ETH: An Architecture for Exploring the Design Space of In-situ Scientific Visualization

Gregory Abram*, Vignesh Adhinarayanan†, Wu-chun Feng†, David Rogers‡, James Ahrens‡
*Texas Advanced Computing Center, The University of Texas at Austin, Austin, TX 78758
†Department of Computer Science, Virginia Tech, Blacksburg, Virginia 24061
‡Computer, Computational, and Statistical Sciences Division, Los Alamos National Laboratory, New Mexico 87545
{gda}@tacc.utexas.edu,{avignesh, wfeng}@vt.edu, {dhr, ahrens}@lanl.gov

***Abstract*—As high-performance computing (HPC) moves towards the exascale era, large-scale scientific simulations are generating enormous datasets. Many techniques (e.g., in-situ methods, data sampling, and compression) have been proposed to help visualize these large datasets under various constraints such as storage, power, and energy. However, evaluating these techniques and understanding the trade-offs (e.g., performance, efficiency, and quality) remains a challenging task.**

To enable exploration of the design space across such trade-offs, we propose the Exploration Test Harness (ETH), an architecture for the early-stage exploration of visualization and rendering approaches, job layout, and visualization pipelines. ETH covers a broader parameter space than current large-scale visualization applications such as ParaView and VisIt. It also promotes the study of simulation-visualization coupling strategies through a data-centric approach, rather than requiring coupling with a specific scientific simulation code. Furthermore, with experimentation on an extensively instrumented supercomputer, we study more metrics of interest than was previously possible. Importantly, ETH will help to answer important *what-if* scenarios and *trade-off* questions in the early stages of pipeline development, helping scientists to make informed choices about how to best couple a simulation code with visualization at extreme scale.

***Index Terms*—In-situ Techniques, High-Performance Computing, Design-space Exploration, Raycasting, Energy Efficiency**

I. Introduction

Power, storage, and data movement have emerged as first-order design constraints in supercomputing systems. For example, the U.S. Department of Energy imposed a power budget of 20-30 MW for candidate exascale systems [2]. The applications running on these machines are also limited by their aggregate I/O bandwidth (60 TB/s) and storage capacity (1000 PB) [2]. Such constraints hamper our ability to visualize and analyze extreme-scale datasets via conventional methods [25], [26]. In response, researchers have developed numerous techniques (e.g., in-situ methods [5], data sampling [34], compression [20], and job layout optimization [6], [29]) to address the challenges associated with large datasets. The goal of this paper is to *empower domain scientists in choosing the best approach for in-situ visualization from a complex space of design trade-offs.*

Scientists using traditional workflows have developed rules of thumb for how to sample, store, and visualize their data on the resources that have been available to them, and these rules of thumb are embodied in the current generation of visualization systems (e.g., Paraview [10] or VisIt [19]). These workflows consist of a computational science/simulation stage, followed by visualization and analysis as a decoupled post-processing stage. Further, current visualization systems generally utilize geometry-pipeline algorithms that leverage high-performance rasterization systems for real-time performance on workstation-level problems.

Abram and Adhinarayanan contributed to this work equally.

However, such rules may *not* apply in the exascale era for many well-recognized reasons. It becomes much too costly (in both time and energy) to write results to secondary storage between the computation and analysis stages, leading to the adoption of in-situ methods [5] that perform visualization and analysis of computation results as they are computed, rather than as a post-process applied to data that has been saved to disk. In effect, we replace the post-processing workflow of Figure 1 (top) with the in-situ workflow of Figure 1 (bottom) and work to address the coupling between the two (represented by the thick black line).

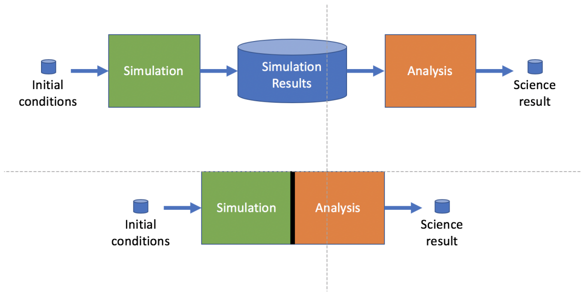


Figure 1 Pictorial representation of post-processing workflow (top) and in-situ workflow (bottom). Exploring the requirements of the design space of possible in-situ workflows is the focus of the ETH architecture.

In-situ methods minimize the need to write large datasets to secondary storage by processing the raw data into *extracts* that reflect the information contained in the raw data that is of actual interest to the scientist. For example, cosmology investigators use n-body smooth-particle hydrodynamics to model the development of the universe; while the algorithm tracks very large numbers of particles, the science is particularly interested in the distribution of *halos*, which are computed by statistically processing the evolving particles. Rather than save the particles themselves and run the statistics later, in-situ methods apply the statistics as the particles are evolved in the simulation and then generate the vastly smaller halo data.

Another example arises in visualizing simulation results. Rather than save the raw data of every time-step to disk, thus resulting in a huge dataset for later visualization into animation, we can compute and save a subset of time-step images as the data is computed. The resulting difference in the size of the raw data and time-step images is so large that we can compute and save many images in-situ in the time it takes to output the raw data [3].

In general, it is difficult to instrument simulation codes for in-situ analysis of all the cases that the simulation code is put through. For instance, MPAS-Ocean [28] can be used to study many aspects of the ocean; different applications of MPAS-Ocean will require different extracts — (1) the distribution and transfer of kinetic energy and (2) the distribution of chemical species through the ocean. Thus, it is impractical to insert the analysis directly into the complex simulation code for *each* use; instead, the developer codes to an interface that communicates with a customized analysis component to perform the case-specific analysis. This two-part architecture — a large simulation computation communicating via an interface to a potentially comparably-sized analysis component is at the heart of in-situ processing.

In-situ pipelines can be executed in many ways. Figure 2 shows four possible architectures of in-situ pipelines. In each case, we see the simulation and analysis positioned in processes residing in nodes in an inter-process communications domain. In the upper cases of Figure 2, the simulation and analysis are co-located on the same nodes, enabling them to communicate using shared memory. In these cases, the resources of the node are necessarily shared by the two tasks and each impacts the other; one can choose to divide the cores and memory between the two tasks, allowing them to run concurrently, or in time so that (for example) each has access to the full computational capability of the node in its turn.

The lower cases in Figure 2 instead use separate computational resources for each. Here we allocate separate sets of nodes for each workload and use the high-speed parallel inter-process communications capabilities of a supercomputer to move data from one set to the other. This, however, leads to a potential load-balancing issue since resources may be difficult to move between the two workloads; in some cases, the analysis may wait for the computation and vice versa.

Thus, the design-space for in-situ computation is large and highly multi-dimensional. Different choices may (should) produce the same results, but potentially at very different costs in time, power, energy, and storage. Importantly, the 'sweet spot' for a given case may be very dependent on the specifics of the case rather than the simulation and analysis codes themselves. Given a specific science workload consisting of simulation code, initial conditions, and analysis task, it is impractical to configure the simulation for the specifics of the analysis and then run alternative in-situ configurations for evaluation.

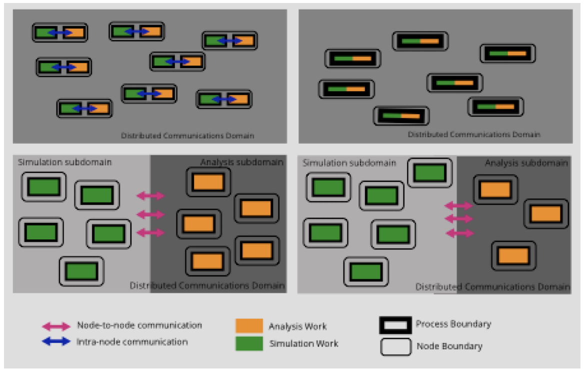

Distributed Communications Domain
Distributed Communications Domain
Simulation subdomain
Analysis subdomain
Distributed Communications Domain
Simulation subdomain
Analysis subdomain
Distributed Communications Domain
Node-to-node communication
Intra-node communication
Analysis Work
Simulation Work
Process Boundary
Node Boundary

Figure 2 Four alternative architectures for in-situ visualization: simulation and analysis co-resident in each processes; simulation and analysis in separate processes on same node; simulation and analysis in separate subsets of the distributed system with same and differing numbers of nodes for each.

Insights gained from our earlier experiments [1] have shown that when studying in-situ approaches, the behavior of the application can be largely factored out. We therefore can replace the simulation with a *proxy* for the simulation; a task that has access to *the same raw data that the simulation produce internally*, but which is much easier to reconfigure for different in-situ architectures. To do so, we make a preliminary run of the simulation itself on the science case, and write data out as if for simple post-processing analysis, but in abbreviated form to produce typical data but much more limited in scope. Our simulation proxy then reads the simulation data into memory and presents it to the simulation/analysis interface as if by the simulation itself. We can then easily test the analysis workload against real simulation data in varying in-situ architectures. In effect, we replace the standard abstract in-situ architecture given in Figure 1 (bottom) with the more flexible architecture given in Figure 3 while operating on data that is representative of that which the simulation will produce in a production run.

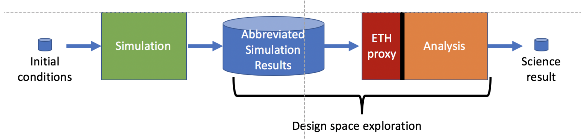


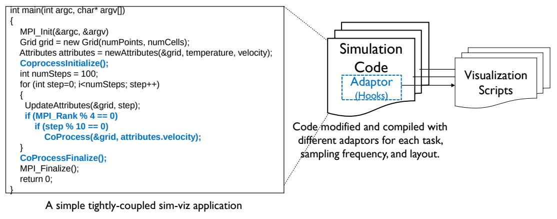
Figure 3 Diagram showing the ETH architecture approach to design space exploration. Simulation data is written to disk by a simulation application. That data can then be read into ETH's proxy appication, and tested in a variety of execution configurations.

A. Contributions

The specific contributions of this paper, and of the *Exploration Test Harness* (ETH) architecture are as follows:

- **A first-of-its-kind architecture for answering "what if" questions**. Our work will facilitate a deep understanding about the trade-offs among different operations, sampling, visualization pipelines, and coupling. ETH code is available at `https://github.com/ascr-ecx/eth`.

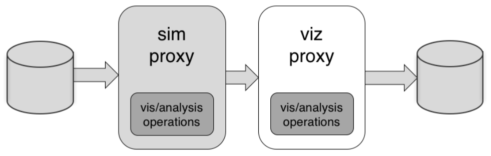
- **An architecture that runs on data, rather than requiring coupling to a science code**. While other frameworks can operate on data, the design of ETH ensures that we can even answer questions pertaining to coupling optimizations without real simulation-visualization code coupling (see Figure 4a). This means experiments can be run without investment in coupling the in-situ pipeline with a specific code - an important gain in flexibility.
- **Built-in exploration of multiple rendering approaches**. In particular, the toolkit includes a raycasting approach that operates on the raw data and a geometry-based approach that performs traditional triangle-based operations (see Figure 5).
- **Results from experiments on representative data types (grid and points) across different configurations**. In particular, we have conducted experiments on two classes of data — points and grids — that represent two important classes of data that extreme-scale applications create.



```
int main(int argc, char* argv[])
{
   MPI_Init(&argc, &argv)
   Grid grid = new Grid(numPoints, numCells);
   Attributes attributes = newAttributes(&grid, temperature, velocity);
   CoprocessInitialize();
   int numSteps = 100;
   for (int step=0; i<numSteps; step++)
   {
     UpdateAttributes(&grid, step);
     if (MPI_Rank % 4 == 0)
        if (step % 10 == 0)
           CoProcess(&grid, attributes.velocity);
   }
   CoProcessFinalize();
   MPI_Finalize();
   return 0;
}
```

Simulation Code
Adaptor
(Hooks)
Visualization Scripts
Code modified and compiled with different adaptors for each task, sampling frequency, and layout.
A simple tightly-coupled sim-viz application

a. Exploration with existing frameworks



b. Exploration with ETH

Figure 4 Existing frameworks require extensive modifications to a large code base for design-space exploration. With ETH, data is read from disk into *simulation proxies* and sent to *visualization proxies* where they are rendered to output artifacts. Design-space exploration requires changes only to configuration files and a single 260 source lines of code (SLOC) *simulation proxy*.

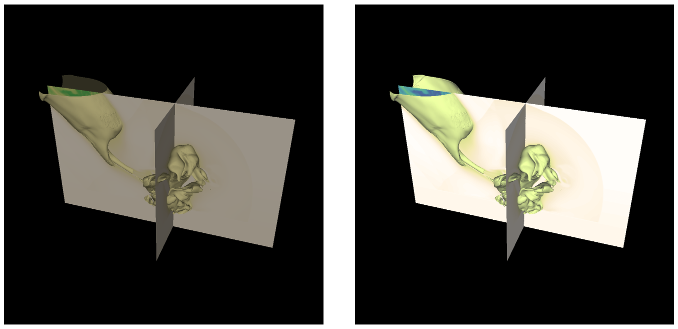
The rest of the paper is organized as follows. Section II discusses the related work. Section III presents an overview of our ETH architecture, and Section IV describes the parameter space explored in this paper. The experimental setup and results are presented in Sections V and VI, respectively. Section VIII concludes our paper.

II. Related Work

There are a number of mature visualization frameworks for in-situ and post-processing visualization of large-scale scientific data. Early instances of in-situ implementations focused on tightly integrating a visualization routine with a simulation routine (e.g., pV3 [15]). These implementations are typically customized for a specific purpose such as for monitoring or steering a simulation. Modern frameworks, on the other hand, provide a great deal of flexibility. Though useful, this flexibility means that contemporary visualization frameworks are largely ill-suited for rapid design-space exploration. A majority of the visualization frameworks can be classified into one of the following two categories:

- **Those designed for a specific purpose and narrow in scope.** `CUMULVS` [18], `EPSN` [8], and `SCIRun` [16] are task-specific for computational steering; `yt` [31] is data-type specific for AMR data; `Nessie` framework [23] is designed to be application-specific; and `QIso` [35] is for extracting and operating upon surfaces. Owing to their narrow focus, they are ill-suited for design-space exploration.
- **Those focused on integration with a real-world application rather than exploratory research.** Some examples in this category include `Catalyst` [4], `ADIOS` [24], `GLEAN` [32], `VisIt` [19], and `Damaris/Viz` [7], and `Libsim` [36]. While these frameworks provide a vast array of options to explore the in-situ visualization space, the effort required to use these tools for design-space exploration is significantly high. For instance, to get an in-situ setup running with `Catalyst` [11], one has to compile and configure `Catalyst`, the simulation code, and write a custom adapter. This added complexity makes this class of in-situ framework ill-suited for exploration of the design space.

Some of the other frameworks are designed to encourage exploration by easing the burden of integration through simplified design or by narrowing the tool's focus (e.g., by operating only with mini-apps). `Strawman` [21] belongs to



a. Raycast image   b. Geometry-based image

Figure 5 Sample images from datasets tested and rendered with ETH. Different rendering pipelines, coupling strategies, and sampling approaches can all be tested in many configurations with the toolkit. Here we show both a raytraced and geometry-based rendering of an asteroid impact from an xRAGE simulation.

this category with one of its stated goal being to explore in-situ solutions. Much like Catalyst, it also requires integration with applications (or mini-apps) by writing adaptors. Like the full-fledged visualization toolkits, using such tools for design-space exploration would also be time-consuming due to the additional time cost involved in writing adaptors, compiling and running the simulation. ETH simplifies the evaluation process by eliminating the need for writing adaptors, compiling and running the simulation, and coupling data management frameworks (e.g., ADIOS and Damaris) with visualization frameworks (e.g., ParaView/Catalyst and VisIt).

Why not use existing toolkits in the "post-processing" mode and directly operate on the data? In our earlier design-space experiments, we attempted to use ParaView/Catalyst to perform the study directly on the data. However, our experiences with the setups presented in Figure 4a showed that these setups are not appropriate for ParaView/Catalyst, which also lacks good transport mechanisms for the data. One would have to setup Catalyst with a data management framework like ADIOS or write one's own transport code, neither of which are simple tasks [12]. Furthermore, getting custom rendering working with existing frameworks is a significantly more involved process compared to a tool such as ETH.

III. Exploration Test Harness (ETH)

The Exploration Test Harness (ETH) is a lightweight, open-source testing harness that promotes exploration of a variety of analysis and visualization pipelines in many different operations, work distributions, and mappings onto hardware. The toolkit is based on VTK [30], which is a core capability for the analysis and visualization community. VTK implements a data-centric pipeline of operators, filters and rendering operations that operate on data, then pass it along to the next element in the pipeline. There are several important capabilities of the test harness, which are discussed below. These capabilities are the classes of parameters that are varied in the tests run for this paper.

ETH operates on data and does not require code coupling. ETH loads test data from disk, and this has several advantages over other systems. First, ETH can run on many different science domains, without changing the toolkit. Because it is based on VTK, any dataset that VTK reads can be used in tests. Existing toolkits tend to be based on *proxy apps*, which contain simplified versions of the physics of their related simulation. Operating on real data is critical to testing the visualization and analysis operators as simulated data does not generally contain enough complexity to test and exercise interesting analysis and visualization operations.

ETH has easily configurable visualization operations. A critical feature of a useful testing system is a way to modify analysis and visualization operations so that they approximate those that are to be tested. Again, since ETH is based on VTK, many operations can be easily added to the pipelines tested, and they can be specific to the data and visualizations that are of interest. This means that ETH can generate specific visualization products that serve as good proxies for perceptual and cognitive test products.

ETH can run with different process-couplings. Given the number of different hardware and process options available today, it is useful to study different configurations of parallel jobs. ETH can be configured to run pairs of *simulation proxy* and *visualization proxy* processes in three ways: 1) coupled into a single process, or 2) as communicating processes, which enables the processes to run on the same nodes, thereby avoiding any I/O operations, or 3) on different nodes, enabling testing using differing numbers of nodes and on heterogeneous systems.

ETH can run different rendering pipelines. Traditional visualization workflows use algorithms that iterate over the input data to extract intermediate geometrical representations of the data that can then be rendered using OpenGL, which then iterates over the intermediate data to determine each element's contribution to the output image. Recent technical advances [17], [27], [33] make it practical to support raycasting renderers that operate directly on data, avoiding the need for intermediate representations and the memory space they require. Further, in many cases, raycasting can produce significantly better images at lower cost, particularly as datasets get large. Since the choice of visualization back-ends significantly affects a system's overall performance and the effectiveness of the results, ETH can be configured for alternative rendering back-ends as shown in Figure 6, enabling this dimension to be explored as it affects the overall system performance.

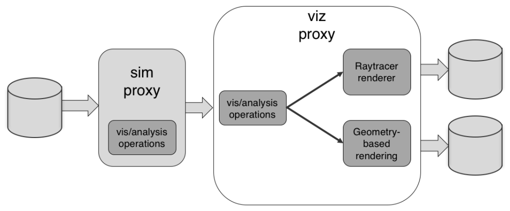


Figure 6 Diagram showing options for pipeline execution inside the visualization process. We can send the result of some set of operations to different rendering pipelines. In this case, we can compare a traditional geometry-based rendering approach with a raycasting approach that operates directly on the data, and takes advantage of new software rendering libraries that are optimized for this.

A. Design of ETH

ETH uses a simulation proxy, executing in place of the simulation itself. The basic unit of granularity is a pair of processes, each of which can perform analysis or visualization operations on real simulation data as shown in Figure 4b. The ETH simulation proxy reads data from the disk and then operates on the data in parallel. An experiment can then be run over different process or node configurations, and different rendering pipelines to produce artifact on disk. The system can accommodate a wide range of analysis and visualization operations in either a simulation process or an analysis process, so the toolkit can be customized to operate on a user's specific data, and perform visualizations that are simplified or specifically tuned to the user's work flow. Modes include unified (single process), core-to-core, and node-to-node. There are two rendering pipelines supported as well – raycasting (non-geometric) and geometry-based rendering.

B. ETH Proxy and User-generated Simulation Data

Because each simulation typically has its own native data format, our design requires that the data is exported as VTK data objects. Instrumenting simulations to produce VTK objects is well understood, and mature toolkits are available (Catalyst, for Paraview and libsim for Visit). This means there is a ready path for a user to export new types of simulation results in this format.

The ETH simulation proxy requires that each parallel process of the proxy is able to load the data that it will pass to the in-situ interface. Thus the idealized architecture of Figure 3 is better represented in the current implementation as Figure 7. ETH requires that the input consists of VTK data. This means that users can adapt their specific simulation data to a common format and can take advantage of the ETH capabilities, regardless of the scientific domain of the user data.

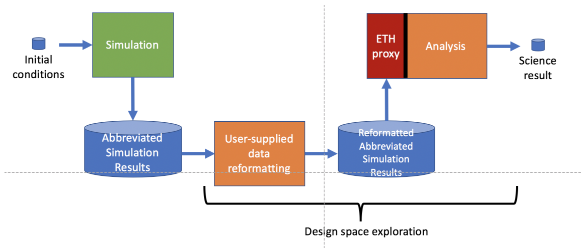

Initial conditions
Simulation
Abbreviated Simulation Results
User-supplied data reformatting
Reformatted Abbreviated Simulation Results
ETH proxy
Analysis
Science result
Design space exploration

Figure 7 Experiment workflow for ETH with user-supplied data reformatting

C. Execution Details

As described above, ETH runs with pairs of simulation and visualization proxies coupled together into one process, or running in separate processes and communicating via the socket layer. In the first case, experiments are easily run using the standard batch scheduler.

When the simulation and visualization proxies run in *separate* processes, the two sets of proxy processes must exchange information that enables the visualization proxy processes to connect to their paired simulation proxy processes. This is done by starting the parallel application in two steps: first, the simulation proxy application is started. Each process of the application then adds its assigned IP address and port number to a globally accessible layout file, then opens its port and waits for connection. The visualization proxy application is then started. Each process of the application then references the global layout file, determines the location of the simulation proxy(s) it will receive data from, waits for the corresponding port to open, and then establishes the connection.

When the simulation and visualization proxy applications run on the same nodes, this is again easy to start: the job script simply begins the simulation proxy application first, in the background, and then starts the visualization proxy application in the foreground.

When the simulation and visualization proxy applications run on separate sets of nodes, several alternatives exist depending on the mix of nodes required; on homogeneous systems, a single job allocating the total number of nodes required, and MPI arguments can be used to start the two parallel processes offset from one another. When heterogeneous collections of nodes are desired, it will be up to the scheduling system to arrange for two separate jobs to be started concurrently.

IV. Methodology

There are several configurable parameters that affect the performance, power, energy, and quality of the images produced by the visualization process. In this paper, we study three such parameters that we believe will lead to insightful results. The rest of this section describes these parameters and the applications used in this study.

A. Applications and Input Datasets

In this paper, we explored two different dataset—a point-based dataset from a cosmology simulation and a grid-based dataset from an asteroid impact simulation. These are representative of large classes of science datasets of interest at extreme scale. The applications and datasets are described in detail next.

Cosmology Simulation (HACC) Hardware/Hybrid Accelerated Cosmology Code (HACC) simulation is a cosmological n-body simulation used to study the evolution of the universe [14]. For our experiments, we use four datasets from the dark sky simulation on 400 nodes. 1 billion particles from the n-body simulation are passed on as input to the in-situ visualization engine for the largest data set each time step. 500 images are rendered in each time step. Each particle's data is composed of its ID, position vector, and velocity vector. The visualization task here is to render the point-cloud data in a manner that makes visual identification of halos easy. The other problems operate on 750 million, 500 million, and 250 million particles, respectively.

Asteroid Simulation (xRAGE) Radiation Adaptive Grid Eulerian (xRAGE) is a radiation-hydrodynamic code used for solving a variety of high-deformation flow problems involving a radiative transfer of energy (e.g., underwater blast) [13]. While the simulation itself normally uses adaptive mesh refinement (AMR) method, the AMR data is typically converted to an unstructured grid data which is then downsampled to a structured grid data before being handed off to the visualization code. The visualization task is to represent a grid of variables such as pressure, density and (in our case) temperature. We run three problem sizes of this application on 216 nodes. The small problem operates on a 610x375x320 grid and processes 7.91 GB/timestep, medium operates on a 1280x750x640 grid and processes 2.46 GB/timestep, and large operates on a 1840x1120x960 grid and processes 292 MB/timestep. Twelve time steps are processed with two sliding planes and a varying isovalue for 1000 images. For strong scaling, 100 images are rendered for each timestep.

B. In-situ Parameters

Once the data have been read into the *simulation proxy*, We study two in-situ methods, namely, sampling and coupling strategy. The options for each parameter are described next.

Sampling Technique Spatial sampling is explored which operates by selecting a subset of points (down sampling) from the original dataset based on some given distribution. We vary the sampling ratio (i.e., number of selected points from the given "raw" data) and study how the metrics included in this study changes.

Coupling Strategy We explore three work-distribution or sim-viz coupling strategies.

- *intercore* coupling – Simulation and visualization processes are time-shared and alternate on same set of nodes.
- *internode* coupling – The processes are space-shared with the simulation process running on half the number of allocated nodes and the visualization process running on the remaining nodes
- *tight* coupling – the visualization and simulation processes are merged to create a single, unified process for running a scientific workflow

These coupling strategies are expected to have distinct advantages based on the scalability of the rendering technique and data size. We seek to quantify the impact of the coupling strategy on our selected metrics.

C. Rendering Parameters

Once the data have passed through the operations requested by the test designer, they are rendered to disk. There are two pipelines available in ETH: (1) a geometry-based renderer, and (2) a raycasting renderer. The geometry-based rendering pipeline utilizes VTK to generate geometry from the data, and then render that geometry in a variety of ways. The raycasting method operates directly on the data rays from a light source (e.g., camera) to all the pixels in the image plane placed in front of the object. If the ray passing through the pixel hits an object, the color for that pixel is calculated and rendered. If the input object is a point cloud, the data is preprocessed into an appropriate data structure so that it becomes possible for the rays to hit an "object." Note that unlike the other two methods, this method is dependent on the number of rays cast, once the initial data structure is built.

Rendering Methods for HACC data The HACC data consists of a very large number of particles; essentially, points in 3D space. Ideally, these points will be rendered as spheres, so that distant particles will be smaller to give the appearance of distance. This case presents a particularly difficult case for traditional geometry-based visualization, since modeling each of a billion or more spheres using sufficient triangles to give the appearance of roundness is impractical. Instead, we use two alternative geometry-based rendering — VTK points and Gaussian splatter — for evaluation.

Geometry-based: VTK Points. This is the simplest of the techniques used in this paper. The technique operates on a collection of points (also referred to as a *point cloud*). Each point in the collection is described by its position in the *x-y-z* plane. Usually, the location of the point in the 3-D space is also accompanied by a scalar field. The technique operates by mapping every point in the 3-D space to the 2-D plane and rendering every pixel with a fixed size (usually 1 to 3 pixels on a side) and fixed color block. This normally results in a loss in 3-D perception.

Geometry-based: Gaussian Splatter. This technique creates a single triangle, sized to reflect a given radius, to represent each point. At rendering time the triangle is transformed to the proper location in eye-space, oriented toward the viewer, and rendered to the screen using a specialized shader function that manipulates the triangle normal at each pixel to model a sphere. In its current implementation this leads to some unfortunate artifacts, but it does limit the amount of geometry required to represent the data for rendering.

Raycast Spheres. This case is particularly well-suited to raycasting. Each particle is represented as a 3-D point and a world-space radius, and placed into an specialized acceleration structure at a cost of roughly O(NlogN). At run-time, the acceleration structure is traversed to determine whether the viewing rays (that is, rays begun at the viewpoint passing through each pixel in the image plane) strikes a sphere with a cost that is sub-linear in the number of particles. If a ray *does* intersect a sphere, a simple geometric calculation produces an intersection depth and orientation for shading.

Rendering Methods for Volumetric Data. The asteroid dataset is scalar and volumetric; the data represents the temperature field in the vicinity of the steroid strike. In our tests we use two traditional visualization techniques that are widely used in such cases: slicing planes and isosurfaces.

Slices and Isosurfaces in Geometry-based Visualization. To visualize slices and isosurfaces of volumetric data, geometry-based visualization systems must first generate geometry representing the slice or isosurface as a set of triangles, which are then rendered using a standard OpenGL pipeline. Generating such geometry consists of two steps: first, identifying the cells of the data grid that contain fragments of the surface, and then determining the geometry within those cells. While efficient algorithms attempt to minimize the number of cells that intersect the surfaces, a large number of the input cells will be examined and and a very large amount of geometry will often be created. In the case of slicing planes, the work and resulting data size is proportional (roughly) to the 2/3 root of the input data size, while in the case of isosurfaces the work is proportionate to the data size and the size out geometry set can range from zero to proportionate to the input data size.

Slices and Isosurfaces in Raycasting. Again, raycasting provides an attractive alternative to geometry-based visualization in these cases. The intersection of an arbitrary ray with an implicitly defined plane to produce a hit point in data space is O(1), and in the case of structured grids looking up the corresponding data value is also O(1), so the cost of rendering slicing planes is O(number of pixels). Isosurfaces are rendered by iterating along each view ray, sampling to find the data value for each iteration, and looking for crossings. Once a crossing is found, a hit point can be interpolated. Note that the appropriate sampling along the ray is proportionate to the resolution of the data in 1-D, so the cost of each ray is proportionate to the 1/3 root of the input data size.

V. Experimental Setup

In this section, we describe the hardware and software setup used for our experiments. Further, we describe the metrics used for evaluation.

A. Hardware Setup

We run our experiments on *Hikari*, a 432-node HPE Apollo 8000 cluster, which is capable of delivering over 400 TFLOPs of peak performance. Each node of the cluster is equipped with *two* 12-core Intel Haswell E5-2600v3 processors operating at 3.5 GHz and up to 64GB of RAM per node. The nodes are interconnected by Mellanox EDR infiniband using a fat tree topology. The cluster operates on high-voltage direct current (HVDC) instead of the traditional alternate current thereby avoiding four AC/DC conversions. This results in the cluster consuming significantly lesser power at both idle and active modes than a similar-sized cluster operating on alternate current. The system is extensively metered to allow quantitative studies. Apollo 8000 system manager is used to record records power every 5 seconds on a half-rack basis. We use *TACC stats* [9], a low-overhead monitoring infrastructure, to collect hardware performance counter data, which we use for analyzing our results.

B. Software Setup

The software setup is as follows: we use CentOS Linux 7.2.1511 running kernel version 3.10.0-327.13.1 IMPI 5.1.2 is used to parallelize our jobs across sockets, TBB 4.4.1 is used for threading, Intel ISPC is used for vectorization.

C. Metrics of Interest and Measurement Techniques

- **Power.** HPC systems are increasingly power limited. Therefore, we wish to examine whether different visualization algorithms, sampling, and data layout plays a role in affecting the power consumption of the HPC system. To measure power, we use the Apollo 8000 system manager which samples instantaneous power and records the average power every 5 seconds. From this power profile, we calculate and report the power consumed over the period of one entire run of an application configuration.
- **Energy.** With carbon emissions of IT infrastructure becoming a major concern across the world, it becomes important to document the energy consumed by large jobs running on HPC system. In this paper, we calculate energy based by multiplying the average power reported by Apollo 8000 system manager with execution time.
- **Performance.** Performance is reported as execution time which is calculated by subtracting the wall time upon the completion of the job from the wall time at the time of the start of the job.
- **Scalability.** In-situ techniques *can* result in additional execution time on a supercomputing cluster (a large resource) rather than on a scientist's workstation (a small resource). While the end-to-end time for a given workflow ultimately matters, understanding how the visualization algorithms scales on a large cluster helps us develop good job partitioning strategies. Therefore we calculate and report scalability which is the ratio of execution time of a visualization algorithm running on N nodes to the execution time on 1 node.

VI. Experimental Results

In this section, we present experimental data for the three datasets previously described. The primary goal of this section is to demonstrate the usefulness of our design-space exploration toolkit rather than perform an in-depth characterization study.

A. Cosmology Simulation (HACC)

We present the performance, power, energy, and relative accuracy of visualization artifacts here.

Algorithms. Table I presents the execution time and average power consumption for the *raycasting*, *Gaussian splat*, and *VTK points* for the *large* dataset in our test suite. As shown in Table I, the execution time for the Gaussian splat algorithm is 36% lower than VTK points which itself is lower than raycasting by 42%. An analysis of performance counter data shows that the raycasting algorithm performs significantly more computations than the other two algorithms for the problem size considered. Most of the additional computations come from an additional setup phase where an acceleration structure is built for the first time. Gaussian splat executing faster than VTK points is an unintuitive result as they both render all the points presented to them as input, but Gaussian incurs an additional step where the points are splatted to nearby voxels. Thus, in theory, this algorithm performs more computations than VTK points. We attribute its fastness to a superior implementation of Gaussian splat. Note that while the raycasting algorithm is the slowest, the quality of the images rendered by this algorithm tends to be better than the other two algorithms. Quantifying the perceptive value of the image produced, however, is an active research problem beyond the scope of this paper.

Finding 1. For HACC running on a mid-sized cluster (400 nodes), Gaussian splat is faster than VTK points which in turn is faster than raycasting.

The power consumption remains relatively constant across the three algorithms with only Gaussian splat exhibiting a

TABLE I: Visualization Algorithm Results for HACC

Algorithm	Time (s)	Power (kW)
Raycasting	464.4	55.7
Gaussian Splat	171.9	55.3
VTK Points	268.7	55.2

marginally higher consumption as shown in Table I. However, this marginal difference is insignificant and should be attributed to noise in the measurement. Similar power consumption for the algorithms is understandable as we expect the resources to be utilized to the same levels. Raycasting is fully parallelized – MPI is used to parallelize across the nodes, Intel TBB is used to parallelize across the cores within a node, and Intel ISPC compiler is used to vectorize across SIMD lanes; same is the case with VTK library implementations. As long as the problem size remains large enough to occupy all the resources, the implementations will utilize the hardware to similar levels and consume the same power. With power being effectively the same across the three algorithms, the energy consumption tracks performance (not explicitly shown in Table I).

Finding 2. Power consumption remains nearly the same for raycasting, Gaussian splat, and VTK points for HACC.

Scalability of the algorithms with data size. We present the *normalized* execution time for the three algorithms as we change the data size and fix the number of nodes in Figure 8. The normalization is performed against the smallest dataset within each algorithm. Gaussian splat and VTK points show a linear increase in execution time as the problem size increases as they both run in O(n). Of these two algorithms, VTK points scale better than Gaussian splat. However, for all the problem sizes considered in this study, Gaussian splat continued to outperform VTK points. Raycasting, unlike Gaussian splat and VTK points, shows a sub-linear increase in execution time as the performance of raycasting is not dependent on the number of points, but the number of rays. While the initial structure-generation phase of raycasting is affected by the number of points, all the other stages depend on the number of rays cast which remains constant. Therefore, we expect raycasting to outperform the other two algorithms for problem sizes larger than those considered in this paper.

Finding 3. Geometry-based methods exhibit a significantly different scaling curve when compared to geometry-free method which indicates that the optimal algorithm is dependent on the problem size or cluster size.

This finding is also corroborated by Larsen et al. where they report that raycasting outperforms rasterization for larger problem sizes but not for smaller problems [22]. Our toolkit makes such discoveries easier as it removes the necessity to integrate visualization frameworks with simulation codes.

Sampling Techniques. Figure 9 presents the performance, dynamic power, and energy consumption for four different spatial-sampling ratios for the three different algorithms. Spatial sampling ensures that fewer points are rendered, so the

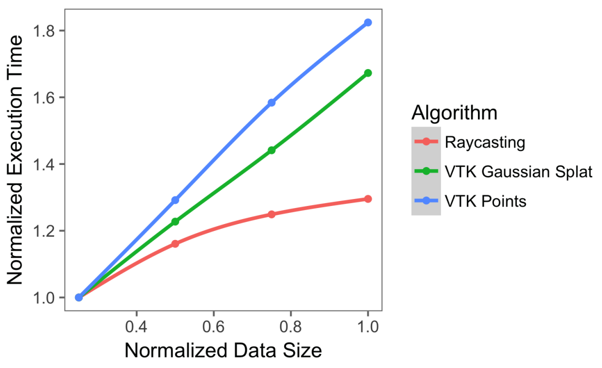


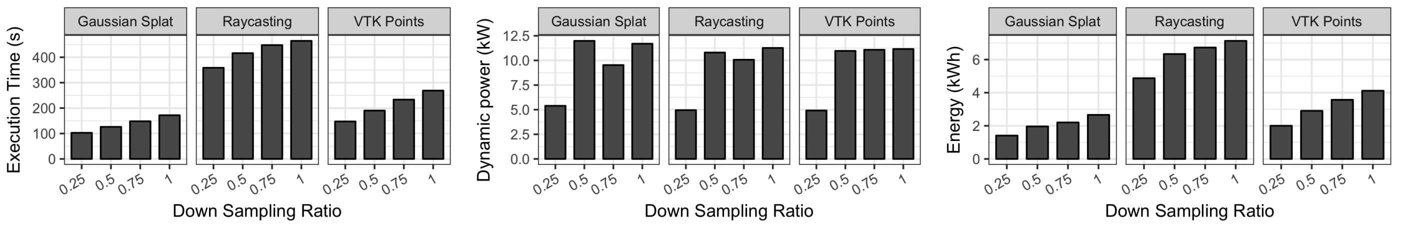
Figure 8 Scalability of raycasting, Gaussian splat, and VTK points with respect to problem size. Results are normalized to the smallest problem size for each algorithm.

execution time goes down as the sampling ratio increases as shown in Figure 9a. The result worth noting here is the dynamic power consumption for the different sampling ratios shown in Figure 9b. At low sampling rates (less than 0.5), we see a significant reduction in power consumption. More specifically, the total power consumption at a sampling rate of 0.25 is 11% lower than the power consumption for the full dataset (i.e., sampling ratio = 1.0). This value actually corresponds to a 39% reduction in the dynamic power of the system. Our hypothesis is that at this point we have reduced the problem size to a level where it becomes difficult to keep all parallel resources busy thus affecting the resource utilization.

Finding 4. Spatial sampling can reduce system power consumption for HACC.

A side effect of reducing sampling ratio is a reduction in the quality of image which may reduce its perception value. We also quantify the error introduced by the sampling technique using the root mean square error (RMSE) metric in Table II. While the energy saved increases with loss in accuracy, as expected, the trade-off curve between the two differ across algorithms significantly. For example, VTK points show the most resilience to error when spatial sampling is applied to a dataset. Its RMSE is as low as 0.13 when one in four points are sampled, whereas for the other algorithms, it is at least 0.42. However, the baseline image (i.e., no sampling is applied) for VTK points is generally worse than the other two. We can also observe from the table that raycasting shows more tolerance to errors at higher sampling rates, but the errors creep up as we sample less. While we have used a simple metric, in practice, we expect users of the toolkit to use more sophisticated metrics explicitly targeted at measuring the perception quality of an image.

Strong Scaling Results. Figure 10 shows the results for two different node counts. For brevity, we discuss only the "full" dataset. The other datasets within the cosmology application follow similar trends. In this figure, we observe that the performance of the raycasting algorithm improves only slightly when we increase the node count as shown in Figure 10a. The



a. Performance b. Power c. Energy

Figure 9 Performance, power, and energy consumption for four different spatial sampling configurations for the cosmology application

TABLE II: Trade-off between accuracy and energy for HACC

Algorithm	Sampling Ratio	0.75	0.50	0.25
Raycasting	RMSE	0.17	0.28	0.42
	Energy Saved	17.4%	28.1%	41.5%
Gaussian Splat	RMSE	0.33	0.37	0.43
	Energy Saved	17.2%	26.3%	47.0%
VTK Points	RMSE	0.04	0.08	0.13
	Energy Saved	13.3%	29.4%	51.4%

average power consumption when 200 nodes are used is nearly 50% lower than when 400 nodes are used resulting in a similar magnitude of energy saved as shown in Figures 10b and 10c. This is true for all three algorithms studied.

Finding 5. For the problem sizes considered, visualization algorithms such as raycasting, Gaussian splatter, and VTK points exhibit poor strong-scaling.

An implication of the above finding is that the commonly used coupling strategy, where simulation and visualization alternates on all the nodes of a supercomputer, may not be the most energy-efficient, as the additional nodes largely "waste" power and energy. We posit that a better way to distribute work is be to allocate a small number of nodes for visualization and the remaining nodes for simulation and space share the nodes between the two.

Coupling Strategies. We verify our hypothesis by performing an experiment with different coupling strategies. As shown in Figure 11, tight- coupling strategy (tightly coupled and intercore coupled) is not always optimal both in terms of performance and energy consumption. This set of results belies expectations and underscores the need for experiment- and data-driven decision making. Our software toolkit along with the experimental hardware platform makes experimentation and data collection easier and facilitates rapid design-space exploration.

Finding 6. Proximity between the simulation and visualization routines does not necessarily equate with optimality as evidenced by the intercore coupling which outperforms the other coupling strategies for the HACC application.

B. Asteroid Simulation (xRAGE)

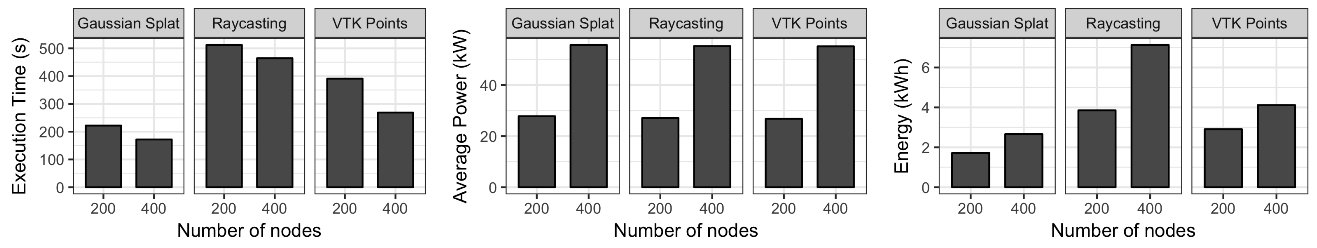
Experimental results are discussed for the xRAGE application in this section. Since this application operates on a structured grid instead of HACC's point-based data, the algorithms considered for evaluation differs.

Algorithms. Figure 12 shows the performance, power, and energy consumption for VTK's geometry-based isosurface algorithm (vtk) and raycasting. As we can see from this figure, vtk takes 28% more time than raycasting to generate the isosurface. While VTK's implementation consumes lesser power than raycasting (Figure 12b), it is offset by a significant increase in execution time resulting in higher energy consumption for VTK as shown in Figure 12c.

Scalability of algorithms with problem size. Though both VTK's and raycasting's execution time increases in O(Data Size), the slope of the line is significantly different. As shown in Figure 13, a 27-fold increase in problem size resulted in VTK taking 5.8 times longer to execute, whereas for raycasting it was only a 1.35-fold increase. In fact, VTK executed faster for the smallest problem size, but the trend reversed when the data size was increased by a factor of 2 in all three dimensions of the 3D grid.

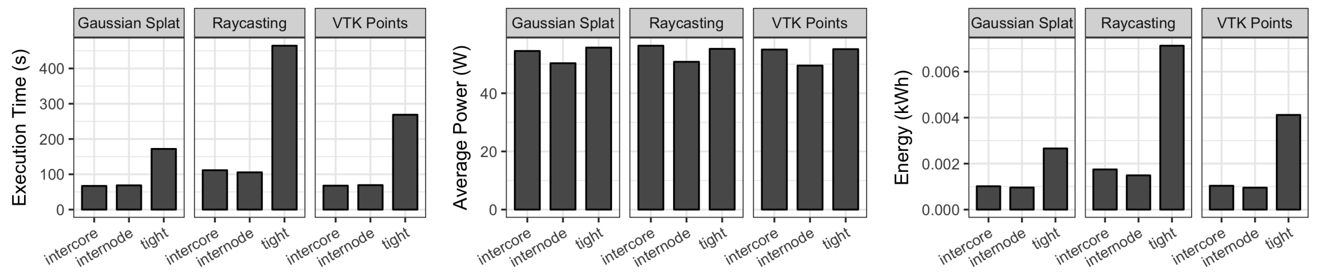
Sampling Technique. We previously identified spatial sampling as a technique to reduce power consumption. Here we run the sampling experiments with the asteroid dataset. As observed in Figure 14b, power consumption does not reduce with sampling ratio even when the sampling ratio is reduced to 0.04 (compared to up to 0.25 in the cosmology application). This indicates that optimization techniques (for both power and performance) are not common across domains. While sampling helped reduce power for HACC, it only helps in reducing energy for xRAGE (shown in Figure 12c).

Strong Scaling Results. Figure 15 shows an interesting trend. In this graph, we plot the normalized performance (proportional to the reciprocal of execution time) versus the number of nodes used in the experiments. We varied the node count from 1 to 216. We can see that the raycasting algorithm scales well. When we double the number of nodes, the performance roughly doubles, which is its expected property. VTK on the other hand, does not only fail to scale, but actually shows performance



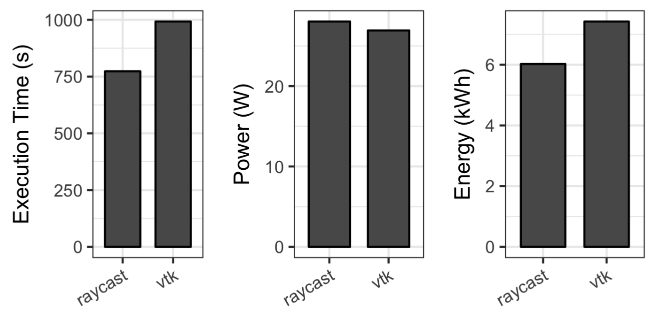
a. Performance b. Power c. Energy

Figure 10 Experimental results for two different node counts for the cosmology application



a. Performance b. Power c. Energy

Figure 11 Experimental results for different simulation-visualization coupling strategies for the cosmology application



a. Performance b. Average Power c. Energy

Figure 12 Execution time and average power consumption of raycasting and vtk algorithm for the xRAGE application

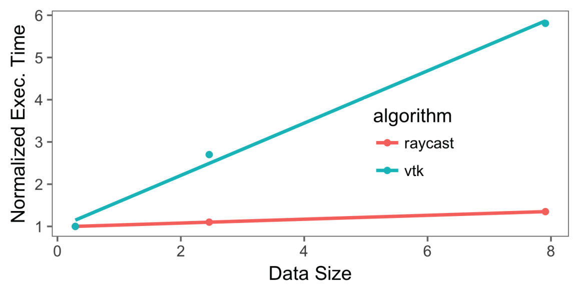


Figure 13 Scalability of raycasting and geometry-based (vtk) methods with respect to problem size. Results are normalized to the smallest problem size for each algorithm.

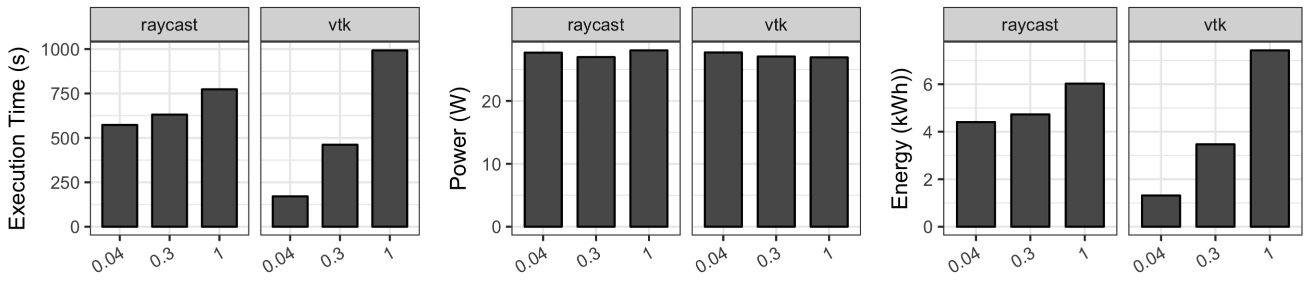
degradation beyond a point. We think this is due to some form of contention in a shared resource arising from parallelism.

Finding 7. Similar to HACC, scaling curve for geometry-based and geometry-free methods are different for HACC. Therefore, the optimal method for rendering depends on problem size and cluster size. For the largest data considered in this paper, raycast started outperforming VTK when the node count is 64 or more.

VII. Discussion

In this section, we discuss the generality of our results. Noting that some of our results are *not* broadly applicable, we explain how a domain scientist would extend ETH to conduct studies on other domains.

Generality of results. Our experimental study on HACC (a representative of particle-based data) and xRage (a representative of structured grid data) revealed the following: raycasting scales better than geometry-based methods especially at high node counts. With exascale systems projected to consist of hundreds of thousands of nodes, we expect raycasting to be an important part of the domain scientists' toolkits, especially in the context of in-situ workflows. Nevertheless, none of the algorithms studied in this paper exhibited good strong scaling. This has important implications on how simulation and visualization routines are coupled. In our studies, we found that the optimal coupling strategy is specific to the application, algorithm, and data size. Scientists would need to conduct separate experimental studies for each application domains and analyze the results to identify optimal coupling



a. Performance b. Power c. Energy

Figure 14 Performance, power, and energy consumption for three different spatial sampling configurations for the xRAGE application

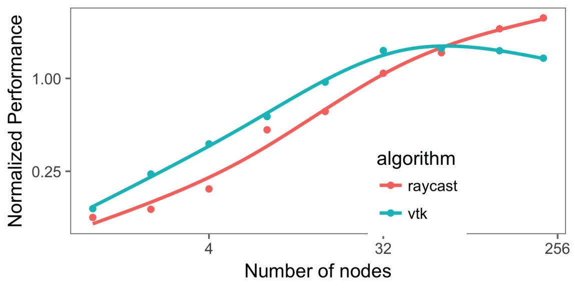


Figure 15 Strong scaling curve for raycasting and VTK geometric rendering for the xRAGE application.

strategies. We believe a tool built based on ETH's architecture would help speed up such an analysis study. Beyond coupling strategies, other techniques such as spatial sampling consistently helped improve performance and reduce power consumption. We anticipate that spatial sampling techniques will help in future scientific workflows to both improve performance and reduce power consumption.

Given that our experimental results showed that the optimal coupling strategy is highly specific to the application under study, the visualization algorithm, and the dataset size and sampling technique, one would have to extend ETH for other domains such as unstructured grid. To conduct studies on other domains, as a pre-processing step, one would need to run the simulation to collect data sets and partition the data thus collected. The job layout (i.e., where the visualization and simulation proxies are run) is specified in a separate file. If necessary, the visualization proxy is extended to include any new algorithm that the user may wish to study. For subsequent exploration of a different layout, the user simply changes the job layout file.

VIII. Conclusion and Future Work

In-situ methods are becoming an indispensable part of the scientific workflow at extreme scale and the number of options for these methods is exploding. With HPC systems being severely constrained by power/energy, storage, and I/O, it has become very important to gain a thorough understanding of the trade-offs among many in-situ approaches. Our experience with ETH and traditional full-featured softwares strongly indicate the need for a light-weight mechanism to quickly explore large parameter spaces so as to make informed choices about setting the visualization pipelines at extreme scale. Our initial experiments already point to important directions in designing a visualization workflow in order to save time, power, energy, while still obtaining good-quality visualizations. We expect ETH to expand the type of toolkits available to a domain scientist, by introducing a testing toolkit in addition to the existing production-oriented visualization frameworks.

Acknowledgment

This material is based upon work supported by Dr. Lucy Nowell of the U.S. Department of Energy Office of Science, Advanced Scientific Computing Research under Award Numbers DE-SC0012513, DE-SC0012637, and DE-SC-0012516. We would like to acknowledge Galen Gisler (LANL) for the asteroid simulation and contributors to the hardware/hybrid accelerated cosmology Code (HACC). We thank Lucas A. Wilson (TACC) for his assistance in collecting power data. We also thank the anonymous reviewers whose comments helped improve this manuscript. This work was published under LANL release LA-UR-17-26715.

References

[1] V. Adhinarayanan, W. Feng, D. Rogers, J. Ahrens, and S. Pakin. Characterizing and modeling energy for extreme-scale in-situ visualization. In *Proceedings of the 31st International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2017.

[2] S. Ahern, A. Shoshani, K.-L. Ma, A. Choudhary, T. Critchlow, S. Klasky, V. Pascucci, J. Ahrens, W. Bethel, H. Childs, et al. Scientific Discovery at the Exascale: Report from the DOE ASCR 2011 Workshop on Exascale Data Management, Analysis, and visualization. *Dept. of Energy Office of Advanced Scientific Computing Research*, 2011.

[3] J. Ahrens, S. Jourdain, P. OLeary, J. Patchett, D. H. Rogers, and M. Petersen. An image-based approach to extreme scale in situ visualization and analysis. In *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 424–434, Nov 2014.

[4] U. Ayachit, A. Bauer, B. Geveci, P. O'Leary, K. Moreland, N. Fabian, and J. Mauldin. Paraview Catalyst: Enabling In Situ Data Analysis and Visualization. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, pages 25–29. ACM, 2015.

[5] A. C. Bauer, H. Abbasi, J. Ahrens, H. Childs, B. Geveci, S. Klasky, K. Moreland, P. O'Leary, V. Vishwanath, B. Whitlock, and E. W. Bethel. In Situ Methods, Infrastructures, and Applications on High Performance Computing Platforms. *Computer Graphics Forum*, 35(3):577–597, 2016.

[6] J. C. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, et al. Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–9. IEEE, 2012.

[7] M. Dorier, R. Sisneros, T. Peterka, G. Antoniu, and D. Semeraro. Damaris/viz: A Nonintrusive, Adaptable and User-friendly In Situ Visualization Framework. In *2013 IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)*, pages 67–75. IEEE, 2013.

[8] A. Esnard, N. Richart, and O. Coulaud. A steering environment for online parallel visualization of legacy parallel simulations. In *Distributed Simulation and Real-Time Applications, 2006. DS-RT'06. Tenth IEEE International Symposium on*, pages 7–14. IEEE, 2006.

[9] T. Evans, W. L. Barth, J. C. Browne, R. L. DeLeon, T. R. Furlani, S. M. Gallo, M. D. Jones, and A. K. Patra. Comprehensive Resource Use Monitoring for HPC Systems with TACC Stats. In *Proceedings of the First International Workshop on HPC User Support Tools (HUST)*, pages 13–21, 2014.

[10] N. Fabian, K. Moreland, D. Thompson, A. C. Bauer, P. Marion, B. Gevecik, M. Rasquin, and K. E. Jansen. The paraview coprocessing library: A scalable, general purpose in situ visualization library. In *2011 IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 89–96. IEEE, 2011.

[11] N. Fabian, K. Moreland, D. Thompson, A. C. Bauer, P. Marion, B. Gevecik, M. Rasquin, and K. E. Jansen. The ParaView Coprocessing Library: A Scalable, General Purpose In-Situ Visualization Library. In *2011 IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 89–96, Oct 2011.

[12] B. Geveci. Personal Communication.

[13] M. Gittings, R. Weaver, M. Clover, T. Betlach, N. Byrne, R. Coker, E. Dendy, R. Hueckstaedt, K. New, W. R. Oakes, et al. The RAGE Radiation-Hydrodynamic Code. *Computational Science & Discovery*, 1(1):015005, 2008.

[14] S. Habib, A. Pope, H. Finkel, N. Frontiere, K. Heitmann, D. Daniel, P. Fasel, V. Morozov, G. Zagaris, T. Peterka, et al. HACC: Simulating Sky Surveys on State-of-the-Art Supercomputing Architectures. *New Astronomy*, 42:49–65, 2016.

[15] R. Haimes. pv3-a distributed system for large-scale unsteady cfd visualization. In *32nd Aerospace Sciences Meeting and Exhibit*, page 321, 1994.

[16] S. Institute, 2016. SCIRun: A Scientific Computing Problem Solving Environment, Scientific Computing and Imaging Institute (SCI), Download from: http://www.scirun.org.

[17] A. Knoll, I. Wald, P. A. Navrátil, M. E. Papka, and K. P. Gaither. Ray tracing and volume rendering large molecular data on multi-core and many-core architectures. In *Proceedings of the 8th International Workshop on Ultrascale Visualization*, UltraVis '13, pages 5:1–5:8, New York, NY, USA, 2013. ACM.

[18] J. A. Kohl, T. Wilde, and D. E. Bernholdt. Cumulvs: Interacting with high-performance scientific simulations, for visualization, steering and fault tolerance. *International Journal of High Performance Computing Applications*, 20(2):255–285, 2006.

[19] T. Kuhlen, R. Pajarola, and K. Zhou. 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.

[20] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. Samatova. Compressing the incompressible with isabela: In-situ reduction of spatio-temporal data. *Euro-Par 2011 Parallel Processing*, pages 366–379, 2011.

[21] M. Larsen, E. Brugger, H. Childs, J. Eliot, K. Griffin, and C. Harrison. Strawman: A Batch In Situ Visualization and Analysis Infrastructure for Multi-Physics Simulation Codes. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV)*, pages 30–35, 2015.

[22] M. Larsen, C. Harrison, J. Kress, D. Pugmire, J. S. Meredith, and H. Childs. Performance modeling of in situ rendering. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16, pages 24:1–24:12, Piscataway, NJ, USA, 2016. IEEE Press.

[23] J. Lofstead, R. Oldfield, T. Kordenbrock, and C. Reiss. Extending Scalability of Collective IO through Nessie and Staging. In *Proceedings of the sixth workshop on Parallel Data Storage*, pages 7–12. ACM, 2011.

[24] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin. Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS). In *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, pages 15–24. ACM, 2008.

[25] K.-L. Ma. In Situ Visualization at Extreme Scale: Challenges and Opportunities. *IEEE Comput. Graph. Appl.*, 29(6):14–19, Nov. 2009.

[26] U. D. of Energy. Synergistic Challenges in Data-Intensive Science and Exascale Computing, Mar 2013.

[27] S. Parker, M. Parker, Y. Livnat, P. P. Sloan, C. Hansen, and P. Shirley. Interactive ray tracing for volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):238–250, Jul 1999.

[28] T. Ringler, M. Petersen, R. L. Higdon, D. Jacobsen, P. W. Jones, and M. Maltrud. A multi-resolution approach to global ocean modeling. *Ocean Modelling*, 69:211 – 232, 2013.

[29] I. Rodero, M. Parashar, A. G. Landge, S. Kumar, V. Pascucci, and P.-T. Bremer. Evaluation of in-situ analysis strategies at scale for power efficiency and scalability. In *Cluster, Cloud and Grid Computing (CCGrid), 2016 16th IEEE/ACM International Symposium on*, pages 156–164. IEEE, 2016.

[30] W. Schroeder, K. Martin, and B. Lorensen. The Visualization Toolkit: An Object-Oriented Approach to 3-D Graphics, 2005.

[31] M. J. Turk, B. D. Smith, J. S. Oishi, S. Skory, S. W. Skillman, T. Abel, and M. L. Norman. yt: A multi-code analysis toolkit for astrophysical simulation data. *The Astrophysical Journal Supplement Series*, 192(1):9, 2010.

[32] V. Vishwanath, M. Hereld, V. Morozov, and M. E. Papka. Topology-aware data movement and staging for I/O acceleration on Blue Gene/P supercomputing systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 19. ACM, 2011.

[33] I. Wald, G. Johnson, J. Amstutz, C. Brownlee, A. Knoll, J. Jeffers, J. Günther, and P. Navratil. Ospray - a cpu ray tracing framework for scientific visualization. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):931–940, Jan 2017.

[34] J. Woodring, J. Ahrens, J. Figg, J. Wendelberger, S. Habib, and K. Heitmann. In-situ sampling of a large-scale particle simulation for interactive visualization and analysis. In *Computer Graphics Forum*, volume 30(3), pages 1151–1160. Wiley Online Library, 2011.

[35] S. Ziegeler, C. Atkins, A. Bauer, and L. Pettey. In situ analysis as a parallel i/o problem. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, pages 13–18. ACM, 2015.

[36] S. B. Ziegeler. An I/O Mini-App Dedicated to In Situ Visualization. In *2016 Second Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV)*, pages 29–34, Nov 2016.