МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

01.03.02 ПРИКЛАДНАЯ МАТЕМАТИКА И ИНФОРМАТИКА

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ И КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ

БАКАЛАВРСКАЯ РАБОТА

на тему Параллельные алгоритмы решения уравнений Максвелла

Студент Д.В. Кирюхин

Руководитель А.В. Очеповский

Допустить к защите
Заведующий кафедрой к.тех.н, доцент, А.В. Очеповский _____

«_____»_____20_____г.

Тольятти 2016

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

УТВЕРЖДАЮ
Зав. кафедрой «Прикладная
математика и информатика»
_____А.В.Очеповский

«____»_____2016 г.

ЗАДАНИЕ
на выполнение бакалаврской работы

Студент Кирюхин Данила Валерьевич
1. Тема Параллельные алгоритмы решения уравнений Максвелла
2. Срок сдачи студентом законченной выпускной квалификационной работы 24.06.2016
3. Исходные данные к выпускной квалификационной работе
- Поддержка компьютером технологии CUDA.

4. Содержание выпускной квалификационной работы (перечень подлежащих разработке вопросов, разделов)

Введение

1. Глава 1 Математические основы для построения модели отражений электромагнитной волны от подстилающей поверхности
   1.1. Исследование земной поверхности методами дистанционного зондирования
   1.2. Уравнения Максвелла
   1.3. Основные задачи электродинамики
   1.4. Известные решения для тел правильной формы
2. Глава 2 Разработка модели отражений электромагнитной волны от подстилающей поверхности
   2.1. Компьютерное моделирование
   2.2. Параллельные вычисления в компьютерном моделировании
   2.3. Численные методы решений уравнений Максвелла
   2.4. Метод FDTD
   2.5. Параллельный метод FDTD для решения задачи рассеяния электромагнитной волны
   2.6. Особенности параллельной реализации FDTD на CUDA

2.7. Программная реализация метода FDTD
3. Глава 3 Оценка полученных результатов
   3.1. Оценка эффективности работы программы на CUDA
   3.2. Сравнение времени работы метода FDTD на CPU и GPU

5. Ориентировочный перечень графического и иллюстративного материала Презентация, включающая блок-схемы работы приложения, графики, диаграммы, экранные формы, демонстрирующие работоспособность программного продукта
6. Дата выдачи задания « 11 » января 2016 г.

Руководитель выпускной квалификационной работы _____ А. В. Очеповский

Задание принял к исполнению _____ Д. В. Кирюхин

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

УТВЕРЖДАЮ
Зав. кафедрой «Прикладная
математика и информатика»
_____А.В.Очеповский

«____»_____2016 г.

КАЛЕНДАРНЫЙ ПЛАН
выполнения бакалаврской работы

Студента Кирюхина Данила Валерьевича
по теме Параллельные алгоритмы решения уравнений Максвелла

Наименование раздела работы	Плановый срок выполнения раздела	Фактический срок выполнения раздела	Отметка о выполнении	Подпись руководителя
Изучение методов ДЗЗ	11.01.2016	11.01.2016	выполнено	
Изучение уравнений Максвелла	19.01.2016	19.01.2016	выполнено	
Изучение задачи рассеяния ЭМВ и методов ее решения	20.02.2016	20.02.2016	выполнено	
Разработка алгоритмов решения для тел правильной формы	8.03.2016	8.03.2016	выполнено	
Разработка параллельных алгоритмов	22.03.2016	22.03.2016	выполнено	
Реализация алгоритмов, написание кода приложения.	14.04.2016	14.04.2016	выполнено	
Написание 3 главы	6.04.2016	6.04.2016	выполнено	
Подведение итогов,	10.04.2016	10.04.2016	выполнено	

редактирование бакалаврской работы. Создание презентационного материала	21.04.2016	21.04.2016		
Проверка на наличие заимствований (плагиата) в системе antiplagiat.ru	30.05.2016	30.05.2016	выполнено	
Предварительная защита	6.06.2016-11.06.2016	31.05.2016	выполнено	
Сдача на кафедру отзыва научного руководителя и ознакомление с ним	20.06.2016	20.06.2016	выполнено	
Сдача на кафедру комплекта документов для защиты	24.06.2016	24.06.2016	выполнено	
Защита ВКР	27-30.06.2016	29.06.2016	выполнено	

Руководитель выпускной квалификационной работы _____ А. В. Очеповский

Задание принял к исполнению _____ Д. В. Кирюхин

Аннотация

Темой данной выпускной квалификационной работы является «Параллельные алгоритмы решения уравнений Максвелла».

Работа выполнена студентом Тольяттинского Государственного Университета, института математики, физики и информационных технологий, группы ПМИб – 1201, Кирюхиным Данилой Валерьевичем.

Объектом исследования данной выпускной бакалаврской работы является процесс рассеяния электромагнитной волны от подстилающей поверхности.

Целью работы является построение алгоритмов для определения характеристик отражения электромагнитной волны от диэлектрических структур.

Предметом исследования являются параллельные алгоритмы моделирующие отражение ЭМВ от подстилающей поверхности.

Для достижения указанной цели необходимо решение следующих задач:

Задачами работы являются**:**

- построение математической модели отражений ЭМВ от подстилающей поверхности;
- разработка алгоритмов для определения характеристик отражения электромагнитной волны (ЭМВ) от диэлектрических структур;
- реализация и исследование алгоритмов для определения характеристик отражения электромагнитной волны (ЭМВ) от диэлектрических структур с использованием технологии CUDA.

Выпускная квалификационная работа состоит из введения, трех глав, заключения, списка литературы и приложения.

В первой главе описаны математические основы для построения модели отражения ЭМВ.

Во второй главе описывается численный метод решения уравнений Максвелла, способы его распараллеливания с учетом технологии паралелльных вычислений CUDA и описание реализации.

В третьей главе производится анализ полученных результатов ускорения и анализ эффективности параллельного алгоритма на CUDA.

В заключении подводится общая оценка, анализ возникших при разработке трудностей, а также перспективы применения параллельных вычислений на GPU.

Выпускная квалификационная работа содержит пояснительную записку объемом 61 страницы, включая 33 иллюстрации, 3 таблицы, список литературы из 20 наименований, приложение.

Оглавление

Введение

В современном стремительно меняющемся мире мы становимся свидетелями непрерывных революционных технологических изменений. Если XX век уже стал веком цифровых технологий, то XXI век можно смело назвать веком космических цифровых технологий. Значительное место в космических технологиях все больше занимает дистанционное зондирование Земли (ДЗЗ) из космоса. Данные, полученные с помощью ДЗЗ, являются важным инструментом для решения практических задач управления различного уровня от государственного до местного, мониторинга природных и техногенных объектов и явлений. Космические снимки активно используются не только в научных и производственных целях, но и в повседневной жизни людей.

Данные полученные с помощью дистанционного зондирования используются в различных областях:

- землепользование и картографирование земельных ресурсов;
- исследования роста городов;
- сельское хозяйство;
- картографирование грунтовых вод;
- борьба с наводнениями;
- гидроморфологические исследования;
- картографирование пустующих земель;
- региональное планирование;
- борьба с природными катастрофами.

На данный момент имеется два способа формирования радиолокационных изображений поверхности [2]:

- дистанционное зондирование земли;
- подповерхностное зондирование.

Для того чтобы синтезировать новые алгоритмы формирования радиолокационных изображений, нам нужно знать статистические

характеристики подстилающей поверхности. Есть три типа сбора характеристик:

- летные эксперименты;
- полунатуральные эксперименты;
- компьютерное моделирование.

Первые два метода являются труднореализуемыми ввиду своей дороговизны, и остаётся прибегать к компьютерному моделированию. Поэтому основной задачей будет являться моделирование характеристик подстилающей поверхности в виде диэлектрически — неоднородных структур. Первым этапом предстоящей работы является разработка и исследование алгоритмов для определения характеристик отражения электромагнитной волны (ЭМВ) от диэлектрических структур правильной формы (сфера, параллелепипед, цилиндр). Такая постановка задачи не требует больших вычислительных ресурсов, но объекты, которые нас окружают, в большинстве своем различной формы и не имеют решения с помощью аналитических алгоритмов, тогда следует применять численные алгоритмы, которые имеют высокую сложность по времени [6]. При необходимости в больших вычислительных ресурсах, используют параллельные вычисления. Параллельные вычисления – вычисления на многопроцессорных системах для одновременного решения различных частей одной задачи. Главной целью параллельных вычислений является снижение времени решения задачи. Задача параллельных вычислений – получение параллельного алгоритма и управление им, с целью достижения наибольшей эффективности использования многопроцессорной вычислительной техники.

Математической основой разработки заявленных алгоритмов являются уравнения Максвелла. Уравнения Максвелла описывают обширную область явлений. Они лежат в основе электротехники и радиотехники и играют важнейшую роль в развитии актуальных направлений современной физики. В данной работе рассматривается решение уравнений Максвелла в задаче рассеяния электромагнитной волны (ЭМВ) неоднородным объектом

произвольной формы. Аналитические подходы решений уравнений, зависимы от жестких упрощающих допущений о геометрии рассеивателя и распределении показателя преломления. Аналитические методы позволяют решать задачу для однородных объектов правильной формы (сфера, цилиндр, параллелепипед). Всё, что находится вокруг нас, в большинстве случаев является неоднородным объектом произвольной формы. Специально для таких объектов есть численные методы, но за точность алгоритма приходиться платить сложностью алгоритма.

Численные алгоритмы, моделирующие задачу рассеяния ЭМВ, обладают высокой вычислительной сложностью. Поэтому применение технологии параллельных вычислений на графических процессорах CUDA для ускорения является **актуальным.**

Новизна работы заключается в применении технологии CUDA для ускорения численных алгоритмов решения задачи рассеяния на ЭВМ.

Объектом исследования является процесс рассеяния ЭМВ от подстилающей поверхности.

Предмет исследования: параллельные алгоритмы моделирующие отражение ЭМВ от подстилающей поверхности.

Целью работы является построение алгоритмов для определения характеристик отражения электромагнитной волны (ЭМВ) от диэлектрических структур.

Задачами работы являются**:**

- построение математической модели отражений ЭМВ от подстилающей поверхности;
- разработка алгоритмов для определения характеристик отражения электромагнитной волны от диэлектрических структур;
- реализация и исследование алгоритмов для определения характеристик отражения электромагнитной волны от диэлектрических структур с использованием технологии CUDA.

Выпускная квалификационная работа состоит из трех глав.

В первой главе описаны математические основы для построения модели отражения ЭМВ.

Во второй главе описывается численный метод решения уравнений Максвелла, способы его распараллеливания с учетом технологии паралелльных вычислений CUDA и описание реализации.

В третьей главе производится анализ полученных результатов ускорения и анализ эффективности параллельного алгоритма на CUDA.

В заключении подводится общая оценка, анализ возникших при разработке трудностей, а также перспективы применения параллельных вычислений на GPU.

Глава 1 Математические основы для построения модели отражений электромагнитной волны от подстилающей поверхности

1.1 Исследование земной поверхности методами дистанционного зондирования

Дистанционное зондирование – это различные способы получения информации об объекте, расположенного на расстоянии, без вступления с ним в прямой контакт, т.е. без непосредственного контакта приемных чувствительных элементов аппаратуры с поверхностью исследуемого объекта. Примерами естественных форм ДЗ являются зрение, обоняние и слух человека. К методам дистанционного зондирования относятся все методы неконтактного получения информации, такие как сейсморазведка, гравиразведка, и т.д. Среди них особое место занимают методы дистанционного зондирования Земли (ДЗЗ) из космоса или воздушного пространства. Под дистанционным зондированием поверхности Земли подразумевается наблюдение и измерение объектов в различных диапазонах электромагнитного спектра с целью определение местоположения, вида, свойств и временной изменчивости объектов окружающей среды без непосредственного контакта с ним измерительного прибора [9].

К методам ДЗЗ относится группа методов получения изображения земной поверхности в определенных участках электромагнитного спектра с авиационных и космических летательных аппаратов для изучения состояния или тематического картографирования поверхности. Таким образом, данные ДЗЗ – это, прежде всего, аэрофотоснимки и КС поверхности Земли.

ДЗЗ имеет широкий круг приложений, начиная с военной разведки. В невоенной сфере большинство приложений относится к категории исследования окружающей среды:

1. Атмосфера: температура, осадки, распределение и тип облаков, концентрации газов.

2. Земная поверхность: топография, температура, альбедо, влажность почвы, тип и состояние растительности, антропогенные нагрузки.
3. Океан: температура, топография, цвет водной поверхности.
4. Криосфера: распределение, состояние и динамические подвижки снега, морского льда, айсбергов, ледников.

Исторически один из наиболее развитых способов получения информации об объектах земной поверхности – это сбор информации «в поле». Сплошное изучение значительных по площади территорий методами наземной съемки требует огромных экономических и временных затрат. Необходимо отметить, что при наземных исследованиях трудно добиться синхронности, одновременности наблюдений на всех участках. Ко всему этому зачастую добавляется такой фактор, как труднодоступность территории. Этих недостатков лишены методы ДЗЗ. Одной из наиболее важных характеристик ДЗЗ является возможность накапливать данные о большой области земной поверхности или объеме атмосферы за короткий промежуток времени, получая практически моментальный снимок. Если этот аспект рассматривать в сочетании с тем фактом, что с помощью спутниковых систем можно получать данные в ситуациях сложных для наземных исследований, когда они медленны, дороги опасны, политически неудобны, то потенциальная мощь ДЗЗ становится еще более очевидной. Дополнительным преимуществом ДЗЗ является возможность систем выдавать калиброванные данные в цифровом виде, которые могут быть введены прямо в компьютер для обработки. Следует отметить, что чем больше территория государства, тем более эффективно применение дистанционных методов [9].

Кроме метода дистанционного зондирования, существует метод подповерхностного зондирования (в общепринятой терминологии — георадиолокация, в англоязычной литературе этот метод называется «Ground Penetrating Radar» или GPR.) основан на изучении распространения электромагнитных волн в среде. Идея метода в излучении импульсов электромагнитных волн и регистрации сигналов, отраженных от границ раздела

слоев зондируемой среды, имеющих различие по диэлектрической проницаемости. Такими границами раздела в исследуемых средах являются, например, контакт между сухими и влагонасыщенными грунтами (уровень грунтовых вод), контакты между породами различного литологического состава, между породой и материалом искусственного сооружения, между мерзлыми и талыми грунтами, между коренными и рыхлыми породами и т.д. Георадарные определения в настоящее время получают широкое применение в различных областях, среди которых в первую очередь необходимо выделить горное дело, транспортное, промышленное и гражданское строительство, экологию [10].

Применение георадарных определений позволяет строить геологические разрезы; определять положение уровня грунтовых вод, толщину льда, глубину и профиль дна рек и озер; определять границы распространения полезных ископаемых, положение карстовых воронок и пустот; выявлять локальные проявления месторождений полезных ископаемых. Для задач горного дела почвенное зондирование с помощью георадарных комплексов дает возможность обследовать борты, уступы и бермы в карьерах; кровлю, потолочины и целики в подземных горных выработках; обнаруживать полости и кварцевые гнезда; выявлять природные и техногенные разрывные нарушения в законтурном массиве пород. С применением современных георадарных технологий возможно производить оценку оснований под транспортные сооружения; определять глубину промерзания в грунтовых массивах и дорожных конструкциях; определять содержание влаги в грунте земляного полотна и подстилающих грунтовых основаниях; определять качество и состояние бетонных конструкций, состояние дамб и плотин; выявлять оползневые зоны, места расположения инженерных сетей. Специально следует выделить решаемые с помощью георадарных технологий задачи экологии: оценка загрязнения почв; обнаружение утечки нефте-, продукто- и водопроводов; идентификация мест захоронения экологически опасных отходов и др.

Упрощенная структурная схема работы дистанционного зондирования представлена на рисунке 1.1. Авиационный или космическо-летательный аппарат фиксирует с помощью сенсора, отраженные электромагнитные волны, которые были получены путём рассеяния электромагнитной волны. Источником электромагнитных волн, может быть пассивным (Солнце) или активным (лазеры, радиолокаторы). На основе отраженных электромагнитных волн, формируется изображение поверхности.

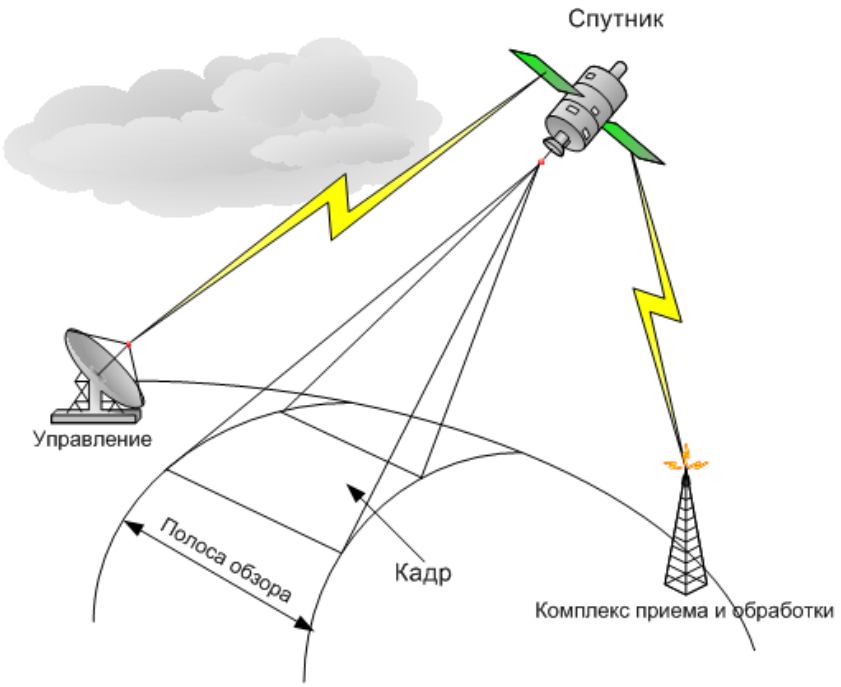


Рисунок 1.1 – Упрощенная схема работы дистанционного зондирования

В зависимости от длины падающей электромагнитной волны – λ и размера частиц – a, изменяется тип рассеяния:

1. Рэлеевское рассеяние – размеры частиц малы по сравнению с длиной волны $a \leq \frac{1}{15}\lambda$.
2. Рассеяние Ми – $a \geq \frac{1}{15}\lambda$.
3. Рассеяние на оптических неоднородностях.
4. Многократное рассеяние – рассеяние в каждом из последовательных актов осуществляется по законам однократного рассеяния.

5. Рамановское рассеяние – характеризуется изменением частоты рассеянной волны по сравнению с падающей волной.

На практике не существует такого сенсора, с помощью которого можно было бы регистрировать все длины волн электромагнитного спектра. Это является очень сложной задачей учесть все типы рассеяния. Из-за сложности ставятся первоначальные задачи:

1. Понять процесс рассеяния.
2. Научиться решать задачу рассеяния для тел правильной формы.

Но во всех типах рассеяния математическим решением задачи рассеяния – является решение системы уравнений Максвелла.

1.2 Уравнения Максвелла

Уравнения Максвелла – система уравнений в дифференциальной или интегральной форме, описывающих электромагнитное поле и его связь с электрическими зарядами и токами в вакууме и сплошных средах [4].

Первое уравнение Максвелла представляет собой закон Гаусса для электрических полей. Выглядит оно следующим образом:

$$\operatorname{div} \mathbf{E} = \frac{\rho}{\varepsilon_0}, \tag{1.1}$$

где div – знак оператора дивергенции (потока);

$\mathbf{E}$ – векторное электрическое поле;

ρ – суммарный заряд;

ε_0 – диэлектрическая постоянная вакуума.

Дивергенция векторного поля определяется как следующая скалярная функция трех переменных:

$$\operatorname{div} \mathbf{F} = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} + \frac{\partial F_z}{\partial z}, \tag{1.2}$$

где div – знак оператора дивергенции (потока);

$\mathbf{F}$ – некоторое векторное поле с декартовыми компонентами F_x, F_y, F_z.

В терминах символического вектора дивергенция $\mathbf{F}$ может быть

формально записана как «скалярное произведение»:

$$\mathrm{div}\,\mathbf{F} = \nabla \cdot \mathbf{F}, \tag{1.3}$$

где div – знак оператора дивергенции (потока);

$\mathbf{F}$ – некоторое векторное поле с декартовыми компонентами F_x, F_y, F_z;

∇ – оператор набла.

Второе уравнение Максвелла – это обобщение закона индукции Фарадея для диэлектрической среды в свободном пространстве:

$$\mathrm{rot}\,\mathbf{E} = -\frac{\partial \mathbf{B}}{\partial \mathrm{t}}, \tag{1.4}$$

где rot – знак оператора ротора (вихрь);

$\mathbf{E}$ – векторное электрическое поле;

$\mathbf{B}$ – векторное магнитное поле;

$\frac{\partial \mathbf{B}}{\partial \mathrm{t}}$ – частная производная $\mathbf{B}$ по времени.

Ротором (вихрем) дифференцируемого векторного поля называется вектор:

$$\mathrm{rot}\,\mathbf{F} = \nabla \times \mathbf{F} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ \frac{\partial}{\partial \mathrm{x}} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \\ F_x & F_y & F_z \end{vmatrix} = \\ = \left(\frac{\partial F_z}{\partial \mathrm{y}} - \frac{\partial F_y}{\partial \mathrm{z}}\right)\mathbf{i} - \left(\frac{\partial F_z}{\partial \mathrm{x}} - \frac{\partial F_x}{\partial \mathrm{z}}\right)\mathbf{j} - \left(\frac{\partial F_y}{\partial \mathrm{x}} - \frac{\partial F_x}{\partial \mathrm{y}}\right)\mathbf{k}, \tag{1.5}$$

где rot – знак оператора ротора (вихрь);

$\mathbf{F}$ – некоторое векторное поле с декартовыми компонентами F_x, F_y, F_z;

∇ – оператор набла;

i, **j**, **k** – единичные векторы.

Третье уравнение Максвелла – это закон Гаусса, но для магнитных полей:

$$\mathrm{div}\,\mathbf{B} = 0, \tag{1.6}$$

где div – знак оператора дивергенции (потока);

$\mathbf{B}$ – векторное магнитное поле.

Четвертое уравнение Максвелла – это закон Андре Ампера, который связывает постоянный ток и магнитное поле вокруг него, но с дополнительным членом $(\frac{1}{c^2})\cdot\frac{\partial \mathbf{E}}{\partial t}$:

$$\text{rot}\,\mathbf{B} = \frac{j}{\varepsilon_0 c^2} + \left(\frac{1}{c^2}\right)\cdot\frac{\partial \mathbf{E}}{\partial t}, \tag{1.7}$$

где rot – знак оператора ротора (вихрь);

B – векторное магнитное поле;

j – ток;

ε_0 – диэлектрическая постоянная вакуума;

c – скорость света;

E – векторное электрическое поле;

$\frac{\partial \mathbf{E}}{\partial t}$ – частная производная **E** по времени.

Четыре уравнения в дифференциальной форме, представленные выше образуют систему уравнений Максвелла.

1.3 Основные задачи электродинамики

Основная задача электродинамики состоит в отыскании электрического (**E**) и магнитного (**B**) поля с помощью системы уравнений Максвелла. Такое решение возможно, если известны плотности заряда и тока во всех точках пространства и во все моменты времени. При интегрировании дифференциальных уравнений систем, получается решения, содержащие некоторые произвольные функции. Такова структура общего решения дифференциальных уравнений в частных производных. Для того чтобы из общего решения найти частное – конкретное поле, необходимо располагать начальными и граничными условиями [6].

Начальные условия – это значения величин **E** и **B** во всех точках пространства в некоторый момент времени, принимаемый за начальный. Граничные условия – это значения векторов поля на границе области пространства, занимаемой полем. Поле системы зарядов в пустоте не

ограничено какими-либо конечными поверхностями. Граничные условия здесь – значения векторов поля при бесконечном удалении от системы зарядов. Физический смысл имеют только те задачи, в которых система зарядов занимает не все бесконечное пространство, а ограниченную его область.

Основная задача электродинамики состоит в отыскании поля по заданному распределению и движению зарядов при известных начальных и граничных условиях. При указанных условиях уравнения Максвелла имеют единственное решение. Задание системы на некоторый начальный момент времени позволяет определить ее состояние во все последующие моменты времени. Имеет смысл, а также практическое значение задача, обратная: по заданному полю определить плотность зарядов и токов. Кроме названных двух задач, в отдельных случаях может решаться и задача о движении заряженных тел, внесенных в поле. Часто оказывается необходимым найти электромагнитное поле, созданное зарядом, движущимся под действием, внешнего поля. Типичный пример – задача на рассеяние света. Заряды, входящие в состав вещества испытывают на себе действие светового электромагнитного поля, приходят в колебания и сами излучают электромагнитные волны – рассеивают свет. В общем случае задача о рассеянии ставится следующим образом. На некоторый объект произвольной формы с диэлектрической проницаемостью ($\varepsilon_p(\vec{r})$) и объемом (V) падает электромагнитная волна в направлении распространения ($\overrightarrow{k_l}$) и с колебаниями электрического вектора в направлении ($\overrightarrow{e_l} = \frac{\overrightarrow{E_l}}{|E_l|}$). Волна движется в пространстве с диэлектрической проницаемостью ε_0. После рассеивания и поглощения результирующая волна имеет направление распространения $\vec{k}_s$ и колебания электрического вектора в направлении $\overrightarrow{e_s} = \frac{\overrightarrow{E_s}}{|E_s|}$. Точное решение задачи рассеяния сводится к решению волнового уравнения. Основные трудности строгого решения волнового уравнения стимулировали поиск и развитие других способов решения задачи рассеяния. Такие альтернативные

решения, которые на практике используются гораздо шире, чем точные, развивались в двух направлениях: 1) приближенные решения теории рассеяния и 2) численные методы решения волновых уравнений. Аналитические подход, зависимы от жестких упрощающих допущений о геометрии рассеивателя и распределении показателя преломления. Совершенно неясно, насколько ограничивающими эти допущения являются и, прежде всего, что делать, если они неприменимы. В последнее время, достигнут существенный прогресс в развитии численных подходов к задаче рассеяния, отвечающих на эти вопросы. Имеется два численных метода, оказавшиеся весьма полезными при исследовании рассеяния света: модель дискретных частиц (МДЧ) и трехмерный конечно-разностный временной метод (КРВМ). МДЧ представляет собой продолжение аналитических методов. Она моделирует сложный неоднородный рассеиватель комбинацией одинаковых дискретных объектов простой, обычно сферической, формы. Для выполнения этих расчетов нужно предположить некоторое распределение размеров и показателей преломления элементарных рассеивателей. Нормальное или логарифмическое нормальное распределения используются наиболее часто. Затем поле рассеянного излучения рассчитываются численно как суперпозиция полей рассеяния от каждой частицы с помощью либо теории Ми, либо ее приближений. Как можно видеть, хотя эта модель и дает некоторое понимание внутренней структуры частицы рассеивающего образца, она ни в коем случае не является строгой [6].

КРВМ представляет собой более общий подход, не имеющий ограничений модели дискретных частиц. Он позволяет вычислить амплитуды рассеяния для неоднородных объектов произвольной формы. КРВМ ставит задачу нахождения численного решения уравнений Максвелла для электромагнитной волны, распространяющейся в среде с заданным распределением показателя преломления. Уравнения дискретизируются во времени и пространстве на четырехмерной сетке. Значение показателя преломления задано в каждом элементе сетки. Очевидно, чем меньше шаг сетки во времени и пространстве, тем точнее модель. Несмотря на общность, КРВМ

страдает рядом недостатков. Во-первых, он требует большого объема вычислений. Во-вторых, давая решение задачи рассеяния, он не всегда помогает понять механизм процесса рассеяния. Наконец, наиболее важным мотивом решения прямой задачи рассеяния является выяснение общих свойств рассеяния, которые позволят хотя бы частично решить обратную задачу. С другой стороны. КРВМ может быть предельно полезен при сравнении различных приближенных аналитических моделей с численным экспериментом, который трудно реализовать в лаборатории, что позволяет выработать наиболее точные аналитические описания.

1.4 Известные решения для тел правильной формы

Реальные радиолокационные цели имеют довольно сложную форму поверхности и выполняются из самых различных материалов. Поэтому приближенные численные методы расчета характеристик рассеяния радиоволн объектами сложной геометрической формы в широком диапазоне частот при воздействии короткого по длительности импульса дают результаты, которые довольно сложно интерпретировать физически. Это приводит к тому, что оценить степень близости результатов к истинным значениям, а также обоснованно выбрать параметры приближенных методов, например, интервалы дискретизации по времени и пространству, не всегда просто. Одним из возможных путей решения этой проблемы является проведение расчетов на тестовых задачах, точный результат решения которых известен.

Точные аналитические решения задач дифракции электромагнитных волн получены для некоторых тел, имеющих правильную геометрическую форму: сфера (шар), цилиндр, часть плоской поверхности. Наиболее простые результаты получаются для тел с идеально проводящими поверхностями при стационарном воздействии гармонических колебаний, когда поле рассматривается в дальней зоне. Анализ этих известных зависимостей позволяет выявить некоторые общие закономерности, которые целесообразно было бы использовать в сверхширокополосной радиолокации.

Самым простым и хорошо изученным объектом с точки зрения рассеяния электромагнитной волны является идеально проводящая сфера. Для нее проведен точный расчет методом криволинейных координат в сферических координатах. Впервые анализ рассеяния сферы был проведен в 1908 г. ученым Густавом Ми, поэтому некоторые формулы и зависимости для сферы носят его имя.

Далее описана физическая задача рассеяния сферой. Векторные поля E и H подчиняющиеся уравнениям Максвелла в условиях поставленной задачи могут быть разделены на три части: поля $\mathrm{E_i}$ падающей волны, поля E_s рассеянной от сферы волны и поля $\mathrm{E_r}$ внутри сферы. Все поля: $\mathrm{E_i}$, $\mathrm{E_s}$ и $\mathrm{E_r}$, удовлетворяют волновым уравнениям. Поля вне сферы удовлетворяют граничным условиям.

$$(\mathbf{B}_2 - \mathbf{B}_1)\cdot n = 0, \qquad (\mathbf{D}_2 - \mathbf{D}_1)\cdot n = K,$$

$$(\mathbf{E}_2 - \mathbf{E}_1)\times n = 0, \qquad (\mathbf{H}_2 - \mathbf{H}_1)\times n = L.$$

Вместо того чтобы решать векторные волновые уравнения, можно использовать электрические векторы Герца $\boldsymbol{\pi_1}$ и $\boldsymbol{\pi_2}$, которые позволяют свести задачу к отысканию решения для скалярного волнового уравнения.

Векторы Герца могут быть представлены через волновые уравнения следующим образом:

$$\nabla^2\pi_1 - \sigma\mu\frac{\partial\pi_1}{\partial \mathrm{t}} - \varepsilon_0\mu\frac{\partial^2\pi_1}{\partial t^2} = -\frac{\mathbf{P}}{\varepsilon_0}, \tag{1.8}$$

$$\nabla^2\pi_2 - \sigma\mu_0\frac{\partial\pi_2}{\partial \mathrm{t}} - \varepsilon_0\mu_0\frac{\partial^2\pi_2}{\partial t^2} = -\mathbf{M}, \tag{1.9}$$

где ∇ – оператор набла;

π_1, π_2 – потенциалы Герца;

σ – сечение рассеяния;

ε_0, μ_0 – индуктивные потенциалы свободного пространства;

$\mathbf{P}$ – электрическая поляризация;

M – магнитная поляризация.

Из уравнения (1.8) выражается ТМ-волна, которая характеризуется нулевым значением радиальной компоненты магнитного поля H_{1r}. Соответственно, из уравнения (1.9) выражается ТЕ-волна, которая характеризуется нулевым значением радиальной компоненты электрического поля E_{2r}.

Вектора Герца могут быть представлены через скалярные функции потенциала, известные по-другому как потенциалы Герца или потенциалы Дебая:

$$-\nabla\cdot\boldsymbol{\pi_1} = \pi_1, \qquad -\nabla\cdot\boldsymbol{\pi_2} = \pi_2, \tag{1.10}$$

где ∇ – оператор набла;

$\boldsymbol{\pi_1}, \boldsymbol{\pi_2}$ – электрические векторы Герца;

π_1, π_2 – потенциалы Герца - Дебая.

Потенциалы Герца-Дебая, π_1 и π_2 являются решениями скалярного волнового уравнения. Если последнее разрешимо для ТМ-волн и ТЕ-волн, тогда векторные поля могут быть представлены через сумму ТМ-волн и ТЕ-волн. Учитывая это, компоненты электромагнитных полей в сферических координатах представимы в виде:

$$E_r = E_{1r} + E_{2r} = \frac{\partial^2(r\pi_1)}{\partial r^2} + k^2 r\pi_1 + 0, \tag{1.11}$$

$$E_r = E_{1\theta} + E_{2\theta} = \frac{1}{r}\frac{\partial^2(r\pi_1)}{\partial \mathrm{r}\,\partial\theta} + k_2\frac{1}{r sin\theta}\frac{\partial r\pi_2}{\partial\,\varphi}, \tag{1.12}$$

$$E_r = E_{1\theta} + E_{2\theta} = \frac{1}{r sin\theta}\frac{\partial^2(r\pi_1)}{\partial r\,\partial\varphi} - k_2\frac{1}{r}\frac{\partial r\pi_2}{\partial\,\varphi}, \tag{1.13}$$

$$H_{\mathrm{r}} = H_{1\mathrm{r}} + H_{2\mathrm{r}} = 0 + \frac{\partial^2(r\pi_2)}{\partial r^2} + k^2 r\pi_2, \tag{1.14}$$

$$H_\theta = H_{1\theta} + E_{2\theta} = -k_1\frac{1}{r sin\theta}\frac{\partial(r\pi_1)}{\partial\varphi} + \frac{1}{r}\frac{\partial^2(r\pi_2)}{\partial r\,\partial\,\theta}, \tag{1.15}$$

$$H_{\varphi} = H_{1\varphi} + E_{2\varphi} = k_1 \frac{1}{r}\frac{\partial r\pi_1}{\partial \theta} + \frac{1}{r sin\theta}\frac{\partial^2 (r^2 \pi_2)}{\partial r\, \partial \varphi}, \quad (1.16)$$

где константа распространения:

$$k^2 = -k_1 k_2, \qquad k_1 = -i\omega\varepsilon + \sigma, \qquad k_2 = -i\omega.$$

Математическая задача сводится к отысканию решения для однородного скалярного волнового уравнения, которое в сферических координатах принимает вид:

$$\frac{1}{r}\frac{\partial^2 r\pi}{\partial r^2} + \frac{1}{r^2 sin\theta}\frac{\partial}{\partial \theta}\left(sin\theta \frac{\pi}{\theta}\right) + \frac{1}{r^2 sin^2\theta}\frac{\partial^2 \pi}{\partial \varphi} + k^2\pi = 0, \quad (1.17)$$

где

$$\pi = R(r)\Theta(\theta)\Phi(\varphi), \quad (1.18)$$

где функции $R(r)$, $\Theta(\theta)$, $\Phi(\varphi)$ удовлетворяют обыкновенным дифференциальным уравнениям:

$$\frac{d^2 rR(r)}{dr^2} + \left[k^2 - \frac{n(n+1)}{r^2}\right] rR(r) = 0, \quad (1.19)$$

$$\frac{1}{sin\theta}\left(sin\theta \frac{d\Theta(\theta)}{d\theta}\right) + \left[\mathrm{n(n+1)} - \frac{m^2}{sin^2\theta}\right]\Theta(\theta) = 0, \quad (1.20)$$

$$\frac{d^2\Phi(\varphi)}{d\varphi^2} + m^2\Phi(\varphi) = 0, \quad (1.21)$$

где n - это интеграл и m это интегральные значения -n, ... , ... , +n. Дополнительные условия задачи:

- решение π должно быть найдено отдельно для падающей волны π_i, рассеянной волны π_s и волны π_r внутри сферы;
- величины π_i, π_s, π_r не должны обращаться в бесконечность в начале, т.е. $\pi_s \rightarrow \infty$ при r = 0;
- величина π_s должна стремиться к нулю в бесконечности, т.е. $\pi_s \rightarrow 0$ при r $\rightarrow \infty$;
- величины π_i, π_s, π_r должны удовлетворять граничному условию на r = a:

$$\left(\frac{\partial}{\partial \mathrm{r}}\right)\left[r\left(\pi_1^i+\pi_1^s\right)\right]=\left(\frac{\partial}{\partial \mathrm{r}}\right)\left[r\,\pi_1^r\right], \tag{1.22}$$

$$\left(\frac{\partial}{\partial \mathrm{r}}\right)\left[r\left(\pi_2^i+\pi_2^s\right)\right]=\left(\frac{\partial}{\partial \mathrm{r}}\right)\left[r\,\pi_2^r\right], \tag{1.23}$$

$$k_1^{(2)} r(\pi_1^i+\pi_1^s)=k_1^{(1)} r\,\pi_1^r, \tag{1.24}$$

$$k_2^{(2)} r(\pi_1^i+\pi_1^s)=k_2^{(1)} r\,\pi_2^r, \tag{1.25}$$

Таким образом, задача рассеяния ЭМВ сферой сводится к решению скалярного волнового уравнения, представленного в формуле (1.17) [13].

В данной главе было представлено дистанционное зондирование и сфера его применения. Были описаны уравнения Максвелла, которые являются математической основой задачи рассеяния ЭМВ, также представлены основные задачи электродинамики, которые решаются с помощью уравнений Максвелла и приведены примеры задачи рассеяния ЭМВ для тел правильной формы.

Глава 2 Разработка компьютерной модели отражений электромагнитной волны от подстилающей поверхности

2.1 Компьютерное моделирование

Моделирование представляет собой процесс замещения объекта исследования некоторой его моделью и проведение исследований на модели с целью получения необходимой информации об объекте. Модель – это физический или абстрактный образ моделируемого объекта, удобный для проведения исследований и позволяющий адекватно отображать интересующие исследователя физические свойства и характеристики объекта. Удобство проведения исследований может определяться различными факторами: легкостью и доступностью получения информации, сокращением сроков и уменьшением материальных затрат на исследование и др. Рассмотрим краткую классификацию видов моделирования систем представленную на рисунке 2.1.

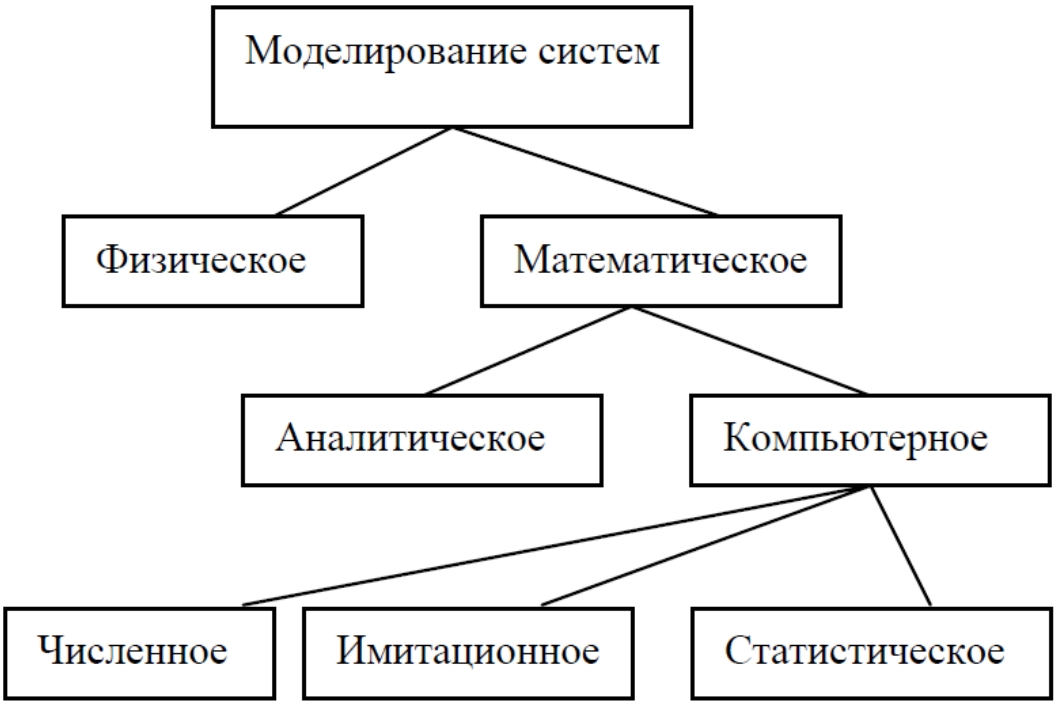


Рисунок 2.1 — Классификация видов моделирования систем

Различают моделирование физическое и математическое. Физическое моделирование предполагает, что в качестве модели используется либо сама исследуемая система (например, в случае производственного эксперимента),

либо другая система с той же или подобной физической природой. Обычно изготавливается макетный или опытный образец объекта, проводятся испытания, в процессе которых определяются его выходные параметры и характеристики, оцениваются надежность функционирования и степень выполнения технических требований, предъявленных к объекту. Если вариант технической разработки оказался неудачным, все повторяется сначала, то есть осуществляется повторное проектирование, изготовление опытного образца, испытания и т.д. Примером такого физического моделирования является продувка моделей самолетов в аэродинамических трубах. Понятно, что физическое моделирование сопряжено с большими временными и материальными затратами. Под математическим моделированием понимается процесс установления соответствия данной реальной системы некоторой математической модели и исследование этой модели, позволяющее получить характеристики реальной системы. Математическое моделирование можно разделить на аналитическое и компьютерное. Для аналитического моделирования характерно то, что процесс функционирования элементов системы записывается в виде некоторых математических соотношений (алгебраических, интегральных, разностных и т.д.) или логических условий. Аналитическая модель исследуется следующими методами: 1) аналитическим, когда стремятся получить в общем виде явные зависимости для искомых характеристик системы; 2) численным, когда, не умея решать уравнения в общем виде, стремятся получить числовые результаты, но при конкретных начальных данных; 3) качественным, когда не имея решения в явном виде, можно найти некоторые свойства решения (например, оценить устойчивость решения). Введем в рассмотрение необходимые определения. Процессом называется серия реальных операций или обработок исходных материалов. Системой (объектом) называется процесс или часть процесса, выбранная для анализа. Математической моделью называется приближенное описание реального процесса, выраженное с помощью математических соотношений. Любая математическая модель описывает реальный процесс лишь с некоторой

степенью приближения к действительности. Математические модели могут представлять собой системы дифференциальных уравнений (обыкновенных или в частных производных), системы алгебраических уравнений, матричные уравнения, линейные, нелинейные уравнения и т.д. Компьютерное моделирование можно разделить на три вида: численное, имитационное, статистическое. Для компьютерного моделирования характерно, что математическая модель системы представлена в виде программы на ЭВМ или компьютерной модели, позволяющей проводить с ней вычислительные эксперименты. При численном моделировании для построения компьютерной модели используются методы вычислительной математики, а вычислительный эксперимент заключается в численном решении некоторых математических уравнений при заданных значениях параметров и начальных условиях. Имитационное моделирование – это вид компьютерного моделирования, для которого характерно воспроизведение на ЭВМ (имитация) процесса функционирования исследуемой системы. При этом имитируются элементарные явления, составляющие процесс, с сохранением их логической структуры, последовательности протекания во времени, что позволяет получить информацию о состоянии системы в заданные моменты времени. Статистическое моделирование – это вид компьютерного моделирования, позволяющий получить статистические данные о процессах в моделируемой системе.

Математическое моделирование какого-либо объекта порождает такой план действий, который можно разбить на три этапа: модель – алгоритм – программа представлена на рисунке 2.2. На первом этапе строится «эквивалент» объекта. Этот «эквивалент» отражает в математической форме важные для данного исследования свойства объекта: законы, которым подчиняется объект, связи, присущие его частям и т.д. Затем математическая модель исследуется теоретическими методами, что позволяет получить предварительные знания об объекте.

Второй этап – разработка алгоритма для реализации модели на компьютере. Модель представляется в форме, удобной для применения численных методов, определяется последовательность вычислительных и логических операций, которые нужно произвести, чтобы найти искомые величины с заданной точностью. Вычислительные алгоритмы не должны искажать основные свойства модели и, следовательно, исходного объекта, быть экономичными и адаптирующимися к особенностям решаемых задач и используемых компьютеров.

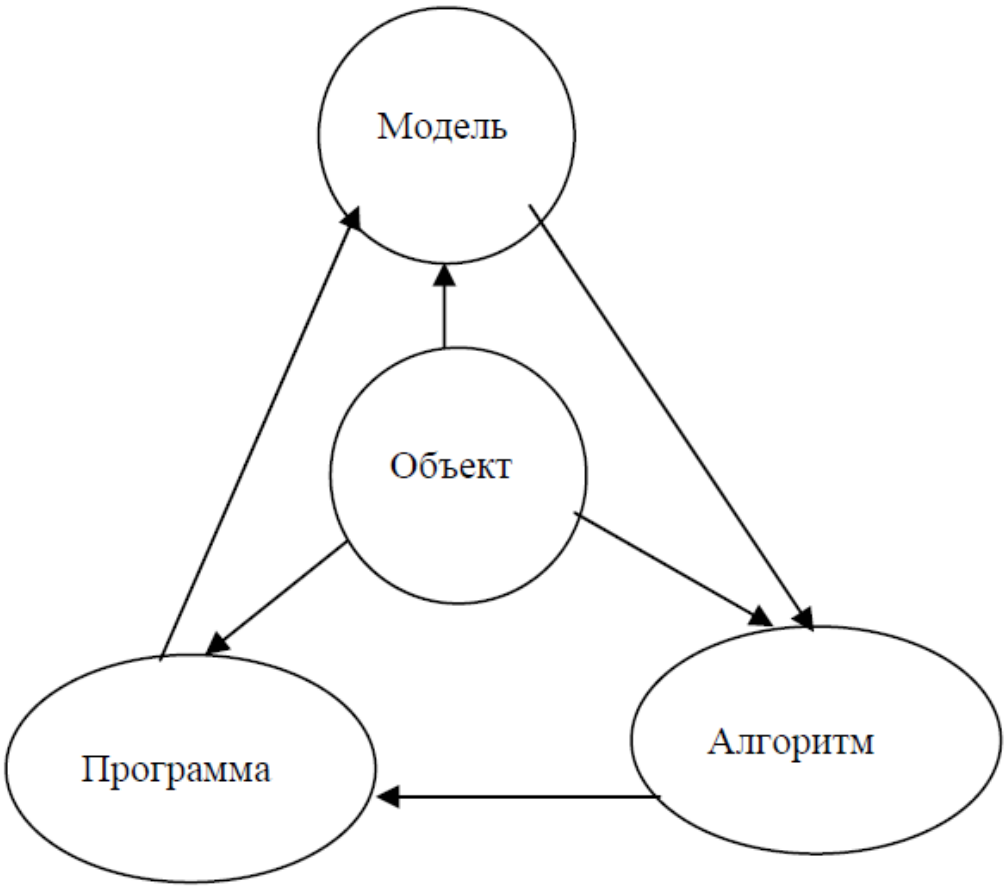


Рисунок 2.2 – Триада математического моделирования

На третьем этапе создаются программы, «переводящие» модель и алгоритм на доступный компьютеру язык. К ним также предъявляются требования экономичности и адаптивности. Их можно назвать «электронным» эквивалентом изучаемого объекта, уже пригодным для испытания на компьютере. Создав триаду «модель – алгоритм – программа», исследователь получает в руки универсальный, гибкий и недорогой инструмент, который

отлаживается, тестируется в «пробных» вычислительных экспериментах. После того, как адекватность (достаточное соответствие) триады исходному объекту имеется, с моделью проводятся разные «опыты», дающие все требуемые свойства и характеристики объекта. Важно, что процесс моделирования сопровождается улучшением и уточнением всех звеньев триады. Такой метод познания сочетает в себе достоинства, как теории, так и эксперимента. Действительно, работа не с самим объектом (явлением или процессом), а с математической моделью дает возможность относительно быстро, без существенных затрат исследовать его свойства и поведение в любых мыслимых ситуациях. Это составляет преимущества теории. В то же время вычислительные эксперименты с моделями объектов позволяют, опираясь на мощь вычислительных методов и компьютеров, глубоко и полно изучать объекты, что недоступно чисто теоретическим подходам. Это уже составляет преимущества эксперимента. Математическое моделирование как методология не подменяет собой математику, физику, биологию и другие научные дисциплины. Оно играет синтезирующую роль. Создание и применение триады невозможно без опоры на самые разные методы и подходы – от качественного анализа нелинейных моделей до современных языков программирования. Но, решая проблемы информационного общества, нельзя уповать только на мощь компьютеров. Необходимо постоянное совершенствование триады математического моделирования.

2.2 Параллельные вычисления в компьютерном моделировании

Параллельные вычисления – вычисления, которые можно реализовать на многопроцессорных системах с использованием возможности одновременного выполнения многих действий, порождаемых процессом решения одной или многих задач. Основной целью параллельных вычислений является уменьшение времени решения задачи. Многие необходимые для нужд практики задачи требуется решать в реальном времени или для их решения требуется очень большой объем вычислений. Задача параллельных вычислений –

создание ресурса параллелизма (получение параллельного алгоритма) в процессах решения задач и управление реализацией этого параллелизма с целью достижения наибольшей эффективности использования многопроцессорной вычислительной техники. Получить параллельный алгоритм решения задачи можно путем распараллеливания имеющегося последовательного алгоритма или путем разработки нового параллельного алгоритма. Возможно, для осуществления распараллеливания алгоритм решения задачи придется заменить или модифицировать (например, устранить некоторые зависимости между операциями).

GPU – это специализированный процессор, который сконструирован, как многопоточный сопроцессор для параллельного вычисления огромных массивов данных. Неспециализированные вычисления на графических процессорах используют графический процессор для выполнения вычислений, а управляет процессом вычислений центральный процессор. NVIDIA Geforce, Quadro и Tesla имеют поддержку неспециализированных вычислений, используя технологию CUDA. CUDA – это технология программирования, которая объединяет между собой центральный и графический процессор, разрабатывается компанией NVIDIA [1].

Графический процессор схож с многоядерным центральным процессором, но с двумя главными различиями. Ядра центрального процессора созданы для исполнения одного потока последовательных инструкций с максимальной производительностью, а графические процессоры проектируются для быстрого исполнения большого числа параллельно выполняемых потоков инструкций. Второе различие – это как организуются потоки. Операционная система планирует потоки различных ядер центрального процессора по приоритету. Графические процессоры имеют специальное аппаратное оборудование для совместного управления потоками. В таблице 2.1 представлена производительность и технические характеристики ряда современных графических и центральных процессоров [7].

Таблица 2.1 – Производительность ряда современных CPU и GPU

Model	**Transistor Count (10^6)**	**Die Size (mm^2)**	**Shader Cores**	**Clock Rate (GHz)**	**RAM Bandwidth (GB/s)**	**Performance (GFLOPS)**
AMD Opteron 6128	1,200	315	8	2.0	42.7	256
AMD A8-3850	758	258	4/400	2.9/0.6	29.8	355
Intel Xeon E5540	731	263	4/8	2.53	25.6	40.5/45
Intel Core i7 990X	1,170	240	6	3.46-3.7	24.5	107.58
Intel Core i7 2600K	995	216	4/48	3.4/0.85	24.5	129.6
NVIDIA Tesla C1060	1,400	576	240	1.296	102.4	622.08
NVIDIA Tesla C2070	3,100	529	448	1.150	144	1,030.40
NVIDIA GTX 480	3,200	529	480	1.41	177.4	1,345
NVIDIA GTX 580	3,000	520	512	1.544	192.4	1,581.1
NVIDIA GTX 690	2 x 3,540	2 x 294	915	1.019	2 x 192.3	2 x 2,810

В модель программирования на графических процессорах входит три компонента: решётка, блок и поток. Поток выполняется ядром с соответствующим ему индексом. Каждый поток использует этот индекс для доступа к элементам в массиве, таким образом, что совокупность всех потоков совместно обрабатывает весь набор данных массива. Блок это группа потоков, которые исполняются параллельно или последовательно, но все в произвольном порядке. Один поток может управлять потоками. Решетка – это группа из блоков. Все потоки в пределах блока исполняются одним ядром и используют общую память, которая показана на рисунке 2.4.

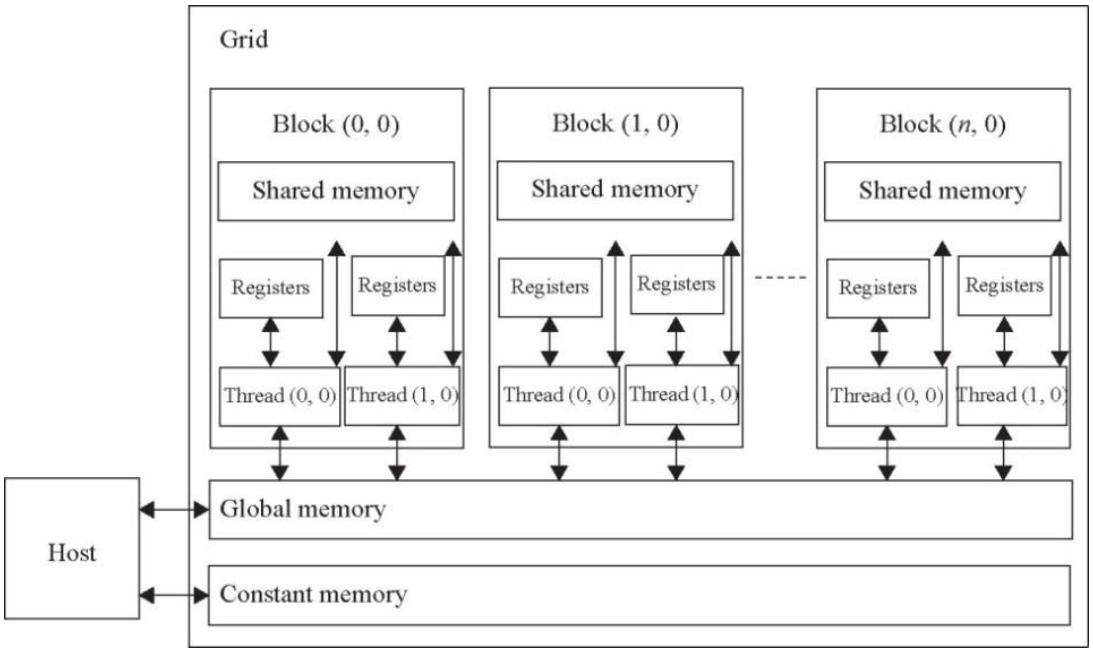


Рисунок 2.4 – Расположение и взаимодействие потоков и памяти в графическом процессоре

На заре компьютерных вычислений, исследования в данной области полагались на мощность центральных процессоров для решения различных вычислительных задач. Спустя годы был достигнут огромный прогресс в использовании мощностей центральных процессоров путем повышения тактовых частот, увеличения кэшей, быстрой памяти, появления многоядерных процессоров. Это разрабатывалось для пользователей, нуждами которых являлись, просмотр видео в интернете и компьютерные игры. Центральный процессор был вынужден стать «мастером на все руки» для вычислительных задач, он должен был выполнять большое количество задач, не специализируясь на какой-то конкретной области.

А графический процессор устроен по своей архитектуре для очень узкого круга задач, которые требуют выполнения относительно небольшого количества операций. Видео карты разрабатывались с одной целью, с целью обработки команд и данных, необходимых для обеспечения графики. За последние несколько лет, процесс разработки графических процессоров начал двигаться быстрее, чем у центральных процессоров в связи с задачами, для которых он был предназначен. Это привело к разработке очень мощных

процессорных блоков для компьютерной графики. Следует отметить, что графический процессор был разработан конкретно для визуализации графики, а не для вычислительных задач. Различные процессы, используемые в визуализации графики, аналогичны для многих математических векторных функций.

CUDA среда разработки от NVIDIA сделала вычисления на графических процессорах проще. Для программирования с использованием архитектуры CUDA, разработчики могут использовать язык C, который позволяет работать с процессорами, поддерживающие CUDA, с высокой производительностью.

CUDA, как и другие языки или платформы программирования имеет свои плюсы и минусы. Самый главный плюс CUDA – простота в изучении по сравнению со своими конкурентами и NVIDIA предоставляет обширную поддержку разработчикам. Одним главным минусом является то, что CUDA работает только на совместимых видеокартах от NVIDIA.

2.3 Численные методы решения уравнений Максвелла

Существует два основных численных метода решения задачи рассеяния света: метод конечных элементов (МКЭ) и метод конечных разностей (МКР).

Метод конечных элементов – численный метод решения дифференциальных уравнений в частных производных. Идея метода заключается в разбиении области, в которой ищется решение дифференциальных уравнений, на конечное количество подобластей. В каждой из подобластей выбирается аппроксимирующая функция. Чаще всего это полином первой степени. За границами подобластей функция обращается в нуль. Решением задачи являются значения функций на границах подобластей, которые заранее неизвестны. Коэффициенты функций ищутся из условия равенства значения соседних функций на границах подобластей. Затем эти коэффициенты выражаются через значения функций в узлах подобластей. Составляется СЛАУ. Количество уравнений в СЛАУ равно количеству неизвестных значений в узлах прямо пропорционально количеству элементов.

Метод конечных разностей – численный метод решения дифференциальных уравнений, также называют методом сеток. Основная идея заключается в замене производных разностными схемами. На область, в которой ищется решения дифференциальных уравнений, наносится сетка с узлами. Каждую производную приближенно заменяют соответствующей разностной схемой и, таким образом, выражаются через неизвестные узловые значения искомой функции. В результате получаем СЛАУ относительно значений функций в узлах сетки. Решение этой системы позволяет, в конечном счете, получить приближенное решение исходной задачи.

Большим преимуществом метода конечных разностей является слабая зависимость от граничных условий задачи, геометрии объекта, что упрощает вычисления на ЭВМ. Далее рассматривается метод конечных разностей во временной области.

2.4 Метод FDTD

Метод конечных разностей во временной области (FDTD) – это численный метод, основанный на замене производных разностными схемами. Он используется, чтобы решать уравнения Максвелла для электрических и магнитных полей во временной и пространственной области. Метод FDTD использует центрально – разностную аппроксимацию для дискретизации двух вихревых уравнений Максвелла, а именно, законов Фарадея (1.4) и Ампера (1.7), во временных и пространственных областях, а затем полученные уравнения решаются численно для получения электрического и магнитного поля в каждый момент времени и точке пространства.

По схеме Yee расчетная область дискретизируется с помощью прямоугольной сетки. Электрические поля, расположены вдоль границы электрических блоков, в то время как магнитные поля располагаются в центре. Это удовлетворяет свойствам электрических и магнитных полей в уравнениях Максвелла. Электрический элемент типичной схемы Yee показан на рисунке 2.5.

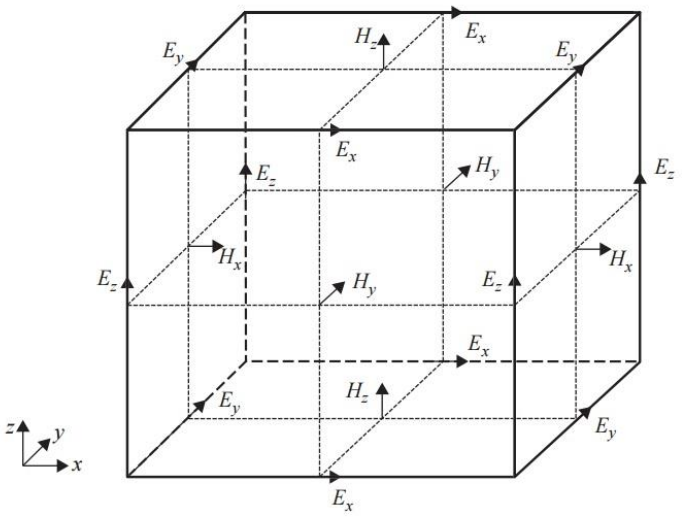


Рисунок 2.5 – Расположение электрических и магнитных полей по схеме Yee.

Электрические поля по времени разбиваются на участки nΔt и считаются что, они однородны в течение периода с (n - 1/ 2Δt) до (n + 1 / 2Δt). Аналогичным образом разбиваются магнитные поля, которые имеют шаг (n + 1 / 2Δt), относительно распределения электрических полей, и считаются что, они однородны в период с nΔt до (n + 1)Δt.

В Декартовой системе координат уравнения (1.4) и (1.7) выражаются в виде шести дифференциальных уравнений в частных производных [14]:

$$\frac{\partial \mathrm{H}_x}{\partial \mathrm{t}} = \frac{1}{\mu_x}\left(\frac{\partial \mathrm{E}_y}{\partial \mathrm{z}} - \frac{\partial \mathrm{E}_z}{\partial \mathrm{y}} - \sigma_{M_x} H_x\right), \tag{2.1}$$

$$\frac{\partial \mathrm{H}_y}{\partial \mathrm{t}} = \frac{1}{\mu_y}\left(\frac{\partial \mathrm{E}_z}{\partial \mathrm{x}} - \frac{\partial \mathrm{E}_x}{\partial \mathrm{z}} - \sigma_{M_y} H_y\right), \tag{2.2}$$

$$\frac{\partial \mathrm{H}_z}{\partial \mathrm{t}} = \frac{1}{\mu_z}\left(\frac{\partial \mathrm{E}_x}{\partial \mathrm{y}} - \frac{\partial \mathrm{E}_y}{\partial \mathrm{x}} - \sigma_{M_z} H_z\right), \tag{2.3}$$

$$\frac{\partial \mathrm{E}_x}{\partial \mathrm{t}} = \frac{1}{\varepsilon_x}\left(\frac{\partial \mathrm{H}_z}{\partial \mathrm{y}} - \frac{\partial \mathrm{H}_y}{\partial \mathrm{z}} - \sigma_x E_x\right), \tag{2.4}$$

$$\frac{\partial \mathrm{E}_y}{\partial \mathrm{t}} = \frac{1}{\varepsilon_y}\left(\frac{\partial \mathrm{H}_x}{\partial \mathrm{z}} - \frac{\partial \mathrm{H}_z}{\partial \mathrm{x}} - \sigma_y E_y\right), \tag{2.5}$$

$$\frac{\partial \mathrm{E}_z}{\partial \mathrm{t}} = \frac{1}{\varepsilon_z}\left(\frac{\partial \mathrm{H}_y}{\partial \mathrm{x}} - \frac{\partial \mathrm{H}_x}{\partial \mathrm{y}} - \sigma_z E_z\right), \tag{2.6}$$

где ε и σ, μ, и σ_M описывают электрические и магнитные свойства объекта. Уравнения (2.1) - (2.6) являются основой алгоритма FDTD для моделирования распространения электромагнитных волн. Электрические и магнитные поля в дискретизованной версии располагаются в шахматном порядке во времени и пространстве. Электрические и магнитные поля разбиваются по времени с шагом $n\Delta t$ и $\left(n+\frac{1}{2}\right)\Delta t$, соответственно, а также смещены относительно друг друга в пространстве, как указано на рисунке 2.5. Используя приятые условные обозначения, дискретизованные поля во временной и пространственной области можно записать в следующем формате [15]:

$$H_x^{n+\frac{1}{2}}\left(i, j+\frac{1}{2}, k+\frac{1}{2}\right) = \frac{\mu_x - 0.5\Delta t\sigma M_x}{\mu_x + 0.5\Delta t\sigma M_x} H_x^{n-\frac{1}{2}}\left(i, j+\frac{1}{2}, k+\frac{1}{2}\right) + \\ + \frac{\Delta t}{\mu_x + 0.5\Delta t\sigma M_x}\left[\begin{array}{l} \dfrac{E_y^n\left(i, j+\frac{1}{2}, k+1\right) - E_y^n\left(i, j+\frac{1}{2}, k\right)}{\Delta z} \\ -\dfrac{E_z^n(i, j+1, k+1/2) - E_z^n(i, j, k+1/2)}{\Delta y} \end{array}\right] \tag{2.7}$$

$$H_y^{n+\frac{1}{2}}\left(i+\frac{1}{2}, j, k+\frac{1}{2}\right) = \frac{\mu_y - 0.5\Delta t\sigma M_y}{\mu_y + 0.5\Delta t\sigma M_y} H_y^{n-\frac{1}{2}}\left(i+\frac{1}{2}, j, k+\frac{1}{2}\right) + \\ + \frac{\Delta t}{\mu_y + 0.5\Delta t\sigma M_y}\left[\begin{array}{l} \dfrac{E_z^n\left(i+1, j, k+\frac{1}{2}\right) - E_z^n\left(i, j, k+\frac{1}{2}\right)}{\Delta x} \\ -\dfrac{E_x^n\left(i+\frac{1}{2}, j, k+1\right) - E_x^n\left(i+\frac{1}{2}, j, k\right)}{\Delta z} \end{array}\right] \tag{2.8}$$

$$H_z^{n+\frac{1}{2}}\left(i+\frac{1}{2}, j+\frac{1}{2}, k\right) = \frac{\mu_z - 0.5\Delta t\sigma M_z}{\mu_z + 0.5\Delta t\sigma M_z} H_z^{n-\frac{1}{2}}\left(i+\frac{1}{2}, j+\frac{1}{2}, k\right) + \\ + \frac{\Delta t}{\mu_z + 0.5\Delta t\sigma M_z}\left[\begin{array}{l} \dfrac{E_x^n\left(i+\frac{1}{2}, j+1, k\right) - E_x^n\left(i+\frac{1}{2}, j, k\right)}{\Delta y} \\ -\dfrac{E_y^n\left(i+\frac{1}{2}, j+1, k\right) - E_y^n\left(i, j+\frac{1}{2}, k\right)}{\Delta x} \end{array}\right] \tag{2.9}$$

$$E_x^{n+1}\left(i+\frac{1}{2},j,k\right)=\frac{\varepsilon_x-0.5\Delta t\sigma_x}{\varepsilon_x+0.5\Delta t\sigma_x}E_x^n\left(i+\frac{1}{2},j,k\right)+$$
$$+\frac{\Delta t}{\varepsilon_x+0.5\Delta t\sigma_x}\left[\begin{array}{l}\dfrac{H_z^{n+\frac{1}{2}}\left(i+\frac{1}{2},j+\frac{1}{2},k\right)-H_z^{n+\frac{1}{2}}\left(i+\frac{1}{2},j-\frac{1}{2},k\right)}{\Delta y}\\ -\dfrac{H_y^{n+\frac{1}{2}}\left(i+\frac{1}{2},j,k+\frac{1}{2}\right)-H_y^{n+\frac{1}{2}}\left(i+\frac{1}{2},j,k-\frac{1}{2}\right)}{\Delta z}\end{array}\right] \quad (2.10)$$

$$E_y^{n+1}\left(i,j+\frac{1}{2},k\right)=\frac{\varepsilon_y-0.5\Delta t\sigma_y}{\varepsilon_y+0.5\Delta t\sigma_y}H_x^n\left(i,j+\frac{1}{2},k\right)+$$
$$+\frac{\Delta t}{\varepsilon_y+0.5\Delta t\sigma_y}\left[\begin{array}{l}\dfrac{H_x^{n+\frac{1}{2}}\left(i,j+\frac{1}{2},k+\frac{1}{2}\right)-H_x^{n+\frac{1}{2}}\left(i,j+\frac{1}{2},k-\frac{1}{2}\right)}{\Delta z}\\ -\dfrac{H_z^{n+\frac{1}{2}}\left(i+\frac{1}{2},j+\frac{1}{2},k\right)-H_z^{n+\frac{1}{2}}\left(i-\frac{1}{2},j+\frac{1}{2},k\right)}{\Delta x}\end{array}\right] \quad (2.11)$$

$$E_z^{n+1}\left(i,j,k+\frac{1}{2}\right)=\frac{\varepsilon_z-0.5\Delta t\sigma_z}{\varepsilon_z+0.5\Delta t\sigma_z}H_z^n\left(i,j,k+\frac{1}{2}\right)+$$
$$+\frac{\Delta t}{\varepsilon_z+0.5\Delta t\sigma_z}\left[\begin{array}{l}\dfrac{H_y^{n+\frac{1}{2}}\left(i+\frac{1}{2},j,k+\frac{1}{2}\right)-H_y^{n+\frac{1}{2}}\left(i-\frac{1}{2},j,k+\frac{1}{2}\right)}{\Delta z}\\ -\dfrac{H_x^{n+\frac{1}{2}}\left(i,j+\frac{1}{2},k+\frac{1}{2}\right)-H_x^{n+\frac{1}{2}}\left(i,j-\frac{1}{2},k+\frac{1}{2}\right)}{\Delta y}\end{array}\right] \quad (2.12)$$

Уравнения с (2.7) по (2.12) не содержат каких-либо четких границ, и мы должны дополнить их соответствующим граничным условием для того, чтобы сократить расчетную область. При моделировании FDTD, наиболее часто используемые граничные условия: идеальный электрический проводник (PEC), идеальный магнитный проводник (PMC), поглощающее граничное условие (ABC), и периодическое граничное условие (PBC).

Электрические и магнитные поля в обновленных уравнениях (2.7) - (2.12) в соответствии с их положением в пространстве. Поля индексируются с нуля, а не с помощью смещения временного шага, связано это с тем, чтобы минимизировать использование памяти. Если есть n_x, n_y, и n_z клетки в x-, y-, и

z-направлениях, соответственно, массивы электрического поля E_x, E_yи E_z, и массивы магнитного поля H_x, H_y, и H_z записываются следующим образом:

$$E_x[n_x][n_y+1][n_z+1] \tag{2.13}$$

$$E_y[n_x+1][n_y][n_z+1] \tag{2.14}$$

$$E_z[n_x+1][n_y+1][n_z] \tag{2.15}$$

$$H_x[n_x+1][n_y][n_z] \tag{2.16}$$

$$H_y[n_x][n_y+1][n_z] \tag{2.17}$$

$$H_z[nx][ny][nz+1] \tag{2.18}$$

Электрические поля начинаются с нулевого индекса, распределенные на границе расчетной области, как это показано на рисунке 2.6 [20].

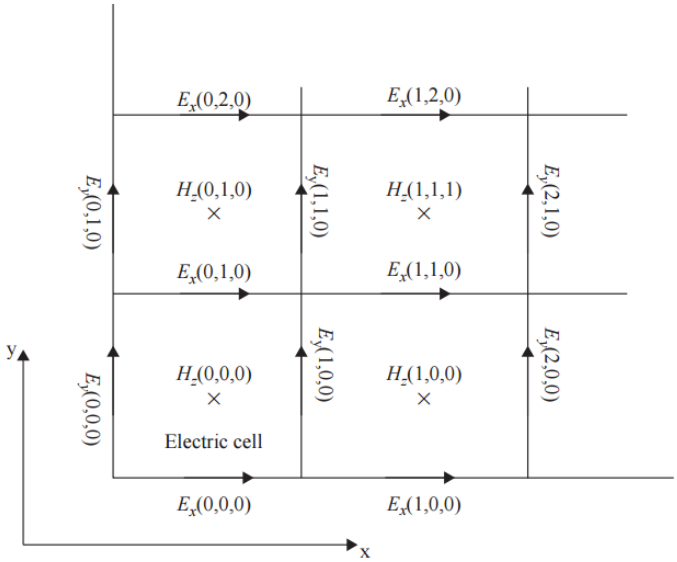


Рисунок 2.6 – Расположение электрических и магнитных полей в методе FDTD.

Одним из важных вопросов, которые должны решаться при смещении по времени в методе конечных разностей является стабильность решения во временной области. Стабильность алгоритма FDTD зависит от природы физической модели, используемой разностной схемы, а также качества

структуры сетки [5]. Для того чтобы, понять природу стабильности, дисперсия выражается следующим образом:

$$\omega = \frac{2}{\Delta t} sin^{-1}\left(c\Delta t \sqrt{\frac{1}{\Delta x^2} sin^2\left(\frac{k_x \Delta x}{2}\right) + \frac{1}{\Delta y^2} \sin^2\left(\frac{k_y \Delta y}{2}\right) + \frac{1}{\Delta z^2} sin^2(\frac{k_z \Delta z}{2})} \right) \quad (2.19)$$

Если ω является мнимым числом, электромагнитная волна, $\psi(t, \vec{r}) = \psi_0 e^{j(\omega t - \vec{k}, \vec{r})}$, то либо будет быстро затухать или будет расти в геометрической прогрессии и расходится, в зависимости от того, что мнимая часть ω является положительной или отрицательной. Для того чтобы гарантировать, чтобы ω являлось действительным числом нужно чтобы выражение внутри круглых скобок в (2.15) удовлетворяло условию [16]:

$$c\Delta t \sqrt{\frac{1}{\Delta x^2} sin^2\left(\frac{k_x \Delta x}{2}\right) + \frac{1}{\Delta y^2} \sin^2\left(\frac{k_y \Delta y}{2}\right) + \frac{1}{\Delta z^2} sin^2(\frac{k_z \Delta z}{2})} \leq 1 \quad (2.20)$$

Поскольку максимальное возможное значение синуса в квадрате равен 1, размер шага времени должен удовлетворять следующему условию:

$$\Delta t \leq \frac{1}{c\sqrt{\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} + \frac{1}{\Delta z^2}}} \quad (2.21)$$

чтобы решение было стабильным. Критерий выше, называется условием устойчивости. Уравнение (2.17) указывает, что размер временного шага определяется размером ячейки в x-, y-, и z-направления и скоростью электромагнитной волны в среде.

2.5 Параллельный метод FDTD для решения задачи рассеяния ЭМВ

Одна вычислительная единица может быть вычислительным ядром, либо CPU с несколькими ядрами, или вычислительным узлом, который содержит несколько CPU. Различное распределение вычислительных единиц значительно влияют на производительность распараллеливания, которое определяется объемом обмена информацией между вычислительными единицами. В параллельных вычислениях, исходная задача разбивается на мелкие части, которые присваиваются каждой вычислительной единице. Каждой

вычислительной единице в кластере, необходимо имитировать одну определенную подобласть, как показано на рисунке 2.7 [19].

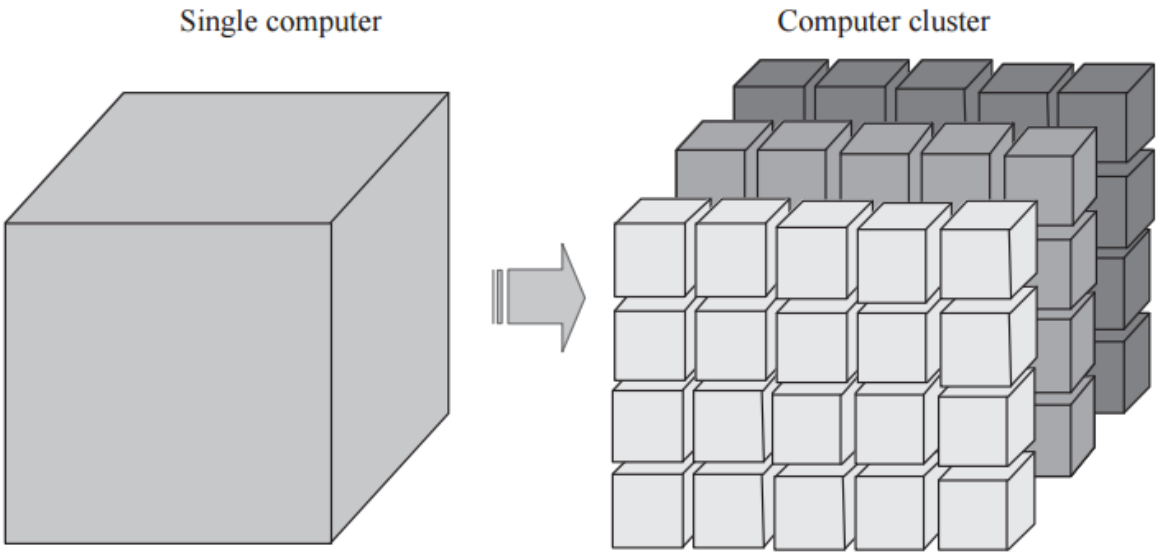


Рисунок 2.7 – Основная идея параллельной обработки FDTD

Каждая подобласть является зависимой и требует наличие информации от своих соседей для расчета полей на границе раздела подобластей. Порядок обмена информацией показан на рисунке 2.8 [19].

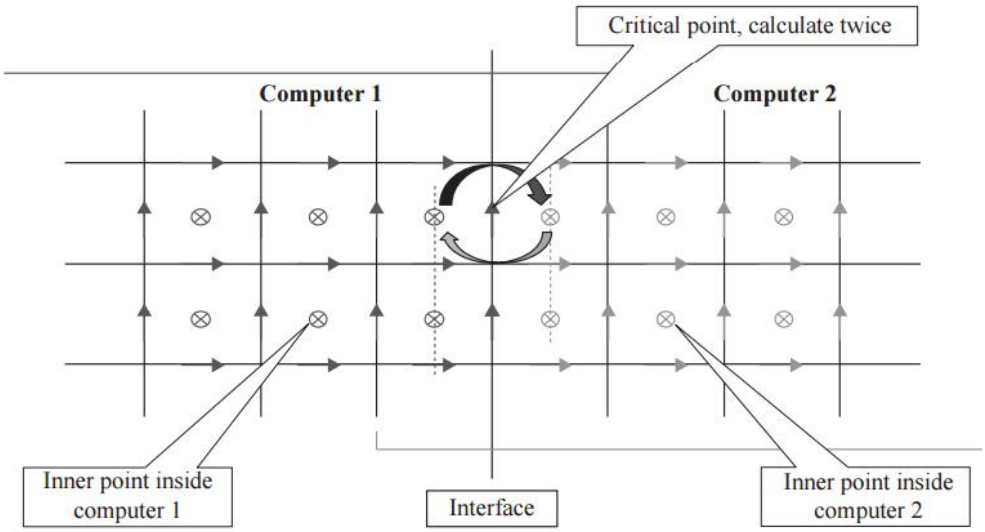


Рисунок 2.8 – Обмен информации на границе подобластей

Полное моделирование FDTD осуществляется в три этапа: предварительной обработки, моделирование объекта, а также пост- обработки.

На первом этапе, параллельный алгоритм FDTD формирует распределение объекта на основе его геометрии и указанной сетки распределения. На этом шаге, каждый вычислительный блок в кластере не должен ожидать другие блоки, так как ячейки независимы от своих соседей. Для объектов со сложной геометрией, сложность вычислений в каждой ячейке может существенно отличаться, а время обработки, которое требуется ячейке, может варьироваться на разных вычислительных машинах. Для того, чтобы ускорить процедуру предварительной обработки, количество подобластей не должно быть равно количеству вычислительных единиц. Кроме того, вычислительные единицы, вычисляющие простые подобласти могут работать одновременно с несколькими подобластями без необходимости ожидания других вычислительных единиц, как показано на рисунке 2.9 [19]. В отличие от предварительной обработки, параллельное обновление в каждом подобласти должно быть синхронизировано на каждом временном шаге. Таким образом, мы не допускаем, вычислительной единице перейти к другой подобласти. Данные пост-обработки не требуют обмена данными между вычислительными единицами.

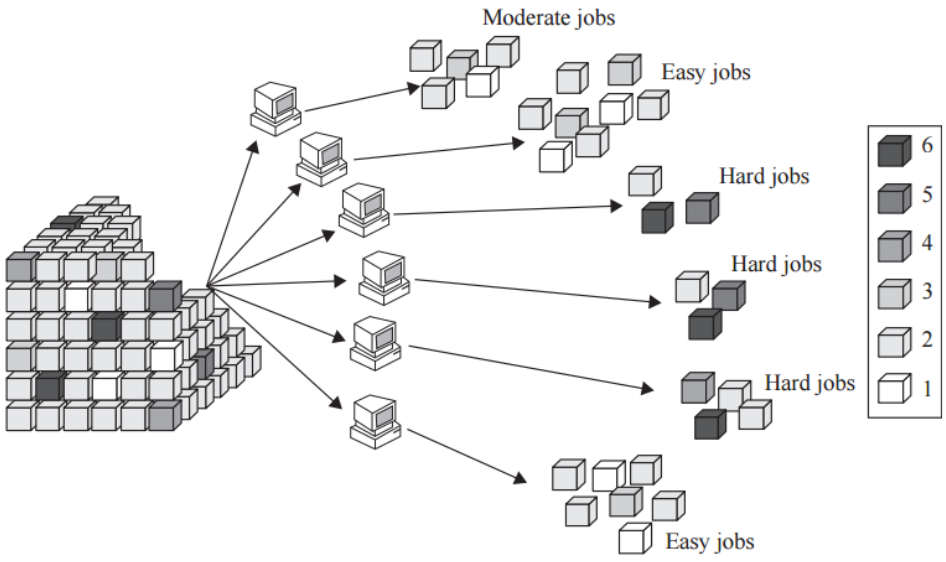


Рисунок 2.9 – Разбиение задачи на мелкие части на стадии предварительной обработки.

Так как обновление поля в методе FDTD требует информации об его соседях, это делает его высоко распараллеленным методом. Для линейного алгоритма FDTD, нахождение поля во всей расчетной области может быть решено путем возбуждения поля, чтобы удовлетворить нужное условие на границе области. Для параллельного алгоритма, поля на границе каждой подобласти неизвестны, но они рассчитываются путем заимствования некоторой информации из соседних подобластей. Для этого используется высокопроизводительная сеть для передачи информации из одной вычислительной единицы на другую на каждом временном шаге. В отличие от других методов моделирования, параллельный метод FDTD требует только расчета полей на границе раздела между смежными вычислительными единицами.

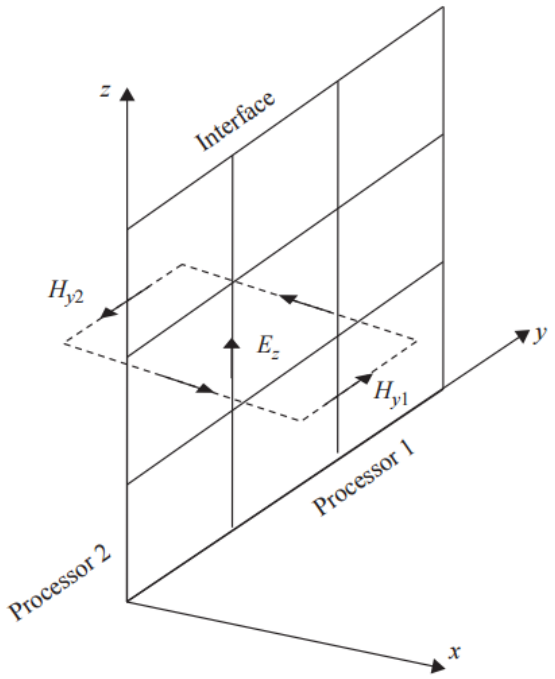


Рисунок 2.10 – Распределение электрических и магнитных полей между двумя соседними процессорами.

На рис. 2.10 электрическое поле E_z, расположено на границе раздела между процессорами 1 и 2 [19]. Для обновления этого электрического поля необходима информация о двух магнитных полях: H_{y1}, и H_{y2}, которые вычисляются процессорами 1 и 2, соответственно.

Магнитные поля H_{y1} и H_{y2} обмениваются информацией на каждом временном шаге. Обмен данными происходит только на границах раздела между двумя соседними подобластями.

2.6 Особенности параллельной реализации FDTD на CUDA

Следующий перечень рекомендаций принимается во внимание для оптимизации параллельного алгоритма FDTD на CUDA:

- структура алгоритма должна представлять максимальный параллелизм данных;
- обеспечение совместного доступа к глобальной памяти, когда это возможно;
- минимизировать использование глобальной памяти;
- использовать число потоков в блоке кратное 32, это обеспечивает оптимальную эффективность вычислений и способствует объединенному доступу к глобальной памяти [17].

Основное пространство памяти на GPU принадлежит глобальной памяти, и доступ к ней осуществляется путем транзакций операций чтения или записи размером в 32-, 64- или 128 байт, блоком потоков. Для лучшей производительности, нужно упорядочить эти операции. Рисунок 2.11 [18] иллюстрирует, непоследовательный доступ к ячейкам глобальной памяти потоками. На рисунке 2.12 [18] представлен упорядоченный доступ к глобальной памяти [20].

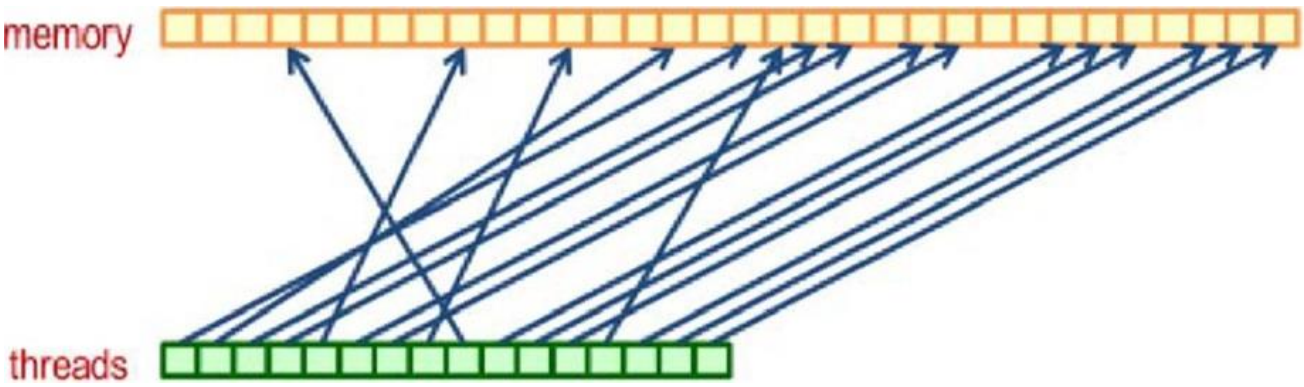


Рисунок 2.11 – Неупорядоченный доступ к ячейкам глобальной памяти потоками.

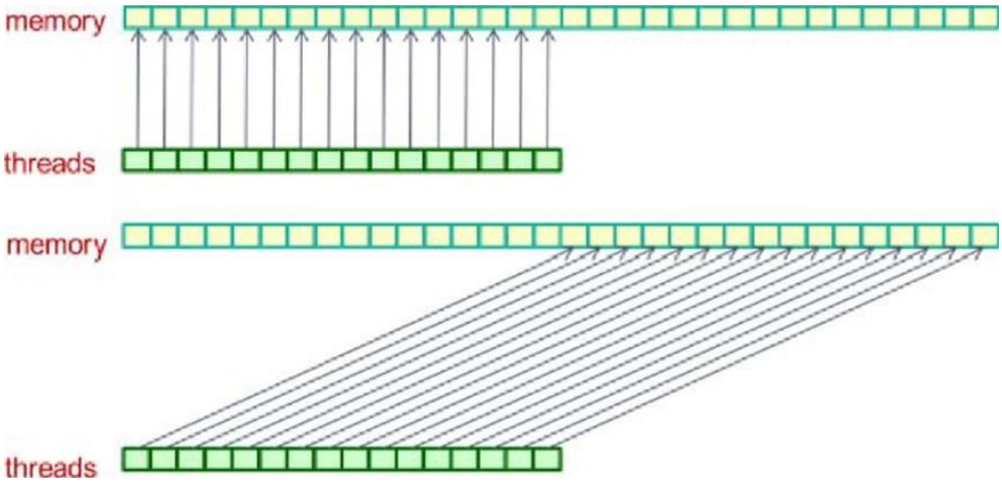


Рисунок 2.12 – Упорядоченный доступ к глобальной памяти.

Для достижения совместного доступа к глобальной памяти расчетная область расширяется в x-направлении добавлением ячеек таким образом, чтобы число ячеек стало кратным 16. Рисунок 2.13[20] иллюстрирует расширенную расчетную область. Благодаря расширению расчетной области обеспечивается совместный доступ к глобальной памяти, что минимизирует обращение к глобальной области.

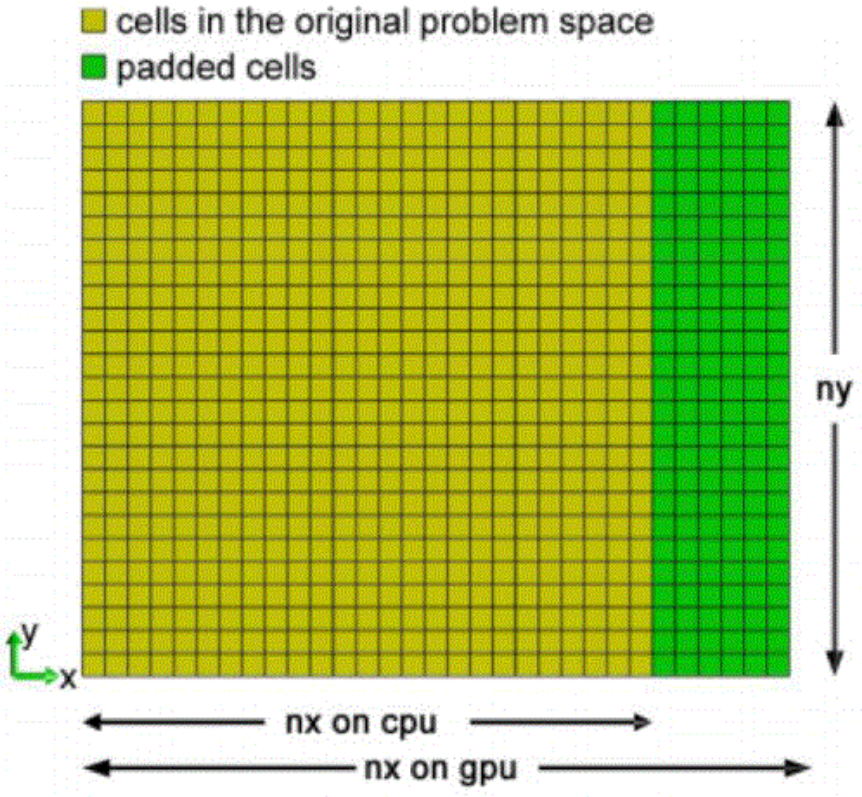


Рисунок 2.13 – Расширенная расчетная область.

GPU представляет собой отдельное вычислительное устройство, которое имеет свои процессоры, и также свою память. В терминологии CUDA, GPU называют устройством (device), а CPU именуется хостом. Если вычисления выполняются на GPU, соответствующие данные должны быть переданы в память устройства до начала вычисления. Основные вычисления – это нахождение электромагнитных полей в каждый момент времени, что требует наличия актуальных коэффициентов на вход. Перед началом вычислений данные должны быть переданы в память устройства (GPU).

В FDTD обновлять поля в ячейке можно независимо от других ячеек, следовательно, обновления полей в различных клетках, может выполняться параллельно друг с другом. Каждой ячейке сопоставляется поток, для выполнения вычислений поля внутри ячейки. В CUDA потоки образуют блок, а ряд блоков образуют сетку (grid). Для параллельного вычисления, каждой ячейке сопоставляется поток, потоки объединяются в блоки таким образом, чтобы сетка (grid) охватывала все ячейки расчетной области. Рисунок 2.14 иллюстрирует разбиение потоков на блоки [18]. Номер ячейки – это номер блока, а индекс указывает на номер потока.

7_4	7_5	7_6	7_7	8_0	8_1	8_2	8_3	8_4	8_5	8_6
6_0	6_1	6_2	6_3	6_4	6_5	6_6	6_7	7_0	7_1	7_2
4_4	4_5	4_6	4_7	5_0	5_1	5_2	5_3	5_4	5_5	5_6
3_0	3_1	3_2	3_3	3_4	3_5	3_6	3_7	4_0	4_1	4_2
1_4	1_5	1_6	1_7	2_0	2_1	2_2	2_3	2_4	2_5	2_6

Рисунок 2.14 – Сопоставление потоков с ячейками.

Ниже представлено формула (2.18) для обновления магнитного поля [18]:

$$H_y^n(i,j,k) = H_y^{n-\frac{1}{2}}(i,j,k) + \\ +C_{hyez} \times (E_z^n(i+1,j,k) - E_z^n(i,j,k)) + \\ + C_{hyex} \times \left((E_x^n(i,j,k+1) - E_x^n(i,j,k)\right) \quad (2.22)$$

По формуле (2.18) для расчета полей в ячейке, информация о соседних ячейках также необходима. Нужна информация о $E_z(i+1,j,k)$ и $E_x(i,j,k+1)$

для вычисления $H_y(i,j,k)$. Поток, который обрабатывает ячейку (i,j,k) может эффективно читать пространство памяти в ячейке $(i,j,k+1)$, так как обеспечивается совместный доступ. Однако доступ к памяти в ячейке $(i+1,j,k)$, не обеспечивается совместный доступ и является дорогим с точки зрения времени вычисления. В этом случае нужно использовать совместную память для эффективности. Потому что на устройстве, доступ к разделенной памяти намного быстрее, чем к локальной и глобальной памяти. Каждый поток может получить доступ к памяти связанной с ним и загрузить соответствующие данные в разделенную память. Когда данные будут загружены, они будут доступны потокам соседних ячеек.

Все равно существует проблема доступа к памяти, если потоку на границе блока необходимо получить доступ к клетке, которая принадлежит другому блоку потоков. Эта проблема решается, путем копирования данных из соседних блоков, на рисунке 2.15 [18] представлена схема работы.

Оператор sEy[ti] = Ey[fi] загружает блок данных в разделенную память и выражение sEy[blockDim.x+ti] = Ey[blockDim.x+fi] загружает соседний блок данных в разделенную память, а также синхронизуется работа потоков оператором __syncthreads ().

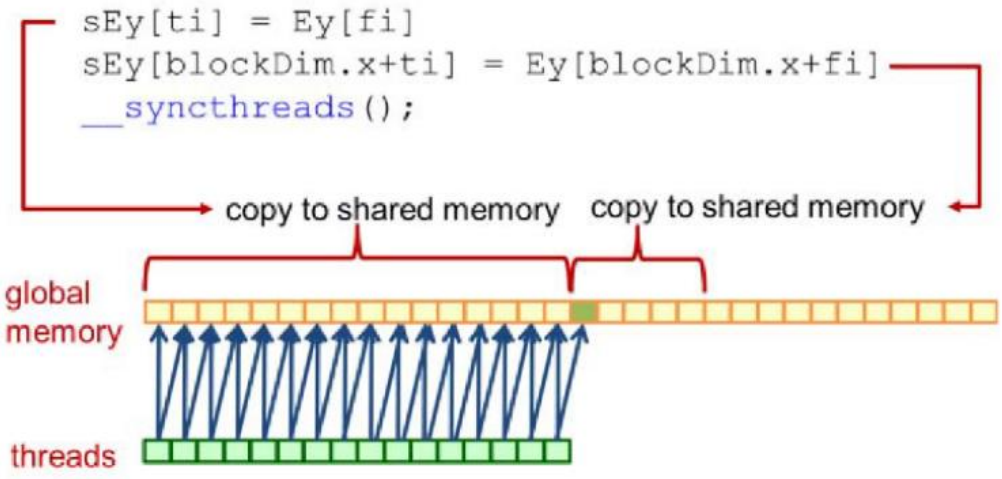


Рисунок 2.15 – Копирование данных из глобальной области памяти в разделенную память.

После произведения всех вычислений, данные полученные, копируются обратно в память хоста (CPU).

2.7 Программная реализация метода FDTD

Ниже на рисунке 2.16 представлен листинг кода функции main, в котором описан весь процесс вычислений. Функция initializeStartParameters() инициализирует начальные параметры: размер расчетной области, размер ячеек в x-, y- и z-направлениях, количество временных шагов. Далее вызывается функция setupProblemSpace(), которая отвечает за разбиение расчетной области. В зависимости от параметра run_on_gpu, изменяется разбиение расчетной области в x-направлении, делается это для обеспечения совместного доступа к глобальной области памяти. После разбиения расчетной области, происходит инициализация переменных, с помощью функции initializeVariables(). Функция startTimer() запускает таймер для вычисления времени работы программы. После запуска таймера происходит запуск процесса вычислений. В зависимости от параметра run_on_gpu, вычисления происходят на CPU либо на GPU. Если параметр run_on_gpu равен false, вычисления происходят с использованием центрального процессора, если true, то вычисления происходят на GPU.

```
int main()
{
    initializeStartParameters();
    setupProblemSpace();
    initializeVariables();
    startTimer();

    if(run_on_gpu)
    {
        setupGPU();
        runTimeMarchingLoopOnGPU();
        copyDataBackAndClearGPU();
    }
    else
    {
        runTimeMarchingLoopOnCPU();
    }
    stopTimer();
}
```

Рисунок 2.16 – Листинг кода функции main

Далее описывается действие программы, если входной параметр run_on_gpu равен false. Вызывается фукнция runTimeMarchingLoopOnCPU(), которая выполняет запуск процесса вычислений на центральном процессоре. Здесь находится цикл, который высчитывает электромагнитные поля в каждый момент времени, количество итераций цикла (временных шагов) зависит от входных параметров. На рисунке 2.17 представлен листинг кода функции runTimeMatchingLoopOnCPU().

```
void runTimeMarchingLoopOnCPU()
{
    for (int time_step = 0; time_step < number_of_time_steps; time_step++)
    {
        updateMagneticFields();
        updateElectricFields();
    }
}
```

Рисунок 2.17 – Листинг кода функции runTimeMarchingLoopOnCPU()

Функция updateMagneticFields() выполняет обновление магнитных полей на каждую итерацию цикла (временной шаг). Листинг кода функции updateMagneticFields() представлен на рисунке 2.18. В функции происходит обновление электрических полей в каждой точке расчетной области.

```
void updateMagneticFields()
{
    for (int i = 0; i < n_fields - z_offset; i++)
    {
        Hx[i] = Hx[i] + Chxey*(Ey[i + z_offset] - Ey[i]) + Chxez*(Ez[i + y_offset] - Ez[i]);
        Hy[i] = Hy[i] + Chyez*(Ez[i + x_offset] - Ez[i]) + Chyex*(Ex[i + z_offset] - Ex[i]);
        Hz[i] = Hz[i] + Chzex*(Ex[i + y_offset] - Ex[i]) + Chzey*(Ey[i + x_offset] - Ey[i]);
    }
}
```

Рисунок 2.18 – Листинг кода функции updateMagneticFields

Происходит обновление магнитных полей, в каждой точке пространства используя формулы (2.7) – (2.9). После вычисления магнитных полей в каждой ячейке расчетной области, рассчитываются электрические поля, используя

функцию updateEletricFields(), листинг кода которой представлен на рисунке 2.19.

```
void updateElectricFields()
{
    for (int i = z_offset; i < n_fields; i++)
    {
        Ex[i] = Ex[i] + Cexhz[i] * (Hz[i] - Hz[i - y_offset]) + Cexhy[i] * (Hy[i] - Hy[i - z_offset]);
        Ey[i] - Ey[i] + Ceyhx[i] * (Hx[i] - Hx[i - z_offset]) + Ceyhz[i] * (Hz[i] - Hz[i - x_offset]);
    }

    for (int i = y_offset; i< n_fields - z_offset; i++)
    {
        Ez[i] = Ceze[i] * Ez[i] + Cezhy[i] * (Hy[i] - Hy[i - x_offset]) + Cezhx[i] * (Hx[i] - Hx[i - y_offset]);
    }
}
```

Рисунок 2.19 – Листинг кода функции updateElectricFields

Если параметр run_on_gpu равен true, то вычисления происходят на GPU. Изначально нужно подготовить вычисления на GPU, за это отвечает функция setupGPU(), листинг которой представлен ниже на рисунке 2.20.

```
void setupGPU()
{
    if (copyArraysToGpuMemory() != 0)
    {
        printf("Ошибка при копирование данных на GPU!\n");
    }
    setThreadBlocks();
}
```

Рисунок 2.20 – Листинг кода функции setupGPU

В подготовку к вычислениям входит инициализация данных в памяти устройства и копирование данных в память устройства, путем вызова функции copyArraysToGpuMemory(). Инициализация памяти на устройстве происходит вызовов функции cudaMalloc(), а копирование данных на память устройства выполняет функция cudaMemcpy(). Фрагмент кода функции copyArraysToGpuMemory() представлен на рисунке 2.21.

```
et = cudaMalloc((void**)&dvEx, array_size);
et = cudaMalloc((void**)&dvEy, array_size);
et = cudaMalloc((void**)&dvEz, array_size);
et = cudaMalloc((void**)&dvHx, array_size);
et = cudaMalloc((void**)&dvHy, array_size);
et = cudaMalloc((void**)&dvHz, array_size);
cudaMemcpy(dvEx, Ex, array_size, cudaMemcpyHostToDevice);
cudaMemcpy(dvEy, Ey, array_size, cudaMemcpyHostToDevice);
cudaMemcpy(dvEz, Ez, array_size, cudaMemcpyHostToDevice);
cudaMemcpy(dvHx, Hx, array_size, cudaMemcpyHostToDevice);
cudaMemcpy(dvHy, Hy, array_size, cudaMemcpyHostToDevice);
cudaMemcpy(dvHz, Hz, array_size, cudaMemcpyHostToDevice);
```

Рисунок 2.21 – Фрагмент функции copyArraysToGpuMemory

После завершения копирования данных в память устройства, выполняется фукнция setThreadsBlocks(), которая распределяет потоки на блоки. Листинг кода функции setThreadsBlocks() представлен на рисунке 2.22.

```
void setThreadBlocks()
{
    int n_blocks = ((n_fields - z_offset) / maximum_threads_per_block) +
        ((n_fields - z_offset) % maximum_threads_per_block == 0 ? 0 : 1);
    block_eh = dim3(maximum_threads_per_block, 1, 1);
    grid_eh = dim3(n_blocks, 1, 1);
    shared_memory_size = 2 * (maximum_threads_per_block + 16) * sizeof(float);
}
```

Рисунок 2.22 – Листинг кода функции setThreadsBlocks

После выполнения функции setupGPU(), происходит выполнение основных вычислений электромагнитных полей, путем вызова функции runTimeMarchingLoopOnGPU(), которая вычисляет электромагнитные поля в каждый момент времени.

```
void runTimeMarchingLoopOnGPU()
{
    for (int time_step = 0; time_step < number_of_time_steps; time_step++)
    {
        updateMagneticFieldsOnGPU <<< grid_eh, block_eh, shared_memory_size >>>
            (Chxey, Chxez, Chyez, Chyex, Chzex, Chzey, dvEx, dvEy,
            dvEz, dvHx, dvHy, dvHz, y_offset, z_offset, n_fields);
        updateElectricFieldsOnGPU <<< grid_eh, block_eh, shared_memory_size >>>
            (dvCexhy, dvCexhz, dvCeyhz, dvCeyhx, dvCezhx, dvCezhy, dvCeze,
            dvEx, dvEy, dvEz, dvHx, dvHy, dvHz, y_offset, z_offset, n_fields);
    }
}
```

Рисунок 2.23 – Листинг кода функциии runTimeMarchingLoopOnGpu

Для обновления электромагнитных полей происходит запуск ядра (kernel), путем передачи имени (адреса) функции, объявленной со спецификатором __global__, передачи данных, полученных в результате выполнения функции setThreadsBlocks(): размерность решетки (grid) в блоках, количество потоков в блоке и размер разделенной памяти, а после передаются аргументы функции. Ядро – это параллельная часть алгоритма. В случае обновления магнитных полей передается имя функции updateMagneticFieldsOnGpu, листинг кода которой представлен на рисунке 2.24.

```
__global__ void updateMagneticFieldsOnGPU(float Chxey, float Chxez, float Chyez, float Chyex,
    float Chzex, float Chzey, float* Ex, float* Ey, float* Ez, float* Hx, float* Hy,
    float* Hz, int y_offset, int z_offset, int n_fields)
{
    extern  __shared__ float sE[];
    float *sEy = (float*)sE;
    float *sEz = (float*)&sEy[blockDim.x + 16];
    int ti = threadIdx.x;
    int fi = blockIdx.x * blockDim.x + threadIdx.x;

    sEy[ti] = Ey[fi];
    sEz[ti] = Ez[fi];
    if (ti < 16)
    {
        sEy[blockDim.x + ti] = Ey[blockDim.x + fi];
        sEz[blockDim.x + ti] = Ez[blockDim.x + fi];
    }
    __syncthreads();
    if (fi >= n_fields - z_offset) return;
    Hx[fi] = Hx[fi] + Chxey*(Ey[fi + z_offset] - sEy[ti]) + Chxez*(Ez[fi + y_offset] - sEz[ti]);
    Hy[fi] = Hy[fi] + Chyez*(sEz[ti + 1] - sEz[ti]) + Chyex*(Ex[fi + z_offset] - Ex[fi]);
    Hz[fi] = Hz[fi] + Chzex*(Ex[fi + y_offset] - Ex[fi]) + Chzey*(sEy[ti + 1] - sEy[ti]);
}
```

Рисунок 2.24 – Листинг кода функции updateMagneticFieldsOnGPU

Перед началом вычислений магнитных полей, происходит загрузка блоков данных из глобальной памяти в разделенную память, при необходимости происходит загрузка в разделенную область памяти данных соседнего блока из глобальной области памяти. После загрузки данных происходит барьерная синхронизация потоков в блоке путем вызова оператора __syncthreads(). Далее происходят вычисления магнитных полей.

Аналогично происходит запуск ядра в случае обновления электрических полей, передается имя функции updateElectricFieldsOnGPU, листинг кода которой представлен на рисунке 2.25, размерность решетки (grid) в блоках, количество потоков в блоке и размер разделенной памяти, и аргументы функции.

Процесс вычислений электрических полей происходит аналогичным образом, как и у магнитных, сначала происходит загрузка блоков данных из глобальной памяти в разделенную память, при необходимости происходит загрузка в разделенную область памяти данных соседнего блока из глобальной области памяти. После загрузки данных происходит барьерная синхронизация потоков в блоке путем вызова оператора __syncthreads(). После непосредственно происходит вычисление электрических полей.

```
__global__ void updateElectricFieldsOnGPU(float* Cexhy, float* Cexhz, float* Ceyhz, float* Ceyhx, float* Cezhx,
    float* Cezhy, float* Ceze, float* Ex, float* Ey, float* Ez, float* Hx, float* Hy,
    float* Hz, int y_offset, int z_offset, int n_fields)
{
    extern  __shared__   float sH[];
    float *sHy = (float*)sH;
    float *sHz = (float*)&sHy[blockDim.x + 16]; int ti = threadIdx.x;
    int fi = blockIdx.x * blockDim.x + threadIdx.x;
    sHy[ti + 16] = Hy[fi];
    sHz[ti + 16] = Hz[fi];
    if (ti < 16)
    {
        sHy[ti] = Hy[fi - 16];
        sHz[ti] = Hz[fi - 16];
    }
    __syncthreads();
    if (fi >= n_fields - z_offset) return;
    if (fi < y_offset) return;
    Ez[fi] = Ceze[fi] * Ez[fi] + Cezhy[fi] * (sHy[ti + 16] - sHy[ti + 15]) + Cezhx[fi] * (Hx[fi] - Hx[fi - y_offset]);
    if (fi < z_offset) return;
    Ex[fi] = Ex[fi] + Cexhz[fi] * (sHz[ti + 16] - Hz[fi - y_offset]) + Cexhy[fi] * (sHy[ti + 16] - Hy[fi - z_offset]);
    Ey[fi] = Ey[fi] + Ceyhx[fi] * (Hx[fi] - Hx[fi - z_offset]) + Ceyhz[fi] * (sHz[ti + 16] - sHz[ti + 15]);
}
```

Рисунок 2.25 – Листинг кода функции updateElectricFieldsOnGPU

После вычисления электромагнитных полей в каждый момент времени, нужно скопировать данные в память хоста (CPU). Для этого используется функция copyDataBackAndClearGPU, фрагмент кода которой представлен ниже на рисунке 2.26.

```
cudaMemcpy(Ex, dvEx, array_size, cudaMemcpyDeviceToHost);
cudaMemcpy(Ey, dvEy, array_size, cudaMemcpyDeviceToHost);
cudaMemcpy(Ez, dvEz, array_size, cudaMemcpyDeviceToHost);
cudaMemcpy(Hx, dvHx, array_size, cudaMemcpyDeviceToHost);
cudaMemcpy(Hy, dvHy, array_size, cudaMemcpyDeviceToHost);
cudaMemcpy(Hz, dvHz, array_size, cudaMemcpyDeviceToHost);

cudaFree(dvEx);
cudaFree(dvEy);
cudaFree(dvEz);
cudaFree(dvHx);
cudaFree(dvHy);
cudaFree(dvHz);
```

Рисунок 2.26 – Фрагмент кода функции copyDataBackAndClearGPU

Для копирования данных с устройства на хост используется фукнция cudaMemcpy, последний параметр которой является типом передачи данных в данном случае это cudaMemcpyDeviceToHost. Для освобождения памяти на устройстве, используется функция cudaFree, которой передается единственный параметр – указатель на область памяти. После копирования данных обратно на память хоста, вызывается функция stopTimer, которая останавливает таймер времени работы программы.

Для тестирования корректности работы алгоритмов было решено сравнивать с аналитическим решением для тела правильной формы, а именно для сферы, полученной путем решения скалярного волнового уравнения, представленного в формуле (1.17). Радиус сферы $a = 10$, размер расчетной области в x-, y- и z-направлениях $N_x = N_y = N_z = 20$, размер ячейки в x-, y- и z- направлениях $\Delta x = \Delta y = \Delta z = 1$, количество временных шагов $t_{max} = 10$. Рассеянные поля в дальней зоне определяются E_φ, E_θ [13]. Значения E_φ= (-4.13553e+052, -1.61117e+052) и E_θ = (-7.65412e+052, -2.98197e+052) были

получены аналитическим решением. Ниже на рисунке 2.27 представлены результаты работы алгоритма на CPU и GPU.

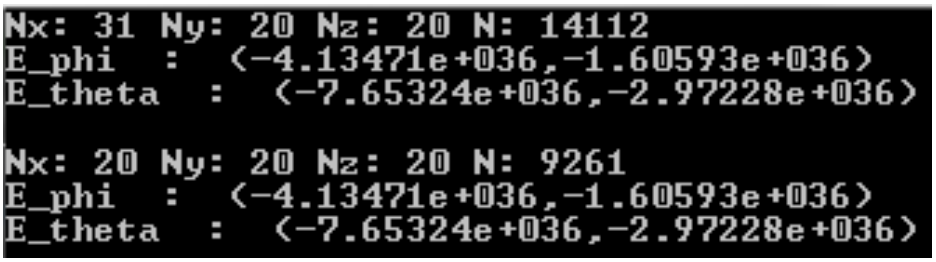


Рисунок 2.27 — Результаты работы алгоритма на GPU и CPU

В первой половине рисунка 2.27 представлены результаты вычислений на GPU, а во второй половине результаты вычислений на GPU. Как можно заметить значения, полученные на GPU и CPU, следовательно, можно сделать вывод о том, что потоки правильно синхронизируются. Значения, полученные в ходе вычислений отличаются от аналитических. Алгоритм работает точнее, если уменьшить размер ячейки. Ниже на рисунке 2.28 представлен результат работы алгоритма при размере ячеек в x-, y- и z- направлениях $\Delta x = \Delta y = \Delta z = 0.5$.

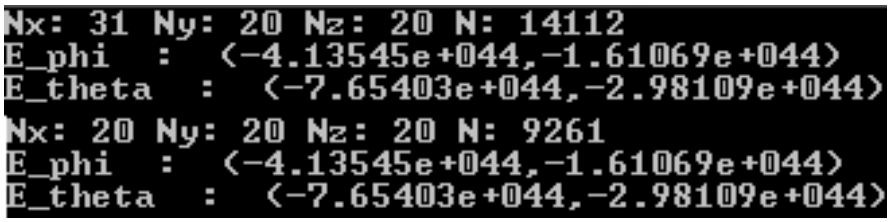


Рисунок 2.28 – Результаты работы алгоритма на GPU и CPU

Как можно заметить из рисунка 2.28 точность решения при уменьшении размера ячеек стала выше.

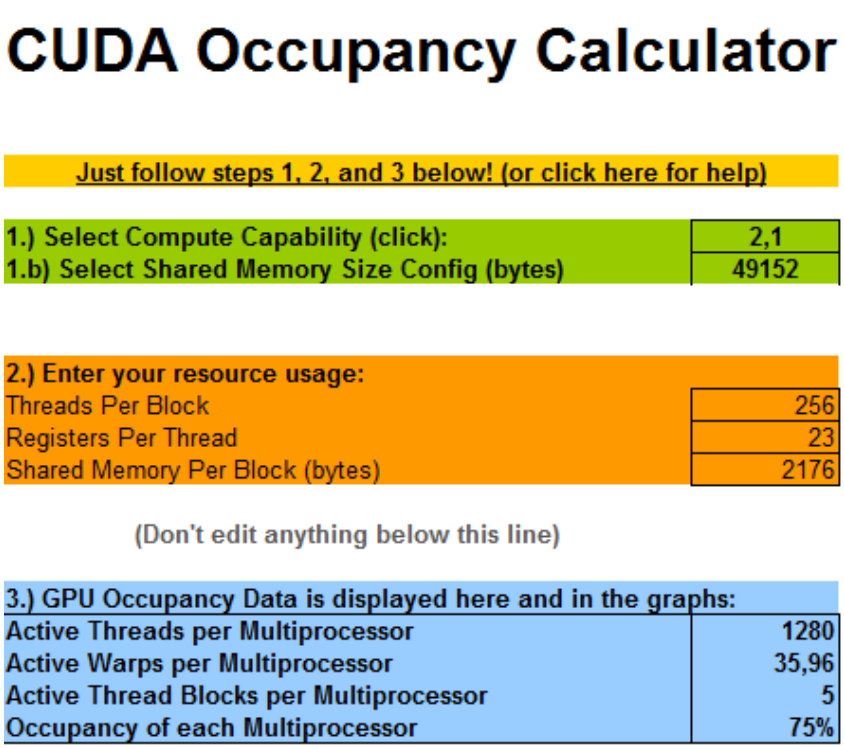
В данной главе был представлен численный метод решения задачи рассеяния ЭМВ – FDTD. Способ его распараллеливания и особенности ускорения вычислений на графических процессорах, с помощью технологии параллельных вычислений CUDA. Также была представлена реализация алгоритма, тестирование и проверка корректности алгоритма.

Глава 3 Оценка полученных результатов

3.1 Оценка эффективности работы программы на CUDA

Эффективность программы определяется использованием двух ресурсов. Первый из них – это время работы программы, а второй параметр – память. Время исполнения является более важным фактором для программиста, так как в большинстве случаев программа оценивается количеством машинного времени, необходимого для ее выполнения.

Эффективность программы с использованием технологии CUDA зависит от двух параметров. Первый параметр – это работа с глобальной памятью. Дело в том, что латентность глобальной памяти очень высокая и составляет порядка 400-600 тактов. Для оптимизации эффективности требуется минимизировать использование глобальной области памяти, что и было сделано в данной реализации, путем использования разделенной памяти. Вдобавок к этому использовались объединенные (coalesced) запросы в глобальную память, которые сокращают количество реальных запросов. Объединённый запрос – это когда одновременный доступ к памяти потоками во время выполнения одного чтения или записи, соединяется в одну транзакцию объемом памяти 32, 64 или 128 байт. Вторым параметром является gpu occupancy. Занятость GPU (Occupancy) – отношение количества текущих загруженных нитей на мультипроцессоре к теоретическому максимуму. Занятость может изменяться в зависимости от размера блоков, размера сетки. Компания NVIDIA выпустила калькулятор CUDA Occupancy Calculator, который позволяет рассчитать приблизительную степень занятости. С помощью данного калькулятора была проанализирована occupancy для размеров блоков в 256, 512 и 1024 потока. В ходе анализа было выявлено, что размер блока равным 256 потоков, показывает наибольший результат, равный 75 процентам. Можно остановиться на 75 процентах, ведь дальнейшее повышение загруженности не всегда приводит к более высокой производительности [12]. Результаты для 512 и 1024 потоков в блоке получились в районе 60 процентов. Ниже на рисунке 3.1 представлен результат работы CUDA Ocuppancy Calculator для размера блока в 256 потоков.



CUDA Occupancy Calculator

Just follow steps 1, 2, and 3 below! (or click here for help)

1.) Select Compute Capability (click):	2,1
1.b) Select Shared Memory Size Config (bytes)	49152

2.) Enter your resource usage:	
Threads Per Block	256
Registers Per Thread	23
Shared Memory Per Block (bytes)	2176

(Don't edit anything below this line)

3.) GPU Occupancy Data is displayed here and in the graphs:	
Active Threads per Multiprocessor	1280
Active Warps per Multiprocessor	35,96
Active Thread Blocks per Multiprocessor	5
Occupancy of each Multiprocessor	75%

Рисунок 3.1 – Результат работы CUDA Occupancy Calculator

Для того чтобы проверить загруженность на практике нужно прибегнуть к профилированию. Профилирование – это сбор различных характеристик работы приложения для последующего анализа, с целью повышения эффективности. С помощью профилирования происходит поиск так называемых «узких мест», в которых возможно повысить эффективность программы. Для профилирования NVIDIA выпустила инструмент NVIDIA Visual Profiler [11]. Ниже на рисунке 3.2 представлены данные полученные в процессе профилирования с помощью NVIDIA Visual Profiler. В верхней половине указаны данные для функции updateEletricFields, где значение загруженности составляет 74,9 процентов, как и было, подсчитано с помощью калькулятора, а в нижней половине данные для функции updateMagneticFields, где значение загруженности составляет 92,7 процентов, что превысило ожидания.

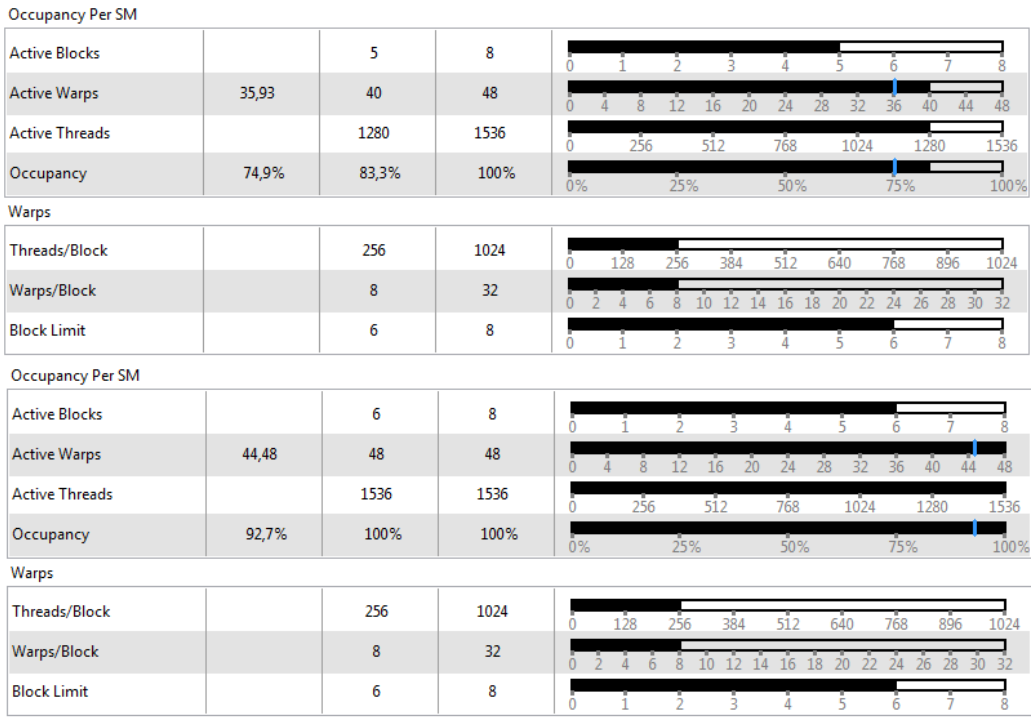


Рисунок 3.2 — Данные из профайлера

3.2 Сравнение времени работы метода FDTD на CPU и GPU

Сравнения двух реализаций алгоритма FDTD производились на видеокарте NVIDIA GT 540M и процессоре Intel Core i5 – 2430M, технические характеристики, которых представлены в таблицах 3.1 и 3.2. Эксперимент производился на операционной системе Microsoft Windows 7 (Service Pack 1) и установленным драйвером NVIDIA CUDA compiler версии 7.5.17.

Таблица 3.1 – Технические характеристики GPU NVIDIA GT 540M

Характеристика	Значение
Количество мультипроцессоров, шт.	2
Размер видеопамяти, Мб	1024
Максимальное число потоков в блоке, шт.	1024
Максимальная размерность блока потоков (x, y, z), шт.	1024x1024x64
Максимальная размерность сетки блоков, шт.	65535x65535x65535
Тактовая частота ядра, МГц	672
Тактовая частота памяти, МГц	900

Таблица 3.2 – Технические характеристики CPU Intel Core i5- 2430M

Характеристика	**Значение**
Тактовая частота ядра, ГГц	2.4
Тактовая частота шины CPU, ГТ/с	5
Кэш L1, Кб	128
Кэш L2, Кб	512
Кэш L3, Кб	3072

В ходе эксперимента расчетная область бралась в размере от 10 до 200 ячеек в каждом из трех направлений ($N_x = N_y = N_z = N$), тогда расчетная область принимает размер ($N_x \times N_y \times N_z$), дискретизация по времени составляет N_t = 5000. Результаты времени работы алгоритма приведены в таблице 3.3. Рисунок 3.3 иллюстрирует зависимость времени выполнения от числа узлов сеточной области.

Таблица 3.3 – Время выполнения алгоритма

Количество ячеек в каждом из направлений (N)	**Время выполнения, с**	
	CPU	**GPU**
10	0,13	0,9
20	0.8	1,08
30	2,4	1,4
40	5,8	2,2
50	12,5	3,7
75	84,6	8,9
100	168,0	20,7
125	325,2	36,2
150	624,2	65,2
175	1102,9	96,3
200	2016,1	147,5

Размер блока равен ($MTPB \times 1 \times 1$), где MTPB – это максимальное количество потоков в блоке равное 256, размер сетки выбирается по формуле:

$$\frac{\left(\left(N_{fields} - (N_x + 1) * \left(N_y + 1\right)\right)\right)}{MTPB}, \tag{3.1}$$

где N_{fields} – общее количество ячеек;

N_x, N_y – количество ячеек в x- и y- направлениях;

MTPB – максимальное количество потоков в блоке.

Размер разделенной памяти выбирается по формуле, указанной ниже:

$$2 * (MTPB + 16) * sizeof(float), \quad (3.2)$$

где MTPB – максимальное количество потоков в блоке;

sizeof(float) – размер в байтах типа float.

Рисунок 3.3 – Зависимость времени выполнения от числа ячеек расчетной области

Как видно из рисунка 3.3, зависимость времени выполнения от размеров расчетной области на GPU – линейная, а для алгоритма на CPU – кубическая. Кубическая зависимость времени работы алгоритма от размера расчетной области объясняется тем, что с ростом увеличения расчетной области число скалярных операций увеличивается пропорционально кубу размера сеточной области. Для расчетной области малых размеров существенной разницы нету, где производить вычисления, это связано с возможностью CPU быстро обрабатывать небольшое количество данных, которое может храниться в кэше (cache). С ростом расчетной области большая часть времени затрачивается на

передачу данных из оперативной памяти в кэш. Для наглядности на рисунке 3.4 представлена зависимость времени выполнения работы алгоритма от размеров расчетной области с использованием логарифмической шкалы.

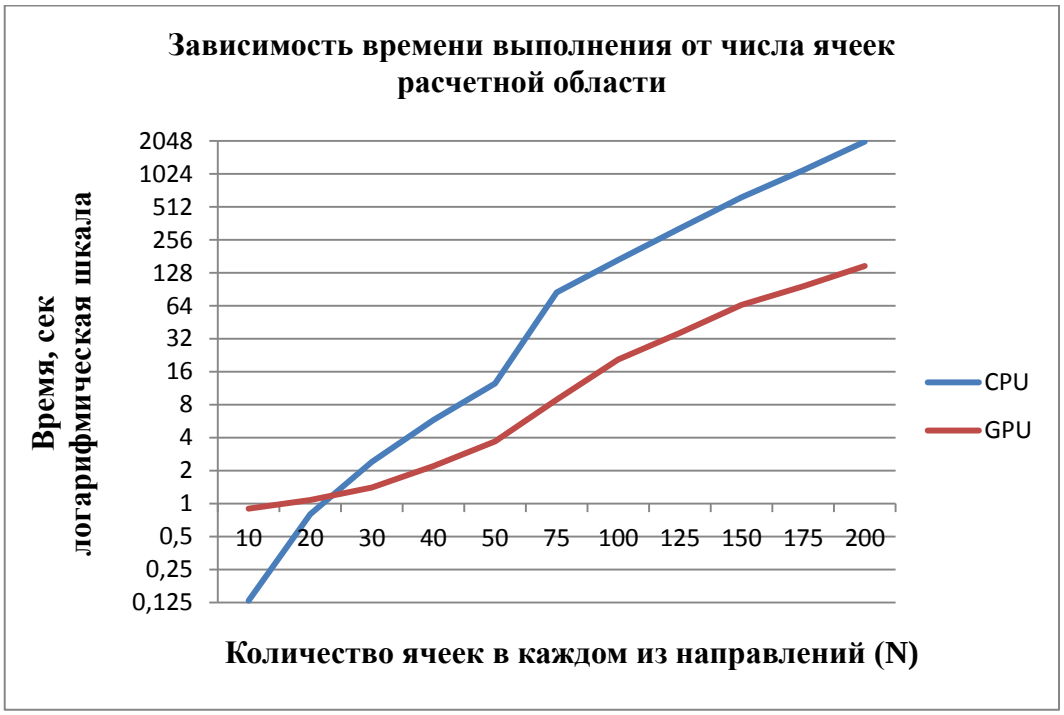


Рисунок 3.4 – Зависимость времени работы алгоритма от размеров расчетной области

При больших размерах расчетной области использование алгоритма FDTD с использованием технологии CUDA, даёт существенное ускорение относительно CPU: для размера расчетной области в 100 ячеек в x-, y-, z-направлениях в 8 раз, а для размеров в 200 ячеек в 13 раз. Отсутствие выигрыша по времени при использовании GPU в случаях, когда размеры исходной области не превышают 20 ячеек во всех трех направлениях, обусловлено значительными накладными расходами на вызов ядра при небольшом объеме обрабатываемых данных, большая часть мультипроцессоров простаивает, следовательно, значение gpu occupancy низкое и происходит нерациональное использование ресурсов GPU.

В этой главе было проанализирована эффективность программы с использованием технологии параллельных вычислений CUDA, с помощью

NVIDIA Visual Profiler. А также сравнение скорости работы алгоритмов на CPU и алгоритма GPU.

Заключение

В ходе проделанной работы были изучены математические основы построения модели отражения ЭМВ, задача рассеяния ЭМВ для объектов правильной формы и численные методы решения для объектов произвольной формы.

В ходе работы был реализован численный метод FDTD линейно на CPU и параллельно на GPU с помощью технологии параллельных вычислений CUDA. В результате исследования мы пришли к выводу что, использование GPU и технологии CUDA для неграфических вычислений является эффективным решением для ускорения вычислений, когда есть необходимость в большом объеме вычислений. При разработке параллельной программы с помощью технологии CUDA очень важно учитывать архитектуру, только с учетом архитектуры можно достичь высокой эффективности программы.

Для задач с небольшой расчетной областью, нецелесообразно использовать вычисления на GPU, так как большие накладные расходы на создание и управление потоками, вызов ядра и большая часть мультипроцессоров простаивает, для задач маленькой размерности хорошо справится CPU. При увеличении расчетной области применение вычислений на GPU, начинает себя оправдывать, при расчетной области размером в 200 ячеек во всех трех направлениях получили выигрыш в 13 раз. Дело в том что, зависимость времени от размера расчетной области на CPU кубическая, это обусловлено тем, что число операций прямо пропорционально кубу размера сеточной области. Так что можно прийти к выводу, что применение GPU эффективно и рационально для больших размеров расчетной области.

Список использованной литературы

Учебники и учебные пособия

1. Боресков, А. В. Основы работы с технологией CUDA / А. В. Боресков, А. А. Харламов. – М.: ДМК Пресс, 2010. – 232 с.

2. Воробьева, А. А. Дистанционное зондирование Земли: учебное пособие / А. А. Воробьева; Петербургский Национальный Исследовательский Университет Информационных Технологий, Механики и Оптики. – СПб.: Изд-во С. – Петерб. ун-та ИТМО, 2012. – 14 с.

3. Григорьев, А. Д. Методы вычислительной электродинамики / А. Д. Григорьев. – М.: ФИЗМАТЛИТ, 2012. — 432 с.

4. Ландау, Л. Д. Теоретическая физика. Теория поля. Том 2 / Л. Д. Ландау, Е. М. Лифшиц. – М.: ФИЗМАТЛИТ 2012 – 536 с.

5. Мареев, В. В. Основы методов конечных разностей / В. В. Мареев, Е. Н. Станкова. – СПб.: Изд-во С. – Петерб. ун-та, 2012. 64 с.

6. Оптическая биомедицинская диагностика. В 2 т. Т. 2 / Пер. с англ. под ред. В.В. Тучина. – М.: ФИЗМАТЛИТ, 2007. – 368 с.

7. Параллельные вычисления на GPU. Архитектура и программная модель CUDA: учебное пособие / А. В. Боресков и др. Предисл.: В.А. Садовничий. – М.: Издательство Московского университета, 2012. – 336с.

8. Токарева, О. С. Обработка и интерпретация данных дистанционного зондирования Земли: учебное пособие / О. С. Токарева; Томский политехнический университет. – Томск: Изд-во политехнического университета, 2010. – 148 с.

Периодические издания

9. Вицентий, А. В. Применение дистанционного зондирования Земли и космических технологий для развития арктических и субарктических территорий Российской Федерации [Текст] / А.В. Вицентий // Труды Кольского научного центра РАН, №5, 2013. – С. 40-45.

10. Мельников, Е. В. Инновационные георадарные технологии изучения подповерхностной структуры и состояния природно-технических

систем [Текст] / Е. В. Мельников, А. И. Калашник // Вестник Кольского научного центра РАН, №3, 2010. – С. 4 – 8.

Электронные ресурсы

11. CUDA Toolkit Documentation v7.5 [Electronic resource]: [Документация по набору инструментов CUDA]. – Electronic data.–NVidia Corp., [2015]. – Mode of access: http://docs.nvidia.com/cuda/

12. OpenCL Best Practices Guide [Electronic resource]: [Руководство по OpenCL]. – Electronic data. – NVidia Corp., [2011]. – Mode of access: http://camlunity.ru/swap/Library/Conflux/OpenCL/NVIDIA_OpenCL_Best_Practices_Guide.pdf

13. Каюмов, И.И. Моделирование рассеяния электромагнитных волн сферой в случае наноразмерности: дипломная работа / И.И. Каюмов; [Место защиты: Казанский Федеральный Университет]. – Казань, 2015. – 73 с. [Электронный ресурс]: http:// kpfu.ru/portal/docs/F1700892384/Kayumov.pdf

Литература на иностранном языке

14. Inan U. S. Numerical Electromagnetics: The FDTD Method / U. S. Inan, R. A. Marshall // Cambridge University Press, 1 edition, 2011 – 404 p.

15. Schneider J. B. Understanding the Finite-Difference Time-Domain Method / J. B. Schneider // Источник: www.eecs.wsu.edu/~schneidj/ufdtd – 2015

16. Sullivan D. M. Electromagnetic Simulation Using the FDTD Method / D. M. Sullivan // Wiley-IEEE Press, 2 edition, 2013 - 192 p.

17. V. Demir and A. Elsherbeni, Compute Unified Device Architecture (CUDA) Based FiniteDifference Time-Domain (FDTD) Implementation, Journal of the Applied Computational Electromagnetics Society (ACES), Vol. 25, No. 4, April 2010, pp. 303-314.

18. Yu W. Advanced Computational Electromagnetic Methods and Applications/ W. Yu, W. Li, A. Elsherbeni and Y. Rahmat - Samii // Artech House, 2015 - 574 p.

19. Yu W. Advanced FDTD Method: Parallelization, Acceleration, and Engineering Applications / W. Yu, X. Yang, Y. Liu, R. Mittra, A. Muto // Artech

House Electromagnetic Analysis, 1 edition, 2011 - 254 p.

20. Yu W. VALU, AVX and GPU Acceleration Techniques for Parallel FDTD Methods / W. Yu, X. Yang and W. Li // The ACES Series on Computational Electromagnetics and Engineering (CEME) - SciTech Publishing 2014 - 300 p.

Приложение А

Листинг кода файла fdtd.cu

```
#define _USE_MATH_DEFINES // for C
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

#include <cuda.h>
#include <cuda_runtime_api.h>
#include <device_launch_parameters.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <ctime>

using namespace std;

int search_time = 0; // Время в миллисекундах
int number_of_time_steps = 5000; // Количество временных шагов

float cell_size_x = 0.0004064; // Размер ячейки в направлении x-
float cell_size_y = 0.0004233; // Размер ячейки в направлении y-
float cell_size_z = 0.00265; // Размер ячейки в направлении z-

int c0 = 299792458; // Скорость света
float free_space_permeability = 4 * M_PI * pow(10, -7); // Константа распространения в
свободном пространстве
float vacuum_permittivity = 1 / (4 * M_PI * c0*c0) * pow(10, 7); // Диэлектрическая
проницаемость вакума

float box_size_x = 0.028448; // Размер расчетной области в направлении x-
float box_size_y = 0.027938; // Размер расчетной области в направлении y-
float box_size_z = 0.003445; // Размер расчетной области в направлении z-

float * Hx, * Hy, *Hz; // Магнитные поля
float * Ex, * Ey, * Ez; // Электрические поля
float Chxey, Chxez, Chyez, Chyex, Chzex, Chzey, * Cexhz, * Cexhy, * Ceyhx, * Ceyhz, * Ceze,
* Cezhy, * Cezhx; // Коэффициенты

int x_offset, y_offset, z_offset; // Смещение по x, y и z

unsigned int n_fields; // Количество ячеек

int nx, ny, nz; // Количество ячеек по x, y и z;
int nx_cpu, nx_gpu;// Количество ячеек по x на CPU и GPU
int maximum_threads_per_block; // Максимальное количество потоков в блоке
float dx, dy, dz, dt; // dt, dy, dz, dt

// Переменные с префисом dv- для хранения на GPU
float * dvEx, * dvEy, * dvEz, * dvHx, * dvHy, * dvHz, * dvChxey, * dvChxez, * dvChyez, *
dvChyex, * dvChzex, * dvChzey, *dvCexhz, *dvCexhy, *dvCeyhx, *dvCeyhz, *dvCeze, *dvCezhy,
*dvCezhx;

dim3 block_eh; // Блок потоков
dim3 grid_eh; // Решетка

int shared_memory_size;

bool run_on_gpu = false; // Параметр

// Разбиение расчетной области
void setupProblemSpace()
```

```
{
	nx_cpu = round(box_size_x / dx);
	ny = round(box_size_y / dy);
	nz = round(box_size_z / dz);
	nx_gpu = (16 * (nx_cpu / 16) + 15);

	if (run_on_gpu)
		nx = nx_gpu;
	else
		nx = nx_cpu;

	n_fields = (nx + 1) * (ny + 1) * (nz + 1);
}

//Инициализация переменных
void initializeVariables(){
	Ex = new float[n_fields]();
	Ey = new float[n_fields]();
	Ez = new float[n_fields]();

	Hx = new float[n_fields]();
	Hy = new float[n_fields]();
	Hz = new float[n_fields]();

	Cexhy = new float[n_fields]();
	Cexhz = new float[n_fields]();
	Ceyhx = new float[n_fields]();
	Ceyhz = new float[n_fields]();
	Ceze = new float[n_fields]();
	Cezhx = new float[n_fields]();
	Cezhy = new float[n_fields]();

	for (int i = 0; i < n_fields; i++){

		Ex[i] = 0.0;
		Ey[i] = 0.0;
		Ez[i] = 0.0;
		Hx[i] = 0.0;
		Hy[i] = 0.0;
		Hz[i] = 0.0;

		Cexhy[i] = 0.0;
		Cexhz[i] = 0.0;
		Ceyhx[i] = 0.0;
		Ceyhz[i] = 0.0;
		Cezhx[i] = 0.0;
		Cezhy[i] = 0.0;
		Ceze[i] = 0.0;
	}

	maximum_threads_per_block = 256;
}
// Задание старторвых параметров
void initializeStartParameters(){
	dx = cell_size_x;
	dy = cell_size_y;
	dz = cell_size_z;

	dt = 1;

	x_offset = 1;
	y_offset = nx + 1;
	z_offset = (nx + 1) * (ny + 1);
```

```
	Chxey = dt / free_space_permeability * dz;
	Chxez = - dt / free_space_permeability * dy;
	Chyex = dt / free_space_permeability * dx;
	Chyez = -dt / free_space_permeability * dz;
	Chzex = dt / free_space_permeability * dy;
	Chzey = -dt / free_space_permeability * dx;

	for (int i = 0; i < n_fields; i++){
		Cexhy[i] = -dt / vacuum_permittivity * dz;
		Cexhz[i] = dt / vacuum_permittivity * dy;
		Ceyhx[i] = dt / vacuum_permittivity * dz;
		Ceyhz[i] = -dt / vacuum_permittivity * dx;
		Cezhx[i] = -dt / vacuum_permittivity * dy;
		Cezhy[i] = dt / vacuum_permittivity * dz;
	}
}

void updateMagneticFields()
{
	for (int i = 0; i < n_fields - z_offset; i++)
	{
		Hx[i] = Hx[i] + Chxey*(Ey[i + z_offset] - Ey[i]) + Chxez*(Ez[i + y_offset] -
Ez[i]);
		Hy[i] = Hy[i] + Chyez*(Ez[i + x_offset] - Ez[i]) + Chyex*(Ex[i + z_offset] -
Ex[i]);
		Hz[i] = Hz[i] + Chzex*(Ex[i + y_offset] - Ex[i]) + Chzey*(Ey[i + x_offset] -
Ey[i]);
	}
}

void updateElectricFields()
{
	for (int i = z_offset; i < n_fields; i++)
	{
		Ex[i] = Ex[i] + Cexhz[i] * (Hz[i] - Hz[i - y_offset]) + Cexhy[i] * (Hy[i] -
Hy[i - z_offset]);
		Ey[i] = Ey[i] + Ceyhx[i] * (Hx[i] - Hx[i - z_offset]) + Ceyhz[i] * (Hz[i] -
Hz[i - x_offset]);
	}

	for (int i = y_offset; i< n_fields - z_offset; i++)
	{
		Ez[i] = Ceze[i] * Ez[i] + Cezhy[i] * (Hy[i] - Hy[i - x_offset]) + Cezhx[i] *
(Hx[i] - Hx[i - y_offset]);
	}
}

void runTimeMarchingLoopOnCPU()
{
	for (int time_step = 0; time_step < number_of_time_steps; time_step++)
	{
		updateMagneticFields();
		updateElectricFields();
	}
}

int copyArraysToGpuMemory()
{
	int size_int = sizeof(int);
	int size_float = sizeof(float);
	int array_size = (n_fields + maximum_threads_per_block)	* size_float;

	cudaError_t et;
	et = cudaMalloc((void**)&dvEx, array_size);
```

```
	et = cudaMalloc((void**)&dvEy, array_size);
	et = cudaMalloc((void**)&dvEz, array_size);
	et = cudaMalloc((void**)&dvHx, array_size);
	et = cudaMalloc((void**)&dvHy, array_size);
	et = cudaMalloc((void**)&dvHz, array_size);

	et = cudaMalloc((void**)&dvCexhy, array_size);
	et = cudaMalloc((void**)&dvCexhz, array_size);
	et = cudaMalloc((void**)&dvCeyhz, array_size);
	et = cudaMalloc((void**)&dvCeyhx, array_size);
	et = cudaMalloc((void**)&dvCeze, array_size);
	et = cudaMalloc((void**)&dvCezhy, array_size);
	et = cudaMalloc((void**)&dvCezhx, array_size);

	array_size = n_fields * size_float;

	cudaMemcpy(dvEx, Ex, array_size, cudaMemcpyHostToDevice);
	cudaMemcpy(dvEy, Ey, array_size, cudaMemcpyHostToDevice);
	cudaMemcpy(dvEz, Ez, array_size, cudaMemcpyHostToDevice);
	cudaMemcpy(dvHx, Hx, array_size, cudaMemcpyHostToDevice);
	cudaMemcpy(dvHy, Hy, array_size, cudaMemcpyHostToDevice);
	cudaMemcpy(dvHz, Hz, array_size, cudaMemcpyHostToDevice);

	cudaMemcpy(dvCexhy, Cexhy, array_size, cudaMemcpyHostToDevice);
	cudaMemcpy(dvCexhz, Cexhz, array_size, cudaMemcpyHostToDevice);
	cudaMemcpy(dvCeyhz, Ceyhz, array_size, cudaMemcpyHostToDevice);
	cudaMemcpy(dvCeyhx, Ceyhx, array_size, cudaMemcpyHostToDevice);
	cudaMemcpy(dvCeze, Ceze, array_size, cudaMemcpyHostToDevice);
	cudaMemcpy(dvCezhy, Cezhy, array_size, cudaMemcpyHostToDevice);
	cudaMemcpy(dvCezhx, Cezhx, array_size, cudaMemcpyHostToDevice);

	return 0;
}

void setThreadBlocks()
{
	int n_blocks = ((n_fields - z_offset) / maximum_threads_per_block) + ((n_fields -
z_offset) % maximum_threads_per_block == 0 ? 0 : 1);
	block_eh = dim3(maximum_threads_per_block, 1, 1);
	grid_eh = dim3(n_blocks, 1, 1);
	shared_memory_size = 2 * (maximum_threads_per_block + 16) * sizeof(float);
}

void setupGPU()
{
	if (copyArraysToGpuMemory() != 0)
	{
		printf("Ошибка при копировании данных на GPU!\n");
	}
	setThreadBlocks();
}

__global__ void updateMagneticFieldsOnGPU(float Chxey, float Chxez, float Chyez, float
Chyex,
	float Chzex, float Chzey, float* Ex, float* Ey, float* Ez, float* Hx, float* Hy,
	float* Hz, int y_offset, int z_offset, int n_fields)
{
	extern		__shared__ float sE[];
	float *sEy = (float*)sE;
	float *sEz = (float*)&sEy[blockDim.x + 16];
	int ti = threadIdx.x;
	int fi = blockIdx.x * blockDim.x + threadIdx.x;

	sEy[ti] = Ey[fi];
```

```
	sEz[ti] = Ez[fi];
	if (ti < 16)
	{
		sEy[blockDim.x + ti] = Ey[blockDim.x + fi];
		sEz[blockDim.x + ti] = Ez[blockDim.x + fi];
	}
	__syncthreads();
	if (fi >= n_fields - z_offset) return;
	Hx[fi] = Hx[fi] + Chxey*(Ey[fi + z_offset] - sEy[ti]) + Chxez*(Ez[fi + y_offset] -
sEz[ti]);
	Hy[fi] = Hy[fi] + Chyez*(sEz[ti + 1] - sEz[ti]) + Chyex*(Ex[fi + z_offset] - Ex[fi]);
	Hz[fi] = Hz[fi]     + Chzex*(Ex[fi + y_offset] - Ex[fi]) + Chzey*(sEy[ti + 1] -
sEy[ti]);
}

__global__ void updateElectricFieldsOnGPU(float* Cexhy, float* Cexhz, float* Ceyhz, float*
Ceyhx, float* Cezhx,
	float* Cezhy, float* Ceze, float* Ex, float* Ey, float* Ez, float* Hx, float* Hy,
	float* Hz, int y_offset, int z_offset, int n_fields)
{
	extern		__shared__	float sH[];
	float *sHy = (float*)sH;
	float *sHz = (float*)&sHy[blockDim.x + 16]; int ti = threadIdx.x;
	int fi = blockIdx.x * blockDim.x + threadIdx.x;
	sHy[ti + 16] = Hy[fi];
	sHz[ti + 16] = Hz[fi];
	if (ti < 16)
	{
		sHy[ti] = Hy[fi - 16];
		sHz[ti] = Hz[fi - 16];
	}
	__syncthreads();
	if (fi >= n_fields - z_offset) return;
	if (fi < y_offset) return;
	Ez[fi] = Ceze[fi] * Ez[fi] + Cezhy[fi] * (sHy[ti + 16] - sHy[ti + 15]) + Cezhx[fi] *
(Hx[fi] - Hx[fi - y_offset]);
	if (fi < z_offset) return;
	Ex[fi] = Ex[fi] + Cexhz[fi] * (sHz[ti + 16] - Hz[fi - y_offset]) + Cexhy[fi] *
(sHy[ti + 16] - Hy[fi - z_offset]);
	Ey[fi] = Ey[fi] + Ceyhx[fi] * (Hx[fi] - Hx[fi - z_offset]) + Ceyhz[fi] * (sHz[ti +
16] - sHz[ti + 15]);
}

void copyDataBackAndClearGPU()
{
	int size_int = sizeof(int);
	int size_float = sizeof(float);
	int array_size = (n_fields + maximum_threads_per_block)	* size_float;

	cudaMemcpy(Ex, dvEx, array_size, cudaMemcpyDeviceToHost);
	cudaMemcpy(Ey, dvEy, array_size, cudaMemcpyDeviceToHost);
	cudaMemcpy(Ez, dvEz, array_size, cudaMemcpyDeviceToHost);
	cudaMemcpy(Hx, dvHx, array_size, cudaMemcpyDeviceToHost);
	cudaMemcpy(Hy, dvHy, array_size, cudaMemcpyDeviceToHost);
	cudaMemcpy(Hz, dvHz, array_size, cudaMemcpyDeviceToHost);

	cudaFree(dvEx);
	cudaFree(dvEy);
	cudaFree(dvEz);
	cudaFree(dvHx);
	cudaFree(dvHy);
	cudaFree(dvHz);

}
```

```
void startTimer(){
        search_time = clock();
}

void stopTimer(){
       search_time = clock() - search_time;
}

void runTimeMarchingLoopOnGPU()
{
       for (int time_step = 0; time_step < number_of_time_steps; time_step++)
       {
              updateMagneticFieldsOnGPU <<< grid_eh, block_eh, shared_memory_size >>>
                     (Chxey, Chxez, Chyez, Chyex, Chzex, Chzey, dvEx, dvEy,
                     dvEz, dvHx, dvHy, dvHz, y_offset, z_offset, n_fields);
              updateElectricFieldsOnGPU <<< grid_eh, block_eh, shared_memory_size >>>
                     (dvCexhy, dvCexhz, dvCeyhz, dvCeyhx, dvCezhx, dvCezhy, dvCeze,
                     dvEx, dvEy, dvEz, dvHx, dvHy, dvHz, y_offset, z_offset, n_fields);
       }
}


int main()
{
       initializeStartParameters();
       setupProblemSpace();
       initializeVariables();
       startTimer();

       if(run_on_gpu)
       {
              setupGPU();
              runTimeMarchingLoopOnGPU();
              copyDataBackAndClearGPU();
       }
       else
       {
              runTimeMarchingLoopOnCPU();
       }
       stopTimer();
}
```