Introduction to GPU Programming

Volodymyr (Vlad) Kindratenko

Innovative Systems Laboratory @ NCSA

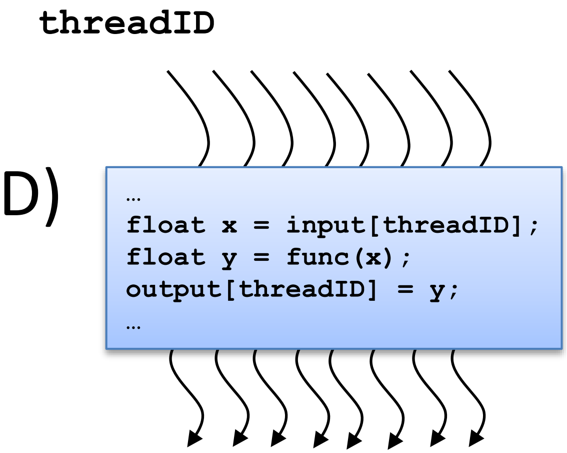
Institute for Advanced Computing Applications and Technologies (IACAT)

Part II

- GPU programing model
- Hands-on: Mandelbrot set fractal renderer
  - Reference implementation
  - GPU implementation

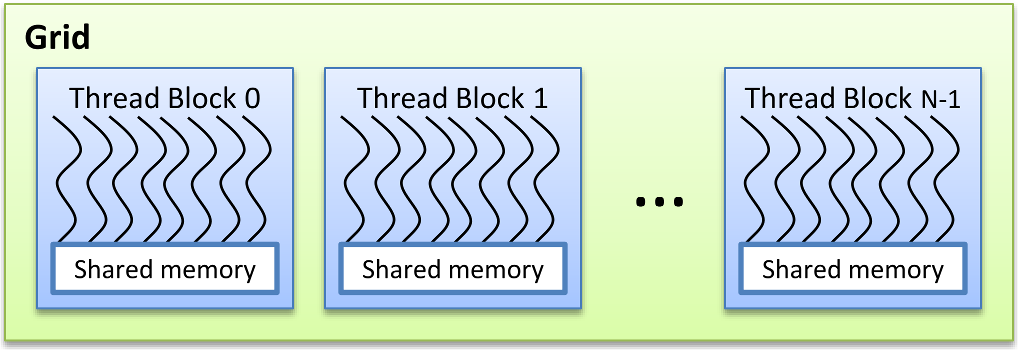
CUDA Programming Model

- A CUDA kernel is executed by an array of threads
  - All threads run the same code (SPMD)
  - Each thread has an ID that it uses to compute memory addresses and make control decisions



```
…
float x = input[threadID];
float y = func(x);
output[threadID] = y;
…
```

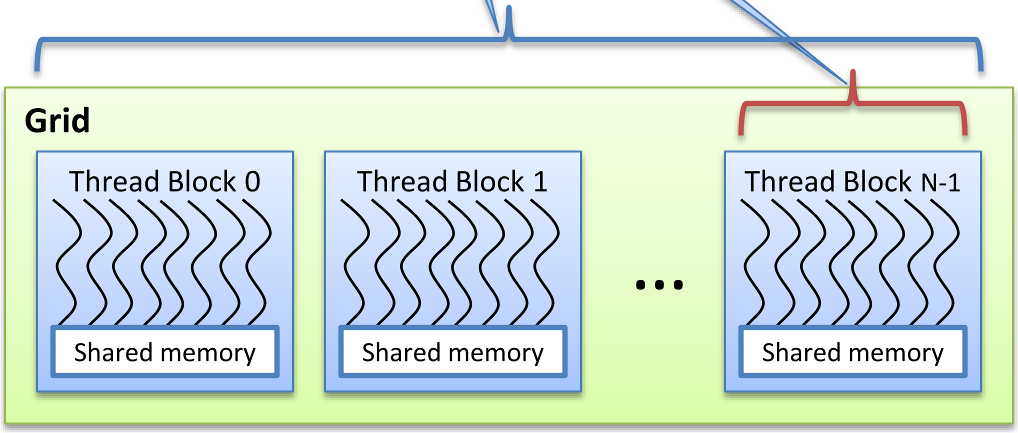
- Threads are arranged as a grid of thread blocks
  - Threads within a block have access to a segment of shared memory



Kernel Invocation Syntax

grid & thread block dimensionality

```
vecAdd<<<32, 512>>>(devPtrA, devPtrB, devPtrC);
```



```
int i = blockIdx.x * blockDim.x + threadIdx.x;
```

block ID within a grid

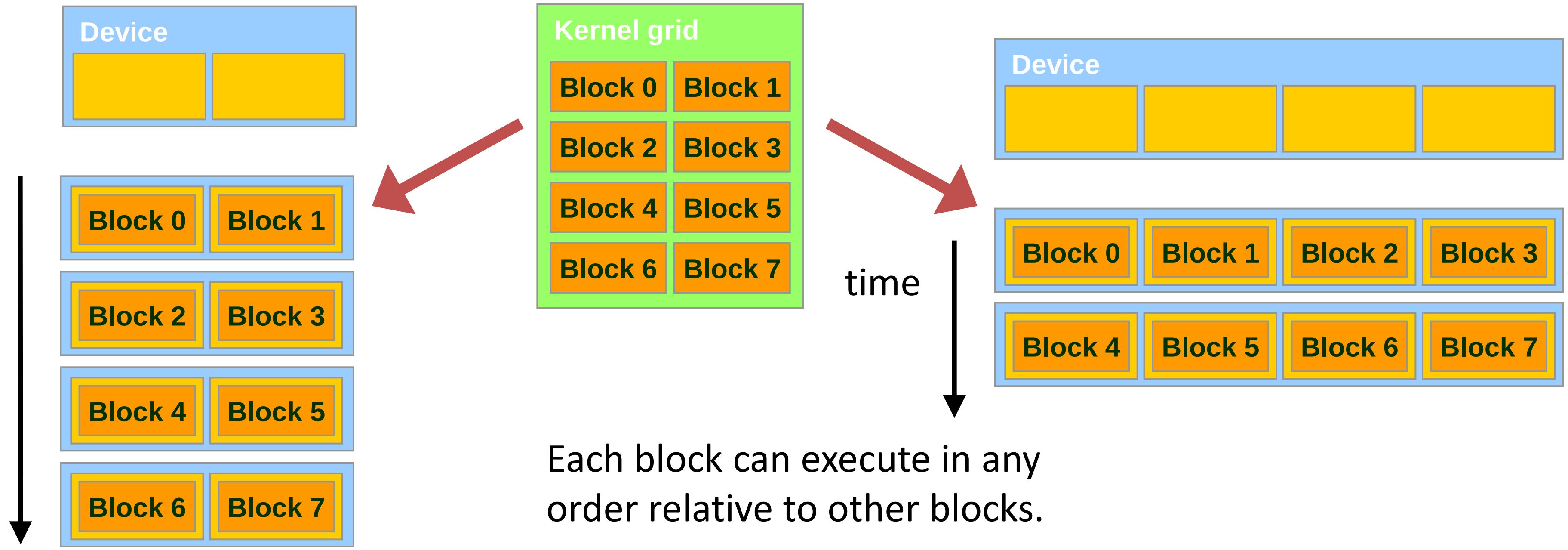
number of threads per block

thread ID within a thread block

Mapping Threads to the Hardware

- Blocks of threads are transparently assigned to SMs
  - A block of threads executes on one SM & does not migrate
  - Several blocks can reside concurrently on one SM

- Blocks must be independent
  - Any possible interleaving of blocks should be valid
  - Blocks may coordinate but not synchronize
  - Thread blocks can run in any order

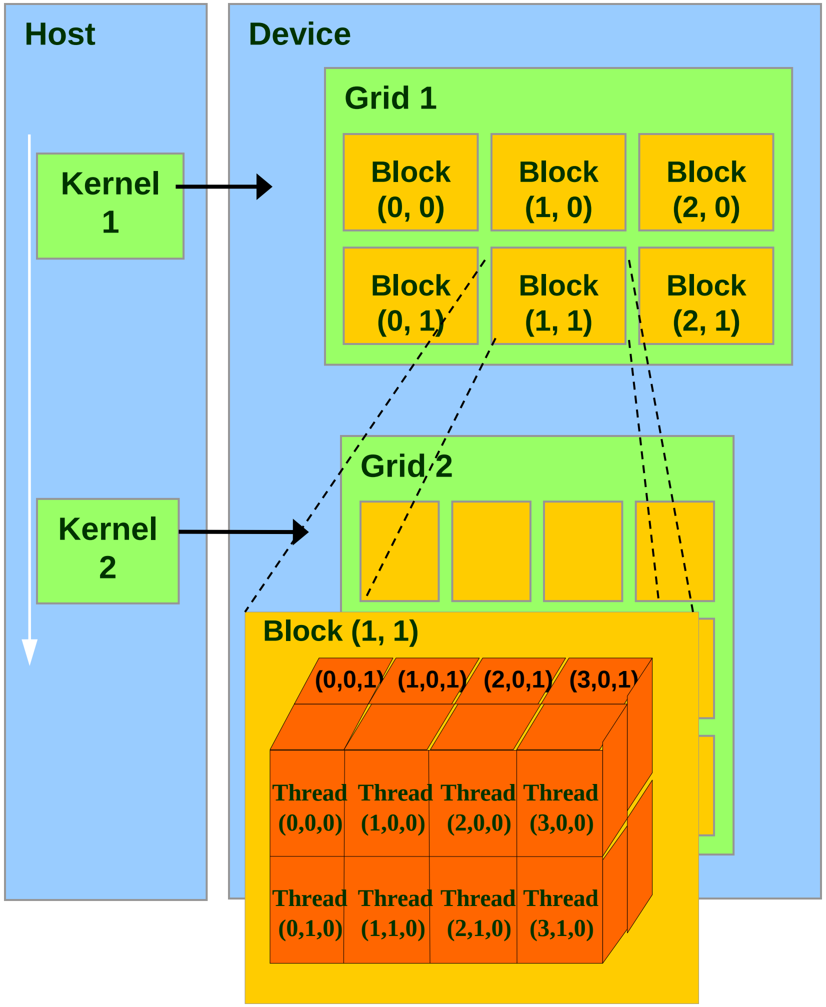


Each block can execute in any order relative to other blocks.

Slide is courtesy of NVIDIA

CUDA Programming Model

- A kernel is executed as a grid of thread blocks
  - Grid of blocks can be 1 or 2-dimentional
  - Thread blocks can be 1, 2, or 3-dimensional
- Different kernels can have different grid/block configuration
- Threads from the same block have access to a shared memory and their execution can be synchronized

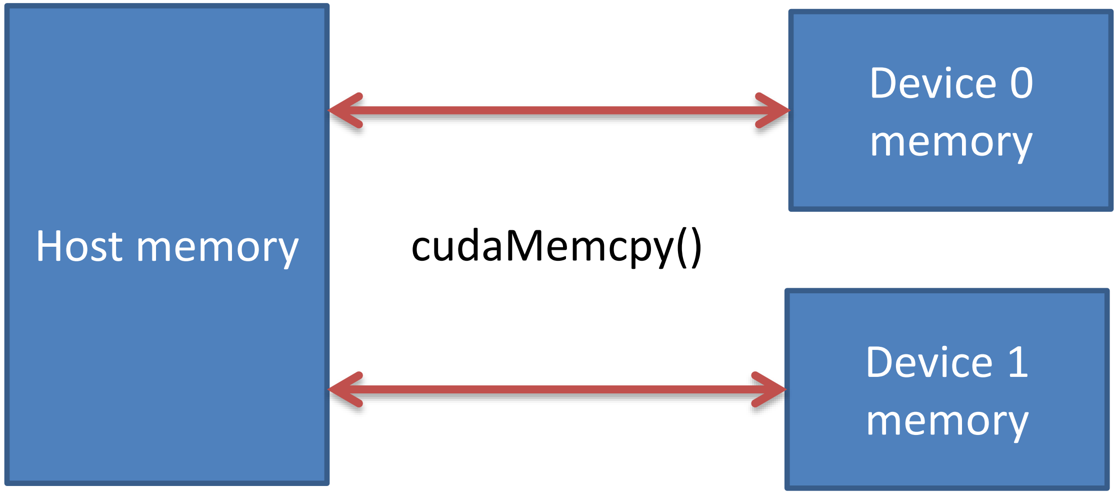


V. Kindratenko, **Introduction to GPU Programming (part II)**, December 2010, The American University in Cairo, Egypt

Slide is courtesy of NVIDIA

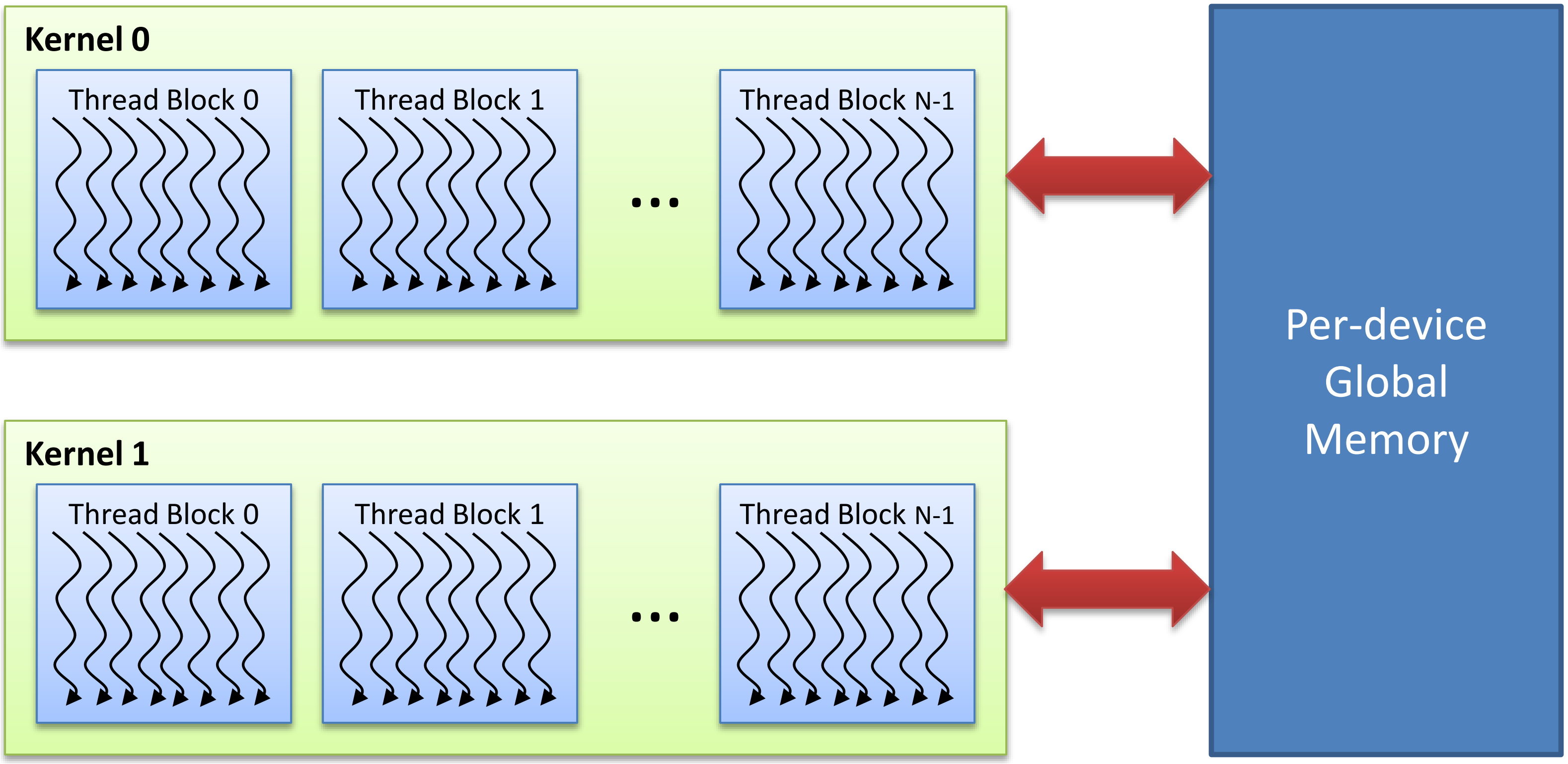
GPU Memory Hierarchy

- Global (device) memory
  - Accessible by all threads as well as host (CPU)
  - Data lifetime is from allocation to deallocation



GPU Memory Hierarchy

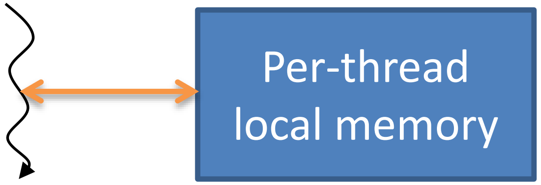
- Global (device) memory

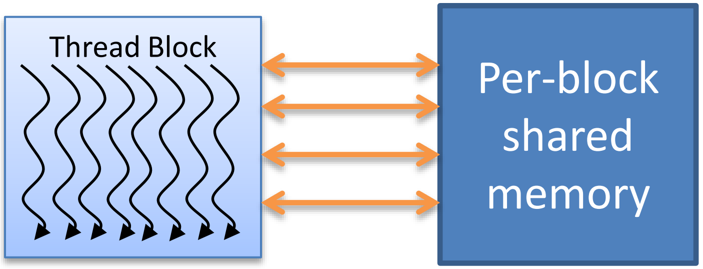


GPU Memory Hierarchy

- Local storage
  - Each thread has own local storage
  - Mostly registers (managed by the compiler)
  - Data lifetime = thread lifetime

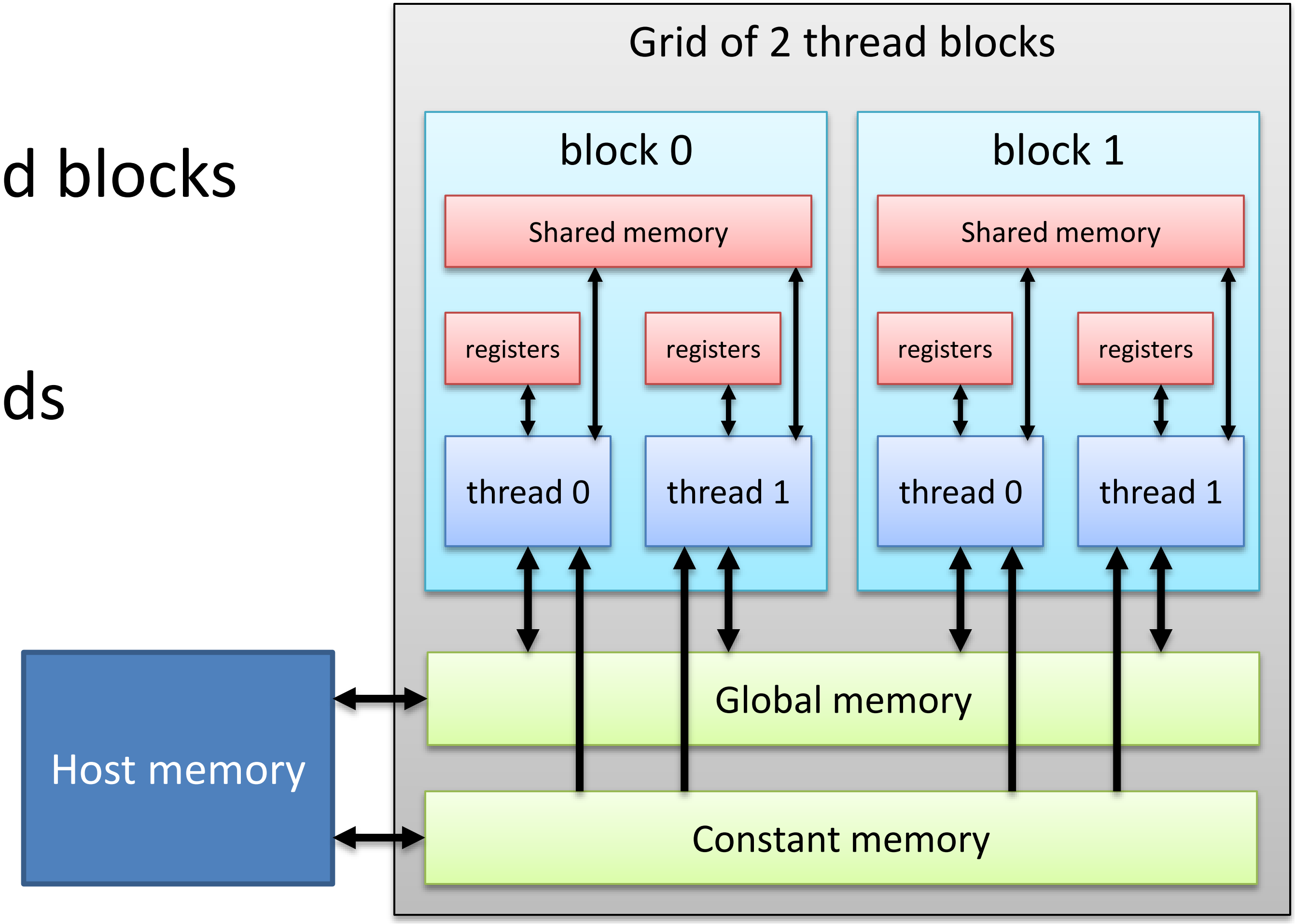
- Shared memory
  - Each thread block has own shared memory
    - Accessible only by threads within that block
  - Data lifetime = block lifetime



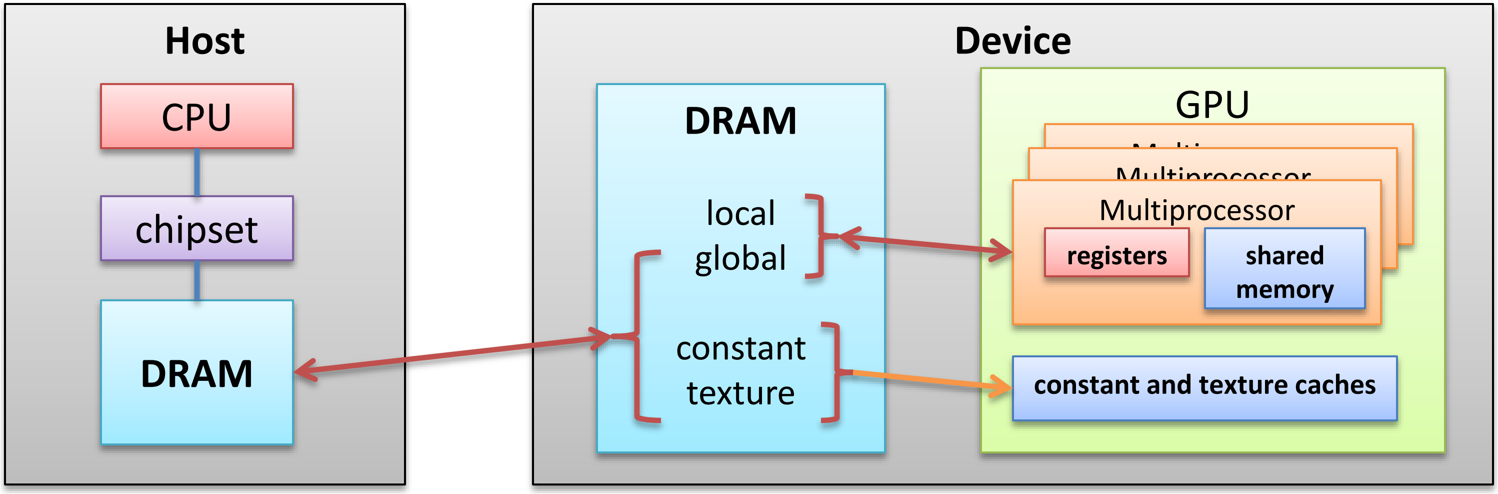


GPU Memory Hierarchy

- 1D grid
  - 2 thread blocks
- 1D block
  - 2 threads



GPU Memory Hierarchy



Memory	Location	Cached	Access	Scope	Lifetime
Register	On-chip	N/A	R/W	One thread	Thread
Local	Off-chip	No	R/W	One thread	Thread
Shared	On-chip	N/A	R/W	All threads in a block	Block
Global	Off-chip	No	R/W	All threads + host	Application
Constant	Off-chip	Yes	R	All threads + host	Application
Texture	Off-chip	Yes	R	All threads + host	Application

Porting Mandelbrot set fractal renderer to CUDA

- Source is in ~/tutorial/src2
  - fractal.c – reference C implementation
  - Makefile – make file
  - fractal.cu.reference – CUDA implementation for reference

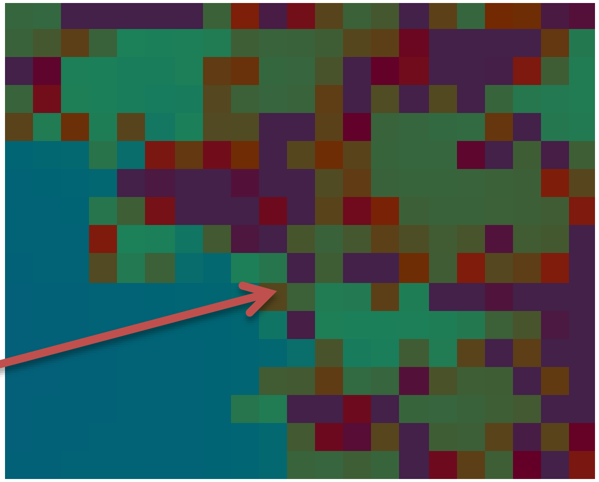
Getting started

- **cd tutorial/src2**
- **make cpu**
- **./fractal_cpu**
- **make convert**

- *copy fractal.bmp to your desktop*
- *display fractal.bmp on your desktop*

Reference C Implementation

```
void makefractal_cpu(unsigned char *image, int width, int height, double xupper,
 double xlower, double yupper, double ylower)
{
    int x, y;

    double xinc   = (xupper - xlower) / width;
    double yinc   = (yupper - ylower) / height;

    for (y = 0; y < height; y++)
    {
        for (x = 0; x < width; x++)
        {
            image[y*width+x] = iter((xlower + x*xinc), (ylower + y*yinc));
        }
    }
}
```



Reference C Implementation

```
inline unsigned char iter(double a, double b)
{
   unsigned char i = 0;
   double c_x = 0, c_y = 0;
   double c_x_tmp, c_y_tmp;
   double D = 4.0;

   while ((c_x*c_x+c_y*c_y < D) && (i++ < 255))
   {
      c_x_tmp = c_x * c_x - c_y * c_y;
      c_y_tmp = 2* c_y * c_x;
      c_x = a + c_x_tmp;
      c_y = b + c_y_tmp;
   }

   return i;
}
```

The Mandelbrot set is generated by iterating complex function $\mathbf{z^2 + c}$, where $\mathbf{c}$ is a constant:

$$\mathbf{z_1 = (z_0)^2 + c}$$
$$\mathbf{z_2 = (z_1)^2 + c}$$
$$\mathbf{z_3 = (z_2)^2 + c}$$

and so forth. Sequence $\mathbf{z_0, z_1, z_2,...}$ is called the *orbit* of $\mathbf{z_0}$ under iteration of $\mathbf{z^2 + c}$. We stop iteration when the orbit starts to diverge, or when a maximum number of iterations is done.

CUDA Kernel Implementation

```
__global__ void makefractal_gpu(unsigned char *image, ~~int width, int height, double
xupper, double xlower, double yupper, double ylower~~)
{
   int x = blockIdx.x;
   int y = blockIdx.y;

   int width = gridDim.x;
   int height = gridDim.y;

   double xupper=-0.74624, xlower=-0.74758, yupper=0.10779, ylower=0.10671;

   double xinc = (xupper - xlower) / width;
   double yinc = (yupper - ylower) / height;

   image[y*width+x] = iter((xlower + x*xinc), (ylower + y*yinc));
}
```

CUDA Kernel Implementation

```
~~inline~~ __device__ unsigned char iter(double a, double b)
{
   unsigned char i = 0;
   double c_x = 0, c_y = 0;
   double c_x_tmp, c_y_tmp;
   double D = 4.0;

   while ((c_x*c_x+c_y*c_y < D) && (i++ < 255))
   {
      c_x_tmp = c_x * c_x - c_y * c_y;
      c_y_tmp = 2* c_y * c_x;
      c_x = a + c_x_tmp;
      c_y = b + c_y_tmp;
   }

   return i;
}
```

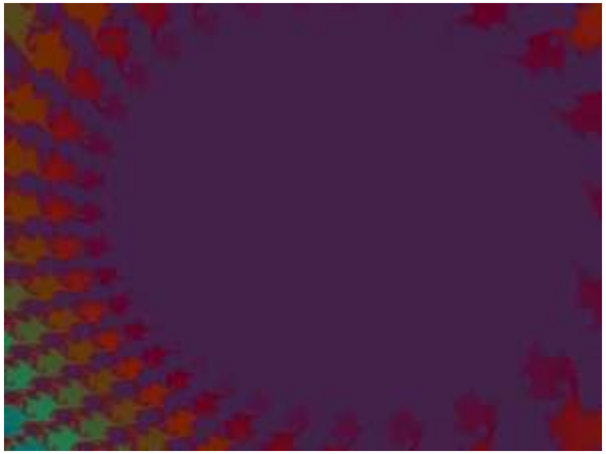
Host Code

```
int width = 1024;
int height = 768;
unsigned char *image = NULL;
unsigned char *devImage;

image = (unsigned char*)malloc(width*height*sizeof(unsigned char));
cudaMalloc((void**)&devImage, width*height*sizeof(unsigned char));

dim3 dimGrid(width, height);
dim3 dimBlock(1);

makefractal_gpu<<<dimGrid, dimBlock>>>(devImage);

cudaMemcpy(image, devImage, width*height*sizeof(unsigned char), cudaMemcpyDeviceToHost);

free(image);
cudaFree(devImage);
```

Few Examples

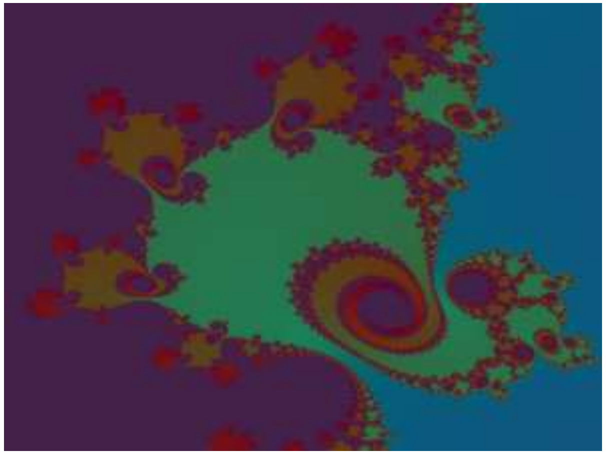
- xupper=-0.74624
- xlower=-0.74758
- yupper=0.10779
- ylower=0.10671

- CPU time: 2.27 sec
- GPU time: 0.29 sec



- xupper=-0.754534912109
- xlower=-.757077407837
- yupper=0.060144042969
- ylower=0.057710774740

- CPU time: 1.5 sec
- GPU time: 0.25 sec



Lab/Homework Exercises

- Exercise 1: Modify fractal code to improve efficiency
  - hint: launch multiple threads per block

Documentation

- NVIDIA's documentation
  - http://developer.nvidia.com/object/gpucomputing.html
  - Programming Guide
  - Best Practices Gide
  - Reference Manual
- CUDA C SDK Code Samples
  - http://developer.nvidia.com/object/cuda_3_2_downloads.html
- Books
  - David Kirk, Wen-mei W. Hwu, Programming Massively Parallel Processors: A Hands-on Approach, Morgan Kaufmann, 2010
  - Jason Sanders, Edward Kandrot, CUDA by Example: An Introduction to General-Purpose GPU Programming, Addison-Wesley, 2010