GPU Computing the Easi(er) Way:

Introduction to Running MATLAB and Mathematica on a GPU

Peter Ruprecht
peter.ruprecht@colorado.edu

www.rc.colorado.edu

Outline

- What is a GPU?
- Computing on a GPU vs a CPU
- MATLAB
  - Creating arrays on the GPU
  - Using GPU-enabled functions
  - Improving element-wise parts of the program
  - Vectorization and beyond
- Mathematica
  - GPU-enabled functions
  - Moving data to/from GPU

Slides and code samples available by next week at
http://researchcomputing.github.io

What is a GPU?

- Graphics Processing Unit (“video card”)
- Specialized multi-core hardware designed for rendering images
- Designed for a highly parallelized applications (often embarrassingly parallel)
- Best at handling single-precision data
- Great at vector calculations

- Some GPUs are now modified to make them better at non-graphics computing
  - Double-precision
  - Error correction

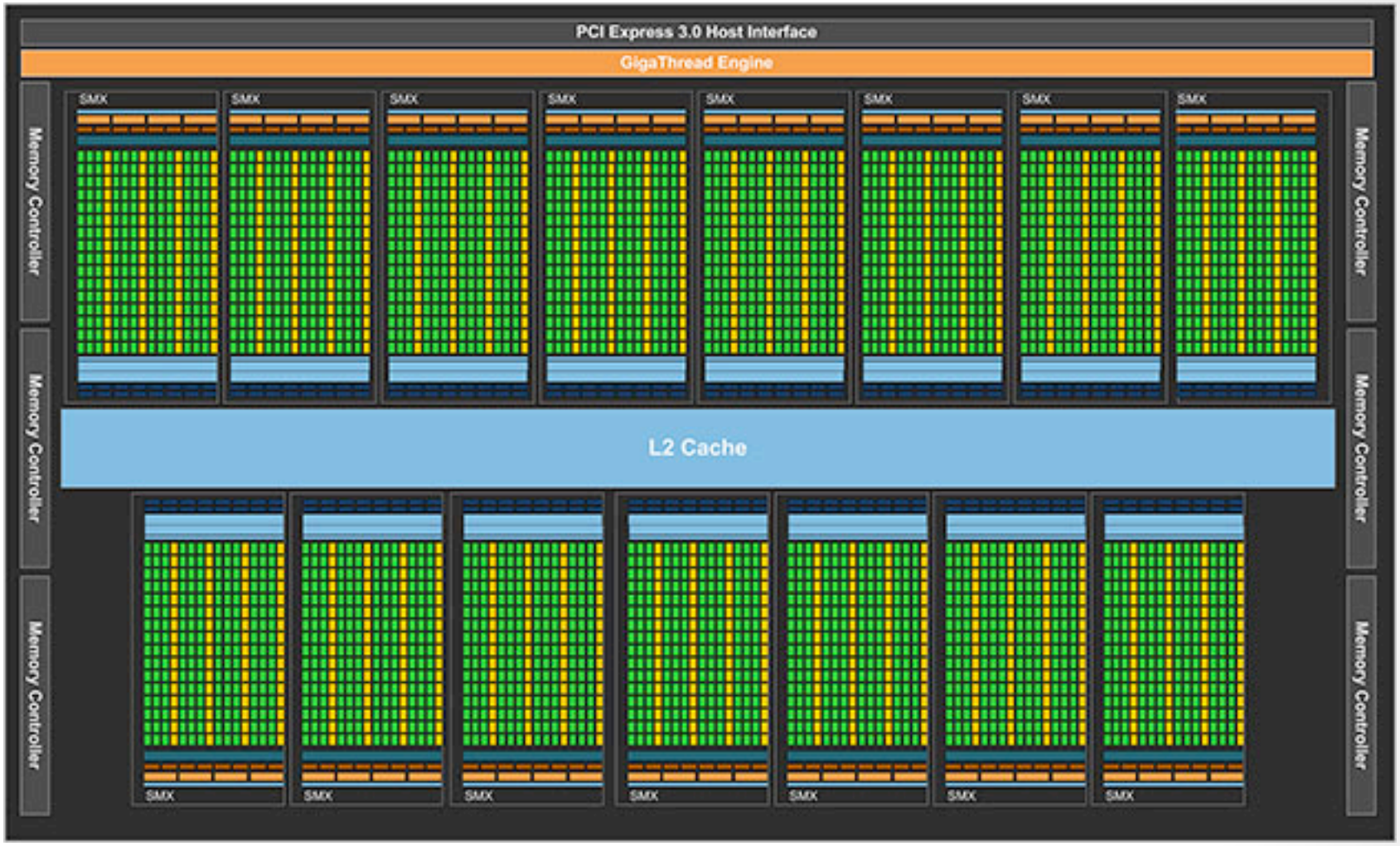

Image: NVIDIA

Computer Bus Layout

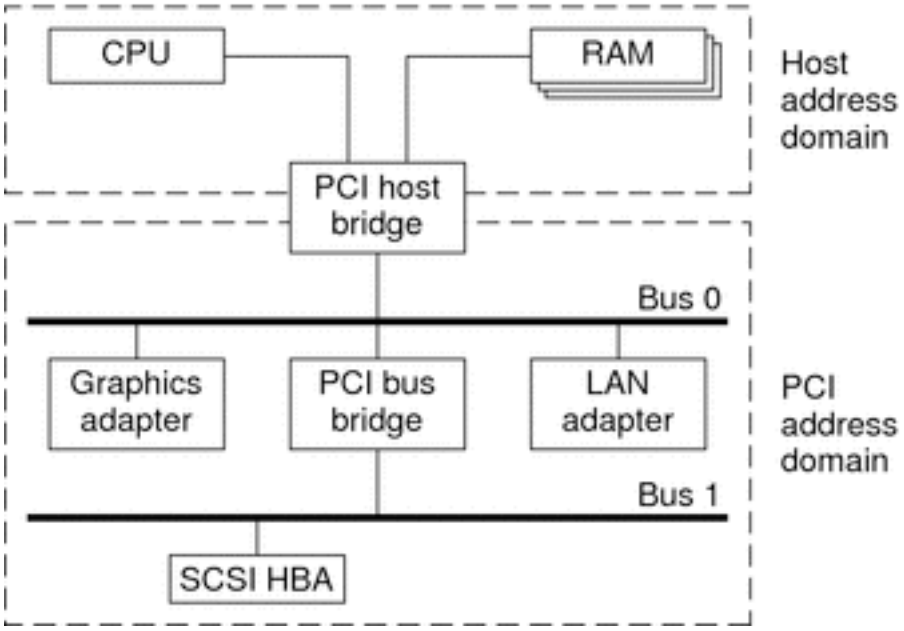

Image: Oracle

GPGPU

- General Purpose computing on Graphics Processing Unit
  - A software or workflow concept; not a piece of hardware
- CPU has several cores that are designed to handle a wide range of tasks
- GPU has many cores that are highly specialized
- Getting CPU and GPU to work together efficiently is tricky, especially moving data back and forth
- Programming models:
  - OpenCL – more general, higher level
  - CUDA – NVIDIA only

GPU Systems in CU RC

- “gpu” QOS (aka queue) consists of 2 nodes:
- 2x 6-core CPUs
- 128 GB RAM
- 1x NVIDIA Tesla M2070 GPU
  - 448 cores
  - 1.15 GHz
  - 6 GB onboard memory

MATLAB

- High-level language for numerical computation
- Excellent support for matrix and vector operations
- Includes Parallel Computing Toolkit, which supports GPGPU
- Many built-in functions and operators include near-automatic GPU support
- Requires CUDA, hence NVIDIA GPUs
- If you have an NVIDIA video card in your laptop or desktop you can try MATLAB right on it *

MATLAB – first steps

- Start an interactive job

```
salloc --qos=gpu
```

- Load MATLAB module

```
module load matlab/matlab-2014a
```

- Start MATLAB

```
matlab
```

- Can we see the GPU?

```
gpuDeviceCount
d=gpuDevice
```

MATLAB – gpuArray

- Follow example from MATLAB docs:
- Create array (matrix) and transfer it to GPU

```
X=rand(1000);
G=gpuArray(X);
```

- Create array directly on GPU

```
G=ones(100,100,50,'gpuArray');
```

- Return the data to the CPU

```
C=gather(G);
```

MATLAB – built-in functions

- Over 200 MATLAB functions accept gpuArrays as input
- When a GPU-enabled function operates on a gpuArray, the computation automatically happens on the GPU
- May be slight differences in a function's behavior or restrictions when using gpuArray
- Operators such as +, *, ^, \

MATLAB – timing example

```
tic;
Ca = rand(10000,'single');
Cfft = fft(Ca);
Cb = (real(Cfft) + Ca) * 6; toc
Elapsed time is 2.393863 seconds.

tic;
Ga = rand(10000,'single','gpuArray');
Gfft = fft(Ga);
Gb = (real(Gfft) + Ga) * 6;
G = gather(Gb); toc
Elapsed time is 0.441605 seconds.
```

MATLAB – arrayfun

arrayfun – apply function to each element of an array

- Useful for functions that can't be vectorized and thus must be handled element-wise
- Can use on CPU or GPU
- Creates custom CUDA code and runs it on the GPU for extra performance
- Not all MATLAB functions can be passed into arrayfun

MATLAB – arrayfun example

```
function Y = randommult(X)
    R = rand();
    Y = R.*X;
end

>> tic; C = 2*ones(1000,1000);
D=arrayfun(@randommult, C); toc
Elapsed time is 5.965878 seconds.

>> tic; G=2*ones(1000,1000,'gpuArray');
H=arrayfun(@randommult, G); gather(H); toc
Elapsed time is 2.828656 seconds.
```

MATLAB – vectorization

- Reduce loop-based element-wise algorithms
- Rather, use vector and matrix operations whenever possible
- Important for both CPU and GPU performance
- Details left as an exercise for the interested user

Other GPU options:

- Turn PTX file into a CUDA Kernel (requires CUDA Toolkit from NVIDIA)
- Compile MEX file that contains CUDA code, (not C)

MATLAB – vectorization example

- Calculate volume of a square prism, given two vectors containing lengths and heights (`L` and `H`)
- Scalar or element-wise:

```
for n = 1:10000
    V(n) = (L(n)^2)*H(n);
end
```

- Vectorize using array operators and no loop:

```
V=(L.^2).*H;
```

Mathematica

- Integrated system for symbolic and numerical computation
- Includes CUDALink and OpenCLLink, which support GPGPU
  - We are going to focus on CUDALink
- Many built-in functions and operators include near-automatic GPU support
- Requires CUDA, hence NVIDIA GPUs
- If you have an NVIDIA video card in your laptop or desktop you can use it with Mathematica

Mathematica – first steps

- Start an interactive job

```
salloc --qos=gpu
```

- Load Mathematica module

```
module load mathematica/mathematica-9.0
```

- Start Mathematica CLI

```
math
```

- Load the CUDALink application and test

```
In[1]:= Needs["CUDALink`"]
In[2]:=CUDAQ[]
In[3]:=CUDAInformation[]
```

Mathematica – built-in functions

- Names begin with “CUDA”

```
CUDADot, CUDAFourier, . . .
```

- Create matrix on CPU and multiply it by itself

```
C=RandomReal[1,{1000,1000}];
AbsoluteTiming[C.C;]
```

- Use CUDADot instead

```
AbsoluteTiming[CudaDot[C,C];]
```

Mathematica – use GPU memory

- Move data to GPU with CUDAMemoryLoad

```
G=CUDAMemoryLoad[C]
```

- Use data on GPU for multiply

```
AbsoluteTiming[res=CUDADot[G,G]]
```

- Bring data back to main memory (CPU)

```
CUDAMemoryGet[res]
```

`CUDAFunctionLoad` lets you write your own functions in C and run them on the GPU

Thank you!

Slides available at:

http://researchcomputing.github.io