GeMTC: ManyGPU-enabled Many-Task Computing

Ioan Raicu
Computer Science Department
Illinois Institute of Technology

CS554: Data-Intensive Computing
February 16th, 2015

Logistics

- Quiz#2 graded
  - Minimum Value: 0
  - Average: 6.40
  - Median: 6.00
  - Maximum Value: 10.00
  - Standard Deviation: 2.28

Logistics

- Project proposal writeup
  - Will be posted today
- Project brainstorming ideas
  - Will be posted today
  - This is your reading assignment, plus papers cited in these writeups
- Project brainstorming next 3 lectures
- Project proposals and team formations due March 2nd (midnight)

More Information

- More information:
  - http://www.cs.iit.edu/~iraicu/
  - http://datasys.cs.iit.edu/
- Contact:
  - iraicu@cs.iit.edu
- Questions?

Design and Evaluation of the GeMTC Framework for GPU-enabled Many-Task Computing

Scott J. Krieder, Justin M. Wozniak, Timothy Armstrong, Michael Wilde, Daniel S. Katz, Benjamin Grimmer, Ian T. Foster, Ioan Raicu

ILLINOIS INSTITUTE OF TECHNOLOGY

DataSys
Data-Intensive Distributed Systems Laboratory

HPDC 2014

Vancouver, Canada

Acknowledgements

Dr. Justin Wozniak, Argonne (ANL) Computer Scientist

Timothy Armstrong, UChicago, CS Ph.D. Student

Michael Wilde, UChicago CI Fellow & ANL Software Architect

Dr. Daniel S. Katz, Senior Fellow, Computation Institute, Univer

Dr. Ian T. Foster, Director, Computation Institute at UChicago & ANL Senior Scientist

Dr. Ioan Raicu, Advisor, Illinois Institute of Technology

Benjamin Grimmer, Undergraduate, Illinois Institute of Technology






Outline

- Background Information
  - Many-task computing
  - Hardware Accelerators
- Proposed Work
  - GeMTC = GPU enabled Many Task Computing
- GeMTC Architecture
- Swift and the dataflow model
- Performance Evaluation
- Closing Remarks & Future Directions

Distributed Paradigms

HPC:

- Tightly coupled
- Large jobs
- Hours/days
- Low latency

MTC

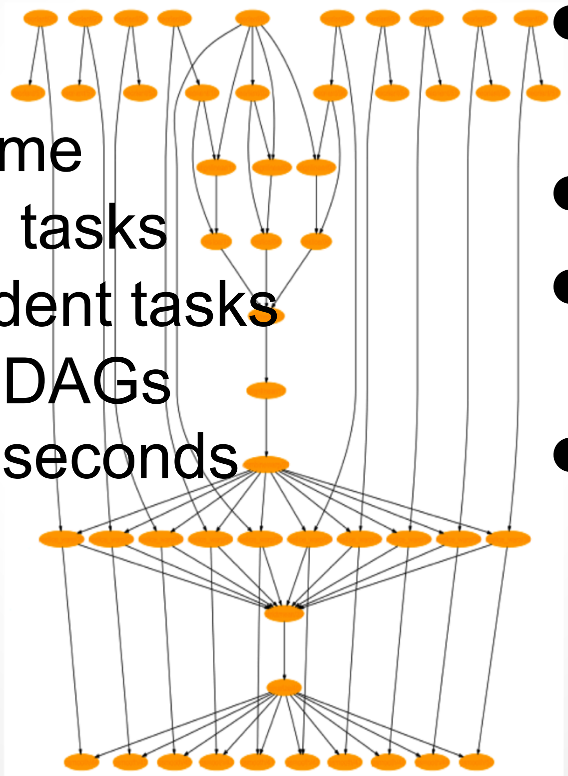
HTC:

- Loosely coupled
- Days/Months
- Fault tolerance

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

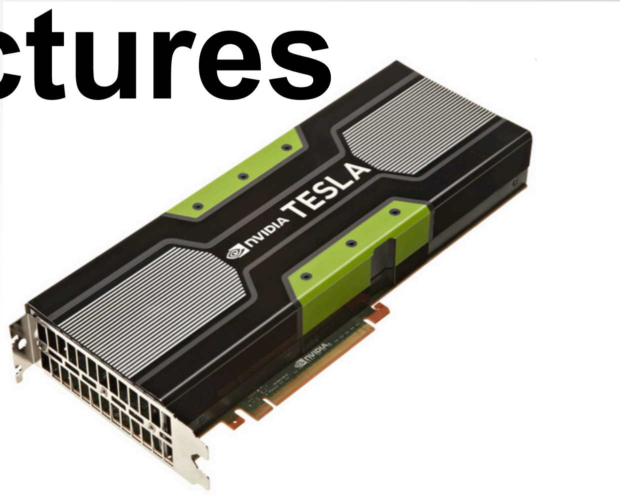


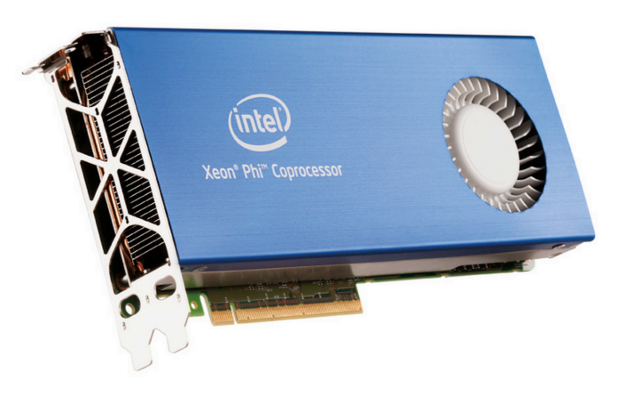
Advantages:

- Improve fault tolerance
- Maintain efficiency
- Programmability & Portability
- Support pleasingly parallel and complex applications

Accelerator Architectures

- GPU - Streaming Multiprocessors (15 SMXs on Kepler K20)
- Warps
  - 32 threads in a warp
  - 192 warps
    1. hardware available
    2. independent compute element
- Intel Xeon Phi
  - 60 cores * 4 threads per core = 240 hardware threads





Proposed Work

GeMTC: **G**PU **e**nabled **M**any-**T**ask **C**omputing

Motivation: No support for MTC on Accelerators!

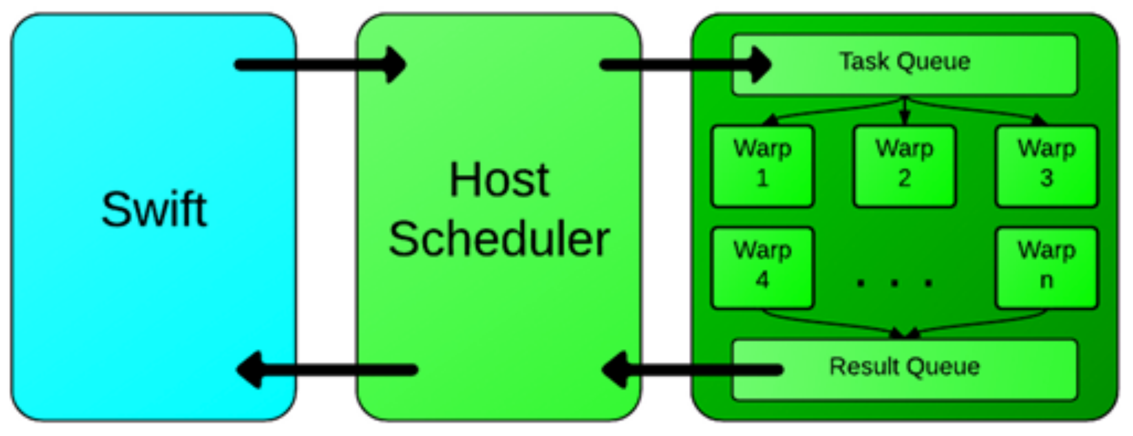
Goals:

1) MTC support
2) Programmability
3) Efficiency
4) MIMD on SIMD
5) Increase concurrency 15 to 192 (~13x)

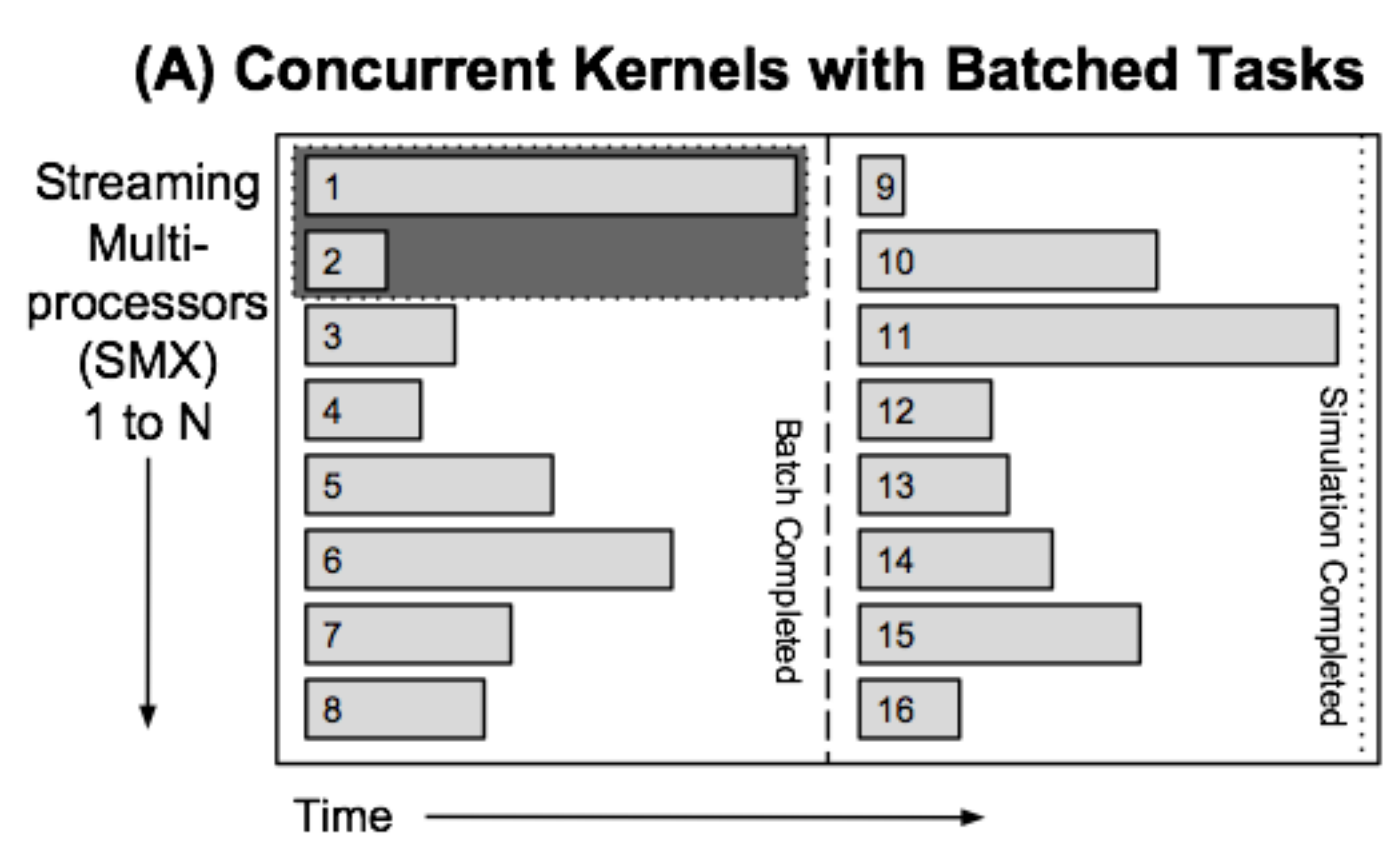
Approach:

Design, implement middleware:

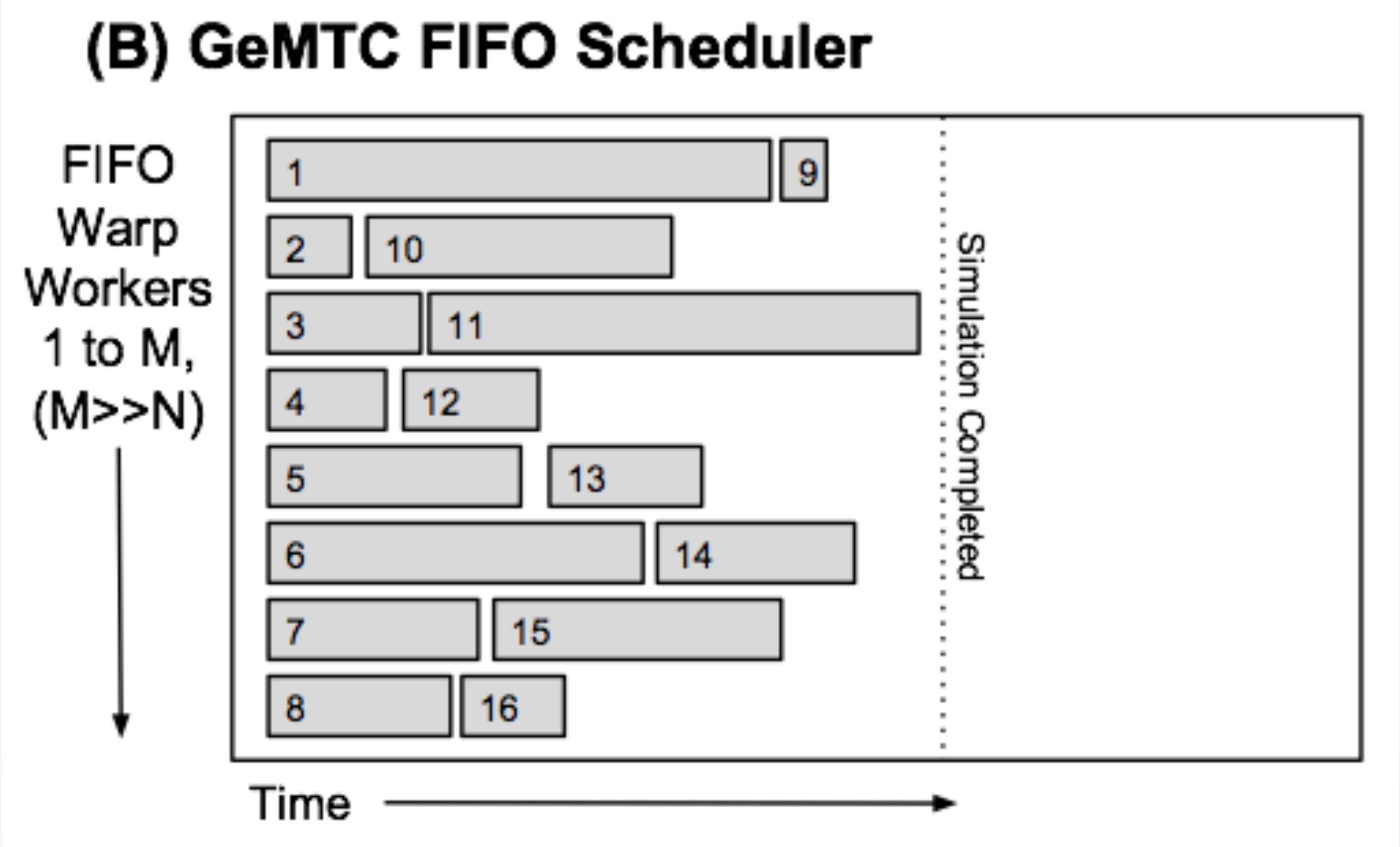
1) Manages GPU
2) Spread host/device
3) Workflow system integration (Swift/T)



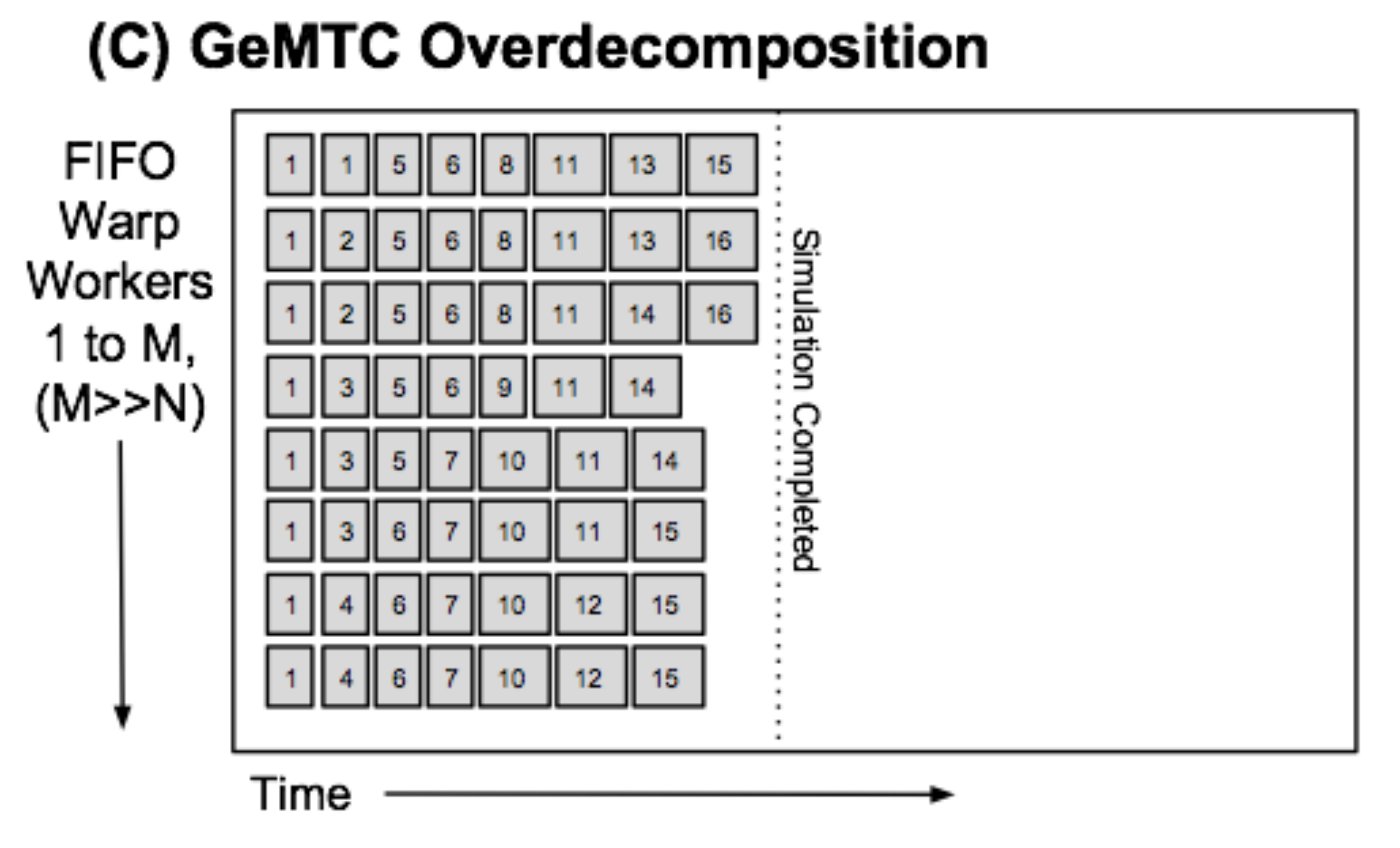
CUDA Concurrent Kernels



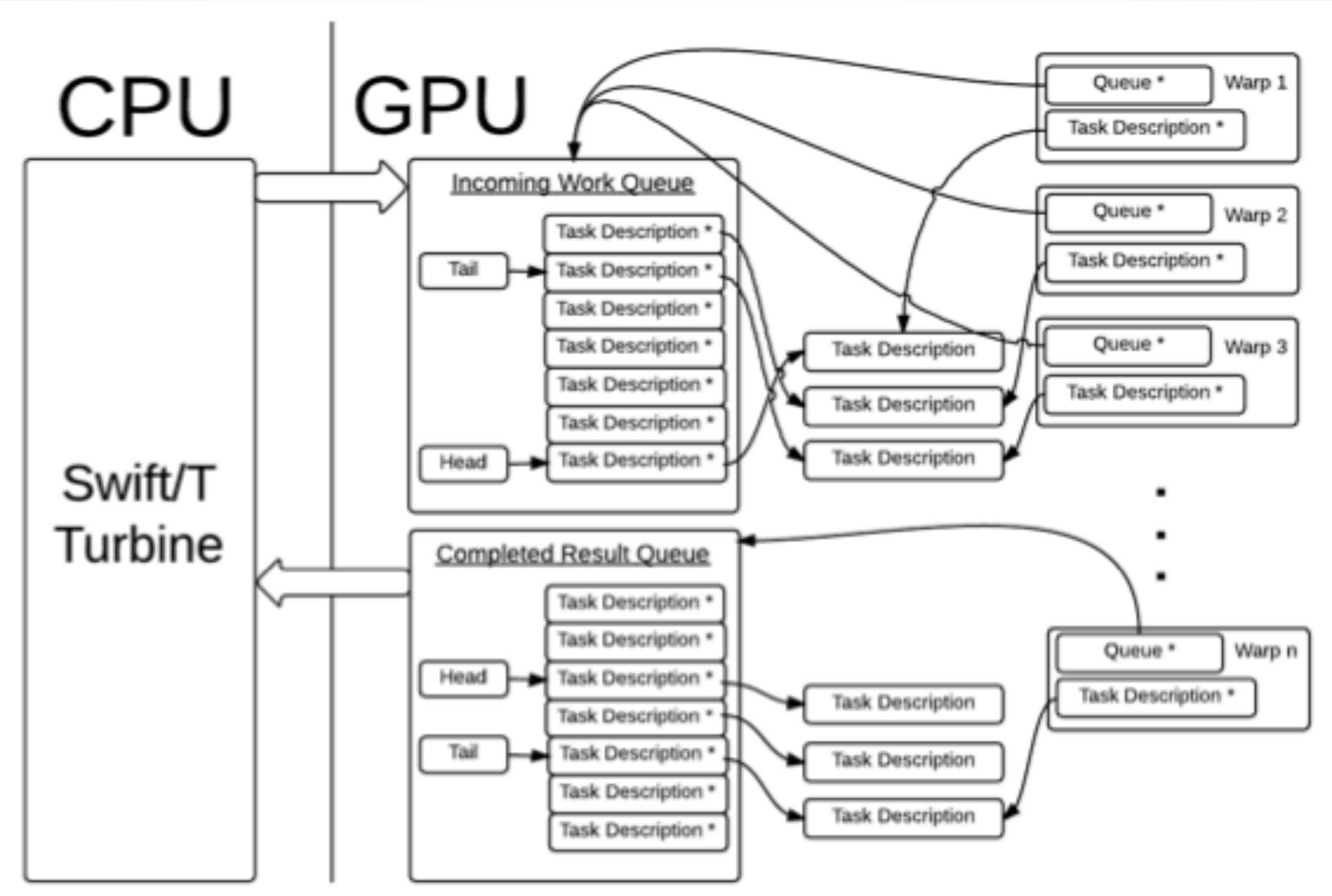
GeMTC FIFO


(B) GeMTC FIFO Scheduler
FIFO Warp Workers 1 to M, (M>>N)
1
9
2
10
3
11
4
12
5
13
6
14
7
15
8
16
Simulation Completed
Time

GeMTC Overdecomposition



GeMTC Architecture



GeMTC API

Device Management:

- gemtcSetup()
- gemtcCleanup()

Task Management:

- gemtcPush()
- gemtcPoll()

Data Movement:

- gemtcMemcpyDevToHost()
- gemtcMemcpyHostToDev()

Memory Management*:

- gemtcGPUMalloc()
- gemtcGPUFree()

**EuroSys'13 Poster*

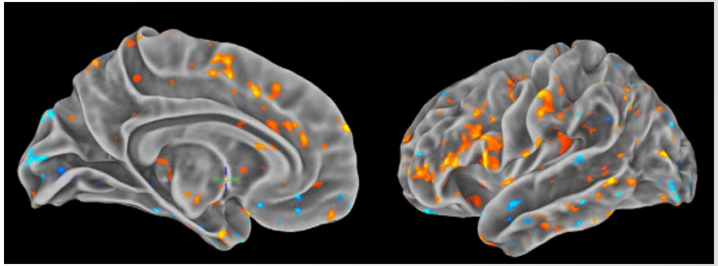
GeMTC AppKernels

- Precompiled into GeMTC Framework
- Optimized for Single Warp Execution
  - (Future: Strap together multiple warps)
- Previous AppKernel Work:
  - Molecular Dynamics, Synthetic Benchmarks
- Current AppKernel Work:
  - BLAS functionality, etc.
    - SAXPY, SGEMM, Image processing, Black Scholes

```
#include "gemtc.cu"
main(){
   // Start GeMTC
   gemtcSetup(QUEUE_SIZE);
   // Allocate device memory
   d_array = gemtcGPUMalloc(MALLOC_SIZE);
   // Populate device memory
   gemtcMemcpyHostToDevice(d_array,
     h_array, MALLOC_SIZE);
   // Push a task to the GPU
   gemtcPush(MATMUL, NUM_THREADS,
       TaskID, d_array);
   // Poll for completed results
   gemtcPoll(TaskID, pointer);
   // Copy back results
   gemtcMemcpyDeviceToHost(h_array,
       pointer, MALLOC_SIZE);
   // Free GPU memory
   gemtcGPUFree(pointer);
   // Shutdown GeMTC
   gemtcCleanup();
}
```

Swift and Applications

- Swift
  - 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



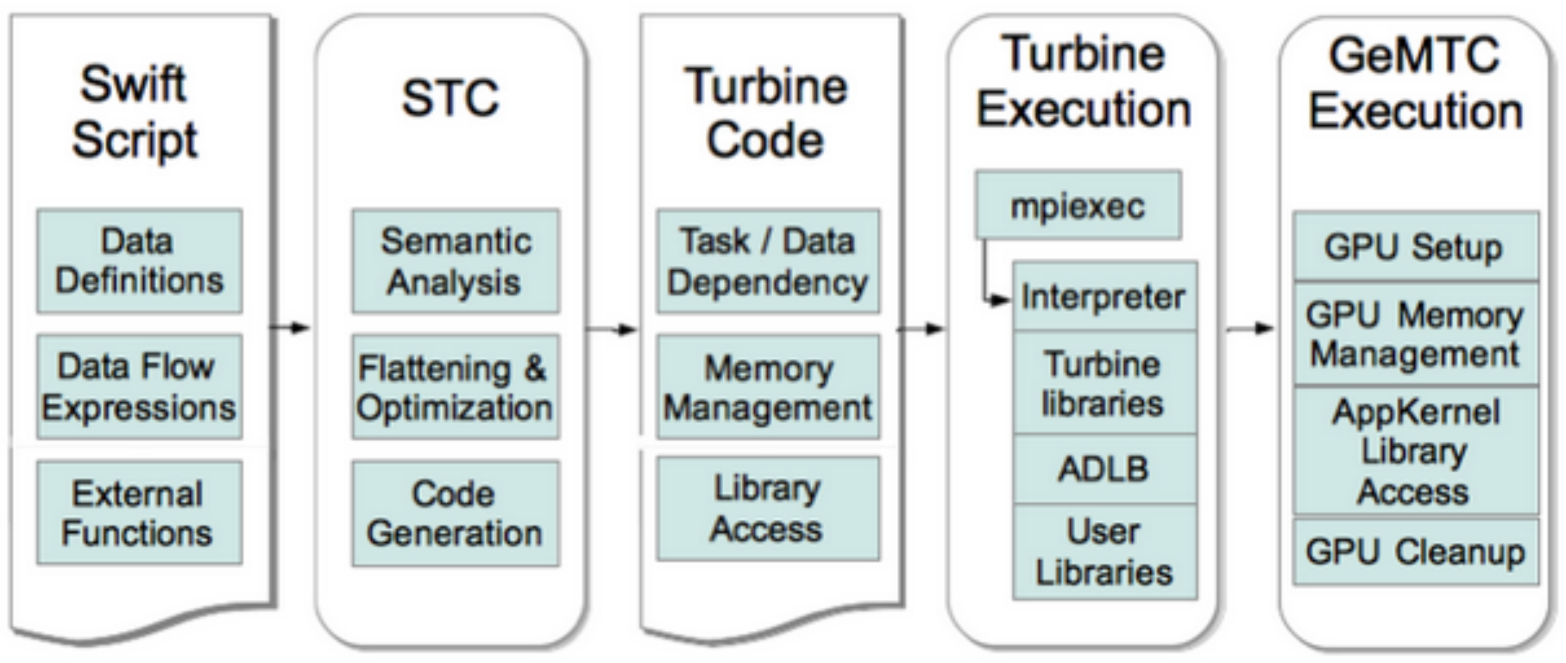
Swift lets you write parallel scripts that run many copies of ordinary programs concurrently, using statements like this:

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

Images from Swift Case Studies - http://www.ci.uchicago.edu/swift/case_studies/

Swift Dataflow & Integration

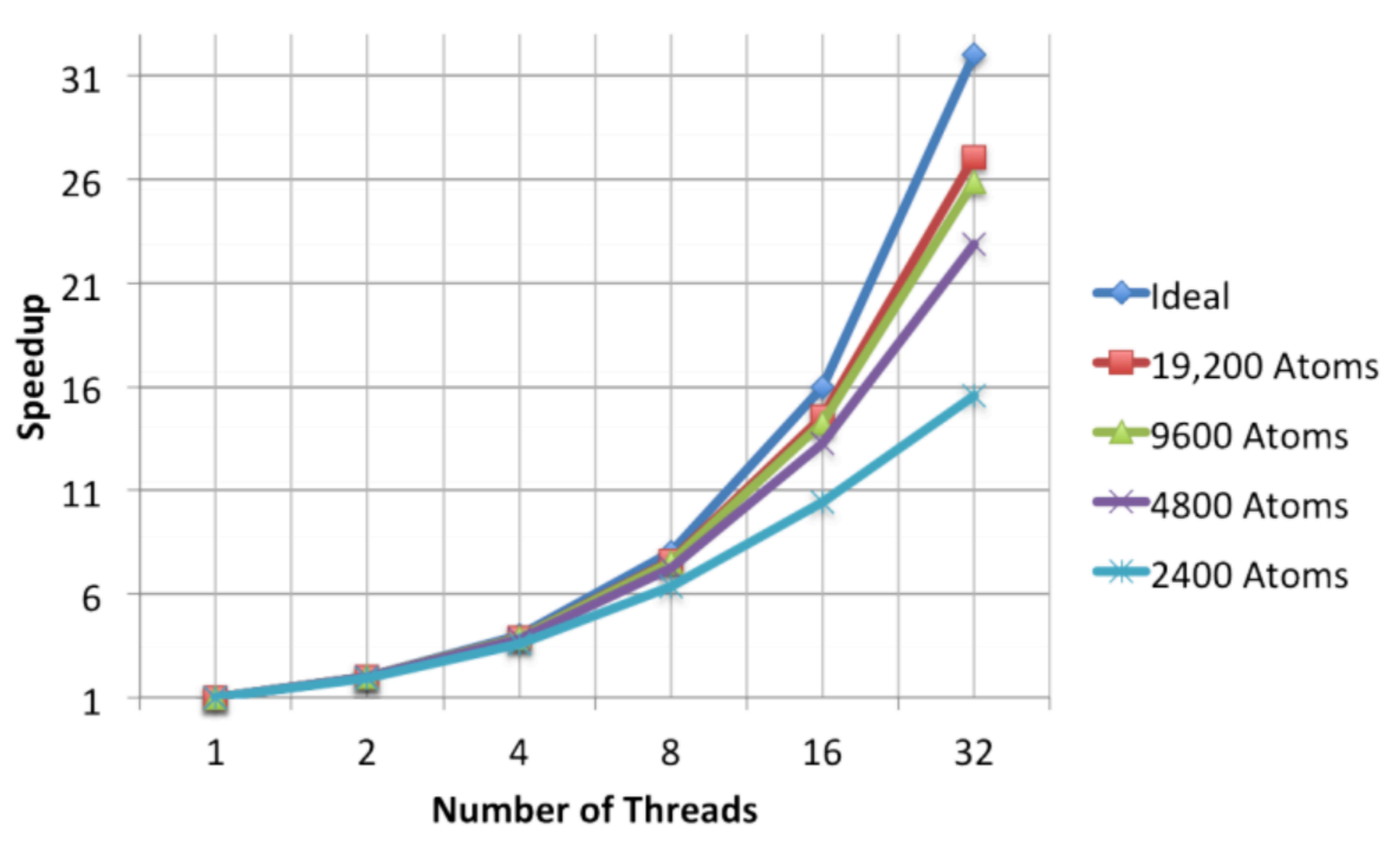
```
…
x = f(a);
y = f(b);

c = g(x, y);
...
```



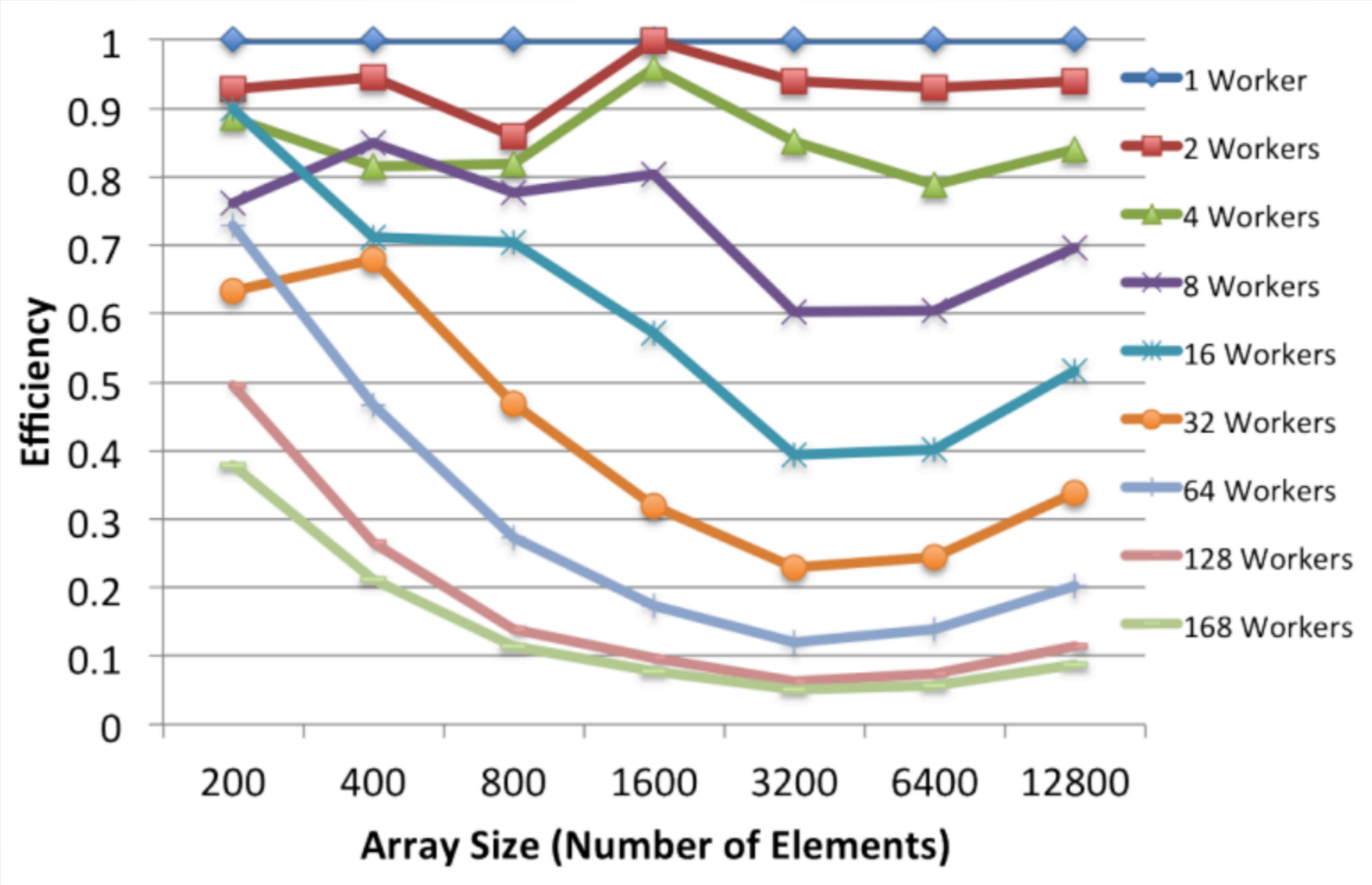
Performance Evaluation

- **GeMTC and Molecular Dynamics**
- GeMTC Throughput and Efficiency (Leveraging Swift)
- Preliminary Results on Intel Xeon Phi

Speedup within a Single Warp



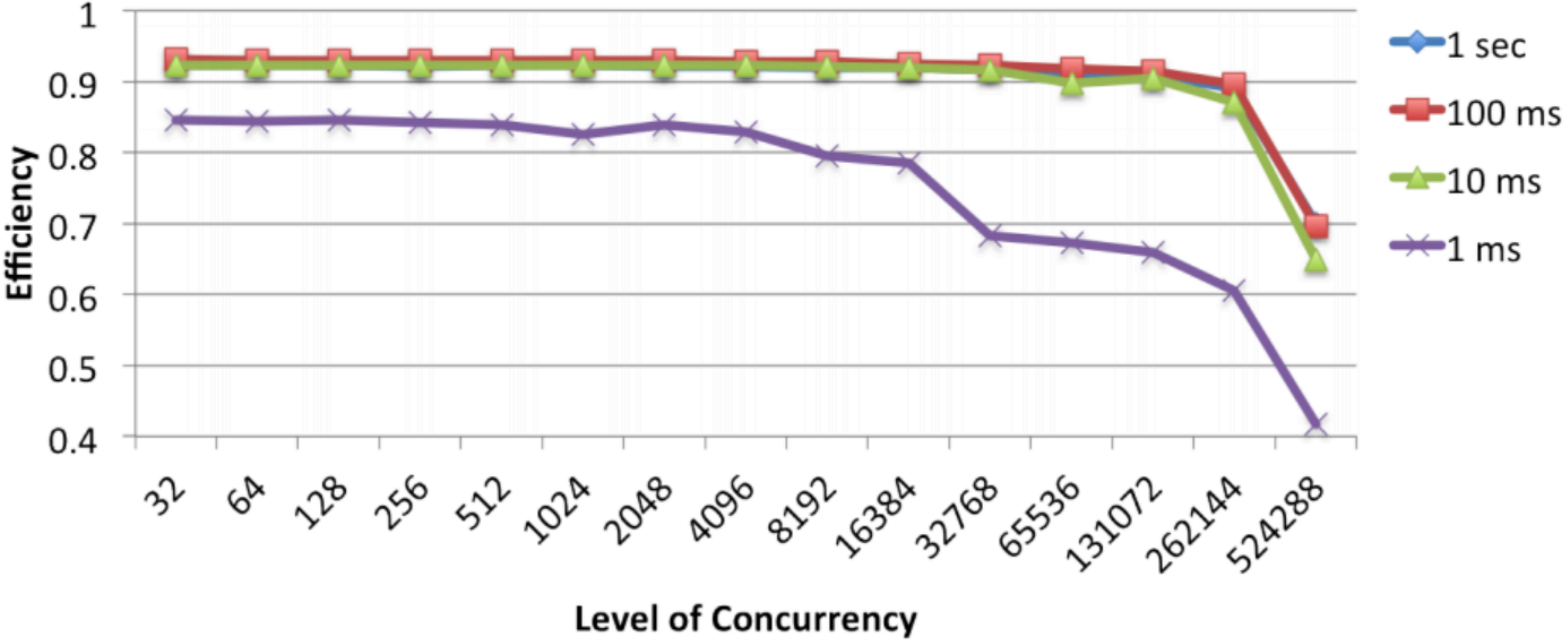
GeMTC Utilization on K20X



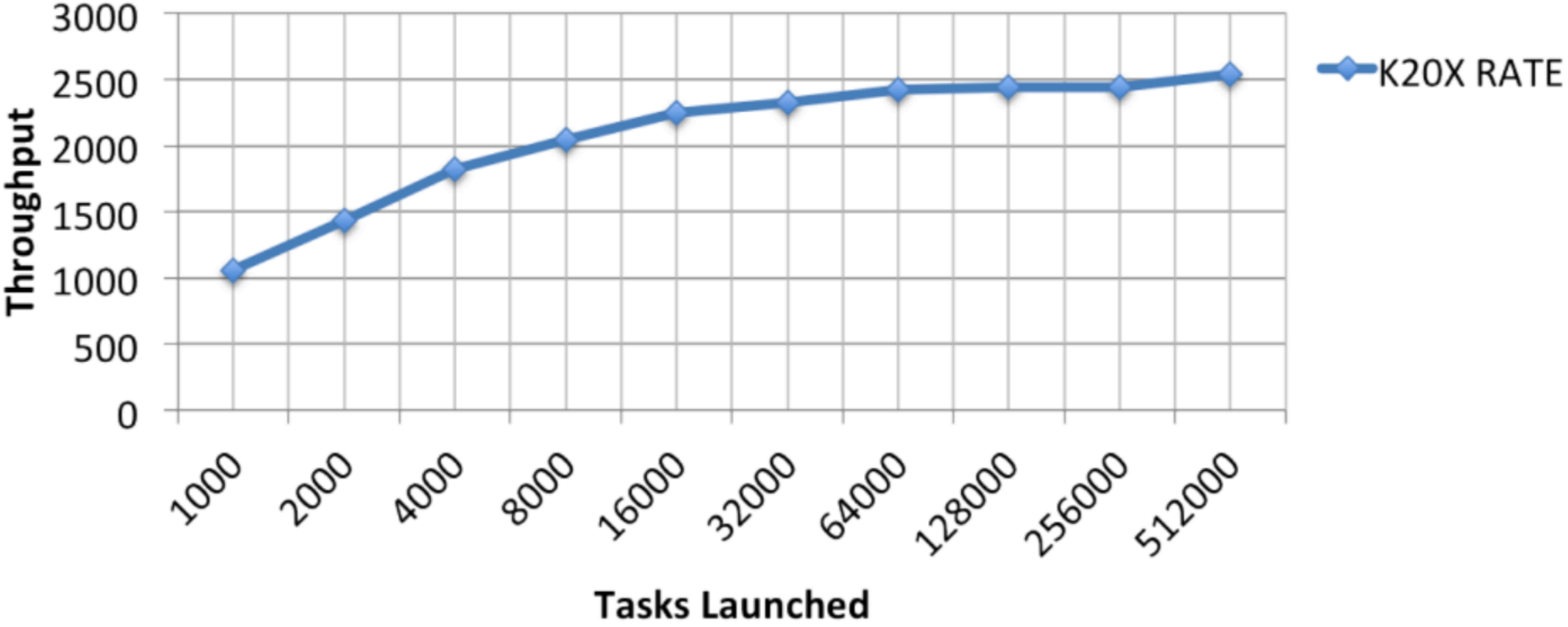
Performance Evaluation

- GeMTC and Molecular Dynamics
- **GeMTC Throughput and Efficiency (Leveraging Swift)**
- Preliminary Results on Intel Xeon Phi

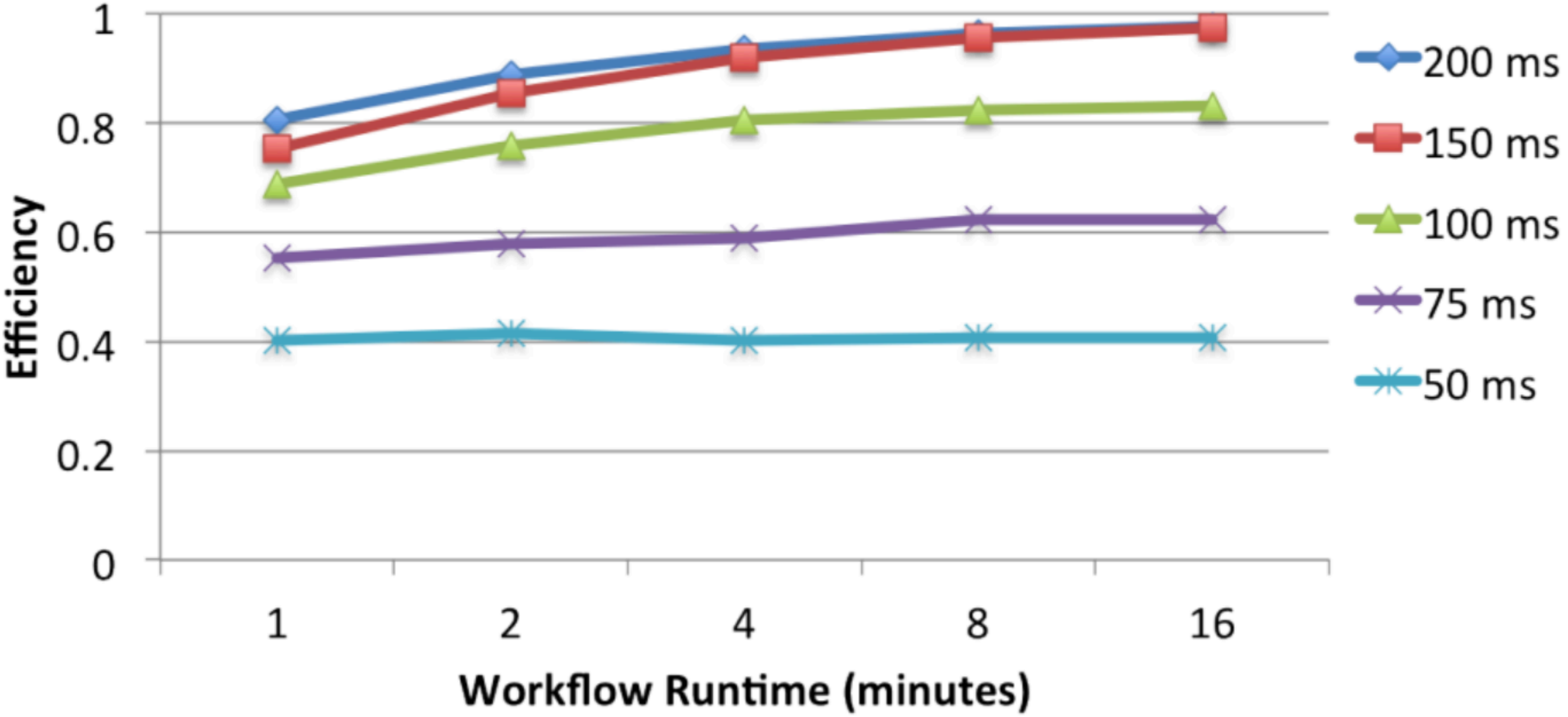
Fine-grained Swift CPU Workloads



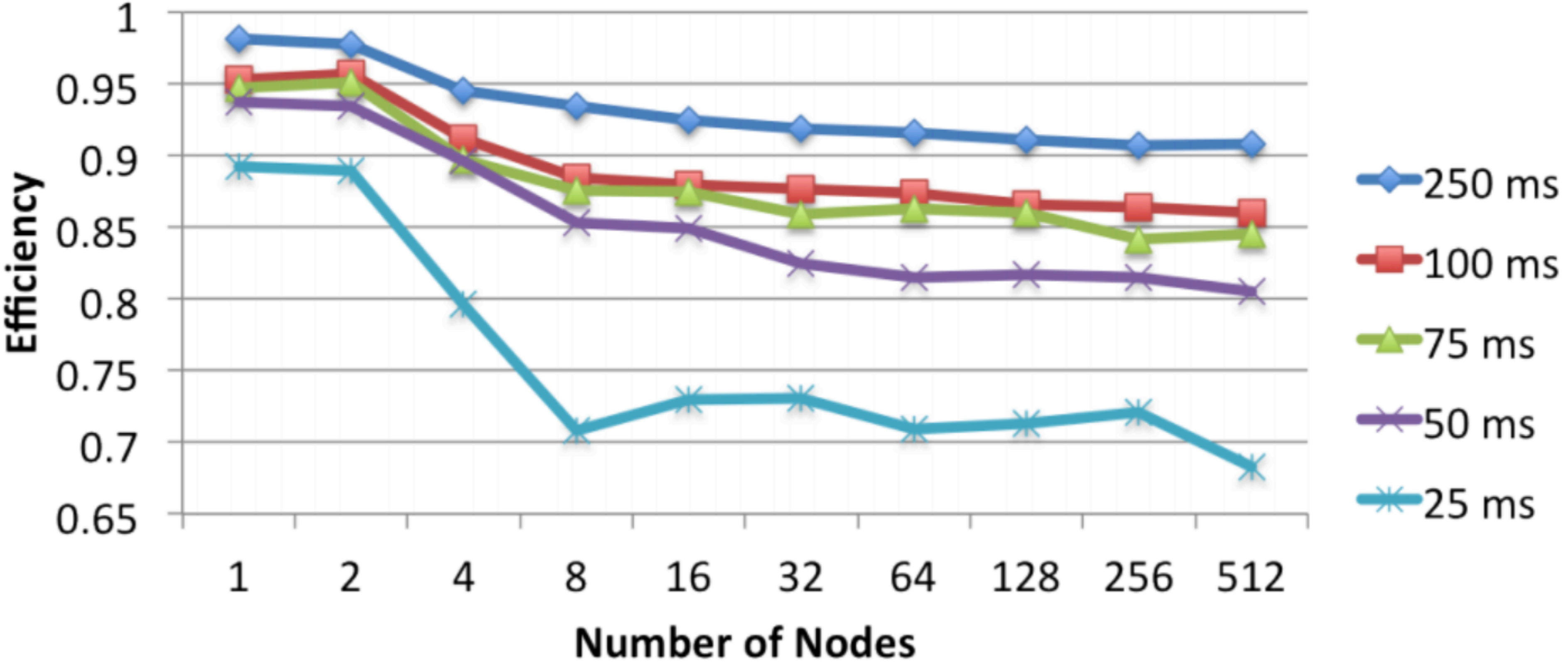
GeMTC + Swift on XK7 of Blue Waters



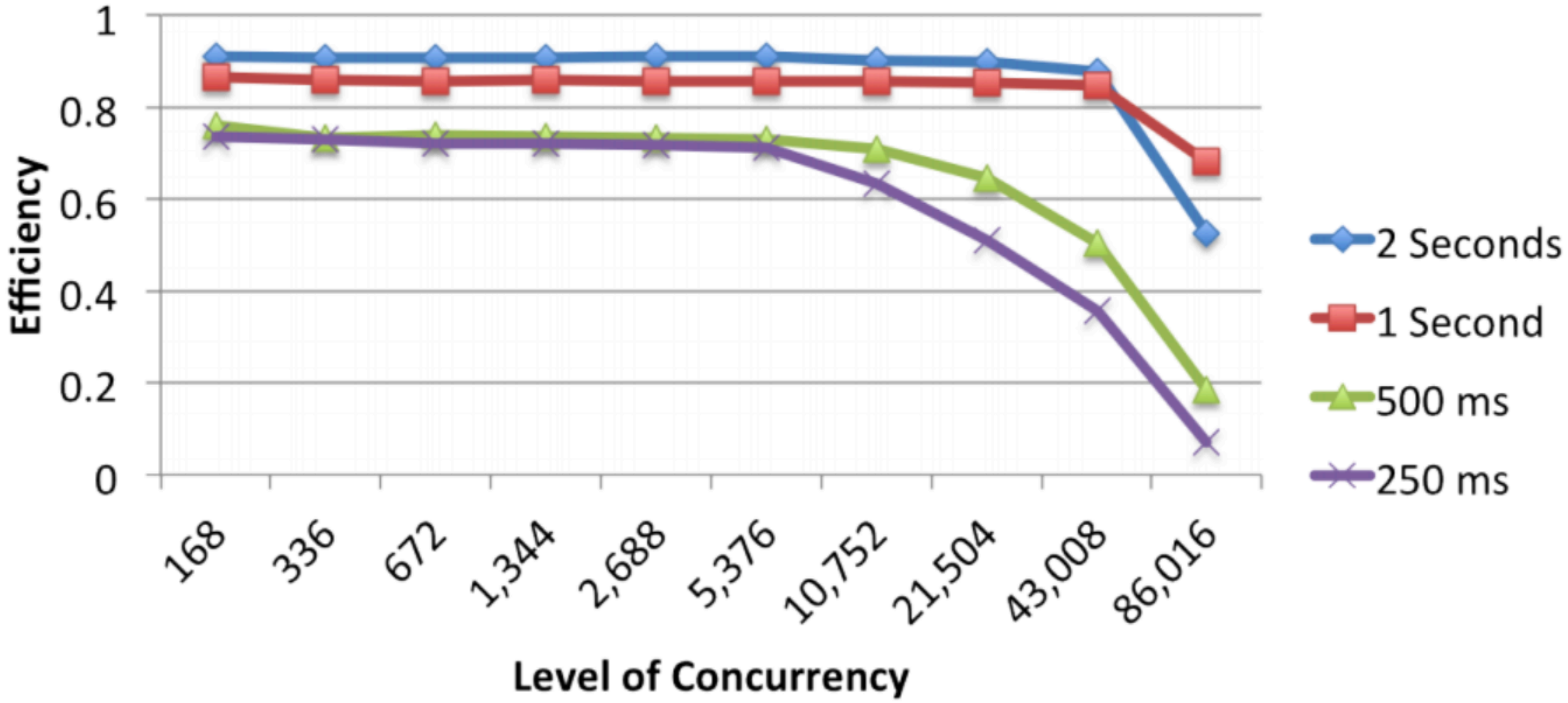
Single XK7 Node Efficiency



GeMTC + Swift 512 Nodes 1 Worker/GPU



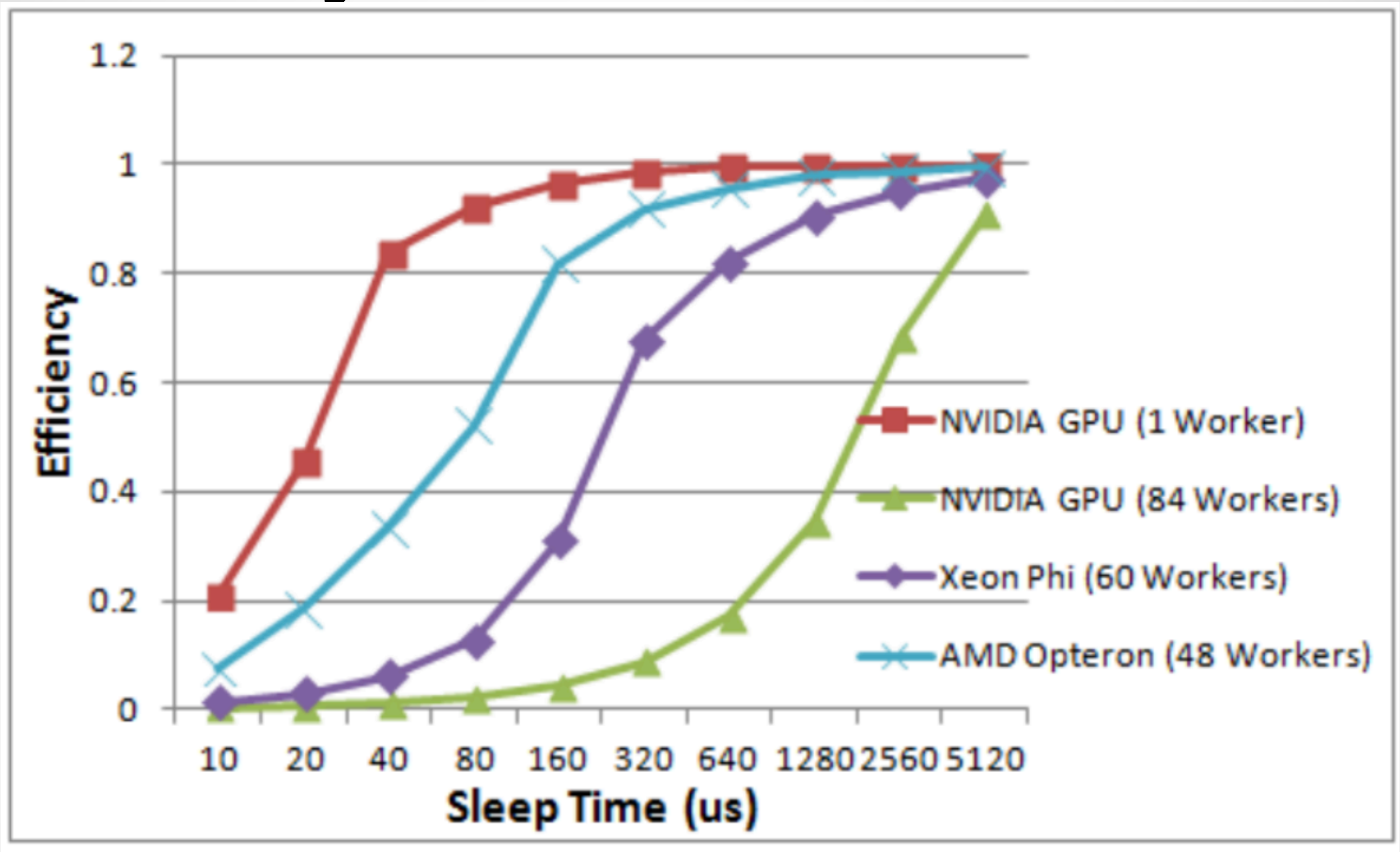
GeMTC + Swift 512 Nodes, 168W/GPU



Performance Evaluation

- GeMTC and Molecular Dynamics
- GeMTC Throughput and Efficiency (Leveraging Swift)
- **Preliminary Results on Intel Xeon Phi**

Preliminary Results on Intel Xeon Phi



Conclusion & Future Work

- Efficient MTC on NVIDIA GPUs
- MIMD on SIMD

- More efficient node utilization (CPU)
- Strap together multiple warp workers
- Support alt. accelerators (OpenCL, OpenACC)
- CUDA 6 Enhancements (Unified Memory, etc.)

Code Repositories

GeMTC:
http://datasys.cs.iit.edu/projects/GeMTC
https://github.com/skrieder/gemtc
Swift:
http://swift-lang.org/main/

Questions?

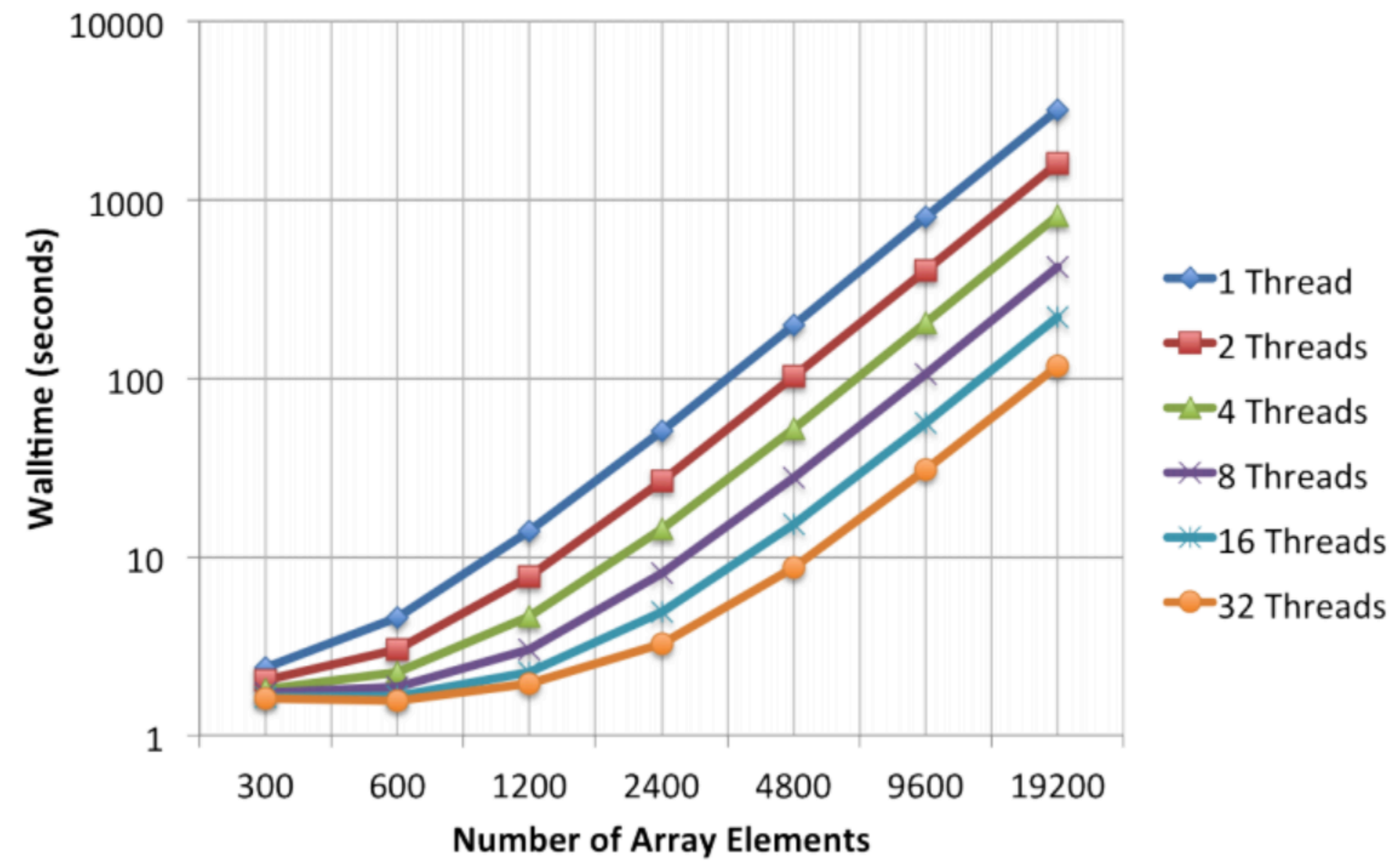
Scott J. Krieder
Illinois Institute of Technology
skrieder@iit.edu
@skrieder
http://datasys.cs.iit.edu/~skrieder

Appendix:
Additional Slides and Details

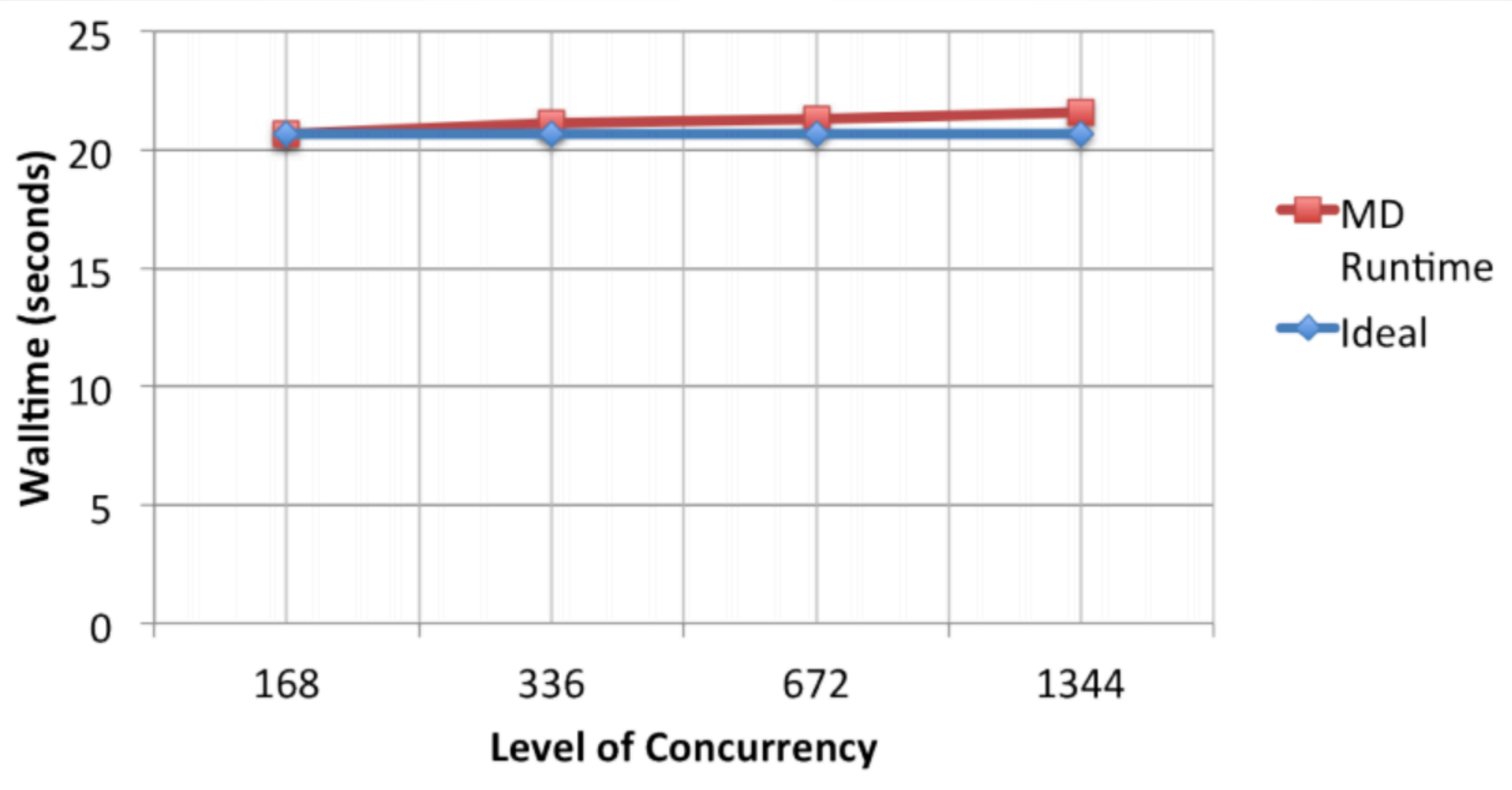
Related Work

- Warp-level execution
  - Graph processing - Hong et. al., PPoPP’11
- Dataflow on Accelerators
  - PTask, Rossbach et al., MSR
- Accelerator Virtualization
  - Becchi et. al., Ravi, Pegasus
- Runtime systems
  - StarPU, COSMIC

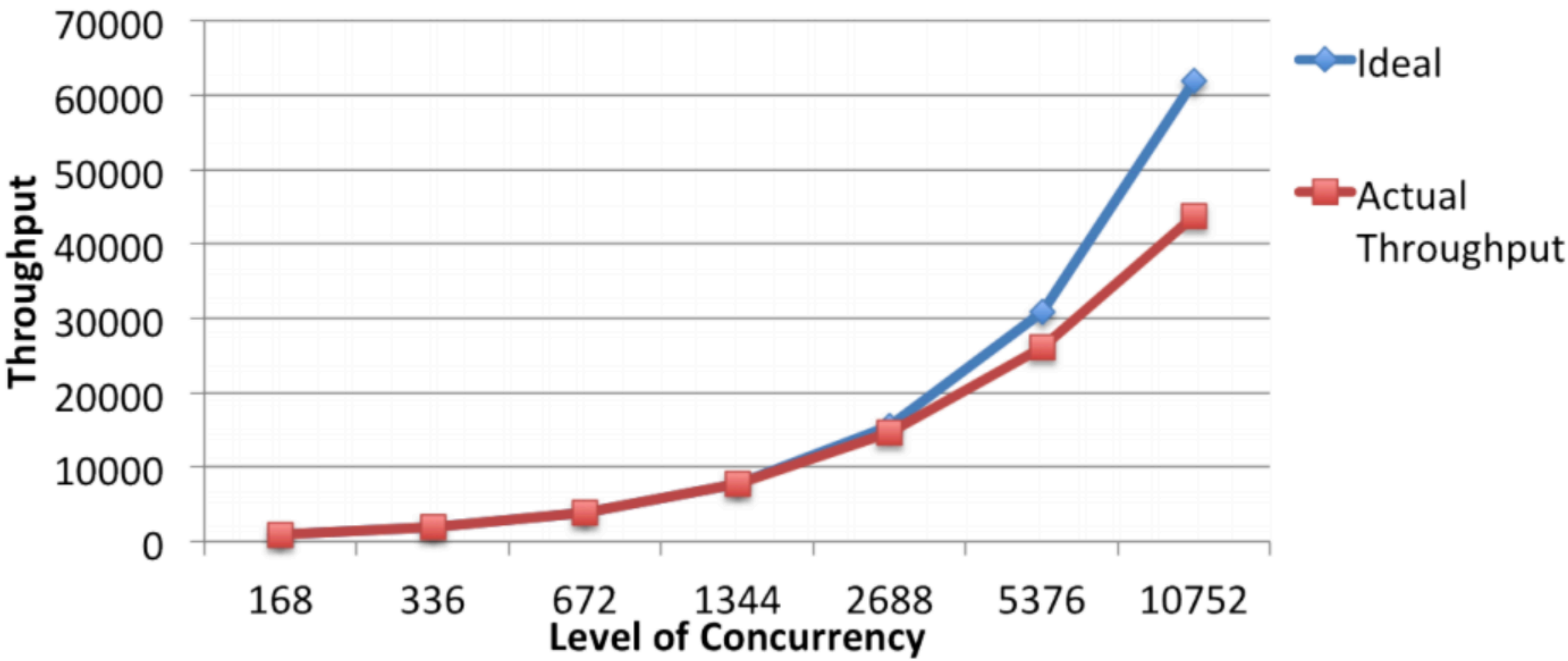
GeMTC and MD over Single Warp



GeMTC and MDLite over 1344 Workers



GeMTC + Swift over 10,000 GPU Workers



GeMTC Memory Management