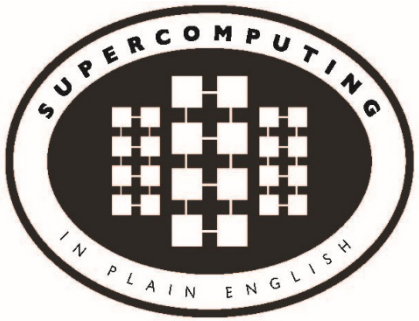
Supercomputing in Plain English

GPGPU: Number Crunching in Your Graphics Card

Henry Neeman, Director
Director, OU Supercomputing Center for Education & Research (OSCER)
Assistant Vice President, Information Technology – Research Strategy Advisor
Associate Professor, College of Engineering
Adjunct Associate Professor, School of Computer Science
University of Oklahoma
Tuesday April 14 2015




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.

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 e-mail.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

PLEASE REGISTER

If you haven't already registered, please do so.

You can find the registration link on the SiPE webpage:

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

Our ability to continue providing Supercomputing in Plain English depends on being able to show strong participation.

We use our headcounts, institution counts and state counts (since 2001, over 2000 served, from every US state except RI and VT, plus 17 other countries, on every continent except Australia and Antarctica) to improve grant proposals.

Download the Slides Beforehand

Before the start of the session, please download the slides from the Supercomputing in Plain English website:

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

That way, if anything goes wrong, you can still follow along with just audio.

PLEASE MUTE YOURSELF.

H.323 (Polycom etc) #1

If you want to use H.323 videoconferencing – for example, Polycom – then:

- If you AREN'T registered with the OneNet gatekeeper (which is probably the case), then:
  - Dial **`164.58.250.51`**
  - Bring up the virtual keypad.
    On some H.323 devices, you can bring up the virtual keypad by typing: #

    (You may want to try without first, then with; some devices won't work with the #, but give cryptic error messages about it.)
  - When asked for the conference ID, or if there's no response, enter: **`0409`**
  - On most but not all H.323 devices, you indicate the end of the ID with: **#**

H.323 (Polycom etc) #2

If you want to use H.323 videoconferencing – for example, Polycom – then:

- If you ARE already registered with the OneNet gatekeeper (most institutions aren't), dial:

  2500409

Many thanks to James Deaton, Skyler Donahue, Jeremy Wright and Steven Haldeman of OneNet for providing this.

PLEASE MUTE YOURSELF.

Wowza #1

You can watch from a Windows, MacOS or Linux laptop using Wowza from the following URL:

http://jwplayer.onenet.net/stream6/sipe.html

Wowza behaves a lot like YouTube, except live.

Many thanks to James Deaton, Skyler Donahue, Jeremy Wright and Steven Haldeman of OneNet for providing this.

PLEASE MUTE YOURSELF.

Wowza #2

Wowza has been tested on multiple browsers on each of:

- Windows (7 and 8): IE, Firefox, Chrome, Opera, Safari
- MacOS X: Safari, Firefox
- Linux: Firefox, Opera

PLEASE MUTE YOURSELF.

RTMP

If you have a video player that can handle RTMP, you can watch the Wowza feed that way:

`rtmp://stream3.onenet.net/live/mp4:sipe-wowza`

Toll Free Phone Bridge

IF ALL ELSE FAILS, you can use our toll free phone bridge:

800-832-0736

* 623 2874 #

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 **ONLY** if you cannot connect any other way: the phone bridge can handle only 100 simultaneous connections, and we have over 500 participants.

Many thanks to OU CIO Loretta Early for providing the toll free phone bridge.

PLEASE MUTE YOURSELF.

Please Mute Yourself

No matter how you connect, **PLEASE MUTE YOURSELF**, so that we cannot hear you.

(For Wowza, you don't need to do that, because the information only goes from us to you, not from you to us.)

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 e-mail.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

Questions via E-mail Only

Ask questions by sending e-mail to:

sipe2015@gmail.com

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

PLEASE MUTE YOURSELF.

Onsite: Talent Release Form

If you're attending onsite, you **MUST** do one of the following:

- complete and sign the Talent Release Form,

OR

- sit behind the cameras (where you can't be seen) and don't talk at all.

If you aren't onsite, then **PLEASE MUTE YOURSELF.**

TENTATIVE Schedule

Tue Jan 20: GPGPU: What the Heck is Supercomputing?
Tue Jan 27: The Tyranny of the Storage Hierarchy
Tue Feb 3: Instruction Level Parallelism
Tue Feb 10: Stupid Compiler Tricks
Tue Feb 17: Shared Memory Multithreading
Tue March 3: Distributed Multiprocessing
Tue March 10: Applications and Types of Parallelism
Tue March 17: **NO SESSION** (OU's Spring Break)
Tue March 24: **NO SESSION** (Henry has a huge grant proposal due)
Tue March 31: GPGPU Madness
Tue Apr 14: High Throughput Computing
Tue Apr 14: GPGPU: Number Crunching in Your Graphics Card
Tue Apr 21: Grab Bag: Scientific Libraries, I/O Libraries, Visualization

Thanks for helping!

- OU IT
  - OSCER operations staff (Brandon George, Dave Akin, Brett Zimmerman, Josh Alexander, Patrick Calhoun)
  - Horst Severini, OSCER Associate Director for Remote & Heterogeneous Computing
  - Debi Gentis, OSCER Coordinator
  - Jim Summers
  - The OU IT network team
- James Deaton, Skyler Donahue, Jeremy Wright and Steven Haldeman, OneNet
- Kay Avila, U Iowa
- Stephen Harrell, Purdue U

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.

PLEASE MUTE YOURSELF.

Coming in 2015!

- Linux Clusters Institute workshop May 18-22 2015 @ OU
  http://www.linuxclustersinstitute.org/workshops/
- Great Plains Network Annual Meeting, May 27-29, Kansas City
- Advanced Cyberinfrastructure Research & Education Facilitators (ACI-REF) Virtual Residency May 31 - June 6 2015
- XSEDE2015, July 26-30, St. Louis MO
  https://conferences.xsede.org/xsede15
- IEEE Cluster 2015, Sep 23-27, Chicago IL
  http://www.mcs.anl.gov/ieeecluster2015/
- OKLAHOMA SUPERCOMPUTING SYMPOSIUM 2015, **Sep 22-23 2015** @ OU
- SC13, Nov 15-20 2015, Austin TX
  http://sc15.supercomputing.org/

PLEASE MUTE YOURSELF.

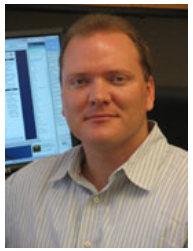
OK Supercomputing Symposium 2015

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

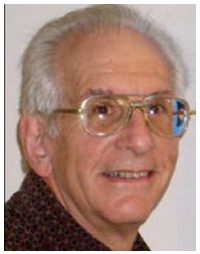


2007 Keynote:
Jay Boisseau
Director
Texas Advanced Computing Center
U. Texas Austin

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:
Barry Schneider
Program Manager
National Science Foundation

2012 Keynote:
Thom Dunning
Director
National Center for Supercomputing Applications

2015 Keynote:
John Shalf
Dept Head CS
Lawrence Berkeley Lab
CTO, NERSC

2014 Keynote:
Irene Qualters
Division Director
Advanced Cyberinfarstructure Division, NSF

FREE!
Wed Sep 23 2015 @ OU

Reception/Poster Session
Tue Sep 22 2015 @ OU
Symposium
Wed Sep 23 2015 @ OU

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 can be hard to program;
- your code on them may not be portable to other accelerators, so the labor you invest in programming them can have 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://blog.xcelerit.com/benchmarks-nvidia-kepler-vs-fermi/

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

What does 1 TFLOPs Look Like?

1997: Room

ASCI RED[13]
Sandia National Lab

2002: Row

`boomer.oscer.ou.edu`
In service 2002-5: 11 racks

2012: Card

AMD FirePro W9000[14]

NVIDIA Kepler K20[15]

Intel MIC Xeon PHI[16]

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 FY2013, NVIDIA sold about $2-3B of GPUs (out of something like $4B total revenue).
- For example, in 2013, GPUs sold 413 million units.
  http://www.tgdaily.com/enterprise/126591-graphics-chips-market-is-showing-some-life
- 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***.

Intel MIC

- First production (non-research) model: Xeon Phi.
- Not a graphics card.
- But, has similar structure to a graphics card, just without the graphics.
- Based on x86: can use a lot of the same tools as CPU.
- 61 x86 cores, 512-bit vector widths (8-way double precision floating point vectors, up to 16 DP floating point calculations per clock cycle using Fused Multiply-Add).
- 8 GB GDDR5 Graphics RAM, 352 GB/sec
- Peak ~1070 GFLOPs per card (i.e., OSCER's first cluster supercomputer in 2002).

http://www.tacc.utexas.edu/user-services/user-guides/stampede-user-guide
https://secure-software.intel.com/sites/default/files/article/334766/intel-xeon-phi-systemsoftwaredevelopersguide.pdf

How to Program a GPU

- Proprietary programming language or extensions
  - NVIDIA: CUDA (C/C++)
  - AMD/ATI: StreamSDK/Brook+ (C/C++) – defunct
- 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).
  - PGI is now part of NVIDIA.
- OpenACC accelerator directives for NVIDIA and AMD
- OpenMP version 4.0 includes accelerator and vectorization directives.

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.
  - PGI is now part of NVIDIA.

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);
}
```

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.
- Currently on version 2.1 (released March 3 2015).

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.
- 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

OpenACC Compiler Directives

- Open standard for accelerator directives
- Developed by NVIDIA, Cray, PGI, CAPS
- Available in PGI and CAPS compilers for general cluster user, in Cray compilers for use on Crays

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

OpenACC Example Part 1 (C)

```
#include <stdio.h>
#include <stdlib.h>
void vecaddgpu( float *restrict r, float *a, float *b, int n ){
  #pragma acc kernels loop copyin(a[0:n],b[0:n]) copyout(r[0:n])
  for( int i = 0; i < n; ++i ) r[i] = a[i] + b[i];
}

/* http://www.pgroup.com/doc/openacc_gs.pdf */
```

OpenACC Example Part 2 (C)

```
int main( int argc, char* argv[] ){
  int n; /* vector length */
  float * a; /* input vector 1 */
  float * b; /* input vector 2 */
  float * r; /* output vector */
  float * e; /* expected output values */
  int i, errs;

  if( argc > 1 ) n = atoi( argv[1] );
  else n = 100000; /* default vector length */
  if( n <= 0 ) n = 100000;
  a = (float*)malloc( n*sizeof(float) );
  b = (float*)malloc( n*sizeof(float) );
  r = (float*)malloc( n*sizeof(float) );
  e = (float*)malloc( n*sizeof(float) );
  for( i = 0; i < n; ++i ){
    a[i] = (float)(i+1);
    b[i] = (float)(1000*i);
  }
```

OpenACC Example Part 3 (C)

```
  /* compute on the GPU */
  vecaddgpu( r, a, b, n );
  /* compute on the host to compare */
  for( i = 0; i < n; ++i ) e[i] = a[i] + b[i];
  /* compare results */
  errs = 0;
  for( i = 0; i < n; ++i ){
    if( r[i] != e[i] ){
      ++errs;
    }
  }
  printf( "%d errors found\n", errs );
  return errs;
}
```

OpenACC Example Part 1 (F90)

```
module vecaddmod
  implicit none
  contains
    subroutine vecaddgpu( r, a, b, n )
      real, dimension(:) :: r, a, b
      integer :: n
      integer :: i
!$acc kernels loop copyin(a(1:n),b(1:n)) copyout(r(1:n))
      do i = 1, n
        r(i) = a(i) + b(i)
      enddo
    end subroutine
end module

! http://www.pgroup.com/doc/openacc_gs.pdf
```

OpenACC Example Part 2 (F90)

```
program main
  use vecaddmod
  implicit none
  integer :: n, i, errs, argcount
  real, dimension(:), allocatable :: a, b, r, e
  character*10 :: arg1
  argcount = command_argument_count()
  n = 1000000 ! default value
  if( argcount = 1 )then
    call get_command_argument( 1, arg1 )
    read( arg1, '(i)' ) n
    if( n <= 0 ) n = 100000
  endif
  allocate( a(n), b(n), r(n), e(n) )
  do i = 1, n
    a(i) = i
    b(i) = 1000*i
  enddo
```

OpenACC Example Part 3 (F90)

```
  ! compute on the GPU
  call vecaddgpu( r, a, b, n )
  ! compute on the host to compare
  do i = 1, n
    e(i) = a(i) + b(i)
  enddo
  ! compare results
  errs = 0
  do i = 1, n
    if( r(i) /= e(i) )then
      errs = errs + 1
    endif
  enddo
  print *, errs, ' errors found'
  if( errs ) call exit(errs)
end program
```

OpenMP 4.0 Accelerator Directives

- OpenMP's 4.0 standard was released in 2013.
- It appears has both accelerator directives and vectorization directives.
- It's the *lingua franca* of the Intel MIC accelerator.

OpenMP Accelerator Example (F90)

```
! snippet from the hand-coded subprogram...
!dir$ attributes offload:mic :: my_sgemm
subroutine my_sgemm(d,a,b)
real, dimension(:,:) :: a, b, d
!$omp parallel do
do j=1, n
  do i=1, n
    d(i,j) = 0.0
    do k=1, n
      d(i,j) = d(i,j)+a(i,k)*b(k,j)
    enddo
  enddo
enddo
end subroutine
```

http://www.cac.cornell.edu/education/training/ParallelFall2012/OpenMPNov2012.pdf

Digging Deeper: CUDA on NVIDIA

NVIDIA Tesla

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

NVIDIA Tesla K80 Card Specs

- 4992 GPU cores
- 562 MHz core clock, 875 GHz boost clock
- Single precision floating point performance:
  8740 GFLOPs (2 single precision flops per clock per core)
- Double precision floating point performance:
  2910 GFLOPs (2/3 double precision flop per clock per core)
- Internal RAM: 24 GB DDR5
- Internal RAM speed: 480 GB/sec (compared ~100 GB/sec for regular RAM)
- Has to be plugged into a PCIe slot (at most 16 GB/sec per GPU card)

http://www.nvidia.com/object/tesla-servers.html
http://www.anandtech.com/show/8729/nvidia-launches-tesla-k80-gk210-gpu
http://en.wikipedia.org/wiki/Nvidia_Tesla

Compare Top x86 vs NVIDIA K80

Let's compare the best dual socket x86 server today vs K80.

	Dual socket, Intel 2.3 GHz 18-core	**NVIDIA Tesla K80 dual cards in an x86 server**
Peak DP FLOPs	1324.8 GFLOPs DP	5820 GFLOPs DP (4.4x)
Peak SP FLOPS	2649.6 GFLOPs SP	17,480 GFLOPs SP (6.6x)
Peak RAM BW	~100 GB/sec	~480 GB/sec (4.8x)
Peak PCIe BW	N/A	16 GB/sec
Needs x86 server to be part of?	No	Yes
Power/Heat	~350 W	2 × 300 W + ~400 W (~2.9x)
Code portable?	Yes	No (CUDA) Yes (OpenACC, OpenCL)

Compare Top x86 vs NVIDIA K80

Here are some interesting measures:

	Dual socket, Intel 2.3 GHz 18-core	**NVIDIA Tesla K80**
DP GFLOPs/Watt	~3.8 GFLOPs/Watt	~5.8 GFLOPs/Watt (~1.5x)
SP GFLOPS/Watt	~7.6 GFLOPs/Watt	~17.5 GFLOPs/Watt (~2.3x)
DP TFLOPs/sq ft	~10.6 TFLOPs/sq ft	~23.3 TFLOPs/sq ft (2.2x)
SP TFLOPs/sq ft	~21.2 TFLOPs/sq ft	~69.9 TFLOPs/sq ft (3.3x)
Racks per PFLOP DP	16 racks/PFLOP DP	8 racks/PFLOP DP (50%)
Racks per PFLOP SP	8 racks/PFLOP SP	3 racks/PFLOP SP (38%)

What Are the Downsides?

- You have to rewrite your code into CUDA or OpenCL or PGI accelerator directives (or someday maybe OpenMP).
  - CUDA: Proprietary
  - OpenCL: portable but cumbersome
  - OpenACC, OpenMP 4.0: portable, but which to choose?
    - They may be merging ….

Programming for Performance

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

- PCI 3.0 x16: 16 GB/sec
- 2133 MHz current architectures: up to ~100 GB/sec per server
- GDDR5 accelerator card RAM: 480 GB/sec per card

Your goal:

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

TENTATIVE Schedule

Tue Jan 20: GPGPU: What the Heck is Supercomputing?
Tue Jan 27: The Tyranny of the Storage Hierarchy
Tue Feb 3: Instruction Level Parallelism
Tue Feb 10: Stupid Compiler Tricks
Tue Feb 17: Shared Memory Multithreading
Tue March 3: Distributed Multiprocessing
Tue March 10: Applications and Types of Parallelism
Tue March 17: **NO SESSION** (OU's Spring Break)
Tue March 24: **NO SESSION** (Henry has a huge grant proposal due)
Tue March 31: GPGPU Madness
Tue Apr 14: High Throughput Computing
Tue Apr 14: GPGPU: Number Crunching in Your Graphics Card
Tue Apr 21: Grab Bag: Scientific Libraries, I/O Libraries, Visualization

Thanks for helping!

- OU IT
  - OSCER operations staff (Brandon George, Dave Akin, Brett Zimmerman, Josh Alexander, Patrick Calhoun)
  - Horst Severini, OSCER Associate Director for Remote & Heterogeneous Computing
  - Debi Gentis, OSCER Coordinator
  - Jim Summers
  - The OU IT network team
- James Deaton, Skyler Donahue, Jeremy Wright and Steven Haldeman, OneNet
- Kay Avila, U Iowa
- Stephen Harrell, Purdue U

Coming in 2015!

Linux Clusters Institute workshop May 18-22 2015 @ OU

http://www.linuxclustersinstitute.org/workshops/

Great Plains Network Annual Meeting, May 27-29, Kansas City

Advanced Cyberinfrastructure Research & Education Facilitators (ACI-REF) Virtual Residency May 31 - June 6 2015

XSEDE2015, July 26-30, St. Louis MO

https://conferences.xsede.org/xsede15

IEEE Cluster 2015, Sep 23-27, Chicago IL

http://www.mcs.anl.gov/ieeecluster2015/

OKLAHOMA SUPERCOMPUTING SYMPOSIUM 2015, **Sep 22-23 2015** @ OU

SC13, Nov 15-20 2015, Austin TX

http://sc15.supercomputing.org/

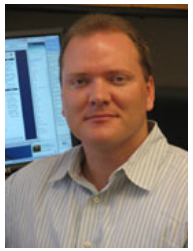
OK Supercomputing Symposium 2015

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

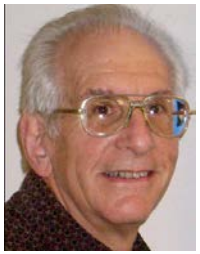


2007 Keynote:
Jay Boisseau
Director
Texas Advanced Computing Center
U. Texas Austin

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:
Barry Schneider
Program Manager
National Science Foundation

2012 Keynote:
Thom Dunning
Director
National Center for Supercomputing Applications

2015 Keynote:
John Shalf
Dept Head CS
Lawrence Berkeley Lab
CTO, NERSC

2014 Keynote:
Irene Qualters
Division Director
Advanced Cyberinfarstructure Division, NSF

FREE!
Wed Sep 23 2015
@ OU

Reception/Poster Session
Tue Sep 22 2015 @ OU
Symposium
Wed Sep 23 2015 @ OU

Thanks for your attention!

Questions?

www.oscer.ou.edu

Does CUDA Help?

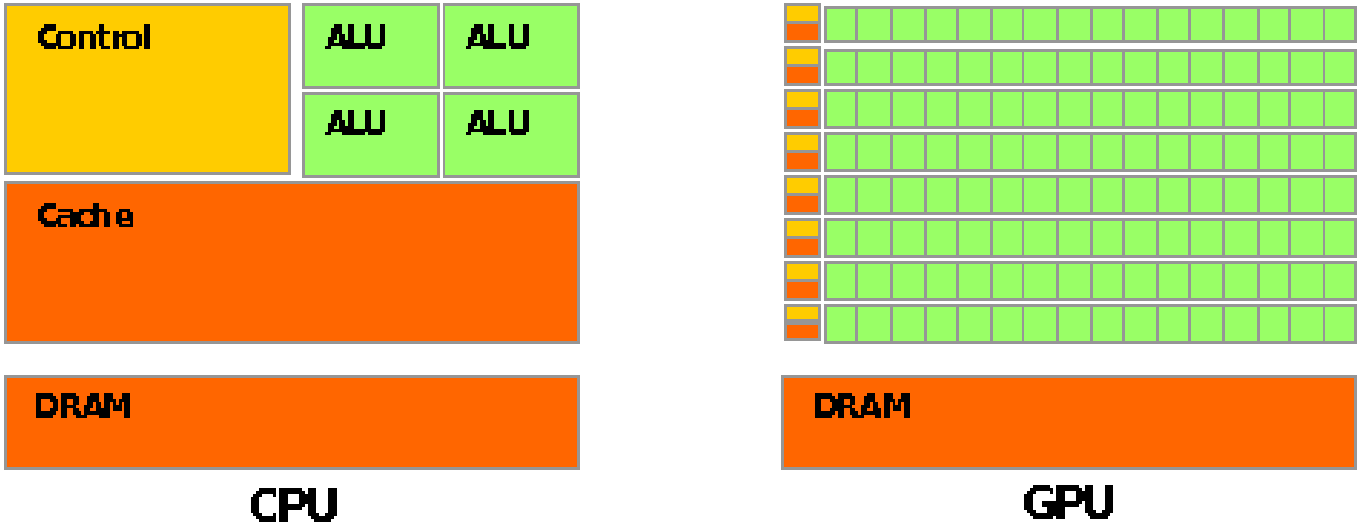
Example Applications	URL	Speedup
Seismic Database	http://www.headwave.com	66x – 100x
Mobile Phone Antenna Simulation	http://www.accelware.com	45x
Molecular Dynamics	http://www.ks.uiuc.edu/Research/vmd	21x – 100x
Neuron Simulation	http://www.evolvedmachines.com	100x
MRI Processing	http://bic-test.beckman.uiuc.edu	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

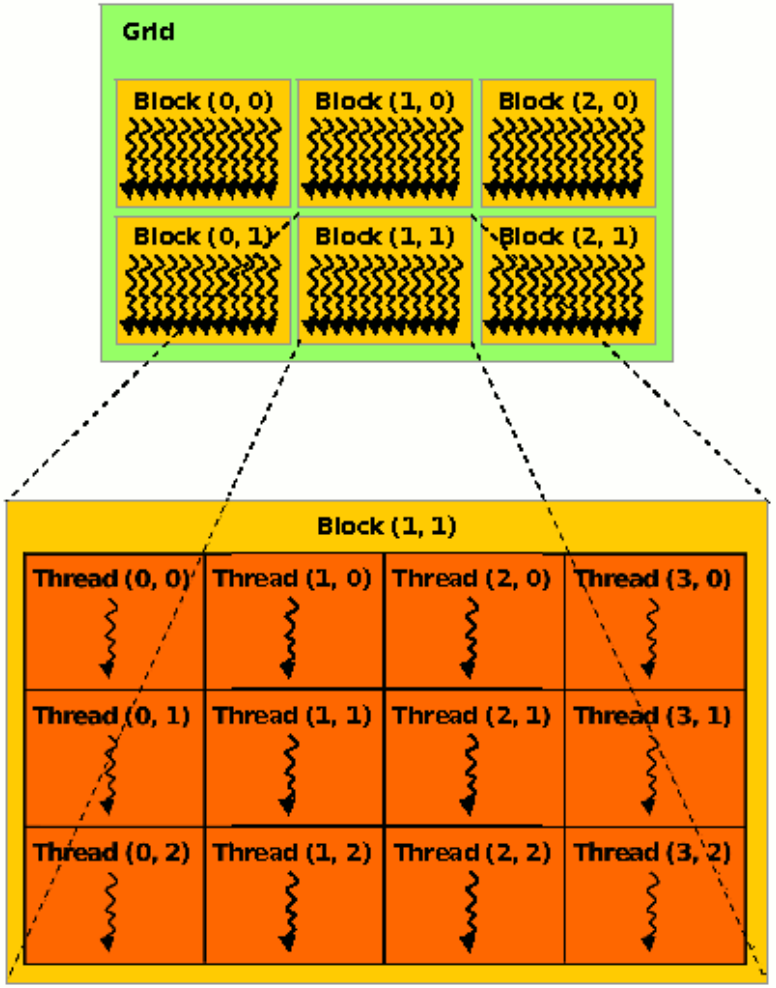


Image Source:
NVIDIA CUDA Programming Guide

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

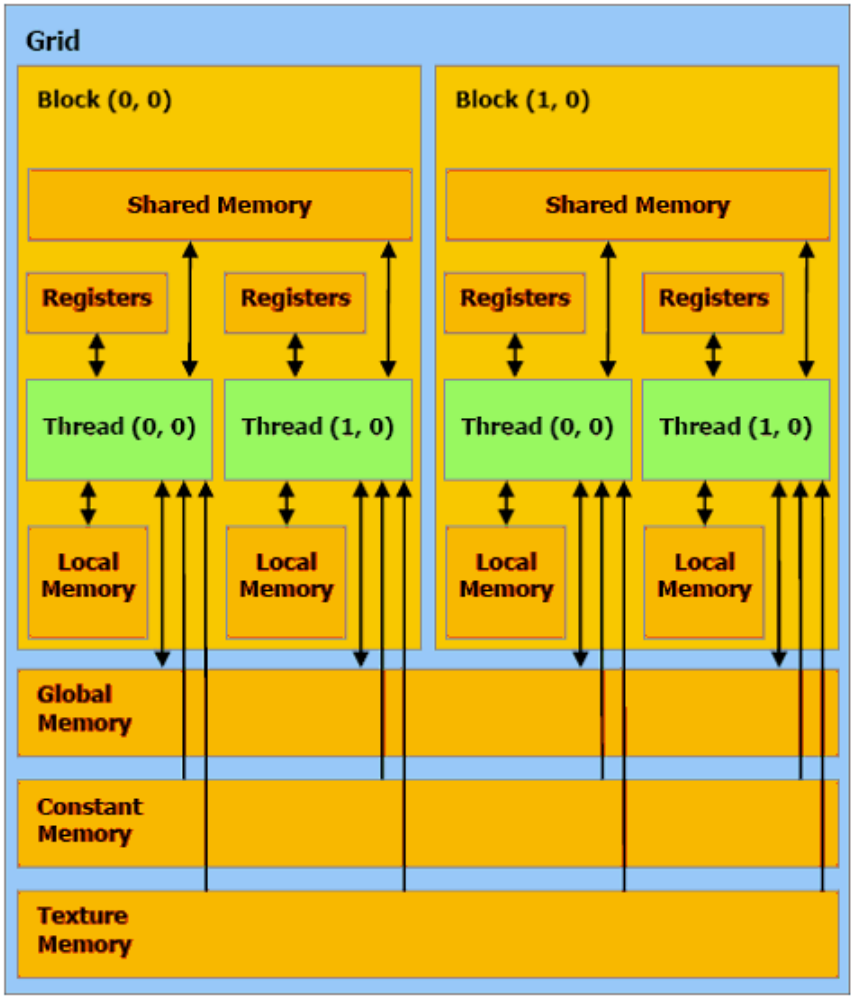
If data need to move from the host (where presumably the data are initially input or generated), then a copy has to exist in both places.

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:

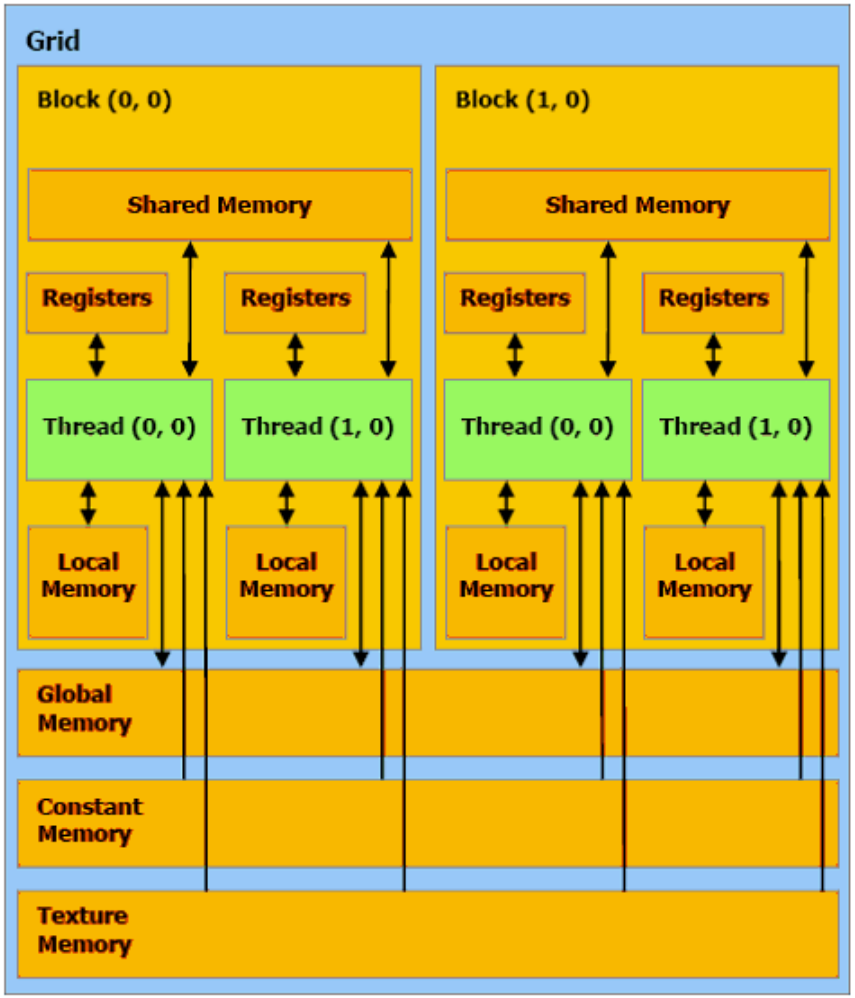
- Host memory (x86 server)
- Device memory (GPU)
  - ***Global***: visible to all threads in all blocks – largest, slowest
  - ***Shared***: visible to all threads in a particular block – medium size, medium speed
  - ***Local***: visible only to a particular thread – smallest, fastest



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.

Thanks for your attention!

Questions?

www.oscer.ou.edu