Summary of previous lectures

- Pthreads: low-level multi-threaded programming
- OpenMP: simplified interface based on `#pragma`, adapted to scientific computing
- OpenMP for and scheduling (static, dynamic)
- Reduction and atomic
- Data layout and vectorization (SoA vs AoS)
- Arithmetic intensity and roofline model
- NUMA and first-touch allocation
GPU computing!
Performance numbers (Karl Rupp)

40 Years of Microprocessor Trend Data

- Transistors (thousands)
- Single-Thread Performance (SpecINT x 10^3)
- Frequency (MHz)
- Typical Power (Watts)
- Number of Logical Cores

Year:

Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten.
New plot and data collected for 2010-2015 by K. Rupp.

https://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/
https://www.karlrupp.net/2015/06/40-years-of-microprocessor-trend-data/
Key to performance: lots of cores!
Peak performance: single precision

Theoretical Peak Performance, Single Precision

INTEL Xeon CPUs
NVIDIA GeForce GPUs
AMD Radeon GPUs
INTEL Xeon Phis
Memory is key for many applications

Theoretical Peak Memory Bandwidth Comparison

Memory is key for many applications.
To build supercomputers, perf/watt is the relevant metric

Theoretical Peak Floating Point Operations per Watt, Single Precision

- INTEL Xeon CPUs
- NVIDIA GeForce GPUs
- AMD Radeon GPUs
- INTEL Xeon Phis
Arithmetic intensity: are you memory or compute bound?

![Theoretical Peak Floating Point Operations per Byte, Double Precision](image_url)
What is a major issue in building a supercomputer?

- Amount of memory
- Processing speed
- Network speed

Start the presentation to activate live content.
Performance for scientific applications

- Example: Tesla P100
- 5.3 teraflops double-precision performance
- 10.6 teraflops single-precision performance
- 732 GB/sec memory bandwidth
Performance of P100

- Deep Learning
- Quantum Mechanics
- Particle Weather
- Quantum ChromoDynamics
- Molecular Dynamics
- Cosmology
**Alphago**

1,202 CPUs and 176 GPUs

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Search threads</th>
<th>No. of CPU</th>
<th>No. of GPU</th>
<th>Elo rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>40</td>
<td>48</td>
<td>2</td>
<td>2,738</td>
</tr>
<tr>
<td>Single</td>
<td>40</td>
<td>48</td>
<td>4</td>
<td>2,850</td>
</tr>
<tr>
<td>Single</td>
<td>40</td>
<td>48</td>
<td>8</td>
<td>2,890</td>
</tr>
<tr>
<td>Distributed</td>
<td>12</td>
<td>428</td>
<td>64</td>
<td>2,937</td>
</tr>
<tr>
<td>Distributed</td>
<td>24</td>
<td>764</td>
<td>112</td>
<td>3,079</td>
</tr>
<tr>
<td>Distributed</td>
<td>40</td>
<td>1,202</td>
<td>176</td>
<td>3,140</td>
</tr>
<tr>
<td>Distributed</td>
<td>64</td>
<td>1,920</td>
<td>280</td>
<td>3,168</td>
</tr>
</tbody>
</table>
What is the technology behind GPU processors?
It’s started with 3D graphics
Parallelism

- For graphics rendering, GPUs can reach amazing performance.
- Characteristics of calculations:
  - Simple arithmetic operations at each pixel
  - Large amounts of parallelism
  - Same operation needs to be performed on many different elements
- Graphics processing sounds a lot like scientific computing!
Multicore processors

• The mean giant.
• A fast processor because:
  • Pipelining of execution
  • Out of order execution
  • Branch prediction
  • Pre-fetching
  • Large amount of multilevel cache
  • Hyperthreading
  • …
• Fast and general; does not assume anything about the computation.
Streaming processors, e.g., GPUs

- The ants model
- Many computing units operating in parallel
- Ideal for simple but repetitive tasks.
- Example:
  - Dense linear algebra
  - Finite-difference
  - Finite-element
  - Neural network
  - Large data set processing
- Bad when calculation involves significant branching, e.g., each thread is doing something different
How is this possible?

• There has to be a trade-off.

• More computing units means you need to give up something:
  • No processor space is dedicated to complex optimizations, e.g., out-of-order execution
  • Cache space is limited because it has to be shared amongst the threads
  • Light threads: hardware supports the ability to switch threads every cycle; this allows a large number of threads in-flight or live
  • Limited logic for program control: 32 threads are grouped into warps; all threads in warp execute the same instruction at the same time.
  • Each computing unit is less powerful but there are more of them.
Modern processors

Intel Xeon

Intel Xeon Phi

Specialization

Performance

NVIDIA Tesla
Short History of NVIDIA GPU architectures

- G70: 2006
- Tesla: 2006
- Fermi: 2010
- Kepler: 2012
- Maxwell: 2014
- Pascal: 2016
Certainty

What we have:

• Fermi M2050; 4 GPUs/node
• Release date: July 2011
• Clock rate: 1.147 GHz
• 14 multiprocessors
• 448 cores
• Total global memory: 2.8 GB
• Bandwidth: 148 GB/sec
• Peak performance: single 1.03 Tflop/sec; double 515 Gflop/sec
• Compute capability: 2.0
Host and coprocessor

• Xeon Phi accelerators support a full-fledged operating system like Linux.
• NVIDIA GPUs are different. They only work as a co-processor.
• This means you need a host processor (e.g., Intel Xeon).
• Your program runs on the host and uses the CUDA API to move data back and forth to the GPU and run programs on the GPU.
• You cannot log on the GPU directly or run an OS on the GPU.
Allocate memory on GPU
Copy data
Launch kernel
Copy data
Free memory on GPU

Allocation
Host to Device
Kernel execution
Device to Host

Asynchronous
GPU architecture

You cannot program a GPU without understanding the architecture of the processor.
1 instruction buffer  
+ warp scheduler 
Large register file 

32 cores 

Maxwell architecture  
Maxwell Streaming Multiprocessor (SM) 

Shared memory
Global view

Common L2 cache

No. of cores: \((8 \times 4 \times 2) \times 8 \times 4\)  
= 2,048 cores!
Basic breakdown

To program GPUs you need to understand the basic breakdown, at least as a programming model:

• Bottom-most level: 32 cores = 32 threads, grouped in a warp. The hardware is optimized to have all threads in a warp execute the same instruction at the same time. SIMT (single instruction multiple threads) model.

• Groups of warps form a block. A block executes on 1 SM (streaming multiprocessor). Threads within a block have some ability to exchange data and synchronize.

• All the blocks constitute the “entire dataset” that will be operated on by a kernel.
What's a CUDA thread block?

A set of threads running on a multiprocessor

32 threads

A set of warps grouped together
What does SIMT mean?

Full efficiency is reached when 32 threads execute the same instruction.

All threads execute the same instruction.

Threads can execute vector instructions efficiently.

Start the presentation to activate live content.
If you see this message in presentation mode, install the add-in or get help at PollEv.com/app.
Execution model

• Calculations on a GPU always follow the same model.
• This is very constrained.
• GPUs run kernels that are designed to execute calculations in the following way.
Code example

• Let’s start with a very basic example
  firstProgram.cu

.bashrc:

# User specific aliases and functions
module load null
module load gnu/4.9.2
module load cuda/7.5.18
Code overview

```c
int* d_output;
cudaMalloc(&d_output, sizeof(int) * N);
kernel<<<1, N>>>(d_output);
vector<int> h_output(N);
cudaMemcpy(&h_output[0], d_output, sizeof(int) * N, cudaMemcpyDeviceToHost);
for(int i = 0; i < N; ++i) {
    printf("Entry %3d, written by thread %2d\n", 
            h_output[i], i);
}
cudaFree(d_output);
```
Function to execute

Input variables

kernel<<<1, N>>>(d_output);

Number of threads to use
Device kernels

```c
__device__ __host__
int f(int i) {
    return i*i;
}

__global__
void kernel(int* out) {
    out[threadIdx.x] = f(threadIdx.x);
}
```
global host device

What are these mysterious keywords?

__global__ kernel will be
• Executed on the device
• Callable from the host

__host__ kernel will be
• Executed on the host
• Callable from the host

__device__ kernel will be
• Executed on the device
• Callable from the device only

Compiled for host

Compiled for device
We will see on Wednesday how to use more threads and run larger cases.