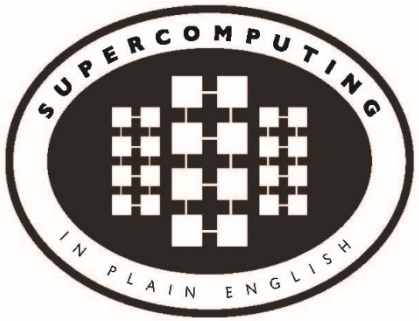
Supercomputing in Plain English

GPGPU: Number Crunching in Your Graphics Card

Henry Neeman, University of Oklahoma
Director, OU Supercomputing Center for Education & Research (OSCER)
Assistant Vice President, Information Technology – Research Strategy Advisor
Associate Professor, Gallogly College of Engineering
Adjunct Associate Professor, School of Computer Science
Tuesday April 24 2018



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 phone bridge to fall back on.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

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:

supercomputinginplainenglish@gmail.com

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

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.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

Zoom

Go to:

http://zoom.us/j/979158478

Many thanks Eddie Huebsch, OU CIO, for providing this.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

YouTube

You can watch from a Windows, MacOS or Linux laptop or an Android or iOS handheld using YouTube.

Go to YouTube via your preferred web browser or app, and then search for:

`Supercomputing InPlainEnglish`

(**`InPlainEnglish`** is all one word.)

Many thanks to Skyler Donahue of OneNet for providing this.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

Twitch

You can watch from a Windows, MacOS or Linux laptop or an Android or iOS handheld using Twitch.

Go to:

http://www.twitch.tv/sipe2018

Many thanks to Skyler Donahue of OneNet for providing this.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

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/streams/sipe.html

If that URL fails, then go to:

http://jwplayer.onenet.net/streams/sipebackup.html

Many thanks to Skyler Donahue of OneNet for providing this.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

Wowza #2

Wowza has been tested on multiple browsers on each of:

- Windows 10: IE, Firefox, Chrome, Opera, Safari
- MacOS: Safari, Firefox
- Linux: Firefox, Opera

We've also successfully tested it via apps on devices with:

- Android
- iOS

Many thanks to Skyler Donahue of OneNet for providing this.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

Toll Free Phone Bridge

IF ALL ELSE FAILS, you can use our US TOLL phone bridge:

405-325-6688

684 684 #

NOTE: This is for **US** call-ins **ONLY**.

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 1000 participants.

Many thanks to OU CIO Eddie Huebsch for providing the phone bridge..

Please Mute Yourself

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

(For YouTube, Twitch and 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.

Questions via E-mail Only

Ask questions by sending e-mail to:

supercomputinginplainenglish@gmail.com

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

DON'T USE CHAT OR VOICE FOR QUESTIONS!

No one will be monitoring any of the chats, and if we can hear your question, you're creating an **echo cancellation** problem.

PLEASE MUTE YOURSELF.
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 23: Storage: What the Heck is Supercomputing?
Tue Jan 30: The Tyranny of the Storage Hierarchy Part I
Tue Feb 6: The Tyranny of the Storage Hierarchy Part II
Tue Feb 13: Instruction Level Parallelism
Tue Feb 20: Stupid Compiler Tricks
Tue Feb 27: Multicore Multithreading
Tue March 6: Distributed Multiprocessing
Tue March 13: **NO SESSION** (Henry business travel)
Tue March 20: **NO SESSION** (OU's Spring Break)
Tue March 27: Applications and Types of Parallelism
Tue Apr 3: Multicore Madness
Tue Apr 10: **NO SESSION** (Henry business travel)
Tue Apr 17: High Throughput Computing
Tue Apr 24: GPU: Number Crunching in Your Graphics Card
Tue May 1: Grab Bag: Scientific Libraries, I/O Libraries, Visualization

Thanks for helping!

- OU IT
  - OSCER operations staff (Dave Akin, Patrick Calhoun, Kali McLennan, Jason Speckman, Brett Zimmerman)
  - OSCER Research Computing Facilitators (Jim Ferguson, Horst Severini)
  - Debi Gentis, OSCER Coordinator
  - Kyle Dudgeon, OSCER Manager of Operations
  - Ashish Pai, Managing Director for Research IT Services
  - The OU IT network team
  - OU CIO Eddie Huebsch
- OneNet: Skyler Donahue
- Oklahoma State U: Dana Brunson

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 phone bridge to fall back on.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

Coming in 2018!

- Linux Clusters Institute workshops

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

  - Introductory HPC Cluster System Administration: May 14-18 2018 @ U Nebraska, Lincoln NE USA
  - Intermediate HPC Cluster System Administration: Aug 13-17 2018 @ Yale U, New Haven CT USA
- Great Plains Network Annual Meeting: details coming soon
- Advanced Cyberinfrastructure Research & Education Facilitators (ACI-REF) Virtual Residency Aug 5-10 2018, U Oklahoma, Norman OK USA
- PEARC 2018, July 22-27, Pittsburgh PA USA

  https://www.pearc18.pearc.org/
- IEEE Cluster 2018, Sep 10-13, Belfast UK

  https://cluster2018.github.io
- **OKLAHOMA SUPERCOMPUTING SYMPOSIUM 2018, Sep 25-26 2018 @ OU**
- SC18 supercomputing conference, Nov 11-16 2018, Dallas TX USA

  http://sc18.supercomputing.org/

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 (though often cheaper per unit of computing speed);
- 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.**

https://www.amd.com/en/products/professional-graphics/radeon-pro-ssg

https://www.nvidia.com/en-us/data-center/tesla-v100/

https://www.xcelerit.com/computing-benchmarks/insights/benchmarks-intel-xeon-phi-knl-vs-broadwell-cpu/

What does 1 TFLOPs Look Like?

1 TFLOPs: trillion calculations per second

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]

CPU Chip 2017

AMD EPYC

Intel Skylake

https://www.top500.org/static/media/uploads/.thumbnails/epyc-vs-xeon.jpg/epyc-vs-xeon-742x382.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 calendar year 2017, NVIDIA sold ~$5.7B of GPUs, which was ~80% of their total revenue.
  https://marketrealist.com/2017/08/data-center-nvidias-key-growth-driver-fiscal-2018
- This means that the GPU companies have been able to recoup the huge fixed costs.

GPUs 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 ten years or so – 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*** (GPGPU).

Intel MIC

- Also known as 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.
- Current series: “Knights Landing” (model numbers 72xx)
  - Comes in chip form only (not card), for single socket server.
  - 64 to 72 x86 cores, each with 512-bit vector widths (8-way double precision floating point vectors), 2 Fused Multiply-Add units per core, so up to 32 DP floating point calculations per core per cycle.
  - 16 GB MCDRAM @ 400+ GB/sec.
  - Server can have up to 384 GB RAM @ 102.4 GB/sec.
  - Peak 2.6624 to 3.4560 TFLOPs per card.

https://en.wikipedia.org/wiki/Xeon_Phi

How to Program a GPU

- Proprietary programming language or extensions
  - NVIDIA: CUDA (C/C++)
- OpenCL (Open Computing Language): an industry standard for doing number crunching on GPUs.
- OpenACC accelerator directives for NVIDIA and AMD
- OpenMP version 4.x includes accelerator and vectorization directives (works well with Intel Xeon Phi).

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)

https://www.khronos.org/opencl/

- Initial version of OpenCL standard released in Dec 2008.
- Many companies have created their own implementations.
- Currently on version 2.2 (released May 12 2017).

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);
}
```

OpenACC

- Open standard for expressing accelerator parallelism.
- Fortran and C.
- Similar to OpenMP in structure: uses directives.
- If the compiler doesn't understand the directives, it ignores them, so the same code can work:
  - **with** accelerators or **without** accelerators;
  - with compilers that **do** understand OpenACC and with compilers that **don't** understand OpenACC.
- The OpenACC directives tell the compiler which parts of the code happen in the accelerator (and some details about how to run them in the accelerator).
  - The parts of the code that **don't** have OpenACC directives happen in the regular server CPU/RAM hardware.

OpenACC Compiler Directives

- Developed by NVIDIA, Cray, PGI, CAPS
- Available in PGI compilers for general cluster user, and in Cray compilers for use on Crays.
- Also some less commonly used and experimental compilers.

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.x Accelerator Directives

- OpenMP's 4.5 standard was released in 2015.
- It 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 V100 Card Specs

- 5120 GPU cores
- 1455 GHz turbo clock speed
- Double precision (FP64) floating point performance:
  7 TFLOPs (32 double precision flop per cycle per core)
- Single precision (FP32) floating point performance:
  14 TFLOPs (64 single precision flops per cycle per core)
- Half precision (FP16) floating point performance:
  112 TFLOPs (512 half precision flops per cycle per core)
- Internal RAM: 16 GB, 900 GB/sec (vs ~200 GB/sec for regular RAM)
- Has to be plugged into a PCIe slot (32 GB/sec per GPU card)

http://www.nvidia.com/content/PDF/Volta-Datasheet.pdf
https://en.wikipedia.org/wiki/Nvidia_Tesla

Compare Top x86 vs NVIDIA V100

Let's compare the best dual socket x86 server today vs V100.

	Dual socket, Intel 8180 28-core	**NVIDIA Tesla V100 dual cards in an x86 server**
Peak DP FLOPs	3.0464 TFLOPs DP	14 TFLOPs DP (4.6x)
Peak SP FLOPs	6.0928 TFLOPs SP	28 TFLOPs SP (4.6x)
Peak HP FLOPs	N/A	224 TFLOPs HP
Peak RAM BW	~200 GB/sec	~1800 GB/sec (9x)
Peak PCIe BW	N/A	32 GB/sec
Needs x86 server to be part of?	Is x86 server	Yes
Power/Heat	~400 W	2 × 250 W + ~400 W (~2.25x)
Code portable?	Yes	Yes (OpenACC, OpenCL)

Compare Top x86 vs NVIDIA V100

Here are some interesting measures:

	Dual socket, Intel 8180 28-core	**NVIDIA Tesla V100 dual cards in an x86 server**
DP GFLOPs/Watt	~7.6 GFLOPs/Watt	~15.6 GFLOPs/Watt (~2x)
SP GFLOPS/Watt	~15.2 GFLOPs/Watt	~31.1 GFLOPs/Watt (~2x)
DP TFLOPs/sq ft	~19.3 TFLOPs/sq ft	~44.3 TFLOPs/sq ft (2.3x)
SP TFLOPs/sq ft	~38.6 TFLOPs/sq ft	~88.6 TFLOPs/sq ft (2.3x)
Racks per PFLOP DP	9 racks/PFLOP DP	4 racks/PFLOP DP (44%)
Racks per PFLOP SP	5 racks/PFLOP SP	2 racks/PFLOP SP (40%)

What Are the Downsides?

- You have to rewrite your code into CUDA or OpenCL or PGI accelerator directives (or maybe OpenMP).
  - CUDA: Proprietary
  - OpenCL: portable but cumbersome
  - OpenACC, OpenMP 4.x: portable, but which to choose?

Programming for Performance

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

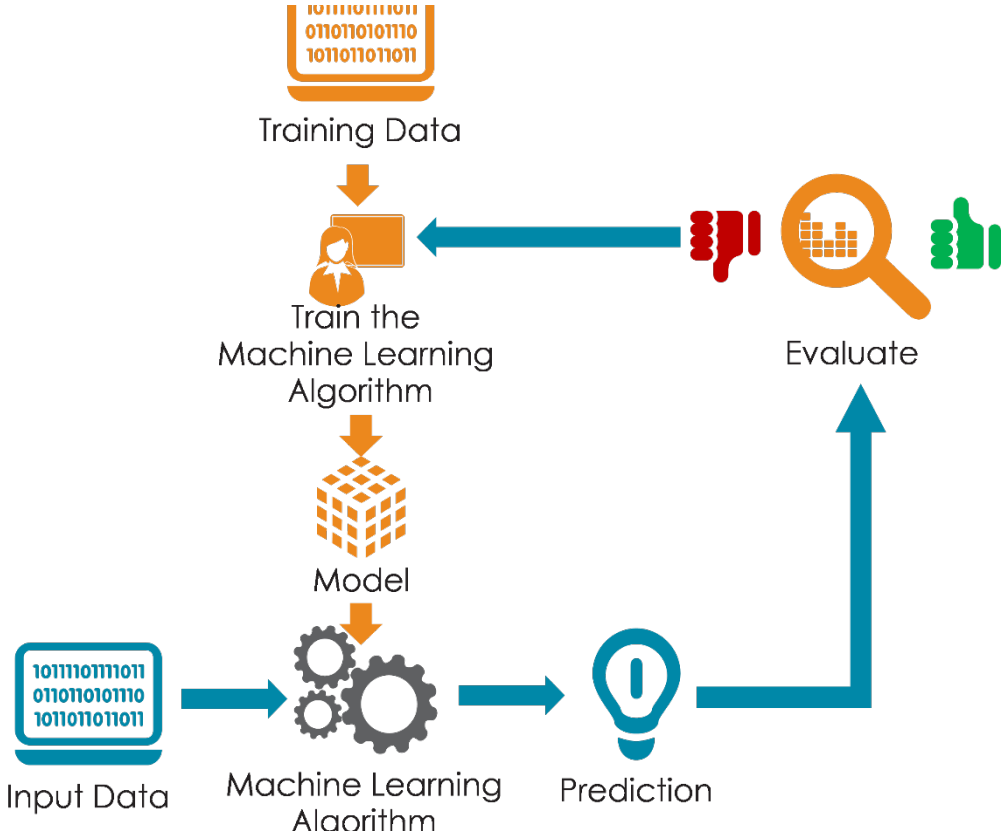
- PCI 3.0 x16: 16 GB/sec
- 2666 MHz current architectures: up to ~200 GB/sec per server
- Accelerator card RAM: 900 GB/sec per card

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.

NOTE: This **DOESN'T** apply to current Intel MIC.

Machine Learning on Accelerators


Training Data
Train the
Machine Learning
Algorithm
Evaluate
Model
Input Data
Machine Learning
Algorithm
Prediction

What is Machine Learning?

- Machine Learning is the part of Artificial Intelligence (AI) that involves a computer "learning" how to produce a good output, or action, given a set of inputs.
  - How do humans react when a basketball is heading toward us?
  - How about a truck?
  - How do we decide which gallon of milk to buy?
  - Or where to get the best haircut?
- For Machine Learning to be appropriate, there must exist three conditions:
  1. There must exist a large data set of inputs and outputs.
  2. There must be some sort of pattern in the data set.
  3. It must be difficult for humans to discern a mathematical pattern to the data.
- If the above three conditions are not met, applying Machine Learning through structured inference learning is futile.

Machine Learning Components

- Components of Machine Learning include:
  - Data for the Machine Learning algorithm
  - Output (or Decision)
  - Structured learning component, which is performed by the Machine Learning algorithm to understand the pattern of the Data input to produce Output (or Decision).
  - The expression that the Machine Learning formulates is called the Mapping Function.
  - The Mapping Function is used to learn the Target Function, which is the ultimate output of Machine Learning, and presumably is not a function that can be solved perfectly by mathematics.

What Does Machine Learning Give Us?

- Machine Learning ultimately provides us an **estimate** of a **predictive model** that best generalizes to particular type of data.
- The Data used to teach the model is critical. The larger and cleaner the dataset is, the better the estimate of a predictive model will be.
- Because the requirements for **effective** use of Machine Learning, it is not an analysis methodology that is in any way universal.

Machine Learning and GPGPUs

- Many Machine Learning algorithms do calculations on fast GPGPUs, using half-precision arithmetic.
- Half Precision (less precise) arithmetic can be use because:
  - learning data sets should be (and are) large;
  - precision is not nearly as important as the data having a trend, or pattern;
  - a few inaccurate variables in the learning data should be overwhelmed by data that **is** accurate.
- Using Half Precision data speeds up GPGPUs communications, and therefore the total processing time.
- It's not unusual to find many 4x4 matrix operations in ML code – so it's easy to map ML code onto rendering hardware.

Why 4x4 matrices?

- GPUs were designed to solve 4x4 matrices very fast and in parallel, to serve their original purpose of doing very fast 3D graphics.
  - Using 4x4 matrices, you can describe ***rotation***, ***translation*** (moving things around) and ***scaling*** (making things bigger or smaller) at the same time, vital for calculating the "next color" of a pixel.
- Programmers writing the linear algebra part of their ML system are rewarded with increased performance if they can translate it to as much 4x4 matrix algebra as possible.
- The other parts of ML systems generally involve statistical and probability calculations, which are not nearly as quick on GPUs.

TENTATIVE Schedule

Tue Jan 23: Storage: What the Heck is Supercomputing?
Tue Jan 30: The Tyranny of the Storage Hierarchy Part I
Tue Feb 6: The Tyranny of the Storage Hierarchy Part II
Tue Feb 13: Instruction Level Parallelism
Tue Feb 20: Stupid Compiler Tricks
Tue Feb 27: Multicore Multithreading
Tue March 6: Distributed Multiprocessing
Tue March 13: **NO SESSION** (Henry business travel)
Tue March 20: **NO SESSION** (OU's Spring Break)
Tue March 27: Applications and Types of Parallelism
Tue Apr 3: Multicore Madness
Tue Apr 10: **NO SESSION** (Henry business travel)
Tue Apr 17: High Throughput Computing
Tue Apr 24: GPU: Number Crunching in Your Graphics Card
Tue May 1: Grab Bag: Scientific Libraries, I/O Libraries, Visualization

Thanks for helping!

- OU IT
  - OSCER operations staff (Dave Akin, Patrick Calhoun, Kali McLennan, Jason Speckman, Brett Zimmerman)
  - OSCER Research Computing Facilitators (Jim Ferguson, Horst Severini)
  - Debi Gentis, OSCER Coordinator
  - Kyle Dudgeon, OSCER Manager of Operations
  - Ashish Pai, Managing Director for Research IT Services
  - The OU IT network team
  - OU CIO Eddie Huebsch
- OneNet: Skyler Donahue
- Oklahoma State U: Dana Brunson

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 phone bridge to fall back on.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

Coming in 2018!

- Linux Clusters Institute workshops

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

  - Introductory HPC Cluster System Administration: May 14-18 2018 @ U Nebraska, Lincoln NE USA
  - Intermediate HPC Cluster System Administration: Aug 13-17 2018 @ Yale U, New Haven CT USA
- Great Plains Network Annual Meeting: details coming soon
- Advanced Cyberinfrastructure Research & Education Facilitators (ACI-REF) Virtual Residency Aug 5-10 2018, U Oklahoma, Norman OK USA
- PEARC 2018, July 22-27, Pittsburgh PA USA

  https://www.pearc18.pearc.org/
- IEEE Cluster 2018, Sep 10-13, Belfast UK

  https://cluster2018.github.io
- **OKLAHOMA SUPERCOMPUTING SYMPOSIUM 2018, Sep 25-26 2018 @ OU**
- SC18 supercomputing conference, Nov 11-16 2018, Dallas TX USA

  http://sc18.supercomputing.org/

Thanks for your attention!

Questions?

www.oscer.ou.edu

Does CUDA Help?

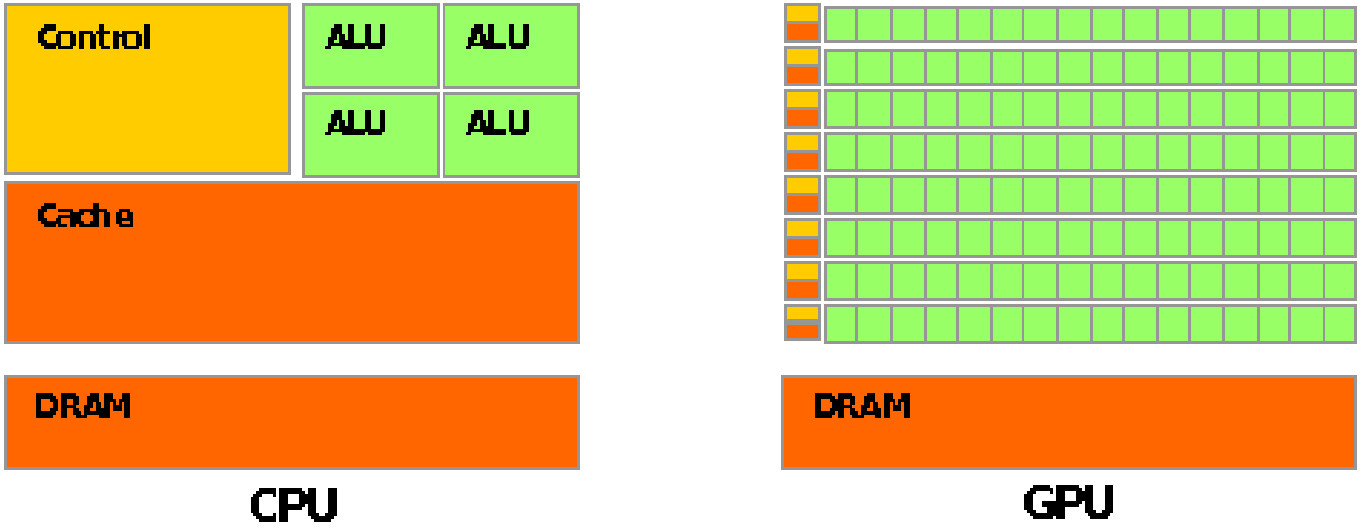
Example Applications	URL	Speedup
Seismic Database	http://www.headwave.com	66x – 100x
Mobile Phone Antenna Simulation	http://www.accelware.com	45x
Molecular Dynamics	http://www.ks.uiuc.edu/Research/vmd	21x – 100x
Neuron Simulation	http://www.evolvedmachines.com	100x
MRI Processing	http://bic-test.beckman.uiuc.edu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

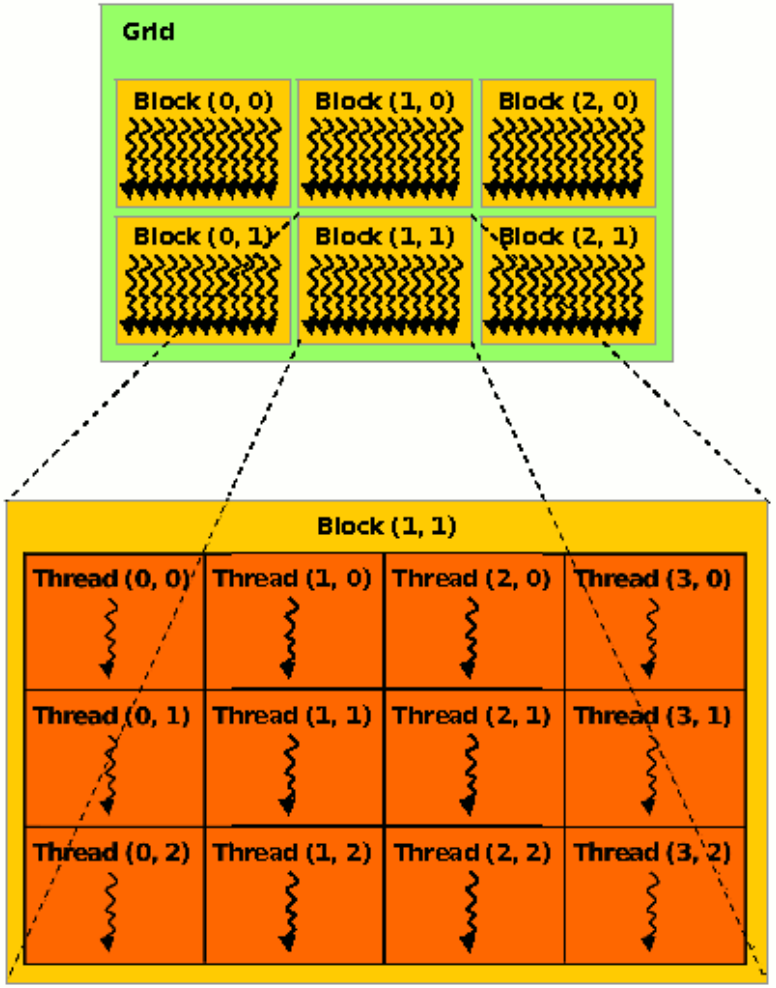


Image Source:
NVIDIA CUDA Programming Guide

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

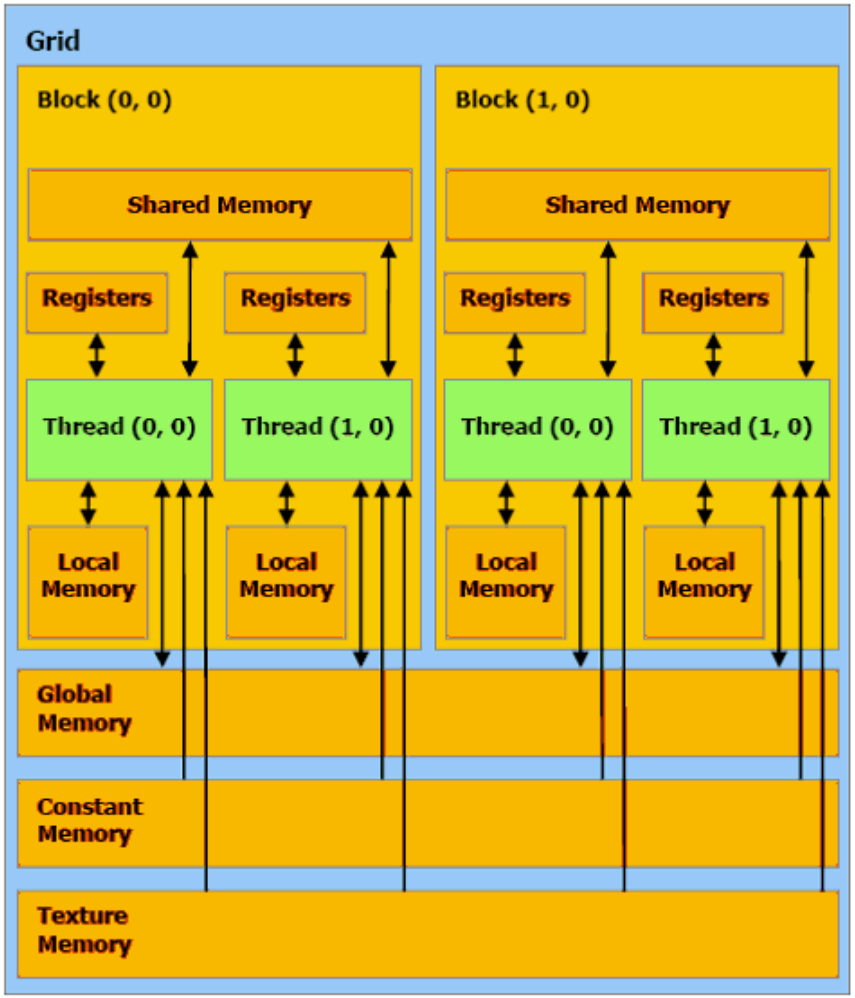
If data need to move from the host (where presumably the data are initially input or generated), then a copy has to exist in both places.

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:

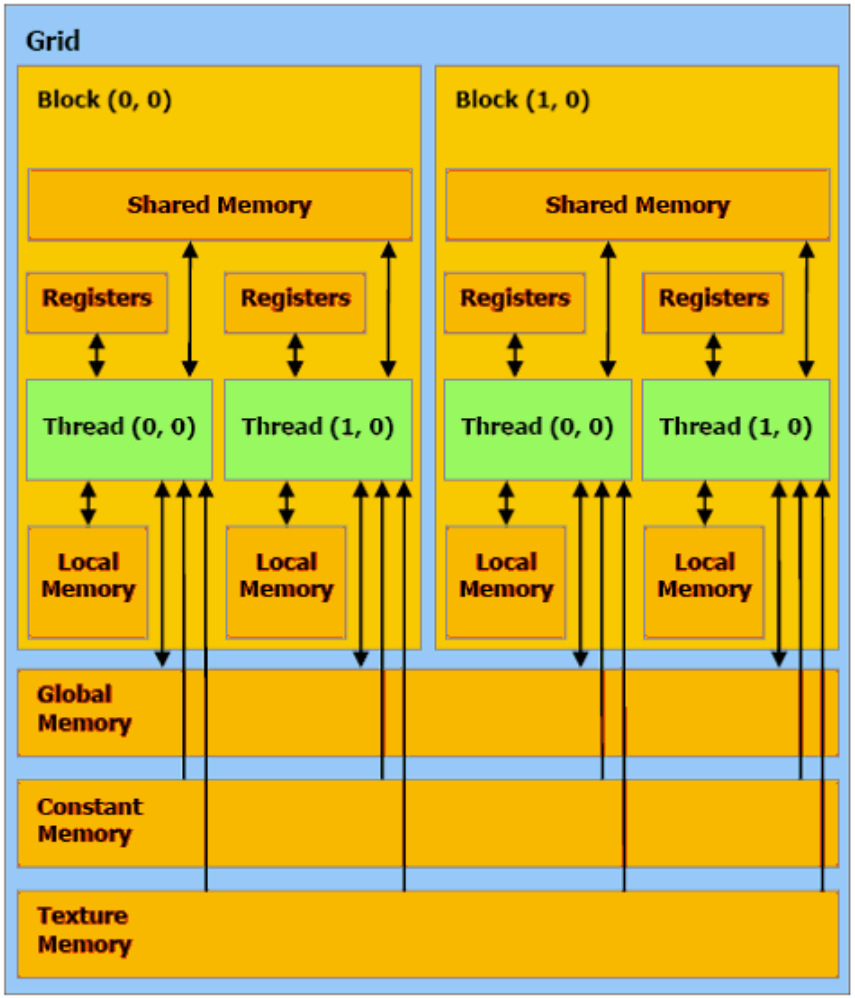
- Host memory (x86 server)
- Device memory (GPU)
  - ***Global***: visible to all threads in all blocks – largest, slowest
  - ***Shared***: visible to all threads in a particular block – medium size, medium speed
  - ***Local***: visible only to a particular thread – smallest, fastest



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

TENTATIVE Schedule

Tue Jan 23: Storage: What the Heck is Supercomputing?
Tue Jan 30: The Tyranny of the Storage Hierarchy Part I
Tue Feb 6: The Tyranny of the Storage Hierarchy Part II
Tue Feb 13: Instruction Level Parallelism
Tue Feb 20: Stupid Compiler Tricks
Tue Feb 27: Multicore Multithreading
Tue March 6: Distributed Multiprocessing
Tue March 13: **NO SESSION** (Henry business travel)
Tue March 20: **NO SESSION** (OU's Spring Break)
Tue March 27: Applications and Types of Parallelism
Tue Apr 3: Multicore Madness
Tue Apr 10: **NO SESSION** (Henry business travel)
Tue Apr 17: High Throughput Computing
Tue Apr 24: GPU: Number Crunching in Your Graphics Card
Tue May 1: Grab Bag: Scientific Libraries, I/O Libraries, Visualization

Thanks for helping!

- OU IT
  - OSCER operations staff (Dave Akin, Patrick Calhoun, Kali McLennan, Jason Speckman, Brett Zimmerman)
  - OSCER Research Computing Facilitators (Jim Ferguson, Horst Severini)
  - Debi Gentis, OSCER Coordinator
  - Kyle Dudgeon, OSCER Manager of Operations
  - Ashish Pai, Managing Director for Research IT Services
  - The OU IT network team
  - OU CIO Eddie Huebsch
- OneNet: Skyler Donahue
- Oklahoma State U: Dana Brunson

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 phone bridge to fall back on.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

PLEASE MUTE YOURSELF.

Coming in 2018!

- Linux Clusters Institute workshops

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

  - Introductory HPC Cluster System Administration: May 14-18 2018 @ U Nebraska, Lincoln NE USA
  - Intermediate HPC Cluster System Administration: Aug 13-17 2018 @ Yale U, New Haven CT USA
- Great Plains Network Annual Meeting: details coming soon
- Advanced Cyberinfrastructure Research & Education Facilitators (ACI-REF) Virtual Residency Aug 5-10 2018, U Oklahoma, Norman OK USA
- PEARC 2018, July 22-27, Pittsburgh PA USA

  https://www.pearc18.pearc.org/
- IEEE Cluster 2018, Sep 10-13, Belfast UK

  https://cluster2018.github.io
- **OKLAHOMA SUPERCOMPUTING SYMPOSIUM 2018, Sep 25-26 2018 @ OU**
- SC18 supercomputing conference, Nov 11-16 2018, Dallas TX USA

  http://sc18.supercomputing.org/

Thanks for your attention!

Questions?

www.oscer.ou.edu