GPU Cluster Computing

Advanced Computing Center for Research and Education
What is GPU Computing?

- Graphics processing units (GPUs) are used for rendering graphics on a display (e.g. laptop monitor).
- The gaming and high-definition graphics industries drive the development of fast graphics processing.
- Recently, programming GPUs has been made generic enough to work with scientific computing problems.
- Due to their massively parallel architecture, GPUs enable the completion of computationally intensive tasks much faster than conventional CPUs.
- Hybrid CPU-GPU system:
  - GPU: Computationally-intensive (massively parallel) part.
  - CPU: sequential part.
GPU Performance: Computing

Image taken from CUDA Programming Guide
GPU Performance: Memory Bandwidth

Image taken from CUDA Programming Guide
Quiz Time!

Which of the following make up a CPU?
(a) Flow unit
(b) Compute unit
(c) Control unit
(d) Math unit
(e) Arithmetic logic unit
(f) G unit
(g) Cache
(h) Registers
Quiz Time!

Which of the following make up a CPU?

(a) Flow unit
(b) Compute unit
(c) Control unit — brain of the operation; decides which instructions need to be executed; fetches data from main memory
(d) Math unit
(e) Arithmetic logic unit — operates on data
(f) G unit
(g) Cache — small, fast bank of memory
(h) Registers — smaller, faster bank of memory
Why is the GPU So Fast?

- The GPU is specialized for compute-intensive, highly data parallel computation (owing to its graphics rendering origin).
- More transistors can be devoted to data processing rather than data caching and control flow.
- Where GPUs excel: high arithmetic intensity (the ratio between arithmetic operations and memory operations), matrix and vector operations (remember, your graphics monitor is just a matrix of pixels!).
How are GPUs Different than CPUs?

“CPUs are designed to handle complexity well, while GPUs are designed to handle concurrency well.” - Axel Kohlmeyer
- Modern multicore processors
- Handful of CPUs each supporting ~1-2 hardware threads
- On-chip memory near processors (registers, cache)
- Shared global memory space (external DRAM)
- Programmable GPUs
- Several processors each supporting numerous hardware threads
- On-chip memory near processors (registers, cache)
- Shared global memory space (external DRAM)
• Typically the GPU and CPU coexist in a heterogeneous setting.
• Less computationally intensive part runs on CPU (coarse-grained parallelism), and more intensive parts run on GPU (fine-grained parallelism).
• GPU is typically a computer card, installed into a PCI Express slot.
• Market leaders: NVIDIA, AMD (ATI)
  • Example NVIDIA GPUs
  
GeForce GTX 480

Tesla 2070
**NVIDIA GPU**

**SM**
- Streaming multiprocessor
- 32xSP (or 16, 48 or more)
- Fast local ‘shared memory’ (shared between SPs)
  - 16 KiB (or 64 KiB)

**SP**
- Scalar processor ‘CUDA core’
- Executes one thread

**GLOBAL MEMORY**
- On device
NVIDIA GPU

GPU:

• SMs
  • 30 SM on GT200,
  • 15 SM on Fermi

• For example, GTX 480:
  • 15 SMs x 32 cores
    = 480 cores on a GPU

GDDR memory
512 MiB - 6 GiB

GLOBAL MEMORY
(ON DEVICE)

SM

SHARED MEMORY
The **GPU** is viewed as a compute device that:

- Is a co-processor to the **CPU** or **host**.
- Has its own DRAM (device memory, or global memory in CUDA parlance).
- Runs many threads in parallel.

Data-parallel portions of an application run on the device as kernels which are executed in parallel by many threads.

Differences between GPU and CPU threads:

- GPU threads are extremely lightweight: very little creation overhead.
- GPU needs 1000s of threads for full efficiency.
  - Multi-core CPU needs fewer heavy threads.
A master process running on the CPU performs the following steps:

- Initialize card
- Allocates memory in host and on device
- Copies data from host to device memory
- Launches multiple blocks of execution “kernel” on device
- Copies data from device memory to host
- Repeats 3-5 as needed
- Deallocates all memory and terminates
Execution Model

Serial Code

Kernel A

Serial Code

Kernel B

Host

Device

Host

Device
Software view

- Threads launched for a parallel section are partitioned into thread blocks.
- Grid = all blocks for a given launch
- Thread block is a group of threads that can:
  - Synchronize their execution
  - Communicate via shared memory
Whole Grid Runs on GPU

Many blocks of threads

Global Memory
• Each block of the execution kernel executes on a SM (streaming multiprocessor).
• If the number of blocks exceeds the number of SMs, then more than one will run at a time on each SM if there are enough registers and shared memory, and the others will wait in a queue and execute later.
• All threads within one block can access local shared memory but can’t see what the other blocks are doing (even if they are on the same SM).
• There are no guarantees on the order in which the blocks execute.
void function(...) {
    // (1) Allocate memory on the GPU
    // (2) Transfer input data to the GPU
    // (3) Launch kernel on the GPU
    // (4) Transfer output data to CPU
}

__global__ void kernel(...) {
    // Code executed in parallel by
    // GPU here...
}
ACCRE GPU Nodes

- 48 compute nodes with:
  - Two quad-core 2.4 GHz Intel Westmere processors, 48 GB of memory
  - 4 Nvidia GTX 480
  - 10 Gigabit Ethernet
- Nvidia GTX 480
  - 15 Streaming Multiprocessors per GPU
  - 480 Compute cores per GPU
  - Peak performance: 1.35 TFLOPS SP / 168 GFLOPS DP
  - Memory: 1.5 GB
  - Memory Bandwidth: 177.4 GB/sec
Software Stack

- Operating system:
  - CentOS 6.5
- Batch scheduler system:
  - SLURM
- All compilers and tools are available on the GPU gateway (vmps81)
  - type: “ssh vunetid@vmps81.accre.vanderbilt.edu” OR “rsh vmps81” if you’re already logged into the cluster
  - GCC, Intel compiler
  - Compute nodes share the same base OS and libraries
CUDA

• Based on C with some extensions.
• C++ support increasing steadily.
• FORTRAN support provided by PGI compiler.
• Lots of example code and good documentation.
• Large user community on NVIDIA forums.
CUDA Components

- Driver
  - Low-level software that controls the graphics card
- Compute Capability
  - Refers to the GPU architecture, and features supported by hardware
- Toolkit (CUDA version)
  - nvcc CUDA compiler, libraries, profiling/debugging tools
- SDK (Software Development Kit)
  - Lots of demonstration examples
  - Some error-checking utilities
  - Not officially supported by NVIDIA
  - Almost no documentation
CUDA Architecture
Quiz Time!

*Describe what the command mpicc is used for.
CUDA Toolkit

- Installed at `/usr/local/cuda`
- To compile CUDA program:

```bash
export PATH="/usr/local/cuda/bin:$PATH"
export LD_LIBRARY_PATH="/usr/local/cuda/lib64:/usr/local/cuda/lib:$LD_LIBRARY_PATH"
```

- Source files with CUDA language extensions (.cu) must be compiled with `nvcc`.
- Actually, `nvcc` is a compile driver.
  - Works by invoking all the necessary tools and compilers like gcc, cl, ...
CUDA Toolkit

```c
int main(void) {
    printf("Hello World!\n");
    return 0;
}
```

$ nvcc hello_world.c
$ a.out
Hello World!
$

NVIDIA compiler (nvcc) can be used to compile programs with no device code.
CUDA Toolkit

CUDA C/C++ keyword `__global__` indicates a function (i.e. kernel) that:

- Runs on the device
- Called from host (CPU) code

`nvcc` separates source code into host and device components

- Device functions, e.g. `mykernel()`, processed by NVIDIA compiler
- Host functions, e.g. `main()`, processed by standard host compile, e.g. `gcc`, `cl.exe`
CUDA SDK

- Provides hundreds of code samples to help you get started on the path of writing software with CUDA C/C++.
- Code samples cover a wide range of applications and techniques.
- Installed at `/opt/NVIDIA_GPU_Computing_SDK` on the GPU gateway (vmps81).
Job Submission

Do not ssh to a compute node.

Must use SLURM to submit jobs.
  • Either as batch job or interactively.
  • Provides exclusive access to a node and all 4 GPUs.
  • Must include partition and account information.

If you need interactive access to a GPU for development or testing:

```
salloc --time=6:00:00 --partition=gpu --account=sc290_gpu
```

For batch jobs, include these lines:

```
#SBATCH --partition=gpu
#SBATCH --account=sc290_gpu
```
Must be in a GPU group and include the --partition and --account options

Single job has exclusive rights to allocated node (4 GPUs per node)

See example-5 at https://github.com/accire/SLURM.git