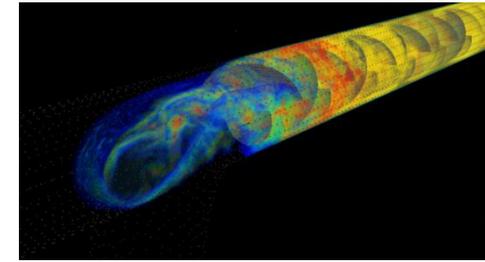


Portability and Performance with Task-Based Frameworks - Experiences with Uintah

Martin Berzins



Open source at
www.uintah.utah.edu

- Introduction, machines
- Uintah abstractions for portability
- Uintah key kernels
- Uintah performance examples and challenges
- Portable Performance



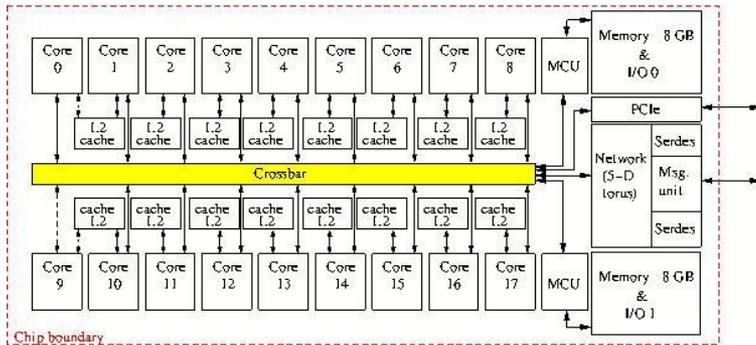
Thanks to DOE ASCI (97-10), NSF , DOE NETL+NNSA, ARL,
NSF , INCITE, XSEDE, ALCC, ORNL, ALCF



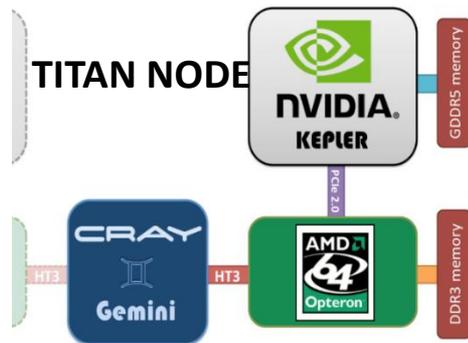
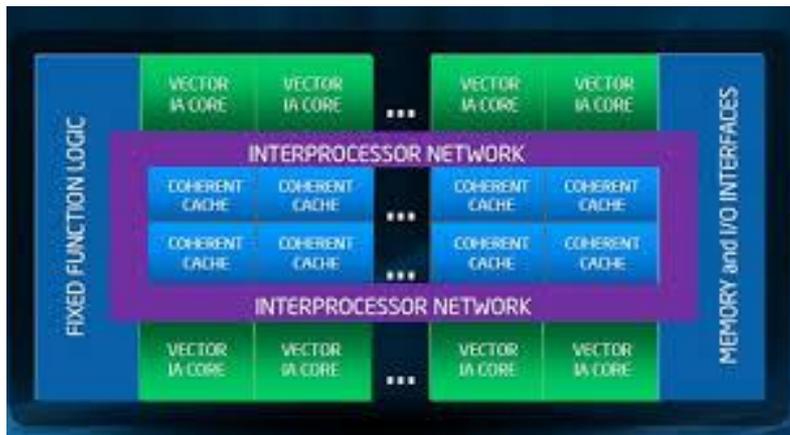
INTRODUCTION,

- Peak computer power grows by a factor of a thousand every decade and so has a profound impact on our ability to undertake science
- Often in the past we have had to change our codes...
- A significant change is coming as we try to go forward for the next decade.
- New machines are all very different architecturally due to energy and manufacturing process requirements
- Can we achieve portability and performance?
- What are the right abstractions for ensuring portability and performance?

IBM BGQ 16 core

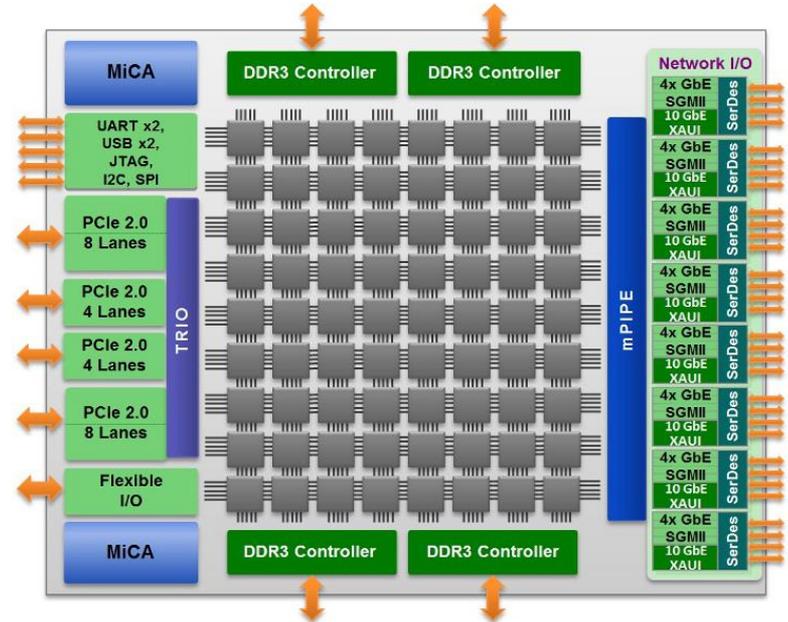


Intel Xeon Phi KNC

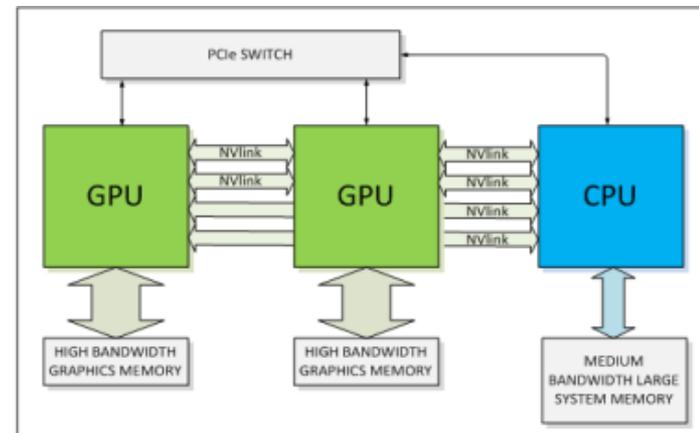


Present nodes used here and Future Nodes

Intel Xeon Phi KNL



FUTURE GPU NODEs



Present PetaFlop (10**15 ops)Architectures used here and Future Architectures

System attributes	OLCF Now	ALCF Now	OLCF Upgrade	ALCF Upgrade
Name/Planned Installation	TITAN	MIRA	Summit 2017-2018	Aurora 2018-2019
System peak (Flops)	27 PF	10PF	8-10x Titan	>180PF
Peak Power (MW)	8.2MW	4.8MW	10MW	~13MW
System memory per node	38 GB	16 GB	> 512 GB	TBA
Node performance (TF)	1.452	0.204	>40	>15 times Mira
Node processors	AMD Opteron Nvidia Kepler	64-bit PowerPC A2	Multiple IBM Power9 CPUs & multiple Nvidia Voltas GPUS	Intel Knights Hill
System size (nodes)	18,688 nodes, 268K cores 18.6K GPUs	49,152 nodes 768K cores	~3,500 nodes	~50,000 nodes 3M cores?
Interconnect	Gemini	5D Torus	Mellanox	Intel

Plus one more generation of machines and then exascale 10**18

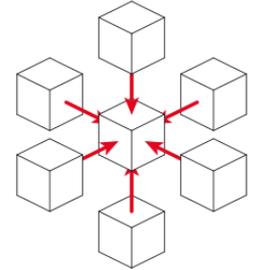
Programming Challenges for New Architectures I

Summit 3.5K nodes **TITAN** 18.6K nodes

At 8-10x Titan each Summit nodes has to have 40 → 60 x flops of a Titan node

Titan injection bw is 6.4GB/s vs 23 GB/s for Summit

Factor of 4 roughly.



Consider Stencil Calculation

flops proportional to volume

communications proportional to surface area

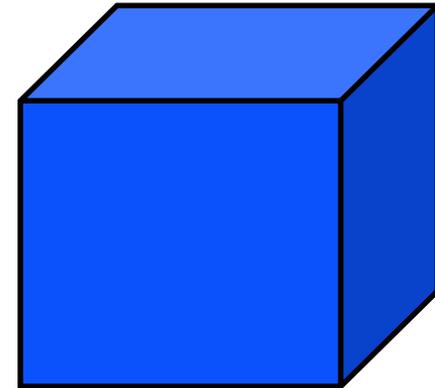
$6N^2$ where $N \times N \times N$ is volume. Compare $6N^2$ values

Summit communications per node are then

$(40 \rightarrow 60)^{2/3}$ of Titan or **13x Titan**

Hence Summit's network might be easier to overload than is desirable

Hiding communications costs may will be key on Summit



Programming Challenges for New Architectures II

Mira 48K nodes

16 cores

10 PF

Theta 2.5K nodes

72? cores

8.5PF

Aurora 50K nodes

60+ cores?

180-450- PF

Each Theta core has to be 3-4x faster than a Mira core

Similarly with Aurora with about 4x cores and 18x performance each core has to be 3-4 X faster than a Mira core.

This is achieved today with standard “heavyweight” Xeon cores e.g. NSFs Stampede.

Achieving this with vectorization could be challenging particularly if KNH performance relies on long vectors.

The Intel Omnipath technology seems to provide a network that builds upon Mira.

Five abstractions for Portability and Performance

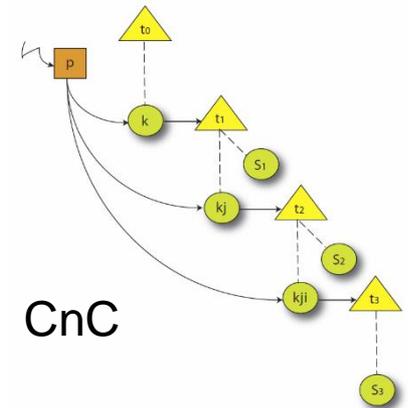
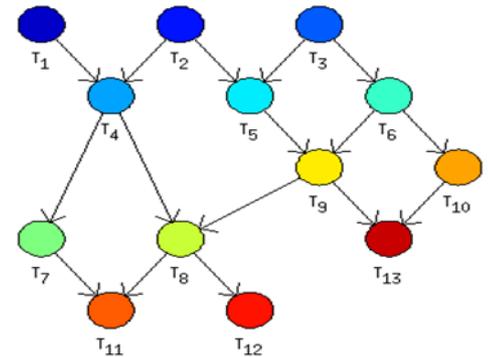
1. A DAG task-based formulation of problems
2. A programming model to write these tasks as code
3. A runtime system to execute these tasks
4. A low-level portability layer to allow tasks to run across different machines
5. A high-level domain specific language to ease problem specification

Five abstractions for Portability and Performance

1. A DAG task-based formulation of problems
2. A programming model to write these tasks as code
[Uintah](#)
3. A runtime system to execute these tasks
[Uintah Runtime System](#)
4. A low-level portability layer to allow tasks to run across different machines
[Kokkos](#)
5. A high-level domain specific language to ease problem solving
[Nebo Wasatch](#)

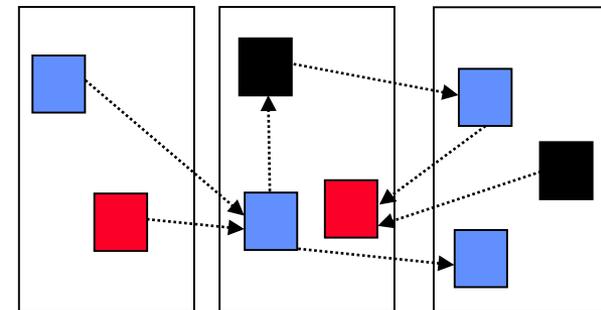
Why DAG Task-Based Approaches?

- Allows an abstract portable task specification independent of execution
- Allows ADAPTIVE execution through architecture-aware runtime system with over-decomposition
 - Adapts to delays in communications or execution
 - Allows replication of execution and fault-tolerance
- Task migration allows varying workloads to be addressed
- Performance can be achieved through a combination of runtime system and abstract machine specific layers



CnC

Charm++



Uintah Open Source Software Portability and Scalability

Uintah Software Team

Phase 1 Uintah software developed 1998-2008 by a team led by **Steve Parker**
Novel design and reasonable scalability (~1000 cores 2005) fixed execution

Phase 2 (2008 ...)NSF, ARL and DOE funded expansion of both scalability and software with adaptive task execution

NSF PetaApps

Qingyu Meng* John Schmidt,, Todd Harman Justin Luitjens*, Jacqueline Beckvermit



* Now at Google



* Now at NVIDIA



PSAAP Extreme Scaling team

Alan Humphrey John Holmen



SANDIA

Dan Sunderland



NSFXPS

Brad Peterson



Nan

Xiao



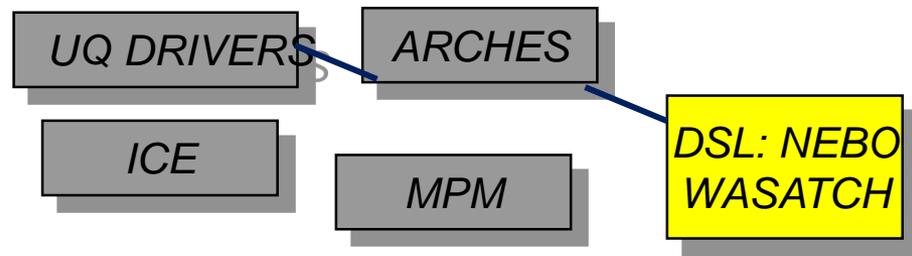
Harish

Dasari



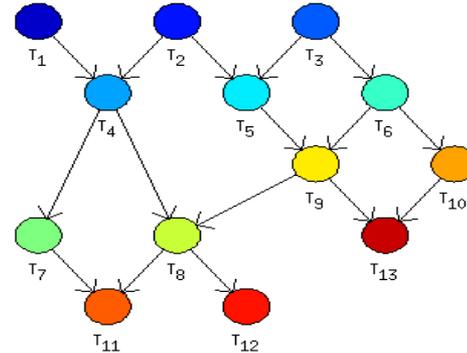
Applications code Programming model

Some components have not changed as we have gone from 600 to 600K cores



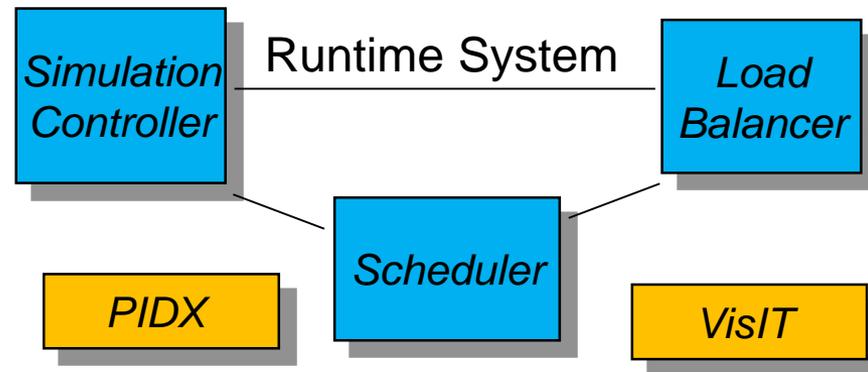
Abstract C++ Task Graph Form

Compilation into C++ Cuda etc



Adaptive Execution of tasks

asynchronous out-of-order execution, work stealing, Overlap communication & computation.



Intermediate layer for node Performance on specific cores/processors

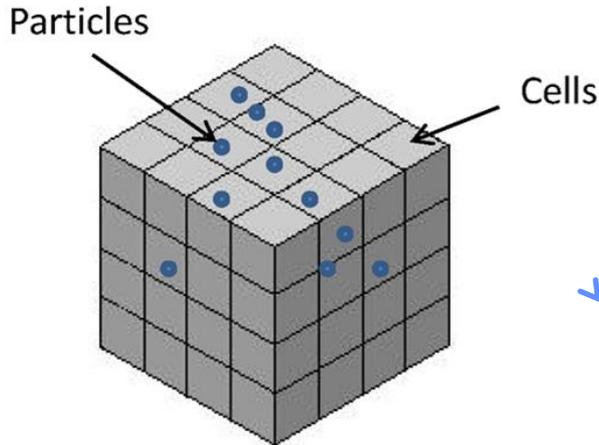
Kokkos Intermediate Layer

Uintah Architecture



Uintah Patch, Variables and AMR Outline

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

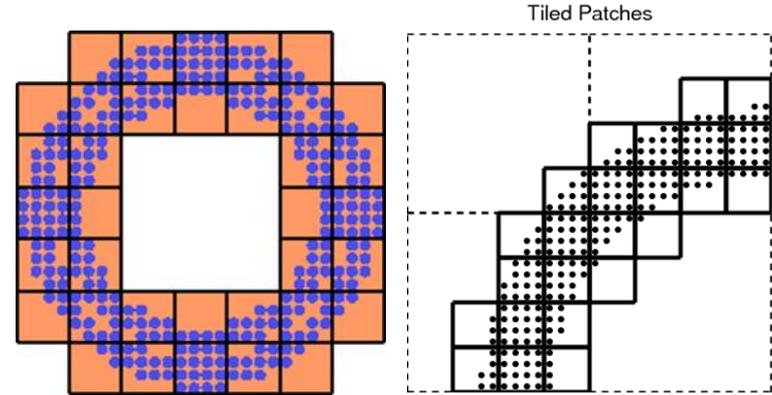


Uintah Patch

ICE Structured Grid Variable (for Flows) are Cell Centered Nodes, Face Centered Nodes.

Unstructured Points (for Solids) are **MPM** Particles PIC for solids

ARCHES is a combustion code using several different radiation models and linear solvers



- Structured Grid + Unstructured Points
- Patch-based Domain Decomposition
- Regular Local Adaptive Mesh Refinement
- Dynamic Load Balancing
 - Profiling + Forecasting Model
 - Parallel Space Filling Curves
- Works on MPI and/or thread level

Uintah **Programing Model** and **Runtime System**

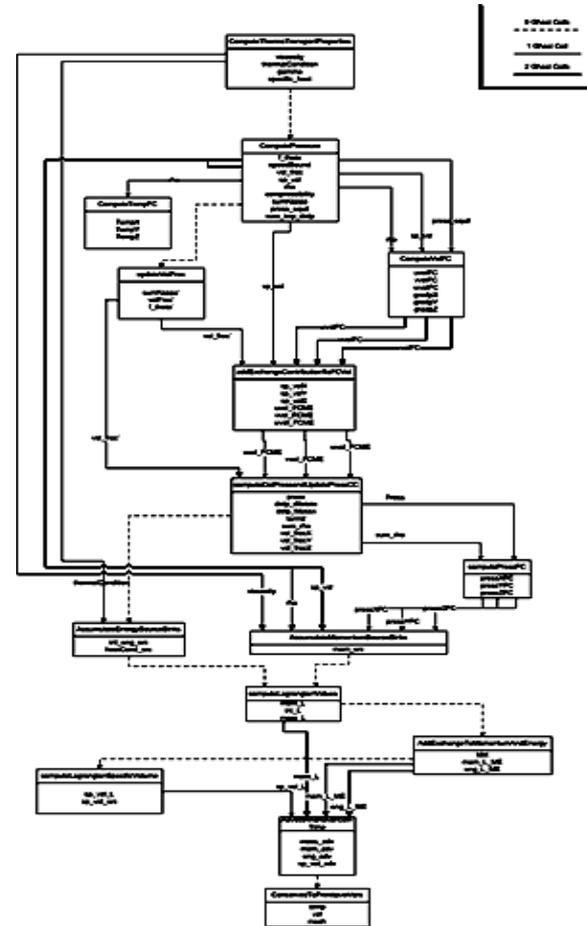
Each task defines its computation with required inputs and outputs

Tasks do not explicitly define communications but only what inputs they need from a data warehouse and which tasks need to execute before each other

Uintah uses this information to create a task graph of computation (nodes) + communication (along edges)

Communication is overlapped with computation

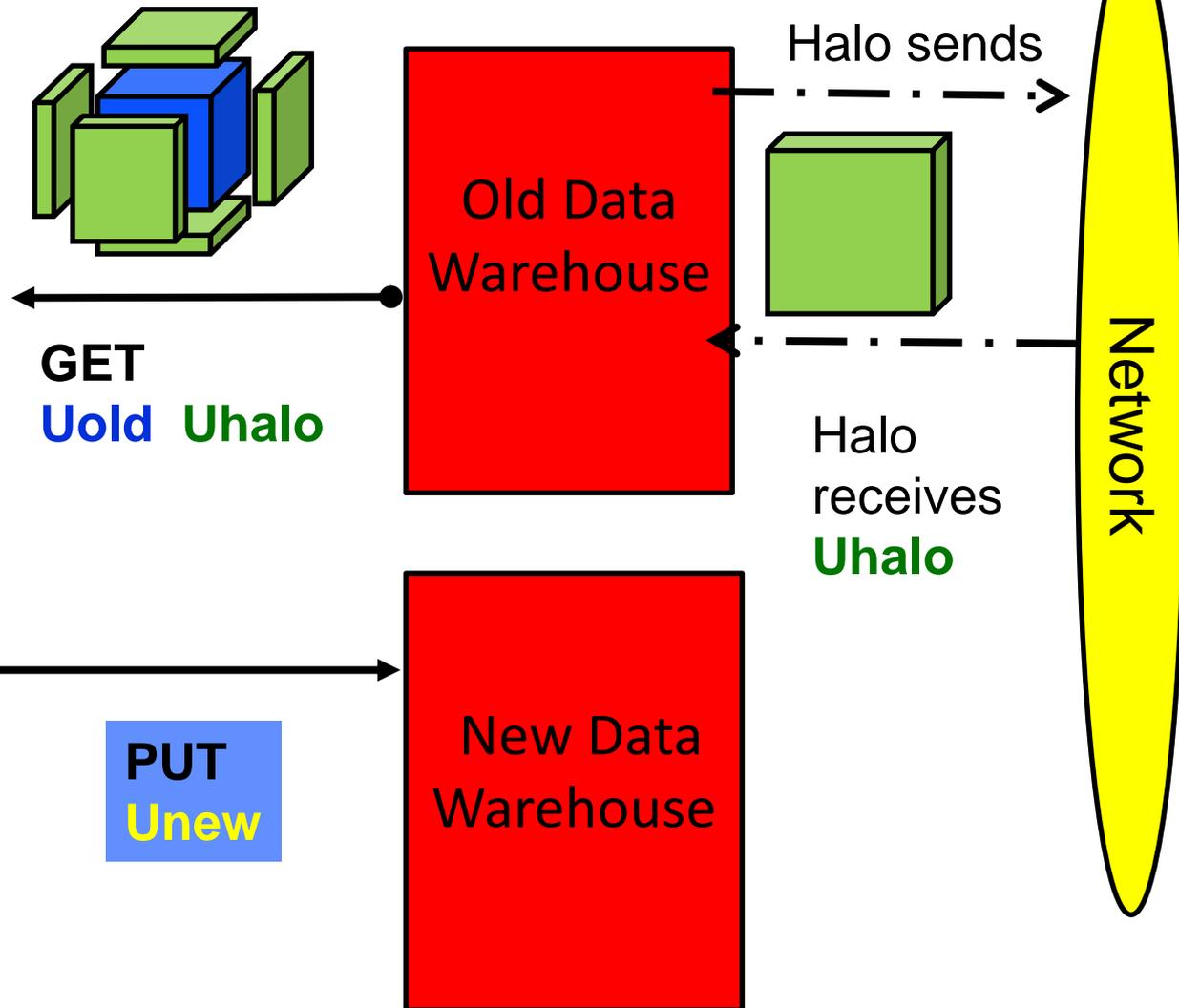
Taskgraph is executed adaptively and sometimes out of order, inputs to tasks are saved



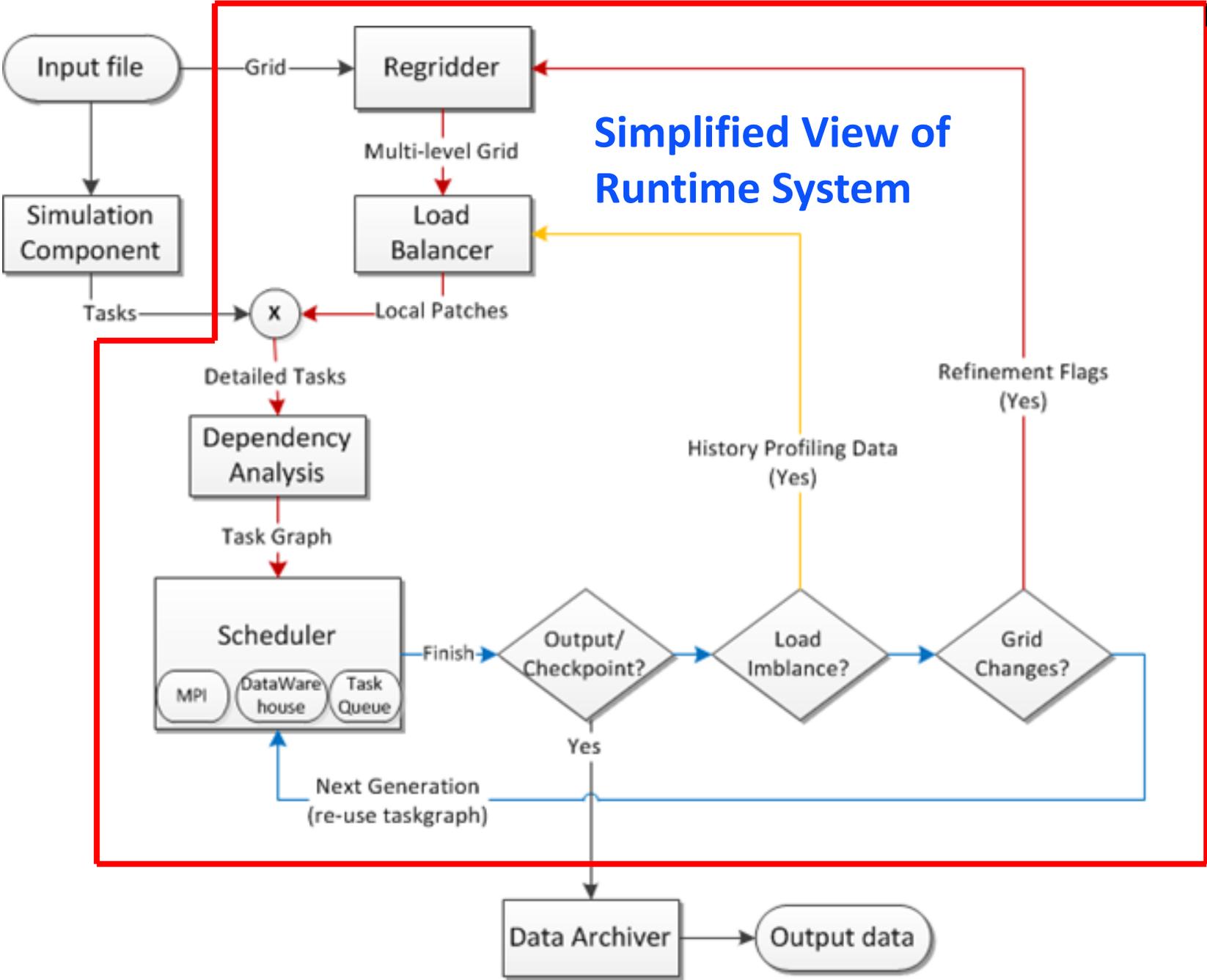
Programming Model for Stencil Computation on a Timestep

Example Stencil Task

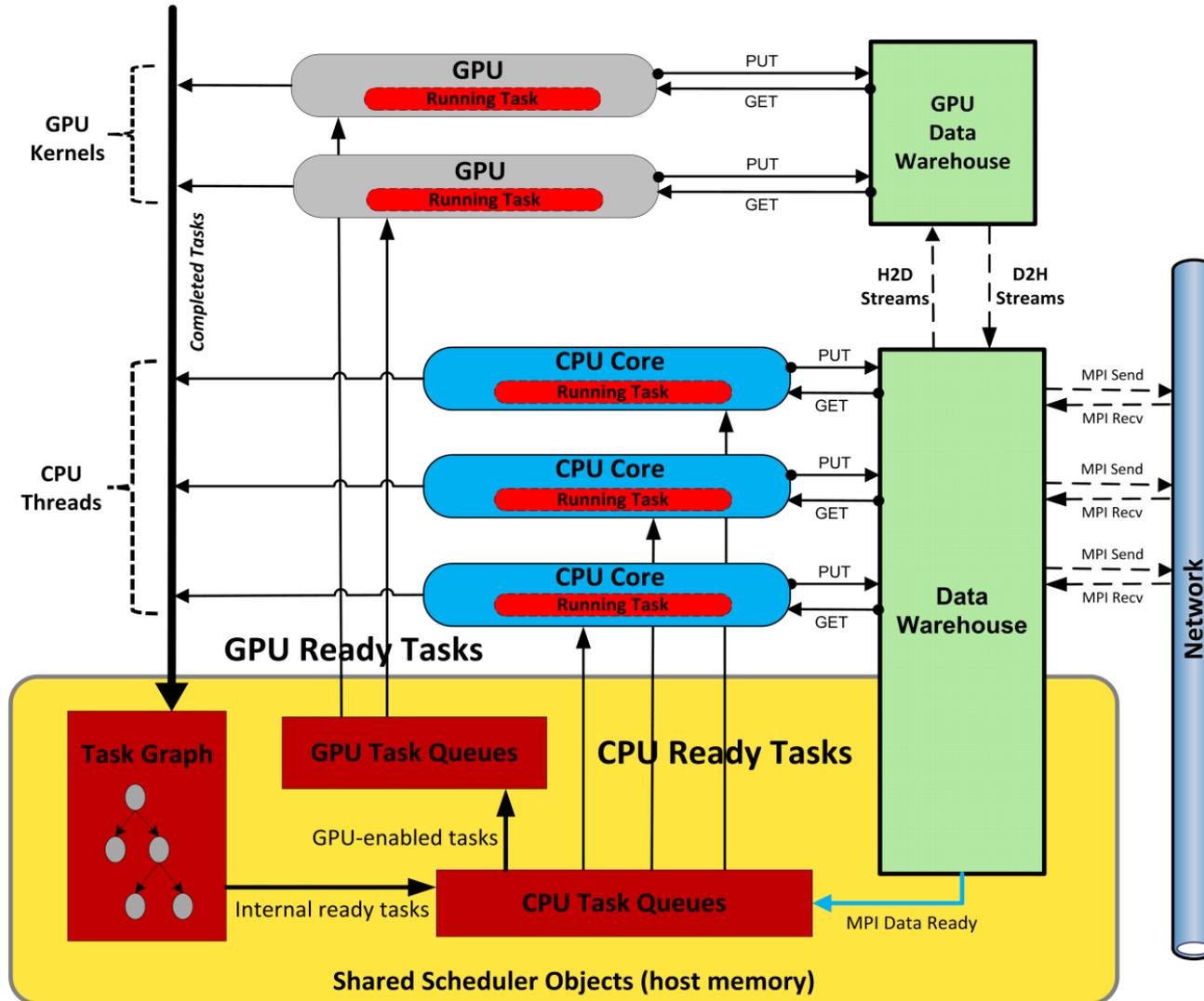
$$U_{\text{new}} = U_{\text{old}} + dt * F(U_{\text{old}}, U_{\text{halo}})$$



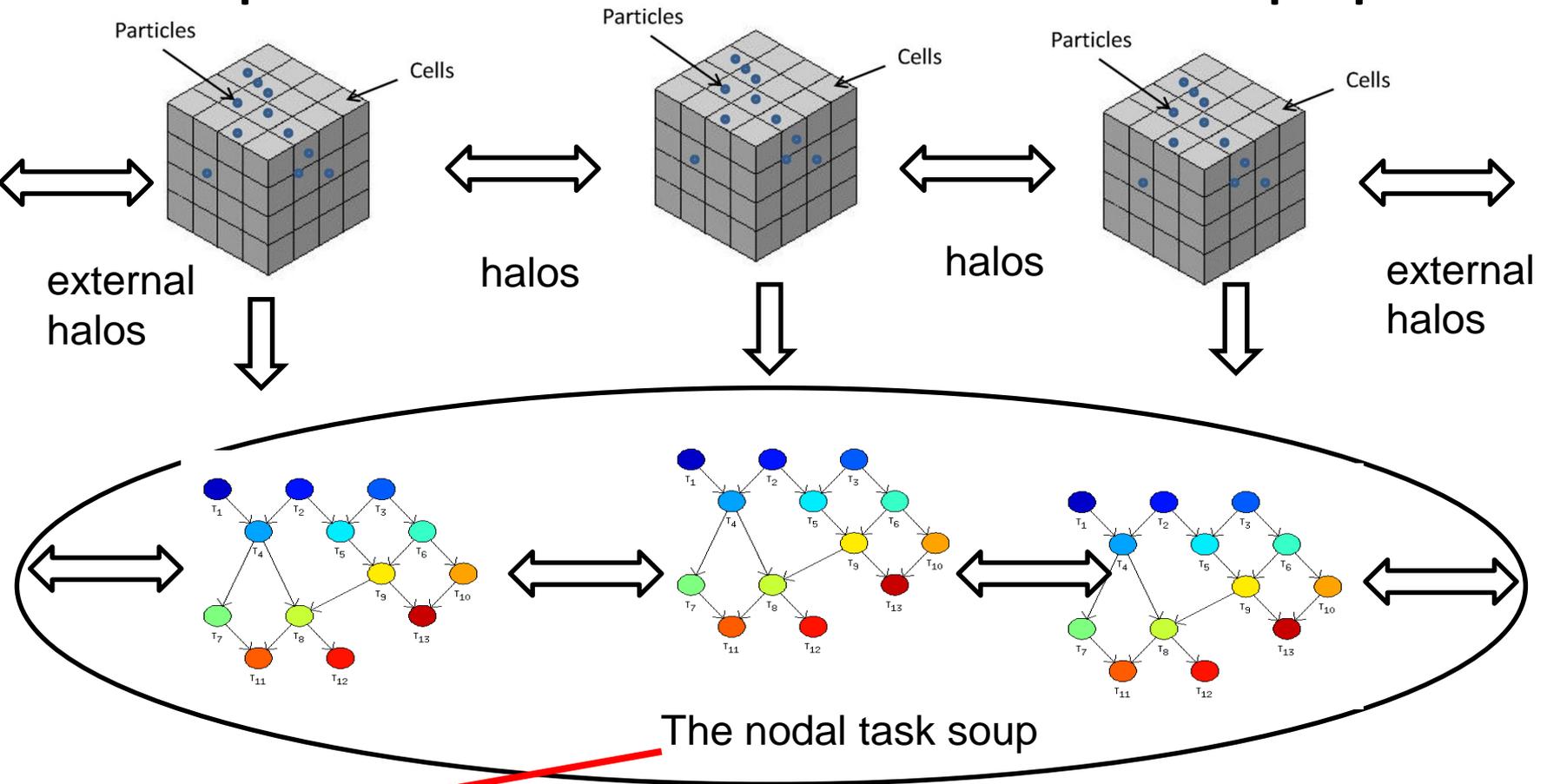
Simplified View of Runtime System



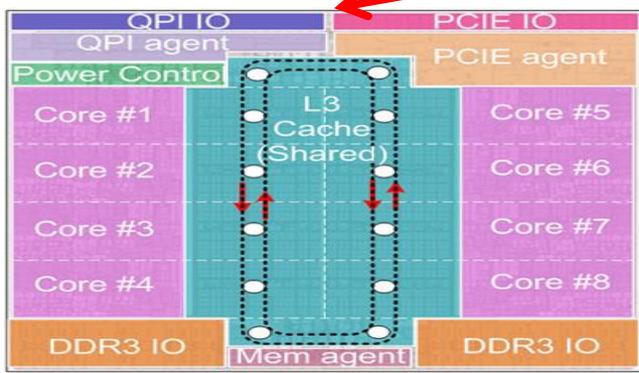
Uintah Heterogeneous Runtime System (Multiple GPUs and Intel Xeon Phis)



Task Graph Structure on a Multicore Node with multiple patches



The nodal task soup

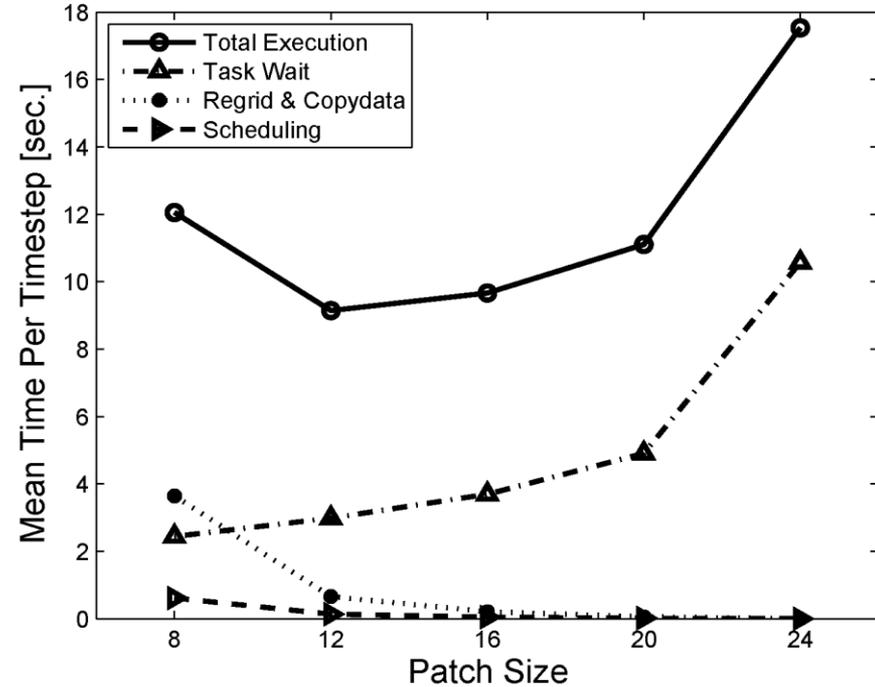


This is not a single graph. Multiscale and Multi-Physics merely add flavor to the “soup”. There are many adaptive strategies and tricks that are used in the execution of this graph soup.

Summary of Uintah Scalability Improvements

- (i) Move to a one MPI process per multicore node reduces memory to less than 10% of previous for 100K+ cores
- (ii) **Work on Runtime System involved substantial rewrites**
- (iii) Use optimal size patches to balance overhead and granularity
- (iv) Use only one data warehouse but allow all cores fast access to it, through the use of atomic operations.
- (v) **Prioritize tasks with the most external communications**
- (vi) **Use out-of-order execution**
- (vii) **Static order task execution of limited value**

ICE with different patch sizes (Kraken, with 24K cores)



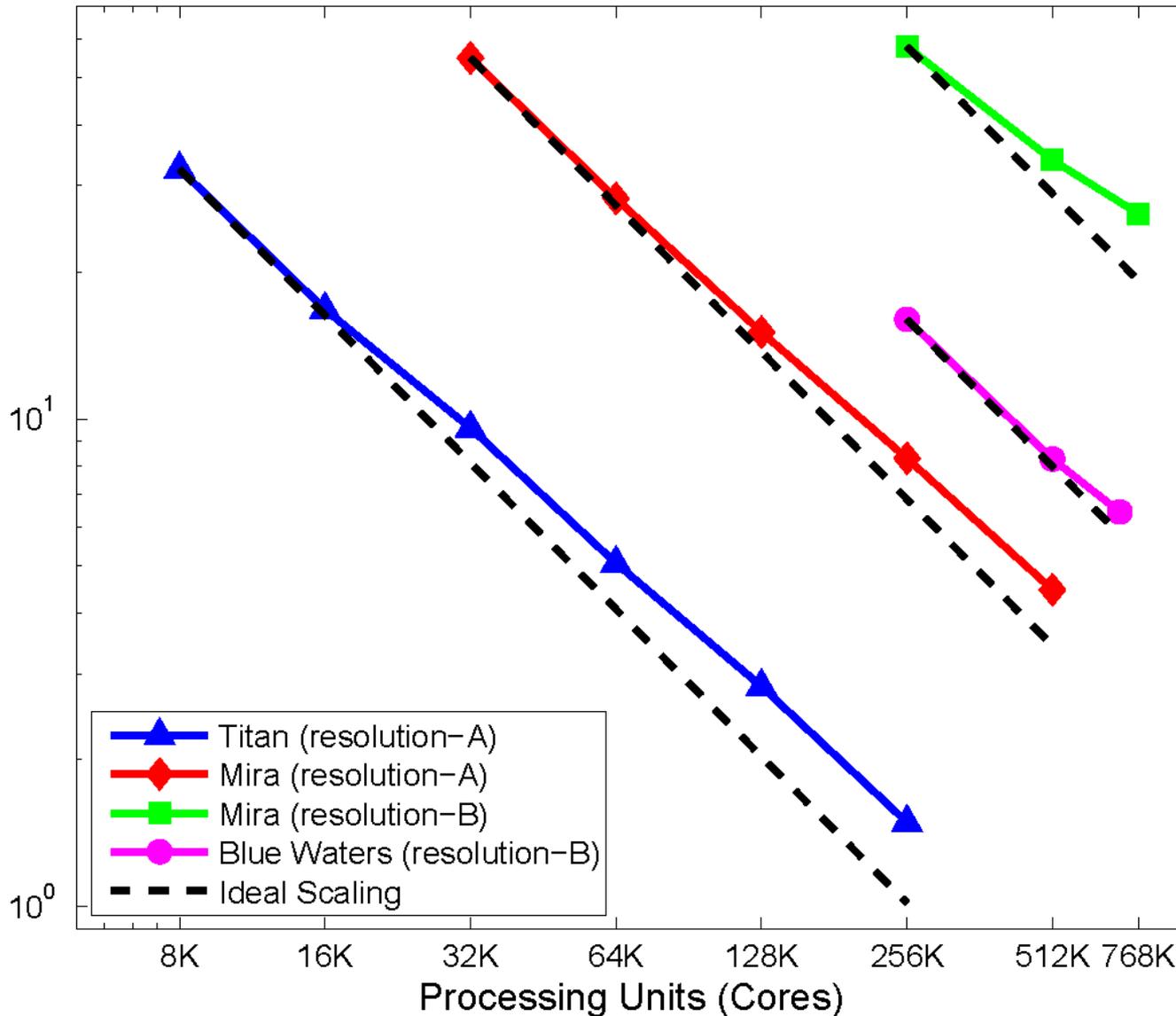
Algorithm	Random	FCFS	PatchOrder	Most Msg.
Queue Length	3.11	3.16	4.05	4.29
Wait Time	18.9	18.0	7.0	2.6
Overall Time	315.35	308.73	187.19	139.39

MPM AMR ICE Strong Scaling

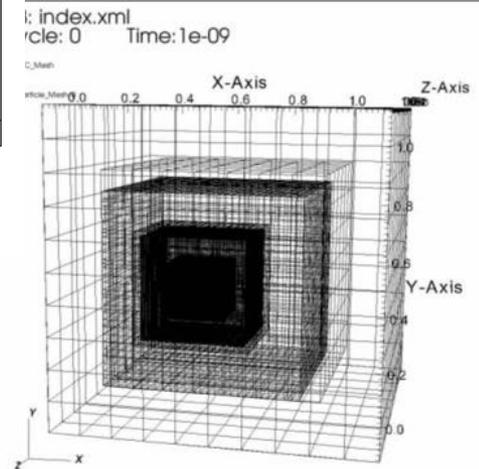
Mira DOE BG/Q
768K cores
Blue Waters Cray
XE6/XK7 700K+
cores

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

Mean Time Per Timestep(second)



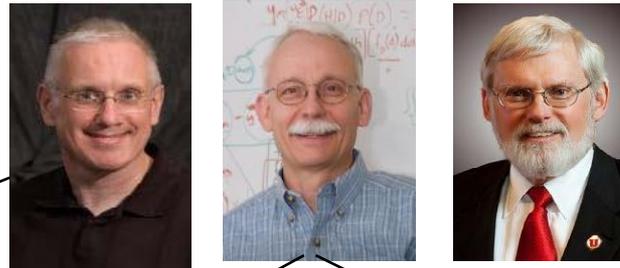
Complex fluid-structure interaction problem
with adaptive mesh refinement, see SC13/14 paper
NSF funding.



Utah PSAAP2 Center

Motivating Problem

The Exec



Computer Science

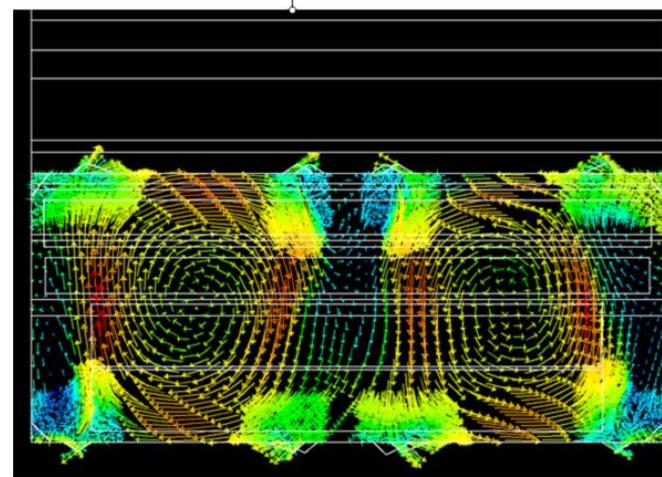
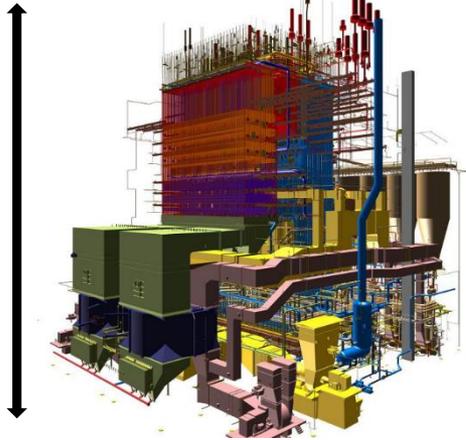
Predictive Modeling

Uncertainty Quantification



Exascale Target Problem DOE NNSA PSAAP II Center –

92 meters



boiler simulation

- Alstom Power 1000MWe “Twin Fireball” boiler
- Supply power for 1M people
- 1mm grid resolution = 9×10^{12} cells
- 1000 times larger than largest problems solved today

Target Problem

Radiation dominant mode of heat transfer

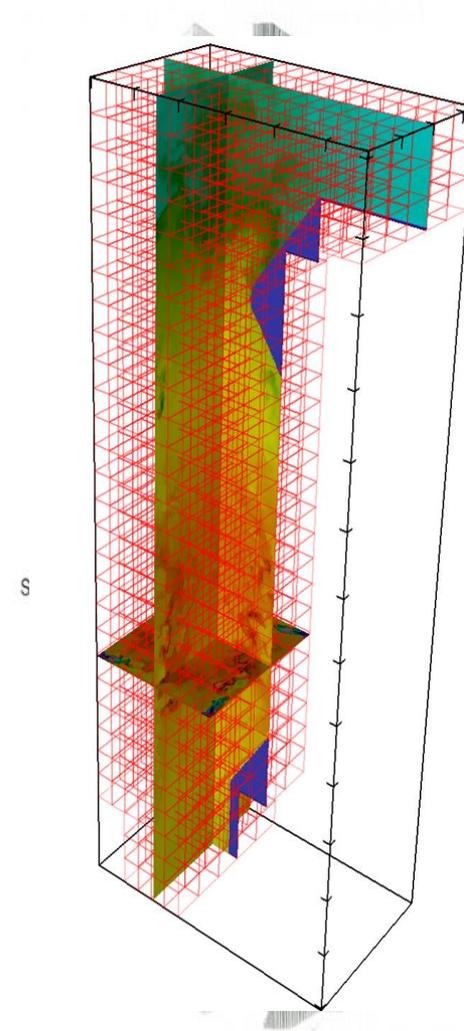
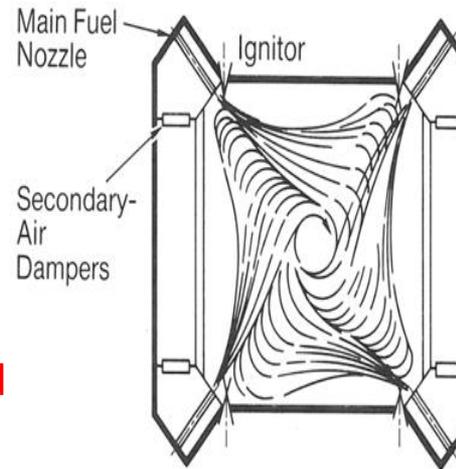
Radiative heat transfer (RTH) rates
~proportional to 4th power of the
temperature difference.

Combustion simulations highly influenced
by the accuracy of models with radiative
effects

Participating Media Particle laden,
scattering, absorbing, emitting

Domain Contains:

Coal particles, soot
CO₂ & H₂O (and other minor gases)
Spectral, Specular reflections
Walls & tubes with deposits



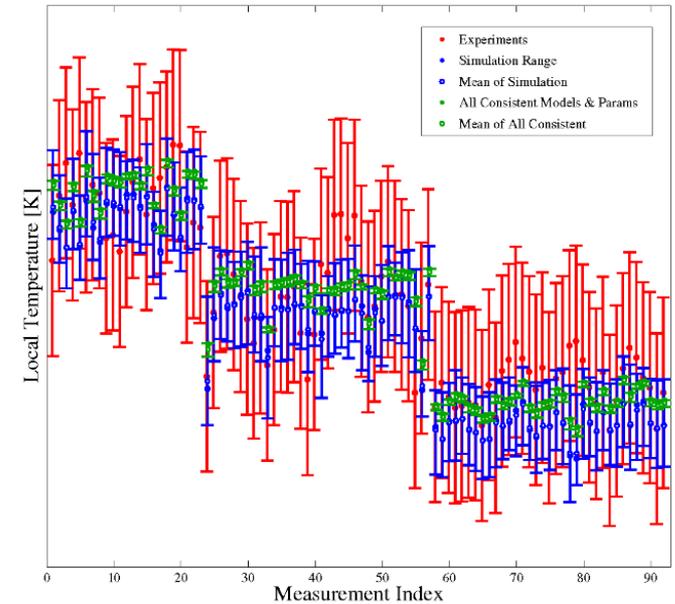
Existing Simulations of Clean coal Boilers using ARCHES in Uintah

(i) Traditional Lagrangian/RANS approaches do not address well particle effects

(ii) LES has potential to predict oxy---coal flames and to be an important design tool

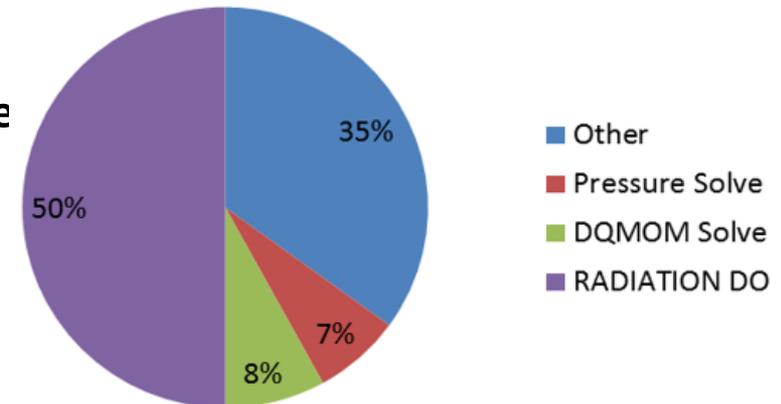
(iii) LES is “like DNS” for coal

- Structured, high order finite-volume
- 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)**
- **Radiation via Discrete Ordinates – massive solves or Ray tracing.**



Red is expt. Blue is sim.
Green is consistent

ARCHES CPU %



Linear Solves arises from Navier–Stokes Equations

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho \mathbf{u} = 0,$$

Full model includes turbulence, chemical reactions and radiation

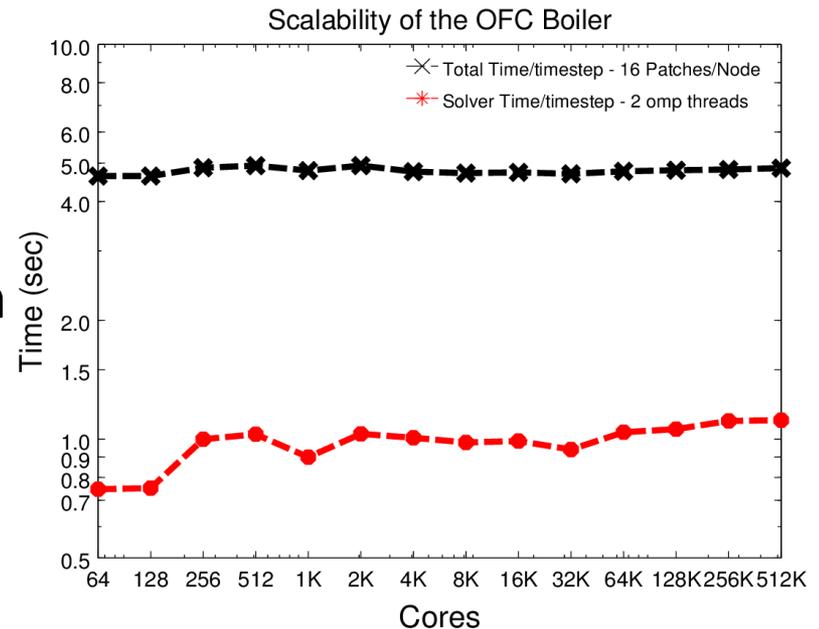
where ρ is density, \mathbf{u} is velocity vector and p is pressure

$$\frac{\partial \rho \mathbf{u}}{\partial t} = \mathbf{F} - \nabla p, \quad \text{where} \quad \mathbf{F} = -\nabla \cdot \rho \mathbf{u} \mathbf{u} + \nu \nabla^2 \mathbf{u} + \rho \mathbf{g}$$

Arrive at pressure Poisson equation

$$\nabla^2 p = R, \quad \text{where} \quad R = \nabla \cdot \mathbf{F} + \frac{\partial^2 p}{\partial t^2}$$

Use hypre Solver distributed by LLNL
Preconditioned Conjugate Gradients on regular mesh patches used Multi-grid pre-conditioner used. 20^3 patch with one MPI process per core no AMR or radiation.



Uintah Portability

Wide range of cpu + infiniband machines

Vulcan and Mira IBM BG/Q

Cray Blue Waters, Titan

Variety of Cray XC 30s

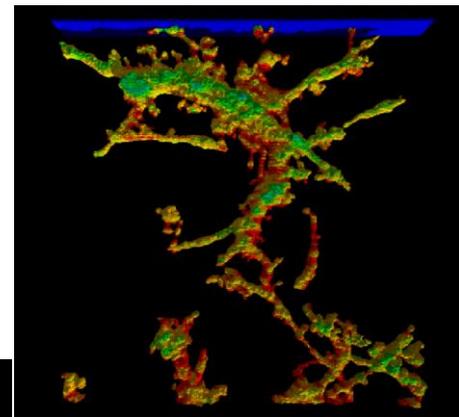
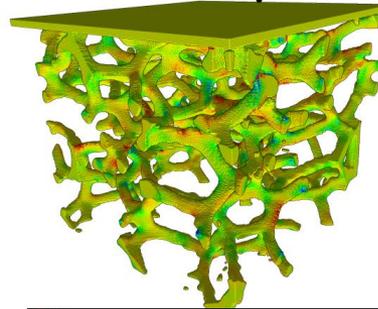
IBM Dataplex DOD

TACC Stampede CPU + Xeon Phi

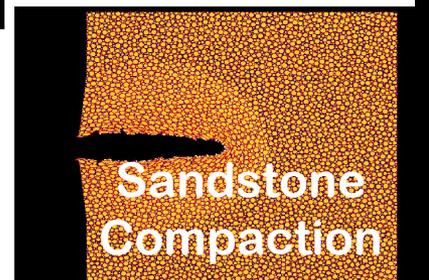
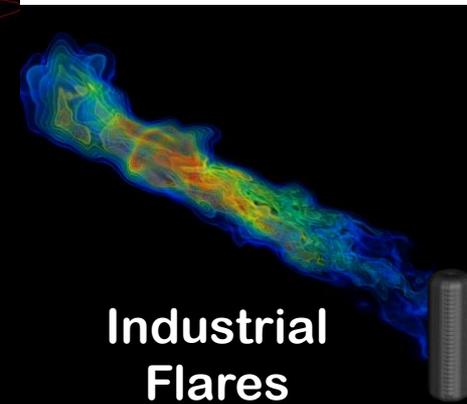
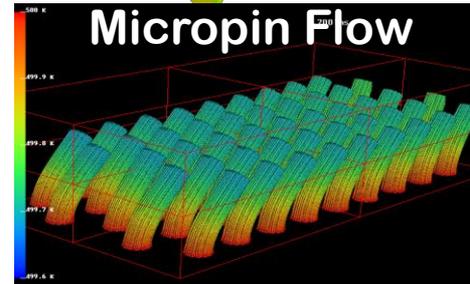
Many machines worldwide.....



Foam Compaction



Angiogenesis



Industrial Flares

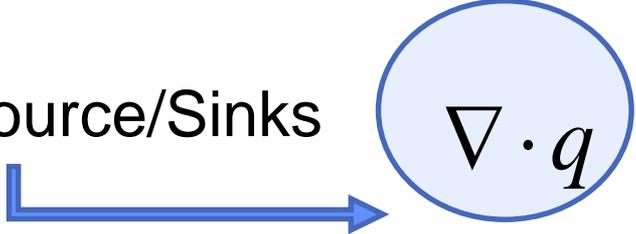
Utah PSAAP2 Center

Radiative Heat Transfer

**A key computational kernel using as much as
50% of run time**

Ray Tracing Radiation Overview

Solving **energy** and **radiative heat transfer** eqns

$$\frac{\partial T}{\partial t} = \text{Diffusion} - \text{Convection} + \text{Source/Sinks}$$


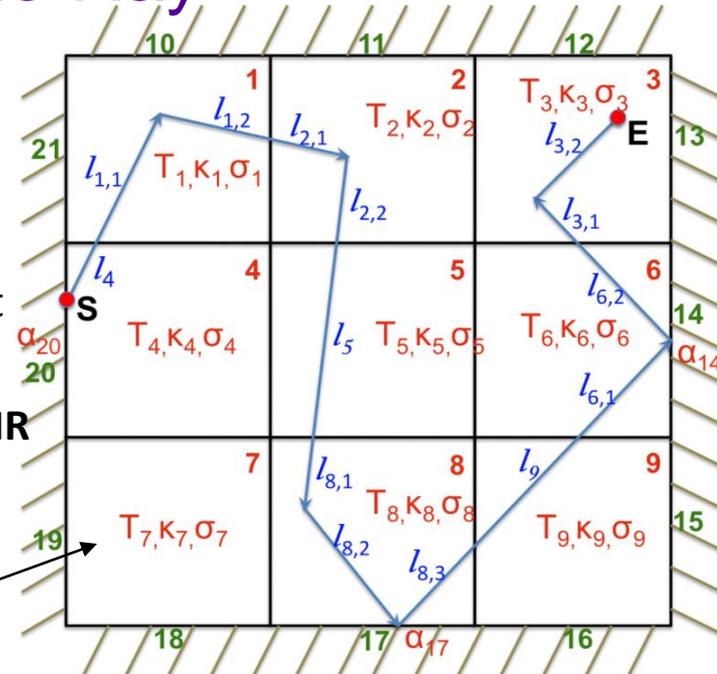
Need to compute the **net radiative source term** 

- *Energy equation conventionally solved by ARCHES (finite volume)*
- *Temperature field, T used to compute **net radiative source term***
- *Radiative sources fed back into energy equation for CFD calculation*

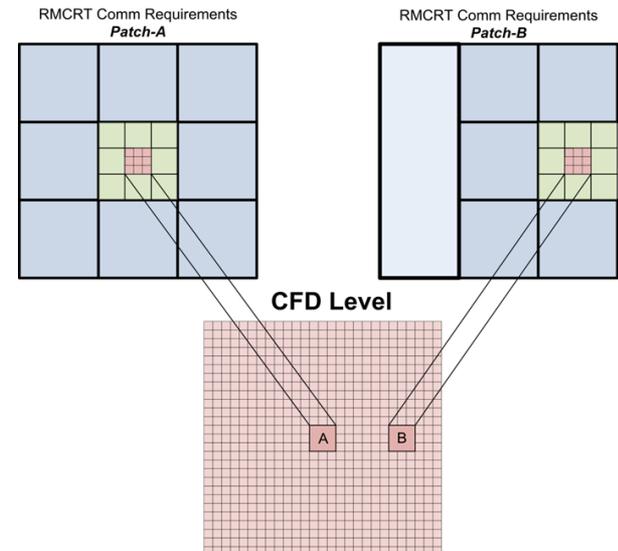
$$\nabla \cdot q = \kappa(4\pi I - \int_{4\pi} I d\Omega) \rightarrow \sum_{rays} \alpha_r I_r$$

Radiation: Reverse Monte Carlo Ray Tracing Algorithm

- Loose coupling to CFD due to time-scale separation.
 - Radiation timescales are typically longer than turbulent mixing timescales.
 - Required resolution decreases with distance → **use AMR to trace rays locally.**
- **RMCR: Incorporate dominant physics**
 - *Emitting / Absorbing Media and walls*
 - *Ray Scattering*
- **User controls # rays per cell**
 - *Each cell has Temp Absorb and Scattering Coeffs ,*
- **Radiative Heat Transfer key**
 - *Replicate Geometry on every node*
 - *Calculate heat fluxes on Geometry*
- *transfer cell information globally on coarse mesh except locally.*



4-Level Data Onion

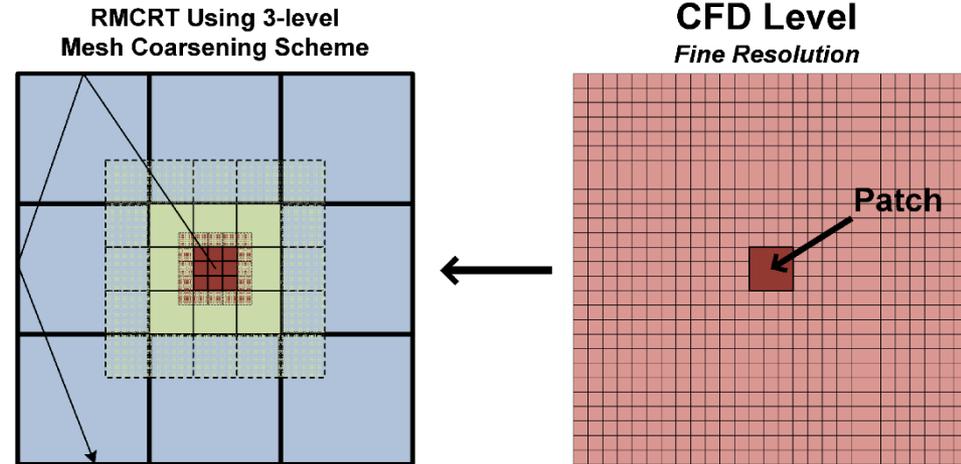


Multi-Level RMCRT Approach

- Principal single-level challenges:
 - all-to-all communication requirements
 - problem size limited - memory constraints with large, highly resolved physical domains

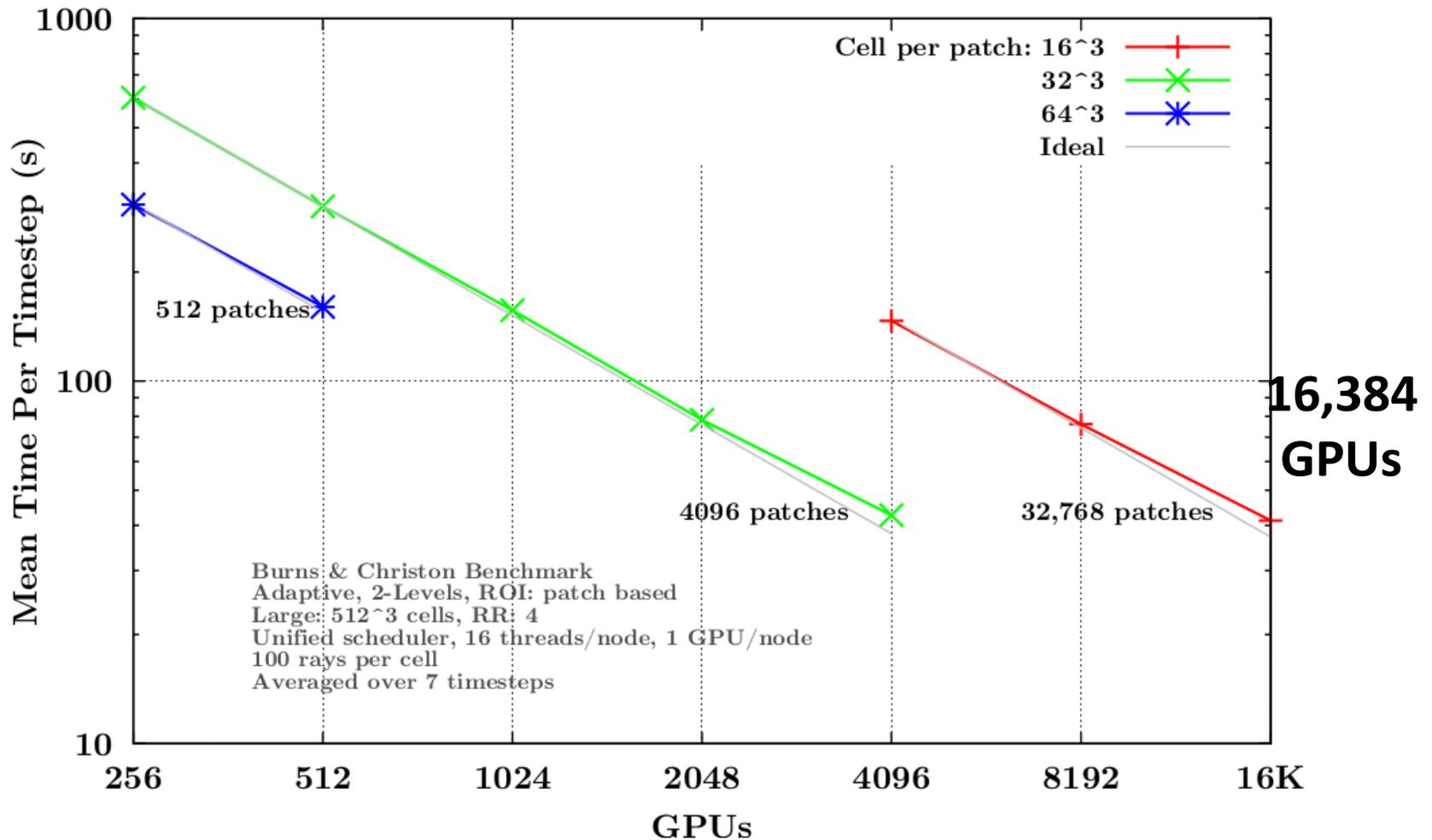
SOLUTION: AMR – multiple levels

- Locally use fine mesh - *local halo exchange*
- Globally use coarser approximation of global domain - *all-to-all* exchange of radiative properties
- As rays travel away from **local patch**, stride between cells becomes larger
- Significant reduction in computational cost, memory usage and **all-to-all** costs



RMCRT - 2D diagram of three-level mesh refinement scheme, illustrating how a ray from a fine-level patch (right) might be traced across a coarsened domain (left).

Two-Level RMCRT GPU Scalability on Titan



Preliminary Results – July 1, 2015

GPU Strong scaling of the two-level benchmark RMCRT problem on the DOE Titan system

Patch sizes: 16^3 , 32^3 and 64^3

Current Efforts on porting RMCRT to Xeon Phi

- **Native execution experiments have explored :**
 - optimal run configurations
 - use of the 61st (reserved) core on the Xeon Phi
 - thread affinity
 - mesh decomposition
- **Key Outcomes:**
 - **two Xeon E5s outperformed one Xeon Phi by 40%**
 - mesh decomposition had a profound impact on performance
 - No one thread placement strategy dominated performance
 - Differing caches of Sandy Bridge and MIC matter
 - **Simple vectorization approaches failed (e.g. SIMD directives)**
 - Many options are available to explore (e.g. manual intrinsics and software packages such as Kokkos (SNL) or RAJA (LLNL))
 - **Complete code overhaul necessary e.g. 1.5x speedup by changing data access mechanism**

Performance at Node Level

Problematic for stencil codes on real applications

Easy stencils are easy.....

Few flops per byte.

Performance governed by data transfer rates often

Percentage of peak low 3-5% or 10-12% with a lot of work (PPM)

Corresponds to HPCG benchmark much more closely than Linpack

Opportunities with Low-level portability layers

Domain specific Languages may help

An Example of A Problematic Uintah Task

Computing the pressure in the ICE explicit fluids code values defined at nodes $j, j+1$. Need to define values at midpoints and **pressure halfway in time $n+1/2$**

$$U_{j+1/2}^{FC} = U_{j+1/2}^{AVG} - \frac{\Delta t}{2\Delta x \rho_{j+1/2}^{avg}} \left[\left(p_{j+1}^n - p_j^n \right) + \left(\delta p_{j+1}^n - \delta p_j^n \right) \right]$$

where $\delta p_{j+1}^n = \left(p_j^{n+1/2} - p_j^n \right),$

The iteration is

$$\delta p_{j+1}^n = -\frac{\Delta t}{2\Delta x} u_j^n \left(p_{j+1}^n - p_{j-1}^n \right) - \frac{\Delta t}{2\Delta x} \left(c^2 \rho \right)_j^n \left(U_{j+1/2}^{FC} - U_{j-1/2}^{FC} \right)$$

Easy in 1D for one material, but in 3D for multiple materials?

```
void ICE::computeEquilibrationPressure(const ProcessorGroup*,  
    const PatchSubset* patches, const MaterialSubset* /*matls*/,  
    DataWarehouse* old_dw, DataWarehouse* new_dw)
```

Loop over patches

Get data from Old data warehouse and allocate space in new data Warehouse

Compute volume fractions and initial density

now loop over every cell

```
for (CellIterator iter=patch->getExtraCellIterator();!iter.done();iter++) {  
    while ( count < d_max_iter_equilibration && converged == false) {        count++;  
        evaluate press_eos at cell i,j,k
```

Compute delPress - update press_CC

Compute updated volume fractions

Test for convergence

Find the speed of sound based on converged solution

Save iteration data for output in case of crash

BULLET PROOFING CHECKS positivity, mass fraction etc

Boundary conditions

Indeterminate loops while testing for convergence , calls to 2 functions

11 loops over number of materials about 350 lines of robust safe C++

Ice Code Pressure Calculation

Problem Size	CPU 12 patches and threads	GPU 12 patches and gpu tasks	GPU 1 large patch and GPU task	Speedup
50x50x50	0.00036	0.00032	0.0026	1.37
64x64x64	0.0072	0.00611	0.0035	2.03
100x100x100	0.0255	0.0203	0.00923	2.76
128x128x128	0.0488	0.037	0.0196	2.48

Significant penalty for running lots of smaller tasks on smaller arrays on a GPU. Need to reduce GPU overhead

GPU Engine and Scheduler Improvements

New GPU engine can now keep data resident on the GPU, avoiding unnecessary PCIe bus data transfers and automatically handles ghost/halo cells across multiple CPU threads, multiple GPUs per node, and multiple nodes. Also reduced overhead so that fast tasks do not have undue overhead.

Poisson Problem	Problem Size	Number of Patches	Avg Time Per Iteration
CPU engine (Xeon 12 cores)	400x400x400	12 (3x2x2)	0.653
Current GPU engine (K20)	400x400x400	12 (3x2x2)	0.344
New GPU engine (K20)	400x400x400	2 (1x1x2)	0.098

DSL: “Matlab for PDEs on Supercomputers”

Field & stencil
operations:

(and much more)

$$\text{rhs} = -\frac{\partial}{\partial x}(J_x + C_x) - \frac{\partial}{\partial y}(J_y + C_y) - \frac{\partial}{\partial z}(J_z + C_z)$$

```
rhs <<= -divOpX( xConvFlux + xDiffFlux )  
        -divOpY( yConvFlux + yDiffFlux )  
        -divOpZ( zConvFlux + zDiffFlux );
```

Can “chain” stencil operations.

- 70+ natively supported discrete operators (extensible).
- CPU (serial, multicore), GPU, Xeon Phi backends.
- Field can live in multiple locations (GPU, CPU) *simultaneously*.

Auto-generate code for efficient execution on CPU, GPU, XeonPhi, etc. during compilation.

Express complex pde functions as DAG - automatically construct algorithms from expressions

Define field operations needed to execute tasks (fine grained vector parallelism on the mesh)

User writes only field operations code . Supports field & stencil operations directly - no loops!

Strongly typed fields ensure valid operations at compile time. *Allows implementations to be tried without modifying application code.*

Scalability on a node - use **Uintah** infrastructure to get scalability across whole system

NEBO/Wasatch Example

[Sutherland Earl Might]
Energy equation

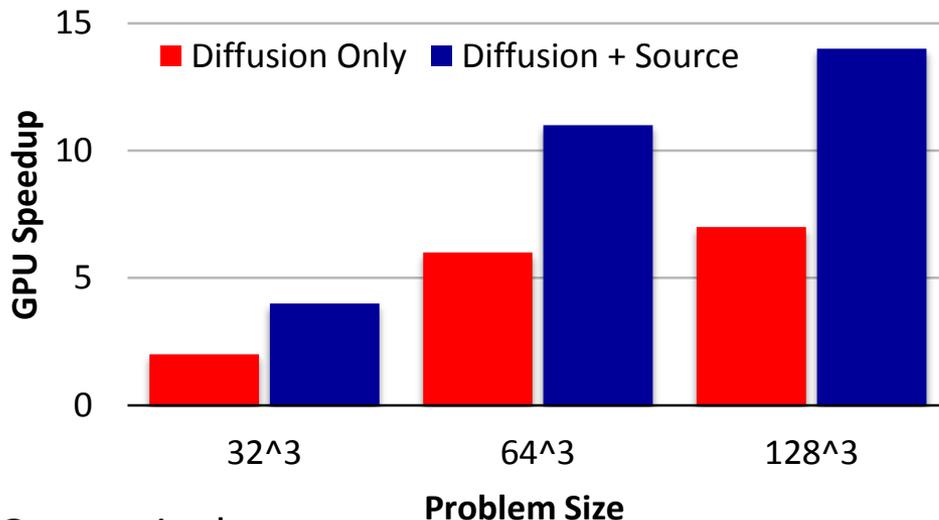
$$\frac{\partial \rho e}{\partial t} + \nabla \cdot (\rho e \underline{u}) + \nabla \cdot \underline{J}_h + terms = 0$$

Enthalpy diffusive flux

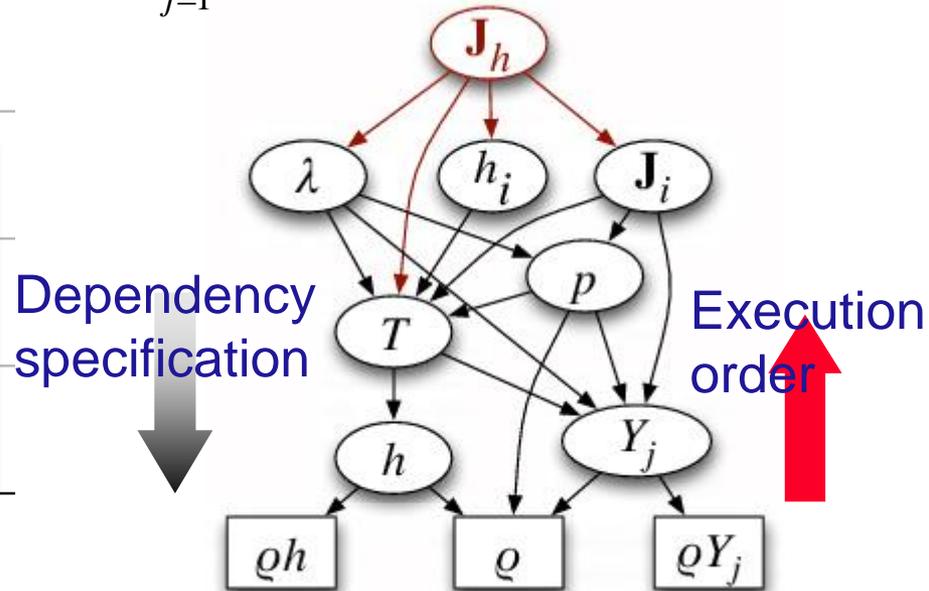
$$\underline{J}_h = -\lambda(T, Y_j) \nabla T - \sum_{i=1}^n h_i \underline{J}_i$$

$$\underline{J}_i = -\sum_{j=1}^{ns} D_{ij}(T, Y_j) \nabla Y_j - D_i^T(T, Y_j) \nabla T$$

Standalone

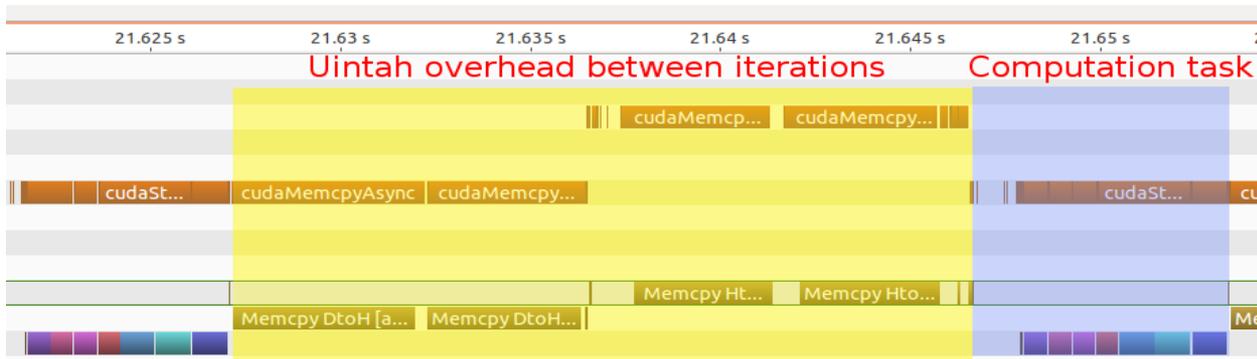


Over a single core

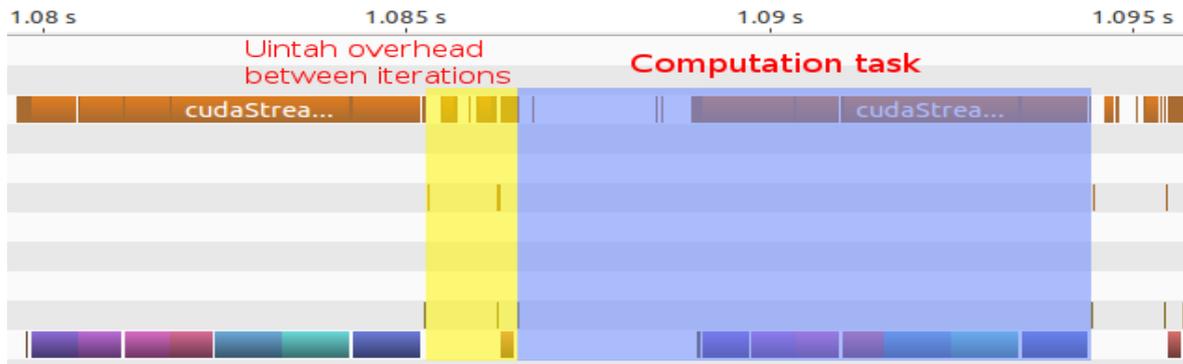


Example of Improvements – Wasatch

□ Before



□ Current



□ For this problem, Uintah overhead was reduced from ~19 milliseconds to 1-2 milliseconds. Work is being done to improve this further.

□

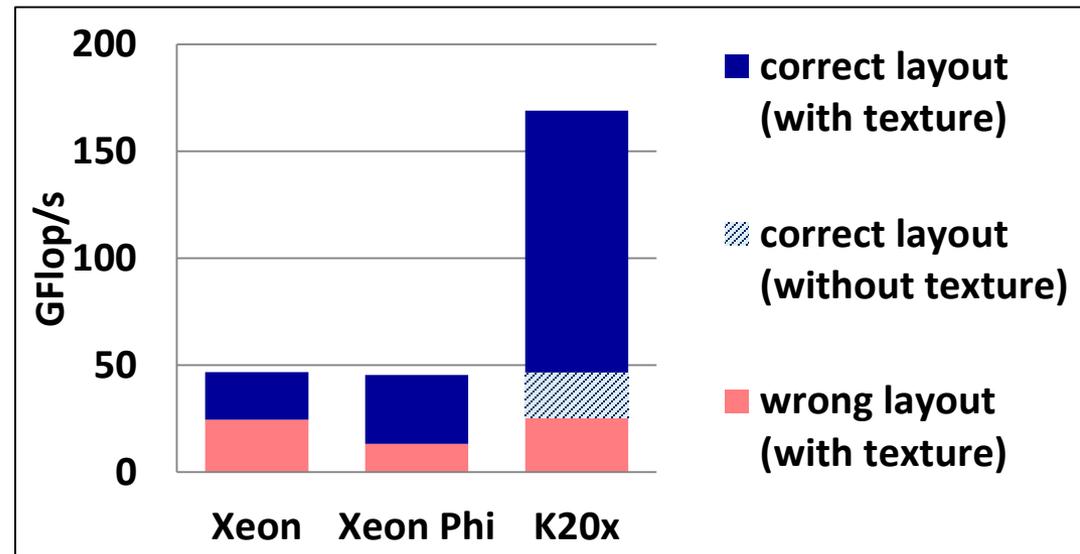
□ Results were obtained on a Maxwell NVIDIA GPU. We have noticed that Kepler GPUs have somewhat higher GPU API latencies.

Kokkos: A Layered Collection of Libraries

- **Standard C++, Not a language extension** [source Carter Edwards Dan Sunderland]
 - *In spirit* of TBB, Thrust & CUSP, C++AMP, LLNL's RAJA, ...
 - *Not* a language extension like OpenMP, OpenACC, OpenCL, CUDA, ...
- **Uses C++ template meta-programming**
- **Multidimensional Arrays, *with a twist***
 - **Layout mapping: multi-index (i,j,k,...) ↔ memory location**
 - **Choose layout to satisfy device-specific memory access pattern**
 - **Layout changes are invisible to the user code**

- **MD LJ Kernel Test**

- **864k atoms, ~77 neighbors**
 - **2D neighbor array**
 - **Different layouts CPU vs GPU**
 - **Random read 'pos' through GPU texture cache**
- **Large performance loss with wrong array layout**



Uintah Laplace Eqn Example

```
for(NodeIterator iter(l, h); !iter.done(); iter++)
```

OLD

```
{ IntVector n = *iter;
newphi[n]=(1./6)*( phi[n+IntVector(1,0,0)] + phi[n+IntVector(-
1,0,0)] + phi[n+IntVector(0,1,0)] + phi[n+IntVector(0,-1,0)] +
phi[n+IntVector(0,0,1)] + phi[n+IntVector(0,0,-1)]);
double diff = newphi[n] - phi[n]; residual += diff * diff;
}
void operator()(int i, int j, int k, double & residual)
```

NEW

```
const { const IntVector n(i,j,k); (*m_newphi)[n]=(1./6)*(
m_phi[IntVector(i+1,j,k)] + m_phi[IntVector(i-1,j,k)] +
m_phi[IntVector(i,j+1,k)] + m_phi[IntVector(i,j-1,k)] +
m_phi[IntVector(i,j,k+1)] + m_phi[IntVector(i,j,k-1)]);
double diff = (*m_newphi)[n] - m_phi[n]; residual += diff * diff;
}
```

NEW is 2X faster than OLD on cpus

Summary

- **The five abstractions are all important for portability, scaling and for not needing to change applications code**
- **Most significant challenge is development cost perhaps?**
- **Scalability** still requires tuning the runtime system. **Cannot develop nodal code in isolation.**
- **Future Portability:** use Kokkos for rewriting legacy applications +Wasach/Nebo DSL for new code. MIC and GPU ongoing. Aiming at future DOE machines Summit and Aurora.
- **What about really challenging apps?**

Acknowledgments. This material is based upon work supported by the Department of Energy, National Nuclear Security Administration, under Award Number(s) DE-NA0002375, and by DOE ALCC award CMB109, "Large Scale Turbulent Clean Coal Combustion", for time on Titan. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. We would also like to thank all those involved with Uintah past and present, Isaac Hunsaker and Qingyu Meng in particular.