Reducing Overhead in the Uintah Framework to Support Short-Lived Tasks on GPU-Heterogeneous Architectures

Brad Peterson, Alan Humphrey, Harish Dasari, James Sutherland, Tony Saad, Martin Berzins

Scientific Computing and Imaging Institute, University of Utah

SCI INSTITUTE

THE UNIVERSITY OF UTAH

Outline:

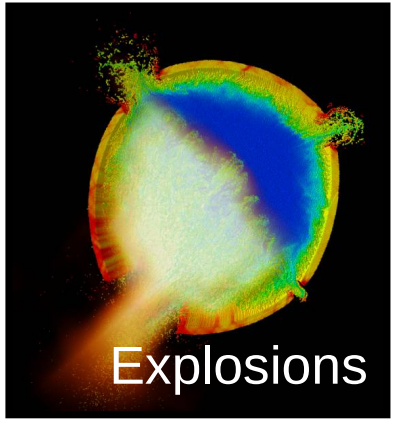
<u>Central Theme</u>: Improvements in Uintah's runtime system allows many additional classes of computations to achieve speedups by utilizing GPUs.

I. **Uintah Framework – Overview**
II. **Challenges certain computations pose with Uintah's runtime**
III. **Simplifying shared GPU data stores**
IV. **Allowing for GPU data persistence, including halo information**
V. **Summary and Questions**

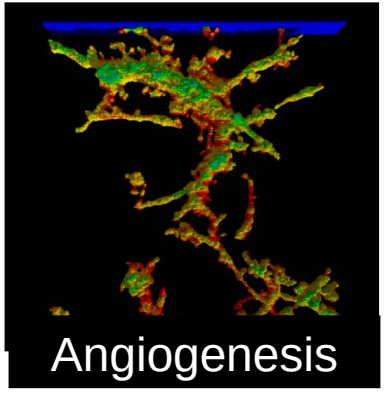
DOE Titan – 18,688 nodes
Each with one NVidia GPU and one AMD Opteron
27.1 Petaflops in total
24.5 petaflops for the GPUs
2.6 petaflops for the CPUs

Uintah: Runtime System for Multiphysics Simulations

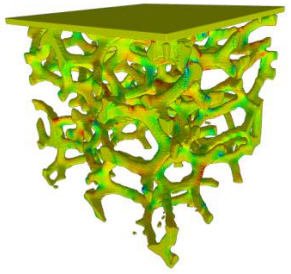
Plume Fires



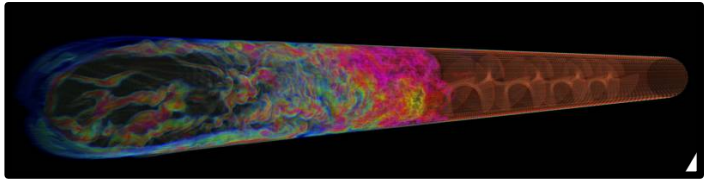
Explosions



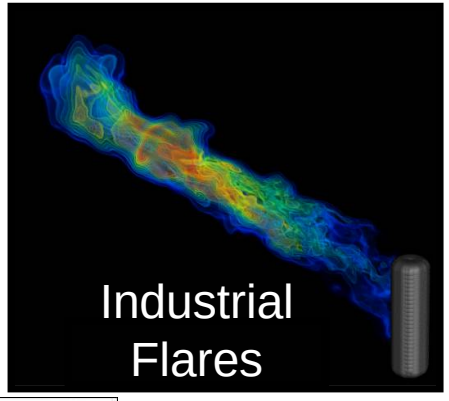
Angiogenesis



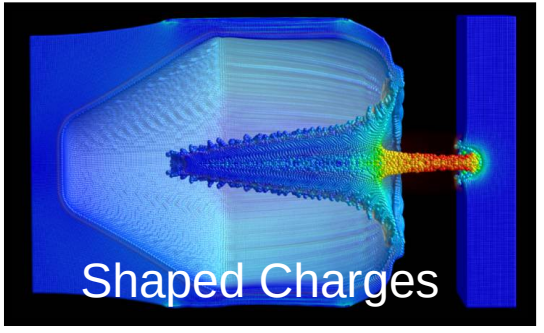
Foam Compaction



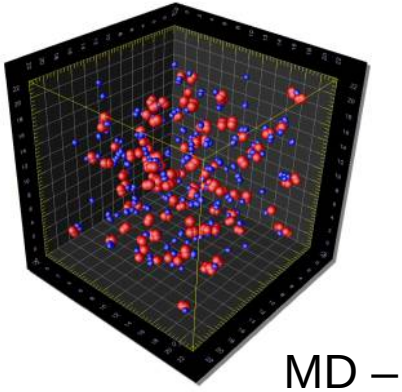
Chemical/Gas Mixing



Industrial Flares



Shaped Charges



MD – Multiscale Materials Design

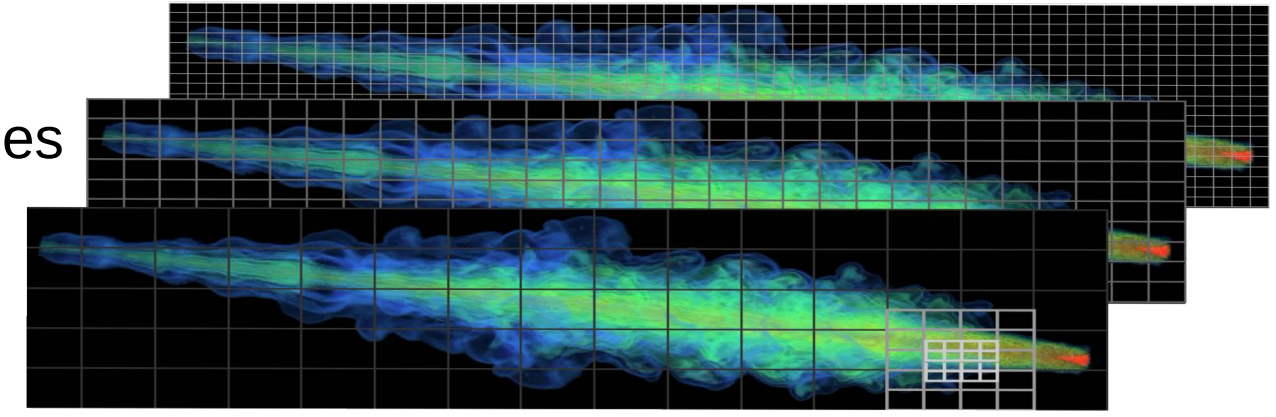
Sandstone Compaction

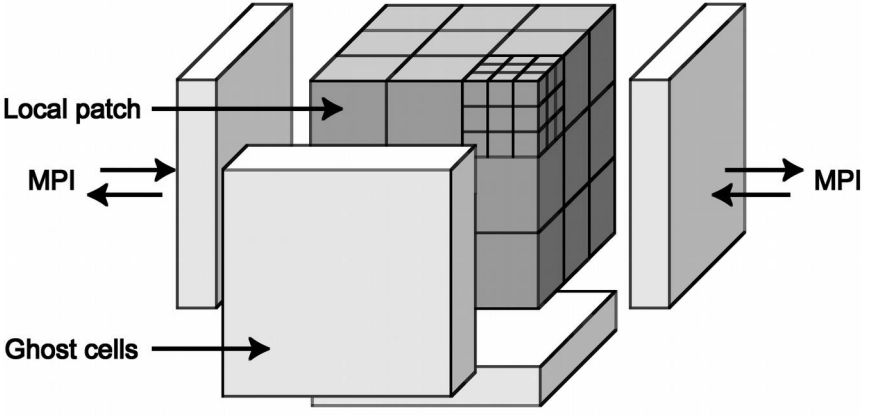
SCI INSTITUTE

THE UNIVERSITY OF UTAH

Uintah Domain Decomposition

- A structured mesh grid is partitioned into uniform patches
- One node is assigned many patches (usually one patch per logical core)
- One MPI rank per node
- Halo (ghost cells) exchanged with MPI rank neighbors
- Adaptive mesh refinement

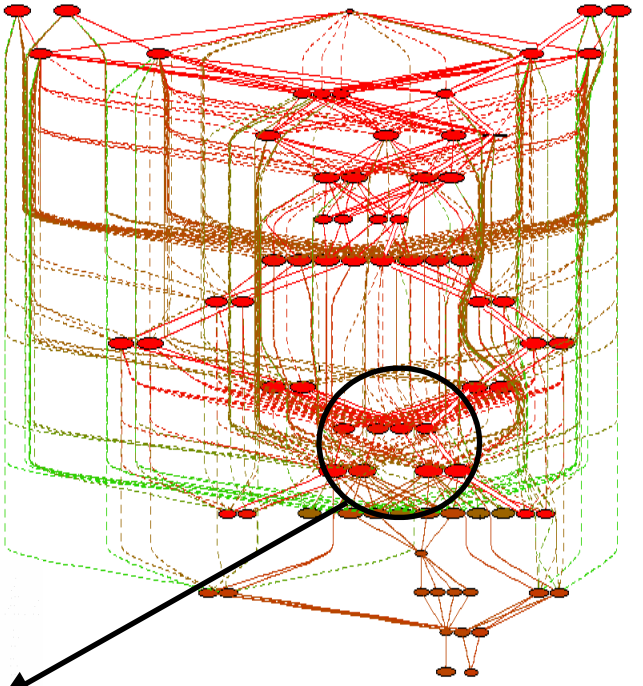


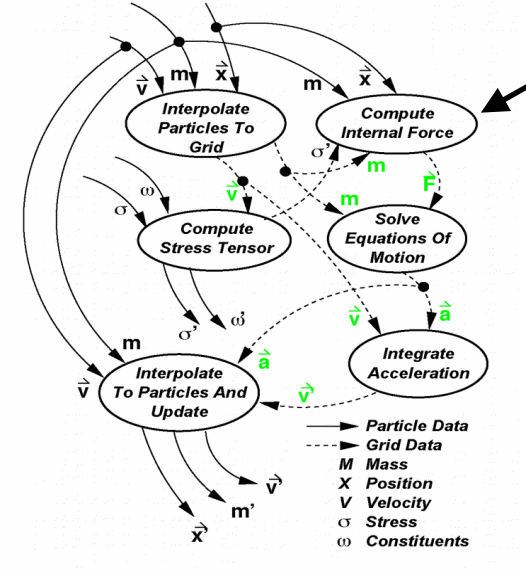


Patch structure within one MPI rank

Uintah Task Parallelism and Task Graph

- User defines Uintah Tasks:
  - **Input** and **output** variables
  - A callback function
- Uintah analyzes task dependencies
  - Creates a task graph
  - Exchange halo (ghost cells) with neighbors
  - Prepares tasks and executes them asynchronously



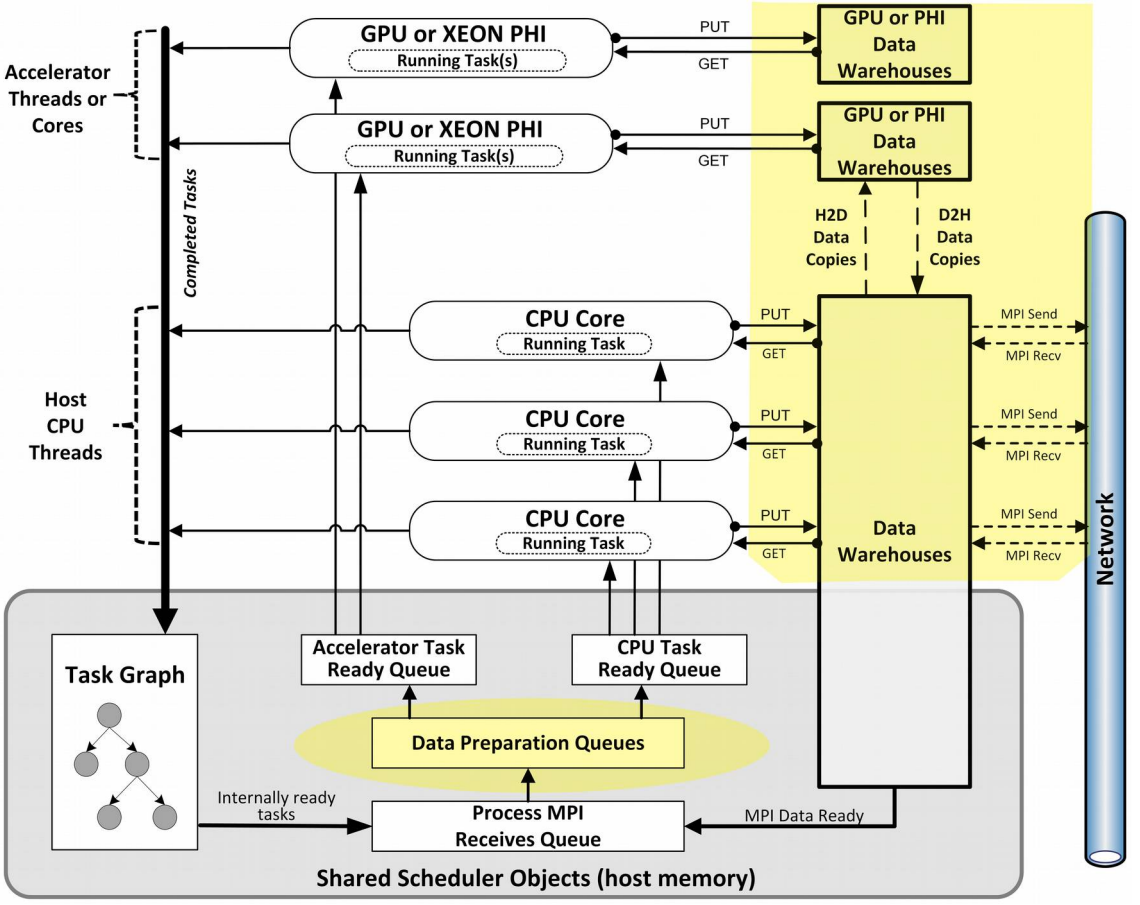


Uintah Full Overview

- Scales to tens of thousands of nodes, and hundreds of thousands of cores
- Parallelism through MPI, Pthreads, GPUs, and Xeon Phis.
- Scheduler fully decentralized, no master thread
- Shared and lock free data warehouses

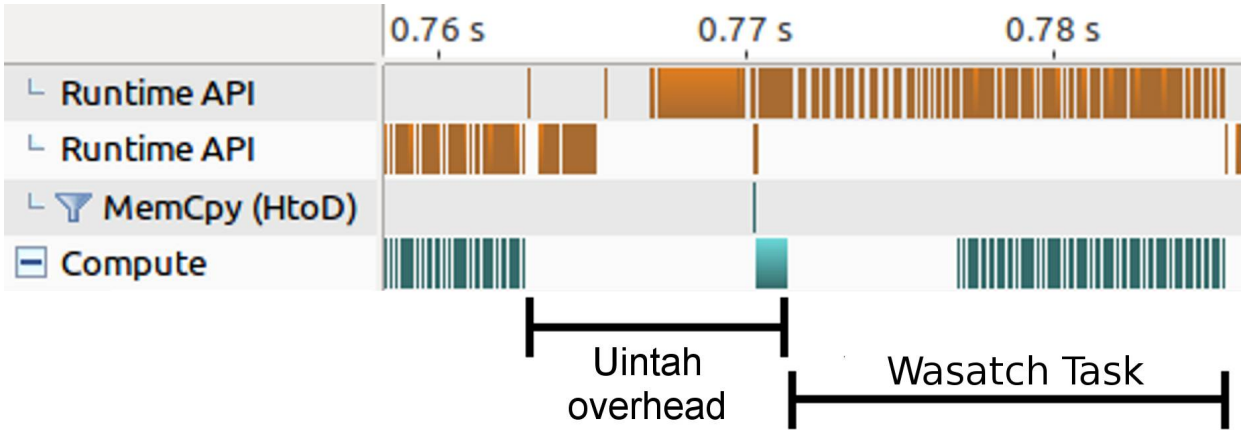
Overall goals:

- Hide runtime from the user
- Keep CPU and GPU programming models similar



This work focuses on the highlighted regions

Difficulties With Previous Runtime



If a GPU task executes quickly, the overhead becomes substantial

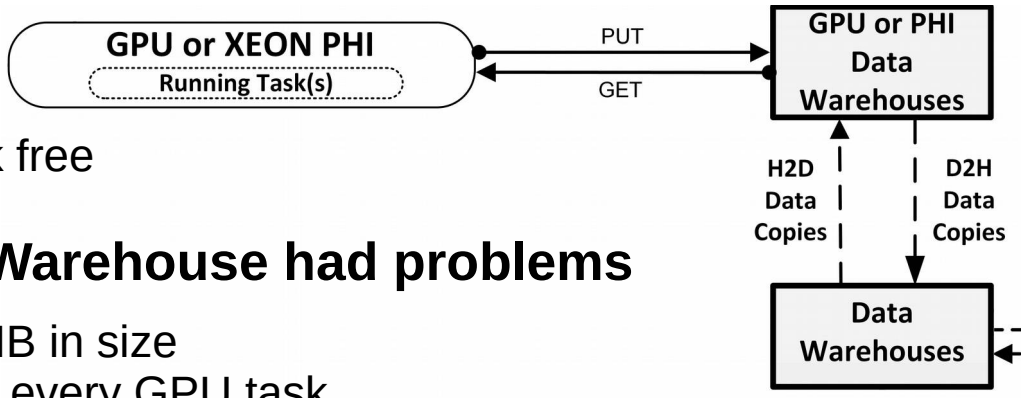
Unwanted overhead was mainly caused by:

- Copying full GPU Data Warehouses for each task
- Tasks with many variables using the PCIe bus (This pictured task required 120 variables)
- Many materials in the computation

Shared GPU Data Warehouses

The previous design had a philosophy:

1. Any task on any CPU thread can access the host-side Data Warehouse
2. This CPU Data Warehouse is shared and lock free



The same philosophy for a GPU Data Warehouse had problems

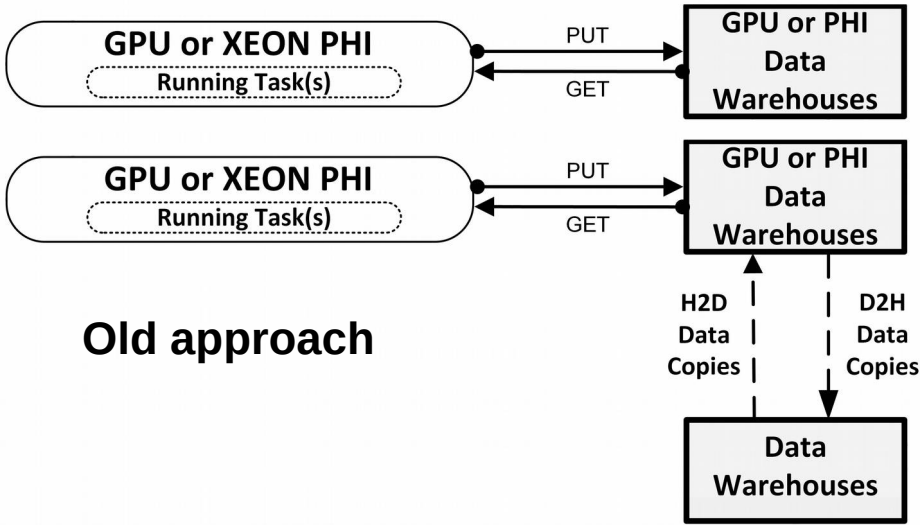
- The GPU Data Warehouse kept growing, 2-3 MB in size
- More CPU cores meant more full syncs prior to every GPU task
- We halted all GPU computations to allow full sync to GPU
- Trying to avoid halting and copy task portions into the GPU DW is problematic
- Underlying code was turning into maintaining two separate Data Warehouse objects

Root issue: Lock free intrinsics work well for host DW. No easy way to sync to and get data from an accelerator DW in a lock free way.

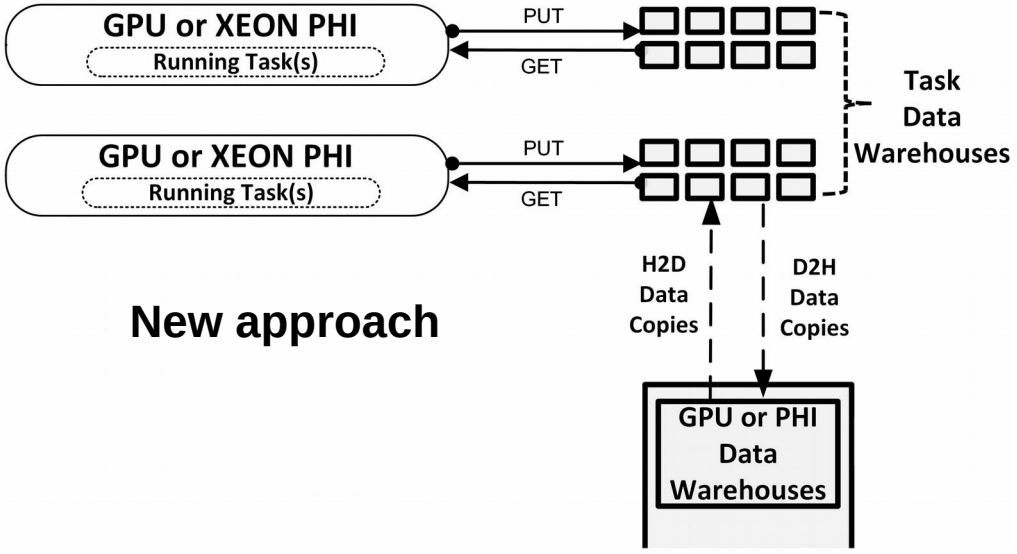
Solution - Task Specific Data Warehouses

New philosophy:
- Each task has its own distinct GPU DW
- No sharing or lock free intrinsics required

- Can overlap computation with communication
- Get and put interface format does not change
- Required no changes to existing GPU task code



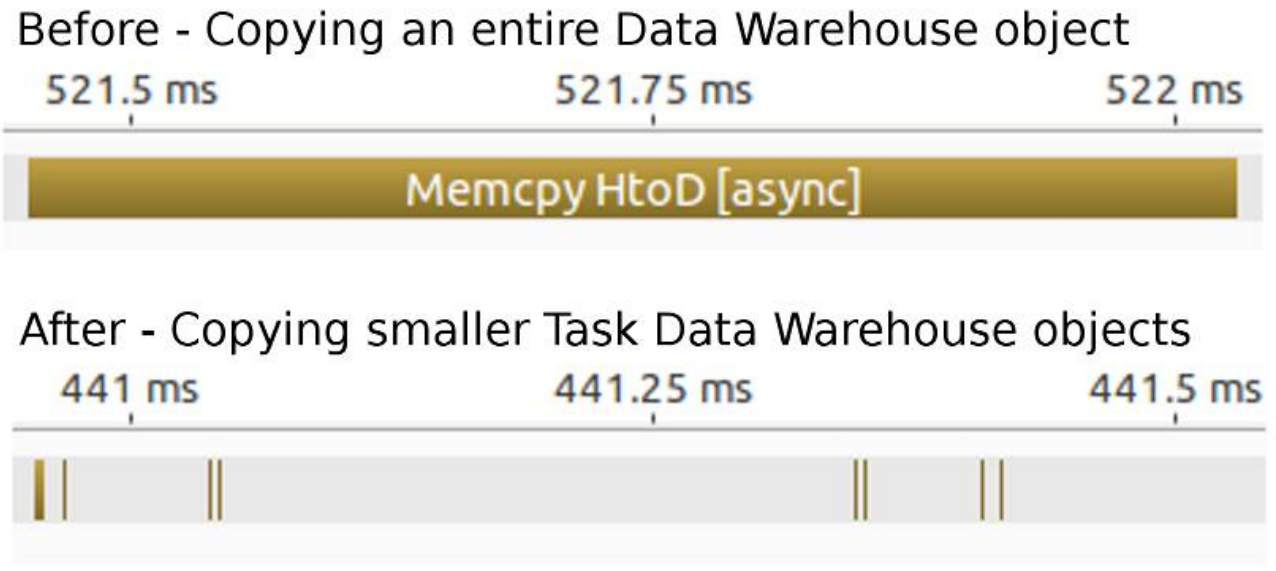
Old approach



New approach

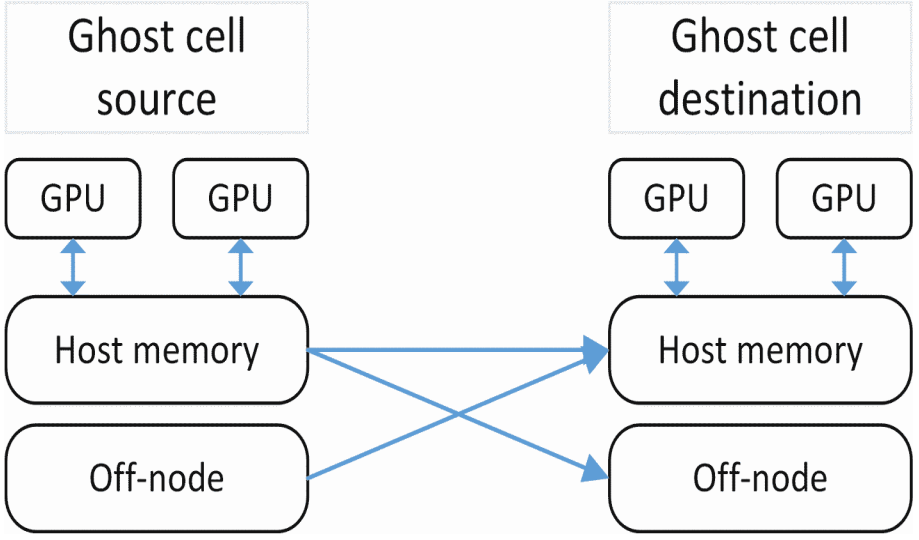
Making Task Data Warehouses Small

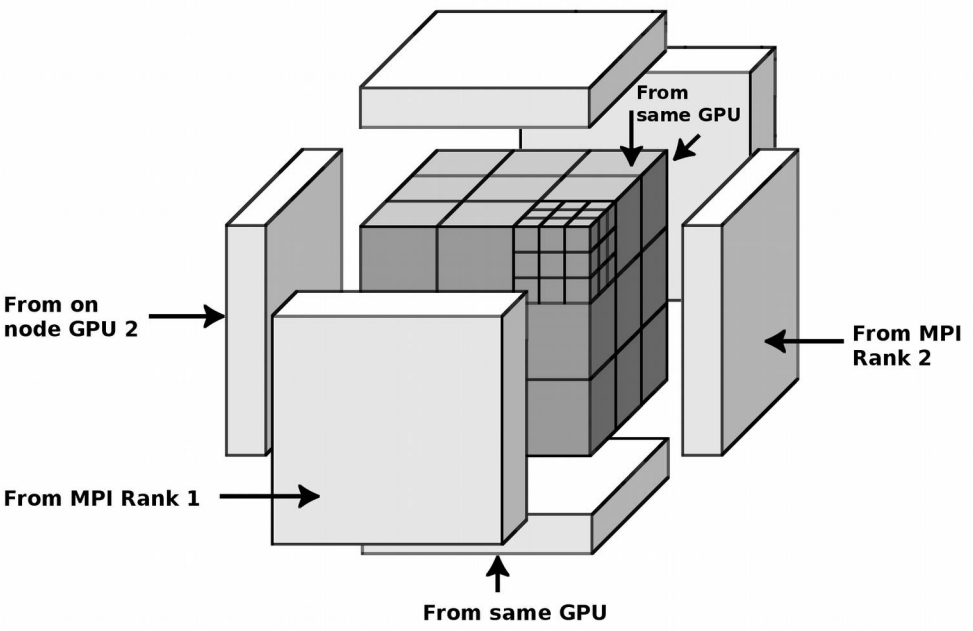
- Perform a dry run. All variables and total amount of copies must be accumulated and counted.
- A serialized object is allocated to minimum size, reduces allocs and API calls.
- Flexible design can work for Xeon Phis or any future accelerator with its own memory.



Ghost Cells with Previous Runtime

- Quick and easy. Designed for problems with computations where each timestep took at least 1 second to complete
- Lets all ghost cell management occur in host memory
- Too much movement across the PCIe bus for short-lived tasks





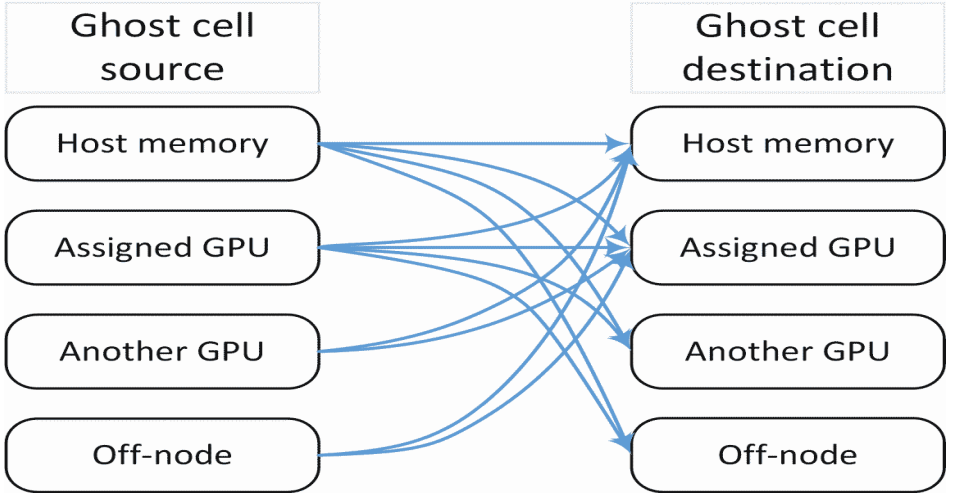
Ghost Cells Management Refactor

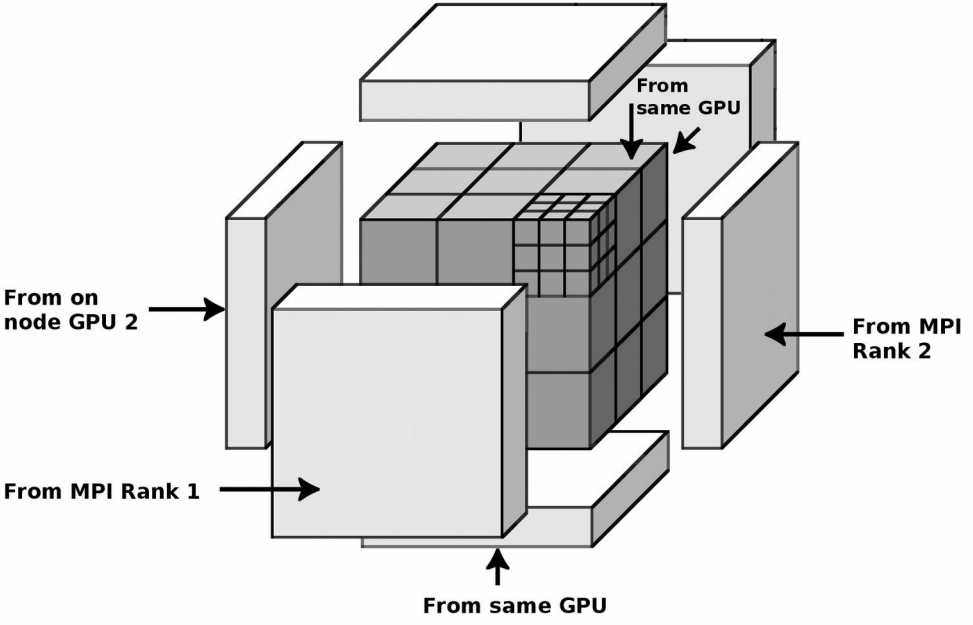
- **New philosophy: Keep data persistent where it's needed.**
  - Data in GPU memory stays in GPU memory between timesteps.
  - The model should be applicable for other accelerators.
  - Ghost cells can arrive from or be sent to many various sources.

- **Implementation is complicated.**
  - Many of Uintah's strengths come from having one MPI rank per node.
  - Ghost cells in GPU memory need to be buffered into contiguous memory (whether or not CUDA aware MPI is used).

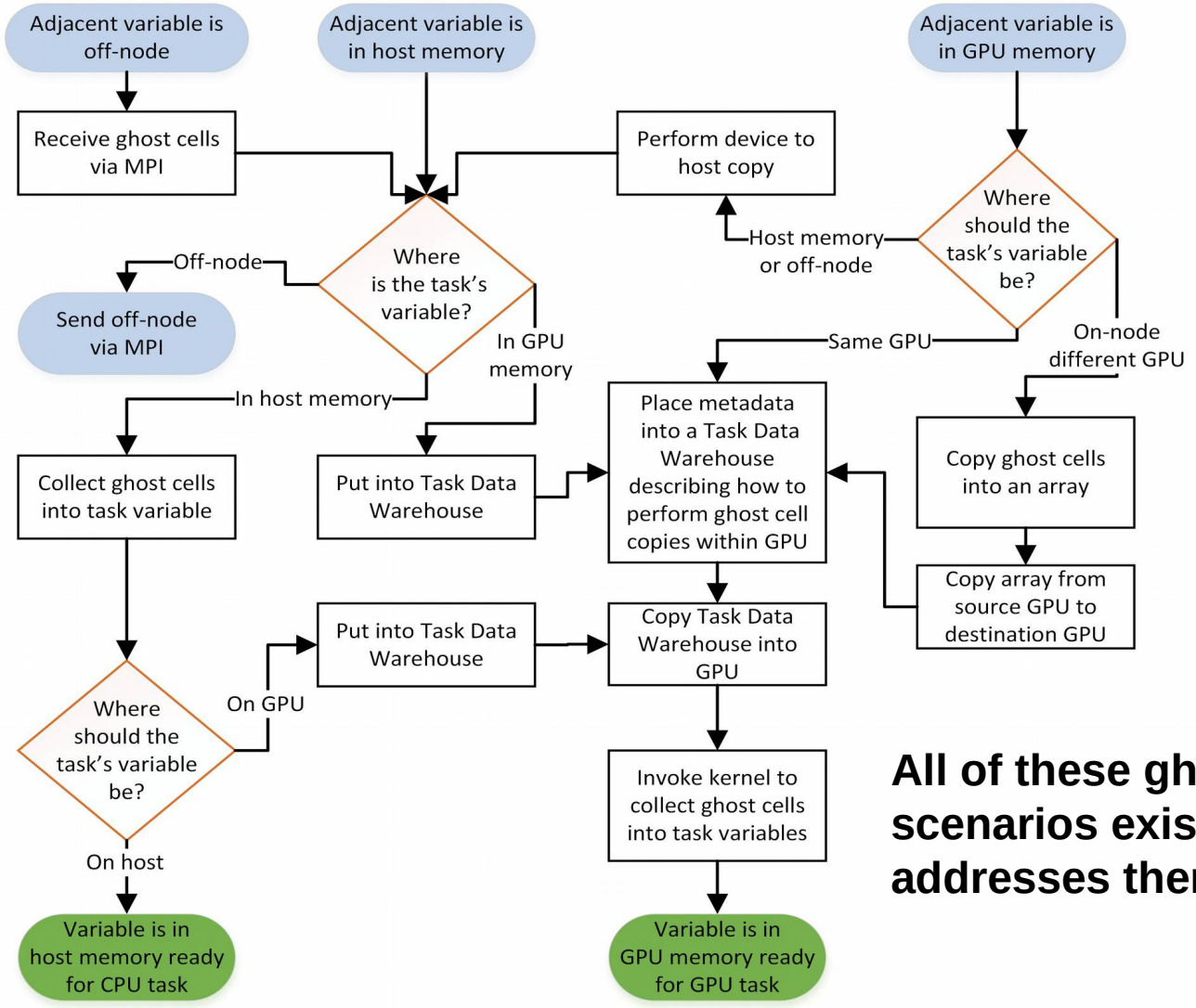
Ghost Cells Management Refactor

- Many new scenarios to manage.
  - Includes intra-GPU ghost cell variable-to-variable copies
  - Includes on-node GPU to GPU copies.
  - Scenarios can be queued into similar work units





Ghost Cells – Many Complex Scenarios



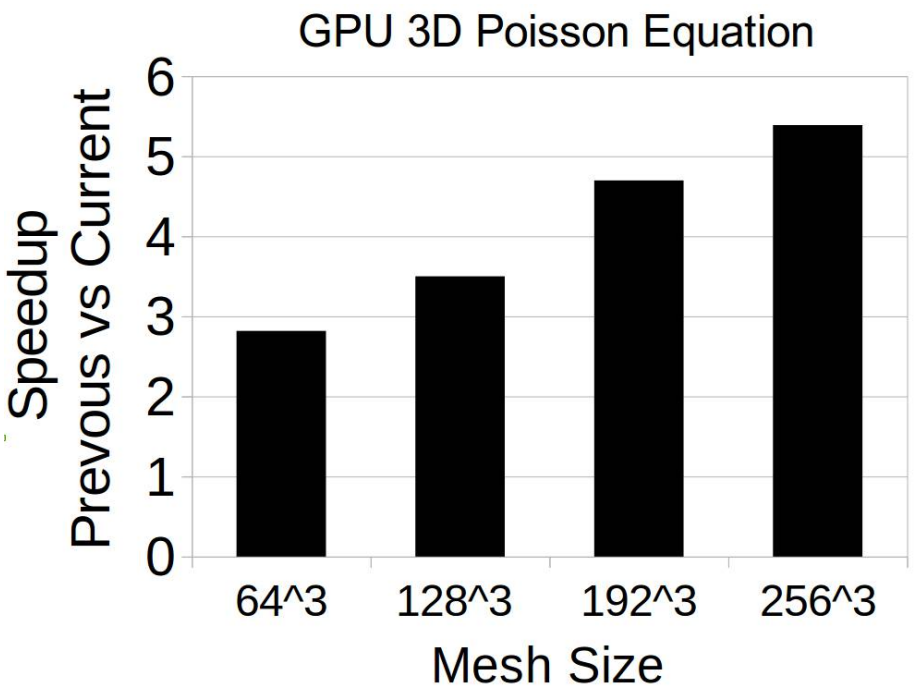
All of these ghost cell scenarios exist, and this work addresses them in Uintah

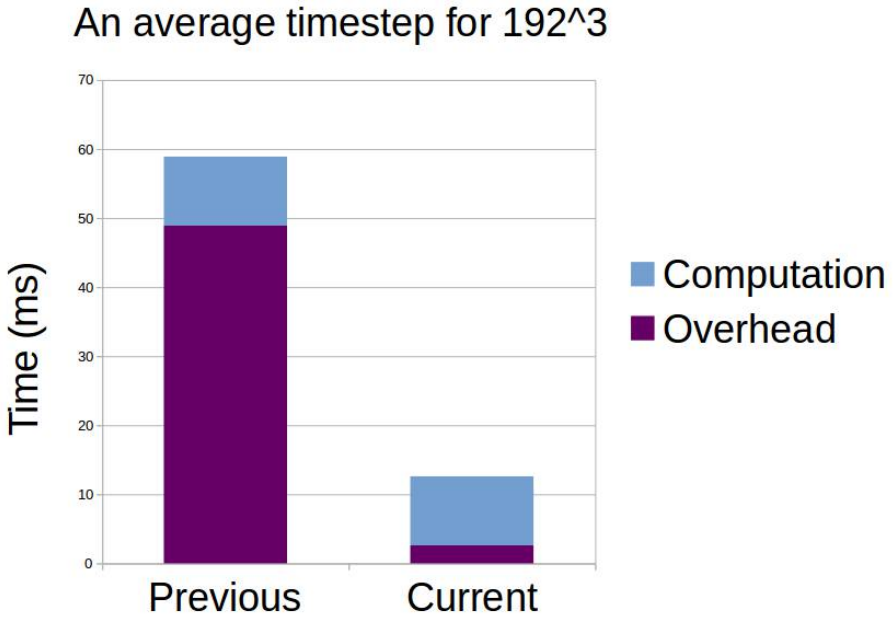
SCI INSTITUTE

THE UNIVERSITY OF UTAH

Results – 3D Poisson Equation

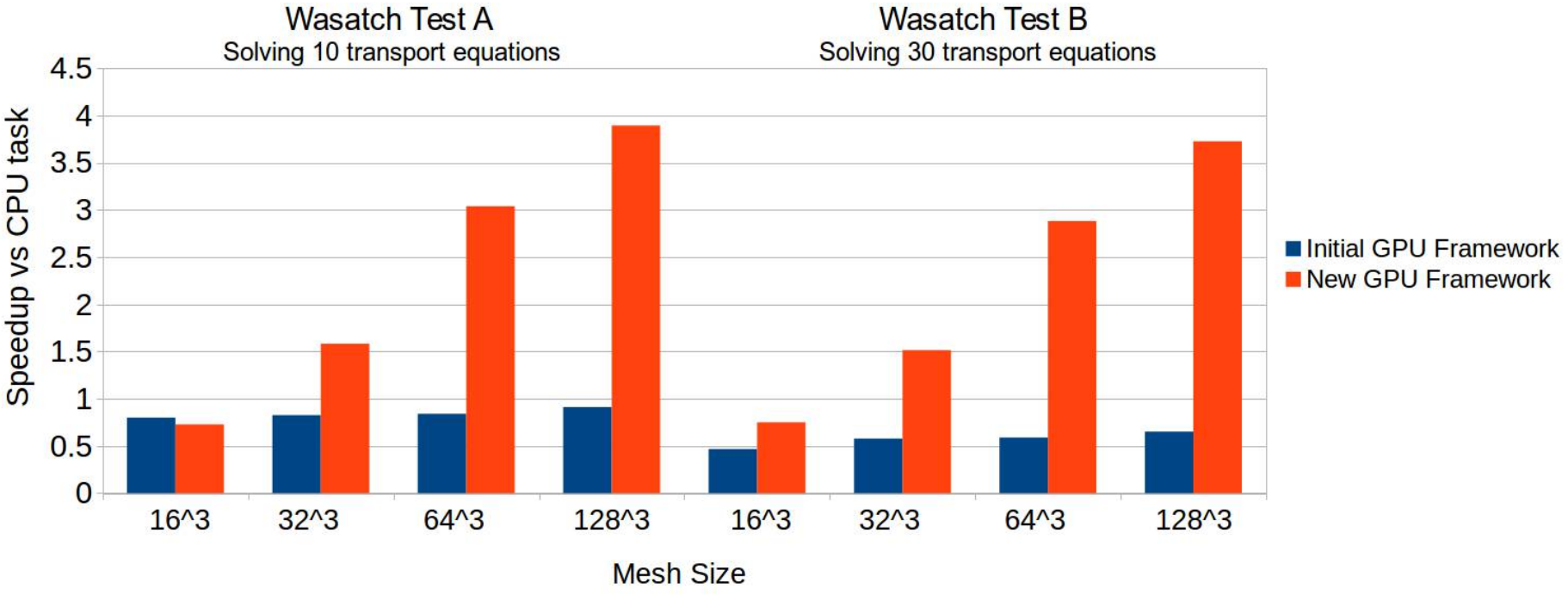
- One of Uintah's hardest tests for GPU performance.
- Good “stress” test. Just a simple stencil computation per timestep.





Results – Wasatch Scalability Tests

- In the 32^3 cases:
  - Before it was faster to run on the CPU
  - Now it is faster to now run on the GPU



Conclusions

- Uintah provides powerful abstraction for solving challenging engineering problems
- Shields application developers from challenges in heterogeneous systems
- This work significantly reduces overhead in the original GPU engine
- Allows a much broader range of computational tasks to leverage GPUs
- Wasatch simulations weak scale to 12,800 GPUs on Titan
- Naturally extends to other accelerators such as Xeon Phis
- Sets the stage for integration with performance portability libraries such as Kokkos

The work of Peterson, Dasari, Berzins and Sutherland was funded by NSF XPS Award 1337135 and which benefited from background work on the Uintah framework and Wasatch by Humphrey, Saad, and Sutherland. This material is based upon work supported by the Department of Energy, National Nuclear Security Administration, under Award Number(s) DE-NA0002375. We would also like to thank all those involved with Uintah past and present, Qingyu Meng in particular.