

An Introduction to Wasatch

James C. Sutherland

SEPTEMBER 7, 2010

Levels of Parallelism

Domain decomposition (data parallel)

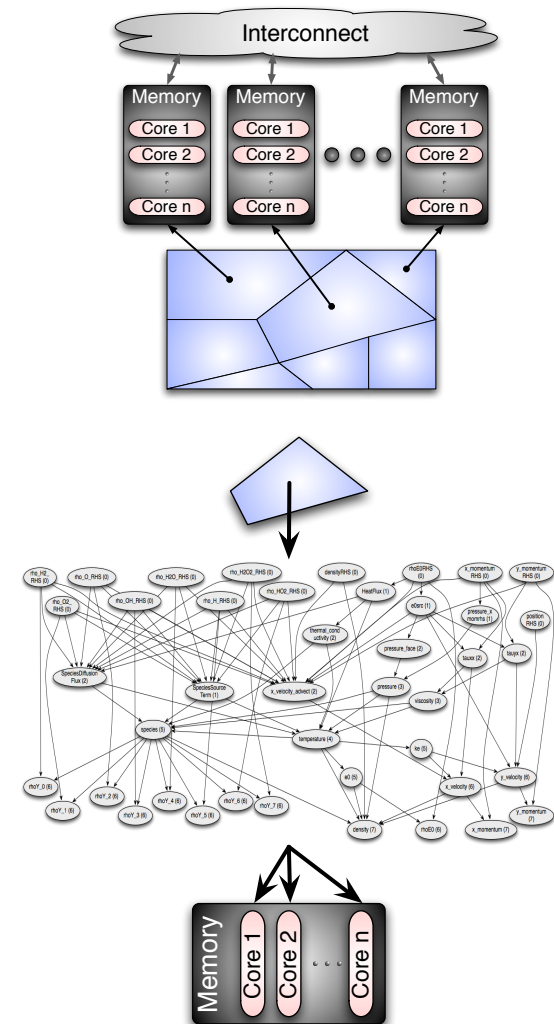
- subdivide the physical domain into patches/worksets & distribute across nodes/cores

Algorithm decomposition (task parallel)

- Identify tasks on a patch/workset that can be evaluated concurrently

Fine-grained (vector) parallelism

- Perform operations independently on various grid points
- Field operations (mesh loops)
- Discrete calculus operations (sparse matvec)
- Some can be hidden from the user
 - expression templates to collect series of mesh operations and dispatch them optimally in parallel...



$$\nabla^2 \phi + s$$

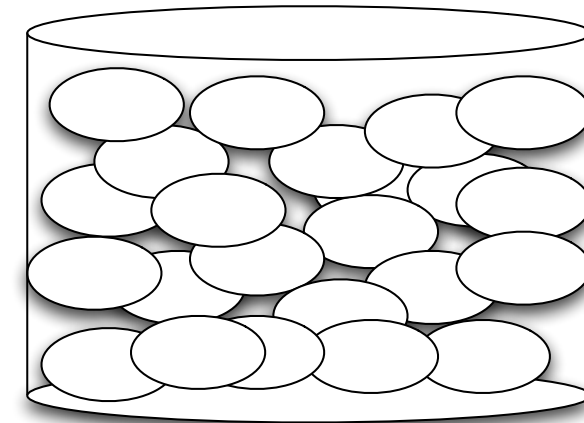
One expression
(calculated on a patch/workset)



Expression Library

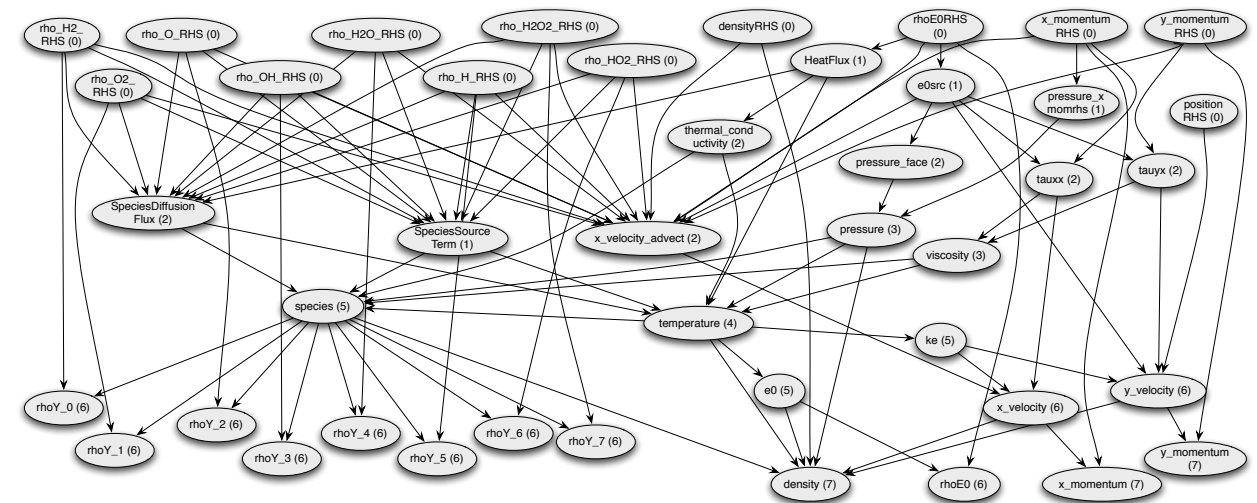
(Algorithm Decomposition)

- Uses graph theory to automatically construct algorithms from dependencies among expressions
- Thread-based parallelism via algorithm decomposition (exposing concurrency in the algorithm)
- Key classes:
 - Expression: a node in a graph. Calculates one or more fields of the same type.
 - ExpressionTree: a graph containing one or more nodes. Automatically constructed given a set of root nodes.
- Expressions are where the bulk of user code in Wasatch will be written.



Create individual expressions and register them.

Request calculation of any # of fields. The graph to accomplish this is automatically created!



<https://software.crsim.utah.edu/trac/wiki/ExprLib/GettingStarted>

THE INSTITUTE FOR CLEAN AND SECURE ENERGY

SpatialOps Library

(type-safe vectorized operations)

- Supports field & stencil operations directly - no more loops!
- Strongly typed fields ensure valid operations at compile time.
- Allows a variety of implementations to be tried without modifying application code.

an excerpt from:

Wasatch/Expressions/DiffusiveFlux.h

```
typedef typename GradT::DestFieldType FluxT;  
typedef typename GradT::SrcFieldType  ScalarT;  
  
const GradT* gradOp_  
const ScalarT* phi_  
const FluxT* coef_;
```

The “work” part
of an expression
to calculate a
diffusive flux.

```
template< typename GradT >  
void  
DiffusiveFlux<GradT>::evaluate()  
{  
    FluxT& result = this->value();  
  
    gradOp_->apply_to_field( *phi_, result ); // J = grad(phi)  
    result *= *coef_;                        // J = gamma * grad(phi)  
    result *= -1.0;                          // J = -gamma * grad(phi)  
}
```



<https://software.crsim.utah.edu/trac/wiki/SpatialOps>

THE INSTITUTE FOR CLEAN AND SECURE ENERGY

Why Wasatch?

Concept: interface the Expression and SpatialOps libraries with Uintah.

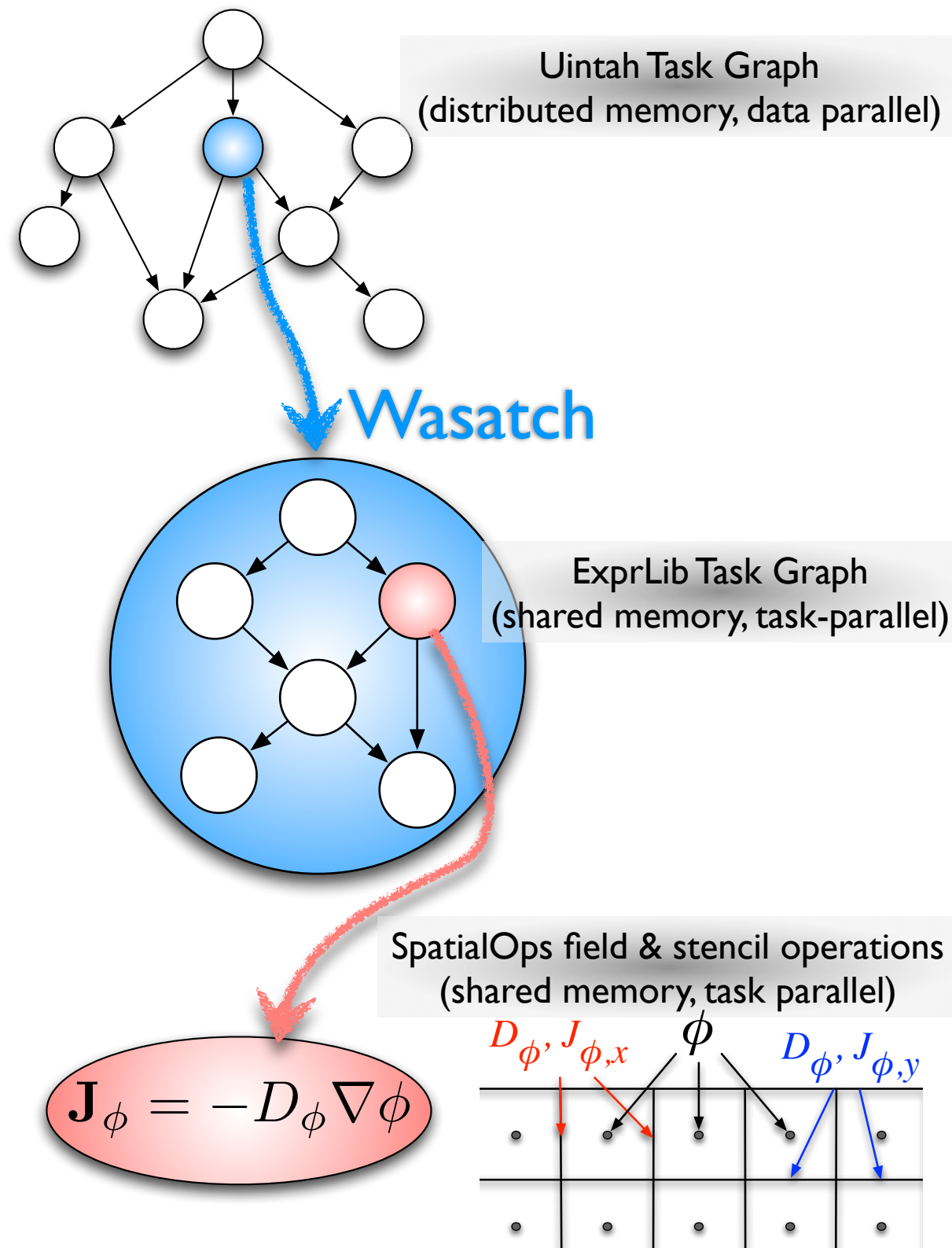
- Expression Library provides a simple way of handling complexity in code and also gives thread parallelism.
- SpatialOps library provides compact ways of dealing with discretization.
- Uintah gives MPI parallelism

Wasatch:

- Parses input and turns this into transport equations, expressions, etc.
- Generates expression graph(s) and exposes them to Uintah as task(s)
- Allows code to be developed in a Uintah-agnostic way.

Result: rapid development of robust code.

- heavy usage of C++ templates.



A Few Key Uintah Concepts

- 📌 Patch: a sub-region in space where we are solving PDEs
- 📌 Task: some work to do on a patch to calculate the value of one or more fields.
- 📌 A few required tasks
 - Initialization
 - Setting the value of the time step
 - Advance solution over time step increment.



Key Wasatch Abstractions

“Core” Wasatch Developers will need to understand these.



TaskInterface: wraps ExpressionTree objects as Uintah tasks

- This happens largely behind the scenes.
- All field requirements are gathered from the ExpressionTree and advertised to Uintah, and the Uintah task is automatically created.



TransportEquation: a class to support formation of PDE solvers

- Register expressions required to set initial conditions for this equation.
- Register expressions required to calculate the RHS of this equation.
- Set BCs for this equation.



TimeStepper: a class to support advancing PDEs in time (Runge-Kutta)

- creates an ExpressionTree (graph) and wraps it to expose it to Uintah as a (heavy-weight) Task.
 - Plug in an expression to calculate the RHS of an equation and it does the rest!
- Easily add new transport equations to the TimeStepper, resulting in automatic generation of new algorithms with appropriate coupling/ordering.
- When TransportEquations are parsed from the input file, an “adaptor” is created that plugs them into a TimeStepper.



Wasatch Skeleton



Wasatch::problemSetup

- Registers the material(s) for use in solving this problem.
- Creates any expressions triggered directly from the input file. These are typically associated with calculating initial conditions.
- Sets up property evaluation expressions (may be required by any transport equation).
- Creates a time integrator for solving the set of PDEs.
- Parses and constructs transport equations. This, in turn, registers relevant expressions required to solve each transport equation and also plugs them into the time integrator.



Wasatch::scheduleInitialize

- Set up spatial operators associated with each patch.
- Set up the ExpressionTree (and then the Uintah::Task(s)) that will set the initial conditions.



Wasatch::scheduleComputeStableTimestep



Wasatch::scheduleTimeAdvance

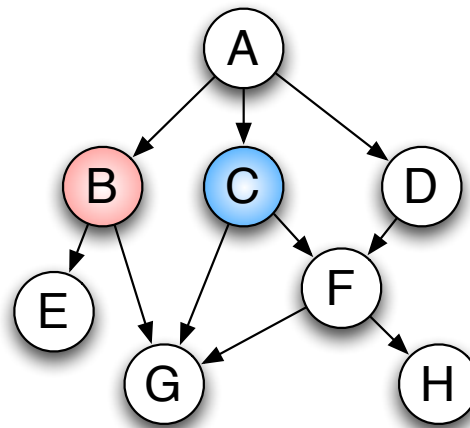
- Plug each transport equation that has been set up into the time stepper and generate the full ExpressionTree that will calculate the RHS for all of the equations.
- Create the Uintah::Task(s) associated with the ExpressionTree to advance the solution for the PDE system forward one time step.



Graph/Algorithm Cleaving

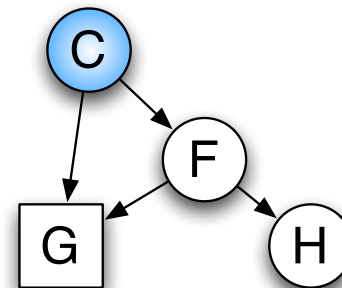
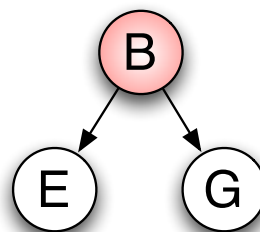
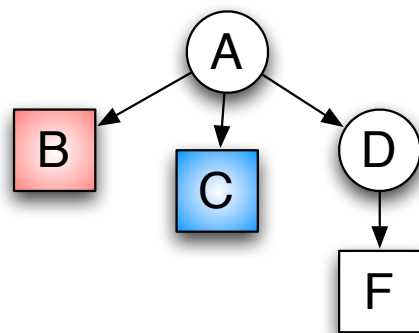
Programmer “tags” expressions that require ghost updates.
Graphs are automatically cleaved.

Original “full” graph
(translated into a Uintah Task):



“B” and “C” require ghost updates
prior to execution of “A”

Modified to three Uintah Tasks:



Order of execution:

1. “B” tree
2. “C” tree
3. “A” tree

Three separate graphs (Uintah Tasks).
Uintah manages ghost updates.

