A Portable Applications- Driven Approach to Scalability on Present and Future Exascale Systems

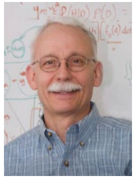
John Holmen* Alan Humphrey, Brad Peterson, Damodar Sahasabarude, John Schmidt & Martin Berzins
Scientific Imaging and Computing Institute
University of Utah

1. Exascale challenges
2. Nodal scalability via Uintah, runtimes and programming models
3. Scaling challenging global problems (radiation)
4. Performance portability using Kokkos
5. Conclusions

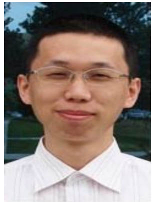
*Intel Parallel Computing Center

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

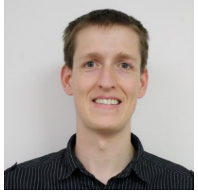


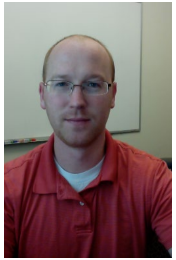



Part of Utah PSAAP Center

PSAAP2 Applications Team

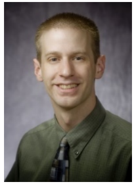
Todd Harman Jeremy Thornock Derek Harris Ben Isaac

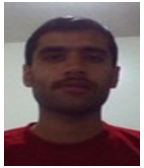




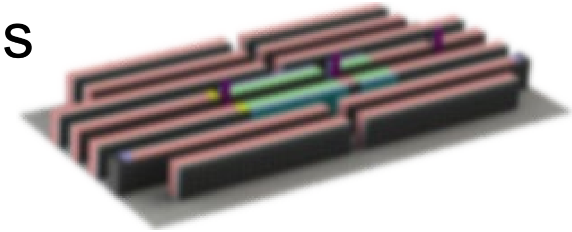
PSAAP Extreme Scaling team

John Schmidt Alan Humphrey John Holmen Brad Peterson Damaodar Sahasbaude





Exascale Machines Possible Timelines



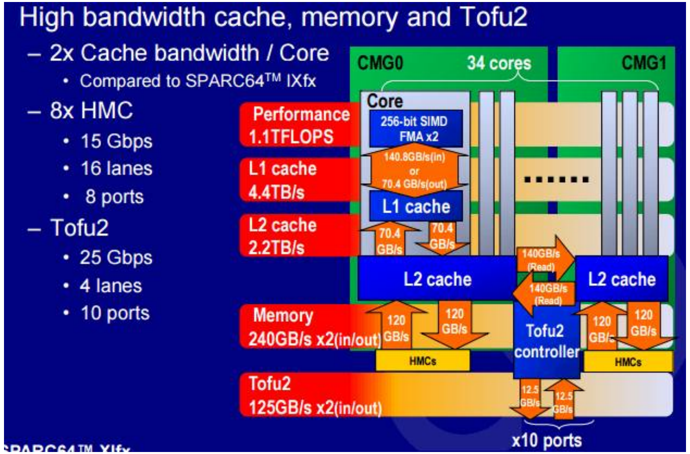
2018 Summit (Oak Ridge) and Sierra (LLNL)   <4,500 nodes with 2 power 9 + 6 Volta GPUs 120- 200F?

2020 Tianhe 3

2020 Post K Machine

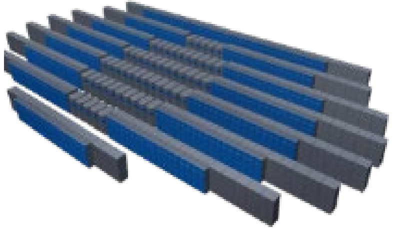
2020/21 Sunway Exascale

2020/21 Sugon Exascale



2020/21   Argonne A21   Intel Architecture

2021   Oak Ridge Frontier 1,000–3,000 PF
LLNL “El Capitain”



All of these are “novel” architectures GPU Arm Custom etc

Addressing the challenges of multi-scale multi-physics applications on varied future architectures

(i) Use asynchronous many task (AMT) approaches to ensure that the compute nodes always have work to do .

(ii) Look at the scalability of challenging .nonstandard algorithms.

(iii)Make sure that tasks on nodes can run in a portable fashion and as efficiently as possible without code changes.

Addressing the challenges of multi-scale multi-physics applications on varied future architectures

(i) Use asynchronous many task (AMT) approaches to ensure that the compute nodes always have work to do .

Illustrate this with the Uintah software

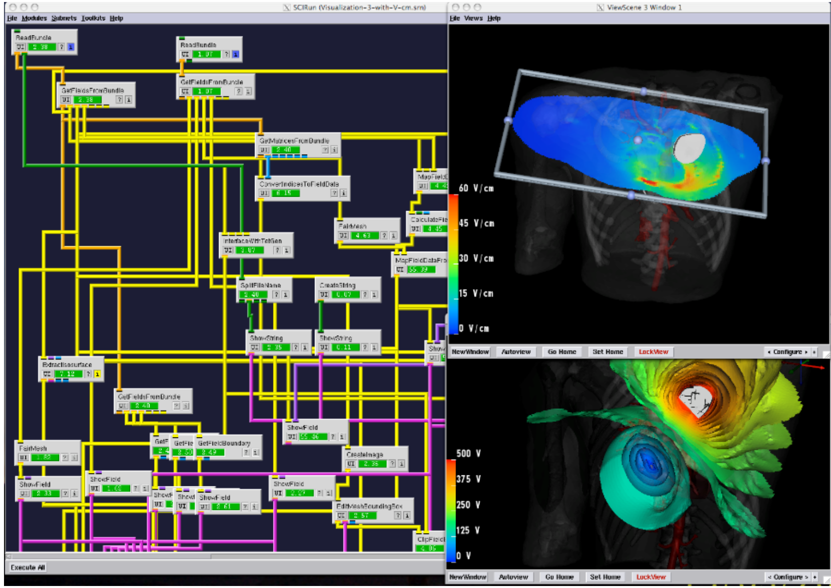
(ii) Look at the scalability of challenging .nonstandard algorithms.

Consider the scalability of global radiation problems

(iii)Make sure that tasks on nodes can run in a portable fashion and as efficiently as possible without code changes.

Use the Kokkos scalability library

Uintah development timeline



Task Based approach by Steve Parker Originated in SCIRun problem solving (workflow) environment for large scale biomedical problems .

Simple programming model- separation physics and computer science

Developed independently of Charm++ and Sarkar.

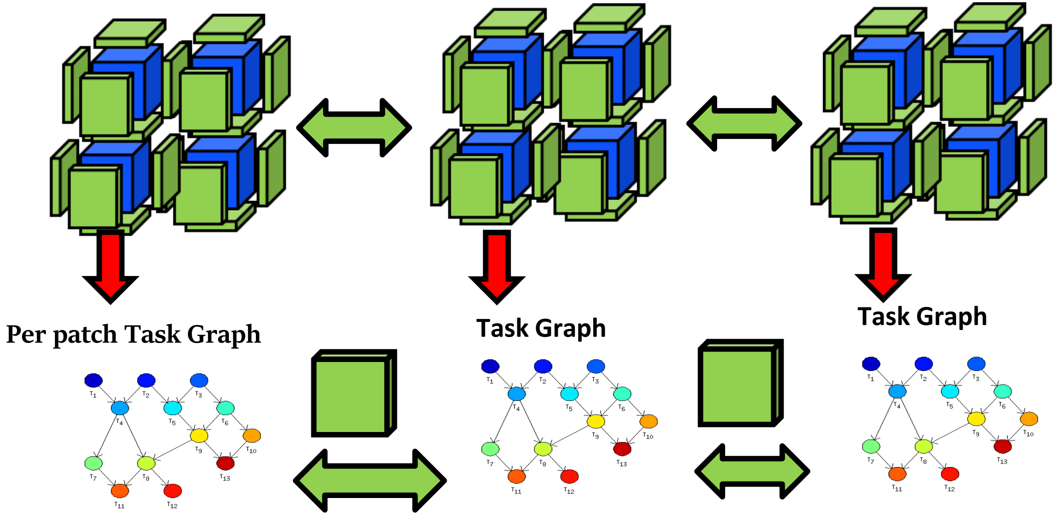
- 1998-2007 CSAFE ASCI Center- static execution of task graphs, complex multiphysics . Steve goes to NVIDIA ☹.
- 2008-2010 CSAFE Full physics, AMR for fluid-structure
- 2010-2015 Adaptive asynchronous. out-of-order task execution
- 2014- PSAAP2 Center Turbulent Combustion - full scalability on Titan Mira, Blue Waters – moving to exascale portability?

THE UNIVERSITY OF UTAH

Uintah Asynchronous Many Task (AMT) Approach 2008…

In Uintah dynamic task graph execution needed for more than 100K cores

e.g. three compute nodes 12 mesh patches

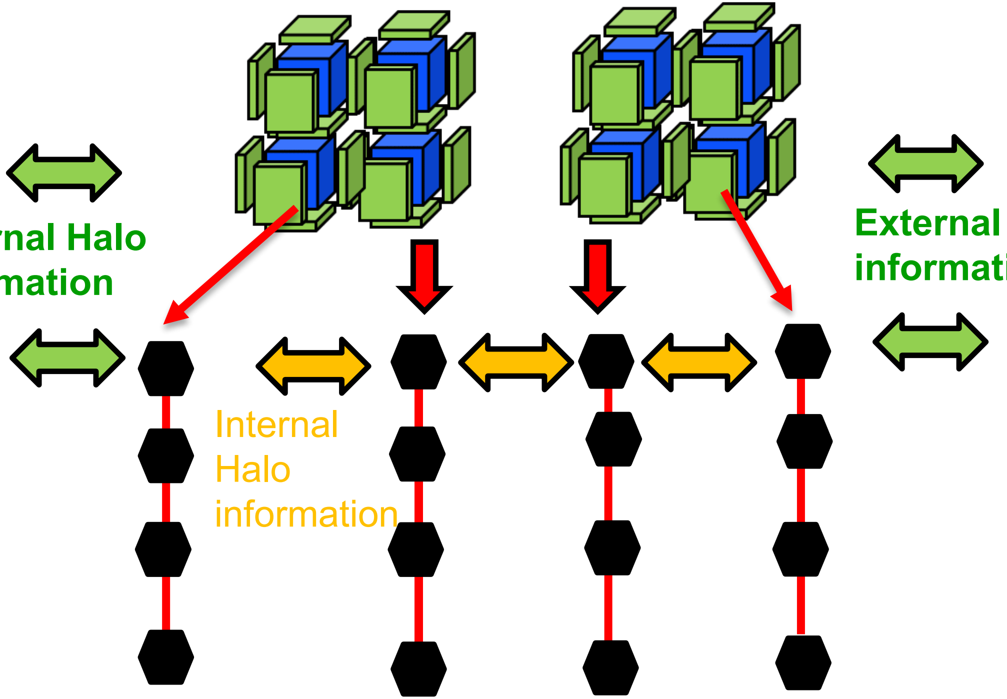


Execute tasks when possible communicating as needed. Do useful work instead of waiting. Execute tasks out of order if possible

Over-decomposition in the Uintah AMT Approach

e.g. one compute core 8 mesh patches consider bottom 4

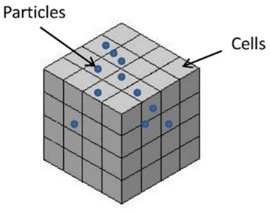
Multiple Patches on a single core
allow flexibility of execution



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

Uintah Architecture Review

PDE Applications
Code
Components



Uintah Patch

About 1.2 M lines 500K "core' C++

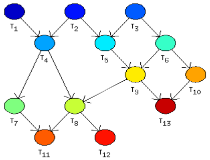
ICE FV Fluids — 50K Lines

MPM Particles — 250K lines

ARCHES — 250K lines

Task Graph Compilation

Automatically generated abstract C++ task graph form With mpi compiled in



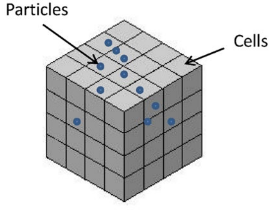
Asynchronous Task Runtime System

Kokkos Portability Library

GPUs ***CPUs*** ***Xeon Phis*** ***A.N.OTHER***

Uintah Architecture Review

PDE Applications Code Components

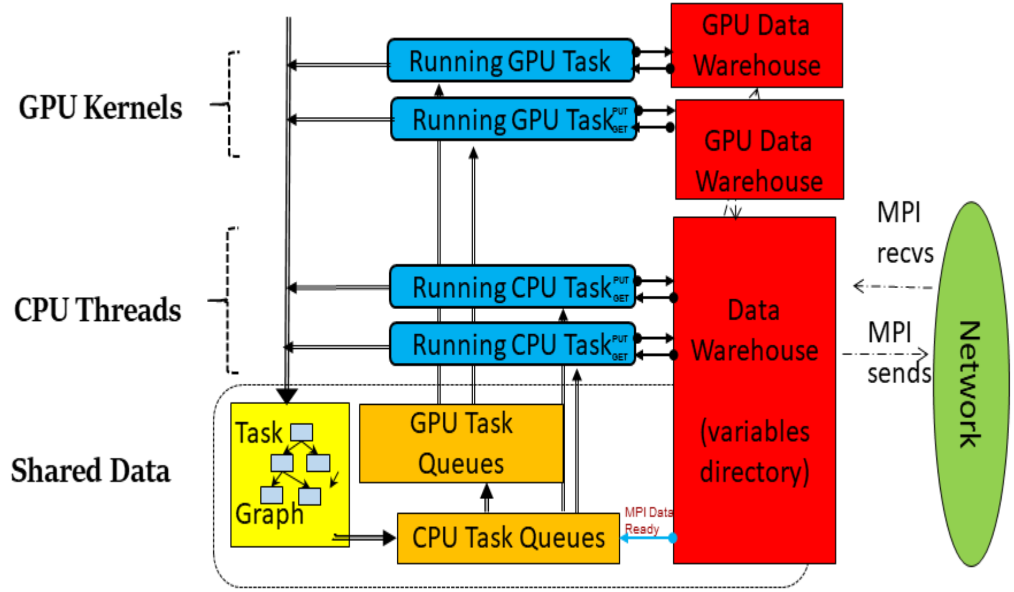


Uintah Patch

About 1.2 M lines 500K "core' C++

ICE FV Fluids	50K Lines
MPM Particles	250K lines
ARCHES	250K lines

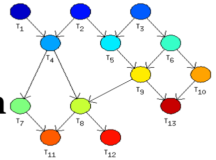
Uintah: Node Architecture



CPU cores and GPUs pull work from task queues

Task Graph Compilation

Automatically generated abstract C++ task graph form With mpi compiled in

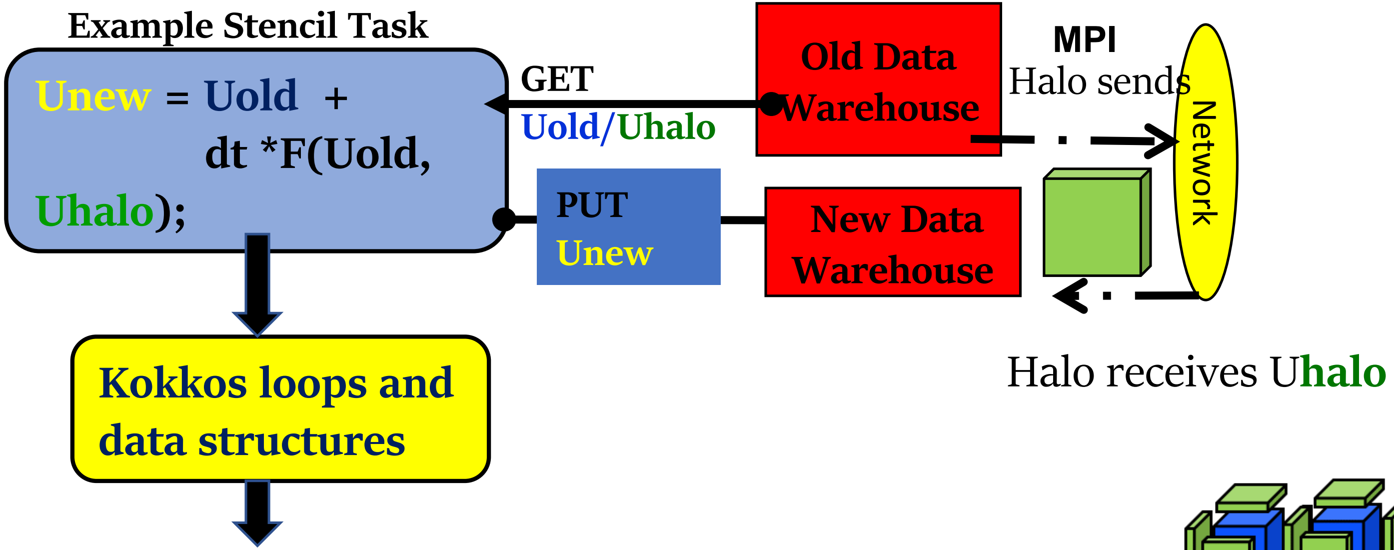


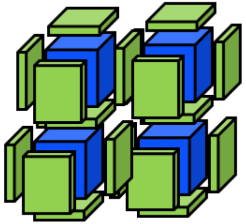
Asynchronous Task Runtime System

Kokkos Portability Library

GPUs	*CPUs*	*Xeon Phis*	*A.N.OTHER*

New Uintah Programming Model for Stencil Timestep

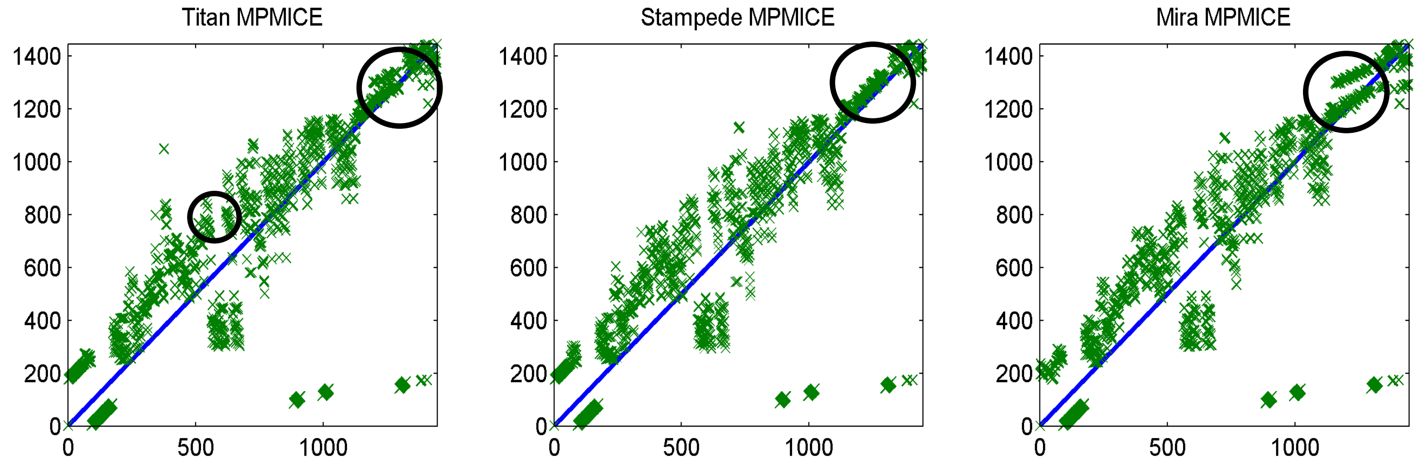




```
Uintah::BlockRange range ( patch->getCellLowIndex(),
patch->getCellHighIndex() );
Uintah::parallel_for ( range, [&]( int i,  int j, int k ) {
  char_rate[I,j,k] = 0.0;
  . . .
}
```

Automatically calls non-Kokkos, Kokkos OpenMP or Kokkos cuda, depending on build

Scalability is at least partially achieved by not executing tasks in order e.g.

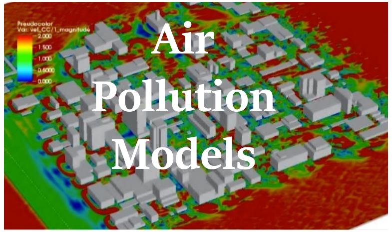


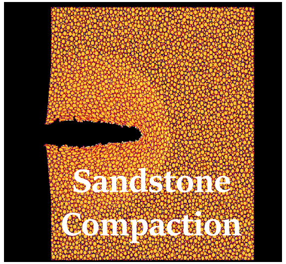
Straight line represents given order of tasks . X   shows when a task is actually executed.
Above the line means late execution while below the line means early execution took place. More “late” tasks than “early” ones as e.g.

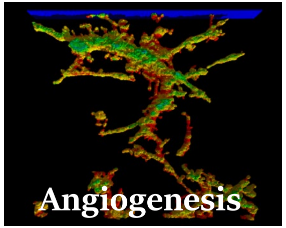
TASKS: 1 2 3 4 5

1 4 2 3 5

Early Late execution

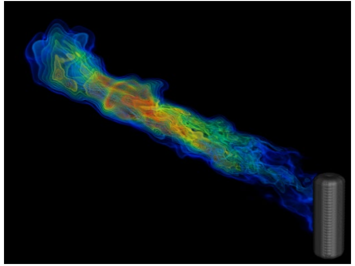


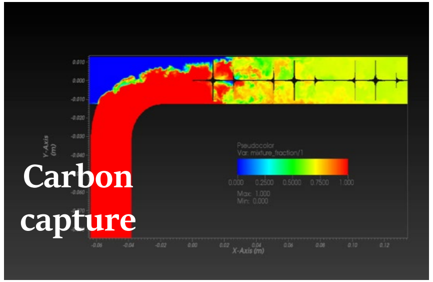


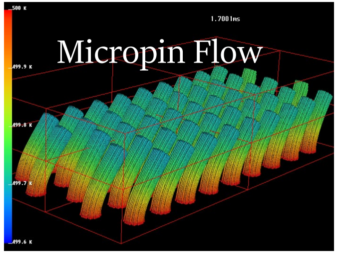


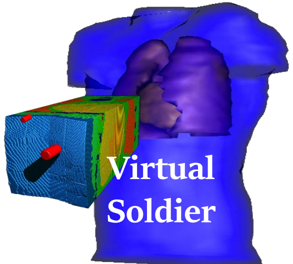
A Few Uintah Apps Codes Examples

- **Arches:** industrial flares John Zink, ultra Low Nox: Chevron Fives, CO2 mineralization Calera Corp, LES with REI consulting, Mitsubushi Heavy Industries low Nox, General Electric Boilers + many universities. Radiation and LES models
- **ICE:** semiconductor devices, flow over cities, accidental detonations, turbulence , reactive models Air Force
- **MPM:** fundamental analysis. Army Research Lab Center in Materials Modeling, novel battery models with silicon, penetration and fracture models for oil industry , Darpa heart injuries, angiogenesis. Many different solid mechanics models.



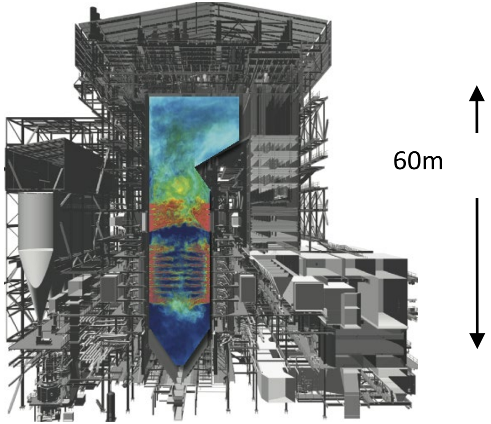






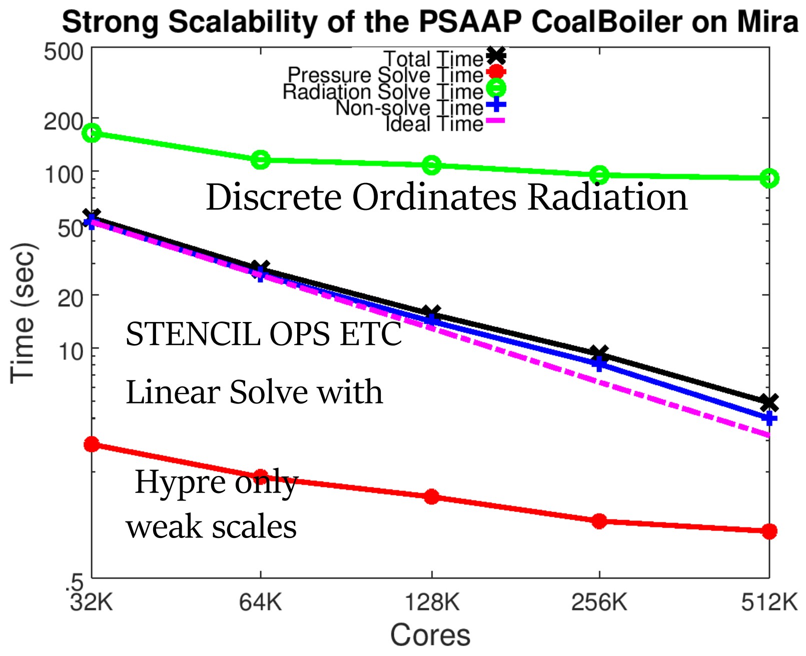
NNSA PSAAP2 Existing Simulations of GE Clean(er) Coal Boilers

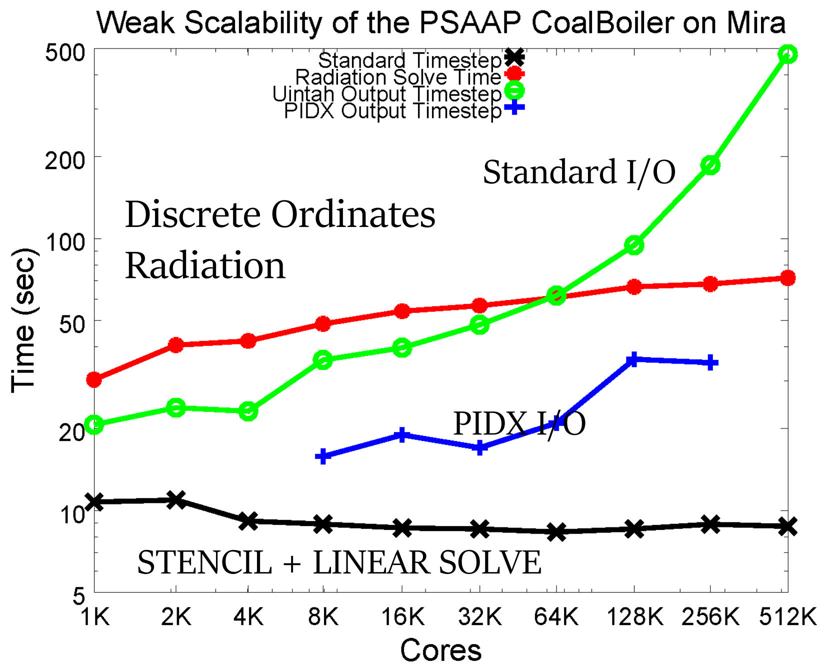
- Large scale turbulent combustion needs mm scale grids 10^14 mesh cells 10^15 variables (1000x more than now)
- Structured, high order finite-volume discretization
- Mass, momentum, energy conservation
- LES closure, tabulated chemistry
- PDF mixing models
- DQMOM (many small linear solves)
- Uncertainty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

Uintah scales for the Boiler problem on the largest machines that we have access to



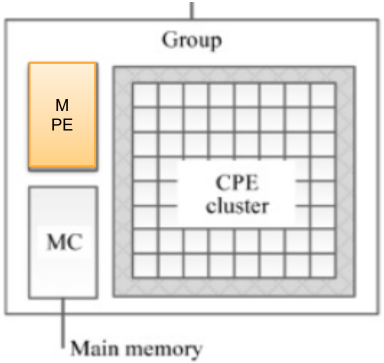


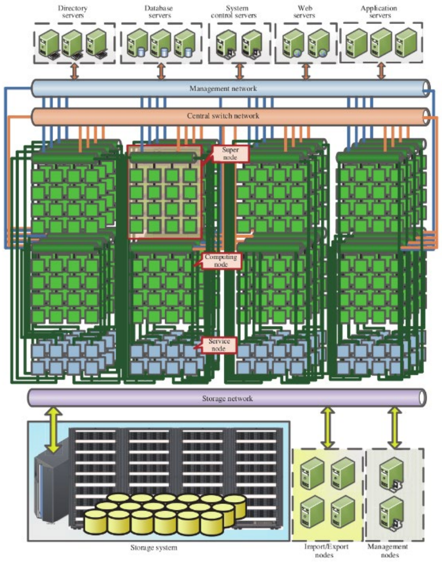
Full physics multi-level GPU-RMCRT strong scales on Titan

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

Shenwei TaihuLight Architecture:

- Each Sunway Compute node contains 4 **core groups (CGs).**
- **CG :** 1 Management Processing Element (**MPE)** and **64 CPEs** Computing Processing Elements
- **MPE** handles the main control flow / management, communications and computations and shares its memory with.....
- **cpes** are used to perform computations. These can be considered as "coprocessor" used to offload computations. With 256 vector instructions. Cacheless but with shared scratch memory 64K (LDM)
- 10M cores 93PF **vectorization and comms hiding keys to success.**





Source https://science.energy.gov/~/media/ascr/ascac/pdf/meetings/201609/Dongarra-ascac-sunway.pdf

Sunway specific changes Damodar and Zhang Yang IAPM (NSF)

Infrastructure and Scheduler: 200 lines of new code

- Updated offloading and polling mechanism using OpenACC

Computational Kernel / Task: 200 lines of new code

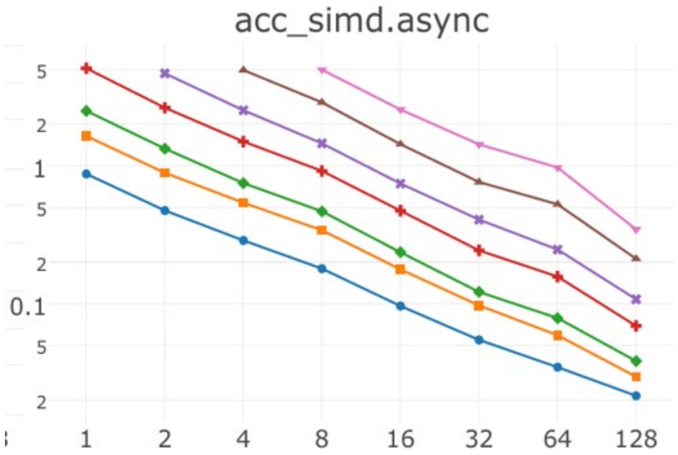
- Porting of Kernel: -main comp. kernel rewritten using Fortran, C, OpenACC and native athread runtime as CPEs do not support C++ low level SIMD instructions
- Need to use athreads low-level SIMD commands to overcome OpenACC slowdowns

Optimizations:

Tiling: The CPE part of scheduler divides tiles among CPEs.

Vectorization: Used native SIMD vector intrinsics for vectorization

Perfect scaling out to 8192 cores on Sunway development queue. IPDPS PDSEC 2018 paper



Weak and Strong Scalability of a Challenging Thermal Radiation Case

Radiation Overview

<u>Solving **energy** and **radiative heat transfer** equations simultaneously</u>

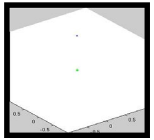
$$\frac{\partial T}{\partial t} = \text{Diffusion} - \text{Convection} + \text{Source/Sinks} \rightarrow \nabla \cdot q$$

Divergence of heat flux

- *Energy equation conventionally solved by ARCHES (finite volume)*
- *Temperature field, **T** used to compute **net radiative source term,** requires integration of incoming intensity about a sphere*

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

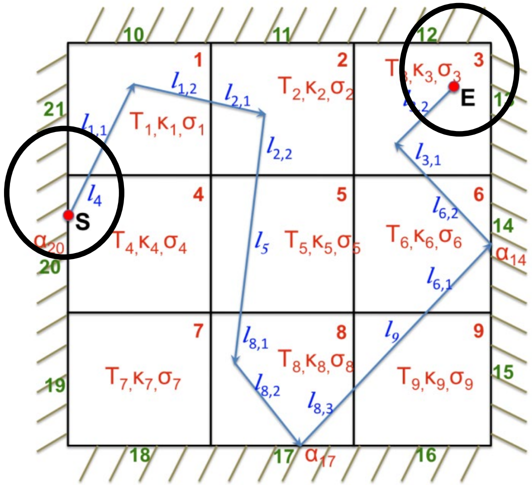
- ***Net radiative source term** goes back into ongoing CFD calculation*

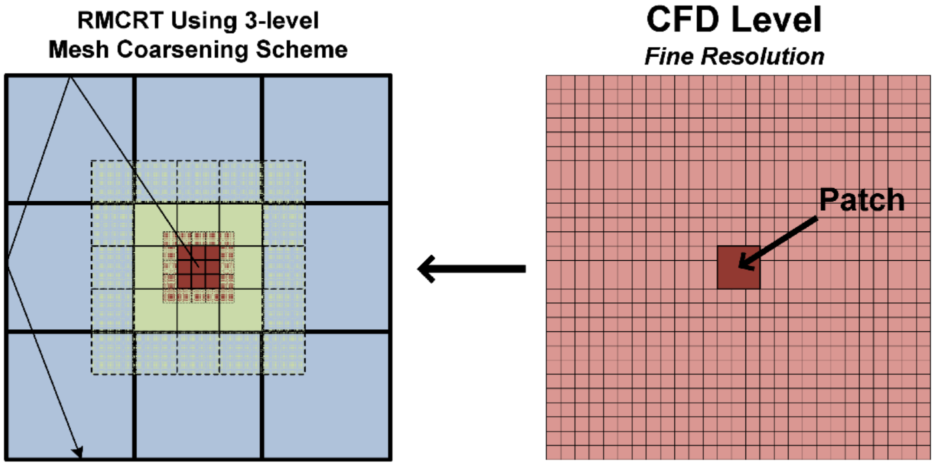


```
for all cells in a mesh patch do
     sumI = 0     // init sum of radiative intensity
     for all rays in a cell do
       findRayDirection()
        findRayLocation()
        updateSumI()  // sum incoming intensities
     end for
     compute  ∇·q
end for
add              back into ongoing CFD calculation
```

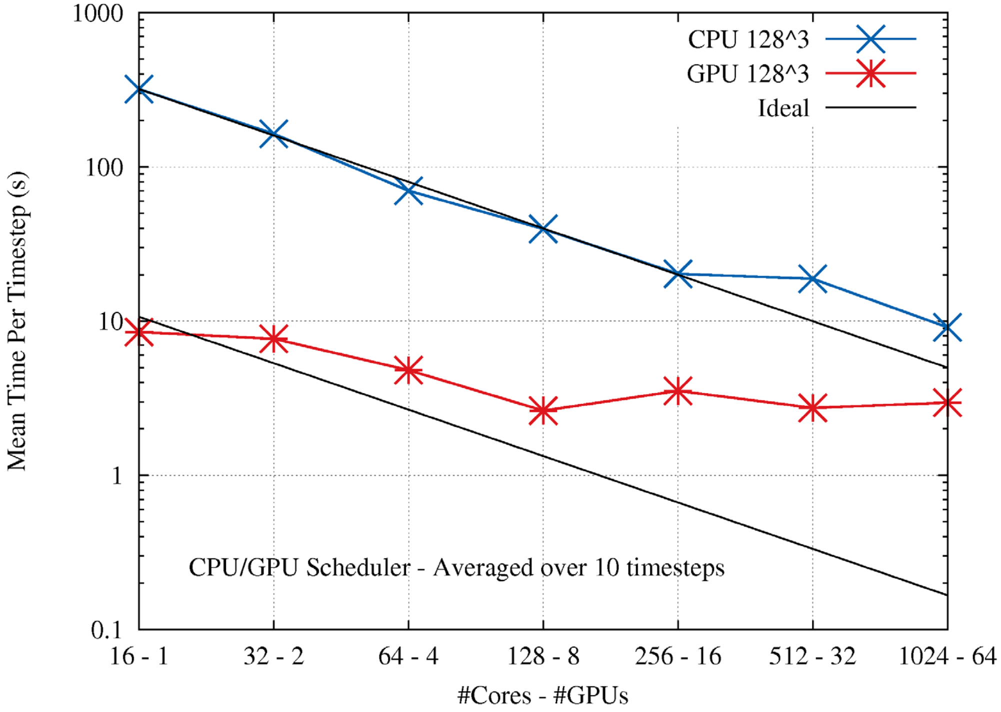
Radiation Overview

- Including Radiation means that every one of 10^10 cells may be connected to every other cell
- Model radiation using Monte Carlo ray tracing (RMCRT)
- Replicate AMR versions of the mesh on each node
- Ray trace in parallel
- Radiative properties and radiative fluxes calculated on each node and their **AMR** values transmitted to minimize communication volume in all-to-all.





No AMR GPU-Based RMCRT Scalability



Mean time per timestep for GPU lower than CPU (up to 64 GPUs)

GPU implementation runs out of work, **communication dominates**

All-to-all nature of problem limits size that can be computed due to memory constraints with large, highly resolved physical domains

Strong scaling results for both CPU and GPU implementations of single-level RMCRT on TitanDev

Nested AMR mesh of p levels

Each box:

8x volume of the one inside it

with same number of n^3 points .

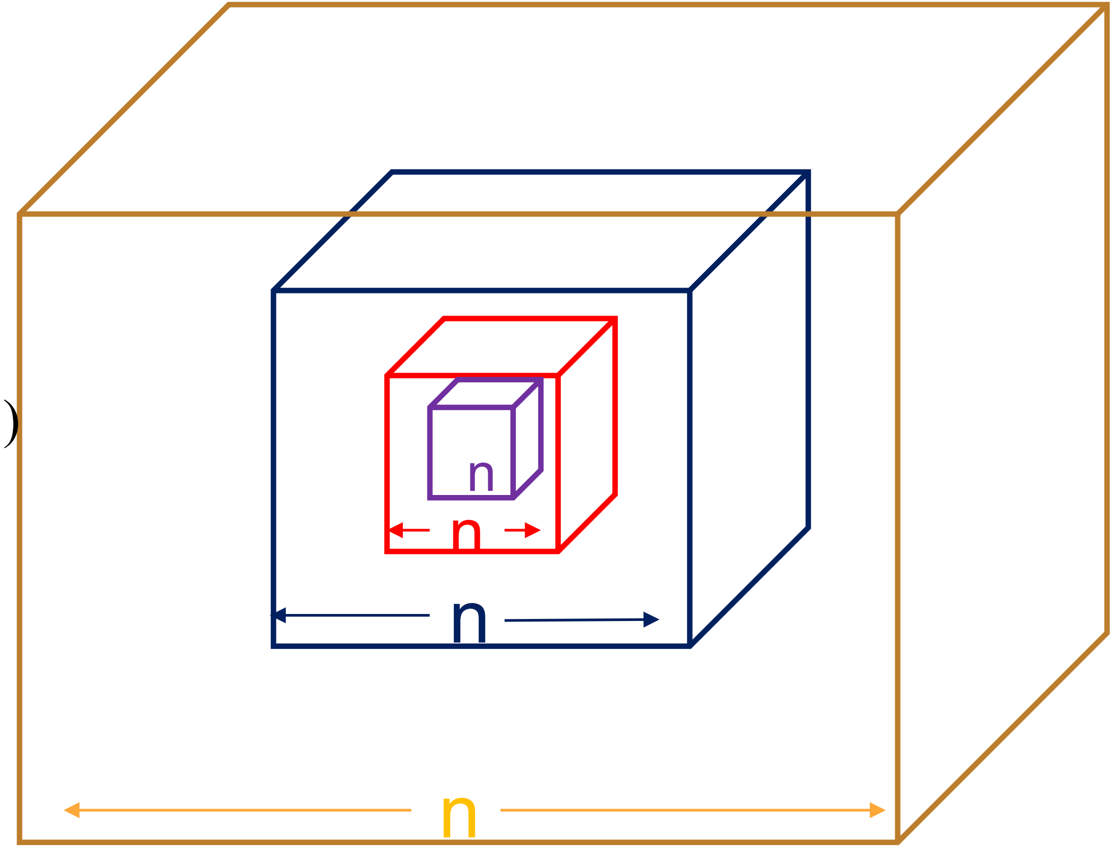
AMR communication volume of mesh

values from innermost box $p(n^3 - \frac{1}{8} n^3)$

Fine mesh communication is $(p^3-1)n^3$

AMR reduces communication volume

by a factor of $\frac{8}{7}(p^3-1)/p \approx p^2$

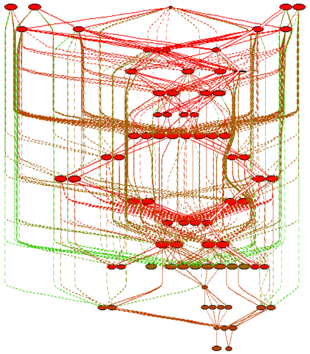


AMR
RMCRT

Each compute node traces rays on this AMR version of the whole mesh
But only “owns” the innermost mesh patch(es).

Task Graph Scaling Challenges at Large Scale

- Apparent deadlock at 32,000 CPU cores – difficult to debug, commercial debuggers
- RMCRT “RayTrace” task requests a “**global halo”** for ray marching – **new challenges**
- Uintah task-graph (TG) compilation algorithm overcompensating when constructing lists of neighboring patches for local halo exchange on fine mesh.
  - Load balancer considering **all patches on fine level** as potential neighbors
  - Cost of this operation grew when patches/node stayed constant



Complexity reduction:

n_1 = # coarse-level patches
n_2 = # fine-level patches
p = # processor cores

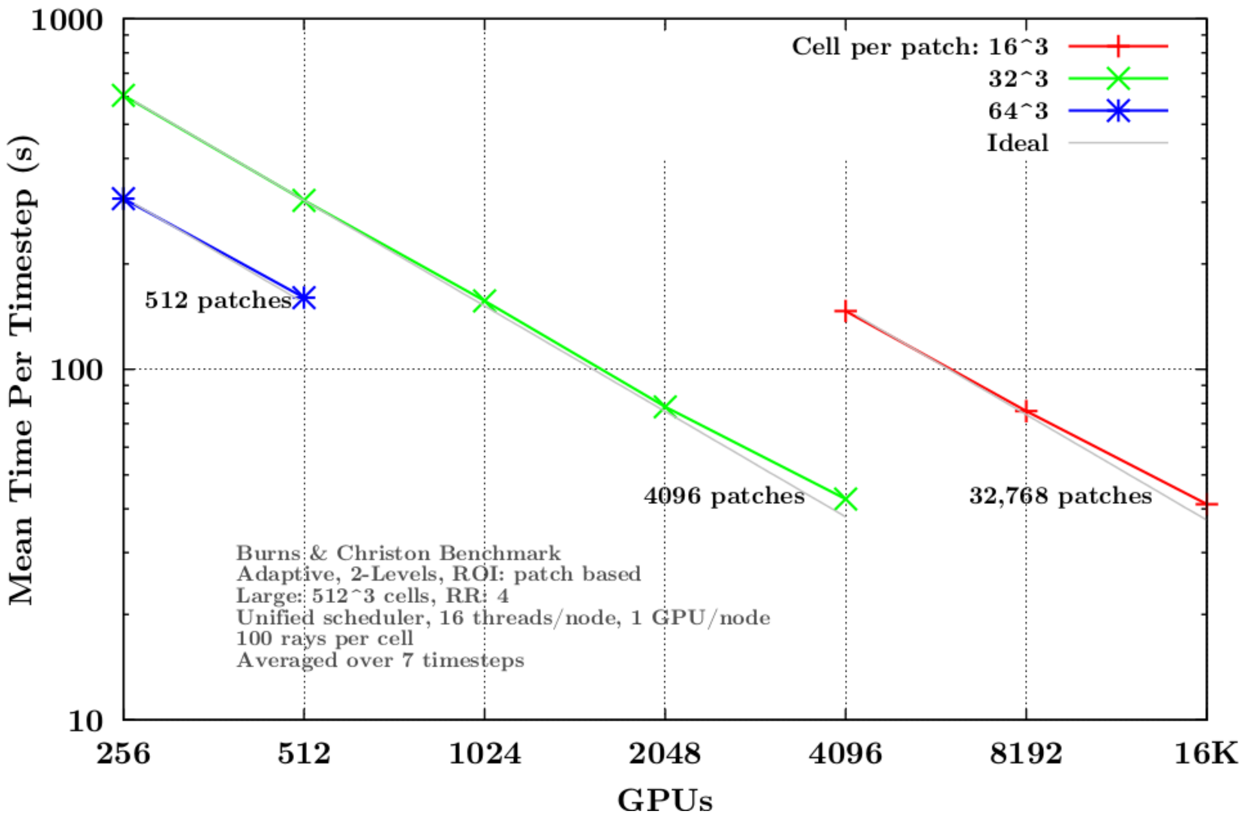
$$O(n_1 \cdot \log(n_1) + n_2 \cdot \log(n_2))$$

$$\Downarrow$$

$$O(n_1 \cdot \log(n_1)) + O(n_2 / p \cdot \log(n_2))$$

Reduced 4 hour TG compile times at 32k cores to under 1 minute, making initial large scaling results possible

GPU Strong Scaling on DOE Titan



4.2X Speedup over 256K CPU version at 16,384 GPUs

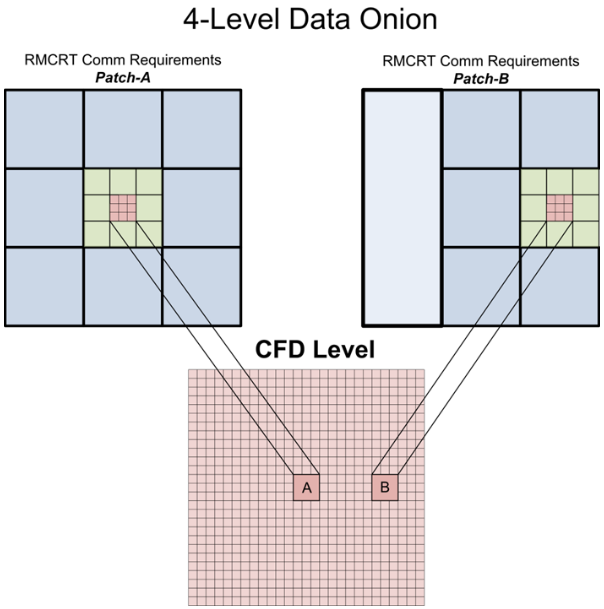
S. P. Burns and M.A Christon. Spatial domain-based parallelism in large-scale, participating-media, radiative transport applications. Numerical Heat Transfer, Part B, 31(4):401-421, 1997.

Challenge: RMCRT strong scales but does it weak scale?

- Fixed mesh domain size grows by 8 and then communications per node grows by a factor of 8 too. Computation per node locally grows by 8 too with a uniform mesh

- What about using the adaptive mesh paradigm?

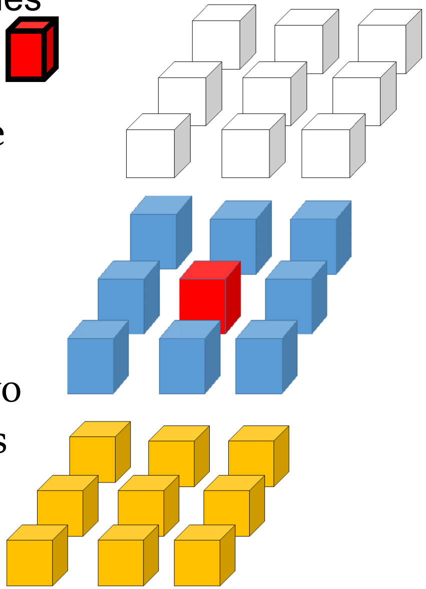
- When the mesh size increases use adaptive mesh coarsening for the new mesh.



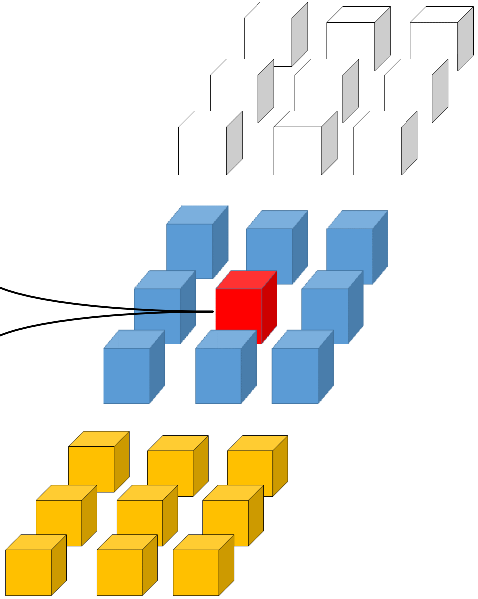
RMCRT Communications AMR Weak scaling

26 Level 1 nodes around node

Each compute node has to communicate with neighboring nodes one, two or more levels away



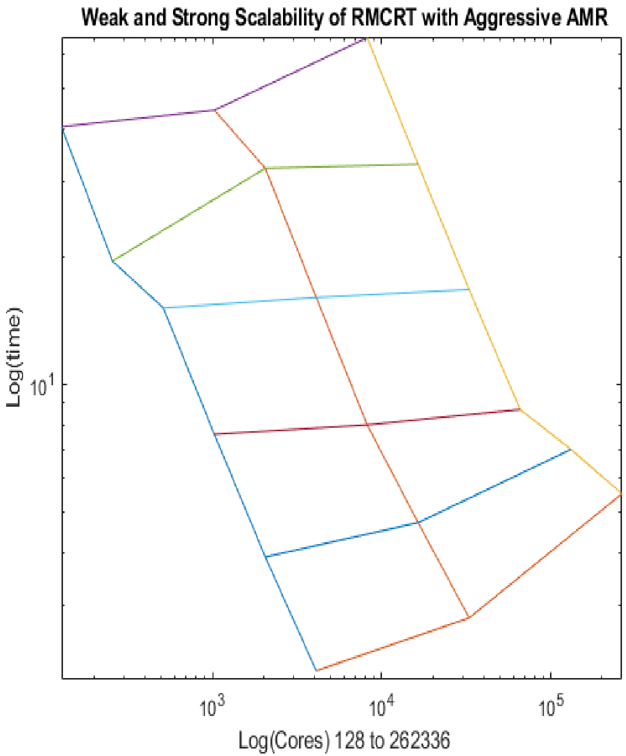
Aggressive coarsening The next level treats these 27 patches as the new fine node



When the full problem size increases by 26 at the coarse level and there are already 27 patches per node the workload only increases by (27+26)/26.

More generally if M coarse levels on a node then adding N more levels for weak scaling at most only multiplies the computational and communication s work by a factor of (N+M)/M the work- hence weak scaling with a factor of two if M = N **if aggressive coarsening is used**

RMCRT WEAK Scaling Results 100 Rays per cell



128^3	RR=2	256^3	RR=4	512^3	RR=8
CORES	TIME	CORES	TIME	CORES	TIME
128	40.5	1k	44.3	8k	65.7
256	20.0	2k	32.4	16K	33
512	15.0	4k	16.0	32k	16.7
1K	7.6	8k	7.94	64k	8.67
2K	3.9	16k	4.66	128K	6.98
4K	2.13	32K	2.85	256K	4.77

Roughly 2X growth in weak scaling as theory predicts

Performance Portability Using Kokkos

Performance Portability Using Kokkos and C ++11 Functors/Lambdas

- Kokkos:C++11 Library for implementing portable thread-parallel codes
- Application identifies parallelizable grains of ***computation*** and ***data***
- **Few changes** to enable Kokkos support via lambdas as they implement an unnamed functor class behind-the-scenes
- Many changes to enable Kokkos support via functors as developers manually implement the functor class.
- Kokkos ***maps*** those computations onto cores and that data onto memory Supported architectures Multicore CPU, Intel Xeon Phi and NVIDIA GPU,IBM Power AMD etc

Functors and Lambdas in C++11

Functor - function object that looks like a function but persists – need to instantiate – stored state.

Lambda* - "syntactic sugar " for writing a functor. Enables functor approach to be applied more quickly. Inline function.

* terminology goes back to the LISP notion of a function

Kokkos Abstractions Patterns Policies and Spaces

- **Parallel Pattern** user's computations (kernel)
  parallel_for, parallel_reduce, parallel_scan, task_graph, ...

- **Execution Policy** how the kernel should be executed static scheduling, dynamic scheduling, thread-teams, ...

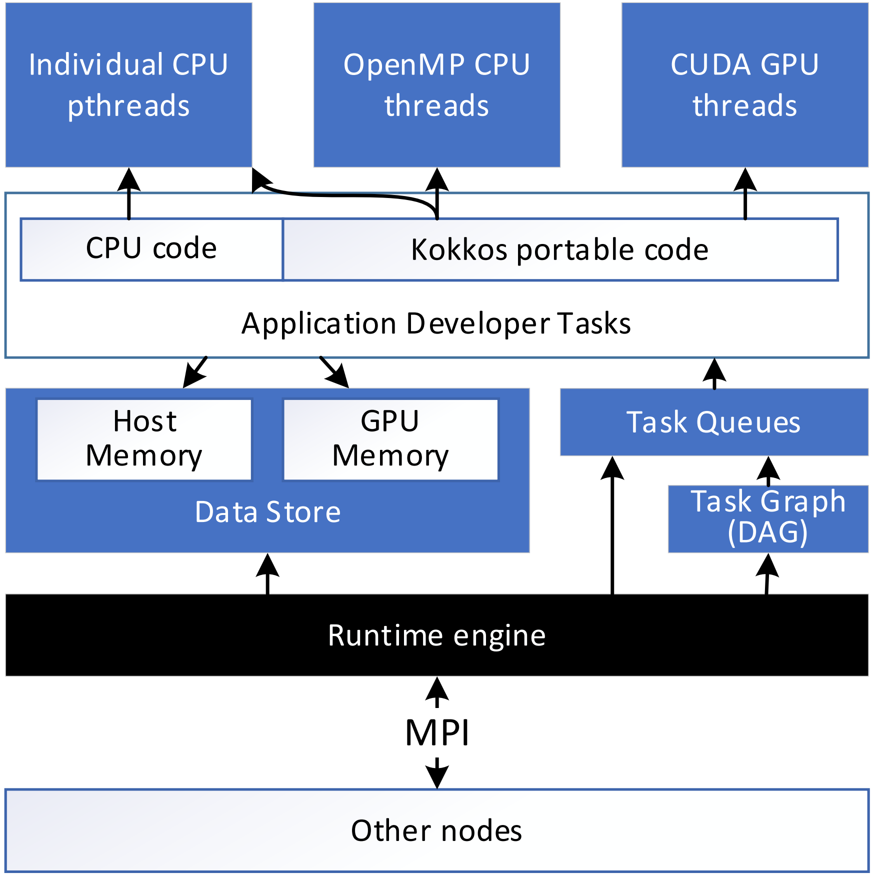
- **Execution Space** where the kernel will execute, Which cores, numa regions, GPUs, ...

- **Memory Space** where the data is allocated
  Host memory, GPU memory, High Bandwidth memory, ...

- **Layout** how the data is mapped to memory Row-major, Column-major, Tiled, ...

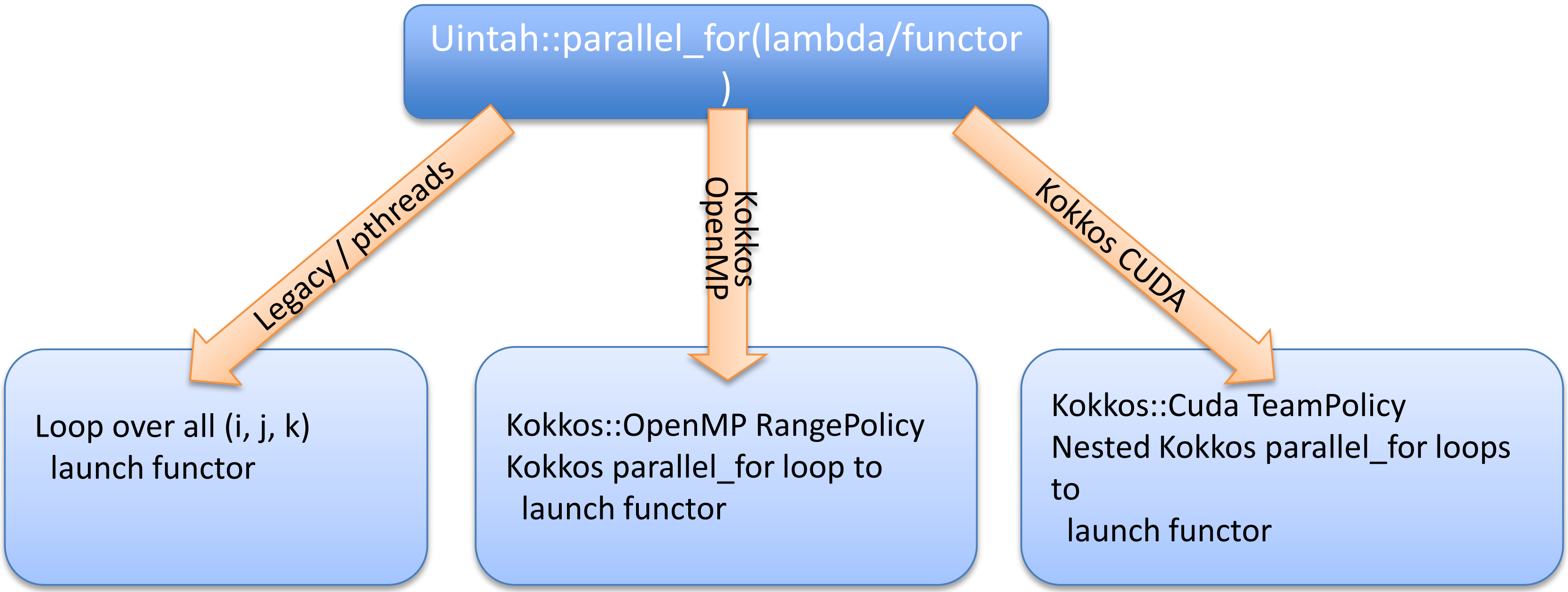
- **View** multiple dimensional array that is allocated in a *memory space* with the appropriate *layout*

Portable Uintah Tasks



- **Uintah tasks can run three ways.**
  - pthreads (for backwards compatibility of legacy tasks).
  - OpenMP CPU or Cuda GPU threads for Kokkos enabled tasks.
- Tasks portably access data store variables from host memory or GPU memory.
- Different tasks can execute in different portable modes. Can mix CPU and GPU tasks in the same build.

Under the Hood



```
// Legacy approach without Kokkos
for ( CellIterator iter = patch->getCellIterator(); !iter.done(); iter++ ) {
  IntVector c = *iter;

  double particle_absorption = abs_scat_coeff[abs_coef][c] * weightQuad[ix][c] *
                               portable_absorption_modifier;
  abskpQuad[ix][c] = ( vol_fraction[c] > 1e-16 ) ? particle_absorption : 0.0;
  abskp[0][c] += abskpQuad[ix][c];
}
```

blue is unchanged code

```
// Lambda –based Kokkos  approach
Uintah::BlockRange range( patch->getCellLowIndex(), patch->getCellHighIndex() );
Uintah::parallel_for( executionObject, range,  KOKKOS_LAMBDA(int i, int j, int k) {

  double particle_absorption = abs_scat_coeff[abs_coef](i,j,k) * weightQuad[ix](i,j,k) *
                               portable_absorption_modifier;
  abskpQuad[ix](i,j,k) = ( vol_fraction(i,j,k) > 1e-16 ) ? particle_absorption : 0.0;
  abskp[0](i,j,k) += abskpQuad[ix](i,j,k);
});
```

Functor version

Internal vars

Parameters passed

Set internal= external

blue is unchanged code.

Note Code bloat

```
//FUNCTOR-BASED APPROACH WITH KOKKOS SUPPORT
namespace {
struct eval_functor {

KokkosView3<double, Kokkos::HostSpace>       abs_scat_coeff;
KokkosView3<const double, Kokkos::HostSpace> weightQuad;
const double                                 portable_absorption_modifier;
KokkosView3<double, Kokkos::HostSpace>       abskpQuad;
KokkosView3<const double, Kokkos::HostSpace> vol_fraction;
KokkosView3<double, Kokkos::HostSpace>       abskp;

  eval_functor( KokkosView3<double, Kokkos::HostSpace>       & m_abs_scat_coeff
              , KokkosView3<const double, Kokkos::HostSpace> & m_weightQuad
              , const double                                 & m_portable_absorption_modifier
              , KokkosView3<double, Kokkos::HostSpace>       & m_abskpQuad
              , KokkosView3<const double, Kokkos::HostSpace> & m_vol_fraction
              , KokkosView3<double, Kokkos::HostSpace>       & m_abskp
              )
    : abs_scat_coeff               ( m_abs_scat_coeff )
    , weightQuad                   ( m_weightQuad )
    , portable_absorption_modifier ( m_portable_absorption_modifier )
    , abskpQuad                    ( m_abskpQuad )
    , vol_fraction                 ( m_vol_fraction )
    , abskp                        ( m_abskp )
  {}

  void operator() ( int i, int j, int k ) const {
    double particle_absorption = abs_scat_coeff(i,j,k) * weightQuad(i,j,k) *
                                 portable_absorption_modifier;
    abskpQuad(i,j,k) = ( vol_fraction(i,j,k) > 1e-16 ) ? particle_absorption : 0.0;
    abskp(i,j,k) += abskpQuad(i,j,k);
  }
};}

Uintah::BlockRange range( patch->getCellLowIndex(), patch->getCellHighIndex() );
eval_functor functor( abs_scat_coeff[abs_coef], weightQuad[ix], portable_absorption_modifier,
                      abskpQuad[ix], vol_fraction, abskp[0] );
Uintah::parallel_for( executionObject, range, functor );
```

Optimizing Serial ARCHES Char-Ox Loop

- **The most challenging of Arches 500 loops (1.6 flops per word.)**
- **Computational bottleneck with legacy C++ features 75% of runtime**
- ~350 lines of code with e.g. 60 Newton iterations and many calculations to determine reaction rates and compute char particle destruction rates:
- Loop (#Reactions + #Reactions * #Reactions) * #NewtonIterations * #Environments times *per cell:*
- Replaced use of std:vector with arrays of plain old data
- Removed memory allocations from loop
- Hard-coded calls to virtual functions and optimized math calls
- Setup DW variables as unmanaged Kokkos views (**Uintah::KokkosView3)** for Kokkos-based Uintah builds

2.66x speedup of serial code

Simple Radiative Properties Loop

for all mesh patches **do**
  for all cells in a mesh patch **do**
    apply a weight to a particle's absorption coefficient
    store the weighted coefficient for flow cells
    store a zero for non-flow cells
  end for
end for

Weighted properties are then used to compute global radiative heat flux

Up to 4.93x serial performance improvement on CPU by:

i. Replacing legacy loop statement with Uintah::parallel_for

ii. Replacing legacy data structures with Uintah::KokkosView3

Results: Adding Loop-Level Parallelism vs Xeon Core

CPU: Two Intel Xeon E5-2680 Sandy Bridge processors 2.7 GHz; 16 cores; 2 threads per core, 64 GB, **GPU Maxwell** 12 GB, **KNL**: 1.3 GHz; 64 cores; 4 threads/core 96 GB

- **Complex CharOX Loop:**
- 16^3 32^3 64^3   patches
  - **14x, 15x 15x speedup CPU** (Kokkos::OpenMP 16 cores)
  - **50x 68x 67x** speedup **GPU** (Kokkos::Cuda 24 blocks 256 threads each)
  - **46x 65x 76x speedup KNL** (Kokkos::OpenMP 64 cores 64 threads)

- **Simple Radiation Props Loop (not enough work):**
  - Up to **12.83x** performance improvement on **CPU** (Kokkos::OpenMP)
  - Up to **6.04x** performance improvement on **KNL** (Kokkos::OpenMP)

Results: Using More Threads per Core

Complex CharOX Loop:

i. Up to **1.11x** performance improvement on CPU (2 threads per core)

ii. Up to **1.47x** performance improvement on KNL (4 threads per core)

Simple Radiation Props Loop:

i. Up to **1.19x** performance improvement on CPU (2 threads per core)

ii. Up to **1.45x** performance improvement on KNL (4 threads per core)

Up to 2X slowdowns when not enough per-core work (16^3 cells per patch)

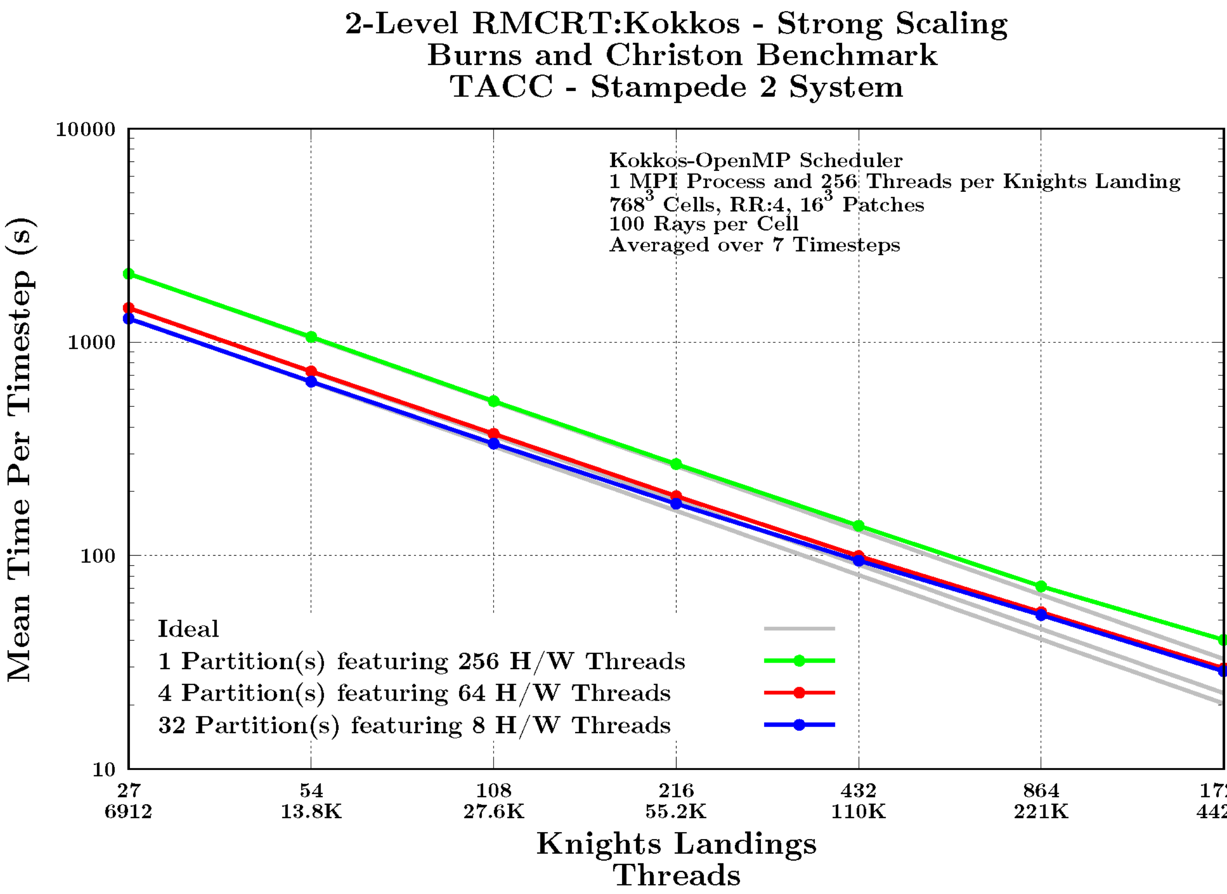
RMCRT

Kokkos delivers amost 1.7x over original cuda /cpu/ MIC code.

Low peak as 0.7 Flops / DP word

Good strong scaling to 1728 KNLs

RMCRT speedups lower on CPU/GPU/KNL 1.2-2.9x



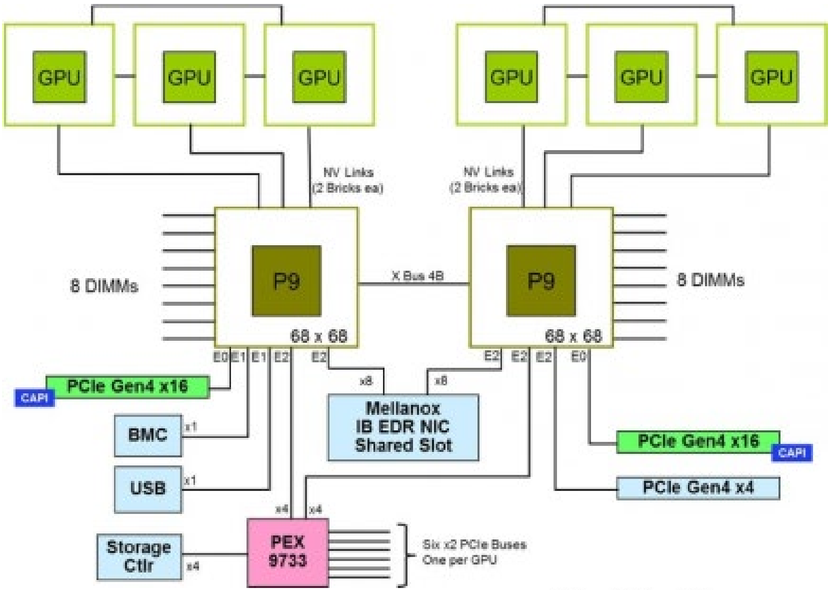
Future Work

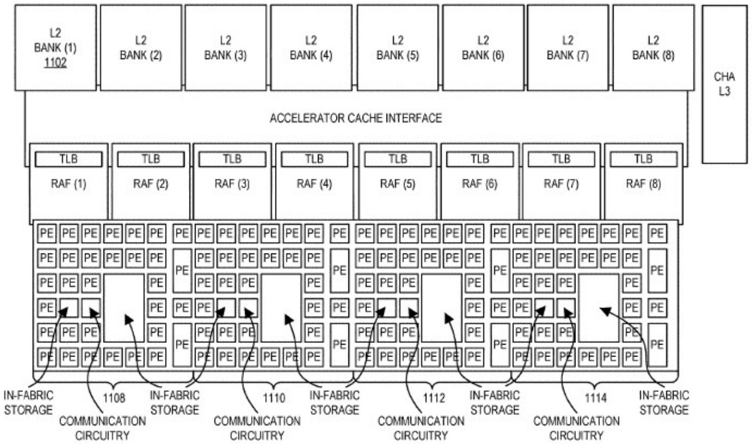
Finish Arches Kokkos Port and SIMD Kokkos

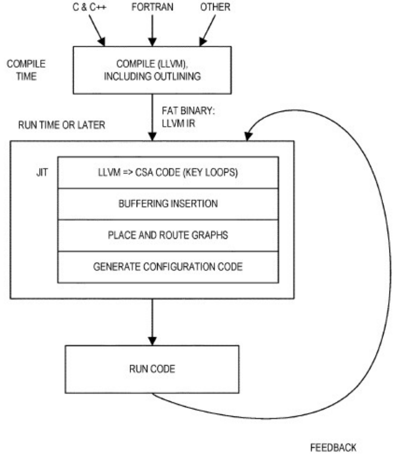
Move Uintah Arches to Lassen multiple GPU Machine .

Experiment with Sandia ARM machine

Start working towards A21 Dataflow Machine (see NextPlatform.com







Summary

Past and present investments in

I. People
II. good code and algorithm design of
III. a programming model and an
IV. adaptive asynchronous communication-hiding runtime system
V. with a portability layer

Make it possible to:

(i) independently develop complex physics code which is then unchanged
(ii) while scaling complex engineering calculations and
(iii) Using results to drive engineering design
(iv) Provide a viable path to exascale