Hybrid Dataflow Programming on Blue Waters

Ioan Raicu, Michael Wilde

Blue Waters Symposium, 2014

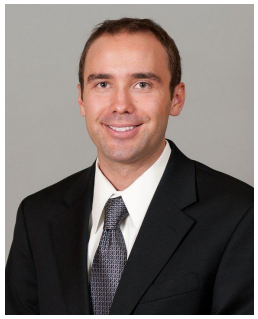
Department of Computer Science, Illinois Institute of Technology
Mathematics and Computer Science Division, Argonne National Laboratory
Computation Institute, University of Chicago

http://swift-lang.org

Collaborators on this work

Scott J. Krieder, Illinois Institute of Technology
Timothy Armstrong, University of Chicago
Justin Wozniak, Argonne
Daniel S. Katz, University of Chicago
Ian T. Foster, University of Chicago and Argonne
Ioan Raicu, Advisor, Illinois Institute of Technology
Michael Wilde, Argonne / University of Chicago

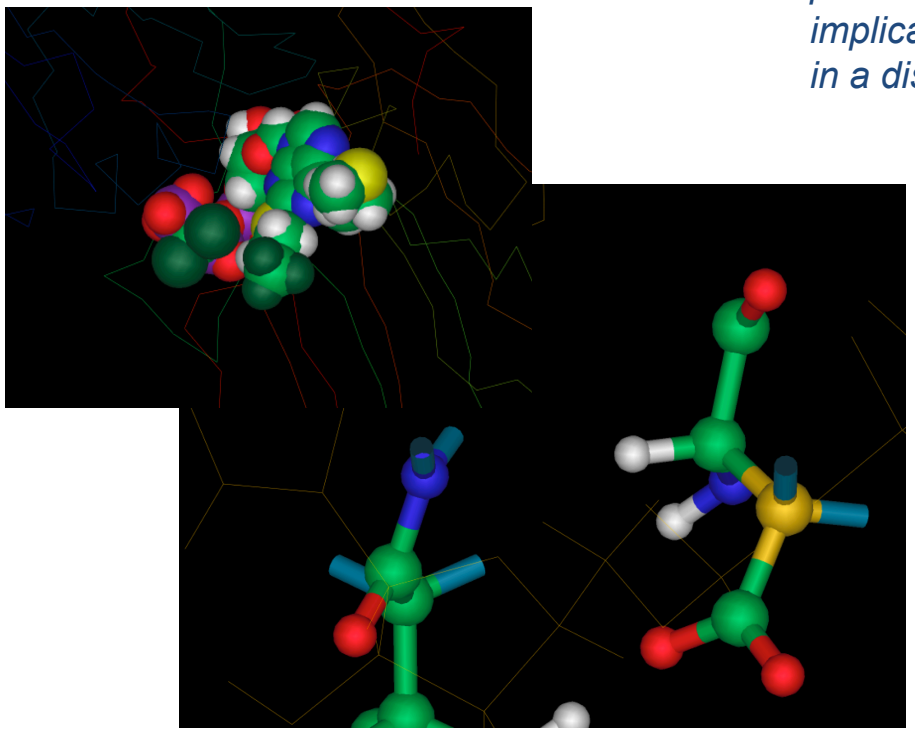
Supported in part by Blue Waters/GLCPC, NSF SI2, and DOE-ASCR X-Stack




When do you need parallel scripting?

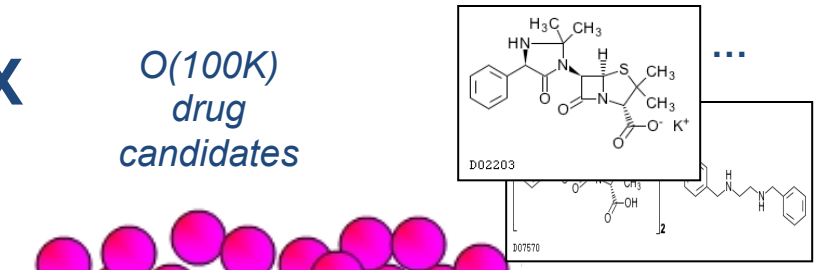
Typical application: protein-ligand docking for drug screening

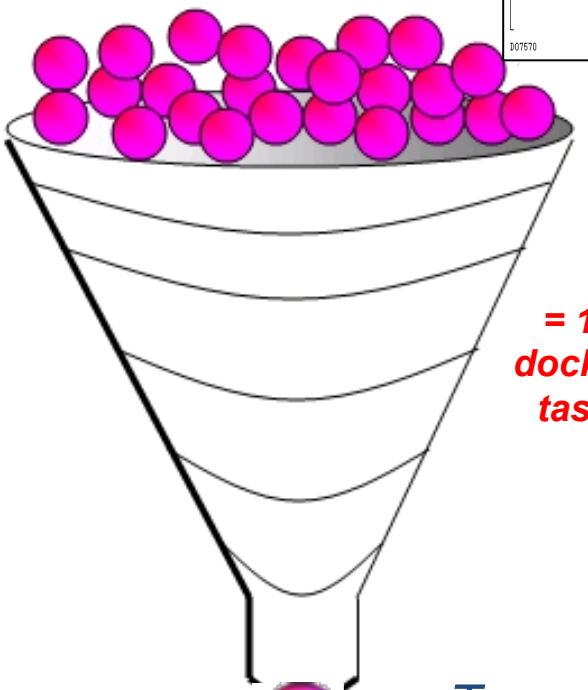


O(10) proteins implicated in a disease

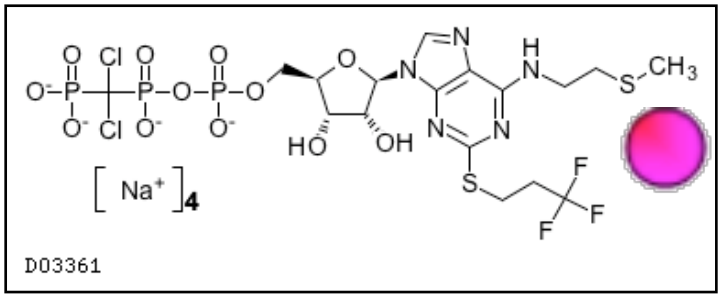
X

O(100K) drug candidates





= 1M docking tasks

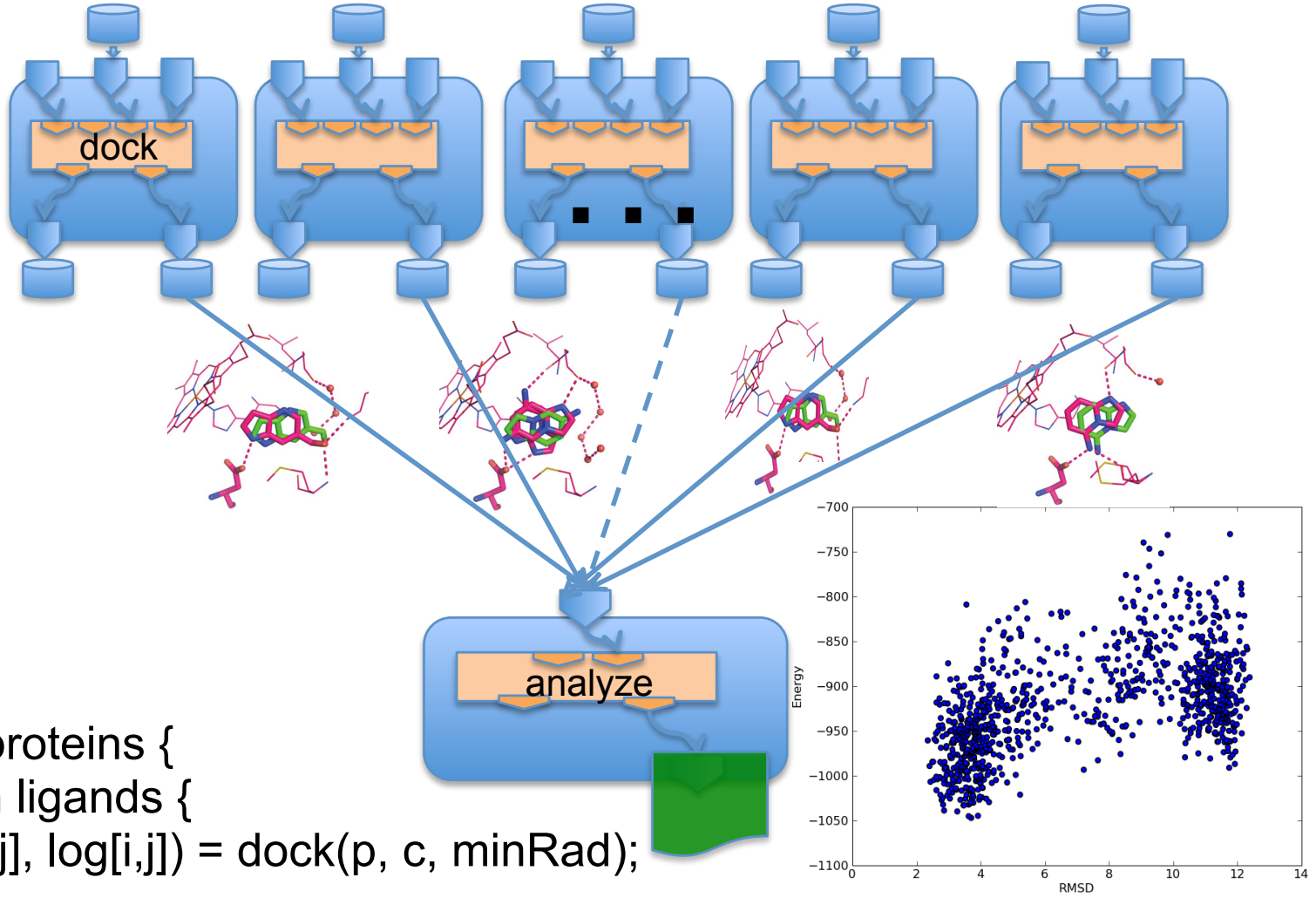


Tens of fruitful candidates for wetlab & APS

Work of M. Kubal, T.A.Binkowski, and B. Roux

Problem: *How to code such applications?*

1,000,000 runs of the "dock" application




```
foreach p, i in proteins {
   foreach c, j in ligands {
      (structure[i,j], log[i,j]) = dock(p, c, minRad);
   }
}
scatter_plot = analyze(structure)
```

Solution: *A compact, portable script*

Swift code excerpt:

```
foreach p, i in proteins {
   foreach c, j in ligands {
      (structure[i,j], log[i,j]) =
       dock(p, c, minRad, maxRad);
   }
}
scatter_plot = analyze(structure)
```

To run:

```
swift –site stampede,trestles     \
      docksweep.swift
```

Programming model: all execution driven by parallel data flow

```
(int r) myproc (int i)
{
    j = f(i);
    k = g(i);
    r = j + k;
}
```

f() and g() are computed in parallel

myproc() returns r when they are done

This parallelism is *automatic*

Works recursively throughout the program's call graph

Pervasive parallel data flow

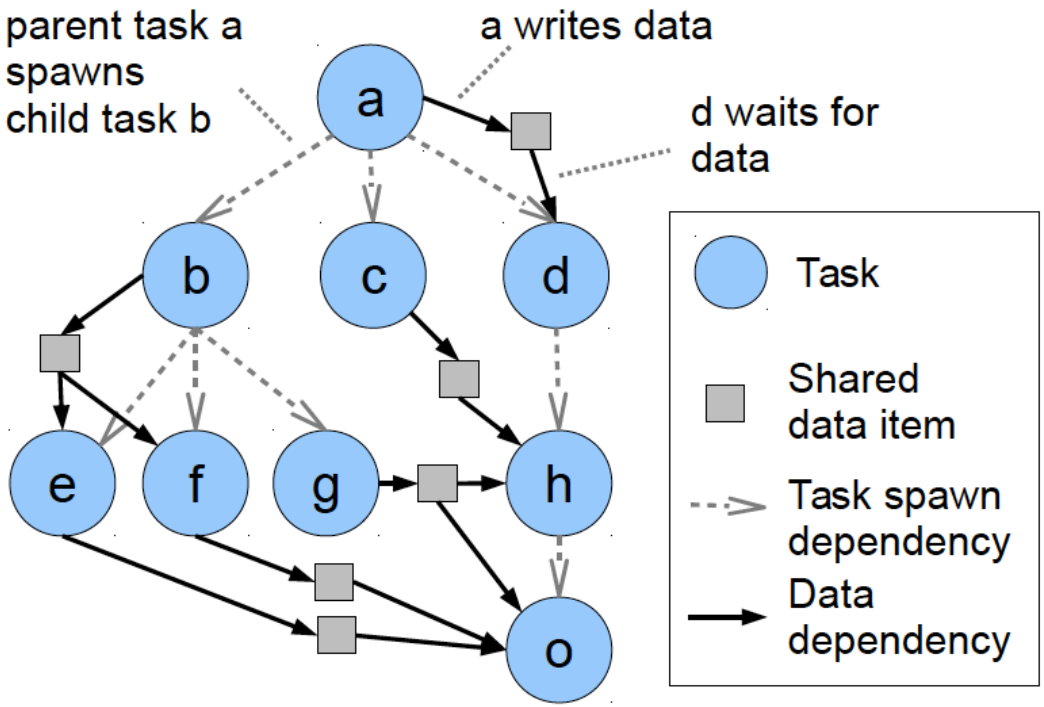
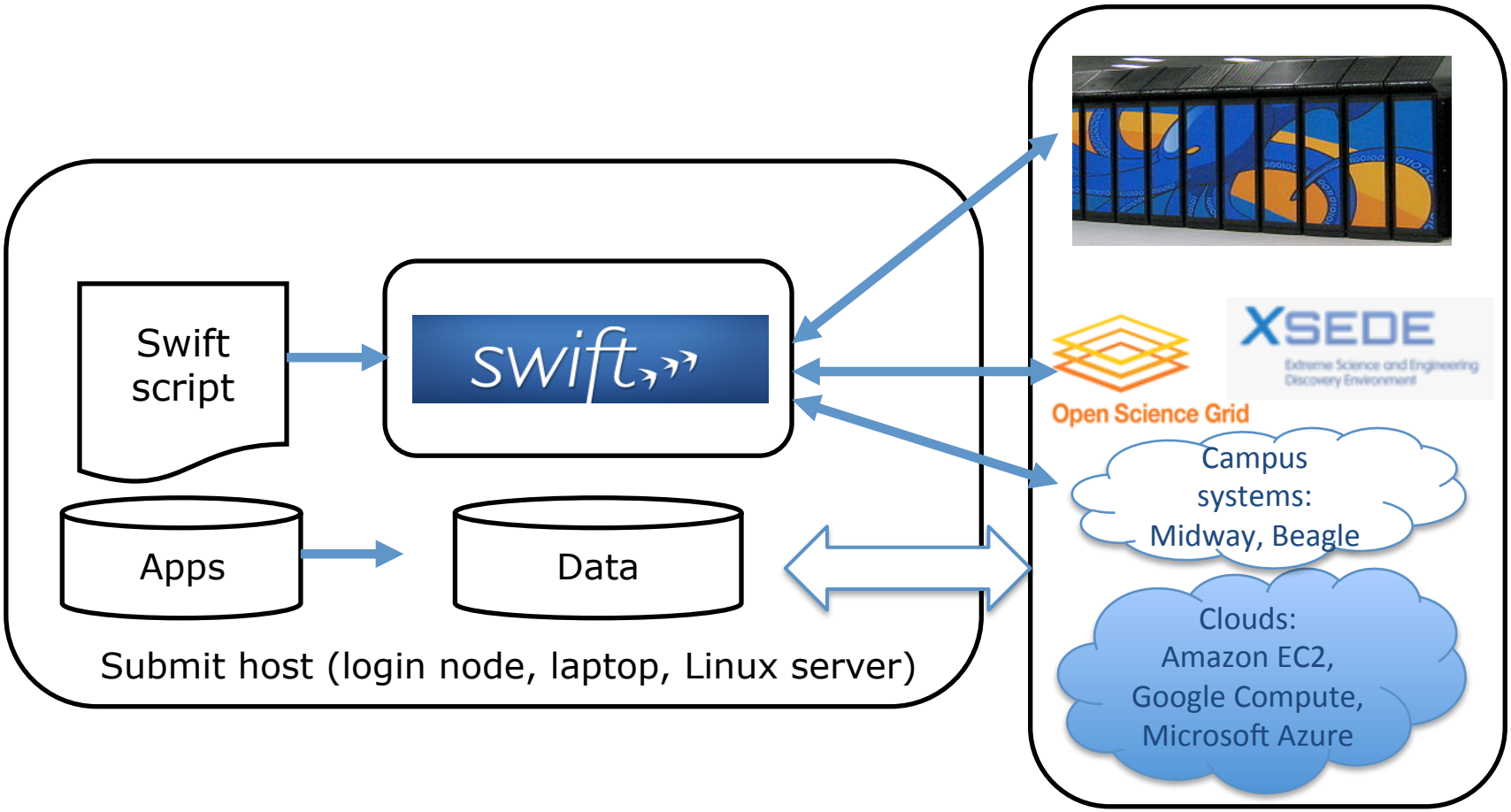
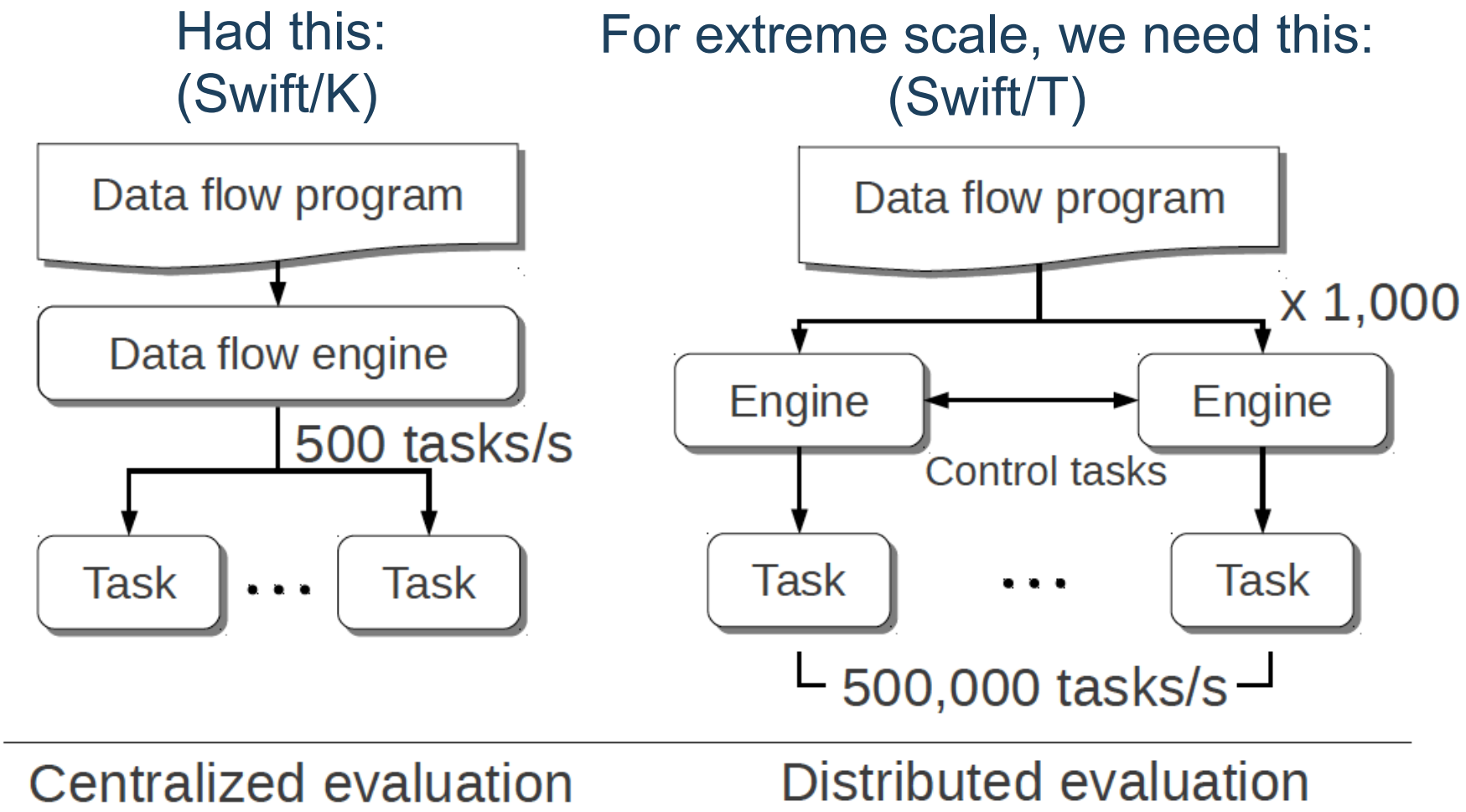


Fig. 1: Task and data dependencies in data-driven task parallelism, forming a spawn tree rooted at task a. Data dependencies on shared data defer execution of tasks until the variables are finalized.

Swift runs across diverse parallel platforms



…but centralized evaluation is a bottleneck at extreme scales



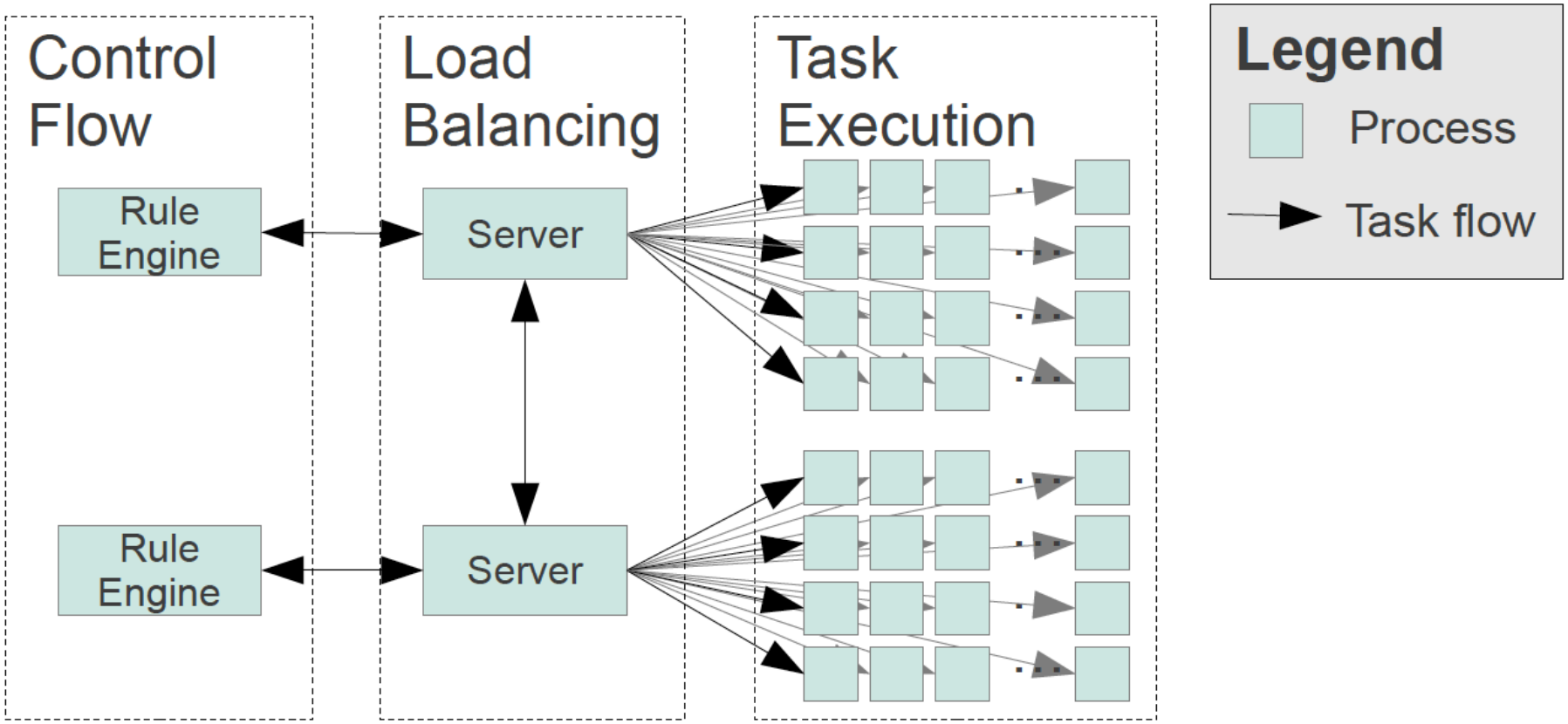
ExM – Extreme scale Many-task computing

Compiler Techniques for Massively Scalable Implicit Task Parallelism

Timothy G. Armstrong,* Justin M. Wozniak,†‡ Michael Wilde,†‡ Ian T. Foster*†‡
*Dept. of Computer Science, University of Chicago, Chicago, IL, USA
†Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, USA
‡Computation Institute, University of Chicago and Argonne National Laboratory, Chicago, IL, USA

http://people.cs.uchicago.edu/~tga/pubs/stc-preprint-apr14.pdf

MPI process architecture for parallel evaluation in Swift/T



GeMTC: GPU-enabled Many-Task Computing

Motivation: Support for MTC on all accelerators!

Goals:

1) MTC support 2) Programmability

3) Efficiency 4) MPMD on SIMD

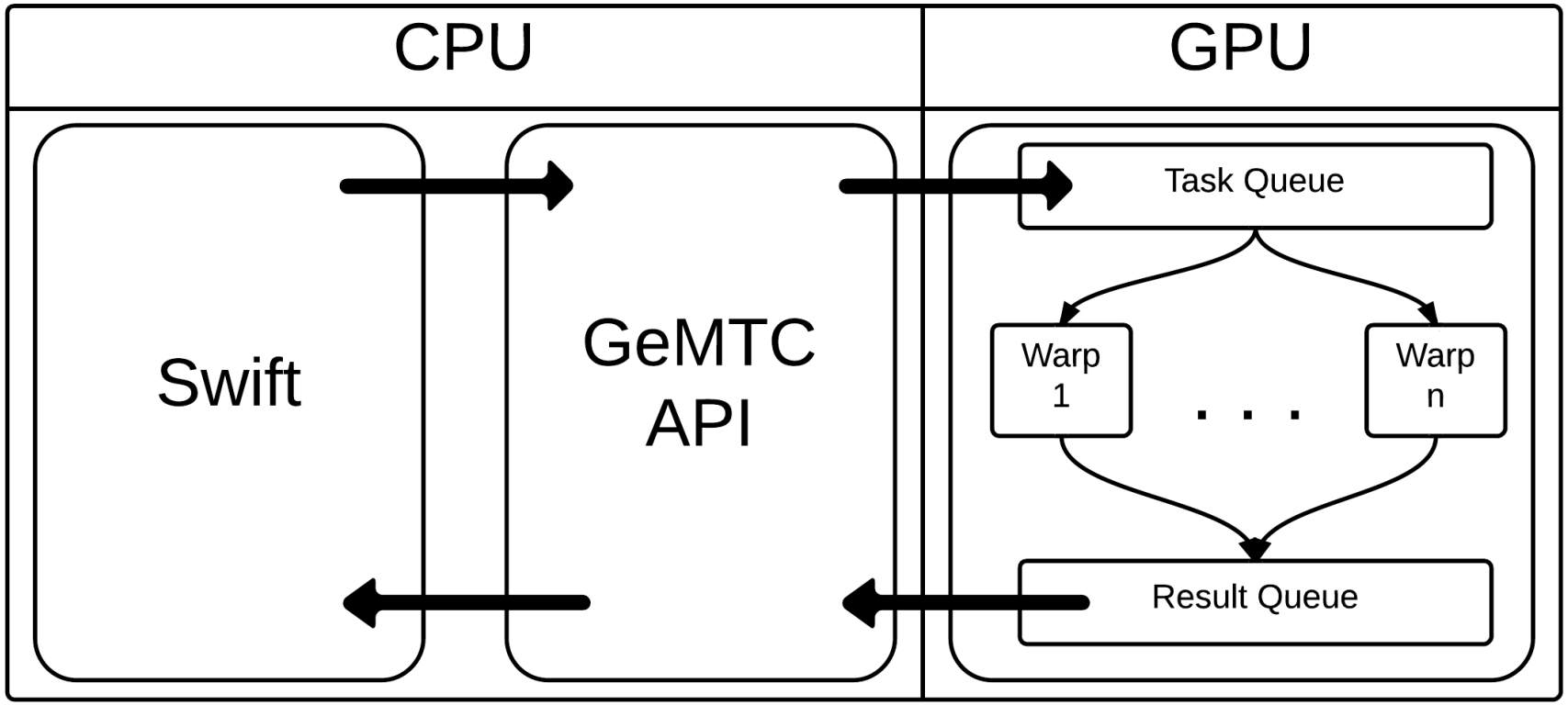
5) Increase concurrency from 15 to 192 (~13x)

Approach:

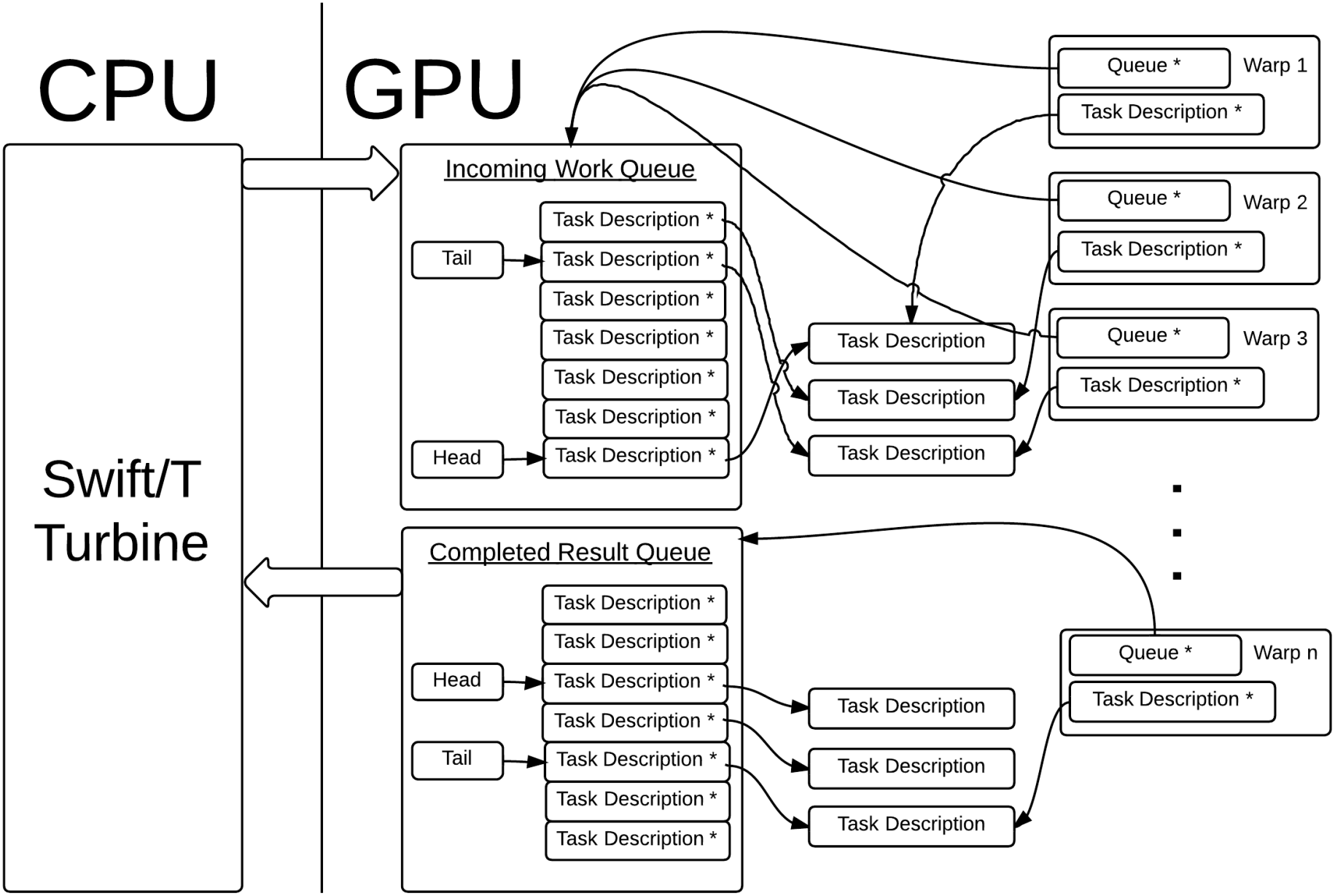
Design & implement GeMTC middleware:

1) Manages GPU 2) Spread host/device

3) Workflow system integration (with Swift/T)



GeMTC Architecture



GeMTC API

Device Management

- gemtcSetup()
- gemtcCleanup()

Data Movement

- gemtcMemcpyDevToHost()
- gemtcMemcpyHostToDev()

Task Management

- gemtcPush()
- gemtcPoll()

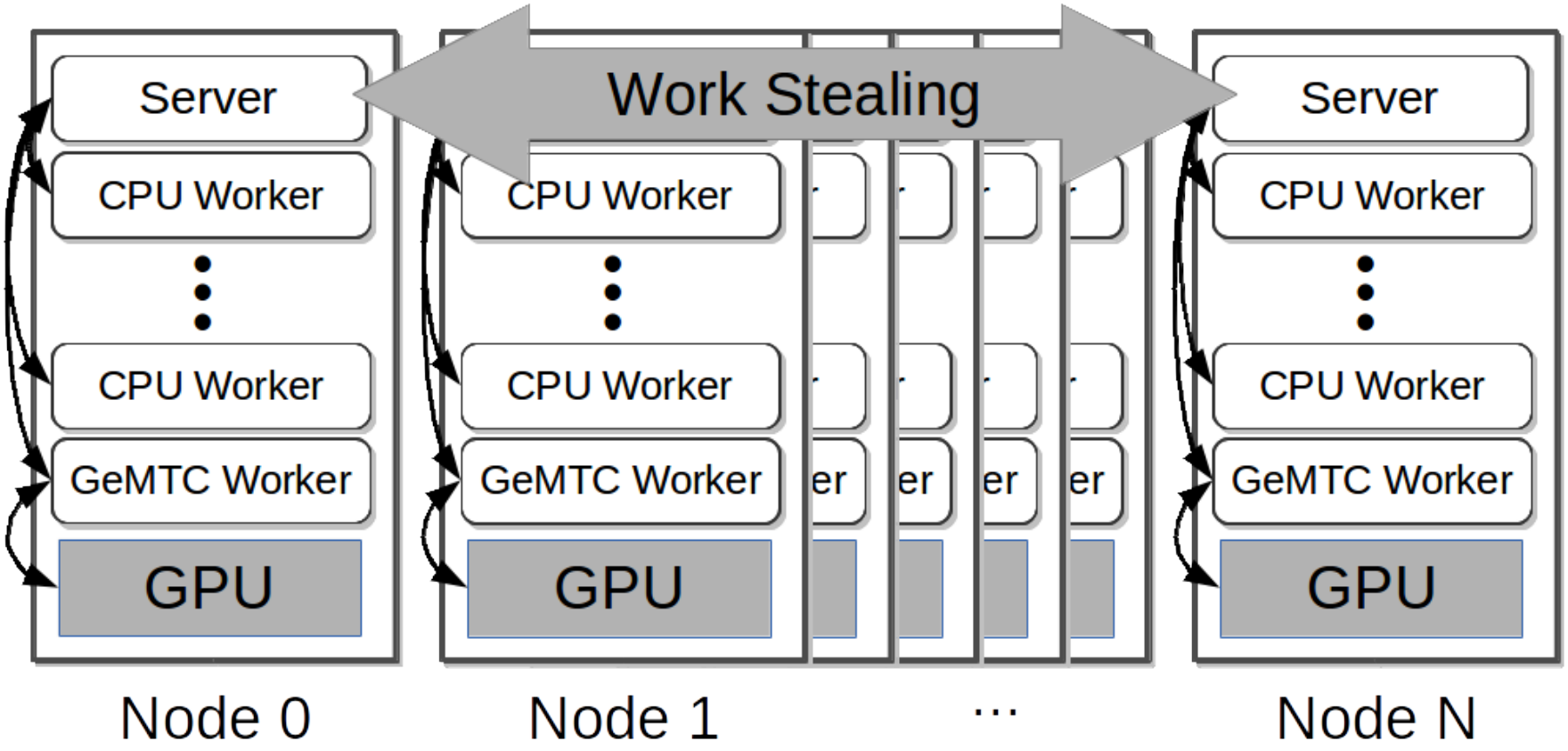
Memory Management

- gemtcGPUMalloc()
- gemtcGPUFree()

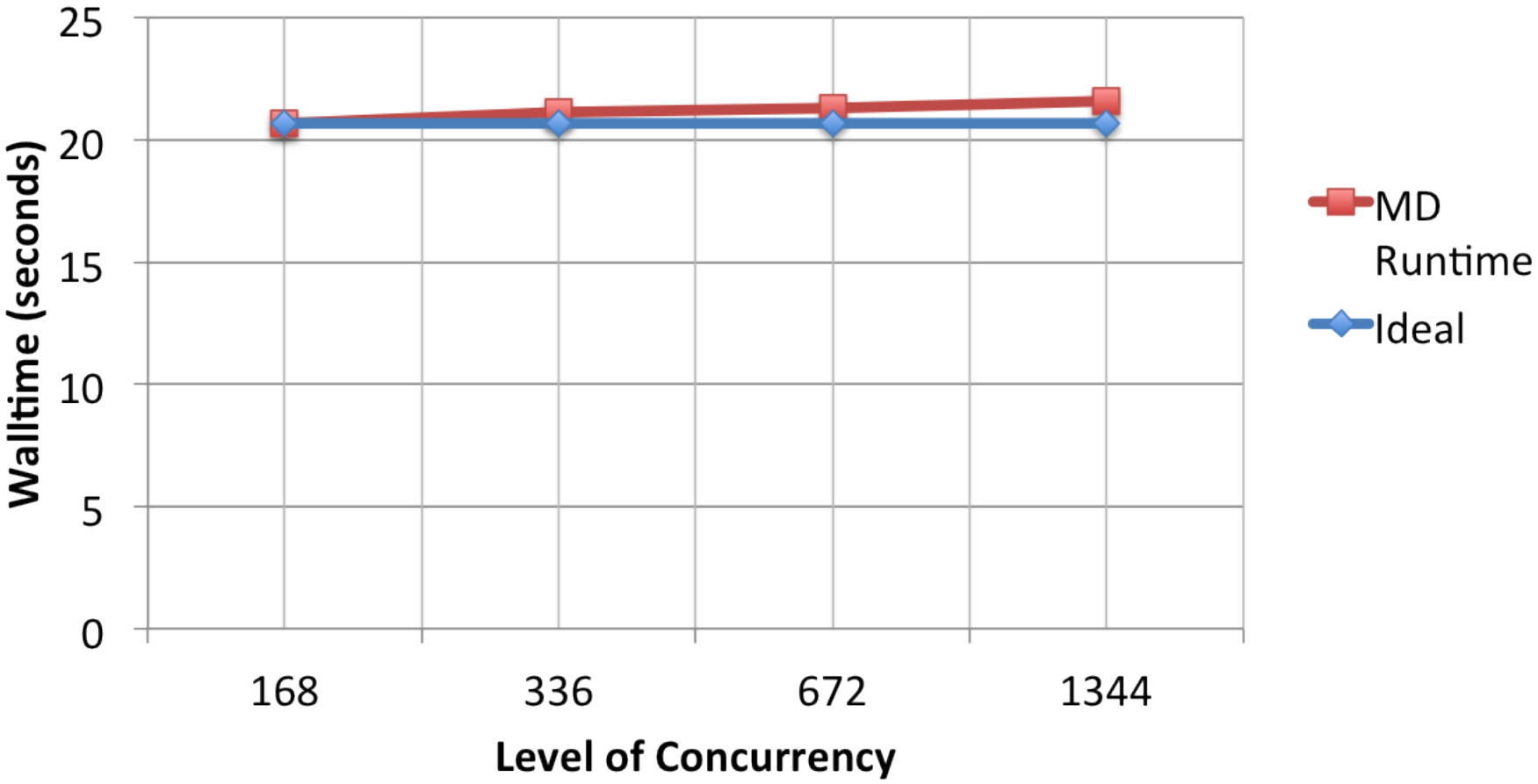
GeMTC AppKernels

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

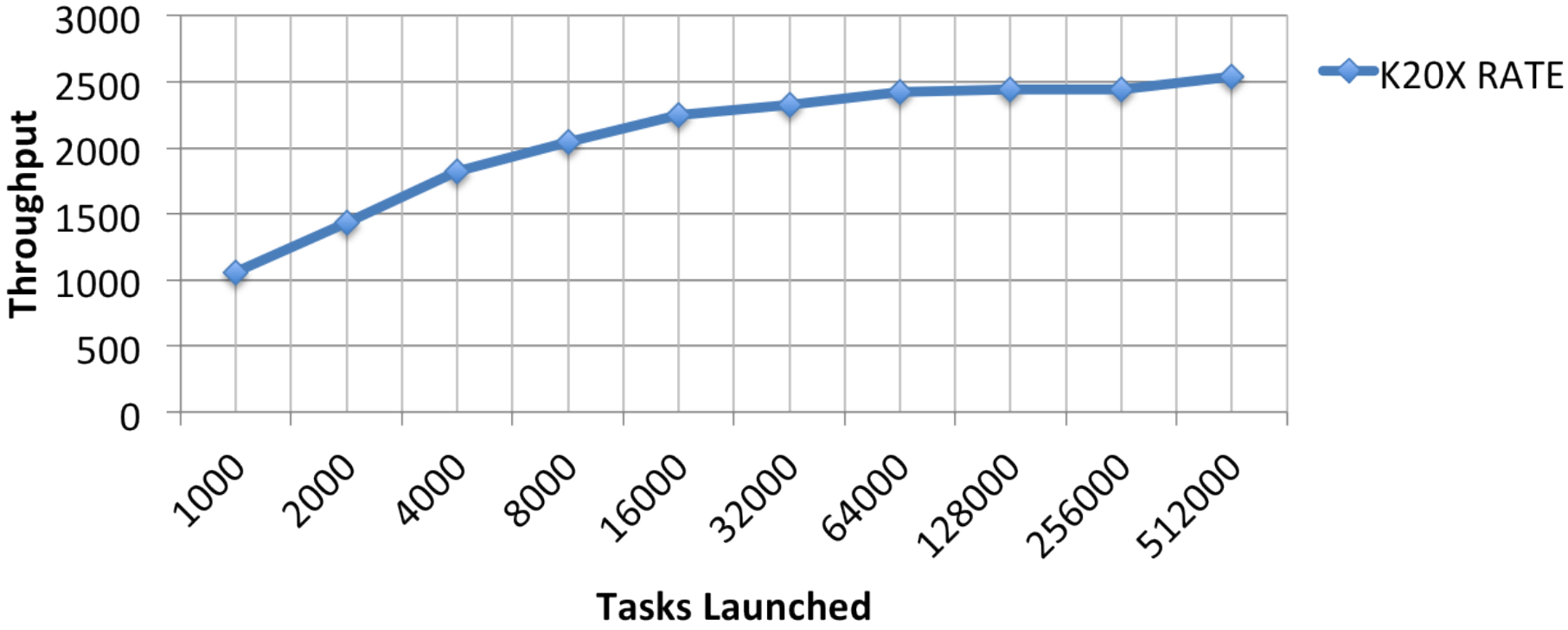
Swift/T + GeMTC Node Layout



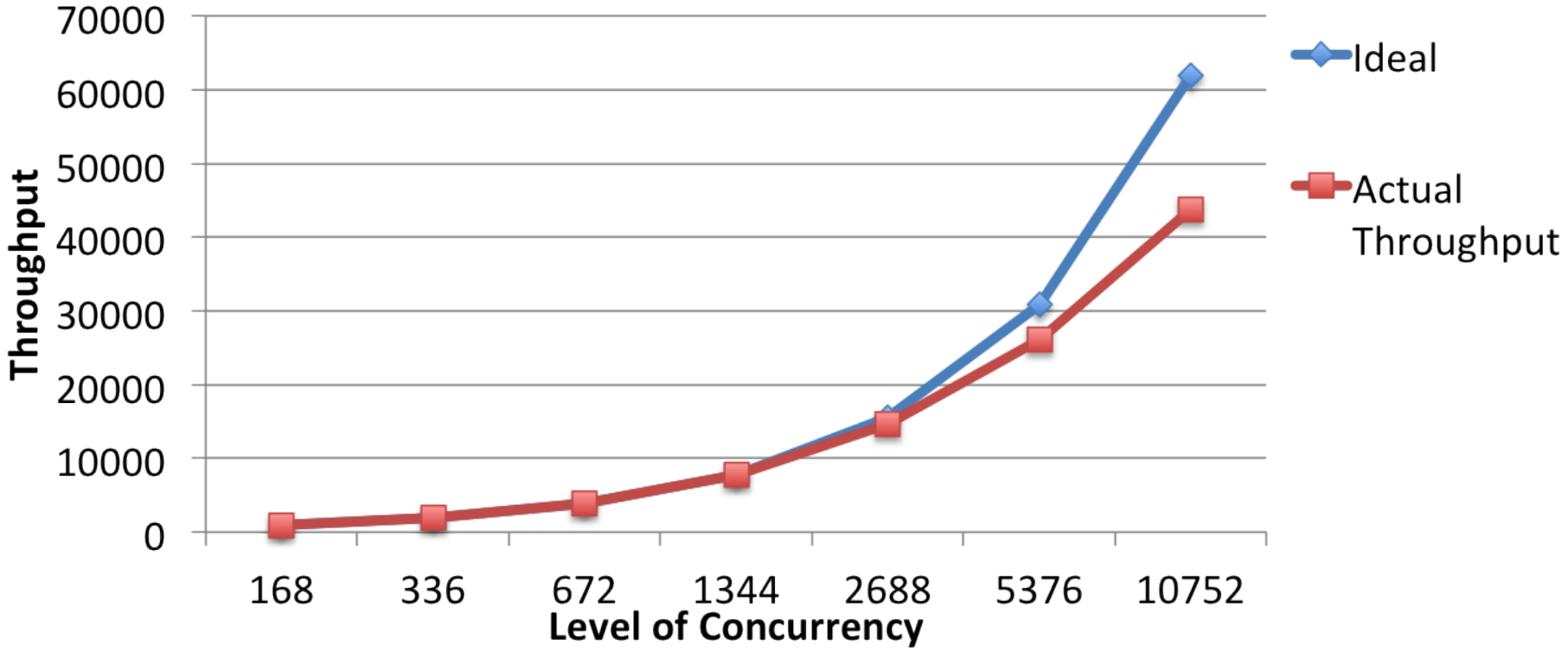
Multi Node Scaling (MD proxy app)



GeMTC throughput on Blue Waters

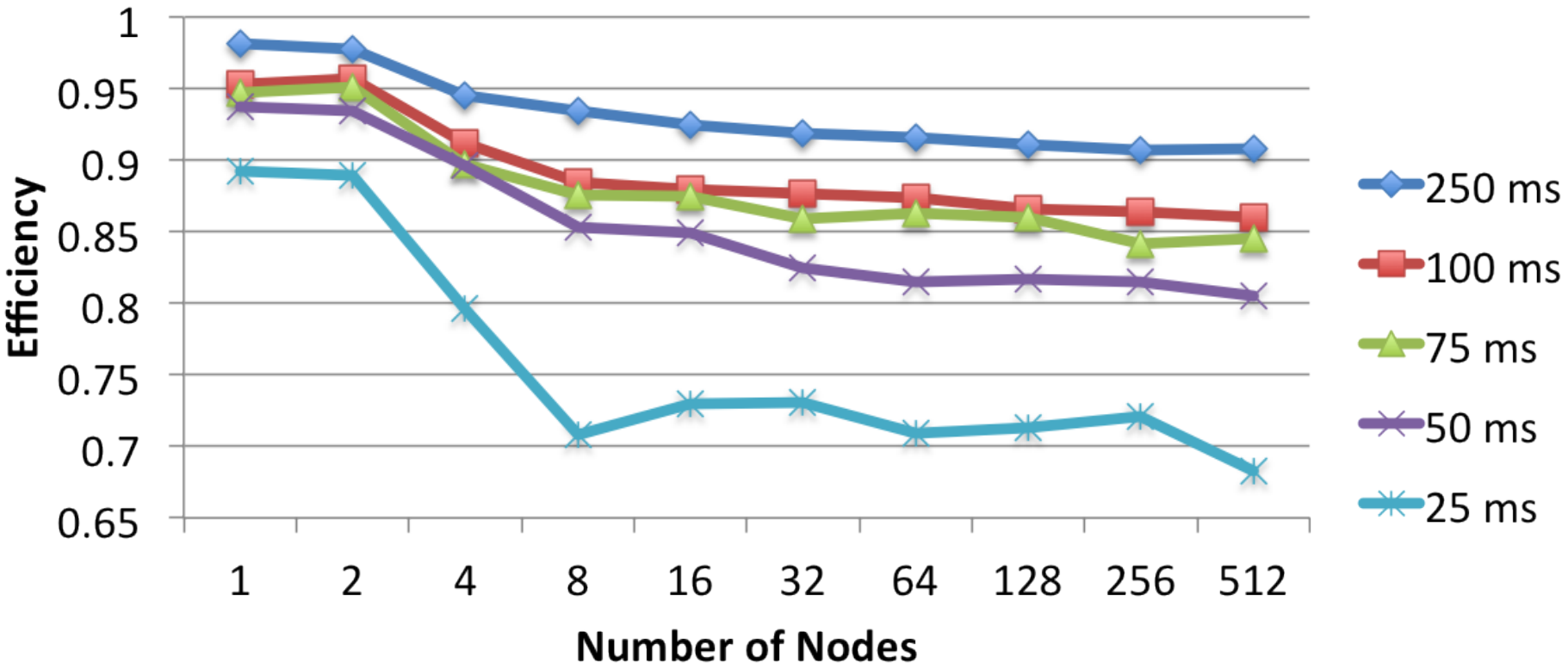


GeMTC + Swift Throughput 10K GPU Workers



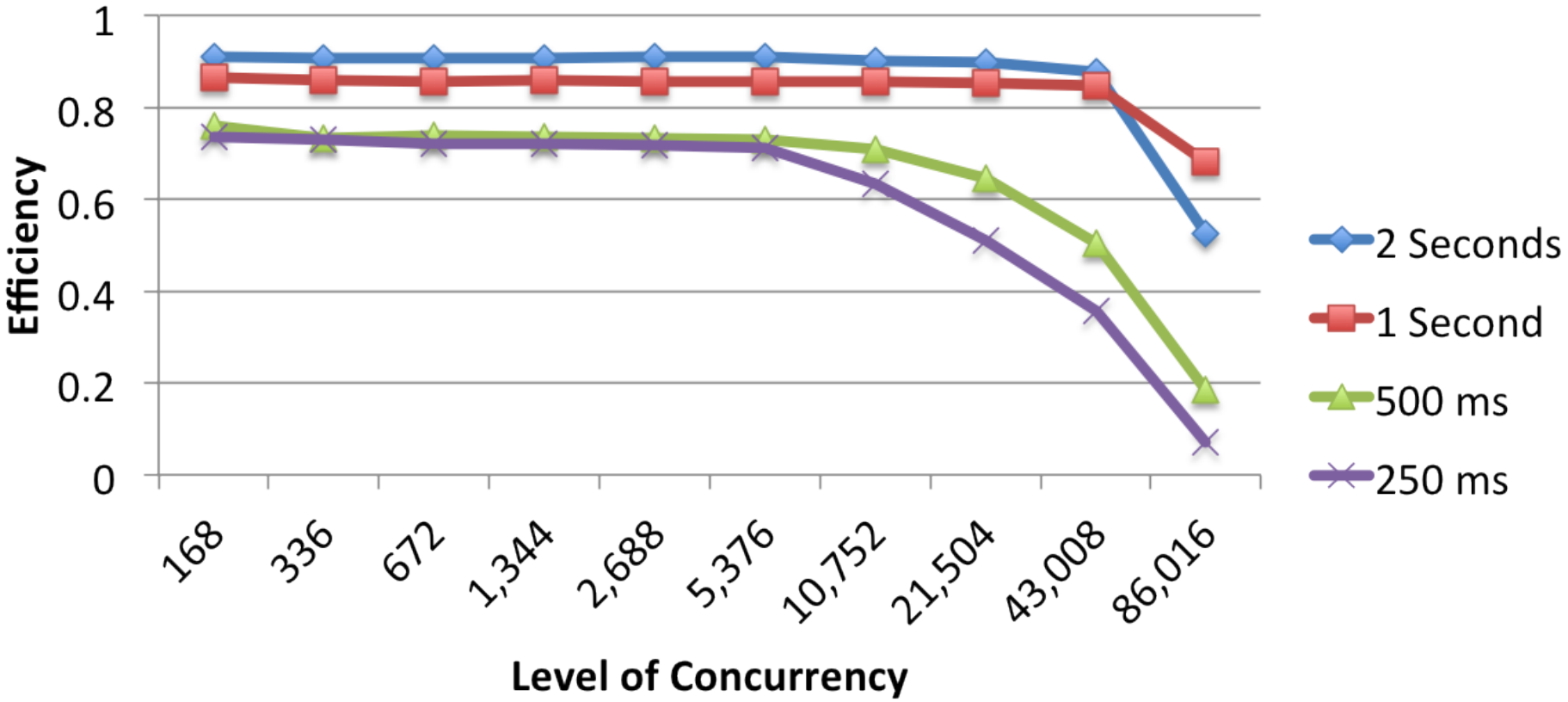
GeMTC Efficiency 86K GPU Workers

One worker per GPU



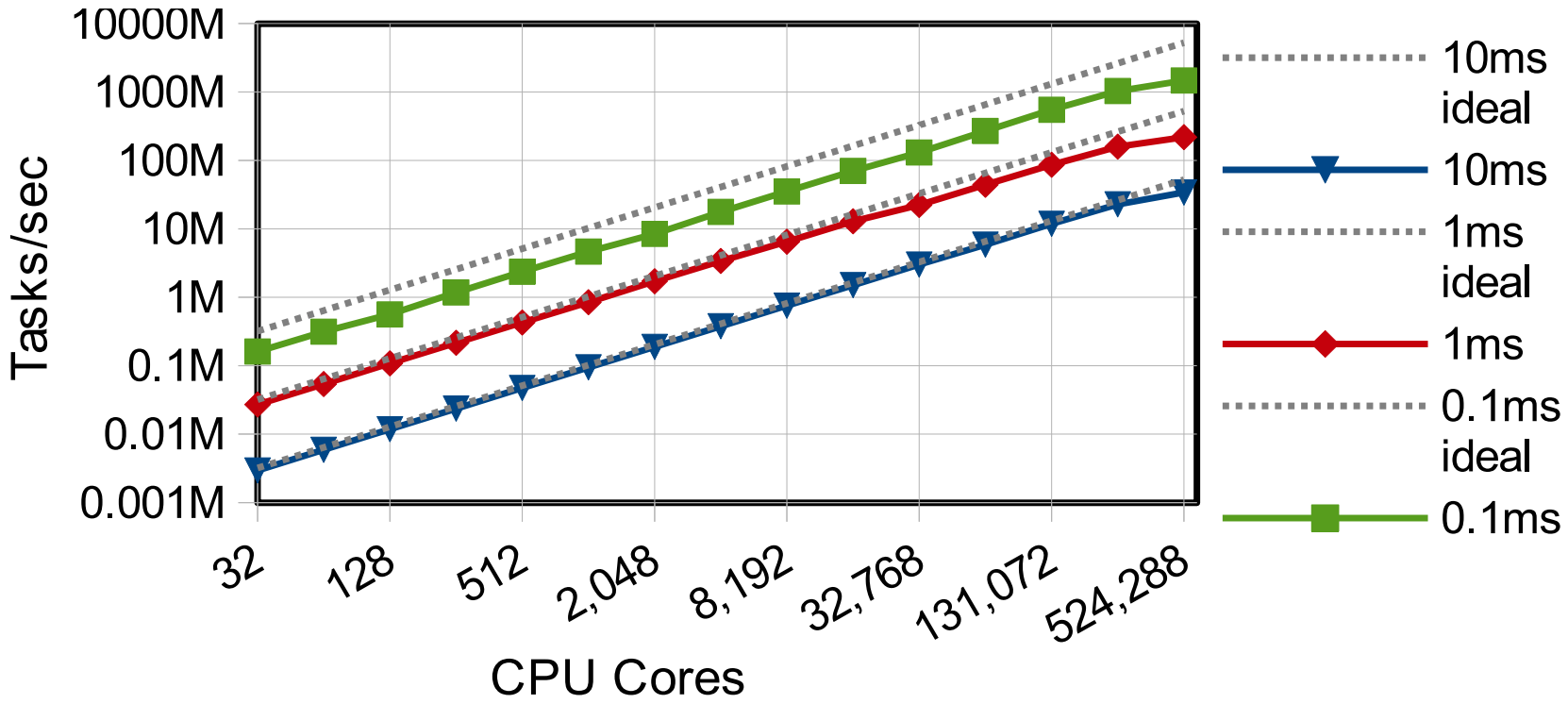
GeMTC Efficiency: 86K GPU Workers

168 active workers per GPU



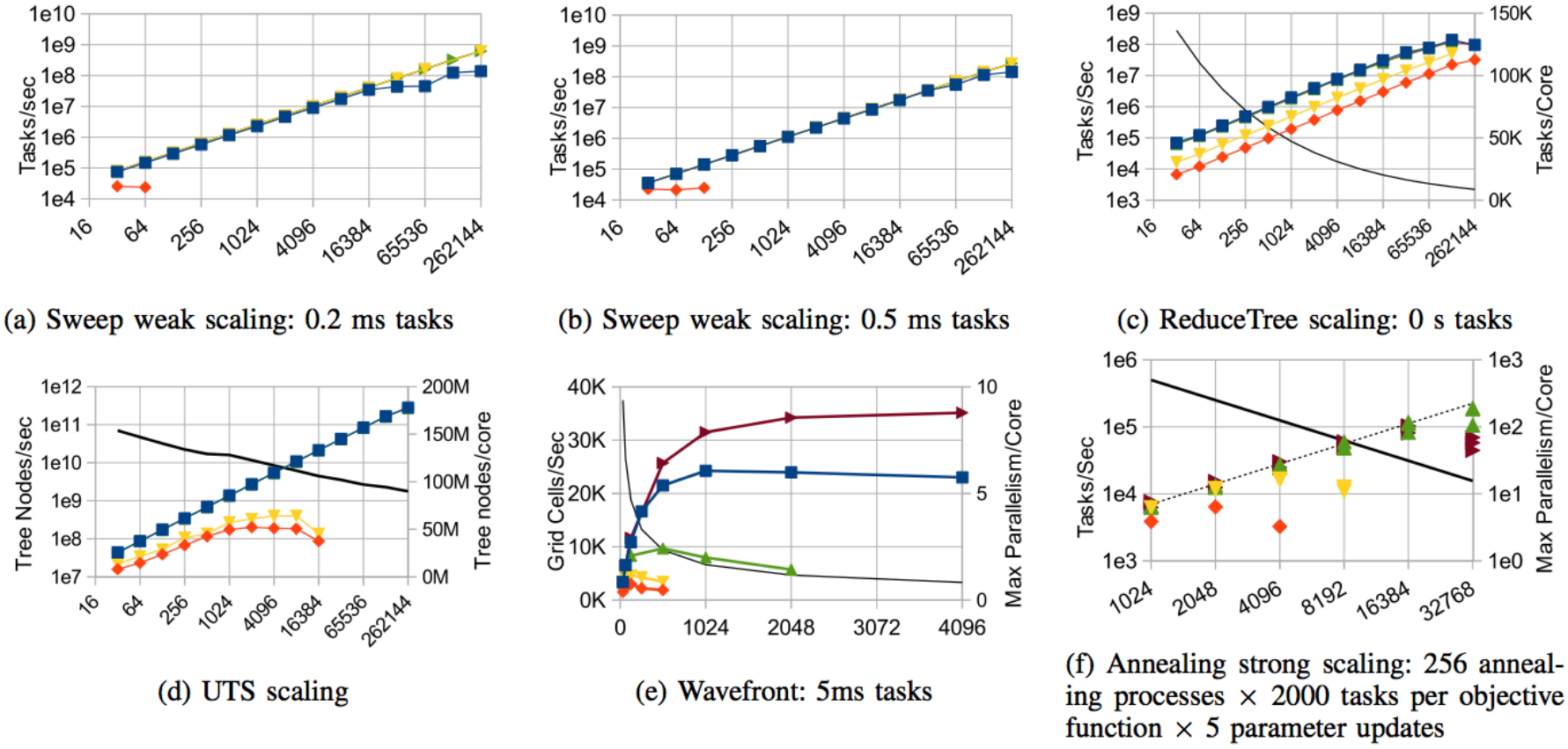
Swift/T: scaling of trivial foreach { } loop

100 microsecond to 10 millisecond tasks on up to 512K integer cores of Blue Waters



Advances in compiler and runtime optimization enable Swift to be applied in new in-memory programming models.
http://people.cs.uchicago.edu/~tga/pubs/stc-preprint-apr14.pdf

Swift/T application benchmarks on Blue Waters



(a) Sweep weak scaling: 0.2 ms tasks

(b) Sweep weak scaling: 0.5 ms tasks

(c) ReduceTree scaling: 0 s tasks

(d) UTS scaling

(e) Wavefront: 5ms tasks

(f) Annealing strong scaling: 256 annealing processes × 2000 tasks per objective function × 5 parameter updates

Fig. 10: Application speedup and scalability at different optimization levels. X axes show scale in cores. Primary Y axes show application throughput in application-dependent terms. Secondary Y axes show problem size or degree of parallelism where applicable.

Future Work

- Evaluate additional applications
  - Cancer gene detection, glass modelling
- Support other accelerators (Phi, AMD)
- Simplify the development of task kernels (OpenCL, OpenACC?)
- More efficient node utilization (CPU)
- Bond multiple warps for a single worker
- CUDA 6 Enhancements (Unified Memory…)

Code, docs and downloads

GeMTC:
http://datasys.cs.iit.edu/projects/GeMTC
https://github.com/skrieder/gemtc

Swift:
http://swift-lang.org

Swift/T and Turbine:
http://mcs.anl.gov/exm/local/guides/swift.html

Publications

- Scott J. Krieder, Justin M. Wozniak, Timothy Armstrong, Michael Wilde, Daniel S. Katz, Benjamin Grimmer, Ian T. Foster, Ioan Raicu. “ Design and Evaluation of the GeMTC Framework for GPU-enabled Many-Task Computing”, HPDC'14
- Benjamin Grimmer, Scott Krieder, Ioan Raicu. "Enabling Dynamic Memory Management Support for MTC on NVIDIA GPUs", EuroSys 2013 (poster)
- Scott J. Krieder, Ioan Raicu, “ Towards the Support for Many-Task Computing on Many Core Computing Platforms” - IEEE/ACM Supercomputing 2012 (SC’12) - Salt Lake City, UT (11/2012)
- Scott J. Krieder, Ioan Raicu - “Early Experiences in running Many-Task Computing workloads on GPUs” - XSEDE 2012 - Chicago, IL (07/2012)
- Scott J. Krieder, Ioan Raicu - “An Overview of Current and Future Accelerator Architectures” - Greater Chicago Area System Research Workshop - Chicago, IL (05/2012)

Conclusions

- More efficient MTC on NVIDIA GPUs
- MPMD on SIMD
- Evaluated synthetic benchmarks and proxy molecular dynamics app on Blue Waters
- Integrated GeMTC with a higher-level programming model (Swift's implicitly parallel functional dataflow)
- Achieved new scaling breakthroughs for an implicitly-parallel programming model