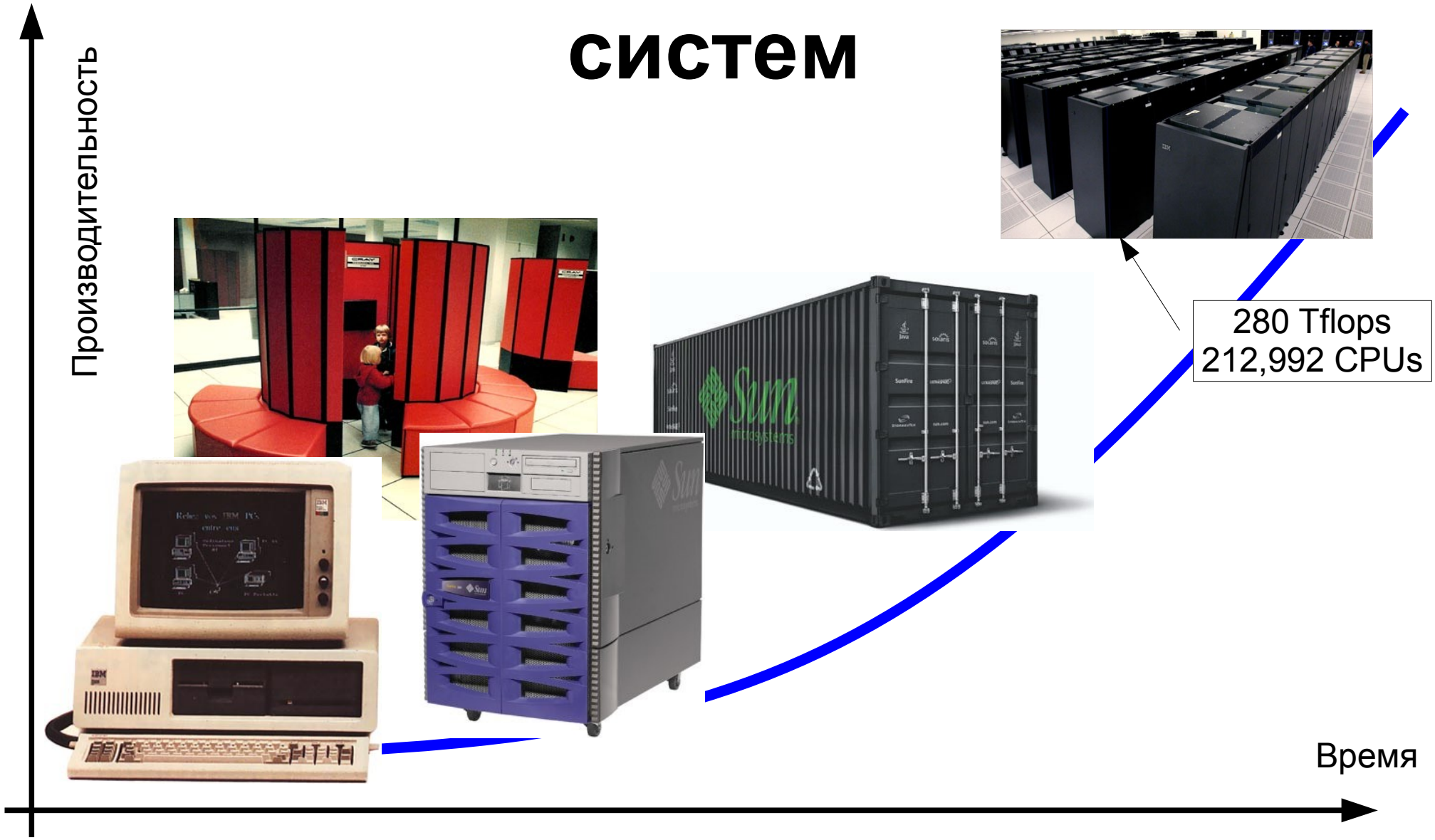
ВВЕДЕНИЕ В CUDA

Романенко А.А.
arom@ccfit.nsu.ru

Новосибирский государственный университет

Мощность вычислительных систем



Рост производительности

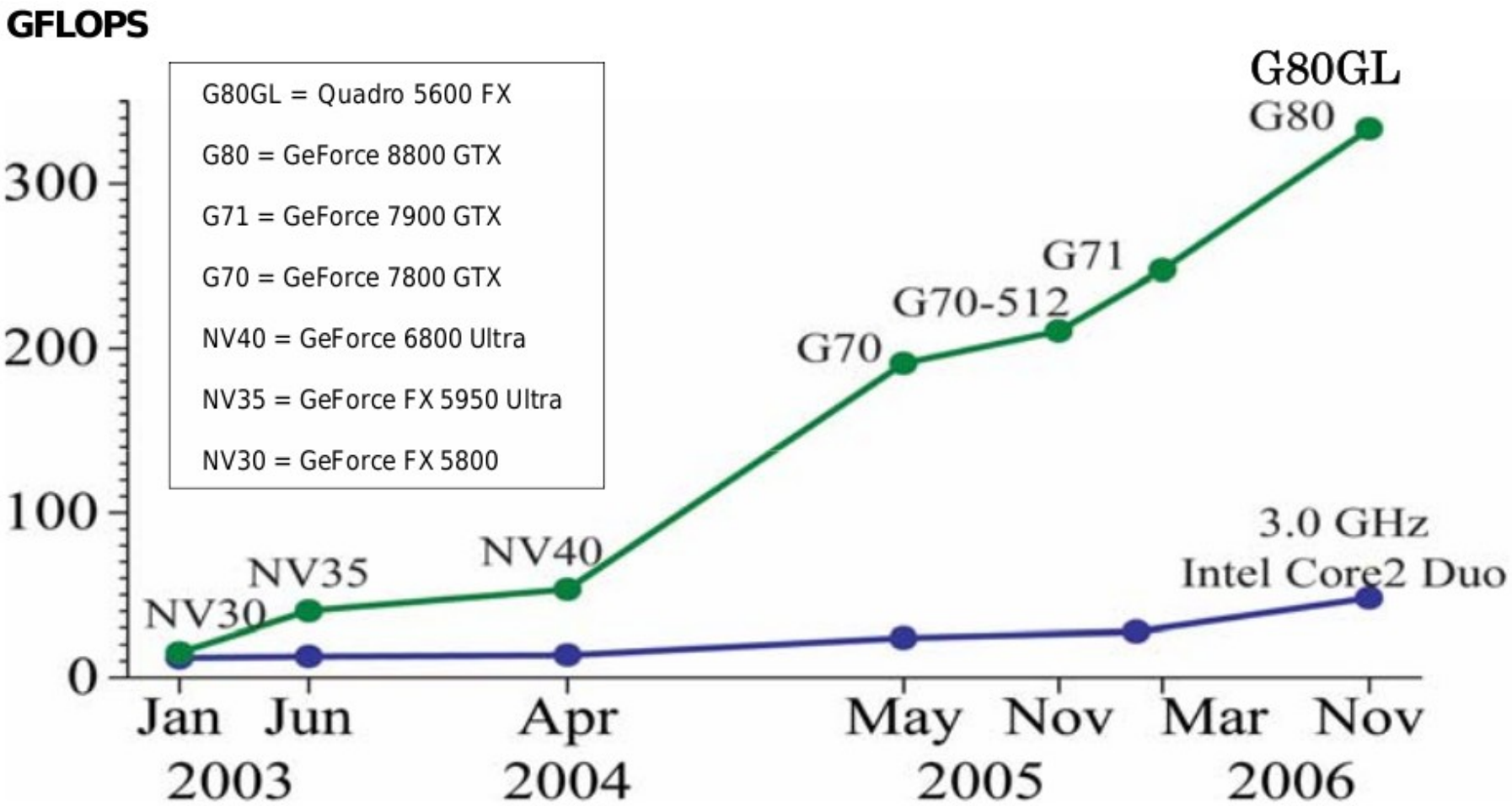
- За счет увеличения частоты процессоров
- За счет увеличения количества ядер/процессоров
- За счет усложнения архитектуры самих процессоров
  - Увеличение количества регистров
  - Изменение длинны конвейера
  - Увеличение разрядности
  - пр.

Обработка графики

- Работа с векторами
- Работа с маленькими матрицами
- Фильтры/post-processing
- Вычисление проекций
- пр.

- Однотипные операции над большим количеством данных

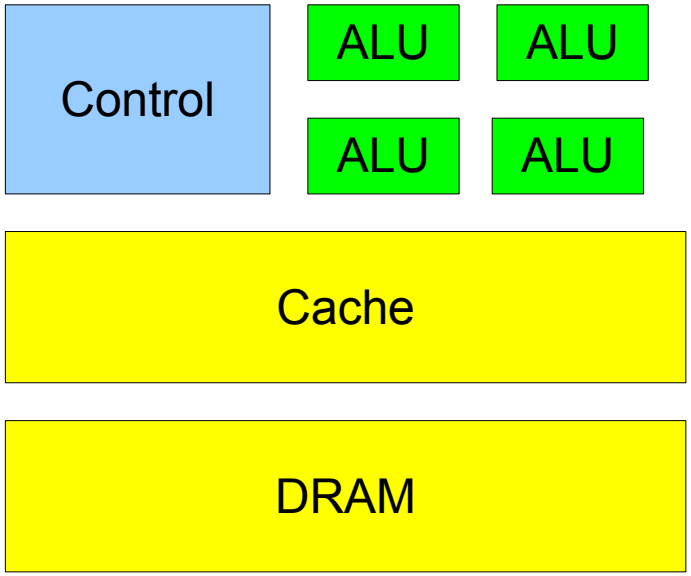
Производительность видеокарт



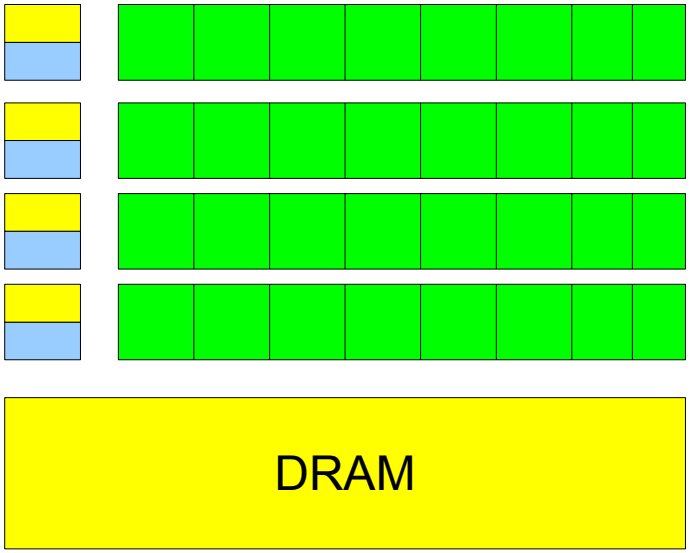
GPU — Graphical Processing Unit

- GPU — процессор на видеокарте. Имеет свою архитектуру
- Программа на GPU не может общаться с хостом
- Программа на GPU не может писать в память хоста
- Загрузка и выгрузка данных на видеокарту производятся через шину PCI Express 1 (2). Передача данных инициируется на стороне хоста
- Видеокарта - сопроцессор

CPU vs. GPU



CPU

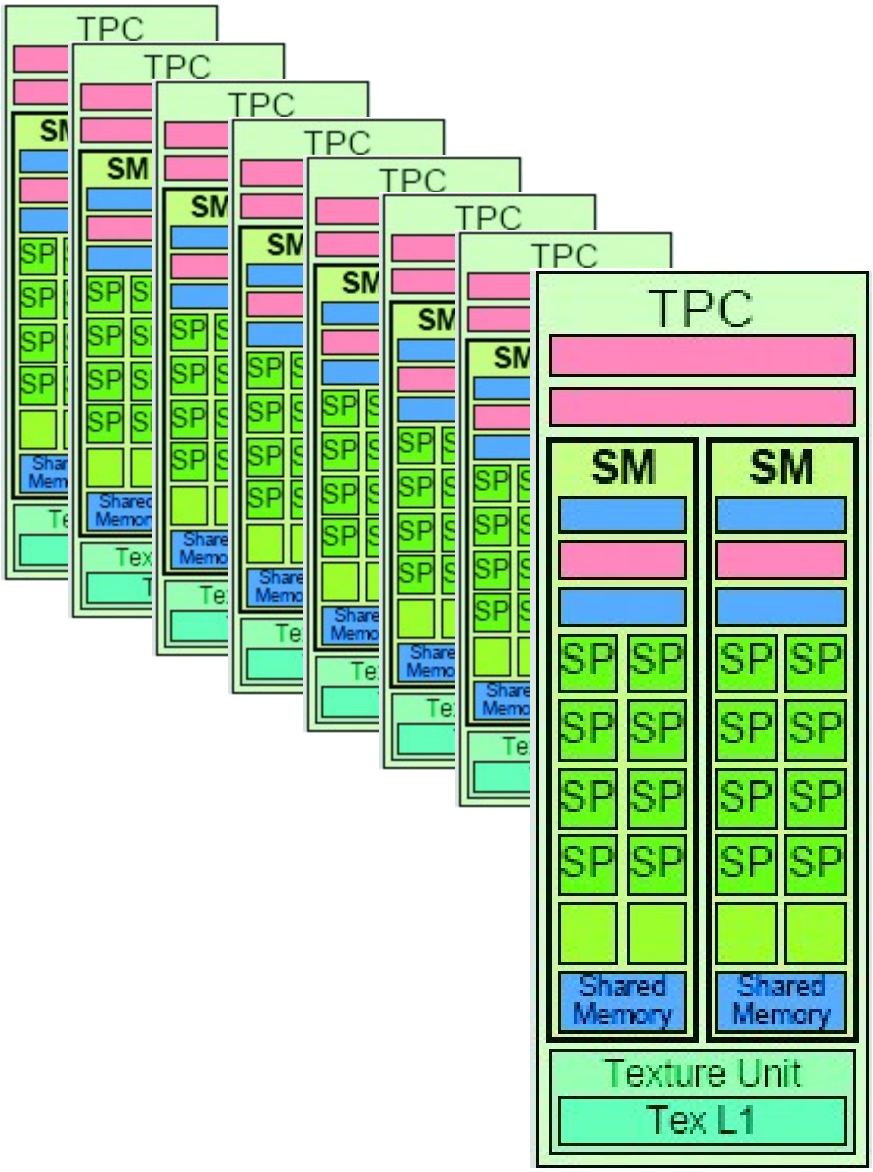


GPU

- Меньше транзисторов на управление и кэш
- Больше на АЛУ

Аппаратная архитектура GPU

Streaming Processor Array



- TPC - Texture Processor Cluster
- SM — Streaming Multiprocessor
  - Multi-threaded processor core
  - Fundamental processing unit for CUDA thread block
- SP — Streaming Processor
  - Scalar ALU for a single CUDA thread

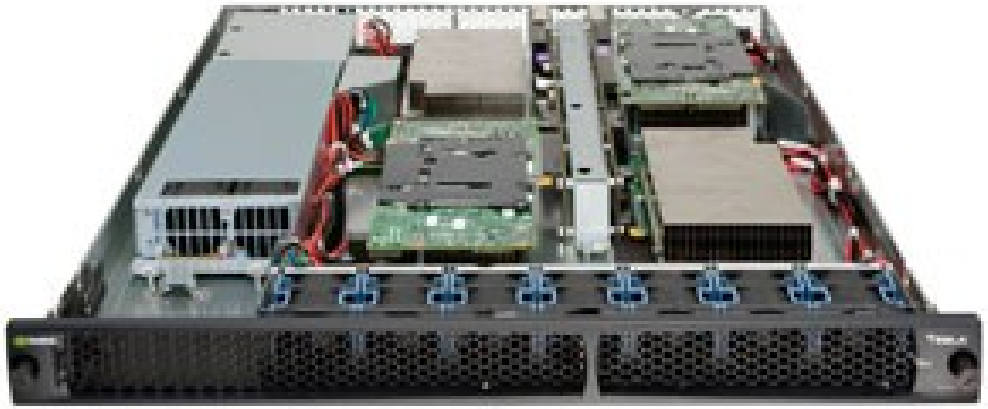
	Количество SM
GeForce 8800 GTX	16
GeForce 8800 GTS	12
Tesla D870	2x16
Tesla S870	4x16
Tesla C1060, GT200, Tesla T10	30
Tesla S1070	4x30

Tesla C1060

1 TFlops

Tesla S1070

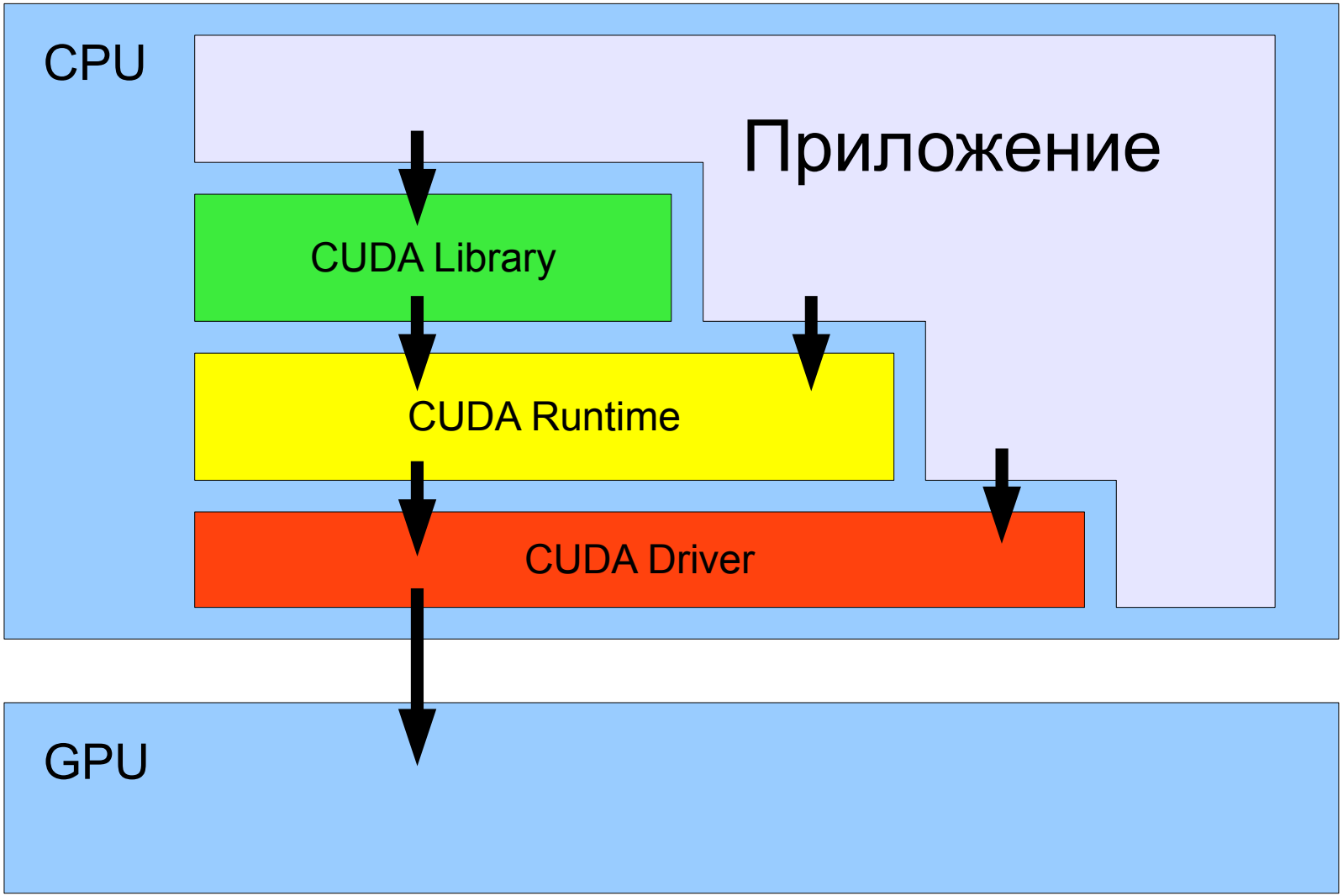
4 TFlops



Производительность для различных приложений

Example Applications	URL	Application Speedup
Seismic Database	http://www.headwave.com	66x to 100x
Mobile Phone Antenna Simulation	http://www.acceleware.com	45x
Molecular Dynamics	http://www.ks.uiuc.edu/Research/vmd	21x to 100x
Neuron Simulation	http://www.evolvedmachines.com	100x
MRI processing	http://bic-test.beckman.uiuc.edu	245x to 415x
Atmospheric Cloud Simulation	http://www.cs.clemson.edu/~jesteel/clouds.ht	50x

CUDA - Compute Unified Device Architecture



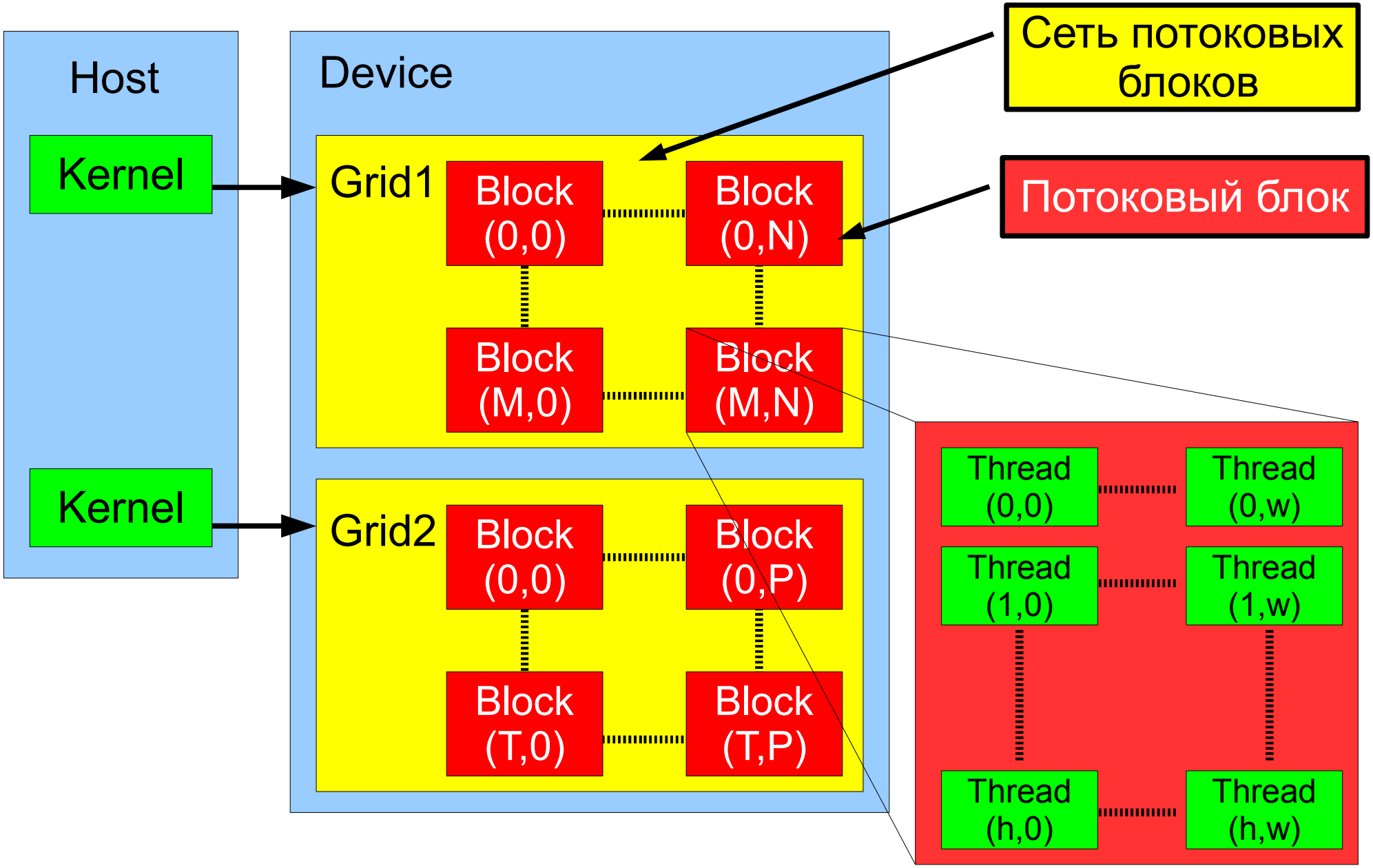
Термины

- **Поток** (Thread) — единица исполнения потока команд
- **Потоковый блок** (Thread blok) — группа связанных между собой потоков.
- **Варп** (Warp)— группа потоков внутри потокового блока, которая исполняется физически одновременно (32 потока)
- **Сеть** (Grid) — набор блоков, который должен быть обработан прежде чем исполнение программы пойдет дальше.

Программная модель

- GPU имеет свою память
- Программа в виде потоков выполняется на SP
- SP имеет доступ только к разделяемой памяти внутри своего SM и памяти GPU
- Синхронизация потоков возможна только внутри SM
- Исполнение организовано как сеть (GRID) потоковых блоков (thread block)
- Программа выполняемая потоком — ядро (kernel)

Запуск потоков



Пример

- Пусть при запуске задана двумерная сеть из блоков размером HxW и каждый блок содержит MxK потоков
- Таким образом область моделирования разбивается на
  - H*M потоков по вертикали
  - W*K потоков по горизонтали
- координаты потока в пространстве
  - (blockID.x*M + threadID.x, blockID.y*K + threadID.y)

Модель выполнения

- Блоки выполняются на Stream Multiprocessor
  - Один блок только на одном SM
  - Последовательность исполнения блоков не определена
- Количество блоков на SM определяется количеством регистров, требуемых потоку и количеством разделяемой памяти на блок
- Исполняемый в текущий момент поток называется **активным блоком**

Модель выполнения

- Каждый активный блок разбивается на SIMD группы потоков — варпы (**warps**). Каждый варп содержит одинаковое количество потоков. WarpSize = 32
- Планировщик потоков периодически передает управление от одного варпа к другому
- Распределение потоков по варпам всегда одинаковое

CUDA, компоненты

- Драйвер
  - /lib/modules/...
- Toolkit
  - /usr/local/cuda
- SDK
  - /usr/local/cudasdk

Сборка программы

- Компилятор — nvcc
- Исходные коды - *.cu или *.cuh
- Рекомендуется собирать с помощью **make**
  - Скопировать к себе из примеров Makefile и common.mk
  - Поправить пути для выходных файлов
- Сборка
  - **make emu=1** — в режиме эмуляции определен макрос __DEVICE_EMULATION__
  - **make dbg=1** — с отладочной информацией
  - **make** — финальной версии программы

Модификаторы функций

- **__device__**
  - Исполняется на GPU
  - Запускается только из GPU
- **__host__**
  - Исполняется на CPU
  - Запускается только с CPU
- **__global__**
  - Исполняется на GPU
  - Запускается только с CPU

Модификаторы функций Ограничения

- **__device__** и **__global__** не поддерживают рекурсию
- В теле **__device__** и **__global__** не должны объявлять статические переменные
- В **__device__** и **__global__** не может быть переменное число параметров
- **__global__** и **__host__** не могут использоваться одновременно
- **__global__** должна возвращать void и суммарный объем параметров должен быть не больше 256 байт

Модификаторы типов

- __device__
  - Располагается в глобальной памяти устройства
  - Имеет время жизни приложения
  - Доступна всем потокам в сети. Инициализируется через библиотечные функции на стороне CPU
- __constant__
  - Располагается в константной памяти устройства
  - Имеет время жизни приложения
  - Доступна всем потокам в сетии. Инициализируется через библиотечные функции на стороне CPU
- __shared__
  - Располагается в разделяемой памяти потокового блока
  - Имеет время жизни потокового блока
  - Доступна только потокам внутри потокового блока

Модификаторы типов Ограничения

- **__shared__** переменная не может быть инициализирована при объявлении
- **__constant__** переменная может быть инициализирована только со стороны CPU
- Область видимости переменных **__device__** и **__constant__** - файл
- **__shared__** и **__constant__** переменные неявно имеют статическое хранилище
- Модификаторы не могут применяться к полям типов **struct** и **union**

Конфигурация времени выполнения

- Определяется при запуске ядра (__global__ функции)
  - **__global__ void Func(float* data);**
  - **Func<<<Dg, Db, Ns, S>>>(data);**
- Dg — размер сети. Тип dim3
  - Dg.x, Dg.y задают размер. Dg.z не используется
- Db — размер блока. Тип dim3
  - Db.x * Db.y * Db.z — количество потоков в блоке
- Ns — размер дополнительной разделяемой памяти на блок. Тип size_t. Опциональный. По-умолчанию - 0
- S — номер потока. Тип cudaStream_t. Опциональный. По-умолчанию — 0.
- **Выполнение ядра асинхронно**

Встроенные переменные

- **gridDim** — размер сети. Тип dim3.
- **blockIdx** — индекс блока в сети. Тип uint3.
- **blockDim** — размерность блока. Тип dim3.
- **threadIdx** — индекс потока в блоке. Тип uint3.
- uint3 и dim3 - структуры из трех полей: x, y, z
- Встроенные переменные нельзя модифицировать
- Нельзя получить адрес встроенной переменной

Пример

```
__global__ void my_sum(float* a, float* b,
                                    float* c, int len){
    unsigned int index;
    index  = blockIdx.x *blockDim.x + threadIdx.x;
    if(index<len){
        c[index] = a[index] + b[index];
    }
}

dim3 GS(100);
dim3 BS(512);

my_sum<<<GS, BS>>>(a, b, c, 5000);
```

Встроенные векторные типы данных

- char1, char2, char3, char4
- uchar1, uchar2, uchar3, uchar4
- short1, short2, short3, short4
- ushort1, ushort2, ushort3, ushort4
- int1, int2, int3, int4
- uint1, uint2, uint3, uint4
- long1, long2, long3, long4
- ulong1, ulong2, ulong3, ulong4
- float1, float2, float3, float4
- Поля: x,y,z,w

Инициализация устройства

- **cudaGetDeviceCount** — количество устройств GPU
- **cudaGetDeviceProperties**(cudaDeviceProp*, uint) — получить параметры устройства
- **cudaSetDevice**(uint) — сделать устройство активным
  - Перед первым вызовом любого ядра или функции из runtime API
- Если требуется работать с несколькими устройствами, необходимо несколько потоков (threads) в программе

Параметры устройства

```
struct cudaDeviceProp{
  char   name[256];
  size_t totalGlobalMem;
  size_t sharedMemPerBlock;
  int    regsPerBlock;
  int    warpSize;
  size_t memPitch;
  int    maxThreadsPerBlock;
  int    maxThreadsDim[3];
  int    maxGridSize[3];
  int    clockRate;
  size_t totalConstMem;
  int    major;
  int    minor;
  size_t textureAlignment;
  int    deviceOverlap;
  int    multiProcessorCount;
  int    __cudaReserved[40];
};
```

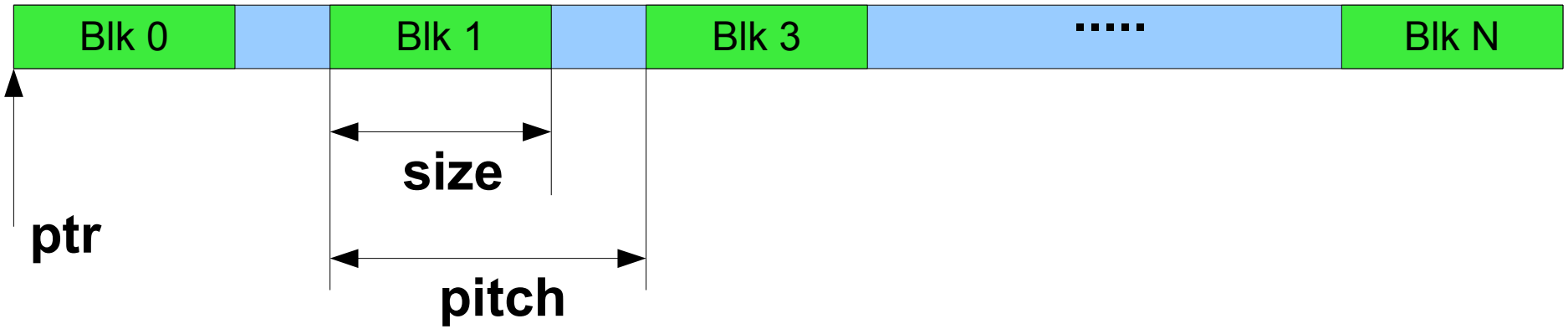
Выделение памяти

- CPU
  - malloc, calloc, free, cudaMallocHost, cudaFreeHost
- GPU
  - cudaMalloc, cudaMallocPitch, cudaFree,
  -

```
float* ptr;
cudaMalloc((void**)ptr, 256*sizeof(float));
....
cudaFree(ptr);
....
cudaMallocPitch((void**)ptr, &pitch,
                        256*sizeof(float), 256);
....
```

Выделение памяти

- **cudaMallocPitch((void**)ptr, &pitch, size, blocks);**



- **cudaMallocArray(struct cudaArray **array,**
  const struct cudaChannelFormatDesc* desc,
  size_t width, size_t height);
- **cudaFreeArray(struct cudaArray *array);**
- **cudaCreateChannelDesc(int x, int y, int z, int w, enum cudaChannelFormatKind f);**

Копирование данных в/из GPU

- cudaMemcpy(void* dst, void* src, size_t size, direction)
- direction:
  - cudaMemcpyHostToDevice
  - cudaMemcpyDeviceToHost
  - cudaMemcpyDeviceToDevice
- cudaMemcpy2D(void* dst, size_t dpitch, const void* src, size_t spitch, size_t width, size_t hight, direction)
- и т.д.

Атомарные операции

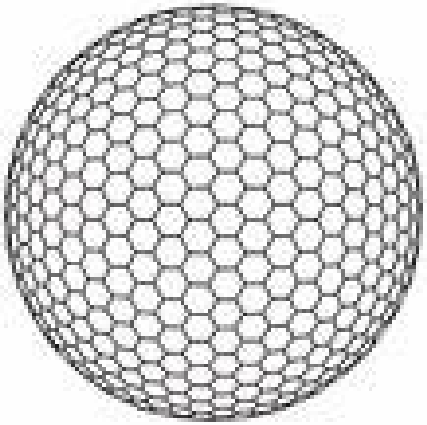
- Только знаковые и беззнаковые целые
- Операции над 32-битными словами в глобальной памяти
- atomicAdd, atomicSub, atomicExch, atomicMin, atomicInc, atomicDec, atomicCAS, atomicOr, atomicAnd, atomicXor

Математические функции

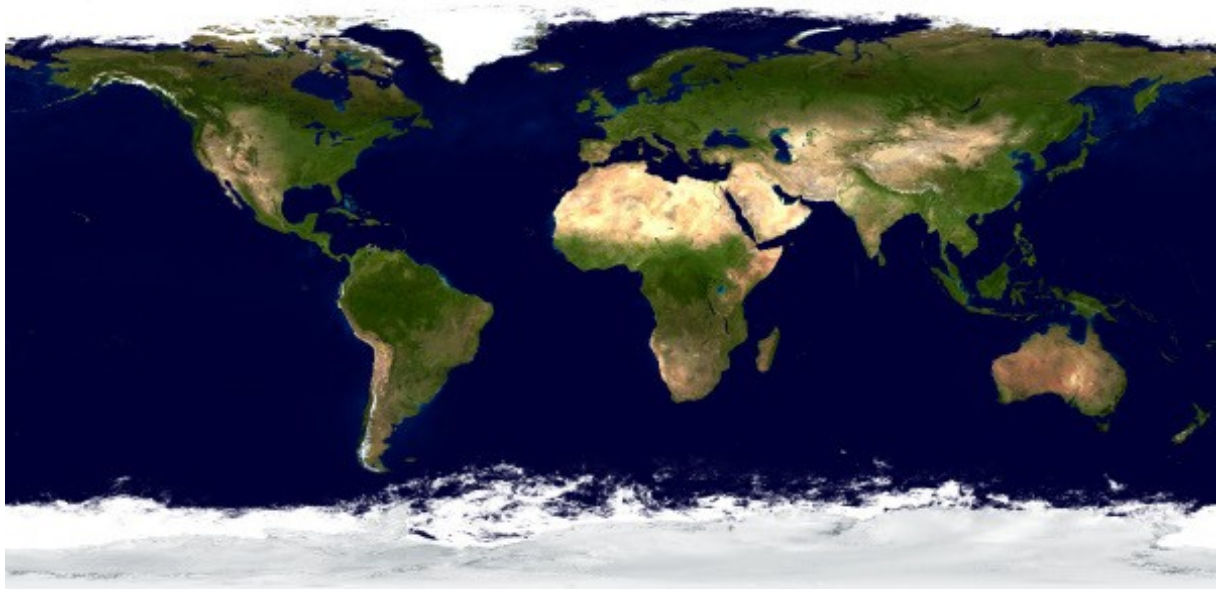
- Есть функции, которые исполняются как на GPU так и на CPU; есть те, которые выполняются только на GPU
- Вычисление может идти с погрешностью (см. документацию)
- Точность указана в ULP -**U**nit in the **L**ast **P**lace или **U**nit of **L**east **P**recision
- Время вычисления функций различно
- Существуют быстрые аналоги функций, но с ограничениями на диапазон/точность

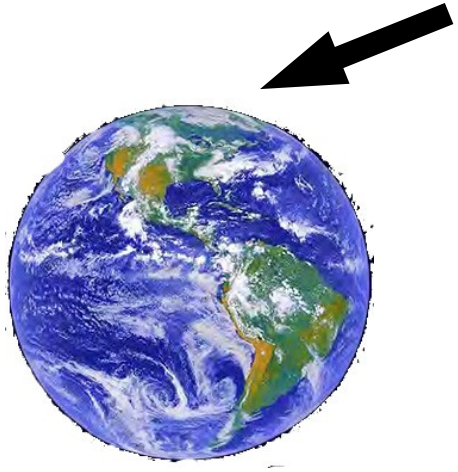
Что такое текстура?

- Способ доступа к данным



+





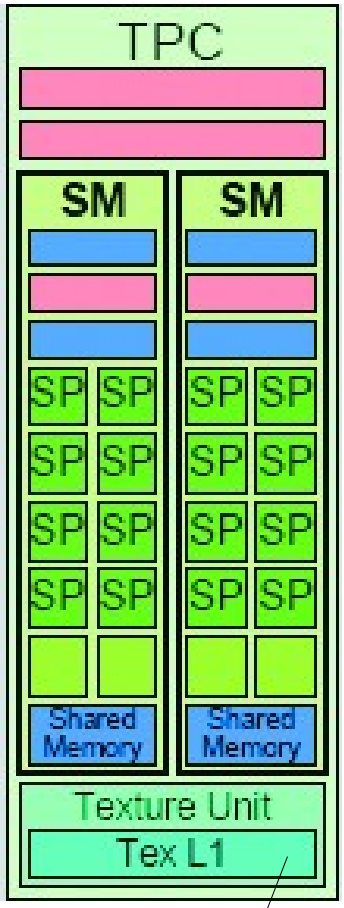
Особенности текстур

- Латентность больше, чем у прямого обращения в память
  - Дополнительные стадии в конвейере:
    - Преобразование адресов
    - Фильтрация
    - Преобразование данных
- Но зато есть кэш
- Разумно использовать, если:
  - Объем данных не влезает в shared память
  - Паттерн доступа хаотичный
  - Данные переиспользуются разными потоками

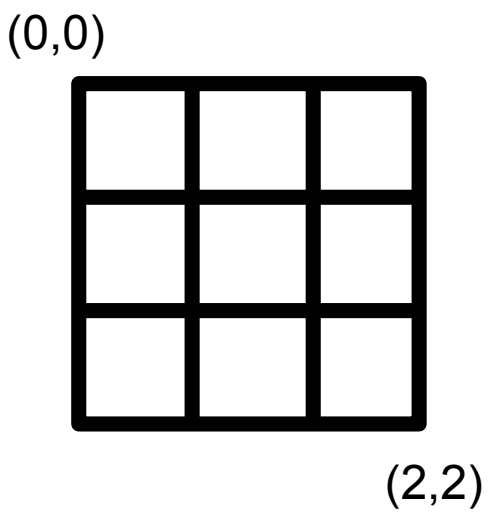
Свойства текстур

- Доступ к данным через кэш
  - Оптимизированы для доступа к данным которые расположены рядом в двумерном пространстве
- Фильтрация
  - Линейная/квадратичная/кубическая
- Свертывание (выход за границы)
  - Повторение/ближайшая граница
- Адресация в 1D, 2D и 3D
  - Целые/нормализованные координаты

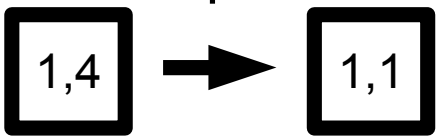
Свойства в картинках



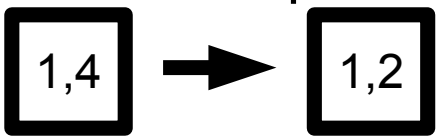
Кэш текстуры



Повторение



Ближайшая граница



- tex1Dfetch(texRef, x)
- tex1D(texRef, x)
- tex2D(texRef, x, y)
- tex3D(texRef, x, y, z)

Свойства в картинках

- Нормализация координат
- Фильтрация

Свойства текстур

- Преобразование данных:
  - **cudaReadModeNormalizedFloat** :
    - Исходный массив содержит данные в integer,
    - возвращаемое значение во **floating point** представлении (доступный диапазон значений отображается в интервал [0, 1] или [-1,1])
  - **cudaReadModeElementType**
    - Возвращаемое значение то же, что и во внутреннем представлении

Типы/свойства текстур

- Привязанные к линейной памяти
  - Только 1D
  - Целочисленная адресация
  - Фильтры и свертывание отсутствуют
- Привязанные к массивам CUDA
  - 1D, 2D или 3D
  - целые/нормализованные координаты
  - Фильтрация
  - Свертывание

Работа с текстурами

- Host:
  - Выделить память (malloc/cudaMallocArray/...)
  - Объявить указатель на текстуру
  - Связать указатель на текстуру с областью памяти
  - После использования:
    - Отвязать текстуру, освободить память
- Device:
  - Чтение данных через указатель текстуры
  - Текстуры для линейной памяти: tex1Dfetch()
  - Текстуры на массивах: tex1D() or tex2D() or tex3D()

Работа с текстурами (Host)

```
texture<float, 2, cudaReadModeElementType> tex;

  ...

cudaChannelFormatDesc channelDesc =
        cudaCreateChannelDesc(32, 0, 0, 0, cudaChannelFormatKindFloat);

cudaArray* cu_arr;

cudaMallocArray(&cu_arr, &channelDesc, width, height );
cudaMemcpyToArray(cu_arr, 0, 0, h_dta, size, cudaMemcpyHostToDevice);

// 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
  cudaBindTextureToArray(tex, cu_arr, channelDesc);
```

Работа с текстурами (Device)

```
__global__ void Kernel( 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;

    // transform coordinates
    u -= 0.5f;
    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);
}
```

CUDA Utilities library

- #include <cutil.h>
- Не является частью CUDA
- Назначение
  - Разбор командной строки
  - Чтение/запись бинарных файлов и изображений (PPM)
  - Сравнение массивов данных
  - Таймеры
  - Макросы проверки ошибок/инициализации
  - Проверка конфликтов банков разделяемой памяти

CUDA Utilities library

- CUT_DEVICE_INIT(ARGC, ARGV)
- CUT_EXIT(ARGC, ARGV)
- CUDA_SAFE_CALL(call) — в режиме отладки
- CUT_BANK_CHECKER(array, index) — в режиме эмуляции + отладки
- cutCreateTimer(unsigned int* name);
- cutDeleteTimer(unsigned int name);
- cutStartTimer(const unsigned int name);
- cutStopTimer(const unsigned int name);
- cutGetTimerValue(const unsigned int name);

Оптимизация программ на CUDA

Профилирование

- clock_t clock() - счетчик, который увеличивается с каждым тактом GPU
  - Разность значений в начале и конце ядра — количество тактов, затраченных GPU на выполнение потока. НЕ ПРОВЕДЕННОЕ в исполнении потока.
- Переменные окружения
  - CUDA_PROFILE=1
  - CUDA_PROFILE_LOG
  - CUDA_PROFILE_CONFIG

Профилирование. Файл конфигурации

- timestamp
- gridsize
- threadblocksize
- dynsmemperblock
- stasmemperblock
- regperthread
- memtransferdir
- memtransfersize
- streamid

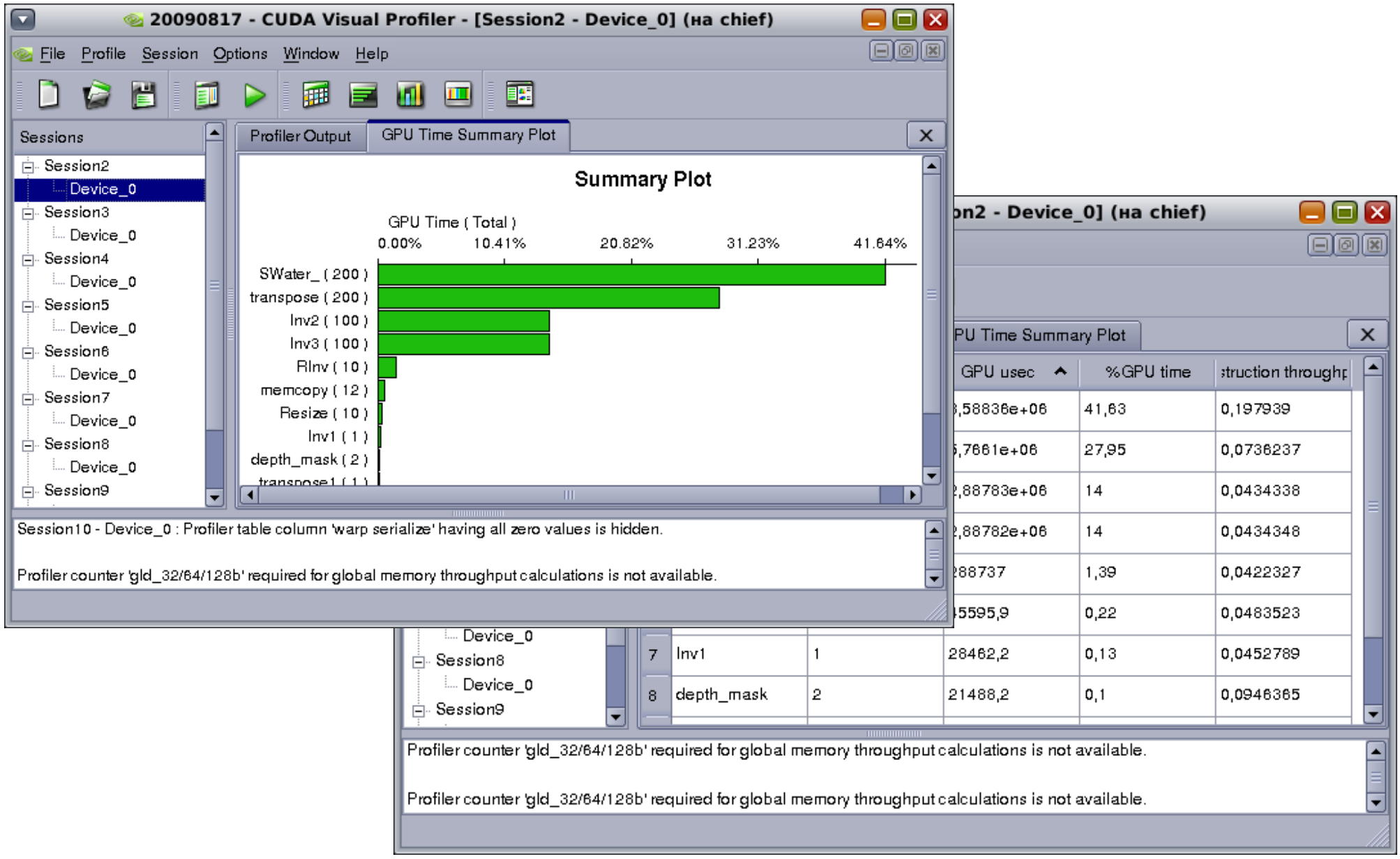
Профилирование. Файл конфигурации

- gld_incoherent
- gld_coherent
- gld_32b / gst_32b
- gld_64b / gst_62b
- gld_128b / gst_128b
- gld_request
- gst_incoherent
- gst_coherent
- gst_request
- local_load
- local_store
- branch
- divergent_branch
- instructions
- warp_serialize
- cta_launched
- gputime
- cputime
- occupancy

Профилирование. Вывод

```
method,gputime,cputime,occupancy
method=[ memcopy ] gputime=[ 5519.680 ]
method=[ memcopy ] gputime=[ 5516.992 ]
method=[ memcopy ] gputime=[ 5517.728 ]
method=[ memcopy ] gputime=[ 5508.288 ]
method=[ __globfunc__Z10depth_maskPfS_fii ] gputime=[ 1791.424 ] cputime=[ 14.000 ]
                                                                  occupancy=[ 0.812 ]
method=[ __globfunc__Z10depth_maskPfS_fii ] gputime=[ 1769.920 ] cputime=[ 2.000 ]
                                                                  occupancy=[ 0.812 ]
method=[ __globfunc__Z9transposePfS_ii ] gputime=[ 1598.336 ] cputime=[ 2.000 ] occupancy=[ 0.812 ]
method=[ __globfunc__Z12Invariants_XPfS_S_S_S_S_S_ii ] gputime=[ 5061.920 ] cputime=[ 3.000 ]
                                                                  occupancy=[ 0.812 ]
method=[ __globfunc__Z7SWater_PfS_S_S_S_S_fS_S_S_ii ] gputime=[ 10506.752 ] cputime=[ 2.000 ]
                                                                  occupancy=[ 0.406 ]
method=[ __globfunc__Z9transposePfS_ii ] gputime=[ 1599.776 ] cputime=[ 2.000 ] occupancy=[ 0.812 ]
method=[ __globfunc__Z9transposePfS_ii ] gputime=[ 1613.472 ] cputime=[ 1.000 ] occupancy=[ 0.812 ]
method=[ __globfunc__Z9transposePfS_ii ] gputime=[ 1609.056 ] cputime=[ 2.000 ] occupancy=[ 0.812 ]
method=[ __globfunc__Z7SWater_PfS_S_S_S_S_fS_S_S_ii ] gputime=[ 10758.784 ] cputime=[ 2.000 ]
                                                                  occupancy=[ 0.406 ]
method=[ __globfunc__Z13RInvariants_YPfS_S_S_S_S_S_ii ] gputime=[ 5423.456 ] cputime=[ 3.000 ]
                                                                  occupancy=[ 0.812 ]
```

Профилировщик cudaprof



Анализ отчета о профилировании

- Значение имеет не цифры, а их приращение
- Для ядер надо стремиться чтобы стремились к нулю несогласованное обращение к памяти (gld_incoherent, gst_incoherent)

Оптимизация. Инструменты

- Количество регистров на поток, разделяемой памяти на блок, локальной памяти на блок, сщтстантной памяти на блок
  - --ptxas-options=-v
- CUDA Occupancy calculator

CUDA Occupancy calculator

Just follow steps 1, 2, and 3 below! (or click here for help)

1.) Select Compute Capability (click):	1,2

2.) Enter your resource usage:	
Threads Per Block	256
Registers Per Thread	18
Shared Memory Per Block (bytes)	512

(Don't edit anything below this line)

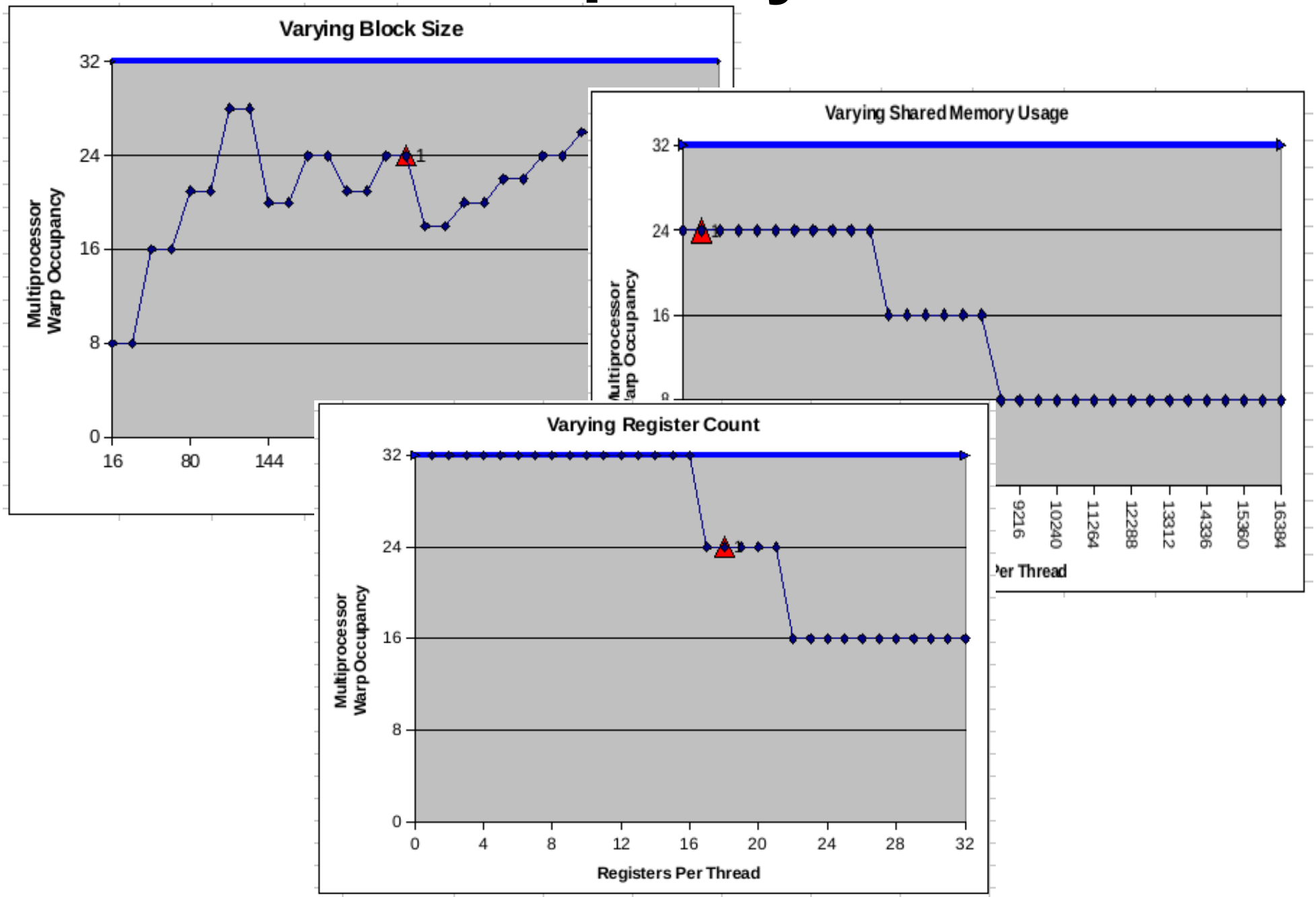
3.) GPU Occupancy Data is displayed here and in the graphs:	
Active Threads per Multiprocessor	768
Active Warps per Multiprocessor	24
Active Thread Blocks per Multiprocessor	3
Occupancy of each Multiprocessor	75%

Physical Limits for GPU:
Threads / Warp
Warps / Multiprocessor
Threads / Multiprocessor
Thread Blocks / Multiprocessor
Total # of 32-bit registers / Multiprocess
Shared Memory / Multiprocessor (bytes)

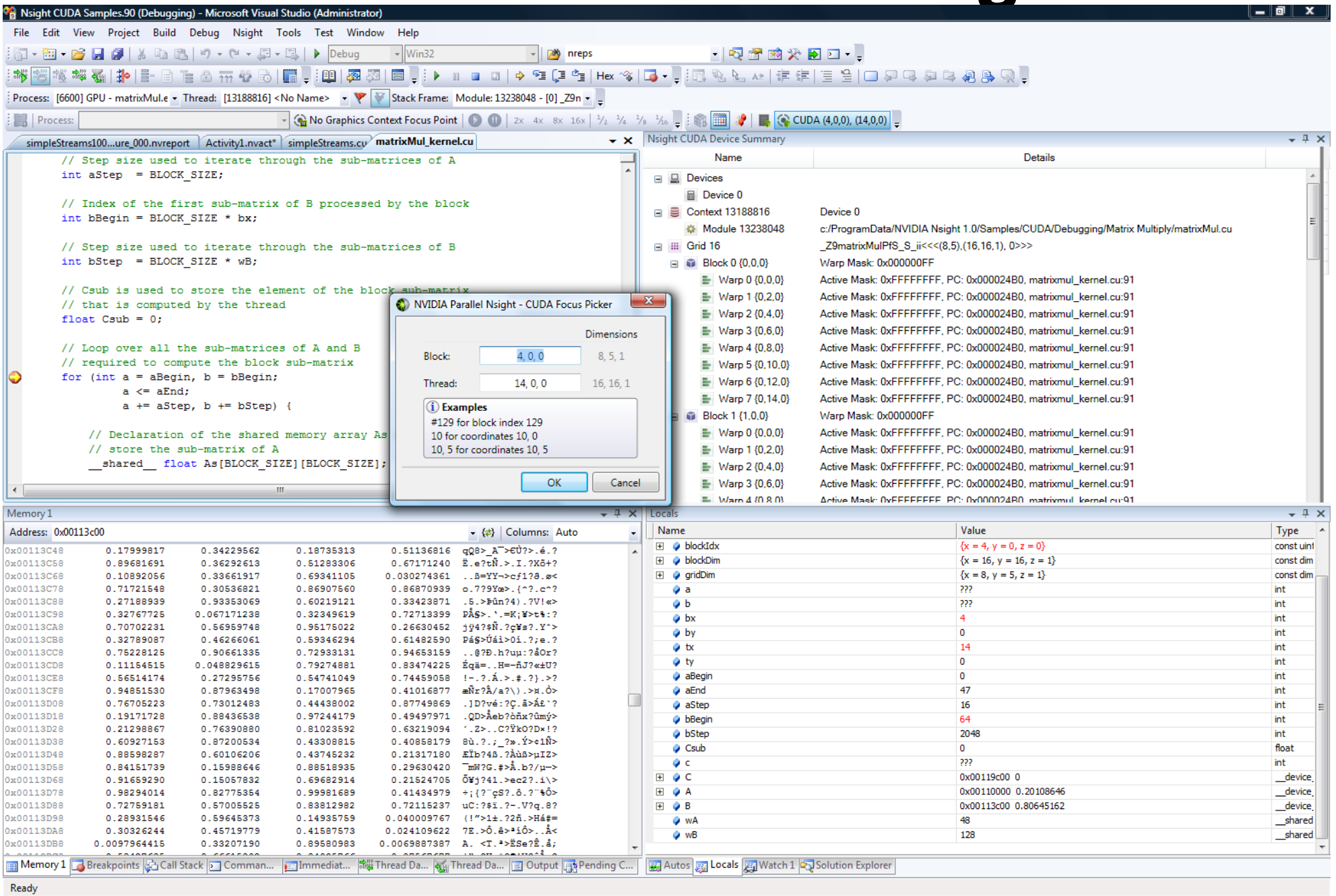
Allocation Per Thread Block
Warps
Registers
Shared Memory
These data are used in computing the o

Maximum Thread Blocks Per Multiprocessor	Blocks
Limited by Max Warps / Multiprocessor	4
Limited by Registers / Multiprocessor	3
Limited by Shared Memory / Multiprocessor	32
Thread Block Limit Per Multiprocessor highlighted	RED

CUDA Occupancy calculator



NVIDIA Parallel Nsight



Оптимизация

- Обработка инструкций
  - Чтение операндов
  - Выполнение инструкции
  - Сохранение результата
- Для оптимизации
  - Использовать более быстрые инструкции
  - Минимизировать задержки на обращение к памяти
  - По максимуму использовать пропускную способность шин данных

Исполнение инструкций

- Арифметические операции
  - 4 такта для FMUL, FADD, FMAD IADD, бинарные операции, сравнение, MIN, MAX, приведение типов
  - 16 тактов для __log, 1/sqrt, IMUL (1.x)
  - 32 такта для sqrt, __sin, __cos, __exp, FDIV
  - 20 тактов для __fdividef(x, y)

Условные переходы

- Если в ворпе есть две ветви исполнения (условный переход), то сначала исполняются потоки, которые проходят одну ветвь, затем потоки, которые проходят вторую.
- Минимизировать количество ветвлений. В частности внутри ворпа. Например за счет предвычислений.

Доступ к памяти

- 4 такта на обработку одной инструкции по работе с памятью (разделяемая, константная*, текстуры*).
- 400-600 тактов задержка по доступу к глобальной памяти
- Метод работы:
  - Загрузить данные из глобальной памяти в разделяемую (через текстуры)
  - Обработать данные
  - Выгрузить в глобальную

* - если нет промахов по кэшу

Доступ к памяти

- Используйте cudaMalloPitch вместо cudaMalloc, если двумерный массив
- Используйте cudaMallocArray для 2D и 3D массивов
- Используйте текстуры

Передача данных на/с устройства

- Скорость копирования на устройстве выше, чем между Хостом и Устройством
- Рекомендуется не выгружать данные, а запустить ядро с малым уровнем параллелизма, если это возможно.
- На Хосте выделять память с помощью cudaMallocHost()
- Совмещать передачу данных с вычислениями.

Литература

- http://www.nvidia.com/object/cuda_home.html
- http://steps3d.narod.ru/tutorials/cuda-tutorial.html
- http://steps3d.narod.ru/tutorials/cuda-2-tutorial.html