ECP VTK-m: Updating HPC Visualization Software for Exascale-Era Processors

Kenneth Moreland	Sandia National Laboratories	kmorel@sandia.gov
David Pugmire	Oak Ridge National Laboratory	pugmire@ornl.gov
Christopher Sewell	Los Alamos National Laboratory	csewell@lanl.gov
Berk Geveci	Kitware, Inc.	berk.geveci@kitware.com
Hank Childs	University of Oregon	hank@cs.uoregon.edu

Software Technology Category: Data Analytics and Visualization

Primary Funding Sources: [Funding sources removed for unlimited release.]

SAND 2016-12912 O

1 Executive Summary

The scientific visualization community, with significant past funding support from DOE, provides a suite of production-quality software tools that address the large-scale data analysis and visualization needs of scientists across many application domains. This software, which includes the end-user tools ParaView and VisIt and the in situ technologies Catalyst and LibSim, is proven to be very versatile and to be scalable enough to address the scientific discovery needs of today's largest simulation runs.

However, as with much of the HPC software existing today, the bulk of our scientific visualization software is designed for clusters of traditional serial CPU processors. Unfortunately, the model of clusters of serial processors is quickly becoming obsolete. Exascale systems will use a variety of new processor technologies that require a massive amount of parallelization to leverage technologies such as multiple cores, hyperthreading, and vectorized operations as well as systems with heterogeneous processors and deep memory hierarchies. Our existing visualization software will not run effectively on these systems. **If we do not update our scientific visualization algorithms to run effectively on exascale processor technology, we will lose our ability to make discoveries with computational science at scale.**

The VTK-m software product seeks to remedy this technology gap by providing the underlying algorithmic engine for advanced processors. The VTK-m library provides a framework and infrastructure to implement massively threaded visualization algorithms with high productivity. Furthermore, VTK-m provides a cross-platform framework that allows a single algorithm implementation to port across all the foreseeable DOE exascale platforms. Multiple studies have shown that the archictecture-agnostic approach utilized by VTK-m has comparable performance to architecture-specific implementations.

Although there will be some time spent in the VTK-m project building up the infrastructure, the majority of the work is in redeveloping, implementing, and supporting necessary visualization algorithms in the new system. We plan to leverage a significant amount of visualization software for the exascale, but there is still a large base of complex, computationally intensive algorithms built over the last two decades that need to be redesigned for advanced architectures. Although VTK-m simplifies the design, development, and implementation of such algorithms, **updating the many critical scientific visualization algorithms in use today requires significant investment**. And, of course, all this new software needs to be hardened to be ready for production, which adds a significant overhead to development.

Our proposed effort will in turn impact key scientific visualization tools. Up until now, these tools — ParaView, VisIt, and their in situ forms — have been underpinned by the Visualization ToolKit (VTK) library. VTK-m builds on the VTK effort, with the "-m" referring to many-core capability. The VTK-m name was selected to evoke what VTK has delivered: a high-quality library with rich functionality and production software engineering practices, enabling impact for many diverse user communities. Further, VTK-m is being developed by some of the same people who built VTK, including Kitware, Inc., which is the home to VTK (and other product lines). Developers of ParaView and VisIt are in the process of integrating VTK-m, using funding coming from SciDAC and ASC. However, while VTK-m has made great strides in recent years, it is missing myriad algorithms needed to be successful within the ECP. Developing those algorithms is the focus of this ECP proposal.

For the ECP/VTK-m project, we have assembled a team that are experts both in addressing HPC visualization needs for DOE and in developing multi-/many-core visualization algorithms in VTK-m. The ECP/VTK-m project is organized with a definitive set of measurable milestones, which will be implemented through established formal development practices. VTK-m fills a critical gap in the implementation of visualization for scientific discovery, and the flexibility of VTK-m will allow it to fill this role throughout ECP software.

2 Exascale Software Challenge

Visualization and analysis has long been recognized as an important enabling technology for computer simulation [2]. It is used by computational scientists for use cases ranging from visual debugging to exploring data in efforts to uncover new science. However, exascale I/O constraints will force the processing paradigm for scientific visualization and analysis to change from post hoc processing to in situ processing. While this transition has many implications, the central focus of this proposal is that visualization and analysis routines will need to run on the same hardware as the simulation codes. Moreover, since visualization and analysis algorithms can be very computationally intensive (e.g., particle advection or volume rendering), efficient performance is critical.

Among the many fundamental changes we are seeing in architectures likely to be used for ECP systems, one of the most profound is a reliance on new processor types optimized for execution bandwidth over latency hiding (via massively threaded processors). Our current production scientific visualization software is not designed to efficiently deal with such architectures, and upgrading this software to efficiently support ECP architectures is one of the most critical challenges for HPC scientific visualization. Important community visualization tools — namely ParaView, VisIt, and their in situ variants — are planning on closing this gap using our VTK-m framework.

2.1 Exascale Problem Target

Although the basic architecture for high-performance computing platforms has remained homogeneous and consistent for over a decade, revolutionary changes are appearing on leading edge supercomputers, with plans for future supercomputers containing even larger changes. One noteworthy attribute of future HPC machines is the massive increase in concurrency required to sustain peak computation; most project billions of threads to achieve an exaflop [3]. This increase is partially accredited to requiring more cores to achieve faster aggregate computing rates [14] and partially accredited to using additional threads per core to hide memory latency [34]. Due to limitations of cost and power, the system memory will not commensurately increase, which signifies algorithms will need good strong scaling (that is, more parallelism per unit of data) [32].

This trend to massive threading can be seen in high-performance computing today. The current leadership-class computer at Oak Ridge National Laboratory, Titan, requires between 70 million and 500 million threads to run at peak, which is over 300× more than required by its predecessor, JaguarPF. In contrast, the system memory grew only by a factor of 2.3.

The increasing reliance on concurrency to achieve faster execution rates invalidates the scalability of much of our scientific HPC code. New processor architectures are leading to new programming models and new algorithmic approaches. The design of new algorithms and their practical implementation are a critical extreme-scale challenge [6, 12].

To address the needs for HPC scientific visualization, earlier research has established the VTK-m library [29, 30]. VTK-m provides a framework for simplifying the design of visualization algorithms on current and future architectures. Some prototype algorithms in VTK-m are shown in Figure 1. VTK-m also provides a flexible data model that can adapt to many scientific data types and operate well on multi-

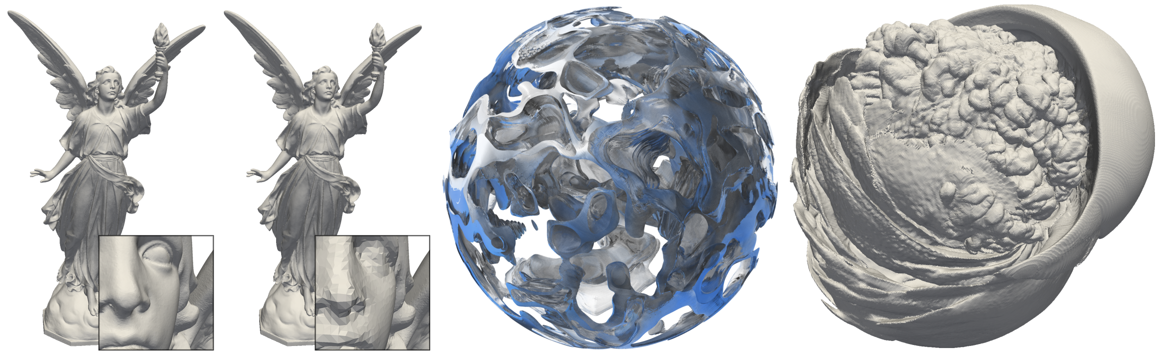


Figure 1: Prototype implementations of mesh simplification, rendering, and contouring in VTK-m.

threaded devices. Finally, VTK-m serves as a container for algorithms designed in the framework and gives the visualization community a common point to collaborate, contribute, and leverage massively threaded algorithms. VTK-m gives us the software development tools to engineer the necessary multi-/many-core visualization algorithms so critical for exascale. It is now time to apply this foundation to redesign our scientific visualization algorithms and update our HPC software.

2.2 Impact and Urgency

The scientific visualization research community has been building scalable HPC algorithms for over 15 years, and today there are multiple production tools that provide excellent scalability. However, our current visualization tools are based on a message-passing programming model. They expect a coarse decomposition of the data that works best when each processing element has on the order of a hundred thousand to a million data cells [13, 25, 26].

HPC systems are undergoing a revolution in processor hardware architecture for exascale. New HPC systems require a much higher degree of parallelism that could require threads operating on only a small number of data cells. At this fine degree of parallelism, our conventional visualization approach breaks down in multiple different ways.

2.2.1 Load Imbalance

Typically data are partitioned under the assumption that the amount of work is uniform for equal-sized data regions. However, this is not true for all visualization algorithms, many of which generate data conditionally based on the input values. With only a few exceptions, current parallel visualization functions completely ignore this load imbalance, which is considered tolerable when amortized over larger partitions.

When the data gets decomposed to the cell level, this amortization no longer occurs, which results in a much more severe load imbalance. Finely threaded visualization algorithms need to be cognizant of potential load imbalance and schedule work accordingly.

2.2.2 Dynamic Memory Allocation

When the amount of data a visualization algorithm generates is dependent on the values of the input data, the size and structure of the output data are not known at the execution outset. In such a case, the algorithm must dynamically allocate memory as data are generated.

Processing elements in our conventional parallel visualization algorithms that operate on coarse partitions in distributed memory spaces can dynamically allocate memory completely independent from one another. In contrast, dynamic memory allocation from many threads within a shared memory environment requires explicit synchronization that inhibits parallel execution.

2.2.3 Topological Connections

Scientific visualization algorithms are dominated by topological operations on meshes. Care must be taken when defining topological connections across boundaries of data assigned to different processing elements. Mutual data being read must be consistent and mutual data being written must be coordinated.

A conventional parallel visualization algorithm typically manages topological connections across processes by replicating data on the boundaries using ghost regions [4]. Since the partitions are coarse, there is a limited amount of replication. However, this replication cannot be sustained as the data decomposition approaches single cells. Instead, threads must share connected data, and generated data may require further processing to identify coincident topology.

2.2.4 SIMD Execution

In addition to increasing execution bandwidth through multiple processing cores, modern HPC processors are also increasing the width and number of vector processing units, allowing the same instruction to be executed simultaneously on a large number of data values. Whereas conventional parallel visualization algorithms allow for completely independent execution, efficient vector processing requires algorithms that can leverage data level parallelism to run in single instruction, multiple data (SIMD) mode. New algorithms will need to minimize code execution divergence.

3 Exascale Software Description

3.1 Software Overview and Status

Earlier this decade, DOE visualization researchers began investigating the changes needed to support the heavily threaded processors that would soon be appearing on leadership class facilities. Three separate efforts emerged, each endeavoring to be the "new VTK" for many-core architectures. Each effort had its own key strength: Dax [28] focused on building blocks for threaded visualization algorithms, PISTON [19] focused on data parallel primitives for architecture portability, and EAVL [22] focused on flexible and advanced data models that work well on accelerator processors. However, the leads of these efforts began to collaborate, leading to a new, combined product (VTK-m) which has the backing of the community, and encompasses all of the developers from the original three projects. The original projects (Dax, EAVL, PISTON) have now all effectively been retired, leaving VTK-m as the path forward for ParaView, VisIt, and their in situ variants. VTK-m's name reflects the desire to emulate many of the elements of VTK: open source, large community, diverse usage, and production quality. Kitware Inc. is endorsing the effort and has been a driving force in the initial development of the combined framework.

VTK-m is powerful because it serves as a container for algorithms, provides flexible data representation, and simplifies the design of portable and efficient visualization algorithms across new and future computer architectures. The XVis project has provided the base research and development for VTK-m's multi-/many-core visualization framework, and the framework's support for portable algorithms, multiple architectures, and data representation is reaching production level. Using the existing framework, we have made headway implementing some fundamental visualization algorithms including field operations, contouring, thresholding, and simplification. However, there are other fundamental features, such as searching, particle advection, and connected components, that have yet to be addressed. Furthermore, those operations that are implemented are missing features critical for their use in packages like VisIt and ParaView. As the adage goes, "the last 10% takes 90% of the time," meaning transitioning software from working example to production-ready is itself a significant development effort. With this proposal, we focus on this missing element, namely implementing the many critical visualization and analysis algorithms needed by ECP stakeholders.

3.1.1 Performance Portability

Unlike many "in house" HPC software applications that can be written for a specific HPC installation, the software our team develops is used across all the DOE supercomputer facilities. This means our software must perform well on a variety of hardware types and configurations. Furthermore, the critical code in a visualization system is not limited to a small number of iterative loops. Our software contains numerous algorithm implementations to address visualization needs across many scientific disciplines.

Upgrading this functionality is a daunting challenge. Attempting to create multiple algorithm implementations targeted to each known platform (and possibly currently unknown platforms in the future) creates a combinatorial explosion of work. This approach would require significant developer investment, likely exceeding worldwide funding available for this task.

Instead, VTK-m uses generic data parallel primitives to achieve performance portability. VTK-m defines an abstract device model constructed from a set of well-known operations such as map, scan, sort, and reduce, that run in parallel on the device. The VTK-m team is rethinking visualization algorithms as a sequence of these parallel primitive operations. With this abstraction, the entirety of algorithms in VTK-m can be ported to a new device by providing only the parallel primitives for that device. Algorithms need only be written once to VTK-m's abstract device model, which dramatically reduces the implementation work

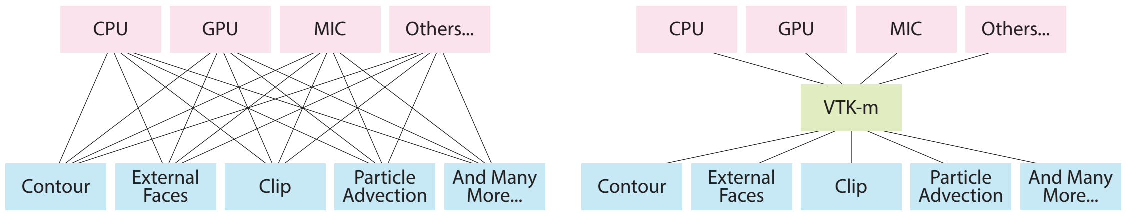


Figure 2: At left, implementing all of our visualization algorithms separately for every architecture we need to support leads to an unmanageable amount of software. At right, by using VTK-m's abstract device model, we can reduce the amount of software to implement to a feasible level.

for various architectures, as shown in Figure 2.

The indirection of using data parallel primitives to implement algorithms can limit the amount of optimization that can be done, which raises the question of how much portability impacts performance. Evidence collected from recent research using data parallel primitives [18, 19, 29, 30] suggests that this approach to performance portability is efficient and does not introduce significant memory overhead. The small loss of performance is greatly outweighed by the benefits of simplified implementation and performance portability.

Figure 3: Comparison of the Marching Cubes algorithm in VTK-m and the CUDA SDK reference implementation. Implementations in VTK-m are simpler, shorter, more general, and easier to maintain.

3.1.2 Toolkit for Visualization Algorithms

In addition to reducing the complexity of supporting visualization algorithms across many architectures, VTK-m simplifies the development of parallel scientific visualization algorithms by providing a framework of supporting functionality that allows developers to focus on visualization operations. Consider the listings in Figure 3 that compare the size of the implementations for the Marching Cubes algorithm in VTK-m with the equivalent algorithm implemented as a reference implementation in the CUDA software development kit. The VTK-m implementation is shorter and easier to maintain because VTK-m internally manages the parallel distribution of work and data, and reduces the amount of boilerplate code required.

In addition to being shorter and easier to write, algorithms implemented in VTK-m tend to be more versatile. The CUDA reference implementation is only able to operate on a single type of data using a particular memory layout on a single type of mesh structure on a single class of architecture. The VTK-m algorithm, in contrast, can be run on multiple data types using different memory layouts and on meshes with different data models without modifying a single line of this code. This is because the VTK-m algorithm is using flexible templated data structures that can be modified independently from the algorithm. VTK-m also provides the infrastructure that allows the algorithm to be trivially ported across different processor architectures without changing the code.

A key property of VTK-m is that its architecture-agnostic approach is able to achieve performance comparable to architecture-specific implementations. When comparing to the CUDA Marching Cubes algorithm from NVIDIA discussed earlier in this section, VTK-m can achieve performance within 15% to 25%. Further, in practical terms we believe the performance gap is even less than this amount since the CUDA implementation makes multiple assumptions, such as only dealing with powers-of-two grids, that optimize performance, but these assumptions are infeasible for a general-purpose use in an application like VisIt or ParaView. Generalizing their code would undoubtedly make it slower and thus reduce the 15% to 25% performance difference.

3.2 Software Development Needs

The ECP/VTK-m project represents a major development push to add the necessary features and algorithms using the VTK-m framework to enable scientific visualization tools at the exascale. Although we anticipate some level of research as we address new algorithmic challenges, ECP/VTK-m is primarily a development push.

We can estimate the development effort by considering the existing VTK library [33]. VTK is the base visualization library on which our production HPC scientific visualization tools like ParaView [5] and VisIt [11] are built. VTK wraps algorithms in what it calls filters, which is where the majority of the computation happens and which is where we predict we will have the most influence with VTK-m. VTK contains hundreds of filters, most of which do not work with multiple threads and none of which work on accelerators like GPUs. It is necessary for us to enhance critical filters with updated multi-/many-core algorithms in VTK-m. Many of these filters have been identified as important for the tools ASCR application scientists use, and it is this list we must improve in ECP. Details of which algorithms we are addressing in ECP/VTK-m are given in Section 5.

3.2.1 System Requirements

We have no intention of attempting a complete rewrite of the entire scientific visualization software used by DOE, which comprises millions of lines of code. Instead the VTK-m project is focused on implementing the underlying computational engine for visualization software. Within this engine are the computationally intensive visualization algorithms, which will be re-engineered to work efficiently on exascale platforms.

Figure 4 gives an overview of the software used for large-scale scientific visualization. At the base of this software stack is the core visualization library, VTK, which provides the foundation for the rest of the software. Built on top of the VTK library are the end user tools ParaView and VisIt. In addition to providing the user interfaces used by DOE scientists on a daily basis, these software products manage the distributed memory parallel processing and remote interaction required for large-scale data analysis. It makes sense to leverage these software products for our in situ visualization needs because they are a flexible and versatile collection of visualization products. Hence, the Catalyst and LibSim libraries make this rich software stack available for use in running simulations.

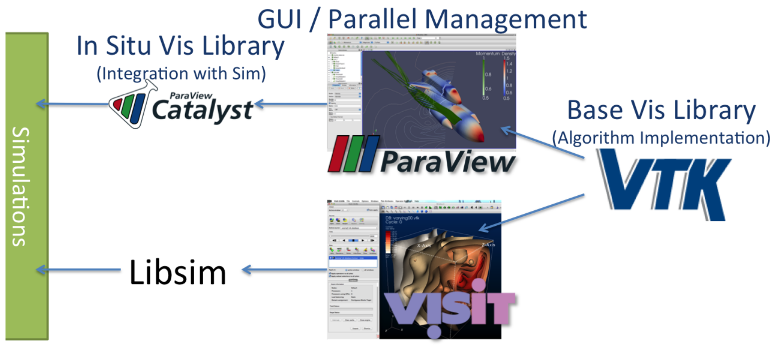


Figure 4: The scientific visualization software stack.

Our plan for VTK-m is to heavily leverage this existing visualization software stack. Although they share similar names, VTK and VTK-m are independent products. However, VTK-m is not intended to be a replacement for VTK. The investment in VTK development is simply too large. Instead, our plan is to integrate VTK-m into VTK. VTK is a modular library that makes it possible to add components to the system. By supporting the VTK data model, we can seamlessly integrate algorithms implemented in VTK-m, which can be deployed as "filters" in the VTK environment. This also allows us to target those filters that are necessary for ECP and loosely collaborate with others interested in implementing other algorithms for their own needs.

Although our integration plan allows us to leverage a significant portion of visualization software at the exascale, the transition still necessitates updating the computationally intensive algorithms. This update is no small task. There are hundreds of independent visualization algorithms that are developed and used in scientific visualization software. This update is not a simple scaling or transformation of the software. Each implementation requires a redesign using new execution models and a new development environment.

Much of the development in ECP/VTK-m can be performed with readily available computer hardware because VTK-m is designed to allow developers to create platform-independent software. There will be a small set of core developers that will need early access to testbed machines to ensure that the core functionality of VTK-m works well in these environments. However, once the initial design for a given computer platform is complete, the remaining update to new hardware should be minimal.

3.2.2 Technology Maturity

VTK-m is a relatively young product. However, VTK-m is the descendant of three earlier software research projects (Dax, PISTON, and EAVL), which has allowed VTK-m to quickly grow and supplant its antecedents. Furthermore, since VTK-m is a collaboration across multiple national laboratories, it makes efficient use of DOE research and development resources with increased impact across a range of DOE mission applications.

Although VTK-m is in its early development stages, from the start we have been using an agile iterative software development process similar to Kanban to maintain high-quality software throughout. Our development process requires the introduction of regression tests for any added software, and there are currently well over a hundred tests. These tests are run nightly to ensure continued software quality. Furthermore, all changes are sequestered to be tested, reviewed, and vetted before being merged into the main branch. This software process is managed by Kitware who also manages VTK, ParaView, and many other similar products.

3.2.3 Cross-Team Collaboration and Integration

The main deployment vehicle for VTK-m will be the integration within the VTK library. As is evident in Figure 4, this integration will make VTK-m accessible at least indirectly to the entire visualization software stack. The ECP/VTK-m project expects to rely on related projects to perform the majority of the integration and deployment work. (The SciDAC-4 and ECP-ALPINE proposals each respectively address a different aspect of this integration and deployment.) As a contingency for this integration not happening in another project, we plan for the minimal amount of integration to run VTK-m in our visualization software stack.

Assuming they are funded, we will collaborate with the SciDAC institute to ensure a smooth integration of VTK-m with the ParaView and VisIt tools. We will collaborate with other ECP projects that can benefit from VTK-m integration and vice versa. The DIY project will better enable algorithms that require special communication in distributed memory machines, and we can work together to link the data models and integrate the algorithms. Several ECP projects support code in ParaView, VisIt, Catalyst, LibSim, and ADIOS, all of which make excellent deployment platforms for VTK-m. We have also demonstrated that VTK-m can be directly integrated into science applications should it be necessary for a targeted, lightweight solution.

3.2.4 Related Research

The most direct related research is ongoing in the ASCR-funded XVis project, which continues through FY17. Planned research includes characterizing data access and function execution as well as evaluating hybrid parallel execution strategies. These efforts will be highly complementary to VTK-m development in ECP.

If this proposal is funded, ATDM has agreed to provide additional funds to support complementary work to implement several classes of filters in VTK-m that are not included in the scope of this proposal. Specifically, these would include field reductions (e.g., generating bounding-box outlines or computing functions of quality over a full mesh), extraction of arbitrary subsets (e.g., thresholding), and the generation of simplices (e.g., triangulation and tetrahedralization). Additionally, ATDM will contribute to the implementation of several modifications to the infrastructure including dynamic types, cell sets, and execution modes.

VTK-m will also respond to other complementary research. For example, work on Kokkos [15] and parallel extensions to the C++ standard provide new tools for managing memory and execution. We are also interested in collaborating with research projects to integrate VTK-m technologies into systems using Legion [8], DHARMA [9], Charm++ [1], Uintah [10], and other asynchronous many-task solutions. Of course, we will also ensure that VTK-m integrates well with the existing software stack (Figure 4), which comprises VTK [33], ParaView [5], VisIt [11], Catalyst [7, 16], and LibSim [35], as well as the supporting infrastructure tools like ADIOS [20], DIY [31], and IceT [27]. In this way, VTK-m fills the gap of executing visualization algorithms on advance processor architectures in a variety of software systems.

VTK-m is able to fill this role by providing flexible data structures that adapt to the needs of the client software. One of these data structures is an "array handle" object used throughout VTK-m. The array handle is templated on a "storage" object that allows it to access different memory layouts [28] including those defined outside of VTK-m as illustrated in Figure 5.

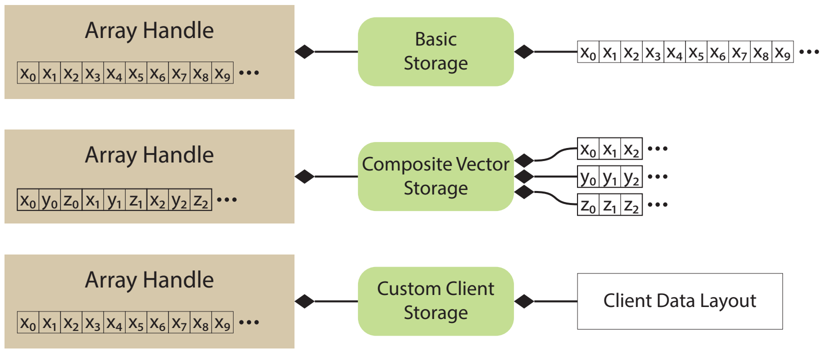

Array Handle
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 •••
Basic Storage
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 •••
Array Handle
x_0 y_0 z_0 x_1 y_1 z_1 x_2 y_2 z_2 •••
Composite Vector Storage
x_0 x_1 x_2 •••
y_0 y_1 y_2 •••
z_0 z_1 z_2 •••
Array Handle
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 •••
Custom Client Storage
Client Data Layout

Figure 5: Array handles, storage objects, and the underlying data source.

VTK-m also provides a very flexible data model. The data model comprises sets of cells, sets of fields, and sets of coordinate systems. All three of these items can be defined independently of each other, which provides significant flexibility and efficiency in the definition of the data structure [23]. It is also possible to extend the data model with new cell set structures, as illustrated by the extruded mesh concept shown in Figure 6, which we cover in the work section of this project (Section 5).

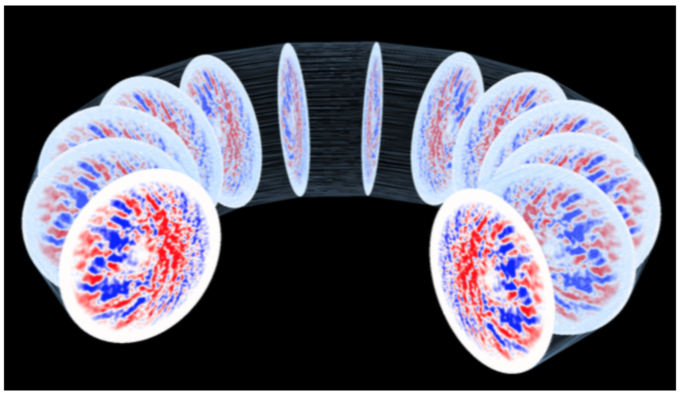


Figure 6: Many fusion codes represent the 3D simulation mesh with a single plane of 2D geometry and a number that specifies the angular spacing. An extruded mesh data structure allows a zero copy representation of the simulation data.

4 Exascale System Utilization

It is critical that we optimize VTK-m for the many-core architectures that will be used on ECP's eventual exascale machine. We believe that we can make significant progress by optimizing on the leadership machines in the pre-exascale era. For the duration of this proposed project, we believe the CORAL, Trinity, and Cori machines will be the best surrogates, and we plan to assess performance on these machines regularly. Our proposal team should have access to the pre-exascale machines and be able to use them to evaluate VTK-m because we have connections to multiple DOE Labs and other programs (specifically SciDAC). Further, since we plan to deploy our software within ECP applications code (both directly and through ECP-ALPINE technology), we should have regular access within ECP as well. Looking forward, we predict a similar situation with the APEX and CORAL2 procurements.

Our project is primarily focused on leveraging on-node parallelism, and representative many-core/multi-core devices, such as Nvidia GPUs and Intel Xeon Phis, are readily available. However, we will need access to testbeds as new processors and accelerators become available so that we can test and optimize our code for them. Our focus on on-node parallelism means we will require fewer hours on large machines than most other ECP projects. Nevertheless, we do anticipate the need for roughly one million hours to ensure that our filters do indeed run efficiently within externally-developed distributed memory infrastructures, especially for algorithms such as stream lines that are known to be challenging on distributed systems.

5 Expected Outcomes: Milestones and Deliverables

Getting to exascale requires the most dramatic change in algorithm design VTK has seen in its over two decades of development; we are replacing software developed over many man-years of effort. As such, there are many items that need to be addressed to run effectively at exascale. To organize this work, we break the milestones into groups of related tasks, which are itemized and numbered in the following text. Figure 7, located after the descriptions of our milestones on page 15, provides a Gantt chart of the overall expected progress of our milestones.

M1 Infrastructure

The infrastructure work is dedicated to supporting the algorithms and use cases of VTK-m. Work on the infrastructure is very general and can impact the capabilities of all algorithms in VTK-m.

M1.a Better Dynamic Types Design (Year 1/Q1, SNL, Kitware) For the best efficiency across all platforms, VTK-m algorithms use static typing with C++ templates. However, many libraries like VTK, ParaView, and VisIt use dynamic types with virtual functions because data types often cannot be determined at compile time. We have an interface in VTK-m to merge these two typing mechanisms by generating all possible combinations of static types when faced with a dynamic type. Although this mechanism works, it generates very large executables and takes a very long time to compile. As we move forward, it is clear that these problems will get worse and become infeasible at exascale. We will rectify the problem by introducing some level of virtual methods, which require only a single code path, within VTK-m algorithms. This first milestone produces a design document to propose an approach to the new system.

M1.b Better Dynamic Types Implementation (Year 2/Q1, SNL, Kitware) Builds on Milestone M1.a to implement the proposed solution.

M1.c Configurable Cell Sets Design (Year 2/Q1, Kitware, SNL) Currently there is a fixed set of cell types supported within VTK-m. One issue with having a fixed set of cells is that it becomes difficult to interface VTK-m's data model with another that uses different cell types or different interpretations of cells. This change would make it possible to reconfigure the types of cells used so that it can be applied to different sets of cells. This milestone produces a design document to propose an approach to the new system.

M1.d Configurable Cell Sets Implementation (Year 3/Q1, Kitware, SNL) Builds on Milestone M1.c to implement the proposed solution.

M1.e Execution Modes (Year 3/Q2, Kitware) VTK-m currently supports two main execution modes: a simple map of array values and a more complex map allowing access to incident topology. However, other operations require different organization of the parallel execution. For example, statistical queries require scheduling of parallel reduction. Computational geometry needs the ability to produce cells of different shapes with points that may be shared or interpolated in different ways. Particle advection algorithms require the incremental building of geometry. Connected components algorithms will likely require an iterative "pointer jumping" operation.

M1.f Lightweight Cell Library (Year 3/Q4, Kitware) VTK and VTK-m each have their own code to perform operations on cells. This leads to code duplication as well as further code to resolve differences between the two. To better provide code sharing between VTK, VTK-m, and other visualization software, we will separate out the cell operations into a shared, lightweight library that can be included in both projects without heavy overhead.

M1.g Multiblock Data (Year 1/Q4, ORNL) The data structure in VTK-m is currently defined as a single block. It is easy for other software like VTK to build multiblock structures containing multiple VTK-m data sets, but multiblock data structures could potentially provide specialized optimizations. We expect it will be worthwhile to support multiblock data as a first class citizen.

M1.h Specialized Data Models (Year 2/Q4, ORNL) The data model in VTK-m has been designed to efficiently represent a large class of data layouts used by simulation codes. VTK-m currently supports a basic set of sturctured and unstructured meshes, but a variety of specialized representations have not been implemented yet. These include extruded meshes, which are used to represent the toroidal mesh in fusion simulations, and molecular meshes used in chemistry and biological simulations. In the particular case of fusion simulations, support for extruded meshes will make it possible to directly represent the toroidal mesh with zero-copy.

M2 Field Computation

Derived fields, mesh warping, interpolations, derivatives, and other computed field values are a fundamental part of scientific visualization. These field computation algorithms tend to parallelize well although they may involve searching local neighborhoods, interpolating values, or performing reduction operations.

M2.a Faceted Surface Normals (Year 1/Q2, Kitware) Surface normals are a critical component for lighting calculations that provide the major visual cues for surface shape. The first approximation for the normal to a surface represented by a polygonal mesh is the perpendicular direction to each flat polygon. This is simple to calculate, but provides a faceted representation of the surface.

M2.b Smooth Surface Normals (Year 1/Q3, Kitware) This milestone improves on the normals provided by Milestone M2.a by averaging the normals to each point in the mesh. These normals on points can then be interpolated by the rendering system to estimate the lighting on a smooth surface.

M2.c Predefined Function Evaluation (Year 1/Q4, Kitware, ORNL) Predefined functions such as vector math (dot product, cross product, normalization, magnitude), distance measurements, field component extraction, and texture coordinate generation are necessary components of scientific visualization. This milestone will ensure that these core functions will be implemented in VTK-m and made available to ParaView, and VisIt.

M2.d Point Movement (Year 2/Q2, Kitware, ORNL) Point movement is the basis for deformations, transformations, and projections of geometric and image data inside VTK, ParaView, and VisIt. Further, the fundamental concept also can be used to apply transformations, and warps to arbitrary fields. This milestone will ensure that VTK-m will provide sufficient point movement algorithms for the needs of ParaView and VisIt.

M2.e Cell Metrics (Year 1/Q4, UO) Over the past decade, mesh-quality metrics, such as “skew,” “aspect ratio,” “diagonal ratio,” and others have helped simulation code developers identify problematic cells in tools such as VisIt and ParaView. Further, volume and area calculations are needed for integration-based analysis tasks. This milestone will ensure that these filters are available in VTK-m to fit the needs of VisIt and ParaView.

M2.f Surface Metrics (Year 2/Q4, Kitware) Surface metrics like curvature provide vital information about the form of a mesh. This information helps identify important features, create silhouettes, smooth surfaces, and identify uncertainty [17].

M2.g Gradient (Year 1/Q4, SNL) There are several mechanisms to estimate derivatives and related metrics, and each has its own place in visualization and analysis. VTK-m must support estimating gradients on all mesh types and all cell types. The gradient operation should also support secondary gradient computations like divergence and vorticity.

M3 Computational Geometry

Operations on meshes are fundamental to scientific visualization. These can involve extracting, coarsening, smoothing, intersecting, following topology, or deriving geometry. Performing computational geometry algorithms on many threads can be complex. Although parallel tasks can be broken down to finite elements, these task can have many interdependencies as items generated in parallel connect to each other.

M3.a Merge Points (Year 2/Q4, SNL) For a variety of reasons it is possible for a mesh structure to contain points that are incident with each other, and it may be necessary to identify these incident points and merge them. In the most general case, coincident points must be identified by locating points with spatial coordinates. In many specialized situations, coincident points can be identified via hashed keys. Although less general, these approaches tend to be faster and more robust [24].

M3.b Clip (Year 3/Q4, UO) Clip operations intersect meshes with implicit functions. It is the foundation of spatial subsetting algorithms, such as "box," and also the foundation of data-based subsetting, such as "isovolume." The algorithm requires considering thousands of possible cases, and is thus quite difficult to implement. This milestone will implement clipping to be sufficient for VisIt's and ParaView's needs.

M3.c Connected Components (Year 2/Q4, LANL) Connected components algorithms identify groups of cells that are connected to each other, either directly by a shared face or edge, or indirectly by some path. The "friend of friends" halo definition often used in cosmology simulations is one example application. Multiple threads must be coordinated to agree on group identification. Likewise, special support is needed to achieve consistent group identifications when used in a distributed memory parallel environment.

M3.d External Surface (Year 1/Q4, UO, SNL) External facelist calculation produces the facets on the exterior of a 3D mesh. This algorithm is used ubiquitously within VisIt and ParaView. UO previously implemented a version of this algorithm for tetrahedral meshes, and, for this milestone, will extend this algorithm to work for all mesh types. Further, Sandia will integrate new execution modes (milestone M1.e) with this algorithm.

M3.e Path Geometry (Year 3/Q4, ORNL) Algorithms such as streamlines can generate 1D paths through space (typically represented as polylines). Representing these paths with 3D tube or ribbon geometry can provide important visual cues for the curve's shape and can add further properties to the curve like orientation, magnitude, or uncertainty.

M3.f Contouring (Year 3/Q4, LANL, SNL) Creating contours, also known as isosurfaces in 3D, is a particular class of cell generation algorithms that is of great importance to scientific visualization. The initial Marching Cubes based implementation for contouring exists in VTK-m, but there are a variety of specializations that can be applied both to support contouring on more generalized, unstructured meshes, and to optimize the performance for more specific types of meshes. Contouring algorithms can also be used as a building block for several other algorithms, such as slicing and extracting parts from volume fractions. LANL will lead the algorithm development, while SNL will assist in integrating these algorithms with new execution modes (milestone M1.e) and in completing the implementation across all mesh types (under SciDAC funded work).

M3.g Ghost Cells (Year 2/Q2, ORNL) A common way to support continuity across multi-block data boundaries is the use of ghost cells. While full support for multi-block data will be provided by another layer in the software stack , the data model in VTK-m needs to provide a representation for ghost cells, and VTK-m algorithms and filters need to appropriately handle ghost cells.

M3.h Feature-Sensitive Surface Normals (Year 3/Q4, Kitware) Milestone M2.b provides the basic surface normal generation required for rendering. This algorithm usually provides a reasonable representation, but it often behaves poorly around certain features of the data. For example, the "winding" of the polygons in a surface can be inconsistent, which means normals might point in opposite sides of the surface, causing poor shading and coloring when rendered. The averaging of normals should make sure that the directionality is consistent. Also, smooth normals do not shade sharp bends in the mesh correctly. Edges on these sharp bends must be "split" to correct the shading. These considerations take additional design and implementation.

M4 Searching

Often mesh elements are found by iterating over topologies and following connections. However, it is sometimes necessary to find mesh elements from other criteria such as position in space. Such queries require building search structures to quickly find the appropriate data. These search structures must be built in parallel and operate well across multiple platforms.

M4.a Spatial Division (Year 1/Q4, LANL) Spatial division algorithms create a partitioning in space and identify the location of points in that partitioning. Examples include kD-trees and OBB (oriented bounding box) trees. Typically the spatial partitioning is designed to evenly divide the points at each level of a recursive partitioning. Many search algorithms use spatial division to rapidly locate or intersect elements.

M4.b Locate Point (Year 1/Q4, LANL) Point location makes use of a search data structure built from input data points to efficiently query for the points that are located within a specified region of interest.

M4.c Locate Cell (Year 2/Q3, UO, ORNL) Cell location, i.e., deciding which cell a point lies within, is a critical building block for many visualization algorithms, including picking and particle advection. For this milestone, UO will produce an efficient cell location module that can be used within other algorithms. This will be done in conjunction with ORNL, who will use the algorithm for particle advection.

M4.d Resample (Year 3/Q4, UO, ORNL) Resampling a mesh onto a regular mesh is used for various purposes in visualization, including coarsening for interactivity and doing quick and approximate neighbor searches. For this milestone, UO will lead, and ORNL assist in implementing a resampling capability within VTK-m.

M5 Particle Advection

Particle advection is a base operation in many flow visualization algorithms. Thus, a base particle advection system is needed as well as multiple algorithms built on top of this system. It is worth noting that particle advection algorithms require special handling of multiple nodes in a distributed memory parallel job. Thus, we plan to incoporate other technologies such as DIY to enable the algorithm on HPC. It is also worth noting that particle advection must identify the location of points as they are advected in space, and thus likely to rely on the search structures of Milestone M4.c.

M5.a Advect Steady State (Year 2/Q3, ORNL, UO) This includes the ability to advect many particles in 2D and 3D flows where the flow direction is not changing over time. We will support multiple integration methods and multiple mesh structures.

M5.b Advect Time Varying (Year 3/Q1, ORNL, UO) This includes the ability to advect many particles in 2D and 3D flows where the flow direction does change over time. Supporting multiple time steps usually requires streaming data over time from another source. Hence, this version of the algorithm might require integration into other parts of the software stack.

M5.c Advanced Flow Algorithms (Year 3/Q4, ORNL, UO) Particle advection enables many other advanced flow visualization algorithms such as FTLE and Poincaré plots. This milestone will explore which of these algorithms are most worthwhile in VTK-m and implement select algorithms.

M6 Rendering

Producing an image representation of VTK-m data is a foundational requirement. VTK-m currently has a basic set of rendering options avalable that needs to be expanded to make it more generally usable by applications.

M6.a Rendering Topological Entities (Year 1/Q4, ORNL) VTK-m currently supports surface rendering by tesselation of data structures, and rendering the resulting triangles. We will extend current functionality to include face, edge, and point rendering.

M6.b Particle Rendering (Year 2/Q4, ORNL) Particle representations are important to a number of ECP applications, and particle rendering is currently not supported in VTK-m. We will implement rendering of particle data in VTK-m.

M6.c Optimization of Rendering Methods (Year 3/Q4, ORNL) We will benchmark and optimize the performance and memory requirements for rendering of VTK-m data on accelerator hardware. This work will include support for CUDA-OpenGL interop in order to reduce movement of data to GPUs.

M7 Software Management and Releases

As the ECP/VTK-m project progresses, we will maintain a stable software product on which applications can depend. (See Section 6.2 for details on the software process.) We will provide regular releases of the VTK-m library to provide dependent software access to new features.

M7.a VTK-m Release 1 (Year 1/Q3, Kitware, SNL) We will provide a release of VTK-m software and associated documentation during FY17.

M7.b VTK-m Release 2 (Year 2/Q3, Kitware, SNL) We will provide a release of VTK-m software and associated documentation during FY18.

M7.c VTK-m Release 3 (Year 3/Q3, Kitware, SNL) We will provide a release of VTK-m software and associated documentation during FY19.

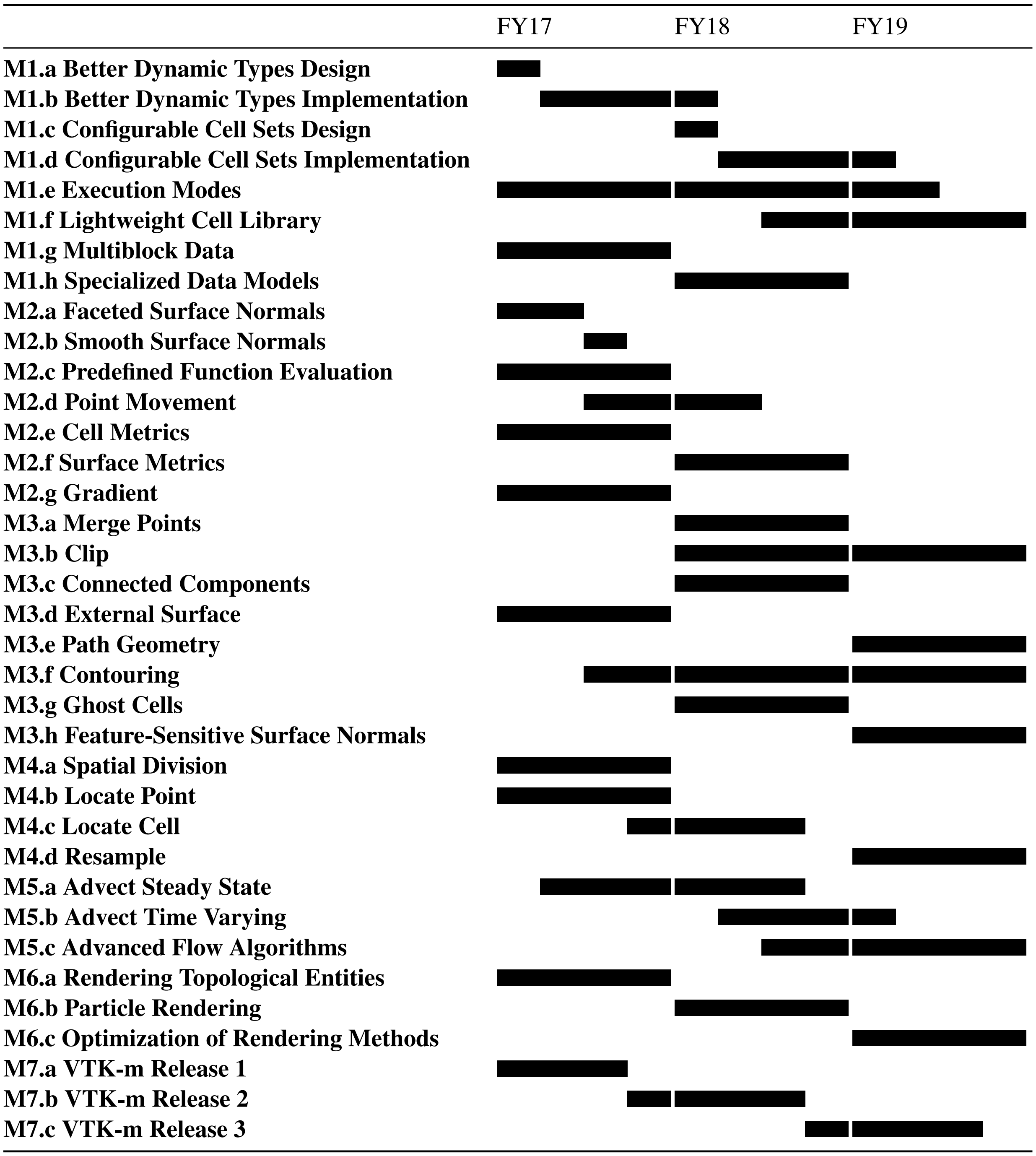


Figure 7: Gantt chart for the planned progress of the ECP/VTK-m milestones.

6 Project Team and Management

6.1 Project Team Capabilities

Name	Affiliation	Role and Responsibility
Kenneth Moreland	SNL	**Project lead.** Participating in the design and implementation of core infrastructure and other algorithms.
Thomas Otahal	SNL	Participating in the design and implementation of multiple algorithms.
Berk Geveci	Kitware	**Kitware lead.** Participating in the design and implementation of core infrastructure.
Robert Maynard	Kitware	Lead developer. Participating in the design and implementation of core infrastructure and algorithms. Lead developer of core infrastructure.
Chris Sewell	LANL	**LANL lead.** Participating in the design and implementation of connected components, contouring, spatial division, and locate point algorithms.
Hank Childs	UO	**UO lead.** Participating in the design and implementation of mesh quality, clip, resample, search, and particle advection algorithms.
David Pugmire	ORNL	**ORNL lead.** Participating in the the design and implementation of particle advection algorithms, rendering, search, ghost and multi-block support, and data models.
Mark Kim	ORNL	Participating in the design and implementation of multiple algorithms.

The team assembled for the ECP/VTK-m project has extensive knowledge in HPC scientific visualization. The team members both have very good practical experience with the DOE needs for scientific discovery and have a history of delivering quality software to DOE facilities. Furthermore, we have included the major designers and developers of the VTK-m software.

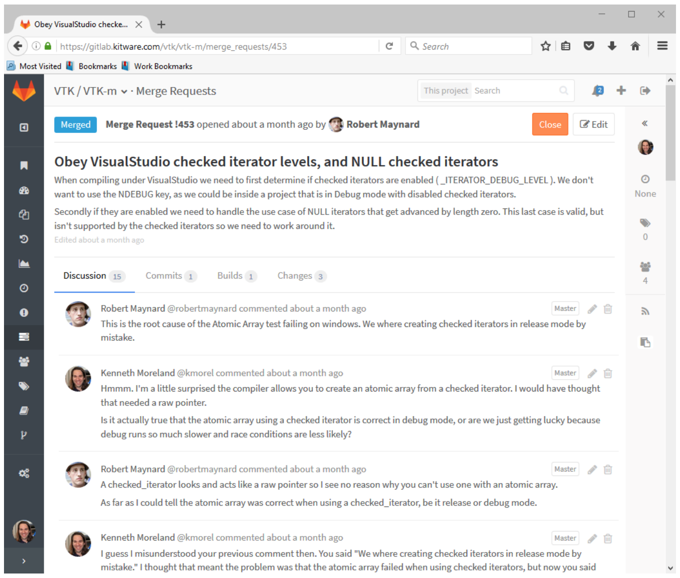
6.2 Project Management Approach

Our project follows in the successful footsteps of the VTK, ParaView, and VisIt projects by leveraging the software itself and drawing personnel from these projects. These projects have demonstrated how agile methods can be applied for bootstrapping complicated software infrastructures and then for maintaining them as long term as community efforts. We will follow proven software methodologies described in more detail in the following sections. Our approach depends on agile techniques, continuous integration, and tight teams that regularly (weekly or more often) communicate and coordinate. We will continue to leverage infrastructure maintained by Kitware for repository hosting, code review, testing, and documentation.

The VTK-m project evolved from three separate projects funded through different mechanisms that came together under ASCR funding. We believe that the successful convergence of these efforts is a clear demonstration that the project management and software development techniques are sound. The team we propose under ECP/VTK-m is practiced in working together, and we will follow a similar style of project management. To manage coordination across the partners, we will hold quarterly teleconferences and annual face-to-face meetings. These meetings will augment the already regular interaction the PIs have through professional activities like conferences and other DOE projects. Also, as previously mentioned, our developers will hold weekly meetings to coordinate software dependencies, development, and requirements. This follows the software process originally established with XVis.

6.2.1 Design and Implementation

Many of the software processes used to develop VTK-m have already been implemented. The software is hosted in a Git repository maintained by Kitware. This repository is served by a customized version of Gitlab which allows for tightly coupled code review and continuous testing/integration. VTK-m is mainly a header only library (using C++ templates). However, the testing suite is built using the cross-platform build tool CMake [21], and we leverage CTest for testing and CDash as a testing dashboard. The integration process is as follows. Each developer who wants to contribute to VTK-m creates a fork of the VTK-m repository under Gitlab. Then the user pushes their own branch under this fork and asks for it to be merged to the official VTK-m repository. This request is reviewed by core VTK-m developers — Gitlab allows for visual reviewing, demonstrated in Figure 8, as well as the ability to pull to a local repository for more extensive testing and reviewing. Each merge request also launches a set of automated testing by a set of testing machines hosted by the VTK-m team for pre-integration testing. This review then leads to either an approval and merge of the branch or a request for further improvements, which leads to another round of code review. We have demonstrated that this workflow scales from small projects to larger community efforts like VTK.



Merge Request !453 opened about a month ago by Robert Maynard

Obey VisualStudio checked iterator levels, and NULL checked iterators

When compiling under VisualStudio we need to first determine if checked iterators are enabled (_ITERATOR_DEBUG_LEVEL). We don't want to use the NDEBUG key, as we could be inside a project that is in Debug mode with disabled checked iterators.

Secondly if they are enabled we need to handle the use case of NULL iterators that get advanced by length zero. This last case is valid, but isn't supported by the checked iterators so we need to work around it.

Edited about a month ago

Robert Maynard @robertmaynard commented about a month ago
This is the root cause of the Atomic Array test failing on windows. We where creating checked iterators in release mode by mistake.

Kenneth Moreland @kmorel commented about a month ago
Hmmm. I'm a little surprised the compiler allows you to create an atomic array from a checked iterator. I would have thought that needed a raw pointer.
Is it actually true that the atomic array using a checked iterator is correct in debug mode, or are we just getting lucky because debug runs so much slower and race conditions are less likely?

Robert Maynard @robertmaynard commented about a month ago
A checked_iterator looks and acts like a raw pointer so I see no reason why you can't use one with an atomic array.
As far as I could tell the atomic array was correct when using a checked_iterator, be it release or debug mode.

Kenneth Moreland @kmorel commented about a month ago
I guess I misunderstood your previous comment then. You said "We where creating checked iterators in release mode by mistake." I thought that meant the problem was that the atomic array failed when using checked iterators, but now you said

Figure 8: An example of using the Gitlab tools to capture comments during a VTK-m code review.

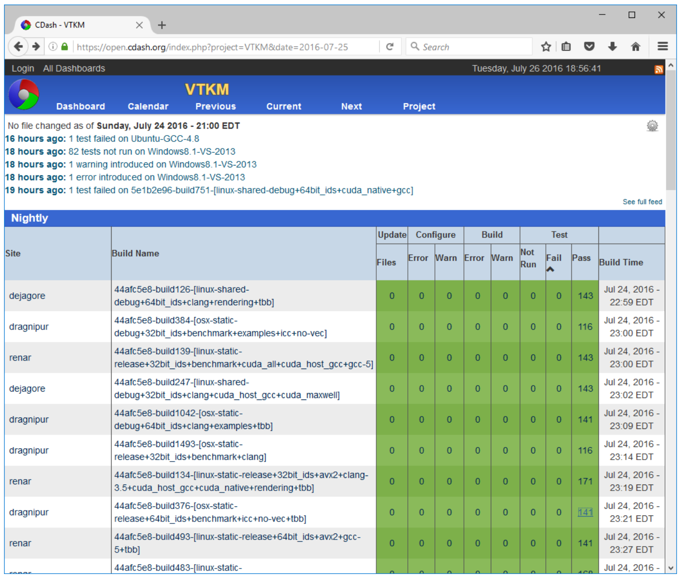
VTK-m was designed from the ground up to use modern approaches to parallelization, so we do not expect to need to significantly redesign existing code to replace dated approaches. This design involves a portable front-end API used by algorithm developers and various back-ends that map to on-the-node programming models such as TBB, CUDA and OpenMP. We will develop other back-ends and/or interface with other ECP software activities as needed during the course of the project. We expect that this approach will minimize the risks while moving forward. We identify potential risks in Section 7.

6.2.2 Testing and Assessment

The VTK-m project was developed with testability and stability as primary concerns. A robust testing infrastructure had to be developed because VTK-m is built around the core idea that it can execute user worklets on a number of different heterogeneous platforms.

VTK-m has three major types of tests. These are device agnostic infrastructure tests, per device infrastructure tests, and per device per visualization algorithm tests. All the infrastructure tests are also tested across 15+ types (float, double, int, etc.). So while VTK-m has 160+ test modules, each module is tested for all the types, allowing us to verify the full combinatorial space. This combination allows us to reach a 95% code coverage rate. We verify that these tests also work across numerous platforms and configurations because VTK-m needs to work on multiple different architectures. We do this by having an automatic testing infrastructure that covers the Linux, OSX, and Windows platforms plus the GCC 4.8, GCC 5.X, Intel 15, Clang 3.1, Clang 3.4, and MSVC 2013 compilers. We also test across multiple different accelerator hardware architectures (Kepler and Maxwell, with Pascal and KNL being added). As the project progresses, we will add new platforms to our testing farm as they are introduced. Furthermore, we expect more testing as VTK-m is included in VTK, ParaView, and VisIt.

We use the CDash tool to automate the running of the test suite on all these platforms and configurations. The tests are automatically run every night on the master branch and also automatically run during the code review cycle. The results of the tests are easily accessible from the CDash web services as shown in Figure 9.



No file changed as of **Sunday, July 24 2016 - 21:00 EDT**
16 hours ago: 1 test failed on Ubuntu-GCC-4.8
18 hours ago: 82 tests not run on Windows8.1-VS-2013
18 hours ago: 1 warning introduced on Windows8.1-VS-2013
18 hours ago: 1 error introduced on Windows8.1-VS-2013
19 hours ago: 1 test failed on 5e1b2e96-build751-[linux-shared-debug+64bit_ids+cuda_native+gcc]

		Update	Configure		Build		Test			
Site	Build Name	Files	Error	Warn	Error	Warn	Not Run	Fail	Pass	Build Time
dejagore	44afc5e8-build126-[linux-shared-debug+64bit_ids+clang+rendering+tbb]	0	0	0	0	0	0	0	143	Jul 24, 2016 - 22:59 EDT
dragnipur	44afc5e8-build384-[osx-static-debug+32bit_ids+benchmark+examples+icc+no-vec]	0	0	0	0	0	0	0	116	Jul 24, 2016 - 23:00 EDT
renar	44afc5e8-build139-[linux-static-release+32bit_ids+benchmark+cuda_all+cuda_host_gcc+gcc-5]	0	0	0	0	0	0	0	143	Jul 24, 2016 - 23:00 EDT
dejagore	44afc5e8-build247-[linux-shared-debug+32bit_ids+clang+cuda_host_gcc+cuda_maxwell]	0	0	0	0	0	0	0	143	Jul 24, 2016 - 23:02 EDT
dragnipur	44afc5e8-build1042-[osx-static-debug+64bit_ids+clang+examples+tbb]	0	0	0	0	0	0	0	141	Jul 24, 2016 - 23:09 EDT
dragnipur	44afc5e8-build1493-[osx-static-release+32bit_ids+benchmark+clang]	0	0	0	0	0	0	0	116	Jul 24, 2016 - 23:14 EDT
renar	44afc5e8-build134-[linux-static-release+32bit_ids+avx2+clang-3.5+cuda_host_gcc+cuda_native+rendering+tbb]	0	0	0	0	0	0	0	171	Jul 24, 2016 - 23:19 EDT
dragnipur	44afc5e8-build376-[osx-static-release+64bit_ids+benchmark+icc+no-vec+tbb]	0	0	0	0	0	0	0	141	Jul 24, 2016 - 23:21 EDT
renar	44afc5e8-build493-[linux-static-release+64bit_ids+avx2+gcc-5+tbb]	0	0	0	0	0	0	0	141	Jul 24, 2016 - 23:27 EDT
	44afc5e8-build483-[linux-static-									Jul 24, 2016 -

Figure 9: A report of automated testing from CDash.

6.2.3 Software Release

The current version of VTK-m has already been released under the 3 Clause BSD License. We will continue to host and develop VTK-m openly under Git, with the Kitware Gitlab being the official repository. It is possible to clone VTK-m under Gitlab or from Github for development and contributing. VTK-m currently has minimal dependencies to fairly stable software stacks such as TBB and compiler / compiler features such as OpenMP and CUDA. We expect that at least part of these technologies will be available on upcoming DOE platforms. Hence, we expect minimal risks to our deployment timeline. We will release VTK-m annually in source code form. We expect that other projects, such as VTK, ParaView, VisIt, Catalyst, and LibSim, will also deploy VTK-m in binary form on DOE platforms although this is outside the scope of this project.

7 Challenges and Risks

Risk/Challenge	Exposure	Mitigation Action
Reliance on Other Projects The ECP/VTK-m project relies on other projects such as SciDAC-4/Data and ECP/ALPINE in particular to provide integration into post hoc and in situ software stacks.	Medium	As a contingency for this integration not happening in another project, we plan for the minimal amount of integration to run VTK-m in our visualization software stack. The scope of the work might need to be scaled back if ECP/VTK-m took the brunt of the integration.
Staffing Developing many-threaded visualization algorithms within VTK-m requires a different mindset than the traditional algorithm development, which will require some training for new developers and potential staffing issues.	Medium	We are writing comprehensive documentation (hundreds of pages) for VTK-m to help new developers get familiar with the system. Co-PI Childs uses VTK-m as part of his graduate curriculum, which gives us an influx of knowledgeable students.
Performance Tradeoffs We are faced with making multiple tradeoffs, such as between portability and performance, between code simplicity and performance, and between compile-time complexity and run-time performance	Medium	The design of VTK-m is intended to be extensible to enable users to customize many of these trade-offs to their needs. In general, the data-parallel programming model enables reasonable default trade-offs to be made.

Architectural Changes Although we have a better idea about what an exascale computer will look like than a few years ago, there is still a great deal of uncertainty about exascale hardware.	Low	VTK-m has been designed to be hardware agnostic. Evolutionary hardware changes can be addressed with modifications to the VTK-m execution environment for the device. Significant hardware changes or new hardware can be addressed with a new execution environment module.
Unknown ECP Requirements Although we have scoped our work to deliver a functioning VTK-m to ECP codes, we acknowledge that some code teams may have requirements of which we are not aware. This is particularly true for codes that are producing outputs different in nature than those found on petascale simulations (e.g., ensembles).	Low	We have budgeted to address specific application science needs that are not addressed by the milestones in Section 5. We feel confident that we have assembled a team capable of responding to these needs.
Adoption of VTK-m To be successful, VTK-m must be included in the visualization software that application scientists use for science discovery. Furthermore, we intend to make an impact to visualization science at large.	Low	We are closely working with the ParaView, VisIt, Catalyst, Libsim and ADIOS teams to make sure that VTK-m is integrated into these tools as it is developed. We will also reach out to the wider community through presentations and tutorials. We are also working on releasing a VTK-m book.

8 Budget Summary

[Budget removed for unlimited release.]

9 Integration

The majority of work outlined in Section 5 comes from analyzing the current visualization software stack and the known needs for scientific discovery. However, as the ECP/VTK-m project progresses, the needs of related ECP activities will shape our work. As such, we expect to collaborate with other projects and fullfill different roles.

Software Technology: Our biggest impact will be integrating with the scientific visualization software stack. We expect the majority of this work to take place in SciDAC, ECP/ALPINE, and other projects. However, this ECP/VTK-m project will perform enough integration to make sure all of this is possible. Additionally, the integration of VTK-m to other software products could necessitate internal changes to VTK-m to make it possible, easier, or more efficient. We will take care of these changes within the ECP/VTK-m project.

Additionally, other ECP software technology projects are likely to need on-node parallelism support for visualization and analysis algorithms. The primary responsibility for creating these multi-core algorithms falls in this ECP/VTK-m project. As necessary, we will collaborate with other ECP software technology projects to develop the APIs and algorithms they need to be successful at exascale. Likewise we expect to leverage other ECP technologies to address off-node parallelism, as well as other problems out of the scope for this project. Although we expect much of this integration with other technologies to be assisted by other projects, the ECP/VTK-m project will facilitate this integration where necessary.

We have been in contact with other proposed ECP projects, and plan to collaborate with them if our project is funded. The ECP/ALPINE project is dedicated to deploying in situ infrastructures to ECP code teams. They plan to use VTK-m for on-node parallelism, meaning that our proposed work is a critical dependency for their effort. We are committed to coordinating with ECP/ALPINE and guiding our development as needed to support their project. The ECP/DIY project is dedicated to providing efficient communication for visualization and analysis infrastructures. This project is complementary to our own, and, together, VTK-m and DIY provide the building blocks for ECP/ALPINE to construct in situ infrastructures. We will coordinate with the DIY team to make sure that the changes we make to VTK and VTK-m's data models are suitable to DIY's needs. The ECP/zfp project is implementing compression technologies designed to be used in place with computation. Such methods are highly complementary with VTK-m where memory storage and bandwidth are often at a premium. We will work with the zfp team to adapt our array structures to integrate compression into VTK-m.

Application Development: The biggest impact VTK-m software will have on ECP applications is through our integration with the existing scientific visualization software stack. This software stack is currently supported at DOE computational facilities, and widely used by application scientists. Looking to the future, many of the ECP applications list VisIt, ParaView, VTK, and VTK-m as key software for their projects. Over the course of the VTK-m project, we will need to support the visualization and analysis needs of ECP applications as they arise. We anticipate that some of these needs might be more generally useful, while others might be special cases. Members of this ECP/VTK-m project have ongoing collaborations with a number of the ECP applications that cover a wide range of scientific disciplines, and types of codes. These include fusion, material science, molecular dynamics, astrophysics, cosmology, neutron transport, and biofuels. These existing relationships put us in an ideal situation to understand and assess the visualization needs for these applications, as well as to collaborate with them on development, testing and deployment of solutions for production use. While it may not be possible to support all of the ECP application requests, we will provide directed support, with guidance from ECP management, of VTK-m functionality in a way that maximizes overall impact and applicability.

We also anticipate cases where providing support for ECP applications involves other ECP software technologies, e.g., visualization tools and in situ frameworks. In these cases we will coordinate and work with both the ECP applications and software projects to understand and assess the requirements and develop a plan for providing the VTK-m functionality.

A References Cited

[1] Acun, B., Gupta, A., Jain, N., Langer, A., Menon, H., Mikida, E., Ni, X., Robson, M., Sun, Y., Totoni, E., Wesolowski, L., and Kale, L. Parallel programming with migratable objects: Charm++ in practice. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (November 2014), pp. 647–658. DOI 10.1109/SC.2014.58.

[2] Ahern, S., Shoshani, A., Ma, K.-L., Choudhary, A., Critchlow, T., Klasky, S., Valerio Pascucci, Ahrens, J., Bethel, E. W., Childs, H., Huang, J., Joy, K. I., Koziol, Q., Lofstead, J., Meredith, J., Moreland, K., Ostrouchov, G., Papka, M., Vishwanath, V., Wolf, M., Wright, N., and Wu, K. J. Scientific Discovery at the Exascale: Report for the DOE ASCR Workshop on Exascale Data Management, Analysis, and Visualization, July 2011.

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

[4] Ahrens, J., Brislawn, K., Martin, K., Geveci, B., Law, C. C., and Papka, M. Large-scale data visualization using parallel data streaming. *IEEE Computer Graphics and Applications 21*, 4 (July/August 2001), 34–41.

[5] Ahrens, J., Geveci, B., and Law, C. Paraview: An end-user tool for large data visualization. In *Visualization Handbook*. Elesvier, 2005. ISBN 978-0123875822.

[6] Ashby, S., et al. The opportunities and challenges of exascale computing. Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee, Fall 2010.

[7] Ayachit, U., Bauer, A., Geveci, B., O'Leary, P., Moreland, K., Fabian, N., and Mauldin, J. 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 (ISAV 2015)* (November 2015), pp. 25–29. DOI 10.1145/2828612.2828624.

[8] Bauer, M., Treichler, S., Slaughter, E., and Aiken, A. Legion: Expressing locality and independence with logical regions. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (November 2012), no. 66. DOI 10.1109/SC.2012.71.

[9] Bennett, J., Clay, R., et al. ASC ATDM level 2 milestone #5325: Asynchronous many-task runtime system analysis and assessment for next generation platforms. Tech. Rep. SAND2015-8312, Sandia National Laboratories, September 2015.

[10] Berzins, M., Schmidt, J., Meng, Q., and Humphrey, A. Past, present and future scalability of the Uintah software. In *Proceedings of the Extreme Scaling Workshop (BW-XSEDE '12)* (2012).

[11] Childs, H., Brugger, E., Whitlock, B., Meredith, J., Ahern, S., Pugmire, D., Biagas, K., Miller, M., Harrison, C., Weber, G. H., Krishnan, H., Fogal, T., Sanderson, A., Garth, C., Bethel, E. W., Camp, D., Rübel, O., Durant, M., Favre, J. M., and Navrátil, P. VisIt: An end-user tool for visualizing and analyzing very large data. In *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*. October 2012, pp. 357–372.

[12] Childs, H., Geveci, B., Schroeder, W., Meredith, J., Moreland, K., Sewell, C., Kuhlen, T., and Bethel, E. W. Research challenges for visualization software. *IEEE Computer 46*, 5 (May 2013), 34–42. DOI 10.1109/MC.2013.179.

[13] CHILDS, H., PUGMIRE, D., AHERN, S., WHITLOCK, B., HOWISON, M., PRABHAT, WEBER, G. H., AND BETHEL, E. W. Extreme scaling of production visualization software on diverse architectures. *IEEE Computer Graphics and Applications 30*, 3 (May/June 2010), 22–31. DOI 10.1109/MCG.2010.51.

[14] DONGARRA, J., BEECHMAN, P., ET AL. The international exascale software project roadmap. Tech. Rep. ut-cs-10-652, University of Tennessee, January 2010.

[15] EDWARDS, H. C., AND TROTT, C. R. Kokkos: Enabling performance portability across manycore architectures. In *Extreme Scaling Workshop (XSW)* (August 2013), pp. 18–24. DOI 10.1109/XSW.2013.7.

[16] FABIAN, N., MORELAND, K., THOMPSON, D., BAUER, A. C., MARION, P., GEVECI, B., RASQUIN, M., AND JANSEN, K. E. The ParaView coprocessing library: A scalable, general purpose in situ visualization library. In *Proceedings of the IEEE Symposium on Large-Scale Data Analysis and Visualization* (October 2011), pp. 89–96. DOI 10.1109/LDAV.2011.6092322.

[17] KINDLMANN, G., WHITAKER, R., TASDIZEN, T., AND MÖLLER, T. Curvature-based transfer functions for direct volume rendering: Methods and applications. In *IEEE Visualization* (October 2003), pp. 513–520. DOI 10.1109/VISUAL.2003.1250414.

[18] LARSEN, M., MEREDITH, J. S., NAVRATIL, P. A., AND CHILDS, H. Ray tracing within a data parallel framework. In *IEEE Pacific Visualization Symposium (PacificVis)* (April 2015), pp. 279–286. DOI 10.1109/PACIFICVIS.2015.7156388.

[19] LO, L., SEWELL, C., AND AHRENS, J. PISTON: A portable cross-platform framework for data-parallel visualization operators. In *Eurographics Symposium on Parallel Graphics and Visualization* (2012). DOI 10.2312/EGPGV/EGPGV12/011-020.

[20] LOFSTEAD, J., ZHENG, F., KLASKY, S., AND SCHWAN, K. Adaptable, metadata rich IO methods for portable high performance IO. In *IEEE International Symposium on Parallel & Distributed Processing, IPDPS'09* (May 2009). DOI 10.1109/IPDPS.2009.5161052.

[21] MARTIN, K., AND HOFFMAN, B. An open source approach to developing software in a small organization. *IEEE Software 24*, 1 (January/February 2007), 46–53. DOI 10.1109/MS.2007.5.

[22] MEREDITH, J. S., AHERN, S., PUGMIRE, D., AND SISNEROS, R. EAVL: the extreme-scale analysis and visualization library. In *Eurographics Symposium on Parallel Graphics and Visualization* (2012), The Eurographics Association, pp. 21–30.

[23] MEREDITH, J. S., AHERN, S., PUGMIRE, D., AND SISNEROS, R. EAVL: The extreme-scale analysis and visualization library. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)* (2012), pp. 21–30. DOI 10.2312/EGPGV/EGPGV12/021-030.

[24] MILLER, R., MORELAND, K., AND MA, K.-L. Finely-threaded history-based topology computation. In *Eurographics Symposium on Parallel Graphics and Visualization* (2014), pp. 41–48. DOI 10.2312/pgv.20141083.

[25] MORELAND, K. Oh, $#*@! Exascale! The effect of emerging architectures on scientific discovery. In *2012 SC Companion (Proceedings of the Ultrascale Visualization Workshop)* (November 2012), pp. 224–231. DOI 10.1109/SC.Companion.2012.38.

[26] MORELAND, K. The ParaView tutorial, version 4.4. Tech. Rep. SAND2015-7813 TR, Sandia National Laboratories, 2015.

[27] MORELAND, K., KENDALL, W., PETERKA, T., AND HUANG, J. An image compositing solution at scale. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11)* (November 2011). DOI 10.1145/2063384.2063417.

[28] MORELAND, K., KING, B., MAYNARD, R., AND MA, K.-L. Flexible analysis software for emerging architectures. In *2012 SC Companion (Petascale Data Analytics: Challenges and Opportunities)* (November 2012), pp. 821–826. DOI 10.1109/SC.Companion.2012.115.

[29] MORELAND, K., LARSEN, M., AND CHILDS, H. Visualization for exascale: Portable performance is critical. *Supercomputing Frontiers and Innovations 2*, 3 (2015). DOI 10.14529/jsfi150306.

[30] MORELAND, K., SEWELL, C., USHER, W., TA LO, L., MEREDITH, J., PUGMIRE, D., KRESS, J., SCHROOTS, H., MA, K.-L., CHILDS, H., LARSEN, M., CHEN, C.-M., MAYNARD, R., AND GEVECI, B. Vtk-m: Accelerating the visualization toolkit for massively threaded architectures. *IEEE Computer Graphics and Applications 36*, 3 (May/June 2016), 48–58. DOI 10.1109/MCG.2016.48.

[31] PETERKA, T., ROSS, R., KENDALL, W., GYULASSY, A., PASCUCCI, V., SHEN, H.-W., LEE, T.-Y., AND CHAUDHURI, A. Scalable parallel building blocks for custom data analysis. In *Proceedings of Large Data Analysis and Visualization Symposium LDAV'11* (October 2011), pp. 105–112. DOI 10.1109/LDAV.2011.6092324.

[32] RICHARDS, M., ET AL. Exascale software study: Software challenges in extreme scale systems. Tech. rep., DARPA Information Processing Techniques Office (IPTO), September 2009.

[33] SCHROEDER, W., MARTIN, K., AND LORENSEN, B. *The Visualization Toolkit: An Object Oriented Approach to 3D Graphics*, fourth ed. Kitware Inc., 2004. ISBN 1-930934-19-X.

[34] STEVENS, R., WHITE, A., ET AL. Architectures and technology for extreme scale computing. Tech. rep., ASCR Scientific Grand Challenges Workshop Series, December 2009.

[35] WHITLOCK, B. Getting data into VisIt. Tech. Rep. LLNL-SM-446033, Lawrence Livermore National Laboratory, July 2010.