Large-Scale Compute-Intensive Analysis via a Combined In-Situ and Co-Scheduling Workflow Approach

Christopher Sewell
CCS-7
Los Alamos National Lab
Los Alamos, NM 87545
csewell@lanl.gov

Katrin Heitmann
High Energy Physics
Argonne National Lab
Lemont, IL 60439
heitmann@anl.gov

Hal Finkel
ALCF
Argonne National Lab
Lemont, IL 60439
hfinkel@anl.gov

George Zagaris
Computing Applications
Lawrence Livermore Nat. Lab
Livermore, CA 94551
zagaris2@llnl.gov

Suzanne T. Parete-Koon
OLCF
Oak Ridge National Lab
Oak Ridge, TN 37831
paretekoonst@ornl.gov

Patricia K. Fasel
CCS-3
Los Alamos National Lab
Los Alamos, NM 87545
pkf@lanl.gov

Adrian Pope
ALCF
Argonne National Lab
Lemont, IL 60439
apope@anl.gov

Nicholas Frontiere
University of Chicago
and
Argonne National Laboratory
nfrontiere@anl.gov

Li-ta Lo
CCS-7
Los Alamos National Lab
Los Alamos, NM 87545
ollie@lanl.gov

Bronson Messer
OLCF
Oak Ridge National Lab
Oak Ridge, TN 37831
bronson@ornl.gov

Salman Habib
High Energy Physics
Argonne National Lab
Lemont, IL 60439
habib@anl.gov

James Ahrens
CCS-7
Los Alamos National Lab
Los Alamos, NM 87545
ahrens@lanl.gov

ABSTRACT

Large-scale simulations can produce hundreds of terabytes to petabytes of data, complicating and limiting the efficiency of workflows. Traditionally, outputs are stored on the file system and analyzed in post-processing. With the rapidly increasing size and complexity of simulations, this approach faces an uncertain future. Trending techniques consist of performing the analysis *in-situ*, utilizing the same resources as the simulation, and/or off-loading subsets of the data to a compute-intensive analysis system. We introduce an analysis framework developed for HACC, a cosmological N-body code, that uses both *in-situ* and co-scheduling approaches for handling petabyte-scale outputs. We compare different analysis set-ups ranging from purely off-line, to purely *in-situ* to *in-situ*/co-scheduling. The analysis routines are implemented using the PISTON/VTK-m framework, allowing a single implementation of an algorithm that simultaneously targets a variety of GPU, multicore, and many-core architectures.

1. INTRODUCTION

The analysis of ever-increasing simulation output sizes poses a major challenge in many areas of scientific computing. Traditionally, the simulation outputs, which can reach hundreds of terabytes to petabytes in supercomputing applications, are written to disk and analyzed off-line. Storing these amounts of data for extended periods on disk for analysis tasks is impossible if the number of simulations of this size and larger rapidly increases. In the future we will therefore be forced to develop alternatives to the off-line analysis approach. One such alternative is to carry out the analysis *in-situ*. This works particularly well if the analysis step is appropriately sized and load-balanced and is able to run on the same large partition of the HPC system as the simulation itself. Other analysis tasks, however, may not meet these requirements and not lend themselves to efficient *in-situ* analysis.

There are a number of reasons why a subset of the analysis component may not be fully suited to an *in-situ* approach. An obvious reason is that these analysis tasks can have strong scaling bottlenecks. These bottlenecks may arise from a number of causes; a typical example is one where the task runs efficiently on the majority of the nodes, but the workload on some nodes may become very heavy, slowing the analysis step down, and destroying the overall scalability of the approach. Off-loading the tasks to other nodes will increase the complexity of the analytics tasks, especially given that a large number of them may have to be performed, each with its own specific (parallel) solution. Such an approach – even in the cases where it applies – leads to problems with portability, maintainability, and code management.

In this paper, "*in-situ*" means that the analysis runs in the same process as the simulation (e.g., a library call), while "off-line" means that the analysis runs in a separate process (either on the same machine as the simulation, or on a different machine), which takes output from the simulation process as its input. In a combined *in-situ* and off-line workflow, some analysis is performed *in-situ* and some

SC '15, November 15 - 20, 2015, Austin, TX, USA

ACM ISBN 978-1-4503-3723-6/15/11…$15.00

DOI: http://dx.doi.org/10.1145/2807591.2807663

off-line. "Co-scheduling" means that analysis that is performed off-line is computed by multiple independent processes, which can be automatically queued on an analysis cluster as the data becomes available, and run while the main simulation is still running on the primary HPC system, rather than as a single off-line process that is queued and run only after the main simulation terminates. "*In-transit*" means that data is transferred from the main simulation to co-scheduled off-line analysis processes through a separate memory device (such as NVRAM) that is shared between the main HPC system and the analysis cluster, rather than by the analysis processes reading files written by the simulation.

This paper presents a comparison of alternative workflows for scientific simulation and analysis, including new ones enabled by emerging *in-situ* and co-scheduling technologies. We implement and compare three different approaches: purely *in-situ*, purely off-line, and combined *in-situ* and off-line. We also discuss possible variations of the last approach, including co-scheduling the off-line analysis, and passing data to the off-line analysis through a shared memory system rather than through file I/O.

The results reported here, as well as the approaches we have devised to portably and efficiently handle a diverse range of analysis tasks, with widely varying data sizes and computational loads, should help inform decisions made for other large-scale simulations as they encounter ever larger data sizes, more heterogeneous computing platforms, and more complex analysis tasks.

We focus our discussion on computational cosmology and related analysis challenges of large N-body simulations, which represent some of the largest simulations carried out on HPC platforms. Two examples in this area are the recently completed Outer Rim simulation [12] that evolved more than a trillion particles on Argonne's IBM BG/Q Mira system and generated approximately 5Pbytes of raw outputs (not including check-point restart files) and the Q Continuum simulation [13] – more than half a trillion particles on Oak Ridge's CPU/GPU Titan system and approximately 2.5Pbytes of data. The very high mass resolution in the Q Continuum simulation of $\sim 10^8 M_\odot$ led to major challenges for the final analysis steps. These challenges motivated a more complex analysis workflow to separate out very compute intensive tasks, discussed below.

An example for an efficient *in-situ* analysis task in cosmology is the determination of the density fluctuation power spectrum. This calculation requires a density estimation on a regular grid via, e.g., a Cloud-In-Cell (CIC) algorithm and very large FFTs. Both of the algorithms are efficiently parallelizable and in the case of the above mentioned simulations, the determination of the power spectrum takes only a few minutes, a small fraction of the computational time required for a single time step. Therefore, the power spectrum was determined at regular intervals as an *in-situ* operation during the full runs. Another analysis example in cosmology is finding matter clumps called halos and evaluating their properties. Halos are regions of high density and are of scientific interest for diverse reasons. They provide information about structure formation as well as galaxy formation and are the basis for building sophisticated synthetic sky catalogs from N-body simulations. The identification of halos via standard cluster-finding algorithms is also efficiently parallelizable, just like the power spectrum calculation. However, the vast range of sizes of halos (in the Q Continuum simulation we found a handful of halos with up to 25 million particles in the late stages of the evolution, while billions of halos with 40 particles were found) poses a problem for more compute-intensive analysis tasks, e.g., finding the center of a halo as given by its gravitational potential minimum. For a 40 particle halo, center finding takes less than a second, while in the case of a halo with 25 million particles, it can take hours, depending on the center definition. Following our strategy, this suggests that the analysis be broken up into two parts: small halos to be analyzed *in-situ* while large halos are sifted out and handed over to another system. A detailed discussion including timing information can be found in the main body of the paper.

More generally, our contention is that the nature of many analytics tasks is suited to a workflow approach that combines an *in-situ* step with an off-line step, which may be co-scheduled. In our case, we first carry out an *in-situ* data analysis step to 1) reduce the amount of data to be analyzed (in our example by a factor of five), and 2) separate the data-intensive analysis tasks to be handled by a different machine (or by a different sector of the same machine). In this way, the data-intensive tasks are handed over to a computational resource that is responsible for the analysis tasks while the main simulation code continues to run. We describe the analysis workflow set up for the Hardware/Hybrid Accelerated Cosmology Code (HACC) and explain our proposed combined *in-situ*/co-scheduling approach.

Finally, a successful analysis strategy requires flexible tools that can run on a variety of architectures. For example, most analysis clusters have a very different architecture compared to the main HPC resource. In order to ensure that our analysis routines run on a variety of architectures, we have implemented them using the PISTON framework. PISTON, a part of the VTK-m project, is built on top of NVIDIA's Thrust library, and allows a single implementation of an algorithm to be compiled to multiple backends, enabling one to target a variety of multi-core and many-core architectures. We have previously demonstrated the performance and portability of PISTON for algorithms such as isosurfacing, KD-tree construction, and dendrogram-based halo finding with variable linking lengths [20, 33, 40]. More broadly, this data-parallel programming model has been used for a wide variety of algorithms in data structures, computational geometry, graphs, and numerical analysis [7].

The overall organization of this paper is as follows. We briefly review related work in the field in Section 2. We then give a general description of our workflow implementation, in particular the *in-situ* and co-scheduling approach, in Section 3. We also provide details about our flexible halo center finding implementation in the same section. Finally, we show results from our implementation in Section 4 and demonstrate the efficiency of our combined *in-situ*/off-line approach. In the future, on new architectures that provide burst-buffer capabilities, we will be well prepared to take full advantage of these set-ups. Instead of writing out the files to be analyzed to disk, we will be able to keep them in memory and off-load them to other compute resources.

2. RELATED WORK

Increases in available computational power are outpacing growth in I/O bandwidth and capacity [32]. The challenge this poses to traditional post-processing visualization and analysis workflows are widely recognized [23]. The need for *in-situ* workflows, in which visualization and analysis products are computed in the same process space as the simulation, has been described in a variety of workshop reports from agencies such as the National Science Foundation and the Department of Energy [3, 17, 25]. Libraries have been developed to make algorithms from popular post-processing tools available directly to the simulation *in-situ*. These include Catalyst [9], based on ParaView, and Libsim [39], based on VisIt. An *in-situ* workflow typically has the advantage of needing to save much less data (such as images or summary statistics) than the raw data dumps required for post-processing workflows. While not saving all the raw data can potentially limit the ability to explore the data in post-processing, traditional workflows also entail the loss

of data (often in the form of decreased time resolution), and *in-situ* tools such as Cinema [4] can provide the user with greater flexibility in making trade-offs among factors such as time, space, and resolution. Additional *in-situ* research has focused on such issues as reducing the memory footprint by sharing data between simulations and visualization libraries with "zero-copy" data structures [42] and automatically evaluating the relative importance of data in order to produce reduced data products [29]. The increased temporal resolution available *in-situ* has been exploited by analysis algorithms such as flow field analysis [2, 44].

In-transit workflows transfer data from a simulation over a network to a separate process that computes visualization and analysis products while the simulation is running. The simulation and analysis processes are co-scheduled on the same machine, or on a separate system. Several frameworks have been developed to provide network and/or storage services that enable *in-transit* coupling of simulations with visualization and analysis processes. Examples of these include Sandia National Laboratory's Nessie (NEtwork Scalable Service Interface) [21, 22] and Argonne National Laboratory's GLEAN [38]. ADIOS (Adaptable I/O System) [1] from Oak Ridge National Laboratory includes support for staging data for analysis before final output to disk. Nessie, GLEAN, and ADIOS are compared and contrasted in Ref. [28]. A detailed study [31] at Sandia compared *in-situ* and *in-transit* workflows using Catalyst and Nessie for a shock physics code. They found their *in-transit* workflow to be slower than *in-situ* except in a few cases involving analysis algorithms that did not scale well. They also noted that schedulers available at the time were generally inadequate for the needs of *in-transit* workflows. Studies have demonstrated the potential value of using solid state memory as a part of an *in-transit* pipeline, computing visualization products on "burst buffer" I/O nodes separate from the simulation nodes and the file system nodes [6]. Task parallel systems take the co-scheduling approach even further, allowing many asynchronous tasks to be scheduled to execute at specified levels of the memory hierarchy [43]. General-purpose task parallel runtimes include Legion [5] from Stanford University and Uintah [26] from the University of Utah, while the idea of a loosely-coupled distributed system of cooperating tasks reaches its natural extension in grid computing [11].

Our initial co-scheduling scheme was derived from the Bellerophon software stack. Bellerophon is a workflow management system developed with the goals of automating data analysis, workflow management, and software engineering tasks for HPC simulation campaigns [18, 19]. It was initially developed and deployed to monitor and analyze, in near real-time, long-running core-collapse supernova simulations with the CHIMERA code [8, 27]. The design of Bellerophon's *n*-tier architecture includes a logic, data, and presentation tier as well as a supercomputing tier. Elements of the Bellerophon supercomputing tier monitor simulation progress, process and analyze results, archive data, and transmit new data to Bellerophon's data server. Several analogous tasks are accomplished by the *in-situ* and co-scheduled components described here.

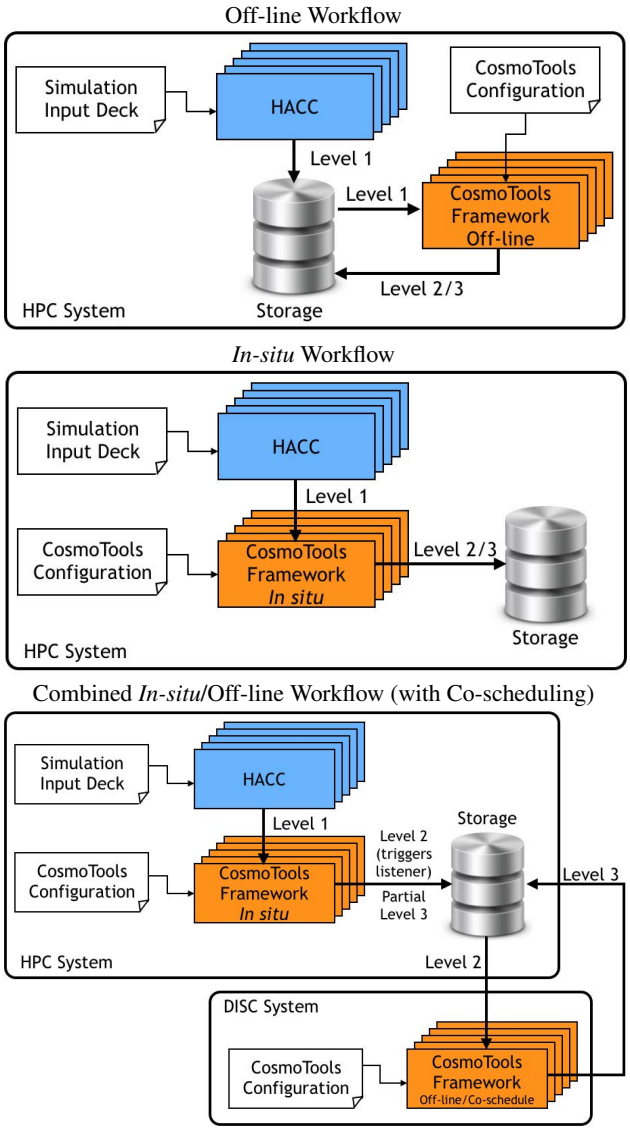

Figure 1: Schematics of the analysis workflows we examined. In the off-line approach, all analysis is computed in a separate process after the simulation has completed, by reading data written to disk by the simulation. In the *in-situ* approach, all analysis is computed in the same process as the simulation, with no I/O required to communicate between them. In the combined approach, one part of the analysis is carried out *in situ*; a subset of the results are stored on either disk or external memory and then read in and analyzed by an off-line process (which may be co-scheduled). Results from both analysis steps are reconciled and a complete analysis output is written to storage.

3. WORKFLOWS AND ALGORITHMS

The simulations and analysis methods presented in this paper use the HACC cosmology code. HACC solves for the evolution of cosmic structure, from very small initial fluctuations on a uniform background, to the highly nonlinear, clustered regime of the present epoch. Once the first bound objects (halos) form, analysis tasks are carried out to not only capture these structures within one time snapshot but also to track their evolution to the end of the simulation. Over time, halos merge and accrete mass and contain structures within them (subhalos).

In order to enable a smooth analysis workflow, HACC is instrumented with CosmoTools, a flexible *in-situ* analysis capability. Details of the CosmoTools design are discussed in Section 3.1. Figure 1 gives a general overview of the HACC simulation and analysis set-up for the different workflows we examined.

The simulation 'input deck' contains all the simulation parameters for the main run. It also includes a trigger for CosmoTools and a pointer to the CosmoTools configuration file. That file has all the details about the separate analysis tools, at which time steps

to run them, and which parameters to use for each. While HACC is running, CosmoTools is called at the requested time steps and analysis tasks are performed. The output is then directly written to the storage system, or in principle could be written to memory for further analysis steps. As we describe later, some analysis tasks are optimally performed on a different data-intensive platform or on a smaller portion of the HPC resource. For these tasks, a co-scheduled job can be triggered and data is read back in from storage or memory. We will describe and analyze different analysis approaches in Section 4 that are enabled with this general set-up.

The data hierarchy within HACC can be described in terms of three levels. The first (Level 1) is the raw HACC output which can either be the simulation particles (including positions, velocities, and particle tags) or full grid information. HACC uses uniform grids for calculating long-range forces, making these data structures very straightforward to store. The size of Level 1 data obviously depends on the number of particles simulated (typically, the particle number and grid size are the same), as each particle carries 36 bytes of information. Analysis tasks that require all Level 1 data, such as halo finding or power spectrum determination, result in Level 2 data. The analysis usually leads to a data volume reduction by a factor of a few – in the case of the power spectrum the resulting data is only one table, for the halo files the reduced data volume depends on the data of interest but is often an order of magnitude smaller. Finally, Level 2 data can be further analyzed to generate Level 3 data. This step is often carried out off-line and can require an independent set of analysis codes. Examples are properties of halos, including halo centers, shapes, and subhalo populations, the creation of synthetic sky maps, or summary statistics such as mass functions and halo concentrations. For a more extensive discussion, see, e.g., Ref. [14]. Table 1 gives two explicit examples for the simulations discussed in this paper later in Section 4. Next, we will give a brief overview of the CosmoTools implementation, our co-scheduling set-up, and the main analysis tools used in our examples.

3.1 In-Situ Analysis via CosmoTools

CosmoTools is a parallel *in-situ* analysis framework that is developed as an integral component of HACC. Performing the analysis *in-situ* enables analysis tasks to utilize the same memory and compute resources with the simulation, alleviating I/O and storage bottlenecks. However, embedding analysis tasks in the critical path of the simulation code, i.e., within a simulation time-step, requires careful design and poses additional challenges and considerations. Chief among the considerations are the *suitability* of an analysis task to be conducted *in-situ*, in conjunction with its net effect on *performance* and *memory footprint* throughout the lifetime of a simulation run. To address these challenges, the design and development of CosmoTools employs several key principles. It is minimally intrusive, providing a simple interface that can be invoked within the main physics loop. It is lightweight, with negligible overhead, and algorithms are designed to operate directly on an already distributed HACC particle dataset (i.e., Level 1 data), avoiding the need for deep copies, data transformations and/or data redistribution ("zero copy"). It is extensible to support new analysis algorithms, and is easily configurable in the problem setup, even while the simulation is running for computational steering.

CosmoTools defines a pure abstract base class, *InSituAlgorithm*, from which specific analysis tasks inherit. Each algorithm subclass must implement three virtual functions: `SetParameters()` for configuration, `ShouldExecute()` to determine if the analysis should be exectued at a given time step, and `Execute()` to perform the analysis. The *InSituAnalysisManager* class holds a list of references to concrete *InSituAlgorithm* instances and serves as the primary object interacting with the simulation code. There is a very small overhead for the virtual function calls, which could in principle be avoided by using the *Curiously Recurring Template Pattern* (CRTP) [37].

In our target domain, computational cosmology, we found that analysis tasks operating on Level 1 data (raw particles) are suitable candidates for *in-situ* analysis, while off-line analysis of Level 2 data can yield the best overall performance. To enable the combined *in-situ* and off-line/co-scheduled workflow, CosmoTools also provides a stand-alone driver that allows the algorithms to be invoked asynchronously by co-scheduling another analysis run, executed *in tandem* with the simulation using different resources. The details of our co-scheduled approach are out-lined in Section 3.2. Section 4 further elaborates on the advantages of our workflow with a specific example.

3.2 Co-Scheduling

Co-scheduling allows data to be analyzed as soon as it is made available. Since the jobs generating and analyzing data can overlap, co-scheduling can reduce the wall time for the overall task. However, the main advantage over the off-line approach is the convenience of automatically enabling the analysis workflow as the main application is still running. Optimally, secondary analysis jobs should be run on resources with sufficient capacity to allow a short queue time. Even under non-optimal conditions co-scheduling still allows analysis to become an automated part of the simulation workflow, even with some level of "pile-up" in the analysis stack, where many analysis jobs are queued while others run. The co-scheduling approach allows the analysis jobs to be spread over multiple resources if they are available. For the work presented here, a "listener" script, based on a scheme derived from the Bellerophon software stack [18, 19], is set up to run in the background of the main application in the Titan batch script. The listener launches analysis jobs when pre-specified output files are generated by the main application. While the listener and the main job run asynchronously, the rate at which the listener checks for new output files should be chosen to be much higher than the rate at which the main code generates new output files. When a new output file is found the listener generates a new batch script and input parameters, based on the timestep of the data and template files, and then submits the new job. The listener then resumes checking for new data.

Either the listener script or the main job can run in the background. In this case, the listener's backgrounding allows the job to end when the main application has completed rather than allowing the listener to fruitlessly check for new data until the end of the user defined wall time. If the last output files come at the very end of the main application's execution time, an additional instance of the listener would run after the job completes to catch the last output data.

For co-scheduling on resources at a user facility, it is important to consider the facility's queue policies and resource capabilities. In HACC, the secondary analysis code is well optimized for GPUs. OLCF's designated analysis cluster, Rhea, has the capacity to ensure that enough nodes are available for smaller jobs to have short queue waits. However, Rhea does not currently have GPUs. The secondary job could be co-scheduled on Titan with the main job, and use Titan's GPUs. However, Titan's queue is designed to favor large jobs. The queue policy only allows two jobs that use less than 125 nodes to run simultaneously. Thus, co-scheduling on Titan would require a queue exemption if more than one secondary analysis job is to run on Titan while the primary job is running.

	Level 1	Level 2	Level 3
Examples	Raw particles, grid information	Halo particles, density fields, subsamples of particles	Halo properties, galaxy catalogs, subhalos, mass functions concentrations
Size, 1024^3, last step only	∼40GB (raw particles)	∼5GB (halo particles, > 300,000 particles)	Halo centers, ∼43MB
Size, 8192^3, last step only	∼20TB (raw particles)	∼4TB (halo particles, > 300,000 particles)	Halo centers, ∼10GB

Table 1: Examples for Level 1, 2, and 3 data products and sizes for the two case studies discussed in Section 4. The data sizes obviously depend on the halo definition used. In the case of the 8192^3 particle simulation, very small halos were discarded, leading to a smaller Level 3 data set.

3.3 Halo Analysis

3.3.1 Halo Identification

There are several definitions commonly used for identifying halos within cosmology simulations (e.g., [30, 36]). In this study, we use the percolation-based Friends-Of-Friends (FOF) halo definition [16], which imposes no restrictions on halo shape. An FOF halo consists of all particles that are within the "linking length" of at least one other particle in the halo (the choice of linking length is connected to the choice of an isodensity surface that defines the boundary of the halo). To avoid spurious identifications, halos with fewer than a specified number of particles are discarded. Finding FOF halos is equivalent to finding the connected components of a graph in which each particle is a vertex, and there exists an edge between two vertices if and only if the distance between them is less than the specified linking length. However, the number of edges (up to $O(n^2)$ for n particles) makes it intractable to explicitly compute and store an edge list or adjacency matrix.

We compute FOF halos in parallel across MPI ranks by distributing particles across the processors according to a domain decomposition. "Overload regions" are defined at the boundaries of the processors, with each of the neighboring processors receiving a copy of the particles in this region. The size of the overload regions are defined to be large enough relative to the maximum feasible halo extent such that each halo is assured of being found in its entirety by at least one processor. Within each rank, FOF halos are found using a serial algorithm which constructs and then recursively traverses a balanced k-d tree. Starting at the leaf nodes, which contain individual particles, it merges particles into halos. At higher levels of the tree, bounding boxes which define the space covered by the subtree rooted at a node are used to reduce the number of particle-to-particle distance comparisons, allowing whole subtrees to be merged into a halo or excluded from a halo at once. After each processor has found its halos locally, the parallel halo finder identifies halos found in whole or in part by multiple processes, and assigns them to a unique processor, see Refs. [15, 41] for details.

Halos themselves contain sub-structure, which can be identified using subhalo finders in simulations with sufficient resolution. For this study, we used our implementation [10] of the subhalo finding algorithm presented in Refs. [24, 35]. The local density for each particle in the parent FOF halo is estimated by finding a specified number of nearest neighbor particles, and computing a density based on the total mass of these particles and the distance to the furthest of these. A Barnes-Hut tree, similar to an octree but with support for more efficient traversals, is used for calculating the local densities using an SPH (Smoothed Particle Hydrodynamics) kernel. A subhalo candidate tree is then constructed by iterating over the particle list in sorted order according to density. Finally, candidate particles with high total energy are "unbound" from subhalos in a multi-pass algorithm, removing no more than one-quarter of the particles with positive energy at each step.

3.3.2 Halo Center Finding

Once a halo has been identified, accurately determining its center is essential for comparison of results with observations and for calculating halo properties such as concentration. The concentration is determined from the density profile of the halo as a function of radius – if the center is not exactly at the density maximum, the concentration will be underestimated. An accurate halo center is also important for placing the central galaxy into the halo. Many cosmological measurements that are extracted from the simulation depend on accurate galaxy placements, such as galaxy-galaxy lensing, the galaxy correlation function, or strong lensing measurements. Computation of spherical overdensity (SO) halos [36] may also be seeded at FOF halo centers. In this study, we use the Most Bound Particle (MBP) definition for a halo center, which identifies the center as the particle with the minimal potential (equivalent to the density maximum), where the potential for a given particle is computed as the sum over all other particles of the negative of mass divided by its distance to the particle. A small constant offset term may be added to the distance to avoid numerical issues caused by extremely close particles.

Within each MPI rank, the center for each halo can be computed with an A* search algorithm, which uses an optimistic heuristic to estimate the potential for each particle, allowing it to locate the particle with minimum potential without having to explicitly compute the potentials for all particles. This algorithm is reported to be faster than a brute force approach of computing potentials for all particles by a problem-dependent factor of roughly eight [10], but it is still a serial $O(n^2)$ algorithm. More recently, using PISTON/VTK-m, we have implemented a simple algorithm that computes the potentials for all particles and finds the minimum. The algorithm is easily parallelizable, since the potential for each particle can be computed in parallel.It runs portably across multi-core and many-core accelerators using Thrust primitives. On Titan's GPUs, it is much faster than the serial A* algorithm run on Titan's CPUs [34]. Nevertheless, as an $O(n^2)$ algorithm, it can lead to significant load imbalance. For example, finding the MBP center of a halo with 10 million particles can take 10,000 times longer than for a halo with 100,000 particles.

We note that more computationally efficient, but less accurate, methods do exist for finding halo centers. We have experimented with approximate center finding methods but none of them satisfied our strict accuracy requirements. Furthermore, as previously discussed, significant load imbalance is inevitable for many analysis tasks, thus motivating the off-line and co-scheduling workflows described in this paper. Our MBP center finder serves as a good

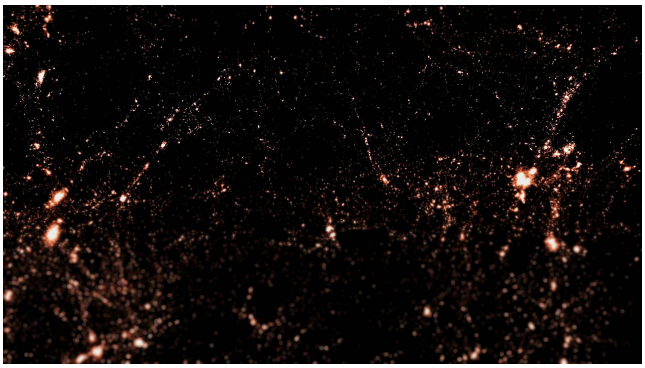

Figure 2: Visualization of the Q Continuum simulation's particle distribution, zoomed in to a sub-region of the volume of a single node (one out of 16,384 nodes in total), showing the halos that have formed in this region at the final time step.

test case for these approaches. In the context of cosmology simulations, many subhalo finding algorithms also exhibit significant load imbalance, with subhalo finding taking much longer for large halos than for small ones.

4. RESULTS

4.1 Analysis of the Q Continuum Simulation

The implementation of our integrated *in-situ*/co-scheduling analysis approach was partially motived by the challenges we encountered when analyzing the very large, high-resolution, highly clustered data resulting from the final time step of the Q Continuum simulation. The simulation, described in detail in Ref. [13], evolved 8192^3 particles on Titan and we stored 100 time snapshots, resulting in more than 2Pbytes of raw (Level 1) data. A visualization produced based on the results of this simulation is shown in Figure 2. HACC is equipped to carry out a complete *in-situ* analysis and in principle we could have carried out the following tasks while the code was running on Titan: 1) Power spectrum calculation, based on cloud-in-cell density estimation on a uniform grid and FFTs of size 8192^3; 2) Halo identification, based on the clustering algorithm discussed in Section 3.3; 3) Halo center finding, based on the algorithm discussed in Section 3.3; 4) Sub-halo finding (identifying sub-structure within halos); 5) Halo mass estimation based on a spherical overdensity definition.

The power spectrum calculation, the halo identification, and the spherical overdensity mass estimator all lend themselves well to efficient parallel implementation. The halo center finder and subhalo finder, though, as explained in the previous section, slow down rapidly with increasing halo size. In addition, the three halo analysis steps have to be carried out in sequence – hence, although the over density mass estimator is very fast, it relies on information obtained by the center finder. When the Q Continuum simulation was started, the halo identification and center finding algorithms were tightly coupled and only implemented to run on CPU systems. The very high mass resolution of the Q Continuum simulation forced us to rethink this strategy and, in the meantime, store the full raw outputs on disk to enable a later off-line analysis. As a first important improvement step we re-implemented the center finder within PISTON, to take full advantage of Titan's GPUs as explained above. This provided approximately a factor of fifty speed-up (here we compare the halo finder run on one rank per node on the CPUs with the GPU implementation – the memory requirements for this simulation, as well as limitations of OpenCL, used for some physics kernels, did not allow us to use more ranks per node).

SLICE	Redshift z	Max Find	Min Find	Max Center	Min Center
60	1.680	433	352	449	19
64	1.433	483	385	668	19
73	0.959	663	532	1819	19
100	0	2143	1859	21250	2.4

Table 2: Examples of timing results for different analysis steps. Reported are the time the slowest and the fastest nodes took to do the halo identification (Find) and the center finding (Center). All times are quoted in seconds. The analysis of the last step was split into two parts, one for small and medium halos carried out on Titan and one for large halos carried out on Moonlight, a separate GPU system. The time for the slowest center finding result at $z = 0$ from Moonlight was adjusted by a factor of 0.55 for ease of comparison with the Titan results. The time for the fastest center-finding result at $z = 0$ from Titan has improved compared to earlier redshifts since particles are more tightly clustered the further the evolution has progressed.

After the analysis of the first 64 time slices, the load imbalance of the center finder outweighed the cost of I/O and distribution code necessary to carry out the analysis. We therefore implemented a division of labor for the center finding algorithm for small to medium and heavy halos; this point is illuminated further with the timing data shown in Table 2. Table 2 shows the timing for the full halo analysis for four of our one hundred outputs. The redshift z indicates the cosmic time at which the data was stored; the redshift today (and therefore the end of the simulation) is $z = 0$. The simulation started at $z = 200$ and the first output was stored at $z = 10$. We report the timing for the halo identification (Find) and the center finder on the GPU (Center) for the slowest and the fastest node. As is easily seen, the identification is well balanced for each time step shown. Since more and larger structures form over time, the overall identification time increases. The calculation for the halo center finder on the other hand is already rather unbalanced at the 60th analysis step. Nevertheless, in order to carry out the analysis of the time step off-line, we are forced to read in the particle data (20TB per snapshot) and distribute them among the processors. Reading the full particle set from one snapshot on Titan takes roughly 10 minutes (see Ref. [13] for details on HACC's I/O implementation on Titan) and another 10 minutes to distribute the particles among the 16384 nodes which are needed to hold the particles. At that point, 10 minutes of total analysis time for the center finding (even if out of balance) is justified. For later time steps, this situation dramatically worsened and we broke up the center finding into two steps.

We use the final time step as an example to explain our strategy and timings. We first read in the particles for the last time step and carried out the halo identification which took approximately one hour on 16,384 nodes. We then found the centers for all halos that have less than 300,000 particles, which took just over one minute. We printed out all the particles that reside in halos with more than 300,000 particles to the file system – the resulting data (Level 2) was a factor of 5 less than the raw data at Level 1. Figure 3 shows the number of halos as a function of mass to illustrate this cut. The Level 2 data was moved to a different machine (Moonlight, a GPU cluster at Los Alamos) where center finding was carried out. Moonlight provided more flexibility for the queuing of small, long analysis jobs which was important for our analysis. In a final step, the

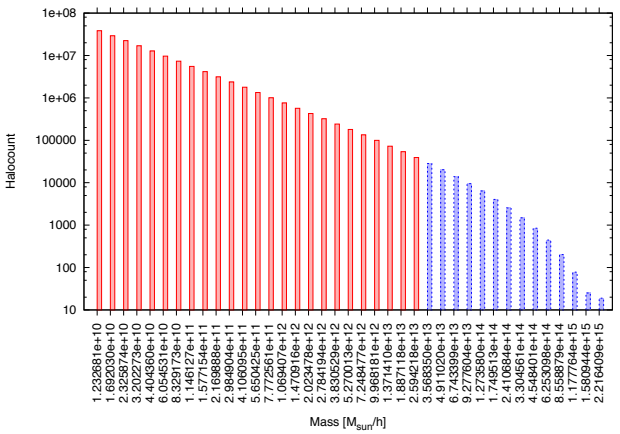

Figure 3: Log-log plot of halo counts as a function of mass at $z = 0$. The red histograms mark the halos that were fully analyzed after the halo finding step while the blue histograms show the halos which were off-loaded to Moonlight for the center finding step. Overall, we found 167,686,789 halos, out of which 84,719 were off-loaded. The center finding for the remaining halos (99.9%) took approximately one minute on 16,384 nodes of Titan.

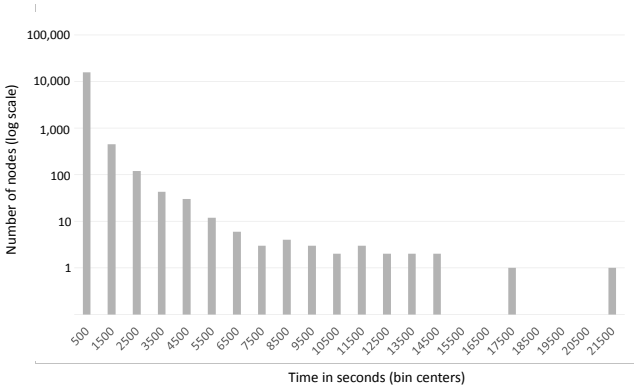

Figure 4: Histogram for the distribution of projected center finding times on Titan, showing node counts on a log scale for bins of width 1000 seconds. Centers for large halos (over 300,000 particles) were actually computed off-line on Moonlight, so these timings are projected based on the sizes of all large halos output by each node on Titan. Times for finding centers for small halos (under 300,000 particles), which were computed *in situ* on Titan, are not included here, but no node required more than approximately 60 seconds to complete all its *in situ* center finding.

two files from the Titan and Moonlight analysis were merged to provide a complete set of halo centers and properties. Shown in Figure 4 is the distribution of the time it would have taken each node on Titan if all halo center finding had been computed *in-situ*, with timings projected based on the sizes of the large halos.

While we chose this threshold of 300,000 manually, making the choice of what analysis to perform *in-situ* and what to perform off-line could be made more automated. First, one would estimate the time the code will spend in I/O, t_{io}, if the analysis were off-line. This depends on the total number of particles. The mass of the largest halo, m_{max_io}, that could be analyzed in time less than t_{io}, would then be estimated. (Halo mass is just a function of the number of particles, with all particles having equal mass.) During the simulation, all halo finding occurs *in-situ*, and the mass of the largest halo, m_{max_sim}, can be found. If $m_{max\_sim} < m_{max\_io}$, the centers for all halos can be computed *in-situ*. If $m_{max\_sim} > m_{max\_io}$, then all particles in halos with mass greater than m_{max_io} should be saved out for off-line center-finding. To set up an optimized co-scheduling job, one would first estimate the time, T, to analyze all halos, which is easily available from the halo masses found *in-situ*. From this, the time, t_{max}, it will take to analyze the largest halo can be estimated. The number of ranks for the co-scheduling task should be set equal to T/t_{max}. The halos should be distributed so that each rank has roughly the same workload (estimated again from halo masses).

For optimal I/O performance, the results from 128 nodes from Titan were aggregated in one file, resulting in 128 files containing 128 blocks each. Each file was analyzed separately by a set of single-node jobs on Moonlight. The longest analysis job took 37.8 hours, the shortest 6.0 hours. The longest single block analysis took 10.6 hours. This block held a halo with $\sim$25 M particles, and several other large halos. Overall, the center finding required about 1770 node hours on Moonlight. On Titan, the analysis would have been faster by a factor of roughly 0.55 (due to newer hardware – the estimate is based on direct comparisons of our timings on the two systems, and on the specifications of Moonlight's M2090 and Titan's K20X GPUs), resulting in 985 node hours, or $\sim 30,000$ core hours. Adding 0.5M core hours for the halo finding and center finding for the small and medium size halos, the analysis required 0.52M core hours. If we would have carried out this step fully *in-situ* or off-line, the time would have been dictated by the slowest block. Therefore, we would have spent 5.9 hours on 16,384 nodes plus one hour for halo identification on Titan, leading to 3.4M core hours, a factor of 6.5 more expensive than the approach taken. We emphasize that this large factor is due to the fact that this simulation covers a very large volume at very high mass resolution, resulting in very rare, large objects.

As should be clear from the above, the analysis workflow for this large simulation is sub-optimal. The overheads due to I/O (writing out the Level 1 files and reading them in again for analysis) and particle distribution are not small – on a simulation of the size of the Q Continuum this by itself amounts to $\sim$ 0.16M core hours (taking into account the extra charges on Titan due to the GPUs) per analysis step (and we would like to emphasize that these steps were heavily optimized for HACC and the I/O almost reached peak performance on the Lustre file system). For future runs, we therefore developed a more sophisticated analysis workflow with different options for *in situ*, co-scheduling, and off-line analysis.

4.2 Comparison of Different Approaches

We now discuss our implementations of different analysis strategies with CosmoTools' *in-situ* capability and co-scheduling options implemented on the OLCF machines. The analysis tasks we perform are the halo finding and the halo center determination. Since the analysis of the Q Continuum run is very expensive, we focus our discussion on a downscaled version. While this simulation does not have clusters at the extreme mass scales found in the Q Continuum run (the largest halo in this smaller run contains 2,548,321 particles, an order of magnitude smaller than from the Q Continuum run), due to its smaller volume, the mass resolution is very similar and the results can be approximately scaled up. To be more specific, the simulation evolves 1024^3 particles in a $(162.5\ \mathrm{Mpc})^3$ volume on 32 nodes of Titan. This reduces the problem by exactly

a factor of 512. The memory requirements for this run (including the overhead for the overload regions) do not allow us to reduce the number of nodes to smaller than 32. Table 3 summarizes our different approaches, while Table 4 shows more detailed results. We only demonstrate results for the last time step – the reader should keep in mind though that running the full analysis would involve 100 snapshots.

The first set-up is a complete *in-situ* analysis. This approach has two major advantages: It does not require I/O nor a redistribution of the particles after read-in since all the particles are readily distributed in memory already. In addition, it does not require extra wait time in the queue since only the main simulation job has to be submitted. For our test run, this approach takes overall 722 secs for the halo and center finding task. On 32 nodes and with the additional charging factor of 30 for a node hour on Titan, this amounts to just under 200 core hours. The major drawback is the load imbalance of the center finder. In this smaller test case, the imbalance between the fastest and the slowest node is a factor of 15, accounting for the time the halo with more than 2M particles takes.

With the second strategy, we perform a full off-line analysis step. In this case, we have additional computational requirements for 1) writing Level 1 data during the simulation, 2) reading Level 1 data for analysis, and 3) redistribution of the particles after reading for the analysis. In addition, the queuing time can be considerable, since this approach requires as many nodes as the simulation run itself to fit in all the particles. This can add days to a week of wait time. The one advantage of the off-line approach is that the data is now available for analysis tasks that were not foreseen when the original simulation was set up. In cosmology, simulations are science-rich and new questions often come up after the simulation run has finished and the outputs are being investigated. Having the full raw particle output in this case can be very important. In our test case, we carry out the off-line analysis (halo finding and center finding) in one step – we could certainly follow a more complex strategy as we did for the Q Continuum simulation by breaking up the center finding step into two parts (small and medium size halos vs. large halos). Since the timing advantages of this approach are shown in the next examples, we will not discuss it for a full off-line approach separately. For the full off-line approach, more than 400 secs are added to the analysis for writing, reading, and redistributing the full Level 1 data set. Overall, this approach takes 1332 secs (not counting the queuing time), leading to a charge of 355 core hours (plus one core hour for writing Level 1 data during the simulation run), 163 more than the cost of the *in-situ* approach. Most of this difference is due to the time required to write, read, and redistribute the data for off-line analysis (118 core hours), although some (the 45 core hour difference between the analysis for the two approaches) may be due only to noise in Titan's performance. We emphasize though that for a larger volume run, the center finder time would outweigh the I/O and redistribution by a much larger fraction so the difference between *in-situ* and off-line would be not as large.

We describe three variations on the final strategy, which combines an *in-situ* analysis step to reduce Level 1 to Level 2 data and then a separate off-line analysis step to further investigate Level 2 data, possibly on a different machine. In the *in-situ* step in all three cases, all halos and particles within the halos are identified. For halos with less or equal than 300,000 particles (following our Q Continuum analysis) we carry out the center finding on the fly as well. Both steps combined (identifying all halos and finding centers for halos with less or equal 300,000 particles) takes $\sim$361 secs, reducing the full *in-situ* time by a factor of two.

Particles in halos with more than 300,000 particles are treated in three different ways. In the first, simple variation of this approach, the particles in large halos are written to disk. Compared to the full off-line approach, this reduces the I/O time and time for redistribution of the particles by more than a factor of two. Compared to the full *in-situ* analysis, this time is still non-negligible, amounting to $\sim$75 secs in our example. For the off-line analysis of the Level 2 data, we are now flexible with regard to our approach. Due to the data reduction from Level 1 to Level 2 data, we only need 4 nodes for the following analysis (we tested that even one node is sufficient, but the computational costs between one node and four nodes are roughly the same while the wall clock reduced for four nodes by a factor of four). The center finding took 1075 secs for this test on Titan. Overall, the combined *in-situ*/off-line approach reduces the cost in our example compared to the full *in-situ* approach by $\sim 30\%$. Again, we point out that for a larger simulation this cost-reduction would be even more, as discussed in the previous section. In this variation, this task is carried out off-line, so some additional waiting time in the queue is required.

This waiting time could be reduced in the second variation of the combined *in-situ*/off-line strategy. Here we co-schedule the main run and therefore the *in-situ* analysis with the off-line analysis of the Level 2 data. While we were able to run an *in situ*/co-scheduled job successfully on the Titan/Rhea combination for one time step, in order to verify that the listener on the OLCF system automatically queues an analysis job when data becomes available (i.e., when a pre-specified file is written by the simulation), direct comparisons to *in-situ* analysis timings are not meaningful if the off-line and co-scheduled analysis tasks are run on an analysis cluster with different hardware. Rhea's hardware is non-optimal for our analysis tasks, and, even though our portable PISTON analysis algorithms can run on CPUs as well as GPUs, the lack of GPUs slowed down the center finding considerably. Therefore, we report timing results only for analysis carried out on Titan from the simple variation of the *in-situ*/off-line workflow. The core hours for the co-scheduled workflow would be the same if run on an analysis cluster with equivalent hardware to the main HPC system, since the only difference between these variations is the scheduling. The advantage of the co-scheduling variation is that the workflow is simplified because the off-line analysis is queued automatically. Moreover, if

Method	I/O	Redist.	Queueing	Core hrs
in-situ	none	none	none	193
off-line	Level 1	Level 1	full	356
in-situ/off-line				
simple	Level 2	Level 2	partial	135
co-scheduled	Level 2	Level 2	partial simult	(same)
in-transit	none	Level 2	partial simult	(n/a)

Table 3: Analysis workflows investigated in this paper. The I/O step always includes both reading and writing. The queuing only includes the analysis job itself. Partial and full refer to the allocation request compared to the simulation run, while simult means that multiple small analysis jobs are queued automatically as data becomes available from the simulation and can run simultaneously with the simulation and each other. In the case of the Q Continuum simulation, Level 2 data contains only 20% of Level 1 data. Core hours are calculated following the Titan charge policy – an hour per node leads to a charge of 30 core hours. Core hours for the *in-situ*/co-scheduled workflow would in theory be equal to the simple *in-situ*/off-line workflow (135) if run on equivalent hardware. The last implementation would be optimal but does not currently exist on systems to which we have access.

In-situ Only Workflow

	Simulation					Post-processing					
	Queuing	Sim	Analysis	Write	Total	Queuing	Read	Redistribute	Analysis	Write	Total
Time (sec)	One 32-node job	772	722	0.3	1494 + queuing	--	--	--	--	--	0
Core hours	--	206	193	0.1	399	--	--	--	--	--	0
Data	--	Level 1 in memory, Level 3 output				--	--	--	--	--	--
Resources	--	32 nodes				--	None				

Off-line Only Workflow

	Simulation					Post-processing					
	Queuing	Sim	Analysis	Write	Total	Queuing	Read	Redistribute	Analysis	Write	Total
Time (sec)	One 32-node job	779	0	5	784 + queuing	One 32-node job, queued after sim	5	435	892	0.3	1332 + queuing
Core hours	--	208	0	1	209	--	1	116	238	0.1	355
Data	--	Level 1 in memory, Level 1 output				--	Level 1 input	Level 1 comm.	Level 1 in mem.	Level 3 output	
Resources	--	32 nodes				--	32 nodes				

Combined In-situ / Off-line Workflows

	Simulation					Post-processing					
	Queuing	Sim	Analysis	Write	Total	Queuing	Read	Redistribute	Analysis	Write	Total
Time (sec)	One 32-node job	774	361	3	1138 + queuing	One 4-node job covering all timesteps (only 1 in this test), queued after sim *Co-scheduled or in-transit: a 4-node job for each timestep, queued as data is available*	3	75	1075	0.2	1153 + queuing
Core hours	--	206	96	1	303	--	0.1	2	36	0.01	38
Data	--	Level 1 in memory, Level 2 and partial Level 3 output *In-transit: Level 1 in memory, Level 2 in external memory, partial Level 3 output*				--	Level 2 input *In-transit: None*	Level 2 comm.	Level 2 in mem.	Level 3 output	
Resources	--	32 nodes				--	4 nodes				

Table 4: Detailed results showing the trade-offs between time, core hours, data (memory usage and I/O), and resource usage for each of the considered workflow strategies. Timings are for the last timestep of the simulation. For the *in-situ*/off-line strategy, timings are given only for the simple variation, while differences for the proposed co-scheduled and *in-transit* variations are shown in italics. The core hours shown in Table 3 correspond to the sum of the core hours for the analysis and write steps of the simulation run, plus the total core hours for the post-processing run.

the full simulation were to be run, and analysis jobs co-scheduled for each desired time step, the total core hours would still be expected to be the same as if all off-line analysis were performed after the completion of the simulation (assuming negligible start-up time for the analysis tasks), but the scientist may have to wait a shorter time for his/her results, since the co-scheduled analysis jobs could run simultaneously with the main simulation.

The third variation we discuss is at this point only a hypothetical implementation, but might be enabled on some machines in the near future. (We did not have access to any machines that would have allowed us to carry out this test.) In this case, the Level 1 to Level 2 reduction as well as the partial Level 2 data analysis are carried out *in-situ*. Instead of writing out the Level 2 data that require further analysis to disk, the data is now stored on a separate memory device and the analysis is done *in-transit*. This could be either NVRAM or an external memory set-up that is connected to both the main HPC system as well as the analysis cluster. This set-up would not require any additional I/O for the Level 2 data or additional queuing of analysis jobs. It would, however, require the redistribution of Level 2 data for the analysis system. This is a small overhead compared to the additional computational costs for a full *in-situ* analysis with a non-optimal load-balanced code.

As mentioned in Section 3.3, identifying subhalos within halos is another analysis task that is not easily fully load-balanced. In addition, our current implementation based on a tree-algorithm does not take advantage of GPUs. In our test problem, subhalos were found for halos with more than 5000 particles. (Smaller halos will not exhibit much substructure and the identification is unreliable if the subhalo size itself is too small.) Subhalo finding carried out *in-situ* on 32 nodes of Titan's CPUs took 8172 secs for the slowest and 1457 secs for the fastest node, an imbalance of more than a factor of five. Co-scheduling to off-load this work to a fast analysis system would again be expected to save compute resources for this analysis step.

These halo analysis tasks fall into a broader category of feature extraction algorithms, which tend to require significant computational resources, and result in feature catalogs and/or summary statistics. Other algorithms in this category may also be able to benefit from this combined *in-situ*/off-line approach. In contrast, some other classes of analysis tasks, such as the generation of image databases, which can quickly compute and save an output, may be best suited to a simple *in-situ* workflow. Also, analysis tasks, such as the computation of streamlines, which do not operate independently on data from different time steps, may be more difficult to adapt to a co-scheduled workflow.

5. DISCUSSION

The analysis of very large datasets is a nontrivial and significantly time-consuming task and can easily become more complicated – from the workflow perspective, often substantially more complicated – than the main simulation run. In the application example presented here, taken from computational cosmology, the increasing size of simulations leads to the occurrence of large, rare objects in very large simulation volumes. The existence of these objects can lead to difficulties in the development of fully load-balanced analysis routines. In our specific case, the computational

burden for the determination of halo properties (their potential minimum center and their subhalos) depends strongly on the size of the halos. In current state-of-the-art large-volume, high-mass resolution simulations such as the Q Continuum or the Outer Rim runs, the sizes of the halos vary vastly, but only a very small number of extreme-size halos exists. The combination of well load-balanced and difficult to load balance analysis tasks demands a flexible workflow implementation that combines *in-situ* and co-scheduled/offline approaches. The major advantage of a full *in-situ* analysis is the savings in I/O and redistribution time, as well as queue wait times. Additionally, it also helps to avoid problems with the file system. Nevertheless, this approach can become too costly, and software solutions at the algorithmic level can present their own problems. We therefore advocate for a combined approach that can take advantage of the HPC system for some *in situ* tasks and optimize the analysis of a subset of the raw simulation data on either a smaller portion of the HPC system or specialized data-intensive machines. To fully automate the analysis process and avoid the need of manual user interference, future systems will be well served by sophisticated submission systems that are designed from the outset to allow and promote co-scheduling.

6. ACKNOWLEDGMENTS

The work performed by CS, LL, PF, and JA was supported by the Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research under the Institute of Scalable Data Management, Analysis and Visualization (SDAV). An award of computer time was provided by the Innovative and Novel Computational Impact on Theory and Experiment (INCITE) program. This research used resources of the OLCF, which is supported by DOE/SC under contract DE-AC05-00OR22725. HF, SH, KH, and AP were supported by the U.S. Department of Energy, Basic Energy Sciences, Office of Science, under contract No. DE-AC02-06CH11357. We would like to thank Silvio Rizzi and Joe Insley of Argonne National Laboratory for the visualization shown in Figure 2.

7. REFERENCES

[1] H. Abbasi, J. Lofstead, F. Zheng, K. Schwan, M. Wolf, and S. Klasky. Extending I/O through high performance data services. IEEE International Conference on Cluster Computing and Workshops, 2009.

[2] A. Agranovsky, D. Camp, C. Garth, E.W. Bethel, K.I. Joy, and H. Childs. Improved Post Hoc Flow Analysis Via Lagrangian Representations. In Proceedings of the IEEE Symposium on Large Data Visualization and Analysis (LDAV), pages 67 – 75, Paris, France, November 2014.

[3] S. Ahern, A. Shoshani, K.-L. Ma, et al. Scientific discovery at the exascale. DOE ASCR 2011 Workshop on Exascale Data Management, Analysis, and Visualization, 2011.

[4] J. Ahrens, S. Jourdain, P. O'Leary, J. Patchett, D.H. Rogers, and M. Petersen. An Image-based Approach to Extreme Scale In-Situ Visualization and Analysis. International Conference on Supercomputing, November 2014.

[5] M. Bauer, S. Treichler, E. Slaughter, and A. Aiken. Legion: Expressing Locality and Independence with Logical Regions. International Conference on Supercomputing, 2012.

[6] J. Bent, S. Faibish, J. Ahrens, G. Grider, J. Patchett, P. Tzelnic, and J. Woodring, Jitter-free co-processing on a prototype exascale storage stack. IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST), 2012.

[7] G. Blelloch. Vector models for data-parallel computing. MIT Press. ISBN 0-262-02313-X. 1990.

[8] S.W. Bruenn, A. Mezzacappa, W.R. Hix, J.M. Blondin, P. Marronetti, O.E.B. Messer, C.J. Dirk, and S. Yoshida. Mechanisms of Core-Collapse Supernovae & Simulation Results from the CHIMERA Code. Probing Stellar Populations Out To The Distant Universe: Cefalu 2008.

[9] N. Fabian, K. Moreland, D. Thompson, A.C. Bauer, P. Marion, B. Geveci, M. Rasquin, and K.E. Jansen. The ParaView coprocessing library: A scalable, general purpose in-situ visualization library. IEEE Symposium on Large-Scale Data Analysis and Visualization, October 2011.

[10] P. Fasel. Cosmology analysis software. Technical report, Los Alamos National Laboratory, 2011.

[11] I. Foster and C. Kesselman. The Grid: Blueprint for a new computing infrastructure. Morgan Kaufmann, 1999.

[12] S. Habib, A. Pope, H. Finkel, N. Frontiere, K. Heitmann, D. Daniel, P. Fasel, V. Morozov, G. Zagaris, T. Peterka, V. Vishwanath, Z. Lukic, S. Sehrish, and W.-k. Liao. HACC: Simulating Sky Surveys on State-of-the-Art Supercomputing Architectures arXiv:1410.2805, New Astronomy, submitted.

[13] K. Heitmann, N. Frontiere, C. Sewell, S. Habib, A. Pope, H. Finkel, S. Rizzi, J. Insley, and S. Bhattacharya. The Q Continuum Simulation: Harnessing the Power of GPU Accelerated Supercomputers, The Astrophysical Journal Supplement, accepted for publication.

[14] K. Heitmann, S. Habib, H. Finkel, N. Frontiere, A. Pope, V. Morozov, S. Rangel, E. Kovacs, J. Kwan, N. Li, S. Rizzi, J. Insley, V. Vishwanath, T. Peterka, D. Daniel, P. Fasel, and G. Zagaris. Large-Scale Simulations of Sky Surveys. Comput. Sci. Eng. 16, 14, 2014.

[15] C.-H. Hsu, J. Ahrens, K. Heitmann. Verification of the time evolution of cosmological simulations via hypothesis-driven comparative and quantitative visualization. Pacific Vis, 2010.

[16] J. Huchra and M. Geller. Groups of Galaxies I. Nearby Groups, ApJ, 257(423), 1982.

[17] C. Johnson, R. Ross, et al. Visualization and knowledge discovery. DOE/ASCR Workshop on Visual Analysis and Data Exploration at Extreme Scale, 2007.

[18] E.J. Lingerfelt, O.E.B. Messer, J.A. Osborne, R.D. Budiardja, and A. Mezzacappa. A Multitier System for the Verification, Visualization and Management of CHIMERA. Procedia Computer Science, 4(0), 2011.

[19] E.J. Lingerfelt, O.E.B. Messer, S.S. Desai, C.A. Holt, and E.J. Lentz. Near Real-time Data Analysis of Core-collapse Supernova Simulations with Bellerophon. Procedia Computer Science, 29(0), 2014.

[20] L.-T. Lo, C. Sewell, and J. Ahrens. PISTON: A Portable Cross-Platform Framework for Data-Parallel Visualization Operators. Proceedings of the Eurographics Symposium on Parallel Graphics and Visualization, May 2012.

[21] J. Lofstead, R. Oldfield, T. Kordenbrock, and C. Reiss. Extending scalability of collective I/O through Nessie and staging. In Proceedings of the 6th Parallel Data Storage Workshop, Seattle, WA, November 2011.

[22] J. Lofstead, R.A. Oldfield, and T.H. Kordenbrock. Experiences applying data staging technology in unconventional ways. In 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Delft, The Netherlands, May 2013. IEEE/ACM.

[23] K.-L. Ma. In-Situ Visualization at Extreme Scale: Challenges and Opportunities. IEEE Computer Graphics

Appl., vol. 29, no. 6, pp. 14-19, Nov. 2009.

[24] M. Maciejewski, S. Colombi, V. Springel, C. Alard, and F.R. Bouche. Phase-space structures II. Hierarchical Structure Finder. Monthly Notices of the Royal Astronomical Society, July 2009.

[25] B.H. McCormick, T.A. DeFanti, and M.D. Brown, editors. Visualization in Scientific Computing (special issue of Computer Graphics), volume 21. ACM, 1987.

[26] Q. Meng, A. Humphrey, and M. Berzins. The Uintah Framework: A Unified Heterogeneous Task Scheduling and Runtime System. Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing at the International Conference on Supercomputing, 2012.

[27] O.E.B Messer, S.W. Bruenn, J.M. Blondin, W.R. Hix, A. Mezzacappa, and C.J. Dirk. Petascale supernova simulation with CHIMERA. J. Phys.: Conf. Ser., July 2007.

[28] K. Moreland, R. Oldfield, P. Marion, S. Joudain, N. Podhorszki, V. Vishwanath, N. Fabian, C. Docan, M. Parashar, M. Hereld, M.E. Papka, and S. Klasky. Examples of in transit visualization. In Proceedings of the PDAC 2011 : 2nd International Workshop on Petascale Data Analytics: Challenges and Opportunities, Seattle, WA, November 2011.

[29] B. Nouanesengsy, J. Woodring, J. Patchett, K. Myers, and J. Ahrens. Adr visualization: A generalized framework for ranking large-scale scientific data using analysis-driven refinement. In Large Data Analysis and Visualization, 2014.

[30] M. A. Patwary, D. Palsetia, A. Agrawal, W.-K. Liao, F. Manne, A. Choudhary. A new scalable parallel DBSCAN algorithm using the disjoint-set data structure. International Conference on Supercomputing, 2012.

[31] D. Rogers, K. Moreland, R. Oldfield, and N. Fabian. Data Co-Processing for Extreme Scale Analysis Level III ASC Milestone (4745). Technical Report SAND2013-1122, Sandia National Laboratories, November 2013.

[32] R.B. Ross, T. Peterka, H.-W. Shen, Y. Hong, K.-L. Ma, H. Yu, and K. Moreland. Visualization and parallel I/O at extreme scale. Journal of Physics: Conference Series, 2008

[33] C. Sewell, L.-T. Lo, and J. Ahrens. Portable Data-Parallel Visualization and Analysis in Distributed Memory Environments. Proceedings of the IEEE Symposium on Large-Scale Data Analysis and Visualization, October 2013.

[34] C. Sewell, K. Heitmann, L.-T. Lo, S. Habib, and J. Ahrens. Utilizing Many-Core Acclerators for Halo and Center Finding within a Cosmology Simulation. Subm., 2015.

[35] V. Springel, N. Yoshida, and S.D.M. White. GADGET: a code for collisionless and gasdynamical cosmological simulations. New Astronomy, 6, 79-117, 2001.

[36] J.L. Tinker, A.V. Kravtsov, A. Klypin, K. Abazajian, M.S. Warren, G. Yepes, S. Gottlober, and D.E. Holz. Toward a halo mass function for precision cosmology: the limits of universality. Astrophysics Journal, 2008.

[37] D. Vandervoorde and N. Josuttis. C++ Templates: The Complete Guide, 2002.

[38] V. Vishwanath, M. Hereld, and M.E. Papka. Toward simulation-time data analysis and I/O acceleration on leadership-class systems. IEEE Symposium on Large Data Analysis and Visualization, 2011.

[39] B. Whitlock, J.M. Favre, and J.S. Meredith. Parallel in-situ coupling of simulation with a fully featured visualization system. In Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization, 2011.

[40] W. Widanagamaachchi, P.-T. Bremer, C. Sewell, L.-T. Lo, J. Ahrens, and V. Pascucci. Data-Parallel Halo Finding with Variable Linking Lengths. Proceedings of the IEEE Symposium on Large-Scale Data Analysis and Visualization, November 2014.

[41] J. Woodring, K. Heitmann, J. Ahrens, P. Fasel, C.-H. Hsu, S. Habib, and A. Pope. Analyzing and visualizing cosmological simulations with Paraview. The Astrophysical Journal Supplement Series, 195(11), 2011.

[42] J. Woodring, J. Ahrens, T. J. Tautges, T. Peterka, V. Vishwanath, and B. Geveci. On-demand unstructured mesh translation for reducing memory pressure during in-situ analysis. Proceedings of the 8th International Workshop on Ultrascale Visualization, ACM, 2013.

[43] Y. Yan, et. al. Hierarchical place trees: A portable abstraction for task parallelism and data movement. Languages and Compilers for Parallel Computing, 2010.

[44] H. Yu, C. Wang, R. W. Grout, J. H. Chen, and K.-L. Ma. In situ visualization for large-scale combustion simulations. IEEE Computer Graphics and Applications, 30(3), 2010.