УДК 004.02

Ляпустин Вячеслав Павлович,
аспирант, Ангарский государственный технический университет,
e-mail: lyapustinvp@gmail.com
Кривов Максим Викторович,
к.т.н., доцент, заведующий кафедрой «Вычислительных машин и комплексов»,
Ангарский государственный технический университет,
e-mail: vmk@angtu.com

РЕАЛИЗАЦИЯ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ НА ОСНОВЕ ПРОГРАММНО-АППАРАТНОГО СТЕКА CUDA

Lyapustin V.P., Krivov M.V.

REALIZATION OF PARALLEL COMPUTING ON THE BASIS OF THE CUDA HARDWARE-SOFTWARE STACK

Аннотация. В данной работе подробно описана проблематика реализации параллельных вычислений на основе программно-аппаратного стека CUDA. В ней произведён всесторонний анализ ключевых аспектов использования CUDA для параллельного программирования и обработки данных. В работе подробно рассматривается структура архитектуры CUDA, описывается модель программирования и процесс оптимизации.

Ключевые слова: параллельные вычисления, CUDA, графические процессоры, оптимизация, производительность.

Abstract. This paper describes in detail the problems of implementing parallel computing based on the CUDA hardware and software stack. It provides a comprehensive analysis of the key aspects of using CUDA for parallel programming and data processing. The paper examines in detail the structure of the CUDA architecture, describes the programming model and the optimization process.

Keywords: parallel computing, CUDA, graphic processors, optimization, performance.

Растущие требования к производительности вычислительных систем порождают необходимость использования параллельных вычислительных архитектур. Одним из лидирующих решений в этой области является программно-аппаратный стек CUDA от компании NVIDIA, позволяющий эффективно использовать вычислительные мощности графических процессоров (GPU) для выполнения массивно-параллельных вычислений.

Параллельные вычисления стали важнейшим элементом благодаря своей способности выполнять множество вычислений или процессов одновременно. Важным игроком в развитии и реализации параллельных вычислений является программно-аппаратный стек CUDA. Цель этого очерка - вникнуть в тонкости реализации параллельных вычислений с помощью программно-аппаратного стека CUDA, изучить его структуру, широкий спектр применения и многочисленные преимущества.

Когда мы говорим о параллельных вычислениях, мы имеем в виду тип вычислений, при котором несколько вычислений или процессов выполняются одновременно. Именно такое одновременное выполнение задач позволяет более эффективно решать большие и сложные проблемы. Каким образом? Раз-

бивая их на более мелкие и управляемые компоненты, которые можно решать параллельно.

Параллельные вычисления существуют в различных формах, каждая из которых имеет свои уникальные характеристики и преимущества. Параллелизм на уровне битов, например, увеличивает размер слова процессора, что позволяет одновременно обрабатывать больше данных. Существует также параллелизм на уровне инструкций, который направлен на уменьшение времени ожидания выполнения инструкций за счет их одновременного выполнения.

Параллелизм данных - это еще одна форма, которая предполагает одновременное выполнение одной и той же функции в нескольких точках данных. Параллелизм задач, с другой стороны, предполагает одновременное выполнение различных функций на одних и тех же или разных данных.

Основное преимущество параллельных вычислений заключается в их способности значительно сократить время, необходимое для решения задачи. Они позволяют выполнять сложные вычисления и задачи гораздо быстрее, чем это было бы возможно при использовании последовательных вычислений. Это приводит к повышению эффективности и производительности, что делает их предпочтительным выбором для многих [1].

В основе реализации параллельных вычислений лежит программно-аппаратный стек CUDA. Compute Unified Device Architecture (CUDA) - это платформа для параллельных вычислений и модель интерфейса прикладного программирования (API), созданная компанией Nvidia. Она позволяет разработчикам программного обеспечения и инженерам использовать графический процессор (GPU) с поддержкой CUDA для обработки данных общего назначения - такой подход называется GPGPU (General-Purpose computing on Graphics Processing Units).

Структура платформы CUDA очень интересна. Она состоит из программного слоя, который обеспечивает прямой доступ к виртуальному набору инструкций GPU и параллельным вычислительным элементам, облегчая выполнение вычислительных ядер. Проще говоря, она позволяет разработчикам использовать возможности GPU не только для рендеринга графики. Это открывает перед разработчиками возможности для использования его в более сложных задачах, требующих больших объемов данных.

Универсальность программно-аппаратного стека CUDA проявляется в его широком применении в различных областях. Например, в вычислительных финансах он используется для проведения сложных симуляций и оценки рисков. Аналогично, в вычислительной биологии он играет важнейшую роль в обработке больших массивов данных и выполнении сложных алгоритмов для понимания биологических процессов [2].

Сфера обработки изображений и видео также выигрывает от использования CUDA. Ее способность выполнять параллельные вычисления значитель-

но сокращает время, необходимое для процессов рендеринга и редактирования. Высокая вычислительная мощность позволяет быстрее обрабатывать отдельные пиксели, что приводит к ускорению обработки изображений и видео.

Одним из основных преимуществ CUDA, заслуживающих упоминания, является ее потенциал для повышения эффективности вычислений, особенно для приложений, требующих значительной вычислительной мощности. Она предоставляет простой интерфейс и абстракции для управления параллелизмом. В сочетании с массивным параллелизмом GPU это позволяет значительно повысить производительность многих типов вычислений.

Развитие науки и технологий в последние десятилетия неразрывно связано с ростом вычислительных мощностей компьютерных систем. Многие современные задачи, такие как моделирование физических процессов, обработка больших данных, машинное обучение и др., требуют огромных вычислительных ресурсов. Для удовлетворения этих потребностей постоянно ведутся разработки в области повышения производительности вычислительных систем [3].

Одним из наиболее перспективных направлений является параллельная обработка данных, основанная на использовании многопроцессорных систем. Параллельные вычисления позволяют распределять задачи между множеством вычислительных ядер, что существенно повышает общую производительность по сравнению с последовательной обработкой. Ключевым аспектом эффективной реализации параллельных вычислений является выбор соответствующей программно-аппаратной платформы.

В последние годы особую популярность в области параллельных вычислений приобрели графические процессоры (GPU), изначально разработанные для обработки графических данных. Современные GPU состоят из сотен и даже тысяч вычислительных ядер, способных выполнять параллельные вычисления с высокой степенью эффективности. Компания NVIDIA, один из лидеров в области графических решений, создала программно-аппаратный стек CUDA (Compute Unified Device Architecture), позволяющий программистам использовать вычислительные мощности GPU для решения общих вычислительных задач.

Решение дифференциальных уравнений, описывающих технологические процессы, является одной из важных областей применения параллельных вычислений на графических процессорах с использованием технологии CUDA. Рассмотрим подробнее подход к разработке таких решений.

Дифференциальные уравнения часто используются для моделирования динамических систем, таких как распространение тепла, течение жидкостей, распространение волн и многих других физических процессов. Численное решение этих уравнений требует значительных вычислительных ресурсов, особенно при высоких требованиях к точности и разрешению.

Графические процессоры (GPU) с их массивно-параллельной архитектурой идеально подходят для решения таких задач. CUDA предоставляет программистам возможность эффективно распараллеливать вычисления на GPU, что позволяет существенно ускорить решение дифференциальных уравнений по сравнению с традиционными последовательными реализациями на центральных процессорах (CPU).

Разработка решения для дифференциальных уравнений с использованием CUDA обычно включает следующие основные шаги:

1. Дискретизация уравнения. Первым шагом является преобразование дифференциального уравнения в дискретную форму, пригодную для численного решения. Это может быть достигнуто с помощью различных методов, таких как метод конечных разностей, метод конечных элементов или спектральные методы. Выбор метода зависит от характера уравнения и требований к точности.
2. Распараллеливание вычислений. После дискретизации уравнения необходимо разработать параллельный алгоритм для его решения на GPU. Это достигается путем распределения вычислений между множеством потоков CUDA, каждый из которых отвечает за вычисление значений в определенных точках пространственно-временной сетки.
3. Организация данных и доступа к памяти. Одним из ключевых аспектов эффективной реализации на GPU является оптимизация доступа к памяти. Необходимо тщательно спроектировать структуры данных и распределение данных между различными типами памяти GPU (глобальная, разделяемая, текстурная) для минимизации задержек и максимального использования пропускной способности памяти.
4. Ядра CUDA. Основная часть вычислений реализуется в виде ядер CUDA - специальных функций, выполняемых параллельно на множестве потоков GPU. Ядра CUDA должны быть тщательно спроектированы с учетом специфики решаемой задачи и архитектуры GPU для достижения максимальной производительности.
5. Обработка границ и граничных условий. При решении дифференциальных уравнений часто необходимо учитывать граничные условия, описывающие поведение системы на границах расчетной области. Эти условия должны быть правильно реализованы в параллельном алгоритме, что может потребовать дополнительной обработки для потоков, расположенных вблизи границ.
6. Верификация и валидация результатов. После разработки параллельного решения необходимо провести тщательную верификацию и валидацию полученных результатов, сравнивая их с аналитическими решениями или другими численными методами. Это позволит гарантировать корректность реализации и точность вычислений.

7. Оптимизация производительности. Для достижения максимальной производительности на GPU часто требуется применение различных оптимизационных стратегий, таких как оптимизация доступа к памяти, распределения ресурсов и использования специальных инструкций GPU. Инструменты профилирования CUDA могут помочь выявить узкие места и направления для оптимизации [4].

Следует отметить, что разработка эффективных решений для дифференциальных уравнений на GPU с использованием CUDA может быть сложной задачей, требующей глубокого понимания архитектуры GPU, программной модели CUDA и методов численного решения уравнений. Однако достижимые ускорения по сравнению с традиционными реализациями на CPU делают этот подход привлекательным для широкого круга задач моделирования технологических процессов.

Технология CUDA основана на использовании специализированных функций, которые доступны ядрам. Они выполняются параллельно на графическом процессоре в качестве различных потоков. Таким образом мы можем установить связь ядро – аналоговая функция. Потоки разделены по ядрам и выполняются с использованием своих индивидуальных инструкций и локальной памяти. Группировка потоков одинакового размера и последующее их распределение по различным мультипроцессорам позволяет нам извлечь максимальную эффективность от каждого из ядер. Ограничение числа потоков в цепи можно уточнить при помощи функций API. Основой является качество взаимодействия потоков внутри общего блока данных, принадлежащих отдельному ядру. Оно позволяет нам эффективно взаимодействовать с памятью, добиваясь синхронизации.

CUDA представляет собой программно-аппаратную архитектуру, разработанную NVIDIA для реализации массивно-параллельных вычислений на GPU. Она включает в себя аппаратную часть, состоящую из специализированных вычислительных ядер на GPU, и программную часть, представляющую собой расширение языков программирования C и C++ для написания параллельных программ.

Графические процессоры NVIDIA, поддерживающие CUDA, содержат множество вычислительных ядер, объединенных в так называемые мультипроцессоры (Streaming Multiprocessors, SM). Каждый мультипроцессор содержит несколько потоковых процессоров (Streaming Processors, SP), которые являются основными вычислительными единицами GPU. Кроме того, на GPU присутствуют различные типы памяти, такие как:

1. Глобальная память: большой объем памяти, доступный для чтения и записи всеми потоками на GPU.

2. Разделяемая память: небольшой объем памяти с низкой латентностью, доступный для чтения и записи потоками внутри одного мультипроцессора.
3. Регистровая память: набор регистров, используемых для хранения локальных переменных потока.
4. Константная память: кэшируемая область памяти для хранения констант, доступных для чтения потоками.
5. Текстурная память: кэшируемая область памяти, оптимизированная для доступа к данным по массивным шаблонам.

CUDA позволяет программистам писать параллельный код, используя расширения языков C и C++. Основной единицей параллельных вычислений в CUDA является ядро (kernel) - специальная функция, которая выполняется одновременно на множестве потоков GPU. Потоки объединяются в блоки (blocks), а блоки - в сетку (grid) [2].

Когда программа CUDA запускает ядро, она определяет конфигурацию сетки и блоков, а также передает необходимые данные из памяти CPU в память GPU. Затем каждый поток выполняет свою часть вычислений параллельно с другими потоками. После завершения вычислений результаты могут быть скопированы обратно в память CPU.

Выделяются следующие ключевые аспекты программной модели CUDA:

1. Иерархия потоков: потоки организованы в иерархическую структуру блоков и сеток, что позволяет эффективно распараллеливать вычисления и управлять доступом к памяти.
2. Разделяемая память: потоки внутри одного блока могут совместно использовать разделяемую память для обмена данными и синхронизации.
3. Барьеры синхронизации: механизмы синхронизации, позволяющие координировать выполнение потоков внутри блока.
4. Взаимодействие устройств: механизмы передачи данных между памятью CPU (хост) и памятью GPU (устройство).

Одним из простейших примеров параллельных вычислений с CUDA является операция векторного сложения двух векторов. Каждый поток в этом случае отвечает за сложение соответствующих элементов векторов.

```
__global__ void vecAdd(float* a, float* b, float* c, int n)
{
    int idx = blockDim.x * blockIdx.x + threadIdx.x;
    if (idx < n)
        c[idx] = a[idx] + b[idx];
}
```

В данном примере ядро vecAdd выполняется параллельно многими потоками. Каждый поток определяет свой индекс idx в соответствии с его позицией в сетке блоков и внутри блока. Затем поток выполняет сложение элементов векторов a и b с соответствующим индексом и записывает результат в вектор c.

Матричное умножение: Другим распространенным примером является операция умножения матриц. В данном случае каждый поток отвечает за вычисление одного элемента результирующей матрицы путем выполнения скалярного произведения соответствующей строки и столбца исходных матриц.

```
__global__ void matMul(float* a, float* b, float* c, int M, int N, int K)
{
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

    if (row < M && col < N) {
        float sum = 0.0f;
        for (int i = 0; i < K; i++)
            sum += a[row * K + i] * b[i * N + col];
        c[row * N + col] = sum;
    }
}
```

В этом примере ядро matMul запускается с конфигурацией сетки и блоков, соответствующей размерам исходных матриц. Каждый поток определяет свои индексы строки row и столбца col в результирующей матрице. Затем поток вычисляет элемент c[row * N + col] путем перемножения соответствующей строки матрицы a и столбца матрицы b с использованием цикла для суммирования произведений элементов.

Обработка изображений: CUDA также широко применяется для обработки изображений и компьютерного зрения. Одним из примеров может служить применение свертки для выделения границ на изображении.

```
__global__ void sobelFilter(unsigned char* in, unsigned char* out, int width, int height)
{
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;

    if (x >= width || y >= height)
        return;
```

```
    // Применение ядра свертки Собеля
    int gx = 0, gy = 0;
    for (int j = -1; j <= 1; j++) {
        for (int i = -1; i <= 1; i++) {
            int pos = (y + j) * width + (x + i);
            gx += in[pos] * sobelX[j + 1][i + 1];
            gy += in[pos] * sobelY[j + 1][i + 1];
        }
    }

    int g = sqrt(gx * gx + gy * gy);
    out[y * width + x] = (unsigned char)min(g, 255);
}
```

В этом примере ядро sobelFilter применяет фильтр Собеля для выделения границ на входном изображении in. Каждый поток обрабатывает один пиксель изображения, вычисляя градиенты gx и gy путем свертки с ядрами sobelX и sobelY. Затем поток вычисляет общую величину градиента g и записывает результат в выходное изображение out.

Для достижения максимальной производительности при параллельных вычислениях на GPU с использованием CUDA необходимо учитывать ряд оптимизационных стратегий и рекомендаций. Некоторые из ключевых аспектов оптимизации включают:

1. Оптимизация доступа к памяти: правильная организация доступа к глобальной памяти, использование разделяемой памяти и кэширования.
2. Распределение ресурсов: эффективное распределение потоков, блоков и регистров для максимального использования вычислительных ресурсов GPU.
3. Оптимизация инструкций: использование специальных инструкций GPU, таких как операции с числами с плавающей запятой, векторные операции и др.
4. Стратегии параллелизма: выбор подходящих стратегий параллелизма (параллелизм по данным, по задачам, комбинированный) в зависимости от характера задачи.
5. Профилирование и анализ производительности: использование инструментов профилирования и анализа производительности CUDA для выявления узких мест и оптимизации кода.

Программно-аппаратный стек CUDA от NVIDIA предоставляет мощный инструментарий для реализации массивно-параллельных вычислений с использованием вычислительных ресурсов графических процессоров. Благодаря своей архитектуре и программной модели, CUDA позволяет эффективно рас-

параллеливать широкий спектр задач, от простых векторных операций до сложных вычислений в области обработки изображений, моделирования и машинного обучения.

Успешная реализация параллельных вычислений с CUDA требует глубокого понимания архитектуры GPU, программной модели и оптимизационных стратегий. Тщательное проектирование алгоритмов, организация доступа к памяти и распределение вычислительных ресурсов имеют решающее значение для достижения высокой производительности.

По мере дальнейшего развития технологий и появления новых вычислительных задач, параллельные вычисления на основе CUDA, несомненно, будут играть важную роль в обеспечении требуемых вычислительных мощностей. Совершенствование программно-аппаратных решений и методик параллельного программирования открывает перспективы для решения всё более сложных задач в области науки, инженерии и других областях человеческой деятельности.

ЛИТЕРАТУРА

1. **Малявко, А. А.** Параллельное программирование на основе технологий openmp, cuda, opencl, mpi : учебное пособие для вузов / А. А. Малявко. - 3-е изд., испр. и доп. - Москва : Издательство Юрайт, 2023. - 135 с. - (Высшее образование). - ISBN 978-5-534-14116-0. - Текст: электронный // Образовательная платформа Юрайт [сайт]. - URL: https://urait.ru/bcode/514199 (дата обращения: 12.04.2024).
2. **Немнюгин, С.А.** Введение в программирование на кластерах / С.А.Немнюгин. – 2 - е изд., испр. - Москва : Национальный Открытый Университет «ИНТУИТ», 2016. - 247 с.
3. **Николаев, Е.И.** Параллельные вычисления : учебное пособие / Е.И.Николаев ; Министерство образования и науки Российской Федерации, Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Северо-Кавказский федеральный университет». - Ставрополь : СКФУ, 2016. - 185 с.
4. Основы высокопроизводительных вычислений : учебное пособие / **К.Е. Афанасьев, С.Ю. Завозкин, С.Н. Трофимов, А.Ю. Власенко. - Кемерово** : Кемеровский государственный университет, 2011. - Т. 1. Высокопроизводительные вычислительные системы. - 246 с.