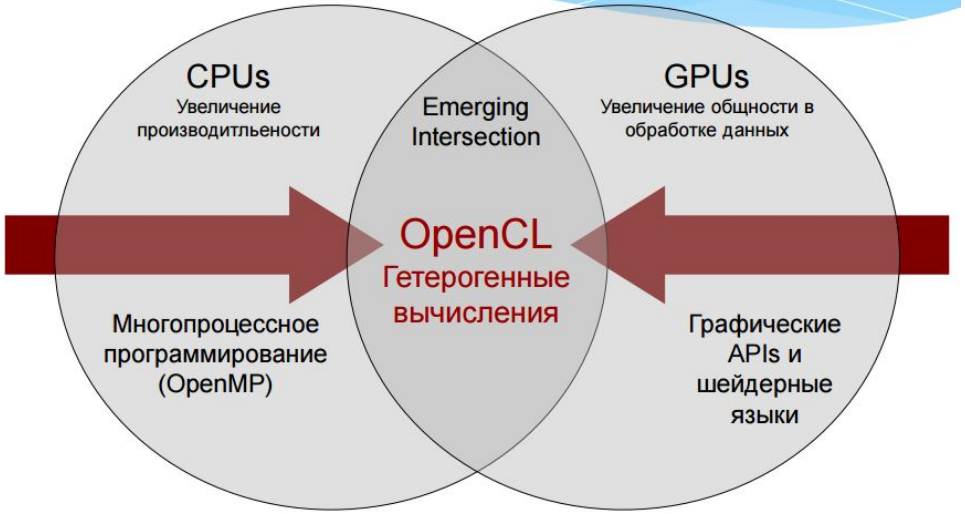


Параллельное и распределенное программирование.

Технология программирования гетерогенных систем OpenCL.

Лекция 1

Балльно-рейтинговая система

	4 курс		5 курс	
	Кол-во	Балл	Кол-во	Балл
Лабораторные работы	6+6	5-10	4	5-10
Лекции	4+4	0,5	8	0,5
Контрольные работы	1+1	5-10	2	7-14
Активность	-	6	-	6
Экзамен/диф. зачет	1+1	10-22	1	10-22

Технологии программирования

Разные ускорители

- Разные языки и библиотеки
  - NVidia – CUDA
  - AMD – Brook+
  - CELL BE – SPU-C
  - x86 – C/C++/Fortran + OpenMP
- Непереносимость
  - Код
  - Навыки программиста

Цели OpenCL

- Стандарт программирования
  - Многоядерные процессоры
  - Ускорители, GPU
  - Мобильные медиапроцессоры
- Стандарт функциональности
  - Производителям процессоров

Назначение OpenCL

OpenCL (Open Computing Language) – открытый стандарт параллельного программирования для гетерогенных платформ, включающих центральные, графические процессоры и другие дискретные вычислительные устройства.

Компоненты:

- библиотечные функции (API) для управления параллельными вычислениями на устройствах со стороны центрального процессора;
- языки программирования для реализации вычислений;
- система времени выполнения для поддержки разработки.

Рабочая группа OpenCL

Рабочая группа OpenCL

Что такое OpenCL

- Открытая спецификация
- Спецификация, разрабатываемая мировыми лидерами в области разработки и производства вычислительных устройств
- Спецификация, поддерживаемая Khronos Group

План лекции

- Введение в параллельные вычисления
- Параллельные вычисления на GPU
- Программная модель вычисления на GPU
- Аппаратная модель вычисления на GPU
- Проблемы реализации параллельных вычислений

Параллелизм

- Параллелизм описывает возможность одновременного выполнения нескольких частей задачи

- Чтобы использовать параллелизм, мы должны иметь физические ресурсы (например, оборудование) для работы по нескольким причинам одновременно

Параллелизм

- Закон Амдаля : максимальное теоретическое ускорение, которое мы можем добиться с использованием параллелизма в задаче, пропорционально отношению доли последовательных вычислений к параллельным частям и числу процессоров, которые мы имеем

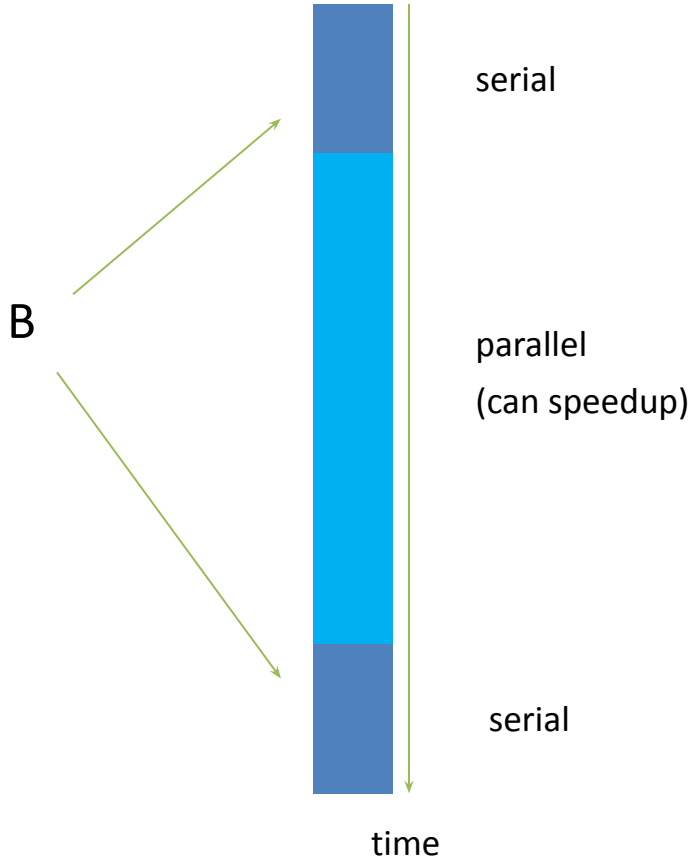
$$S = \frac{1}{B + \left(\frac{1}{n}\right)(1 - B)}$$

S = ускорение

B = доля последовательных вычислений

n = количество процессоров

- If an algorithm is 95% parallel (B = .05), then with a large enough n, we can approach a 20X speedup



Параллелизм

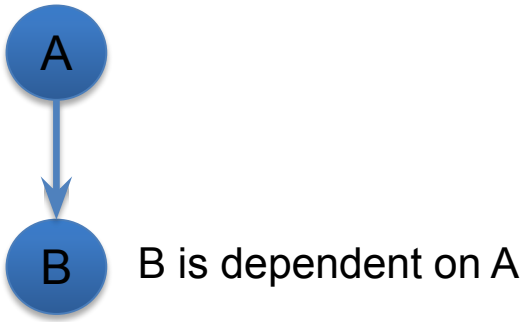
- Для традиционных архитектур процессоров часто говорим о параллелизме на уровне инструкций (ILP)
  - Высокопроизводительные процессоры часто имеют логику большого объема, предназначенную для суперскалярного и нестандартного оборудования для использования ILP
- Для GPU-вычислений с OpenCL используются другие типы параллелизма:
  - Параллелизм задач - возможность одновременного выполнения различных задач в рамках задачи
  - Параллелизм данных - возможность выполнять части одной задачи (т. е. Разные данные) одновременно
- В OpenCL мы увидим, что задачи часто могут соответствовать разным ядрам, а параллелизм данных используется несколькими программными потоками в ядрах

Декомпозиция

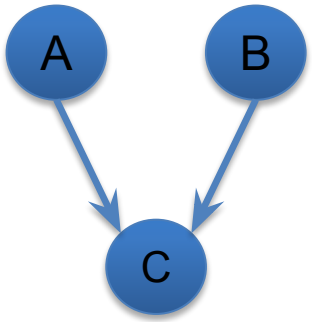
- Для нетривиальных задач декомпозиция помогает иметь более формальные понятия для определения параллелизма
- Когда мы думаем о том, как распараллелить программу, мы используем понятия декомпозиции:
  - Декомпозиция задачи: деление алгоритма на отдельные задачи (не фокусироваться на данных)
  - Декомпозиция данных: разделение набора данных на отдельные фрагменты, которые могут обрабатываться параллельно

Декомпозиция задачи

- Декомпозиция задачи сводит алгоритм к функционально независимым частям
- Задачи могут иметь зависимости от других задач
  - Если ввод задачи B зависит от выхода задачи A, то задача B зависит от задачи A
  - Задачи, которые не имеют зависимостей (или чьи зависимости завершены), могут быть выполнены в любое время для достижения параллелизма
  - Графики зависимостей задач используются для описания взаимосвязи между задачами



B is dependent on A

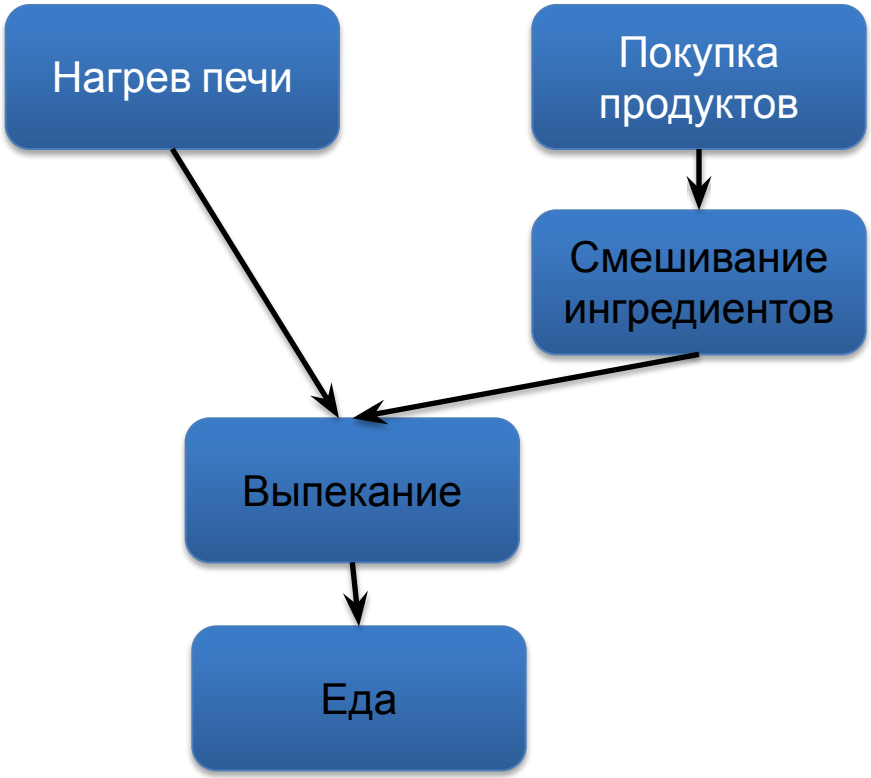


A and B are independent of each other

C is dependent on A and B

Граф зависимости задач

- Мы можем создать простой график зависимостей задачи для выпечки
- Любые задачи, которые не связаны через график, могут выполняться параллельно (например, предварительный нагрев печи и покупки продуктов)



Декомпозиция данных

- Декомпозиция данных - это способ разбить работу на несколько независимых задач, но каждая задача несет ту же ответственность
  - Каждая задача может рассматриваться как обработка части данных
- Использование декомпозиции данных позволяет использовать параллелизм данных
- Используя OpenCL, декомпозиция данных позволяет сопоставлять параллельную обработку данных с параллельным оборудованием данных

Декомпозиция выходных данных

- Для большинства научных и инженерных приложений декомпозиция выполняется на основе выходных данных
- Примеры декомпозиции выходных данных:
  - Каждый выходной пиксель свертки изображения получается путем применения фильтра к области входных пикселей
  - Каждый выходной элемент матричного умножения получается путем умножения строки на столбец входных матриц
- Этот метод действителен в любое время, когда алгоритм основан на взаимно однозначных или многозначных функциях

Декомпозиция выходных данных

(пример)

- Фильтр Box
  - выполняет операции над каждым пикселем независимо
  - результаты помещаются в независимых ячейки
  - Большое количество ВУ приведет у существенному увеличению производительности

Декомпозиция входных данных

- Разделение входных данных аналогично, за исключением алгоритмов, которые являются функцией «один ко многим»
- Примеры входного разложения
  - Гистограмма создается путем помещения каждого входного сигнала в одно из фиксированного количества делений
  - Функция поиска может принимать строку как входную информацию и искать появление различных подстрок
- Для этих типов приложений каждый поток создает «частичный счет» вывода, а для вычисления конечного результата требуются синхронизация, атомные операции или другие задачи

Параллельные вычисления

- Декомпозиция основана исключительно на алгоритме

- Однако при реализации параллельного алгоритма необходимо учитывать как аппаратные, так и программные соображения

Параллельные вычисления

- Существуют аппаратные и программные подходы к параллелизму
- Большая часть 90-х годов была потрачена на то, чтобы заставить CPU автоматически использовать Параллельность Уровня Инструкции (ILP)
  - Несколько команд (без зависимостей) выдаются и выполняются параллельно
  - Автоматическая аппаратная распараллеливание не будет рассматриваться в этом курсе
- Более высокий уровень параллелизма (например, потоковой передачи) не может выполняться автоматически, поэтому программные конструкции, вставленные программистами или компиляторами, указывают аппаратному обеспечению, в котором существует параллелизм
- При параллельном программировании программист должен выбрать модель программирования и параллельное вычислительное устройство (ВУ), которые подходят для решения задачи

Параллелизм на уровне вычислительных устройств

- Аппаратное обеспечение обычно предназначено для определенного типа параллелизма

Hardware type	Examples	Parallelism
Multi-core superscalar processors	Phenom II CPU	Task
Vector or SIMD processors	SSE units (x86 CPUs)	Data
Multi-core SIMD processors	Radeon R9 290X GPU	Data

- В настоящее время графические процессоры состоят из множества независимых «процессоров», которые имеют элементы обработки SIMD
  - Одновременно на GPU запускается одна задача *
  - «Разворачивание» цикла (следующий слайд) используется для разделения параллельной задачи данных между независимыми процессорами
  - Каждая инструкция должна быть параллельна данным, чтобы в полной мере использовать преимущества SIMD-оборудования GPU

*если одновременно выполняются несколько задач, невозможно взаимодействие

Извлечение параллелизма: «разворачивание» цикла

- Loop strip mining (разворачивание цикла) - это метод преобразования цикла, который разбивает итерации цикла так, что может быть несколько итераций:
  - выполняются одновременно (векторные / SIMD-единицы);
  - разделяются между различными процессорами (многоядерные процессоры);
  - или выполняется оба условия (графические процессоры).
- Пример с «разворачиванием» цикла показан на следующих слайдах

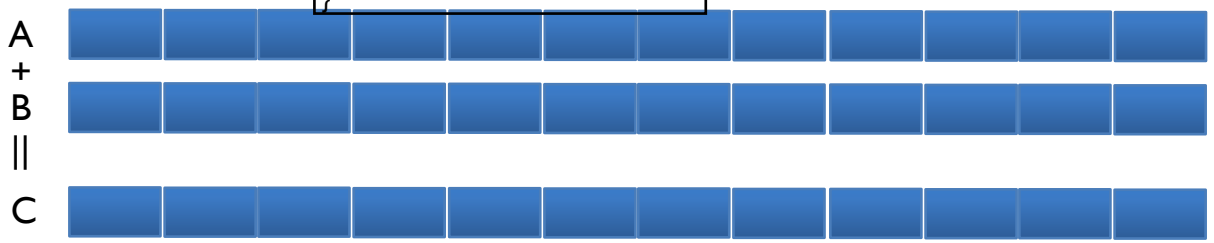
Параллелизм на уровне программы – SPMD

- Программы GPU называются «ядрами» и записываются с использованием модели программирования Single Program Multiple Data (SPMD)
  - SPMD выполняет несколько экземпляров одной и той же программы независимо, где каждая программа работает с другой частью данных
- Для параллельных данных и инженерных приложений объединение SPMD с «разворачиванием» цикла является очень распространенным методом параллельного программирования
  - Интерфейс передачи сообщений (MPI) используется для запуска SPMD на распределенном кластере
  - Потоки POSIX (pthreads) используются для запуска SPMD в системе с разделяемой памятью
  - «Ядра» запускают SPMD в графическом процессоре

Параллелизм на уровне программы – SPMD

- Consider the following vector addition example

```
for( i = 0:11 ) {
   C[ i ] = A[ i ] + B[ i ]
}
```



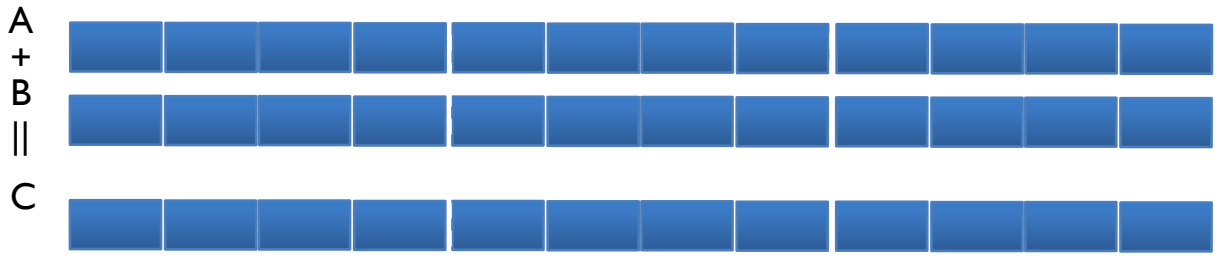
Serial program:
one program completes the entire task

- Combining SPMD with loop strip mining allows multiple copies of the same program execute on different data in parallel

```
for( i = 0:3 ) {
   C[ i ] = A[ i ] + B[ i ]
}
```

```
for( i = 4:7 ) {
   C[ i ] = A[ i ] + B[ i ]
}
```

```
for( i = 8:11 ) {
   C[ i ] = A[ i ] + B[ i ]
}
```



SPMD program:
multiple copies of the same program run on different chunks of the data

Параллелизм на уровне программы – SPMD

- В примере добавления векторов каждый фрагмент данных может быть выполнен как независимый поток
- На современных процессорах накладные расходы на создание потоков настолько высоки, что куски должны быть большими
  - На практике обычно несколько потоков (примерно столько же, сколько количество ядер процессора), и каждому из них предоставляется большой объем работы
- При программировании графических процессоров создание потоков требует небольших накладных расходов, поэтому мы можем создать один поток для каждой итерации цикла

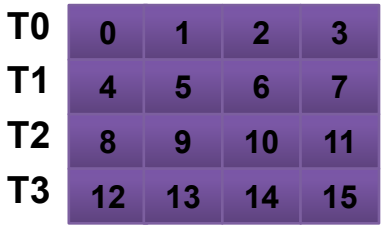
Параллелизм на уровне программы – SPMD

Single-threaded (CPU)

```
// there are N
elements
for(i = 0; i < N; i++)
  C[i] = A[i] + B[i]
```

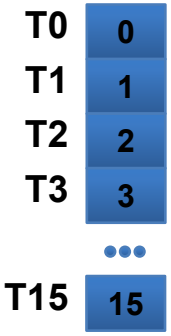
Multi-threaded (CPU)

```
// tid is the thread id
// P is the number of cores
for(i = tid*(N/P); i < (tid+1)*N/P;
i++)
  C[i] = A[i] + B[i]
```



Massively Multi-threaded (GPU)

```
// tid is the thread id
C[tid] = A[tid] +
B[tid]
```

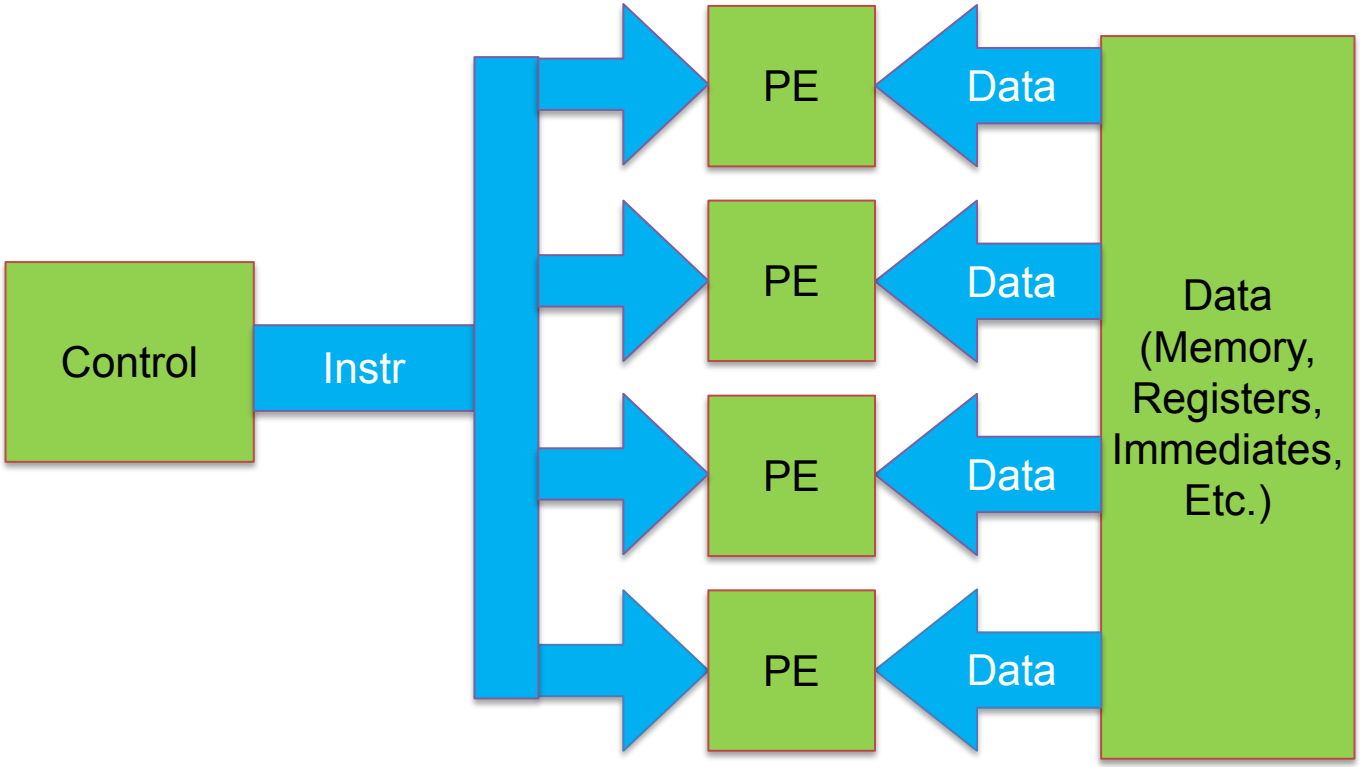


Параллелизм на уровне ВУ – SIMD

- Каждый обрабатывающий элемент ВУ выполняет одну и ту же инструкцию с разными данными одновременно
  - Одна команда выдается для одновременного выполнения на многих блоках ALU
  - Мы говорим, что количество блоков ALU - это ширина SIMD-блока
- SIMD-процессоры эффективны для параллельных алгоритмов данных
  - Они уменьшают количество управляющих потоков в пользу аппаратного обеспечения ALU

Параллелизм на уровне ВУ – SIMD

- A SIMD hardware unit



Параллелизм на уровне ВУ – SIMD

- В примере сложения вектора SIMD-блок с шириной в четыре может одновременно выполнить четыре итерации цикла
- Все текущие графические процессоры основаны на оборудовании SIMD
  - аппаратное обеспечение GPU неявно отображает каждый поток SPMD в SIMD-ядро,
  - модель работы потоков на SIMD-оборудовании часто называется Single Instruction Multiple Threads (SIMT)

Проблемы реализации параллельных вычислений

- На процессорах аппаратные атомарные операции обеспечивают параллелизм
  - Атомарные операции позволяют считывать и записывать данные без вмешательства из другого потока

- Графические процессоры поддерживают некоторые общесистемные атомарные операции, но с большим компромиссом производительности
  - Обычно код, требующий глобальной синхронизации, плохо подходит для графических процессоров (или должен быть реструктурирован)
  - Любая проблема, которая решается с использованием разбиения входных данных (т. е. требует, чтобы результаты были объединены в конце), скорее всего потребует провести реструктуризацию, чтобы хорошо работать на графическом процессоре

Почему OpenCL

- Производительность зависит не от частоты процессора, а от уровня и типа параллелизма задачи

- Гетерогенное программирование: использование ресурсов CPU и GPU

- Единый интерфейс для множества устройств

- поддержка параллельного вычисления задач **общего назначения**
- Портируемая
- Низкий барьер для начала работы

OpenCL совместимые устройства

- Большинство CPU и GPU
- Следующие типы устройств могут быть совместимы:
  - мультимедийные чипы;
  - FPGA;
  - встраиваемые процессоры.

Назначение OpenCL

- Простая модель вычисления – чистый API
- Поддержка ANSI-C99
- Дополнительные спецификаторы, встроенные типы данных и функции
- Управление потоками
- Синхронизация на уровне потоков и приложений
- Уменьшение ошибок в математических функций
- Стандарт для разработчиков аппаратного обеспечения

Где OpenCL может использоваться

- Обработка видео, аудио – информации и изображений
- Научно-исследовательские вычисления
- Медицинские расчеты
- Финансовые модели
- Др.

Средства параллельного программирования

	ВУ с общей памятью	ВУ с распределенной памятью	Гетерогенные системы (CPU + GPU)
Системные средства	threads	процессы + сокеты	-
Языки	C/C++, Fortran 95	HPF, DVM, mpC, Charm ++, Occam	
Специальные библиотеки и пакеты	OpenMP	MPI PVM	OpenCL, CUDA

Краткая история OpenCL

Таблица - Основные этапы развития OpenCL

Год	Событие	Возможности
18 ноября 2008	OpenCL 1.0	Поддержка открытого GPGPU-стандарта OpenCL
14 июля 2010	OpenCL 1.1	– новые типы данных (3-d векторы, изображения); – команды из нескольких потоков хоста и обработки буфера между несколькими устройствами; – улучшенное взаимодействие с OpenGL за счет эффективного обмена изображениями.
15 ноября 2011	OpenCL 1.2	- партицирование устройств; - раздельная компиляция и связывание объектов; - расширенная поддержка изображений; - встроенные OpenCL-ядра; - улучшенное взаимодействие OpenCL и API DirectX 9/11.
22 июля 2013	OpenCL 2.0	- общая виртуальная память; - вложенный параллелизм; - универсальное адресное пространство; - атомарные операции C11 со стороны устройства; - каналы.
3 марта 2015	OpenCL 2.1	Поддержка C++14

OpenCL и OpenGL

- OpenCL и OpenGL совместно работают хорошо
- Основное вычисление выполняется с помощью OpenCL и отображаются результаты с помощью OpenGL

Когда не следует применять OpenCL

- Последовательные задачи
- Вычисления с зависимостью по данным
- Вычисления, которые подразумевают значительное количество обмена промежуточными результатами
- Вычисления, зависимые от устройства

Почему вычисления на GPU?

- Высокая производительность на операциях с плавающей точкой;
- Спроектирован для высоко масштабируемого параллелизма;
- Скорость увеличения производительности GPU несколько выше CPU;
- Пропускная способность GPU (внутренние шины, память- АЛУ) намного выше, чем в CPU

Ограничения CPU

- Системы на GPU ограничены
- На GPU «тяжело» реализуются аналитические алгоритмы
- GPU тяжело отлаживать
- GPU требуется определенный способ организации данных для достижения требуемой производительности

OpenCL Demo

Выводы

- Выбор подходящих параллельных аппаратных и программных моделей сильно зависит от проблемы, которую мы пытаемся решить
  - Проблемы, которые соответствуют модели декомпозиции выходных данных, обычно довольно легко сопоставляются с аппаратно-параллельным оборудованием

- Наивно полагать, что модель параллельного программирования OpenCL проста, поскольку упрощено программирование SPMD
  - Мы часто можем сопоставлять итерации цикла for-loop непосредственно с рабочими элементами OpenCL
  - Однако, получение высокой производительности требует глубокого понимания аппаратного обеспечения (включая аппаратный параллелизм + подсистему памяти) и усложняет модель программирования