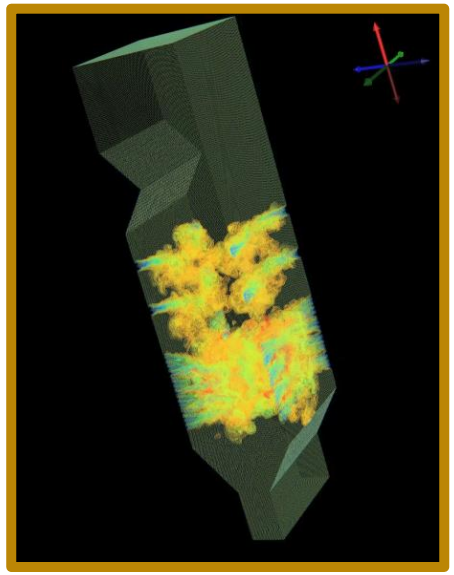
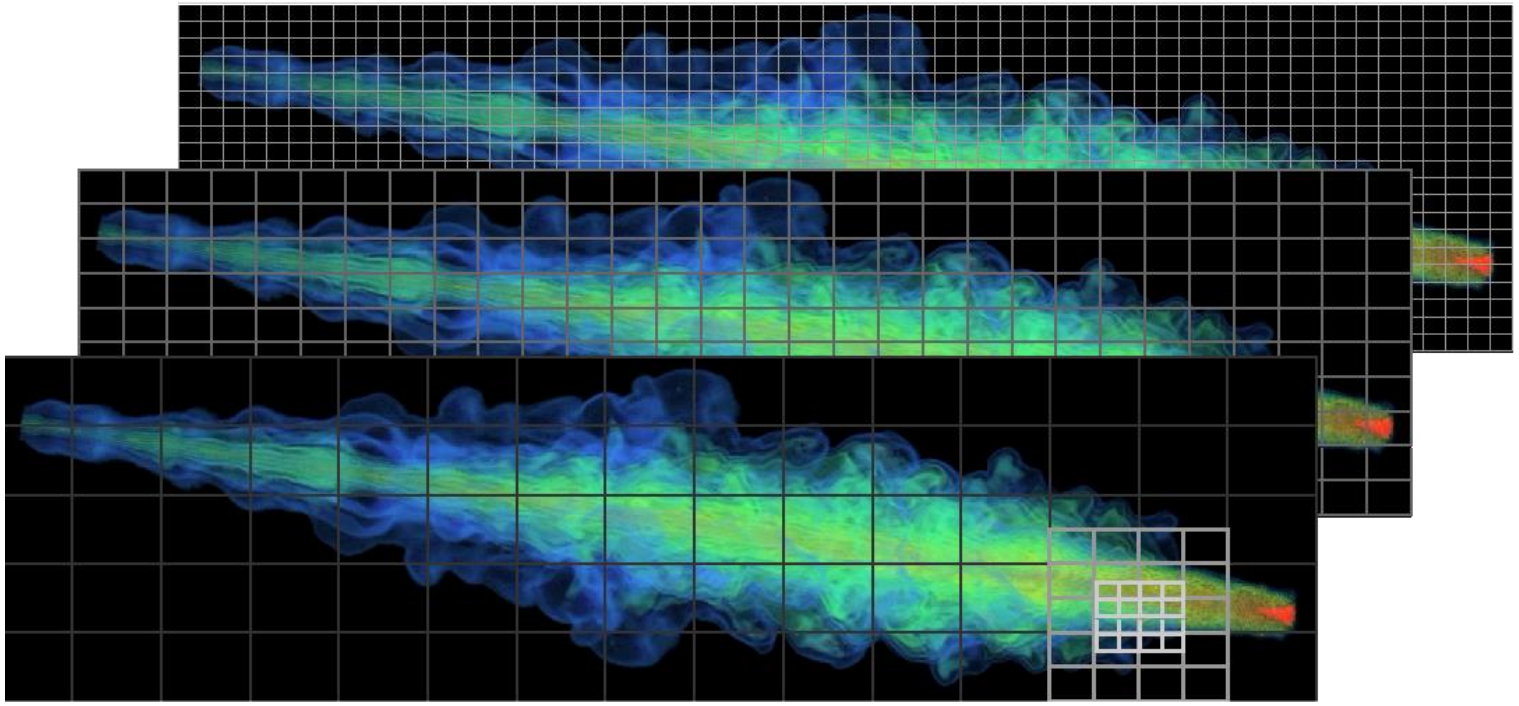
Harnessing the Power of Titan with the Uintah Computational Framework



CARBON CAPTURE
MULTIDISCIPLINARY
SIMULATION CENTER



Alan Humphrey, Qingyu Meng, Brad Peterson, Martin Berzins

Scientific Computing and Imaging Institute, University of Utah

SCI INSTITUTE

THE UNIVERSITY OF UTAH

Outline:

I. **Uintah Framework – Overview**
II. **Extending Uintah to Leverage GPUs**
III. **Target Application – DOE NNSA PSAAP II Multidisciplinary Simulation Center**
IV. **A Developing GPU-based Radiation Model**
V. **Summary and Questions**

<u>Central Theme</u>: Shielding developers from complexities inherent in heterogeneous systems like Titan & Keeneland

Thanks to:

- John Schmidt, Todd Harman, Jeremy Thornock, J. Davison de St. Germain
- Justin Luitjens and Steve Parker, NVIDIA
- DOE for funding the CSAFE project from 1997-2010,
- DOE NETL, DOE NNSA, INCITE, ALCC
- NSF for funding via SDCI and PetaApps, XSEDE
- Keeneland Computing Facility
- Oak Ridge Leadership Computing Facility for access to Titan
- DOE NNSA PSAPP II (March 2014)

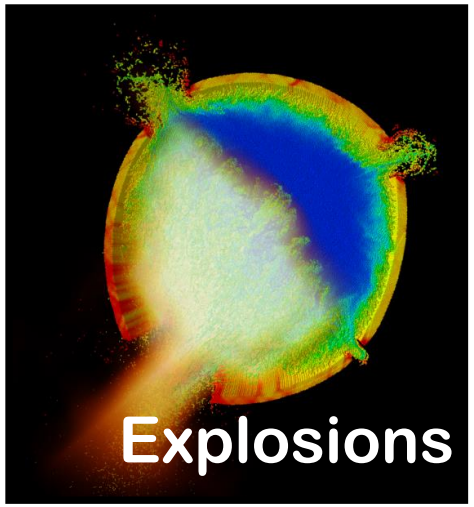
DOE Titan – 20 Petaflops
18,688 GPUs

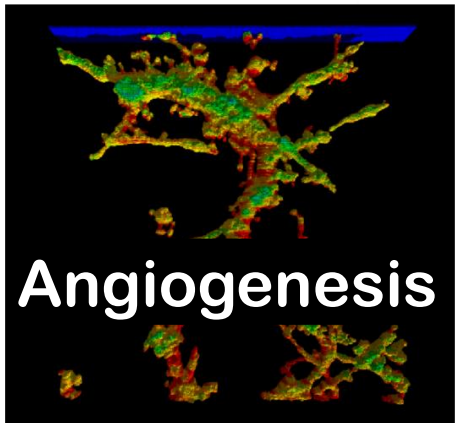
NSF Keeneland
792 GPUs

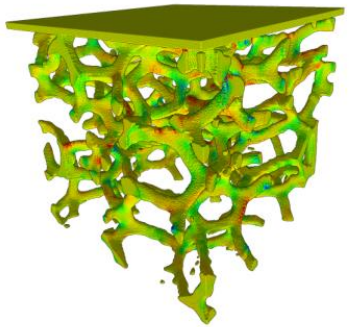
Uintah Overview

- Parallel, adaptive multi-physics framework
- Fluid-structure interaction problems
- Patch-based AMR:
- Particle system and mesh-based fluid solve

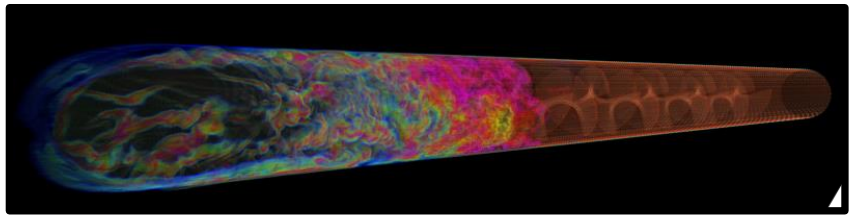
Plume Fires



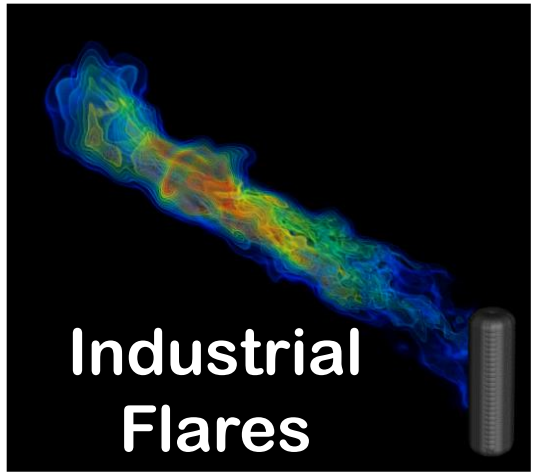


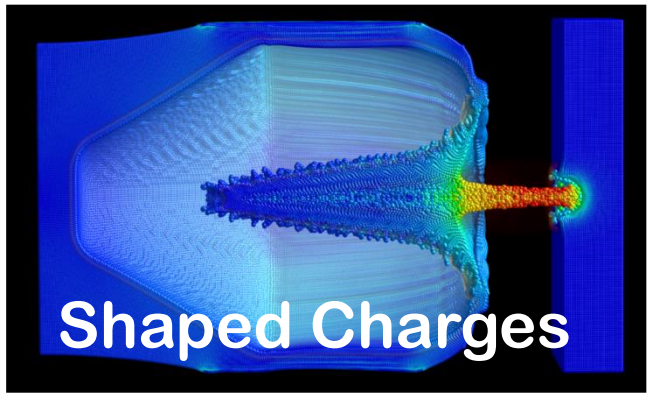


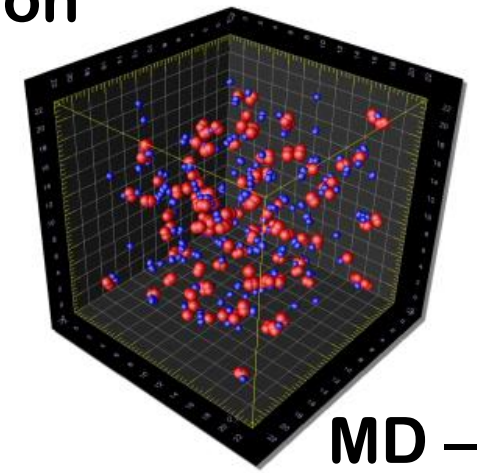
Foam Compaction



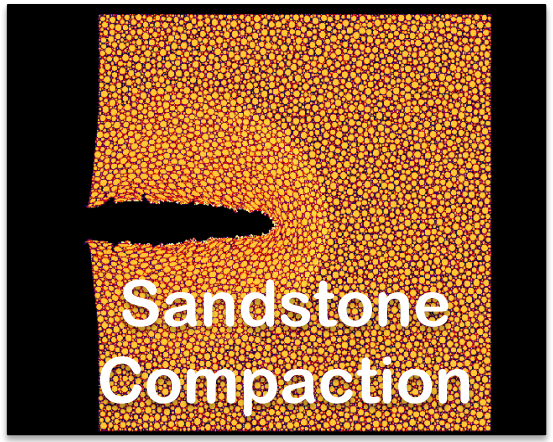
Chemical/Gas Mixing







MD – Multiscale Materials Design

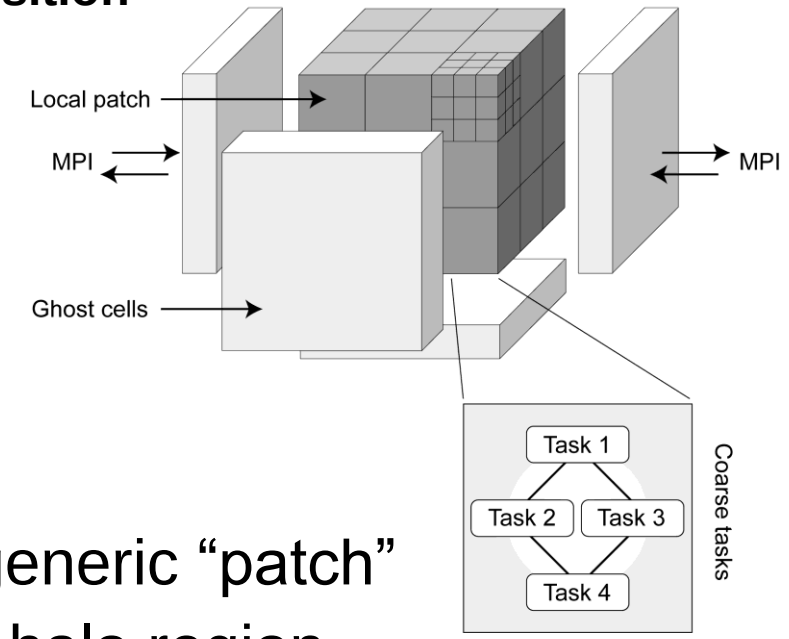


SCI INSTITUTE

THE UNIVERSITY OF UTAH

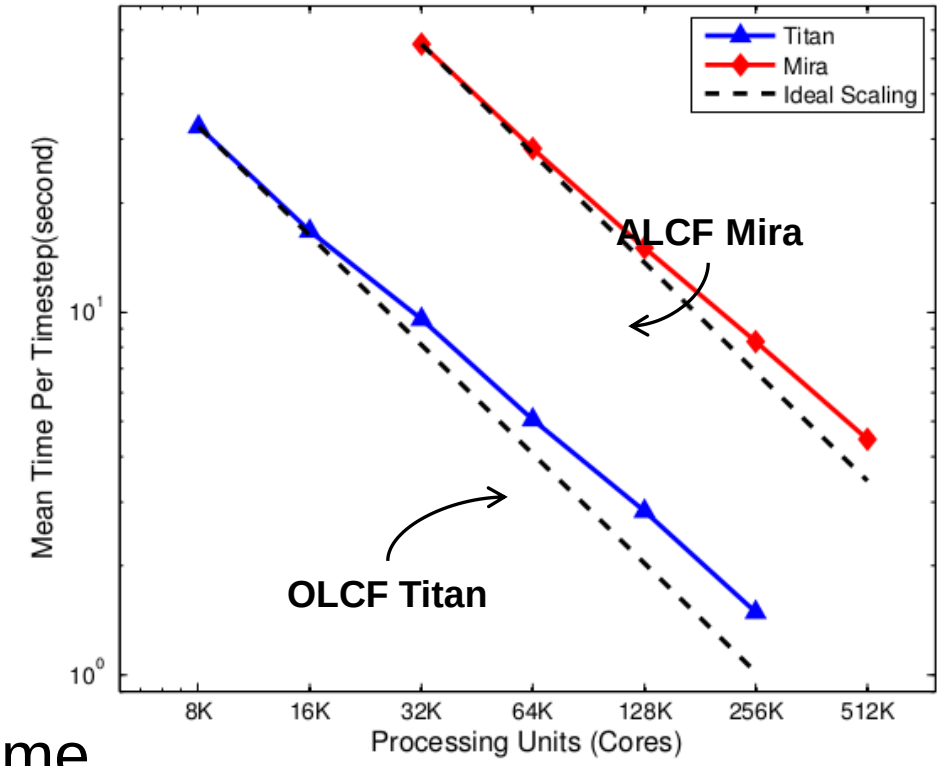
Uintah: Parallelism & Scaling

Patch-based domain decomposition



Asynchronous task-based paradigm

- Task - serial code on generic “patch”
- Task specifies desired halo region
- Clear separation of user code from parallelism/runtime
- Uintah infrastructure provides:
  - *automatic MPI message generation*
  - *load balancing*
  - *particle relocation*
  - *check pointing & restart*

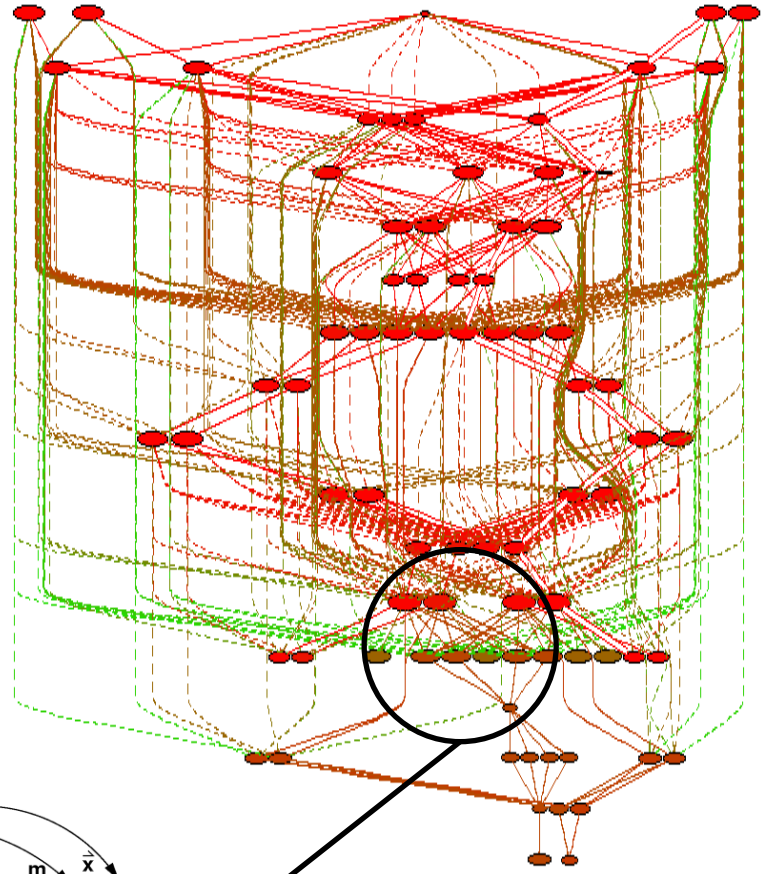


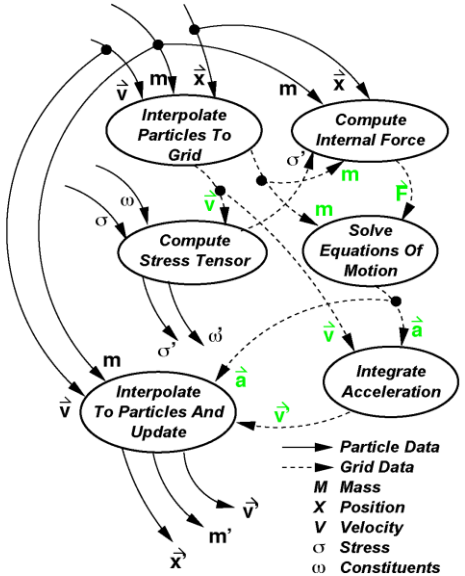
Strong Scaling:
Fluid-structure interaction problem using MPMICE algorithm w/ AMR

Existing components scale from 5k to 500K cores w/o changing any user code

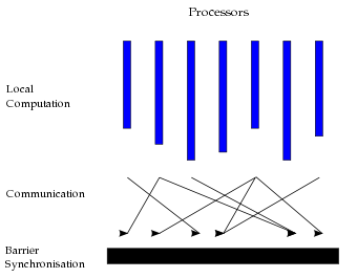
Uintah: Task-Graph Approach

- Task Graph: Directed Acyclic Graph (DAG)
- **Task –** basic unit of work
  - C++ method with computation (user written callback)
- Asynchronous, dynamic, out of order execution of tasks - **key idea**
- Overlap communication & computation
- Allows Uintah to be generalized to support accelerators
- GPU extension is realized without massive, sweeping code changes
  - **Infrastructure handles device API details**
  - **Provides convenient GPU APIs**
  - **User writes only GPU kernels for appropriate CPU tasks**



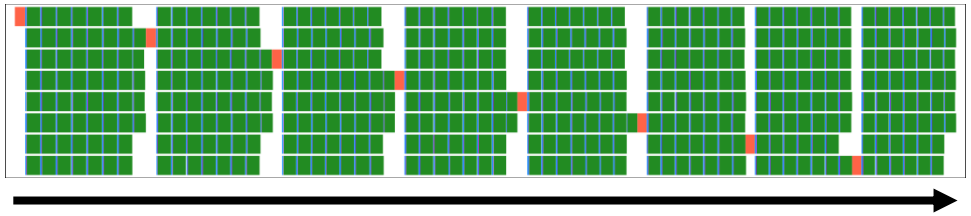


Why Does Dynamic Execution of Directed Acyclic Graphs Work So Well?

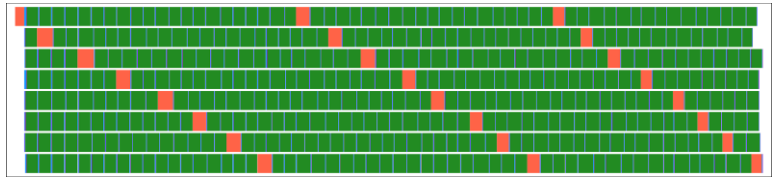


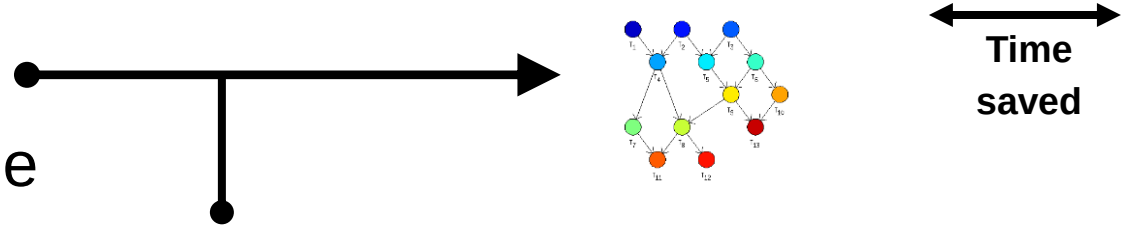
- Eliminate spurious synchronization points
- Multiple task-graphs across multicore (+GPU) nodes – parallel slackness
- Overlap communication with computation executing tasks as they become available – avoid waiting (out-of order execution).
- Load balance complex workloads by having a sufficiently rich mix of tasks per multicore node that load balancing is done per node (not core)

Bulk Synchronous Approach



DAG-based: dynamic scheduling

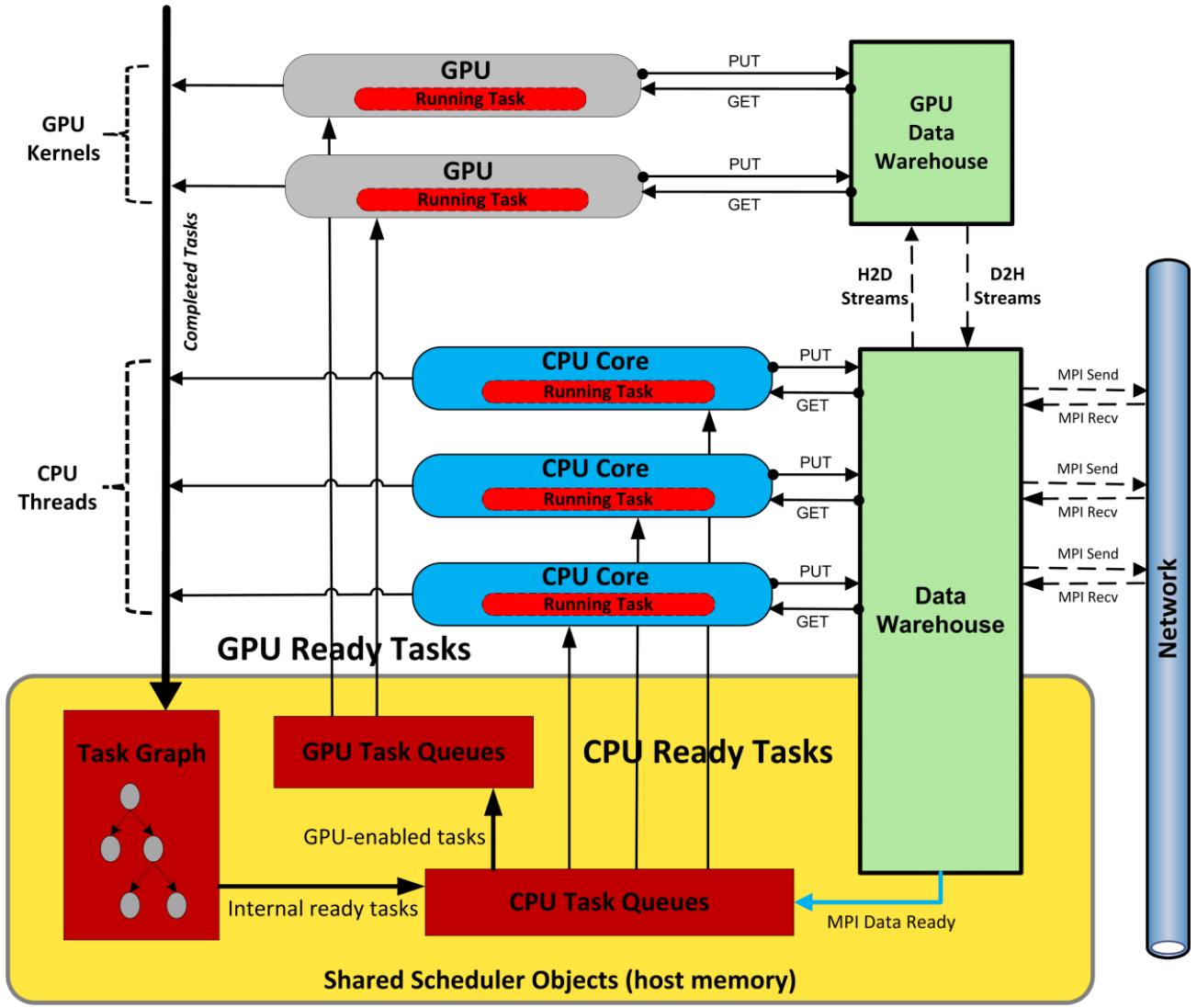




DATA FLOWS SPECIFY ORDER OF EXECUTION WITHOUT SYNCHRONIZATION

Uintah Heterogeneous Runtime System

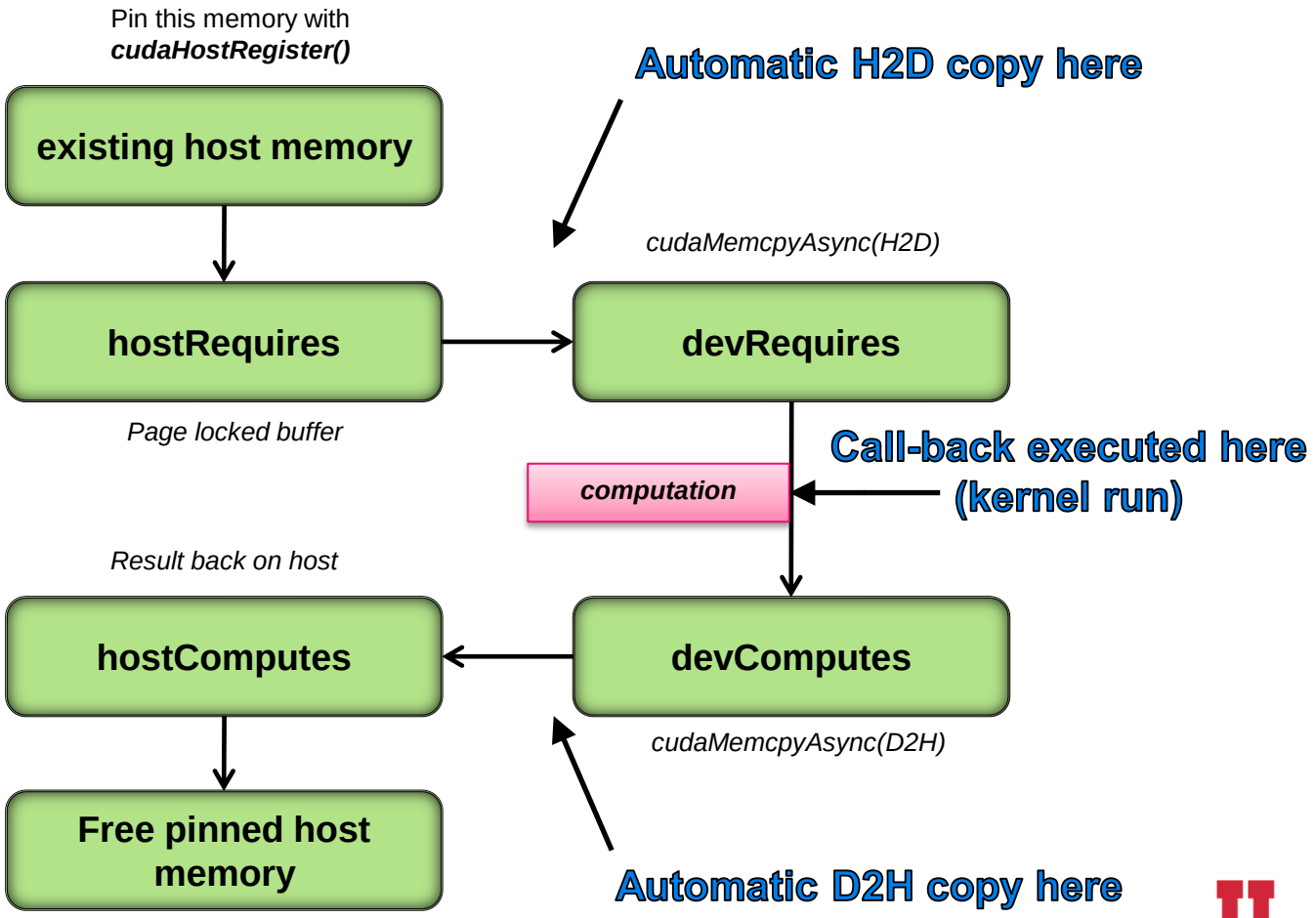
- Shared memory model on-node: 1 MPI rank per node
- MPI + Pthreads + CUDA
- Better load-balancing
- Decentralized: All threads
  - *access CPU/GPU task queues*
  - *process their own MPI*
  - *interface with GPUs*
- Scalable, efficient, lock-free data structures
- Task code must be thread-safe



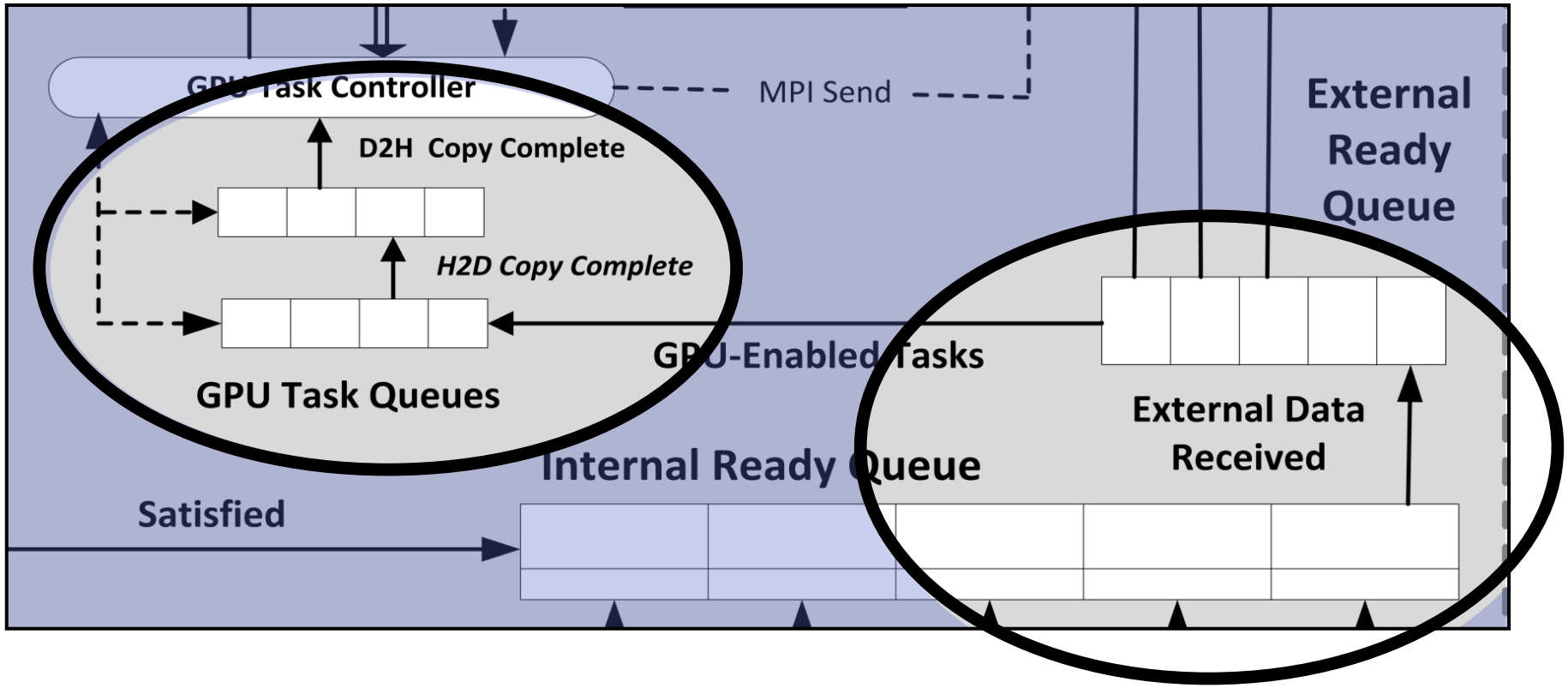
GPU Task and Data Management

Framework Manages Data Movement & Streams
Host ← → Device

- Use ***CUDA Asynchronous API***
- **Automatically** generate CUDA streams for task dependencies
- **Concurrently** execute kernels and memory copies
- **Preload** device data before task kernel executes
- **Multi-GPU** support

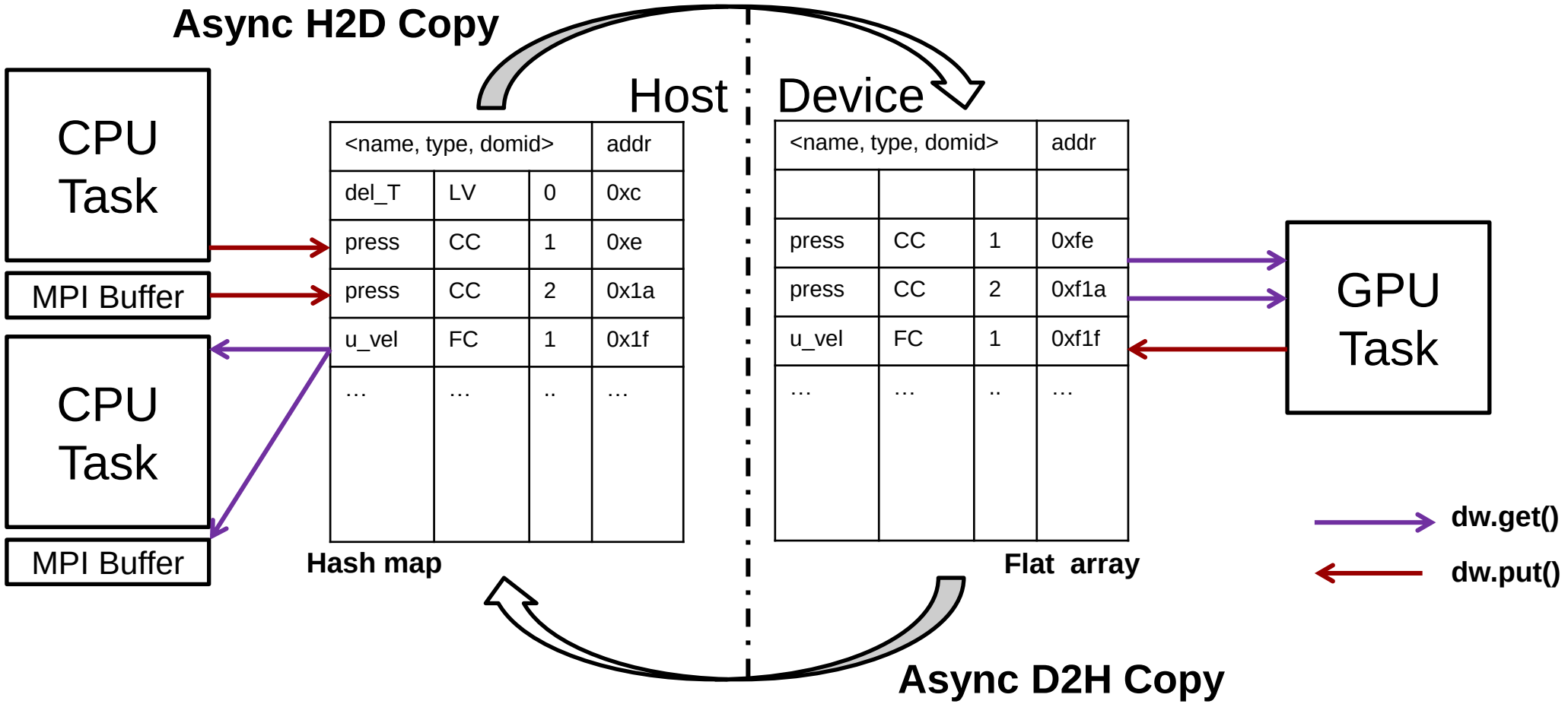


Multistage Task Queue Architecture



- Overlap computation with **PCIe transfers** and **MPI communication**
- Uintah can ***"pre-fetch"*** GPU data
  - *scheduler queries task-graph for a task's data requirements*
  - *migrate data dependencies to GPU and backfill until ready*

GPU DataWarehouse

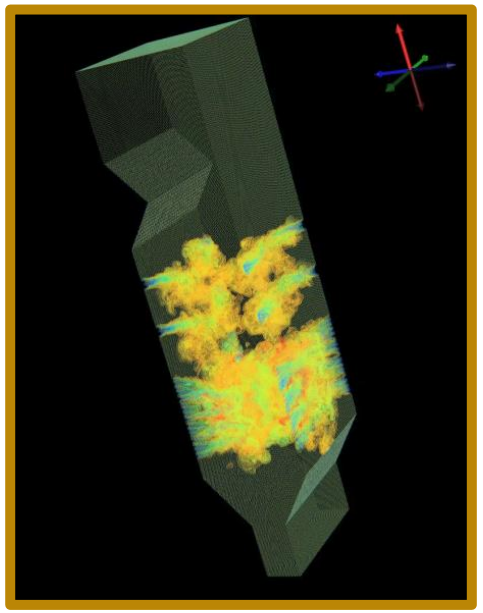


- Automatic, on-demand variable movement to-and-from device
- Implemented interfaces for both CPU/GPU Tasks

Target Problem

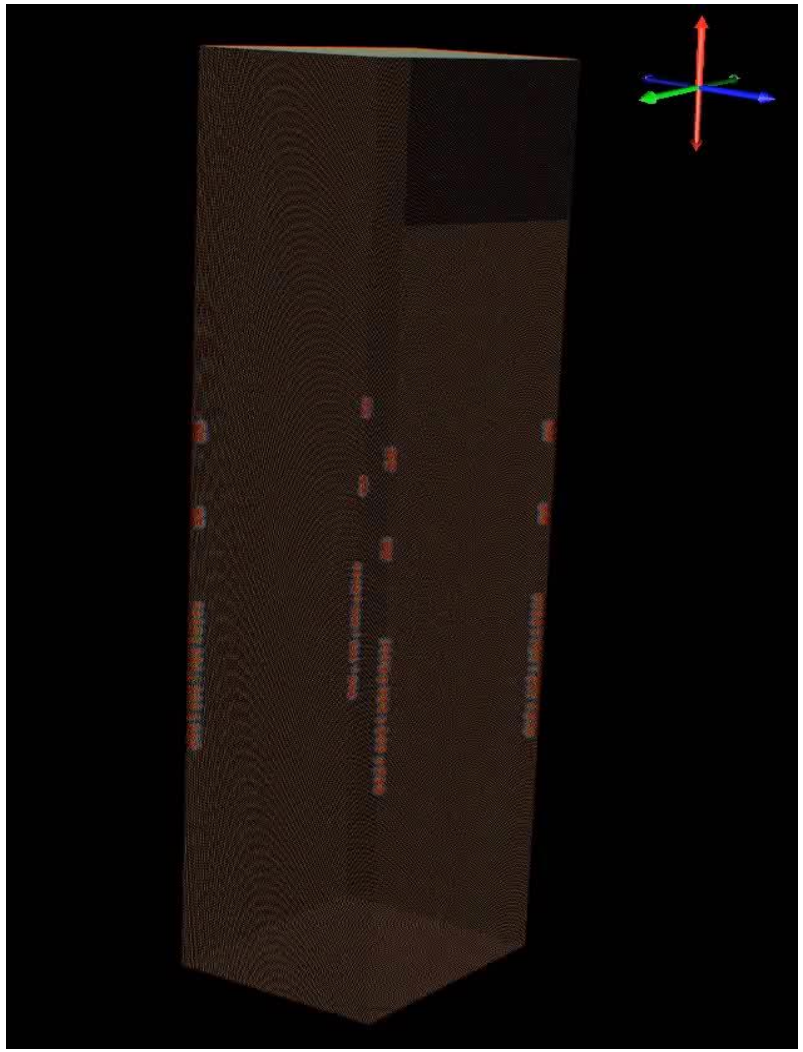
DOE NNSA PSAAP II Center – U. Utah

Alstom Power Boiler Facility



- Use simulation to facilitate design of clean coal boilers
- 350MWe boiler problem
- 1mm grid resolution, 9 x 10^{12} cells
- To simulate problem in 48 hours of wall clock time:
  - "require estimated 50-100 million fast cores"

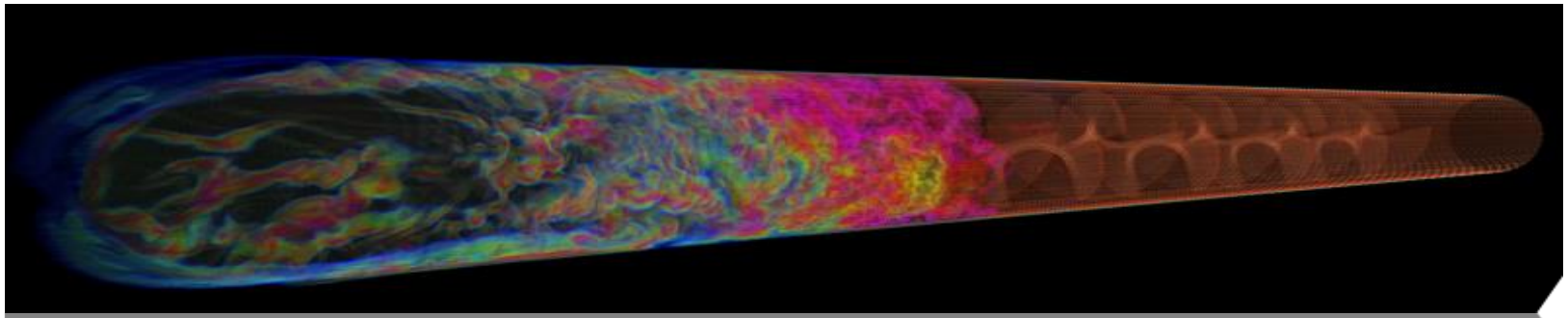
Professor Phil Smith - ICSE, Utah



O_2 concentrations in a clean coal boiler

ARCHES Combustion Component

- **Designed for simulating turbulent reacting flows with participating media radiation**
  - *Heat, mass, and momentum transport*



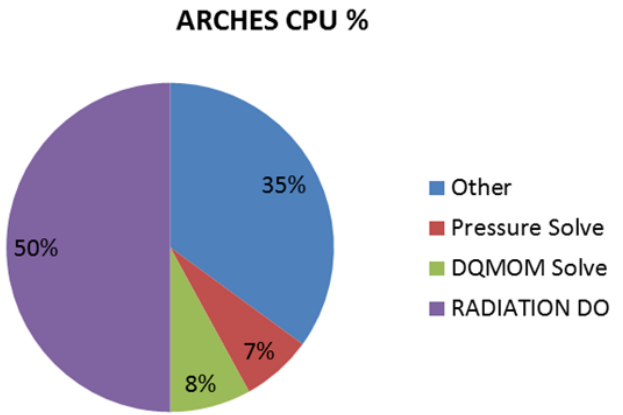
- **3D Large Eddy Simulation** (LES) code
  - Evaluate large clean coal boilers that alleviate CO_2 concerns
- **ARCHES** is massively parallel & highly scalable through its integration with Uintah

ARCHES Radiation Model

- **Approximate radiative heat transfer equation**
- Methods Considered
  - **Discrete Ordinates Method (DOM)**: slow and expensive (solving linear systems) and is difficult to add more complex radiation physics, specifically scattering – Working to leverage ***NVIDIA AmgX***
  - **Reverse Monte Carlo Ray Tracing (RMCRT)**: faster due to ray decomposition and naturally incorporates physics (such as scattering) with ease. No linear solve. Easily ported to GPUs

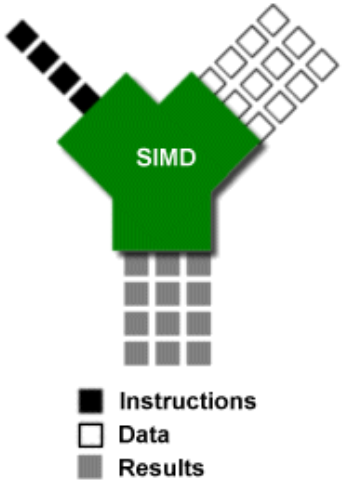
Radiation via DOM performed every timestep

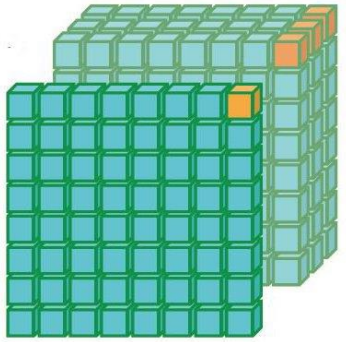
50% CPU time



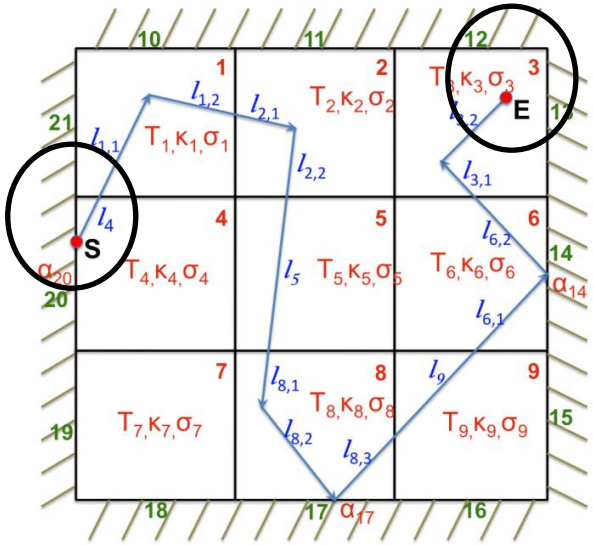
Reverse Monte Carlo Ray Tracing

- ***Lends itself to scalable parallelism***
  - Amenable to GPUs – SIMD
  - Rays mutually exclusive
  - Can be traced simultaneously
    - ✓ any given cell and time step





Map CUDA threads to cells on Uintah mesh patches



Rays traced backwards from computational cell, eliminating the need to track rays that never reach that cell

Figure shows the back path of a ray from **S** to the **emitter E**, on a 2D, nine cell structured mesh patch

Prototype GPU-RMCRT Speedups

Single Node: All CPU Cores vs. Single GPU				
Machine	**Rays**	**CPU (sec)**	**GPU (sec)**	**Speedup (x)**
Keeneland *12-cores Intel*	**25**	**4.89**	**1.16**	**4.22**
	50	**9.08**	**1.86**	**4.88**
	100	**18.56**	**3.16**	**5.87**
TitanDev *16-cores AMD*	**25**	**6.67**	**1.00**	**6.67**
	50	**13.98**	**1.66**	**8.42**
	100	**25.63**	**3.00**	**8.54**

Speedup: mean time per timestep

- **GPU – NVIDIA Tesla M2090**
- **Keeneland CPU Cores – Intel Xeon X5660 (Westmere) @2.8GHz**
- **TitanDev CPU Cores – AMD Opteron 6200 (Interlagos) @2.6GHz**

SCI INSTITUTE

THE UNIVERSITY OF UTAH

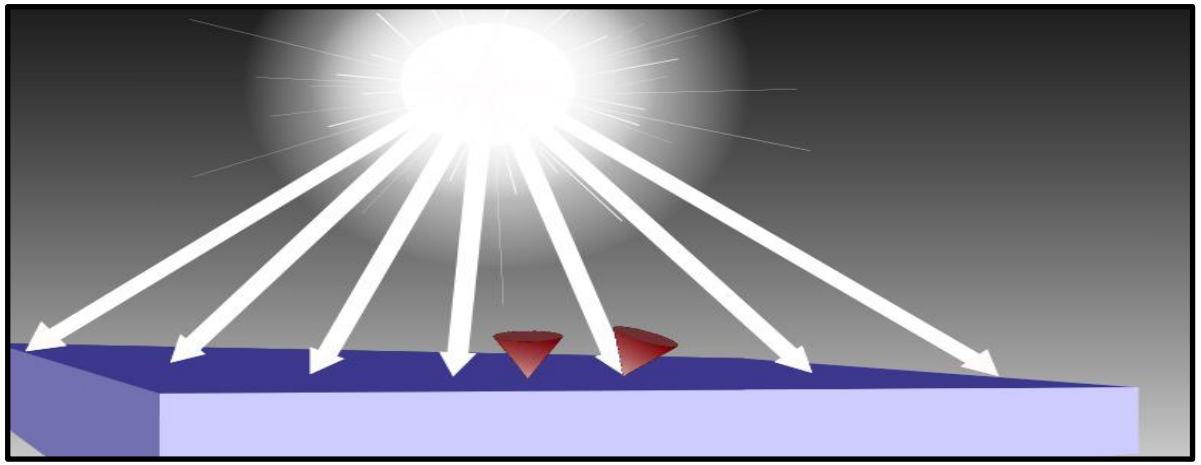
Production-Grade GPU-RMCRT

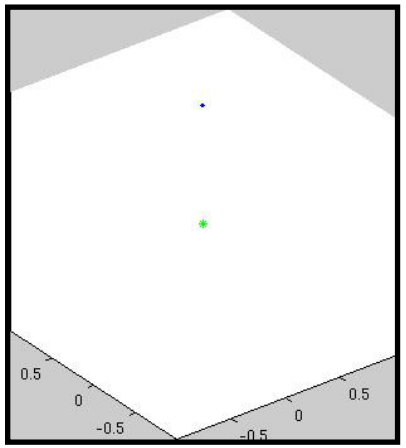
- **Incorporate dominant physics**
  - *Emitting / Absorbing Media*
  - *Emitting and Reflective Walls*
  - *Ray Scattering*
- **User controls # rays per cell**

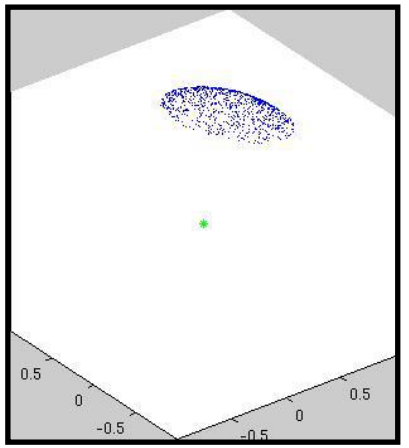
NVIDIA K20m GPU
3.8x faster than
16 CPU cores
(Intel Xeon E5-2660 @2.20 GHz)

Speedup: mean time per timestep

Virtual Radiometer Still Needed

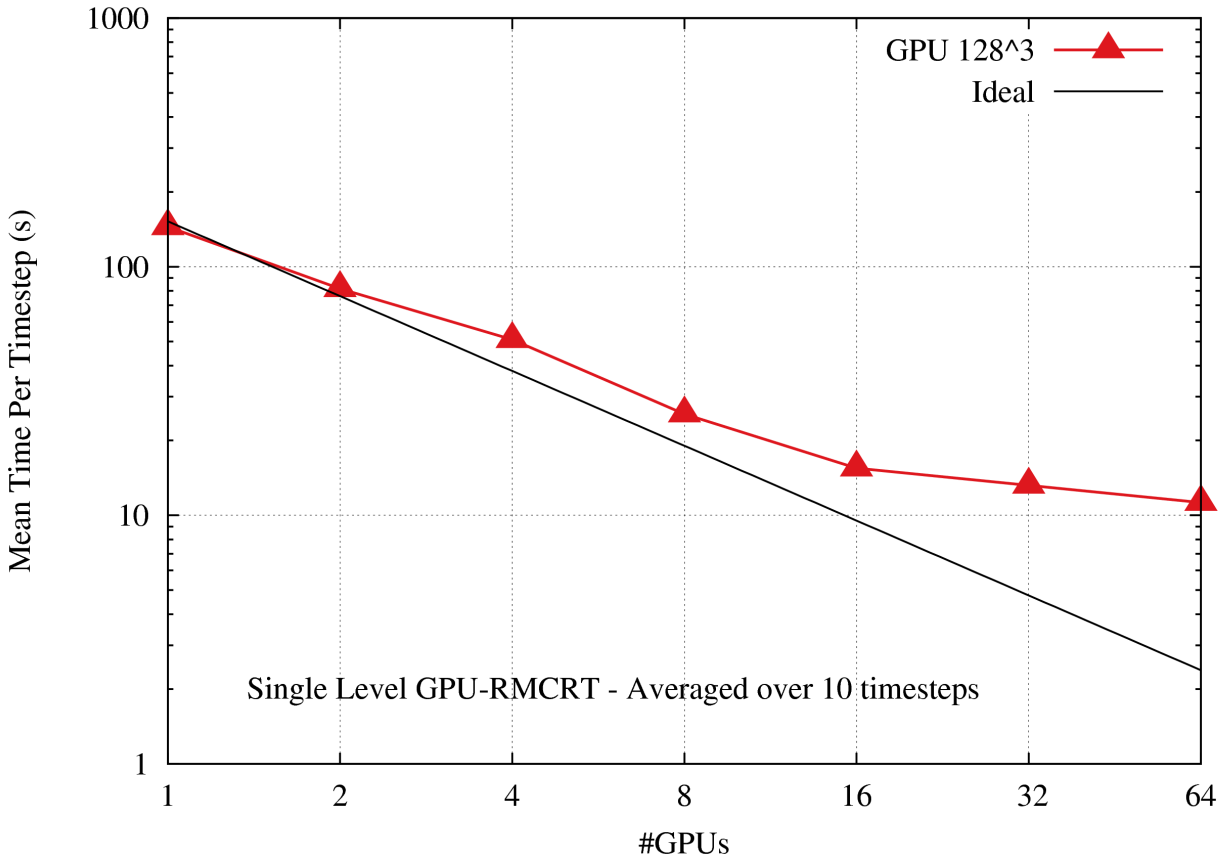






- **All possible view angles**
- **Arbitrary view angle orientations**

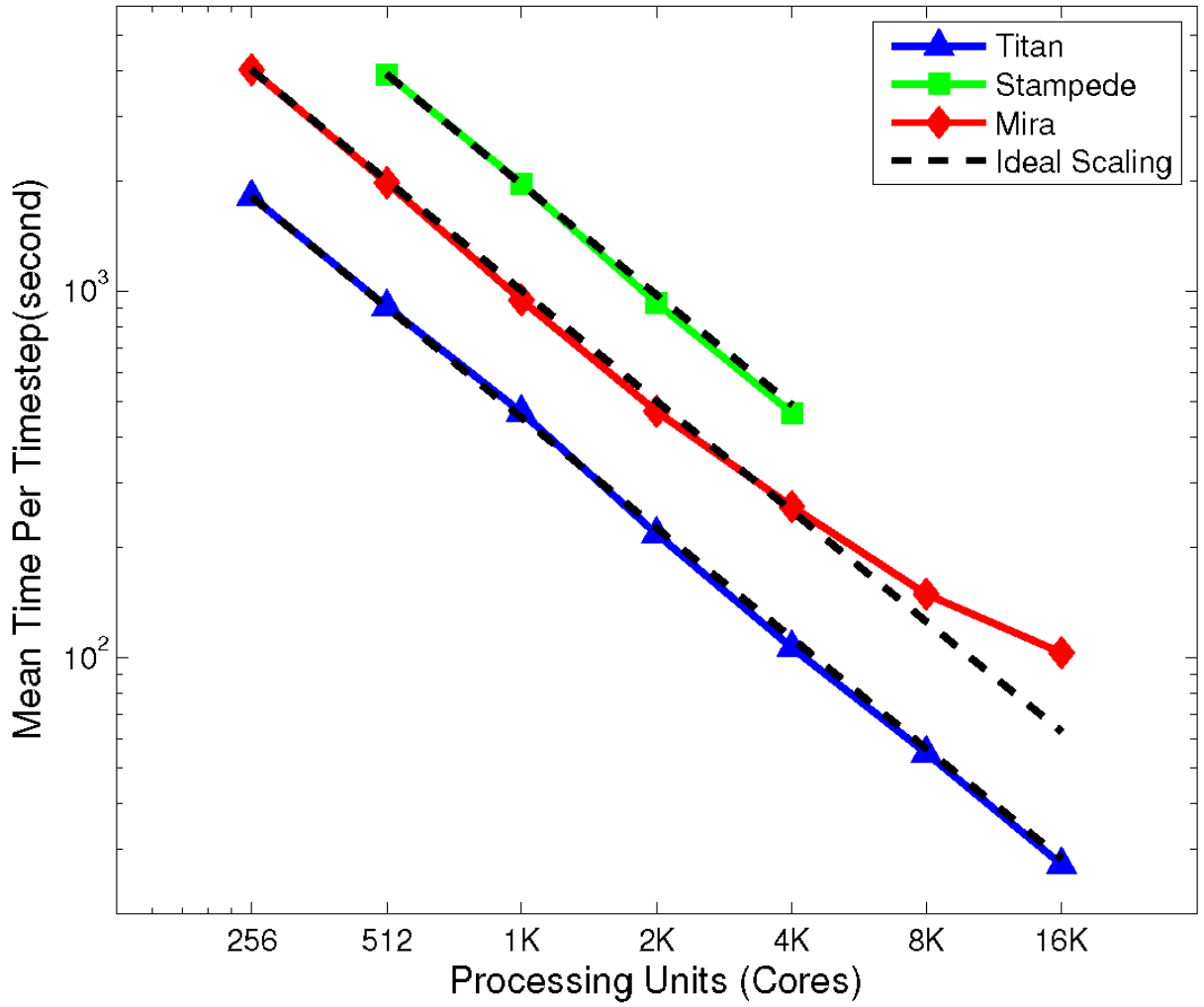
Single-Level GPU-RMCRT Scalability



Strong scaling results for production GPU implementations of RMCRT
NVIDIA - K20 GPUs

- Mean time per timestep for GPU lower than CPU (up to 64 GPUs)
- GPU implementation quickly runs out of work
- **All-to-all** nature of problem limits size that can be computed due to memory and comm. constraints with large, highly resolved physical domains

Two-Level RMCRT CPU Scalability

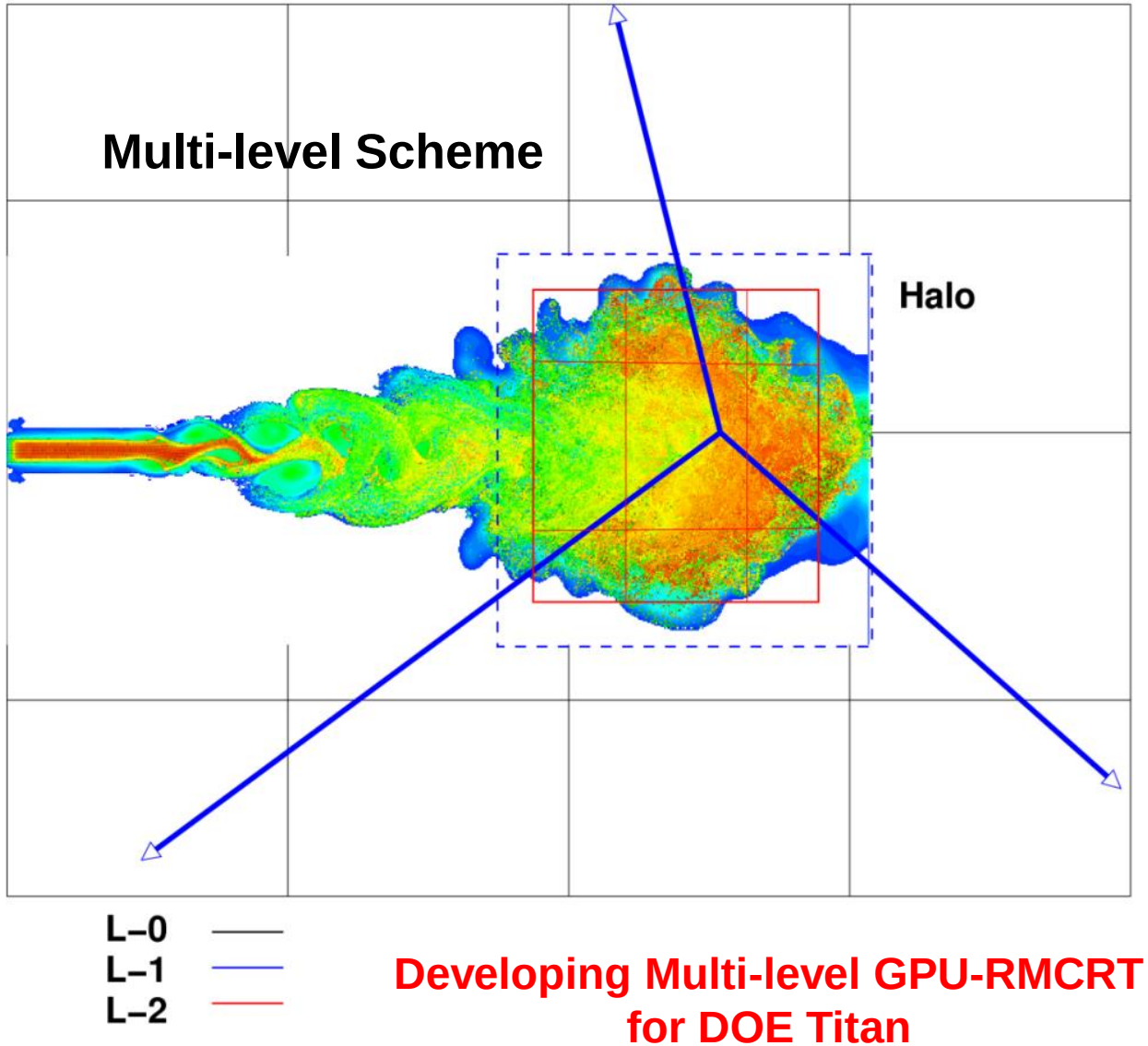


Strong Scaling: **Two-level CPU Prototype in ARCHES**

- **How far can we scale with 3 or more levels?**
- **Can we utilize the whole of systems like Titan with GPU approach**

Multi-Level RMCRT (Research)

- Use coarser representation of computational domain with multiple levels
- Define Region of Interest (**ROI**)
- Surround **ROI** with successively coarser grid
- As rays travel away from **ROI**, the stride taken between cells becomes larger
- This reduces computational cost, memory usage and MPI message volume.

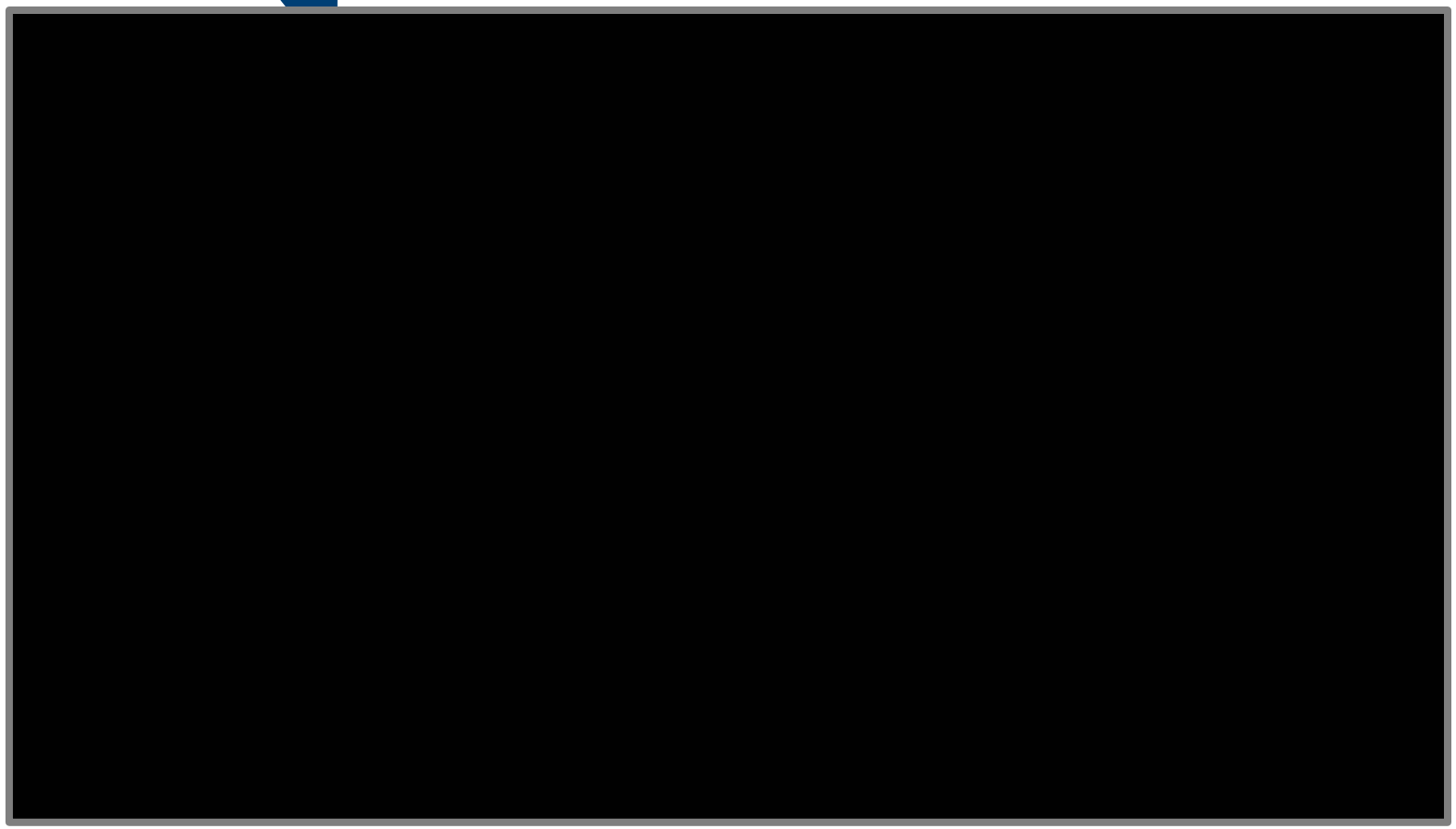


Developing Multi-level GPU-RMCRT for DOE Titan

Summary

- **Uintah Framework - DAG Approach**
  - Powerful abstraction for solving challenging engineering problems
  - Extended with relative ease to efficiently leverage GPUs
  - Provides convenient separation of problem structure from data and communication – application code vs. runtime
  - Shields applications developer from complexities of parallel programming involved with heterogeneous HPC systems
  - Allows scheduling algorithms to optimize for scalability and performance

Questions?



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