Особенности разработки под графические ускорители на примере OpenCL: от архитектуры вычислителя до особенностей оптимизации

Семён Григорьев

Снакт-Петербургский Государственный Университет

27 января 2025

Семён Григорьев

- Доцент кафедры системного программирования Санкт-Петербургского Государственного Университета
- Области интересов
  - **Высокопроизводительный анализ графов**
    - ★ В том числе, с использованием **графических ускорителей**
  - **Высокопроизводительная линейная алгебра** для анализа графов
    - ★ **Обобщённая**: матрицы и вектора параметризованы типом элемента, операции над ними могут быть заданы пользователем
    - ★ **Разреженная**: специализированные структуры для хранения матриц и векторов, специализированные алгоритмы для их обработки
    - ★ В том числе, с использованием **графических ускорителей**
  - Высокоуровневые средства разработки для графических ускорителей

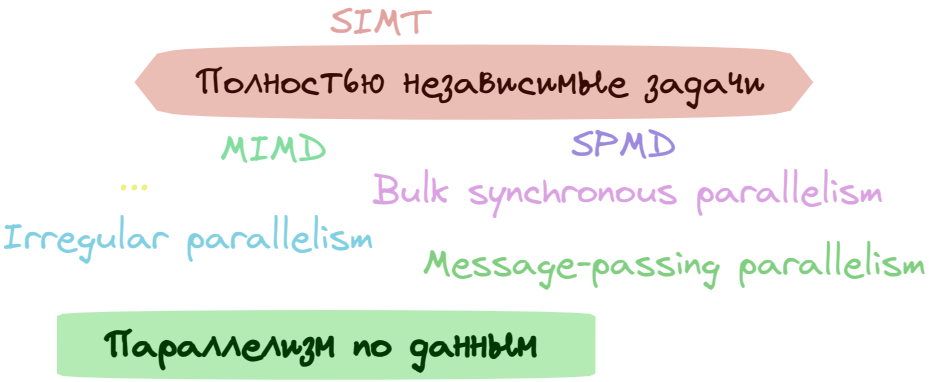
- Email: rsdpisuy@gmail.com
- GitHub: gsvgit
- Google Scholar: Semyon Grigorev
- DBLP: Semyon V. Grigorev

План лекции

Цель лекции: познакомить с графическими ускорителями, показать, на что стоит обращать внимание при первых попытках программировать под них

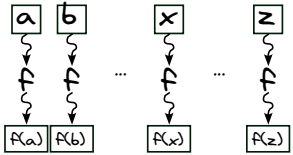
1. Параллельность: что, зачем, какая бывает, какая нас будет интересовать
2. CPU и GPGPU: сходства, различия, взаимосвязь
3. Основные принципы архитектуры GPGPU
4. Основы OpenCL
5. Немного про оптимизации для GPGPU

Параллелизм



Параллелизм по данным

- Одна и та же функция применяется **независимо** ко всем элементам набора данных[1]



- Single Instruction Multiple Data (SIMD)
  - Не функция, но **инструкция**: поток управления функции должен быть максимально простым
    - ⇒ Много ветвлений — **плохо**[2]
    - ⇒ Много однотипных вычислений — **хорошо**
    - ⇒ Данные должны быть **однородными**
  - Типично для обработки изображений, линейной алгебры
    - ★ Вектора, матрицы — хорошие массивы данных
    - ★ Много вычислительно интенсивной, но «однотипной» работы

[1] Можно представить map, iter и прочие аналоги на коллекциях

[2] Современные GPGPU стараются бороться с этим ограничением

CPU

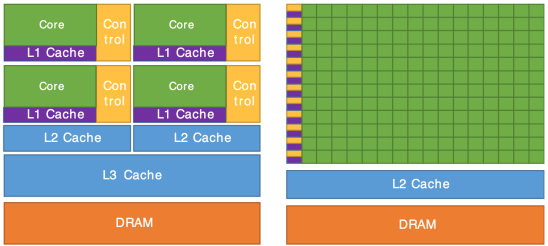
- Процессор общего назначения
- Независимые задачи
- Много сложных кэшей
- Сложный поток управления

GPGPU

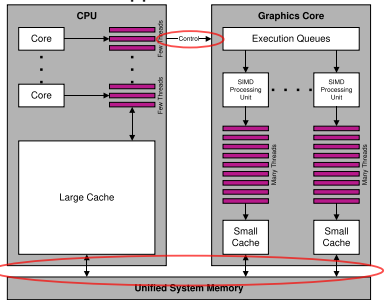
- Специализированный (co)процессор
- Параллелизм по данным
- Высокая пропускная способность
- Большое количество простых ALU

Типичный «взгляд издалека»

Сравнение архитектур[3]



Взаимодействие в SoC[4]



[3] https://cvw.cac.cornell.edu/gpu-architecture/gpu-characteristics/design

[4] PowerVR Hardware. Architecture Overview for Developers

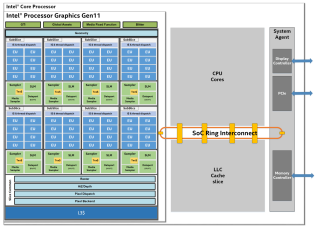
Проблемы[5]

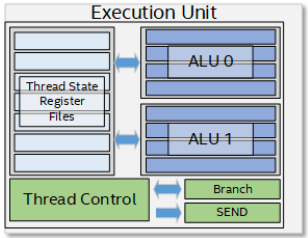
- Масса вендоров:
  - Nvidia
  - AMD
  - Intel
  - Imagination Technologies
  - Qualcomm
  - ...
- Микроархитектура различных графических ускорителей сильно различается
- Архитектура различных графических ускорителей сильно различается
- Терминология сильно различается

Nvidia	AMD	Intel	Im.Tech.
Streaming Processor	SIMD lane	Processing element	ALU Group
SIMT unit	SIMD unit	Vector engine	ALU Pipeline
Streaming Multiprocessor	Computing Unit	Execution Unit	Unified Shading Cluster
GPU processing clusters	Compute Engine	Xe-slice	Core

[5] как всегда, в деталях

Интегрированная графика от Intel[6,7]



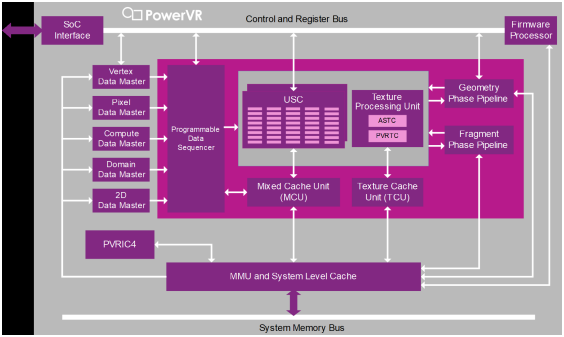


- Много специфики по обработке изображений
- Иерархическая группировка элементов
- Общий с процессором кэш

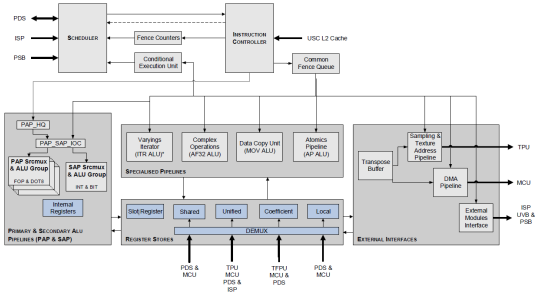
[6] Древняя, Gen11

[7] Изображения из «Intel® Processor Graphics Gen11 Architecture»

Графичекий ускоритель от Imagination Technologies (PowerVR)[8,9]



Общий вид

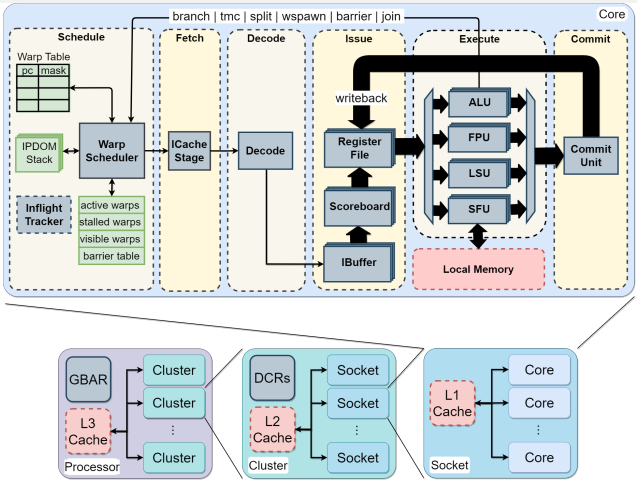


Детали Unified Shading Cluster

[8] Примерно то, с чем некоторые столкнутся на школе

[9] Изображения из документации: `https://docs.imgtec.com/performance-guides/compute-recommendations/html/topics/architecture/overview.html`

Микроархитектура RISC-V GPGPU Vortex[10,11]



[10] Примерно то, с чем другие некоторые столкнутся на школе

[11] Изображение со страницы проекта:
https://github.com/vortexgpgpu/vortex/blob/master/docs/microarchitecture.md

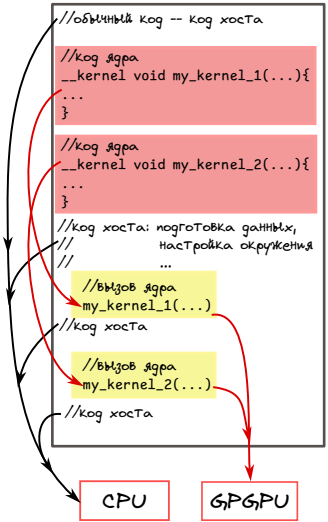
OpenCL: Open Computing Language

- Khronos[12]: «Open Standard for **Parallel** Programming of **Heterogeneous** Systems»
  - Многоядерные CPU
  - **Графические ускорители, GPGPU**
  - Программируемые логические интегральные семы, ПЛИС/FPGA
  - Процессоры для цифровой обработки сигналов, DSP
  - ...
- Активно разрабатывается и используется
  - Intel, Google, Nvidia, AMD, Imagination Technologies, ...
  - OpenCV, FFmpeg, ViennaCL, GNU Octave, Matlab, ...
- Версии стандарта
  - 1.2: 2011 год, обязательная база для 3.0
  - 2.x: 2013 год, «откатили» в 3.0
  - 3.0: 2020 год, 3.0.17 в октябре 2024

[12] Консорциум для работы над открытыми стандартами в области параллельных вычислений, машинного обучения, компьютерной графики: больше 180 организаций, 21 стандарт. `https://www.khronos.org/`.

OpenCL: структура программы

- **Часть кода выполняется на центральном процессоре, часть на ускорителе**
- Центральный процессор (**«хост», host**)
  - Взаимодействие с «внешним миром», подготовка данных, запуск кода на ускорителе
  - Хочется писать код на «типичном» языке программирования
- Ускоритель (**«устройство», device**)
  - Обработка данных «по указке» центрального процессора
  - Придётся пользоваться специализированным языком
  - Процедура, предназначенная для выполнения на ускорителе — **«ядро», kernel**[13]



[13] OpenCL kernel, CUDA kernel, ...

OpenCL: взаимодействие хоста и устройства

- Обнаружение, подключение, конфигурация устройств[14]
- Управление памятью[15]
  - Выделение/освобождение
  - Контроль доступа и синхронизации между хостом и устройствами
- Выполнение команд
  - Передача данных, запуск ядер, синхронизация, ...
  - Основной интерфейс взаимодействия — **очередь команд**[16]
    - Неблокирующая: центральный процессор ставит задачи, но не дожидается их исполнения
    - Нужны дополнительные действия чтобы узнать, когда закончилась определённая задача
    - In-order: команды выполняются строго друг за другом
    - Может быть out-of-order
    - Синхронизация между несколькими очередями возможна, но требует отельной работы

[14] ICD-Loader: `https://github.com/KhronosGroup/OpenCL-ICD-Loader`

[15] Основной примитив — буфер:
`https://registry.khronos.org/OpenCL/specs/3.0-unified/html/OpenCL_API.html#_buffer_objects`

[16] Command queue:
`https://registry.khronos.org/OpenCL/specs/3.0-unified/html/OpenCL_API.html#_command_queues`

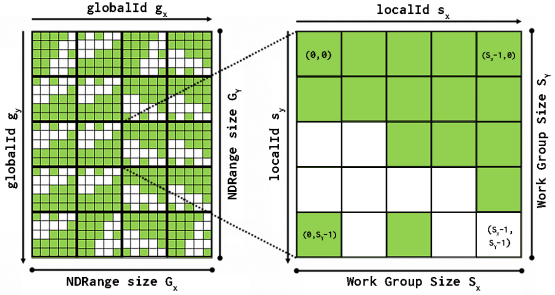
Язык для написания ядер: OpenCL C/C++

- C++ for OpenCL language[17]
- **OpenCL C language**[18]
  - Диалект C99: специальные типы данных[19], модификаторы памяти[20]
  - JIT-компиляция[21], но бывают и AOT решения
  - Дополнительные библиотеки времени выполнения
  - Расширения[22]
    - Предусмотренные стандартом[23]: `cl_khr_fp64, cl_khr_int64_extended_atomics, ...`
    - Расширения от вендоров[24]: `cl_nv_pragma_unroll, cl_amd_fp64, ...`

[17] В 2.x был OpenCL C++ Kernel Language
[18] Спецификация: `https://registry.khronos.org/OpenCL/specs/3.0-unified/html/OpenCL_C.html`
[19] `https://registry.khronos.org/OpenCL/specs/3.0-unified/html/OpenCL_C.html#supported-data-types`
[20] `https://registry.khronos.org/OpenCL/specs/3.0-unified/html/OpenCL_C.html#address-space-qualifiers`
[21] Для переносимости: вендору целевого устройства «лучше знать», как компилировать
[22] Знакомя ситуация, не правда ли?
[23] Поискать «Khronos Extension Specifications» на `https://registry.khronos.org/OpenCL/`
[24] Поискать «Vendor and Multi-Vendor Extension Specifications» на `https://registry.khronos.org/OpenCL/`

OpenCL C: параллельность

- Виртуальная решётка (**NDRange**) «потоков» (**work-items**)
  - одно-, двух-, трёхмерная
  - Группировка узлов в рабочие группы (**work-group**) фиксированного размера
  - Возможность получить глобальные и локальные координаты «потока»
- Параметры (размеры) решётки связаны с «размерами» обрабатываемых данных



Пример вдухмерной виртуальной решётки[25]

- `globalId` — глобальные координаты
- `localId` — локальные координаты

[25] Изображение из поста: `http://thebeardsage.com/opencl-kernel-space/`

И снова проблемы[29]

? Как отобразить абстракции OpenCL в реальное устройство?[26]

? Как писать код, чтобы он показывал хорошую[27] производительность?

☹ Придётся писать много вендоро/платформо/device-специфичного кода.

- ☺ Кое-что полезное об устройстве можно узнать во время выполнения[28]
- 😐 От перебора вороха вариантов в коде всё равно не уйти

! Есть некоторые общие принципы

[26] При таком-то зоопарке устройств/микроархтектур

[27] Хотелось бы, конечно, максимальную но ...

[28] Стандартный API для получения данных об устройстве: `https://registry.khronos.org/OpenCL/specs/3.0-unified/html/OpenCL_API.html#platform-querying-devices`

[29] «Шо, опять?»

Общие принципы работы с очередью команд

Помни!

- Постановка задачи в очередь — накладные расходы
- Разные задачи в очереди задействуют разные ресурсы
- Одна задача может не утилизировать какой-то ресурс полностью
- Ожидание завершения задачи может привести к потере вычислительного времени

Потому

- Не дроби ядра
- Используй разные очереди или out-of-order[30]
- Ставь в очередь как можно больше задач[31]
- Пользуйся тем, что очередь неблокирующая
- Уменьшай количество точек синхронизации

[30] Там, где это возможно

[31] В разумных пределах

Общие принципы работы с памятью

Помни!

- Передача данных[32] — накладные расходы
- Память бывает разная: объём и время доступа различаются на порядки
- Последовательный доступ — хорошо, произвольный — плохо

Потому

- Помни что где лежит и клади нужное поближе[33]
- Думай над тем, как представить данные в памяти
- Структурируй код ядер так, чтобы минимизировать непоследовательный доступ к памяти

[32] Как между хостом и устройством, так и между устройствами

[33] Изучи модификаторы памяти в OpenCL C и настройки буферов

Общие принципы написания ядер

Помни!

- Ветвления[34] в ядрах — зло[35]
  - Соответствующее неприятное явление называется **thread divergence** или **branch divergence**
- На аппаратном уровне **вполне определённая**[36] группа «потоков» гарантированно выполняет одну инструкцию и рабочая группа как правило отображается в неё

Потому

- Старайся писать как можно более линейный код
  - Иногда ради этого приходится писать несколько ядер
- Подбирай размер рабочей группы так, чтобы максимально использовать аппаратные ресурсы

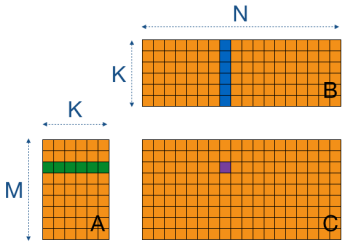
[34] И речь не только об `if-then-else`

[35] К сожалению, иногда — необходимое зло

[36] В зависимости от вендора: Task, Thread Group, Warp, Wavefront, ...

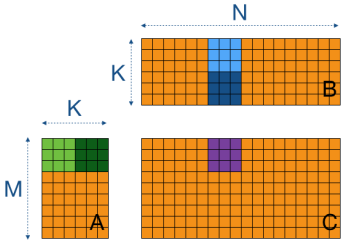
Классический пример[37]

Наивная реализация



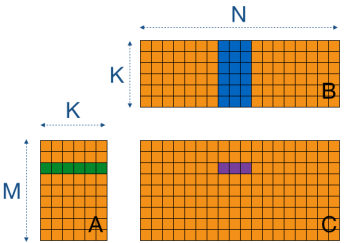
139 GFLOPS

Локальная память



373 GFLOPS

Регистры



689 GFLOPS

- cuBLAS в тех же условиях: 3056 GFLOPS

[37] Материалы из туториала https://cnugteren.github.io/tutorial/pages/page1.html.
Соответствующий код: https://github.com/cnugteren/myGEMM

Немного полезных ссылок

- Khronos OpenCL
  - https://www.khronos.org/opencl/
- Курс по разработке под GPGPU от EuroCC National Competence Center Sweden
  - https://enccs.github.io/gpu-programming/
- OpenCL для ускорителей от Imagination Technologies
  - https://blog.imaginationtech.com/a-quick-guide-to-writing-opencl-kernels-for-rogue/
- Рекомендации разработчику под ускорители от Imagination Technologies
  - https://docs.imgtec.com/performance-guides/compute-recommendations/html/index.html
- Cornell University, Understanding GPU Architecture
  - https://cvw.cac.cornell.edu/gpu-architecture
- Антонюк В. А. OpenCL Открытый язык для параллельный программ, учебное пособие, 2017.
  - https://istina.msu.ru/publications/book/76785028/