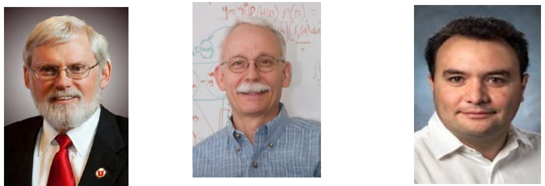
An Evaluation of An Asynchronous Task-Based Dataflow Approach for Uintah

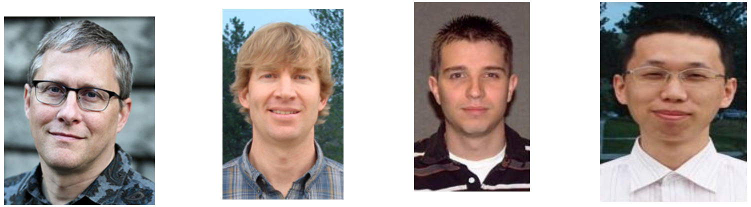
Martin Berzins Alan Humphrey

1. Introduction to dataflow and Uintah
2. AMT Uintah, runtimes and programming models
3. Uintah AMR Algorithms and the need for AMT
4. Scalability Evaluation
5. Conclusions

Uintah Background and Acknowledgements DOE NSF People

- DOE ASC Strategic Academic Alliance Program 1998 -2010
- ALCC and Directors Discretionary time awards
- INCITE (4 awards 700M cpu hours in total)
- Argonne , Oak Ridge and NNSA Facilities
- **NNSA PSAAP2 center funding 2014-2020**
- Argonne A21 Exascale early science program
- Sandia Kokkos group and Livermore Hypre Group
- NSF software funding and Peta-Apps 2007- 2015
- NSF XSEDE TACC Blue Waters computer time and facilities
- The 50 or so people on Uintah and its related projects, since 2003 particularly The Uintah "wizards" Steve Parker, Justin Luitjens, Qingyu Meng and Alan Humphrey .
- NNSA PSAAP2 Co PIs Dave Pershing, Phil Smith Valerio Pascucci





Dataflow Origins and Developments

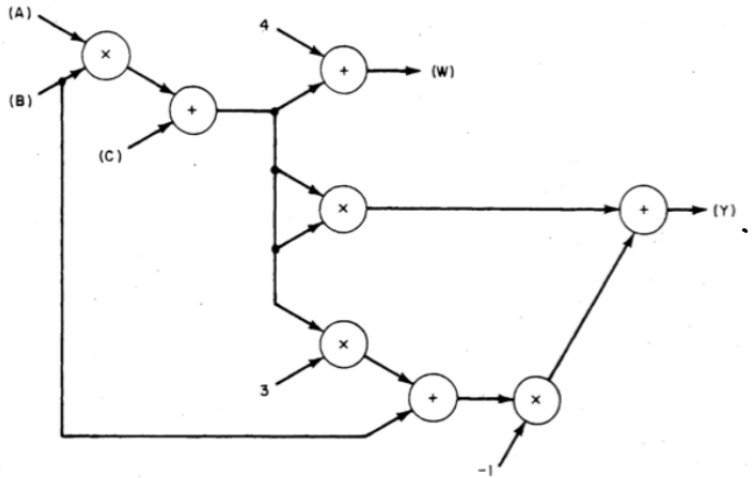
Original Dataflow Sutherland 1966

WRITTEN STATEMENT

$Z = A \times B + C$

$W = Z + 4$

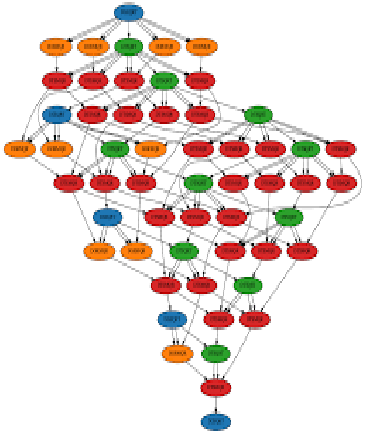
$Y = Z^2 - (3Z + B)$



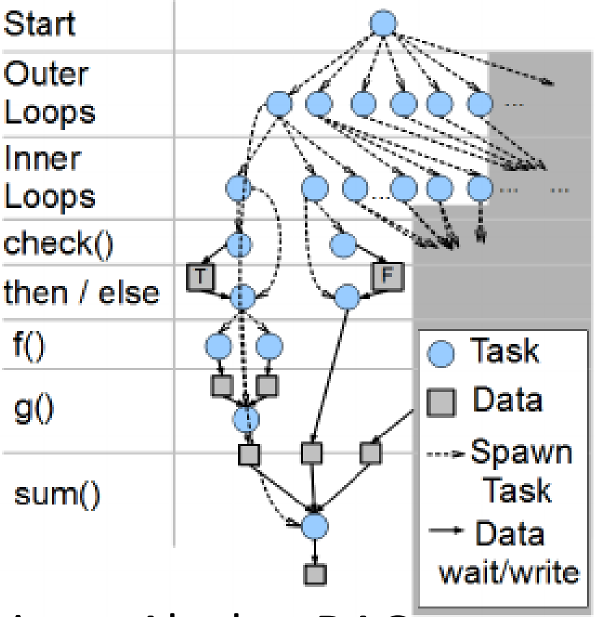
GRAPHICAL STATEMENT

Vivek Sarkar’s Thesis 1980s

```
int X = 1000, Y = 1000;
int A[][];
int B[];
foreach x in [0:X-1] {
foreach y in [0:Y-1] {
  if (check(x, y)) {
    A[x][y] = g(f(x), f(y));
  } else {
    A[x][y] = 0;
  }}
B[x] = sum(A[x]);
}
```



Swift Dataflow System



Linear Algebra DAG Dataflow System

Asynchronous Many Task Runtime Systems

SC16 Survey by Thomas Sterling

Darma – Sandia Labs
Legion – Stanford
Charm++ Illinois
Uintah - University of Utah
STAPL – Texas A & M
OCR - Rice
Qthreads – Sandia
LFRIC - UK met Office
PaRSEC - Tennessee
StarPU - Barcelona/INRIA
HTGS - NIST
FleCSI - LANL
HPX - Indiana / Louisiana

https://www.youtube.com/watch?v=rStVp19tXqk

Key features:

Adaptive execution of tasks

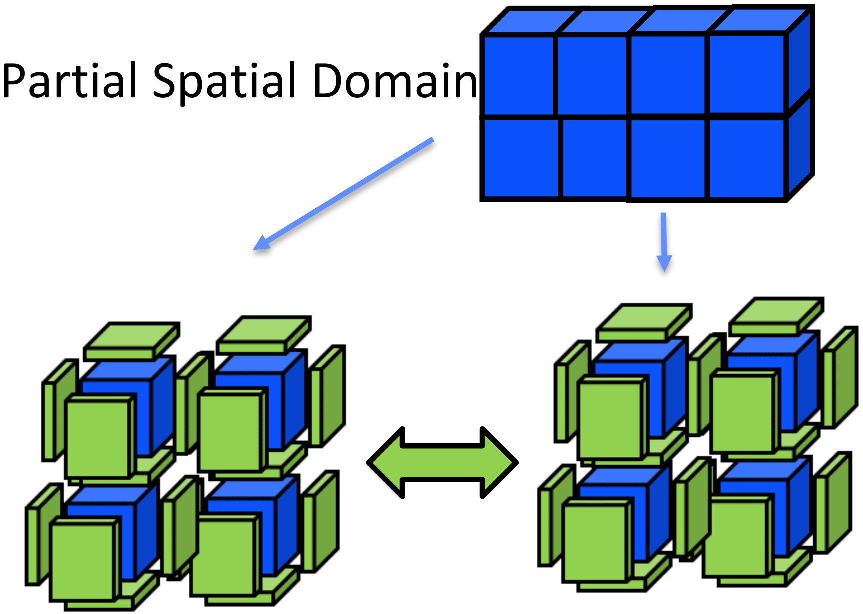
Ability to hide communications costs including delays

Ability to address heterogeneity

Task specification may not change as code ported , even though some of runtime does

Many US activities have some DOE funding

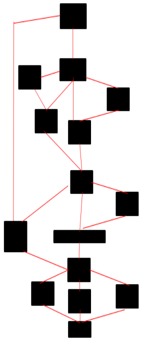
Scientific Computing Partial Differential Equations Dataflows Have a Particular Spatial Structure



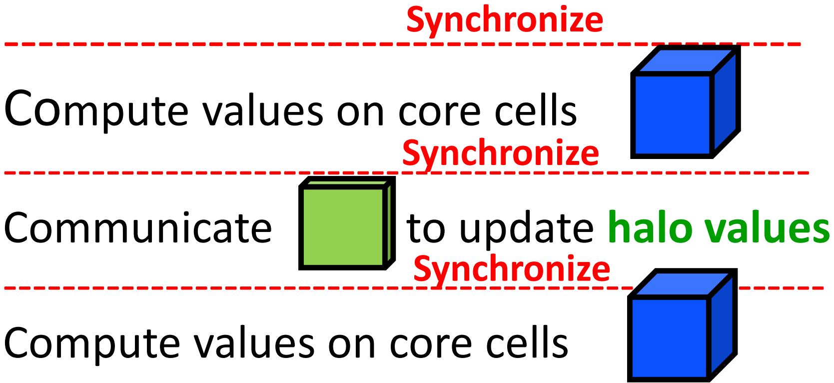
Partial Spatial Domain

Decomposed onto
Two compute nodes

🟦 **Mesh Blocks**
With SAME
Dataflows



🟩 **Halo Communications between mesh blocks**



Synchronize

Compute values on core cells 🟦

Synchronize

Communicate 🟩 to update **halo values**

Synchronize

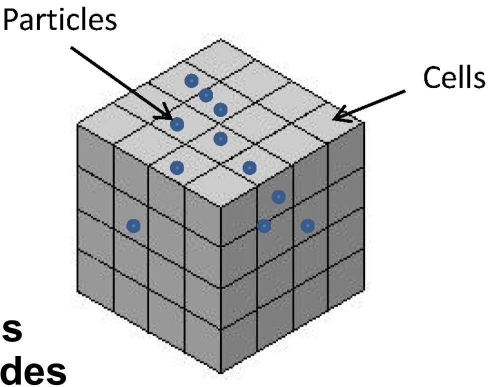
Compute values on core cells 🟦

Uintah ARCHES MPM-ICE-AMR Software

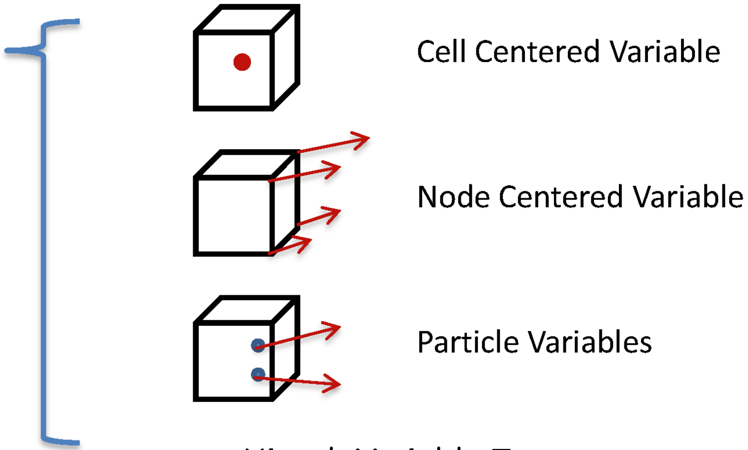
MPM (solids) and ICE (fluids) exchange data several times per timestep (not just boundary condition exchange

Arches finite volume combustion code with a low-Mach number approximation Particle methods and LES algorithms

ICE is a cell-centered finite volume method for Navier Stokes equations



Uintah Patch

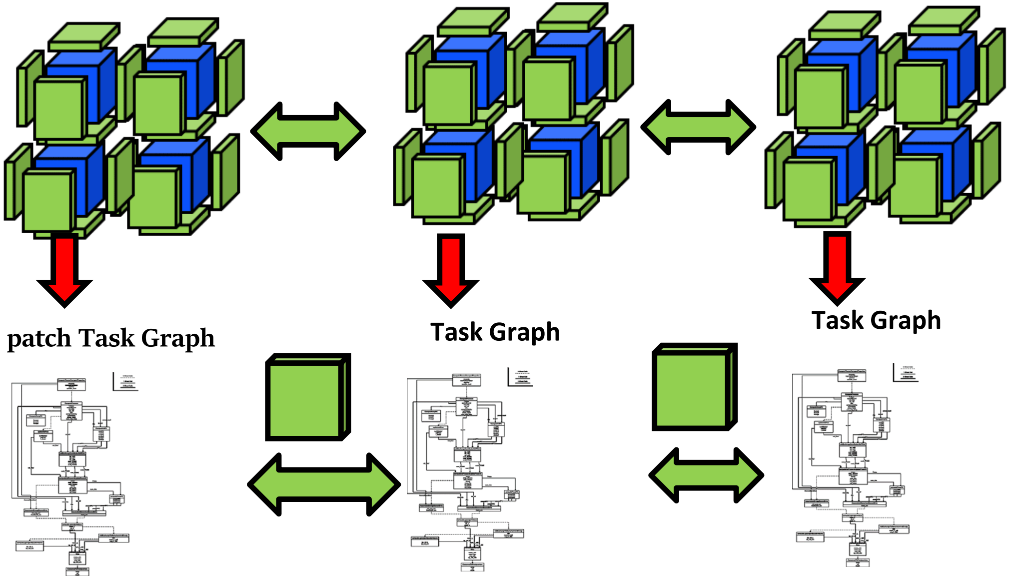


Uintah Variable Types

MPM is a novel method that uses particles and nodes Cartesian grid used as a common frame of reference

Uintah Asynchronous Many Task (AMT) Approach 2010…

e.g. three compute nodes 12 mesh patches



In Uintah each mesh patch has its own graph weakly coupled to others
Tasks are typically 50K to 100K flops

Uintah Architecture Overview

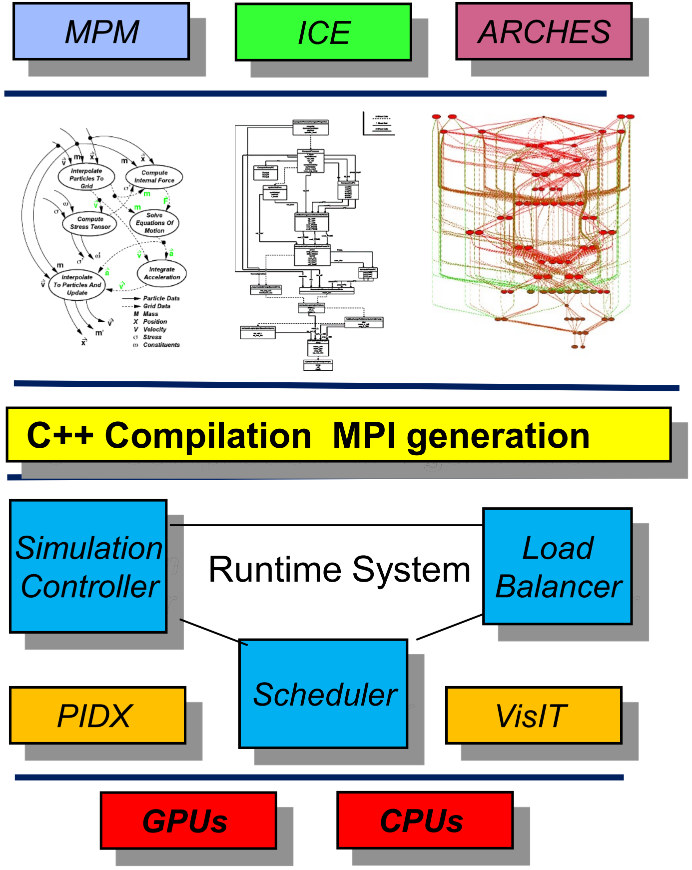
Applications code: about 800K lines written as tasks in Uintah programming model form

Abstract C++ Task Graph Form

Fully automated MPI message generation

Static or Adaptive Execution of Tasks

On specific cores/processors



MPM

ICE

ARCHES

C++ Compilation MPI generation

Simulation Controller

Runtime System

Load Balancer

Scheduler

PIDX

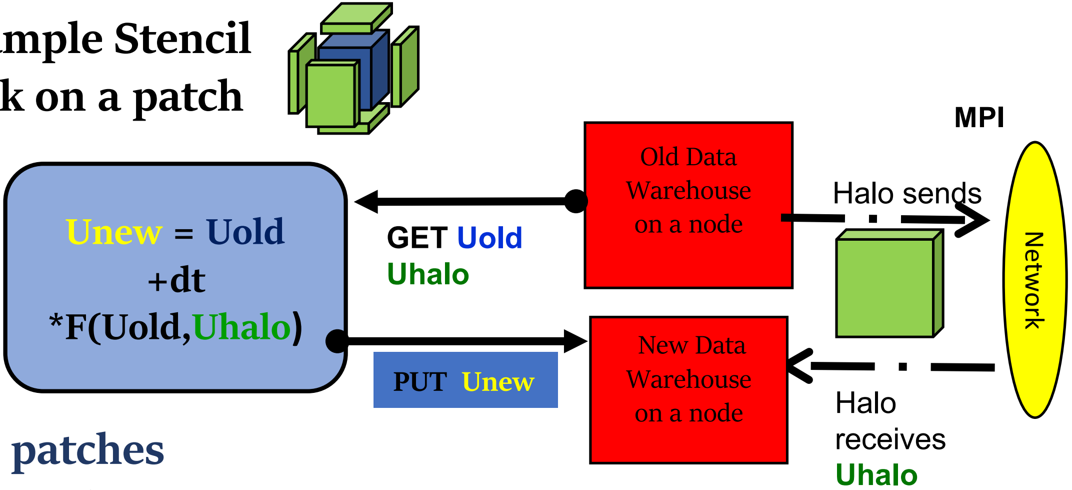
VisIT

GPUs

CPUs

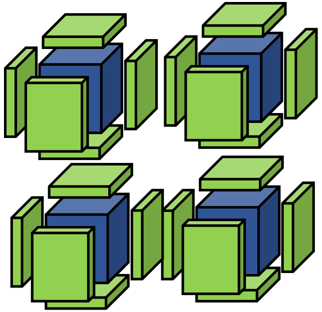
Uintah Programing Model for Stencil Timestep [Parker 1998]

Example Stencil
Task on a patch

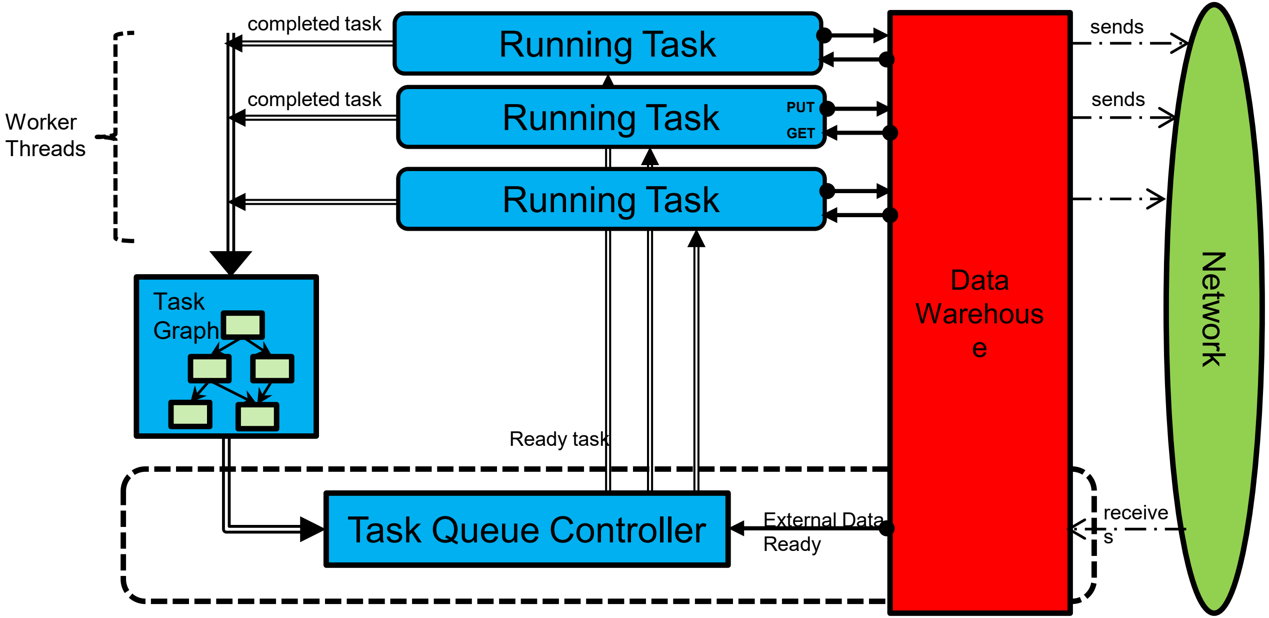


User specifies **mesh patches**
halo levels and connections

Clean separation between physics and CS runtime system
Problems specified 15 years ago run on todays architectures with one significant change as we go to Exascale we use Kokkos to get nodal performance.



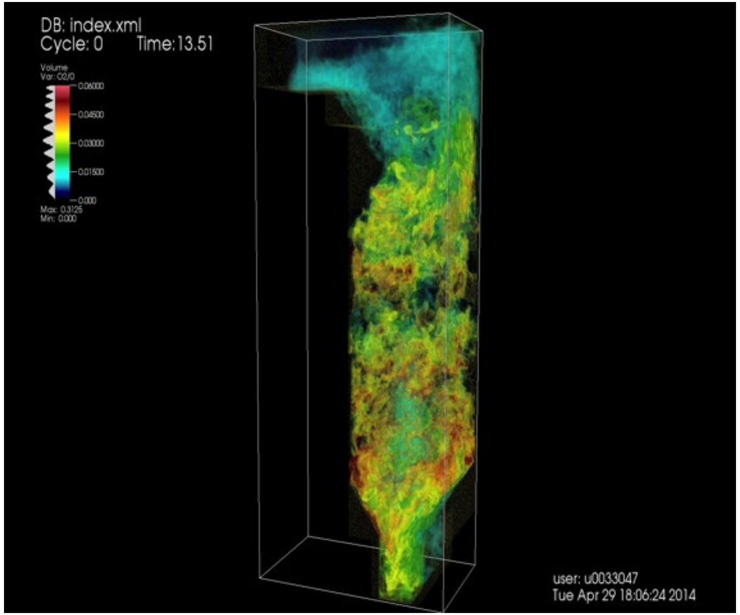
Uintah MPI Task Scheduler on a Compute Node

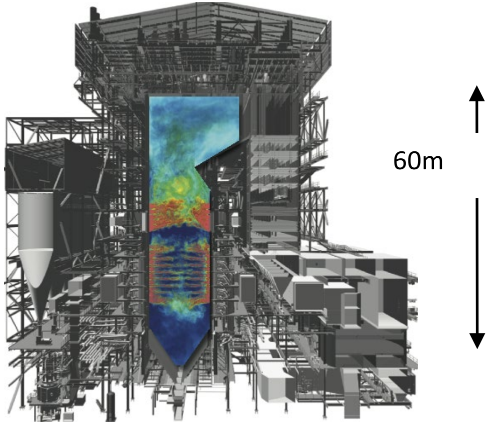


- MPI Tasks linked to cores. Tasks in MPI DAG execute when ready on that core.
- Uses DAG and Asynchronous MPI execution
- Different DAG execution policies (e.g. most connections first) may not always make much difference – the evidence is mixed

NNSA PSAAP2 Existing Simulations of GE Clean(er) Coal Boilers

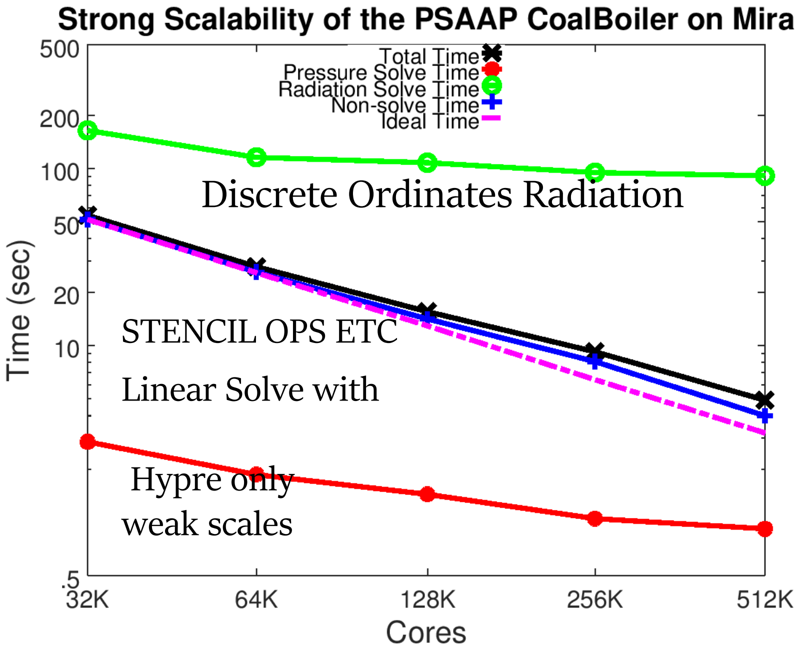
- Large scale turbulent combustion needs mm scale grids 10^14 mesh cells 10^15 variables (1000x more than now)
- Structured, high order finite-volume discretization
- Mass, momentum, energy conservation
- LES closure, tabulated chemistry
- PDF mixing models
- DQMOM (many small linear solves)
- Uncertainty quantification

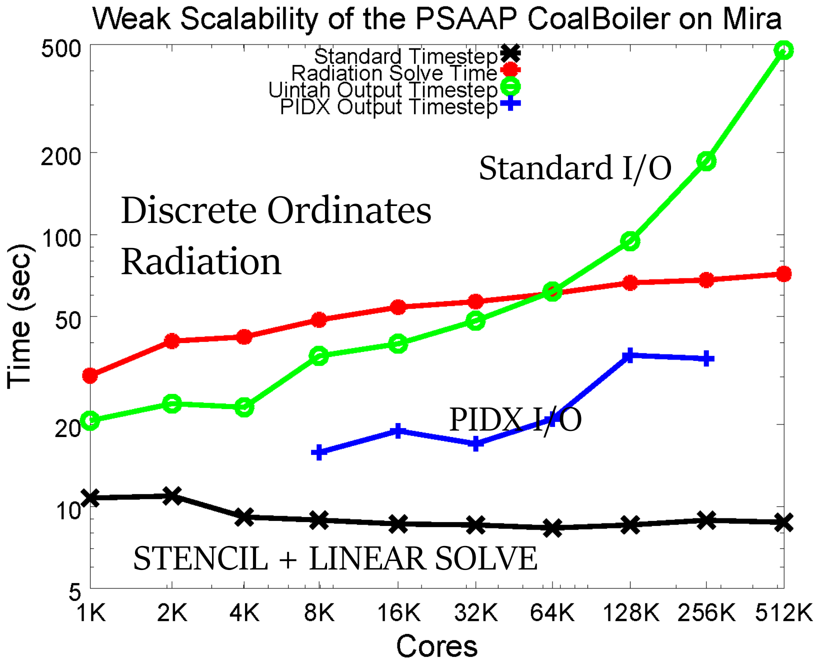




- Low Mach number approx. (pressure Poisson solve up to 10^12 variables. 1M patches 10 B variables
- Radiation via Discrete Ordinates – many hypre solves Mira (cpus) or ray tracing Titan (gpus).
- FAST I/O needed PIDX

For fixed mesh calculations Uintah scales for the Boiler using MPI Scheduler.

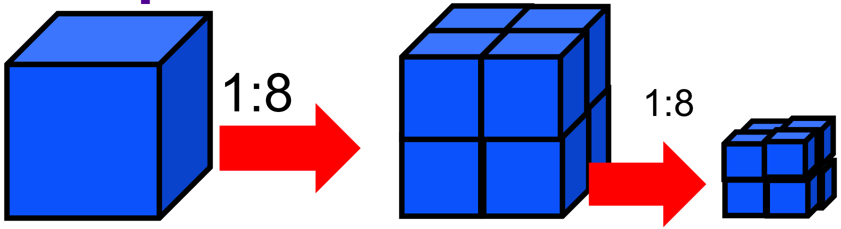




Full physics multi-level GPU-RMCRT strong scales on Titan

[Thornock, Schmidt Kumar Harman Humphrey]

Improved Accuracy via Scalable Adaptive Mesh Regridding Algorithm



Tiles that contain flags are refined

Simple and easy to parallelize

Levels of patches independently refined

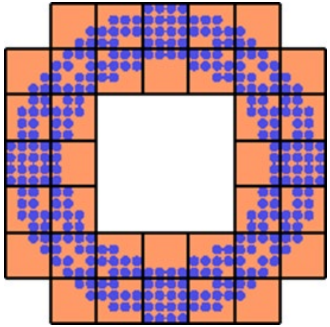
BVH tree used to find patches

Very robust and successful.

1:8

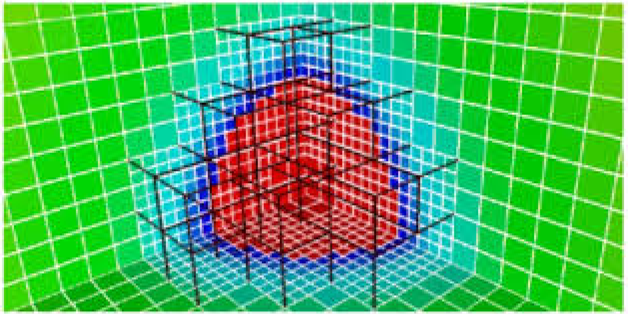
(i) Fast space filling curves [Luitjens Thesis 2011]*
(ii) Tiled regular refinement
(iii) Data assimilation based workload prediction and rebalancing.
(iv) Works with Fluid-structure interaction [Qingyu Meng Thesis 2014]*
and Raytracing radiation [Humphrey Thesis 2019]*

(ii) **DYNAMICALLY CHANGES DAG AT DISCRETE TIMES**



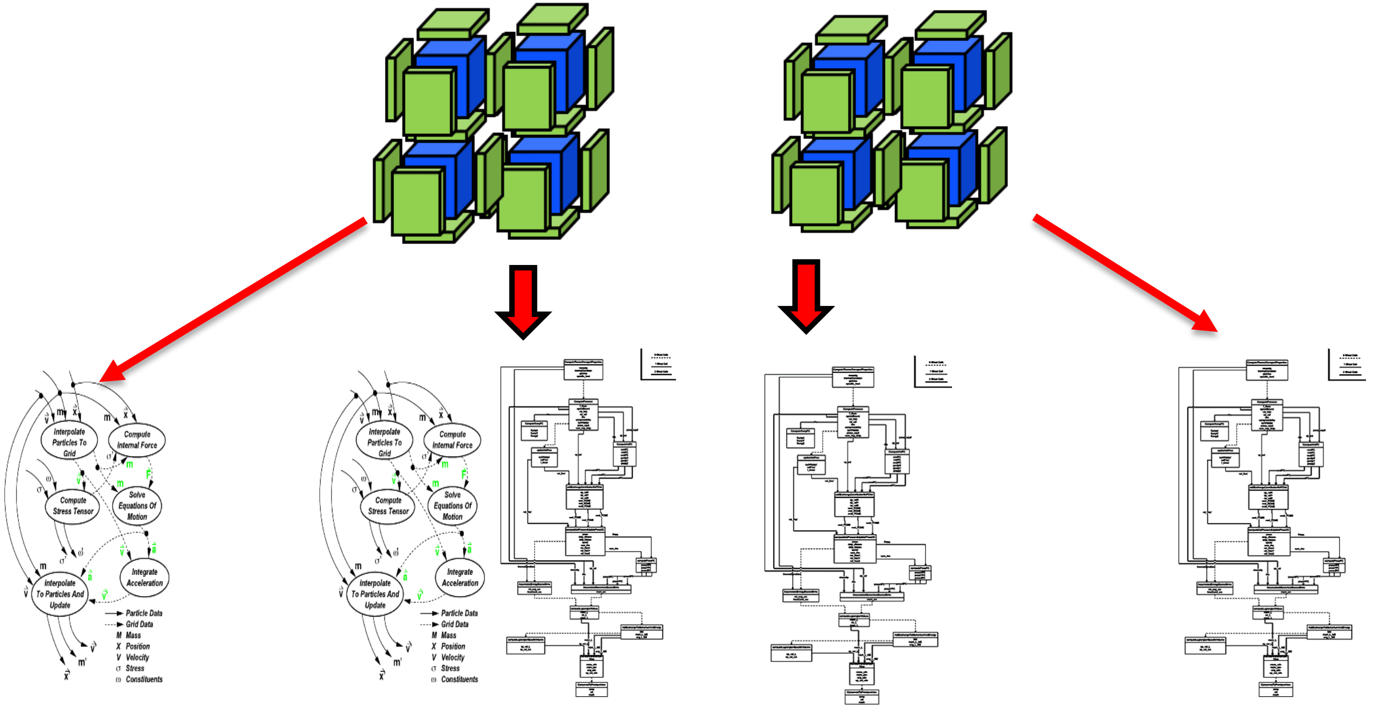
2D illustration Refinement flags and patches

3D Example



* Available from http://www.sci.utah.edu/publications

Fluid Structure Interaction – Some Mesh Patches are Fluids and some are Particles. At the interface we need both DAGs



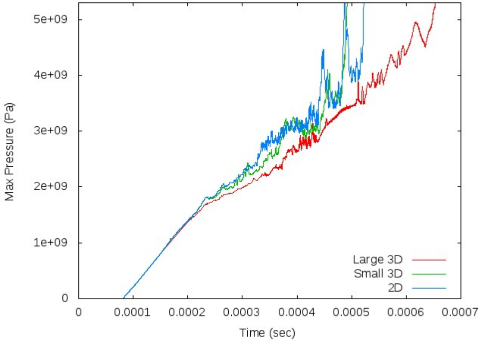
Particles Particles and Fluid Fluid Fluid

NSF funded modeling of Spanish Fork Accident 8/10/05

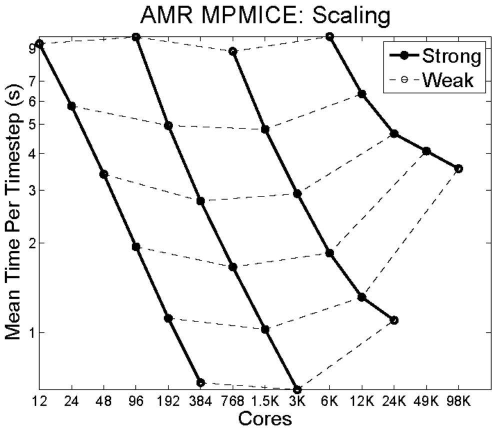
Speeding truck with 8000 explosive boosters each with 2.5-5.5 lbs of explosive overturned and caught fire

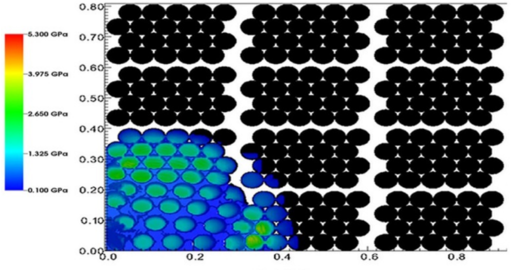
Experimental evidence for a transition from deflagration to detonation?

Deflagration wave moves at ~400m/s not all explosive consumed. Detonation wave moves 8500m/s all explosive consumed.

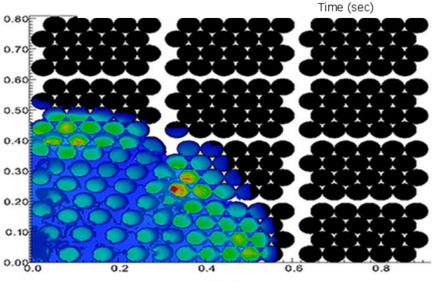


Static MPI Scheduler doesn't scale

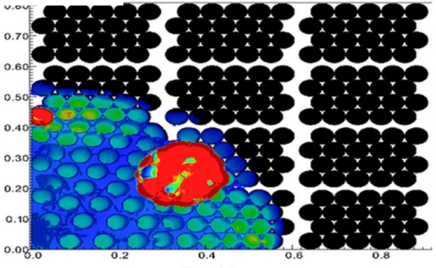




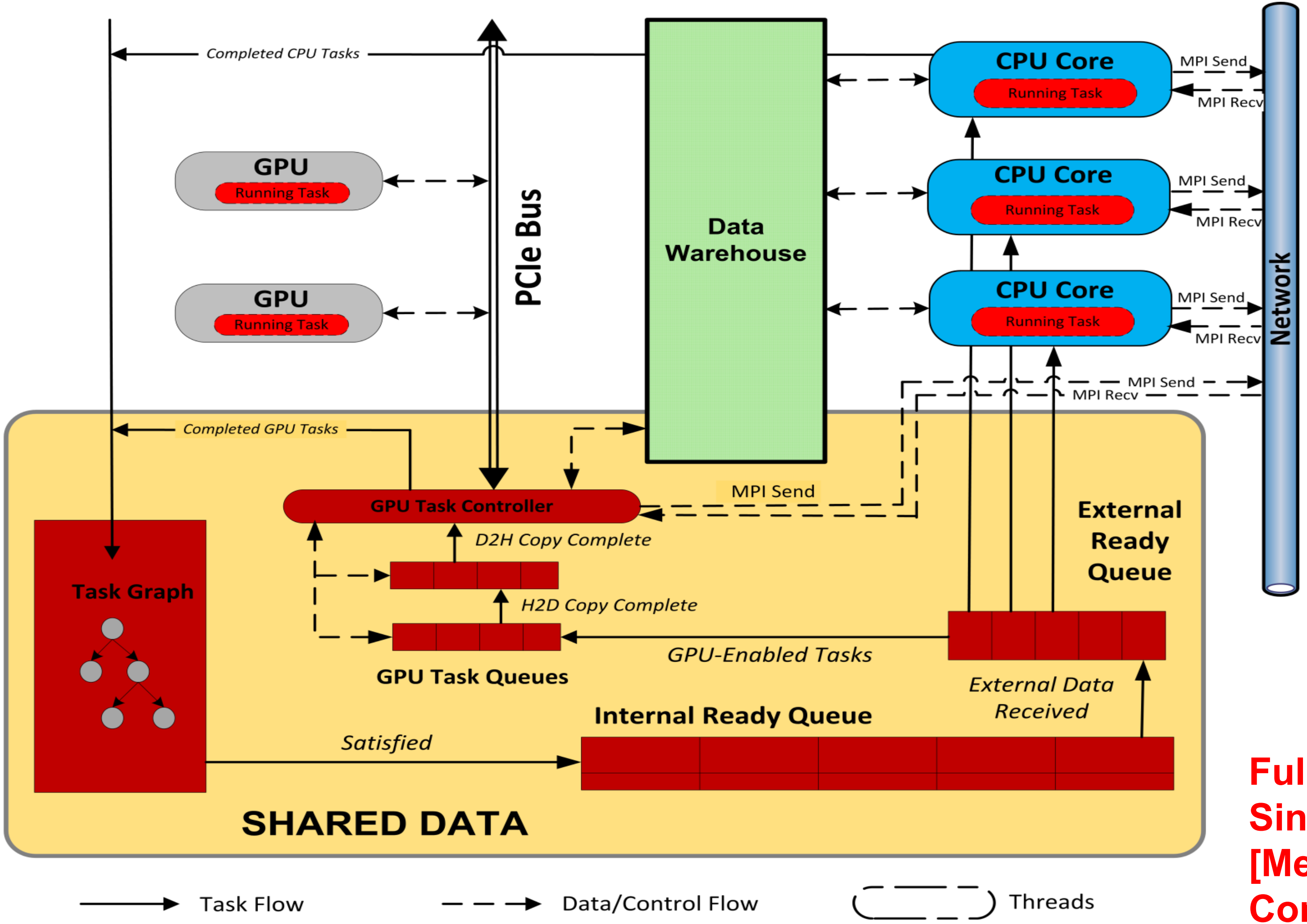
0.534 msec



0.634 msec



0.644 msec



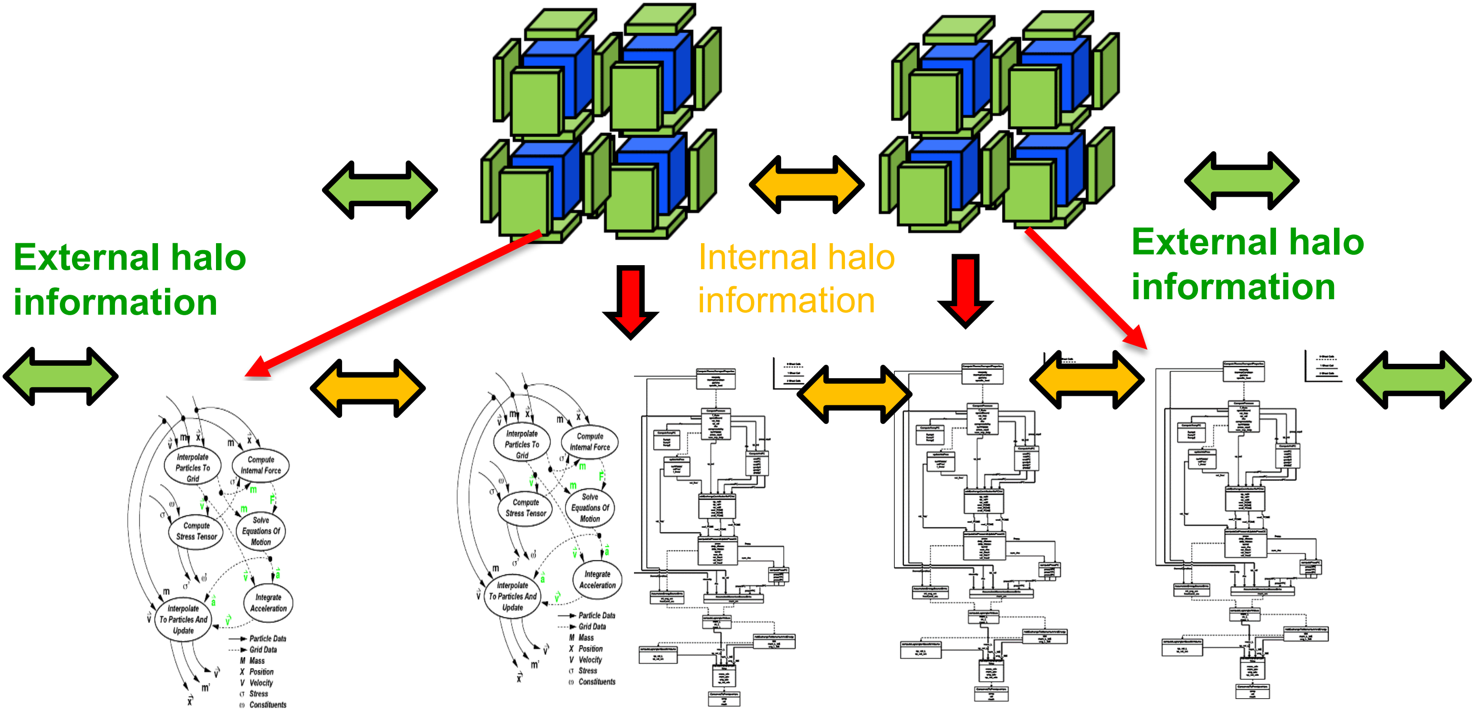
Nodal Shared Memory Model

Unified Heterogeneous Scheduler & Runtime [2012]

Fully Asynchronous Since 2012 [Meng and Berzins Concurrency 2014]

Over decomposition

One compute core 8 mesh patches consider bottom 4 inner 2 only need internal information



External halo information

Internal halo information

External halo information

Execute tasks from whichever patch has its tasks ready as this avoids delays – prioritize tasks with external communications

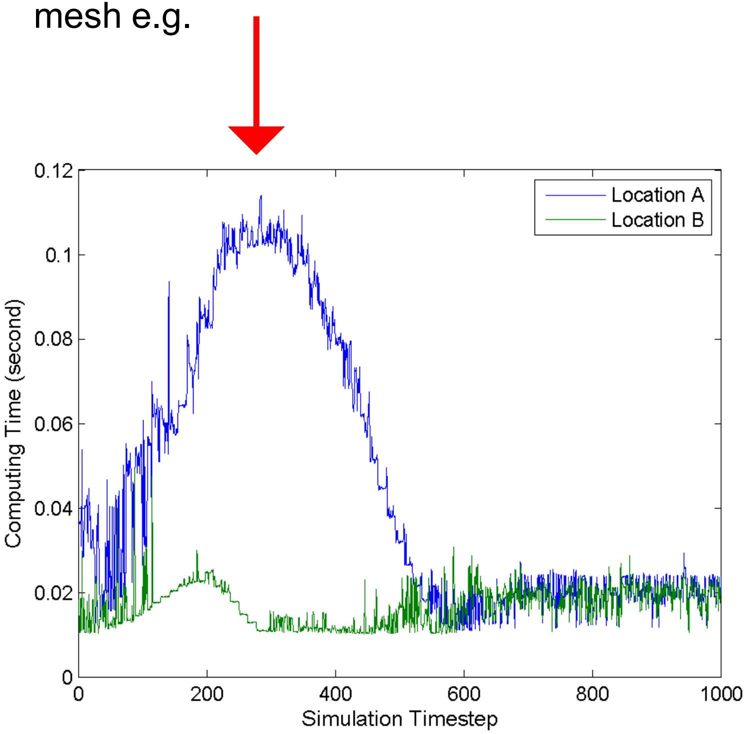
Wait for external halo information

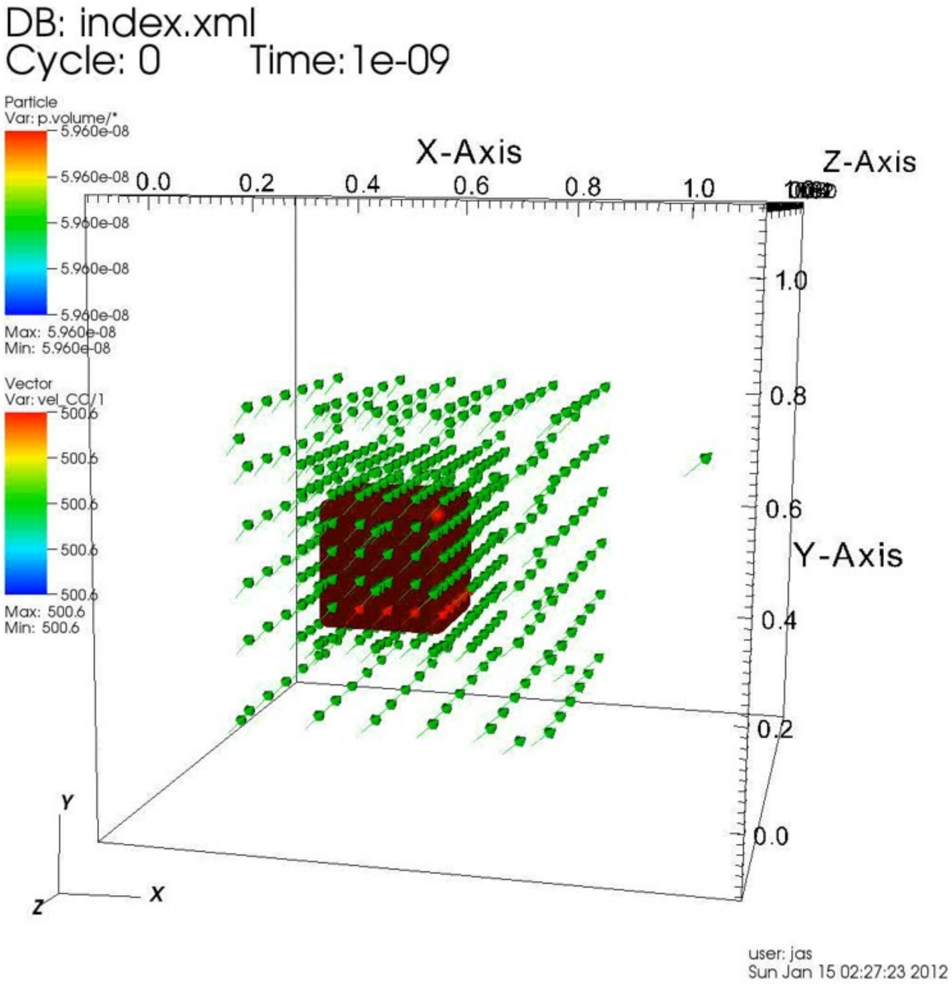
Start immediately

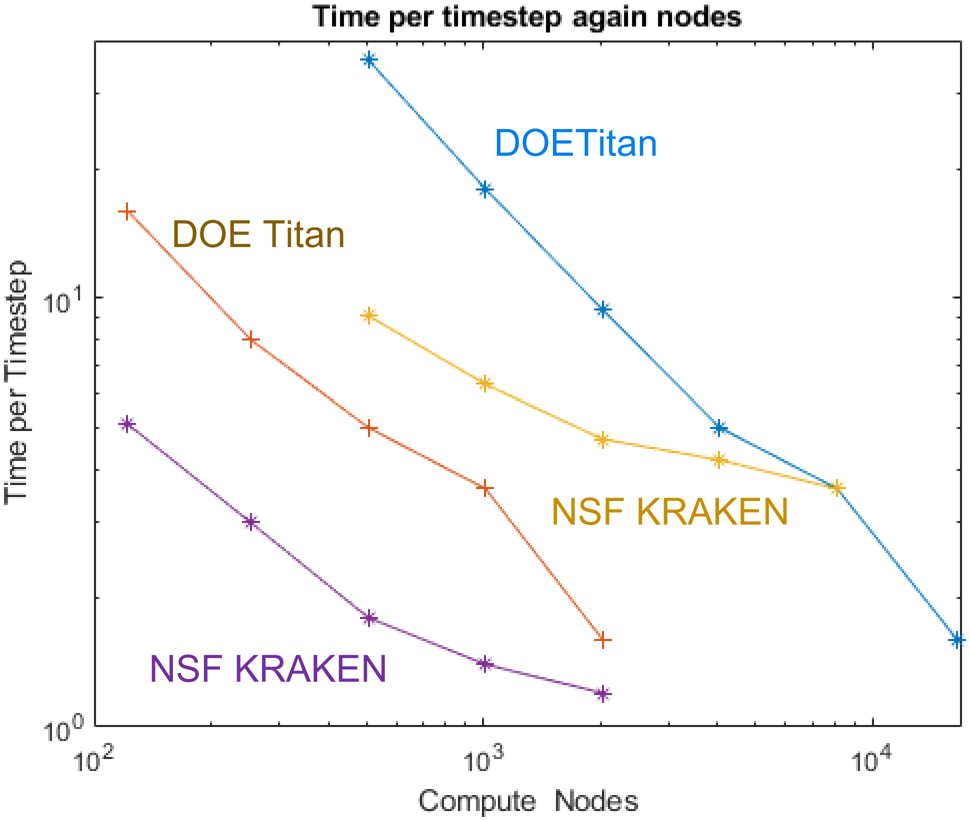
Wait for external halo information

Benchmark Problem

Complex fluid-structure interaction problem with adaptive mesh refinement, following a moving structure consisting of particles . Work varies greatly at each point in the mesh e.g.







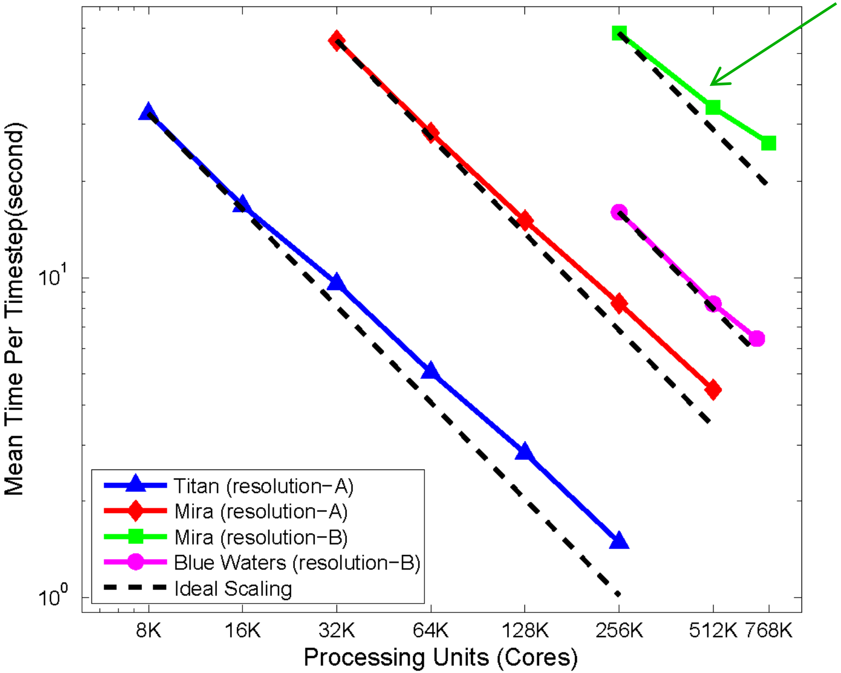
Small core counts

Kraken wins out with Static Approach

Larger core counts

Titan perhaps wins out with dynamic approach

Nodes are either 12x2.6GHz Kraken
Or 16x2.2 Ghz Titan



MPM AMR ICE Strong Scaling [Qingyu Meng 2014]

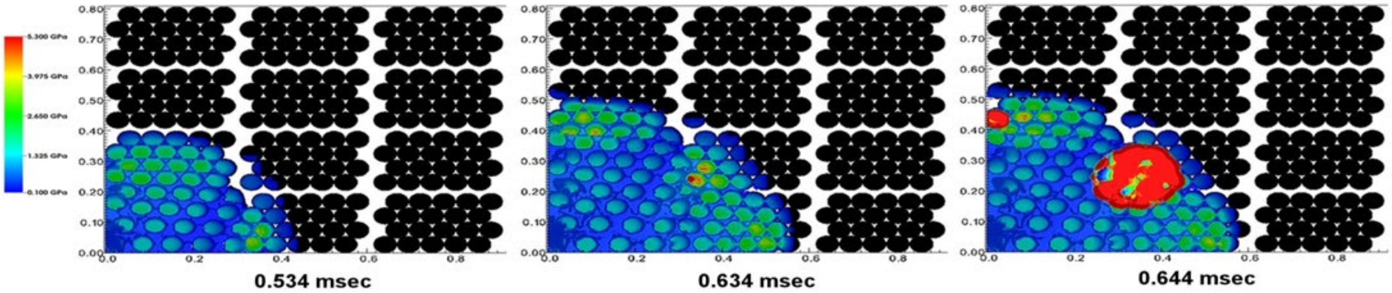
Mira DOE BG/Q
768K cores
Blue Waters Cray
XE6/XK7 700K+
cores

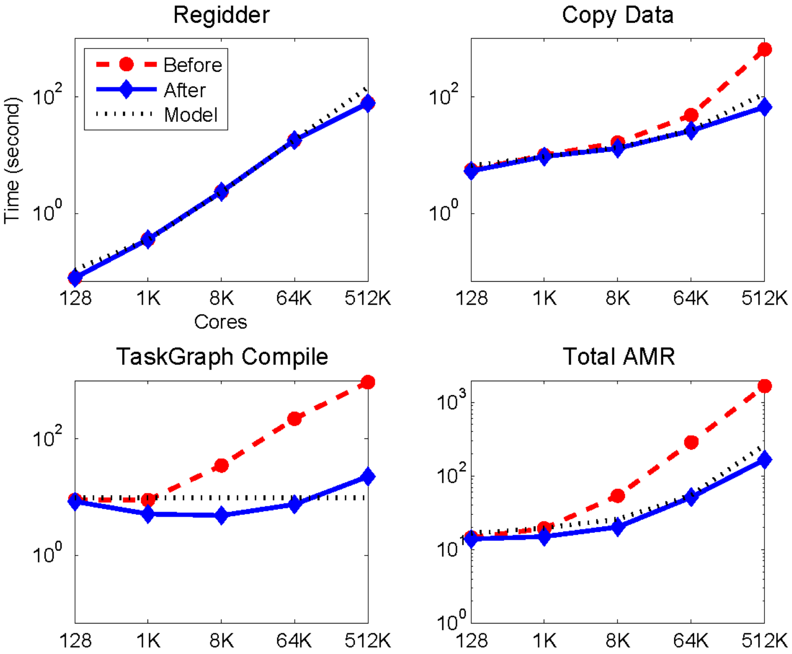
Resolution B
29 Billion particles
4 Billion mesh cells
1.2 Million mesh
patches

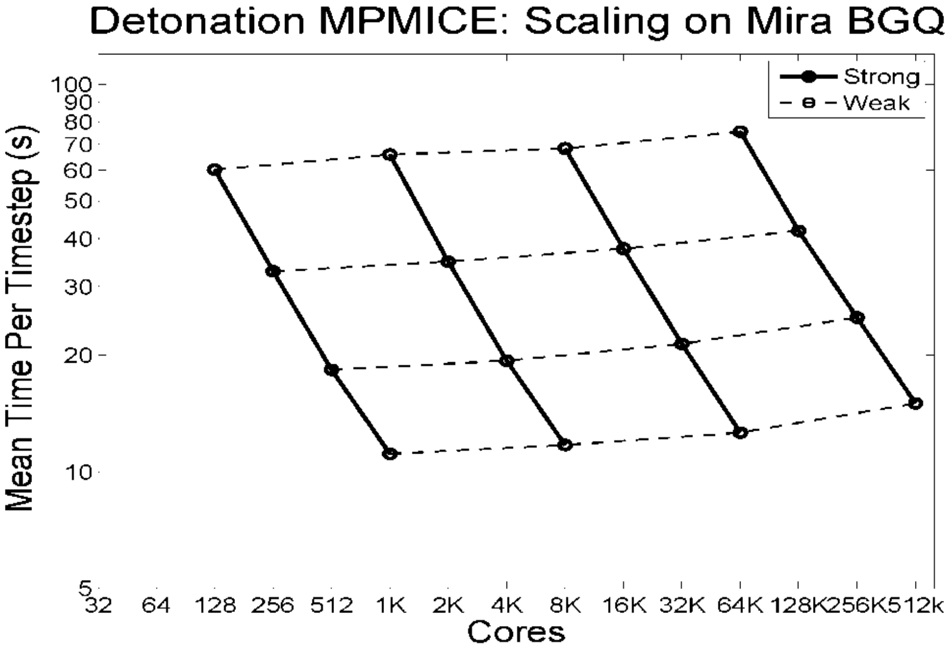
The fluid –structure interaction part of the problem gives very unequal workloads per patch and cannot Be predicted in advance .

Spanish Fork Accident

500K mesh patches
1.3 Billion mesh cells
7.8 Billion particles







Initial Results

- Fast AMR for fluid or fluid/structure interaction
- Unified Scheduler implements full asynchrony
- Care is needed with shared memory approach on a node
- No obvious penalty < 100k cores applied to fluids
- Fluid-structure less clear but good scalability
- Since then many changes to runtime system
- Can new runtime work better with fluid-structure interaction problems?

Unified Scheduler Improvements

DFM’19 Results

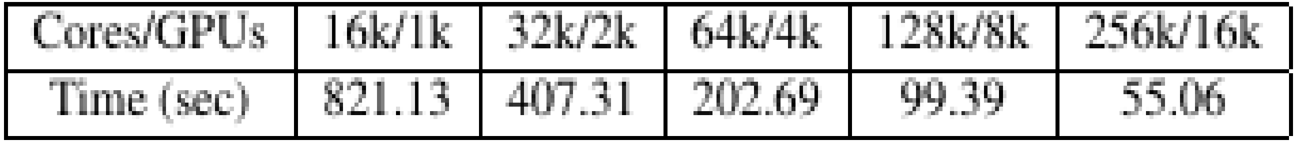
- Improve how threads make an MPI Request -removed code with many locks. Replaced with one instruction e.g. halos
- Many of the operations on Uintah::DependencyBatch must be atomic, but do NOT need to be sequentially consistent. Relaxed memory ordering, in automated MPI engine
- Removed 3-4 usages of std::mutex around std::atomics dealing with external- and internal-ready task queues. All on critical path from all threads on a shared-memory node
- Removed or simplified overly coarse-grained critical sections some of which were in the code critical path, e.g., processing of MPI receives, which for MPI_THREAD_MULTIPLE, encounters locks within the MPI library – was a significant serialization point for code that sees heavy thread traffic

Scheduler Improvements on Titan [Humphrey 2019]

AMR+ICE Benchmark	Res A	Res A	Res B	Res B
Scheduler Cores	MPI	Unified	MPI	Unified
32K	17	14		
64K	13	7	**71.5**	**65**
128K	10	5	**36.7**	**31**
256K	8	3.6	**26**	**17.6**

UNIFIED SCHEDULER FASTER AND SCALES BETTER

Full Boiler Calculation
With Radiation Raytracing



Cores/GPUs	16k/1k	32k/2k	64k/4k	128k/8k	256k/16k
Time (sec)	821.13	407.31	202.69	99.39	55.06

Summary

Static DAG execution with asynchronous communications works for

I. Standard (even if complex) stencil codes

II. Models that have modest memory needs and don't need large shared data

III. Models with predictable workloads

Dynamic DAG Execution makes possible to:

(i) Solve problems with large shared memory or memory that is too big for one MPI process

(ii) Solve problems with adaptive unpredictable workloads . AMR, multiple physics etc

(iii) Care is needed to manage shared memory on a node and thread MPI interactions