Практическая работа №1

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

Цель

Освоить базовые навыки программирования CUDA: работу с одномерными сетками нитей и динамической памятью устройства. Изучить паттерн проектирования `data + view`.

Задание

1. Разработать класс `Data`, который реализует идиому RAII и предоставляет интерфейс для работы с данными на устройстве: динамическое выделение и освобождение массива глобальной памяти, копирование и перемещение данных.

2. Разработать класс `VectorView`, который предоставляет интерфейс для обращения к данным на устройстве: доступ к элементам данных для чтения и записи. Класс должен быть тривиально-копируемым.

3. Разработать класс `Vector` (см. рис. 1), который представляет собой `фасад`, объединяющий:

   - агрегацию с классом `Data` через `std::shared_ptr` (для разделения данных между векторами);
   - композицию с классом `VectorView` (каждый экземпляр `Vector` имеет собственное представление);
   - единый упрощённый интерфейс для работы с векторами.

4. Разработать кёрнел `kernel_vecadd`, который в качестве аргументов принимает по значению объекты класса `VectorView` и вычисляет сумму векторов.

5. Перегрузить оператор `operator+` для класса `Vector`, используя указанный кёрнел.

6. Используя фреймворк `Google Test`, разработать модульные тесты для `operator+` с размерами векторов $n \in \{1, 2, 3, 127, 128, 129, 512, 1024, 1029\}$. В качестве эталона для сравнения использовать результат аналогичной операции для `Eigen::VectorXf`; для верификации результатов применять метод `Eigen::VectorXf::isApprox` с абсолютной точностью 10^{-6}.

7. Используя фреймворк `Google Benchmark`, разработать бенчмарки для `operator+` с размерами векторов $n \in \{8, 8^2, 8^3, 8^4, 8^5, 8^6, 8^7, 8^8\}$. Бенчмарки должны игнорировать время, затраченное на выделение, копирование и освобождение памяти. Для корректного измерения времени выполнения CUDA-кода необходимо использовать `CUDA Events API`.

8. Построить график реальной вычислительной сложности (`real complexity`, пример на рис. 2) `operator+` для сложения векторов типа `Vector` и аналогичный график для сложения векторов типа `Eigen::VectorXf`.
9. Построить график ускорения (`speedup`, пример на рис. 3) `operator+` для сложения векторов типа `Vector` относительно `operator+` для `Eigen::VectorXf`.
10. Сравнить результаты измерений с теоретическими оценками.
11. Подготовить отчёт о проделанной работе, который содержит:
    - фрагменты исходного кода (`Data`, `VectorView`, `Vector`, `kernel_vecadd`);
    - ссылку на репозиторий с полным кодом;
    - результаты измерений в виде графиков;
    - интерпретацию результатов измерений.

Критерии оценки

- **Корректность реализации и тестирование (50%):**
  - отсутствие утечек памяти, корректная работа с `CUDA API`;
  - правильность результатов сложения векторов 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 к выполнению работы

- Сделайте классы `Data`, `VectorView` и `Vector` шаблонами, зависящими от числовых типов.
- Основной код оформите в виде библиотеки.
- Для сборки проекта используйте `CMake + Ninja`.
- Используйте `C++20`. Более новые стандарты не поддерживаются `NVCC`.
- Вы можете использовать `NVIDIA Nsight Visual Studio Code` в качестве IDE.
- Вы можете использовать `Zed Editor + clangd + clang-format + clang-tidy` в качестве IDE.
- Для построения графиков используйте `Python` и пакет `plotly`.
- Вы можете использовать `LaTeX` для создания отчёта.
- Ознакомьтесь с официальным руководством CUDA.
- Пример структуры проекта:

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

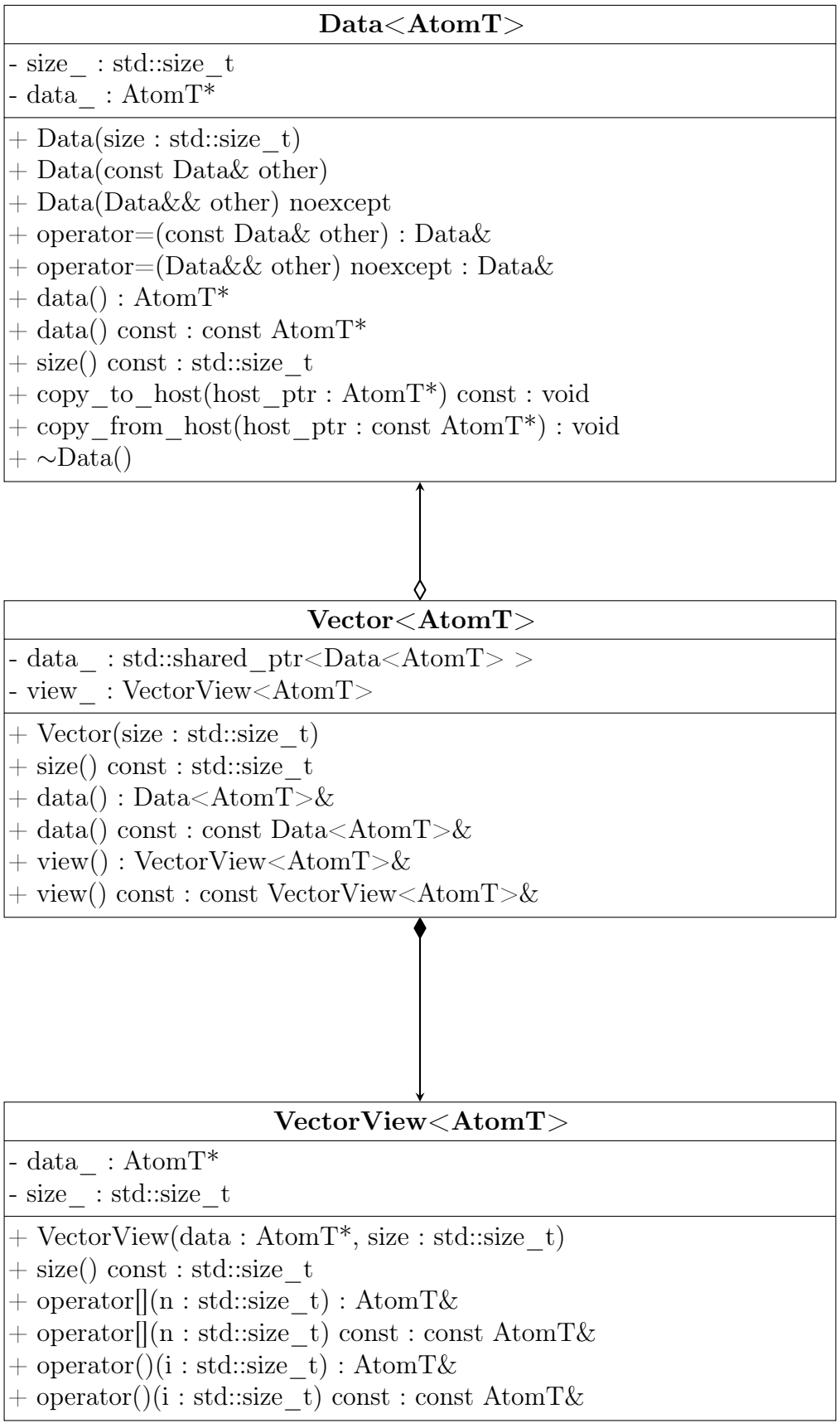

Рис. 1: Диаграмма классов

Рис. 2: Графики реальной вычислительной сложности

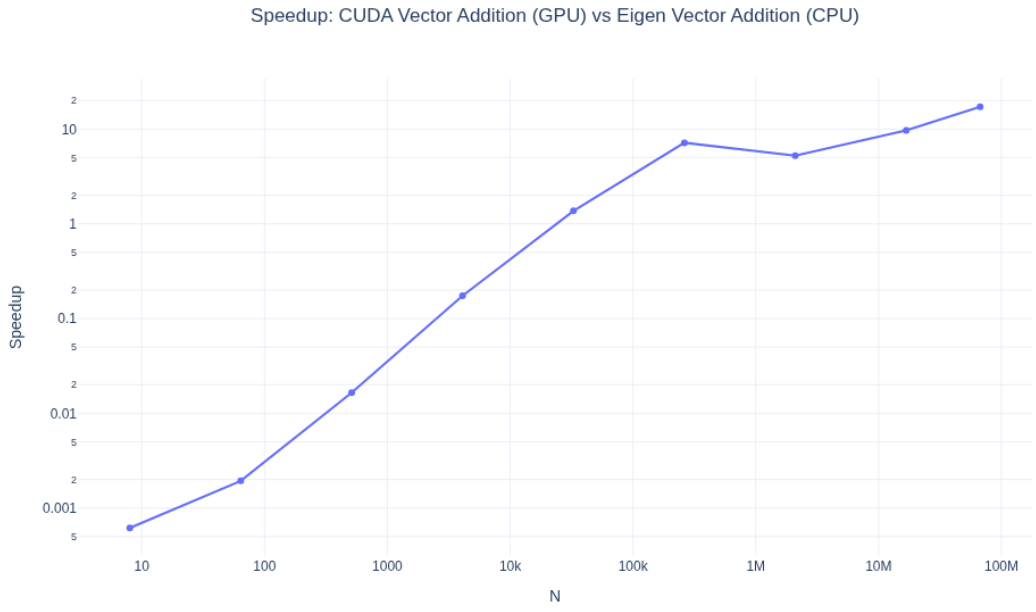

Рис. 3: График ускорения