About Me

● 3rd year PhD Student
● Research Assistant DataSys Laboratory
● Teaching Assistant, Dept. Computer Science
  ○ CS350, CS351, CS450 (involved w/ CS550, CS554)
● IIT Starr/Fieldhouse Research Fellow
  ○ In collaboration w/ Argonne National Laboratory and Computation Institute (UChicago)
● Guest Graduate Student, Argonne National Lab
● Research
  ○ Many-Task Computing
  ○ Hardware Accelerators
  ○ HPC, HTC, Distributed Systems
Lecture Outline

Lecture: (11:25pm-12:40pm)

● GeMTC (:55)
  ○ Motivation
    ■ Distributed Systems, HPC, MTC GPGPU.
  ○ GeMTC
    ■ Architecture, Design
    ■ Apps
  ○ Future Work
    ■ MTACS
    ■ Xeon Phi

● Swift/T (:20)
  ○ Slides as paper was presented. (CCGrid’13)
Hands on Outline

Hands On: (12:45pm-1:45pm)

- **CUDA**
  - SDK Examples
    - DeviceQuery
  - Vector-Add

- **GeMTC**
  - Vector-Add

- **Swift/T**
  - Vector-Add
Acknowledgements

Thank you to:

Dr. Ioan Raicu - Advisor
Benjamin Grimmer - IIT Undergrad
Dustin Shahidehpour, Jeff Johnson - IIT Alumni (Orbitz, Microsoft)
Dr. Justin Wozniak - ANL Computer Scientist
Michael Wilde - ANL Software Architect & UChicago CI Fellow

Publications

● CCGrid'14 - [Preparing Submission]
● STARR/Fieldhouse Research Fellowship '12-'13
● Scott Krieder, Ioan Raicu - “Early Experiences in running Many-Task Computing workloads on GPUs” - XSEDE 2012 - Chicago, IL (07/2012)
● Scott Krieder, Ioan Raicu - “An Overview of Current and Future Accelerator Architectures” - Greater Chicago Area System Research Workshop - Chicago, IL (05/2012)
Lecture Outline

Lecture: (11:25pm-12:40pm)

- **GeMTC (:45)**
  - **Motivation**
    - Distributed Systems, HPC, MTC GPGPU.
  - **GeMTC**
    - Architecture, Design
    - Apps
  - **Future Work**
    - MTACS
    - Xeon Phi

- **Swift/T (:30)**
  - Slides as paper was presented. (CCGrid’13)
Distributed Systems

- Many machines
- Network
- Common Goal
- Fault Tolerant
- Heterogeneous

*image from Tannebaum - "Distributed Systems*
Supercomputing

Key Characteristics:
- Evolving from homogeneous to increasingly hybrid
- High speed network / Fast
- High Performance Computing (HPC)

Advantages
- "Tightly Coupled"
- Large Resources
- Short Time
  - hours/days/weeks

Growing Shortcomings:
- programmability
- fault tolerance
High Throughput Computing (HTC)

Key Characteristics:
- Loosely coupled
- Robustness
- Reliability
- Jobs per month/year

Advantages:
- Programmability
- Fault tolerance

Shortcomings:
- Efficiency
- Large focus on pleasingly parallel
- Bag-of-tasks pattern
Many-Task Computing (MTC)

MTC emphasizes:

- bridging HPC/HTC
- many resources
  - short period of time
- many computational tasks
- dependent/independent tasks
- tasks organized as DAGs
- primary metrics are seconds

Advantages:

- Improve fault tolerant
- Maintain efficiency
- Programmability & Portability
- support embarrassingly parallel and parallel applications
GPGPU
- Host CPU offload work to GPU
- Relieves CPU
- 52 in June 2012, 62 in November 2012
Accelerator Architecture

GPU
- Streaming Multiprocessors
  (15 SMXs on Kepler K20)
- Warps
  - 32 threads in a warp
  - 192 warps
    i. hardware available
    ii. ind. compute

Coprocessors
- Intel Xeon Phi
  - 60 cores * 4 threads per core
    = 240 hardware threads
GPU Block Diagram - Highlighting SMX
Highlighting SMX and Warps
Concept Overview

- Several works combine GPUs
- Split single GPU into many pieces
  - tiny cluster for compute
- 2 ideas
  - Framework managing the GPU = GeMTC (GPU enabled Many Task Computing)
  - Virtualization = Palacios + GEMTC
Lecture Outline

Lecture: (11:25pm-12:40pm)

● GeMTC (:45)
  ○ Motivation
    ■ Distributed Systems, HPC, MTC GPGPU.
  ○ GeMTC
    ■ Intro
    ■ Swift/T + Apps
    ■ Architecture, Design
    ■ Features: Memory, API
  ○ Future Work
    ■ MTACS
    ■ Xeon Phi

● Swift/T (:30)
  ○ Slides as paper was presented. (CCGrid’13)
Proposed Work

"GEMTC: GPU Enabled Many-Task Computing"

Motivation: No support for Many-Task Computing (MTC) on Accelerators!

Goals:
1) MTC support
2) programmability
3) efficiency
4) MIMD on SIMD
5) Increase concurrency 16 to 192 (12x)

Approach:
Design, implement middleware:
1) manages GPU
2) spread host/device
3) Workflow system support (Swift/T)
How do you program GPUs?

User Runtime

C / C++  Fortran  Swift/T

CUDA/OpenCL/OACC → GPU Code

Operating System / Device Driver

NVIDIA Graphics Processing Unit
Swift/T and Applications

- **Swift/T**
  - Active research project (CI UChicago & ANL)
  - Parallel Programming Framework
  - Throughput ~25k tasks/sec per process
  - Shown to scale to 128k cores

- **Application Domains Supported**
  - Astronomy, Biochemistry, Bioinformatics, Economics, Climate Science, Medical Imaging

**Swift** lets you write parallel scripts that run many copies of ordinary programs concurrently, using statements like this:

```swift
foreach protein in proteinList {
    runBLAST(protein);
}
```

Images from Swift Case Studies - http://www.ci.uchicago.edu/swift/case_studies/
Swift/T Fine Grain, 80 W= 20 Nodes*4PPN

Swift/T Fine Grain Tasks on Kaibab

- Efficiency
- Rate

Sleeptime in Seconds

1, 0.1, 0.01, 0.001
GEMTC Applications

- **Performance Benchmarks**
  - sleep
  - sleep-data-move
  - matrix-multiply
  - vector-add
- **Proxy Applications**
  - MDProxy, Molecular Dynamics
- **Scientific Applications (under development)**
  - OOPS, Protein Folding
  - SciColSim, Collaboration graph analysis
GEMTC Task Flow

- Task submitted
  - description, (taskID, taskType, parameters)
- SuperKernel runs as daemon on device
  - incoming work queue
  - outgoing results queue
  - warp picks up task, executes task, return result
- Host checks results, returns to Swift/T
Warps and Incoming Queue

Mirror image for outgoing results queue
Sub Allocator - Motivation

- Malloc without free, terrible performance
- Malloc + Free $\approx 110$ usec
- Grab all GPU memory at the start
  - Manage memory on our own
- Memory operations
  - gemtcMalloc()
  - gemtcFree()
Sub Allocator - Theory

- `cudaMalloc()` called each task
- Communication times much lower!!
Sub Allocator - Results

- `cudaMalloc() \approx 110\text{ usec}`
- `gemtcMalloc() \approx 14\text{ usec}`
GeMTC API

- C API to interact with workflow systems (Swift/T)
- 8 functions including:
  - **Initialization/Deconstruction**
    - gemtcSetup() // starts our SuperKernel
    - gemtcCleanup() // kills SuperKernel
  - **Enqueue/Dequeue Tasks**
    - gemtcPush() // gemtcPush(1, 1000000)
    - gemtcPoll()
  - **Memory Transfer**
    - gemtcMemcpyHostToDevice()
    - gemtcMemcpyDeviceToHost()
  - **Memory Management**
    - gemtcGPUMAlloc()
    - gemtcGPUFree()
GeMTC Throughput

Efficiency of 84 Workers

Number of Tasks vs. Efficiency for different task durations:
- 16384usec
- 8192usec
- 4096usec
- 2048usec
- 1024usec
- 512usec
- 256usec

The graph shows the efficiency of 84 workers as the number of tasks increases for different task durations.
MDProxy CPU vs. GPU

CPU vs. GPU Comparison of MDProxy - 2688 Particles

Time (sec.)

Steps (#)

Parallel
CPU (sec)

GPU (sec)
Hardware Comparison

The graph compares the efficiency of different hardware configurations over sleep time. The x-axis represents the sleep time in microseconds (us), and the y-axis represents efficiency. The configurations include:

- NVIDIA GPU (1 Worker)
- NVIDIA GPU (84 Workers)
- Xeon Phi (60 Workers)
- AMD Opteron (48 Workers)
Conclusions

- *Integrated GeMTC + Swift/T*
  - programmability
  - scalability (multi-node support)
  - application domain

- *Evaluated Performance Benchmarks*

- Framework with ~200 independent workers per node

- *MIMD on SIMD*
  - MIMD collection of SIMD workers
  - unclear max efficiency for scientific applications
  - highly efficient for synthetic benchmarks
Lecture Outline

Lecture: (11:25pm-12:40pm)

● GeMTC (:45)
  ○ Motivation
    ■ Distributed Systems, HPC, MTC GPGPU.
  ○ GeMTC
    ■ Architecture, Design
    ■ Apps
  ○ Future Work
    ■ MTACS
    ■ Xeon Phi
    ■ GeMTC organization chart

● Swift/T (:30)
  ○ Slides as paper was presented. (CCGrid’13)
Future Work

- *Develop* Swift frameworks to evaluate additional applications
  - MDProxy
  - Protein Structure Prediction
  - Modeling of scientific knowledge acquisition in collaborative networks
- *Abstract* the Swift + GeMTC integration process for fast application deployment
- *Implement* an MTC solution for Intel Xeon-Phi and AMD GPUs.
Thanks! Questions?

Scott Krieder
skrieder@iit.edu
Dr. Ioan Raicu
iraicu@cs.iit.edu

Open Source:
https://github.com/skrieder-datasys/gemtc
NVIDIA CUDA Teaching & Research

- Expect to see more GPU projects in CS5XX
- Monthly CUDA Workshops
  - Starting September 3rd 1:50 - 2:40 LS 111
- IIT Free GPUs
Jarvis

- Rocks Cluster Management
  - CentOS
- 10 GPU Nodes
  - 2 Kepler K20
  - 3 GTX 650
  - 5 GTX 480
- Node Types
  - Frontend
  - Compute
Rocks Cluster Management

- Install to
  - frontend
- Compute nodes
  - PXE boot
- NFS
  - across nodes
- SGE
  - Batch scheduler
  - Interactive
  - Scheduled
Any Questions so far?

- Recap
  - HPC
  - GPGPU, CUDA
  - GeMTC
  - Jarvis

- Switch to Swift/T slides.
Hands on Outline

Hands On: (12:45pm-1:45pm)

● CUDA
  ○ Installing CUDA
  ○ SDK Examples
    ■ DeviceQuery
  ○ Vector-Add

● GeMTC
  ○ Vector-Add

● Swift/T
  ○ Vector-Add
How can I run CUDA?

● Getting started tips:
  ○ [https://sites.google.com/site/iitcuda/cuda-quickstart](https://sites.google.com/site/iitcuda/cuda-quickstart)
● Do you have an NVIDIA GPU on your machine?
  ○ Is it CUDA compliant?
● Jarvis
  ○ pending accounts/projects
Installing CUDA

- Download CUDA here:
CUDA Registered Developer

- Forum access
- Extra downloads
- Developer tools
- Register here:
Hands on Outline

Hands On: (12:45pm-1:45pm)

● CUDA
  ○ Installing CUDA
  ○ SDK Examples
    ■ DeviceQuery
  ○ Vector-Add

● GeMTC
  ○ Cloning the git repo.
  ○ Code overview.
  ○ Vector-Add

● Swift/T
  ○ Vector-Add