Uintah

Uintah

1. A parallel mesh library aimed at solving partial differential equations
2. Features include
   - 2.1 Adaptive mesh refinement
   - 2.2 Particles
   - 2.3 An advanced task-graph system
   - 2.4 Fully parallel execution
   - 2.5 Parallel Hilbert space filling curves
   - 2.6 Support for Hypre linear solver developed at LLNL
   - 2.7 A multitude of mesh operations (interpolation, fetching and operating on ghost cells, ..)

A small list of other mesh libraries

1. SAMRAI
2. BoxLib
3. Chombo

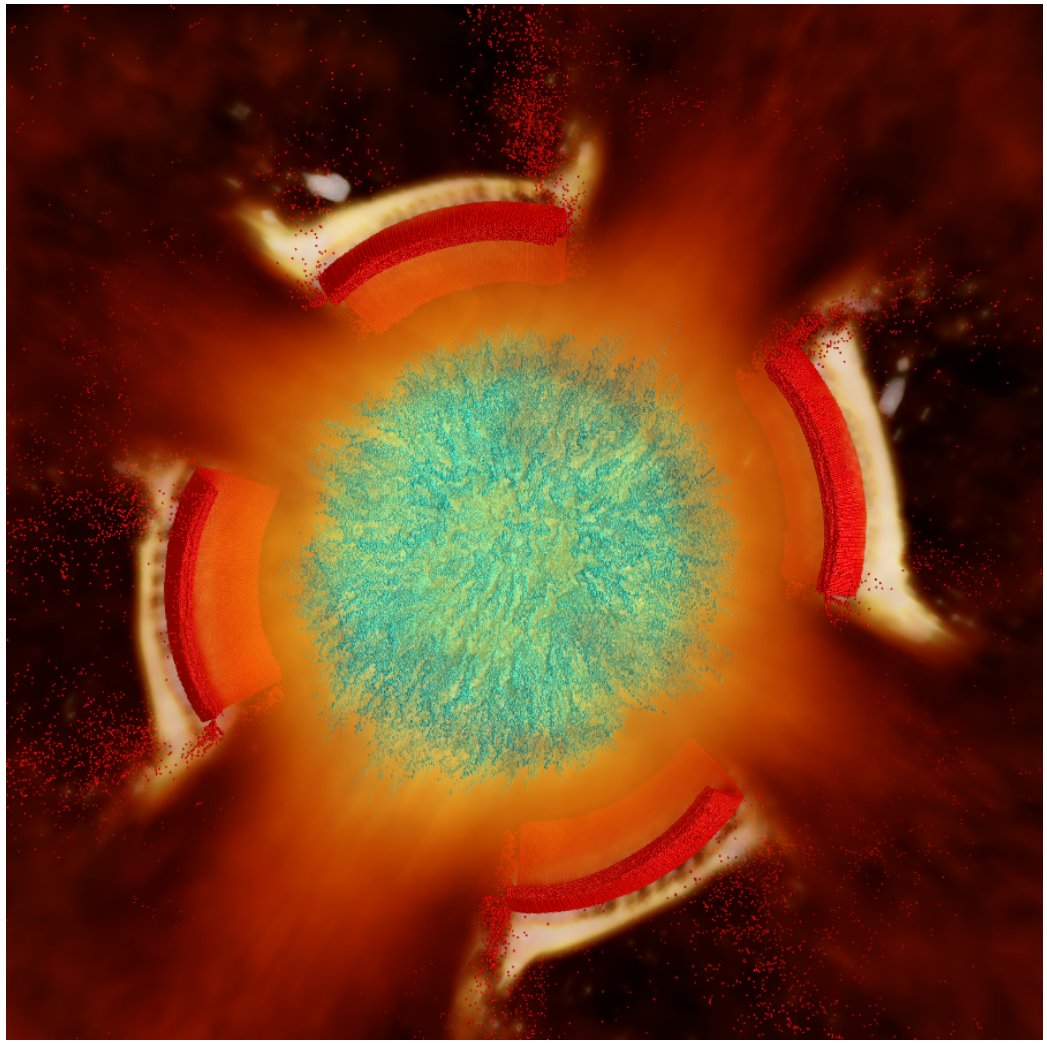
Why use a ready library?

1. Making own algorithms is hard
2. Save time
3. If actively developed and tested, makes your code less error-prone

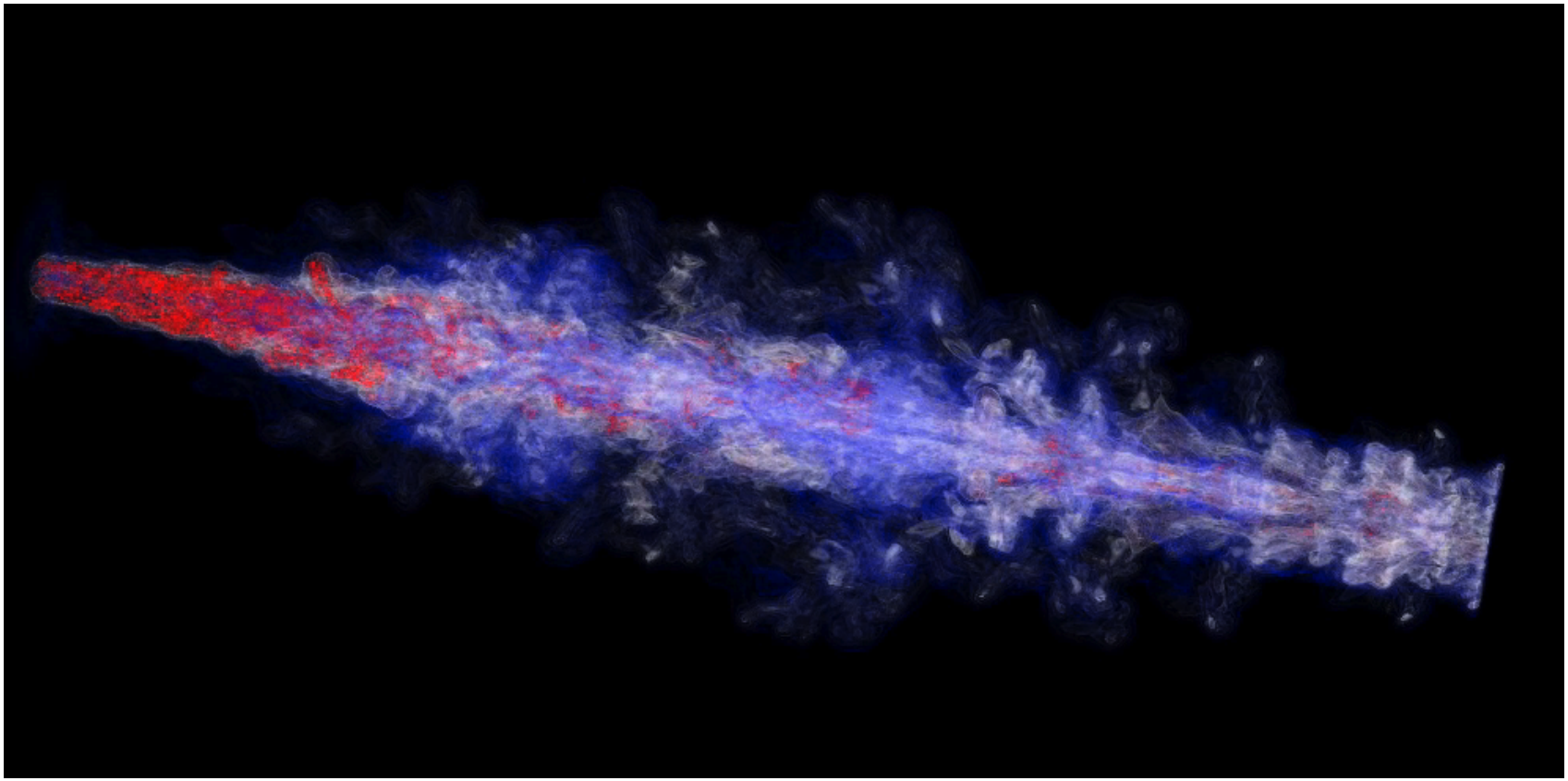
The cons and pros of Uintah

1. Pros
   1.1 Fast
   1.2 Boasts scalability up to over 200 000 cores
   1.3 Relatively simple
   1.4 Is actively developed at the University of Utah
   1.5 Aimed at specific problems
2. Cons
   2.1 Cartesian geometry
   2.2 Aimed at specific problems
   2.3 Uintah owns user-given data.

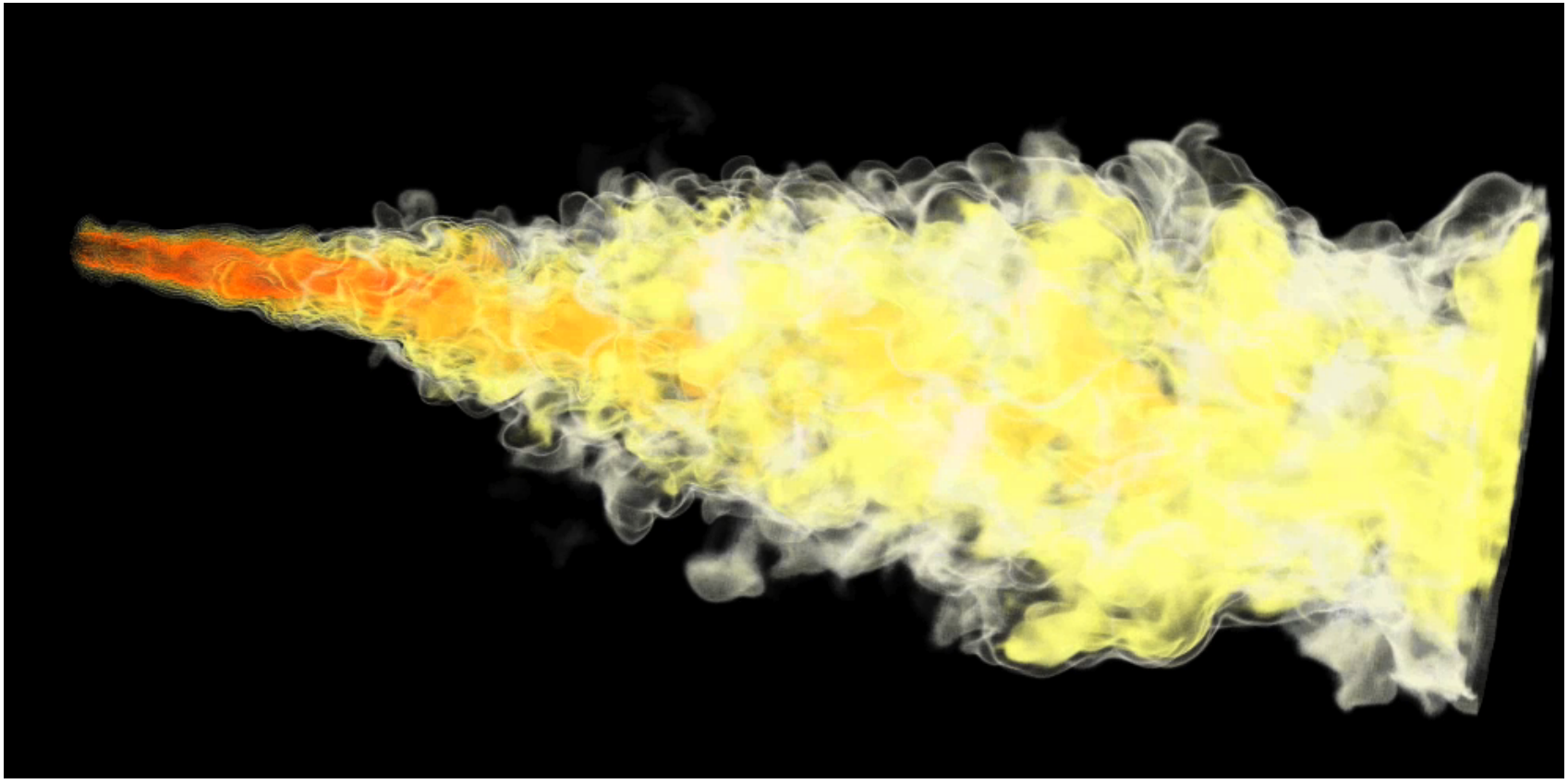
Uintah simulations



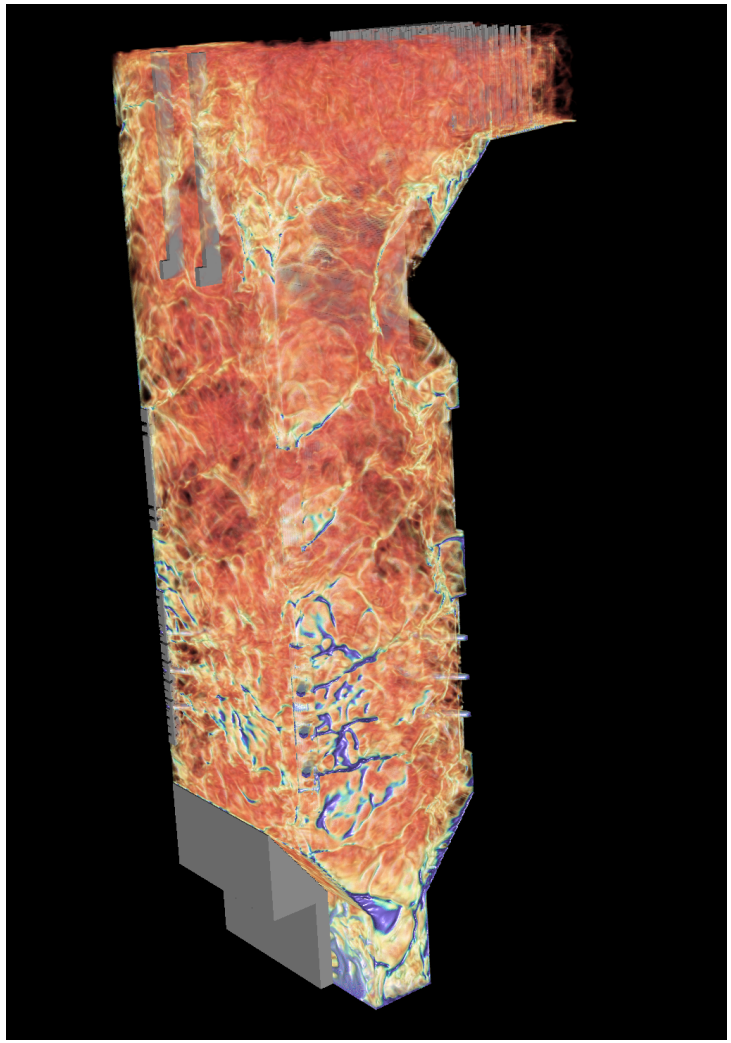
Uintah simulations



Uintah simulations



Uintah simulations



What Uintah can be used for

A few examples

1. Solving Poisson equation
2. Making PIC simulations
3. MHD simulations
4. Simulations of container explosions
5. ..

What Uintah *can not* be used for

1. *Vlasov equation* with mesh-based methods
   - 1.1 Would require 6D
   - 1.2 Would require support for sparse data types
2. PDEs[1] in *special geometry*
   - 2.1 Would require support for arbitrary geometries
3. PDEs with adaptive RBF[2] or other basis methods
   - 3.1 Would again require support for sparse data types

[1] PDE = Partial Differential Equation

[2] RBF = Radial Basis Function

What we used it for

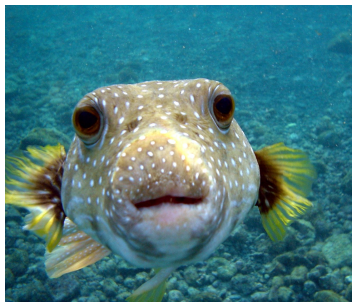


Figure : Poisson simulation

1. Creating a PIC simulation with a Poisson solver which is
   1.1 Fully parallel
   1.2 Scalable
   1.3 No AMR yet (and wont be unless we finish it by Sunday)

What we used it for

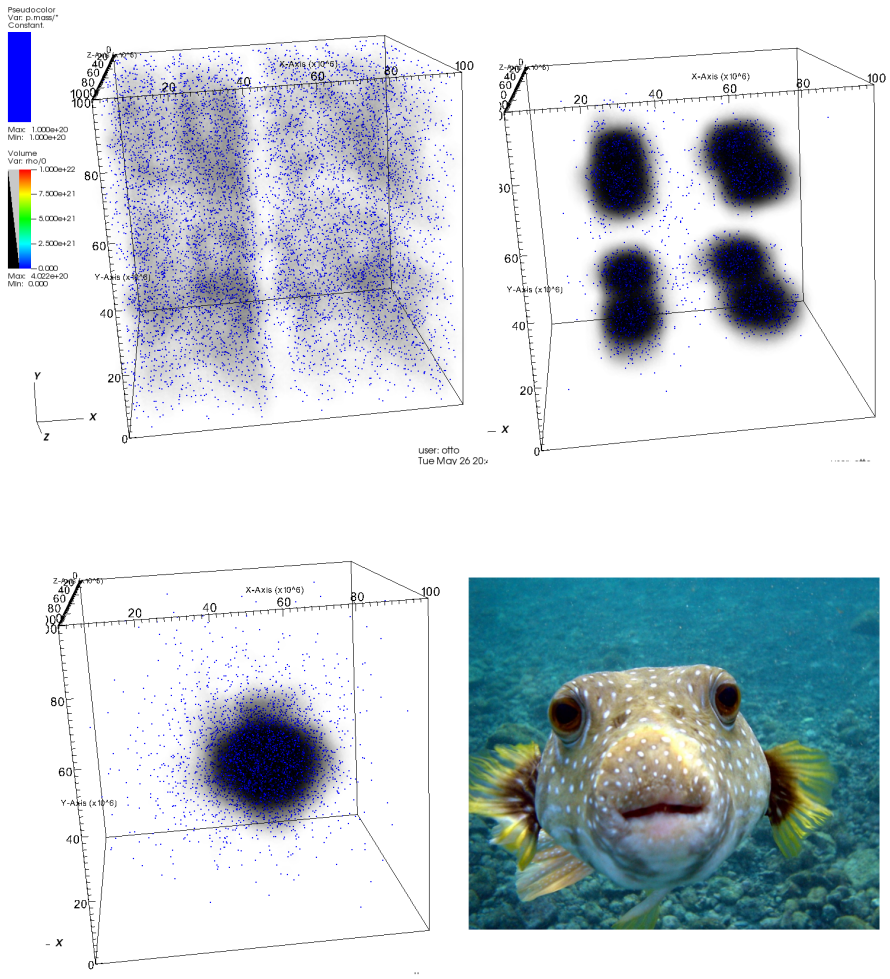


Figure : Plot of our PIC simulation

What we used it for

1. In short, we solved:
   1.1 $\nabla^2\phi(\mathbf{x}) = 4\pi G\rho(\mathbf{x})$
   1.2 $\mathbf{F} = -\nabla\phi$
   1.3 $m\frac{d^2\mathbf{x}}{dt^2} = \mathbf{F}$

Our program (for those interested)

Set up boundary conditions
Set initial ρ
Set initial guess for ϕ
loop
    Calculate ρ
    Calculate $\phi \leftarrow \phi_{next}$
    for Every particle p **do**
        Calculate new particle velocity and position for p
    end for
    $t \leftarrow t + dt$
end loop

Parallelism

A bit on parallelism

Parallelism

1. Uses a task-based approach
2. Supports both threads and MPI
3. One thread for MPI communication, rest for executing code
4. Note: requires latest MPICH and support for MPI_THREAD_MULTIPLE
5. Future: Support for GPU parallelism (now experimental)

Parallelism

Yes, the relative speedup is over 8 with 70 cores

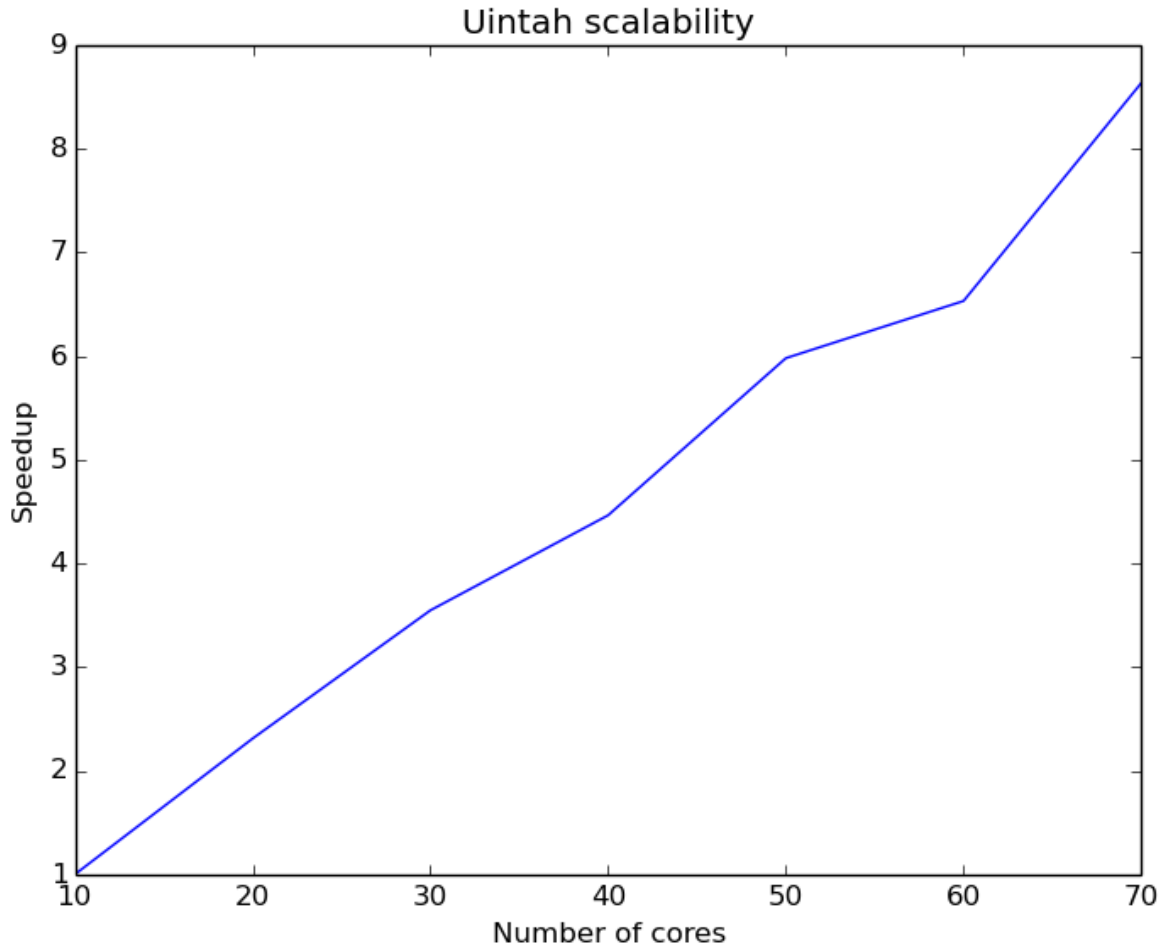


Figure : Our project's scalability run with 5 million particles, 300^3 cells

Uintah overview

Uintah overview

1. Few key concepts about Uintah
2. Coding Examples
3. Parallelism
4. Summary

A few important concepts on using Uintah

1. Uintah operates on tasks, there is no "*main*" program
2. A *new project* will be written as a *new class*
3. Variables and boundary conditions are in an *xml run file*

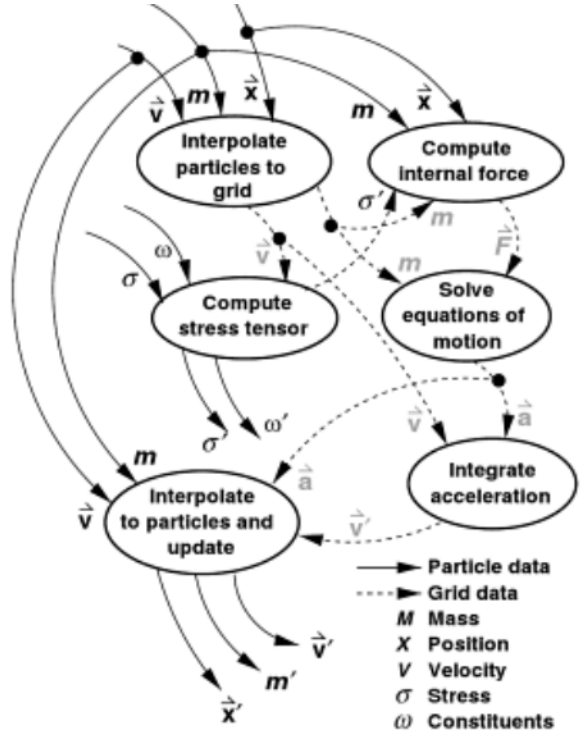


Figure : Example Uintah taskgraph

Uintah operates on tasks, there is no "*main*" program

Functions are called via *tasks* in Uintah:

Task $\leftarrow$new task(*solvePoissonEquation*)
Task $\leftarrow$ old ϕ
Task $\leftarrow$ new ρ
Scheduler $\leftarrow$ Task

Reminder: $\nabla^2\phi(\mathbf{x}) = 4\pi G\rho(\mathbf{x})$

Uintah operates on tasks, there is no "*main*" program

The *previous* function call, but now in the Uintah framework:

```
void
PICPoissonSimulation::scheduleTimeAdvance( const LevelP& level, SchedulerP& sched)
{
..
  // Poisson solver task
  // Create a new task
  Task* task = scinew Task("poissonSolverTask", this, &PICPoissonSimulation::poissonSolver, le
sched.get_rep());

  // The task needs to fetch the phi value from last timestep
  task->requires(Task::OldDW, phi_label, Ghost::AroundNodes, 1);

  // The task needs to fetch the rho value from current timestep
  task->requires(Task::NewDW, rho_label, Ghost::AroundNodes, 1);

  // The task computes a new phi value
  task->computes(phi_label);

  // Add the task to the task schedule:
  schedule->addTask(task, level->eachPatch(), sharedState_->allMaterials() );
..
}
```

A *new project* will be written as a *new class*

To make a project, create a new class that inherits a Uintah class:

```
class PICPoissonSimulation : public UintahParallelComponent, public SimulationInterface {
  PICPoissonSimulation(const ProcessorGroup* myworld);
  virtual ~PICPoissonSimulation();
  virtual void problemSetup(..);
  virtual void scheduleInitialize(..);
  virtual void scheduleComputeStableTimestep(..);
  virtual void scheduleTimeAdvance(..);
  ..
  Functions
  ..
}
```

Variables and boundary conditions are in an *xml run file*

Example XML file:

```
<SimulationComponent type="PICPoissonSimulation" />

<Time>
    <maxTime>1.0e14</maxTime>
    <initTime >0.0</initTime>
    <delt_min >1.0e13</delt_min>
    <delt_max >1.0e13</delt_max>
    <timestep_multiplier >1</timestep_multiplier>
</Time>

<LoadBalancer type="DLB">
  <timestepInterval >2</timestepInterval>
  <dynamicAlgorithm>patchFactorParticles </dynamicAlgorithm>
</LoadBalancer>

 <Grid>
    <Level>
        <Box label = "1">
            <lower >[0,0,0]</lower>
            <upper >[30000,30000,30000]</upper>
             <resolution >[200,200,200]</resolution>
            <patches >[5,5,5]</patches>
        </Box>
    </Level>
    ..
 </Grid>
```

Words of warning

While Uintah is fairly easy to use, scalable and overall likeable, it has some drawbacks

1. It is meant for very specific use (namely solving PDEs on a grid with support for particles)
2. Uintah is not a template library, and Uintah owns the data it operates on
3. No support for more than 3 dimensions
4. Some support for spherical and cylinder coordinates, but not for more complex structures

Summary/Discussion/Questions/Suggestions

1. Uintah is a powerful tool for simulating meshes
2. It is simple to use and has a small learning curve
3. It is parallel and supports adaptive mesh refinement and particle interactions
4. It has fully automated load balancing
5. Perhaps the most scalable grid library currently, with scalability up to over 200k cores

Thank you for your attention!