Санкт-Петербургский государственный университет
Кафедра компьютерного моделирования и многопроцессорных систем

Выпускная квалификационная работа

Моделирование качки судна в реальном времени с использованием вычислений общего назначения на графическом процессоре

Петряков Иван Владимирович

Направление 01.03.02
«Прикладная математика, фундаментальная информатика и программирование»
Бакалаврская программа «Прикладная математика, фундаментальная информатика и программирование»

Научный руководитель
Дегтярев А.Б.
Ганкевич И.Г.

Санкт-Петербург
2019

Содержание

Введение

До недавнего времени центральный процессор являлся единственным вычислительным элементом компьютера. Растущие потребности научного сообщества и индустрии вынудили производителей видеокарт предоставить сторонним разработчикам ПО возможность программировать шейдерные блоки под свои нужды. Эта возможность позволила увеличить производительность ПК, подключив к расчетам графическое устройство.

Эта работа проводилась в рамках проекта «Виртуальный полигон», целью которого является создание системы поддержки принятия решений для моделирования, прогнозирования и предотвращения опасных ситуаций, вызванных различными физическими явлениями. Ситуации могут включать в себя: затопление отсека, пожар в отсеке, потеря остойчивости корабля, большие морские волны и многие другие явления. Данная работа выделяется на фоне предшественников возможностью моделирования явлений и морских объектов в режиме реального времени, и в то же время обеспечивая высокую точность моделирования.

Система принятия решений получает данные в режиме реального времени с морских судов и моделирует их движение. Технология, которая обеспечивает производительность в режиме реального времени, - это GPGPU. Большинство вычислений в проекте включают в себя большое количество трансцендентных математических функций, линейный доступ к массивам и матрицам и отсутствие сложных зависимостей между этапами вычислений. Это позволяет эффективно использовать фреймворк OpenCL для действительно быстрых вычислений.

В нашем случае недостаточно просто переписать исходный код проекта для OpenCL. Чтобы повысить эффективность вычислений на GPU, нам по-

требовалось адаптировать алгоритмы и математические формулы к виду, оптимальному для графического процессора. Прежде всего был применен новый метод вычисления потенциала скорости, численно эквивалентный известным формулам из теории линейных волн, и использующий быстрое преобразование Фурье, для которого существует множество высокопроизводительных программных реализаций на GPU. Следующим этапом стало вычисление частной производной функции в многомерном пространстве, что позволило избавиться от многочисленных переносов данных из памяти процессора в память GPU. И наконец, последним шагом был реализован расчет основных математических функций на GPU.

Моделирование движения судна в океанских волнах выполняется различными компьютерными программами, которые отличаются тем, какие физические явления они моделируют (маневрирование в волнах, затопление отсека, регулярные и нерегулярные волны, ветер, моделирование в реальном времени и т. д.) и области применения (научные исследования, образование или развлечения). Эти программы являются виртуальными аналогами опытовых бассейнов, которые используются для моделирования характеристик и поведения корабля в определенных морских условиях. Преимущество использования виртуального опытового бассейна перед физическим состоит в том, что эксперименты проводятся в реальном пространственном и временном масштабе (с кораблями и океанскими волнами реальных размеров) и на персональном компьютере без необходимости доступа к высокотехнологичному оборудованию.

Хотя все численные эксперименты проводятся на ПК, один компьютер недостаточно силен для их быстрого выполнения. Часто эта проблема решается с помощью кластера компьютерных узлов или суперкомпьютера; однако суперкомпьютер или кластер — это еще одно высокотехнологичное средство, к которому должен получить доступ исследователь. В этом случае виртуальный опытовый бассейн имеет мало преимуществ перед физическим: исследования замедляются из-за бюрократической волокиты и вынужденного разделения

вычислительных ресурсов между остальными участниками сообщества.

Одним из способов устранения этого барьера является использование графического ускорителя для увеличения производительности. В этом случае моделирование может быть выполнено на обычной рабочей станции, которая имеет дискретную видеокарту. Большинство исследователей используют графический процессор для визуализации в реальном времени, но он редко используется для ускорения частей моделирования, не говоря уже о всей программе. В [1] авторы используют графический процессор для ускорения расчета движения свободной поверхности внутри резервуара. В [2] авторы переписывают свой код с использованием быстрых преобразований Фурье и предлагают использовать GPU для увеличения производительности. В [3] авторы используют GPU для моделирования океанских волн. Тем не менее, наиболее эффективный способ использования графического процессора — использовать его как для вычислений, так и для визуализации: он позволяет минимизировать копирование данных между памятью процессора и графического процессора и использовать математические модели, структуры данных и численные методы, адаптированные для графических ускорителей.

Как уже было отмечено выше, Виртуальный полигон — один из первых проектов, в котором делается упор как на точность вычислений, не пренебрегая известными физическими законами, так и на высокую производительность.

Теоретическая и практическая значимость работы

Программные пакеты для моделирования судов обычно используются для статистических исследований динамики судов, а также в качестве тренажера

для подготовки экипажа судна в опасных ситуациях. Одной из проблем, возникающих во время тренировки, является ускорение этапа тренировки, который не связан с действиями экипажа. Чтобы обеспечить такую производительность, необходимо научиться эффективно решать системы уравнений, описывающих движение корабля. Эти уравнения описывают динамику маневрирования судов на волнистой поверхности моря и являются центральными для симулятора. Уравнения решаются численно методом Рунге—Кутты—Фельберга [4] [5]. Из-за большого числа операций с плавающей точкой, вычисления на GPU позволяют достичь значительного ускорения по сравнению с CPU вычислениями. Производительное решение не только позволяет сделать тренировки экипажа более эффективными, но также будет полезно для статистических исследований, поскольку сокращает время моделирования.

Особенность, отличающая виртуальный полигон от аналогичных проектов, заключается в том, что это не только симулятор движения корабля, но и система поддержки принятия решений. Это приводит к жестким требованиям к производительности, которых нет в других симуляторах. Часто они являются либо инструментами исследования без каких-либо требований к производительности, либо симуляторами для компьютерных игр или фильмов.Виртуальный полигон нуждается как в точности, так и производительности, и это его главное отличие от аналогичных программ.

Методология и методы исследования

На ранних этапах разработки Виртуальный полигон не имел реализации, ускоренной на видеокарте, а все вычисления проводились на центральном процессоре. Анализ исходного кода программы помог выявить возможность приме-

нения алгоритмов параллельного вычисления с помощью технологий GPGPU. Разработка такой редакции программы производится одновременно с процессорной. Сравнение результатов обоих версий приложения позволило найти и исправить ряд ошибок как в первоначальном варианте, так и в ускоренном.

Для эффективного выполнения поставленной задачи, исследование проводилось в несколько этапов:

- профилирование участков кода, чтобы выявить наиболее продолжительные промежутки времени выполнения программы;
- анализ возможности применения алгоритмов параллельного вычисления, характерных архитектуре графического устройства;
- реализация намеченных изменений;
- дополнительное профилирование примененных алгоритмов;
- оценка проведенной работы.

На последнем этапе определяется насколько успешно выполнен алгоритм. Если прирост производительности не наблюдается, то проводится дополнительное исследование на предмет оптимизаций исходного кода функции.

Постановка задачи

Основной задачей этой работы было исследование возможности увеличения производительности за счет подключения к вычислениям графических устройств.

Для увеличения быстродействия проекта требуется перенести основные алгоритмы, занимающие большую часть процессорного времени, на многоядерную архитектуру GPU. Таким образом, задействуется больше ресурсов компьютера и разгрузится центральный процессор. Формат входных данных зачастую представляет собой трехмерный или четырехмерный массив значений какой-либо функции. На выход программа ожидает преобразованные (в зависимости от ситуации) данные, которые участвуют в дальнейших вычислениях. Добиться максимального быстродействия поможет только умелое управление памятью, кэшем и синхронизацией потоков.

Обзор литературы

Математическое описание поставленной задачи приведено в [6]. В ней описана динамика движения корабля, рассматриваемого как твердое тело в общем движении с шестью степенями свободы. Описывается связь нелинейностей применяемых подмоделей с физическими явлениями, связанными с поведением корабля, а так же представлена теория поверхностных волн. Кратко представлена математическая модель для оценки нагрузок на внутренние балки корпуса

В [7] авторы разрабатывают метод для моделирования движения корабля, основанный на физических теориях с упрощениями, которые необходимы для работы в реальном времени. Под упрощением понимается разложение волн на продольные и поперечные для ускорения расчета возбуждающих сил. Авторы отмечают, что использование теории потенциала скорости для вычисления этих сил нецелесообразно для реализации в приложениях реального времени. Вопреки этому утверждению мы вычисляем силы волнового возбуждения с помощью потенциала скорости. Причина, по которой этот подход работает в режиме реального времени, двояка. Во-первых, мы используем явную формулу, основанную на преобразованиях Фурье, которая вычисляется быстрее в сравнении с неявными формулами. Во-вторых, мы используем графический процессор для ускорения вычислений преобразований Фурье.

В [2] авторы используют передаточные функции для моделирования движения судна в режиме реального времени. Для повышения производительности они также используют быстрые преобразования Фурье, но производят вычисления только на процессоре. Наконец, в [8] авторы изучают, как использовать GPU для моделирования поверхности океана.

1. Используемое программное обеспечение

1.1. OpenCL

В качестве основной технологии для вычисления на графическом ускорителе был выбран фреймворк OpenCL. Это открытый стандарт, позволяющий выполнять вычисления общего назначения не только на видеокарте, но и на центральном процессоре, что упрощает разработку кода и поиск ошибок. Он не требует определенных устройств (как Nvidia CUDA) и позволяет запускать разработанные программы практически на любом устройстве.

Платформа OpenCL состоит из центрального устройства (например, процессор), соединяемый с устройствами, поддерживающие OpenCL. OpenCL устройство состоит из вычислительных блоков, которые, в свою очередь, разделены на элементы-обработчики. Выполнение программы на OpenCL состоит из набора команд, отправляемых процессором на устройство для дальнейшего вычисления на элементах-обработчиках. Такая организация процесса вычислений способствует кроссплатформенности технологии, так как для каждого устройства можно реализовать собственный интерпретатор потока команд.

OpenCL приложение можно представить в виде двух частей: исполняющаяся на главном устройстве, и исполняющаяся на устройстве. Процессорная часть программы определяет контекст, в котором выполняются подпрограммы (ядра). Ядро попадает в очередь исполнения, в котором для него задаются аргументы и пространство индексов. Для каждого индекса этого пространства выполняется копия подпрограммы, называемая рабочей единицей. Индекс, соот-

ветствующий этой копии, называется глобальным индексом. Рабочие единицы могут быть объединены в рабочие группы. Они имеют свой индекс с размерностью вышеупомянутого пространства индексов. Внутри группы, рабочая единица имеет локальный индекс с той же размерностью, а значит каждая копия ядра может иметь как глобальный адрес, так и адрес, являющийся комбинацией локального индекса и индекса группы, к которой принадлежит этот элемент.

Каждая рабочая единица может работать с различными видами памяти:

- Глобальная память. Доступна всем рабочим единицам устройства для чтения и записи.
- Локальная память. Доступна всем рабочим единицам устройства в пределах одной рабочей группы.
- Приватная память. Этот тип памяти имеет каждая рабочая единица. Одна рабочая единица не имеет доступа к приватной памяти другой.

В работе была применена модель параллелизма SIMD (single instruction, multiple data — одна инструкция, множество данных), при котором каждому объекту памяти соответствует одна рабочая единица устройства.

Дополнительно тестирование проводилось с помощью программы Oclgrind [9]. Oclgrind — это платформа для анализа и отладки приложений OpenCL. Это приложение моделирует окружение OpenCL в соответствии со стандартом и соблюдает описанную выше модель управления памятью и индексации вычислительных единиц, в результате чего программы, написанные для выполнения на видеокарте, могут быть запущены на центральном процессоре. Большое количество инструментов позволяет отслеживать появление ошибок, которые трудно было бы обнаружить на реальном устройстве.

1.2. OpenMP

Современные многоядерные CPU достаточно хорошо справляются с параллелизмом инструкций. Для параллельного программирования на центральном процессоре был использован программный интерфейс OpenMP — открытый стандарт разработки, API (application programming interface — программный интерфейс приложения), поддерживающий мультиплатформенное многопоточное программирование на языках C, C++ и Fortran для многопроцессорных систем с общей памятью. В стандарте описываются различные переменные среды выполнения программы, функции и директивы компилятора, которые используются для наиболее подробного описания параллелизма. Этот стандарт поддерживается всеми наиболее известными компиляторами (например Visual C++, GCC, Clang),

Программа, написанная с использованием OpenMP, выполняется последовательно, пока не встретится участок, который необходимо выполнить параллельно. В этот момент программа создает нужное количество потоков (заданное программистом или используемое по умолчанию), которые являются копиями родительского процесса. После выполнения этого блока кода, дочерние потоки ожидают синхронизации, а затем прекращают свою работу.

2. Виртуальный полигон

Виртуальный полигон — это приложение, которое моделирует морские волны, движение судов и затопление отсеков. Одной из особенностей, которая отличает его от существующих предложений, является использование графических ускорителей для увеличения производительности, благодаря чему появилась возможность визуализации моделирования в реальном времени.

Программа состоит из следующих модулей: считыватель модели корпуса корабля из входного файла определенного формата, графический интерфейс, который отображает текущее состояние виртуального мира, и ядро, определяющее следующее положение корабля в пространстве, вычисляя каждый этап моделирования. Основной модуль состоит из компонентов, которые связаны друг с другом в конвейере, при котором выходные данные одного компонента являются входными данными другого. Вычисление выполняется параллельно с визуализацией, и синхронизация происходит после каждого шага моделирования. Это делает графический интерфейс пользователя отзывчивым, даже когда рабочая станция недостаточно производительна для вычислений в режиме реального времени.

Внутри ядра присутствуют следующие компоненты: генератор волнистой поверхности, вычислитель потенциала скорости, и вычислитель силы давления. Каждый компонент в модуле ядра является взаимозаменяемым, что означает, например, что разные генераторы волнистой поверхности могут использоваться с одним и тем же механизмом определения потенциала скорости. После инициализации эти компоненты выполняются в цикле, в котором каждая итерация вычисляет следующий временной шаг моделирования. Хотя итерации цикла являются последовательными, каждый компонент внутри ядра выполняется параллельно, то есть каждый компонент использует OpenMP или OpenCL для вы-

числений на каждом процессоре или графическом ядре. Другими словами, виртуальный испытательный полигон следует модели BSP [10] для организации параллельных вычислений, в которых программа состоит из последовательных шагов, каждый из которых является внутренне параллельным.

2.1. Генератор волнистой поверхности

В виртуальном полигоне используются три модели для создания волнистой поверхности: модель авторегрессионного скользящего среднего (АРСС), волна Стокса и плоская синусоида/косинусоида. С точки зрения производительности невыгодно вычислять модель АРСС на графическом ускорителе: ее алгоритм не использует трансцендентные математические функции, имеет нелинейный шаблон доступа к памяти и сложные информационные зависимости. Гораздо эффективнее (даже без серьезных оптимизаций) выполнить его на процессоре. Напротив, две другие волновые модели волн не имеют препятствий к распараллеливанию, и их легко переписать под архитектуру OpenCL.

Каждая волновая модель выводит трехмерное (одно временное и два пространственных измерения) поле высот волнистой поверхности. Полученное поле является стохастическим, но имеет те же интегральные характеристики, что и исходное. В частности, сохраняется функция распределения вероятности возвышения волнистой поверхности, высоты волны, длины и периода.

Подводя итог, можно сказать, что генератор волнистой поверхности создает поле возвышения волнистой поверхности, используя одну из моделей, описанных выше. Модель АРСС нецелесообразно генерировать с помощью графического ускорителя, а для других моделей эта задача решается тривиально. Это поле высот является входом для модуля вычисления потенциала скорости.

2.2. Потенциал скоростей

Давление воды выводится из потенциала скоростей, с допущением, что морская вода является несжимаемой невязкой жидкостью. Потенциал скоростей определяется следующей системой уравнений:

$$\begin{cases} \nabla^2\phi = 0, \\ \phi_t + \frac{1}{2}|\vec{v}|^2 + g\zeta = -\frac{p}{\rho}, & \text{at } z = \zeta(x,y,t), \\ D\zeta = \nabla\phi \cdot \vec{n}, & \text{at } z = \zeta(x,y,t), \end{cases} \tag{1}$$

где первое уравнение — уравнение непрерывности, второе — уравнение движения, а третье — кинематическое граничное условие на свободной поверхности. Поскольку свободная поверхность известна, второе уравнение становится явной формулой для волнового давления, а задача сводится к нахождению потенциала скорости ϕ:

$$\begin{cases} \phi_{xx} + \phi_{yy} + \phi_{zz} = 0, \\ \zeta_t = \underbrace{\left(\frac{\zeta_x}{\sqrt{1+\zeta_x^2+\zeta_y^2}} - \zeta_x \right)}_{f_x} \phi_x + \underbrace{\left(\frac{\zeta_y}{\sqrt{1+\zeta_x^2+\zeta_y^2}} - \zeta_y \right)}_{f_y} \phi_y - \underbrace{\frac{1}{\sqrt{1+\zeta_x^2+\zeta_y^2}}}_{f_z} \phi_z \end{cases} \tag{2}$$

Система решается методом Фурье с некоторыми физическими и математическими допущениями. Полное решение представлено в виде свертки некоторой оконной функции с суперпозицией производных волнистой поверхности. Если используются волны малой амплитуды, решение сводится к решению из

теории линейных волн.

$$\phi(x, y, z, t) = W_2(x, y, z) \cdot \frac{\zeta_t(x, y, t)}{F(f_1, f_2, f_3)} \tag{3}$$

Свертка реализована с помощью трех преобразований Фурье, которые очень эффективно вычисляются на графических процессорах. Формула подходит для любой волнистой поверхности, независимо от математической модели, используемой для ее создания, если она физически выполнима: одну и ту же формулу можно использовать как для плоских, так и для нерегулярных волн произвольной амплитуды. Полный вывод формулы приведен в [11].

Решение системы уравнений сходится, если интегрирование происходит в диапазоне значений волновых чисел. Этот диапазон определяется с помощью численных методов среди всех значений на водной поверхности для каждого измерения. Такой подход предлагает те же значения потенциала скоростей, что и формулы из линейной теории волн для волн малой амплитуды.

2.3. Сила давления воды

В проекте задействованы три этапа для расчета силы давления: определение смоченной поверхности корпуса судна, вычисление поля давлений под волнистой поверхностью и вычисление силы давления, действующей на корпус судна.

Выталкивающая сила пропорциональна массе воды, замещаемой судном. Для ее расчета корпус корабля разбивается на треугольные панели и определяется, какие из них находятся под водой. Для частично погруженных панелей вычисляется их пересечение с уровнем воды (предполагая, что это прямая ли-

ния) и учитывается только погруженная часть в последующих расчетах. Для каждой панели рассчитывается давление, полученное из-за смещения воды и давление вызванное движением морской волны.

$$\rho(x, y, z, t) = -\rho\phi_t - \rho\frac{1}{2}(\phi_x^2 + \phi_y^2 + \phi_z^2) - \rho g z \tag{4}$$

В таком случае, сила — это вектор в направлении, противоположном поверхности нормали к панели, длина которого пропорциональна давлению и площади панели:

$$F(x, y, z, t) = -\rho S\vec{n} \tag{5}$$

Волновое давление в точке под волнистой поверхностью вычисляется с использованием граничного условия из (1). Затем давление интерполируется в центре каждой панели для вычисления силы давления, действующей на корпус судна. Общая выталкивающая сила — это сумма всех отдельных сил, действующих на каждую погруженную панель. Если опустить потенциал скорости, формула становится законом Архимеда. Этот алгоритм имеет линейный шаблон доступа к памяти и включает в себя множество операций с плавающей точкой, а также векторную арифметику, что позволяет эффективно использовать возможности GPU.

2.4. Вычисление производной функции

Вычисление силы давления довольно просто реализуется на графическом ускорителе. Профилирование приложения показало, что большую часть времени занимает копирование массивов данных между видеопамятью графического

устройства и оперативной памятью процессора при вычислении производной. Нахождение производной функции не требует больших вычислительных мощностей, но чтобы избавиться от лишнего копирования было решено использовать для этой цели видеокарту. Для этого был применен алгоритм, который описан в статье [12].

Производная функции — предел отношения приращения функции к приращению ее аргумента. Если функция представлена некоторым фиксированным набором значений, в дело вступает численное дифференцирование. В основе этого метода лежит интерполирование функции некоторым многочленом. Чем выше порядок многочлена — тем выше точность вычислений. В общем случае формулу численного дифференцирования можно представить в виде (6), где f_i‘ и f_i значения производной функции и самой функции соответственно, а c_j — некоторый коэффициент.

$$f'_i = \sum_j c_j f_{i+j} \tag{6}$$

Алгоритм принимает на вход набор значений некоторой функции, определенной на трехмерном или четырехмерном пространстве, заданный в виде многомерной таблицы значений. Результатом работы алгоритма ожидается таблица той же размерности, в каждой ячейке которой находится значение производной функции в соответствующем порядке. Реализация этого алгоритма позволила отказаться от большого количества копирования данных между оперативной памятью компьютера и внутренней памятью графического устройства.

Для Виртуального полигона в OpenMP версии для вычисления производной использовалась формула с конечными разностями первого порядка (3, 4, 5), что обеспечивало необходимую точность в вычислениях. При проектировании этой функции на графическом ускорителе было решено не отклоняться от

этой формулы, чтобы обе версии программы выдавали одинаковый результат.

$$\begin{cases} f_i' = \frac{f_{i+1}+f_{i-1}}{2} \\ f_i' = \frac{-3f_i+2f_{i+1}-f_{i+2}}{2} \\ f_i' = \frac{3f_i-2f_{i-1}+f_{i-2}}{2} \end{cases} \tag{7}$$

Реализация производной на видеокарте затрагивает различные аспекты оптимизаций, характерные для графического процессора. Основной сложностью такого метода для графического процессора можно отметить множественное количество обращений к памяти с высокими задержками доступа, а именно три раза, если точка находится на границе сетки значений функции (последние две формулы в 7), и два раза если точка не является крайней в этой таблице (первая формула в 7). Для этого, части массива копировались из глобальной памяти устройства в локальную память рабочей группы, что позволило существенно сократить количество обращений к «медленной» памяти.

Такой подход повел за собой следующую проблему, а именно недостаточный объем локальной памяти одной рабочей группы. Очевидный выход из этой ситуации — разбить таблицу значений на такие части, которые с легкостью бы поместились в локальную память. Но, так как для вычисления производной требуется использовать соседние значения функции, то появились бы такие участки памяти, которые подгружались бы несколько раз. Поэтому необходимо найти значение производной в одном вычислительном блоке «от края до края» сетки. Важное свойство, о котором необходимо помнить — архитектура GPU использует такой подход обращения к памяти, при котором в кэш вычислительного блока ускорителя подгружается сразу несколько последовательных ячеек данных, и количество этих ячеек совпадает с количеством вычислительных единиц в блоке. Если определить локальную группу из большего числа элементов, чем находится в одном вычислительном блоке, то такое разделение будет не оптимально, так как придется синхронизировать данные из разных

вычислительных блоков. Если определить меньше — такое разбиение не задействует вычислительные мощности в полной мере. Таким образом алгоритм выстраивается следующим образом:

1. Скопировать начальный объём данных из глобальной памяти в локальную.

2. Выполнить необходимые вычисления.

3. Скопировать следующий набор значений и перейти к пункту 2, пока не вычисления не достигнут границы сетки.

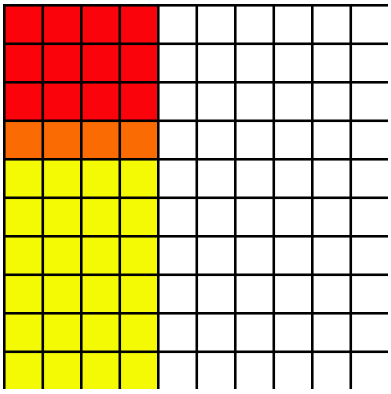

Рис. 1: Схема расположения ячеек памяти. Красным цветом выделены ячейки, которые уже находятся в локальной памяти вычислительного блока. Оранжевым отмечены ячейки, которые будут вскоре загружены в локальную память устройства, желтым отмечены ячейки, которые будут обработаны одной рабочей группой. Подразумевается, что последовательные ячейки памяти расположены слева направо, а вычислительный блок состоит из четырех вычислительных единиц.

При таком подходе (Рис. 1) на каждой итерации вычисления ячейки локальной памяти, в которых хранилось значение f_{i-1} освобождаются, так как значения, хранящиеся в них больше не нужны. На их место загружаются значения f_{i+2}. Следует отметить, что данный алгоритм применим только к тем измерениям многомерной функции, последовательные значения которой вдоль этого измерения не лежат последовательно в памяти устройства.

Для оставшегося измерения (оно, обычно, первое) алгоритм остается, в некотором смысле, наивный. В таком случае, при вычислении производной, все необходимые данные для нахождения значения автоматически будут загружаться в кэш вычислительного блока. Таким образом, Виртуальный полигон

имеет две реализации производной: для случая, когда данные выстроены вдоль измерения, по которому берется производная, и в противоположном случае.

3. Итог проведенной работы

3.1. Рабочее окружение

Виртуальный полигон позиционируется как решение для настольных компьютеров. Был проведен ряд испытаний, при которых измерялось время выполнения описанных выше функций. Это позволило оценить уровень производительности приложения в процессорной версии программы и ускоренной с помощью GPU, а так же сравнить полученный прирост производительности. В обоих случаях операции с плавающей точкой выполнялись с одинарной точностью, так как графические ускорители более оптимизированы для работы с таким типом вычислений.

Испытание представляло собой стандартное выполнение программы, которое длилось в течение минуты. Для определения полученной производительности измерялось время выполнения каждой итерации глобального цикла Виртуального полигона и время выполнения каждого шага в цикле. Каждая версия программы запускалась на одном и том же устройстве с неизменяемой конфигурацией оборудования.

Для тестирования была выбрана модель шара с 5120 полигонами. Испытание проводилось на одном компьютере со следующими аппаратными и программными характеристиками:

- центральный процессор AMD Ryzen 2700x, 8 ядер/16 потоков, работающий на частотах 3.7 – 4.3 ГГц,
- графический ускоритель Nvidia GTX 1060 6gb, версия драйвера 418.74, версия CUDA 10.1,

- компилятор GCC, версии 9.1.1, с дополнительными параметрами компилятора «-O3 -march=native» под управлением ОС Fedora 30,
- для графического процессора использовалась версия OpenCL 1.2 CUDA.

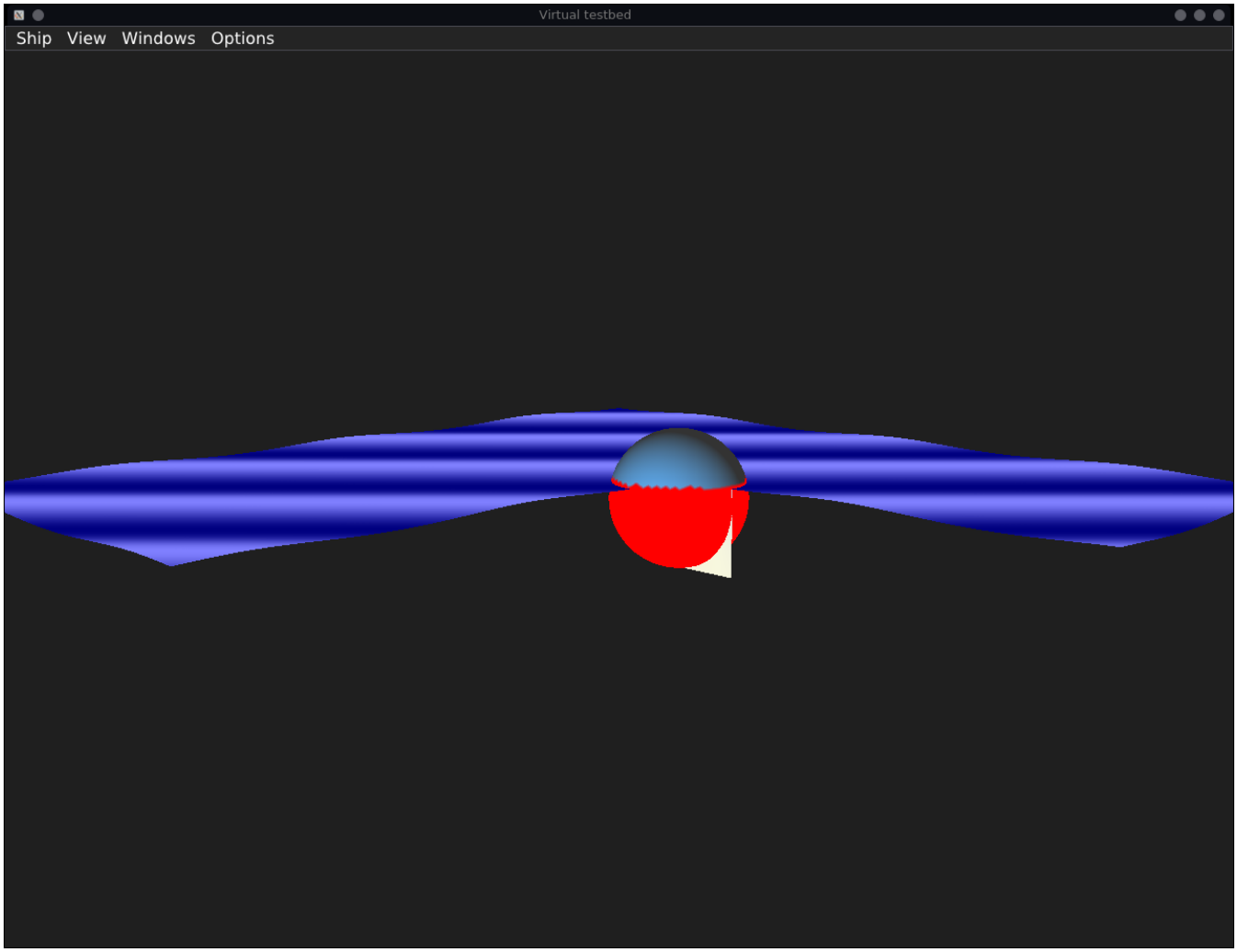

Рис. 2: Модель шара, на котором производилось испытание.

3.2. Сравнение полученных результатов

По результатам проведенных испытаний можно отметить, что Виртуальный полигон показывает более высокую производительность в работе с гра-

фическим ускорителем. На описанном выше компьютере реализация с OpenCL достигла пятидесяти итераций глобального цикла в секунду, в то время как OpenMP версия приложения позволила достигнуть лишь сорока итераций глобального цикла в секунду.

Рассмотрим поближе функции вычисления давления и потенциала скоростей. На графике 3 представлена зависимость времени выполнения функции вычисления силы давления от количества погруженных под воду панелей.

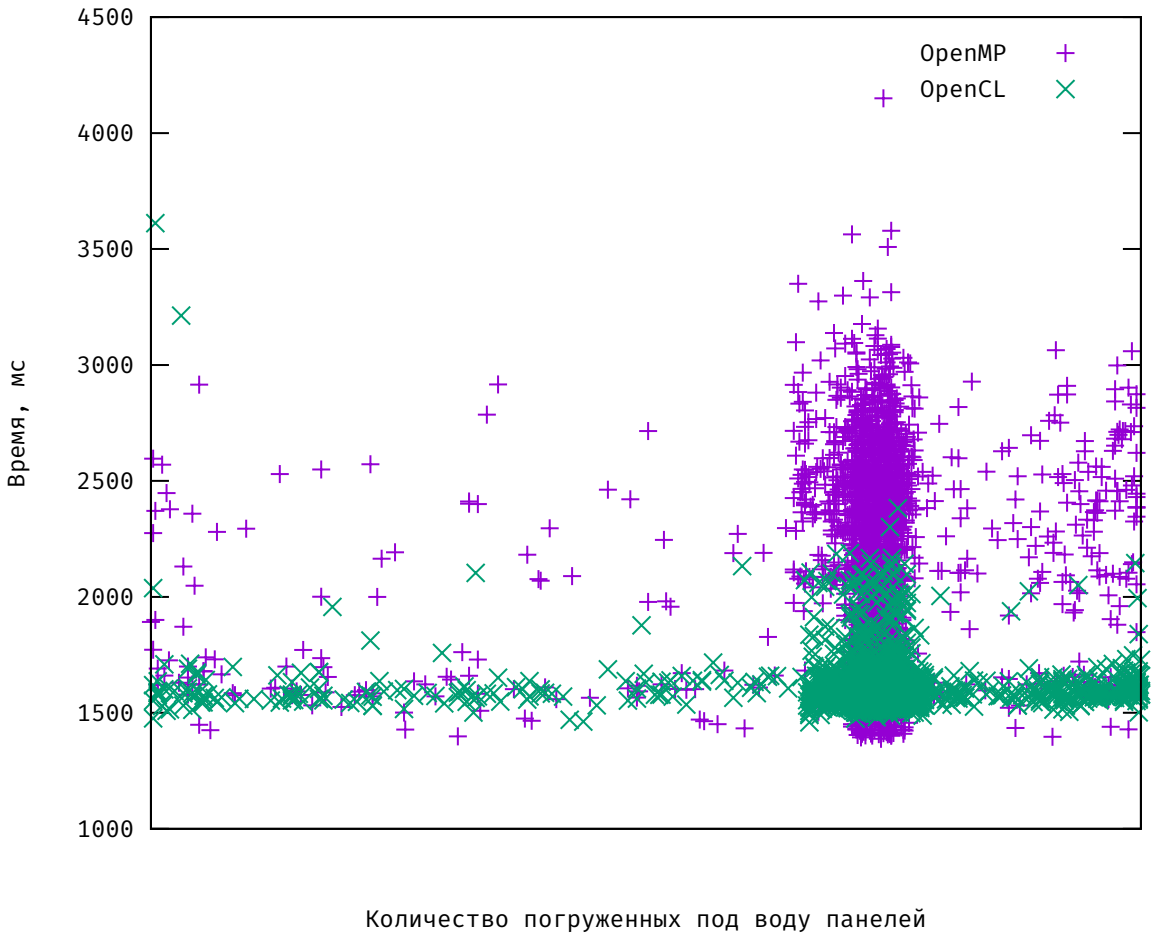

Рис. 3: Зависимость времени вычисления силы давления, действующих на судно, от количества погруженных панелей.

На графике можно заметить значения времени выполнения функции в реализации OpenMP, которые получились меньше соответствующих значений в версии с OpenCL, но в целом ускоренная на видеокарте версия программы выглядит более стабильной при увеличении количества панелей. Об этом свойстве может говорить тот факт, что среднее время выполнения этой функции в

процессорной версии проекта составляет примерно 2,086 мс, при среднем отклонении в 0,453 мс, в то время как использование GPGPU предлагает 1,613 мс в среднем на вычисление силы давления, при среднем отклонении величиной 0,105 мс.

На следующем графике изображена зависимость между временем вычисления потенциала скоростей от количества панелей под водой. На данном примере легко заметить, что использование OpenCL в проекте заметно сократило время выполнения расчета потенциала скоростей.

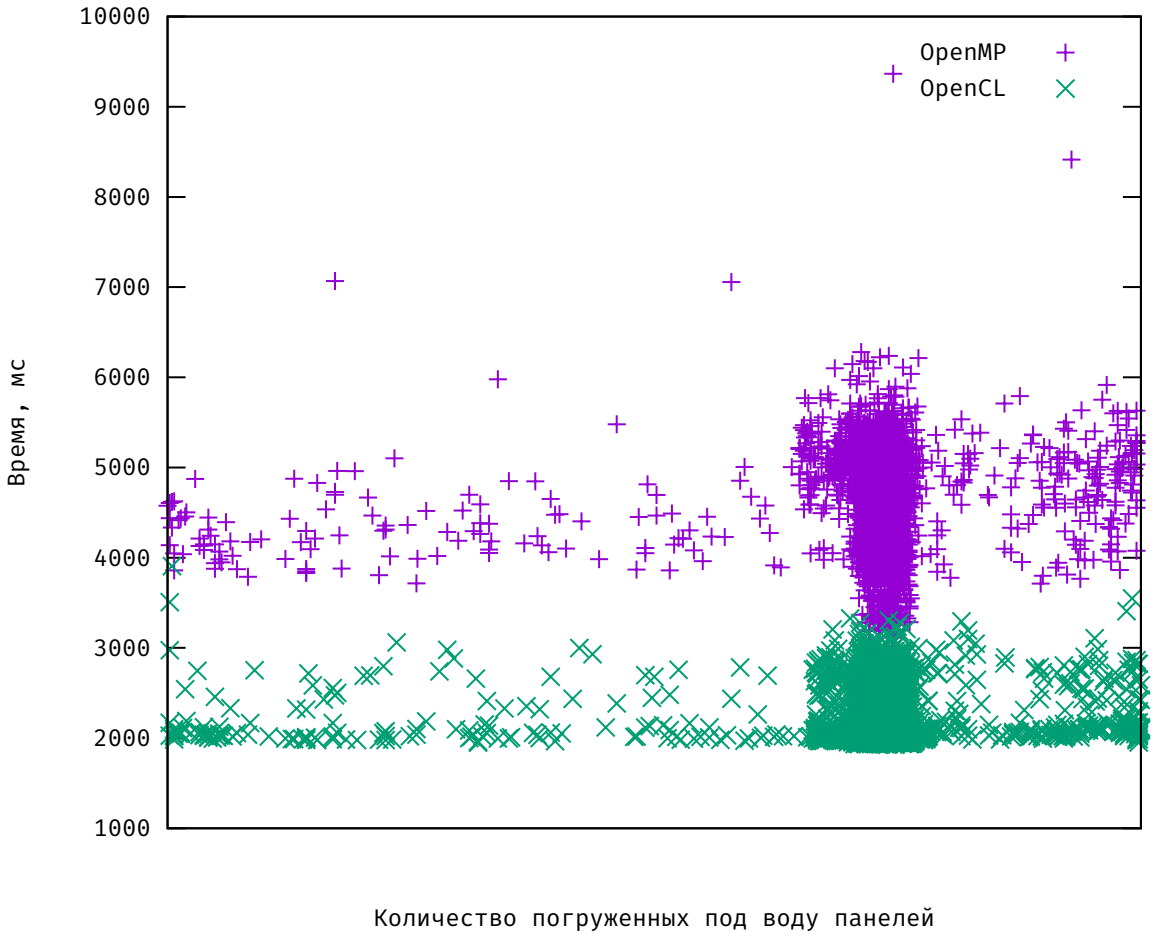

Рис. 4: Зависимость времени вычисления потенциала скоростей от количества погруженных под воду панелей.

При вычислении на процессоре среднее время выполнения этой функции 4,579мс при среднем отклонении 0,596мс, OpenCL версия требует на это в среднем 2,191мс при среднем отклонении 0,294мс.

Такие статистические параметры говорят не только о повышении про-

изводительности проекта, но и о повышенной точности вычислений (увеличенном числе этапов моделирования в секунде), что повышает плавность перехода при моделировании новых шагов. Вычисление производной происходит каж-

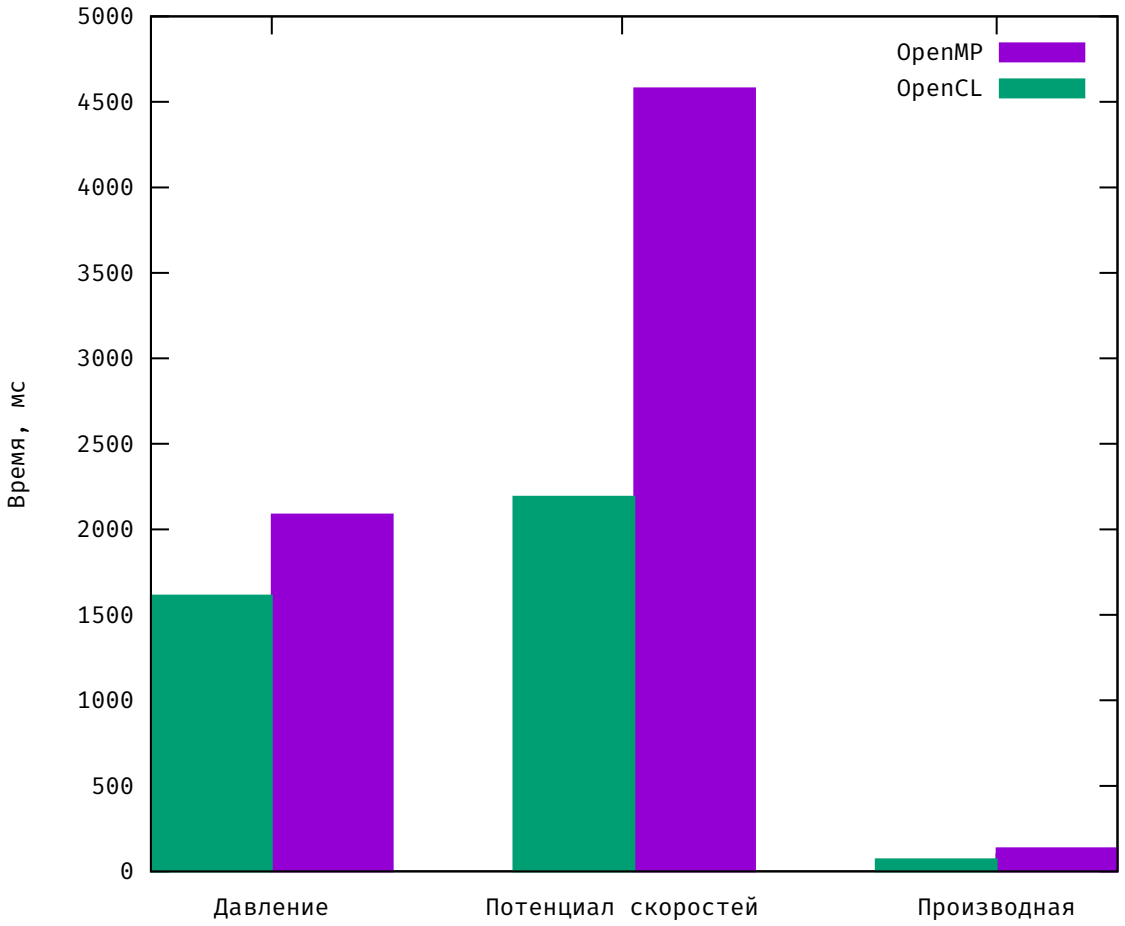

Рис. 5: Сравнение среднего времени выполнения обозреваемых функций.

дый раз на различных размерах четырехмерной сетки, так как для увеличения быстродействия приложения пространство «обрезается» до размеров корабля и не зависит от количества погруженных под воду панелей. Чтобы проиллюстрировать полученные изменения в качестве метрики было выбрано только среднее значение и среднеквадратичное отклонение времени выполнения. Для версии OpenCL эти показатели составляют 0.07мс и 0.05мс соответственно, а для OpenMP 0.134мс и 0.037мс. Сложно сказать, насколько существенный прирост производительности был получен в сравнении с процессорной версией приложения. Самое важное достижение этого этапа разработки — избавиться от лишнего копирования.

4. Вывод

Использование графического ускорителя позволило увеличить производительность компьютера в Виртуальном полигоне по сравнению с версией этого приложения, выполненной с использованием технологии OpenMP. Это вызвано большим количеством вычислительных блоков на видеокарте, при которых массивные объемы данных обрабатываются быстрее, чем на процессоре. В процессе исследования было показано, что применение технологии GPGPU целесообразно и дальнейшее развитие проекта будет сопровождаться использованием OpenCL. Процессорная версия приложения будет использоваться в качестве эталона программы, результаты ускоренной на видеокарте части программы будут сверяться с процессорной версией.

5. Заключение

В работе был рассмотрен Виртуальный полигон, целью которого является создание системы поддержки принятия решений для моделирования, прогнозирования и предотвращения опасных ситуаций на судне в море. Его основной особенностью является возможность моделирования положения корабля в режиме реального времени. Это достигается за счет подключения к вычислениям видеокарты и использования технологии GPGPU. Описана структура этого приложения, его отличительные черты и различные модули. В ходе работы подверглись анализу различные модули программы на предмет возможности применений технологий GPGPU для увеличения производительности. После этого, было произведено сравнение двух версий этого приложения: выполняемой на центральном процессоре, и на графическом ускорителе.

Список литературы

[1] Coupled simulation of nonlinear ship motions and a free surface tank / Jose Luis Cercos-Pita, Gabriele Bulian, Luis Pérez-Rojas, Alberto Francescutto // Ocean Engineering. — 2016. — Vol. 120. — P. 281–288.

[2] Varela José Miguel, Soares Carlos Guedes. Interactive Simulation of Ship Motions in Random Seas based on Real Wave Spectra. // GRAPP. — 2011. — P. 235–244.

[3] Keeler Todd, Bridson Robert. Ocean waves animation using boundary integral equations and explicit mesh tracking // Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation / Eurographics Association. — 2014. — P. 11–19.

[4] Mathews John H., Fink Kurtis D. Numerical methods using MATLAB. — 4th edition. — London : Pearson Prentice Hall, 2004.

[5] Mathews John H., Fink Kurtis D. Runge—Kutta–Fehlberg method (RKF45). — 2004. — Access mode: `http://maths.cnam.fr/IMG/pdf/RungeKuttaFehlbergProof.pdf`.

[6] Matusiak Jerzy et al. Dynamics of a rigid ship. — Aalto University, 2013.

[7] Real-time simulation of ship motions in waves / Xiao Chen, Guangming Wang, Ying Zhu, G Scott Owen // International Symposium on Visual Computing / Springer. — 2012. — P. 71–80.

[8] Ma Xiaohu, Chen Zhiwei, Shi Gang. Real-time ocean wave motion simulation based on statistic model and GPU programming // The 2nd International Conference on Information Science and Engineering / IEEE. — 2010. — P. 3876–3880.

[9] Price James, McIntosh-Smith Simon. Oclgrind: An Extensible OpenCL Device Simulator // Proceedings of the 3rd International Workshop on OpenCL. — IWOCL '15. — New York, NY, USA : ACM, 2015. — P. 12:1–12:7. — Access mode: http://doi.acm.org/10.1145/2791321.2791333.

[10] Valiant Leslie G. A bridging model for parallel computation // Communications of the ACM. — 1990. — Vol. 33, no. 8. — P. 103–111.

[11] Gankevich Ivan, Degtyarev Alexander. Simulation of standing and propagating sea waves with three-dimensional ARMA model // The Ocean in Motion. — Springer, 2018. — P. 249–278.

[12] Micikevicius Paulius. 3D finite difference computation on GPUs using CUDA // Proceedings of 2nd workshop on general purpose processing on graphics processing units / ACM. — 2009. — P. 79–84.