Supercomputing in Plain English

GPGPU: Number Crunching in Your Graphics Card

Henry Neeman, Director
OU Supercomputing Center for Education & Research
University of Oklahoma Information Technology

Tuesday April 26 2011

This is an experiment!

It’s the nature of these kinds of videoconferences that **FAILURES ARE GUARANTEED TO HAPPEN! NO PROMISES!**

So, please bear with us. Hopefully everything will work out well enough.

If you lose your connection, you can retry the same kind of connection, or try connecting another way.

Remember, if all else fails, you always have the toll free phone bridge to fall back on.

Access Grid

If you aren’t sure whether you have AG, you probably don’t.

Tue Apr 26	Monte Carlo
Tue May 3	Helium

Many thanks to Patrick Calhoun of OU for setting these up for us.

H.323 (Polycom etc)

From an H.323 device (e.g., Polycom, Tandberg, Lifesize, etc):

- If you **ARE** already registered with the OneNet gatekeeper:
  Dial
  `2500409`
- If you **AREN'T** registered with the OneNet gatekeeper (probably the case):
  1. Dial:
     `164.58.250.47`
  2. Bring up the virtual keypad.
     On some H.323 devices, you can bring up the virtual keypad by typing:
     `#`
  3. When asked for the conference ID, enter:
     `0409`
  4. On some H.323 devices, you indicate the end of conference ID with:
     `#`

Many thanks to Roger Holder and OneNet for providing this.

H.323 from Internet Explorer

From a Windows PC running Internet Explorer:

1. You **MUST** have the ability to install software on the PC (or have someone install it for you).
2. Download and install the latest Java Runtime Environment (JRE) from here:
   `http://www.oracle.com/technetwork/java/javase/downloads/`
   (Click on the Java Download icon, because that install package includes both the JRE and other components.)
3. Download and install this video decoder:
   `http://164.58.250.47/codian_video_decoder.msi`
4. Start Internet Explorer.
5. Copy-and-paste this URL into your IE window:
   `http://164.58.250.47/`
6. When that webpage loads, in the upper left, click on "Streaming."
7. In the textbox labeled Sign-in Name, type your name.
8. In the textbox labeled Conference ID, type this:
   `0409`
9. Click on "Stream this conference."
10. When that webpage loads, you may see, at the very top, a bar offering you options.
    If so, click on it and choose "Install this add-on."

H.323 from XMeeting (MacOS)

From a Mac running MacOS X:

1. Download XMeeting from
   `http://xmeeting.sourceforge.net/`
2. Install XMeeting as follows:
   a. Open the .dmg file.
   b. Drag XMeeting into the Applications folder.
3. Open XMeeting from Applications.
4. Skip the setup wizard.
5. In the call box, type
   `164.58.250.47`
6. Click the **Call** button.
7. From the Remote Control window, when prompted to join the conference, enter :
   `0409#`

EVO

There's a quick tutorial on the OSCER education webpage.

QuickTime Broadcaster

If you cannot connect via the Access Grid, H.323 or EVO, then you can connect via QuickTime:

`rtsp://129.15.254.141/test_hpc09.sdp`

We recommend using QuickTime Player for this, because we've tested it successfully.

We recommend upgrading to the latest version at:

`http://www.apple.com/quicktime/`

When you run QuickTime Player, traverse the menus

File -> Open URL

Then paste in the rstp URL into the textbox, and click OK.

Many thanks to Kevin Blake of OU for setting up QuickTime Broadcaster for us.

WebEx

We have only a limited number of WebEx connections, so please avoid WebEx unless you have **NO OTHER WAY TO CONNECT**.

Instructions are available on the OSCER education webpage.

Thanks to Tim Miller of Wake Forest U.

Phone Bridge

If all else fails, you can call into our toll free phone bridge:

US: 1-800-832-0736, *6232874#

International: 303-330-0440, *6232874#

Please mute yourself and use the phone to listen.

Don’t worry, we’ll call out slide numbers as we go.

Please use the phone bridge **<u>ONLY</u>** if you cannot connect any other way: the phone bridge is charged per connection per minute, so our preference is to minimize the number of connections.

Many thanks to Amy Apon and U Arkansas for providing the previous toll free phone bridge.

Please Mute Yourself

No matter how you connect, please mute yourself, so that we cannot hear you.

At OU, we will turn off the sound on all conferencing technologies.

That way, we won't have problems with echo cancellation.

Of course, that means we cannot hear questions.

So for questions, you'll need to send some kind of text.

Questions via Text: E-mail

Ask questions via e-mail to **`sipe2011@yahoo.com`**.

All questions will be read out loud and then answered out loud.

Thanks for helping!

- OSCER operations staff: Brandon George, Dave Akin, Brett Zimmerman, Josh Alexander
- Horst Severini, OSCER Associate Director for Remote & Heterogeneous Computing
- OU Research Campus staff (Patrick Calhoun, Mark McAvoy)
- Kevin Blake, OU IT (videographer)
- John Chapman, Jeff Pummill and Amy Apon, U Arkansas
- James Deaton and Roger Holder, OneNet
- Tim Miller, Wake Forest U
- Jamie Hegarty Schwettmann, i11 Industries

This is an experiment!

It’s the nature of these kinds of videoconferences that **FAILURES ARE GUARANTEED TO HAPPEN! NO PROMISES!**

So, please bear with us. Hopefully everything will work out well enough.

If you lose your connection, you can retry the same kind of connection, or try connecting another way.

Remember, if all else fails, you always have the toll free phone bridge to fall back on.

Supercomputing Exercises

Want to do the "Supercomputing in Plain English" exercises?

- The first exercise is already posted at:

  http://www.oscer.ou.edu/education.php

- If you don't yet have a supercomputer account, you can get a temporary account, just for the "Supercomputing in Plain English" exercises, by sending e-mail to:

  hneeman@ou.edu

Please note that this account is for doing the **exercises only**, and will be shut down at the end of the series.

- There isn't a GPGPU exercise, because we don't really learn how to do the programming.

Summer Workshops 2011

- Introduction to Computational Thinking
  - Modeling and Simulation Across the Curriculum
    - June 19 - 25, Wayne State College, Wayne NE
    - July 10 - 16, Texas State Technical College, Waco
  - Preparing In-service and Pre-service Educators for Computational Thinking
    - July 24 - 30, Oregon State University, Corvallis
  - Computational Thinking from a Parallel Perspective
    - July 31 - Aug 6, Louisiana State University, Baton Rouge

Summer Workshops 2011 (cont'd)

- Computational Biology for Biology Educators
  - June 12 - 18, Lafayette College, Easton PA
  - June 26 - July 2, Calvin College, Grand Rapids MI
- Computational Chemistry for Chemistry Educators
  - July 24 - 30, Washington and Lee University, Lexington VA

  JOINTLY PRESENTED VIA VIDEOCONFERENCING WITH
  - July 24 - 30, Oklahoma State University, Stillwater
- Data-driven Computational Science: Modeling and Visualization
  - June 19 - 25, Richard Stockton College of New Jersey, Pomona

Summer Workshops 2011 (cont'd)

- Introduction to Parallel Programming & Cluster Computing
  - June 26 - July 1, Idaho State University, Pocatello

  JOINTLY PRESENTED VIA VIDEOCONFERENCING WITH
  - June 26 - July 1, University of Washington, Seattle
- Intermediate Parallel Programming & Cluster Computing
  - July 31 - Aug 6, University of Oklahoma, Norman

  JOINTLY PRESENTED VIA VIDEOCONFERENCING WITH
  - July 31 - Aug 6, Polytechnic University of Puerto Rico, Hato Rey

OK Supercomputing Symposium 2011

2003 Keynote:
Peter Freeman
NSF
Computer & Information Science & Engineering Assistant Director

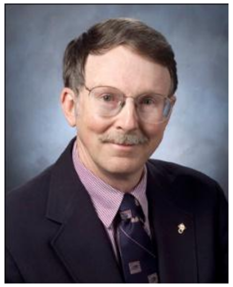
2004 Keynote:
Sangtae Kim
NSF Shared Cyberinfrastructure Division Director

2005 Keynote:
Walt Brooks
NASA Advanced Supercomputing Division Director

2006 Keynote:
Dan Atkins
Head of NSF's Office of Cyberinfrastructure

2007 Keynote:
Jay Boisseau
Director
Texas Advanced Computing Center
U. Texas Austin

2008 Keynote:
José Munoz
Deputy Office Director/ Senior Scientific Advisor
NSF Office of Cyberinfrastructure


2009 Keynote:
Douglass Post
Chief Scientist
US Dept of Defense HPC Modernization Program

2010 Keynote:
Horst Simon
Deputy Director
Lawrence Berkeley National Laboratory

?

2011 Keynote to be announced

FREE! Wed Oct 12 2011 @ OU

http://symposium2011.oscer.ou.edu/

Parallel Programming Workshop
FREE! Tue Oct 11 2011 @ OU
FREE! Symposium Wed Oct 12 2011 @ OU

REGISTRATION IS NOW OPEN!

SC11 Education Program

- At the SC11 supercomputing conference, we’ll hold our annual Education Program, Sat Nov 12 – Tue Nov 15.
- You can apply to attend, either fully funded by SC11 or self-funded.
- Henry is the SC11 Education Chair.
- We’ll alert everyone once the registration website opens.

Outline

- What is GPGPU?
- GPU Programming
- Digging Deeper: CUDA on NVIDIA
- CUDA Thread Hierarchy and Memory Hierarchy
- CUDA Example: Matrix-Matrix Multiply

What is GPGPU?

Accelerators

No, not this ....

http://gizmodo.com/5032891/nissans-eco-gas-pedal-fights-back-to-help-you-save-gas

Accelerators

- In HPC, an accelerator is hardware component whose role is to speed up some aspect of the computing workload.
- In the olden days (1980s), supercomputers sometimes had ***array processors***, which did vector operations on arrays, and PCs sometimes had ***floating point accelerators***: little chips that did the floating point calculations in hardware rather than software.
- More recently, ***Field Programmable Gate Arrays*** (FPGAs) allow reprogramming deep into the hardware.

Why Accelerators are Good

Accelerators are good because:

- they make your code run faster.

Why Accelerators are Bad

Accelerators are bad because:

- they’re expensive;
- they’re hard to program;
- your code on them may not be portable to other accelerators, so the labor you invest in programming them has a very short half-life.

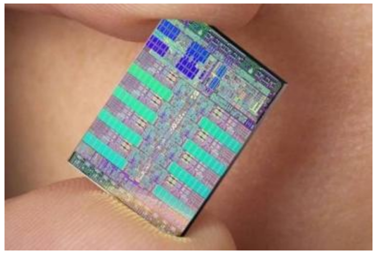
The King of the Accelerators

The undisputed champion of accelerators is:

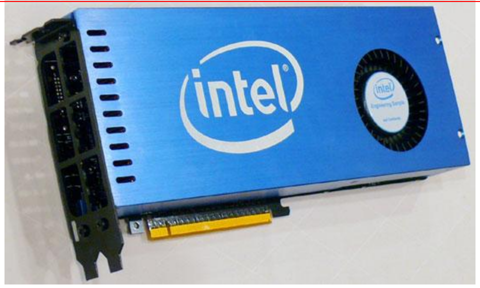
the **graphics processing unit.**

http://www.amd.com/us-en/assets/content_type/DigitalMedia/46928a_01_ATI-FirePro_V8700_angled_low_res.gif

http://images.nvidia.com/products/quadro_fx_5800/Quadro_FX5800_low_3qtr.png



http://www.overclockers.ua/news/cpu/106612-Knights-Ferry.jpg



http://www.gamecyte.com/wp-content/uploads/2009/01/ibm-sony-toshiba-cell.jpg

Why GPU?

- ***Graphics Processing Units*** (GPUs) were originally designed to accelerate graphics tasks like image rendering.
- They became very very popular with videogamers, because they've produced better and better images, and lightning fast.
- And, prices have been extremely good, ranging from three figures at the low end to four figures at the high end.

GPUs are Popular

- Chips are expensive to design (hundreds of millions of $$$), expensive to build the factory for (billions of $$$), but cheap to produce.
- For example, in 2006 – 2007, GPUs sold at a rate of about 80 million cards per year, generating about $20 billion per year in revenue.
  http://www.xbitlabs.com/news/video/display/20080404234228_Shipments_of_Discrete_Graphics_Cards_on_the_Rise_but_Prices_Down_Jon_Peddie_Research.html
- This means that the GPU companies have been able to recoup the huge fixed costs.

GPU Do Arithmetic

- GPUs mostly do stuff like rendering images.
- This is done through mostly floating point arithmetic – the same stuff people use supercomputing for!

GPU Programming

Hard to Program?

- In the olden days – that is, until just the last few years – programming GPUs meant either:
  - using a graphics standard like OpenGL (which is mostly meant for rendering), or
  - getting fairly deep into the graphics rendering pipeline.
- To use a GPU to do general purpose number crunching, you had to make your number crunching pretend to be graphics.
- This was hard. So most people didn’t bother.

Easy to Program?

More recently, GPU manufacturers have worked hard to make GPUs easier to use for general purpose computing.

This is known as ***General Purpose Graphics Processing Units***.

How to Program a GPU

- Proprietary programming language or extensions
  - NVIDIA: CUDA (C/C++)
  - AMD/ATI: StreamSDK/Brook+ (C/C++)
- OpenCL (Open Computing Language): an industry standard for doing number crunching on GPUs.
- Portland Group Inc (PGI) Fortran and C compilers with accelerator directives; PGI CUDA Fortran (Fortran 90 equivalent of NVIDIA's CUDA C).
- OpenMP version 4.0 may include directives for accelerators.
- Others are popping up or in development now ….

NVIDIA CUDA

- NVIDIA proprietary
- Formerly known as “Compute Unified Device Architecture”
- Extensions to C to allow better control of GPU capabilities
- Modest extensions but major rewriting of the code
- Portland Group Inc (PGI) has released a Fortran implementation of CUDA available in their Fortran compiler.

CUDA Example Part 1

```
// example1.cpp : Defines the entry point for the console applicati
    on.
//

#include "stdafx.h"

#include <stdio.h>
#include <cuda.h>

// Kernel that executes on the CUDA device
__global__ void square_array(float *a, int N)
{
  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx<N) a[idx] = a[idx] * a[idx];
}
```

http://llpanorama.wordpress.com/2008/05/21/my-first-cuda-program/

CUDA Example Part 2

```
// main routine that executes on the host
int main(void)
{
  float *a_h, *a_d;  // Pointer to host & device arrays
  const int N = 10;  // Number of elements in arrays
  size_t size = N * sizeof(float);
  a_h = (float *)malloc(size);        // Allocate array on host
  cudaMalloc((void **) &a_d, size);   // Allocate array on device
  // Initialize host array and copy it to CUDA device
  for (int i=0; i<N; i++) a_h[i] = (float)i;
  cudaMemcpy(a_d, a_h, size, cudaMemcpyHostToDevice);
  // Do calculation on device:
  int block_size = 4;
  int n_blocks = N/block_size + (N%block_size == 0 ? 0:1);
  square_array <<< n_blocks, block_size >>> (a_d, N);
  // Retrieve result from device and store it in host array
  cudaMemcpy(a_h, a_d, sizeof(float)*N, cudaMemcpyDeviceToHost);
  // Print results
  for (int i=0; i<N; i++) printf("%d %f\n", i, a_h[i]);
  // Cleanup
  free(a_h); cudaFree(a_d);
}
```

AMD/ATI Brook+

- AMD/ATI proprietary
- Formerly known as “Close to Metal” (CTM)
- Extensions to C to allow better control of GPU capabilities
- No Fortran version available

Brook+ Example Part 1

```
float4 matmult_kernel (int y, int x, int k,
                       float4 M0[], float4 M1[])
{
    float4 total = 0;
    for (int c = 0; c < k / 4; c++)
    {
        total += M0[y][c] * M1[x][c];
    }
    return total;
}
```

http://developer.amd.com/gpu_assets/Stream_Computing_Overview.pdf

Brook+ Example Part 2

```
void matmult (float4 A[], float4 B’[], float4 C[])
{
    for (int i = 0; i < n; i++)
    {
        for (j = 0; j < m / 4; j+)
        {
            launch_thread{
                C[i][j] =
                    matmult_kernel(j, i, k, A, B’);}
        }
    }
    sync_threads{}
}
```

OpenCL

- Open Computing Language
- Open standard developed by the Khronos Group, which is a consortium of many companies (including NVIDIA, AMD and Intel, but also lots of others)
- Initial version of OpenCL standard released in Dec 2008.
- Many companies are creating their own implementations.
- Apple was first to market, with an OpenCL implementation included in Mac OS X v10.6 ("Snow Leopard") in 2009.

OpenCL Example Part 1

```
// create a compute context with GPU device
context =
  clCreateContextFromType(NULL, CL_DEVICE_TYPE_GPU, NULL, NULL, NULL);
// create a command queue
queue = clCreateCommandQueue(context, NULL, 0, NULL);
// allocate the buffer memory objects
memobjs[0] = clCreateBuffer(context,
                 CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
                 sizeof(float)*2*num_entries, srcA, NULL);
memobjs[1] = clCreateBuffer(context,
                 CL_MEM_READ_WRITE,
                 sizeof(float)*2*num_entries, NULL, NULL);
// create the compute program
program = clCreateProgramWithSource(context, 1, &fft1D_1024_kernel_src,
                                    NULL, NULL);
```

http://en.wikipedia.org/wiki/OpenCL

OpenCL Example Part 2

```
// build the compute program executable
clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
// create the compute kernel
kernel = clCreateKernel(program, "fft1D_1024", NULL);
// set the args values
clSetKernelArg(kernel, 0, sizeof(cl_mem), (void *)&memobjs[0]);
clSetKernelArg(kernel, 1, sizeof(cl_mem), (void *)&memobjs[1]);
clSetKernelArg(kernel, 2, sizeof(float)*(local_work_size[0]+1)*16, NULL);
clSetKernelArg(kernel, 3, sizeof(float)*(local_work_size[0]+1)*16, NULL);
// create N-D range object with work-item dimensions and execute kernel
global_work_size[0] = num_entries; local_work_size[0] = 64;
clEnqueueNDRangeKernel(queue, kernel, 1, NULL,
                       global_work_size, local_work_size, 0, NULL, NULL);
```

OpenCL Example Part 3

```
// This kernel computes FFT of length 1024. The 1024 length FFT is
// decomposed into calls to a radix 16 function, another radix 16
// function and then a radix 4 function
__kernel void fft1D_1024 (__global float2 *in, __global float2 *out,
                          __local float *sMemx, __local float *sMemy) {
    int tid = get_local_id(0);
    int blockIdx = get_group_id(0) * 1024 + tid;
    float2 data[16];

// starting index of data to/from global memory
    in = in + blockIdx;
    out = out + blockIdx;
    globalLoads(data, in, 64); // coalesced global reads
    fftRadix16Pass(data); // in-place radix-16 pass
    twiddleFactorMul(data, tid, 1024, 0);
```

OpenCL Example Part 4

```
    // local shuffle using local memory
    localShuffle(data, sMemx, sMemy, tid, (((tid & 15) * 65) + (tid >>
   4)));
    fftRadix16Pass(data); // in-place radix-16 pass
    twiddleFactorMul(data, tid, 64, 4); // twiddle factor multiplication
    localShuffle(data, sMemx, sMemy, tid, (((tid >> 4) * 64) + (tid &
   15)));
    // four radix-4 function calls
    fftRadix4Pass(data);      // radix-4 function number 1
    fftRadix4Pass(data +  4); // radix-4 function number 2
    fftRadix4Pass(data +  8); // radix-4 function number 3
    fftRadix4Pass(data + 12); // radix-4 function number 4
    // coalesced global writes
    globalStores(data, out, 64);
}
```

Portland Group Accelerator Directives

- Proprietary directives in Fortran and C
- Similar to OpenMP in structure
- If the compiler doesn't understand these directives, it ignores them, so the same code can work with an accelerator or without, and with the PGI compilers or other compilers.
- In principle, this will be able to work on a variety of accelerators, but the first instance is NVIDIA; PGI recently announced a deal with AMD/ATI.
- The directives tell the compiler what parts of the code happen in the accelerator; the rest happens in the regular hardware.

PGI Accelerator Example

```
!$acc region
    do k = 1,n1
        do i = 1,n3
            c(i,k) = 0.0
            do j = 1,n2
                c(i,k) = c(i,k) +
&                        a(i,j) * b(j,k)
            enddo
        enddo
    enddo
!$acc end region
```

http://www.pgroup.com/resources/accel.htm

OpenMP 4.0 Accelerator Directives

- OpenMP's 4.0 standard is very much in discussion (and flux).
- It **may** end up with accelerator directives.
- It's too soon to say what the details will be, if it happens at all.
- But, if it happens, then codes amenable to accelerator directives will be able to get substantial speedups with very modest coding effort.

OpenMP 4.0 Accelerator Example

```
!$omp acc_region
    do k = 1,n1
        do i = 1,n3
            c(i,k) = 0.0
            do j = 1,n2
                c(i,k) = c(i,k) +
&                            a(i,j) * b(j,k)
            enddo
        enddo
    enddo
!$omp end acc_region
```

http://www.pgroup.com/resources/accel.htm http://www.cse.scitech.ac.uk/events/GPU_2010/12_Hart.pdf

Digging Deeper: CUDA on NVIDIA

NVIDIA Tesla

- NVIDIA now offers a GPU platform named Tesla.
- It consists essentially of their highest end graphics card, minus the video out connector.

http://images.nvidia.com/products/geforce_gtx_480/geforce_gtx_480_3qtr_low.png

http://images.nvidia.com/products/tesla_C2050_C2070/Tesla_C2050_C2070_3qtr_low_new.png

NVIDIA Tesla C2050 Card Specs

- 448 GPU cores
- 1.15 GHz
- Single precision floating point performance:
  1030.4 GFLOPs (2 single precision flops per clock per core)
- Double precision floating point performance:
  515.2   GFLOPs (1 double precision flop per clock per core)
- Internal RAM: 3 GB DDR5
- Internal RAM speed: 144 GB/sec (compared 21-25 GB/sec for regular RAM)
- Has to be plugged into a PCIe slot (at most 8 GB/sec per GPU card)

NVIDIA Tesla S2050 Server Specs

- 4 C2050 cards inside a 1U server (looks like a Sooner node)
- 1.15 GHz
- Single Precision (SP) floating point performance: 4121.6 GFLOPs
- Double Precision (DP) floating point performance: 2060.8 GFLOPs
- Internal RAM: 12 GB total (3 GB per GPU card)
- Internal RAM speed: 576 GB/sec aggregate
- Has to be plugged into two PCIe slots (at most 16 GB/sec for 4 GPU cards)

Compare x86 vs S2050

Let’s compare the best dual socket x86 server today vs S2050.

	Dual socket, AMD 2.3 GHz 12-core	**NVIDIA Tesla S2050**
Peak DP FLOPs	220.8 GFLOPs DP	2060.8 GFLOPs DP (9.3x)
Peak SP FLOPS	441.6 GFLOPs SP	4121.6 GFLOPs SP (9.3x)
Peak RAM BW	25 GB/sec	576 GB/sec (23x)
Peak PCIe BW	N/A	16 GB/sec
Needs x86 server to attach to?	No	Yes
Power/Heat	~450 W	~900 W + ~400 W (~2.9x)
Code portable?	Yes	No (CUDA) Yes (PGI, OpenCL)

Compare x86 vs S2050

Here are some interesting measures:

	Dual socket, AMD 2.3 GHz 12-core	**NVIDIA Tesla S2050**
DP GFLOPs/Watt	~0.5 GFLOPs/Watt	~1.6 GFLOPs/Watt (~3x)
SP GFLOPS/Watt	~1 GFLOPs/Watt	~3.2 GFLOPs/Watt (~3x)
DP GFLOPs/sq ft	~590 GFLOPs/sq ft	~2750 GFLOPs/sq ft (4.7x)
SP GFLOPs/sq ft	~1180 GFLOPs/sq ft	~5500 GFLOPs/sq ft (4.7x)
Racks per PFLOP DP	142 racks/PFLOP DP	32 racks/PFLOP DP (23%)
Racks per PFLOP SP	71 racks/PFLOP SP	16 racks/PFLOP SP (23%)

Kepler and Maxwell

- NVIDIA's 20-series is also known by the codename "Fermi." It runs at about 0.5 TFLOPs per GPU card (peak).
- The next generation, to be released in 2011, is codenamed "Kepler" and will be capable of something like 1.4 TFLOPs double precision per GPU card.
- After "Kepler" will come "Maxwell" in 2013, capable of something like 4 TFLOPs double precision per GPU card.
- So, the increase in performance is likely to be roughly 2.5x – 3x per generation, roughly every two years.

http://www.vizworld.com/2010/09/thoughts-nvidias-kepler-maxwell-gpus/

What Are the Downsides?

- You have to rewrite your code into CUDA or OpenCL or PGI accelerator directives (or someday maybe OpenMP).
  - CUDA: Proprietary, but maybe portable soon
  - OpenCL: portable but cumbersome
  - PGI accelerator directives: not clear whether you can have most of the code live inside the GPUs.
- BUT: Many groups are coming out with GPGPU code development tools that may help a lot, such as:
  - Fortran-to-CUDA-C converter (NCAR)
  - CUDA C automatic optimizer (memory, threading etc)
  - OpenMP-to-CUDA converter
  - CUDA-to-x86 converter (CUDA code on non-CUDA system)

Programming for Performance

The biggest single performance bottleneck on GPU cards today is the PCIe slot:

- PCIe 2.0 x16: 8 GB/sec
- 1600 MHz Front Side Bus: 25 GB/sec
- GDDR5 GPU card RAM: 144 GB/sec per card

Your goal:

- At startup, move the data from x86 server RAM into GPU RAM.
- Do almost all the work inside the GPU.
- Use the x86 server only for I/O and message passing, to minimize the amount of data moved through the PCIe slot.

Does CUDA Help?

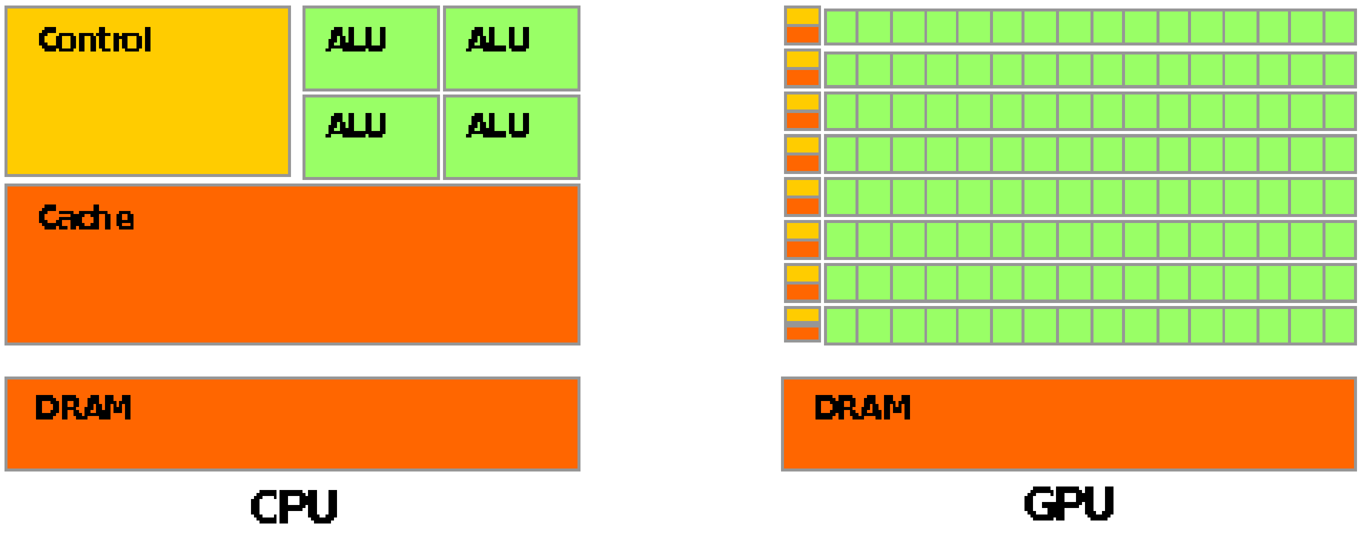
Example Applications	URL	Speedup
Seismic Database	http://www.headwave.com	66x – 100x
Mobile Phone Antenna Simulation	http://www.accelware.com	45x
Molecular Dynamics	http://www.ks.uiuc.edu/Research/vmd	21x – 100x
Neuron Simulation	http://www.evolvedmachines.com	100x
MRI Processing	http://bic-test.beckman.uiuc.edu	245x – 415x
Atmospheric Cloud Simulation	http://www.cs.clemson.edu/~jesteel/clouds.html	50x

http://www.nvidia.com/object/IO_43499.html

CUDA
Thread Hierarchy and Memory Hierarchy

Some of these slides provided by Paul Gray, University of Northern Iowa

CPU vs GPU Layout



Source: NVIDIA CUDA Programming Guide

Buzzword: Kernel

In CUDA, a ***kernel*** is code (typically a function) that can be run inside the GPU.

Typically, the kernel code operates in lock-step on the stream processors inside the GPU.

Buzzword: Thread

In CUDA, a ***thread*** is an execution of a kernel with a given index.

Each thread uses its index to access a specific subset of the elements of a target array, such that the collection of all threads cooperatively processes the entire data set.

So these are very much like threads in the OpenMP or pthreads sense – they even have shared variables and private variables.

Buzzword: Block

In CUDA, a ***block*** is a group of threads.

- Just like OpenMP threads, these could execute concurrently or independently, and in no particular order.
- Threads can be coordinated somewhat, using the `_syncthreads()` function as a barrier, making all threads stop at a certain point in the kernel before moving on en mass. (This is like what happens at the end of an OpenMP loop.)

Buzzword: Grid

In CUDA, a ***grid*** is a group of (thread) blocks, with no synchronization at all among the blocks.

NVIDIA GPU Hierarchy

- ***Grids*** map to GPUs
- ***Blocks*** map to the MultiProcessors (MP)
  - Blocks are never split across MPs, but an MP can have multiple blocks
- ***Threads*** map to Stream Processors (SP)
- ***Warps*** are groups of (32) threads that execute simultaneously

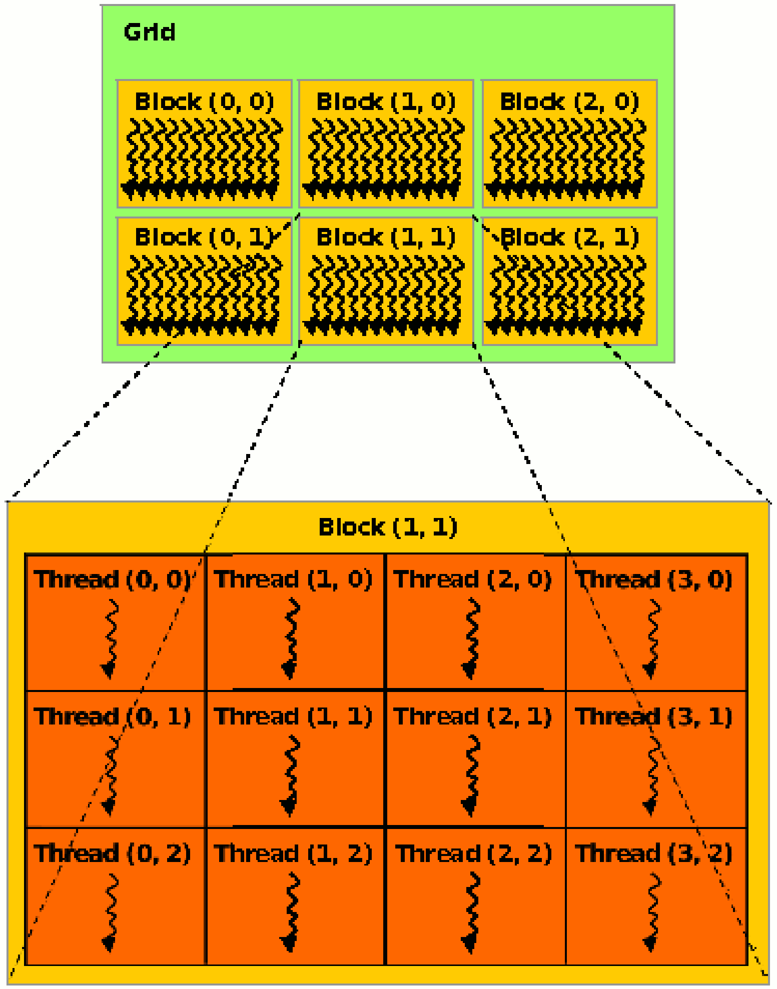


Image Source:
NVIDIA CUDA Programming Guide

CUDA Built-in Variables

- `blockIdx.x, blockIdx.y, blockIdx.z` are built-in variables that returns the block ID in the x-axis, y-axis and z-axis of the block that is executing the given block of code.
- `threadIdx.x, threadIdx.y, threadidx.z` are built-in variables that return the thread ID in the x-axis, y-axis and z-axis of the thread that is being executed by this stream processor in this particular block.

So, you can express your collection of blocks, and your collection of threads within a block, as a 1D array, a 2D array or a 3D array.

These can be helpful when thinking of your data as 2D or 3D.

`__global__` Keyword

In CUDA, if a function is declared with the **`__global__`** keyword, that means that it's intended to be executed inside a GPU.

In CUDA, the term for the GPU is ***device***, and the term for the x86 server is ***host***.

So, a kernel runs on a device, while the main function, and so on, run on the host.

Note that a host can play host to multiple devices; for example, an S2050 server contains 4 C2050 GPU cards, and if a single host has two PCIe slots, then both of the PCIe plugs of the S2050 can be plugged into that same host.

Copying Data from Host to Device

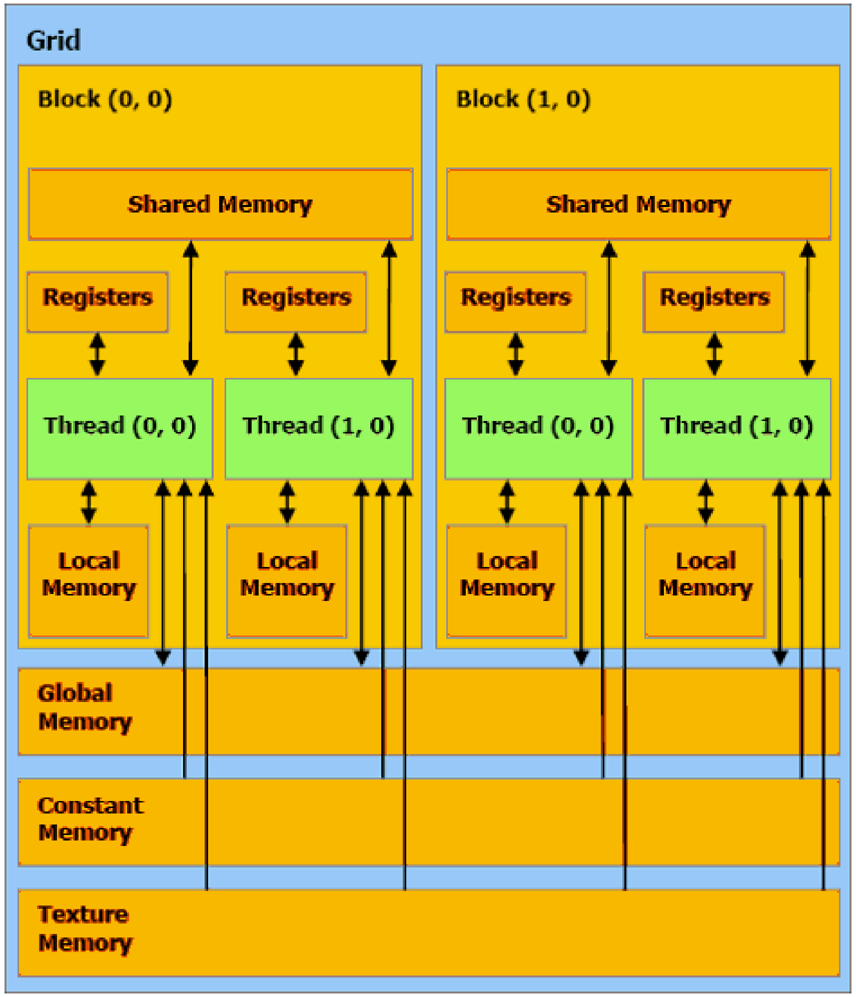
If data need to move from the host (where presumably the data are initially input or generated), then a copy has to exist in both places.

Typically, what's copied are arrays, though of course you can also copy a scalar (the address of which is treated as an array of length 1).

CUDA Memory Hierarchy #1

CUDA has a hierarchy of several kinds of memory:

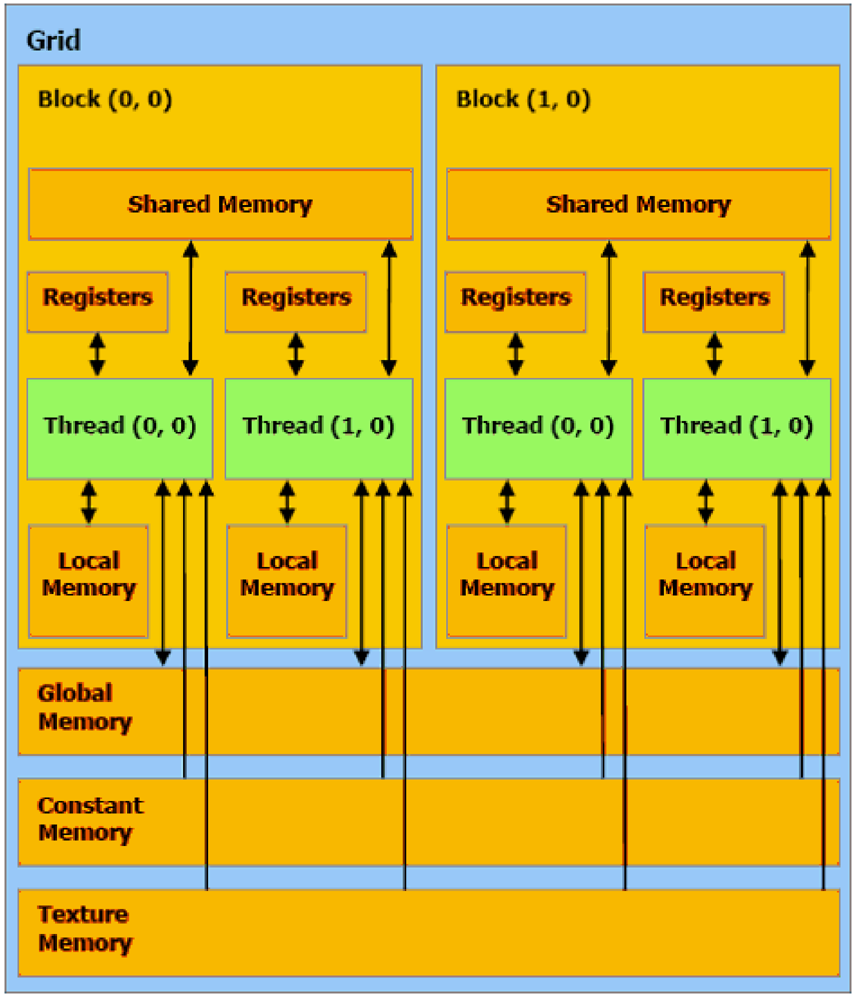
- Host memory (x86 server)
- Device memory (GPU)
  - ***Global***: visible to all threads in all blocks – largest, slowest
  - ***Shared***: visible to all threads in a particular block – medium size, medium speed
  - ***Local***: visible only to a particular thread – smallest, fastest



CUDA Memory Hierarchy #2

CUDA has a hierarchy of several kinds of memory:

- Host memory (x86 server)
- Device memory (GPU)
  - ***Constant***: visible to all threads in all blocks; read only
  - ***Texture***: visible to all threads in all blocks; read only



CUDA Example: Matrix-Matrix Multiply

http://developer.download.nvidia.com/compute/cuda/sdk/website/Linear_Algebra.html#matrixMul

Matrix-Matrix Multiply Main Part 1

```
float* host_A;
float* host_B;
float* host_B;
float* device_A;
float* device_B;
float* device_C;

host_A = (float*) malloc(mem_size_A);
host_B = (float*) malloc(mem_size_B);
host_C = (float*) malloc(mem_size_C);

cudaMalloc((void**) &device_A, mem_size_A);
cudaMalloc((void**) &device_B, mem_size_B);
cudamalloc((void**) &device_C, mem_size_C);

// Set up the initial values of A and B here.

// Henry says: I've oversimplified this a bit from
// the original example code.
```

Matrix-Matrix Multiply Main Part 2

```
// copy host memory to device
cudaMemcpy(device_A, host_A, mem_size_A,
           cudaMemcpyHostToDevice);
cudaMemcpy(device_B, host_B, mem_size_B,
           cudaMemcpyHostToDevice);
// setup execution parameters
dim3 threads(BLOCK_SIZE, BLOCK_SIZE);
dim3 grid(WC / threads.x, HC / threads.y);

// execute the kernel
matrixMul<<< grid, threads >>>(device_C,
                               device_A, device_B, WA, WB);

// copy result from device to host
cudaMemcpy(host_C, device_C, mem_size_C,
           cudaMemcpyDeviceToHost);
```

Matrix Matrix Multiply Kernel Part 1

```
__global__ void matrixMul( float* C, float* A, float* B, int wA, int wB)
{
    // Block index
    int bx = blockIdx.x;
    int by = blockIdx.y;

    // Thread index
    int tx = threadIdx.x;
    int ty = threadIdx.y;

    // Index of the first sub-matrix of A processed by the block
    int aBegin = wA * BLOCK_SIZE * by;

    // Index of the last sub-matrix of A processed by the block
    int aEnd   = aBegin + wA - 1;

    // Step size used to iterate through the sub-matrices of A
    int aStep  = BLOCK_SIZE;

    // Index of the first sub-matrix of B processed by the block
    int bBegin = BLOCK_SIZE * bx;

    // Step size used to iterate through the sub-matrices of B
    int bStep  = BLOCK_SIZE * wB;

    // Csub is used to store the element of the block sub-matrix
    // that is computed by the thread
    float Csub = 0;
```

Matrix Matrix Multiply Kernel Part 2

```
// Loop over all the sub-matrices of A and B
// required to compute the block sub-matrix
for (int a = aBegin, b = bBegin;
         a <= aEnd;
         a += aStep, b += bStep) {

    // Declaration of the shared memory array As used to
    // store the sub-matrix of A
    __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];

    // Declaration of the shared memory array Bs used to
    // store the sub-matrix of B
    __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];

    // Load the matrices from device memory
    // to shared memory; each thread loads
    // one element of each matrix
    AS(ty, tx) = A[a + wA * ty + tx];
    BS(ty, tx) = B[b + wB * ty + tx];

    // Synchronize to make sure the matrices are loaded
    __syncthreads();
```

Matrix Matrix Multiply Kernel Part 3

```
        // Multiply the two matrices together;
        // each thread computes one element
        // of the block sub-matrix
        for (int k = 0; k < BLOCK_SIZE; ++k)
            Csub += AS(ty, k) * BS(k, tx);

        // Synchronize to make sure that the preceding
        // computation is done before loading two new
        // sub-matrices of A and B in the next iteration
        __syncthreads();
    }

    // Write the block sub-matrix to device memory;
    // each thread writes one element
    int c = wB * BLOCK_SIZE * by + BLOCK_SIZE * bx;
    C[c + wB * ty + tx] = Csub;
}
```

Would We Really Do It This Way?

We wouldn’t really do matrix-matrix multiply this way.

NVIDIA has developed a CUDA implementation of the BLAS libraries, which include a highly tuned matrix-matrix multiply routine.

(We’ll learn about BLAS next time.)

There’s also a CUDA FFT library, if your code needs Fast Fourier Transforms.

Summer Workshops 2011

- Introduction to Computational Thinking
  - Modeling and Simulation Across the Curriculum
    - June 19 - 25, Wayne State College, Wayne NE
    - July 10 - 16, Texas State Technical College, Waco
  - Preparing In-service and Pre-service Educators for Computational Thinking
    - July 24 - 30, Oregon State University, Corvallis
  - Computational Thinking from a Parallel Perspective
    - July 31 - Aug 6, Louisiana State University, Baton Rouge

Summer Workshops 2011 (cont'd)

- Computational Biology for Biology Educators
  - June 12 - 18, Lafayette College, Easton PA
  - June 26 - July 2, Calvin College, Grand Rapids MI
- Computational Chemistry for Chemistry Educators
  - July 24 - 30, Washington and Lee University, Lexington VA

  JOINTLY PRESENTED VIA VIDEOCONFERENCING WITH
  - July 24 - 30, Oklahoma State University, Stillwater
- Data-driven Computational Science: Modeling and Visualization
  - June 19 - 25, Richard Stockton College of New Jersey, Pomona

Summer Workshops 2011 (cont'd)

- Introduction to Parallel Programming & Cluster Computing
  - June 26 - July 1, Idaho State University, Pocatello

  JOINTLY PRESENTED VIA VIDEOCONFERENCING WITH
  - June 26 - July 1, University of Washington, Seattle
- Intermediate Parallel Programming & Cluster Computing
  - July 31 - Aug 6, University of Oklahoma, Norman

  JOINTLY PRESENTED VIA VIDEOCONFERENCING WITH
  - July 31 - Aug 6, Polytechnic University of Puerto Rico, Hato Rey

OK Supercomputing Symposium 2011

2003 Keynote:
Peter Freeman
NSF
Computer & Information Science & Engineering
Assistant Director

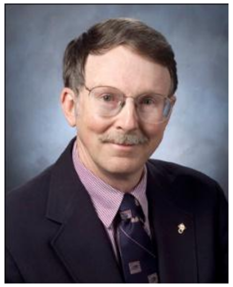
2004 Keynote:
Sangtae Kim
NSF Shared Cyberinfrastructure
Division Director

2005 Keynote:
Walt Brooks
NASA Advanced Supercomputing
Division Director

2006 Keynote:
Dan Atkins
Head of NSF's Office of Cyberinfrastructure

2007 Keynote:
Jay Boisseau
Director
Texas Advanced Computing Center
U. Texas Austin

2008 Keynote:
José Munoz
Deputy Office Director/ Senior Scientific Advisor
NSF Office of Cyberinfrastructure


2009 Keynote:
Douglass Post
Chief Scientist
US Dept of Defense HPC Modernization Program

2010 Keynote:
Horst Simon
Deputy Director
Lawrence Berkeley National Laboratory

?

2011 Keynote to be announced

FREE! Wed Oct 12 2011 @ OU

http://symposium2011.oscer.ou.edu/

Parallel Programming Workshop
FREE! Tue Oct 11 2011 @ OU
FREE! Symposium Wed Oct 12 2011 @ OU

REGISTRATION IS NOW OPEN!

SC11 Education Program

- At the SC11 supercomputing conference, we’ll hold our annual Education Program, Sat Nov 12 – Tue Nov 15.
- You can apply to attend, either fully funded by SC11 or self-funded.
- Henry is the SC11 Education Chair.
- We’ll alert everyone once the registration website opens.

Thanks for your attention!

Questions?

www.oscer.ou.edu