High-Performance Computing with Commodity Hardware

1. Introduction

Instructor: Xiaoming Li

Goals of the Course

- Why High-performance-computing on “consumer” gadgets?
  - GPU
  - Smart phone
- General program optimization techniques.
  - CPU-oriented techniques
  - Architectural factors
  - Manual and compiler optimizations

Goals of the Course (cont.)

- Commodity hardware architecture
  - Nvidia, Intel, AMD, Qualcomm, IBM Cell
- Programming frameworks
- Architecture-specific optimization techniques

This lecture…

- Administrative stuff
- Overview of class
- Project introduction

Administrative stuff

- Course website
  - UD Canvas LMS http://www1.udel.edu/canvas/
  - Announcements, projects, assignments…
  - Check it regularly.
- Lab
  - Share machines with graphic cards.
  - Programming environments installed.
  - You may use your own machines.

Grading

- Small Projects (45%)
- Course Project (55%)
  - Project proposal and midterm presentation (15%)
  - Final report and presentation (25%)
  - Performance and other evaluation metrics (15%)
- Late Policy
  - Up to 1 hour late, -15%
  - Up to 2 hours late, -40%
  - Up to 3 hours late, -70%
  - Zero grade after 3 hours.
- Grades might be normalized separately for undergraduates and graduates

Course Outline

Course Outline

- High Performance Computing
- Program Optimization: general principles and techniques
- Parallel Programming
- NVIDIA CUDA
- OpenCL
- Programming GPUs
- GPU Optimizations
- Case study

High Performance Computing

- History
  - Trends
  - Successes
  - Failures
- Concepts
  - Cost
  - Scalability

Program Optimization

- Profiling
- Compute/Memory Ratio
- Exploiting Memory Hierarchies
- Effective Loops
- Benchmarking

Parallel Programming

- Parallelism
  - Data Level
  - Instruction Level
  - Task Level
- Flynn's Taxonomy
- Synchronization
- Scalability

Nvidia CUDA Architecture

- Introduced in November 06 – G80, G90, GTX 200,GTX 900, ….
- Characteristics
  - Unified Architecture
  - Massive Multi-processors
  - 500 GFLOPS Theoretical for G80, ~900 GFLOPS for GTX 200, ~5000 for P100
- Exhibits
  - Data Level Parallelism
  - Task Level Parallelism

CUDA Compute Unified Device Architecture

- API for interacting with NVIDIA GPUs
- Abstracts details of the GPU from the programmer.

- Usage
  - Explicit memory management.
  - Computational Kernels
- Special APIs
  - BLAS
  - FFT

Applications

- Game Physics
- Electromagnetics
- Image Processing
- Linear Algebra
- Finance

GPU Program Optimization

- Computation
  - Hierarchies
  - Effective Kernels
- Memory
  - Hierarchies
  - Access Patterns
- Compute vs. Memory Tradeoffs
- Multi-GPU

Other GPUs

- OpenCL supported GPUs:
  - AMD
  - Broadcom
  - Intel
- Older technologies:
  - Cell
  - G70 and earlier

Projects

- Goals
  - Application-specific optimizations
  - Performance analysis and modeling
  - General program transformation for GPU
- Small projects
  - 2-3
  - Work by yourself
- Course project
  - Group of two
  - Individual possible. Need instructor’s approval.
  - Expect roughly equal division of workload

Possible topics

- Applications
  - Eigenvalue solver
  - Blocked LU Decomposition
  - Ray Tracing
  - Discrete Cosine Transform (DCT)
  - Finite Difference (heat, static EM)
  - QR Decomposition
- Evaluation
  - Performance, scalability

Possible topics

- GPU architecture research
  - Micro-benchmarks
    - Measure GPU parameters, such as latency.
  - Performance modeling
    - f(program features) -> performance
    - How to achieve theoretical peak performance on GPU?
- Evaluation
  - Profiling, analysis, and verification

Course Project Workflow

- Form project groups
- Talk to instructors
  - Discuss project topics
  - Set project goals
    - Different requirements for undergraduate groups and graduate groups
- Write proposal
- Midterm presentation
- Presentation of a related paper
- Final presentation and report

First exercise

- Implement the following pseudo code in C:

```
For i from 1 to 4 million
        a[2*i] = a[2*i] * 2
        a[2*i-1] = a[2*i-1] *3
```

- Find three different array access orders that implement the same workload but show *meaningfully* different speeds
  - Task 1: find the right way to measure time.
    - Linux, Windows and MacOS
  - Task 2: find mechanisms that cause similar code performs differently.