Utah SCI/CCMSC Programming Models Deep Dive Discussion Notes
University of Utah, Salt Lake City, UT
July 28-29, 2014

SUMMARY NARRATIVE

On July 28-29, 2014 the University of Utah Scientific Computing and Imaging Institute (SCI), which includes the Carbon Capture Multidisciplinary Simulation Center (an NNSA ASC PSAAP II Center) hosted a programming models deep dive discussion with personnel from Los Alamos, Lawrence Livermore and Sandia National Laboratories.

University of Utah participants discussed the capabilities and current status of Uintah, a set of software components and libraries that facilitates the solution of partial differential equations on structured adaptive mesh refinement grids using hundreds to thousands of processors. Uintah is the product of a ten year partnership with the Department of Energy's ASC program through the University of Utah's Center for the Simulation of Accidental Fires and Explosions.

Uintah controls and allocates resources and provides tools to get to massive scale (e.g. Arches, ICE, and MPMICE). As part of the deep dive, University of Utah faculty and staff described current work on two layers of abstraction that would go between the app codes and Uintah-X: SpatialOps (Nebo EDSL for fine-grained data level parallelism) and ExprLib (a DAG-based approach for on-node representation of a task).

Representatives from Los Alamos, Lawrence Livermore and Sandia National Laboratories described current work at the laboratories on programming models and libraries. After background presentations were completed, University of Utah faculty and staff led the laboratory participants through hands-on tutorials using Uintah, Nebo and SpatialOps on selected problems. Lab staff queried University of Utah personnel about current and potential capabilities of the SCI software and identified numerous opportunities for future collaboration. Laboratory personnel expressed particular interest in the potential for extending Uintah to work on unstructured meshes.

FOLLOW UP ITEMS

- Presenters from the University of Utah and the Defense Program labs agreed to make their presentation materials available.
- Utah and lab personnel agreed to pursue collaboration activities focused on Uintah, Nebo and SpatialOps.

PARTICIPANTS

University of Utah: *Martin Berzins, James Sutherland, Alan Humphrey, Todd Harman, Justin Luitjens (NVIDIA), Matt Might, Tony Saad, John Schmidt, Jeremy Thornock, Davison de St Germain*

Sandia: *Eric Phipps, Carter Edwards, Dan Sunderland, Stephen Olivier, Greg Sjaardema, Janine Bennett, Keita Teranishi, Robert Clay, Ted Blacker (AST)*

LANL: *Matt Bement, David Daniel, Curt Canada, Eric Nelson, Joshua Payne, Ben Bergen, Geoffrey Womeldorff*

LLNL: *Rich Hornung, Jeff Keasler, Ian Karlin, Adam Kunen, Bert Still*

NNSA/Leidos: *Tina Macaluso (Scribe)*

DISCUSSION HIGHLIGHTS

Monday July 28, 2014

Greg Jones, Welcome to SCI Overview and Introduction of Participants.

- Welcome to the University of Utah. You have a full day ahead of you. I came to SCI in 2000 when we had ~35 people (and the Institute has close to ~200 people now). Back then we were just getting momentum going on Uintah (we were working on C-SAFE, which was one of the original ASCI Alliance Centers). I believe that some of the best science SCI has done is reflected in the growth in Uintah. It is good to have you here and to see Uintah continue to move forward.

BEGIN BACKGROUND PRESENTATIONS

Martin Berzins, Utah Petascale and Beyond Applications

- The meeting agenda we sent out is only an approximation, and can be modified as needed. It is important that we get these discussions going. First, I will provide a background talk on the models we are using and program drivers for the software work in order to provide context for later discussions. At SCI we use applications to motivate the scalability and algorithmic work we do. Our PSAAP II Center (Uncertainty Quantification-Predictive Multidisciplinary Simulation Center for High Efficiency Electric Power Generation with Carbon Capture or CCMSC) is part of that but it is not the only part. Funding for work at SCI goes back to the original ASCI Alliance Centers, along with NSF, Army, the DOE INCITE Program and the DOE ALCC program.
- Predictive computational (materials) science is changing. Science is based on subjective probability in which predictions must account for uncertainties in parameters, models and experimental data. We cannot deliver predictive materials by design over the next decade without quantifying the uncertainty associated with them.
- The NSF funded our modeling of the Spanish Fork Accident (2005). We have been modeling the scenario with individual boosters in a fire when a speeding truck with 8000 explosive boosters (each with 2-2.5 lbs of explosives) overturned and caught fire. Does this simulation provide experimental evidence for a transition from deflagration to detonation? This is a potential exascale problem.

Still – What is the size of cores you need?

- A fat core (not a MIC core). These results (shown on slide) for the simulation are running on Mira now. We hope to solve this problem in the next few months.
- The CCMSC leadership team includes PI Phil Smith, University of Utah President Dave Pershing and me. We want to take UintahX beyond petascale with our overarching application, which is pulverized coal power generation. You will hear about the UintahX runtime system and the Wasatch Nebo DSL over the next two days.
- An Alstom clean coal boiler is about 60 meters high and is more efficient than the boilers we have had in the past. The plan is to pipe out the CO_2 and put it under the North Sea. Resolution needs to get down to 1 mm per side for each computational volume = $9x10^{12}$ cells (and this is1000x larger than the largest problems we solve today). The next slide shows the specific goals for the high efficiency advanced ultra-supercritical (AUSC) oxy-coal tangentially-fired power boiler. Recent news notes a General Electric takeover of Alstom in progress that injects more uncertainty into the building of such a facility.

Still – This would be larger than any boiler that has ever been built. It has to work the day it is put in place, and it has to work for 50 years – this sounds like things that we focus on.

- The next slide shows a schematic of the plant. There is need to worry about temperatures. We have to get resolution down to 1 mm to capture wave numbers that are important. Jeremy will discuss the existing code this afternoon; radiation is very important and there are both high order

and low order methods with LES model and particulate models that are used. The radiation is challenging and scales to 30K. We have to solve large linear systems.

Nelson – You need 1 mm resolution everywhere?

- There is ongoing debate about that now. I would say that AMR may be needed (but Phil and some of his colleagues are arguing it may not be needed). There is a question as to whether (or not) we could vary the resolution (likely to a factor of 10x, which could make a big difference at this scale).
- The other piece of this challenge is our Multiscale Modeling of Electronic Materials Consortium, which involves eight universities to look at batteries, fuel cells, LEDs & night vision. We want to do continuum scaling down to coarse-grained molecular dynamics (MD), so we can look at new materials in environments where we know current models break. We do not yet know how to do this. It is a big project but there is limited funding available for it. The next slide illustrates an ongoing project involving a Lithium ion battery (this is a huge grand challenge problem) – we are using the same code for all of these projects.
- So, at SCI we have not one but three potential exascale problems and these are what drive us.

LLNL – For the multilevel simulation, is there a resource requirement for each of the simulations (e.g. numbers of cores)?

- We are not at the point where we can pin those details down yet. We know we will be forced to run on the largest machine we can access. At some point there has to be a continuum model and then an MD model (and you go into the MD model, come back with the result and continue). We know how to do that in software but we do not know in reality how it is going to work. We do not presently know how to do that multi-scale modeling and how the uncertainty in each model relates to the uncertainty in the continuum model.

Martin Berzins, Overview of the Uintah Software

- Now I want to discuss the background and motivation for Directed Acyclic Graph (DAG) software. We know many things are going to break as today's bulk synchronous (BSP) processing, distributed memory, and execution models are approaching an efficiency, scalability and power wall. I am going to discuss an approach that we believe has a lot of promise. It has worked well for us and continues to have much potential. I will show you both the strengths and weaknesses of our approach (it is important that we not oversell it).
- Vivek Sarkar's 1989 thesis describes graphical representation for parallel programs. Charm++ came along in the early 1990s and is the best example of this approach today. In 1998 Steve Parker had a different take on the concept when he developed his initial design for Uintah. There is a great deal of work on task graphs in the CS community (but not much of it is directly useful to what we are trying to do). Today, we have the Uintah DAG approach, Jack Dongarra supports a DAG-based parallel linear algebra software approach (Plasma), there is a Star PU task graph runtime system, and Sanjay Kale uses object-based virtualization in Charm++.
- First a little theory – the key idea here is *parallel slackness* – at some point you have more tasks than processors, and this makes your calculations scale. It is not directly useful in its original form but the BSPRAM model extends this notion to shared memory and defines slackness. I think it is unrealistic because it is not clear this is virtually realizable – but the notion is that you can get an optimized simulation if you have enough slackness. This is what Cray did in the past for one platform model line (XMT) using many threads, randomized memory, and parallel slackness via fast switching between threads.
- The model on the slide is a crude realization of the theory, and what we have now looks similar. We have a data warehouse on each multicore node with a task graph and cores running tasks and checking queues to get information. We have parallel slackness through multiplicity of tasks and out-of-order execution. We overlap communication with computation by executing tasks as they

become available to avoid waiting, and we load-balance complex workloads by having a sufficiently rich mix of tasks per multicore node such that load balancing is done per node.

Nelson – Slackness is a measure of how much concurrency you have in the calculation?

- It is a measure of extra currency or over-commitment.

LANL – What determines the granularity of the tasks?

- A good question – there has to be a tradeoff between having too many small tasks (and much overhead) and too few large tasks (because that means waiting). Some components have not changed as we have gone from 600 to 600K cores. You can keep the app code essentially the same as you grow the size of the system (with some caveats), so the app code and the runtime system code can grow independently. Key is expressing the app in the abstract, task-graph approach to compile individual tasks onto CPUs/GPUs and execute on the runtime system in an asynchronous out-of-order execution approach. We have work on scalable I/O and VisIt as well (and it is not clear how this will work out in the future). That is the basic architecture.
- Turning now to solvers (Uintah Patch, variables and AMR) – ICE is a cell-centered finite volume method for Navier-Stokes equations. We have a fluids code, a solid code, a combustion code and now an MD code (due to our ongoing work with the Army Research Lab).

Blacker – Do you think unstructured grids represent a significant change?

- No, I have worked on unstructured grids and load balancing. The main difference is that it is more complicated to define what halos are and to partition them (but we have good partitioning software). The biggest challenge is the fact that things in our infrastructure implicitly assume halos of a certain kind (but if we really wanted to change it then we could). I do not see a major challenge resulting from the use of unstructured grids. Dav St. Germain will discuss the Runtime system in greater detail this afternoon – I am showing you the overall graphic on the slide now only to provide an overall perspective.
- Each task has its computation, defines its inputs and the outputs it will produce, and this information is used to produce a local task graph. There is no MPI in the tasks themselves (just a specification as to what is needed from – and goes back to – the data warehouse). Our task graphs are relatively simple. We can get this to scale because we have multiple task graphs.

Question – Can the tasks be parallel internally?

- Yes – what you do with the loop is an option. We execute one task on one core so we do not have an issue with resource contention.

Humphrey – A single task will run on a single core.

Utah – We have a dialogue taking place to define how the framework could identify resources available, so there is not an over-subscription problem. This dialogue is just beginning.

- You can solve this problem bottom-up (i.e. how things might work on a node) and also top-down (i.e. running things at large scale). My contention is that you cannot design at the node level without understanding what is happening at the global level. (This is a debate I have with people who focus only on the node level and assure me their codes will scale.)
- We have a task graph structure on a multicore node with multiple patches. This is not a single graph; it is closer to multiple structured connected workpools, one per multicore node. Multiscale and multiphysics add flavor because there are more tasks (provided you can handle the logistics). This abstraction is useful in figuring out how to run on many cores and keep everything busy – the more tasks you have, the better.

Nelson – Hundreds or thousands of patches per node?

- No, sometimes there are as few as a couple of patches per core. The DAG approach is not a silver bullet. There have been three phases in Uintah development, the first was 1998-2005 with overlap of communications and computation and static task graph execution. AMR was added in 2005-2010; beginning in 2010 a more hybrid model with a threaded model on the node (one MPI process and one data warehouse per node) was used. Unless you put a lot of work into this, it is not going to help you (discussed graphic of previous CSAFE results using graph-based approach).

What is needed is effort to make sure the task-based approach really works; it is all about parallel slackness. The next slide shows an explosives problem with a fluid-structure benchmark using AMR MPMICE. The thread/MPI scheduler is de-centralized; threads pull tasks from task queues to a single data warehouse.

Karlin – How do you handle NUMA on a multicore node with one MPI task? If you have a multi-socket node, do you run one MPI task across the whole node?

- Yes – and how we handle memory spaces is a function of the specific application code. There is a limit as to what we can do with respect to working with the hierarchy. We want to solve problems at large scale and most of the work I am presenting here has been done by graduate students. We have not yet looked at making the execution of the tasks optimal.

Comment – I do not see that as a weakness. I like having the two functions separate. One reason the bulk synchronous model worked is because there was separation of inter-node runtime versus node level runtime.

Karlin – Can you treat each socket as a node?

Utah – We use one MPI process per socket and use MPI to pin that process to each socket.

- There are multiple things we can do. For task prioritization the different algorithms used to execute tasks determines execution behavior: randomization, first-come/first-serve, PatchOrder and Most Messages. Charm++ does this, as well.
- Turning now to granularity – on the slide we are looking at results for patch size and total execution size on Kraken (24K cores). As you increase patch size, the wait time goes up; overall there is a sweet spot of 12^3 to 15^3 in terms of execution time (on an older machine). We would need to redo experiments on newer machines. The bottom line is that you have to balance patch size with granularity. These are challenges we face presently (i.e. assessing current sweet spots).
- Everything looked good on the Jaguar runs, until ORNL upgraded the nodes and networks, after that we experienced a scaling breakdown at 256K cores. We saw a definite breakdown on Jaguar XK7 with more, faster cores and a faster network because there was contention getting to the data warehouse, so the data warehouse had to be rewritten to make sure all of the cores could access it quickly (and doing that took a few months).
- Scalability is at least partially achieved by not executing tasks in order, e.g. AMR fluid-structure interaction (discussed graphic results for MPMICE runs on Titan, Stampede and Mira). We see some subtle differences; execution order is not the same on every machine. Does that raise issues as to whether we get the same results? Potentially, it could.

Question – Do you see the same situation from run to run?

- If you are on the machine with the same network and nothing else is running, then you expect to see the same deterministic outcome. If other people are on the machine then you could see a different execution pattern. That means we would likely see differences on Titan versus Mira. We have not monitored this.
- If we look at a breakdown of weak scaling for AMR+MPM ICE, we see more differences for Mira, Titan and Stampede with much external communication and not much parallel slackness. We have not tuned the data seen in the histogram on the slide. Wait time is spent polling, packing/unpacking and dealing with the data warehouses.
- Turning now to scalability for the Spanish Fork detonation problem where we go out to ~512K cores. This required work on the AMR algorithm. At every stage, when we move to the next generation of problems, some of the algorithms and data structures need to be replaced (but then the labs know this well). The next slide shows MPM AMR ICE strong scaling on Mira and Blue Waters (where the scaling is better). In summary let me say again that I do not believe it is possible to develop (in isolation) on a single node and get scalability at large node counts. In designing for exascale we see a clear trend in accelerators, e.g. GPU but also Intel MIC. We will have to turn to our Nvidia colleagues to understand scaling better in these environments. The

layered DAG abstraction is important for scaling and for not needing to exchange app codes. We like the work you are doing with Kokkos (a layered collection of libraries).

Nelson – Is the scalability of the radiation solver largely due to radiation solve?

- No, this is ray tracing across many GPUs, and we have not really tried to optimize it yet.

James Sutherland, Node-Level Abstractions for Platform-Agnostic Computing

- Uintah controls and allocates resources and provides tools to get to massive scale (e.g. ARCHES, ICE, and MPMICE). We have been working on two layers of abstraction that would go between the app codes and Uintah-X: SpatialOps (Nebo EDSL for fine-grained data level parallelism) and ExprLib (a DAG-based approach for on-node representation of a task).
- Arches 2.0 is a re-factor to leverage what is in SpatialOps on traditional (CPU) and other architectures. Wasatch is built on both ExprLib and SpatialOps (as are ODT and LBMS).
- Thinking in terms of graphs – we offer a different take. First register all expressions (each "expression" calculates one or more field quantities); each expression advertises all direct dependencies. Next set a "root" expression (all dependencies are discovered/resolved automatically) and construct a graph. This results in highly localized influence of changes in models. Programmers need not worry about the composition of the whole problem (which avoids the issue of "logic creep" because coupling is not exposed to the application programmers). Once you have the graph, you can marry it to a mesh, schedule an evaluation, deduce storage requirements and allocate memory. (Showed and discussed sample coal combustion problem with 55 PDEs and ~35 ODEs/particle along with complex inter-phase coupling). This example sits outside of Uintah (if inside Uintah it would be chopped into smaller pieces to expose finer granularity). We can dial in the granularity of decomposed tasks we expose to Uintah.

Karlin – What is the process to generate the task graph?

- It is a recursive process, based on the input file, until we hit terminal nodes and that generates the graph (based on "what I need to calculate"); it is auto-generated.

LANL – All of the green dots on the graph essentially represent a kernel, so you assess inputs and outputs?

- Yes and you will see examples this afternoon of how to determine what you compute, what you require, how to grab memory and execution.

Comment – How much slackness?

- In this example, if I decompose this so each node is a Uintah task, there is (probably) 50-way task level parallelism. You can analyze the graph and assign priorities to pathways on the graph. It is not just task-parallel that you need to worry with but also execution time on the node because the critical path can be prioritized by depth in the graph (e.g. number of children nodes) or time (to get to hot spots in the graph first), and then you backfill everywhere else. Ultimately, it is the programmer's responsibility to get an appropriate level of granularity. Just adding more granularity – at some point – works against you. Defining optimal level of granularity is a hard thing to do. Optimality can be thought of in different terms: time to execution, power usage, minimizing memory use, minimizing runtime – whatever you want to achieve (and the resulting graph can look very different). We need flexibility to hit targets we have not yet considered.

Bergen – Do you have a mechanism for defining different metrics?

- Once you pick a metric you have to be able to measure it and then you want to optimize it.

Berzins – This work is still in an early stage. We have worked with task graphs for some time and it is taking us awhile to figure out exactly what we can do with the approach. We don't know now how this will work on machines we expect to see in 2017, as an example.

- We are seeing these now with high CPU/GPU architectures and there is still learning that has to happen. Why an EDSL? We want programmers to focus on what is needed to express intent. EDSL brings together model, discretization and architecture. We want the code to perform at least as well as what we can write and hand-tune by hand. We want it to be extensible so it can

insulate the programmer from architecture changes. And we want it to "play well with others". We have embedded it in C++, so C++ app codes can interoperate with it well and there can be a level of incremental adoption. (Discussed comparison of field expressions in C++ and Nebo.) We get compile time guarantee of field compatibility (and you will learn more details tomorrow). We use stencil-based operations to achieve better performance and scalability than without chaining (i.e. nested loops).

- The next slide shows the developer pool from January 2013 to present (and the number of commits is not large). How well do we perform? The Wasatch software stack is 3-4x faster when using the EDSL in Uintah on the mini boiler problem (pure LES without scalar transport or other complicating factors). On some other problems we have seen 5-10x speedup. This is scalable to 260K processes/cores; these are on-node, single-core comparisons (we use each core on the node).
- We have been working to get Wasatch to run on Titan so we can handle hybrid CPU-GPU architectures. If you look at the graphs on the slide, the brown nodes are GPU-ready and the gray are locked CPU (and will ultimately become GPU-ready). Right now we are in a "hybrid mixed" node and this pictorial represents that.
- For the hands-on tutorial this afternoon, I ask that you bookmark this URL: goo.gl/s56Uaz and walk through code examples on your laptops as we describe them.

Ben Bergen, LANL Programming Models for Modern Architectures

- I am in LANL Applied Computer Science (CCS-7), where we work on weapons physics and open science. Our high level goal is to address challenges in multi-physics application development. We want to allow rapid prototyping and development of new physical models and methods, enable cross-physics code optimizations, empower interdisciplinary teams, support flexible and modern data-structure design and allow portability and performance.
- Our current trend is to divide and conquer. We want to separate the kernel and execution logic to avoid unintended (and unnecessary) serialization and expose parallelism of underlying model/method. We also want to separate models and algorithms from implementation via separate roles, so computational physics and math experts can work in an area of their comfort zone and the computational scientists and CS experts can do the same (but this is not what our development environment looks like now).
- Slide #4 shows a current (imperative) coding style (with thanks to Charles Ferenbaugh) – this assumes everything is stored in an array and there are no opportunities to swap out data. This is difficult to parallelize. We are turning to a functional model that will decompose computationally into kernel logic and execution topology. The kernel is applied to an index space (with no implicit ordering). Charles Ferenbaugh describes this as "kernel and driver" and there are advantages to functional representation such as increased flexibility for data layout, exposing task and data parallelism, and improved resilience and transactional semantics.
- A counter example (with thanks to Eric Nelson) would be computing cell centers where common sub-expression elimination introduces loop-carried dependency. This is more portable and may lend itself to optimization without changing the code.
- Slide #9 offers my perception of a proposed development hierarchy (from single thread to single node to full system). I identify where community standards exist and note which steps require internal support. We have been looking at Legion at LANL - but it sounds like we should look also at Uintah because we are interested in a task graph model runtime as a coarse-grained operation. Then we need a node level runtime (i.e. assume single address space and data parallel operations for a particular node architecture). At the bottom two levels we want to take things from the community. For the top four levels of the development hierarchy (compiler support tools, compiler, simulation control and applied methods logic) we believe LANL work will be needed. For the program driver level there is an interesting approach in the University of Chicago FLASH code

with separate, discrete physics packages that can be called out by the program driver. For the compiler support tools layer we would leverage community tools (e.g. LLVM, SPIR, C++11). We would see the physics and applied math guys working at the domain-specific kernels and program driver levels, and the computer and computational science experts working at the compilation tool, runtime library, node-level runtime and system-level runtime levels. Presently at LANL the computational and CS experts do not provide day-in/day-out contributions to the production codes and we need to change that.

- Domain-specific kernel language would provide abstraction for implementing applied methods logic. If we are limited to C++11 semantics then we have to expose some things we might not want to expose - but if we have an embedded DSL then we can map it in C++.

Sandia – Are you thinking about allowing specializations/code to figure out what works well?

- Yes, I am not saying we would only have one kernel implementation for a given algorithm. There is implied database lookup in the code seen on Slide #10. Part of the advantage of the domain-specific kernel language is that details about how things are implemented are open. Pat McCormick is working on the Scout DSL project and Tim Kelly is looking at data-future architectures to see how data value can be made available to algorithms in a good way. We would begin by looking at how we articulate the methods in a way that can be compiled.
- The main purpose of the program driver is to allow different compositions of kernel logic; it would be based on a standard language. The compilation tool produces compiled executables for target architectures. If we can inject these layers then teams of CS can work in support. The runtime library would include mesh and data layout with backend support for different node-level and system-level runtimes, debugging support for kernel language (and the Scout project has made progress) and utilities for kernel language compiler.
- We are looking at Legion, which sounds like it is similar to Uintah. Legion was developed by Alan Aiken at Stanford and it implements a task-graph framework for specifying data-centric qualities. Programmers use these abstractions to organize data, so the Legion runtime can extract task and data-parallelism during execution. Currently at LANL we are working on a conjugate gradient benchmark, along with one for Lagrangian hydrodynamics.
- There are challenges, including standardization of runtimes and node-level runtime models. Barring a standard, we need to ensure interoperability of key components. I think C++11 is a nice development for node-level runtime models and work is being done at Sandia (Kokkos) and LLNL (Raja). SYCL is a C++ specification for OpenCL and CUDA is not portable. We also want portability and performance, along with composability of kernel logic and task logic (and Legion offers an elegant solution to this last challenge).

Sutherland – You are willing to explore a higher language to distill this higher level characterization into an existing language?

- We would introduce data access and involve decisions about data layout – it is limited to the kernel and could only be part of what we are doing (so it would still leverage normal C++ but not necessarily all of the compiler). Right now there are things people are not willing to touch because they are afraid they will break them. I would see this as a reallocation of efforts, and people presently underrepresented on the production code teams would be the ones responsible for maintaining these new features.

Carter Edwards, Sandia, Qthreads Multithreading Library

- This is a library for multithreading, primarily at the node level. The goal is to get the programmer thinking about the way the problem itself decomposes. With Qthreads, the programmer exposes application parallelism as massive numbers of lightweight threads (qthreads). It is problem-centric rather than processor-centric. A dynamic runtime system manages the scheduling of tasks for locality and performance. Locality-aware load balancing of tasks is used to support NUMA and complex cache hierarchies (using a hybrid of shared queuing and work stealing methods). We

have lightweight task context switching and interfaces for higher-level languages/libraries. It provides the only scalable tasking layer for Chapel PGAS language and serves as an interface for OpenMP (with the ROSE compiler at the front end).

- Qthreads has been ported to x86, Phi, PPC, Sparc and Tilera. Qthreads is an option for the lower level layer in today's frameworks and some of its capabilities such as locality awareness are being incorporated into OpenMP.
- Slide #4 shows a diagram of the Qthreads runtime view of locality. It is available online at https://code.google.com/p/qthreads).

Steven Olivier, Sandia, Kokkos

- Kokkos is a layered collection of libraries with core capabilities (on which the layering takes place) and backends such as OpenMP and pthreads. The portability challenge is the requirement for device-specific access patterns needed to get performance. The right question asks what the abstractions should be for performance portability (not whether you should write for GPU versus CPU, which has been the question asked in the past).
- We dispatch computations to an execution space and operate on data in memory spaces. We use multidimensional arrays (with a twist) – we use a C++ abstraction from which you can choose the layout at compile time (and the layout will change depending on the device specifics). The user code need not be changed. You can dispatch computation to one or more execution spaces.
- We evaluate performance from the very beginning (an example of right versus wrong layout for the computational kernel is seen on the slide). We see a large performance loss with the wrong array layout. The user code never changes.
- With respect to overhead, we find Kokkos to be competitive with native programming models (showed MiniFE CG-Solve time for 200 iterations). How do we handle finite element computations with scatter-add from one data structure to another data structure? We can simplify algorithms by using atomic-add units in the hardware and see better performance if the hardware has good atomic units. Now we have a pattern we can use (atomics).
- We are looking at how to manipulate dynamic data structures. We use a thread-scalable algorithm for constructing a data structure and find that the pattern and tools are generally applicable to construction and dynamic modification of data structures. We have had co-design success with CUDA Unified Virtual Memory (UVM) when they gave us early access and we provided them with early feedback. We quickly ported a solver library and provided an early evaluation, so they were able to fix some issues before they released UVM. It was a win-win.
- One has to be able to compose data parallel dispatch and polymorphic array layout. AoS versus SoA has been solved in our codes. We see negligible performance overhead versus native implementation. We have a lock-free unordered map that enables scalable algorithms with dynamic data structures. We are transitioning legacy codes (Tpetra and LAMMPS are now in process). We have a task parallelism LDRD via Kokkos/Qthreads integration and we are looking into re-factoring the core to enable memory space, execution space and execution policy abstraction.

Bergen – Are there double precision atomics?

- We put a layer in which looks like C++14 atomics and I am persuading C++17 to go with the pattern.

Janine Bennett, Sandia, DHARMA (Distributed asynchronous Adaptive Resilient Management of Applications)

- We want to develop performant, portable, scalable, fault-tolerant programming models at extreme scale. I will first discuss a comparative analysis we are doing of programming models for next generation platforms. The languages we are looking at are Uintah, Legion, HPX, Charm++, STAPL and Swift/T. This work began this summer (so we are just getting things underway and

doing initial studies with building block kernels for each code). Once we have completed initial analysis we will down select and run mini apps relevant to apps codes (e.g. MiniFE). The outcome of this work is to obtain guidance for code development on next generation platforms for ASC integrated codes. We see opportunities for collaboration as we work on mini apps.

- Our prime interest is in fault tolerance. Recovery beyond checkpoint/restart is going to be an issue due to distributed coherency challenges. We have focused on fault-tolerant components needed to create a holistic solution. We are looking into options for a fault tolerant layer and have implemented DHARMA in SST to enable coarse-grained simulation that will allow for system-level exploration using skeletonized mini-apps of explicit and implicit solvers. Based on what we learn our plan is to implement the holistic system by the end of next year.

Nelson – Do you cover fault tolerance both across nodes and on node?

- Yes in both the SST macro implementation and with our holistic approach.

St. Germain – What about the memory each task is dealing with?

- Right now we have an abstraction where we assume an NVRAM copy (remember we are looking towards next generation platforms) - but we are also looking at algorithms that implement a "buddy system". There are design tradeoffs.

Berzins – Mike Heroux is looking into the possibility of using AMR, i.e. a coarse version of a patch stored somewhere else. That seems a reasonable price to pay for the copy extension. Do you have a sense of what we can get back that we can use to tell us something has failed?

- Right now we are assuming the transport layer can tell us that something has failed. We have a heartbeat mechanism that provides for quick communication and we also have the notion of timeouts.

St. Germain – Are there statistics on how many CPU hours are lost because of hardware failures in the middle of jobs?

- We have a Sandia turbulent combustion code running now on Titan and they have 1-10 restart experiments, so 9 times/day they will have to trigger a restart due to hardware failure.

Rich Hornung, Raja Portability Layer

- Jeff Keasler and I have been working on this for awhile. We are exploring options for moving things forward and to get people to think differently about abstractions. Given the way our codes have been written, it is hard to manage hardware complexity (e.g. NUMA, increasing importance of SIMD vectorization/SIMT). This is challenging because the integrated codes are large assemblages of multiple codes (each of which can mean millions of lines of code with thousands of loops).
- Raja is a potential path forward. We need algorithms and programming styles that can express various forms of parallelism. At LLNL most of our codes are written in C++. There is no clear choice for a programming model. The aim of RAJA is to decouple the algorithms we have implemented from architecture-specific programming details and enable the developers to think about how to make the algorithms operate more efficiently, change memory, etc. It is critical that we have C++ lambdas support; RAJA can encapsulate various programming models (thus, a code need not be wedded to a particular technology that may work well on one architecture but not well on others).
- The goal is to abstract out what the algorithm is supposed to do from exactly how it is implemented. The code seen on Slide #5 compares code for a C-style for-loop versus a RAJA-style loop. It is important to note that the loop body is the same. Transformations can be adopted incrementally, and each part can be specialized for a particular code or architecture.
- RAJA focuses on inner loops (structured loop nests, structured loop nests) as the fundamental work construct (this is where the majority of the work in our codes is done). We need a single abstraction that handles both coarse and fine-grained threading (because CPUs and GPUs currently map iterations differently). RAJA replaces traditional loop constructs with loop-body and

traversal template (see code comparing use of RAJA traversals with OpenMP on Slide #9). RAJA traversals also work with CUDA (with a caveat because NVCC lambda support is only under development at this time).

- A single loop iteration is associated with a footprint of data values in memory. We think about this hierarchically and bundle iterations into what we call segments – this way we can bundle by data access patterns. Segments are defined at runtime and can be adapted to different memory layouts or architectures. Segment definitions can enable compile-time optimizations. An Indexset is a container of segments. Indexsets enable optimizations when using "indirection" arrays. We ran an ALE3D problem for a long time (a large problem with 10 materials, 512 domains, 16+M zones) and determined that hybrid Indexsets can help to recover some lost performance by exposing traversals of stride-1 ranges. Right now, we have two levels of scheduling available (inter and intra segment). A scheduling/dispatch policy is applied to the collection of segments and a scheduling/execution policy is applied to the iterations within each segment (see examples specified on Slide #19). Then we can dispatch these segments to different architectural resources. All aspects of execution can be tailored (and we want developers to experiment with different aspects of the execution). We want to enable a fine-grained level of execution control (because we often do not know ahead of time what is going to work well). The traversal mechanism provides a bridge between user code and runtime.
- Sequential dispatch with OpenMP execution allows lock-free schedules. Slide #29 shows comparisons of performance for LULESH v1.0 serial performance. The tradeoff between performance and flexibility using LULESH-RAJA appears to be reasonable. Data on the next slide show LULESH v1.0 strong scaling on TLCC2 with only a small change to the source code (and the results appear very good). The RAJA OpenMP version is slower at small thread counts (e.g. 15% overhead at one thread) but lock-free overcomes this issue. There is something we need to work through with respect to what the compilers are doing.
- Issues remain if we are to make this viable. Overheads are due to compiles, runtimes and language/PM standards. C++11 is new (lambda functions), compilers do not aggressively optimize OpenMP and there are aspects of C++ and other standards that do not benefit HPC. Other efforts are exploring C++ abstractions to encapsulate hardware abstractions.

END BACKGROUND PRESENTATIONS AND BEGIN TUTORIALS

URL: goo.gl/s56Uaz

John Schmidt, Uintah Component Developer API Tutorial (John Schmidt, Todd Harman, Jeremy Thornock)

- The focus of my discussion is on writing portable applications for scalability using DAGs. We have used Uintah since 1997 (when we were happy to get 400 cores on the LANL Blue Mountain platform). The overall message to take away from this discussion is that coding for an abstract graph interface provides portability.
- Earlier talks focused on node-scale performance; now we will address large-scale scalability. Uintah has been used in a variety of applications (as Martin explained earlier). Right now, for example, a chemistry grad student is running large-scale simulations. All of our large-scale apps have different task graph representations (e.g. shaped charges, foam compaction).
- Uintah works because – years ago - Steve Parker got the abstractions right. We may not be perfect but we do a good job. Developing new computational components makes things easier when the CS and computational infrastructure is right. The abstract task graph representation encapsulating computation and communication makes things easier. Within the DataWarehouse the developer need only ask for data (not MPI calls, for example).

- We program for a patch, and multiple patches can reside on a single node. There is a representation that Uintah has with patches that creates a convenient model for developers. A task has two features: a point to a function that performs the actual work and a specification of the function inputs and outputs. As part of the specification, we delineate our halo region. Our API sets up four functions and then we do main computational work in the timeAdvance. This is where we set up what tasks are going to take place; the bulk of the work takes place in the timeAdvance (where there is algorithmic implementation).
- We need to do further analysis to determine the way to break-out computational tasks needed to get optimal performance (but our motivation is really to cause the least pain from the developer's point of view). We worry about correctness; and worry about optimization is secondary to that.
- Now we will look at the Burgers Equation, as an example, and how we would program this, specifying details about the grid and setting up an input file with two patches, each in the x, y and z directions. All eight patches could run on one node or we could run a single patch/core. The code I will show you will run either serially or in parallel, depending on the patch layout. It works because the Uintah model makes parallel development easy. The biggest struggle new developers have is getting the dependencies right. (Discussed details of Burgers Equation Code – see slides for actual code).

Bergen – What if you want to implement Red/Black Gauss-Seidel (when you probably want to update in place)?

- Then you use modify – we can do that. Once we have our data set up, we iterate over the patch.

Question – Have you ever worked with two distinct materials with different material models, both of which may need to complete before you do something subsequent or send data to another process?
Harman – For constitutive material evaluation we do a loop-over
Utah – With ARCHES we have several instances where we have the same set of loops going over each task - but certain tasks require much more work because of the model.

- In some sense we have an OS that drives our Uintah framework (showed graphic for runtime system). We need not worry about how to schedule things (because we ask for dependencies). The separation between component code and infrastructure code allows us to stay segregated (to some extent).
- The iterator construct has been in Uintah since the outset. James looked at it and found we are spending much time doing calculations that impact performance. We are excited about James's work to get rid of the iterator ranges and we expect to see (ultimately) performance improvements.

Question – Is there a requirement to use the iterators?

- No, not really.

St. Germain – It was part of the process we used to originally develop the code. It is likely more complicated than it needs to be (and that means overhead).

- The graphic on the next slides was made when we were running codes on 98K Kraken cores. The distributed task graph is complicated but the developer does not need to worry about it. Now, Todd will discuss a more complicated ICE example.

Todd Harman, Three Task Examples (ICE, RMCRT radiation and MPM)

- I am an app developer who has been working at Utah since the CSAFE Alliance Center was in place. We begin with a Uintah domain decomposition forming patches. (Discussed pressure and density functions from ICE (multi material code) Algorithm and the scheduling code needed to calculate – data to come from data warehouse - see 3 slides for details on scheduling and computation).
- There are three phases to the task: (1) get data from the data warehouse, (2) allocate memory and "put" variables into the new data warehouse and (3) run the computation via template function that performs the calculation on the x, y and z bases.

- Reverse Monte Carlo Ray Tracing (RMCRT) with an all-to-all means that every cell potentially needs to "see" all cells in the domain. This is ideal for GPUs because you can have individual threads working on each ray. We want to compute the divergence of the heat flux. The next few slides show the scheduling of the task (and we create both GPU task implementation and GPU task implementation). This way to solve radiation can be used by multiple components in the infrastructure (i.e. Arches, Wasatch and ICE). We designate which data warehouse to use. If the task is GPU capable the Parallel use device command can be used.

St. Germain – We do not presently dynamically schedule to either a GPU or CPU.

- But the logic is there to have it happen dynamically and I don't think it would be hard to set up. A task can modify a variable, or it can compute it. If we compute it then it did not exist before. We add the task, over each level, to the scheduler.

St. Germain – When you schedule a task that modifies a variable, it will create a dependency, based on the order on which the tasks are given to the scheduler.

- Inside the task, we passed in which data warehouse we pull from (i.e. old or new) for each variable and we can set a frequency (in timesteps). From the data warehouse, we convert from the old data warehouse to the new data warehouse and get the data (but first we have to know how big the domain is). We get a region for a data warehouse (old or new) and have the entire computational domain in memory for the three variables (because it is all-to-all).

Question – Why is the GPU version not the same?

- It is the same from the infrastructure point of view - but the implementation is completely different. The last example I will discuss is the Material Point Method (MPM). I will go through the simplest task, which is to update the particle position. You know the old location of the particle, along with the velocity, and you are going to get a new particle position. We treat the variables the same way we did previously, except that we only do the particle calculation at the finest level (see code on slides for MPM task scheduling and calculation). We loop over particles that this task owns, get the node indices that surround the particle and accumulate the contribution from each vertex.

St. Germain – Can tasks do their own parallel implementation? It could be passed to the processor group and the processor can do whatever it wants to do.

- Once you create an ICE object and/or an MPM object you can access them at any time and schedule them. We do a gather/scatter operation at the timestep, as needed.

Karlin – Do you have any concept of where the data will go in a NUMA hierarchy?

- That will be discussed later in some detail.

Jeremy Thornock, Arches

- I developed Arches (our LES combustion code and primary production code) for the PSAAP II Center. Before I go into details let me ask first, are there any questions?

Question – What parts of this can be used for unstructured grids?
Berzins – In all cases we have the concepts of patches and halos. The former are hard to compute because we have subdivisions and/or hierarchies. With an irregular structure I think you could use halos, and conceptually I believe it would be similar. I appreciate that things would be more expensive because you are dealing with unstructured data.
Karlin – What about unstructured code built with structured patches?
Comment – We can do relatively complex geometries with intrusion. Uintah is a block-structured code.
Sandia – Can you have arbitrary connections?
Berzins – I think we could do that but we would need to create the capability.
Nelson – Can you define a different type of data structure you want to store in the data warehouse?
Harman – Yes, we save the file pointers in the data warehouse (we do not open and close the files).

St. Germain – The component does not need to know how the data is stored because it just tells the data warehouse to give it a variable. What you are asking would be straightforward from a developer's point of view (although how to make it scale well is another question).
Utah – We handle particle types in EDSL, and that is on the way to an unstructured field because connectivity has to be specified. EDSL would work today if you had a block of memory from an unstructured framework. If you implement that in Uintah then you could go all the way down the software stack.

- From my standpoint - as a Uintah physics developer - the nice thing is that I need to make no code change (the work is all done at the backend).

Sandia – In Uintah does data get redistributed?
Berzins – Yes (we do not call it AMR capability).
Harman – We transport the particle through the grid and use patches to push things around in the infrastructure. We can do it with or without AMR.
Kunen – Are your patches always spatially defined?

- We define them all in space (i.e. domain decomposition in x, y and z).

Still – Would anything preclude you from defining in higher dimensional space?
Utah – With radiation we assemble a bunch of tasks that would solve across wave-number space. To my knowledge, the framework does not support arbitrary dimensional problems (it is meant to support 4D – space/time).
Still – Photons can scatter off each other and there is a non-trivial intersection in the energy direction that has to be solved, too.

- How does Arches interact with Uintah? There is not much difference from what you have been hearing. We schedule tasks, interact with the data warehouse, and compute. I could talk about how the physics impose demands on the framework and computation, if that is of interest to you. Arches is a combustion code (at heart) that solves mass, momentum and energy balances via a finite volume approach (integral form of the conservation equation) explicit in time (up to third order in time) that staggers velocity from scalars (2^{nd} order in space). With our PSAAP II problem we are working towards representing a coal boiler, so we are stair-stepping and representing geometry on the patch in an intrusion form. The simulation on the next slide shows LES simulation of oxygen on the boiler in a multiphase turbulent flow.

Berzins – When you look at the flow you understand why we are not using AMR.

- The next simulation shows the particle distribution (number density) for one particle size – it is everywhere throughout the entire boiler. The boiler is a giant tea kettle, essentially. The boiler sits inside the water bath with steam coming out the top. There is action everywhere (lots of turbulence and combustion). At the beginning AMR may focus on the inlets - but you will end up being at one level.

Berzins – If we use ray tracing we will have to use AMR.

- We are talking about a boiler that would ultimately go to 350MW (what we are looking at on the slide is ~50MW). Much of what we are doing as an explicit code boils down to scalar updates as we move forward in time. The cost varies, depending on which physics you are trying to represent. As an LES code we have filtering going on, mixture fraction equations that are being updated – the costs of these updates varies. Two critical points on our algorithm require global linear solves (radiation and pressure). We have to be able to project velocities and that takes a linear solve across the whole mesh. We solve externally with Hypre/PETSc (and, hopefully, soon we will use Trilinos as well). We also tabulate a number of tables to provide lookups for the gas phase chemistry. Given the variables we track on the mesh, we have to query a table to obtain values to feed into models downstream.
- A unique feature is the particle capability. We have an experimental facility on campus that is a 100kw down-fired burner, and we run simulations with Arches as part of our UQ work. This boiler feeds in 58M particles/second (on the order of 5 meters/second), and we have to represent that in

some distribution. By comparison, the Boiler Simulation Facility is 15MW Tangentially-fired boiler with multiple injectors and 34B particles/second. We are to go to the 350MW boiler in our PSAAP prediction challenge – how many particles/second is that going to be?

Sandia – Validation experiments?

- We do that on campus. I know they are using a lot of laser diagnostics: number-density functions, velocity distributions, temperatures on walls, distribution out the backends. On the horizontally fired burner there is a single sample point. The BSF is well characterized and the Alstom dataset is good (we have access through a data sharing agreement).
- We represent the number density function (total number of particles per internal coordinate, where the coordinate is anything that characterizes the distribution) as shown on the slide. We focused our research on a Eulerian approach (we are also resurrecting a Lagrangian approach which Phil Smith pioneered for coal). Hopefully, we do not have to model every particle (34B particles coming into the BSF inlets/second). That is a lot of particles and we are not going to be able to represent every particle, so we take a moment (Eulerian) approach or take a Lagrangian approach using collections of particles. (See details on slide for Direct Quadrature Method of Moments or DQMOM approach (which works but there are problems), so CQMOM is being studied because it represents with more fidelity but it is also more expensive). Pure CQMOM is likely not tractable; we are looking at combining DQMOM and CQMOM.

James Sutherland, UintahX Runtime Executing DAGs for Scalability

- (Led group through tutorial on SpatialOps via the URL provided earlier.) Adding new language features is messy and requires code generation. Typically the team on the programming languages side adds what is being requested.
- You have to dig into the C++ template un-roll in order to see where the code is generated but – in the end – the kernels are fairly straightforward. We have seen no problems on the compilers we have tested (we have so far tested Gnu but not Intel yet). This is all template meta-programming (and we will discuss more details about what happens “under the hood” tomorrow).

Utah - So far we have one client for this tool (and that is James). If we have more clients then we would add more features.

- Generally, you want fairly fat kernels (because they perform better) - but it also depends on the architectures (i.e. GPU, multicore). At this point we only have anecdotal evidence. We have been targeting performance on our big calculations and, so far, we are seeing 5x speedup.
- Early incarnations of Nebo used matvec but when we want to do that in-line we need to have a pre-processing stage (and that would be a new project).

Karlin – Can you build a discontinuous finite element?

- You need information about connectivity, so you have a hard time making the compiler chain the operations together. You would only be able to use a subset of the language if you go discontinuous.

Berzins – Discontinuous Galerkin (DG) has a completely different structure. DG has the potential to be better than many methods we have for modern architectures. I think the answer is probably yes, I think you could write a discontinuous finite element in Uintah if the same meshes were used.

- An example that ties the others together is a simple PDE (transient diffusion problem in 2D). We want to solve the problem using Nebo, so we will build on the other examples I have discussed (i.e. basics of field creation, operations using field, performing reduction operators, natively support field types). We begin with constant diffusivity and find it is trivial to accomplish. We need to set boundary conditions, so we build mask points to describe the boundaries and set boundaries quickly on mask points. We build masks on each face and want to move them to the GPU. This method builds stencil and – given a grid – will populate the operator database with 60-70 different stencils supported “out-of-the-box” in Nebo. We retrieve operators from the database we just built. The right-hand-side (RHS) calculation is the one I care about. There is an

interpolation that has to occur. And there is one kernel call that takes place to calculate the RHS. I update the solution and reset boundary conditions so they are in place for the next time step; I do that for each mask. Then I print out.

Bergen - Do you have a mechanism to collect operators?

- Yes they are built outside of the for-loop.

Question – How would you extend this to AMR?

- You would have an outer loop that goes across mesh levels. You have flexibility over what you compute and what level of granularity you compute at.

Hornung – These examples all use uniformly spaced grids. What if I wanted something that was still structured but had different mesh increments?

- The operator database can be done per level or you could build operators that have embedded information about grid metrics (or stretching). We are still limited to structured meshing. Right now because our big app codes are sitting inside Uintah (structured, uniform with AMR) we have taken as many shortcuts as we can. If we go to stretch meshes or meshes with metrics you can embed that in (but it comes at a computational cost).

Hornung – And that is what we do.

- You can do that. The operators in the operator database and resolved in the computation can have as much richness as you need. As soon as you go to unstructured though, things get more complex.

Berzins – The deep question is what is going to persist in the longer term because many people are working on DSLs and task based approaches now. I think this project is more focused on showing that these ideas work and how powerful these approaches can be. It will be interesting to see what persists among the efforts taking place now.

- And right now we lack drivers to move in that direction. If a lab or another partner says they want to explore moving towards generalized coordinates (for example) and the partner is willing to put forth some resources then that is something we would like to pursue.

Davison St. Germain, Uintah Runtime System Tutorial

- The simulation controller has a run routine, sets up the grid, tells each component to run an initial timestep and then goes into the main loop. We re-grid as necessary and re-compile the task graph as necessary and finally execute the timestep. If more information is needed at a certain place, re-gridding uses AMR. We re-grid based on information gleaned from a refinement routine. There are a number of ways the load balancer can decide to put information on components. The data archive component outputs either specified variables or the entire dataset.

Nelson – Where does load balancing happen?

- Only if there is a re-gridding. Usually we create static load balancing and allocate tasks through the individual processor units. We also have profiling in the code that can kick in if the load is out of whack.
- The framework takes care of moving all data for the components, so the component need not move data from one processor to another. There are different ways to schedule MPI/thread/GPU execution - but we settle on the most messages from a task. We distinguish between a task and a detailed task. Patches are multiplied by the number of cores to get the number of tasks and that became unwieldy, so we define detailed tasks as the number of processors in the local area (and doing this took a scheduling bottleneck out of the system). Internal tasks have no communication needs outside of the local processor or node. External tasks are executed on the node or processor but have communication needs. (Discussed notional graphics of tasks and task graphs seen on slide using single processor scheduler, MPI scheduler and unified scheduler). We can duplicate the task graph within a single task so we can have iteration when necessary. The code on the slide is used for task graph compilation – detailed task dependency generation. We look at all requirements and variables to be computed, along with dependencies. What does this gain

us? By putting all the tasks together on a single node we can align them and save much waiting time for the system.

Hoekstra – Have you struggled with scalability of problem set up?

- We did early on when we tried to put all tasks on all processors but when we limited to early neighborhood that eliminated that bottleneck.

Berzins – The key thing with AMR is our use of regular refinement – this is local and limits the size of the refining region. It is a good example of an algorithm when you want to limit the number of patches. On a large machine you want to have lots of patches however. We have an analysis showing why this works well (mesh refinement around a circular front). We performed a scalability study for AMR and produced a number of papers on performance improvements. We are looking at load balancing and Justin looked at algorithmic cost models. Time series analysis was found to be a better approach and is used to forecast time for execution on each patch. It automatically adjusts according to the simulation and architecture with no user interaction. We see a good fit for MPMICE execution time between what we see and what we expect to see. A combination of prediction and estimation is important when thinking in terms of load balancing.

Hornung – Do you do refined time-stepping with your AMR?

Berzins – There are two options. Mostly we use single timestep.

Utah – You get locality through the space-time curve (i.e. split it into equally weighted line segments).

Berzins – And that might be part of the reason why the cost model is not as good as the data simulation.

Bergen – Do you handle sub-cycling on the refined mesh?

Utah - We have two ways to do that, depending on the component and what the problem is.

Alan Humphrey, Uintah Runtime

- I will talk about how we handle and execute tasks. Let me begin with the current incarnation of the Uintah Unified Heterogeneous runtime (RT) System where we use a single MPI process per node and pthreads bound to cores executing tasks. Every thread on the node has the ability to process its own MPI; it accesses task queues and executes tasks. To remedy NUMA effects on a dual socket node you could run two MPI processes (but we have traded that for the scalability in general). I was asked to extend Uintah to GPUs and the task-based approach makes it relatively easy to add in GPU task queues, GPU warehouse and a function that launches a kernel.
- On the slide is the source code for implementation of the unified scheduler. Each thread executes this method, checking task queues for work. There is a task selection phase and a task execution phase. We have internal-ready and external-ready task queues. The code on the slide looks for detailed tasks in the external ready queue, finds a task to execute and goes into a concurrent section of the code where every thread is executing a task.

Question – You rely on thread-safe MPI?

- Yes, multithreaded MPI with interlaced CUDA. If you grab the tarball you can look at the source code file to get a better sense of what is happening in the task execution (per thread). When a task has had its dependency count go to zero it goes into the external ready queue.
- Some challenges and overhead with MPI thread multiple? We have to do communicator duplication to ensure correct message matching. The infrastructure schedules tasks – most Uintah tasks are thread-safe (we run into a few that are not, on occasion). Tasks from the component or user code also have to be thread-safe. Overhead for acquiring locks becomes significant with high thread counts. Multi-threading coding is complex and often painful to debug (and many bugs only manifest at high core counts). The next slide shows two chunks of code for signaling worker threads (we wrap synchronization primitives in C++ classes – when you wake the threads up they begin to execute tasks).
- Global scalability depends on the details of the nodal runtime system. We had things that ran fine on Jaguar but not so well on Titan because more, faster cores and faster communications broke

our runtime system (which worked fine with locks previously). At this point we are running at a minimum of 15 threads per node. (The graph on the slide shows single node performance). The code on the next slide shows details for using atomics. There are still a fair amount of locks we need to pick off as we go. We see almost linear scaling on Kraken on a challenging data problem when we go to decentralized MPI/thread hybrid scheduler.

Berzins – One AMR MPMICE problem now scales well and the other older version fails (they are not the same mesh or the same distribution of patches per core).

- Turning now to GPUs – when we wanted to extend Uintah to GPUs it turned out to be fairly manageable (at the time we were looking at the Fermi architecture). But, we are basically feeding a Ferrari through a tiny straw – how do we hide PCIe latency? That was the motivation for developing our heterogeneous runtime. We leveraged the Nvidia CUDA asynchronous API. Operations from different streams can be interleaved. To extend the scheduler to GPUs we add in GPU task streams and the GPU data warehouse.
- We have to have a staging area on the CPU. The infrastructure from the task graph knows about dependencies for a particular task and can grab CUDA streams and begin asynchronous copies (so it becomes resident on the GPU). Then you grab pointers and use them in your (hand-written) kernel to concurrently execute kernels and memory copies. We preload data before the task kernel executes.

Sutherland – With up to 16^3 problem sizes we find launch latency will dominate the execution - but with significantly larger problem sizes the latency is washed out.

Utah – Launch latency from the host to the GPU is about 10-15 ms, and much of that is software latency that you can pipeline. If you have all of the kernels run at the same time you can overlap that even further.

- Our biggest customer would be James and Wasatch. The data we copy to the GPU they grab as pointers, wrap them, and use them. The Monte Carlo ray-tracing is the first hand-written piece we are working towards. We have a single level version now with all physics that is an order of magnitude faster per timestep. We will push that forward to the multilevel approach. The next slide shows lines of code for the GPU task selection (per thread) with CUDA enabled. We overlap computation, PCIe transfers and MPI communication.

Hornung – Do you try to specialize on thread block geometry based on task?

- It is up to the application (i.e. the kernel being provided to us by the user). It would not be too hard to build “smarts” in. Right now there is no current notion of GPU affinity but that would not be hard to put into the code.
- We wanted to make the GPU data warehouse API look as much like the CPU interface as possible, so we maintain simple get/put operations and overload operators you can index into the grid variables. To the user they appear like the CPU-side.
- With the GPU ray tracing (which is production ready) we have scaled out to 8K cores. We find with the GPU approach that we basically run out of work. Our next step will be to push the ray tracing kernel using our multi-level approach (so we will not do the full all-to-all). We got early access to Stampede when there was a barebones software stack (and the results of runs on Stampede are seen on the next few slides). We were up and running on MIC within 24 hours (although not with great performance). The offload model turned out to be the hardest for Uintah because functions called in MIC. In the end, the symmetric model fit our model the best. We ran into load balancing issues (because Uintah treats every MPI rank equally and they are not equal on the MIC). We are waiting to get access to Knights’ Landing to take this analysis further. We ran into an issue where we got differences in floating point accuracy with the symmetric model (we fixed it with a compiler flag but it was not obvious at first).
- In summary, the DAG abstraction has been important in achieving scaling but it has not come for free. With every new architecture, something breaks and we spend significant amounts of time to fix what breaks. The clear separation between the infrastructure code and component code

shields application developers from details of CUDA, pthread and MPI. Our GPU development is ongoing and we are showing good scaling. We are waiting for access to in-socket MICs (Knight's Landing).

Nelson – How closely is the scheduler linked in with the data warehouse? Could I use the scheduler with a different database?

- No, another database format would not work.

Hoekstra – Could you implement an API?

- It would be complicated. I will be happy to talk with you further about it offline.

Hoekstra – We have some pre-production testbeds at Sandia. When you have the time, please use them. (There are some NDA considerations that have to be addressed first).

- We would love to do that, thank you.

LANL – Do your tasks have to be stateless?

Berzins – We would like them to be stateless - but some of the codes are problematic because people find ways of doing things that work without fully understanding the implications.

- The data warehouse holds state data.

Berzins – Note that we place all of our papers online if you would like more information.

Day 2 – Tuesday, July 29, 2014

Matt Might, Advanced Template Meta-programming for Portable Performance

- What I am going to talk about is actually meta-meta-programming (and is the work of my PhD student Chris Earl). This is about more than just portability and performance (although those are important) - it is also about expressiveness. We would like to also work towards fault tolerance and – by shrinking the code as close to the math as possible – enhancing correctness.
- What is the problem? James will say that what he wants is for us to enable him to be able to write code cleanly and easily. The code has to run on multicore, CPU, GPU or Intel Xeon Phi.
- How do we do this? It was clear to me at the outset that we had to have a new language … so long as it is C++. We are stuck with C++ template programming. What is different about this template meta-programming? At the highest level, there is the fact that - with the same piece of code - we can compile it simultaneously for multiple backends. The other distinguishing feature is the idea of meta-meta-programming while, not strictly necessary, makes this a more tractable engineering problem.
- Our core idea is the notion of building the structure of the computation into the type level at compile time. (Discussed example on slide). Once you have the structure of the computation lifted out at compile time you can inject yourself into the compilation process to perform functions such as execute, evaluate on CPU, evaluate on multicore, evaluate on GPU, evaluate on Phi or debug. For the same expression we can compile via one thread or n threads.

Nelson – How many threads?

- The threads are fixed at compile time.

Sutherland – The cap on the number of threads is fixed but the actual size of the thread pool can be smaller than the cap.

- Let us look at the compilation of one expression to trace through the process of how this happens. To begin we will add two fields together. On the slide you see the source code for "plus" (+) – showed many, many lines of code with definitions to capture types of summation and many overloads. (Discussed the portion of the code that would be used for this example). There are many assertions within the meta-programming to ensure we do not allow the hardware to do something that should be physically impossible.
- Now we look at the programming needed to conduct the assignment (when you see <<= then you know this is a Nebo expression). We have a dot template, wrap the left-hand-side (LHS) in a Nebo field and begin to unfold the expression. We wrapped the LHS in a Nebo field and when

specialized with "initial" three internal Nebo fields are created, so within a field we get three versions of the code at the same time (as fields). At runtime the Nebo field checks to assess whether the RHS and the LHS are compatible. If both sides are on CPU, as an example, then the code runs on the CPU. If they are not compatible then error messages are printed. We do not make implicit moves for the programmer because we do not want to hide the cost of doing so. The code to also run the same calculation on GPU is in place, so you can do that next. Running on the different types of processors is referred to in Nebo as modes.

Hornung – Range of compilers?

- GCC, NCC but not yet Intel-X (which is on our shortlist). We can run many small scale test cases. We are trying to make this not depend on the whims of the compiler.

Hornung – Do you do performance testing? Have you compared to straight for-loops so compilers can optimize well?

Sutherland – Our target is to be at least as fast or faster as hand-optimized code.

- My background is in compilers. There are design decisions made throughout Nebo to make certain we do not violate the compiler (otherwise things would get very slow).

Hoekstra – Could you make this easier?

- Yes.

Karlin – What actually is handed off to the compiler?

- A triple-nested loop.

Utah – Our main goal now is to get everything onto the GPU (there are some pieces that are not there yet). Vectorizing instructions would come after that. I don't think it would be too hard – you would have to create a different backend specialized to vectorize instructions (so the loop would take multiple inputs rather than just one).

- Right now the backend does not have particular parameters (but it could if you know the memory or cache size).

Karlin – Could you specialize on the size of the computation?

- Yes, you could. There is nothing stopping you from adding another backend (except that each backend means another ~11K lines of code). With Fulmar we are engaging C++ template meta-meta-programming. We have Nebo expressions and the template programs got bigger and bigger. Every time James wanted to add a function it meant a week of programming. Fulmar generates the template language (so it is a meta-meta-programming language). Once you have 500 lines of Scheme in place then to go from the first operation to the second means writing one more line. This is our secret attempt to get Scheme running at scale.

Bergen – If you could use C++11, could you use R values?

Utah - It would help. Auto keyword would help a great deal.

Bergen – This seems like table gen in LLVM (they were trying to solve a similar problem).

- I will look into that (it is less error-prone). The code that comes out of Fulmar is surprisingly readable. You can find it here: https://github.com/cwearl/fulmar
- So, that is a top and bottom deep dive on one slice of Nebo.

LANL – What you are calling a walker, in C++ the same concept is called an execution policy.

- That is good to know.

Berzins – There is much activity with respect to DSL space now. What do you think will persist?

- I wanted to throw out C++ and start from scratch but there is too much legacy code out there. The reality is that scratch codes – unless they offer significant performance benefits – are not going to be worth giving up over today's codes. This approach offers flexibility to get from where we are and where we need to be (without having to make discrete jumps along the way). C++ programmers can begin working with this today.

Sutherland – We started a project to leverage Nebo to do source-to-source transformation on large scale granularity decisions. We want to automate that (so one would be able to dial in granularity underneath the kernel execution).

- We are focusing that effort where we think the biggest benefit will be: to help with high level concerns. That is where we think it would pay off.

Hornung – Is Nebo a project? SpatialOps we talked about yesterday.

- Nebo is a language.

Sutherland – Nebo is a subset of SpatialOps (which is built on top of Nebo).
Utah – Nebo is whenever you see <<=.
Hornung – When you get to more complicated things like gradients, how difficult would it be to write more difficult, arbitrary stencil-type operations? What if you are indexing the various arrays differently (i.e. not everything is passing through the same mesh)?

- Nebo does stencil operations layers.

Sutherland – At the Uintah level there is the concept of materials that exist on a patch and that index exists outside of a Nebo kernel.
Hornung – How do I get at other data structures I need down in this layer?

- We could have index transformers for coordinates.

Utah – We have not done it yet.

- If I were to add that, I would add index transformers to get the needed data.

Sutherland – We would need to see a concrete example (there are a number of solutions to what you are asking).
Karlin – We could do it today with masks but that might be inefficient.

- And this is why we stay client-focused. We add features when someone requests them and gives us sample problems.

Karlin – How much time did it take to develop this infrastructure?

- Baseline time was a PhD (and to go further will mean multiple PhDs). Getting Fulmar to the point where you could add more features would mean another PhD.

Hornung – Imagine you are initializing three or four different arrays …

- We want to use auto in that case. You could capture the expression but you have to define the type (and that would be onerous). Auto would help. In principle it can be done (the question is how cleanly we can do it with C++11).

Karlin – If you have a mark to identify things that are fuse-able then that would make it easier.

- Yes, I agree. We probably would come up with a different assignment operator.

Karlin – I would build a DAG at a high level to provide another layer on top.
Utah – We did not want to have to do that in meta-programming. We are building our own language for source-to-source translation because that is an easier way to do it.

Alan Humphrey, More on Scheduler

- (Showed illustration of Uintah task scheduler before re-factoring). We use one MPI process per node and pthreads to pin the task. This is our answer to MPI+X.

Hornung – Is it possible for a task to launch other tasks?

- Yes and I will discuss global synchronization tasks in more detail. We call out to individual libraries (e.g. hypre). There is a convergence loop within a task that can create a sub scheduler. We have done some initial exploration into doing multiple task graphs. Everything is distributed: schedulers, data warehouse, etc – they live on each node.
- Looking at the Uintah nodal runtime layer we see the data warehouse shared resources (one per multicore node). Each thread on a node checks communication records, accesses the data warehouse, processes its own MPI and executes tasks. The slide showing the Uintah static task scheduler reveals how far we have come – this evolved into dynamic execution and from there to multi-threaded. The new hybrid model memory savings are obtained via ghost cells.

Hornung – How do you manage dependencies when you are not exchanging payload?

- You eliminate all the inter-node MPI. The only patches that communicate will be on the outer edges. Our on-demand data warehouse knows what data each patch needs, so when you ask for a variable on a patch from the data warehouse it assembles the data and provides it.

Berzins – There is the old data warehouse and the new data warehouse. You take data from the old data warehouse and write to the new data warehouse – that is how we manage that. It is based on the notion of how halos are passed around and shared.

- We are running two data warehouses per multicore node.

Berzins – It is a clean thing to do (but does take storage).

- Essentially everything is non-blocking. Global reductions (only) are one-sided. We want to look at non-blocking collectives more over the next few months. (Discussed source code on slide for task selection per thread). In the multi-threaded task scheduler there are two phases: (1) each thread selects a task and (2) the task executes. For each iteration of the loop (code shown on the slide) the task may do something different.

LANL – If a task has to block and wait, does the task yield the thread to another task?

- No, it does not. If a task is scheduled it is ready to go, so the thread is dedicated to run the task until the task is complete and then the thread cycles back through this loop.

LANL – If one task spawns three tasks it waits for all three to be finished before it can move on?

- I will discuss that further. Because of the Army MD work we can create a secondary instance of the scheduler with the thread pool.

St. Germain – A task does not spawn sub tasks. The only way it could do so is if it has a complete sub scheduler. And the sub scheduler is a complete task graph that has to be created and is pre-defined.

Karlin – If the result of one task tells me what task to run next, how do you handle that?

- You would only make a decision like that if you are re-gridding in AMR (in which case everything changes).

Karlin – We have instances where if a value changes it changes the physics that is being run.

Harman – For our pressure solve we have something like that. We do not satisfy the criteria so we have to reschedule.

St. Germain – Every component has a read-recompile function. Something that has changed the simulation dramatically sets this flag.

Kunen - How does re-computing a global norm (e.g. Allreduce) fit?

St. Germain – Any global reduction is scheduled in the same order and we push them down to the bottom of the task graph to the extent possible.

- I have some details about that I can show (I skipped this yesterday). You use a reduction variable and the system handles the reduction.

St. Germain – Say a detonation occurs in the simulation. Then we want to start saving more data (as an example of when a simulation needs to change and we need to update).

Kunen – Can you dynamically add?

St. Germain – Yes, at the beginning of the next timestep and then you dynamically update your task graph.

- You can mark a task as using MPI; that should make things safe in the infrastructure and create a synchronization point (but in my experience so far, it is not guaranteed).

Berzins – We treat some things as black boxes (and as such do not attempt to control what goes on inside them).

Sutherland – You might speak to how Uintah interacts with one of your linear solvers.

- We mark this a task that uses MPI and hand over the communicator, let it hijack the MPI runtime and then wait until we return to the solve.

Berzins – And that may not be ideal but we do not have an algebra shop here.

Sutherland – There is a CG solver written in the framework, and it works.

Berzins – We are focused on large scale engineering problems and making the code work (we do not delve into the details of the solvers at this time).

- With hypre with our multithreaded tasks – when we use the unified scheduler we reduce the number of MPI ranks by a factor of 4x – right now when we get to hypre solve phase using the threaded scheduler, we encounter a bottleneck (this is something we need to fix).

Sutherland – Synchronization with respect to linear solvers is something we need to work on. Uintah handles parallel slackness well until you get to the linear solver.

- We need to work with it further.

Berzins – We have a potential hit from synchronization overhead and we need to consider ways to reduce it. We have fairly good weak scaling to 260K cores on Titan (but there is a bug on Mira, so our scaling is not as good there). We do not have a huge incentive to do more right now. We will have to come back to this.

Sutherland – We are nervous now about the GPU side because when we hit the linear solve the GPUs sit idle.

Berzins - We have AMGX but NVIDIA only distributes binaries because they do not want to release the source, and that is an issue for us. What we need is an open source GPU solver. We know AMGX was used to scale up to several thousand GPUs. We think there is a good avenue for research there (and we have good relationships with Nvidia).

Sutherland – We have a side project to hedge bets on linear solves (and bypass the entire linear solve). (We don't like to talk about it openly).

Berzins – I will say that recently I was on the SciDAC review team for FASTmath and note that they were asked why they are not focusing on new architectures more as this is an obvious gap.

Olivier – To introduce (in real time) a new task to an existing task graph requires task graph recompilation?

- Yes.

St. Germain – You can have tasks already compiled in that get called (and are skipped based on other conditions).

Olivier – The sum total of the graph has to be there before execution?

- Yes, that is correct. On the slide is source code looking for internally-ready CPU tasks and to delay reduction. If there is no work then we process MPI and see if work can be generated for successive tasks. ICE, MPM, Arches, RMCRT are all thread-safe.

Still – At LLNL there is a tool call STAT – if you have a bug at high core counts using the tool you will have a stack trace.

- And that is essentially what I did (dump a stack trace).

Kunen – I used STAT on 2M cores on Sequoia and it took me two minutes to find the problem.

Olivier – Were the debugging issues you encountered a result of increasing the thread count?

- Yes we went from 16 cores to 64 and during dependency generation we had multi-threaded code we thought was thread-safe (but was not).

Olivier – Have you encountered a situation where memory reads and writes go out of order because there are hundreds of threads? Be ready for that when you get on lock-free data structures.

- No but I am excited to get on something where we can run more than 64. I will keep that in mind. Our initial threaded scheduler was a central control thread that would check queues and dole out tasks. We ran into issues so we decentralized such that every thread can access the data warehouse, process MPIs and task queues. There is still a master thread to wake the other threads so they can start running tasks.

Question – You signal pthreads on every timestep – what are they doing in-between?

- We wake them up; they execute tasks, and then wait until the next timestep. I/O is done, there is checkpointing by the master thread, and then we move forward.

Sandia – That is where we do the swap context in Qthreads.

- And I will look into that. Right now I don't know whether we are taking an overhead hit there or not. We use low level atomics for our garbage collection and for updating items in the database.

Sandia – What types of data are in the data warehouse to guarantee lock-free?

- Typically there is a pointer to a cell centered variable, pointer variable, etc – what we are actually doing is to swap out pointers. Once we go to a decentralized, hybrid MPI/thread scheduler on node with lock-free data warehouse we find we have good performance, as we scale up. We schedule global synch steps to the last quarter of the timestep.

Olivier – It is not that hard to write your own Allreduce.

Berzins – We are resource-constrained.

Hornung – When you run on Mira, do you run one thread per core?

- We use four threads per core, 64 per node and we can approach the performance we see on Titan. We find BG/Q to be a stable architecture (better than Titan). We have not done anything with transactional memory (and I did look into HPM).

Hornung – Each of your tasks is independent so there would be little value to transactional memory.

Sandia – What do you mean by work stealing (you mentioned it yesterday)?

Berzins – What we mean is that you might execute tasks from the same patch on a different core.

Sandia – It is not that you have separate queues and you are stealing from another queue – the cores are just available to all.

- You may have a set of tasks that reside on a patch and some subset executed by Thread 1, Thread 3, etc.

Berzins – We have to put the data warehouse on the GPUs.

- The task graph is a powerful abstraction because the scheduler knows ahead of time everything that is needed.

Karlin – Do you have a notion of pre-fetching data into the fast memory based on the task graph?

- Not currently. Right now everything is available on the GPU, and addresses of variables from the data warehouse come on-the-fly. If the task is marked as something that can be run on a GPU then copies are posted, so every field and all data the GPU task needs for execution is physically resident, including a GPU data warehouse (with get/put interface).

Karlin – What do you do to copy the data when the task is done?

- We execute the kernel via CUDA stream (i.e. fill up with operations that execute in that order), manage a queue of streams, dole out copies for kernel execution and manage copies to and from the device. This is all posted upfront. Once threads have called a callback function to execute the GPU kernel then (behind the scenes) the device host copies back things that need to be copied back (because they are going to be required as dependencies for execution of other tasks). Our next step is to keep as much resident on the GPU as possible and stage the MPI from the GPU itself.

Karlin – It sounds like you have most of the machinery in place. All you really need are some "smarts" to reduce the data motion.

Kunen – If you are moving memory to the GPU - and the total memory is greater than the GPU memory - how do you handle that?

- We do that with CUDA right now (and it is not particularly graceful). What we need to do when not doing solve in the future is allocate a pool and handle that ourselves. With RMCRT we are running large domains and maxing out memory on Titan's GPUs.

Still – All this will change when NVLINK comes out next year and this will be done on hardware, where it will be much faster (versus using assembly language on CUDA programming).

Berzins – We know things are changing very fast (and it is very frustrating).

- Right now there is no notion of GPU affinity but we have the capability to tie that in. The slide shows a notional illustration of the GPU Data Warehouse.

Bergen – Do you have support for Phi yet?

- Yes and I will describe more details. The symmetric model has worked (albeit not well) out of the box for us. We spawn pthreads and use the procedure for a single MPI process.

Berzins – We looked at OpenMP performance issues. There is no clear relationship between what you write and the performance you get. With OpenMP it is very frustrating (and we are not using it).
Olivier – We agree with that. If you wanted to make your internal tasks parallel, how would you do that?
Sutherland – The DSL in the Wasatch component has four levels of parallelism (two from Uintah and an additional level of data and task parallel beneath that).
Kunen – What kind of overhead do you see from the CUDA kernel for the get to the data warehouse?

- You could pass in the pointer and not do the get. We wrote it the way we did as convenience for the developer. It probably adds some overhead (as do other things that have been added to Uintah as convenience for the developer). Ultimately, you want to re-factor to use Nebo.

Berzins – What scaling results are you seeing from multiple GPUs?

- We can use them on Titan - but they do not scale well.

Berzins – We are not seeing good scaling with multiple GPUs.
Olivier – We find with GPUDirect that where the MPI is handed a GPU pointer makes a difference.
Berzins – And that handles one of the stages (but we are still looking at a tiny straw).
Karlin – It is about algorithms. You are going to be bounded by PCIe either by latency or by bandwidth.
Sutherland – We need to not move data back when that can be avoided. Within Wasatch we have the option of executing a large number of GPU kernels before going back to Uintah. The framework is moving towards being smart enough to know when data can be left on the device because the next task is going to use it.
Still – We are not the only ones with a slow straw – I think the vendors will fix the problem at some point.
Berzins – On resilience, James and I have an NSF project where we are looking at resilience. With the task graph we can deal with tasks failing. We need interfaces at the system level to help us consider core failure (re-route tasks), communications failure (re-route message), and node failure (need to replicate patches using an AMR type approach in which a coarse patch is located on another node – in 3D this means 12.5% overhead. This last item has been suggested by Mike Heroux and others, and we will explore it beginning this fall. This is a fluid area but we do not yet have a full sense of the magnitude of the problem. The fact that we are a small market does not work in our favor. It is hard to be sure that we have the answer when we do not yet understand just how big the problem is.

END TUTORIALS AND BEGIN PARTICIPANT FEEDBACK SESSION

Bert Still, Discussion Moderator

- This is the first time that Utah has held such a deep dive. They would like to know what we think worked and what did not with respect to format and content.

Hoekstra – We recognize this has taken much time and effort and we really appreciate it.
Olivier – For the hands-on examples – I hope you did not invest too much time because we did not have much time to use them. Hopefully you will be able to reuse those examples when people have more time.
Daniel – But it was very useful to actually be able to look at the code; when we go home we can rerun them.
Nelson - I would like to have the same level of detail for the Uintah framework. I would like to see - if I have to write a task - what is involved in the process. I did not get a full sense of that looking at the code on the screen.
Hoekstra – What do you do for students? One-on-one mentoring?

Berzins – We originally structured this as three separate sessions but you wanted to have one session, so we scrambled to prepare material. My thinking was that we would sit in groups of eight and actually code.
Bergen – If we are going to do this again it would be helpful to get the hands-on details.
Berzins – And that is what I had in mind.

- For those who wanted breadth (versus those who wanted depth) was there a good look at the whole project, so you know where to ask questions? (*Multiple Answers* – yes).

Hoekstra – We probably needed another half day.
Nelson – I would like more hands-on to get more out of the discussion.
Sutherland – To prepare for this it would be helpful to have some specifics beforehand on what you would like to see, and how you would like that structured. We could use a bugtracker and assigned bugs (~ hack-a-thon).
Hoekstra – We needed to hear the overview first before we can give you those types of details.
Karlin – There are mini apps we now could work with you to put in Nebo or Uintah (whereas we did not know enough to do that before).
Hornung – This project has been around for some time. It would be interesting to have you enumerate design decisions you have made and explain what led you to certain choices along the way, as architectures or problems evolved.
Berzins – I have one slide on that - we started with a static task graph and then MPI – I can remember looking at performance and that is interesting because what we have done is driven by what we see on machines. If it works to a certain level then we are OK but when the machine changes what happens that can lead to things like out-of-order execution tasks. We made design decisions based on scaling, for example, that enabled other things. It a practical evolutionary approach driven by experiments at scale. You could argue against that to say that at some point you have to start again and we have thrown away several runtime systems (what we have today is unrecognizable compared to what we had originally). That design process has been clear as we have gone through this. Dynamic execution, out-of-order, different memory model, and different architectures for accelerators have been the main reasons for assessing what was stopping us at each level and then adapting so each level would not stop us any longer. You may go through the same process. I am concerned about fear of computing at scales (I refer to this as petaphobia and exaphobia). As we are pushing our science we also work on infrastructure so we can make science happen.
Sandia – I am impressed with the evidence of technical depth – you called on multiple people in the room to answer our questions.

- This work represents a lot of time on the part of many people.

Comment – It is important that you have contact information for people here at Utah so we can work further with you.
Sutherland – We should circulate a contact list of all attendees.

- Yes, that will provide information now.

Sutherland – Should future sessions focus just on runtime, just on DSL or do we bring in a bigger group and run parallel sessions?
Clay – Look at the mini apps to see how far you can get with those. A question some of us are asking is how usable this could be for us in the long term.
Karlin – Send a student to us for the summer to document what works and does not work for the mini apps.
Clay – Jeanine is doing that at Sandia now. What was your core design philosophy? What limited you? What decisions did you make about domain decomposition? Did you ever go back? I think you did an excellent job at this meeting - but we need to understand those questions. If we cannot do unstructured then it has limited value to Sandia in the long term.
Hornung – We have large integrated codes and everything is integrated within your framework. Is it possible to explore having a piece of our app in this framework?

Olivier – How modular is your infrastructure? Could your walkers be developed outside the DSL?
Berzins – It is a good question. There are many efforts right now to develop runtime systems.
Sandia – Argo is the Sandia runtime effort.
Berzins – So, should we dive in and compete with the big teams trying to do similar things?
Hoekstra – That is an interesting question. You now have a concrete set of challenges you are trying to solve that keep you focused, and we would represent a totally new customer base.

- This PSAAP Center is like an LDRD project. What they want is research engagement on things like design philosophy rather than production level support. For a small team I am amazed how much they have done and I think there are collaborative things we can work on together.

Berzins – You [ASC Program] control the funding tap and that gives you opportunity to suggest changes . For example if you say our ray tracing should also work on unstructured meshes then we will investigate how to make it work. Perhaps there are ways we could explore more funding opportunities. It depends on where the opportunities are. It is a question of understanding what is important.
Sandia – If there are things we all identify that we want the hardware vendors to do then we have some influence with them. It would be helpful to know that. There is an OS from Sandia called HOBBES – if there are things you need from those layers, we can test them in-house and reach out to the vendors on your behalf.
Sutherland – We often feel insulated from vendor influence because we are a small fish. Your mini apps are very good. Memory bandwidth is important to us. We don't care about FLOP counts. I think the labs are providing a service to us by pushing against the architectural drivers that have been used for the past two decades.
Hoekstra – We are still figuring out how to involve the PSAAP II Centers in co-design.
Karlin – The FF2 projects will be announced shortly and that will lead to a new set of projects.
Sutherland – If there are efforts like that and you can see that our involvement would be beneficial then we are eager to participate. (We are a small team, however and are resource-limited.)
Hoekstra – So far, hack-a-thons have had the most impact on vendor thinking. It would be nice to get PSAAP Center involvement in those.
Karlin – Just talking about the challenges of the code helps to give the vendors perspective (because they are focused on single components and not so much on what will run on the system).
Bergen – Are there issues of confidentiality with FF2 projects?
Sandia – If we have a student working for Sandia they will be covered via Sandia NDAs.

- One thing that is coming up is whether we can do NDAs with universities. That would involve lawyers but it might be possible. The vendors want the input but they do not want to risk their IP.

Hoekstra – Most PSAAP Centers have some NDAs in place.

- Yes, it means getting the lawyers on both sides to agree.

Berzins – There is a productivity workshop that will take place soon in Bloomington.
Bergen – It would be helpful if you could generalize the Uintah interface so we could use it for task scheduling and managing data (and maybe not through the data warehouse). Perhaps an interface to register data could be considered.
Sutherland – In some sense, Uintah is where SIERRA was ten years ago in terms of being a framework and ecosystem. That work continues to be a major investment on the part of the labs, i.e. to break what was a framework into a toolkit. We have something that is working - and we recognize the value in doing something like that - but we have no resources to accomplish such a task. A well-conceived breakdown of a framework into a toolkit is not a small undertaking.
Sandia – And you cannot do it via an evolutionary process.
Berzins - We have Schlumberger, DOE, and a number of other Uintah users at this time. It is not as heavily used as we would like it to be. I would like it to be a piece of software that is designed to be used more broadly.
Hornung – Do you have design documentation?

Schmidt – We have it for creating new components. Many of our users are using existing components. There are some users looking to implement a new component.
Hornung – We have a culture where anything that is used in production has to be controlled. We try to own everything because we have to be able to rebuild a code version from a specific date, so we maintain packages we can use for that (and they have to match our revision and control system).
Berzins – I understand that and we do that, as well.
Hornung – Is there a way we can better transfer ideas, adopt some of your ideas and put our own twists on them?
Berzins – Certainly, it is a question of what you want to do. The code used for Uintah is MIT open-source so you can do anything you want with it (and we want people to use it).
Hoekstra – I think Rich is asking what the optimal way is for us to collaborate.
Berzins – Some of the choices we made (e.g. OpenMP versus pthreads) were made because we really did not have a choice.
Sutherland – Collaborating with one or many staff members at labs to transfer technologies in both directions would be ideal to me. The outcome could be products that are more useful to us and the labs. We would be less enthusiastic about dumping technology over the fence to you.
Berzins – That is not true collaboration. SCI [The Scientific Computing and Imaging Institute at Utah] is built on the notion that we collaborate, and that concept runs very deep. We do not function as an academic silo. I think the most helpful thing we can do for you is to show you some things are possible that you otherwise might not explore yourselves. DHARMA, Kokkos and other efforts – along with the visualization and I/O (where there is some active collaboration already via SciDAC) – there has to be a notion of how we can get some value, as well.
Clay – The partnering you are describing is natural to the labs. We have small teams, too. I think we should share across the labs to the greatest extent we can. Right now I would characterize some of what we are doing as exploratory so we can assess where to make longer-term investments. Ultimately, the labs are going to have to down-select. We have multi-billion dollar investments in codes and - at some point - we are likely looking at complete rewrites. We have to assess the cost and benefit of making changes both big and small. There are possibilities for interactions. I think you represent a team in a great position to generate requirements for OS teams. What is right below your software stack that you would like to see exist, be modified or otherwise change? I think it would be desirable to talk about interchangeable modules at some point. That can be easy if things are extractable. We are developing open-source code, too and we may have bite-sized pieces that can be useful to you. What about your interactions with other universities?
Berzins – We follow Charm++ and a number of other efforts.
Clay – Jeanine and I talked about scheduling a BOF at SC14.
Berzins – And I would love to participate in that.
Schmidt – Steve Parker (who originally put Uintah together) was part of the Common Component Architecture (CCA) group. I think the hope was (back when Uintah was part of CSAFE) to get Uintah into a space where others could use it - but it never happened. Perhaps there are lessons learned that could be applied to an integration effort now to enable useful technologies to propagate through different areas.
Clay – I was running a parallel group to CCA (a standards body for solvers). There was a great vision with respect to wanting to create shareable components - but it was premature given the state of technology at the time. Standards live on and tend to survive the test of time whereas we plow under and rewrite our own code every 3-5 years. I think it would be premature to try and do a standards effort now. Perhaps you should keep an eye on ongoing efforts.
Karlin – It is premature to standardize things before you know what you need. This is happening now with MPI and runtime approaches.
Clay – I agree but I think the dialogue and the process of working on that has some utility. Some of the things we learned earlier are manifest in today's code (e.g. Trilinos). I agree that trying to bake

things in too soon means you likely will need to break it apart later - but having the dialogue is productive.
Berzins – There are things that are being discussed now that could change the nature of the dialogue (e.g. ECI) we are having now.
Clay – I think you have done spectacularly nice work here. The job of going from a successful, scalable scientific code to highly componentized, shareable (at the component level) modules would be twice your effort to date. I think it is time for us to have a broader discussion and try to do some things on a collaborative basis.
Berzins – We have to have a different level of abstractions to deal with different meshes. Getting some funding to make that happen would be important.
Clay – I think there may be some funding available for that. I will also say that your team should not be diverted in a way that puts your PSAAP work at risk. The fact that you have made the progress you have showing this can work at scale on real problems is huge. Only within the past year have I seen a groundswell of acceptance that MPI (or even MPI+X) is not going to continue to work in the way we need them to. I would not divert significant PSAAP resources to address DOE lab needs if they do not align well with your core thrusts.
Berzins – I understand that but I also know that we grow and evolve when we take on new challenges. (We will continue to work on our PSAAP commitments as well.)
Olivier – I would like to talk further during lunch about the graph work.
Clay – Jeanine is leading the Sandia project to look into options for the Sandia problem space. There is a collaboration that can start immediately.
Bergen – On making more Uintah more stand-alone, right now it is too much of a framework (compared to Legion) for other things to be designed on top of it. It is important to be able to support unstructured task data; Legion has unstructured support, conceptually. Your team has an advantage because you have done this at scale and you have experience running large scale simulations. That could give you a competitive edge.
Berzins – Perhaps we should look at whether there are abstractions that we could use with respect to data dependency and assess how much we could do without incurring additional cost.
Hornung – We abstract out MPI calls, as an example. You could imagine enabling some code to put its MPI communication layer into your product and that would facilitate the unstructured mesh capability.
Olivier – Going through abstractions is an excellent basis for collaboration (we are doing that now ourselves).
Bennett – I think we will have feedback for you as we complete our performance analysis; you can get it for the full array of programming models we are looking at. Legion has one large scale example (ALE3D) - but that is a specialized implementation.
Berzins – There seems to be a lot of interest in STAR PU (Barcelona) – Eric Darve at Stanford is the POC who is using it.
Hoekstra – The LSU version of HPX looks interesting with respect to one-sided communication, asynchronous, small message passing.
Sandia – Your work is very impressive.

END OF PROGRAMMING MODELS DEEP DIVE.