Новые возможности CUDA 4.0

Сахарных Николай

NVIDIA

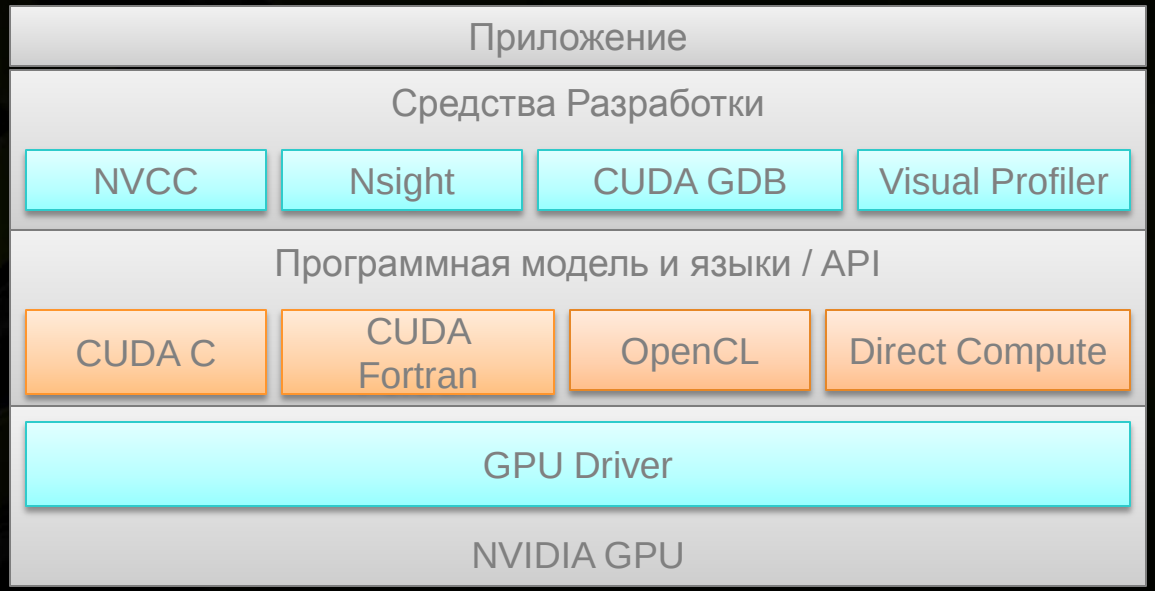
Agenda

- Введение в программную модель CUDA
- Обзор архитектуры Fermi
- Новые возможности CUDA 4.0

NVIDIA Confidential

Программная модель CUDA

- **CUDA – программно-аппаратный стек для программирования GPU**



NVIDIA Confidential

Программная модель CUDA

- Код состоит из последовательных и параллельных частей
- Последовательные части кода выполняютсяна CPU (host)
- Массивно-параллельные части кода выполняются на GPU (device)
  - Является сопроцессором к CPU (host)
  - Имеет собственную память (DRAM)
  - Выполняет одновременно десятки тысяч нитей

NVIDIA Confidential

Программная модель CUDA

- Параллельная часть кода выполняется как большое количество нитей (threads)
- Нити группируются в блоки (blocks) фиксированного размера
- Блоки объединяются в сеть блоков (grid)
- Ядро выполняется на сетке блоков

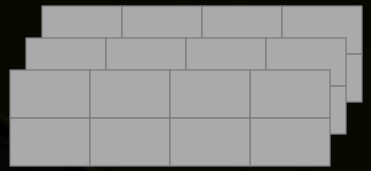
NVIDIA Confidential

Программная модель

- Нити в CUDA объединяются в блоки
  - 1D топология блока

  - 2D топология блока

  - 3D топология блока



- Общее число нитей в блоке ограничено

NVIDIA Confidential

Программная модель

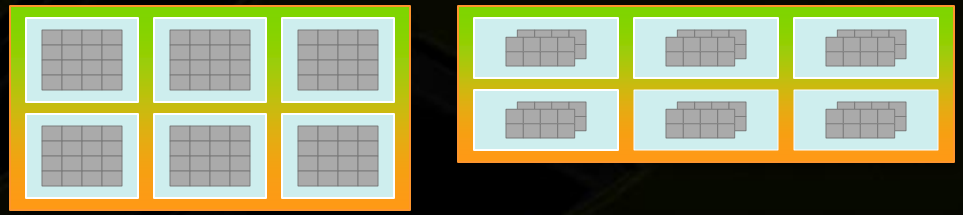
- Блоки могут использовать разделяемую (shared) память
  - Нити могут обмениваться данными
- Внутри блока потоки могут синхронизироваться

NVIDIA Confidential

Программная модель

- Блоки потоков объединяются в сеть (grid) блоков потоков
  - 1D топология блоков потоков

  - 2D топология блоков потоков



- Блоки в сети выполняются **независимо** друг от друга

NVIDIA Confidential

Agenda

- Введение в программную модель CUDA
- Обзор архитектуры Fermi
- Новые возможности CUDA 4.0

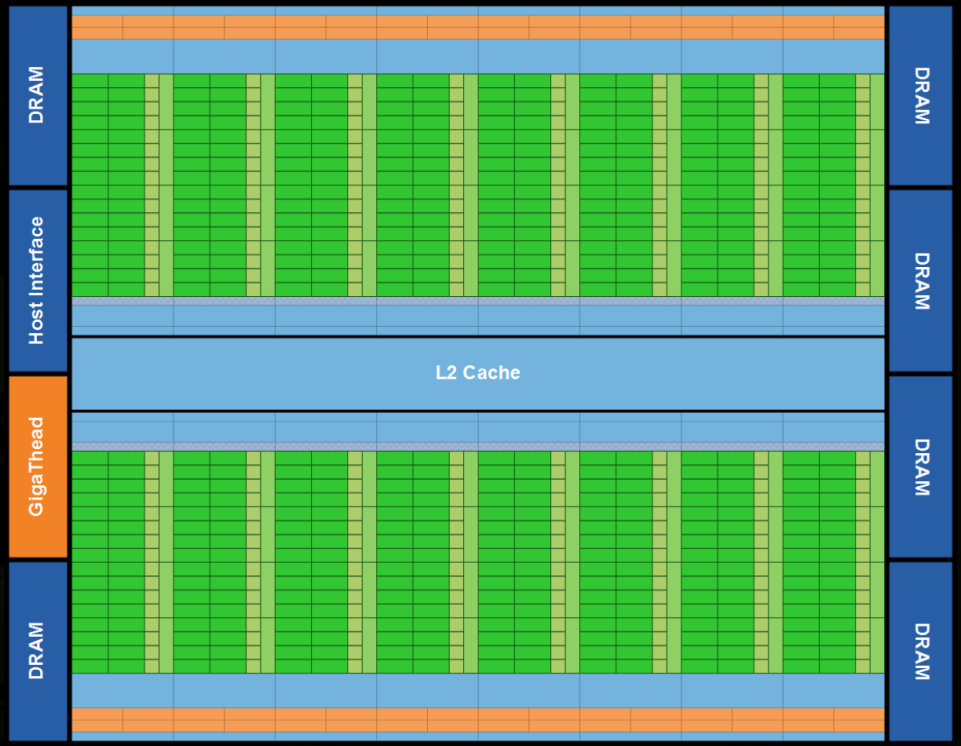
NVIDIA Confidential

Архитектура Tesla (Tesla T10)

30 потоковых мультипроцессоров (SM)
SM: 8 скалярных процессоров (SP), 2 SFU, 1 DP
SM: 16K разделяемой памяти (SMEM)

NVIDIA Confidential

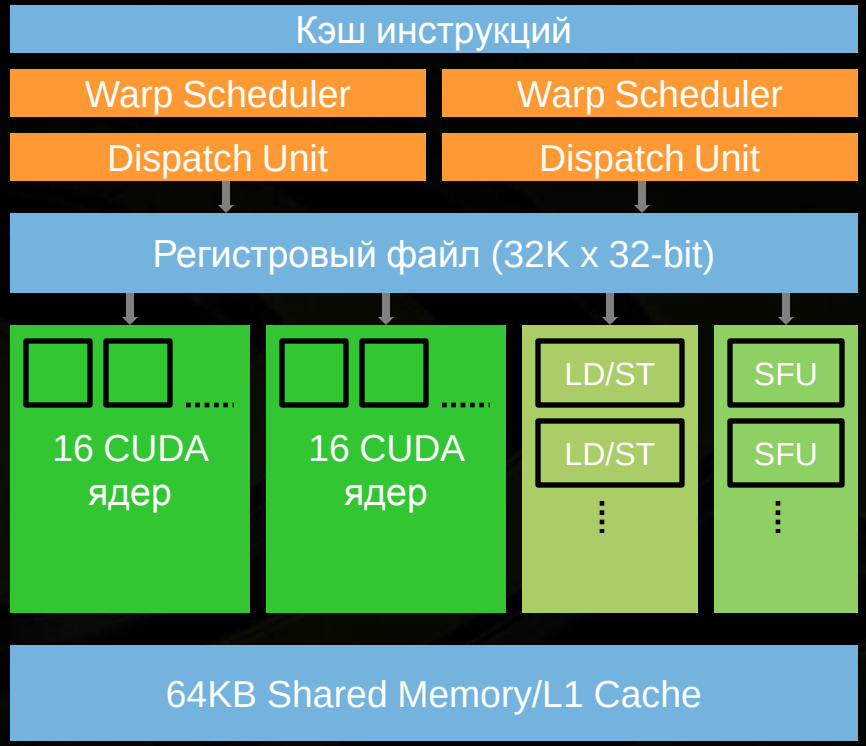
Архитектура Fermi (Tesla T20)



16 потоковых процессоров (SM)

NVIDIA Confidential

Fermi – SM

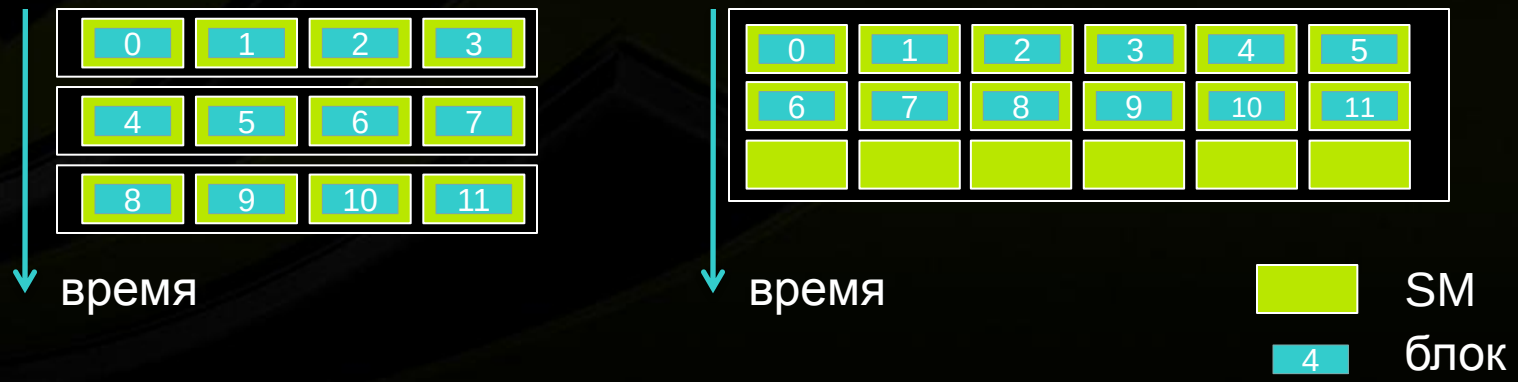


- 32 CUDA ядра на SM
  - INT: 32 инструкции/такт
  - FP: 32 SP / 16 DP за такт
- 4 SFU на SM
  - 4 инструкции за такт
- 32K регистров (32-бит)
- Улучшенная точность
  - IEEE стандарт для SP/DP
  - FMA для SP/DP

NVIDIA Confidential

Связь программной модели с HW

- Блок целиком выполняется на одном SM
  - Блоки могут использовать разделяемую (shared) память
  - Внутри блока нити могут синхронизироваться
- Масштабирование производительности



NVIDIA Confidential

Связь программной модели с HW

- Блоки – абстракция программной модели
- Варп (warp) – реальная единица исполнения устройства
  - = 32 нити на архитектуре Tesla/Fermi

Блок 0

Варп 0
Варп 1
Варп 2
Варп 3
Варп 4

NVIDIA Confidential

Single Instruction Multiple Threads

- Параллельно на каждом SM выполняется большое число отдельных нитей
- Нити в пределах одного варпа выполняются физически параллельно
- Разные варпы могут исполнять разные инструкции
- Большое число варпов покрывает задержки

NVIDIA Confidential

Размер блока

- Задержки покрываются переключением варпов
  - Обращение в память – 400-800 тактов
  - Арифметическая инструкция – 18-22 тактов
  - Больше нитей чтобы покрыть все задержки
- Максимальное число варпов в блоке – 48
  - 1536 нитей (на Tesla – 1024)
- Одновременно не более 8 блоков на SM
  - Слишком большой или малый блок - неэффективно
  - Обычно 128-256 нитей на блок
  - В каждом случае нужно определять отдельно

NVIDIA Confidential

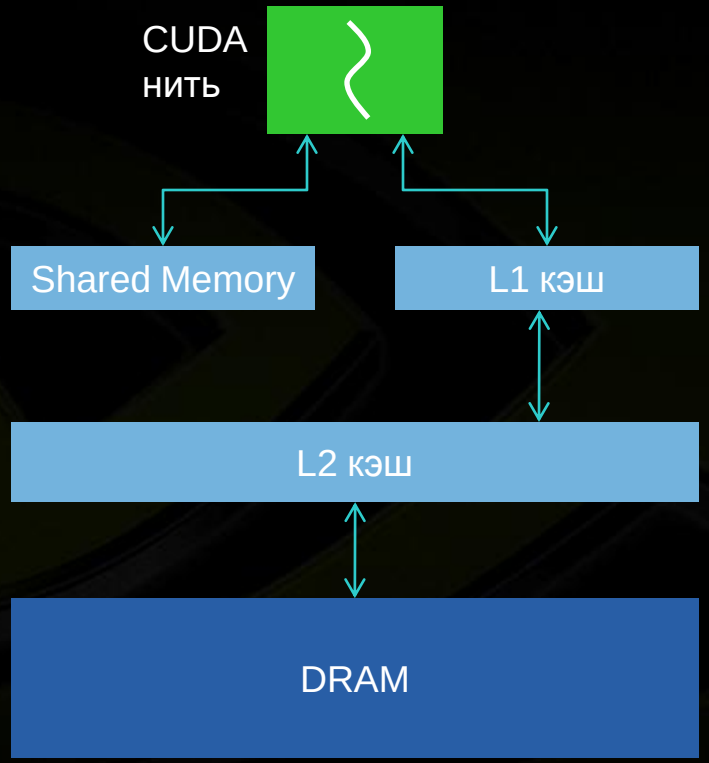
Контроль регистров

- Максимальное число регистров на одну нить – 63
  - На Tesla было 128
- Меньше регистров –> больше блоков может работать на SM –> лучше производительность
- Раньше использовался --maxregcount
- Новая директива для контроля регистров:

```
__global__ void
__launch_bounds__(maxThreadsPerBlock, minBlocksPerMultiprocessor)
MyKernel(...) { ... }
```

NVIDIA Confidential

Fermi – Подсистема памяти



- Настраиваемый L1 кэш для каждого SM
  - 16КБ SMEM, 48КБ L1
  - 48КБ SMEM, 16КБ L1
- Общий L2 кэш для всех SM
  - 768КБ
- Атомарные операции
  - 20x быстрее чем на Tesla
- ECC, коррекция ошибок
  - Single-Bit Correct
  - Double-Bit Detect

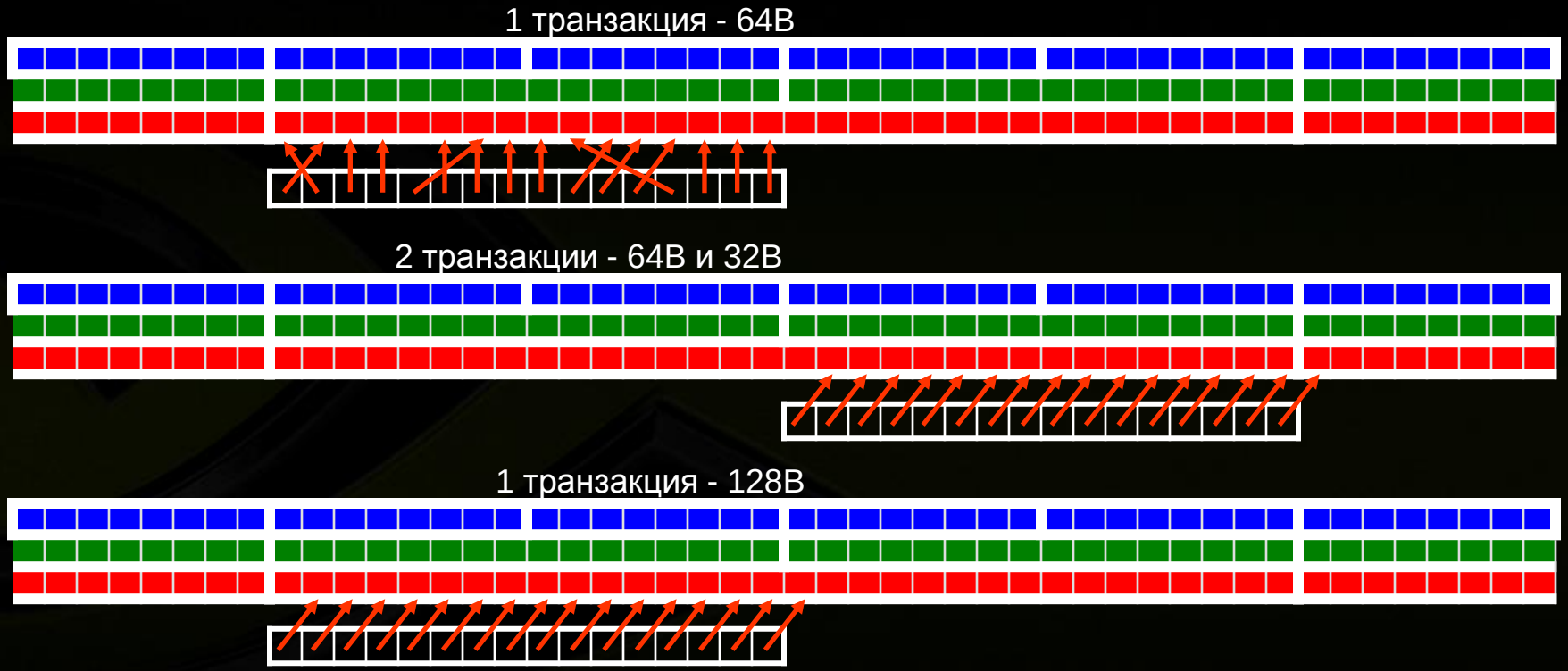
NVIDIA Confidential

Fermi – Кэш L1

- Кэш линия – 128B
- L1 можно отключать
  - Уменьшение размера транзакции с памятью до 32B
- L1 некогерентен
  - Зависимостей между блоками быть не должно
- Чтение и запись в локальную память
  - L1 помогает при интенсивном использовании
  - (нет проблем с когерентностью)

NVIDIA Confidential

Глобальная память на Tesla

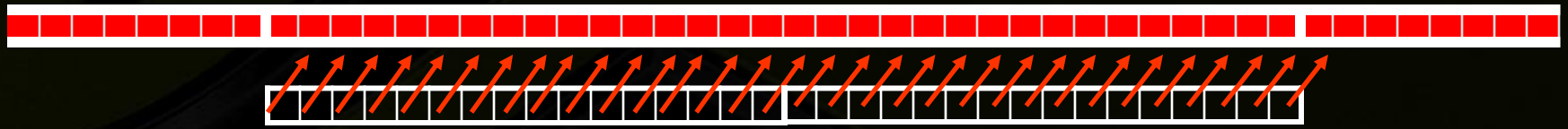


NVIDIA Confidential

Глобальная память на Fermi

- Объединение запросов в память для 32 нитей
- L1 включен – всегда идут запросы по 128B с кэшированием в L1

2 транзакции - 2 x 128B



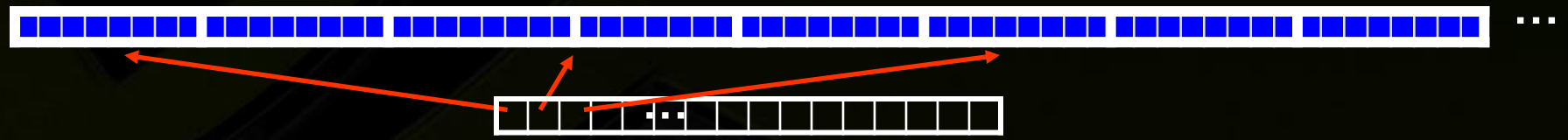
следующий варп скорей всего только 1 транзакция, т.к. попадаем в L1

NVIDIA Confidential

Глобальная память на Fermi

- **L1 выключен – всегда идут запросы по 32B**
- **Лучше для разреженного доступа к памяти**

32 транзакции по 32B, вместо 32 x 128B



NVIDIA Confidential

Fermi: LDU

- На Tesla одинаковые данные предпочтительно хранить в константной памяти (64KB)
  - Константный кэш делает броадкаст всем нитям
- На Fermi появилась инструкция LDU
  - Возможность кэшировать константные значения для блока из глобальной памяти

```
__global__ void kernel( float *g_dst, const float *g_src )
{
    g_dst = g_src[0];              // uniform
    g_dst = g_src[blockIdx.x];     // uniform
    g_dst = g_src[threadIdx.x];    // non-uniform
}
```

NVIDIA Confidential

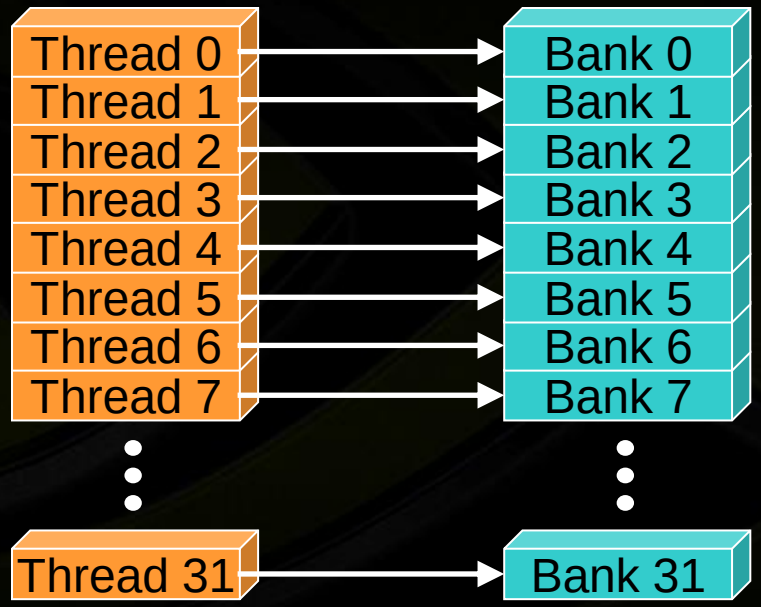
Разделяемая память

- Использование
  - Общение нитей между собой внутри блока
  - Кэширование данных из глобальной памяти
  - Улучшение шаблонов доступа глобальной памяти
- Архитектура Tesla – 16 банков по 32-бит, конфликты возникают на уровне полуварпа
- Архитектура Fermi – 32 банка по 32-бит, конфликты возникают на уровне варпа

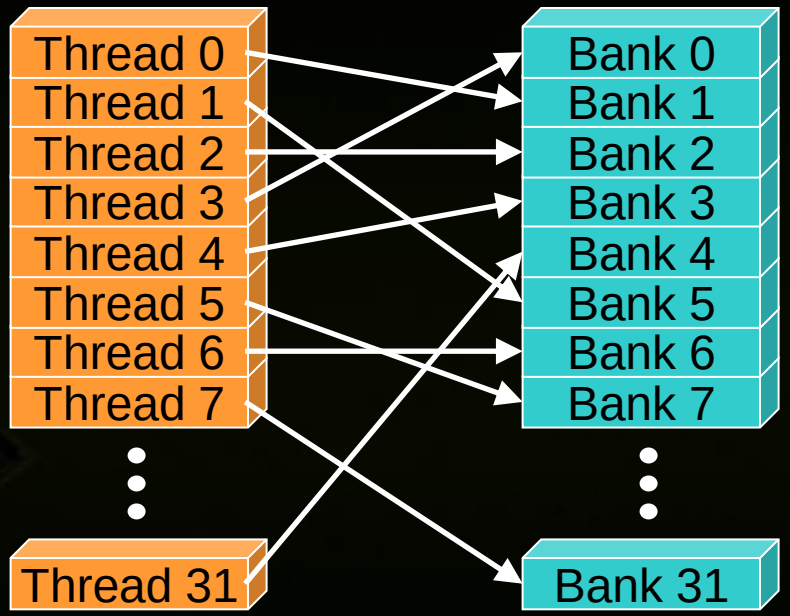
NVIDIA Confidential

Пример доступа к банкам

- Нет конфликтов банков



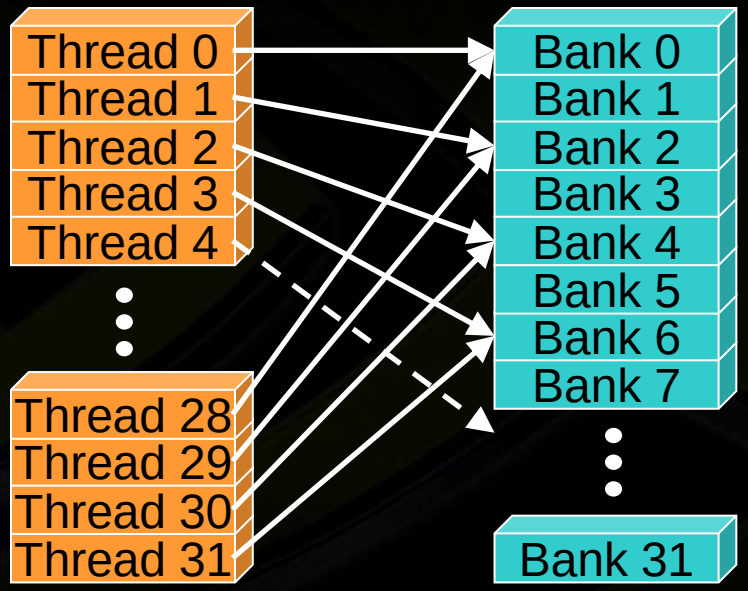
- Нет конфликтов банков



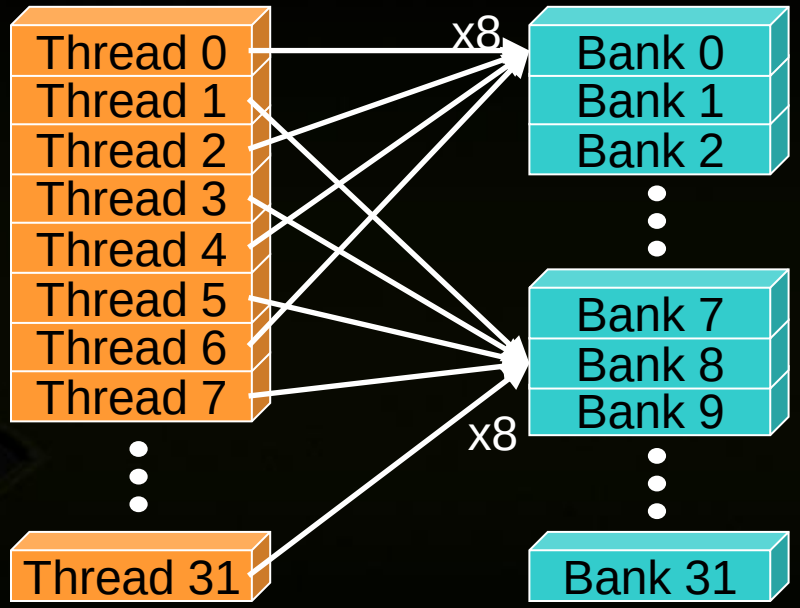
NVIDIA Confidential

Пример доступа к банкам

- 2x конфликты банков



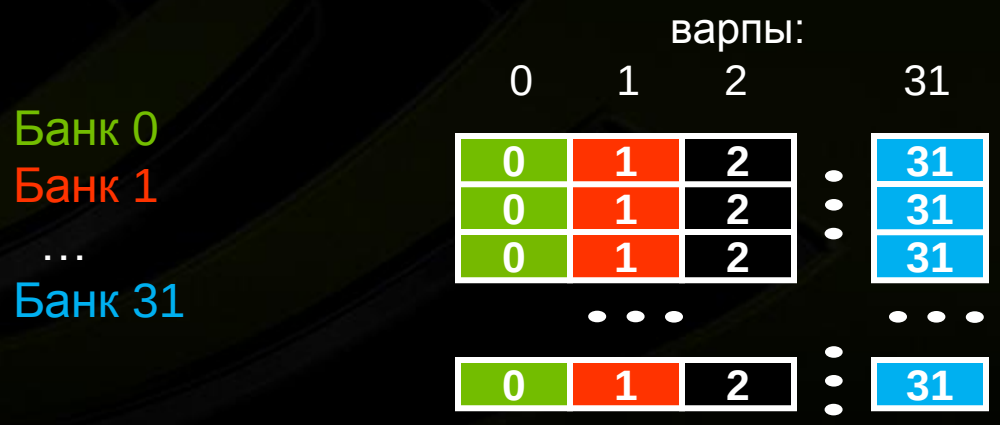
- 8x конфликты банков



NVIDIA Confidential

Устранение конфликтов

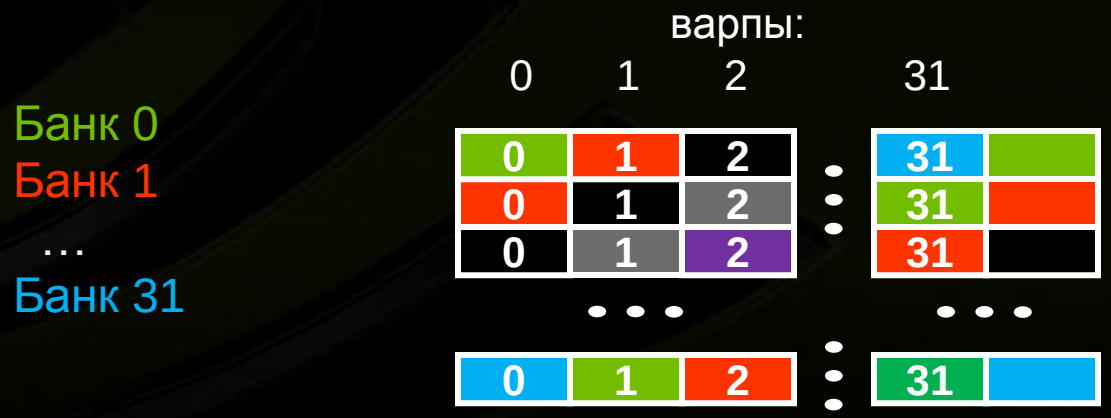
- Двумерный массив разделяемой памяти 32x32
- Варп обращается к столбцу
  - 32x конфликты банков



NVIDIA Confidential

Устранение конфликтов

- Добавляем дополнительный столбец
  - Двумерный массив 32x33
- Варп обращается к столбцу
  - 32 разных банка, конфликтов нет



NVIDIA Confidential

Разделяемая память на Fermi

- **Multi-cast (32 бит)**
  - **Конфликты банков возникают только, если 2 или больше нитей обращаются к разным словам в одном банке памяти**

```
__shared__ char shared[32];
char data = shared[BaseIndex + tid];
```

NVIDIA Confidential

Разделяемая память на Fermi

- **Чтение/запись double (64 бит)**
  - **Специальная обработка структур размера 64 и 128 бит**
  - **При последовательном доступе к массиву из double конфликты не возникают (в отличие от Tesla)**

```
__shared__ double shared[32];
double data = shared[BaseIndex + tid];
```

NVIDIA Confidential

Новые возможности Fermi

- **Одновременное исполнение нескольких ядер**
  - Сначала запускаются все блоки одного ядра
  - Если ресурсы свободны – запускаются блоки ядра из другой очереди исполнения (CUDA stream)

NVIDIA Confidential

Новые возможности Fermi

- **Новые инструкции**
  - Атомарные операции для FP в одинарной точности
  - Инструкция ballot – возвращает 32-битное число, в i-м разряде 0 или 1 в зависимости от выполнения предиката для i-ой нити в текущем варпе
- **2 интерфейса для копирования**
  - Одновременное копирование CPU->GPU и GPU->CPU

NVIDIA Confidential

Agenda

- Введение в программную модель CUDA
- Обзор архитектуры Fermi
- Новые возможности CUDA 4.0

NVIDIA Confidential

CUDA 3.0 – 3.1

- Унифицированное адресное пространство
  - Для Shared, Global, Local памяти
- Поддержка 64-битной адресации
- ABI (Application Binary Interface)
  - Вызовы функций, включая указатели на функции
  - Пользовательский стэк (в том числе для рекурсии)
- printf()

NVIDIA Confidential

CUDA 3.2

- Улучшение производительности
  - CUBLAS от 50% до 300%
  - CUFFT для степеней 3, 5, 7
- Новые библиотеки
  - CUSPARSE, CURAND
- Поддержка malloc(), free()
- Отладка приложений использующих несколько GPU в cuda-gdb и Parallel Nsight

NVIDIA Confidential

CUDA 3.2

- Нет нормального API для работы с multi-GPU
  - Сложное взаимодействие между несколькими GPU
- Неэффективное копирование данных между GPU
  - Необходимо копировать через системную память
- Неполная поддержка C++
  - Виртуальные методы и т.п.

NVIDIA Confidential

CUDA 4.0 – multi-threading

- Каждый GPU может использоваться несколькими CPU потоками
  - Быстрое портирование многопоточных приложений
  - Одновременный запуск ядер с разных CPU потоков
    - Необходимо запускать в разных CUDA stream
  - Новый API для работы с контекстами
    - Один контекст устройства на весь процесс

NVIDIA Confidential

CUDA 4.0 – multi-threading

- **Пример использования одного GPU несколькими CPU потоками**

CPU поток A	CPU поток B
`cudaSetDevice(0);`	`cudaSetDevice(0);`
`cudaMalloc(&p0, size);`	
`pthread_barrier_wait(b);` ⟷	`pthread_barrier_wait(b);`
	`K0<<<1,1>>>(p0);`
	`cudaFree(p0);`

NVIDIA Confidential

CUDA 4.0 – multi-GPU

- Один CPU поток имеет доступ ко всем GPU
  - (до CUDA 4.0 было ограничение на 1 GPU)
  - Простое использование нескольких GPU
  - Координация работы между несколькими GPU
    - cudaStreamWaitEvent() – синхронизация между GPU
    - cudaDeviceSynchronize() – вместо cudaThreadSynchronize()

NVIDIA Confidential

CUDA 4.0 – multi-GPU

- **Пример доступа из одного CPU потока к двум GPU**

```
cudaSetDevice(0);          // активное устройство #0
cudaMalloc(&p0, size);  // выделяем память для p0 на #0
K0<<<1,1>>>(p0);         // вызов ядра K0 на #0

cudaSetDevice(1);        // переключаемся на #1 (невозможно в CUDA 3.x!)
cudaMalloc(&p1, size);  // выделяем память для p1 на #1
K1<<<1,1>>>(p1);         // вызов ядра K1 на #1
```

NVIDIA Confidential

CUDA 4.0 – pinned память

- **Улучшение работы с pinned памятью**

CUDA 3.2	CUDA 4.0
malloc(a)	
cudaMallocHost(b)	
memcpy(b, a)	**cudaHostRegister**(a)
работаем с a	
memcpy(a, b)	**cudaHostUnregister**(a)
cudaFreeHost(b)	

NVIDIA Confidential

CUDA 4.0 – поддержка C/C++

- **Операторы new/delete**
  - Вызов конструкторов
- **Виртуальные функций**
  - Поддержка полиморфизма
- **Вставка PTX**
  - Оптимизации на уровне ассемблера

NVIDIA Confidential

Библиотека Thrust

- Стандартная библиотека шаблонов для CUDA
  - Аналог C++ библиотеки шаблонов STL
- Выбор алгоритма на этапе компиляции
  - Распределение работы между доступными CPU и GPU

Структуры данных	Алгоритмы
thrust::device_vector	thrust::sort
thrust::host_vector	thrust:reduce
thrust::device_ptr	thrust::exclusive_scan

NVIDIA Confidential

Пример Thrust

```
#include <thrust/host_vector.h>
...

int main(void)
{
    // создаем 32M случайных чисел на CPU
    thrust::host_vector<int> h_vec(32 << 20);
    thrust::generate(h_vec.begin(), h_vec.end(), rand);

    // копируем данные на GPU
    thrust::device_vector<int> d_vec = h_vec;

    // сортируем массив на GPU
    thrust::sort(d_vec.begin(), d_vec.end());

    // копируем данные обратно на CPU
    thrust::copy(d_vec.begin(), d_vec.end(), h_vec.begin());

    return 0;
}
```

NVIDIA Confidential

Библиотека CUSPARSE

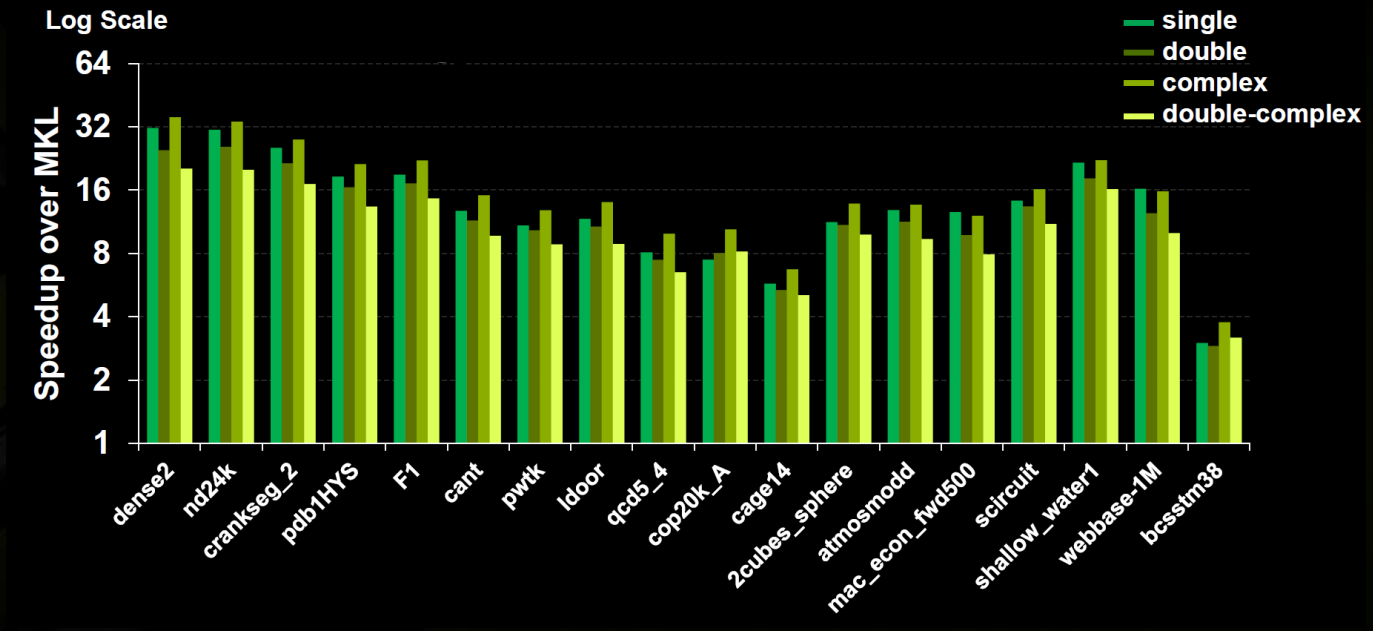
- Новая библиотека для работы с разреженными матрицами
- Преобразования различных форматов COO, CSR, CSC
- Оптимизированные операции умножения матрицы на вектор

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \alpha \begin{bmatrix} 1.0 & & & \\ 2.0 & 3.0 & & \\ & & 4.0 & \\ 5.0 & & 6.0 & 7.0 \end{bmatrix} \begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 4.0 \end{bmatrix} + \beta \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

NVIDIA Confidential

Сравнение с MKL

Умножение разреженной матрицы на несколько полных векторов

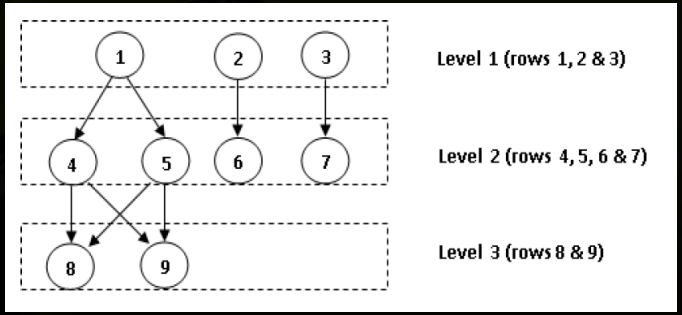


CUSPARSE 3.2, NVIDIA C2050 (Fermi), ECC on
MKL 10.2.3, 4-core Core i7 @ 3.07GHz

NVIDIA Confidential

Решение разреженных систем

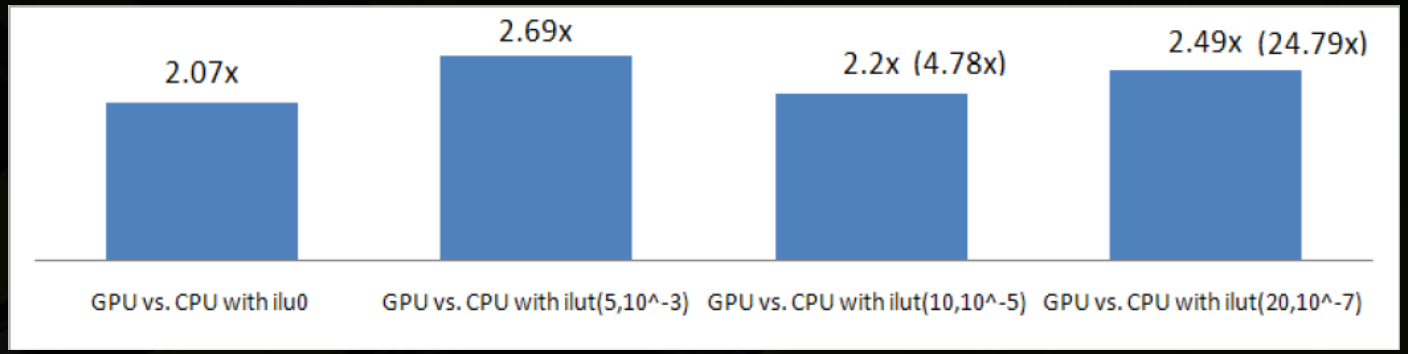
- Решение разреженной треугольной матрицы с помощью CUSPARSE
  - Фаза анализа – построение графа зависимостей
    - `cusparse*csrsv_analysis()`
  - Фаза решения – итерации по уровням
    - `cusparse*csrsv_solve()`



NVIDIA Confidential

Сравнение с MKL

- **Методы Incomplete- LU, Cholesky достигают порядка 2x ускорения на GPU**



Среднее ускорение BiCGStab/CG на GPU (csrilu0)
по сравнению с CPU (4 варианта предобуславливателя)

CUSPARSE 4.0, NVIDIA C2050 (Fermi), ECC on
MKL 10.2.3, 4-core Core i7 @ 3.07GHz

NVIDIA Confidential

Другие библиотеки

- CUBLAS
  - Работа с полными матрицами
- CUSP
  - Решение СЛАУ с разреженными матрицами
  - Различные форматы, солверы CG и BiCGstab
  - http://code.google.com/p/cusp-library/

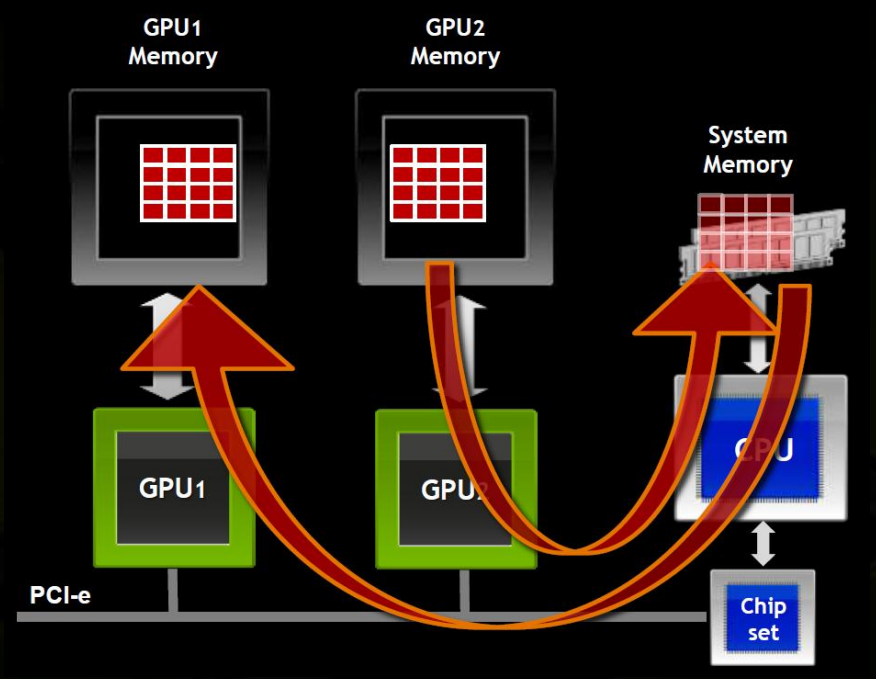
NVIDIA Confidential

Другие библиотеки

- CUFFT
  - Оптимизированная библиотека FFT
  - Поддержка SP/DP, real/complex, 1D/2D/3D, 2/3/5/7-radix
- MATLAB CUDA
  - Поддержка GPU в плагине Parallel Computing Toolbox

NVIDIA Confidential

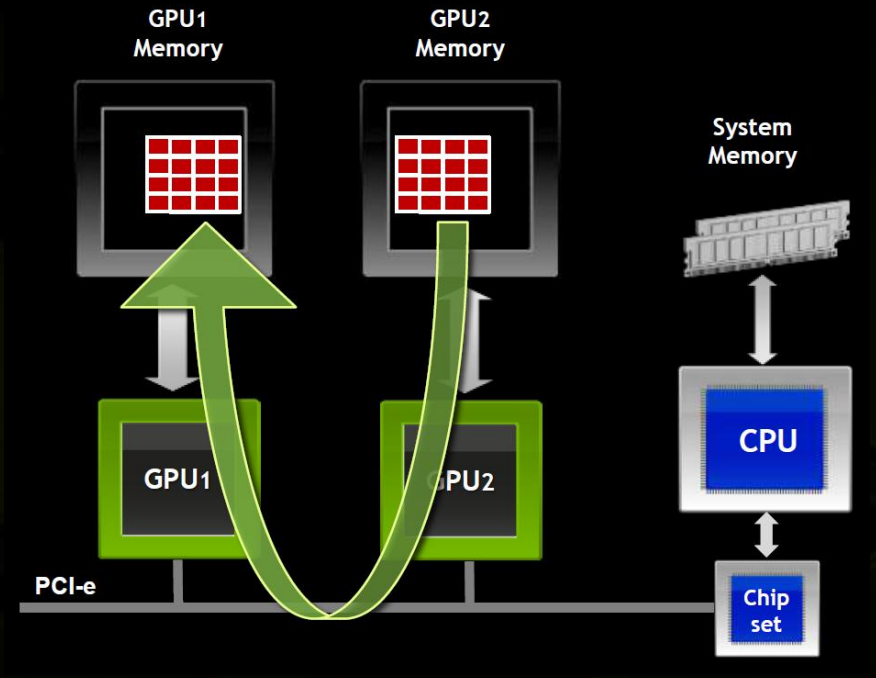
NVIDIA GPUDirect 1.0



- **Улучшение передачи данных внутри сети**
  - **Прямой доступ к памяти GPU для внешнего оборудования**
- **Для передачи данных между двумя GPU внутри узла необходимо копирование в системную память**

NVIDIA Confidential

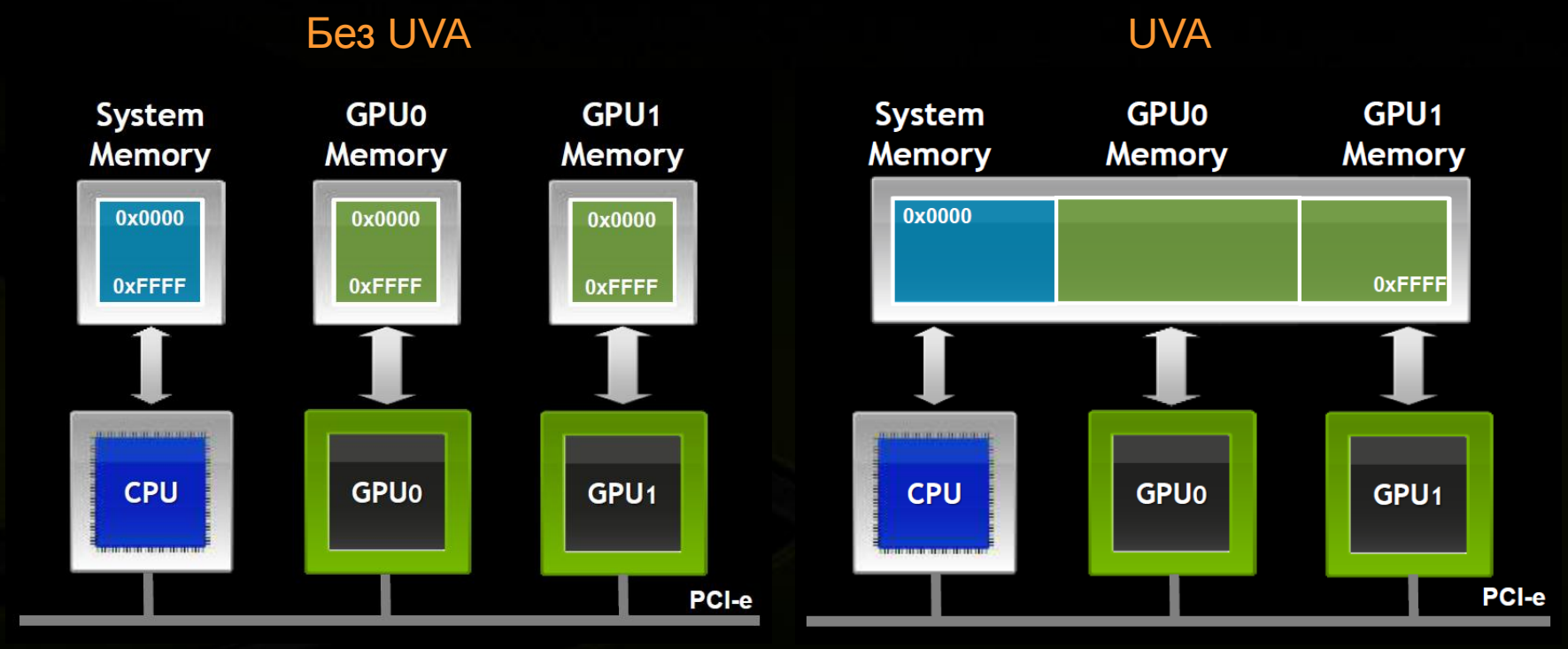
NVIDIA GPUDirect 2.0



- **Прямой доступ к памяти GPU**
  - GPU0 может читать/писать в память GPU1
  - Прямое копирование данных с GPU0 на GPU1
- **Только на Tesla 20xx**
  - x64 Linux или Win TCC

NVIDIA Confidential

Unified Virtual Addressing



NVIDIA Confidential

Unified Virtual Addressing

- Общее адресное пространство для всей памяти CPU и GPU
  - Физическое место определяется через указатель
  - Упрощение интерфейсов (cudaMemcpyDefault)
- Только на Tesla 20xx
  - x64 Linux или Win TCC

NVIDIA Confidential

CUDA 4.0

- Массив текстур (layered textures)
  - Набор нескольких текстур одного типа, размера и формата с фильтрацией только внутри слоя
- Поддержка 3D сеток на Fermi
- cuobjdump: поддержка ассемблера Fermi

NVIDIA Confidential

Вопросы?

NVIDIA Confidential

Полезные ссылки

- CUDA библиотеки:
  - http://developer.nvidia.com/technologies/libraries
- CUDA 4.0 webinar:
  - http://developer.download.nvidia.com/CUDA/training/cuda4_0_feature_intro.mp4
- Курс по CUDA на ВМК МГУ:
  - https://sites.google.com/site/cudacsmsusu/home
- Решение разреженных треугольных матриц
  - http://research.nvidia.com/sites/default/files/publications/nvr-2011-001.pdf

NVIDIA Confidential

Поддержка Fortran

- PGI CUDA Fortran
  - Триал версия доступна для бесплатного скачивания

NVIDIA Confidential