



Казанский
федеральный
университет

ВЫСШАЯ ШКОЛА
информационных технологий
и информационных систем

Оптимизация Отладка OpenCL библиотеки

Эдуард Храмченков

Гетерогенные вычисления

- ▶ Ядра могут запускаться одновременно на нескольких платформах устройствах
- ▶ Необходимо определить контекст, включающий несколько платформ, устройств и очередей
- ▶ Можно создавать несколько контекстов
- ▶ Для работы с несколькими платформами необходима установка соответствующих SDK



Оптимизации

- ▶ Для OpenCL применимы те же принципы «слияния» запросов в память
 - 32 рабочих элемента считывают непрерывный блок в глобальной памяти размером 128 байт одним запросом
 - Выгоднее использовать массивы размер которых кратен 16/32/64 байтам
- ▶ Как минимум 4 рабочих элемента на 1 Processing Element (больше – лучше)



Оптимизации

- ▶ По возможности избегать ветвлений, особенно в сочетании с синхронизацией
- ▶ Структура из массивов предпочтительнее массива структур
- ▶ Развертка многомерных матриц в одномерные массивы
- ▶ Код необходимо тестировать для подбора оптимальных параметров под CPU/GPU конкретных вендоров



Extrae/Paraver

- ▶ Extrae – позволяет построить трассировку вызовов и замерить время исполнения,
- ▶ Paraver – визуализация полученных трассировок
- ▶ <http://www.bsc.es/computer-sciences/performance-tools/trace-generation>
- ▶ <http://www.bsc.es/computer-sciences/performance-tools/paraver>



NVIDIA Visual Profiler

- ▶ Оптимизация OpenCL приложений для GPU Nvidia
- ▶ Позволяет измерять
 - Загруженность устройства
 - Пропускную способность памяти
 - Количество использованных регистров
 - Время потраченное на запуск ядер и операции с памятью
- ▶ Удобный мастер настройки



AMD® CodeXL

- ▶ Оптимизация и отладка OpenCL приложений для GPU AMD
- ▶ Дает информацию
 - Вызовы API и время исполнения ядер
 - Передача данных
 - Использование регистров
 - Использование локальной памяти
 - Подсказки о потенциальных помехах производительности



Отладка

- ▶ Начиная с версии OpenCL 1.2 можно вызывать `printf` из ядер для получения информации
- ▶ Для более старых версий OpenCL можно использовать строковый буфер как параметр
- ▶ Подключить Exceptions для C++
- ▶ Использовать обработчики ошибок и/или блоки `try/catch`
- ▶ Проверять индексы и диапазоны



Отладка gdb

- ▶ Можно проводить отладку для CPU приложений при помощи gdb
- ▶ При исполнении кода на GPU могут возникнуть ошибки, не отловленные gdb
- ▶ Полезен для отлова некорректного доступа в память
- ▶ Компиляция кода с ключом `-g`
- ▶ Для CPU Intel и AMD требуются различные инструкции



Отладка oclgrind

- ▶ Запускает OpenCL приложение в «песочнице»
- ▶ Позволяет отловить
 - Неправильный доступ в память
 - Условия гонки
 - Расхождения рабочих групп
 - Ошибки API



Отладка

- ▶ AMD CodeXL для APU, CPU и GPU от AMD
- ▶ NVIDIA Nsight для GPU Nvidia
- ▶ GPUVerify
- ▶ Visual Studio + Intel SDK for OpenCL



Boost.Compute

- ▶ Библиотека предоставляющая C++ интерфейс для работы с многоядерными CPU и GPU
- ▶ Основана на OpenCL
- ▶ <https://github.com/boostorg/compute>
- ▶ При компиляции кода необходимо указать
 - -I/путь к папке include/
- ▶ Требуемая версия Boost ≥ 1.54
- ▶ Включен в Boost начиная с версии 1.61



Пример 0

- ▶ Получение дефолтного устройства OpenCL



Boost.Compute

- ▶ Идеология применения библиотеки схожа с STL – операции над контейнерами данных
- ▶ В библиотеке реализовано несколько видов контейнеров для работы с данными на устройстве
- ▶ Список алгоритмов:
 - http://www.boost.org/doc/libs/1_62_0/libs/compute/doc/html/boost_compute/reference.html#boost_compute.reference.api_overview.algorithms



Пример 1

- ▶ Передача данных с хоста на устройство посредством Boost.Compute
- ▶ Обработка данных на устройстве через функцию `transform()`
- ▶ Передача обработанных данных с устройства на хост



Boost.Compute

- ▶ Библиотека предоставляет возможность создавать пользовательские функции, для передачи в алгоритмы типа `transform()` и `reduce()`
- ▶ Boost.Compute поддерживает лямбда-выражения для передачи в соответствующие алгоритмы в качестве параметра



Пример 2

- ▶ Пример использования собственных функций и лямбда-выражений для передачи в `transform()`



VexCL

- ▶ Шаблонная библиотека векторных выражений
- ▶ Предназначена для облегчения разработки кода с использованием GPU
- ▶ В качестве бэкенда используются OpenCL/CUDA
- ▶ Поддержка вычислений на нескольких устройствах и платформах



VexCL

- ▶ Библиотека предоставляет STL-подобный интерфейс для работы с контейнерами и алгоритмами с использованием параллельного программирования
- ▶ <https://github.com/ddemidov/vexcl>
- ▶ Для компиляции проекта необходимо
 - Указать папку с библиотеками Boost(флаг -L)
 - Слинковать его с boost_system и boost_filesystem



Пример 3

- ▶ Получения списка GPU поддерживающих вычисления в двойной точностью и включение их в текущий контекст



VexCL

- ▶ Основной фокус библиотеки – операции над векторами
 - Векторы должны иметь одинаковый размер
 - Должны размещаться в памяти одного и того же устройства
- ▶ Каждая операция над векторами приводит к запуску вычислительного ядра



Пример 4

- ▶ Сложение векторов
- ▶ Пример создания пользовательской функции
- ▶ Пример сортировки





Казанский федеральный
УНИВЕРСИТЕТ

ВЫСШАЯ ШКОЛА
информационных технологий
и информационных систем

Вопросы

ekhramch@kpfu.ru