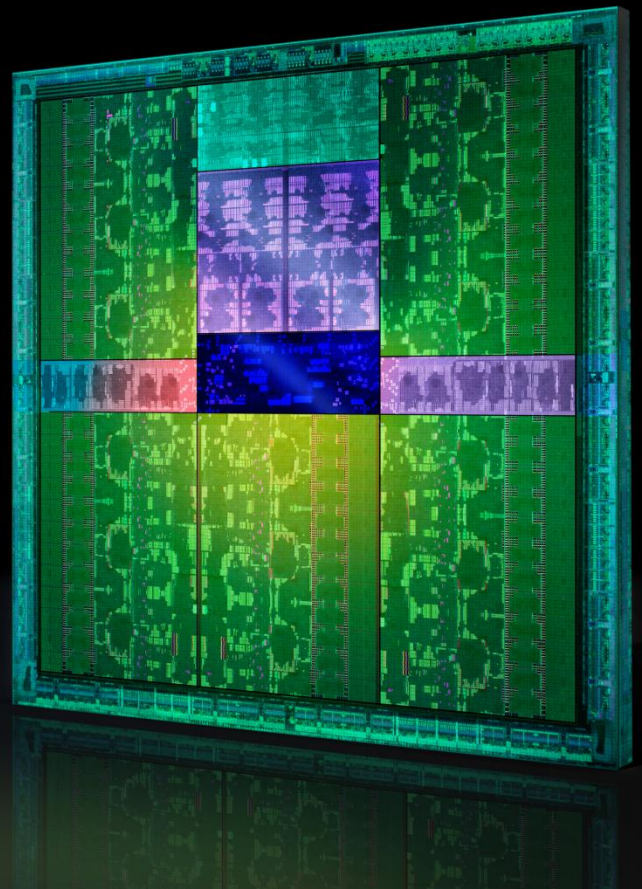

Архитектура Кеплер
Максим Милаков
NVIDIA Corporation
NVIDIA

Представляем GK110 GPU

Производительность

Энергоэффективность

Эффективность программирования

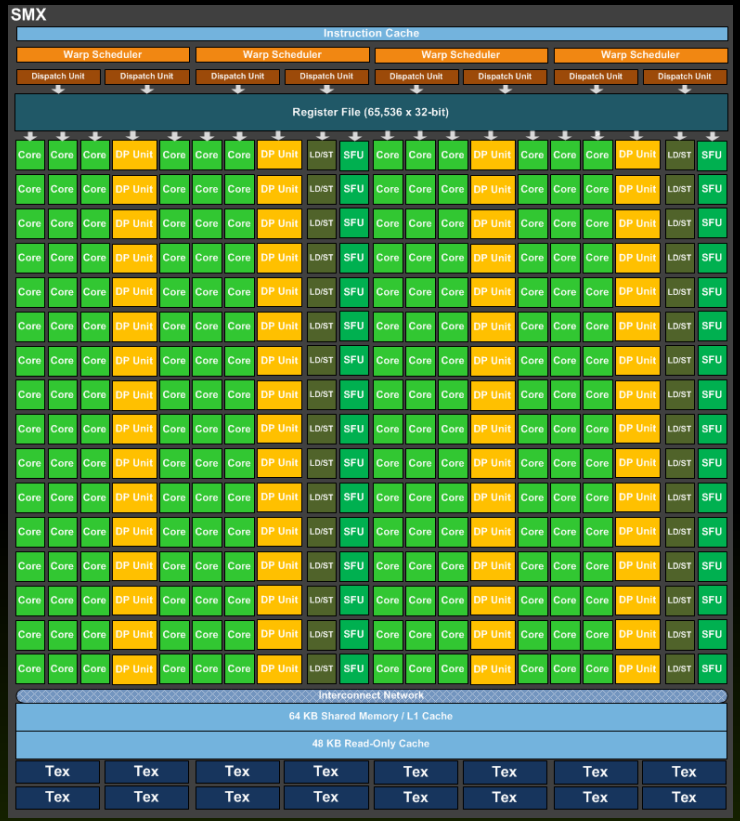


Кеплер GK110 блочная диаграмма

Архитектура

- 7.1 миллиардов транзисторов
- 15 SMX мультипроцессоров
- > 1 TFLOP FP64
- 1.5 MB L2 Cache
- 384-bit GDDR5
- PCI Express Gen3

Kepler GK110 SMX vs Fermi SM

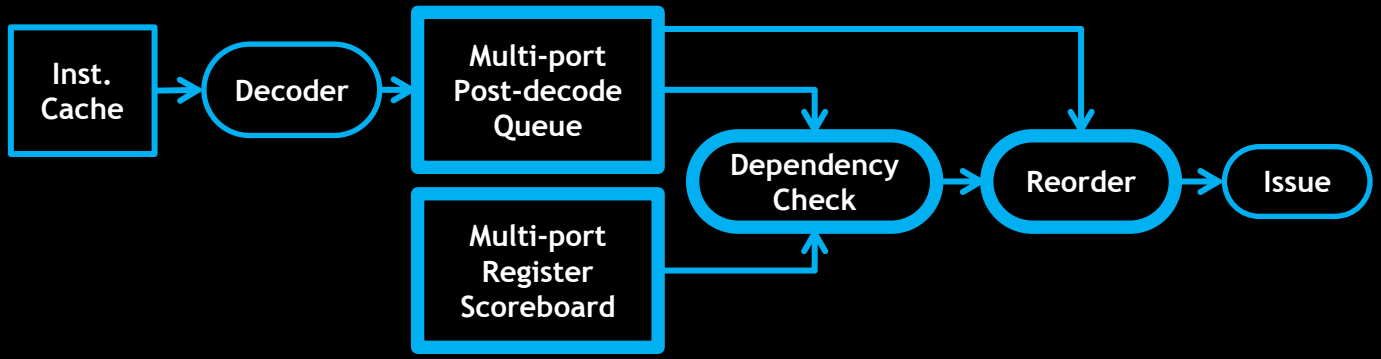


SMX: Еффективная производительность

- Энергоэффективная Архитектура
- Частоты и Новый Техпроцесс
  - Нет удвоения по частоте
  - 28 нм
- Результат:
  - Выросшая производительность
  - Меньшее энергопотребление

Изменения в планировщике/диспетчере

Ферми:
Аппаратная
проверка
зависимостей



Кеплер:
Инструкции
содержат
информацию
для планировщика

Баланс ресурсов в SMX

Ресурс	Кеплер GK110 vs Ферми GF110
Производительность операций с плавающей точкой	**2-3x**
Максимальное количество блоков на один SM/SMX	**2x**
Максимальное количество нитей на один SM/SMX	**1.3x**
Пропускная способность регистрового памяти	**2x**
Размер регистрового памяти	**2x**
Пропускная способность разделяемой памяти	**2x**
Размер разделяемой памяти	**1x**

Разделяемая память

- Пропускная способность разделяемой памяти 2x только в случае чтения/записи 64-битных данных
- Выровненный доступ - нет конфликтов банков
- Доступ невыровненный - структура конфликтов банков зависит от режима адресации разделяемой памяти: cudaFuncSetSharedMemConfig
  - cudaFuncMemBankSizeDefault - по умолчанию
  - cudaFuncMemBankSizeFourByte - последовательные 32-битные значения попадают в различные банки
  - cudaFuncMemBankSizeEightByte - последовательные 64-битные значения попадают в различные банки

Новая кодировка инструкций: 255 регистров на нить

- Ферми: До 63 регистров на нить
  - Часто ограничивает производительность на Ферми
  - Приводит к избыточному вытеснению регистров в локальную память (спиллингу)
- Кеплер: До 255 регистров на нить
  - Особенно важно для приложений с двойной точностью
- Пример: QUDA QCD fp64 код до 5.3x быстрее
  - Спиллинг исчез с возможностью использовать больше регистров

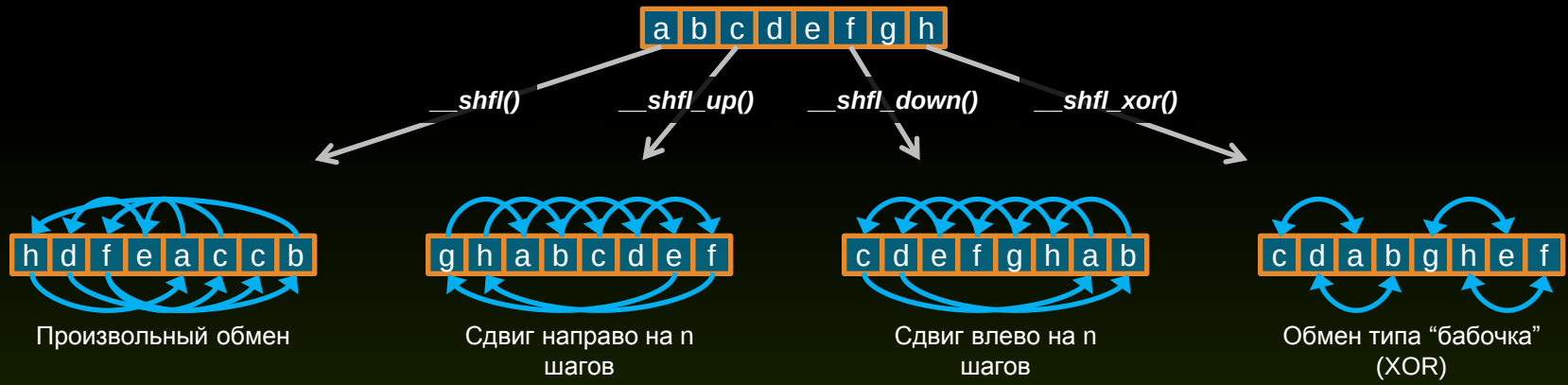
Новые высокопроизводительные инструкции

- SHFL (shuffle) - Обмен данными внутри варпа
- ATOM - Новая функциональность, быстрее
- Read-only кэш
- SHF (Funnel Shift) - 64-битовый сдвиг, циклический сдвиг
- FP32 деление - Ускорения точного деления
- SIMD видео инструкции

SHFL

Обмен данными внутри варпа

- Разделяемая память не используется
- Каждый поток отдает и получает одно 32-битовое значение
- 4 варианта:

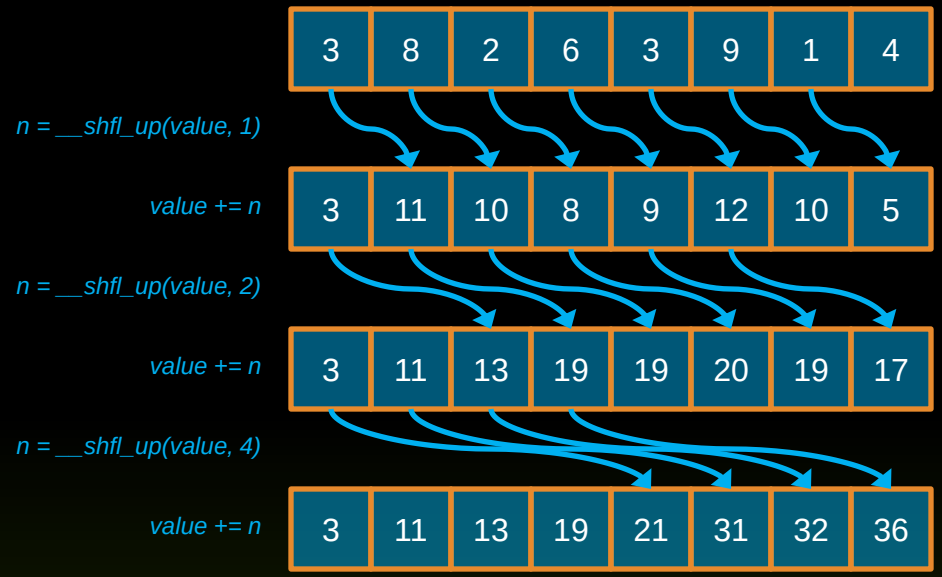


Пример использования SHFL: Префиксная сумма

```
__global__ void shfl_prefix_sum(int *data)
{
    int id = threadIdx.x;
    int value = data[id];
    int lane_id = threadIdx.x & warpSize;

    // Now accumulate in log2(32) steps
    for(int i=1; i<=width; i*=2) {
          int n = __shfl_up(value, i);
          if(lane_id >= i)
                    value += n;
    }

    // Write out our result
    data[id] = value;
}
```



Атомарные операции - новые возможности

- Добавлены int64 функции

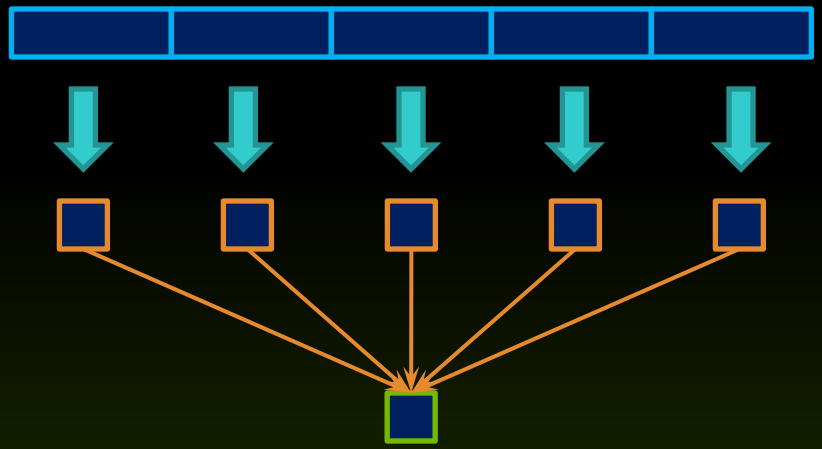
Операция	int32	int64
add	x	x
cas	x	x
exch	x	x
min/max	x	X
and/or/xor	x	X

- 2 – 10x ускорение
  - Стал короче конвейер обработки
  - Больше блоков, выполняющий атомарные операции
  - Самая медленная – 10x
  - Самая быстрая - 2x

Быстрые атомарные операции – новые возможности

Атомарные операции теперь настолько быстры, что их можно использоваться в циклах

- **Пример: Редукция данных (сумма всех значений)**



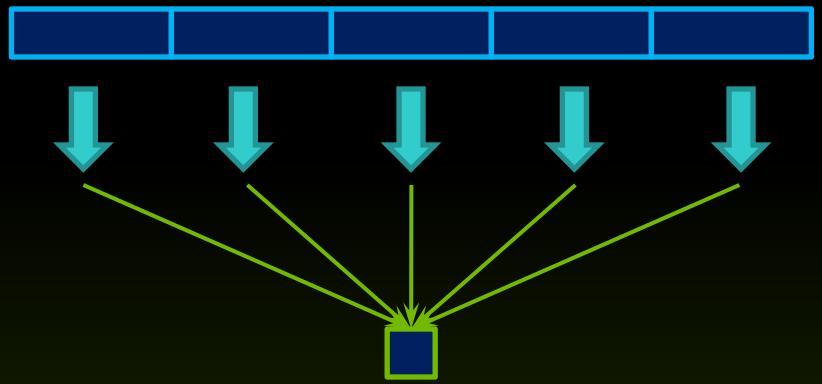
Без атомарных операций

1. Поделить входной массив на N частей
2. Запустить N блоков, каждый выполняет редукцию для своей части
3. Записать N значений в глобальную память
4. Запустить N потоков, редуцировать результат в единственное значение

Быстрые атомарные операции – новые возможности

Атомарные операции теперь настолько быстры, что их можно использоваться в циклах

- **Пример: Редукция данных (сумма всех значений)**

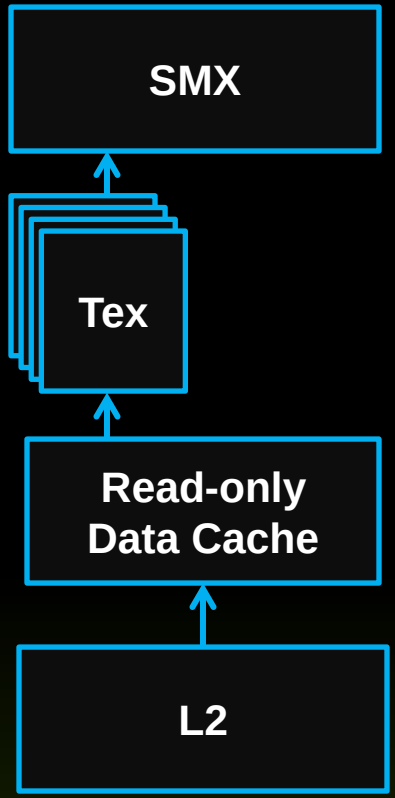


С атомарными операциями

1. Поделить входной массив на N частей
2. Запустить N блоков, каждый выполняет редукцию для своей части
3. Записать результат напрямую посредством атомарных операций

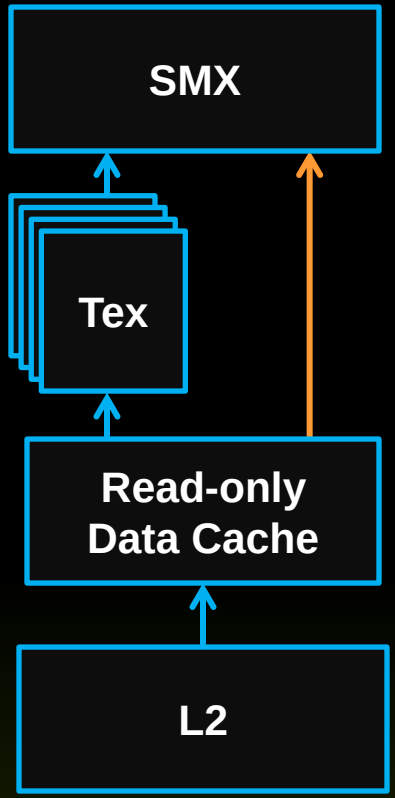
Текстуры: производительность

- Текстуры:
  - Аппаратное ускорение фильтрации данных (1D, 2D, 3D)
  - Read-only (текстурный) кэш
  - Чтение происходит через L2 кэш
- Кеплер SMX vs Ферми SM :
  - 4x операций фильтрации за такт
  - 4x размер кэша (48 КБ)



Прямой доступ к текстурному кэшу

- Новый способ использования
  - В обход текстурного блока
  - Кэшируемый доступ по любому глобальному адресу
  - Не требуется шаг подготовки текстур
- Чем он полезен?
  - Отдельный от smem/L1 путь
  - Высокая полоса пропускания к L2
  - Невыровненный доступ
- Автоматически используется компилятором
  - Для параметров “const __restrict”



Пример с const __restrict

- Укажите `const __restrict` для нужных параметров (полезно и для Ферми)
- Компилятор сгенерирует инструкцию загрузки через текстурный кэш
- Встроенная функция `__ldg`

```
__global__ void saxpy(float x, float y,
                      const float * __restrict input,
                      float * output)
{
    size_t offset = threadIdx.x +
                    (blockIdx.x * blockDim.x);

    // Compiler will automatically use texture
    // for "input"
    output[offset] = (input[offset] * x) + y;
}
```

Funnel Shift

- Произвольный сдвиг 64-битового поля, полученного объединением двух 32-битовых регистров
  - Результат записывается в 32-битовый регистр
  - Таким образом можно реализовать циклический сдвиг
- Встроенная функция: __funnelshift_*
  - PTX: shf
  - Документация по “PTX ISA” и “Inline PTX in CUDA” в CUDA toolkit
- Возможное использование: криптография, SHA*, MD5

fp32 деление

- Инструкция для ускорения точного деления fp32
- Напрямую недоступна, генерируется компилятором для GK110
  - Если указан ключ –prec-div=false, то деление приближенное, инструкция не нужна
- Зачем это разработчику CUDA?
  - Этот слайд здесь исключительно для того, чтобы прояснить вопрос с fp32

SIMD видео инструкции

- Набор инструкций для простейших операций с упакованными целыми числами
  - Пара 16-битовых значений или четверка 8-битовых
  - add, sub, arg, absdiff, min, max, set
- Доступны только в PTX
  - Документация по PTX ISA в CUDA toolkit

Количество инструкций за такт на один SM/SMX

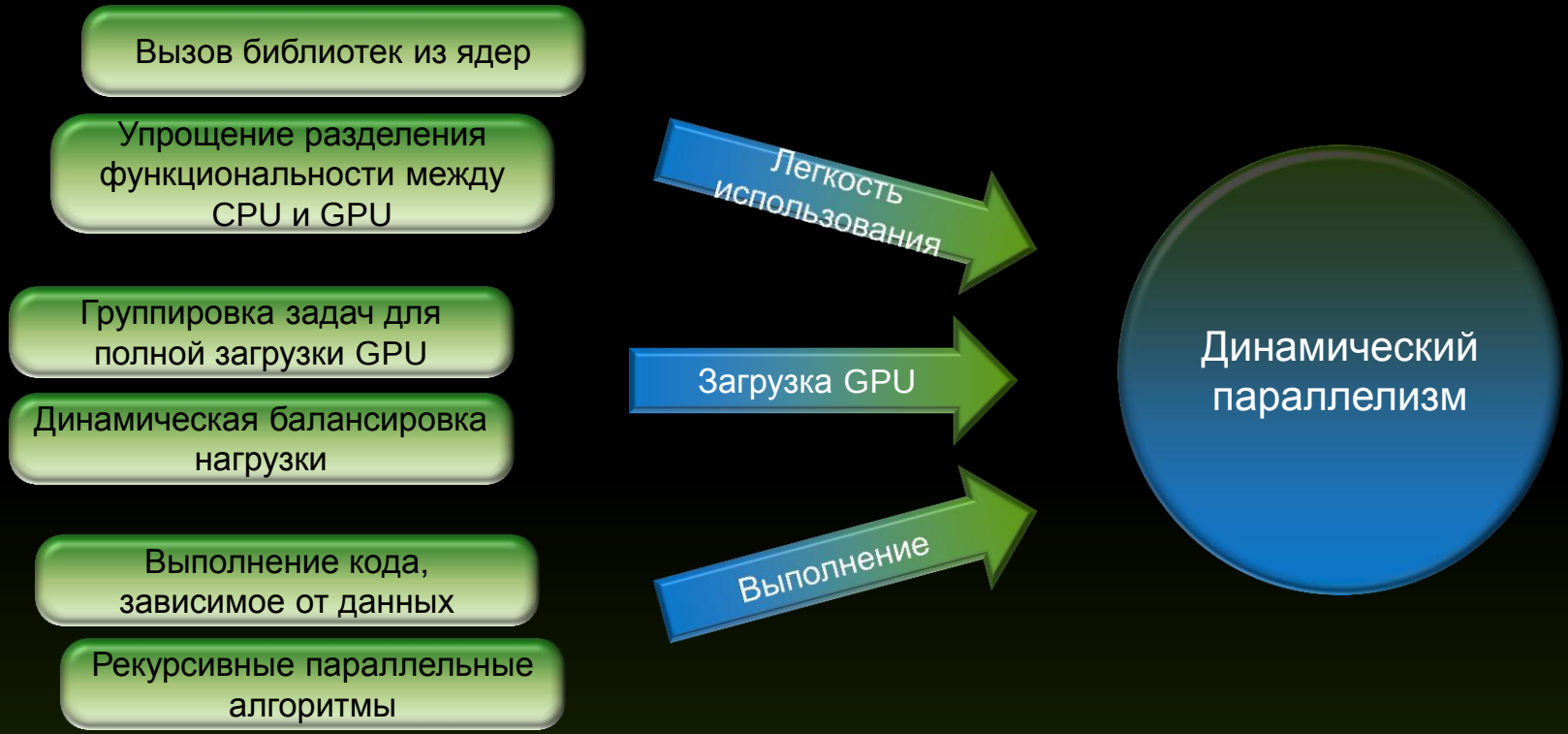
Операция	GF110 (sm_20) Работает на удвоенной частоте!	GK104 (sm_30)	GK110 (sm_35)
32bit fp add, mul, fma	32	192	192
64bit fp add, mul, fma	16*	8	64
32bit int compare, add	32	160	160
32bit int shift	16	32	64
Логические операции	32	160	160
32bit int mul, mad, sad	16	32	32
32bit специальные функции	4	32	32
8bit, 16bit int => 32bit преобразования	16	128	128
Преобразования из или в 64bit	16*	8	32
Все другие преобразования	16	32	32

* Для графических карт (GeForce) производительность ниже

Подсистема памяти Кеплер GK110

- Улучшенный контроллер памяти GDDR5
  - Достигнуты пиковые частоты памяти
- Больший L2 кэш
  - Удвоенная пропускная способность
  - Удвоенный размер (1.5 МБ)
- Эффективная реализация DRAM ECC
  - Издержки при использовании DRAM ECC уменьшены на 66% (среднее по ряду приложений)

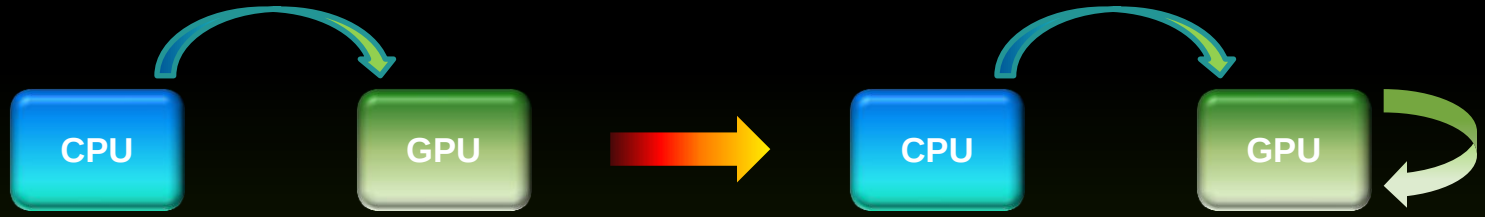
Повышение эффективности программирования



Что такое Динамический Параллелизм?

Возможность запускать новые сетки (ядра) с GPU

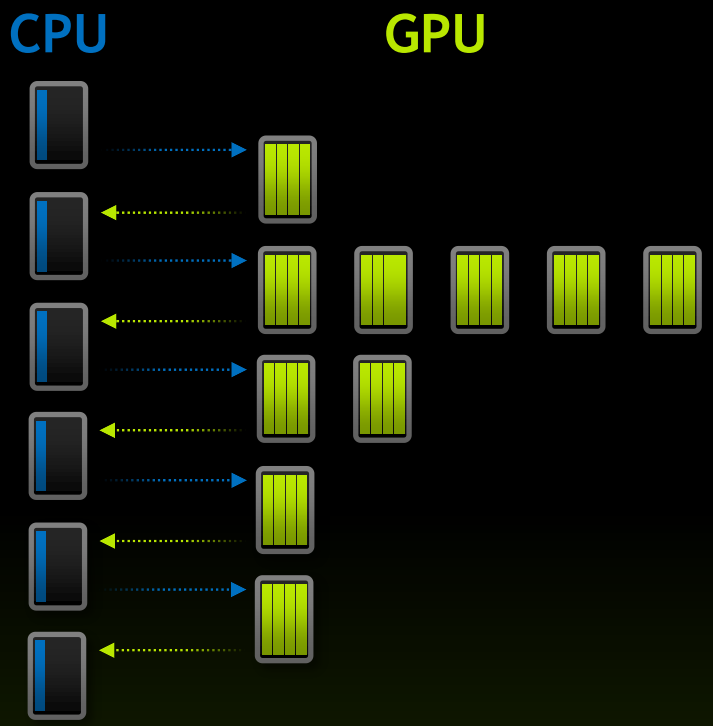
- Динамически
- Одновременно
- Независимо



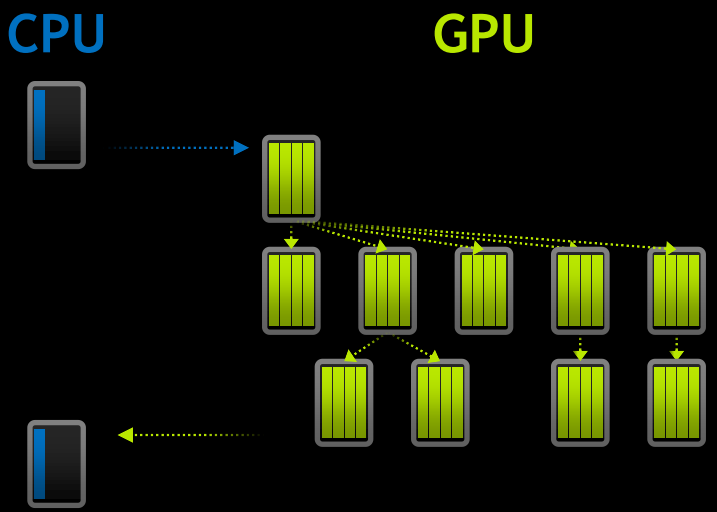
Ферми: Только CPU может создавать работу для GPU

Кеплер: GPU может создавать работу для себя

Что это означает?

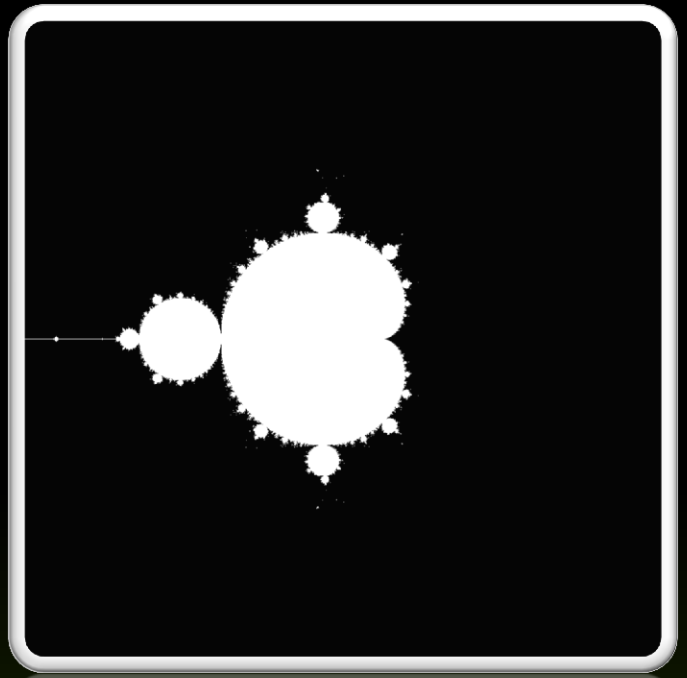


GPU как сопроцессор



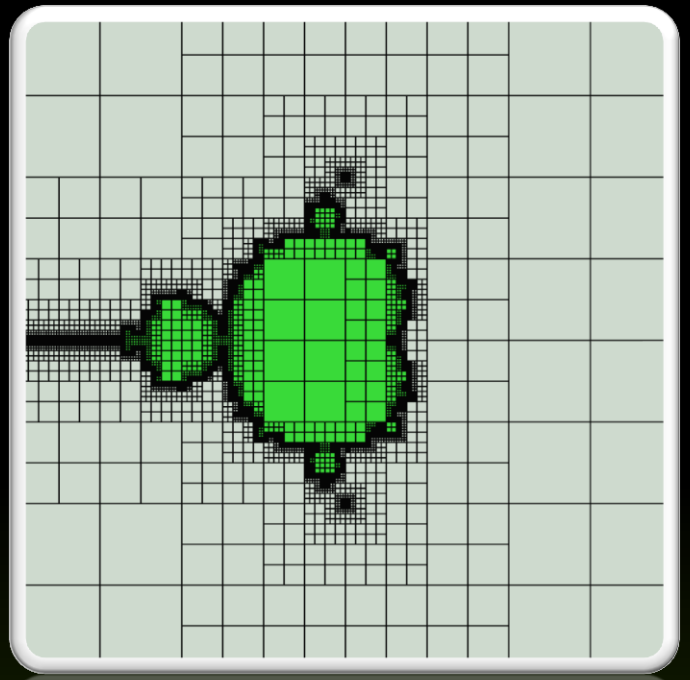
Автономный, Динамический параллелизм

Параллелизм, зависимый от данных



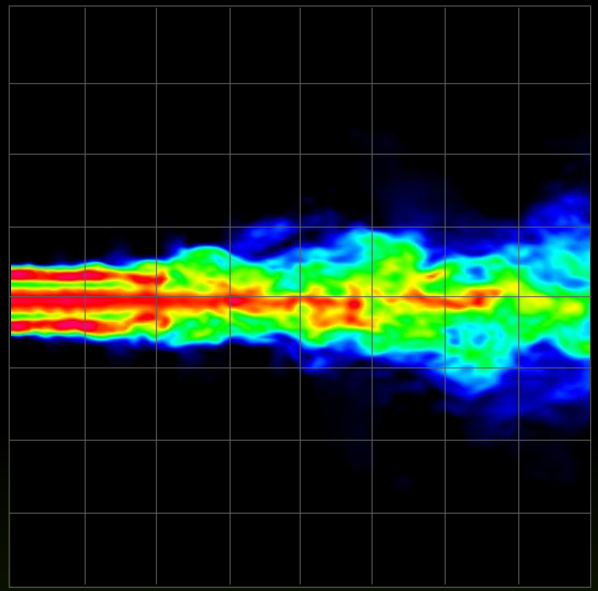
CUDA сегодня

Вычислительные мощности привлекаются для расчета важных областей



CUDA на Кеплере

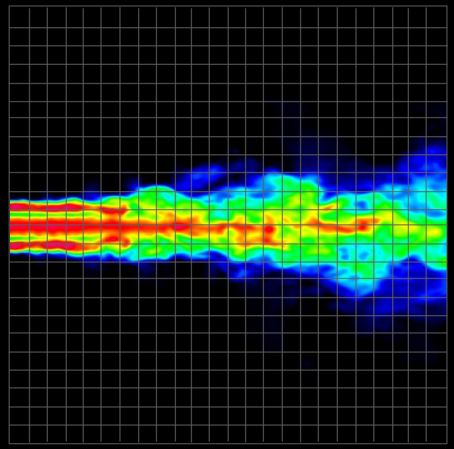
Динамическая генерация задач



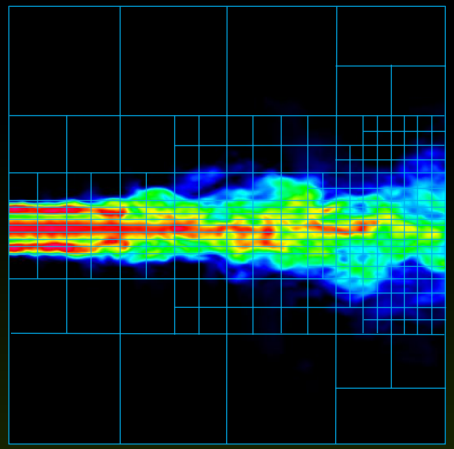
Первоначальная сетка

Статично назначить сетку максимального разрешения

Фиксированная сетка



Динамически определять размер сетки согласно необходимости

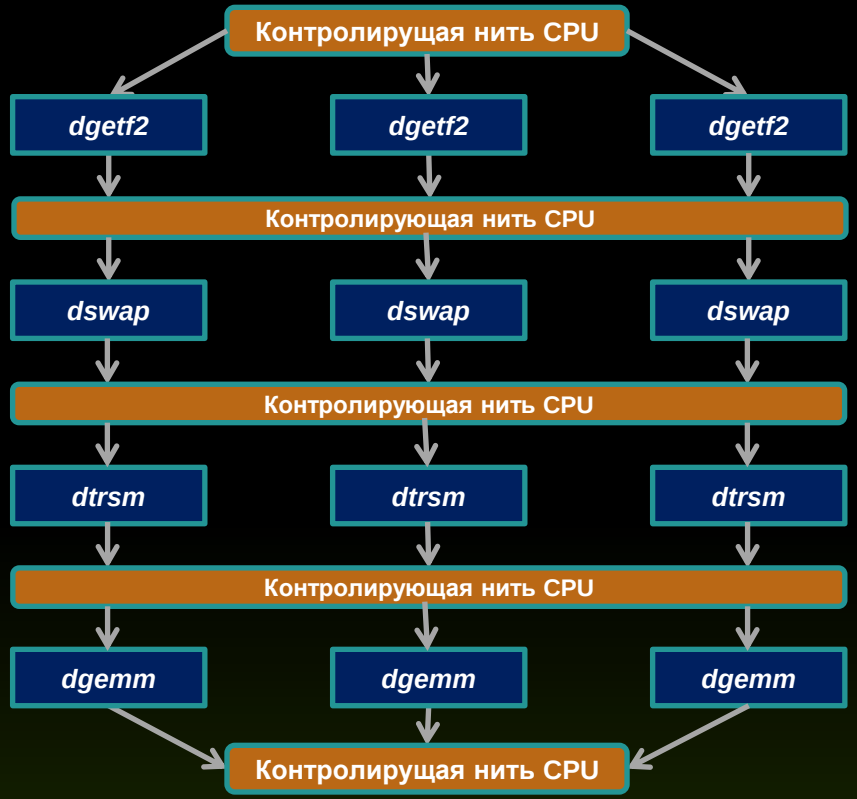


Динамическая сетка

Пакетный и Вложенный Параллелизм

Пакетирование работы с использованием CPU

- Программы на CPU ограничены единственной контрольной точкой
- Можно запускать до десятков нитей
- CPU полностью загружен контролем запусков



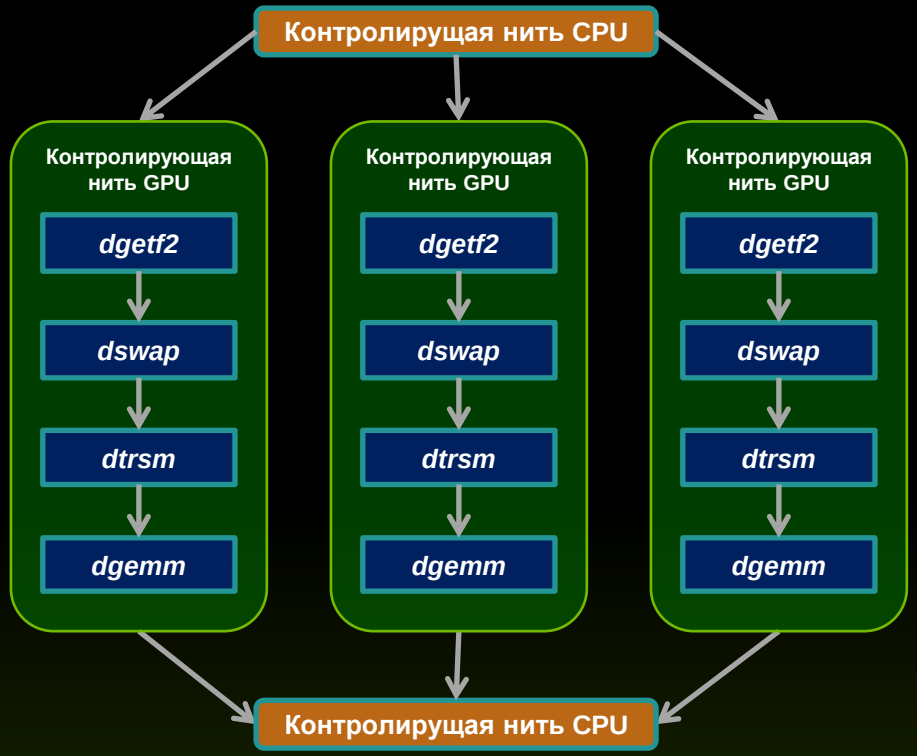
Пакетная LU-Факторизация, до Кеплера

Алгоритм упрощен для простоты иллюстрации

Пакетный и Вложенный Параллелизм

Пакетирование работы с использованием динамического параллелизма

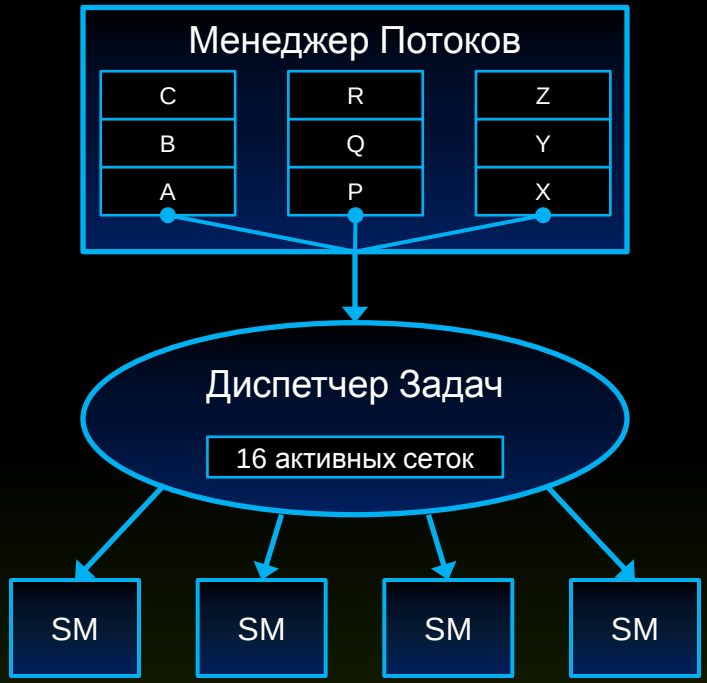
- Перенос внешних циклов на GPU
- Запуск тысяч независимых задач
- Высвобождение CPU



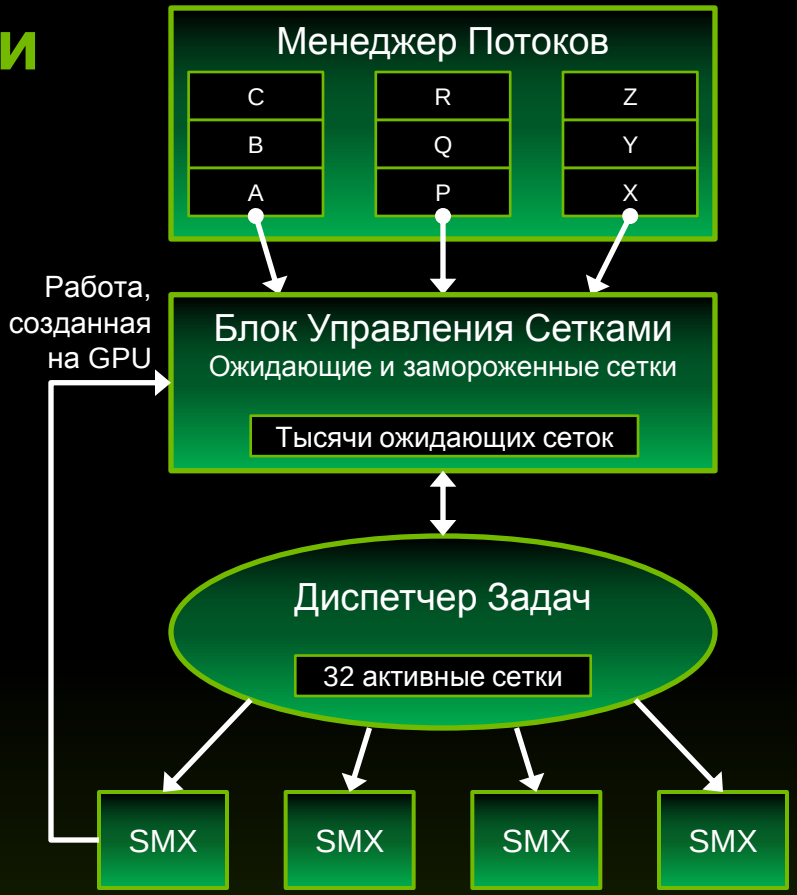
Пакетная LU-Факторизация, Кеплер

Алгоритм упрощен для простоты иллюстрации

Блок Управления Сетками

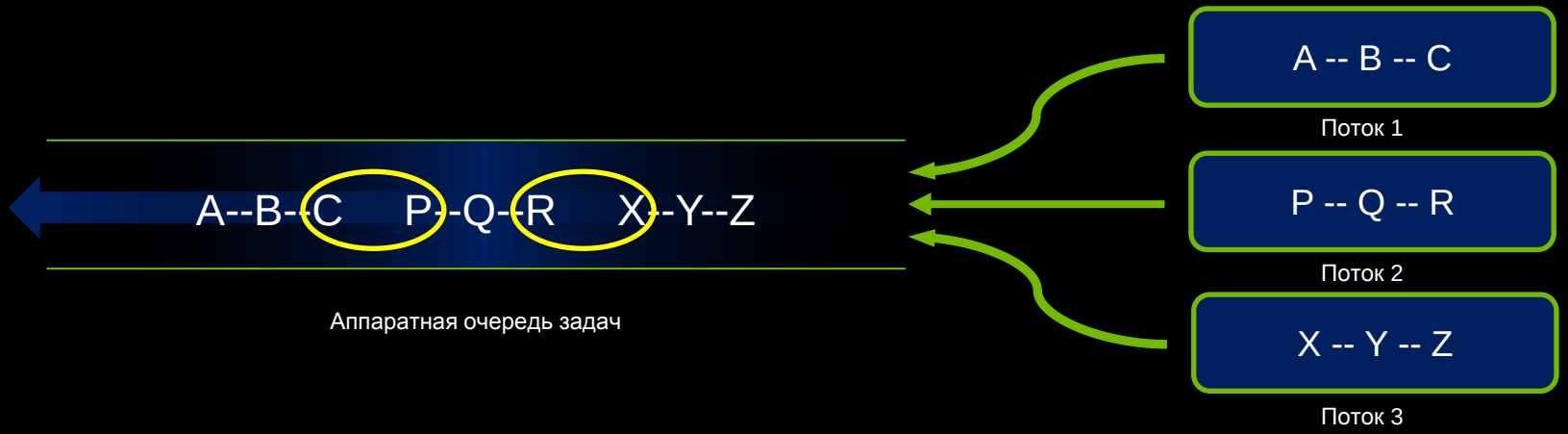


Фермі



Кеплер GK110

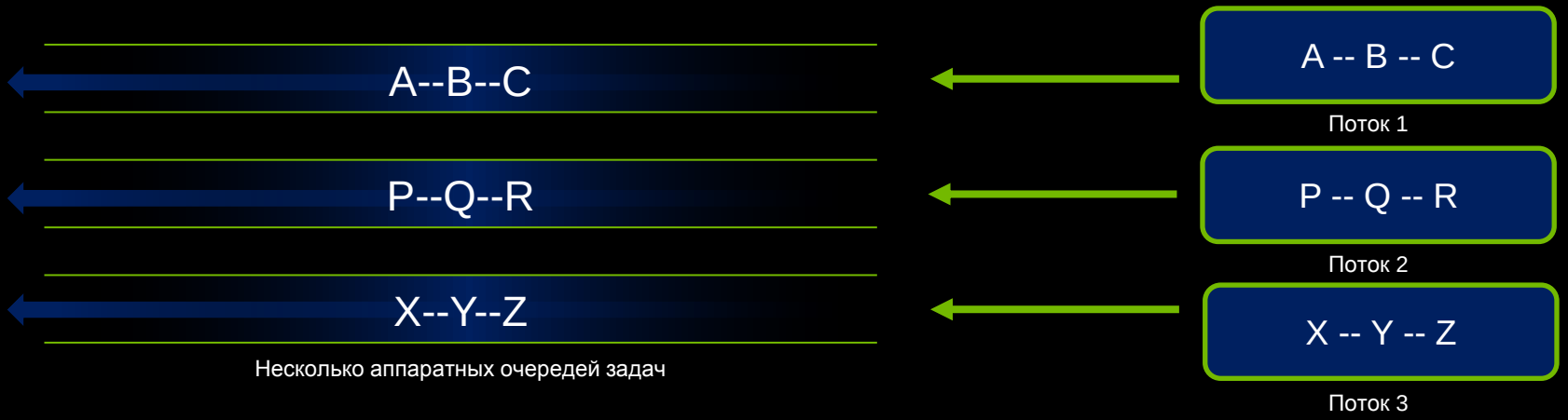
Одновременное выполнение сеток на Ферми



До 16 сеток могут выполняться одновременно. Но!

- Все потоки CUDA объединяются в единственную аппаратную очередь
- Одновременная выполнение только при переходе от одного потока к другому

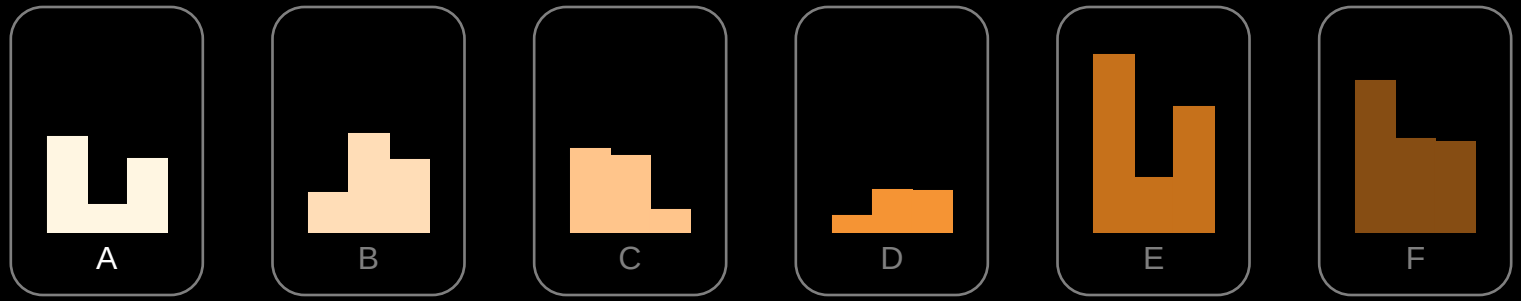
Одновременное выполнение сеток на Кеплере



До 32 сеток может выполняться одновременно

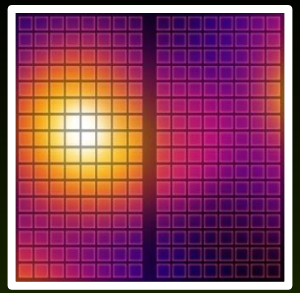
- Одна очередь на поток
- Параллельность для потоков
- При условии отсутствия зависимостей между потоками

Ферми: последовательное выполнение

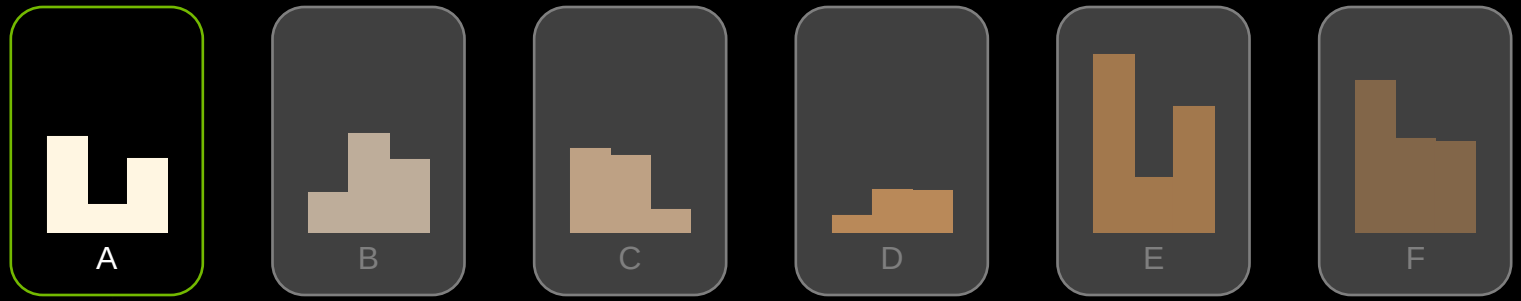


Процессы CPU

Разделяемый GPU

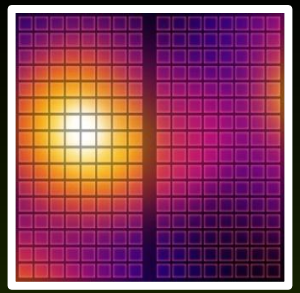


Ферми: последовательное выполнение

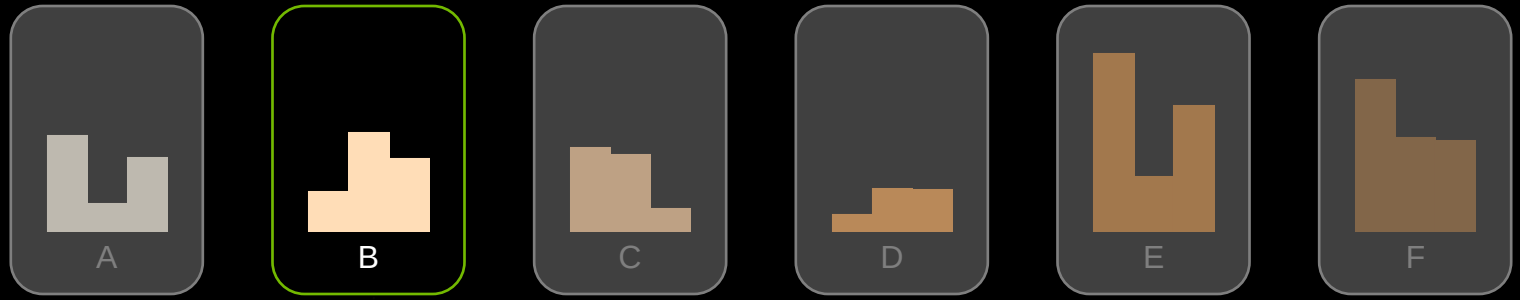


Процессы CPU

Разделяемый GPU

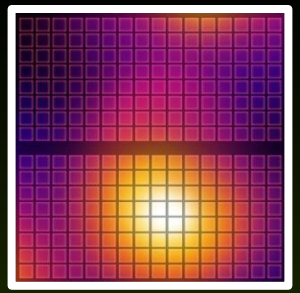


Ферми: последовательное выполнение

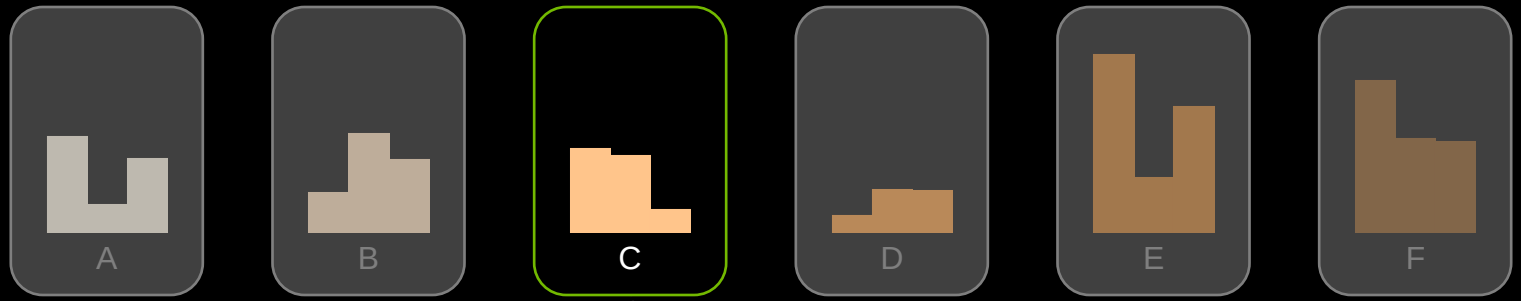


Процессы CPU

Разделяемый GPU

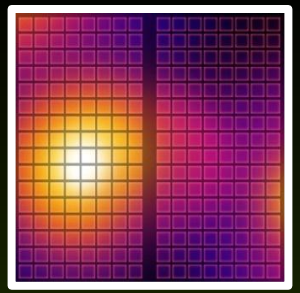


Ферми: последовательное выполнение

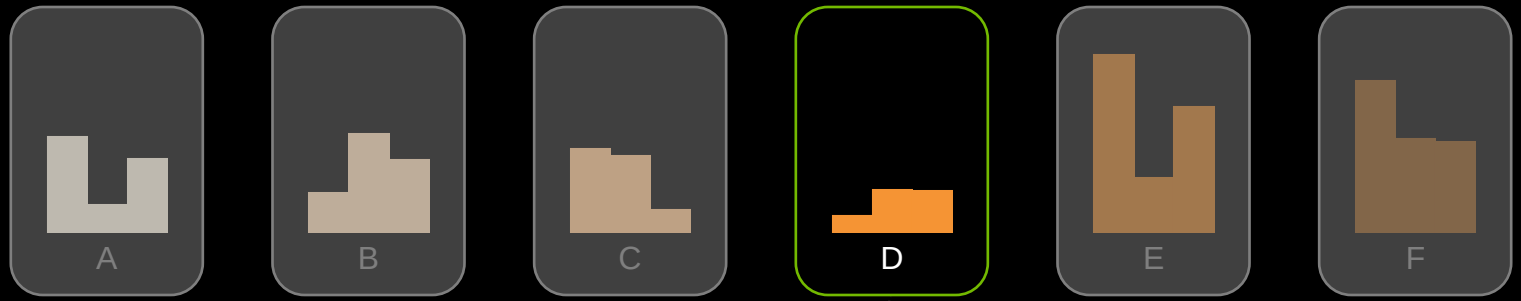


Процессы CPU

Разделяемый GPU

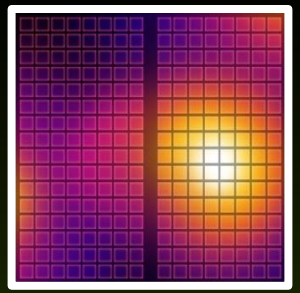


Ферми: последовательное выполнение

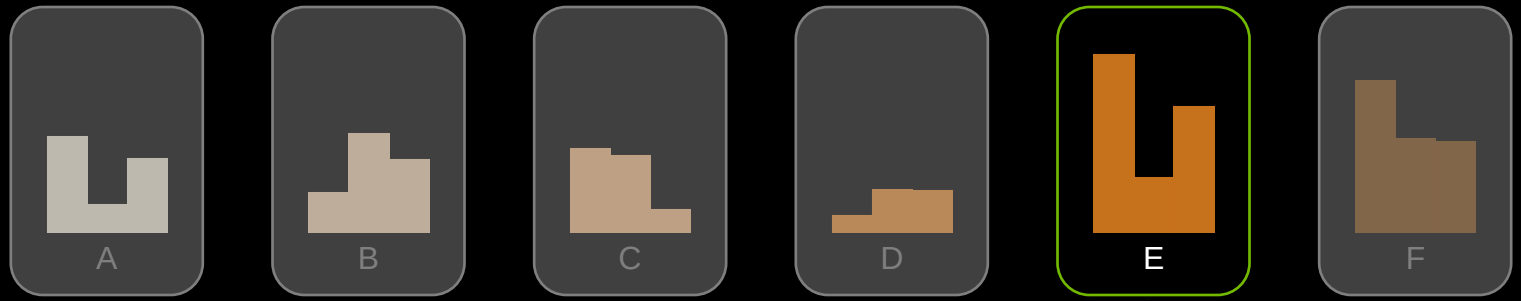


Процессы CPU

Разделяемый GPU

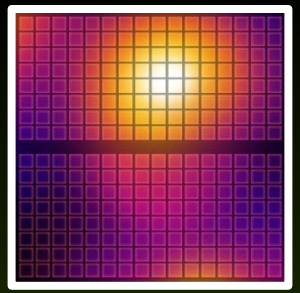


Ферми: последовательное выполнение

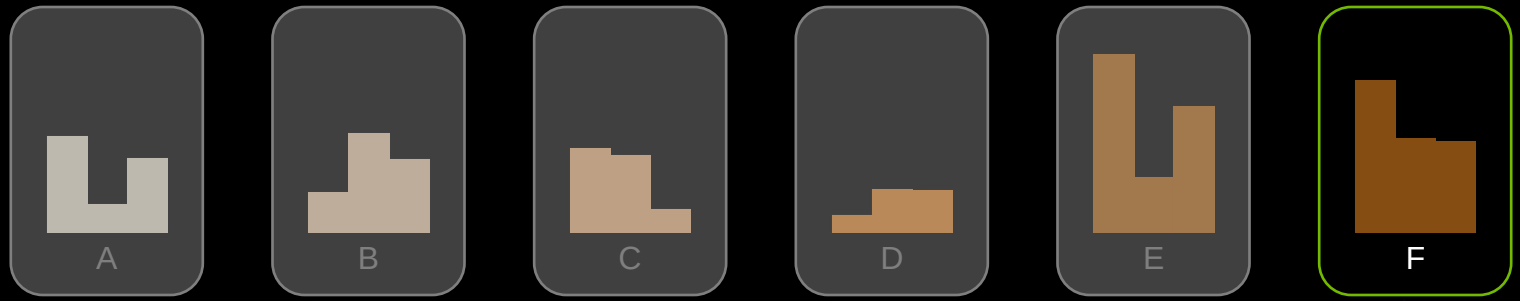


Процессы CPU

Разделяемый GPU

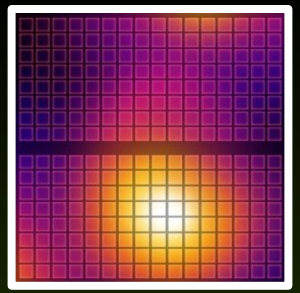


Ферми: последовательное выполнение

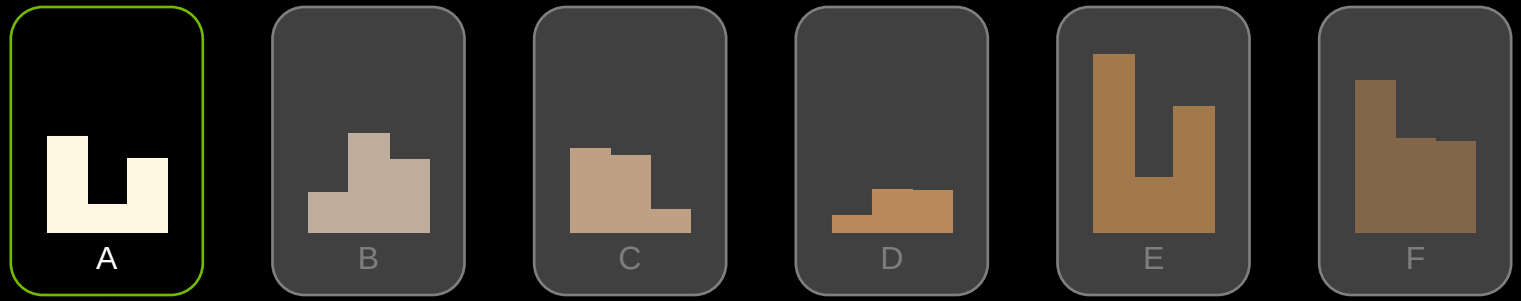


Процессы CPU

Разделяемый GPU

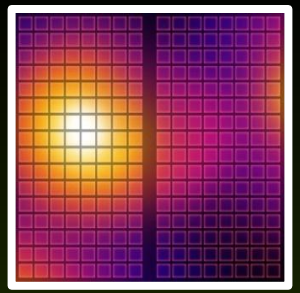


Ферми: последовательное выполнение

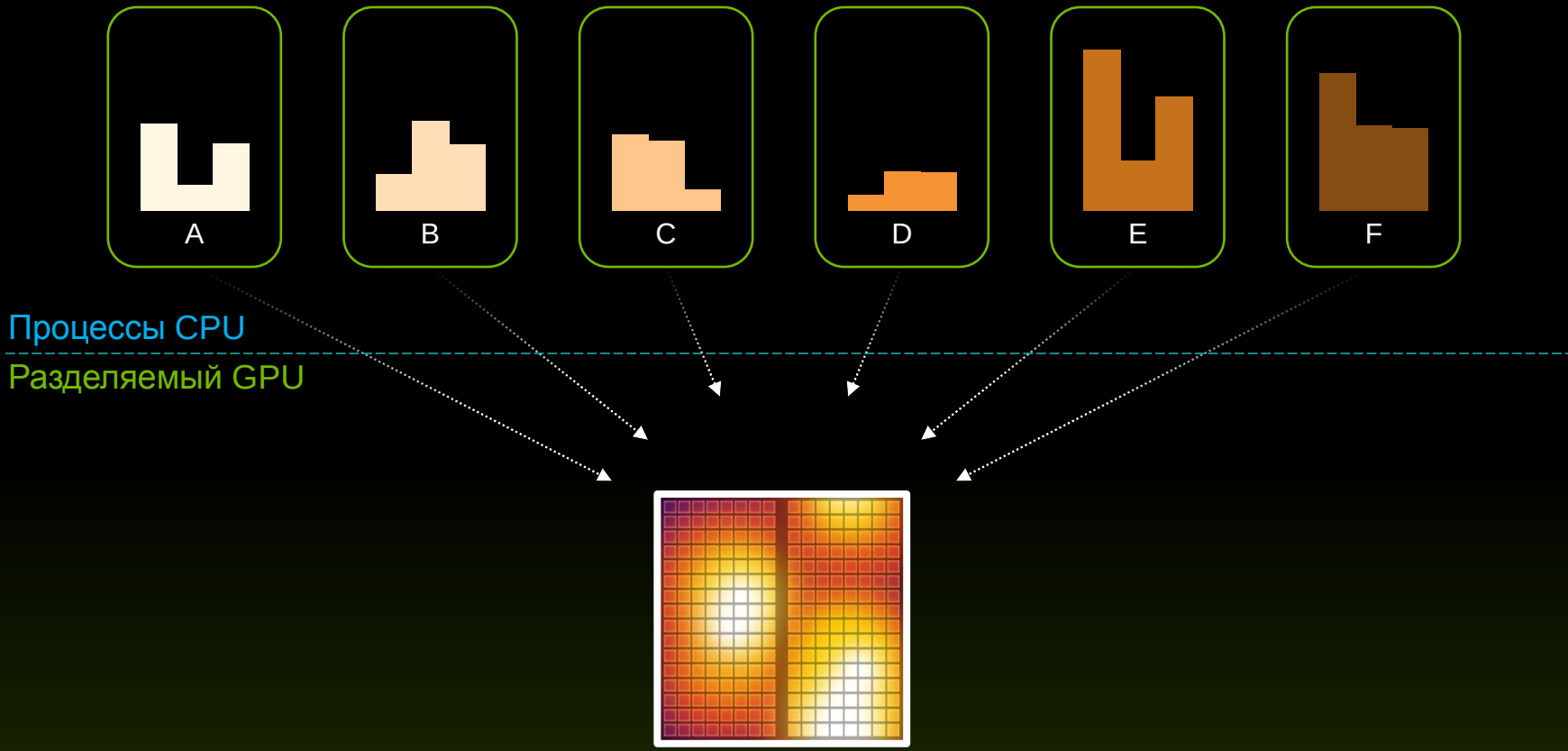


Процессы CPU

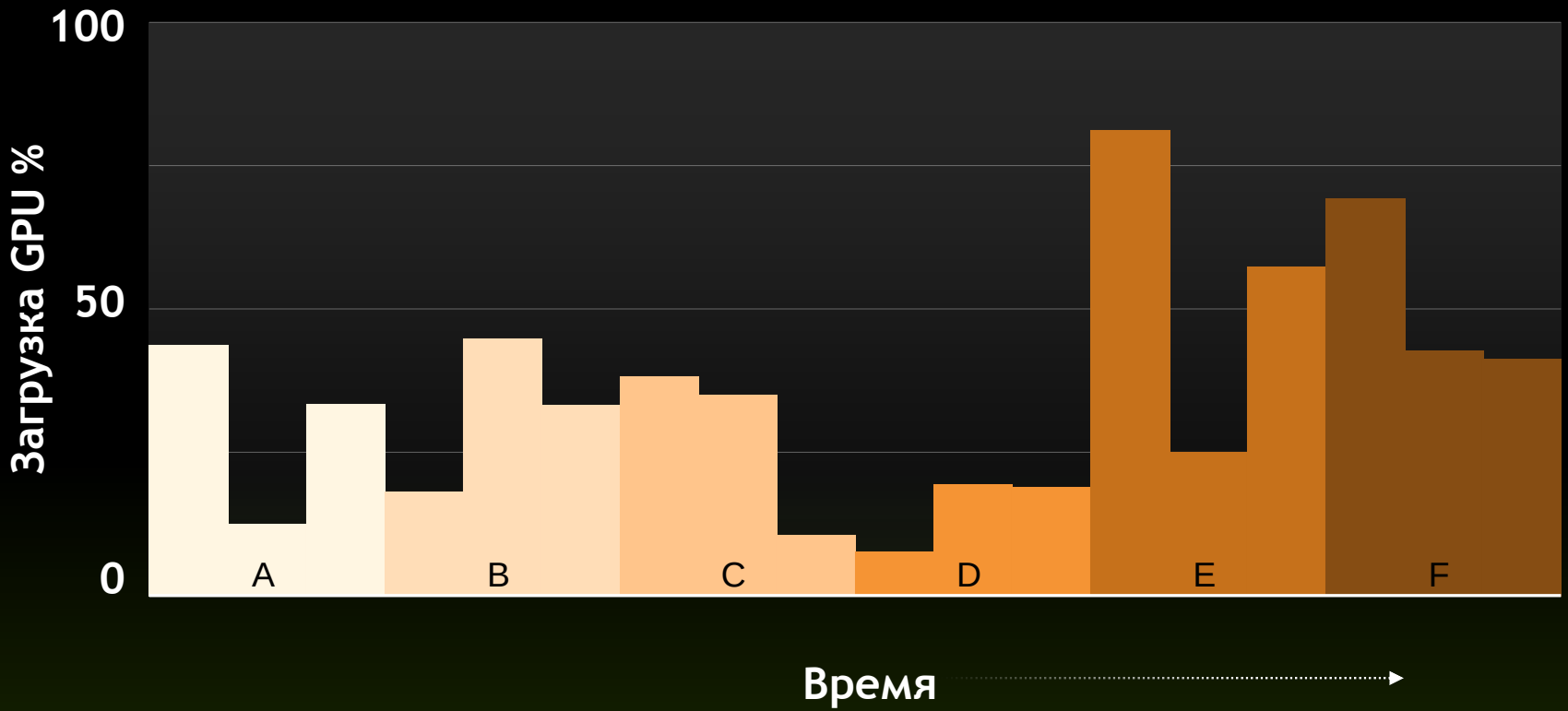
Разделяемый GPU



Кеплер, Hyper-Q: параллельное выполнение



Без Hyper-Q

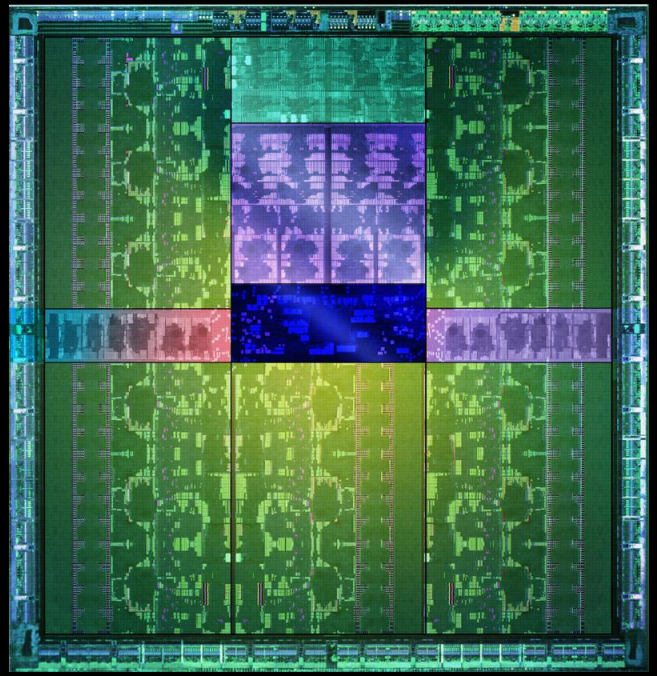


С использованием Hyper-Q

Функциональные отличия GK104 и GK110

Функциональность	GK104 (sm_30)	GK110 (sm_35)
Разделяемая память: 64-битный режим адресации	X	X
До 255 регистров на нить		X
SHFL	X	X
Быстрые атомарные функции	X	X
64-битные атомарные min/max/and/or/xor		X
Текстурный кэш: ускорение и увеличение объема	X	X
Прямая загрузка через текстурный кэш		X
Funnel Shift		X
fp32 деление		X
SIMD инструкции для обработки видео	X	X
Динамический параллелизм		X
Hyper-Q		X

* Для графических карт (GeForce) производительность ниже



Whitepaper: http://www.nvidia.com/kepler
CUDA Toolkit: “CUDA C Programming Guide”, “Kepler Tuning Guide”, “PTX ISA 3.1”