Uintah Task Engine Optimizations for GPU-Heterogeneous Architectures

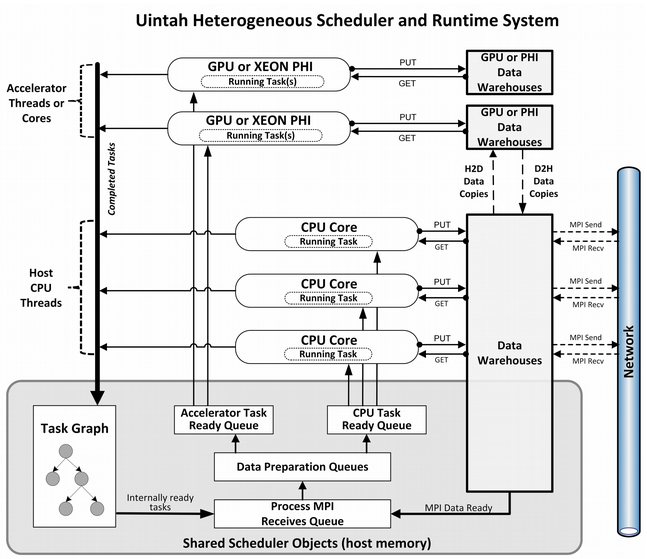
Brad Peterson, Harish Dasari, Nan Xiao, and Martin Berzins
Scientific Computing and Imaging Institute University of Utah

Motivation

To reduce time overhead of Uintah's runtime system, and to better prepare Uintah for accelerator tasks.

- Some fast GPU tasks weren't efficiently managed.
  - Either the PCIe bus was used too often.
  - Or many GPU API calls were slower than computation.
- Creating a full task graph was also unoptimized.

Supporting a heterogeneous mixture of tasks



Tasks:

- **Can be CPU or accelerator tasks.**
- **Can run on many nodes and share halo/ghost cells.**
- **Will require between one to dozens of variables.**
- **Take between 1 ms to over 1 s to compute.**

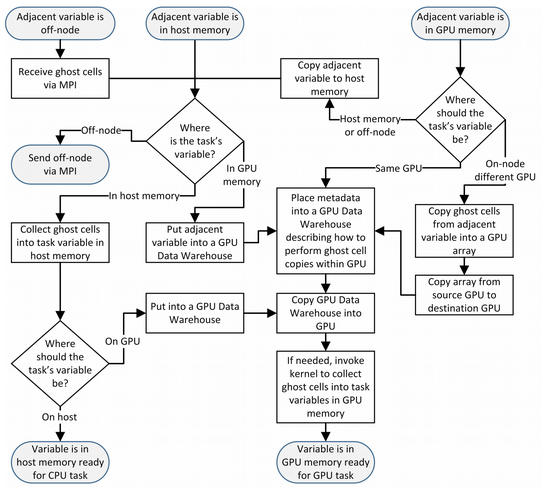
Supported By:

This material is based on upon work supported by the NSF XPS Award 1337135

- *This research utilized equipment donations to the University of Utah's Intel Parallel Computing Center at the SCI Institute.*
- *We wish the acknowledge the prior work and support from Alan Humphrey and Qingyu Meng.*

Supporting short-lived GPU tasks

- Data should stay resident in the GPU as long as possible.
- Many new halo/ghost cells scenarios for a heterogeneous mix of tasks.
- Ghost cells can be sent to and from adjacent regions in host memory, GPU memory, or off node.

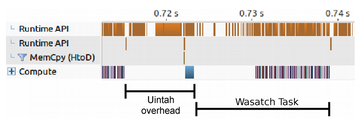


This also means ghost cells can now be managed in groups utilizing a pipeline of work queues.

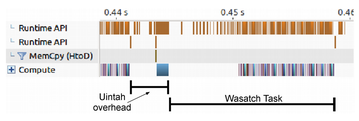
GPU tasks with many variables

- Too many API calls creates large overhead.
- For example, a single CUDA malloc can take longer than computing some GPU task kernels.
- We allocated large GPU memory spaces to hold all variables, instead of allocating one space per variable. Overhead decreases between 1.27x and 2.00x observed.

Before



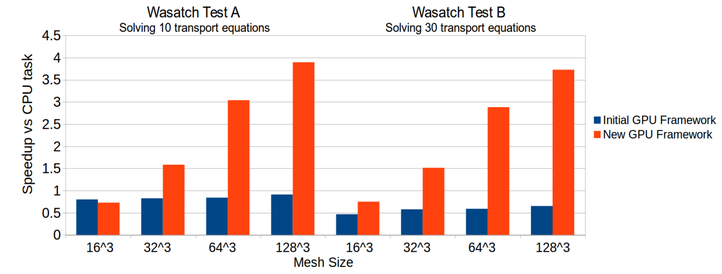
New



A GPU task for the Wasatch component of Uintah requires dozens of computational variables and processes in under 2 milliseconds. The figures above show a decrease in Uintah overhead by allocating all variables into one contiguous memory space.

Results for GPU tasks

- For GPU tasks that compute within milliseconds and/or tasks that require many variables, previously Uintah framework overhead was the dominant factor.



This reduced overhead opens up broad families of new tasks to obtain speedups utilizing Uintah's heterogeneous runtime support.

Faster task graph creations

- Before a simulation starts, and occasionally before any timestep, a full task graph is created. This can be expensive for large problems, on the order of hundreds of seconds!
- The previous approach used a BVH tree and organized data through frequent (and sometimes duplicative) sorting.
- New data structures, such as a set of specialized hash trees, were created. Sections of the mesh grid are organized more compactly and structured according to their region.
- This allowed for faster data structure creation times and faster queries. This ultimately led to observed reductions in task graph creation times by up to 45%.