Архитектура и программирование потоковых многоядерных процессоров для научных расчётов

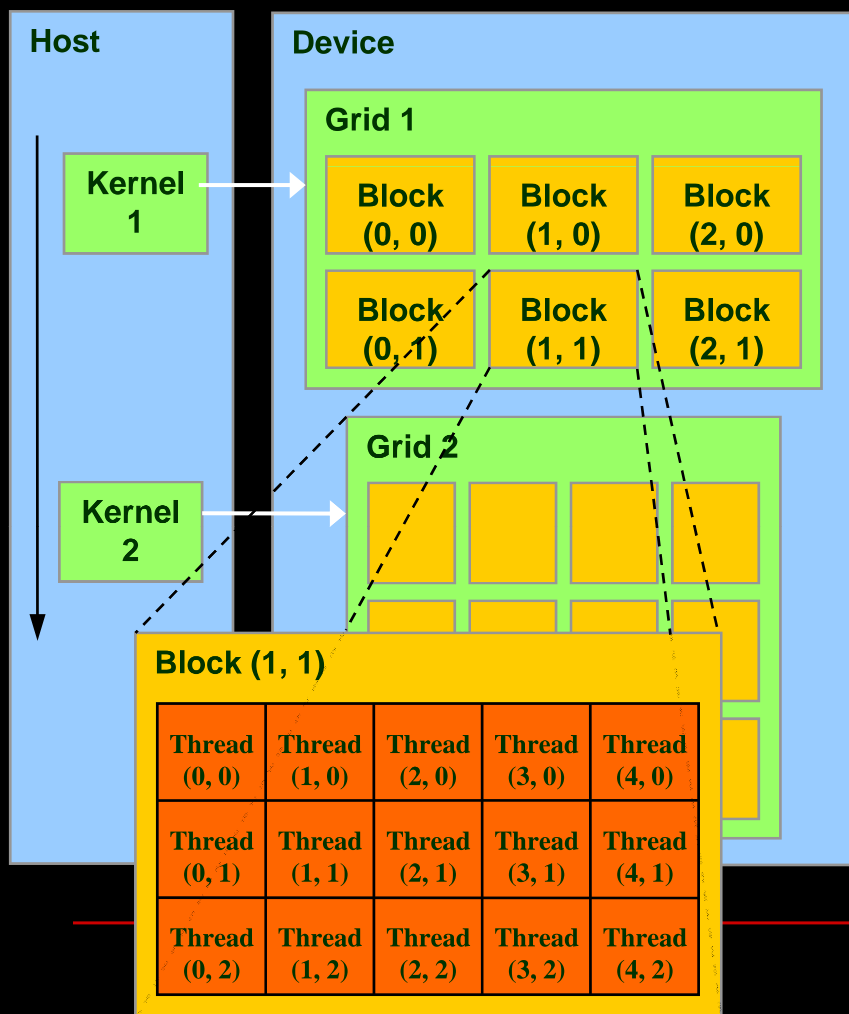
Лекция 5. Пример вычислительного ядра – задача умножения матриц. Текстуры. Атомарные функции. Библиотека CUTIL

NVIDIA®
World Leader in Visual Computing Technologies

Процедура разработки программы

- Общий подход к програмированию
  - Разбить задачу на элементарные блоки данных, над которыми выполняется стандартный алгоритм обработки (**единый для всех блоков**)
  - Разбить за-/вы- гружаемые данные на элементарные **непересекающиеся** блоки, которые каждый из процессов прочтёт/запишет
  - Определить **конфигурацию грида/блока**, позволяющее
    - Оптимальное **размещение промежуточных данных** в регистрах и общей памяти
    - Оптимальную **вычислительную загрузку** потоковых процессоров
  - Определить график **когерентного** обращения к памяти процессами при загрузке данных

Вычислительная конфигурация GPU

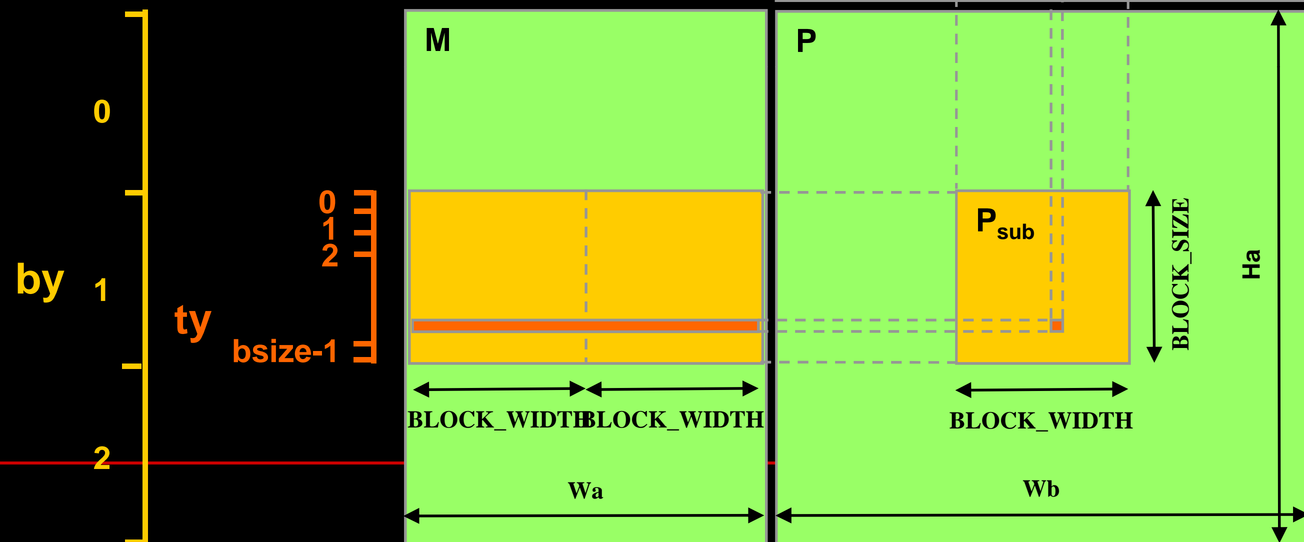


- Процессы объединяются в блоки (**blocks**), внутри которых они имеют общую память (**shared memory**) и синхронное исполнение
- Блоки объединяются в сетки (**grids**)
  - Нет возможности предсказать очерёдность запуска блоков в сетки
  - Между блоками нет и не может быть (см. выше) общей памяти

NVIDIA
World Leader in Visual Computing Technologies

Пример программы для GPU умножение матриц -1

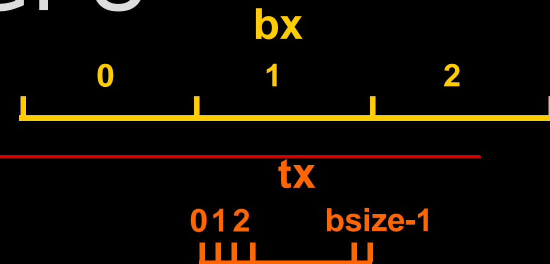
- **Разбить задачу на элементарные блоки данных, над которыми выполняется стандартный алгоритм обработки (единый для всех блоков)**
- Каждый блок вычисляет некоторую небольшую область результата
- Каждый тред в блоке вычисляет один элемент этой области

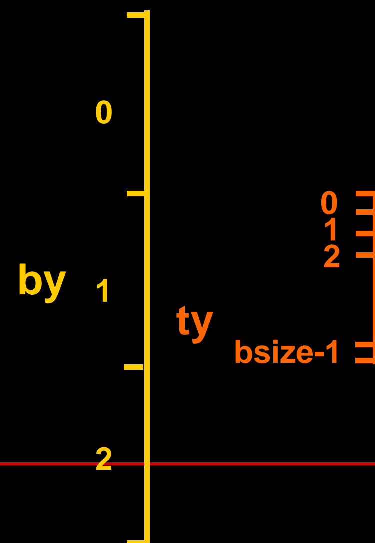


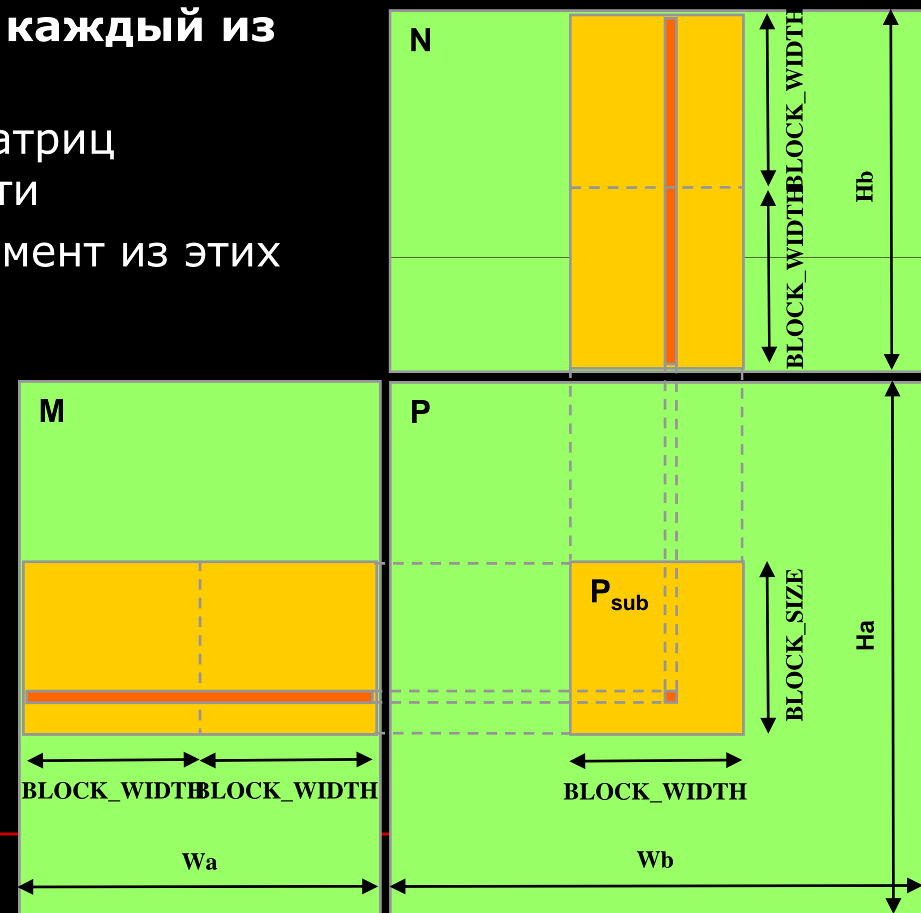
NVIDIA
World Leader in Visual Computing Technologies

Пример программы для GPU умножение матриц -2

- **Разбить данные на элементарные блоки (непересекающиеся), которые каждый из процессов прочтёт/запишет**
- Необходимые области исходных матриц разбиваются на квадратные области
- Каждый из тредов читает один элемент из этих областей в разделяемую память



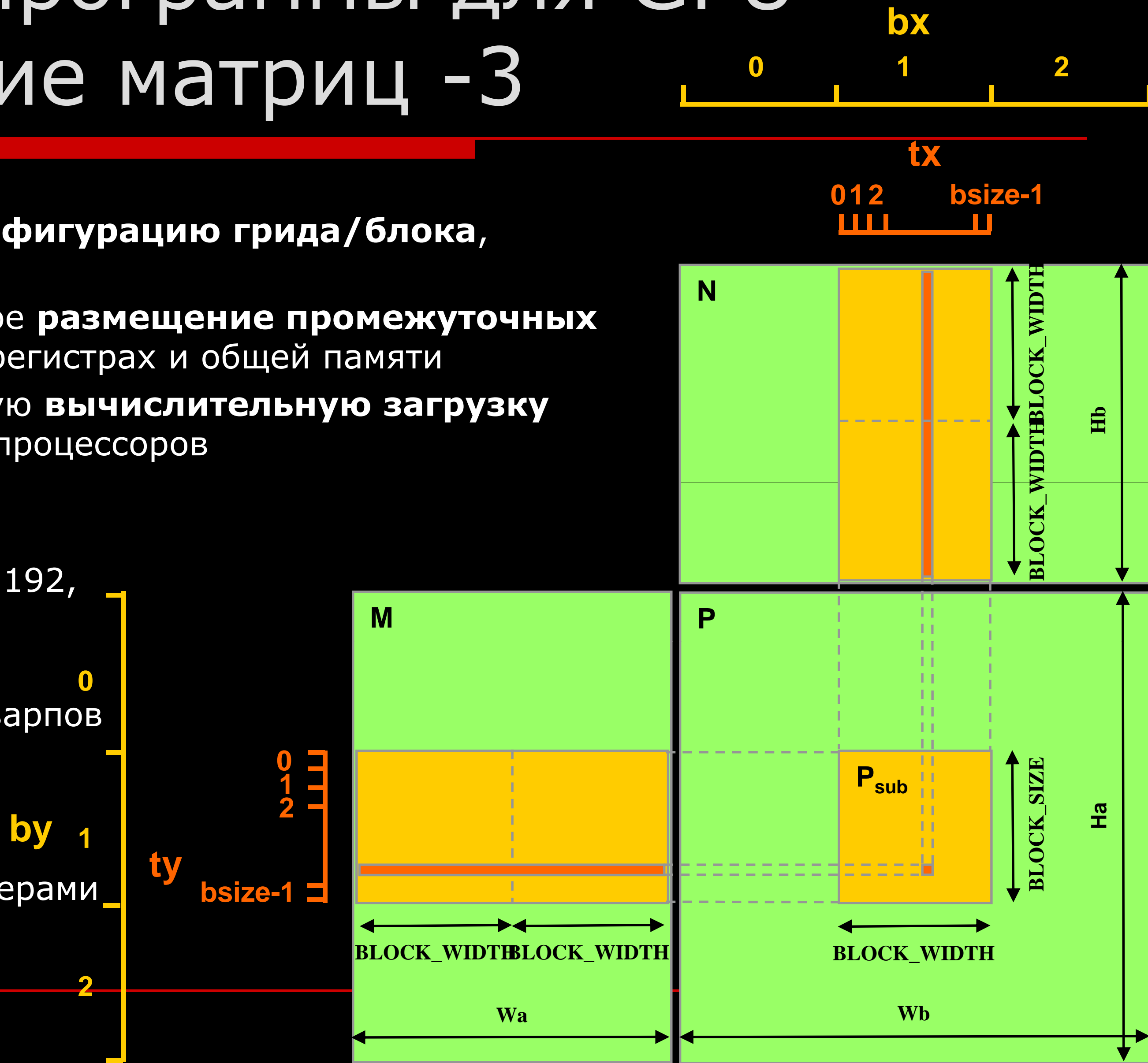




NVIDIA World Leader in Visual Computing Technologies

Пример программы для GPU умножение матриц -3

- Определить **конфигурацию грида/блока**, позволяющее
  - Оптимальное **размещение промежуточных данных** в регистрах и общей памяти
  - Оптимальную **вычислительную загрузку** потоковых процессоров
- Блок = 16x16= 256 тредов больше чем 192, меньше чем 512
- Целое количество варпов в блоке
- Размеры грида определяются размерами исходных матриц

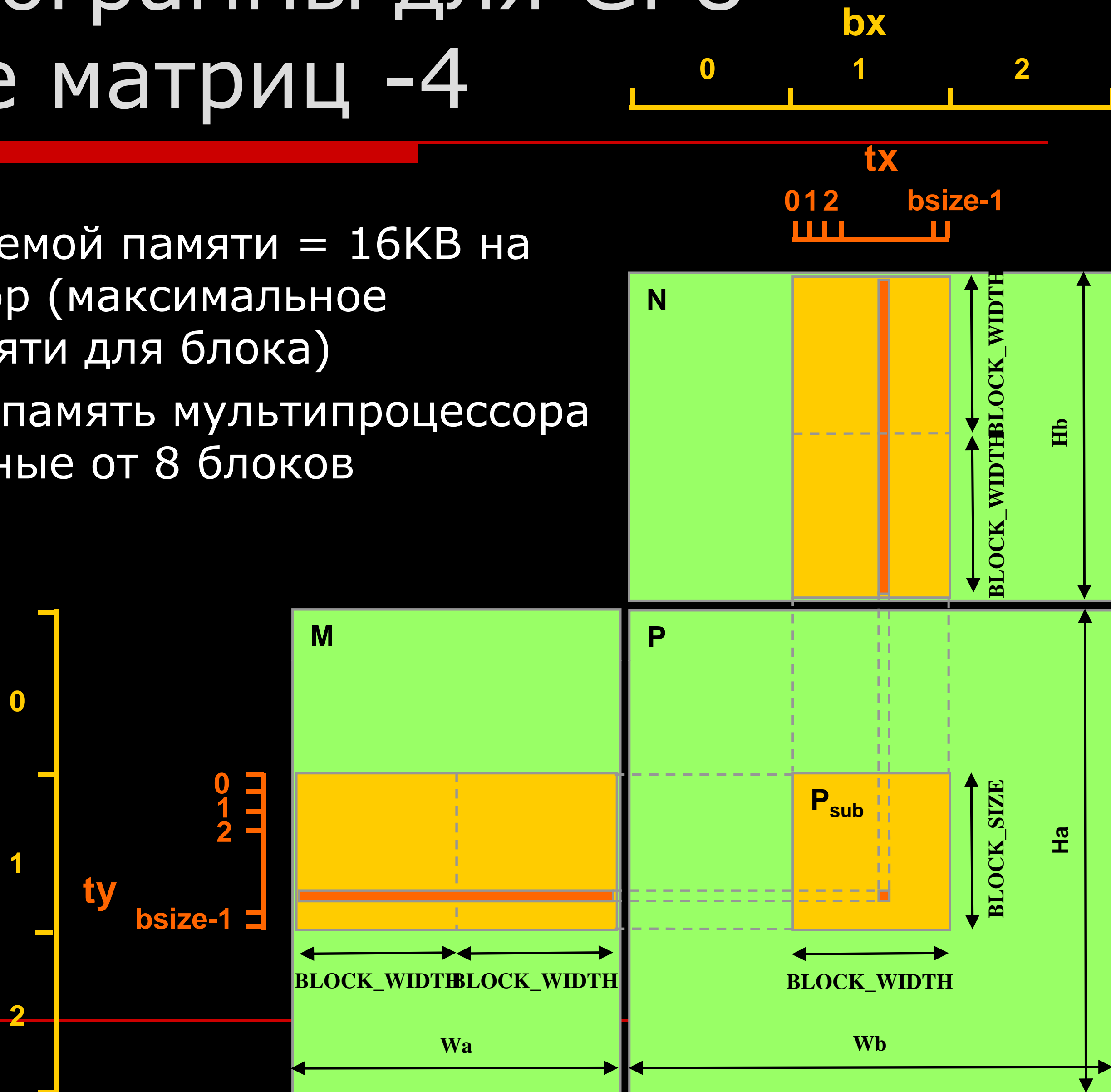


NVIDIA
World Leader in Visual Computing Technologies

Пример программы для GPU умножение матриц -4

- Размер разделяемой памяти = 16KB на мультипроцессор (максимальное количество памяти для блока)
- В разделяемую память мультипроцессора поместятся данные от 8 блоков

- Регистровая память расходуется незначительно
- Для определения количества регистров, приходящееся на один тред нужно смотреть PTX код



NVIDIA
World Leader in Visual Computing Technologies

Хост-функция умножения матриц

```
#define BLOCK_SIZE 16
__global__ void Muld(float*, float*, int, int, float*);
void Mul(const float* A, const float* B, int hA, int wA, int wB, float* C) {
int size;
// Load A and B to the device
float* Ad; size = hA * wA * sizeof(float); cudaMalloc((void**)&Ad, size);
cudaMemcpy(Ad, A, size, cudaMemcpyHostToDevice);
float* Bd; size = wA * wB * sizeof(float); cudaMalloc((void**)&Bd, size);
cudaMemcpy(Bd, B, size, cudaMemcpyHostToDevice);
// Allocate C on the device
float* Cd;
size = hA * wB * sizeof(float);
cudaMalloc((void**)&Cd, size);
// Compute the execution configuration assuming the matrix dimensions are multiples of BLOCK_SIZE
dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE);
dim3 dimGrid(wB / dimBlock.x, hA / dimBlock.y);
// Launch the device computation
Muld<<<dimGrid, dimBlock>>>(Ad, Bd, wA, wB, Cd);
// Read C from the device
cudaMemcpy(C, Cd, size, cudaMemcpyDeviceToHost);
// Free device memory
cudaFree(Ad); cudaFree(Bd); cudaFree(Cd); }
```

NVIDIA
World Leader in Visual Computing Technologies

GPU ядро умножения матриц

```
__global__ void Muld(float* A, float* B, int wA, int wB, float* C) {

int bx = blockIdx.x; // Block index
int by = blockIdx.y;
int tx = threadIdx.x; // Thread index
int ty = threadIdx.y;
int aBegin = wA * BLOCK_SIZE * by; // Index of the first sub-matrix of A processed by the block
int aEnd = aBegin + wA - 1; // Index of the last sub-matrix of A processed by the block
int aStep = BLOCK_SIZE; // Step size used to iterate through the sub-matrices of A
int bBegin = BLOCK_SIZE * bx; // Index of the first sub-matrix of B processed by the block
int bStep = BLOCK_SIZE * wB; // Step size used to iterate through the sub-matrices of B
float Csub = 0; // The element of the block sub-matrix that is computed by the thread

for (int a = aBegin, b = bBegin; a <= aEnd; a += aStep, b += bStep) {
        // Shared memory for the sub-matrix of A
        __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];

        // Shared memory for the sub-matrix of B
        __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];
```

NVIDIA
World Leader in Visual Computing Technologies

GPU ядро умножения матриц (продолжение)

```
As[ty][tx] = A[a + wA * ty + tx]; // Load the matrices from global memory to shared memory;
Bs[ty][tx] = B[b + wB * ty + tx]; // each thread loads one element of each matrix
__syncthreads(); // Synchronize to make sure the matrices are loaded

// Multiply the two matrices together;
// each thread computes one element
// of the block sub-matrix
for (int k = 0; k < BLOCK_SIZE; ++k)
        Csub += As[ty][k] * Bs[k][tx];

// Synchronize to make sure that the preceding
// computation is done before loading two new
// sub-matrices of A and B in the next iteration
__syncthreads();
}
// Write the block sub-matrix to global memory;
// each thread writes one element
int c = wB * BLOCK_SIZE * by + BLOCK_SIZE * bx;
C[c + wB * ty + tx] = Csub;
}
```

Уровень производительности

- Для матрицы 1024*1024
  - Автоматическое распределение : 3.0 сек - **45 GFLOPS**
  - 1 блок на SM : 3.6 сек. - **35 GFLOPS**
  - 2 блока на SM : 3.0 сек. - **45 GFLOPS**
  - Очевидно, ограничивающим фактором является требование запуска 2 блоков на SM, а не ограничение по разделяемой памяти (8 блоков на SM)
- Развёртывание циклов помогает (развёрнуто 16 раз)
  - Развернутая программа (автомат) : 1.6 сек. - **80 GFLOPS**
  - 1 блок на SM : 2.1 сек. - **61 GFLOPS**
  - 2 блока на SM : 1.6 сек. - **80 GFLOPS**

NVIDIA
World Leader in Visual Computing Technologies

Уровень производительности (продолжение)

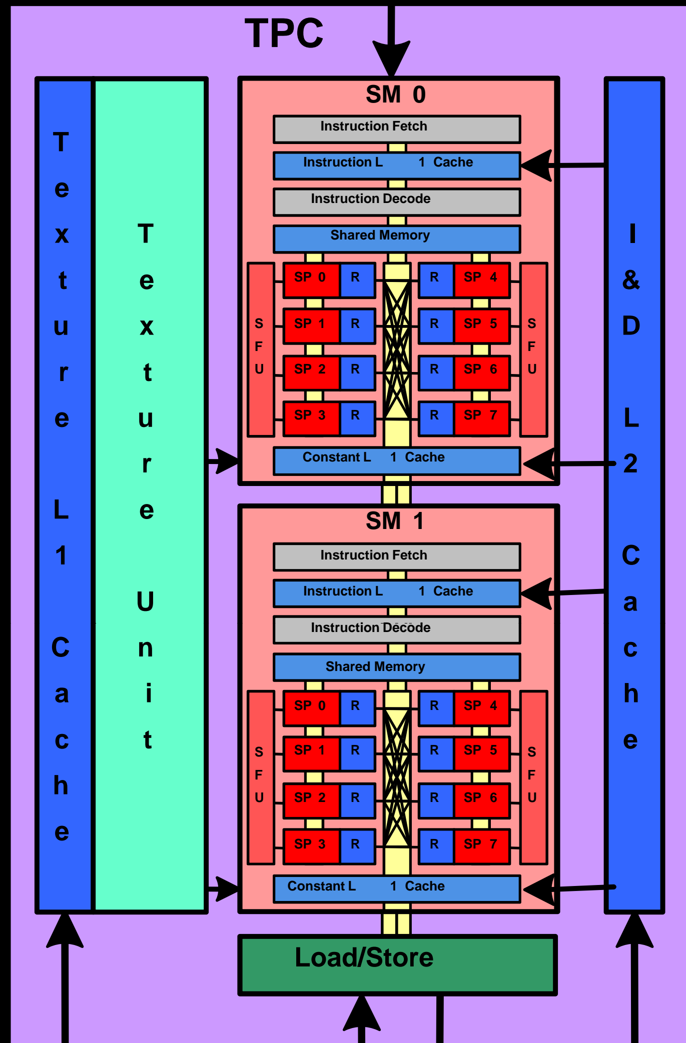
- Возможный альтернативный график вычислений

```
Loop {

Load to Shared Memory

Synchthread()

Compute current subblock

Syncthreads()
}
```

```
Load to Shared Memory
Syncthread()

Loop {

Compute current subblock
Load to Shard Memory

Syncthreads()
}
```

NVIDIA
World Leader in Visual Computing Technologies

Texture Processor Cluster (TPC)



- TEX – Текстурный блок – логика адресации текстурных массивов в 1D, 2D, 3D
  - + L1 Кэш Текстур
- L2 Кэш инструкций и данных для обоих SM
- x2 Потоковых Мультипроцессора (Streaming Multiprocessor)
- x8 потоковых процессоров (streaming processors) = 8 MAD/clock cycle
- Регистры для хранения промежуточных результатов у выполняемых тредов => больше тредов лучше скрыты операции чтения, но межет не хватить регистров

NVIDIA
World Leader in Visual Computing Technologies

Использование текстур (обзор)

- Текстуры - удобный способ обращения с табличными исходными данными
- Кэшированы – для каждых 2-х SM – общий L1 кэш, общий L2 кэш для всего кристалла
  - Замена разделяемой памяти для RO операндов
  - Нет жёстких требований к правильности адресации
- Адреса – числа с плавающей точкой (**возможно с нормализацией**)
  - Позволяют выделять объекты с адресом, находящимся между табличными значениями
    - Аппоксимация ближайшим
    - Линейная и билинейная аппроксимация (исходя из значений соседей)
- Варианты “**tile**” и “**center**”
  - Поведение текстуры при выходе адреса за пределы сетки

NVIDIA®
World Leader in Visual Computing Technologies

Использование текстур (детали)

- **Объявление** вне тела функции - texture<Type, Dim, ReadMode> texRef;
  - Type = тип
  - Dim = 1 | 2
  - ReadMode = **cudaReadModeNormalizedFloat** | **CudaReadModeElementType**

- **Определение** – привязывание объявленной ссылки на 1D (линейный массив) или 2D (CudaArray) объекту в глобальной памяти
  - cudaBindTexture()
  - cudaBindTextureToArray()
  - cudaUnbindTexture()

- **Использование** – "texture fetch"
  - 1D-массив
  - Texture_type tex1Dfetch(texture<uchar4, 1, cudaReadModeNormalizedFloat> texRef,int x);
  - 2D-массив
  - Texture_type tex1D(texture<Type, 1, readMode> texRef, float x);
  - Texture_type tex2D(texture<Type, 2, readMode> texRef, float x, float y);

Использование текстур (пример часть 1)

```
texture<float, 2, cudaReadModeElementType> tex; // declare texture reference for 2D float texture
__global__ void transformKernel( float* g_odata, int width, int height, float theta)
{
   // calculate normalized texture coordinates
   unsigned int x = blockIdx.x*blockDim.x + threadIdx.x;
   unsigned int y = blockIdx.y*blockDim.y + threadIdx.y;

   float u = x / (float) width;
   float v = y / (float) height;

   u -= 0.5f; // transform coordinates
   v -= 0.5f;
   float tu = u*cosf(theta) - v*sinf(theta) + 0.5f;
   float tv = v*cosf(theta) + u*sinf(theta) + 0.5f;

   // read from texture and write to global memory
   g_odata[y*width + x] = tex2D(tex, tu, tv);
}
```

NVIDIA®
World Leader in Visual Computing Technologies

Использование текстур (пример часть 2)

```
// set texture parameters
    tex.addressMode[0] = cudaAddressModeWrap;
    tex.addressMode[1] = cudaAddressModeWrap;
    tex.filterMode = cudaFilterModeLinear;
    tex.normalized = true;    // access with normalized texture coordinates

    // Bind the array to the texture
    CUDA_SAFE_CALL( cudaBindTextureToArray( tex, cu_array, channelDesc));

    dim3 dimBlock(8, 8, 1);
    dim3 dimGrid(width / dimBlock.x, height / dimBlock.y, 1);

    // warmup
    transformKernel<<< dimGrid, dimBlock, 0 >>>( d_data, width, height, angle);

    CUDA_SAFE_CALL( cudaThreadSynchronize() );

    // execute the kernel
    transformKernel<<< dimGrid, dimBlock, 0 >>>( d_data, width, height, angle);
```

NVIDIA
World Leader in Visual Computing Technologies

Обращение с памятью из ворпа

- НЕАТОМАРНЫЕ ИНСТРУКЦИИ (G80)
- ЕСЛИ какая-либо инструкция исполняемая ворпом **пишет** в одно место в **глобальной или общей** памяти
- ТО **количество** записей и их **очерёдность недетерминированы**
- ОДНАКО **по крайней мере одна запись** состоится

- АТОМАРНЫЕ ИНСТРУКЦИИ (G92+)
- ЕСЛИ какая-либо инструкция исполняемая ворпом **пишет/читает/модифицирует** одно место в **глобальной** памяти
- ТО их **очерёдность** записей **недетерминирована**
- ОДНАКО **все записи** состоятся **последовательно**

NVIDIA
World Leader in Visual Computing Technologies

Атомарные функции

- Функции чтения/модификации/записи данных в глобальную память – основа построения алгоритмов **стекового декодирования**
- Работают **только с целыми значениями (!)**
- **Гарантируют неизменность операнда в процессе операции**

  - Арифметические функции: **atomicAdd,atomicSub, atomicExch, atomicMax, atomicInc, atomicDec**
    - `int atomicAdd(int* address, int val);`
  - Функция **atomicCAS – Compare and store**
    - `int atomicCAS(int* address, int compare, int val);`
  - Битовые функции **atomicAnd, atomicOr, atomicXor**
    - `int atomicAnd(int* address, int val);`

Библиотека cutil

- Набор утилит для различных служебных целей (**cut*** - функции)
  - Макросы типа CUDA_SAFE_CALL - различные ситуции, с синхронизацией, без неё, для эмуляции
  - Утилиты профайлинга
    - измерение длительности исполнения
    - CUT_SAFE_CALL(cutCreateTimer(&timer));
    - CUT_SAFE_CALL(cutStartTimer(timer));
    - CUT_SAFE_CALL(cutStopTimer(timer));
    - CUT_SAFE_CALL(cutDeleteTimer(&timer));
    - Определение конфликтных ситуаций операций чтения
    - И т.д. => см **cutil.h**
- Находится в директории common в составе SDK
- В отличии от других библиотек предоставляется в виде исходных текстов
- Необходимо скомпилировать до работы над своим проектом

NVIDIA
World Leader in Visual Computing Technologies

Итоги лекции

- В результате лекции студенты должны :
  - Получить **практический пример** составления вычислительного ядра для типовой задачи.
  - Получить представление о свойствах **текстур** и возможности их применения в научных вычислениях
  - Получить представление о свойствах и возможности применения в научных вычислениях **атомарных функций**
  - **Достаточные знания для начала самостоятельной работы**

NVIDIA
World Leader in Visual Computing Technologies