Lecture 1: Introduction

Rui Wang
Terms you should know...

- **CPU**: Central Processing Unit
- **GPU**: Graphics Processing Unit
- **GPGPU**: General Purpose computation on the GPU
- **FLOPS**: Floating point Operations Per Second
- **GFLOPS**: Giga-FLOPS
- **CUDA**: Compute Unified Device Architecture
- **HPC**: High Performance Computing
Motivation

- Modern GPUs are massively parallel platform for general-purpose computation.
Motivation

• Modern GPUs are massively parallel platform for general-purpose computation.

• The trend of modern microprocessor design:
  • Increase in clock frequency limited by power consumption issues.
  • Virtually all CPU vendors have switched to multi-core and many-core models.
  • Impacts in software development: sequential vs. data-parallel programming.
Parallel Computing

• **Moore's Law**
  • Transistor count doubles every two years.
  • Increase in transistor count is also a rough measure of computer processing speed.
  • Has been given to everything that changes exponentially.

• **New Moore's Law**
  • Microprocessors no longer get faster, just wider.
The Ox vs. Chicken Analogy

• Seymour Cray: If you were plowing a field, which would you rather use: Two strong oxen or 1024 chickens?
The Ox vs. Chicken Analogy

- Seymour Cray: *If you were plowing a field, which would you rather use: Two strong oxen or 1024 chickens?*

- Chicken is winning these days:
  - For many applications, you can run many cores at lower freq and come ahead at the speed game.
  - For example: decrease freq by 20% → 50% cut in power → can add one more dumb core (chicken) → power budget stays the same but with increased performance!
Parallel Computing is not New

- High-performance computing community has been developing parallel algorithms for decades.

*However,*

- Require exotic, large-scale expensive supercomputers.
- Practice and impact of parallel programming limited.
- Massively parallel machines replaced by clusters of affordable commodity microprocessors
GPUs as Parallel Computers

- GPU = Graphics Processing Unit
  - Video cards, game consoles, mobile phones etc.
  - Two major vendors: NVIDIA and ATI (now AMD)
GPUs as Parallel Computers

- Many-core, High raw computation speed
  - NVIDIA 285 GTX has 240 cores, 1 TFLOPS
  - This compares to modern multi-core CPUs which has 2-8 cores, ~100 GFLOPS
GPU vs. CPU

The diagram compares the performance of NVIDIA GPUs and Intel CPUs from 2003 to 2008. It shows the peak GFLOPs per second (GFLOP/s) for each technology over time. Key models include NV35, NV40, G70, G71, G80, G80 Ultra, G92, and 3.2 GHz Harpertown.
GPUs as Parallel Computers

- Many-core, High raw computation speed
  - NVIDIA 285 GTX has 240 cores, 1 TFLOPS
  - This compares to modern multi-core CPUs which has 2-8 cores, ~100 GFLOPS
- High memory bandwidth
  - More than 100GB/s on GPU vs. 10GB/s on CPU
- Users across science and engineering disciplines are achieving 100x or better speedups on GPUs
GPUs as Parallel Computers

- Why such a large performance gap?

- Fundamental differences in design philosophy.
GPUs as Parallel Computers

- GPU design philosophy:
  - Dedicate transistors to maximize FP calculations
  - Simple memory model to maximize bandwidth
  - In contrast, CPUs dedicate large chip area for control logic and memory hierarchies to support complex applications and maximize sequential code performance.
- GPU design driven by the fast growing video game industry.
GPUs as Parallel Computers

- Performance is not the only determining factor.
  - GPUs are becoming popular as parallel processors because they already have a large market share!
  - Over 100 million CUDA-enabled GPUs sold.
  - Relatively low-cost (~$200-400), small form factor

- First time that massively parallel computing is part of a mass-market product.
GPU vs. CPU

- A visual and funny demonstration about the benefits of massively parallel processing on modern GPUs
- Shown at NVISION 08 by the MythBusters guys
Programming Language Support

- Programs used to be implemented through graphics API → inflexible, and lots of limitations.
- Situation fundamentally changed with the arrival of CUDA → does not go through graphics interface at all.

```c
// Sequential Code
void vecAdd(float *v1, float *v2, float *out, int n) {
    for (int i = 0; i < n; i++) {
        out[i] = v1[i] + v2[i];
    }
    return;
}
```
Programming Language Support

// CUDA Code
__global__ void vecAdd(float *v1, float *v2, float *out, int n)
{
    // get thread index
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i >= n) return;
    out[i] = v1[i] + v2[i];
}

void main()
{
    vecAdd<<< n/256, 256 >>>(v1, v2, out, n);
}
GPU Programming Languages

• **Shading Languages**
  • OpenGL shading language (GLSL)
  • DirectX (HLSL)

• **General Purpose**
  • CUDA
  • Brook+
  • OpenCL (Open Computing Language)
Desktop Supercomputing

- “We’ve all heard ‘desktop supercomputer’ claims in the past, but this time it’s for real: NVIDIA and its partners will be delivering outstanding performance and broad applicability to the mainstream marketplace.”

Burton Smith, Microsoft
Previously Chief Scientist at Cray
GPGPU Applications

- Interactive visualization of volumetric white matter connectivity (146X)
- Ionic placement for molecular dynamics simulation on GPU (36X)
- Transcoding HD video stream to H.264 (19X)
- Fluid mechanics in Matlab using .mex file CUDA function (17X)
- Astrophysics N-body simulation (100X)

- Financial simulation of LIBOR model with swaptions (149X)
- GLAM@lab: an M-script API for GPU linear algebra (47X)
- Ultrasound medical imaging for cancer diagnostics (20X)
- Highly optimized object oriented molecular dynamics (24X)
- Cmatch exact string matching to find similar proteins and gene sequences (30X)
Course Goals

- Learn how to program modern GPUs using CUDA and achieve:
  - High performance
  - Scalability across future generations
Course Goals

• Learn how to program modern GPUs using CUDA and achieve:
  • High performance
  • Scalability across future generations

• Gain both knowledge and hand-on experience:
  • Programming API, tools, and techniques
  • Principles and patterns for parallel computing
  • Processor architecture features and constraints
Course Schedule

• First Half:
  • Graphics hardware
  • CUDA programming API
  • OpenGL shading language
  • GPGPU optimizations.

• Second Half:
  • Principles, patterns, building blocks for parallel computing
  • Student presentations
Logistics

- Time: 2:30–3:45pm TuThu
- Location: LGRT 1322
- Office hours: 4:00–5:00pm TuThu (CS 270)
- Course web page: TWiki
- Course forum: phpBB
- Prerequisites: Familiar with C/C++ programming and data structures
Logistics

- Textbooks: *Recommended, not required*
  - NVIDIA CUDA Programming Guide (download)
  - Patterns for Parallel Programming
  - Introduction to Parallel Computing (2nd edition)
Hardware Resources

- List of CUDA-Enabled Hardware

- Linux server: raster.cs.umass.edu (Ubuntu 8.10, 8800 GTS)
Logistics

- Course Workload and Grading:
  - 3 programming assignments: 15% each
  - 1 midterm exam (in-class): 15%
  - 1 final project: 30%
  - Class presentation: 10%
- Final project will be presented in class.
Final Project

- Pick a nontrivial computational problem of your own choice, implement an efficient, GPU-based solution.
Final Project

• Pick a nontrivial computational problem of your own choice, implement an efficient, GPU-based solution

• *Computer Vision and Graphics*
  SIFT extraction, object recognition, Kalman filter…
  Ray tracing, photon mapping, volume rendering…

• *Numerical and Physical Simulation*
  Fluid dynamics, heat transfer, Monte Carlo methods,
  Astrophysics, electrodynamics, sparse linear algebra…

• *Life Science, Medical Imaging*
  Molecular dynamics, MRI image registration, segmentation,
  Protein folding, string matching…
Policies

• Assignments may be discussed with classmates, but you must implement your own solutions.

• Final project is to be completed by each student independently. You may discuss with other students, but you may not share code with each other.

• To cope with unforeseen circumstances, you are allowed \textbf{five} late days in total during the entire semester.

• No late day is permitted for final project.
Keep in Mind

- First time this course is offered
  - There will be rough edges
  - Concepts that need more explanations
  - Questions that I don't have immediate answers
- Questions/comments/feedbacks are all welcome and encouraged.
Warm-up Assignment

- Setup CUDA programming environment
  - Install CUDA driver, toolkit, SDK (2.0 or 2.1)
  - (Links on the class webpage)
Warm-up Assignment

- Setup CUDA programming environment
  - Install CUDA driver, toolkit, SDK
- Study CUDA SDK examples
- Compile a few examples, fix problem if any
- Use CUDA template for creating your own project
  - CUDA code is compiled with `nvcc`
Warm-up Assignment

• *nvcc* basics
  • Code (.cu, .cpp, etc.) is separated into
    – host (CPU) code
    – device (GPU) code
  • Link with necessary libraries
    – cuda, cudart, cutil, cudpp ...
  • Supports device emulate mode
    – threads are created and emulated on host side
    – very handy for debugging