Preliminary Experiences with the Uintah Framework on Intel Xeon Phi and Stampede

Qingyu Meng, Alan Humphrey, John Schmidt, Martin Berzins

UUSCI-2013-002

Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT 84112 USA

March 14, 2013

Abstract:

In this work, we describe our preliminary experiences on the Stampede system in the context of the Uintah Computational Framework. Uintah was developed to provide an environment for solving a broad class of fluid-structure interaction problems on structured adaptive grids. Uintah uses a combination of fluid-flow solvers and particle-based methods, together with a novel asynchronous taskbased approach and fully automated load balancing. While we have designed scalable Uintah runtime systems for large CPU core counts, the emergence of heterogeneous systems presents considerable challenges in terms of effectively utilizing additional on-node accelerators and co-processors, deep memory hierarchies, as well as managing multiple levels of parallelism. Our recent work has addressed the emergence of heterogeneous CPU/GPU systems with the design of a Unified heterogeneous runtime system, enabling Uintah to fully exploit these architectures with support for asynchronous, out-of-order scheduling of both CPU and GPU computational tasks. Using this design, Uintah has run at full scale on the Keeneland System and TitanDev. With the release of the Intel Xeon Phi co-processor and the recent availability of the Stampede system, we show that Uintah may be modified to utilize such a coprocessor based system. We also explore the different usage models provided by the Xeon Phi with the aim of understanding portability of a general purpose framework like Uintah to this architecture. These usage models range from the pragma based offload model to the more complex symmetric model, utilizing all co-processor and host CPU cores simultaneously. We provide preliminary results of the various usage models for a challenging adaptive mesh refinement problem, as well as a detailed account of our experience adapting Uintah to run on the Stampede system. Our conclusion is that while the Stampede system is easy to use, obtaining high performance from the Xeon Phi co-processors requires a substantial but different investment to that needed for GPU-based systems.

Preliminary Experiences with the Uintah Framework on Intel Xeon Phi and Stampede

Qingyu Meng
Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT 84112 USA
qymeng@sci.utah.edu

Alan Humphrey
Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT 84112 USA
ahumphrey@sci.utah.edu

John Schmidt
Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT 84112 USA
jas@sci.utah.edu

Martin Berzins
Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT 84112 USA
mb@sci.utah.edu

ABSTRACT

In this work, we describe our preliminary experiences on the Stampede system in the context of the Uintah Computational Framework. Uintah was developed to provide an environment for solving a broad class of fluid-structure interaction problems on structured adaptive grids. Uintah uses a combination of fluid-flow solvers and particle-based methods, together with a novel asynchronous task-based approach and fully automated load balancing. While we have designed scalable Uintah runtime systems for large CPU core counts, the emergence of heterogeneous systems presents considerable challenges in terms of effectively utilizing additional on-node accelerators and co-processors, deep memory hierarchies, as well as managing multiple levels of parallelism. Our recent work has addressed the emergence of heterogeneous CPU/GPU systems with the design of a Unified heterogeneous runtime system, enabling Uintah to fully exploit these architectures with support for asynchronous, out-of-order scheduling of both CPU and GPU computational tasks. Using this design, Uintah has run at full scale on the Keeneland System and TitanDev. With the release of the Intel Xeon Phi co-processor and the recent availability of the Stampede system, we show that Uintah may be modified to utilize such a co-processor based system. We also explore the different usage models provided by the Xeon Phi with the aim of understanding portability of a general purpose framework like Uintah to this architecture. These usage models range from the pragma based offload model to the more complex symmetric model, utilizing all co-processor and host CPU cores simultaneously. We provide preliminary results of the various usage models for a challenging adaptive mesh refinement problem, as well as a detailed account of our experience adapting Uintah to run on the Stampede system. Our conclusion is that while the Stampede system is easy to use, obtaining high performance from the Xeon Phi co-processors requires a substantial but different investment to that needed for GPU-based systems.

Categories and Subject Descriptors

D.1.3 [**Software**]: Concurrent Programming; G.1.8 [**Mathematics of Computing**]: Partial Differential Equations; G.4 [**Mathematics of Computing**]: Mathematical Software; J.2 [**Computer Applications**]: Physical Sciences and Engineering

General Terms

Design, Performance, Algorithms

Keywords

Uintah, hybrid parallelism, scalability, parallel, adaptive, MIC, Xeon Phi, heterogeneous systems, Stampede, co-processor

1. INTRODUCTION

For the growing number of problems where experiments are impossible, dangerous, or inordinately costly, extreme-scale computing will enable the solution of vastly more accurate predictive models and the analysis of massive quantities of data, producing significant advances in areas of science and technology that contribute to the mission of agencies such as NSF and DOE [2]. It is through compute resources such as those provided by these agencies that we carry out the petascale simulations of today, advancing science and working toward designing software framework architectures to solve problems at massive scale on next-generation systems. Individual processing units consisting solely of CPU's are no longer increasing in speed from generation to generation, yet the demands on system architects for increased density and power efficiency steadily increase. With these demands in mind, traditional systems are now augmented with an increasing number of graphics processing units or co-processors such as the Intel Xeon Phi. This architectural trend is most notable in machines such as the XSEDE

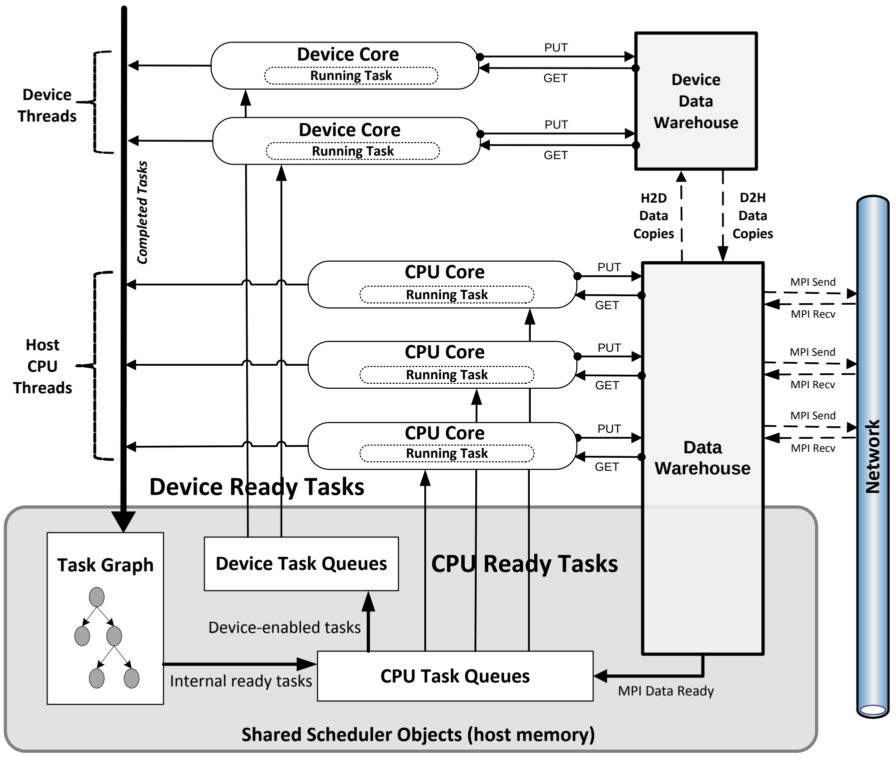


Figure 1: Unified Scheduler and Runtime System with Xeon Phi support

resources Keeneland[1], and Stampede[2]. This trend is also found in the DOE Titan system[3] with its large GPU counts.

Future compute nodes may have hundreds or thousands of cores combined with accelerators or other co-processor designs, and extreme-scale systems could potentially have up to a billion parallel threads of control [2]. Such frameworks must address the formidable scalability and performance challenges associated with running on these systems. For Uintah [4], the challenges of running on accelerator and co-processor based systems begins with developing novel ways to hide latency across deep memory hierarchies, as well as methods of managing multiple levels of parallelism. This must all be accomplished in a manner that insulates the application developer from the inherent complexities involved with programming these systems. Uintah is novel in its use of an asynchronous, task-based paradigm, with complete isolation of the application developer from the parallelism provided by the framework. This approach allows computation to be expressed as individual tasks with input and output data dependencies. The individual tasks are viewed as part of a directed acyclic graph (DAG) and are automatically mapped onto the parallel machine and executed adaptively, asynchronously and often out of order [15].

[1]Keeneland is a hybrid CPU/GPGPU cluster administered by NICS with 264 HP SL250G8 compute nodes, each with 32GB memory, (2) Intel Xeon E5 (8-core Sandy Bridge) processors, and (3) NVIDIA M2090 GPU accelerators [8].

[2]Stampede is a Dell PowerEdge C8220 cluster, administered by TACC with 6,400+ Dell PowerEdge server nodes, each with 32GB memory, (2) Intel Xeon E5 (8-core Sandy Bridge) processors and an Intel Xeon Phi Coprocessor (MIC Architecture) [21].

[3]Titan is a DOE supercomputer located at Oak Ridge National Laboratory with 18,688 compute nodes, each of which contains32GB memory, a single 16-core AMD Opteron 6200 Series (Interlagos cores @2.6GHz) processor and a single Tesla K20 GPU, giving 299,008 processing cores and 18,688 GPU accelerators [16].

Prior to the emergence of heterogeneous systems, a fundamental scalability barrier for Uintah was significantly less memory per core as the numbers of cores per socket grew. In order to address this challenge, as recognized by a number of authors [3,17], Uintah moved from a model that only uses MPI to one that employs MPI to communicate between nodes and a shared memory model using Pthreads to map the work onto available cores in a node [13]. This approach led to the development of a multi-threaded MPI runtime system, including a multi-threaded task scheduler that enabled Uintah to show excellent strong and weak scaling up to 196K cores on the DOE Jaguar XT5 system and good initial scaling to 262k cores on the upgraded DOE Jaguar XK6 system [20]. Using this approach reduced Uintah's on-node memory usage by up to 90% [13].

With the arrival of the Keeneland Initial Delivery System (KIDS) and the upgrade path of the DOE Jaguar system to Titan, Uintah's multi-threaded task scheduler and runtime system were further extended to use a combination of MPI, Pthreads and Nvidia CUDA in order to leverage an arbitrary number of on-node GPUs [11]. In preparation for the imminent release of the Intel Xeon Phi co-processor, this design was refined to become the Unified Scheduler and Runtime System (Figure 1, citewolfhpc12), providing Uintah with a unified approach to supporting and scheduling computational tasks on heterogeneous accelerator/co-processor systems. This design maximizes system utilization by simultaneously using all available processing resources on-node with advanced techniques to harness the additional computational power of both accelerators and co-processors.

In this paper we detail our experiences moving Uintah onto the TACC Stampede system with its Intel Xeon Phi co-processors using the Uintah Unified Scheduler and Runtime System to support, schedule and execute both host CPU and co-processor tasks simultaneously. Throughout this paper, we refer to the **Intel Xeon Phi Coprocessor (MIC Architecture)** as **Xeon Phi** when referring to the co-processor in general, and **MIC** when we talk specifically about the architecture of the Xeon Phi. We explore the various usage models provided by the Xeon Phi with a key aim of understanding the portability of a general purpose framework such as Uintah on such an architecture. Although the Xeon Phi symmetric model is given focus in this work, as it best fits the current Uintah model, our work here clearly illustrates the Directed Acyclic Graph or DAG [5] approach used by Uintah provides the ability to leverage all usage models provided by the Xeon Phi. Ultimately, we provide results from computational experiments using the host-only, native and symmetric models using two challenging computational simulations, one being an incompressible flow calculation (host only) and the other a fluid-structure interaction problem (native and symmetric models) with adaptive mesh refinement (AMR).

In what follows Section 2 provides an overview of the Uintah software, while Section 3 describes the Stampede system and the Intel Xeon Phi co-processor design. Section 4 briefly describes the host-only compute model, and provides scaling results of a production run simulating a helium plume (turbulent reacting flow). Section 4 also compares these results running the same problem on NSF Kraken. Section 5 examines Uintah running the benchmark AMR problem using the Xeon Phi native model. In Section 6 we expose unexpected challenges with the offload model and propose a design solution using techniques in [11]. This solution is left as future work. We concentrate on the Xeon Phi symmetric model in Section 7, as this approach best fits the current Uintah runtime. Here we give scaling results over a range of host and Xeon Phi core counts and provide a subtle floating point accuracy issue encountered. The paper concludes in Section 8 with future work in this area.

2. UINTAH OVERVIEW

The Uintah Software was originally written as part of the University of Utah Center for the Simulation of Accidental Fires and Explosions (C-SAFE) [7]. C-SAFE, a Department of Energy ASC center, focused on providing science-based tools for the numerical simulation of accidental fires and explosions. The aim of Uintah was to be able to solve complex multi-scale, multi-physics problems.

Uintah may be viewed as a sophisticated computational framework that can integrate multiple simulation components, analyze the dependencies and communication patterns between them, and execute the resulting multi-physics simulation [18]. Uintah components are implemented as C++ classes that follow a very simple interface to establish connections with other components in the system. Uintah utilizes an abstract representation (called a task-graph) of parallel computation and communication to express data dependencies between multiple physics components. The task-graph is a directed acyclic graph (DAG) of tasks. Each task consumes some input and produces some output (which is in turn the input of some future task). These inputs and outputs are specified for each patch in a structured AMR grid. Associated with each task is a C++ method which is used to perform the actual computation. Each component specifies a list of tasks to be performed and the data dependencies between them [4].

This component approach allows the application developer to only be concerned with solving the partial differential equations on a local set of block-structured adaptive meshes, without worrying about explicit message passing calls, or notions of parallelization. The Uintah infrastructure even performs automatic load balancing. This approach also allows the developers of the underlying parallel infrastructure to focus on scalability concerns such as load balancing, task (component) scheduling, communications, including accelerator or co-processor interaction. This component-based approach also allows improvements in scalability to be immediately applied to applications without any additional work by the application developer.

Uintah currently contains four main simulation algorithms, or components: the ICE compressible multi-material Computational Fluid Dynamics (CFD) formulation, the particle-based Material Point Method (MPM) for structural mechanics, the combined fluid-structure interaction algorithm MPMICE [10], and the ARCHES combustion simulation component. Development work is also underway on a new MD component to provide basic Molecular Dynamics (MD) capabilities within Uintah. Uintah is regularly released as open source software [9].

3. STAMPEDE AND XEON PHI ARCHITECTURE

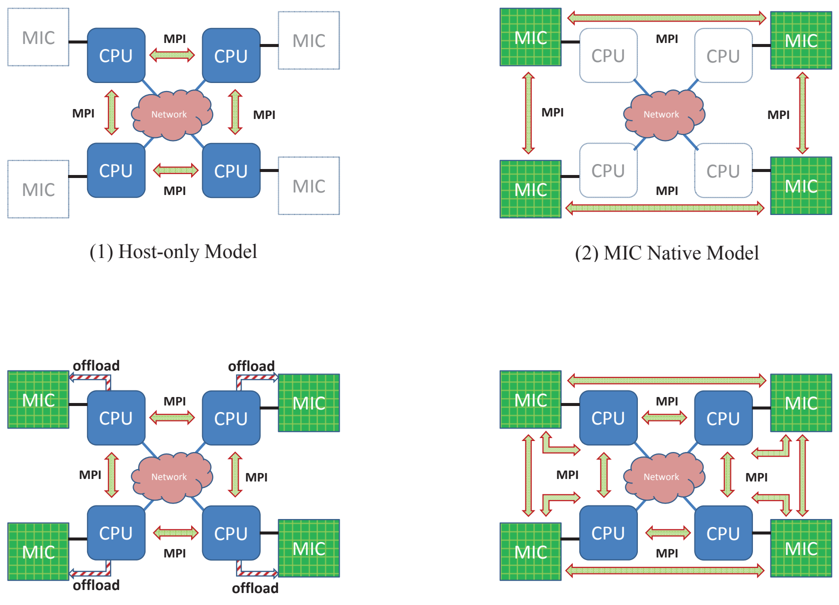
3.1 Stampede

Stampede is the latest, largest, and fastest system (ranking number 7 on the top 500 [1]) that is part of the National Science Foundation's XSEDE program. The Texas Advanced Computing Center (TACC) administers Stampede as well as other resources available under the XSEDE program. Operational since January 7, 2013, Stampede is available to scientists and engineers in all domains of science, as well as offering a research tool in the humanities, digital media and the arts.

Stampede was built by Dell and contains Intel's new co-processor technology, the Xeon Phi. The host processors are an eight core PowerEdge C8220, Xeon E5-2680 operating at 2.7GHz with an Intel Xeon Phi co-processor operating at 1.0GHz. Each compute node has two eight core sockets with 32 GBytes of memory. Stampede is outfitted with $6,400$ compute nodes and $102,400$ cores providing greater than 2 PFlops for the compute cluster and greater than 7 PFlops for the co-processors). The total system memory is 205 TB with over 14 PBytes of shared disk space using the Lustre file system. The system components are connected via a fat-tree FDR InfiniBand interconnect. SLURM (Simple Linux Utility for Resource Management) is used for job submission and scheduling. The operating system is the CentOS Linux distribution [6].

3.2 Xeon Phi Architecture

Stampede provides five programming models: Host-only, MIC native, offload, reverse offload and symmetric. Our focus will be on the four models in Figure 2, and will not cover the reverse offload model, as it is not yet supported by the Intel MPI implementation. In the host-only model, programs run only on host CPUs in the system without any utilization of the Xeon Phi co-processors. Host processors between multiple nodes can communicate though MPI. This model is similar to running on most other CPU-only clusters. The Xeon Phi native model uses only the Xeon Phi co-processors in the system, disregarding the host CPUs. On a Xeon phi card, a very basic version of Linux is installed. After being compiled to MIC binary, a program can then run on the Xeon Phi directly and can use using MPI and OpenMP/Pthreads. The offload model is similar to using accelerators such as a GPU (in conjunction with OpenACC [12]), where the program runs on host CPU and uses offload directives to run certain parts of the computation on Xeon Phi. In this model, all MPI messages are sent and received by host processor. Reverse offload is similar though to offload mode in that the offload region simply runs on host CPU while MPI ranks are run on



(1) Host-only Model

(2) MIC Native Model

(3) Offload Model

(4) Symmetric Model

Figure 2: Xeon Phi Execution Models

the Xeon Phi. For the symmetric model, programs can run on both the host CPU and the Xeon Phi co-processor card natively. MPI messages can be processed by host CPU and Xeon Phi directly.

There are two MPI libraries available on Stampede, Intel MPI and MVAPICH. MVAPICH does not yet have a build for Xeon Phi, but host-only and offload models are supported at this time. Intel MPI has both host and MIC builds and supports four MPI communication modes besides host only:

1. within a single Xeon Phi co-processor,
2. between the Xeon Phi co-processor and the host CPU inside one node,
3. between multiple Xeon Phi co-processors inside one node,
4. between the Xeon Phi co-processors and the host CPU's between several nodes.

4. HOST-ONLY MODEL

In this section a host-only calculation is described. In preparation for the simulations, the standard compiler chain (Intel C++,C, and Fortran) was used to build Uintah as well as the hypre linear solver library. The MVAPICH MPI library was used for all runs, since the Intel MPI version had issues working beyond 2048 cores. The hypre (version 2.8.0b) linear solver package was built with the *-no-global-partition* option using the Intel C compiler.

Uintah has several CFD algorithms that are under active development that are used both in production mode and for bench-marking and performance analysis of new systems. The ARCHES CFD component is an implementation of a three-dimensional, Large Eddy Simulation (LES) algorithm which uses a low-Mach (Ma$<$ 0.3), variable density formulation of the Navier-Stokes equations to simulation heat, mass and momentum transport in reacting flows. The set of filtered equations is discretized in space and time and solved on a staggered finite volume mesh. Flux limiters are used to avoid nonphysical solutions. The low-Mach, pressure formulation requires a solution of an implicit pressure projection at every time sub-step. Various linear solver packages including PETSc and hypre have been used for the solution of these equations. A dynamic large eddy turbulence closure model for momentum and species transport equations is used to account for sub-grid velocity and species fluctuations. Various combustion models exist for doing gas phase and particle phase combustion chemistry. The energy balance includes the effects of radiative heat-loss/gains in the IR spectra by solving the radiative intensity equations using a discrete-ordinance solver. The formulation of the intensity equations at discrete ordinances results in a system of linear equations that are solved using hypre. The solid particulate fuel phases are represented using the direct quadrature method of moments (DQMOM). DQMOM is completely coupled to the the gas phase description resulting in a closed mass, momentum, and energy balances [19].

While the ARCHES finite-volume component is essentially a stencil-based p.d.e code, the implicit formulation of the pressure projection and the concomitant requirement of a linear solve at each time step is the potential bottleneck for achieving scalability at large core counts. Recent results [19] suggest good weak scalability for incompressible flow calculations on other large core count systems.

The illustrative incompressible flow calculation using the ARCHES CFD component is a Helium plume problem, which requires the full solution of the Navier-Stokes equations including density variations. In addition, various sub-models are used to account for any unresolved turbulence scales that are not directly resolved by the computational mesh. The helium plume represents the essential characteristics of a real fire without introducing the full complexities of combustion and thus serves as an important validation problem for the ARCHES code.

The computational scenario consists of a $3m^3$ domain with a $1m$ opening that introduces the helium into a quiescent atmosphere of air with a co-flow of air. Velocity and density conditions at these boundaries are known. The sides and top of the computational cube are modeled using pressure and outlet boundary conditions respectively. The CFD solution procedure exercises major components of the overall ARCHES algorithm, including the modeling of small, sub-grid turbulence scales. Additionally, the coupled problem combines the effects of fluid flow and turbulent scalar mixing for a full spectrum of length and time scales without introducing the complications of combustion reactions.

The overall scalability of the ARCHES algorithm is dictated by the scalability of the linear solver package. Through a judicious choice of solver parameters available in the hypre package, good weak scalability was achieved as in [19]. For the results presented below in 3, a non-symmetric red black Gauss Seidel preconditioner was used while skipping levels during the multi-grid solves. Skipping levels reduces the overall solve time by using interpolated results at the "skipped" levels. The grid resolution of the problem was adjusted such that each core had an equal amount of work. Owing to the number of cores per node for Kraken and Stampede, the actual breakdown of the work load was slightly different, i.e. 330K unknowns/core (Kraken) versus 335K unknowns/core (Stampede).

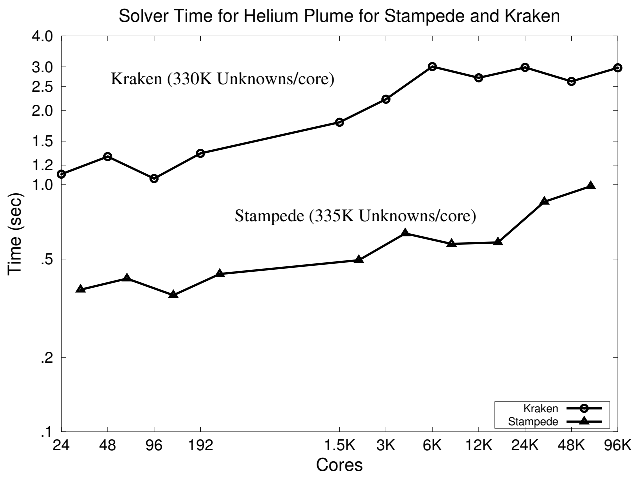


Figure 3: Solver time for the Helium Plume for Stampede and Kraken.

The results show that Stampede is roughly three times faster on a per core basis than Kraken.

5. NATIVE MODEL

As the Intel Xeon Phi is based on X86 technology, porting existing code to the Xeon Phi is relatively easy. Most codes, including Uintah, can be compiled to run on the Xeon Phi by simply adding the *-mmic* compiler flag. The Uintah framework infrastructure code and most of its simulation components are written in C++, with some legacy components written in Fortran. Both C++ and Fortran are supported by the Intel compiler for the MIC architecture. The parallel programming libraries used by Uintah, MPI and Pthreads are also supported natively. However, Uintah depends on many third party libraries such as *libxml2* and *zlib*. Those libraries are not currently installed on the Xeon Phi and needed to be built. To get both Uintah and the other libraries built, cross compiling is required, as the binaries compiled with the *-mmic* compiler flag cannot run on the head nodes of Stampede. As Uintah uses autotools for its build system, only minor changes were made to support cross compiling. We were able to get a native Uintah build up and running on a single Xeon Phi card within 24 hours of having access to the machine.

When running on a single Xeon Phi card, Uintah uses both MPI and Pthreads for parallelization. When running with Pthreads on a shared memory node, Uintah also uses lock-free data structures to allow concurrent access to shared object such as the Data Warehouse

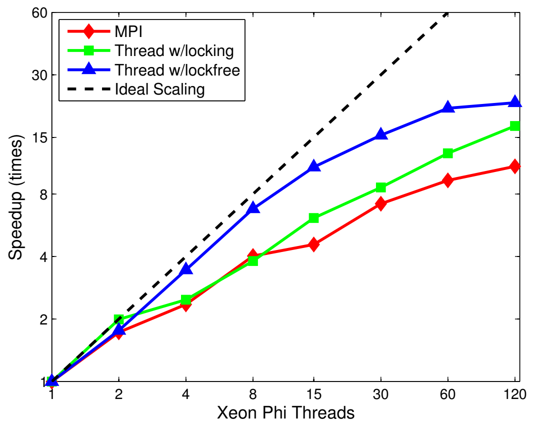


Figure 4: Strong scaling of Uintah AMR MPMICE simulation on single Xeon Phi card (MPI, Pthreads and Pthreads with lock-free data structures)

(a simulation variable repository) without using high-level and typically high-overhead Pthread read-write locks. This lock-free Data Warehouse uses built-in atomic operations that are supported in the gcc compiler such as *fetch_and_add* and *compare_and_swap*. Those gcc built-ins are not supported in earlier versions of the Intel compiler. However, this issue has been solved by using equivalent atomic operations in older Intel compilers or by using the newer Intel compiler. Figure 4 shows strong scaling results of the Uintah AMR MPMICE simulation on a single Xeon Phi card comparing pure MPI, Pthreads with read-write locks and Pthreads with lock-free data structures. Two MPI ranks or Pthreads per Xeon Phi core are used for this benchmark. These results show that Uintah performs and scales better when using a combination of MPI and Pthreads as opposed to an MPI-only approach.

6. OFFLOAD MODEL

Although the directive-based approach, using the Xeon Phi synchronous offload model seems the most attractive to use initially, we discovered this model is more difficult to implement than we originally anticipated for a general purpose framework like Uintah. In order to use this pragma-based offload model, all functions called from the Xeon Phi must be defined with the offload attribute:

```
__target(mic)
```

Due to the complexity of the heavily templated Uintah code, we essentially need to define almost everything with this attribute or rewrite a particular task with a simple C/C++ structure, avoiding the complexities of the infrastructure code. For Uintah to make effective use of this model, the Xeon Phi asynchronous offload features must be used. These features include:

1. asynchronous data transfer
2. asynchronous compute
3. memory management without data transfer.

Using these asynchronous API offerings, PCIe latency can be hidden by overlapping MPI communication with computation on both the host CPU and the Xeon Phi co-processor. The key component in making this work is to implement a mechanism to detect completion of the asynchronous data copies to-and-from the co-processor.

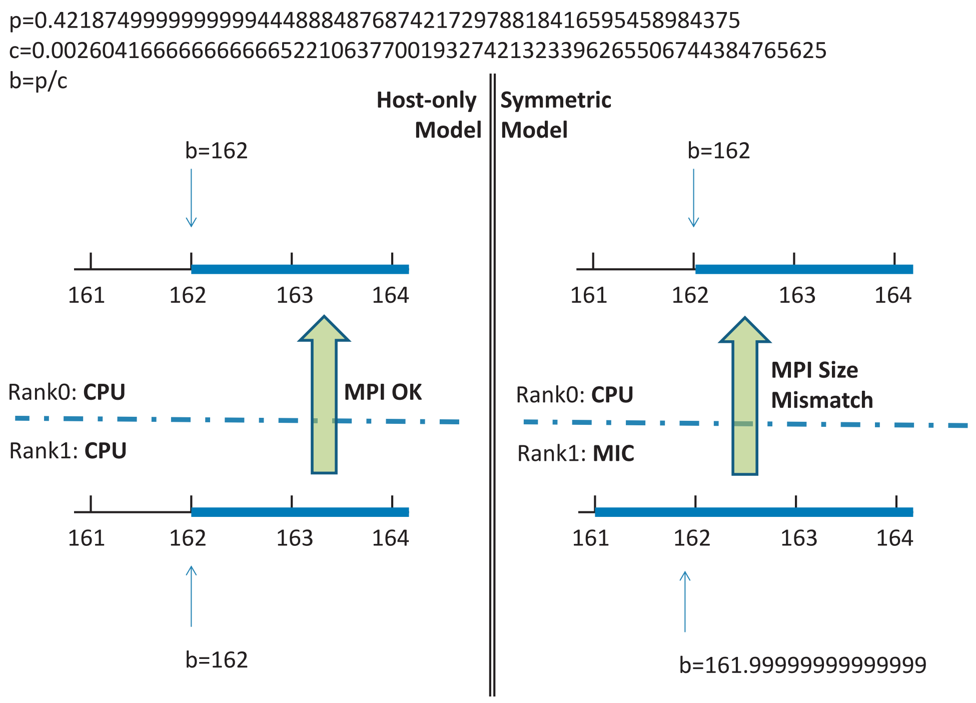


Figure 5: MPI error due to a floating point inconsistency

This approach is nearly a perfect analog to the mechanism created in [11] to orchestrate and manage asynchronous data copies to-and-from on-node GPUs. In the context of the Xeon Phi asynchronous offload model, an offload region can be executed asynchronously when a signal clause is included with the directive. All asynchronously offloaded data and computation can be associated with this signal clause. Detecting completion of this operation is achieved with explicit API calls. For example, the API call:

```
_Offload_signaled(mic_no, &c)
```

tests whether the computation signaled with *c* has finished. This is a non-blocking mechanism to check if offload has been completed.

Using the Xeon Phi asynchronous offload features, we simply generalize the existing *GPU* task queues to become *device* task queues and add the associated logic to the Unified Scheduler and Runtime System from [14] to become what was shown in Figure 1 of Section 1. This implementation is currently underway and testing it is part of our future work.

7. SYMMETRIC MODEL

Uintah's directed acyclic graph (DAG) based runtime system allows full utilization of all available cores on the host CPU and Xeon Phi co-processors easily through the symmetric programming model. The simulation grid in Uintah is partitioned into hexahedral patches by a highly scalable regridder and assigned to nodes by a measurement-based load-balancer [4]. In each MPI process, the Uintah runtime system will schedule tasks on local patches by using a local task graph and data warehouse. The task graph is a DAG [5] which is compiled by making connections on task's required and computed variables. The Uintah scheduler uses the task graph to determine the order of execution, assign tasks to local computing resources and ensure that the correct inter-process communication is performed. Uintah use Pthreads for intra-node task scheduling. Each core directly pulls tasks from multi-stage ready-task queues without any intra-node communications taking place. This runtime system is shown to fully use all available cores on-node, regardless of the number of cores.

When running with the Xeon Phi symmetric model, two binaries are required, one for host CPU(s) and one for Xeon Phi co-processor. Since the Xeon Phi has significantly more cores than the host CPU, more threads are created in MPI ranks running on the Xeon Phi than MPI ranks running on the host CPU. In a typical Uintah run, we create 120 threads per Xeon Phi and 16 threads (one per core) for the host CPU(s). For example, to run symmetric mode, we used the following command line:

```
mpirun.hydra -n 4 ./sus -nthreads 16 input.ups;
-n 4 ./sus-mic -nthreads 120 input.ups
```

This will run Uintah on 4 CPU hosts with 16 threads per host and 4 Xeon Phi cards with 120 threads per card at the same time.

With some MPI ranks running on one architecture while other MPI ranks run on a different architecture, it is important to make sure that all ranks execute in a consistent way. Errors may happen when control logic based results differ between the Xeon Phi and host CPU, such as MPI messages based on floating point calculations. In Uintah, a common operation when running with AMR is to find cells in a finer level based on a point that is computed from coarser level, which are then sent from the finer level cells to coarser level. Figure 5 shows a real AMR example in Uintah, in which a point is computed by the division of two double precision numbers that are known globally to all MPI ranks. The algorithm guaranteed that all ranks should compute this point as the same value such that the sending side will pick the same interval of cells as the receiving side (left side: host-only model). However, while the algorithm is consistent, when one rank runs on the Xeon Phi, the computed value may be inconsistent. In this example, the CPU side receiver picks intervals beginning with 162 however the Xeon Phi sender picks interval beginning with 161. Hence, an MPI buffer mismatch error occurs due to a floating point operation that

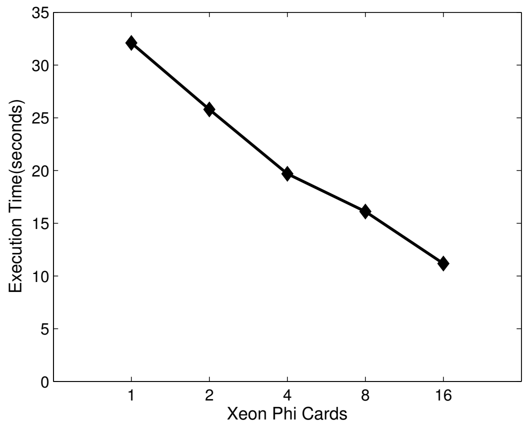


Figure 6: Strong scaling of Uintah AMR MPMICE simulation on multiple Xeon Phi cards using symmetric model

is not consistent between the Xeon Phi co-processor and host CPU (right side: symmetric model). To fix this error, a higher precision compiler flag was used at the cost of lower performance for this method.

Figure 6 shows preliminary scaling results on Stampede with multiple Xeon Phi cards and host nodes using the symmetric model. Usign this model, Uintah can strong scale up to 16 Xeon Phi cards (the current Stampede MIC development queue limit), however the scaling efficiency is limited due to load imbalance between host CPU and Xeon Phi. The reason being that the Uintah load balancer currently assigns host MPI ranks and Xeon Phi ranks the same workload. We detected a load imbalance up to 60% for this benchmark. The workload ratio of CPU to Xeon Phi should be computed based on profiling. We will develop a new load balancer to profile and predict the work load on the host CPU and Xeon Phi card separately to solve this problem.

8. CONCLUSIONS AND FUTURE WORK

We have described our preliminary experiences with Stampede using the Uintah Computational Framework with an emphasis on understanding the performance implications of the new Intel Xeon Phi Coprocessor (MIC Architecture). Using only the host CPUs for computations, Stampede is nearly 3X faster than Kraken for a complex reacting flow CFD calculation. The Uintah architecture has a runtime environment which has been shown to be highly adaptable to the heterogeneous architectures that are emerging in the high performance computing world [11, 14]. This adaptability has allowed Uintah to utilize the range of usage models provided by the Xeon Phi. Of these usage models, we found the symmetric model to best fit Uintah, and required only very small modifications to the Uintah runtime system to use both the host CPUs and Xeon Phi together. Using the Xeon Phi symmetric model yielded excellent strong scaling characteristics up to 16 Xeon Phi cards (the Stampede MIC development queue limit at the time).

Due to different performance characteristics between the host CPU and the Xeon Phi, our scaling efficiency was limited. This will require us to develop an improved load balancer as part of our future work on Stampede to make efficient use of the Xeon Phi symmetric model. Specifically, the load balancer needs to be updated to distribute a given workload according to which processing unit an MPI process is running on. This will expand the current forecast method to profile the host CPU and Xeon Phi separately, as the Xeon Phi and host CPU have different levels of concurrency. For the Xeon Phi, finer patch sizes should be used to keep the many available threads busy and for the host CPU, larger patches are needed to better utilize the larger cache. This change will require the Uintah regridder to be able to generate different patch sizes based on the target processing unit.

To efficiently use the Xeon Phi asynchronous offload model, work is now underway within the Uintah runtime system to generalize its existing *GPU* task queues to become *device* task queues with associated logic. Using this design, we hope to provide the Uintah framework with an additional way to achieve high performance from the Xeon Phi co-processor.

We have also discovered the necessity in making use of the long vector units available on the Xeon Phi, and will so investigate explicitly using its 512-bit vector instructions as the C++ iterator loops currently used throughout Uintah cannot be easily be optimized automatically by the compiler.

9. ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under subcontracts No. OCI0721659, the NSF OCI PetaApps program, through award OCI 0905068 and by DOE INCITE award CMB015 and CMB021 for time on Jaguar and DOE NETL for funding under NET DE-EE0004449. Uintah was written by the University of Utah's Center for the Simulation of Accidental Fires and Explosions (C-SAFE) and funded by the Department of Energy, subcontract No. B524196. This work used early access to the TACC Stampede system. We would like to especially thank Karl Schulz, Bill Barth, Andy Terrel and the entire TACC team for early access to Stampede. We would also like to thank all those previously involved with Uintah, Steve Parker, Justin Luitjens, Todd Harman, Davison de St. Germain and Joseph Peterson in particular.

10. REFERENCES

[1] Top 500. Top500 Web Page, 2012. http://www.top500.org/list/2012/11/.

[2] S. Ashby, P. Beckman, J. Chen, P. Colella, B. Collins, D. Crawford, J. Dongarra, D. Kothe, P. Messina R. Lusk, P Moin T. Mezzacappa, M. Norman, R. Rosner, V. Sarkar, A. Siegel, F. Streitz, A. White, and M. Wright. Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee. Technical report, U.S. Department of Energy, Office of Science, 2010.

[3] P. Balaji, A. Chan, and R. Thakur E. Lusk W. Gropp. Non-data-communication overheads in MPI: analysis on Blue Gene/P. In *Proc. of the 15th Euro. PVM/MPI Users' Group Meeting on Recent Advances in PVM and MPI*, pages 13–22, Berlin, Heidelberg, 2008. Springer-Verlag.

[4] M. Berzins, J. Luitjens, Q. Meng, T. Harman, C.A. Wight, and J.R. Peterson. Uintah - a scalable framework for hazard analysis. In *TG '10: Proc. of 2010 TeraGrid Conference*, New York, NY, USA, 2010. ACM.

[5] M. Berzins, Q. Meng, J. Schmidt, and J. Sutherland. Dag-based software frameworks for pdes. In *Proceedings of HPSS 2011 (Europar, Bordeaux August, 2011)*, 2012.

[6] Texas Advanced Computing Center. Stampede Web Page, 2013. http://www.tacc.utexas.edu/resources/hpc/stampede.

[7] J. D. de St. Germain, J. McCorquodale, S. G. Parker, and C. R. Johnson. Uintah: A massively parallel problem solving environment. In *Ninth IEEE International Symposium on*

High Performance and Distributed Computing, pages 33–41. IEEE, Piscataway, NJ, nov. 2000.

[8] Extreme Science and Engineering Discovery Environment. Georgia Tech Keeneland User Guide, 2013. https://www.xsede.org/gatech-keeneland.

[9] The Center for the Simulation of Accidental Fires and Explosions. Uintah Web Page, 2012. http://www.uintah.utah.edu/.

[10] J. E. Guilkey, T. B. Harman, and B. Banerjee. An eulerian-lagrangian approach for simulating explosions of energetic devices. *Computers and Structures*, 85:660–674, 2007.

[11] A. Humphrey, Q. Meng, M. Berzins, and T. Harman. Radiation Modeling Using the Uintah Heterogeneous CPU/GPU Runtime System. In *Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment (XSEDE 2012)*. ACM, 2012.

[12] OpenACC member companies, CAPS Enterprise, CRAY Inc, The Portland Group Inc (PGI), and NVIDIA. OpenACC Web Page, 2013. http://www.openacc-standard.org/.

[13] Q. Meng, M. Berzins, and J. Schmidt. Using Hybrid Parallelism to Improve Memory Use in the Uintah Framework. In *Proc. of the 2011 TeraGrid Conference (TG11)*, Salt Lake City, Utah, 2011.

[14] Q. Meng, A. Humphrey, and M. Berzins. The Uintah Framework: A Unified Heterogeneous Task Scheduling and Runtime System. In *Digital Proceedings of The International Conference for High Performance Computing, Networking, Storage and Analysis (SC12) - WOLFHPC Workshop*. ACM, 2012.

[15] Q. Meng, J. Luitjens, and M. Berzins. Dynamic task scheduling for the uintah framework. In *Proceedings of the 3rd IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS10)*, 2010.

[16] U.S. Department of Energy Oak Ridge Natioanl Laboratory and Oak Ridge Leadership Computing Facility. Titan Web Page, 2013. http://www.olcf.ornl.gov/titan/.

[17] C.D. Ott, E. Schnetter, G. Allen, E. Seidel, J. Tao, and B. Zink. A case study for petascale applications in astrophysics: simulating gamma-ray bursts. In *Proc. of the 15th ACM Mardi Gras conference: From lightweight mash-ups to lambda grids.*, MG '08, pages 18:1–18:9, New York, NY, USA, 2008. ACM.

[18] S. G. Parker, J. Guilkey, and T. Harman. A component-based parallel infrastructure for the simulation of fluid-structure interaction. *Engineering with Computers*, 22:277–292, 2006.

[19] J. Schmidt, M. Berzins, J. Thornock, T. Saad, and J. Sutherland. Large Scale Parallel Solution of Incompressible Flow Problems using Uintah and hypre. In *Proceedings of CCGrid 2013*. IEEE/ACM, 2013.

[20] J. Schmidt, J. Thornock, J. Sutherland, and M. Berzins. Large Scale Parallel Solution of Incompressible Flow Problems using Uintah and Hypre. Technical Report UUSCI-2012-002, Scientific Computing and Imaging Institute, 2012.

[21] Texas Advanced Computing Center. Stampede User Guide, 2013. http://www.tacc.utexas.edu/user-services/user-guides/stampede-user-guide.