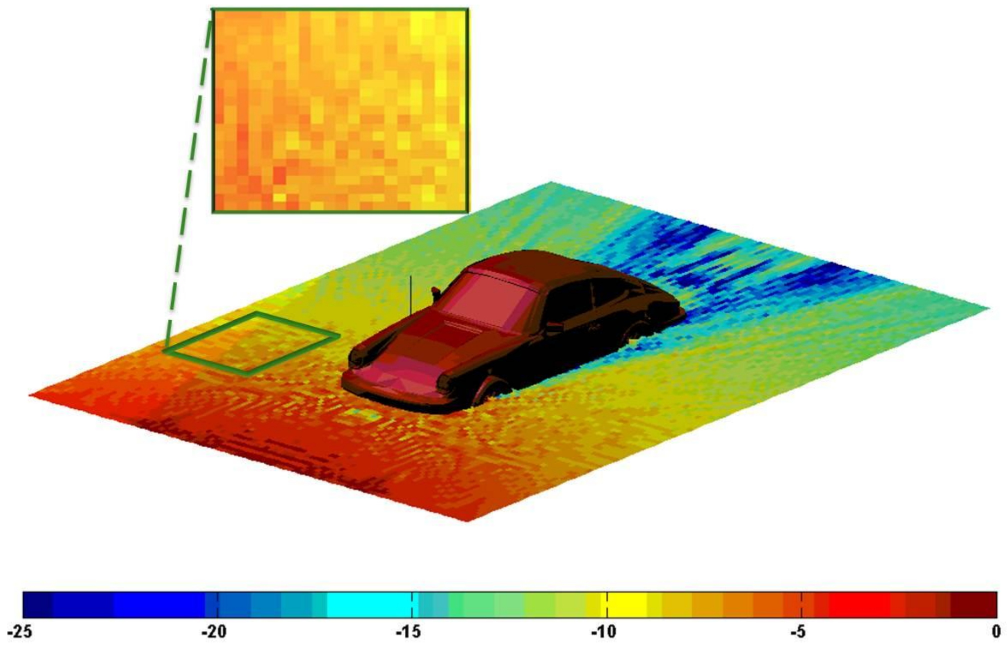


(a)

Figure 4.8 Scattering field with a varying observation points: (a) 100 (b) 2,500 (c) 10,000 and (d) 250,000 observation points.

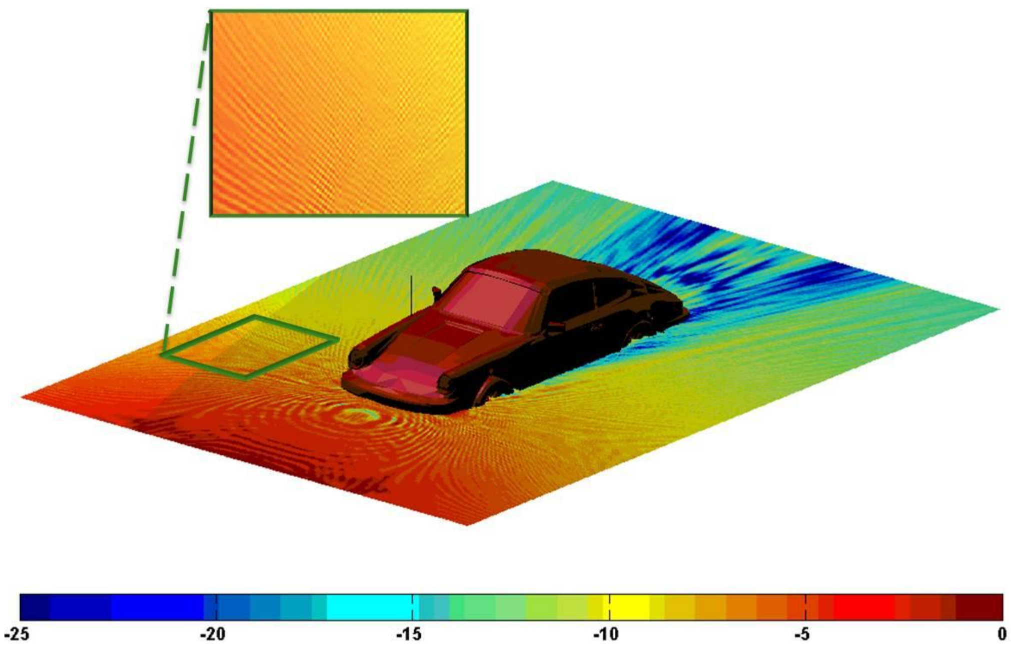


(b)



(c)

Figure 4.8 (continued)



(d)

Figure 4.8 (continued)

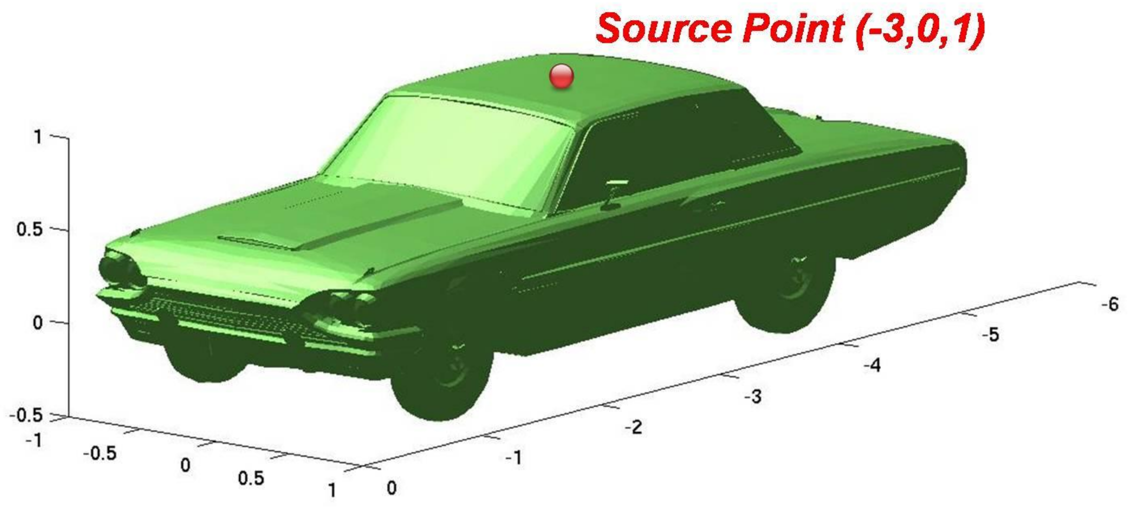
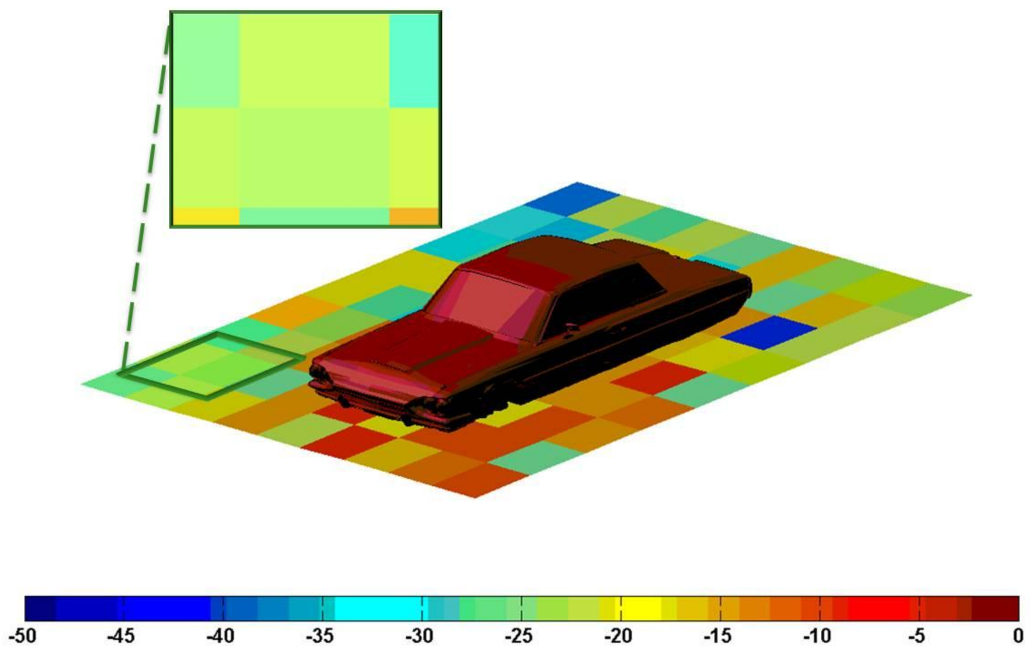
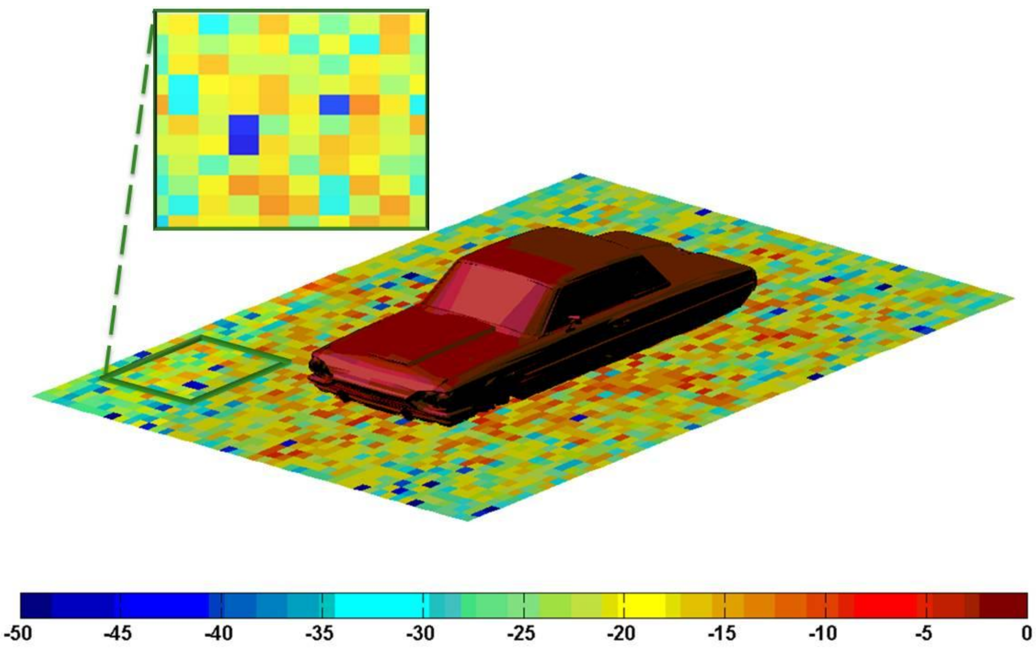


Figure 4.9 1964 Thunderbird with hertzian dipole source located at (-3 m,0,1 m).

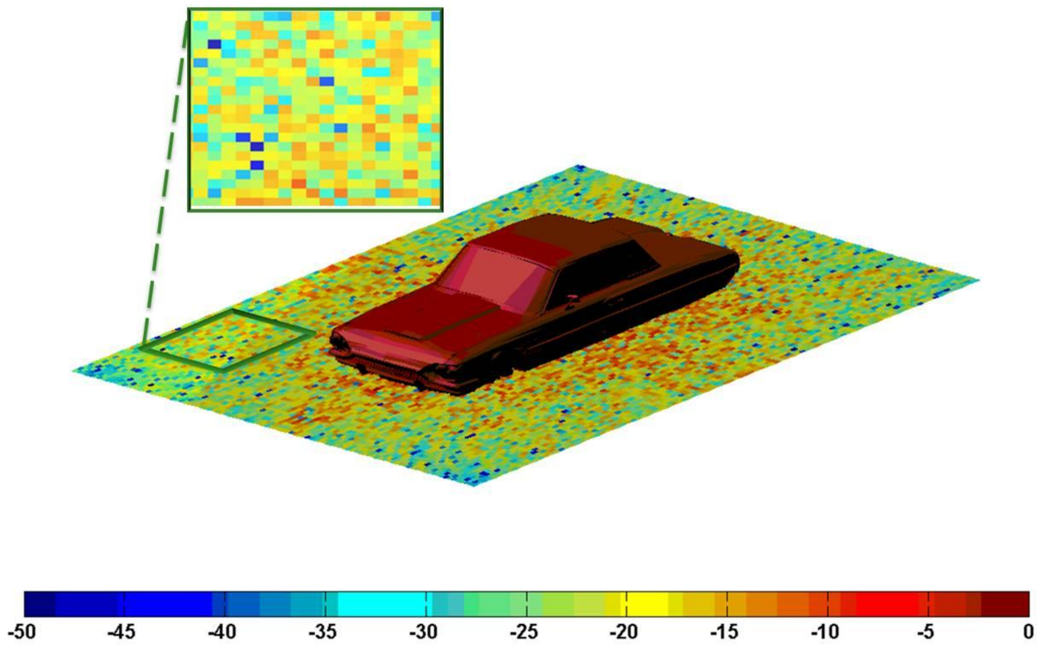


(a)

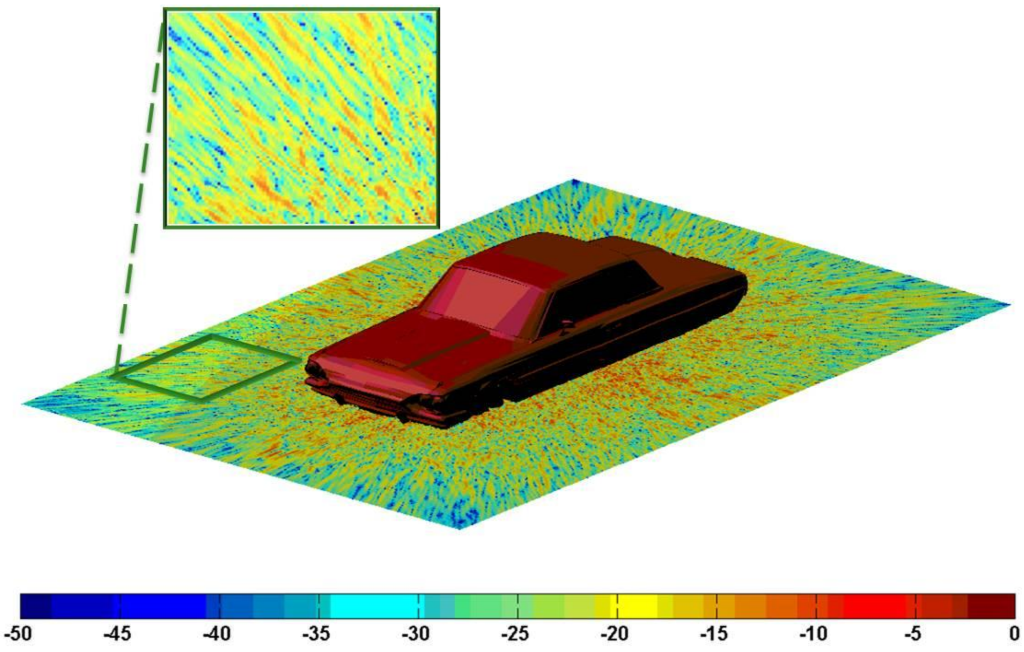


(b)

Figure 4.10 Radiation pattern with a varying observation points: (a) 100 (b) 2,500 (c) 10,000 and (d) 250,000 observation points.



(c)



(d)

Figure 4.10 (continued).

Table 4.1 GPU vs. CPU computational time comparison of electromagnetic aspect of PO-SBR, with 18,184 hit points, 1 frequency, and varying observation points.

Number of Observation Points	CPU time (sec)	GPU time (sec)	Speedup (×)
100	1.61	0.63	2.55
2,500	37.36	0.86	43.44
10,000	147.65	2.39	61.78
250,000	3,697.97	47.41	78.00

Table 4.2 GPU vs. CPU computational time comparison of electromagnetic aspect of PO-SBR, with 10,000 observation points, 1 frequency, and varying hit points.

Number of Hit Points / Ray Degree Spacing	CPU time (sec)	GPU time (sec)	Speedup (×)
7,155 / 0.16°	59.02	1.03	57.30
10,719 / 0.13°	89.98	1.45	62.06
18,184 / 0.10°	147.65	2.39	61.78
36,988 / 0.07°	300.17	4.87	61.63
112,807 / 0.04°	905.49	14.56	62.20

Table 4.3 GPU vs. CPU computational time comparison of electromagnetic aspect of PO-SBR, with 10,000 observation points, 18,184 hit points, and varying frequencies.

Number of Frequencies / Frequencies (GHz)	CPU time (sec)	GPU time (sec)	Speedup (×)
1 / 10	147.65	2.39	61.78
2 / 10, 20	303.88	3.93	77.32
3 / 10, 20, 30	447.70	6.31	70.95
4 / 10, 20, 30, 40	611.58	6.55	93.37
5 / 10, 20, 30, 40, 50	769.42	10.27	74.92

Table 4.4 GPU vs. CPU computational time comparison of electromagnetic aspect of PO-SBR, with 28,321 hit points, 1 frequency, and varying observation points.

Number of Observation Points	CPU time (sec)	GPU time (sec)	Speedup (×)
100	2.71	0.96	2.82
2,500	64.00	1.33	48.12
10,000	249.29	3.79	65.78
250,000	6356.93	78.02	81.29

Table 4.5 GPU vs. CPU computational time comparison of electromagnetic aspect of PO-SBR, with 10,000 observation points, 1 frequency, and varying hit points.

Number of Hit Points / Ray Degree Spacing	CPU time (sec)	GPU time (sec)	Speedup (×)
13,952 / 7°	126.83	1.85	68.56
19,699 / 6°	176.18	2.59	68.02
28,321 / 5°	249.29	3.79	65.78
44,314 / 4°	385.26	5.82	66.20
78,751 / 3°	676.01	10.26	65.89

Table 4.6 GPU vs. CPU computational time comparison of electromagnetic aspect of PO-SBR, with 10,000 observation points, 28,321 hit points, and varying frequencies.

Number of Frequencies / Frequencies (GHz)	CPU time (sec)	GPU time (sec)	Speedup (×)
1 / 10	249.29	3.79	65.78
2 / 10, 20	508.51	5.98	85.04
3 / 10, 20, 30	796.53	9.65	82.54
4 / 10, 20, 30, 40	1078.78	9.83	109.74
5 / 10, 20, 30, 40, 50	1350.1	15.63	86.38

CHAPTER 5
CONCLUSION

This thesis describes a numerical tool based on the PO-SBR method that is capable of computing the fields of electrically large structures or distances, with the computation accelerated using CUDA on NVIDIA's GPU.

Chapter 2 describes the methodology of the PO-SBR method, and derives a step by step version of the method used in this thesis. In Chapter 3, NVIDIA's GPU architecture is introduced and its advantage pointed out in accelerating the method through its parallel computing capability. CUDA is then used as the software interface to control the threads declared in the GPU to distribute the independent calculations onto computation units of the GPU. In Chapter 4, the validation and speedup examples of scattering field and radiation pattern are demonstrated.

Of course, the version of the method described above is far from being perfect. The drawbacks are the lack of dielectric computing capability and the inflexibility of the ray tracing method. Moreover, the acceleration technique employed on CUDA can be improved by a more careful distribution of the threads. Specifically, the looping of hit points through different kernels can be eliminated by distributing them in the same kernel, then using schemes of parallel reduction to sum the fields caused by those different hit points on the same observation point.

Future work includes the addition of dielectric structures and advanced ray tracing method using a surface mesh. Distribution of the threads in CUDA can also be reviewed and improved to achieve higher speedups in all situations. This thesis has provided a starting point for a complete, higher accuracy PO-SBR method accelerated by GPU.

REFERENCES

[1] S. Chang and T. B. A. Senior, “Scattering by a spherical shell with a circular aperture,” Air Force Weapons Lab., Albuquerque, NM, Interaction Note 141, Apr. 1969.

[2] C. S. Lee, S. W. Lee and R. Chou, “RCS reduction of a cylindrical cavity by dielectric coating,” in *Int. IEEE/Antennas Propagat. Soc. Dig.*, Philadelphia, June 1986, pp. 305-308.

[3] R. W. Ziolkowski and W. A. Johnson, “Plane wave scattering from an open spherical shell: a generalized dual series approach,” in *Nat. Radio Sci. Meet. Dig.*, 1984, p.162.

[4] W. A. Johnson and R. W. Ziolkowski, “The scattering of an H-polarized plane wave from an axially slotted infinite cylinder: A dual series approach,” *Radio Sci.*, vol. 19, no. 1, pp. 275-291, 1984.

[5] H. Ling, R. Chou, and S. W. Lee, “Shooting and bouncing rays: Calculating the RCS of anarbitrarily shaped cavity,” *IEEE Trans. on Antennas and Propagat.*, vol. 37, no. 2, pp. 194-205,1989.

[6] G. A. Deschamps, “Ray techniques in electromagnetic,” *Proc. IEEE*, vol. 60, pp. 1022-1035, Sept. 1972.

[7] S. W. Lee, P. Cramer Jr., K. Woo, and Y. Rahmat-Samii, “Diffraction by an arbitrary subreflector: GTD Solution,” *IEEE Trans. Antenna Propagat.*, vol. 27, pp. 305-316, May 1979.

[8] S. W. Lee, M. S. Sheshadri, V. Jamnejad and R. Mittra, “Reflection at a curved dielectric interface: geometrical optics solution,” *IEEE Trans. Microwave Theory Tech.*, vol. MTT-30, pp. 12-19, Jan. 1982.

[9] C. A. Balanis, *Advanced Engineering Electromagnetics*. New York, NY: Wiley, 1989.

[10] M. Pharr and G. Humphreys, *Physically Based Rendering: From Theory to Implementation*. San Francisco, CA: Morgan Kaufmann, 2004.

[11] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn and T. J. Purcell. “A survey of general-purpose computation on graphics hardware,” *Computer*

Graphics Forum, vol. 26, no. 1, pp. 80-113, March 2007.

[12] E. Dunn, N. Smith, R. Hoare, H. T. Meng and J. Jin, “Hardware acceleration of electromagnetic field profile computation: A case study using the PO-SBR method,” in *Proceedings, High Performance Embedded Computing (HPEC) Workshop*, Lexington, MA, September 2010.

[13] *NVIDIA CUDA programming guide*, version 3.0, Feb. 2010. [Online]. Available: http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/NVIDIA_CUDA_ProgrammingGuide.pdf.

[14] H. T. Meng, J. Jin. and E. Dunn, “Acceleration of asymptotic computational electromagnetics Physical Optics - Shooting and Bouncing Ray (PO-SBR) using CUDA,” poster session presented at the GPU Technology Conference, San Jose, CA, September 2010.