

# Программирование Multi-GPU систем

Дмитрий Микушин, NVIDIA

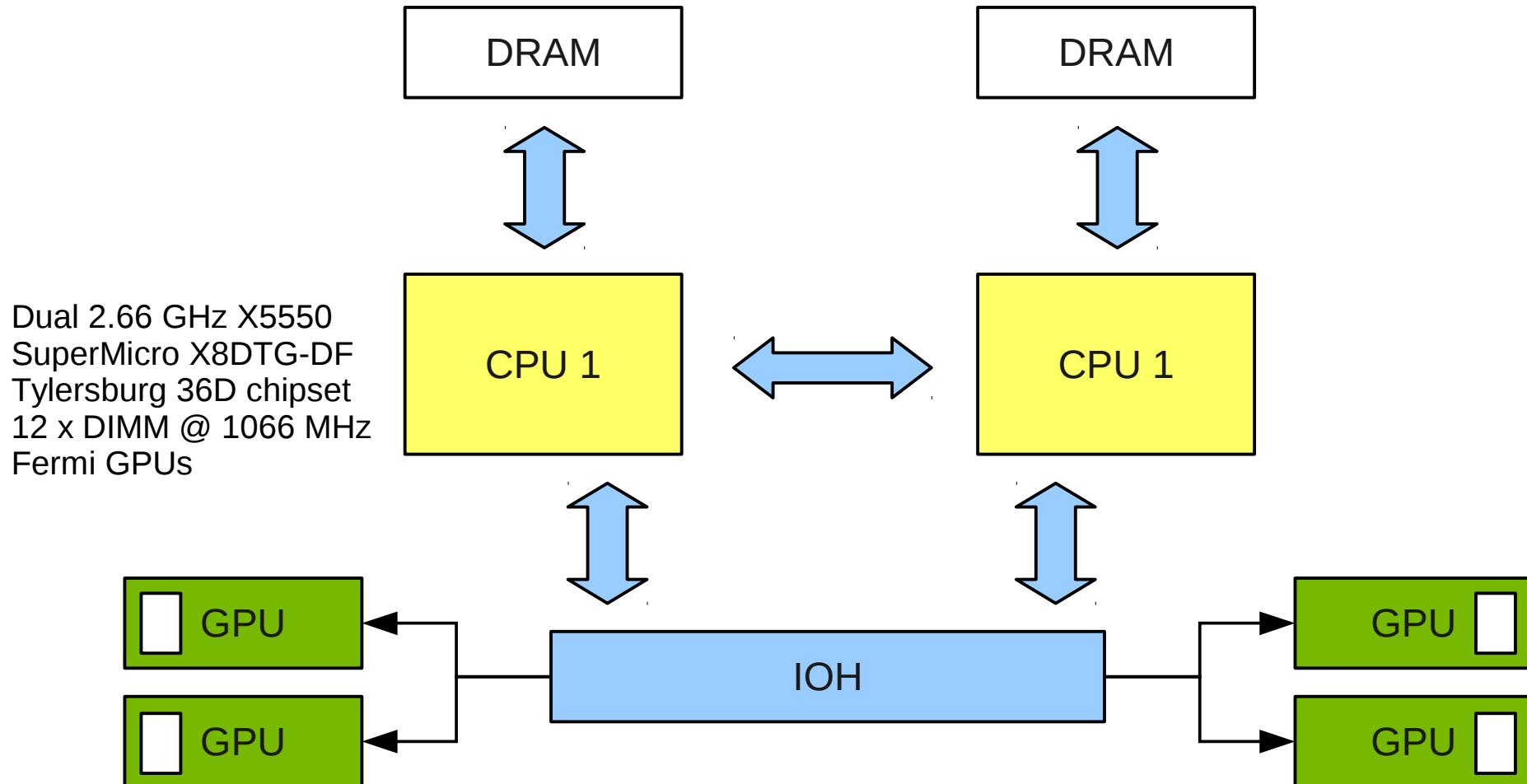
```
__global__ void kernel1(gpu1 const* in, int ex, int ey, float* out)
{
    // Compute absolute (i,j) index for current GPU thread using
    // block indices
    int i = blockIdx.x * BLOCK_LENGTH + threadIdx.x;
    int j = blockIdx.y * BLOCK_LENGTH + threadIdx.y;

    // Compute one data point for the given (i,j) coordinates
    // (i,j) = 0.1f * (ex,ey)
    float val = 0.1f * (float)i + 0.1f * (float)j;
    out[i] = val;
}
```

# План

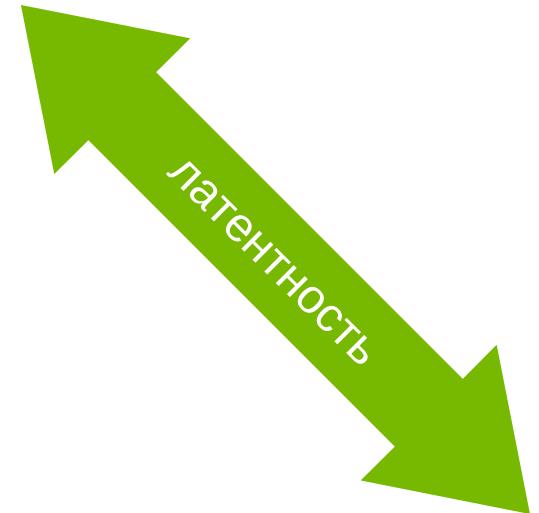
- Гибридная архитектура
- Иерархия памяти, процессы, потоки
- GPU контекст
- Примеры multi-GPU программ
- Асинхронные операции, CUDA streams

# Гибридная архитектура



# Иерархия памяти

- Кеш ядра, сокета
- Ближняя RAM сокета
- Дальняя RAM (через QPI)
- Память PCI-E устройства
- RAM другого выч. узла
- Память чужого PCI-E устройства



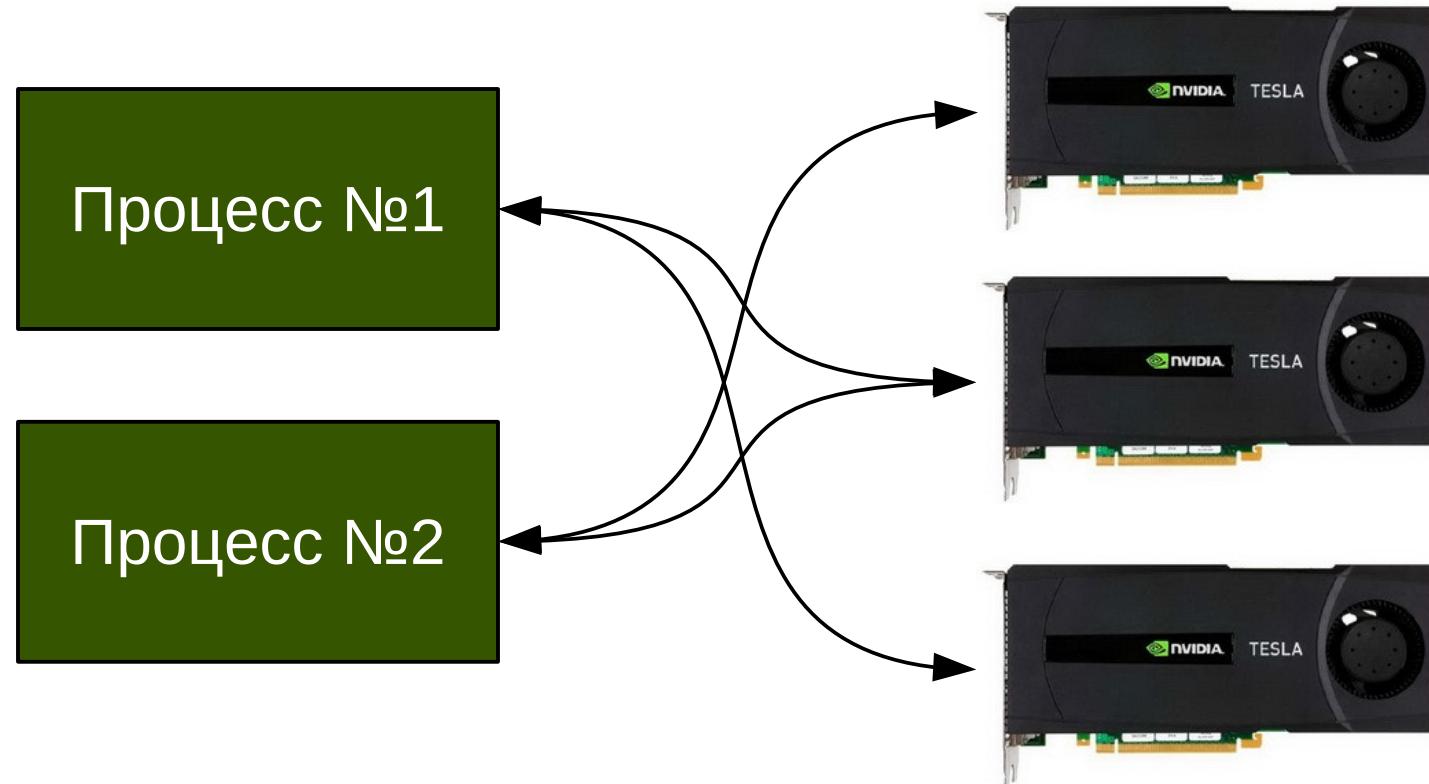
# Роль CUDA

- **CUDA** – это API для организации вычислений на GPU, но не на всём кластере
- Для работы с более сложные системами необходимо использовать дополнительные программные модели

# ОС: процессы

- Собственный контекст в системе (память, файлы, ...)
- Управление процессами происходит через системные вызовы (сравнительно дорого)

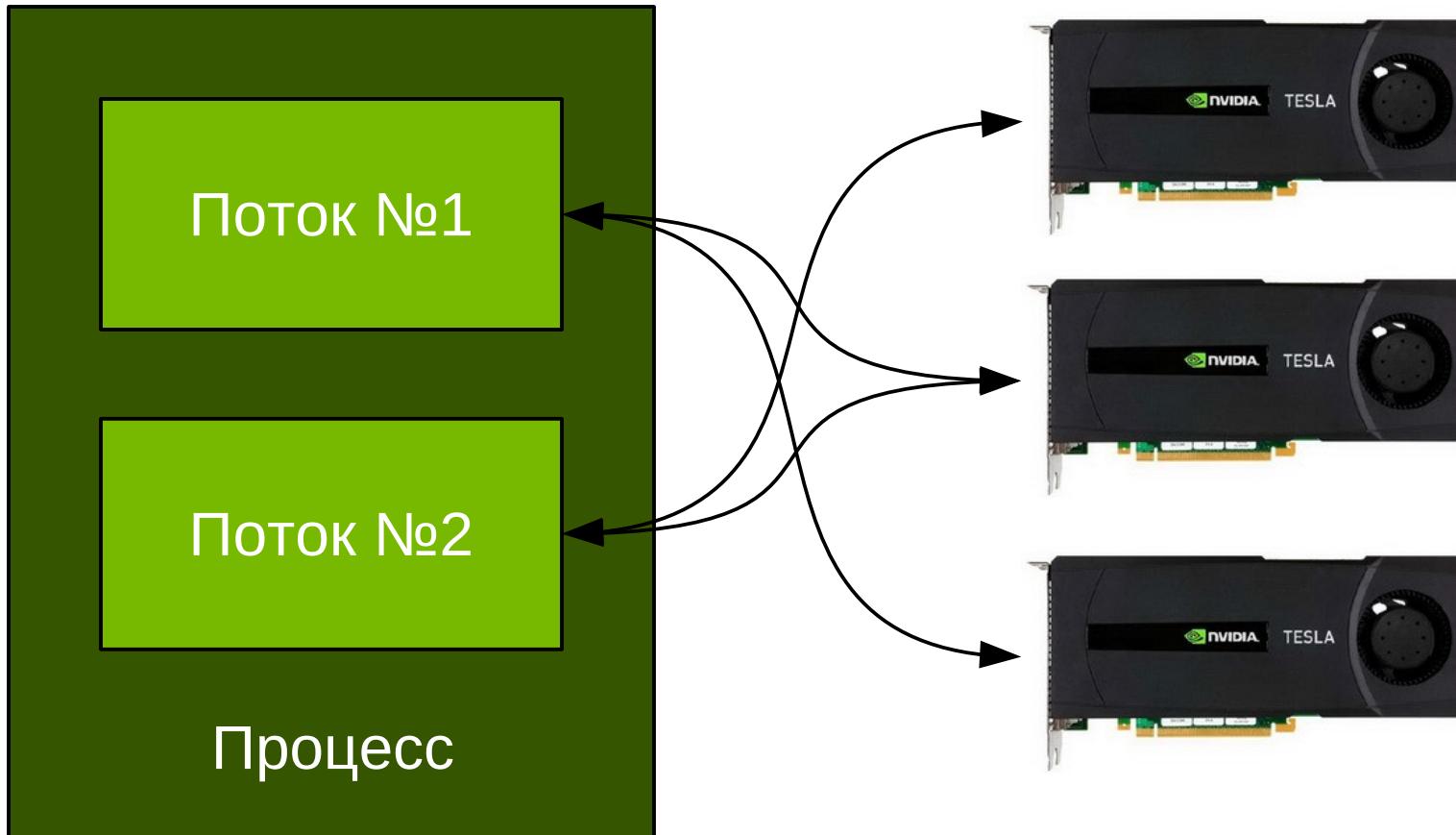
# ОС: процессы



# ОС: потоки

- Общий доступ ко всей памяти родительского процесса, локальная память потока
- В управлении потоками ОС может играть меньшую роль

# ОС: потоки



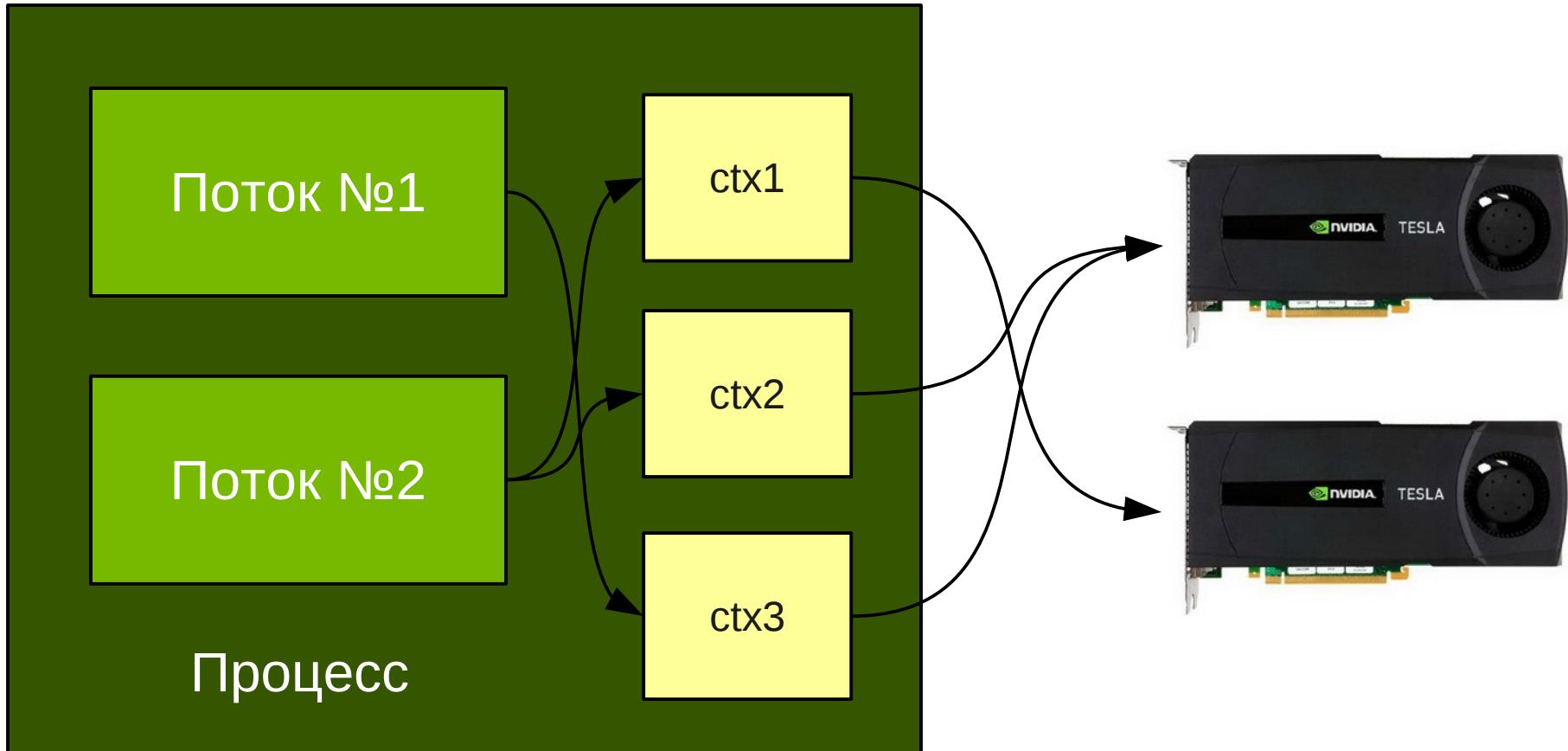
# Контекст устройства (CUDA)

- Контекст – привязанная к определённому устройству управляющая информация (выделенная device-память, результат операции, ...)
- При обращении к устройству многие CUDA-вызовы требуют существование контекста

# Контекст устройства (CUDA)

- Изначально поток/процесс не имеет текущего CUDA-контекста
- Если в процессе/потоке нет текущего контекста, то он будет создан неявно при необходимости
- Одно устройство может иметь несколько контекстов

# Контекст устройства (CUDA)



# Контекст устройства (CUDA)

- Каждый процесс/поток может иметь не более одного текущего контекста
- Процесс/поток может создавать и удалять контексты, менять текущий контекст

# Примеры multi-GPU приложений

- cudaSetDevice
- fork(), MPI, POSIX/thread, OpenMP, Boost
- CUDA streams

Полный исходный код примеров –  
в материалах к лекции

# Управление контекстами

- CUDA API:

- Контекст устройства создаётся неявно
- Переключение контекста:  
`cudaSetDevice(<номер_устройства>)`

- Driver API:

- `cuCtxCreate/cuCtxDestroy`
- `cuCtxPushCurrent/cuCtxPopCurrent`

# Пример №1

Создать процесс, использующий одновременно несколько GPU с помощью cudaSetDevice

Реализация: serial/serial\_cuda/

# cudaSetDevice

```
// Set current CUDA device.  
cuda_status = cudaSetDevice(idevice);  
if (cuda_status != cudaSuccess)  
{  
    fprintf(stderr, "Cannot set current CUDA device, status = %d: %s\n",  
            cuda_status, cudaGetErrorString(cuda_status));  
    return cuda_status;  
}
```

# Open Group / IEEE

- Создание дочернего процесса  
`fork`
- Файлы с общим доступом  
`shm_open`, `shm_unlink`
- Отображение файла в память процесса  
`mmap`, `munmap`, `msync`
- Семафоры  
`sem_open`, `sem_wait`, `sem_post`, `sem_unlink`

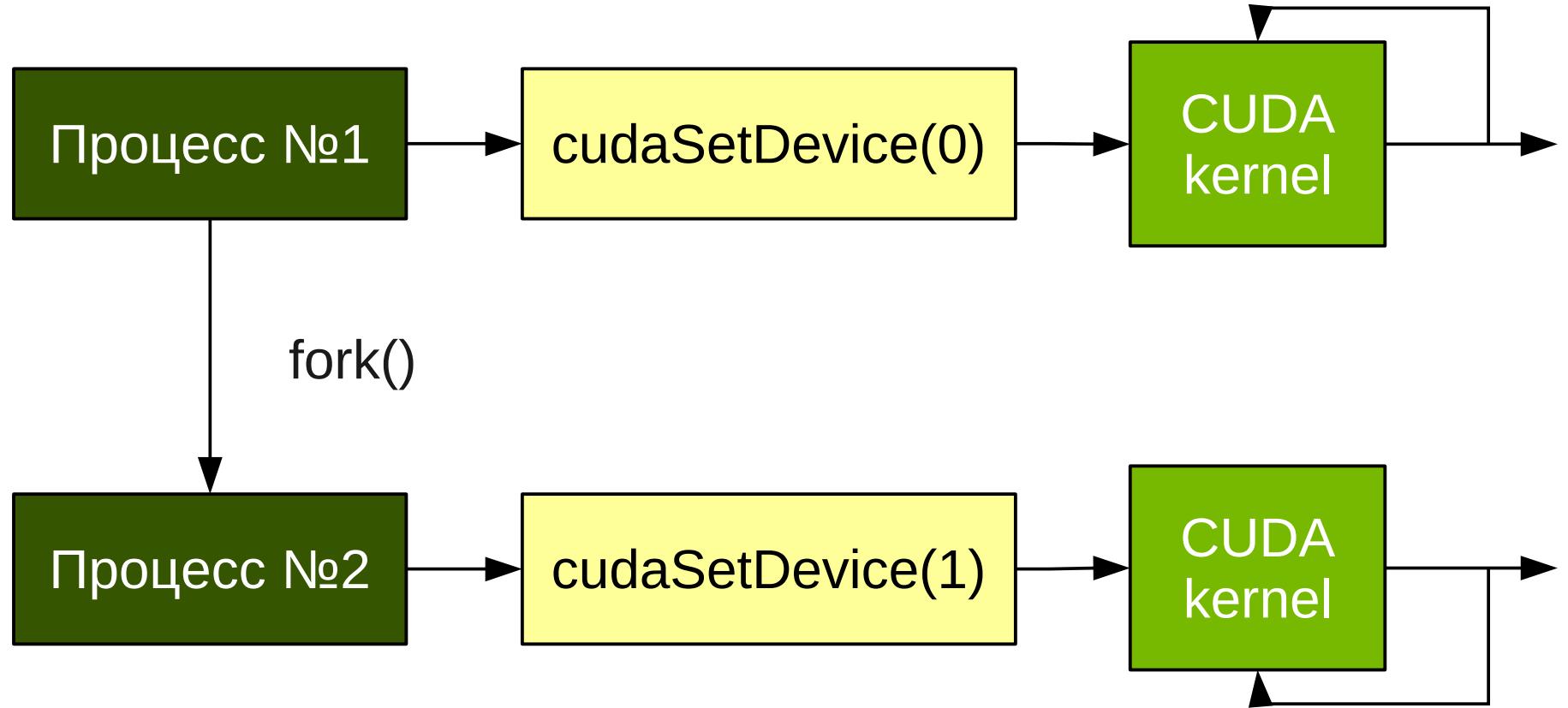
## Пример №2

Создать несколько процессов,  
обрабатывающих независимые данные  
на одном или нескольких GPU

Если в системе один GPU, использовать  
его во всех процессах.

Реализация: unix/process\_fork\_cuda/

# Пример №2



# fork()

```
// Call fork to create another process.  
// Standard: "Memory mappings created in the parent  
// shall be retained in the child process."  
pid_t fork_status = fork();  
  
// From this point two processes are running the same code, if no errors.  
if (fork_status == -1)  
{  
    fprintf(stderr, "Cannot fork process, errno = %d\n", errno);  
    return errno;  
}  
  
// By fork return value we can determine the process role:  
// master or child (worker).  
int master = fork_status ? 1 : 0, worker = !master;  
  
// Get the process ID.  
int pid = (int)getpid();
```

# Message Passing Interface (MPI)

- Реализация – библиотека, демон
- Единый код исполняется множеством параллельных процессов
- Демоны MPI контролируют запуск и состояние процессов на узлах вычислительной сети
- Библиотека MPI реализует интерфейс обмена сообщениями (данные, синхронизация) для имеющейся сети

# Message Passing Interface (MPI)

- Порождение множества процессов  
`mpirun`, `mpiexec`
- Инициализация, деинициализация  
`MPI_Init`, `MPI_Finalize`
- Обмен данными  
`MPI_Send`, `MPI_Recv`, `MPI_Bcast`, ...
- Синхронизация  
`MPI_Barrier`, ...

# GPU-память в командах MPI

Поддержка использования девайс-адресов в MPI  
командах с CUDA 4.0

Доступна в OpenMPI trunk (Rolf vandeVaart)

```
[dmikushin@sm06 forge]$ svn co http://svn.open-mpi.org/svn/ompi/trunk  
ompi-trunk  
[dmikushin@sm06 forge]$ cd ompi-trunk/  
[dmikushin@sm06 ompi-trunk]$ ./autogen.pl  
[dmikushin@sm06 ompi-trunk]$ mkdir build  
[dmikushin@sm06 ompi-trunk]$ cd build  
[dmikushin@sm06 build]$ ../configure  
--prefix=/home/dmikushin/opt/openmpi_gcc-trunk --with-cuda  
[dmikushin@sm06 build]$ make install
```

# Пример №3

С помощью MPI создать несколько процессов, обрабатывающих независимые данные на GPU и обменивающихся результатами через память GPU.

Реализация: mpi/mpi\_cuda\_p2p/

# MPI\_Init / MPI\_Finalize

```
// Initialize MPI. From this point the specified
// number of processes will be executed in parallel.
int mpi_status = MPI_Init(&argc, &argv);
if (mpi_status != MPI_SUCCESS)
{
    fprintf(stderr, "Cannot initialize MPI, status = %d\n",
            mpi_status);
    return mpi_status;
}

mpi_status = MPI_Finalize();
if (mpi_status != MPI_SUCCESS)
{
    fprintf(stderr, "Cannot finalize MPI, status = %d\n",
            mpi_status);
    return mpi_status;
}
```

# MPI\_Comm\_size / \*\_rank

```
// Get the size of the MPI global communicator,  
// that is get the total number of MPI processes.  
int nprocesses;  
mpi_status = MPI_Comm_size(MPI_COMM_WORLD, &nprocesses);  
if (mpi_status != MPI_SUCCESS)  
{  
    fprintf(stderr,  
            "Cannot retrieve the number of MPI processes, status = %d\n",  
            mpi_status);  
    return mpi_status;  
}  
  
// Get the rank (index) of the current MPI process  
// in the global communicator.  
mpi_status = MPI_Comm_rank(MPI_COMM_WORLD, &config.idevice);  
if (mpi_status != MPI_SUCCESS)  
{  
    fprintf(stderr,  
            "Cannot retrieve the rank of current MPI process, status = %d\n",  
            mpi_status);
```

# MPI\_Send / MPI\_Recv

- В MPI\_Send/\_Recv – device-указатели din1/din2

```
float *din1, *din2;
cuda_status = cudaMalloc((void**)&din1, size);
...
cuda_status = cudaMalloc((void**)&din2, size);
...
MPI_Request request;
int inext = (iprocess + 1) % nprocesses;
int iprev = iprocess - 1; iprev += (iprev < 0) ? nprocesses : 0;

// Pass entire process input device buffer directly to input device buffer
// of next process.
mpi_status = MPI_Isend(din1, n * n, MPI_FLOAT, inext, 0, MPI_COMM_WORLD, &request);
mpi_status = MPI_Recv(din2, n * n, MPI_FLOAT, iprev, 0, MPI_COMM_WORLD, NULL);
mpi_status = MPI_Wait(&request, MPI_STATUS_IGNORE);
```

# Дополнительные замечания

- Если процессы зависли из-за некорректной синхронизации, то поможет вызов **killall** ☺

```
[marcusmae@T61p process_fork_cuda]$ ps aux | grep fork
500    6909 52.3  0.0 32988 1152 pts/7   R   16:50  0:46 ./process_fork_cuda
500    6911 75.4  0.0 32988 1152 pts/7   R   16:50  1:03 ./process_fork_cuda
500    6913 46.0  0.0 32988 1152 pts/7   R   16:50  0:35 ./process_fork_cuda
500    7013  0.0  0.0 103384  836 pts/7   S+  16:52  0:00 grep --color=auto fork
[marcusmae@T61p process_fork_cuda]$ killall -9 process_fork_cuda
```

# POSIX threads (pthreads)

- Реализация – библиотека
- Пользователь явно управляет созданием, завершением потоков и их свойствами
- Пользователь явно управляет взаимодействием потоков

# POSIX threads (pthread)

- Порождение и ожидание завершения потока
  - `pthread_create`, `pthread_join`
- Критическая секция
  - `pthread_mutex_lock`, `pthread_mutex_unlock`, ...
- Барьерная и условная синхронизация
  - `pthread_barrier_wait`, `pthread_cond_wait`

# Unified Virtual Address Space

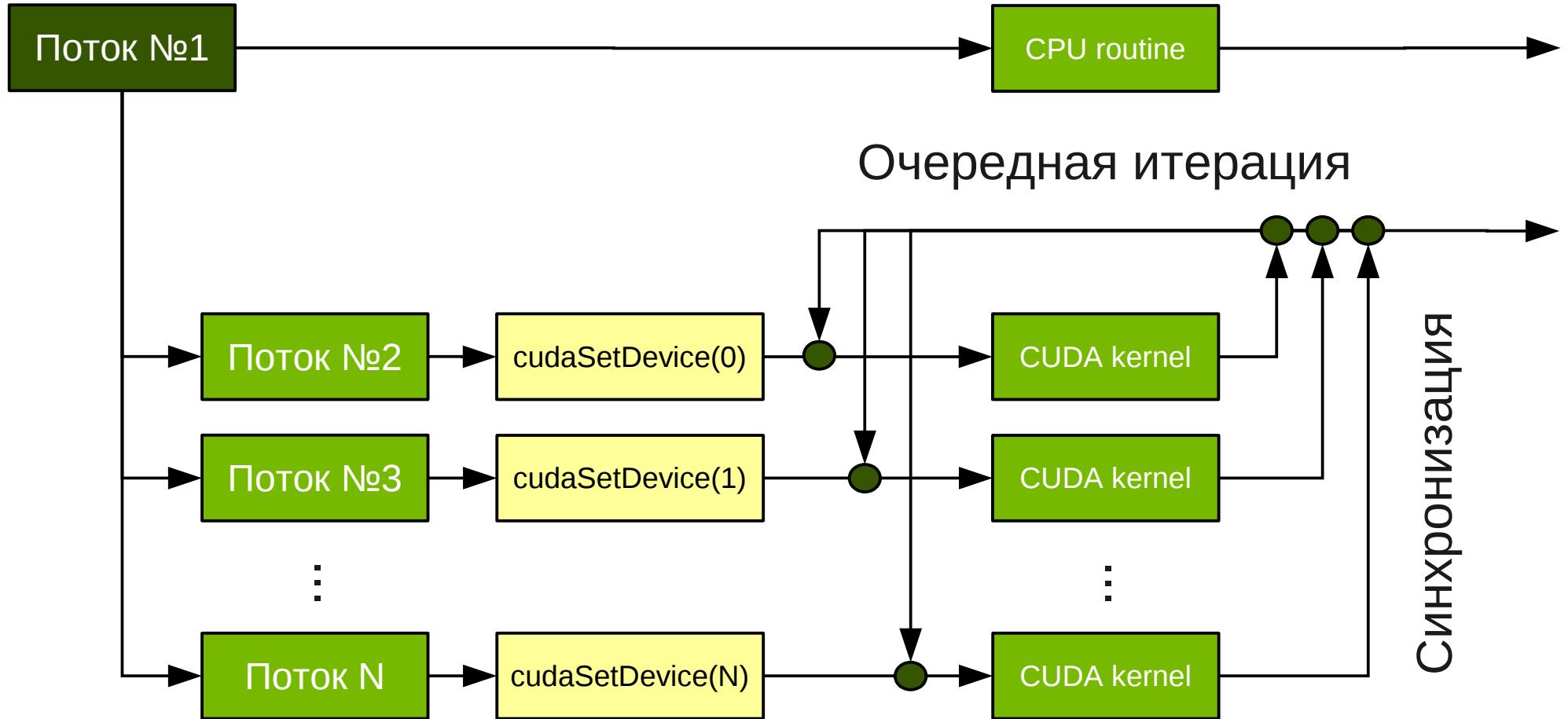
- Общий диапазон адресов для хоста и всех GPU в рамках одного процесса
  - cudaMalloc, cudaMallocHost
- Возможность использования на GPU хост-памяти и памяти другого GPU
  - cudaDeviceEnablePeerAccess
- Отображение и использование на GPU хост-памяти, выделенной обычным malloc
  - cudaHostRegister, cudaHostGetDevicePointer

# Пример №5

С помощью pthread создать несколько потоков, параллельно обрабатывающих независимые данные на GPU и обменивающихся результатами через память GPU.

Реализация: pthread/thread\_cuda\_p2p/

# Пример №5



# pthread\_create

```
// Start threads.
for (int ithread = 0; ithread < nthreads; ithread++)
{
    struct thread_params_t* t = thread_params + ithread;
    t->iThread = ithread;
    t->nThreads = nthreads;
    t->n = n; t->npasses = npasses; t->size = size;
    t->prev = thread_params + ithread - 1;
    t->prev += ithread ? 0 : nthreads;
    t->barrier = &barrier;

    if (pthread_create(&thread_params[ithread].handle, NULL,
                      thread_func, &thread_params[ithread]))
    {
        fprintf(stderr, "Cannot create pthread thread\n");
        return 1;
    }
}
```

# pthread\_exit, pthread\_join

```
pthread_exit(NULL);

...
// Wait for threads completion.
for (int ithread = 0; ithread < nthreads; ithread++)
{
    if (pthread_join(thread_params[ithread].handle, NULL))
    {
        fprintf(stderr, "Cannot join pthread thread\n");
        return 1;
    }
}
```

# pthread\_barrier\_init / \*\_wait

```
// Initialize threads barrier.  
pthread_barrier_t barrier;  
if (pthread_barrier_init(&barrier, NULL, nthreads))  
{  
    fprintf(stderr, "Cannot initialize pthread barrier\n");  
    return 1;  
}  
  
// Synchronize after iteration.  
pthread_barrier_wait(current->barrier);  
  
if (pthread_barrier_destroy(&barrier))  
{  
    fprintf(stderr, "Cannot destroy pthread barrier\n");  
    return 1;  
}
```

# cudaMemcpyPeer

```
// Pass entire process input device buffer directly to input device buffer
// of next process.
status = cudaMemcpyPeer(current->din2, current->ithread,
                      current->prev->din1, current->prev->ithread, current->prev->size);
if (status != cudaSuccess)
{
    fprintf(stderr,
            "Cannot start peer to peer transfer between device %d and %d: %s\n",
            current->ithread, current->prev->ithread,
            cudaGetErrorString(status));
    pthread_exit(NULL);
}

status = cudaDeviceSynchronize();
```

# OpenMP

- Реализация – директивы (расширения языков C, Fortran, ...), библиотека
- Созданием и завершением потоков управляет runtime-библиотека, некоторые свойства потоков могут быть заданы пользователем явно
- Пользователь явно управляет взаимодействием потоков

# OpenMP

- Параллельное исполнение

`#pragma omp parallel`

- Число потоков

`omp_get_num_threads()`, `OMP_NUM_THREADS`

- Параллельные циклы

`#pragma omp parallel for`

# Пример №6

С помощью OpenMP создать несколько потоков, параллельно обрабатывающих независимые данные на GPU и CPU.

Реализация: `openmp/openmp_cuda/`

# omp section-s, parallel for

```
// For each CUDA device found create a separate thread
// and execute the thread_func.
#pragma omp sections
{
    // Section for GPU threads.
    #pragma omp section
    {
    }

    // Section for CPU thread.
    #pragma omp section
    {
    }
}
```

# omp section-s, parallel for

```
// Section for GPU threads.  
#pragma omp section  
{  
    #pragma omp parallel for  
    for (int i = 0; i < ndevices; i++)  
    {  
        config_t* config = configs + i;  
        config->idevice = i;  
        config->step = 0;  
        config->nx = nx; config->ny = ny;  
        config->inout_cpu = inout + np * i;  
        config->status = thread_func(config);  
    }  
}
```

# omp section-s, parallel for

```
// Section for CPU thread.
#pragma omp section
{
    // In parallel main thread launch CPU function equivalent
    // to CUDA kernels, to check the results.
    control = inout + ndevices * np;
    float* input = inout + (ndevices + 1) * np;
    for (int i = 0; i < nticks; i++)
    {
        pattern2d_cpu(1, configs->nx, 1, 1, configs->ny, 1,
                      input, control, ndevices);
        float* swap = control;
        control = input;
        input = swap;
    }
    float* swap = control;
    control = input;
    input = swap;
}
```

# Boost

- Создание потока  
`boost::thread, boost::bind`
- Синхронизация  
`boost::mutex, boost::barrier`

# Пример №7

С помощью Boost создать несколько параллельных потоков, пошагово обрабатывающих независимые данные на GPU и CPU с синхронизацией после каждого шага.

Реализация: boost/boost\_cuda/

# thread, bind, mutex, barrier

```
static boost::mutex m;
boost::barrier* b1, b2;
boost::thread t;

// The function executed by each thread assigned with CUDA device.
void ThreadRunner::thread_func()
{
    ...
}

ThreadRunner::ThreadRunner(int idevice, int nx, int ny, boost::barrier* b) :
    t(boost::bind(&ThreadRunner::thread_func, this)), b2(2), finish(0)
...
...
```

# thread, bind, mutex, barrier

```
// Create a barrier that will wait for (ndevices + 1)
// invocations of wait().
boost::barrier b(ndevices + 1);

// Initialize thread runners and load input data.
ThreadRunner** runners = new ThreadRunner*[ndevices + 1];
for (int i = 0; i < ndevices; i++)
{
    runners[i] = new ThreadRunner(i, nx, ny, &b);
    runners[i]->Load(data);
}
```

# thread, bind, mutex, barrier

```
// Compute the given number of steps.
float* input = data;
float* output = data + np;
for (int i = 0; i < nticks; i++)
{
    // Pass iteration on device threads.
    for (int i = 0; i < ndevices; i++)
        runners[i]->Pass();

    int status = ThreadRunner::GetLastError();
    if (status) return status;

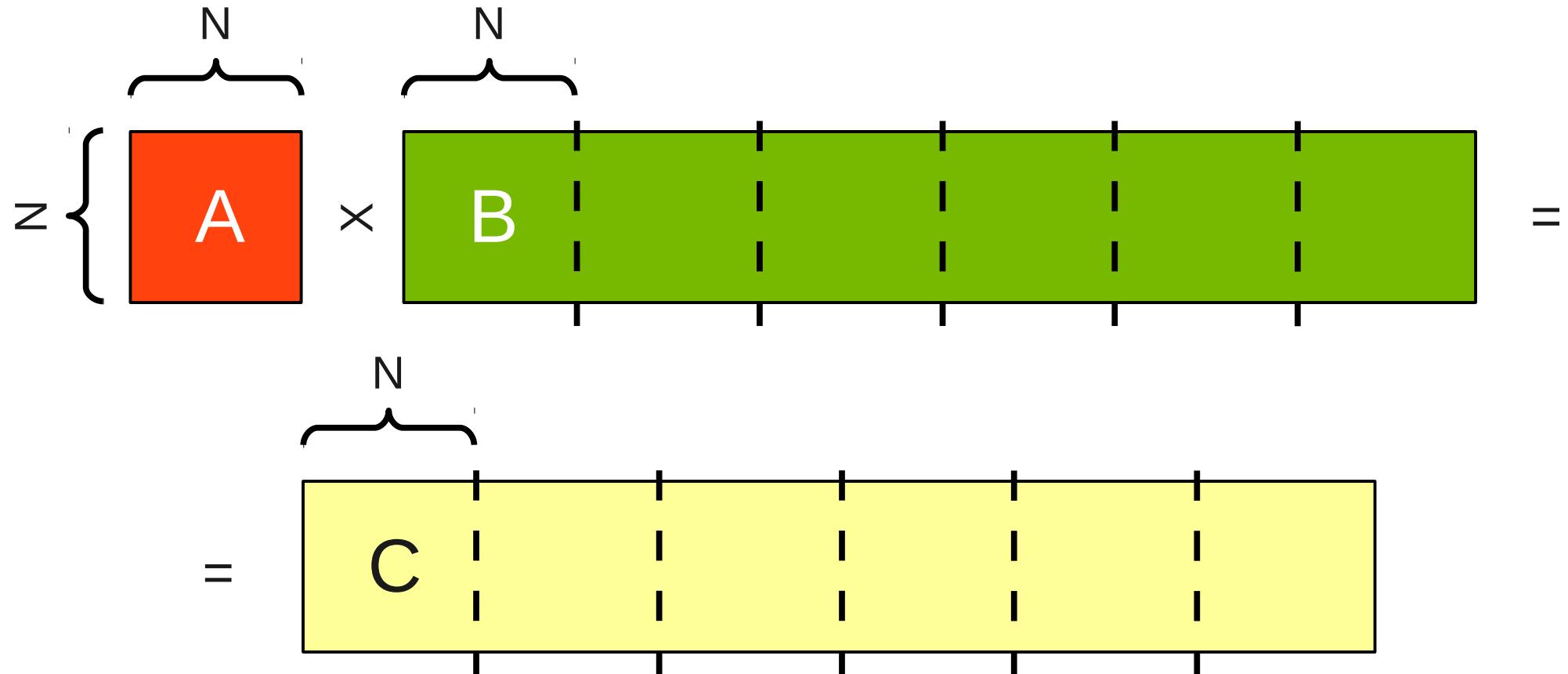
    // In parallel main thread launch CPU function equivalent
    // to CUDA kernels, to check the results.
    pattern2d_cpu(1, nx, 1, 1, ny, 1,
                  input, output, ndevices);
    float* swap = output;
    output = input;
    input = swap;

    b.wait();
}
```

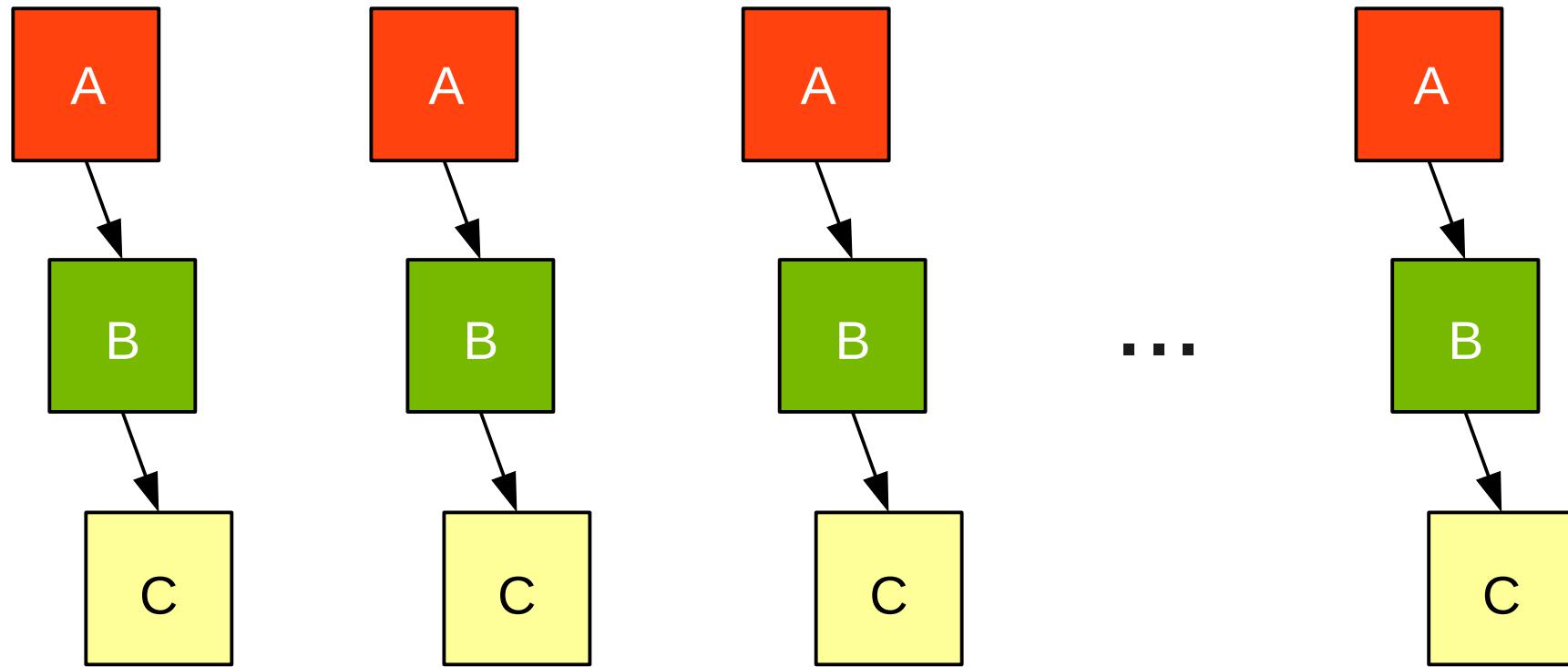
# CUDA Streams

Потоки (streams) – метод выделения последовательностей асинхронных операций, связанных порядком действий в составе одного stream, но независимых для различных streams.

# CUDA Streams



# CUDA Streams



Stream #1

Stream #2

Stream #3

Stream #N

# CUDA Streams

- Параллельные цепочки streams могут эффективнее насыщать вычислительные блоки за счёт передачи данных малыми независимыми блоками.
- Асинхронные операции передачи данных требуют pinned memory (`cudaMallocHost`).

# Пример №8

Блочное перемножение матриц с использованием CUDA streams.

Реализация: gemm\_streams/

```
[dmikushin@tesla-cmc gemm_streamed]$ ./gemm_streamed 4 1024 1025 1 N N 1.0 0.0 16
n      time          gflops       test      enorm      rnorm
1024  0.013909 sec  154.390014 PASSED   0.018676  262177.187500
1024  0.011231 sec  191.204784 PASSED   0.018693  262323.812500
```

```
[dmikushin@tesla-cmc gemm_streamed]$ ./gemm_streamed 4 4096 4097 1 N N 1.0 0.0 16
n      time          gflops       test      enorm      rnorm
4096  0.431783 sec  318.305308 PASSED   0.293618  4194287.250000
4096  0.396524 sec  346.609298 PASSED   0.293707  4194416.500000
```

# cudaStreamCreate / \*Destroy

```
cudaStream_t* stream = (cudaStream_t*)malloc(
    nstreams * sizeof(cudaStream_t));

// Create streams
for (int i = 0; i < nstreams; i++)
{
    cudaerr = cudaStreamCreate(&stream[i]);
    assert(cudaerr == cudaSuccess);
}

// Destroy streams
for (int istream = 0; istream < nstreams; istream++)
{
    cudaerr = cudaStreamDestroy(stream[istream]);
    assert(cudaerr == cudaSuccess);
}
```

# cublasSetVectorAsync

Асинхронная загрузка данных на устройство (аналог cudaMemcpyAsync)

```
for (int istream = 0; istream < nstreams; istream++)
{
    int szpart = n / nstreams;
    size_t shift = n * szpart * istream;
    if (istream == nstreams - 1)
        szpart += n % nstreams;

    status = cublasSetVectorAsync(n * szpart, sizeof(real),
                                h_B + shift, 1, d_B[istream], 1, stream[istream]);
    assert(status == CUBLAS_STATUS_SUCCESS);
    status = cublasSetVectorAsync(n * szpart, sizeof(real),
                                h_C + shift, 1, d_C[istream], 1, stream[istream]);
    assert(status == CUBLAS_STATUS_SUCCESS);
}
```

# cublasSetKernelStream

## Установка stream для CUDA-ядра

```
for (int istream = 0; istream < nstreams; istream++)
{
    int szpart = n / nstreams;
    if (istream == nstreams - 1)
        szpart += n % nstreams;

    // Setup async operations
    status = cublasSetKernelStream(stream[istream]);
    assert(status == CUBLAS_STATUS_SUCCESS);

    // Perform matmul using CUBLAS
    cublas_gemm(transa, transb, n, szpart, n,
                alpha, d_A, n, d_B[istream], n, beta, d_C[istream], n);
    status = cublasGetError();
    assert(status == CUBLAS_STATUS_SUCCESS);
}
```

# cublasGetVectorAsync

Асинхронная выгрузка данных из устройства (аналог cudaMemcpyAsync)

```
// Sync all
for (int istream = 0; istream < nstreams; istream++)
{
    int szpart = n / nstreams;
    size_t shift = n * szpart * istream;
    if (istream == nstreams - 1)
        szpart += n % nstreams;

    // Read the result back
    status = cublasGetVectorAsync(n * szpart, sizeof(real),
                                  d_C[istream], 1, h_C + shift, 1, stream[istream]);
    assert(status == CUBLAS_STATUS_SUCCESS);
}
```

# **cudaStreamSynchronize**

Ожидание выполнения всех операций в  
заданном stream

```
cudaStreamSynchronize(stream[istream]);
```

# Заключение

- Код на CUDA может взаимодействовать с другими программными моделями параллельных вычислений
- Методы из предложенных примеров могут быть использованы в Ваших приложениях и протестированы на сервере tesla-смс

# Отладка приложений для GPU и Multi-GPU

Дмитрий Микушин, NVIDIA

```
__global__ void kernel1(gpu1* gpu,
                        int ex,
                        int n,
                        float* out)

    // Compute absolute (i,j) index
    // for current GPU thread using
    // block indices
    int i = blockIdx.x * BLOCK_LENGTH +
            threadIdx.x;
    int j = blockIdx.y * BLOCK_LENGTH +
            threadIdx.y;

    // Compute one data point
    // for the given coordinates
    float val = 0.1f *
                (2.0f *
```

# План

- Основные сведения об отладке
- Отладчик gdb: пример использования
- Отладчик cuda-gdb: особенности, требования, примеры
- Диагностика девайс-памяти с помощью cuda-memcheck, примеры

# Отладка: терминология

- *Фрейм* (stack frame) – “кадр” состояния функции в момент останова или передачи управления во вложенную функцию
- *Трасса* – полная иерархия фреймов от точки входа в программу до положения останова
- *Breakpoint* – точка останова в коде программы
- *Watchpoint* – точка останова по условию достижения заданного состояния

# Отладка: основные возможности

- Просмотр значений переменных и вычисление несложных выражений
- Назначение точек и условий останова
- Перемещение по стеку фреймов
- Отладка нескольких потоков
- Анализ отладочного дампа (core dump) в offline-режиме

# Исполняемый формат ELF

- Секции – разделы исполняемого файла, в которых информация сгруппирована по типу:
  - `.text` – исполняемый код
  - `.bss` – неинициализированные данные
  - `.data` – инициализированные данные
  - ...
  - `.debug` – **отладочная информация**

# Отладочная информация

- Таблица соответствия обычных имён символов декорированным (*tangled*)
- Таблица соответствия адресов исполняемого кода и строк исходного кода
- Любая дополнительная информация

# GDB: рюшки

- Сокращённые имена команд: b – breakpoint, wa – watchpoint, ...
- В командной строке GDB в любом месте можно нажать <ТАВ>, чтобы показать варианты автозаполнения (или сделать его сразу, если возможен только один вариант)

# GDB: компиляция

- Флаг -g для включения отладочной информации

```
[marcusmae@T61p samples]$ cd multigpu/pthread/pthread_cuda/  
[marcusmae@T61p pthread_cuda]$ make  
gcc -g -std=c99 -I/opt/cuda/include -c pthread_cuda.c -o pthread_cuda.o  
nvcc -g -c pattern2d.cu -o pattern2d.o  
gcc pthread_cuda.o pattern2d.o -o pthread_cuda -lpthread  
-L/opt/cuda/lib64 -lcudart -lm
```

# GDB: запуск

- Запуск отладчика одновременно с запуском приложения

```
[marcusmae@T61p pthread_cuda]$ gdb ./pthread_cuda
GNU gdb (GDB) Fedora (7.2-51.fc14)
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/marcusmae/Samples/cuda-
training/samples/multigpu/thread/thread_cuda/thread_cuda...done.
(gdb)
```

# GDB: запуск (attach)

- Подключение к ранее запущенному приложению

```
[marcusmae@T61p pthread_cuda]$ gdb pthread_cuda 3346
Reading symbols from /home/marcusmae/Samples/cuda-
training/samples/multigpu/thread/thread_cuda/thread_cuda...done.
Attaching to program: /home/marcusmae/Samples/cuda-
training/samples/multigpu/thread/thread_cuda/thread_cuda, process
3346
(gdb)
```

# GDB: breakpoint

- Установка breakpoint по имени функции

```
(gdb) b pattern2d_cpu
Breakpoint 1 at 0x401675: file pattern2d.cu, line 34.
(gdb) r
Starting program: /home/marcusmae/Samples/cuda-
training/samples/multigpu/pthread/pthread_cuda/pthread_cuda
[Thread debugging using libthread_db enabled]
1 CUDA device(s) found
[New Thread 0xfffff6f44700 (LWP 3349)]

Breakpoint 1, pattern2d_cpu (bx=1, nx=128, ex=1, by=1, ny=128, ey=1,
    in=0xfffff6f65010, out=0xfffff6f55010, id=1) at pattern2d.cu:34
34      size_t size = nx * ny * sizeof(float);
(gdb)
```

# GDB: breakpoint

- Установка breakpoint по номеру строки
- Удаление – d <номер>

```
(gdb) b 36
Breakpoint 2 at 0x40169b: file pattern2d.cu, line 36.
(gdb) b pattern2d.cu:37
Note: breakpoint 2 also set at pc 0x40169b.
Breakpoint 3 at 0x40169b: file pattern2d.cu, line 37.
(gdb) c
Continuing.

Breakpoint 2, pattern2d_cpu (bx=1, nx=128, ex=1, by=1, ny=128, ey=1,
    in=0x7fffff6f65010, out=0x7fffff6f55010, id=1) at pattern2d.cu:37
37      for (int j = by; j < ny - ey; j++)
(gdb)
```

# GDB: watchpoint

- Установка watchpoint по адресу в памяти

```
(gdb) b 38
Breakpoint 4 at 0x4016a6: file pattern2d.cu, line 38.
(gdb) c
Continuing.
```

```
Breakpoint 4, pattern2d_cpu (bx=1, nx=128, ex=1, by=1, ny=128, ey=1,
    in=0x7fffff6f55010, out=0x7fffff6f65010, id=1) at pattern2d.cu:38
38          for (int i = bx; i < nx - ex; i++)
(gdb) p &j
$1 = (int *) 0xfffffffffdf3c
(gdb) wa *(int*)0xfffffffffdf3c
Hardware watchpoint 5: *(int*)0xfffffffffdf3c
```

# GDB: watchpoint

- Достижение watchpoint

```
(gdb) d 4
(gdb) c
Continuing.
Hardware watchpoint 5: *(int*)0x7fffffffdf3c

Old value = 1
New value = 2
0x00000000004017a2 in pattern2d_cpu (bx=1, nx=128, ex=1, by=1, ny=128, ey=1,
    in=0x7ffff6f55010, out=0x7ffff6f65010, id=1) at pattern2d.cu:37
37      for (int j = by; j < ny - ey; j++)
```

# GDB: контекст

- list – просмотр исходного кода в месте останова

```
(gdb) list
32     float* in, float* out, int id)
33 {
34     size_t size = nx * ny * sizeof(float);
35     memset(out, 0, size);
36
37     for (int j = by; j < ny - ey; j++)
38         for (int i = bx; i < nx - ex; i++)
39             OUT(i,j) = sqrtf(fabs(IN(i,j) + IN(i-1,j) + IN(i+1,j) -
40                                 2.0f * IN(i,j-1) + 3.0f * IN(i,j+1)));
41
(gdb)
```

# GDB: контекст

- Ctrl-x-a – “оконный” TUI-режим

```
pattern2d.cu
33     {
B+         size_t size = nx * ny * sizeof(float);
35         memset(out, 0, size);
B+
B+> 36
B+> 37         for (int j = by; j < ny - ey; j++)
38             for (int i = bx; i < nx - ex; i++)
39                 OUT(i,j) = sqrtf(fabs(IN(i,j) + IN(i-
40                                     2.0f * IN(i,j-1) + 3.0f * IN(
41                     return 0;
42     }
43
44     // GPU device kernel.
45     __global__ void pattern2d_gpu_kernel(
i,j+1)|
```

multi-thread Thread 0x7fffff In: pattern2d\_cpu  
(gdb)

Line: 37 PC: 0x2011

# GDB: многопоточность

- info th – информация о запущенных потоках
- t <номер> - переключение в поток

```
(gdb) info th
 2 Thread 0x7ffff6f44700 (LWP 3349)  0x0000003befed8997 in ioctl ()
    from /lib64/libc.so.6
* 1 Thread 0x7ffff786c740 (LWP 3346)  0x00000000004017a2 in pattern2d_cpu (
    bx=1, nx=128, ex=1, by=1, ny=128, ey=1, in=0x7ffff6f55010,
    out=0x7ffff6f65010, id=1) at pattern2d.cu:37
(gdb) t 2
[Switching to thread 2 (Thread 0x7ffff6f44700 (LWP 3349))]#0
0x0000003befed8997 in ioctl () from /lib64/libc.so.6
(gdb)
```

# GDB: трасса

- bt – трасса

```
(gdb) bt
#0 0x0000003befed8997 in ioctl () from /lib64/libc.so.6
#1 0x00007ffff710b923 in ?? () from /usr/lib64/libcuda.so
...
#10 0x00007ffff70e7b41 in ?? () from /usr/lib64/libcuda.so
#11 0x00007ffff7daa82e in ?? () from /opt/cuda/lib64/libcudart.so.4
#12 0x00007ffff7daad7b in ?? () from /opt/cuda/lib64/libcudart.so.4
#13 0x00007ffff7dab764 in ?? () from /opt/cuda/lib64/libcudart.so.4
#14 0x00007ffff7da09b6 in ?? () from /opt/cuda/lib64/libcudart.so.4
#15 0x00007ffff7d98659 in ?? () from /opt/cuda/lib64/libcudart.so.4
#16 0x00007ffff7dbdf88 in cudaMalloc () from /opt/cuda/lib64/libcudart.so.4
#17 0x0000000000400dea in thread_func (arg=0x611cf0) at pthread_cuda.c:68
#18 0x0000003bf0206ccb in start_thread () from /lib64/libpthread.so.0
#19 0x0000003befee0c2d in clone () from /lib64/libc.so.6
(gdb)
```

# GDB: переключение фрейма

- `f <номер>` – переключить фрейм

```
(gdb) f 17
#17 0x0000000000400dea in thread_func (arg=0x611cf0) at pthread_cuda.c:68
68      cuda_status = cudaMalloc((void**)&config->in_dev, size);
(gdb) list
63 }
64
65     size_t size = config->nx * config->ny * sizeof(float);
66
67     // Create device arrays for input and output data.
68     cuda_status = cudaMalloc((void**)&config->in_dev, size);
69     if (cuda_status != cudaSuccess)
70     {
71         fprintf(stderr, "Cannot allocate CUDA input buffer on device %d,
status = %d\n",
72                 idevice, cuda_status);
(gdb)
```

# Отладчик: ограничения

- Отладчики могут работать и без отладочной информации, но возможностей будет меньше
- Если код содержит оптимизации, то отладка может быть менее точна (позиционирование по коду, недоступность значений исключённых оптимизацией переменных, ...)

# CUDA-GDB

- Вариант GDB с расширениями для отладки ядер CUDA
- Поддержка отладки на одном GPU или multi-GPU
- Следуя GPL, доступен в исходниках:  
<ftp://download.nvidia.com/CUDAOpen64/>

# CUDA-GDB: требования

- Требует один свободный GPU

fatal: All CUDA devices are used for X11 and cannot be used while debugging.  
(error code = 24)

⇒ в GNU/Linux в текстовом режиме можно работать с cuda-отладчиком на машине с **одним** GPU

- Можно запустить не более одного сеанса cuda-gdb

Found an already running instance of cuda-gdb. If you believe you are seeing this message in error, try deleting /tmp/cuda-gdb.lock

# CUDA-GDB: компиляция

- Флаг -g для включения отладочной информации на хосте, -G – на GPU

```
[marcusmae@T61p pattern2d]$ make
nvcc -g -G -c pattern2d.cu
gcc -g -std=c99 -c -I/usr/include/ImageMagick
draw_diffs.c
nvcc -g pattern2d.o draw_diffs.o -o pattern2d
-lMagickCore
```

# CUDA-GDB: запуск

```
(cuda-gdb) b pattern2d_gpu_kernel
Breakpoint 1 at 0x4032ba: file pattern2d.cu, line 38.
(cuda-gdb) r
Starting program: /home/dmikushin/Programming/cuda-
training/samples/multigpu/pthread/pthread_cuda/pthread_cuda
[Thread debugging using libthread_db enabled]
[New process 26320]
[New Thread 139849021003584 (LWP 26320)]
8 CUDA device(s) found
[New Thread 139849009342208 (LWP 26324)]
[New Thread 139849000949504 (LWP 26325)]
[New Thread 139848992556800 (LWP 26326)]
[New Thread 139848984164096 (LWP 26327)]
[New Thread 139848975771392 (LWP 26328)]
[New Thread 139848899884800 (LWP 26329)]
[New Thread 139848891492096 (LWP 26330)]
[New Thread 139848883099392 (LWP 26331)]
Device 6 initialized
Device 0 initialized
Device 7 initialized
```

# CUDA-GDB: запуск

```
Device 5 initialized
Device 3 initialized
Device 1 initialized
[Launch of CUDA Kernel 0 (pattern2d_gpu_kernel) on Device 6]
Device 4 initialized
Device 2 initialized
[Launch of CUDA Kernel 1 (pattern2d_gpu_kernel) on Device 1]
[Termination of CUDA Kernel 0 (pattern2d_gpu_kernel) on Device 6]
[Launch of CUDA Kernel 2 (pattern2d_gpu_kernel) on Device 2]
[Launch of CUDA Kernel 3 (pattern2d_gpu_kernel) on Device 7]
[Launch of CUDA Kernel 4 (pattern2d_gpu_kernel) on Device 4]
[Termination of CUDA Kernel 3 (pattern2d_gpu_kernel) on Device 7]
[Launch of CUDA Kernel 5 (pattern2d_gpu_kernel) on Device 0]
[Launch of CUDA Kernel 6 (pattern2d_gpu_kernel) on Device 7]
[Switching to CUDA Kernel 5 (<<<(0,0),(0,0,0)>>>)]
```



```
Breakpoint 1, pattern2d_gpu_kernel<<<(3,7),(32,16,1)>>> (bx=1, nx=128,
    ex=1, by=1, ny=128, ey=1, block_length=32, block_height=16,
    in=0x100000, out=0x110000, id=0) at pattern2d.cu:52
```

```
52     int i = blockIdx.x * block_length + threadIdx.x + bx;
```

# CUDA-GDB: расширения

- Отладка ядра: вывод индексов и размерностей

```
(cuda-gdb) b 57
Breakpoint 2 at 0x7fdbac085e98: file pattern2d.cu, line 57.
(cuda-gdb) c
Continuing.
[Termination of CUDA Kernel 8 (pattern2d_gpu_kernel) on Device 4]
```

```
Breakpoint 2, pattern2d_gpu_kernel<<<(3,7),(32,16,1)>>> (bx=1, nx=128,
    ex=1, by=1, ny=128, ey=1, block_length=32, block_height=16,
    in=0x100000, out=0x110000, id=0) at pattern2d.cu:57
57      OUT(i,j) = sqrtf(fabs(IN(i,j) + IN(i-1,j) + IN(i+1,j) -
(cuda-gdb) p blockIdx
$1 = {x = 0, y = 0}
```

# CUDA-GDB: расширения

- Отладка ядра: переключение текущего блока/потока

```
(cuda-gdb) cuda block 2,1 thread 15,4,0
[Switching to CUDA Kernel 1 (device 0, sm 15, warp 4, lane 15, grid 2, block
(2,1), thread (15,4,0))]
57      OUT(i,j) = sqrtf(fabs(IN(i,j) + IN(i-1,j) + IN(i+1,j) -
(cuda-gdb) p i
$2 = 80
(cuda-gdb) p j
$3 = 21
```

# CUDA-GDB: DDD

```
File Edit View Program Commands Status Source Data
0: main
}
// GPU device kernel.
__global__ void pattern2d_gpu_kernel(
    int bx, int nx, int ex, int by, int ny, int ey,
    int block_length, int block_height,
    float* in, float* out, int id)
{
    // Compute absolute (i,j) indexes for
    // the current GPU thread using grid mapping params.
    int i = blockIdx.x * block_length + threadIdx.x + bx;
    int j = blockIdx.y * block_height + threadIdx.y + by;

    // Compute one data point - a piece of
    // work for the current GPU thread.
    OUT(i,j) = sqrt(fabs(IN(i,j) + IN(i-1,j)) + IN(i+1,j) -
                    2.0f * IN(i,j-1) + 3.0f * IN(i,j+1)));
}

// Perform some dummy 2D field processing on GPU.
int pattern2d_gpu(
    int bx, int nx, int ex, int by, int ny, int ey,
    float* in, float* out, int id)
{
    // Configure GPU computational grid:
    // nx = nbBlocks_x * block_length
    // ny = nbBlocks_y * block_height
}

0x00007fcf940a3c40 <pattern2d_gpu_kernel(int, int, int, int, int, int, int, float*, float*, int)+96>: MOV.U16 R0H, g [0x7].U16
0x00007fcf940a3c48 <pattern2d_gpu_kernel(int, int, int, int, int, int, int, float*, float*, int)+104>: MOV.U16 R0H, R0H
0x00007fcf940a3c50 <pattern2d_gpu_kernel(int, int, int, int, int, int, int, float*, float*, int)+112>: I2I.U32.U16 R1, R1L
0x00007fcf940a3c58 <pattern2d_gpu_kernel(int, int, int, int, int, int, int, float*, float*, int)+120>: IMUL32.U16.U16 R5, R4L, R1H
0x00007fcf940a3c5c <pattern2d_gpu_kernel(int, int, int, int, int, int, int, float*, float*, int)+124>: IMAD32.U16.R5, R4R, R1L, R5
0x00007fcf940a3c60 <pattern2d_gpu_kernel(int, int, int, int, int, int, int, float*, float*, int)+128>: SHL R5, R5, 0x10
0x00007fcf940a3c68 <pattern2d_gpu_kernel(int, int, int, int, int, int, int, float*, float*, int)+136>: IMAD.016 R1, R4L, R1L, R5
0x00007fcf940a3c70 <pattern2d_gpu_kernel(int, int, int, int, int, int, int, float*, float*, int)+144>: IADD32 R1, R3, R1
0x00007fcf940a3c74 <pattern2d_gpu_kernel(int, int, int, int, int, int, int, float*, float*, int)+148>: IADD32 R1, R2, R1
0x00007fcf940a3c78 <pattern2d_gpu_kernel(int, int, int, int, int, int, int, float*, float*, int)+152>: MOV R1, R1
0x00007fcf940a3c80 <pattern2d_gpu_kernel(int, int, int, int, int, int, int, float*, float*, int)+160>: MVI R2, 0x1c
0x00007fcf940a3c88 <pattern2d_gpu_kernel(int, int, int, int, int, int, int, float*, float*, int)+168>: R2A A1, R2
0x00007fcf940a3c90 <pattern2d_gpu_kernel(int, int, int, int, int, int, int, float*, float*, int)+176>: MOV R2, g [A1+0x0]
0x00007fcf940a3c98 <pattern2d_gpu_kernel(int, int, int, int, int, int, int, float*, float*, int)+184>: I2I.U32.U16 R3, R0L

[Launch of CUDA Kernel 0 (memset32_aligned1D<<<(128,1,1),(128,1,1)>>>) on Device 0]
[Termination of CUDA Kernel 0 (memset32_aligned1D<<<(128,1,1),(128,1,1)>>>) on Device 0]
[Launch of CUDA Kernel 1 (pattern2d_gpu_kernel<<<(3,7,1),(32,16,1)>>>) on Device 0]
[Switching focus to CUDA Kernel 1, grid 2, block (0,0,0), thread (0,0,0), device 0, sm 0, warp 0, lane 0]

Breakpoint 1, pattern2d_gpu_kernel<<<(3,7,1),(32,16,1)>>> (bx=1, nx=128, ex=1, by=1, ny=128, ey=1, block_length=32, block_height=16, in=0x110000, out=0x120000, id=0) at pattern2d.cu:52
Current language: auto; currently c++
(gdb) n
(gdb)
```

marcusmae@noisy:~/... DDD: pattern2d.cu 2011

# CUDA-GDB: emacs

```
[Switching focus to CUDA kernel 1, grid 2, block (0,0,0), thread (0,0,0), device 0, sm 0, warp 0, lane 0]

Breakpoint 1, pattern2d_gpu_kernel<<<(3,7,1),(32,16,1)>>> (bx=1, nx=128, ex=1, by=1, ny=128, ey=1, block_length=32, block_height=16, in=0x110000, out=0x120000, id=0) at pattern2d.cu:52
Current language: auto; currently c++
(cuda-gdb) n
(cuda-gdb) n
(cuda-gdb) n
-U:*** *gud-pthread_cuda* Bot L35 (Debugger:run [stopped])
Locals Registers
id      @parameter int 0
out    @global float * @parameter (@global float * @parameter) 0x120000
in     @global float * @parameter (@global float * @parameter) 0x110000
block_height @parameter int 16
block_length @parameter int 32
ey     @parameter int 1
-U:*** *locals of pthread_cuda* Top L1 (Locals:pattern2d_gpu_kernel)
// GPU device kernel.
_global_ void pattern2d_gpu_kernel(
    int bx, int nx, int ex, int by, int ny, int ey,
    int block_length, int block_height,
    float* in, float* out, int id)
{
    // Compute absolute (i,j) indexes for
    // the current GPU thread using grid mapping params.
    int i = blockIdx.x * block_length + threadIdx.x + bx;
    int j = blockIdx.y * block_height + threadIdx.y + by;

    // Compute one data point - a piece of
    // work for the current GPU thread.
    OUT(i,j) = sqrtf(fabs(IN(i,j) + IN(i-1,j) + IN(i+1,j) -
        2.0f * IN(i,j-1) + 3.0f * IN(i,j+1)));
}
--:--- pattern2d.cu 28% L57 SVN-9 (Fundamental)
```

marcusmae@noisy:~/... emacs@noisy

# CUDA-МЕМСЧЕК

- Диагностика ошибок в CUDA-ядрах
- Выявление конкретной причины “Kernel launch failure”
- Может быть запущен отдельно или внутри отладчика

```
(cuda-gdb) set cuda memcheck on
```

# CUDA-MEMCHECK

- Адрес вне допустимых границ

```
[marcusmae@T61p pthread_cuda]$ cuda-memcheck ./pthread_cuda
===== CUDA-MEMCHECK
1 CUDA device(s) found
Device 0 initialized
Cannot execute CUDA kernel #2 by device 0, status = 4
Cannot execute pattern 2d on device 0, status = 4
===== Invalid __global__ read of size 4
=====      at 0x00000170 in pattern2d_gpu_kernel
=====      by thread (0,0,0) in block (1,0,0)
=====      Address 0x00100ef4 is out of bounds
=====
===== ERROR SUMMARY: 1 error
```

# Заключение

- Представленный метод отладки универсально полезен, так как кроме GPU применим к широкому классу встраиваемых систем
- Чем раньше Вы столкнётесь с достаточно серьёзной проблемой, чтобы пришлось освоить отладчик, тем лучше! ☺