JÜLICH
FORSCHUNGSZENTRUM

GPGPU

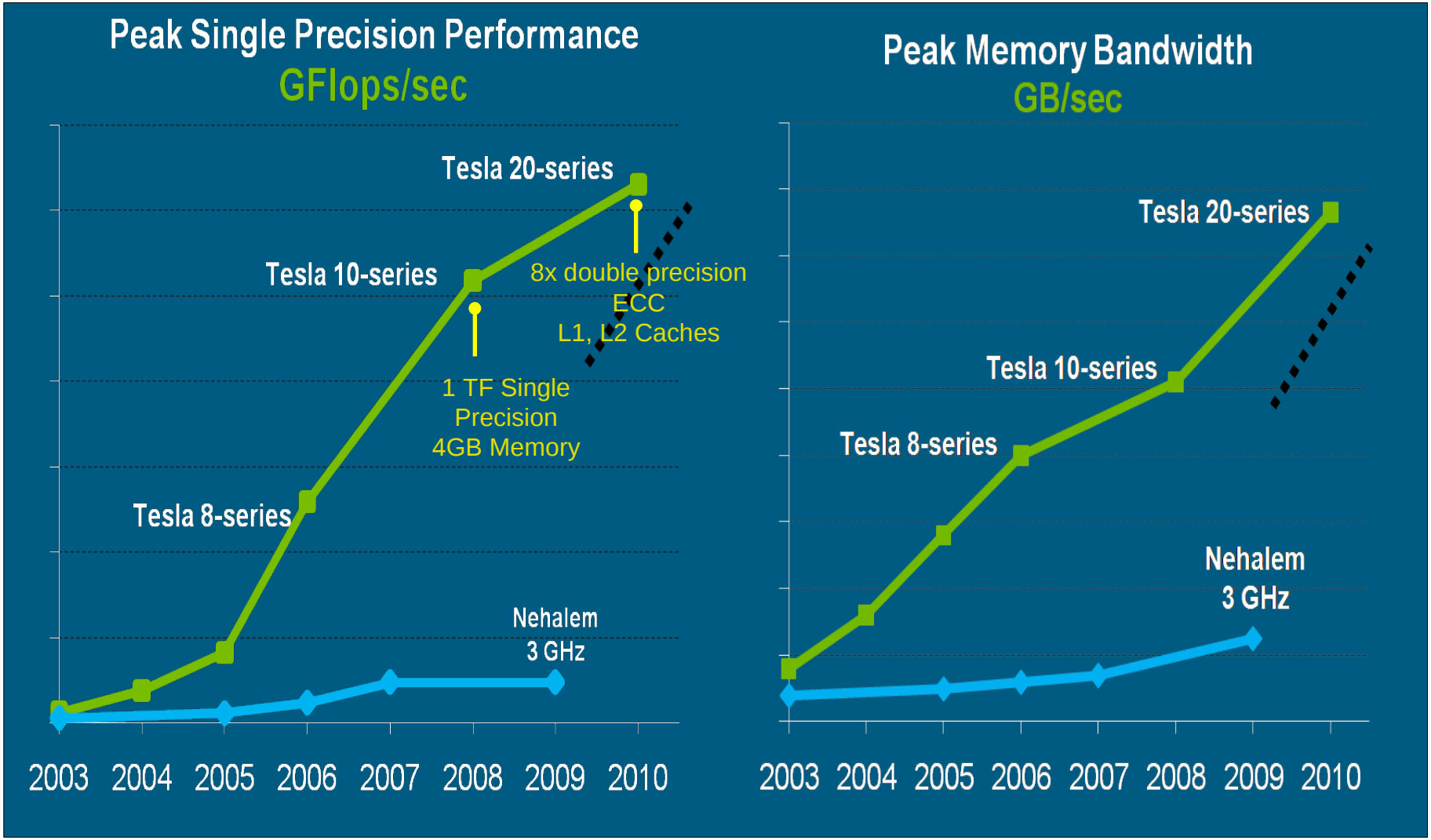
November 16, 2012 Jan H. Meinke

JUDGE

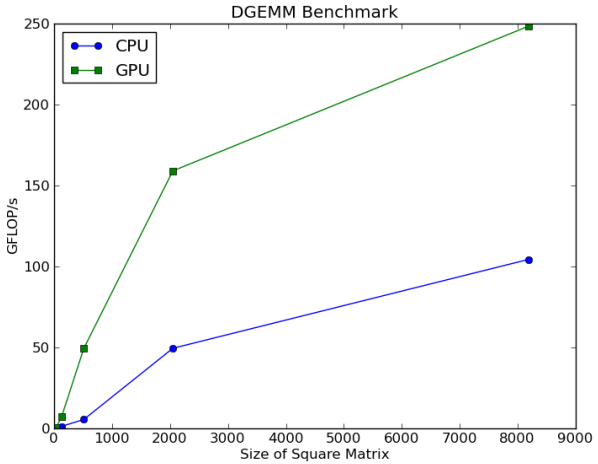
- Login and management nodes
- GPFS nodes
- 208 IBM System x iDataPlex dx360 M3
  - 2 Intel Xeon X5650 6-core processor 2,66 GHz, 96 GB, IB
  - 2 NVIDIA Tesla M2050/M2070 (Fermi), 3/6 GB memory
- 234 TFLOPS:
  - 26.3 TFLOPS (CPU)
  - 208 TFLOPS (GPU)

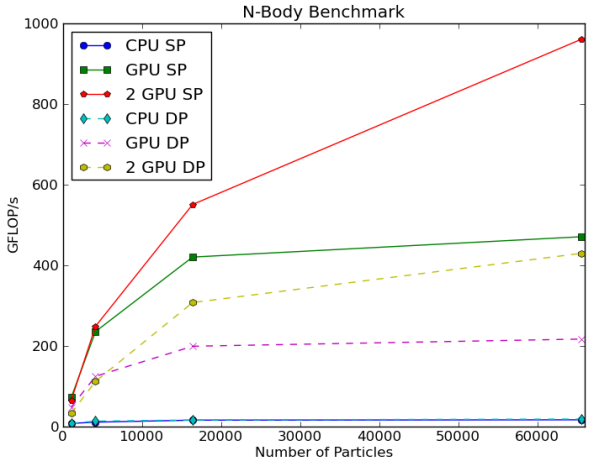
#157 Top500 06/2012

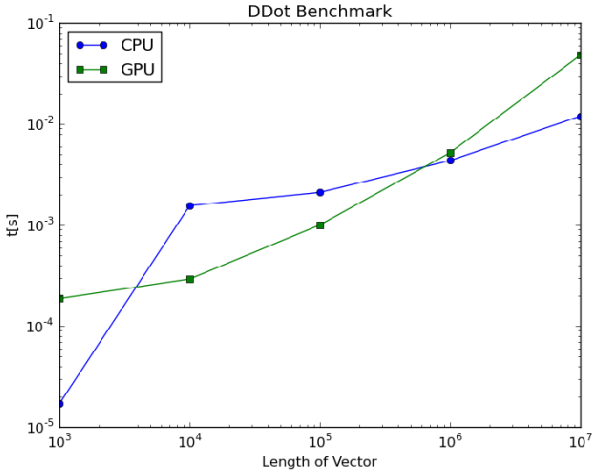
The Performance Gap Widens Further

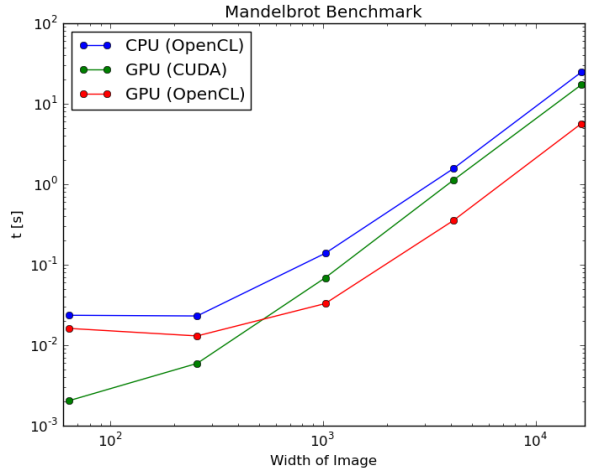


Getting a Feeling for GPU Performance









Programming GPUs

November 16, 2012

Don't!

Applications

Molecular Dynamics

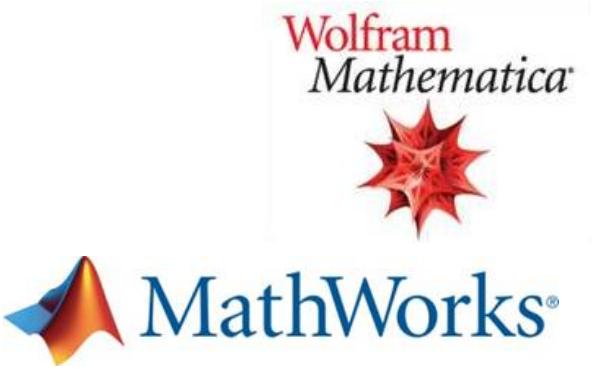
Amber

NAMD

Gromacs

Image Processing

Mathematics



CFD

Libraries

CUBLAS CUSPARSE CUFFT

CURAND Thrust CUSP

Using CUBLAS

- Initialize
- Allocate memory on the GPU
- Copy data to GPU
- Call BLAS routine
- Copy results to Host
- Finalize

Calculates $res = \sum_{i=0}^{n} A_i B_i$

- status = cublasCreate(&handle)
- cudaMalloc((void**)&d_A, n * sizeof(d_A[0]))
- status = cublasSetVector(n, sizeof(A[0]), A, 1, d_A, 1);
- status = cublasDdot(handle, n, d_A, 1, d_B, 1, &res)
- status = cublasDestroy(handle);

Exercise

So, you think you want to write your own GPU code...

Parallel Scaling Primer

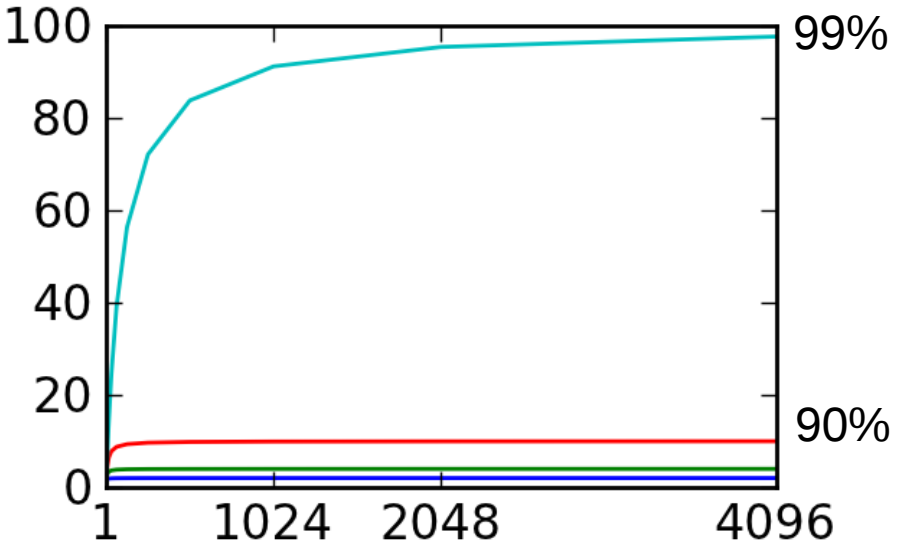
- What is the maximum speedup?

$$t = t_s + t_p$$

$$t(n) = t_s + t_p/n$$

$$s = t/t(n)$$

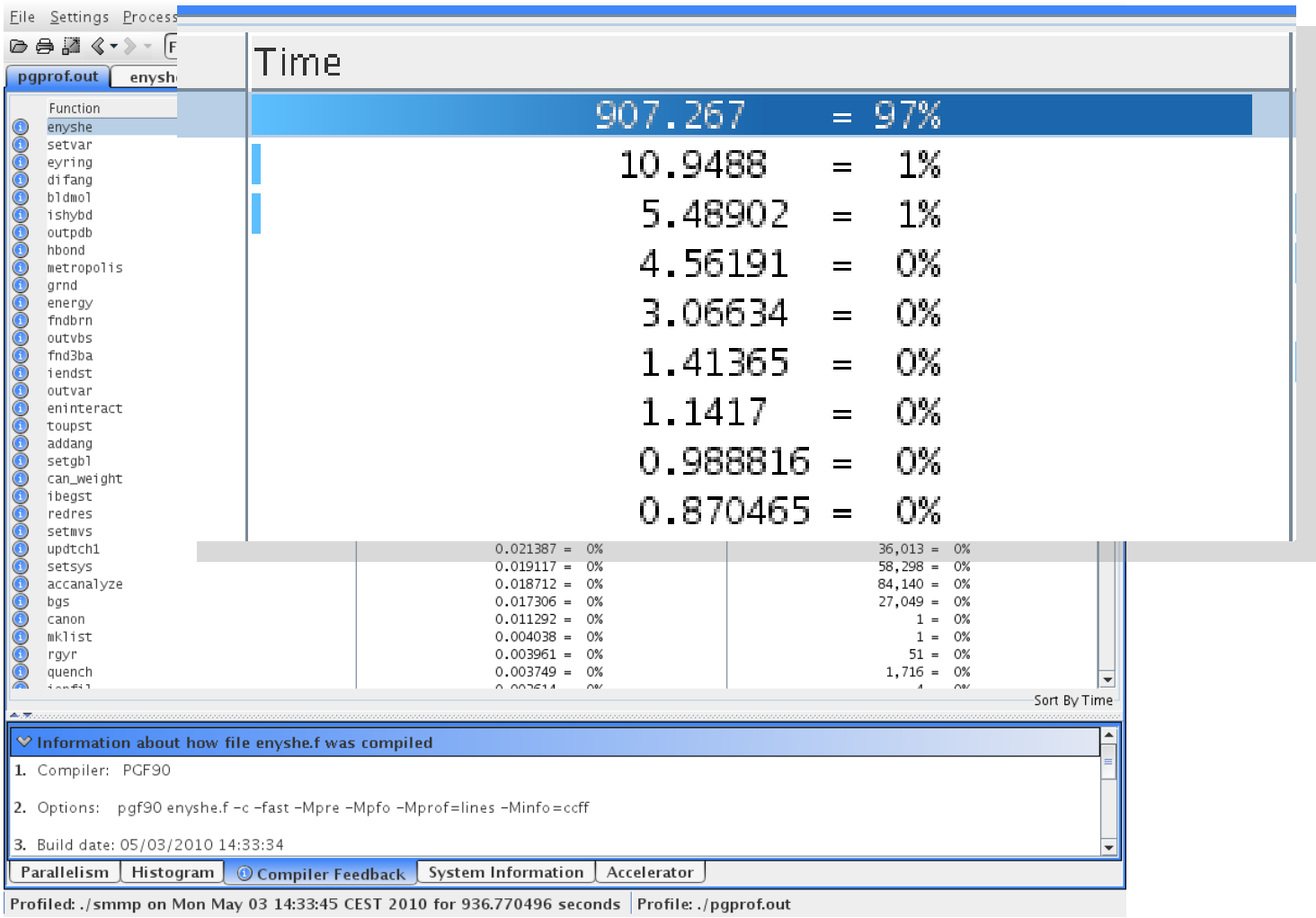
$$= \frac{t_s + t_p}{t_s + t_p/n}$$



Use Profiler

- PGI Profiler is an easy to use tool
  - Supports hardware counters if available
  - First run compiled with -Mpfi -Minfo=ccff
    then compiled with -Mpfo -Mprof=lines -Minfo=ccff
    (profile guided optimization + profiling)
  - Use pgcollect to collect profile
  - or skip profile guided optimization and just use
    -Mprof=lines -Minfo=ccff

Profiling Results



Things to consider

- Is my program computationally intensive?
- How much data needs to be transferred in and out?
- Is the gain worth the pain?

PGI Accelerator

- Pragma/directive based

  #pragma acc region in C

  !acc region … !acc end region in Fortran
- Some additional control statement
  - copyin/copyout
  - vector
  - acc_init
  - acc data region
  - ...

File Edit View Scrollback Bookmarks Settings Help

```
! Simple matmul example

      module mymm
      contains
      subroutine mm1(a, b, c, m)
        real, dimension(:,:) :: a, b, c
        do j = 1, m
           do i = 1, m
               a(i, j) = 0.0
           enddo
           do k = 1, m
               do i = 1, m
                   a(i, j) = a(i, j) + b(i, k) * c(k, j)
               enddo
           enddo
        enddo
      end subroutine
      end module
~
~
~
~
                                                    17,7          All
```

zam839 () jugipsy ()

File Edit View Scrollback Bookmarks Settings Help

```
! Simple matmul example

      module mymm
      contains
      subroutine mm1(a, b, c, m)
        real, dimension(:,:) :: a, b, c
!$acc region
        do j = 1, m
           do i = 1, m
               a(i, j) = 0.0
           enddo
           do k = 1, m
               do i = 1, m
                   a(i, j) = a(i, j) + b(i, k) * c(k, j)
               enddo
           enddo
        enddo
!$acc end region
      end subroutine
      end module
~
~
                                                    12,16         All
```

zam839 () jugipsy ()

File Edit View Scrollback Bookmarks Settings Help

```
meinke@jugipsy:~/tutorials/pgacc/matrixmatrixmultiplication> pgf90 -fast -M
info=all,accel mm2.f90 -c -ta=nvidia
mm1:
      7, Generating copyout(a(1:m,1:m))
         Generating copyin(c(1:m,1:m))
         Generating copyin(b(1:m,1:m))
      8, Loop is parallelizable
      9, Loop is parallelizable
         Accelerator kernel generated
          8, !$acc do parallel, vector(16)
          9, !$acc do parallel, vector(16)
     12, Loop carried reuse of 'a' prevents parallelization
     13, Loop is parallelizable
         Accelerator kernel generated
          8, !$acc do parallel, vector(16)
         12, !$acc do seq
             Cached references to size [16x16] block of 'b'
             Cached references to size [16x16] block of 'c'
         13, !$acc do parallel, vector(16)
             Using register for 'a'
meinke@jugipsy:~/tutorials/pgacc/matrixmatrixmultiplication>
```

zam839 () jugipsy ()

Exercise

module load pgi/12.3

CUDA 4.0: Thrust 1.4

- Template library similar to STL.
- Containers
- Algorithms
- Thrust 1.3 for CUDA 3.2

Thrust by Example

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

int main(void)
{
    // generate 16M random numbers on the host
    thrust::host_vector<int> h_vec(1 << 24);
    thrust::generate(h_vec.begin(), h_vec.end(), rand);

    // transfer data to the device
    thrust::device_vector<int> d_vec = h_vec;

    // sort data on the device
    thrust::sort(d_vec.begin(), d_vec.end());

    // transfer data back to host
    thrust::copy(d_vec.begin(), d_vec.end(), h_vec.begin());

    return 0;
}
```

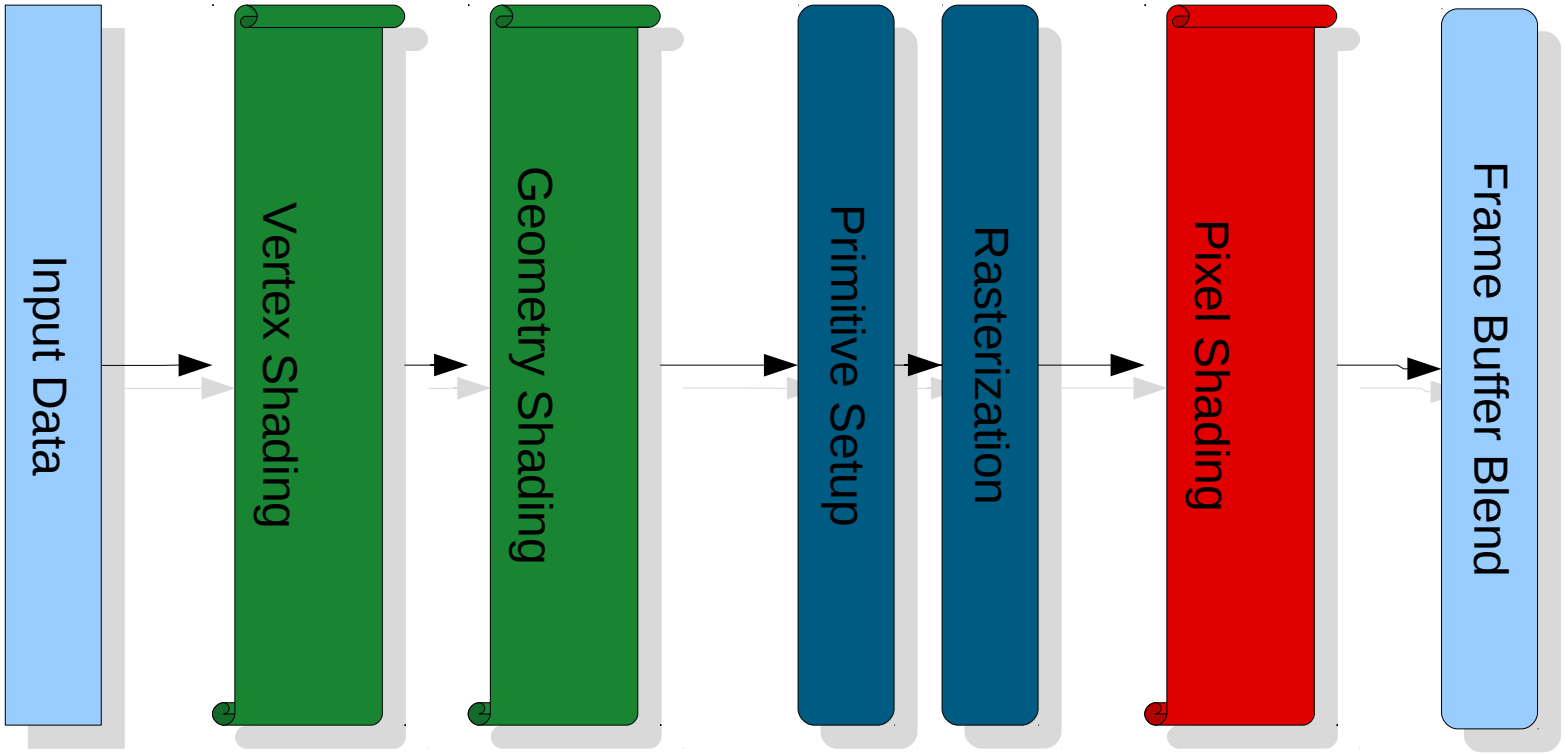
PyCUDA Example

```
from pycuda.reduction import ReductionKernel
dot = ReductionKernel(dtype out = numpy.float32, neutral="0",
        reduce_expr = "a+b", map_expr = "x[i] * y[i]",
        arguments = "const float *x, const float *y")

from pycuda.curandom import rand as curand

x = curand((1000 * 1000), dtype = numpy.float32)
y = curand((1000 * 1000), dtype = numpy.float32)

x_dot_y = dot(x, y).get()
x_dot_y_cpu = numpy.dot(x.get(), y.get())
```

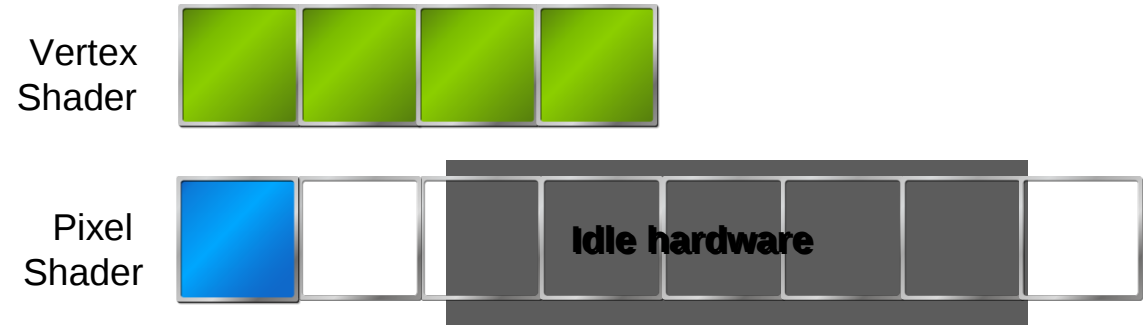
CUDA C Alternatives

- PGI Accelerator
- HMPP
- Thrust
- PyCUDA/PyOpenCL
- CUDA for Fortran
- OpenCL (Wednesday)

Graphics Pipeline

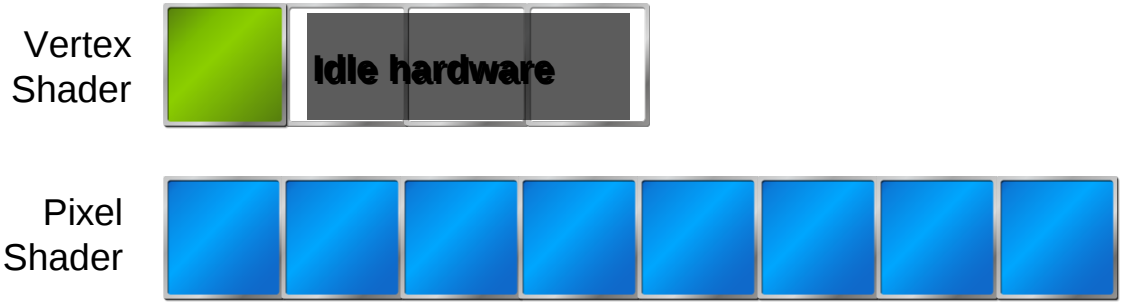


Previous Pipelined Architectures



Heavy Geometry

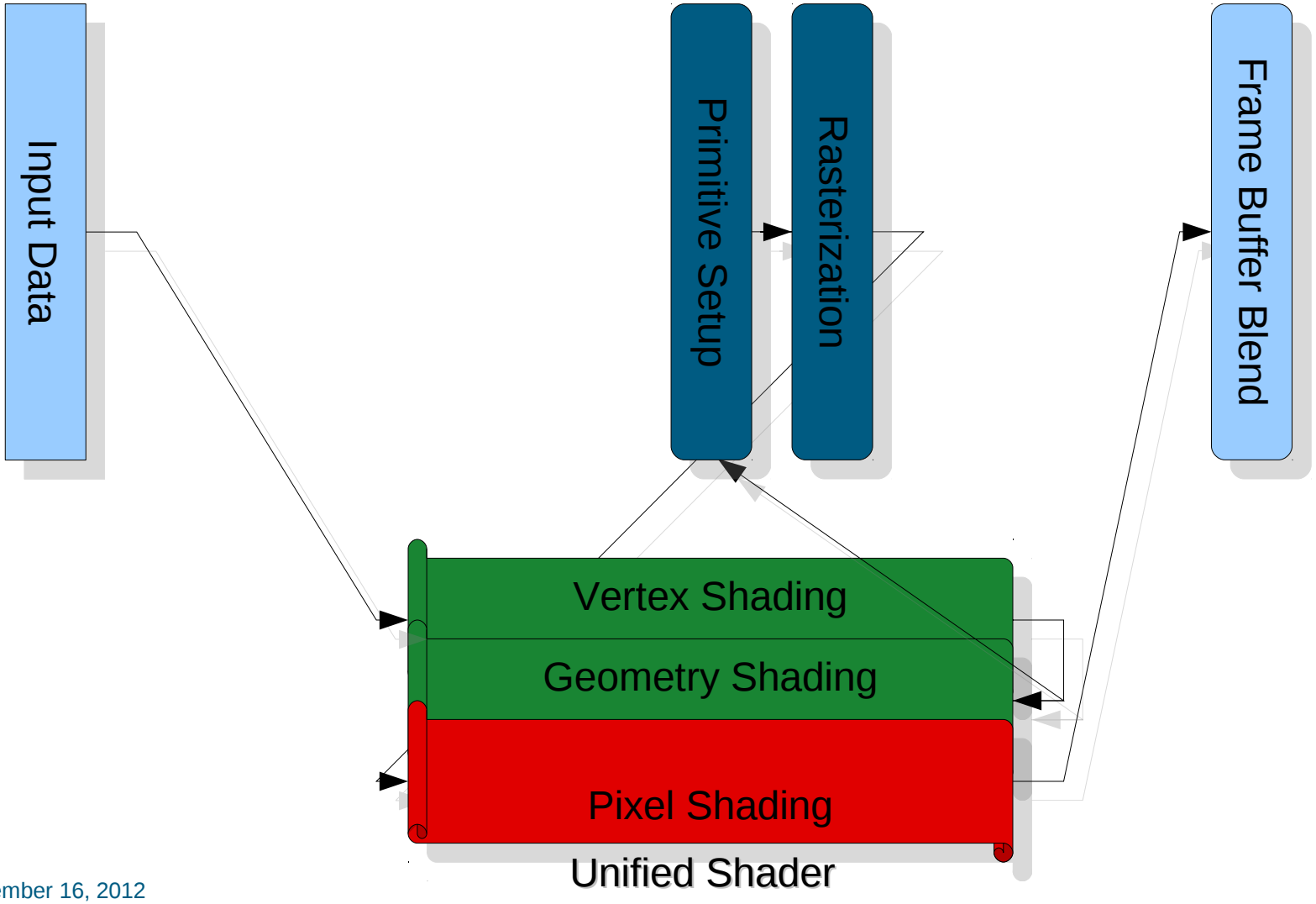
Workload Perf = 4



Heavy Pixel

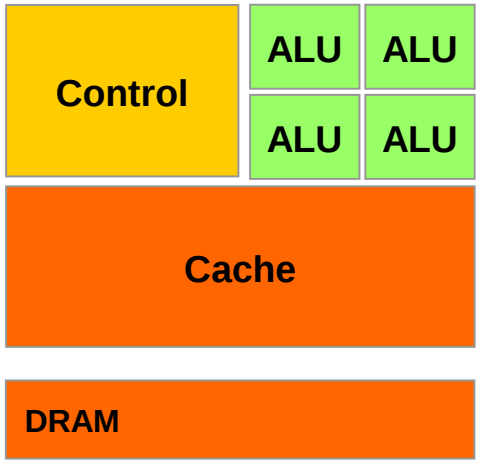
Workload Perf = 8

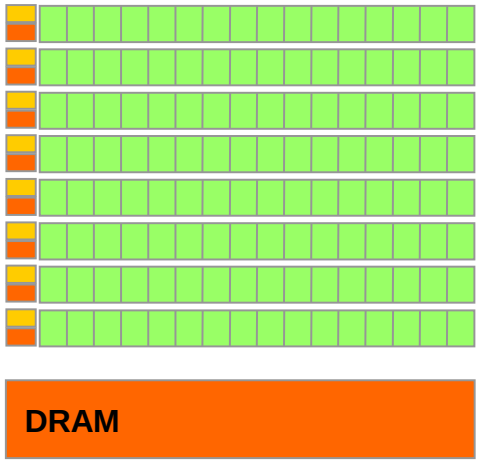
Graphics Pipeline



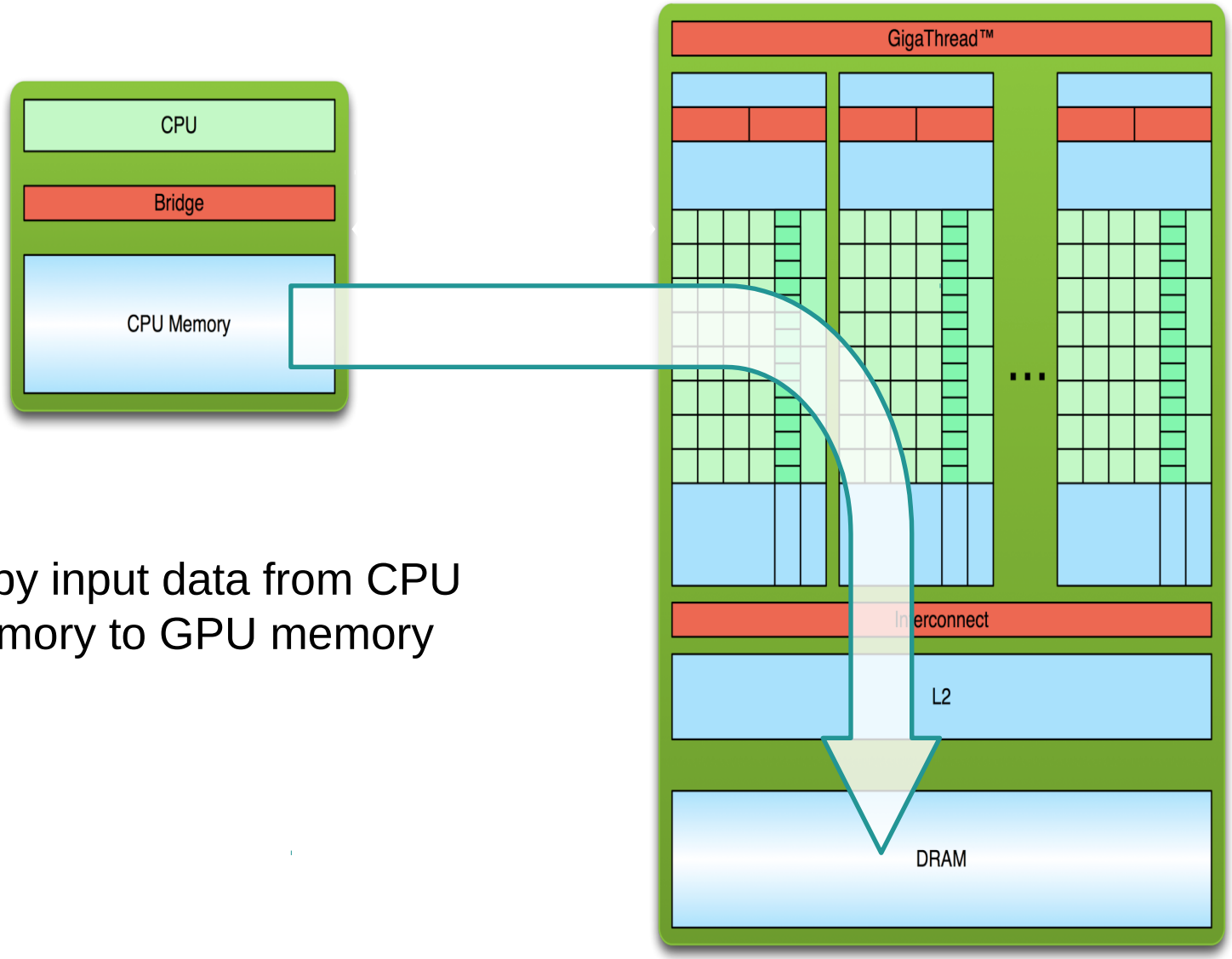
Low Latency or High Throughput?

- CPU
  - Optimized for low-latency access to cached data sets
  - Control logic for out-of-order and speculative execution
- GPU
  - Optimized for data-parallel, throughput computation
  - Architecture tolerant of memory latency
  - More transistors dedicated to computation



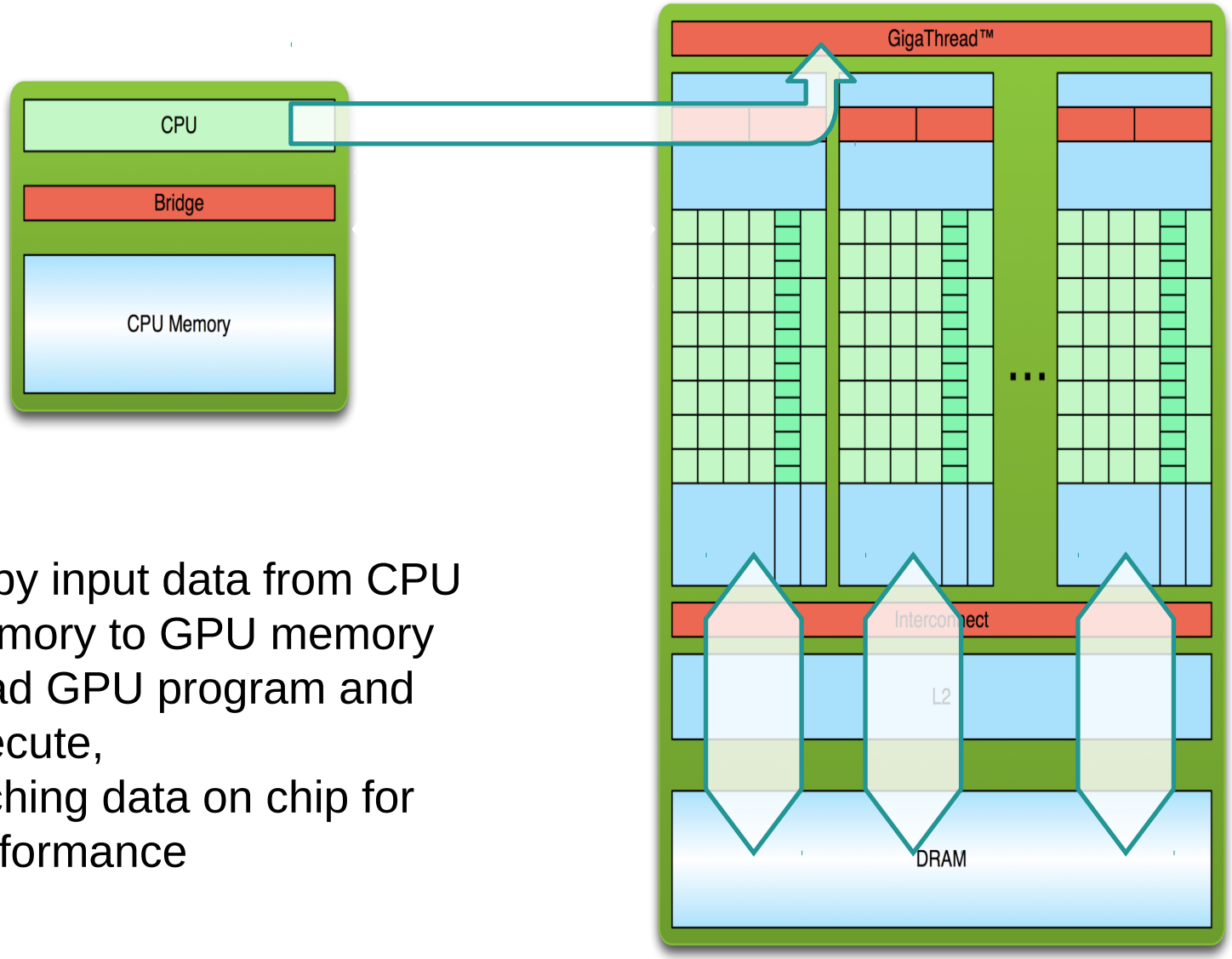


Processing Flow



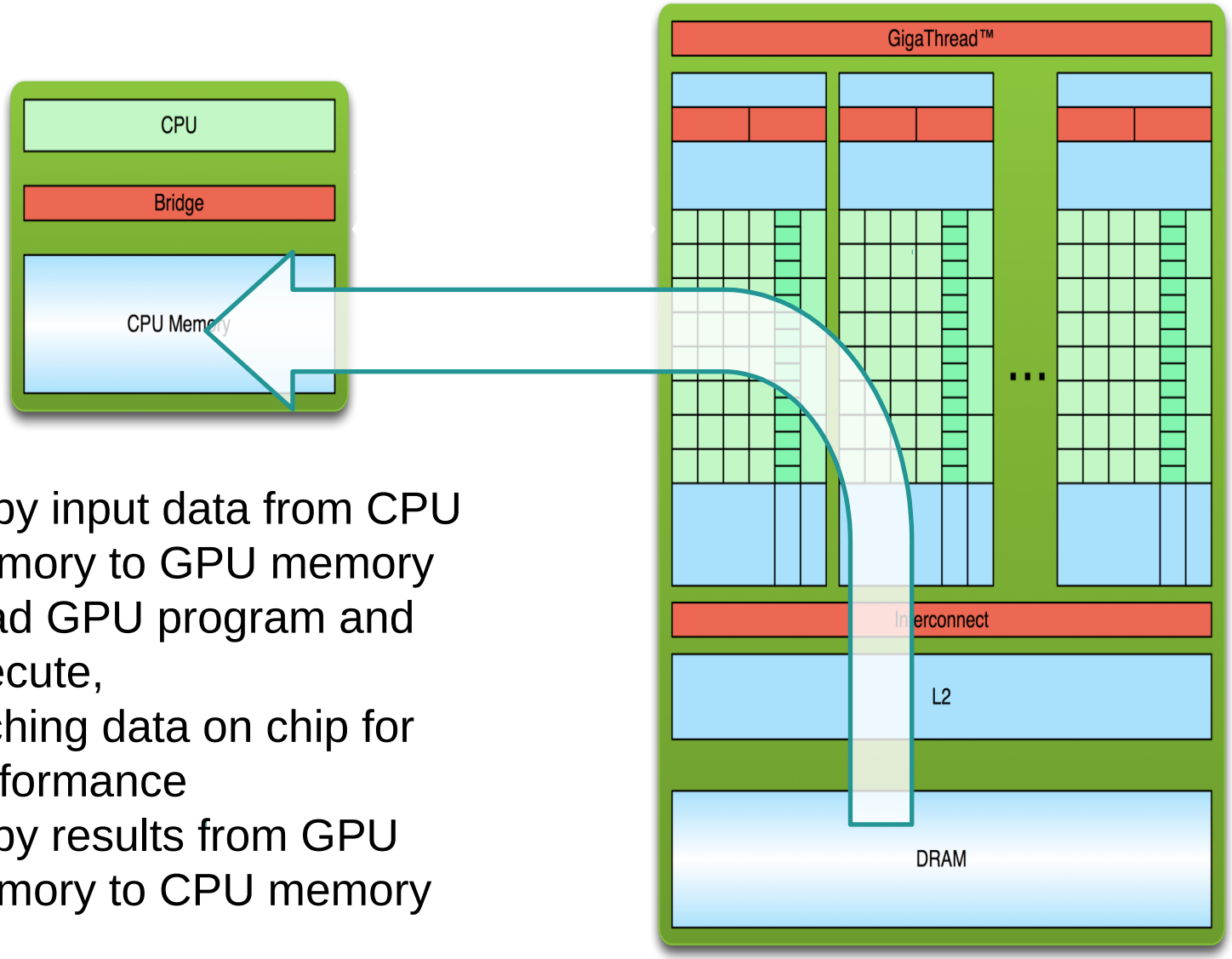
Copy input data from CPU memory to GPU memory

Processing Flow



Copy input data from CPU memory to GPU memory
Load GPU program and execute,
caching data on chip for performance

Processing Flow



Copy input data from CPU memory to GPU memory
Load GPU program and execute,
caching data on chip for performance
Copy results from GPU memory to CPU memory

Programming Model

November 16, 2012

Kernel (C)

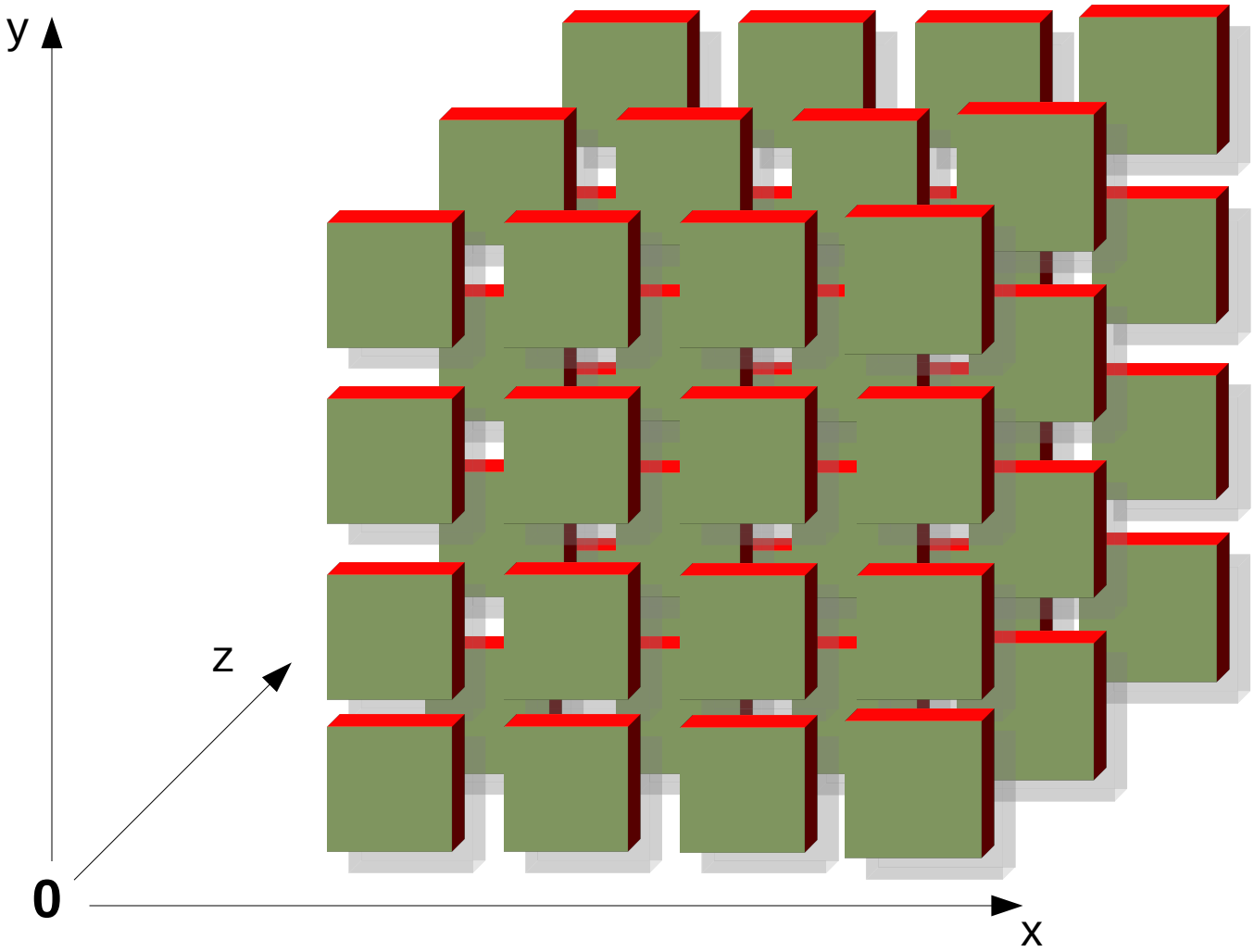
```
void mm(float* A, float* B, float* C, int m){
    for (int i = 0; i < m; ++i){
        for (int j = 0; j < m; ++j){
            for (int k = 0; k < m; ++k){
                C[i * m + j] += A[i * m + k] * B[k * m + j];
            }
        }
    }
}
```

Kernel (C)

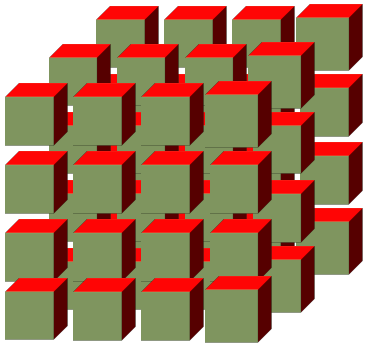
```
void mm_kernel(float* A, float* B, float* C, int m, int i, int j){
            for (int k = 0; k < m; ++k){
              C[i * m + j] += A[i * m + k] * B[k * m + j];
            }
}
```

```
void mm(float* A, float* B, float* C, int m){
    for (int i = 0; i < m; ++i){
        for (int j = 0; j < m; ++j){
              mm_kernel(A, B, C, m, i, j);
        }
    }
}
```

Splitting up the work



Splitting up the work



Thread Block
(4x4x3)

576 Threads

Grid
(4x3x1)

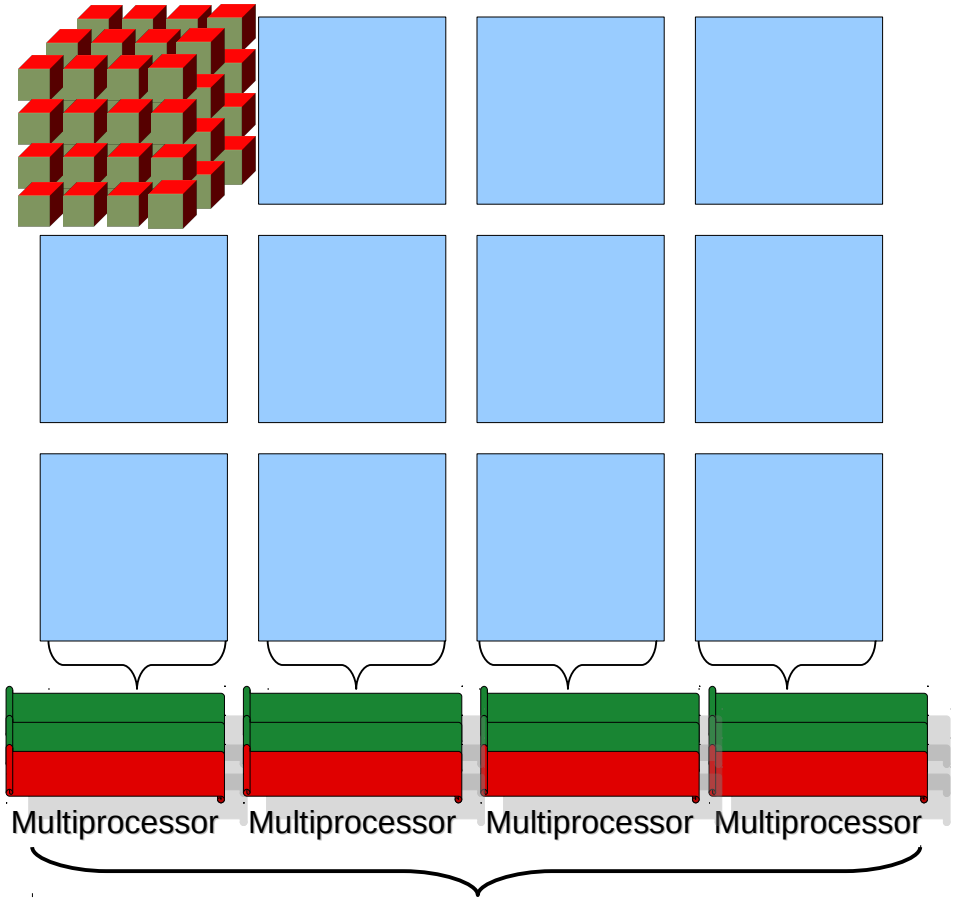
Calculating global coordinates

x = blockIdx.x * blockDim.x + threadIdx.x

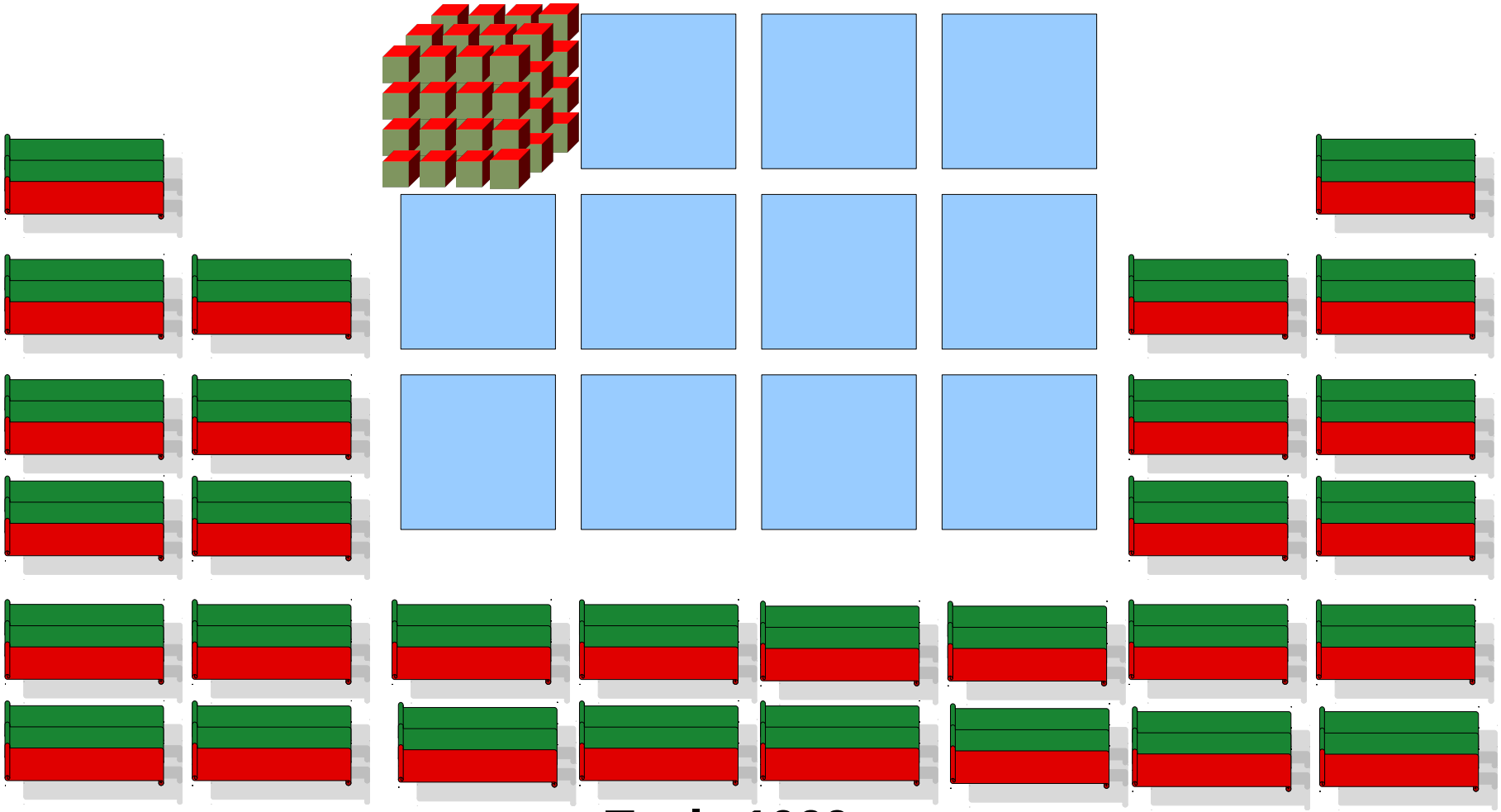
y = blockIdx.y * blockDim.y + threadIdx.y

z = blockIdx.z * blockDim.z + threadIdx.z

Splitting up the work



Splitting up the work



Tesla 1060

deviceQuery

There are 2 devices supporting CUDA

Device 0: "Tesla T10 Processor"

...

CUDA Capability Major revision number: 1
CUDA Capability Minor revision number: 3
Total amount of global memory: 4294770688 bytes
Number of multiprocessors: 30
Number of cores: 240

...

Warp size: 32
Maximum number of threads per block: 512
Maximum sizes of each dimension of a block: 512 x 512 x 64
Maximum sizes of each dimension of a grid: 65535 x 65535 x 1

...

Clock rate: 1.30 GHz
Concurrent copy and execution: Yes
Run time limit on kernels: No
Integrated: No
Support host page-locked memory mapping: Yes
Compute mode: Default (multiple host threads can use this device simultaneously)

Exercise

- Make sure you loaded the pgi compiler using `module load pgi/12.3`
- Run `pgaccelinfo`

JÜLICH
FORSCHUNGSZENTRUM

Getting started with CUDA

November 16, 2012

Kernel (CUDA)

```
__global__ void mm_kernel(float* A, float* B, float* C, int m){

    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;

        for (int k = 0; k < m; ++k){
          C[i * m + j] += A[i * m + k] * B[k * m + j];
        }
}
```

Getting data in and out

- GPU has separate memory
- Allocate memory on device
- Transfer data from host to device
- Transfer data from device to host
- Free device memory

Allocate memory on device

```
cudaMalloc(void** pointer, size_t nbytes)
```

Example:

```
// Allocate a vector of 2048 floats on device
float * a_gpu;
int n = 2048;
cudaMalloc((void**) &a_gpu, n * sizeof(float));
```

Cast to void**

Address of pointer

Get size of a float

Copy from host to device

```
cudaMemcpy(void* dst, void* src, size_t nbytes,
           enum cudaMemcpyKind  dir)
```

Example:

```
// Copy vector of  floats a of length n=2048 to a_gpu on device
cudaMemcpy(a_gpu, a, n * sizeof(float),
                                  cudaMemcpyHostToDevice);
```

Copy from device to host

```
cudaMemcpy(void* dst, void* src, size_t nbytes,
           enum cudaMemcpyKind  dir)
```

Example:

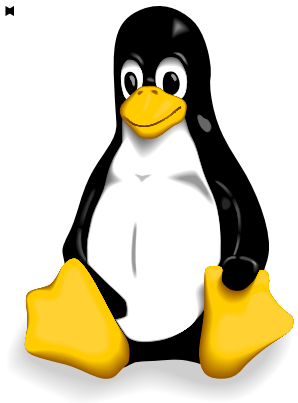
```
// Copy vector of floats a_gpu of length n=2048 to a on host
cudaMemcpy(a, a_gpu, n * sizeof(float),
                                cudaMemcpyDeviceToHost);
```

Note the order

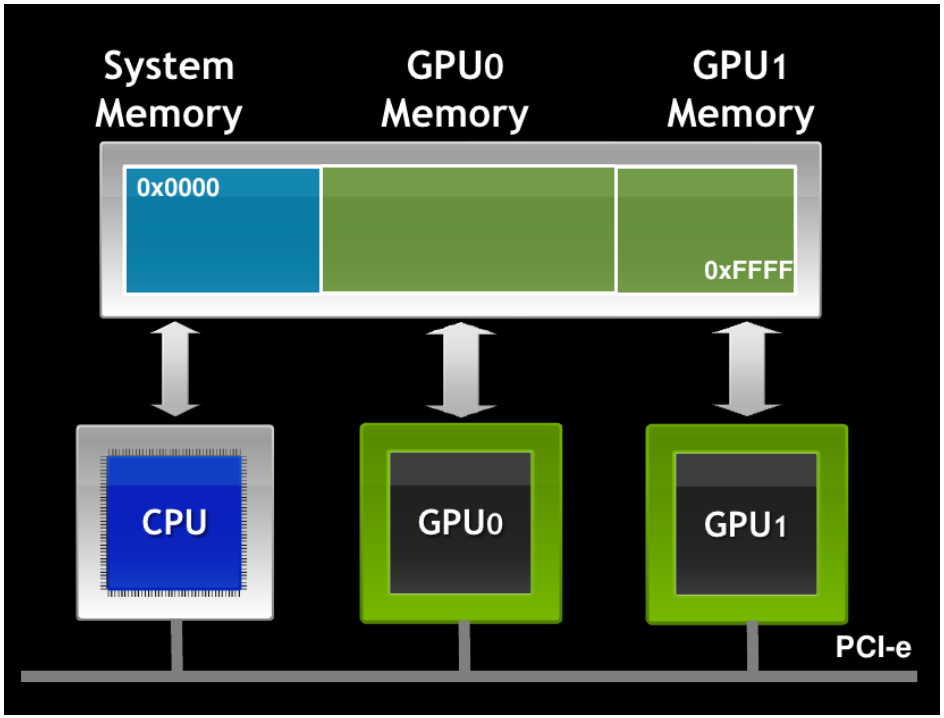
Changed flag

Unified Virtual Address Space (UVA)

Windows



64bit



2.0

cudaMalloc*(...)
cudaHostAlloc(...) } return UVA pointers
cudaMemcpy*(..., cudaMemcpyDefault)

Free device memory

```
cudaFree(void* pointer)
```

Example:

```
// Free the memory allocated by a_gpu on the device
cudaFree(a_gpu);
```

Getting data in and out

- Allocate memory on device

```
cudaMalloc(void** pointer, size_t nbytes)
```

- Transfer data between host and device

```
cudaMemcpy(void* dst, void* src, size_t nbytes,
           enum cudaMemcpyKind  dir)
dir = cudaMemcpyHostToDevice
dir = cudaMemcpyDeviceToHost
```

- Free device memory

```
cudaFree(void* pointer)
```

Calling the kernel

- Define dimensions of thread block
- Define dimensions of grid
- Call the kernel

Define dimensions of thread block

```
dim3 blockDim(size_t blockDimX, size_t blockDimY,
              size_t blockDimZ)
```

On Judge (Tesla 2050):

- Max. dim. of a block: 1024 x 1024 x 64
- Max. number of threads per block: 1024

Example:

```
// Create 3D thread block with 512 threads
dim3 blockDim(16, 16, 2);
```

Define dimensions of grid

```
dim3 gridDim(size_t blockDimX, size_t blockDimY,
             size_t blockDimZ)
```

On Judge (Tesla 2050):

- Max. dim. of a grid: 65535 x 65535 x 1

Example:

```
// Dimension of problem: nx x ny = 1000 x 1000
dim3 blockDim(16, 16) // Don't need to write z = 1
int gx = (nx % blockDim.x==0) ? nx / blockDim.x : nx / blockDim.x + 1
int gy = (ny % blockDim.y==0) ? ny / blockDim.x : ny / blockDim.y + 1
dim3 gridDim(gx, gy);
```

Watch out!

Call the kernel

```
kernel<<<dim3 gridDim, dim3 blockDim>>>([arg]*)
```

Call returns immediately!

Example:

```
// Dimensions as defined in previous slides. The variables
// a_gpu, b_gpu ,c_gpu are arrays, m is the dimension of
// a square matrix
mm_kernel<<<gridDim, blockDim>>>(a_gpu, b_gpu, c_gpu, m)
```

Calling the kernel

- Define dimensions of thread block

```
dim3 blockDim(size_t blockDimX, size_t blockDimY,
              size_t blockDimZ)
```

- Define dimensions of grid

```
dim3 gridDim(size_t gridDimX, size_t gridDimY,
             size_t gridDimZ)
```

- Call the kernel

```
kernel<<<dim3 gridDim, dim3 blockDim>>>([arg]*)
```

Exercise

Exercise Scale Vector

Allocate memory on device

```
cudaMalloc(void** pointer, size_t nbytes)
```

Transfer data between host and device

```
cudaMemcpy(void* dst, void* src,
           size_t nbytes,
           enum cudaMemcpyKind dir)
 dir = cudaMemcpyHostToDevice
 dir = cudaMemcpyDeviceToHost
```

- Free device memory

```
cudaFree(void* pointer)
```

Define dimensions of thread block

```
dim3 blockDim(size_t blockDimX,
              size_t blockDimY,
              size_t blockDimZ)
```

Define dimensions of grid

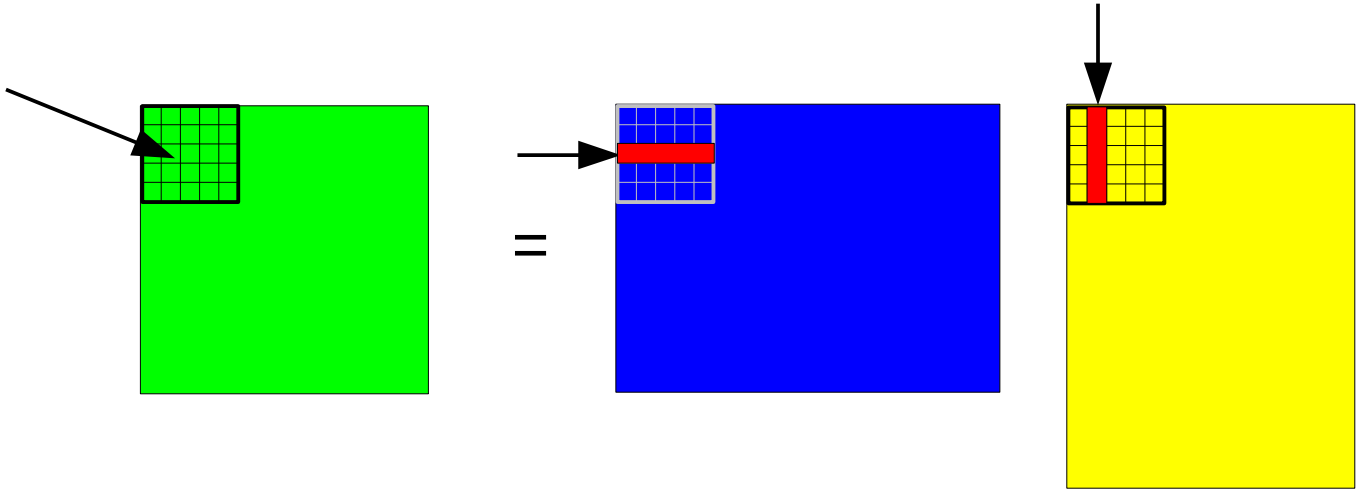
```
dim3 gridDim(size_t gridDimX, size_t
             gridDimY,
             size_t gridDimZ)
```

Call the kernel

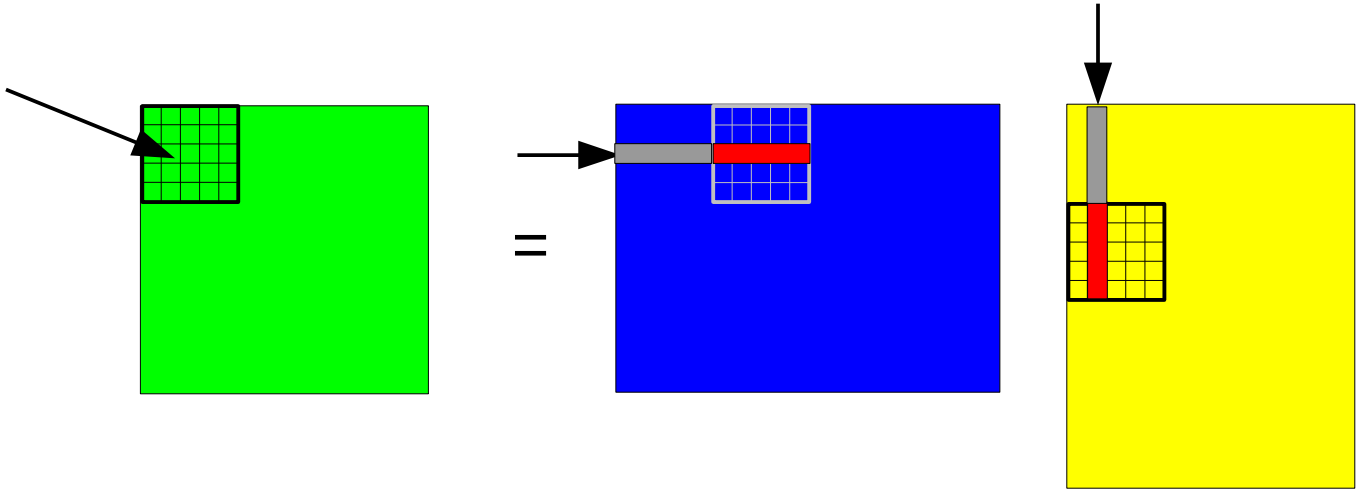
```
kernel<<<dim3 gridDim,
         dim3 blockDim>>>([arg]*)
```

Compile with nvcc -o scale_vector scale_vector.cu

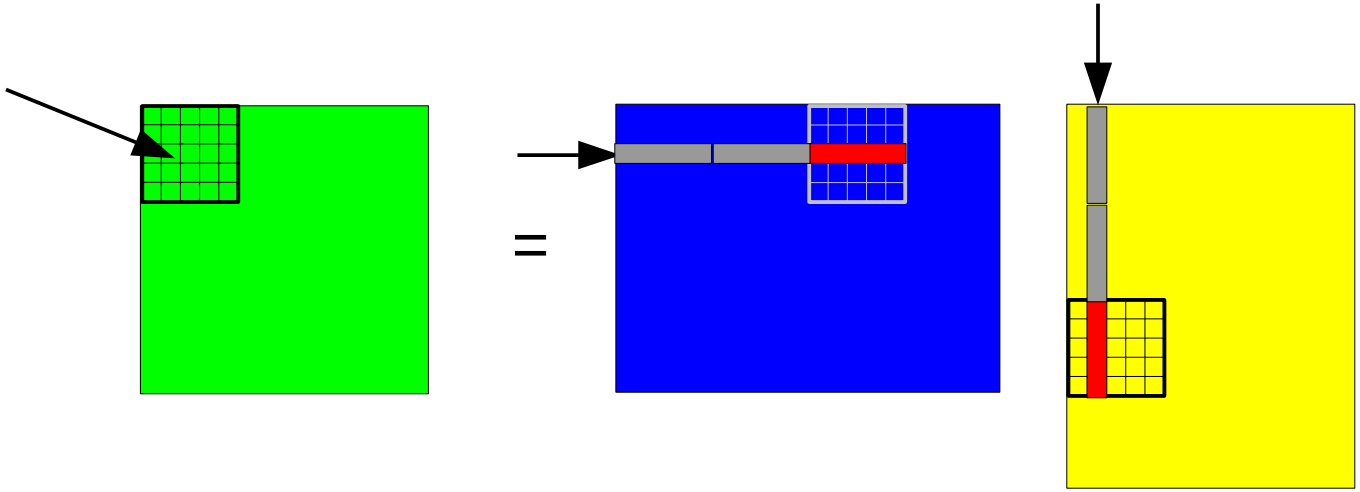
Blockwise Matrix-Matrix Multiplication



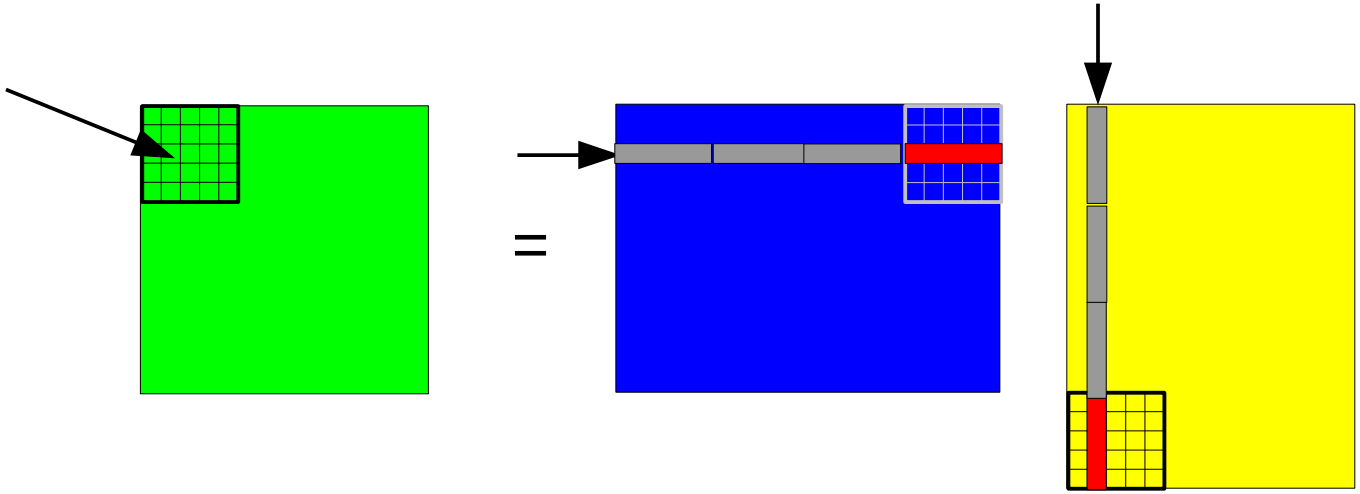
Blockwise Matrix-Matrix Multiplication



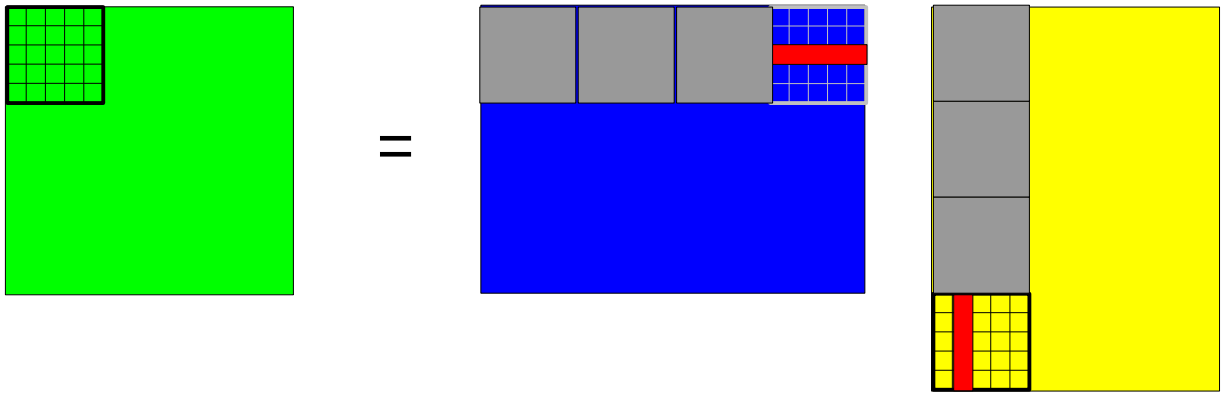
Blockwise Matrix-Matrix Multiplication



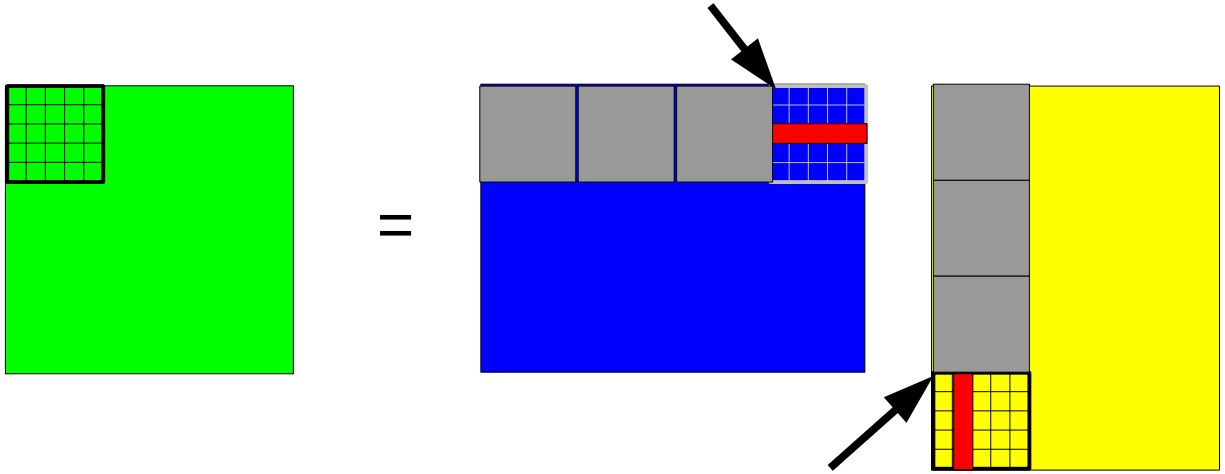
Blockwise Matrix-Matrix Multiplication



Blockwise Matrix-Matrix Multiplication



Blockwise Matrix-Matrix Multiplication



Thread block loops over blocks in blue and yellow matrix:

- Calculate upper left corner
- Load data into shared memory
- Do calculation (one thread is still responsible for an element)
- Add partial sum to result

JÜLICH
FORSCHUNGSZENTRUM

OpenCL and C++11

November 16, 2012

OpenCL C++ Bindings

- Khronos provides C++ binding specification
  - Header file: **cl.hpp**
  - Documentation available at Khronos web page

http://www.khronos.org/registry/cl/specs/opencl-cplusplus-1.1.pdf

  - Namespace **cl**

Getting a Platform

```
int main(int argc, char** argv){
    // Get a list of platforms
    std::vector<cl::Platform> platforms;
    cl::Platform::get(&platforms);
    assert(platforms.size() > 0);

    // Loop over platforms to find a GPU
    std::vector<cl::Device> devices;
    for (auto p :  platforms){
        p.getDevices(CL_DEVICE_TYPE_GPU, &devices);
        if (devices.size() > 0) break;
    }
    assert(devices.size() > 0);
    assert(devices[0].getInfo<CL_DEVICE_TYPE>() ==
CL_DEVICE_TYPE_GPU);
```

Create Context and Queue

```
cl::Context context(devices);
cl::CommandQueue queue(context, devices[0], 0);
```

Load Source and Compile Program

```
cl::Program::Sources source(1, std::make_pair(kernelSource, strlen(
                        kernelSource)));
cl::Program program(context, source);
try {
    program.build(devices, "-x clc++");
}
catch (cl::Error er) {
    std::cout << "Build Status: " << program.getBuildInfo<
    CL_PROGRAM_BUILD_STATUS>(devices[0]) << std::endl;
    std::cout << "Build Options:\t" << program.getBuildInfo<
    CL_PROGRAM_BUILD_OPTIONS>(devices[0]) << std::endl;
    std::cout << "Build Log:\t " << program.getBuildInfo<
    CL_PROGRAM_BUILD_LOG>(devices[0]) << std::endl;
    exit(-1);
}
```

Create the Kernel

```
cl::Kernel kernel(program, "foo");
```

Calling the kernel

```
// Set the kernel argument.
kernel.setArg(0, sizeof(&classObj), &classObj);
// Execute kernel
try {
    queue.enqueueNDRangeKernel(kernel, cl::NullRange,
        cl::NDRange(1), cl::NullRange, NULL, &event);
}
catch (cl::Error er){
        std::cerr << "Error: " << er.what() << "(" << er.err() << ")"
                  << std::endl;
        exit(-1);
}
```

OpenCL Static C++ Kernel Language Extension

- AMD released a preview with AMD APP SDK 2.6
- Needs testing driver
- Supports
  - templates
  - classes
- Doesn't support
  - virtual
  - dynamic memory allocation
  - exceptions

A Simple Template

```
template<class T> kernel void mm_kernel(T* A, T* B, T* C, int m){
    int i = get_global_id(0);
    int j = get_global_id(1);
    T tmpC = 0;
    for(int k = 0; k < m; ++k){
        tmpC += A[i * m + k] * B[k * m + j];
    }
};

template __attribute__((mangled_name(mmfloat)))
        kernel void mm_kernel(float* A, float* B, float* C, int m);
```

Using a Class from a Kernel

```
kernel void foo(__global Test* testClass){
    if (get_global_id(0) == 0){
        int x = testClass->getX() / 2;
        testClass->setX(x);
    }
}
```

A Simple Class

```
class Test {
public:
    void setX(int value);
    int getX();
private:
    int x;
};

void Test::setX(int value){
        x = value;
}
int Test::getX(){
    return x;
}
```

```
Test testClass;
void* mappedPtr=NULL;
// Create device buffer and map it to host space.
cl::Buffer classObj(context, CL_MEM_USE_HOST_PTR, sizeof(Test),
        &testClass);
try {
        mappedPtr = queue.enqueueMapBuffer(classObj, CL_TRUE,
        CL_MAP_READ | CL_MAP_WRITE, 0, sizeof(Test));
} catch (cl::Error er){
        std::cerr << "Error: " << er.what() << "(" << er.err() << ")" <<
        std::endl;
        exit(-1);
}
testClass.setX(10);
// Unmap from host space and transfer changes to device
queue.enqueueUnmapMemObject(classObj, mappedPtr, NULL, &event);
event.wait();
```