Lecture 8

MPI (IV)

CUDA (I)

1

Today's Content

- MPI (IV)
  - Process Topology
  - Process Group
  - Manager-Worker Model
- Introduction
  - Trends in HPC
  - GPGPU
- CUDA Programming

2

Process Topology

3

Process Topology (1)

- It is sometimes convenient to arrange processes in certain topology to match the underlying algorithm that communicates with its neighbors.
- Example: Passing value in a ring
  - 15_ring_5.c
- More examples to come in future lectures
  - Matrix algorithms → 2-D Cartesian
  - 2-D Laplace equation solver → 2-D Cartesian
  - 3-D Laplace equation solver → 3-D Cartesian

4

```
int MPI_Cart_create (MPI_Comm comm_old, int ndims,
      int *dims, int *periods, int reorder, MPI_Comm *comm_cart);

int MPI_Cart_shift ( MPI_Comm comm, int direction, int displ,
                     int *source, int *dest )

int MPI_Cart_get (MPI_Comm comm, int maxdims, int *dims,
        int *periods, int *coords )

int MPI_Cart_rank (MPI_Comm comm, int *coords, int *rank );

int MPI_Cart_coords (MPI_Comm comm, int rank, int maxdims,
      int *coords )
```

5

Note: Italicized & underlined parameters are outputs!

16_cart.c

0 (0,0)	1 (0,1)	2 (0,2)
3 (1,0)	4 (1,1)	5 (1,2)
6 (2,0)	7 (2,1)	8 (2,2)

6

Process Group

7

Communicators

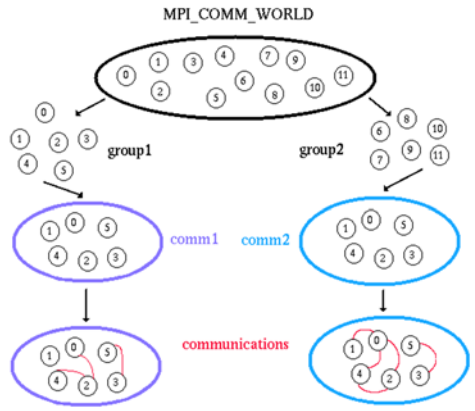
- Communicator: a “handle” or identifier to a group of processes to perform communications
  - Collective communication: communicator
  - Point-to-point communication: communicator + rank
- With communicators
  - Many groups can be created
  - A process may belong to many different groups
- System defined the following two communicators for us after MPI_Init();
  - MPI_COMM_WORLD: all processes the local process can communicate with after initialization (MPI_INIT)
  - MPI_COMM_SELF: the process itself

8

Communicators

- MPI-1: In a static-process-model implementation of MPI, all processes that participate in the computation are available after MPI is initialized. (mpich, lam-mpi)
  - The number of processes (MPI_Comm_size) remains the same from MPI_Init() to MPI_Finalize().
- In MPI-2 with dynamic process mode, processes can dynamically join an MPI execution.
  - In such situations, MPI_COMM_WORLD is a communicator incorporating all processes with which the joining process can immediately communicate.
  - Therefore, MPI_COMM_WORLD may simultaneously have different values in different processes.

9



http://www.llnl.gov/computing/tutorials/mpi/

10

Group communication

- It is sometimes convenient to sub-group MPI_COMM_WORLD to do group communications
- Example: 17_group.c
  - The entire “universe” is divided into two groups, even rank group & odd rank group
  - MPI_COMM_WORLD → group → sub-group → Create communicator from sub-group

11

Six categories in MPI APIs

- Point to point communication
  - MPI_Send, MPI_Recv, …
- Collective communication
  - MPI_Bcast, MPI_Reduce, MPI_Scatter, …
- Process topology
  - MPI_Cart_create, MPI_Cart_shift
- Groups, Contexts, and Communicators
  - MPI_Comm_Group, MPI_Group_incl, MPI_Comm_create, ..
- Environment Inquiry
  - MPI_Get_processor_name, MPI_Get_version, …
- Profiling
  - Not discussed!

12

Manager-Worker Programming Model

Manager-Worker Programming Model

- Originally known as Master-Slave Programming Model
- Manager-worker model can be used in a heterogeneous cluster, and automatically load-balance the available nodes
  - Example: finding n-narcissistic number
  - n-narcissistic number: An n-digit number which is the sum of the nth powers of its digits is called an n-narcissistic number, or sometimes an Armstrong number or perfect digital invariant (Madachy 1979). For example:
    153 = 1^3 + 5^3 + 3^3
    548834 = 5^6 + 4^6 + 8^6 + 8^6 + 3^6 + 4^6
  - 18_manager.cpp

Summary

- Collective communication
- Point-to-point communication
  - Basic point-to-point communication
  - Non-blocking point-to-point communication
- Process Topology
- Process Group
- Manager-Worker Programming Model

15

Trends in High-Performance Computing

16

Trends

- HPC is never a commodity until 1994
- In 1990's
  - Performances of PCs are getting faster and faster
  - Proprietary computer chips offers lower and lower performance/price compared to standard PC chips
- 1994, NASA
  - Thomas Sterling, Donald Becker, … (1994)
  - 16 DX4 (40486) processors, channel bonded Ethernet.

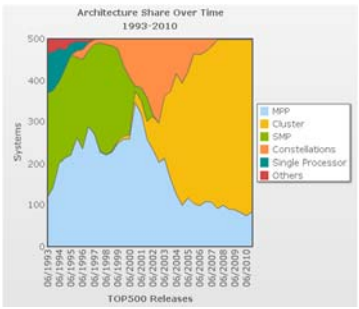
17

Factors contributed to the growth of Beowulf class computers.

- The prevalence of computers for office automation, home computing, games and entertainment now provide system designers with new types of cost-effective components.
- The COTS industry now provides fully assembled subsystems (microprocessors, motherboards, disks and network interface cards).
- Mass market competition has driven the prices down and reliability up for these subsystems.
- The availability of open source software, particularly the Linux operating system, GNU compilers and programming tools and MPI and PVM message passing libraries.
- Programs like the HPCC program have produced many years of experience working with parallel algorithms.
- The recognition that obtaining high performance, even from vendor provided, parallel platforms is hard work and requires researchers to adopt a do-it-yourself attitude.
- An increased reliance on computational science which demands high performance computing.

18

http://www.beowulf.org/overview/history.html



MPP: Massive Parallel Processing

MPP vs. cluster: they are very similar. Multiple processors, each has its private memory, are interconnected through network. MPP tends to have more **sophisticated interconnection** than cluster.

19

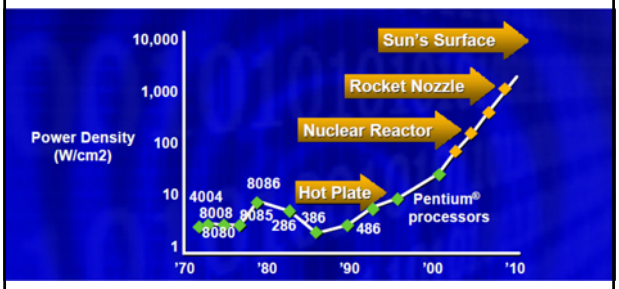
Next 10 years development?

- **Multi-core CPU** is becoming commodity / standard.
  - That's why OpenMP is becoming more and more important ;)
  - Intel has 80-core prototype processors in their lab in 2006…
- Cluster/MPP approach is showing its age with diminishing increase in performance
  - Increasing in clock-speed for general purpose processors (GPP) is slowing down due to power consumption.
- Moore's law suggests number of transistors can be double every 18 – 24 months. These increases can be used better for special-purposed processing.

http://news.cnet.com/Intel-shows-off-80-core-processor/2100-1006_3-6158181.html
http://hpc.pnl.gov/projects/hybrid-computing/

20

Power Density



Source: Patrick Gelsinger, *Intel Developer's Forum*, Intel Corporation, 2004.

Next 10 years of development?

- **Hybrid** computing system
  - Systems that employ more than one type of computing engine to perform application tasks.
  - General purpose processors (e.g. x86) do not necessarily perform well on all tasks. Specialized processors can be built for some special-purpose tasks.
    - GPU → much better at 3D computer graphics
    - FPGA (Field Programmable Gate Arrays) – the processor's hardware can be "reconfigured" or "programmed" to invest its transistors on performing certain tasks.
    - DSP (Digital Signal Processors)

22

http://hpc.pnl.gov/projects/hybrid-computing/

Introduction to GPGPU

23

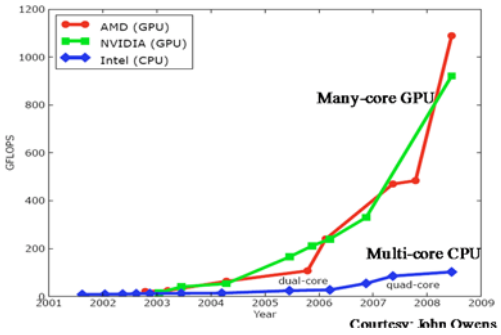
GPGPU

General-**P**urpose computing using **G**raphical **P**rocessing **U**nit

- **GPGPU:** Has been a hot topic in research community since early 2000.
- High-performance (>10x of current generation General Purpose CPUs)
- Highly efficient
  - Cost efficient
  - Space efficient
  - Green / Energy-efficient

24

GPU Performance Growth

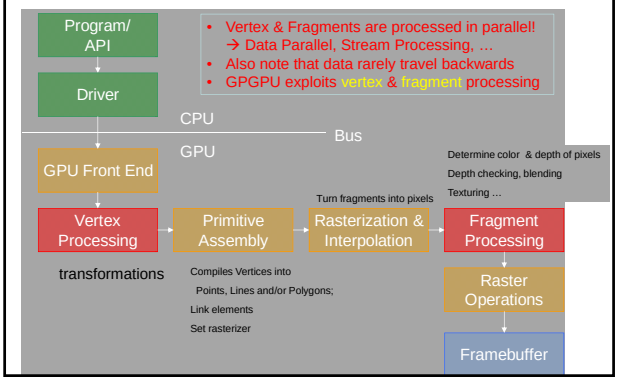


25

Why GPU is faster than CPU ?

- The GPU is ***specialized*** for compute-intensive, highly data parallel computation (graphics rendering)
  - So, more transistors can be devoted to data processing rather than data caching and flow control

GPU Pipeline



Shading Languages

- Cg
  - NVIDIA
- HLSL
  - Microsoft
  - High Level Shader Language
  - DirectX 8+
  - XBox, XBox360
- GLSL
  - OpenGL shading language

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

Barrier into GPGPU

- OpenGL, DirectX ?

- Graphic pipeline ?
- Map between computation versus drawing ?
  - Open a drawing canvas
  - Array → texture
  - Compute kernel → shader
  - Compute → drawing
  - http://www.mathematik.uni-dortmund.de/~goeddeke/gpgpu/tutorial.html#feedback

Efforts for GPGPU

Programming Language Approach

- BrookGPU
  - Around 2003 @ Stanford University
  - Ian Buck, now with NVIDIA
  - Layering over DirectX 9, OpenGL, CTM, Cg
  - Incorporated into the Stream SDK, AMD to replace it CTM
- Sh
  - Conceived @ University of Waterloo, Canada
  - General release around 2005
  - Evolved and commercialized @ RapidMind, Inc.
- **CUDA**
  - Released at 2006 @ NVIDIA
- **OpenCL**
  - 2008 @ Apple WDC, now supported by Nvidia, AMD/ATI
- DirectCompute
  - Microsoft DirectX 10+ on Windows Vista & Windows 7

http://www.gpgpu.org/cgi-bin/blosxom.cgi/High-Level%20Languages/index.html

CUDA

Compute **U**nified **D**evice **A**rchitecture

31

Outline

- CUDA Architecture
- CUDA programming model
- CUDA-C

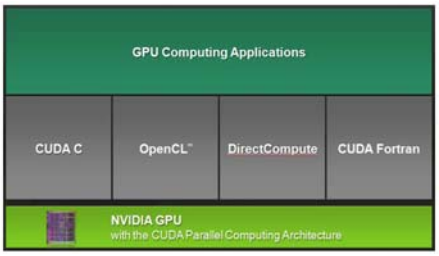
32

CUDA Architecture

33

CUDA™: a General-Purpose Parallel Computing Architecture

- CUDA is designed to support various languages or APIs

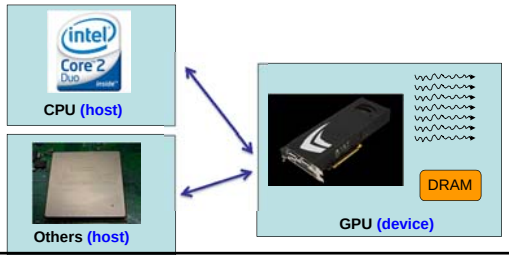


34

NVIDIA_CUDA_Programming_Guide_3.0

CUDA Device and Threads

- Host: usually the general purpose CPU running the computer system
- Host memory: DRAM on the host
- Initiates the execution of kernels
- Device: coprocessor to the CPU/Host
- Device memory: memory on the device
- Runs many threads in parallel

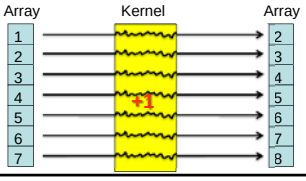


CUDA Programming Model

36

Programming Concepts

- CUDA considers “many-core” processors
- Need **lots and lots of threads** to make it efficient!
- Kernel
  - Kernels are executed by devices through **threads**. One kernel may be executed by many threads.
- Thread
  - Executes a kernel
  - **threadIdx**



Single thread vs. multiple threads

```
Single thread + loop
int a[100]={0};
for(int i=0;i<100;i++) {
    a[i]++;
}
```

38

Single thread vs. multiple threads

```
100 threads, each thread has an
        Id starts from 0
a[id]++;
```

39

Thread Block

- Each thread has its unique ID within a block
- One thread block contains many threads arranged in 1D, 2D, or 3D
  - Process data in linear array, in matrices, or in volumes
  - Threads get their id by threadIdx.x, threadIdx.y, threadIdx.z
- Holds up to 512 threads in current GPU
  - The limit exists because threads in a block are expected to execute on the same processor core
  - Threads in the same block can cooperate
- Threads within a block can cooperate via **shared memory**, **atomic operations** and **barrier synchronization**

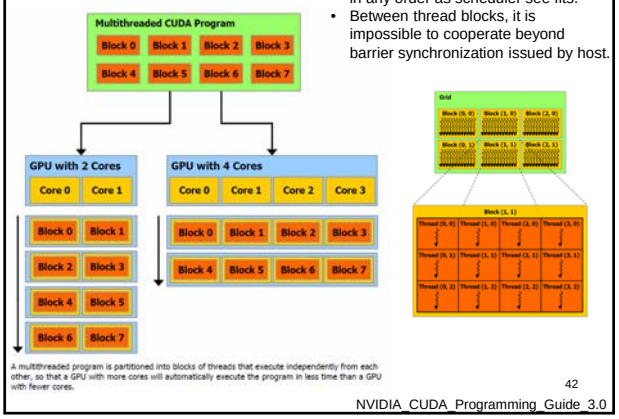
40

Grid

- Thread blocks are arranged into grid (1D or 2D)
- So CUDA programs have grid, each grid point has a thread block, and each thread block contains threads arranged in 1D, 2D, or 3D
- Dimensional Limits:
  - Grid: 65535 x 65536
  - Thread block : 512 x 512 x 64 (the total number must be < 512)
- NO cooperation between thread blocks
- This grid → blocks → threads alllows CUDA programs scale on different hardware.
- Programmers can use **blockIdx, blockDim,** and **threadIdx** to identify their global location/ranking/id to deduce its portion of work (similar to rank in MPI)
  - blockIdx.x, blockIdx.y, blockDim.x, blockDim.y, blockDim.z
  - threadIdx.x, threadIdx.y, threadIdx.z

41

- Thread blocks are executed in any order as scheduler see fits.
- Between thread blocks, it is impossible to cooperate beyond barrier synchronization issued by host.

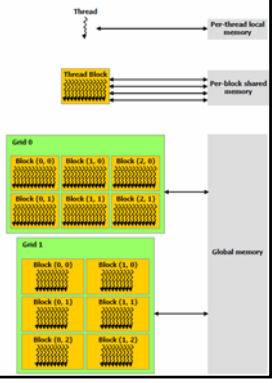


42

NVIDIA_CUDA_Programming_Guide_3.0

Programming Concepts

- Memory Hierarchy
  - Local memory
  - Shared memory
  - Global memory (R/W)
    - Constant memory (R)
    - Texture memory (R)
- Global memories can be accessed by both device & host, but they are optimized for different purposes..
- Need to choose appropriate memory types to use.



Refresh

- Kernel
- Thread
- Thread Block
- Grid
- Memory
  - Local, shared, global / constant / texture

44