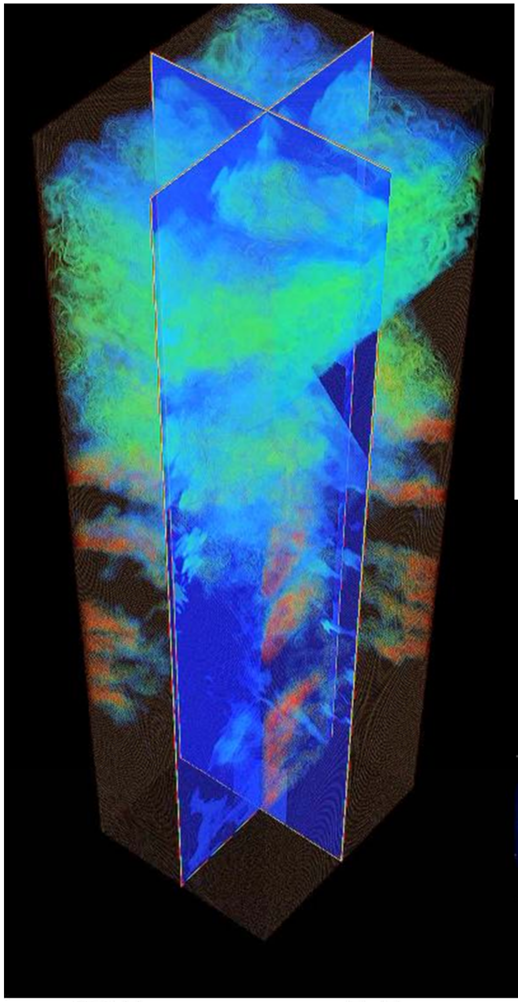
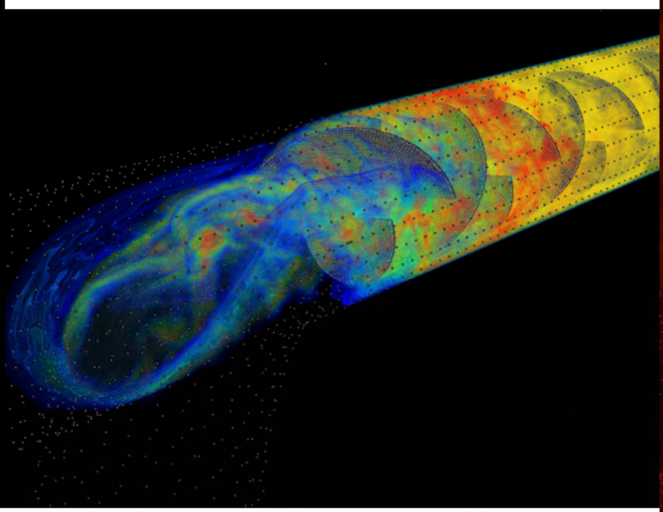
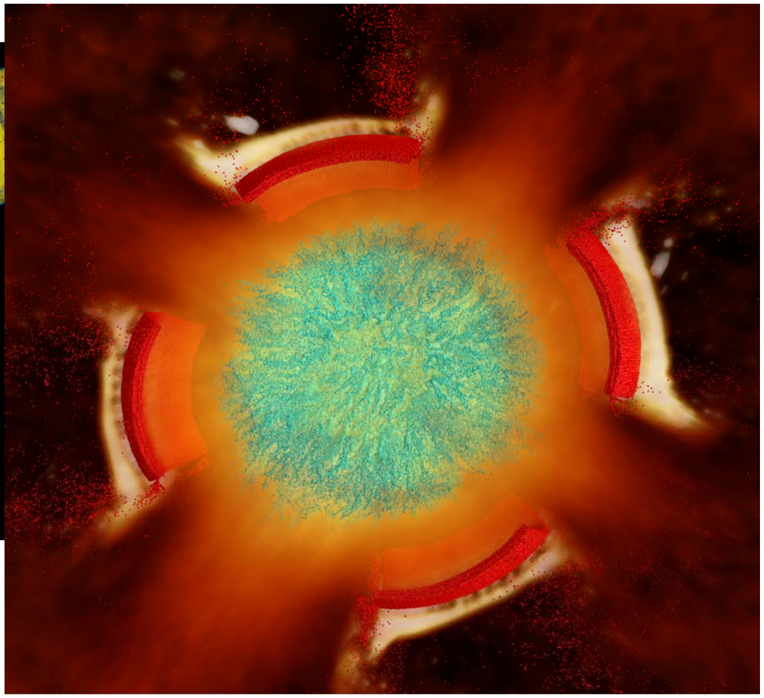




The Uintah Framework: A Unified Heterogeneous Task Scheduling and Runtime System



Qingyu Meng, Alan Humphrey, Martin Berzins

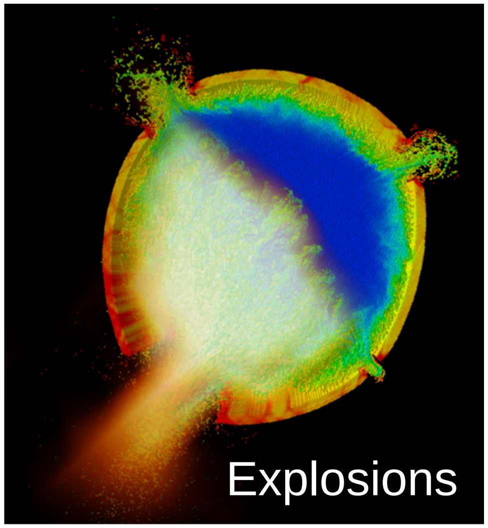
Thanks to: John Schmidt and J. Davison de St. Germain, SCI Institute
Justin Luitjens and Steve Parker, NVIDIA

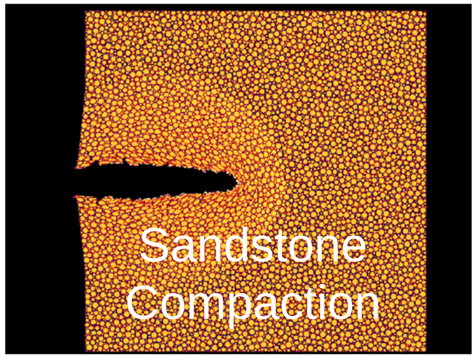
SCI INSTITUTE

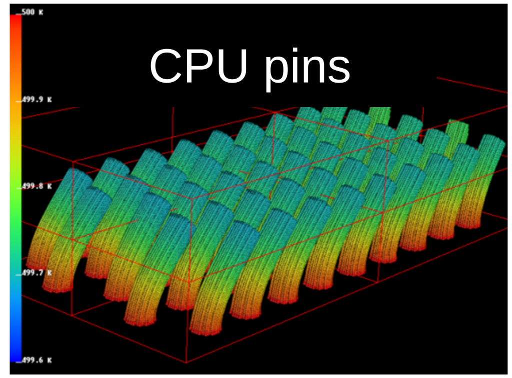
DOE for funding the CSAFE project (97-10), DOE NETL, DOE NNSA
NSF for funding via SDCI and PetaApps

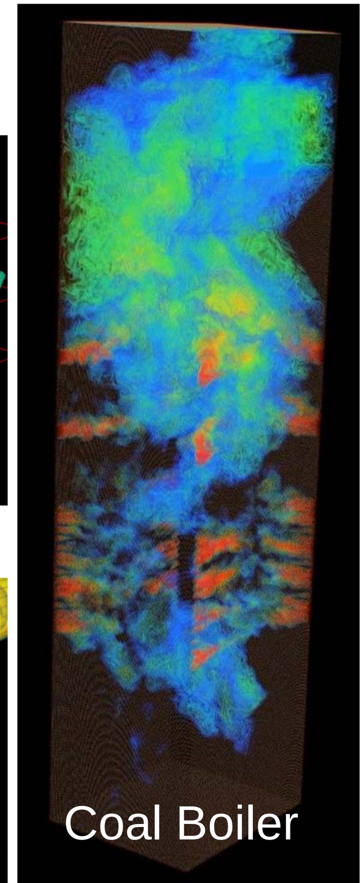
THE UNIVERSITY OF UTAH

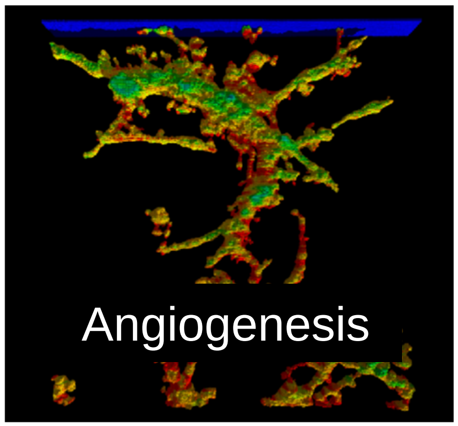
Current and Past Uintah Applications

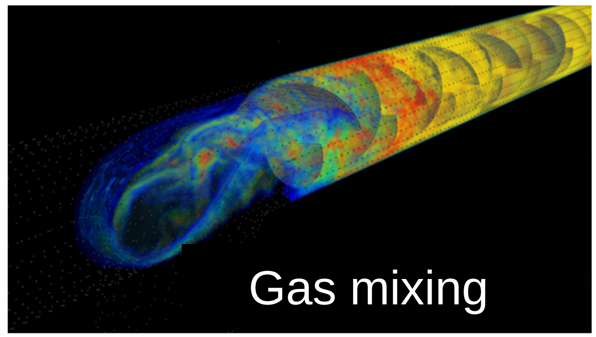

Explosions


Sandstone Compaction

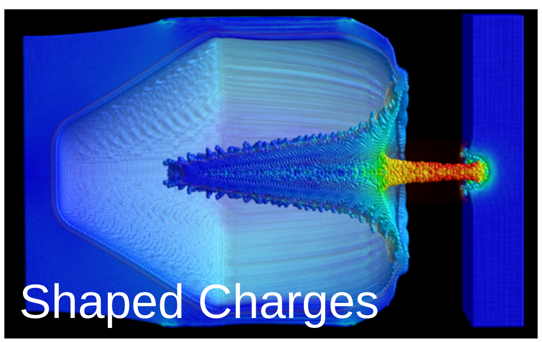

CPU pins

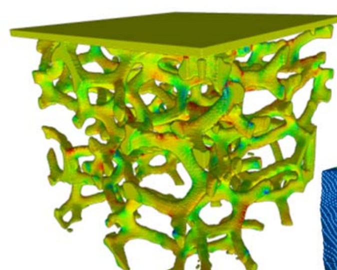

Coal Boiler

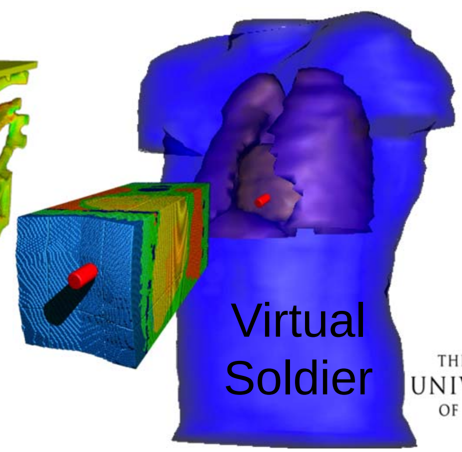

Angiogenesis


Gas mixing


Fires


Shaped Charges

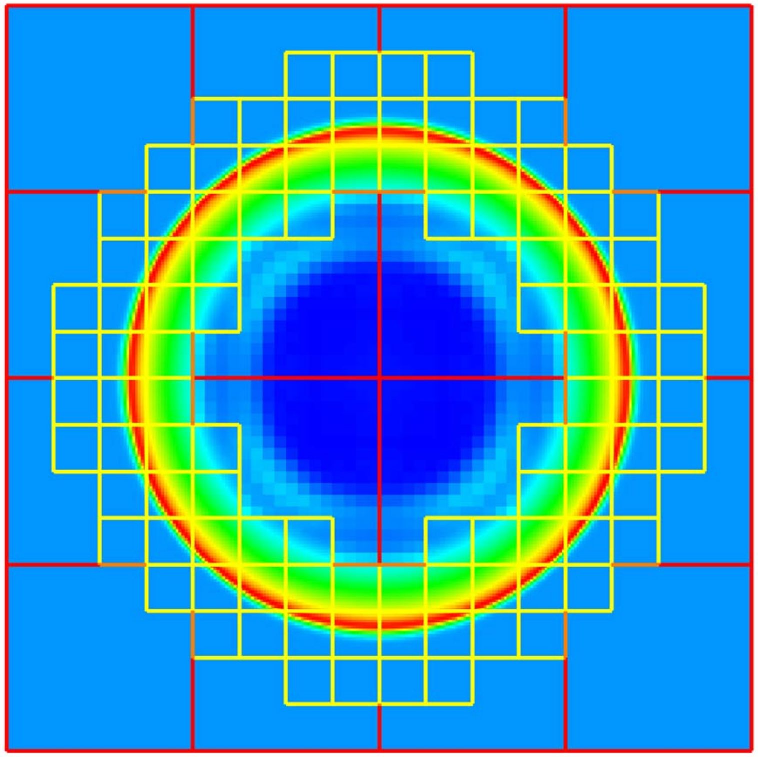

Foam Compaction


Virtual Soldier

THE UNIVERSITY OF UTAH

Uintah Data Parallelism

Uintah uses both data and task parallelism



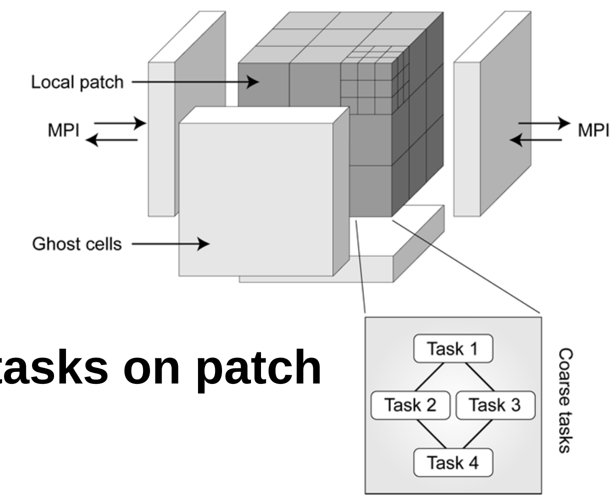
Gird and Patches
(Physical Domain)

- Structured Grid(Flows) + Particles System(Solids)
- Patch-based Domain Decomposition for Parallel Processing
- Adaptive Mesh Refinement
- Dynamic Load Balancing
  - Profiling + Forecasting Model
  - Parallel Space Filling Curves
  - Data Migration

THE UNIVERSITY OF UTAH

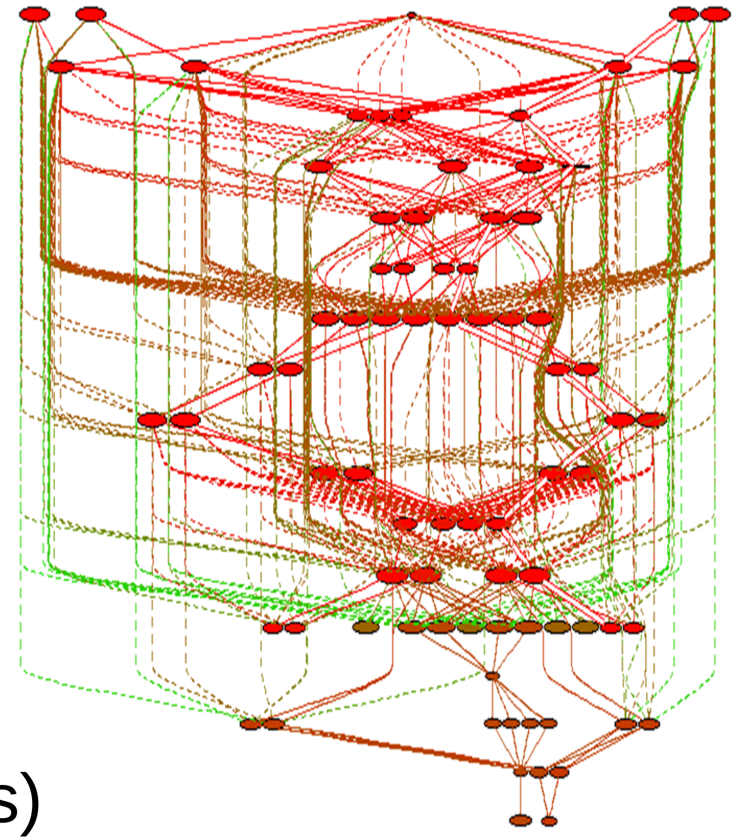
Uintah Task Parallelism and Uintah Task Graph

Patch-based domain decomposition



tasks on patch

Compile



Task Graph

- User defines Uintah Tasks:
  - **Serial** code (call back functions)
  - **Input** and **output** variables
- **Distributed**: only creates tasks on local patches
- Framework analyzes task dependencies and creates TG
  - **Automatic** MPI message generation
  - **Dynamic** Task Execution (Data Driven Overlap)

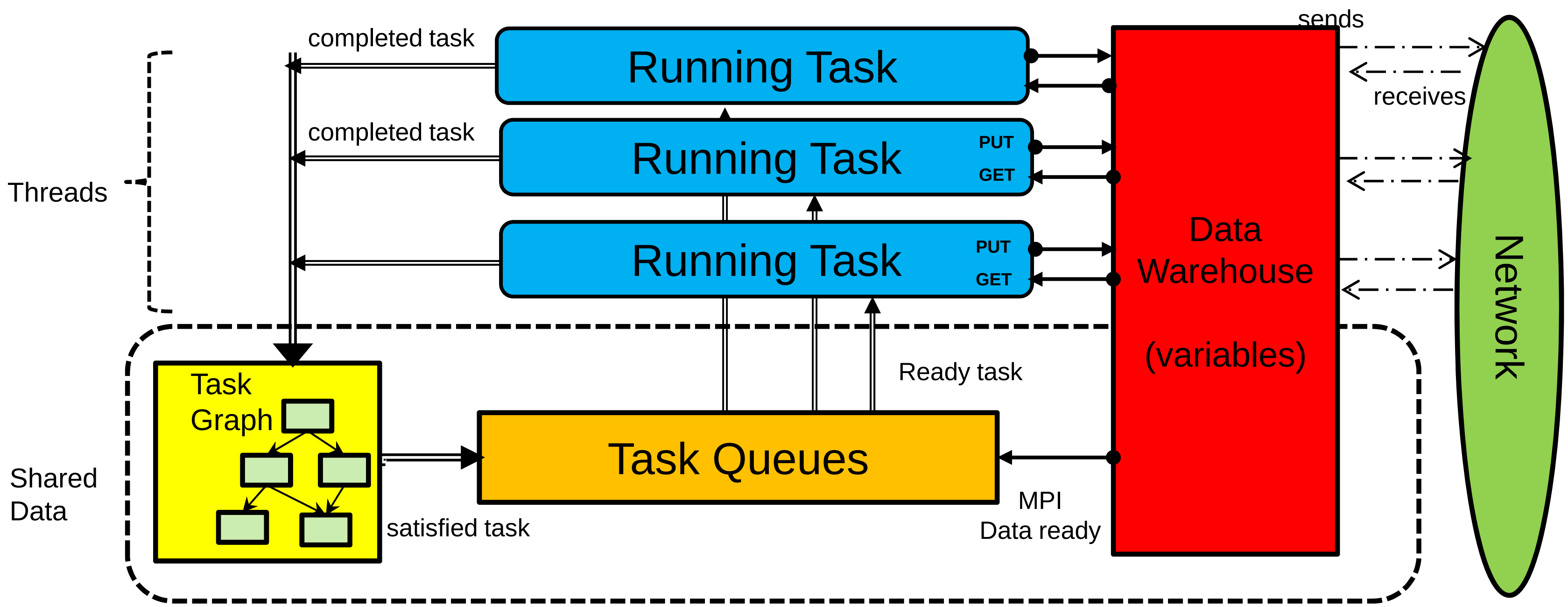
THE UNIVERSITY OF UTAH

Uintah Runtime System: How Uintah Runs Tasks

- **Memory Manager**: Uintah Data Warehouse (DW)
  - Variable dictionary (hashed map from: Variable Name, Patch ID, Material ID keys to memory)
  - Provide interfaces for tasks to
    - Allocate variables
    - Put variables into DW
    - Get variables from DW
  - **Automatic Scrubbing** (garbage collection)
  - **Checkpointing & Restart** (data archiver)
- **Task Manager**: Uintah schedulers
  - Decides when and where to run tasks
  - Decides when to process MPI

THE UNIVERSITY OF UTAH

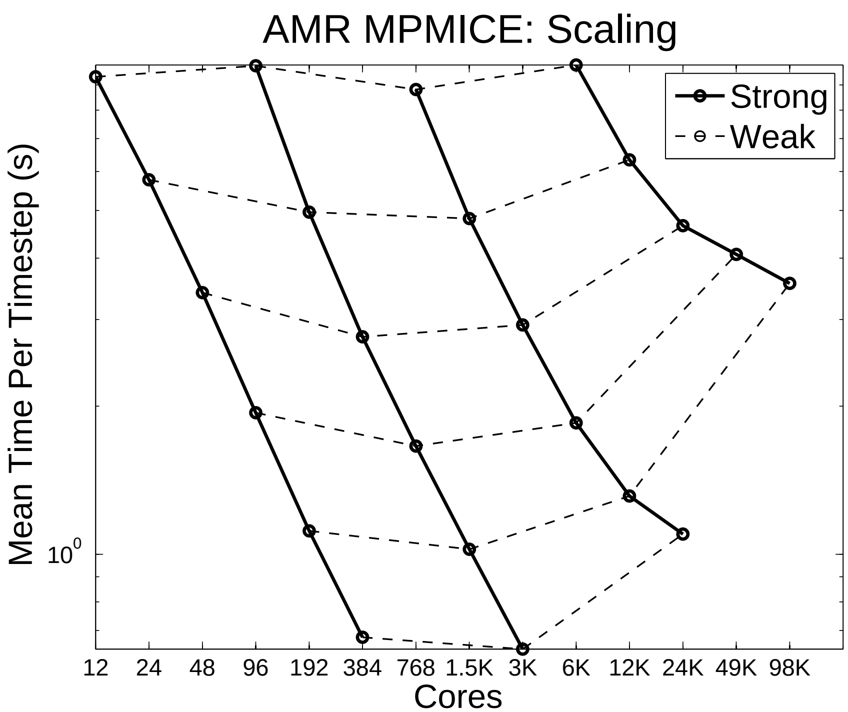
Thread/MPI Scheduler (De-centralized)



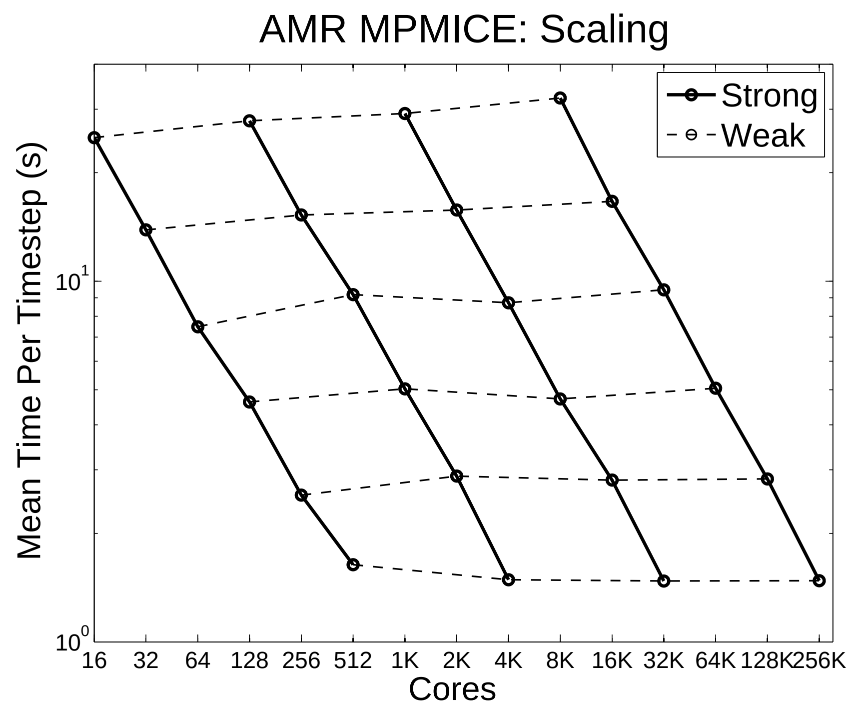
- **Memory saving:** reduce ghost copies and metadata
- **Work stealing** inside node: all threads directly pull tasks from task queues, no on-node MPI
- **Full Overlapping**: All threads process MPI sends/receives and execute tasks
- Use **lock-free** data structure (avoid locking overhead)

THE UNIVERSITY OF UTAH

Scalability Improvement



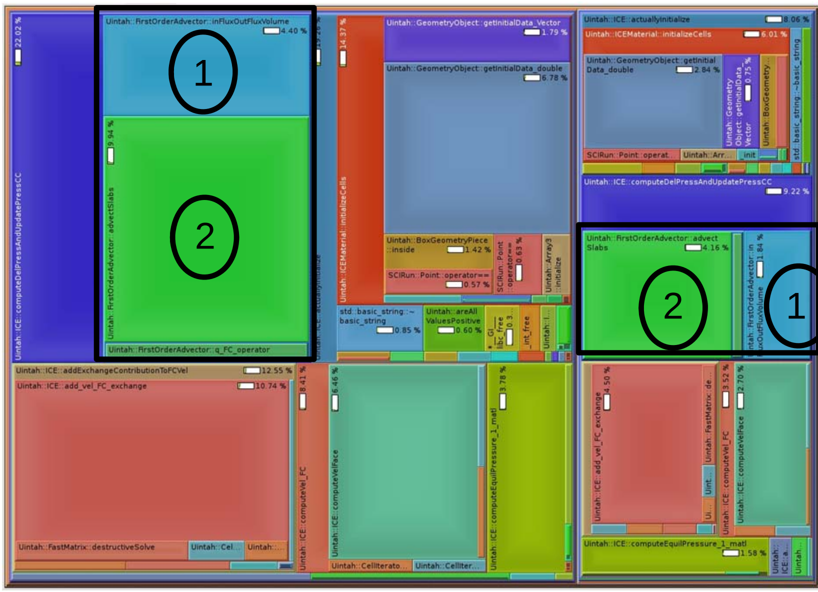
Original Dynamic MPI-only Scheduler



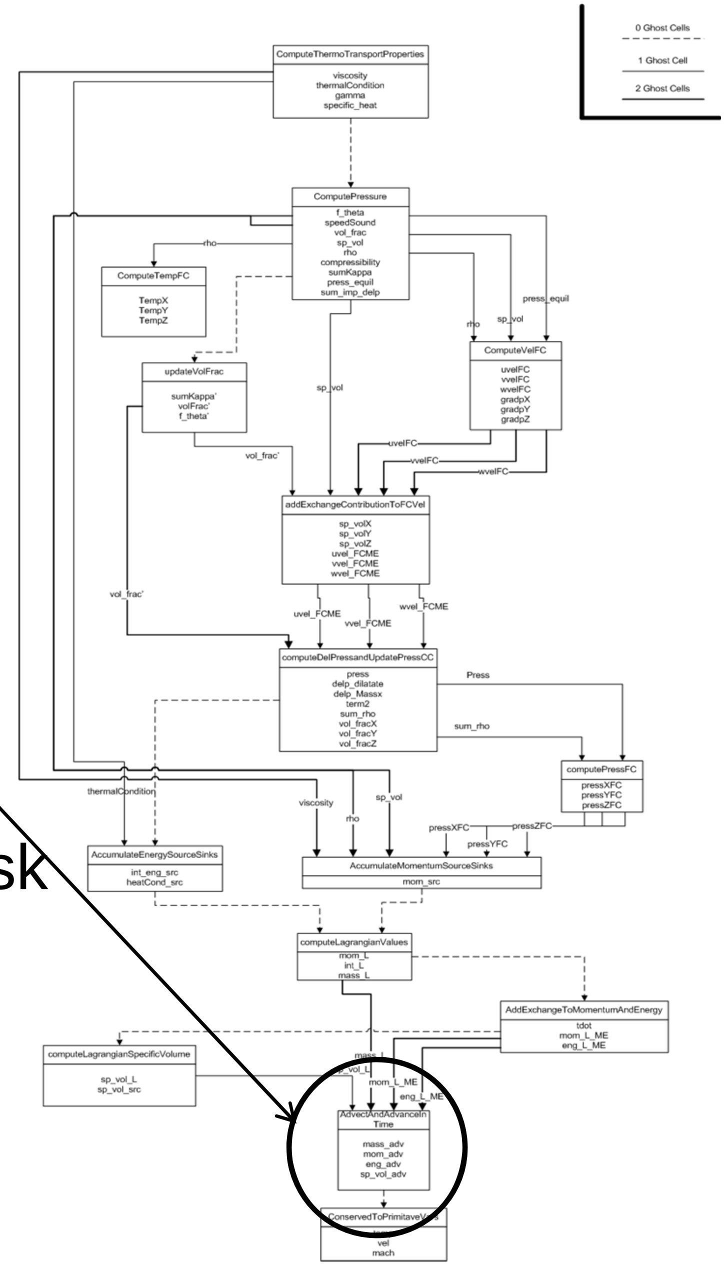
De-centralized MPI/Thread Hybrid Scheduler
(with Lock-free Data Warehouse)

- Achieve much better CPU Scalability
- 95% weak scaling efficiency on 256K cores (Jaguar XK6)
- Use GPUs to accelerate Uintah Components

First step to GPU



Generated by Google profiling tool, visualized by Kcachegrind



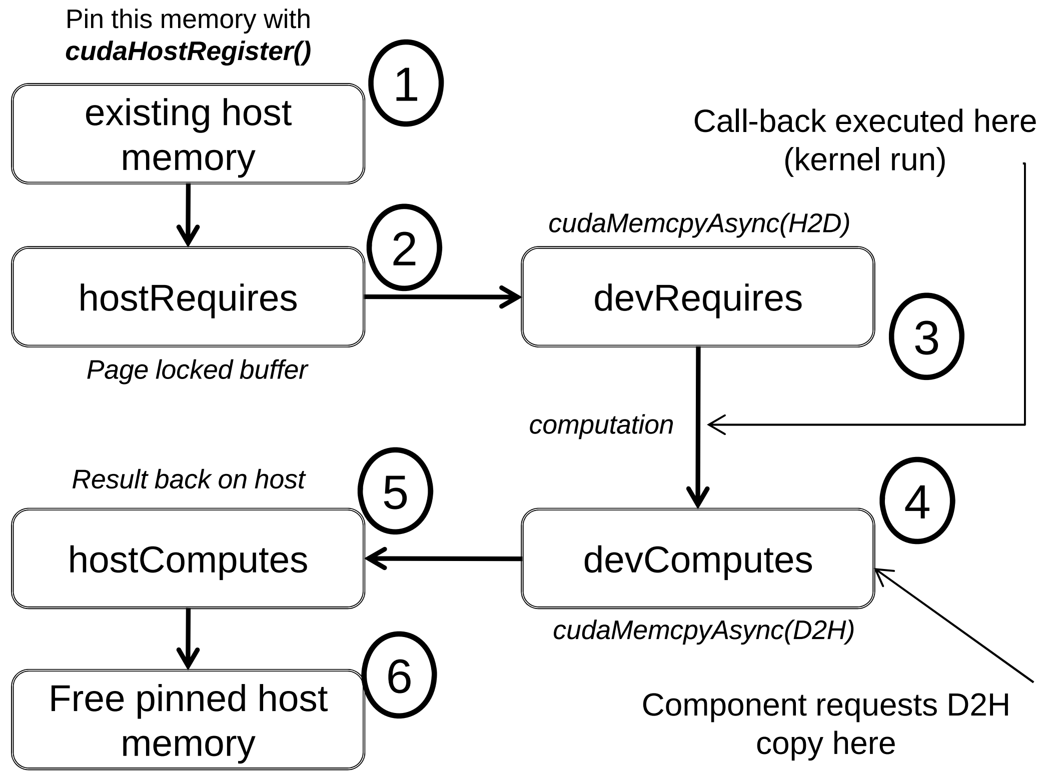
- **Profile** & find most time consuming task
- **Port** task's serial CPU code to GPU
- Call CUDA API **inside task** code
- Framework **unaware** of GPU(s)
- **Result**: ~2x speedup (stencil code)
  - must hide PCIe latency

THE UNIVERSITY OF UTAH

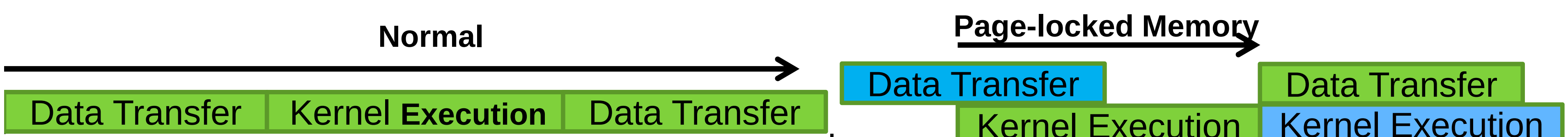
Uintah GPU Task Management

Framework manages all CUDA data movement (NOT inside task)

- Use ***Asynchronous API***
- **Automatically** generate CUDA stream for each dependency
- **Concurrently** execute kernels and memcopies
- **Prefetching** data before task kernel execute
- **Multi-GPU** support
- Two call back functions for both CPU version and GPU version: **Compatible** for non-GPU nodes

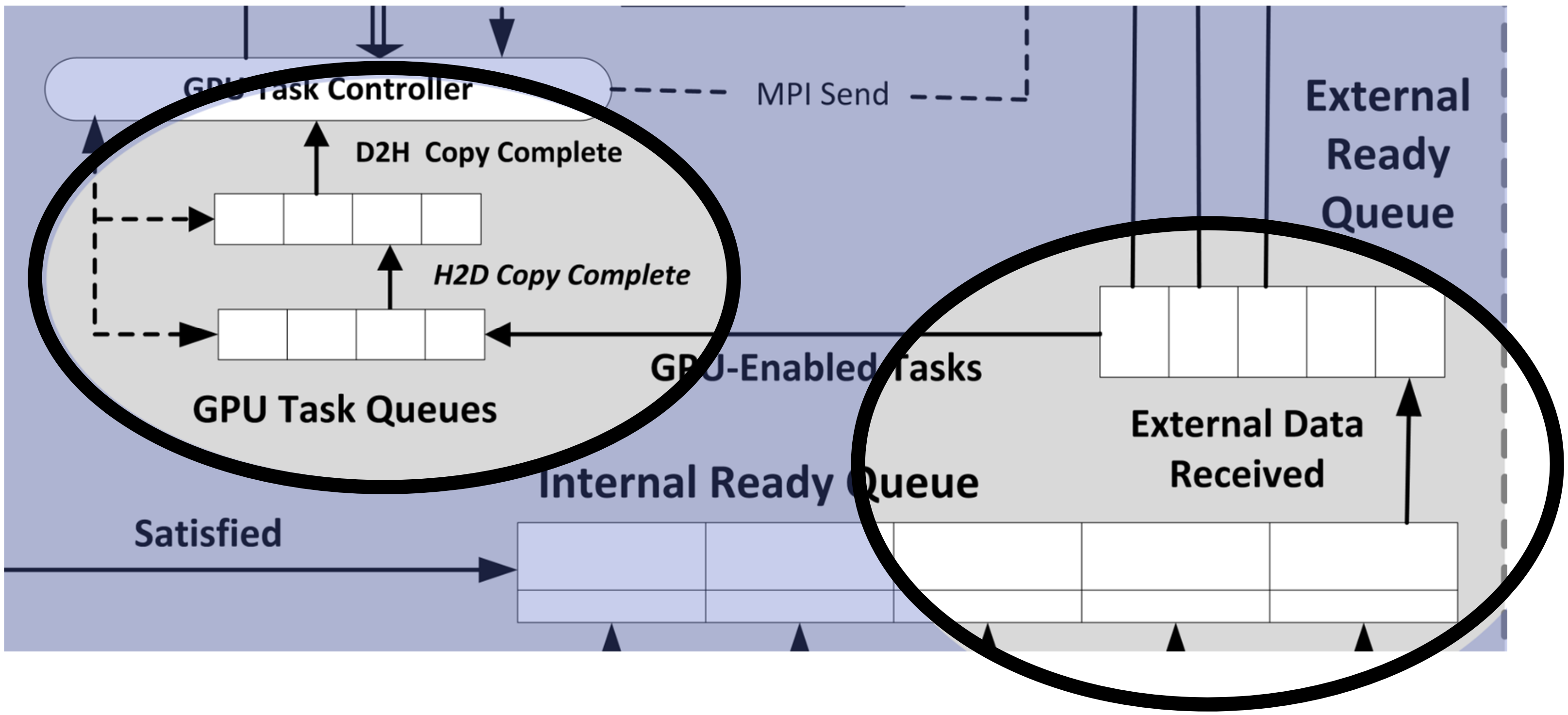


Stages of GPU task in Uintah runtime



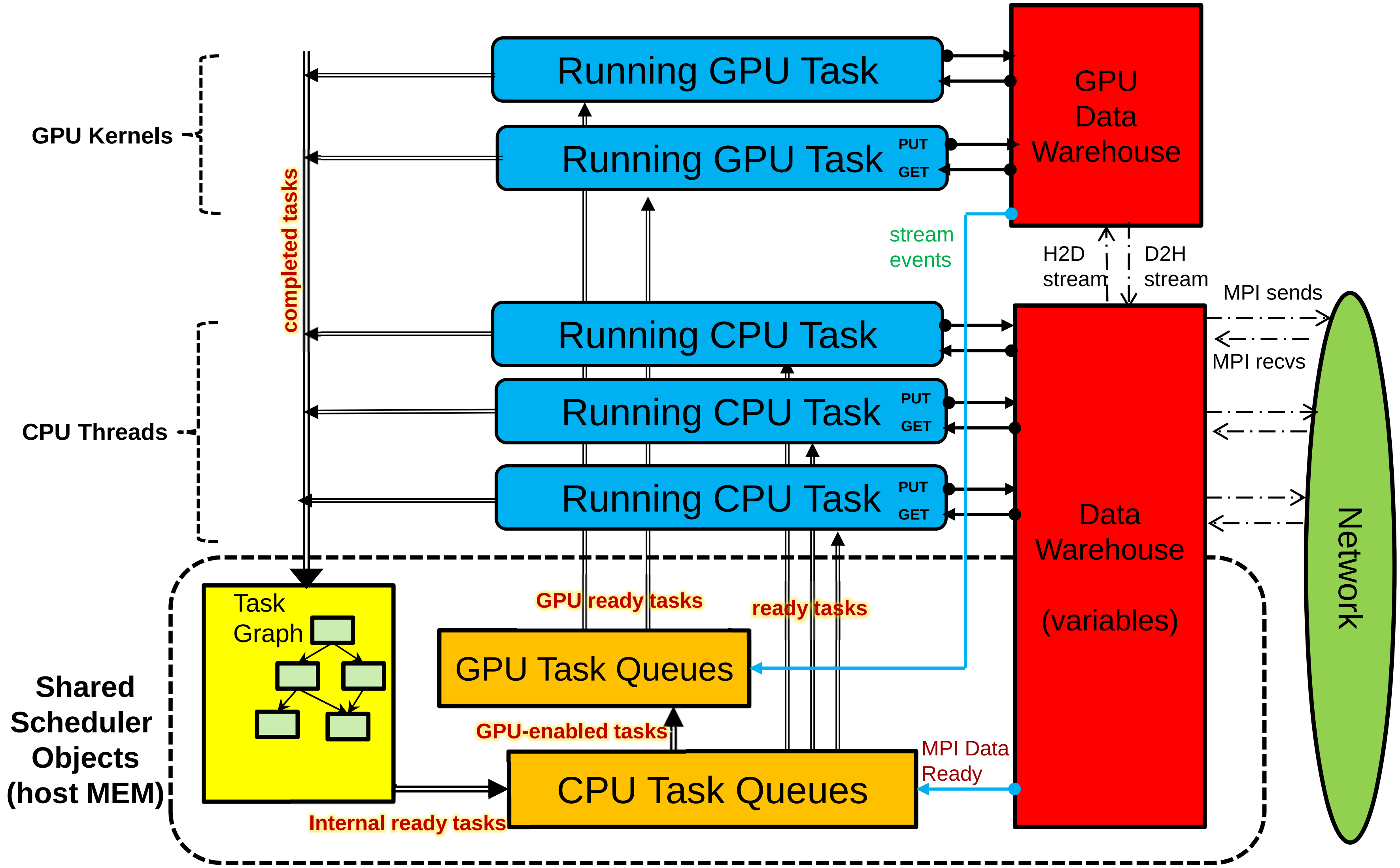
THE UNIVERSITY OF UTAH

Multistage Task Queues Architecture



Fully Overlap computation with PCIe transfers and MPI communication

Unified Heterogeneous Scheduler & Runtime



THE UNIVERSITY OF UTAH

GPU RMCRT Speedup Results
(Multi-Node)

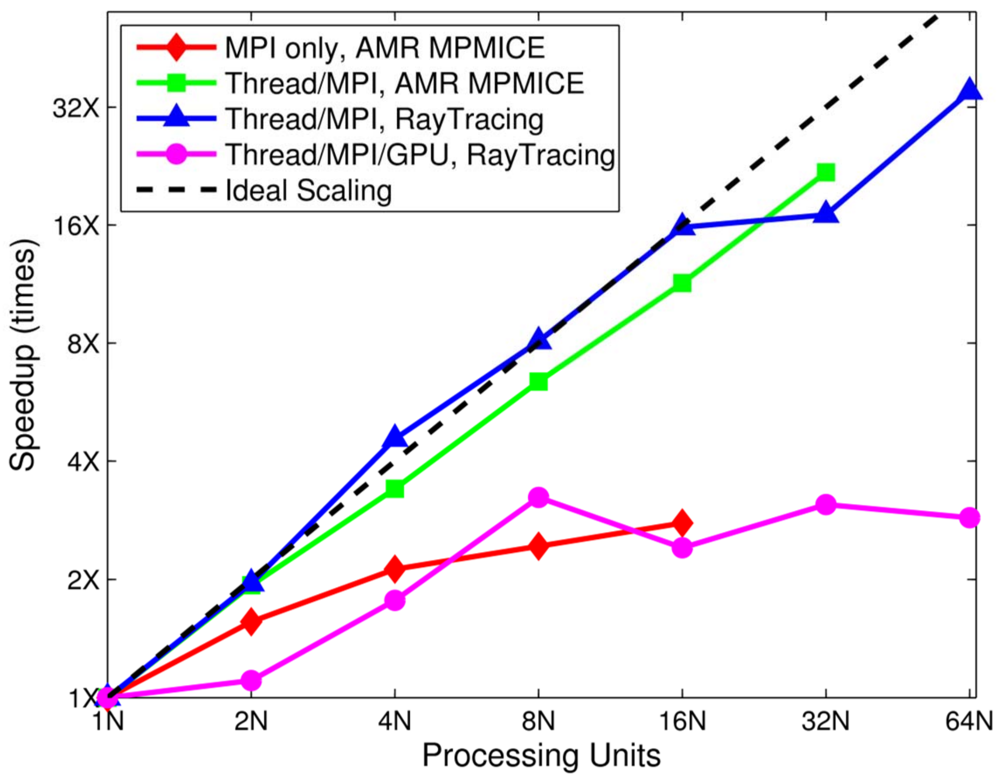
All CPU cores vs Single GPU				
	Nodes	**CPU(sec)**	**CPU+GPU (sec)**	**Speedup (x)**
Keeneland Initial Delivery System	1	319.769	8.49714	38 X
	2	163.773	7.68571	21X
	4	70.0943	4.80571	14X
	8	39.5686	2.62857	15X
	16	20.2414	3.52857	6X *
	32	18.8043	2.73857	6X *
	64	9.11571	2.95714	3X *

- **CPU Core – (2) Intel Xeon 6-core X5660 (Westmere) @2.8GHz**
- **GPU – (1) Nvidia M2090**

*** GPU implementation quickly runs out of work, scaling breaks down**

THE UNIVERSITY OF UTAH

Scaling Comparisons



- Uintah strong scaling results when using:
  - MPI-only
  - Multi-threaded MPI
  - Multi-threaded MPI w/ GPU
- Two Problems:
  - AMR MPMICE

    Nearest neighbors communication

    3.62 billion particles
  - GPU-enabled ray tracer

    All-to-all communication

    100 rays per cell 128^3 cells

Uintah Scaling Overview

- MPI only AMR MPMICE: N=6144 CPU cores; Largest = 98K CPU cores
- Thread/MPI AMR MPMICE: N=8192 CPU cores; **Largest=256K CPU cores**
- Thread/MPI RayTracing: N=16 CPU cores; Largest=1024 CPU cores
- Thread/MPI/GPU RayTracing: N=16 CPU and 1 GPU; **Largest=1024 CPU and 64 GPU**

THE UNIVERSITY OF UTAH

Future Work

- **Scheduler – Infrastructure**
  - GPU affinity for multi socket/GPU nodes
  - Support Intel MIC (Xeon Phi) offload-mode
  - PETSc GPU interface utilization
  - Mechanism to dynamically determine whether to run GPU or CPU version task
  - Optimize GPU codes for Nvidia Kepler
    - CUDA 5.0 – Dynamic Parallelism
    - GPU sub-scheduler

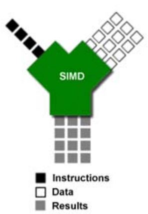
Questions?

Alstom Clean Coal Boiler Simulation
Monte Carlo Ray Tracing on GPU, Flow simulation on CPU

Software Homepage http://www.uintah.utah.edu/

Using GPUs in Energy Applications

- **ARCHES** Combustion Component
  - Alstom Clean Coal Boiler Problem
- Need to approximate radiation transfer equation
- Reverse Monte Carlo Ray Tracing (RMCRT)



  - Rays mutually exclusive - traced simultaneously
  - Ideal for **SIMD** parallelization of GPU
- Offload Ray Tracing and RNG to GPU(s)
  - CPU cores can perform other computation

THE UNIVERSITY OF UTAH

Performance Comparisons

Master-Slave Model vs. Unified

Execution Times – CPU Only					
#Cores	**2**	**4**	**8**	**16**	**32**
Master Slave	57.28	20.72	9.40	4.81	2.95
Unified	29.79	15.70	8.23	4.54	2.78

Execution Times – With GPU						
#Cores	**2**	**4**	**6**	**8**	**10**	**12**
Master Slave	4.55	4.09	3.95	3.68	3.64	3.34
Unified	3.82	3.52	3.09	2.90	2.50	2.09

CPU Problem*:* Combined MPMICE problem using AMR

Run on single Cray XE6 node with two 16-core AMD Opteron 6200 Series (Interlagos cores @2.6GHz) processors

GPU Problem*:* Reverse Monte Carlo Ray Tracer

Run on a single 12-core heterogeneous node (two Intel Xeon X5650 processors each with Westmere 6-core @2.67GHz, (2) Nvidia Tesla C2070 GPUs and (1) Nvidia GeForce 570 GTX GPU)

THE UNIVERSITY OF UTAH