# 2 слайд

Гетерогенные вычислительные системы (ГВС) - электронные системы, использующие различные типы вычислительных блоков. Вычислительными блоками такой системы могут быть процессор общего назначения (GPP), процессор специального назначения (например, цифровой сигнальный процессор (DSP) или графический процессор (GPU)), со-процессор, логика ускорения (специализированная интегральная схема (ASIC) или программируемая пользователем вентильная матрица (FPGA)). В общем, гетерогенная вычислительная платформа содержит процессоры с разными наборами команд (ISA).

Спрос на повышение гетерогенности в вычислительных системах, частично связан с необходимостью в высокопроизводительных, высокореакционных системах, которые взаимодействуют с другим окружением (аудио/видео системы, системы управления, сетевые приложения и т.д.). В прошлом, технологические достижения и масштабируемость частоты процессора позволяли большинству компьютерных приложений увеличивать производительность без структурных изменений или аппаратного ускорения. Хотя эти достижения сохраняются, их влияние на современные приложения не так значительно, как влияние некоторых препятствий, таких как стена памяти и стена мощности. Теперь, с этими дополнительными ограничениями, основным методом получения более производительных вычислительных систем является введение дополнительных специализированных ресурсов, в результате чего вычислительная система становится гетерогенной. Это позволяет разработчику использовать несколько типов вычислительных элементов, каждый из которых способен выполнять задачи, которые лучше всего для него подходят. Добавление дополнительных, независимых вычислительных ресурсов неизбежно приводит к тому, что большинство гетерогенных систем рассматриваются как параллельные вычислительные системы или многоядерные системы. Ещё один термин, который иногда используется для этого типа вычислений «гибридные вычисления». Hybrid-core computing - форма гетерогенных вычислений, в которой асимметричные вычислительные устройства сосуществуют в одном процессоре.

# 3 слайд

Актуальность данной темы обусловлена повышением сложности и объемов вычислений решаемых с использованием вычислительных систем задач. С помощью гетерогенных вычислительных систем могут быть решены многие задачи физики, решение которых без использования ГВС заняло бы очень много времени, либо вообще было бы невозможно.

# 4 слайд

1. Использовать технологию OpenCL для реализации гетерогенной вычислительной системы CPU+GPU.

2. Применить построенную ГВС в алгоритме фильтрации изображений.

# 5 слайд

Для реализации гетерогенной системы, и решения поставленных задач была выбрана технология OpenCL, которая позволяет один и тот же код выполнять на всех, имеющихся в архитектуре компьютера процессорах (единовременно, либо по очереди).

OpenCL одна из технологий, позволяющая создать гетерогенную вычислительную систему, состоящую из различных процессоров, например CPU+GPU. Эта технология была использована в работе, для разработки ГВС и решения трудоемкой задачи.

Ключевыми отличиями используемого языка от Си (стандарт ISO 1999 года) являются:

* Отсутствие поддержки указателей на функции, рекурсии, битовых полей, массивов переменной длины (VLA), стандартных заголовочных файлов.
* Расширения языка для параллелизма: векторные типы, синхронизация, функции для Work-items/Work-Groups.
* Квалификаторы типов памяти: \_\_global, \_\_local, \_\_constant, \_\_private.
* Иной набор встроенных функций.

OpenCL новый стандарт для разработки приложений для гетерогенных систем. Изначально OpenCL задумывался как нечто большее: единый стандарт для написания приложений, которые должны исполняться в системе, где установлены различные по архитектуре процессоры, ускорители и платы расширения.

OpenCL предоставляет программисту низкоуровневый API, через который он взаимодействует с ресурсами устройства. OpenCL API может либо напрямую поддерживаться устройством, либо работать через промежуточный API (как в случае NVidia: OpenCL работает через CUDA Driver API, поддерживаемый устройствами), это зависит от конкретной реализации и не описывается стандартом.

# 6-7 слайд

Для описания основной идеи OpenCL используется иерархия из 4х моделей:

* Модель платформы (Platform Model);
* Модель памяти (Memory Model);
* Модель исполнения (Execution Model);
* Программная модель (Programming Model)

Платформа OpenCL состоит из хоста, соединенного с устройствами, поддерживающими OpenCL. Каждое OpenCL-устройство состоит из вычислительных блоков (Compute Unit), которые далее разделяются на один или более элементов-обработчиков рис.1 (Processing Elements, далее PE).

OpenCL-приложение исполняется на хосте в соответствии с нативными моделями его платформы. OpenCL-приложение отправляет с хоста команды устройствам на выполнение вычислений на PE. PE в рамках вычислительного блока выполняют один поток команд как SIMD блоки (одна инструкция выполняется всеми одновременно, обработка следующей инструкции не начнется, пока все PE не завершат исполнение текущей инструкции), либо как SPMD блоки (у каждого PE собственный счетчик инструкций (program counter)).

То есть OpenCL обрабатывает некие команды, поступающие от хоста. Таким образом приложение не связано жестко с OpenCL, а значит всегда можно подменить реализацию OpenCL, не нарушив работоспособность программы. Даже если будет создано такое устройство, которое не укладывается в модель «OpenCL-устройства», для него можно будет создать реализацию OpenCL, транслирующую команды хоста в более удобный для устройства вид.

Выполение OpenCL-программы состоит из двух частей: хостовая часть программы и kernels (ядра) исполняющиеся на OpenCL-устройстве (рис. 1). Хостовая часть программы определяет контекст, в котором исполняются kernel'ы, и управляет их исполнением.

Основная часть модели исполнения OpenCL описывает исполнение kernel’ов. Когда kernel ставится в очередь на исполнение, определяется пространство индексов (NDRange). Копия kernel'а выполнятся для каждого индекса из этого пространства. Kernel, выполняющийся для конкретного индекса, называется «Work-Item» (рабочей единицей), и определяется точкой в пространстве индексов, то есть каждой «единице» предоставляется глобальный ID (рис. 2). Каждый Work-Item выполняет один и тот же код, но конкретный путь исполнения (ветвления и т.п.) и данные, с которыми он работает, могут быть различными. Work-Item'ы организуются в группы (Work-Groups). Группы предоставляют более крупное разбиение в пространстве индексов. Каждой группе приписывается групповой ID с такой же размерностью, которая использовалась для адресации отдельных элементов. Каждому элементу сопоставляется уникальный, в рамках группы, локальный ID. Таким образом, Work-Item'ы могут быть адресованы как по глобальному ID, так и по комбинации группового и локального ID. Work-Item'ы в группе исполняются параллельно на PE одного вычислительного блока.

Work-Item, исполняющий kernel может использовать четыре различных типа памяти (рис. 3):

* **Глобальная память.** Эта память предоставляет доступ на чтение и запись элементам всех групп. Каждый Work-Item может писать и читать из любой части объекта памяти.
* Запись и чтение глобальной памяти может кэшироваться в зависимости от возможностей устройства.
* **Константная память.** Область глобальной памяти, которая остается постоянной во время исполнения kernel'а. Хост аллоцирует и инициализирует объекты памяти, расположенные в константной памяти.
* **Локальная память.** Область памяти, локальная для группы. Эта область памяти может использоваться, чтобы создавать переменные, разделяемые всей группой. Она может быть реализована как отдельная память на OpenCL-устройстве. Альтернативно эта память может быть размечена как область в глобальной памяти.
* **Частная (private) память.** Область памяти, принадлежащая Work-Item. Переменные, определенные в частной памяти одного Work-Item’а, не видны другим.

Модель исполнения OpenCL поддерживает две программные модели: параллелизм данных (Data Parallel) и параллелизм заданий (Task Parallel), так же поддерживаются гибридные модели. Основная модель, определяющая дизайн OpenCL, - параллелизм данных.

**Программная модель с параллелизмом данных.**

Эта модель определяет вычисления как последовательность инструкций, применяемых к множеству элементов объекта памяти. Пространство индексов, ассоциированное с моделью исполнения OpenCL, определяет Work-Item'ы и то, как данные распределяются между ними. В строгой модели параллелизма данных существует строгое соответствие один к одному между Work-Item и элементом в объекте памяти, с которым kernel может работать параллельно. OpenCL реализует более мягкую модель параллелизма данных, где строгое соответствие один к одному не требуется.

OpenCL предоставляет иерархическую модель параллелизма данных. Существует два способа определить иерархическое деление. В явной модели программист определяет общее число элементов, которые должны исполняться параллельно и так же каким образом эти элементы будут распределены по группам. В неявной модели программист только определяет общее число элементов, которые должны исполняться параллельно, а разделение по рабочим группам выполняется автоматически.

**Программная модель с параллелизмом заданий.**

В этой модели каждая копия kernel'а исполняется независимо от какого-либо пространства индексов. Логически это эквивалентно исполнению kernel'а на вычислительном блоке (CU) с группой, состоящей из одного элемента. В такой модели пользователи выражают параллелизм следующими способами:

* используют векторные типы данных, реализованные в устройстве;
* устанавливают в очередь множество заданий;
* устанавливают в очередь нативные kernel'ы, использующие программную модель, ортогональную к OpenCL.

Существование двух моделей программирования - также дань универсальности. Для современных GPU и Cell хорошо подходит первая модель. Но не все алгоритмы можно эффективно реализовать в рамках такой модели, а также есть вероятность появления устройства, архитектура которого будет неудобна для использования первой модели. В таком случае вторая модель позволяет писать специфичные для другой архитектуры приложения.

В итоге модель OpenCL получилась весьма универсальной, при этом она остается низкоуровневой, позволяя оптимизировать приложения под конкретную архитектуру. Так же она обеспечивает кроссплатформенность при переходе от одного типа OpenCL-устройств к другому. Поставщик реализации OpenCL имеет возможность всячески оптимизировать взаимодействие своего устройства с OpenCL API, добиваясь повышения эффективности распределения ресурсов устройства. Кроме того, правильно написанное OpenCL приложение будет оставаться эффективным при смене поколений устройств.

# 8 слайд

Для каждого пикселя берем окно размером “Глубина фильтрации”\*“Глубина фильтрации”. В стандартном изображении каждый пиксель кодируется тремя цветами RGB. Для качественной фильтрации шумов необходимо фильтровать пиксели по всем трем цветам. Т.е. в действительности нам необходимо провести медианную фильтрацию 3 раза для каждого пикселя.

1. Сначала берем окно для красной компоненты.
2. Расставляем все интенсивности по возрастанию.
3. Величина интенсивности красной компоненты обрабатываемого пикселя будет равна центральному элементу отсортированного окна.
4. Повторяем эту процедуру для зеленого и синего цвета

Схема фильтрации представлена на рис.7.

После этого мы получаем готовый пиксель. Для сортировки был использован алгоритм сортировки выбором, который достаточно быстрый и простой в реализации, что не маловажно, т.к. графические устройства не поддерживают рекурсию и динамическое выделение памяти.

# 9 слайд

Была написана программа фильтрации изображения медианным алгоритмом (рис. 6). Алгоритм был реализован с использованием технологии OpenCL, а также с использованием стандартных средств C++.

# 10-12 слайд

Для демонстрации целесообразности использования данной технологии сравним скорость выполнения одного и того же алгоритма с использованием OpenCL на разных платформах/устройствах и с использованием стандартных средств C++.

Конфигурация компьютера, на котором выполнялось сравнение:

* CPU Intel Core i5-2450M
* GPU AMD Radeon HD 7400M
* ОЗУ 6Гб

Результаты фильтрации изображения размером 584x329 px с разным уровнем зашумленности представлены в диаграммах 1,2.

# 13-15 слайд

Результаты фильтрации изображения размером 1280x697 px с разным уровнем зашумленности представлены в диаграммах 3,4.

Как видно из таблиц и диаграмм, фильтрация изображения с использованием технологии OpenCL проходит до 5-6 раз быстрее, чем с использованием линейных алгоритмов С++.

# 16 слайд

В данной работе была написана программа, использующая технологию OpenCL для реализации гетерогенной вычислительной системы CPU+GPU. Данная гетерогенная вычислительная система была применена для фильтрации изображения с помощью медианного алгоритма. Результаты работы программы были проанализированы для различных уровней зашумленности изображения, а также для изображений различных размеров. Исходя из полученный результатов, следует, что построенная гетерогенная система выполняет работу по фильтрации изображения до 6 раз быстрее, чем если бы мы использовали стандартные средства С++.