GPU cluster computing

Advanced Computing Center for Research and Education
http://www.accre.vanderbilt.edu
What is GPU Computing

• Use of GPU(s) to solve scientific computing problem. Due to their massive-parallel architecture, using GPUs enables the completion of computationally intensive assignments much faster compared with conventional CPUs.

Hybrid CPU-GPU system:
• GPU: Computationally-intensive (massively parallel) part
• CPU: sequential part
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 are GPUs good: high arithmetic intensity (the ratio between arithmetic operations and memory operations)
• Handful of processors each supporting ~1 hardware thread
• On-chip memory near processors (registers, cache)
• Shared global memory space (external DRAM)
Generic Manycore Chip

Multiple processors each supporting many hardware threads
On-chip memory near processors (register, cache, RAM)
Shared global memory space (external DRAM)
Heterogeneous Computing

- Typically 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)
GPUs

- 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 Architecture**

**SM**
- **SP**: scalar processor  
  - ‘CUDA core’  
  -Executes one thread  

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

**GLOBAL MEMORY**  
(ON DEVICE)
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)

SHARED MEMORY

SM

SP

SP

SP

SP

SP

SP

SP

SP

SP

SP

SP
CUDA Programming Model

• 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 only a few heavy ones
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
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 an SM
• 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(...) {
    Allocate memory on the GPU
    Transfer input data to the GPU
    Launch kernel on the GPU
    Transfer output data to CPU
}

__global__ void kernel(...) {
    Code executed on
    the GPU goes here...
}
ACCRE GPU Nodes

- 48 compute nodes with:
  - Two quad-core 2.4 GHz Intel Nehalem, 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 Amount 1.5 GB
  - Memory Bandwidth 177.4 GB/sec
Software Stack

- Operating system:
  - CentOS 6
- Batch system:
  - Torque (PBS)/Moab
- All compilers and tools are available on the GPU gateway (vmps81)
  - GCC, Intel compiler
  - Compute nodes share the same base OS and libraries
CUDA Architecture

• 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 Architecture
CUDA Components

• Driver
  • Low-level software that controls the graphics card

• Toolkit
  • nvcc CUDA compiler
  • Some profiling and debugging tools
  • Various libraries

• SDK
  • Lots of demonstration examples
  • Some error-checking utilities
  • Not officially supported by NVIDIA
  • Almost no documentation
CUDA Toolkit

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

  ```
  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, ...
NVIDIA compiler (nvcc) can be used to compile programs with no device code.
CUDA C/C++ keyword `__global__` indicates a function that:

- Runs on the device
- Is called from host 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`
• Provides hundreds of code samples, white papers, to help you get started on the path of writing software with CUDA C/C++

• Code samples covers a wide range of applications and techniques

• Installed at /opt/NVIDIA_GPU_Computing_SDK on the gpu gateway
Job Submission

• Should not ssh to a compute node

• Must use PBS to submit jobs
  • Either as batch or interactively

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

  `qsub -I -l nodes=1:gpus=1,walltime=01:00:00`

• If you need all 4 GPUs on a single node:

  `qsub -I -l nodes=1:gpus=4,walltime=01:00:00`

• Batch jobs

  `#PBS -l nodes=1:gpus=1`

  `#PBS -l walltime=1:00:00`
Detecting Device ID

- When requesting GPU's from the batch system, the GPUs assigned to your job will be listed in the file pointed to by the `$PBS_GPUFILE` variable

```bash
#!/bin/bash

#PBS -N myGPUJob
#PBS -M xyz@vanderbilt.edu
#PBS -m abe
#PBS -o gpu.log
#PBS -j oe
#PBS -l nodes=1:ppn=1:gpus=1
#PBS -l walltime=00:10:00

cd $PBS_O_WORKDIR

cat $PBS_GPUFILE

gpuNum=`cat $PBS_GPUFILE | sed -e 's/.*-gpu//g' | tr \n ',' | sed 's/.$//'

echo gpuNum = $gpuNum

./myGPUprogram --device=$gpuNum
```
#!/bin/bash

#PBS -N myGPUJob
#PBS -M xyz@vanderbilt.edu
#PBS -m abe
#PBS -o gpu.log
#PBS -j oe
#PBS -l nodes=1:ppn=1:gpus=1
#PBS -l walltime=00:10:00

cd $PBS_O_WORKDIR

cat $PBS_GPUFILE

gpuNum=`cat $PBS_GPUFILE | sed -e 's/.*-gpu//g' | tr '\n' ',' | sed 's/.$//'`

export CUDA_VISIBLE_DEVICES=$gpuNum

./deviceQuery -noprompt | egrep "^Device"
Getting help from ACCRE

• ACCRE website FAQ and Getting Started pages:
  www.accre.vanderbilt.edu/?page_id=35
• ACCRE Help Desk:
  www.accre.vanderbilt.edu/?page_id=369
• Accre-forum mailing list
• ACCRE office hours: open a help desk ticket and mention that you would like to work personally with one of us in the office. The person most able to help you with your specific issue will contact you to schedule a date and time for your visit.