Практическая работа №2

ст. преп. каф. ВпВ ИКИТ СФУ Тарасов С. А.

Цель работы

Закрепить базовые навыки программирования CUDA. Освоить работу с двумерными сетками нитей.

Задание

1. Разработать класс `Matrix` (см. рис. 1), который представляет собой `фасад`, объединяющий:
   - агрегацию с классом `Data` через `std::shared_ptr` (для разделения данных между матрицами);
   - композицию с классом `MatrixView` (каждый экземпляр `Matrix` имеет собственное представление);
   - единый упрощённый интерфейс для работы с матрицами.
2. Разработать класс `MatrixView`, предоставляющий интерфейс для доступа к элементам матрицы на устройстве (чтение и запись по индексам, row-major). Класс должен быть тривиально-копируемым.
3. Разработать кёрнел `kernel_matmul_naive`, который принимает объекты `MatrixView` по значению и вычисляет произведение матриц без использования разделяемой памяти.
4. Перегрузить оператор `operator*` для класса `Matrix`, используя указанный кёрнел.
5. Используя фреймворк `Google Test`, разработать модульные тесты для `operator*` со следующими размерами матриц: A $(m \times k)$ и B $(k \times n)$, где $m, n, k \in \{1, 2, 3, 127, 128, 129, 512\}$. В качестве эталона для сравнения использовать результат аналогичной операции для `Eigen::MatrixXf`; для верификации результатов применять метод `Eigen::MatrixXf::isApprox` с абсолютной точностью 10^{-5}.
6. Используя фреймворк `Google Benchmark`, разработать бенчмарки для `operator*` со следующими размерами матриц: $n \times n$, где $n \in \{16, 32, 64, 128, 256, 512, 1024\}$. Бенчмарки должны игнорировать время, затраченное на выделение, копирование и освобождение памяти. Для корректного измерения времени выполнения CUDA-кода необходимо использовать `CUDA Events API`.

7. Построить график реальной вычислительной сложности (`real complexity`, пример на рис. 2) `operator*` для умножения матриц типа `Matrix` и аналогичный график для умножения матриц типа `Eigen::MatrixXf`.

8. Построить график ускорения (`speedup`, пример на рис. 3) `operator*` для умножения матриц типа `Matrix` относительно `operator*` для `Eigen::MatrixXf`.

9. Сравнить экспериментальные результаты с теоретическими оценками вычислительной сложности.

10. Подготовить отчёт, содержащий:

    - ключевые фрагменты кода;
    - ссылку на репозиторий с полной реализацией;
    - графики результатов измерений;
    - анализ и интерпретацию полученных результатов.

Критерии оценки

- **Корректность реализации и тестирование (50%):**
  - отсутствие утечек памяти, корректная работа с `CUDA API`;
  - правильность результатов умножения матриц различных размеров;
  - полнота тестового покрытия, включая граничные случаи;
  - соответствие результатов эталонной реализации.
- **Качество кода и архитектура (25%):**
  - корректная реализация паттерна `data + view`;
  - чистота архитектуры, разделение ответственности между классами;
  - соблюдение идиомы `RAII`, корректное использование умных указателей;
  - единообразие стиля, качество форматирования и читаемость кода.
- **Качество вычислительного эксперимента (15%):**
  - корректность методики измерений производительности;
  - глубина анализа результатов, сравнение с теоретическими оценками.
- **Документация и оформление (10%):**
  - полнота и структурированность отчёта;
  - ясность изложения;
  - оформление репозитория;
  - оформление отчёта (СТУ 7.5–07–2021).

Рекомендации по выполнению

- Реализуйте классы `MatrixView` и `Matrix` как шаблоны, параметризованные числовыми типами.
- Используйте линейную организацию памяти для матриц (row-major или column-major).
- Для сборки проекта применяйте `CMake + Ninja`.
- Используйте стандарт `C++20` (более новые стандарты могут не поддерживаться `NVCC`).
- Для параметризованного тестирования используйте `Combine (Google Test)`.
- Для построения графиков применяйте `Python` с пакетом `plotly`.
- Учтите, что Eigen по умолчанию использует column-major размещение элементов.
- Ознакомьтесь с официальным руководством CUDA.
- Структура проекта:

```
hsys/
├── work1/
├── work2/
│   ├── core/
│   │   ├── include/
│   │   ├── src/
│   │   └── CMakeLists.txt
│   ├── tests/
│   │   ├── include/
│   │   ├── src/
│   │   └── CMakeLists.txt
│   ├── benchmarks/
│   │   ├── include/
│   │   ├── src/
│   │   └── CMakeLists.txt
│   └── CMakeLists.txt
├── CMakeLists.txt
└── build/
```

Теоретическая справка

Умножение матриц $C = AB$, где A имеет размер $m \times k$, а B — $k \times n$, имеет вычислительную сложность $O(m \cdot n \cdot k)$.

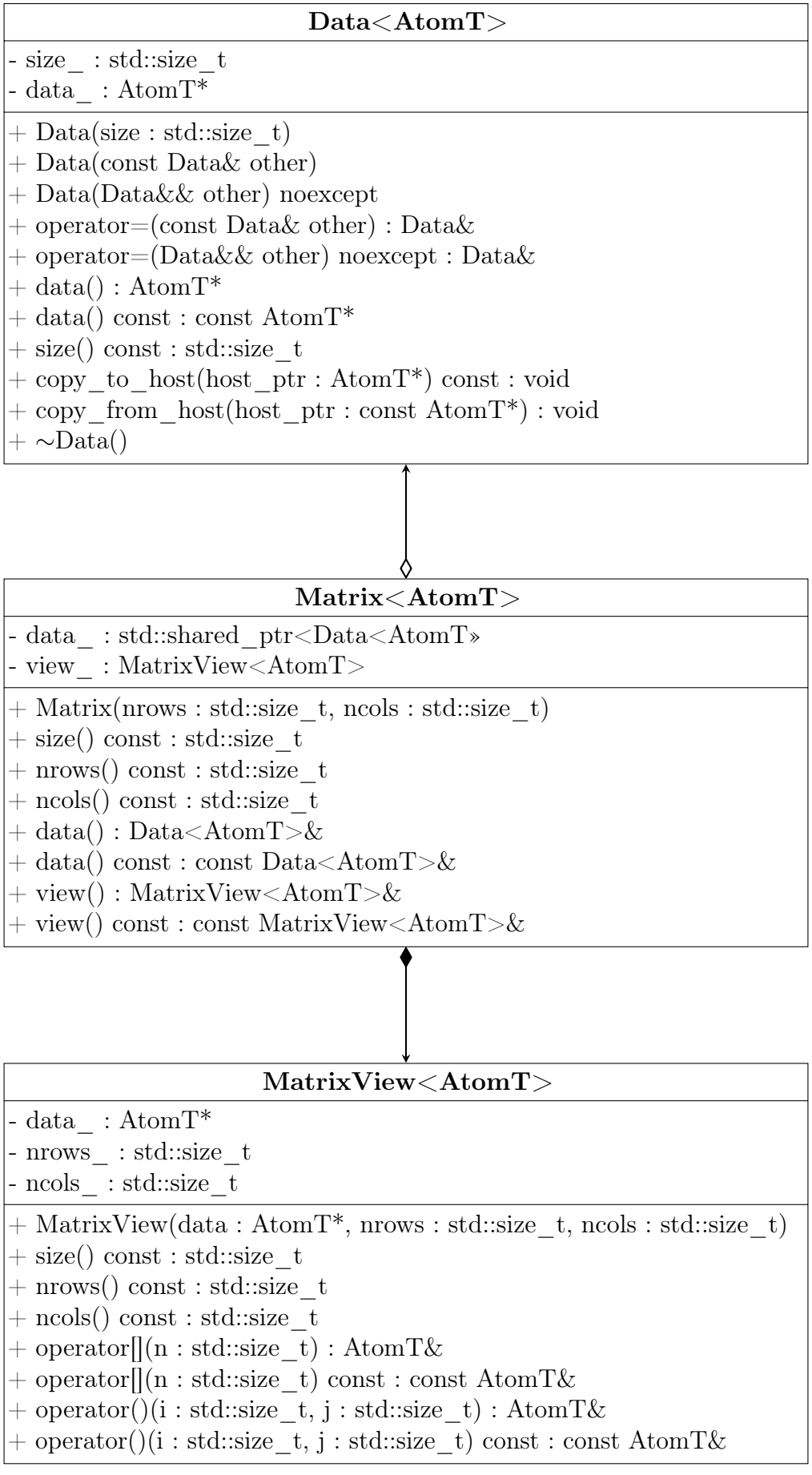

Рис. 1: Диаграмма классов

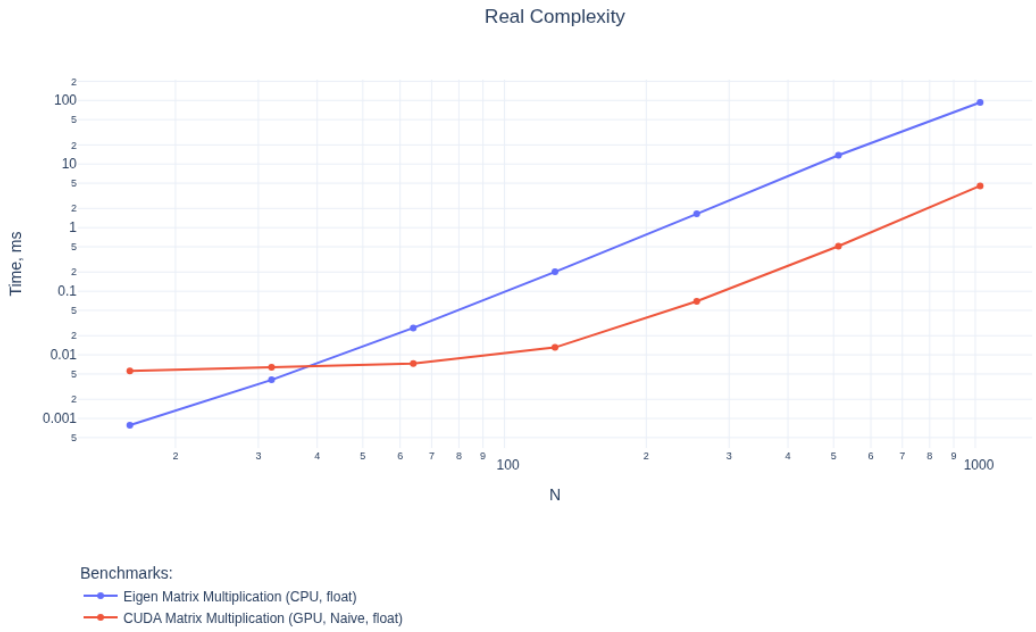

Рис. 2: Графики реальной вычислительной сложности

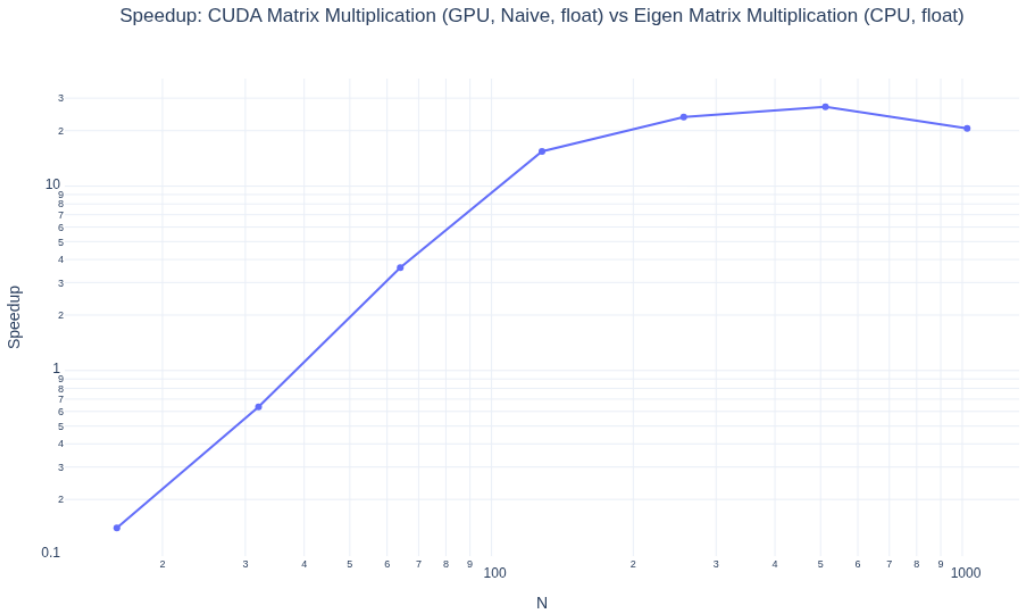

Рис. 3: График ускорения