CSE 591: GPU Programming

CUDA Libraries and OpenCL

Klaus Mueller

Computer Science Department

Stony Brook University

CUDA Libraries

CUBLAS (BLAS = Basic Linear Algebra Subprograms)

level1 (scalar, vector, vector-vector)

level2 (matrix-vector) , level3 (matrix-matrix)

```
void cublasSsymv(char uplo, int n, float alpha, const float *A, int lda, const
   float *x, int incx, float beta, float *y, int incy)
```

performs the matrix-vector operation where *alpha* and *beta* are single-precision scalars, and *x* and *y* are n-element single-precision vectors. *A* is a symmetric $n \times n$ matrix that consists of single-precision elements and is stored in either upper or lower storage mode

CUBLAS Example

Compute a vector's L2 norm

$$\|\mathbf{x}\| := \sqrt{x_1^2 + \cdots + x_n^2}$$

- Single precision

```
float cublasSnrm2 (int n, const float *x, int incx)
```

- Double precision

```
double cublasDnrm2 (int n, const double *x, int incx)
```

```
cublasInit();
float *h_A;
h_A = (float*)malloc(n * sizeof(h_A[0]));
…
cublasAlloc(n, sizeof(d_A[0]), (void**)&d_A);
cublasSetVector(n, sizeof(h_A[0]), h_A, 1, d_A, 1);
float norm2result=cublasSnrm2 (n, d_A, 1);
cublasFree(d_A); free(h_A);
cublasShutdown();
```

initialize library

initialize vector

data transfer

compute norm

Wrap-up

CUDA Libraries (3rd party)

MAGMA (porting from LAPACK to GPU+multicore architectures)

CULA (3rd party implementation of LAPACK)

PyCUDA (CUDA via Python)

Thrust (C++ template for CUDA, open source)

Jasper for DWT (Discrete wavelet transform)

OpenViDIA for computer vision

CUDPP for radix sort

Thrust: Introduction

Offers

- STL compatible containers (vector, list, map)
- ~50 algorithm (reduction, prefix sum, sorting)
- Rapid prototyping

Container

- Hides cudaMalloc & cudaMemcpy
- Iterators behave like pointer

Thrust Example: Sorting

generate 16M random numbers on the host

```
thrust::host_vector<int> h_vec(16*1024*1024);

thrust::generate(h_vec.begin(), h_vec.end(), rand);

thrust::device_vector<int> d_vec = h_vec;

thrust::sort(d_vec.begin(), d_vec.end());

thrust::copy(d_vec.begin(), d_vec.end(), h_vec.begin());
```

transfer data to the device

sort data on the device

transfer data back to host

Thrust: Operators

```
thrust::device_vector<int>   i_vec = ...
thrust:: device_vector<float> f_vec = ...
```

declare storage

```
thrust:: reduce(i_vec.begin(), i_vec.end());
thrust:: reduce(i_vec.begin(), i_vec.end(),    0, thrust:: plus<int>());
```

sum of integers (equivalent calls)

```
thrust:: reduce(f_vec.begin(), f_vec.end());
thrust:: reduce(f_vec.begin(), f_vec.end(), 0.0f, thrust:: plus<float>());
```

sum of floats (equivalent calls)

```
thrust:: reduce(i_vec.begin(), i_vec.end(),    0, thrust:: maximum<int>());
```

maximum of integers

Thrust Example: Vector L2 Norm

More like C++

$$\|\mathbf{x}\| := \sqrt{x_1^2 + \cdots + x_n^2}$$

```
template <typename T> struct square
{ __host__ __device__
T operator()(const T& x) const {
        return x * x;       }
};
 square<float>      unary_op;
plus<float> binary_op;
float init = 0;

device_vector<float>  A(3);
A[0] = 20;  A[1] = 30;  A[2] = 40;
float norm = sqrt( transform_reduce(A.begin(), A.end(),  unary_op, init, binary_op));
```

define transformation f(x) -> x^2

setup arguments

initialize vector

compute norm

To Probe Further

NVIDIA CUDA Zone:

- http://www.nvidia.com/object/cuda_home.html
- Lots of information and code examples
- NVIDIA CUDA Programming Guide

GPGPU community:

- http://www.gpgpu.org
- User forums, tutorials, papers
- Good source: conference tutorials http://www.gpgpu.org/developer/index.shtml#conference-tutorial

OpenCL

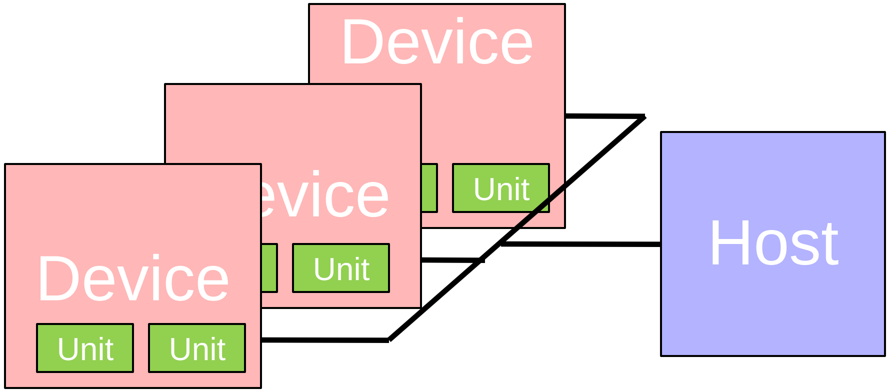
OpenCL: Open Computing Language (based on C)

- support for heterogeneous devices (GPU, CPU, …)
- pick the device best suited for the job
- potential parallelism is key for selection
- recall Amdahl’s law

OpenCL Mindset

Platform model:

- a host is connected to one or more OpenCL devices
- a device is divided into one or more compute units (cores)
- compute units are divided into one or more processing elements



OpenCL Mindset

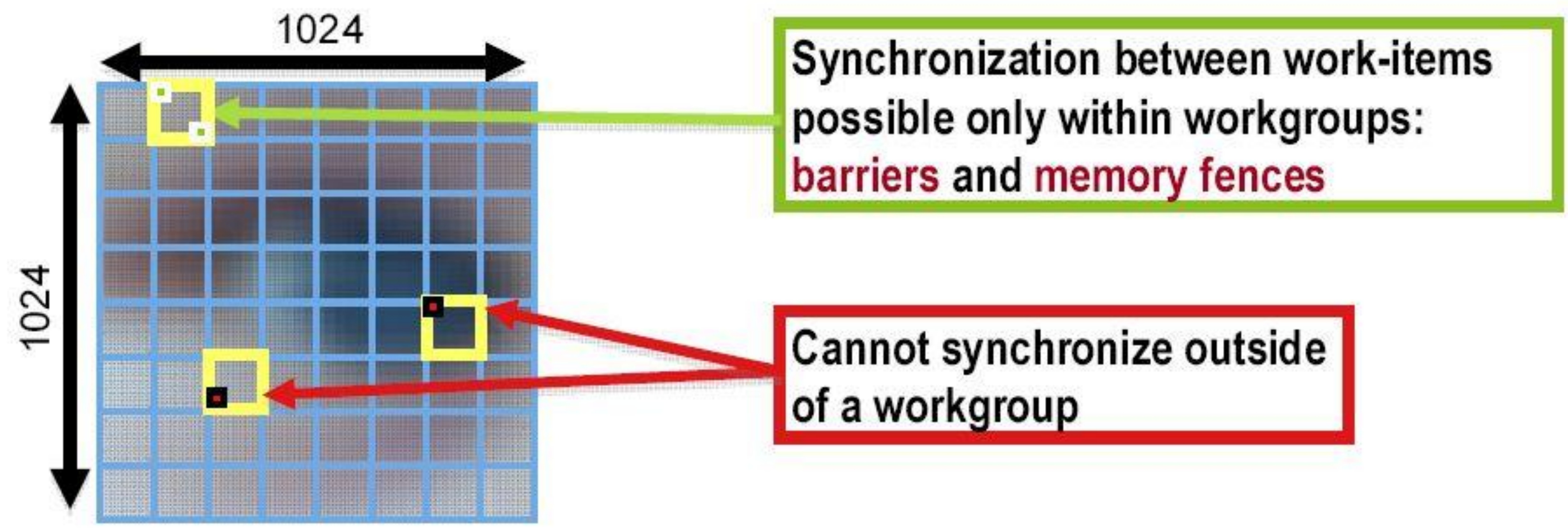
Execution Model

- host programs execute on the host
- kernels execute on one or more OpenCL devices
- each instance of a kernel is called a *work item*
- work items are organized as *work groups*
- work groups and work items are defined into an *index space*
- index space is created upon kernel submission
- work items can be identified by work group and local work item IDs

→ this is all quite similar to CUDA

CUDA Terminology	OpenCL Terminology
Grid	Index Space
Block	Work Group
Thread	Work Item

Global and Local Dimensions



from: Khronos OpenCL Overview

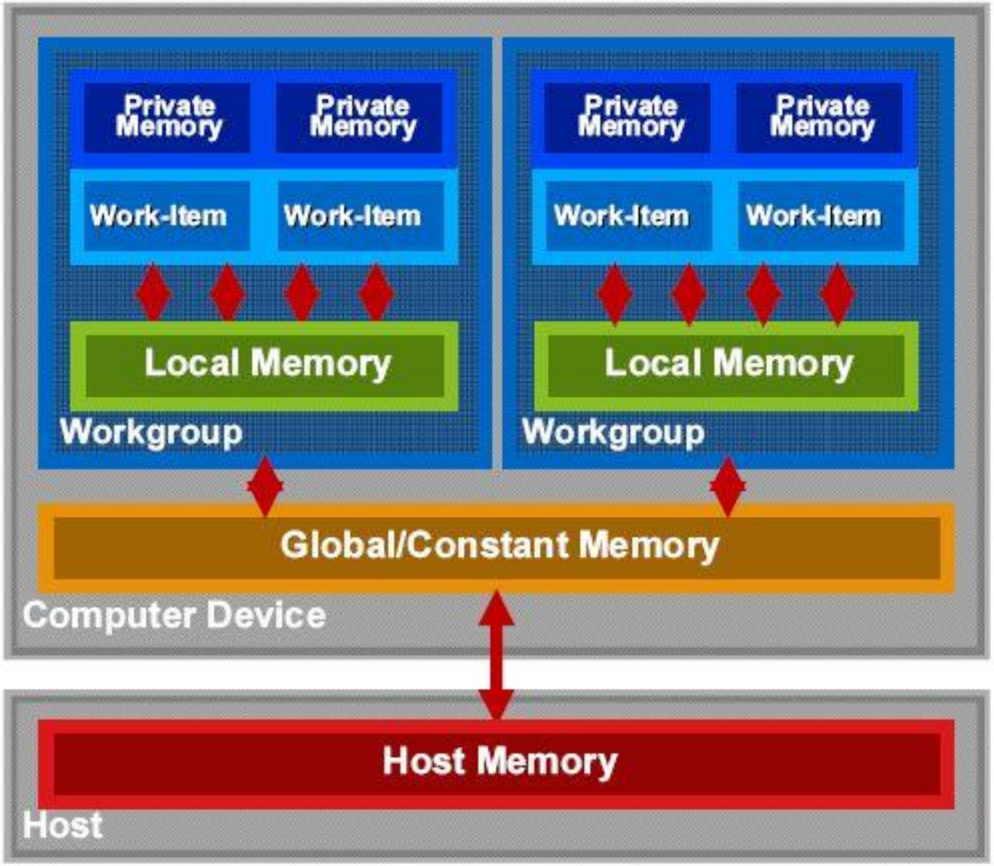
OpenCL Memory Model

Private memory
- per work item

Local memory (16kB)
- shared per work group

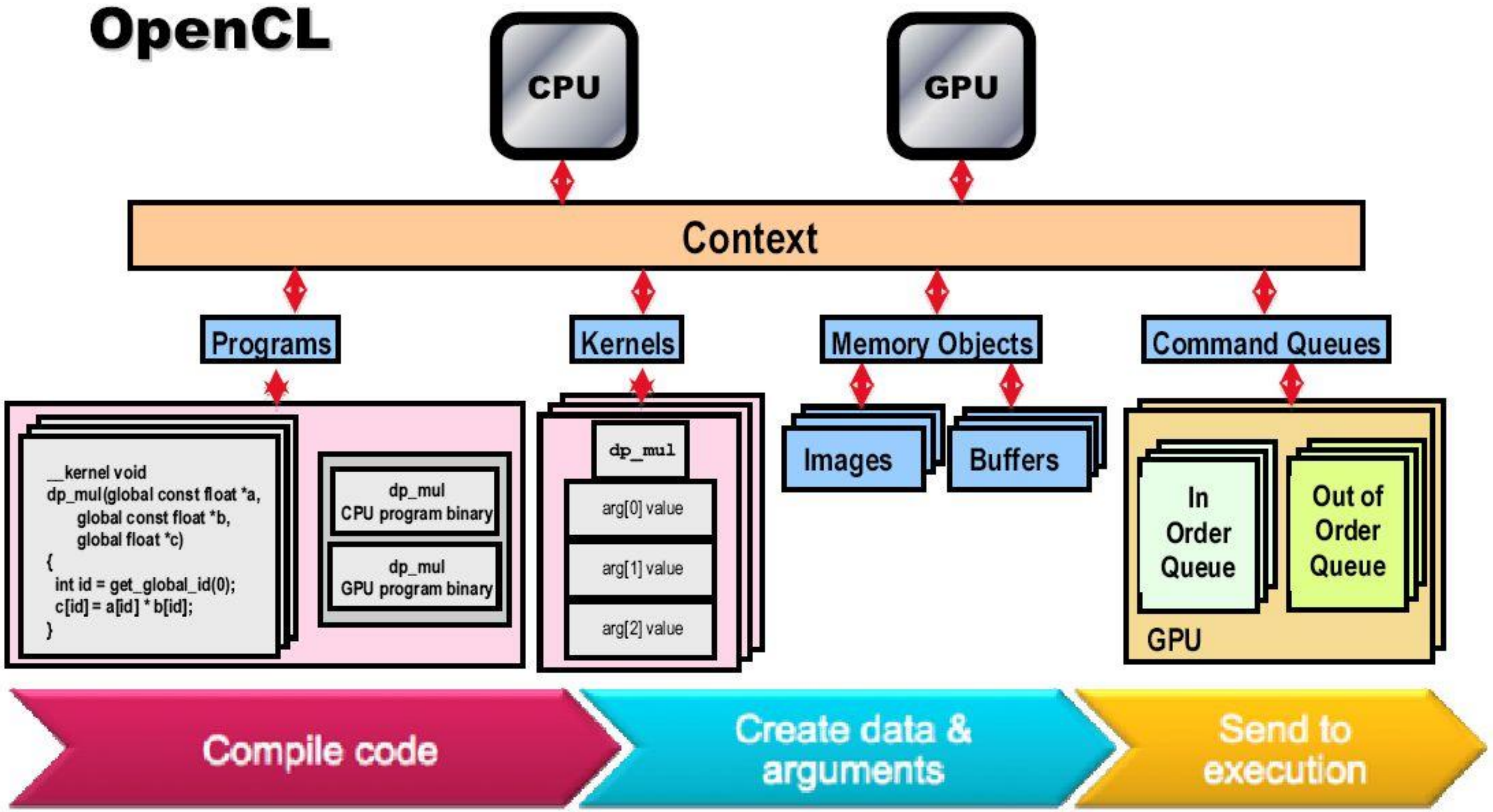
Global/constant memory
- not synchronized

Host memory
- on CPU



from: Khronos OpenCL Overview

Execution Model



from: Khronos OpenCL Overview

More Information

CUDA

- NVIDIA CUDA Programming Guide (version 2.3)
- NVIDIA CUDA Best Practices Guide (version 2.3)
- go to http://developer.download.nvidia.com

Fermi

- go to http://www.nvidia.com/object/fermi_architecture.html
- also informative: “NVIDIA's 'Fermi' GPU architecture revealed“ by Scott Wasson, available at http://techreport.com/articles.x/17670/1

OpenCL

- Khronos Group OpenCL Overview
- go to http://www.khronos.org/opencl