The International Journal of High Performance Computing Applications
2024, Vol. 38(5) 508–526
©The Author(s) 2024
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/10943420241270969
www.sagepub.com/
SAGE

Visualization at exascale: Making it all work with VTK-m

Kenneth Moreland,[1] Tushar M. Athawale,[1] Vicente Bolea,[2] Mark Bolstad,[3] Eric Brugger, Hank Childs,[4] Axel Huebl,[5] Li-Ta Lo,[6] Berk Geveci,[2] Nicole Marsaglia,[7] Sujin Philip,[2] David Pugmire,[1] Silvio Rizzi,[8] Zhe Wang,[1] and Abhishek Yenpure[2]

Abstract
The VTK-m software library enables scientific visualization on exascale-class supercomputers. Exascale machines are particularly challenging for software development in part because they use GPU accelerators to provide the vast majority of their computational throughput. Algorithmic designs for GPUs and GPU-centric computing often deviate from those that worked well on previous generations of high-performance computers that relied on traditional CPUs. Fortunately, VTK-m provides scientific visualization algorithms for GPUs and other accelerators. VTK-m also provides a framework that simplifies the implementation of new algorithms and adds a porting layer to work across multiple processor types. This paper describes the main challenges encountered when making scientific visualization available at exascale. We document the surprises and obstacles faced when moving from pre-exascale platforms to the final exascale designs and the performance on those systems including scaling studies on Frontier, an exascale machine with over 37,000 AMD GPUs. We also report on the integration of VTK-m with other exascale software technologies. Finally, we show how VTK-m helps scientific discovery for applications such as fusion and particle acceleration that leverage an exascale supercomputer.

Keywords
visualization, exascale, VTK-m, parallel, GPU, frontier, auroria, in situ, exascale computing project

1 Introduction

As its name would imply, the goal of the Exascale Computing Project (ECP) was to build and make practical the world's first exascale supercomputers. Although this goal was nominally to stand up computers capable of a billion-billion (10^{18}) floating point operations per second, this advancement required a revolution in both hardware and software. Critical to this advancement was the adoption of GPUs (from a variety of vendors) to be used as the primary computing engine. These GPUs provided unmatched computational throughput relative to the power they consume, albeit at the cost of greater code complexity.

Scientific simulations running on exascale supercomputers can produce datasets of unprecedented scale, and visualization and analysis approaches are frequently used to understand the resulting data and promote discovery. That said, the scale of the data produced by simulations on exascale computers requires the visualization and analysis approaches themselves to utilize significant computational resources. Most commonly, this computation is conducted on the same supercomputer and hardware that produced the data in the first place.

The VTK-m software library makes it possible to process these enormous datasets from exascale computers by implementing classic scientific visualization algorithms that have been redesigned for heavily threaded environments such as supercomputers with GPUs. VTK-m also provides a framework that simplifies the implementation of new algorithms and adds a porting layer to work across multiple processor types. This porting layer allows algorithms written in VTK-m to be written once and run everywhere, which alone saves a substantial amount of developer effort. At the inception of the ECP, VTK-m was the byproduct of a research project. The ECP provided the investment to advance VTK-m to production software. This software now serves as the underlying visualization implementation across

[1]Oak Ridge National Laboratory, USA
[2]Kitware, Inc., USA
[3]Sandia National Laboratories, USA
[4]University of Oregon, USA
[5]Lawrence Berkeley National Laboratory, USA
[6]Los Alamos National Laboratory, USA
[7]Lawrence Livermore National Laboratory, USA
[8]Argonne National Laboratory, USA

Corresponding author:
Kenneth Moreland, Oak Ridge National Laboratory, P.O. Box 2008, 1 Bethel Valley Rd., Bldg. 5700, MS 6164, Oak Ridge, TN 37831-6164
Email: morelandkd@ornl.gov

Table 1. Visualization algorithms accelerated by VTK-m.

Goals of visualization	Algorithms accelerated by VTK-m
Scalar field visualization	Contour (Lo et al. 2012), Threshold (Maynard et al. 2013)
Vector/Flow field visualization	Particle advection (Pugmire et al. 2018), Finite-time Lyapunov exponent (FTLE) (Sane et al. 2021b), Poincaré plot (Suchyta et al. 2022a)
Geometry Refinement	External faces (Lessley et al. 2016, 2017), Surface simplification (Moreland et al. 2016), Point merging (Yenpure et al. 2019)
Rendering	Volume rendering (Larsen et al. 2015a), Surface rendering (Larsen et al. 2015b)
Reduction and compression	Wavelet compression (Li et al. 2017), Statistical models (Wang et al. 2019), Lagrangian Representations (Sane et al. 2021a,b)
Topological analysis	Contour trees (Carr et al. 2021)
Uncertainty visualization	Uncertainty isosurface (Wang et al. 2023)

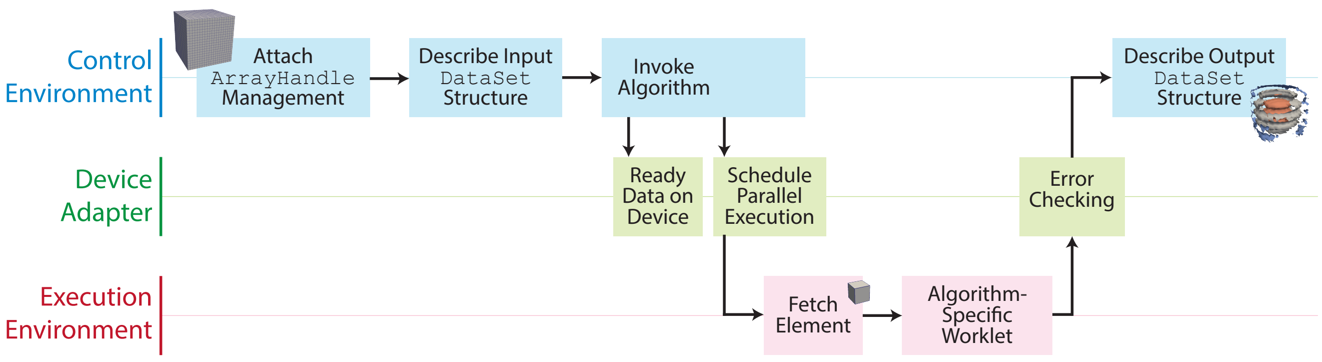

Control Environment
Device Adapter
Execution Environment
Attach ArrayHandle Management
Describe Input DataSet Structure
Invoke Algorithm
Describe Output DataSet Structure
Ready Data on Device
Schedule Parallel Execution
Error Checking
Fetch Element
Algorithm-Specific Worklet

Figure 1. Workflow of an algorithm implemented in VTK-m. The “control environment,” which runs on a single thread on the host, manages data organization and orchestrates operations. The “execution environment,” which runs on many threads on the device, performs parallel execution across elements of data. A “device adapter” manages data movement and control flow between these two environments.

the entire ECP. VTK-m is currently used by ParaView, VisIt, and Ascent for the portion of their visualization algorithms that execute on the accelerator processors of exascale platforms and on other GPU-centric supercomputers.

This paper describes the challenges encountered when making visualization available at exascale. It begins with a brief overview of the VTK-m framework. It then describes the most significant porting challenges encountered during the ECP given the hardware for the production exascale machines differed dramatically from the pre-exascale machines. This is followed by a summary of the performance of VTK-m on the exascale hardware including scaling studies on an exascale computer of over 37,000 AMD GPUs. Finally, the paper concludes with a discussion on the software engineering required to integrate VTK-m into usable visualization tools and successes with using VTK-m to solve problems in real science applications. This includes porting challenges, performance testing and improvement, integration with other ECP software technologies, and the support of ECP applications.

2 Overview of VTK-m

The VTK-m library (Moreland et al. 2016) began as a research project funded by the Advanced Scientific Computing Research program within the US Department of Energy’s (DOE’s) Office of Science. The project’s goal was to enable scientific visualization on emerging high-performance computing (HPC) systems via two approaches: (1) by serving as a repository for interoperable scientific visualization algorithms well suited to accelerator architectures and (2) by providing a framework that simplifies the development of visualization algorithms that can be ported across many accelerator devices.

At the onset of the ECP, VTK-m contained only the most common operations for scientific visualization: contour, threshold, external faces, basic surface simplification, and rendering. Although this initial set of operations is useful, users almost always require more functionality. The ECP enabled this additional functionality to grow with the introduction of new algorithms and performance improvements to the existing ones. Table 1 contains a selection of algorithms currently provided by VTK-m. These added features provided the necessary functionality for the tools and applications that utilized VTK-m to execute on exascale machines and similar hardware.

The basic workflow for an algorithm in VTK-m’s framework is shown in Figure 1. The framework separates code into two environments: control and execution. In a GPU development environment, control corresponds to the “host,” and execution corresponds to the “device.” This separation is maintained even when there is not a clear separation between host and device, which is the case for some accelerators such as the Xeon Phi (Jeffers et al. 2016).

Algorithms in VTK-m execute parallel routines by wrapping them in a functional object that will be passed to the device in the execution environment. There, it will be run on many threads, and each instance will be fed a small, isolated portion of the data. This functional object is called a “worklet” because it works on a small portion of data.

The worklet API in the execution environment is designed to promote thread safety to simplify the implementation and to prevent memory access hazards. It is also well suited for modern devices that require many (thousands or more) parallel threads to run efficiently, which are the type of devices VTK-m is designed for.

To achieve portability, VTK-m contains a device adapter that manages interaction with a variety of devices. The entirety of VTK-m can be ported with a change to the device adapter. Furthermore, all execution code ("device code") is implemented with standard C++14 and can thus be compiled for any device supporting this very common programming environment.

VTK-m uses multiple techniques to achieve efficient parallelization. One technique is to enable data to be divided into small pieces for parallel execution by using a flexible data model (Meredith et al. 2012). A second technique is to utilize data parallel primitive methods (Blelloch 1990). Data parallel primitives allow algorithms to be implemented as a sequence of data parallel operations such as map, scan, sort, and reduce. Early work explored how to implement scientific visualization using data parellel primitives (Lo et al. 2012). Map applies an operation to each datum and is a convenient way to specify parallel operations. Scan, which produces a running sum, product, or other associative operation, is useful for building indicies after counting. Sort reorders data to make duplicates easy to find. Reduce provides a total sum or product for quick accumulations.

To simplify the implementation further, VTK-m provides meta data-parallel primitives (Moreland et al. 2021) that incorporate common patterns for scientific data that were demonstrated to be efficient. For example, VTK-m provides a meta data-parallel primitive to map an operation to every cell of a mesh while internally handling all the indexing of shape and incident point information. Finally, note that the VTK-m design achieves developer efficiency with streamlined algorithm development and automatic porting to new architectures with an efficient implementation.

3 Porting challenges

Although the pre-exascale machines provided good experience to the VTK-m development team for porting to different processor architectures, the design of the Frontier and Aurora exascale machines introduced new technical challenges. This section reports the most significant modifications required to make it feasible to run VTK-m on exascale hardware.

3.1 Adopting Kokkos

As described in Section 2, one of the main features that VTK-m supports is "write once, compile anywhere" for its algorithms. A device portability layer called a device adapter allows algorithms written with standard C++ features to run across all devices. These device adapters wrap a parallel device into a common API that provides functionalities for parallel-task scheduling, memory management, atomic operations, and several commonly used parallel algorithms such as scan, sort, and reduce through a common interface. These are implemented on top of the native libraries used to program these devices.

When the architectures for Aurora and Frontier were announced, it was revealed that they would be using two completely new GPU architectures from two different vendors (i.e., Intel GPUs in Aurora and AMD GPUs in Frontier) with their own native libraries (SYCL for Intel and HIP for AMD). Porting VTK-m directly to these new archictectures would have required two new device adapter back ends. Although the device adapter layer greatly simplifies the porting of VTK-m code, implementing device adapters themselves is not trivial. So, although it would be technically feasible to design device adapters for two new devices, it would have required significant effort.

To overcome this hurdle, we looked to another ECP project called Kokkos (Edwards et al. 2014; Trott et al. 2022). Kokkos is a library for implementing performance-portable applications in C++. Similar to VTK-m, Kokkos also supports multiple device back ends, including SYCL and HIP, which are used by the exascale systems. So, with just one device adapter built on top of Kokkos, we were able to target both the machines. This allows us to write one VTK-m device adapter to target the Kokkos API and defer the work of interfacing with the different ECP device APIs to the Kokkos team, who were already doing this for other ECP projects.

Because the exascale machines were based on completely new hardware, Kokkos was also in active development when we were developing the VTK-m device adapter. Therefore, we faced a few challenges during the development, and some of the most interesting challenges we encountered are described below.

As mentioned previously, VTK-m filters and algorithms are implemented as a sequence of some primitive parallel operations such as scan, sort, and reduce. To achieve good performance, the implementation of these parallel primitives must be as efficient as possible. VTK-m comes with generic implementations for these operations, but generality precludes sufficient optimization for target architectures. Fortunately, device adapters specialize and optimize these operations for their target hardware.

For the Kokkos device adapter, we relied on the implementations provided by the Kokkos library. However, we discovered that Kokkos did not have all of the primitives that VTK-m supported, and, even when available, some implementations were not as flexible as those supported natively by VTK-m. For example, at the time of development, the Kokkos sort only supported floating point values with the less-than operator as the ordering comparator, whereas VTK-m supported all primitive types and also non-primitive types with custom comparison operators. Therefore, we had to implement code paths to use the Kokkos sort for only supported arguments and to fall back to the generic VTK-m implementations for all other cases.

Another issue we encountered were bugs in the Kokkos code base. During development, we hit several critical bugs that would crash the program. Therefore, we also had to work around these failing features with a fallback code path for failing conditions until these issues could be addressed.

We also faced challenges getting the Kokkos library initialization to work with VTK-m. Kokkos only gives one option to specify configuration options at the library initalization during program startup. In contrast, VTK-m's initialization is optional and allows the configuration to be selected more flexibly. This required us to rethink and overhaul our own initialization procedure so that all of our dependencies were initialized correctly and in the correct order. Furthermore, using VTK-m with other libraries that also use Kokkos is a valid use case, but only one of these libraries should initialize and finalize the Kokkos library. To prevent it from "stealing" the Kokkos initialization from other code, VTK-m delays Kokkos initialization until it is required and detects whether the Kokkos system is already initialized. If VTK-m does initialize Kokkos, then it establishes a callback at program termination to finalize the Kokkos library.

3.2 Addition and then removal of virtual methods

One of the challenges of a general purpose visualization library like VTK-m is that it is intended to support data coming from all manner of data producers. It cannot make assumptions about data structures from the base data types (float vs double vs int) to the arrangement and interpolation of values in mesh structures. As such, it requires sophisticated software engineering to enable flexibility and adaptability that is practical both for developers and users of VTK-m.

VTK-m uses C++ templates to customize algorithms for different data types. Ideally, an algorithm will know the data types on which it will operate. Unfortunately, this is seldom the case for VTK-m because data ingested from different sources can have any number of basic types and memory layouts. In the early years of the ECP, this problem was addressed by compiling algorithms in VTK-m for all possible types that could be encountered. However, the amount of potential cases generated is too many to be practical.

C++ objects with virtual methods are a natural solution to this problem, and when they became available on CUDA devices, VTK-m started using virtual methods to hide the structure of arrays. This originally addressed some of the issues with type abstraction. However, this solution worked poorly and was later abandoned for two reasons. First, using virtual methods introduced problems with library linking because the compilers had to collect any possible code that might need to be loaded on the device. In addition to the obvious problem of bloat from replicating object code across all libraries, the process required a fragile and slow build step, and it precluded the possibility of adding further subclasses after the library was built. Second, when the ECP announced that its exascale machines would be using GPUs from Intel and AMD (i.e., not using NVIDIA's CUDA), it was unclear how well they would support virtual methods or even if they would support them at all.

Consequently, the VTK-m development team pivoted, and virtual methods were removed from the VTK-m code that ran on devices. To manage the operations on types without using virtual methods, VTK-m employed a trio of strategies: multiplexing the type in the algorithm, generalizing the stride in arrays, and providing fall backs when unexpected types were encountered.

The first strategy, multiplexing, required using a type-agnostic storage object. For this, a `Variant` class was added to VTK-m. The `Variant` takes a list of types as template parameters, and it can hold exactly one of these objects at a time. At run time, the proper type can be queried and extracted. Although multiplexing from a variant object still requires separate compilation for all possible types, it limits the code that must be recompiled to make it more manageable.

The second strategy required a redesign of the array management in VTK-m. Where the original design of array management completely abstracted the implementation, the new design based the array management on raw buffers of memory that internally can be reinterpreted as C arrays to implement different array types. This in turn provided a way to generalize the representation of an array component by defining a stride in the buffer. In this way, an algorithm no longer had to be compiled for a specific data layout. It could instead apply a stride to the array and use any layout.

The third strategy recognized that although many types are possible, most are rarely encountered in practice. Therefore, instead of attempting to compile a function for any possible type, it is possible to instead compile for the most likely types and then have a fallback for the cases when the type is unexpected. The typical solution is to copy the array of an unknown type to an array of a known type. This feature was included in the filter interface overhaul, which is described in the next section.

3.3 Filter interface overhaul

The majority of algorithms in VTK-m are contained in what is called a "filter" object. A filter takes a dataset, performs some operations to modify it, and returns the resulting dataset. Filters provide the outwardly facing API to process data in VTK-m. However, as previously described, these datasets can have any number of data types and structures. The VTK-m filter base class needs to provide a mechanism to resolve data types. That is, the filter base class needs a way to call a templated method in a derived implementation class with fully resolved data types.

Because a C++ template method cannot also be a virtual function, which is needed for typical run-time polymorphism, the original design used a rather complicated technique called the curiously recurring template pattern (CRTP) (Coplien 1995) to emulate it. CRTP works by making the type of derived class a template argument to the base class. When the base class needs to call a method in the derived class, rather than use a virtual method, it recasts itself as the derived class and calls the method directly. This allows the base class to iterate through a list of supported data types and call a templated method in the derived class to implement the algorithm on each one.

However, using CRTP also made the base filter class itself a class template, and its execution methods required exposed definitions that the compiler internally recompiled each time they were called elsewhere in the code. This made the VTK-m filter library essentially a header-only library, and clients using it had to include all the necessary implementation in the header files. When compiling client code, all those header files had to be parsed, and classes and methods had to be instantiated. As a consequence, it took a long time to compile applications of VTK-m. This issue was most prominent when compiling for GPU devices because most device compilers were not as efficient as host compilers when handling C++ templates. When integrated as part of an in-situ pipeline, simulation code that called VTK-m's filters also had to be compiled by a device compiler, thus exacerbating the problem. In some extreme cases, the entire compilation process took more than 24 hours to complete.

The new design changed the type-dispatching mechanism. The responsibility of type dispatching was shifted from the filter base class to the derived implementation classes. The execution method in the derived class is no longer a template; it accepts a dataset that contains ambiguous types. This increases the burden on the algorithm developers who must now resolve types themselves. To compensate, VTK-m provides an alternate dispatching mechanism by providing various forms of a "cast-and-call" mechanism on the dataset's elements. The derived filter performs the type dispatching by passing a dataset element to a cast-and-call along with a templated, callable object. This callable object is generally the type-dependent, core business logic of the filter implementation wrapped inside a C++14 generic lambda expression. The lambda expression will be instantiated with types from a predefined type list by the cast-and-call mechanism.

The cast-and-call mechanism will attempt to resolve the data to a prescribed list of types. When the data type is not part of the type list, the cast-and-call mechanism can fall back to a known type, as described in Section 3.2. The predefined type list and fallback limit the number of instantiations of the lambda expression. Using cast-and-call with lambda expressions in this way also limits the portion of code to be instantiated. In contrast, the original design required the entire body of the filter implementation to be instantiated, which can increase both compilation time and code bloat.

Because the filter object methods are no longer template methods, they can be virtual methods. Note that the filter methods in question are explicitly run in VTK-m's control environment, and this means they will only be run on the CPU host. Previously discussed issues with virtual methods on GPUs do not apply here, so creating virtual methods is not problematic in this case. Moreover, the virtual methods eliminate the need to use the CRTP technique. Consequently, the entire filter class hierarchy has become a non-template class as well. The library is no longer a header-only library and can be built as a traditional, precompiled library. The filter library is further divided into several modules with each separately compiled into a `.so` file. This allows subsequent linking to potentially load less memory from libraries. It also allows users to turn off the compiling of libraries they do not need for faster compilation. Finally, applications no longer need to be compiled by a device compiler simply because they are using VTK-m.

The new filter structure, which removes exposed templates and encapsulates code into libraries, greatly reduces application build time because applications no longer have to compile many VTK-m templates. Rather, applications simply link to methods in the VTK-m library. The new approach also provides other compile-time saving opportunities within the library. For example, some filters incorporate the functionality of others as a subroutine. The new filter structure allows filters to be compiled once and have their functionality leveraged by other filters. For example, the compilation of VTK-m's material interface reconstruction filter was reduced by a factor of 9 by linking to the mesh quality filter rather than recompiling it.

The new filter structure also enables dividing the code into multiple translation units (i.e., separate C++ source files). Although this does not necessarily reduce the aggregate compile times, it more effectively leverages cores in parallel builds and reduces the possibility of compiler crashes from running out of resources.

3.4 GPU-to-GPU transfers

Modern GPUs used in HPC can perform direct GPU-to-GPU communication. This provides GPUs with an efficient mechanism to send data stored in their device memory directly to another GPU's device memory. This contrasts with the traditional and costly GPU communication pattern that required first copying the desired data from the first GPU's device memory to the system's host memory and then copying the data again from the host memory to the second GPU's device memory.

Enabling this feature in VTK-m required changes in the source code of both VTK-m itself and DIY, which is a third-party library that VTK-m uses to manage its MPI (Message Passing Interface) communication (Peterka et al. 2011; Morozov and Peterka 2016).

The DIY library changes consisted of adding routines that allow sending and receiving raw pointers directly and encapsulated in a newly introduced "blob" data type, which is roughly equivalent to the raw data buffers previously described for VTK-m's array management. This contrasts with the regular operation of DIY, which usually copies and serializes data before sending and receiving. Additionally, we introduced a new API in DIY to control the ownership of the passed raw pointer.

The corresponding VTK-m changes consisted of modifying the VTK-m class that manages memory buffers and their location among host and devices. In particular, the changes overwrite the DIY serialization to directly pass these memory buffers to and from DIY. This enables direct GPU communication in VTK-m because most of the VTK-m storage entities are composed of VTK-m buffers.

The main challenge found during the implementation of this feature was maintaining compatibility with the Frontier target system. Frontier's software stack was a moving target with frequent updates that—in many cases—required us to modify build and runtime parameters and—in some other cases—required us to change the VTK-m and DIY source code. This challenge was minimized by provisioning the VTK-m GitLab project with nightly jobs to build and run tests on the Frontier test-bed system. This extra testing allowed us to quickly identify problems that arose from changes to either VTK-m's source code or to the Frontier software stack.

4 Operation on exascale hardware

The ECP's exascale platforms are unique in many ways. The previous section summarized the major software development hurdles faced when preparing for the platforms. This section describes further technical challenges in compiling and executing VTK-m.

4.1 Frontier

Frontier was the first system delivered under the ECP program (Atchley et al. 2023). Deployed at the Oak Ridge Leadership Computing Facility (OLCF), Frontier is also the first system to achieve exascale by hitting 1.19 exaflops on the High-Performance Linpack benchmark. Frontier is an AMD-based system with 9,472 AMD Epyc CPUs (with 9,408 reserved for compute only) totalling over 600,000 cores. However, the majority of the processing power comes from 37,888 AMD MI250X GPUs (with 37,632 reserved for compute only) totalling over 8.3 million cores. The system is organized into 74 racks, each comprising 64 blades with 2 nodes each. A node consists of a single CPU and 4 GPUs with 4 TB of RAM. Each GPU consists of two Graphics Compute Dies (GCD), which means that a node presents itself to device code as having 8 GPUs. That leaves us with a total count of 75,776 GCDs (with 75,264 reserved for compute only).

Porting to a new system that is also under active development brings many challenges. When the pre-exascale development systems for Frontier became available, VTK-m frequently broke the compiler. At one point in the build, test, and release cycle for the compilers, AMD had the VTK-m team building the compilers from bleeding-edge source to help debug the crashes.

As the compilers stabilized, the team discovered another issue—astronomically long compile times. As an anecdote for a particularly long compile time, VTK-m development switched over to a second system when it came online. A week later, the system administrators for the pre-exascale systems contacted us to ask if they could kill a compile job because it was consuming all the memory on the login node. It turned out that a VTK-m compile was still running on the initial system more than a week later. More commonly, VTK-m would take 20 hours to completely compile. The team would work with AMD to test a new compiler, and the cycle would begin anew. Thankfully, these compile-time issues have been corrected. Currently, a full build of VTK-m with tests and benchmarks can be completed on Frontier in under 20 minutes.

Along the way there were many issues that required solutions from AMD, Kokkos, and the VTK-m teams. Some examples are as follows:

- *Compiler Errors.* VTK-m uses template metaprogramming (Meyers 2005). Although this code is fully C++14 compliant, it sometimes flexes the compiler in unexpected ways and may cause internal compiler errors. Resolving these issues requires dialog with the compiler engineers.

- *Optimizer Internal Looping.* In addition to compiler fixes, certain portions of the code found the compiler optimizing the code too aggressively in extended internal loops. Hints were added to prevent the compiler from optimizing out of control. Furthermore, the VTK-m source was broken into smaller units to reduce template instantiation and to reduce compiler resource utilization, as described in Section 3.3.

- *Degradation of Sorting Parallelism.* The original Kokkos sorting algorithm had not accounted for VTK-m workloads that used integral types (e.g., sorting keys). A binning algorithm that worked well for uniformly distributed values degraded to a largely serial process when many values landed in a single bin. A simple fix by the Kokkos team provided a 12× speedup in these workloads.

- *Function Pointers Unsupported on AMD GPUs.* VTK-m was modified over several months to remove all calls through function pointers as part of the removal of virtual methods (as described in Section 3.2).

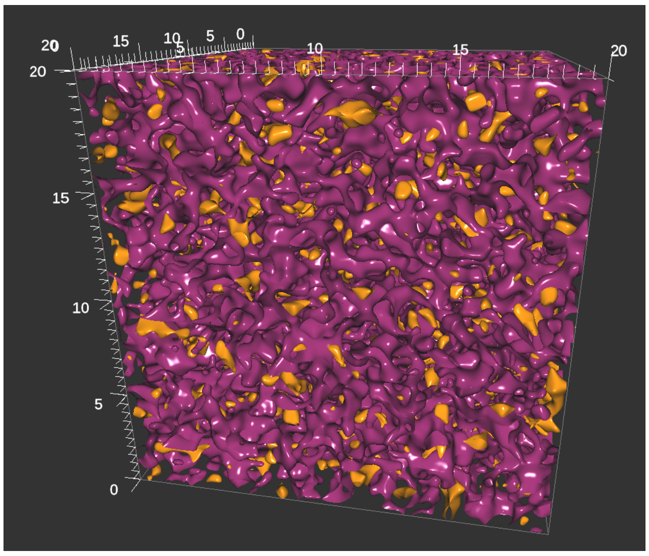


Figure 2. Output from VTK-m scaling study that demonstrated rendering at full scale on the Frontier supercomputer.

When Frontier became available to the various code teams, the VTK-m team at the University of Oregon undertook an experiment to establish VTK-m's scalability. The team created Perlin noise data comprising 80 trillion cells and distributed for a full-system run that used 9,400 nodes and 74,088 GPU GCDs. The total time to render this dataset was 300 ms (not including other processing such as isosurface extraction). Figure 2 shows an example output from this test.

4.2 Aurora

The Aurora supercomputer is deployed at the Argonne Leadership Computing Facility (ALCF) and is expected to deliver over 2 exaflops of computational power. With 166 racks and 10,624 nodes, it boasts 21,248 Intel Xeon Max Series CPUs and 63,744 Intel Data Center Max Series GPUs.

Our journey on Aurora began in August 2020 when we started building VTK-m on the Joint Laboratory for System Integration's test platform at Argonne National Laboratory. We began our work using the Iris test platform, which consisted of 20 nodes equipped with Intel Gen 9 GPUs. By September 2021, we transitioned to the Arcticus test bed, which featured 17 nodes equipped with 2× Intel

Server GPUs code-named Arctic Sound. In August 2022, we conducted tests on the Florentia test platform, which contained early versions of the Aurora GPUs. Finally, by January 2023, our efforts shifted to Sunspot, the Aurora test and development system, which had 128 nodes with 2× Intel Xeon Max Series CPUs and 6× Intel Data Center Max Series GPUs. The first light on Aurora occurred in August 2023.

Throughout our porting and testing efforts, we maintained close collaboration with Intel engineers and ALCF performance engineering staff. This collaboration led to the identification and resolution of several bugs in successive versions of the Intel oneAPI software development kit. In November 2020, we completed a first set of early VTK-m benchmarks on Iris by using Kokkos with OpenMP target offload. However, in February 2021, we encountered issues with virtual methods when transitioning to Kokkos with SYCL. Fortunately, the team anticipated this problem and had already started the process of replacing virtual methods (see Section 3.2), and this issue was resolved soon after. Additionally, we worked with Intel to address long build times.

By December 2023, the latest VTK-m and Kokkos development branches were successfully running on Aurora and achieving a 97% success rate in regression testing. Porting, deployment, and integration efforts are set to continue on Aurora as we move beyond the ECP era.

5 Integration into visualization tools

Throughout the life of the ECP, the VTK-m team collaborated heavily with other ECP software technology teams. The scope of the VTK-m project was to provide the fundamental technology to run scientific visualization algorithms on GPUs, which account for the vast majority of the computational power of current exascale machines. Other ECP teams, most notably the ALPINE project, developed tools that would leverage VTK-m while directly addressing application needs. This arrangement avoided the redundant work of multiple teams developing their own visualization solutions and prevented users from having to use yet another software interface. In this section, we discuss the major visualization tools that we integrated VTK-m with.

5.1 ParaView

The VTK-m library provides high-performance implementations of several visualization algorithms for highly parallel processors. However, features such as file I/O, rendering, and pipeline management, which are all essential parts of a full-featured visualization toolkit, are beyond the scope of VTK-m. On the other hand, ParaView is a mature visualization software that has robust implementations of these features. Therefore, we wanted to integrate VTK-m into ParaView in such a way that ParaView can use VTK-m filters to accelerate its operations when a VTK-m implementation and hardware that concurrently executes many threads is available.

We also wanted the VTK-m accelerated filters to be as easy to use as possible. Therefore, we chose to integrate VTK-m using VTK and ParaView's factory-instantiation feature. Because ParaView is implemented on top of VTK and internally relies on VTK filters, both VTK and ParaView will be mentioned interchangeably in this section. Filters in ParaView are instantiated via a factory method. There can be multiple implementations available for a filter, and the factory method chooses an appropriate implementation at run time based on given criteria. With this method, we can override the default ParaView filters with VTK-m-based filters. Currently, VTK-m accelerated filters that override traditional CPU implementations are available for some commonly used filters such as contour, threshold, and gradient. Additional VTK-m filters can be added by providing additional overrides.

To override a ParaView filter, we first need to implement a VTK-m wrapper filter in VTK to provide the interface of the base VTK/ParaView filter and use VTK-m filters and routines for its operation. The following steps provide a high-level overview of how a VTK-m wrapper filter is implemented in VTK/ParaView:

1. Check the filter parameters and only proceed with VTK-m processing for configurations supported by the VTK-m filter implementation.
2. Convert the input VTK datasets to VTK-m datasets.
3. Execute the VTK-m filter on the data.
4. Convert the output of the VTK-m filter back to VTK datasets.
5. If an error occurs at any point during the above steps, then fall back to the default VTK implementation. Errors in VTK-m are typically signaled via C++ exceptions.

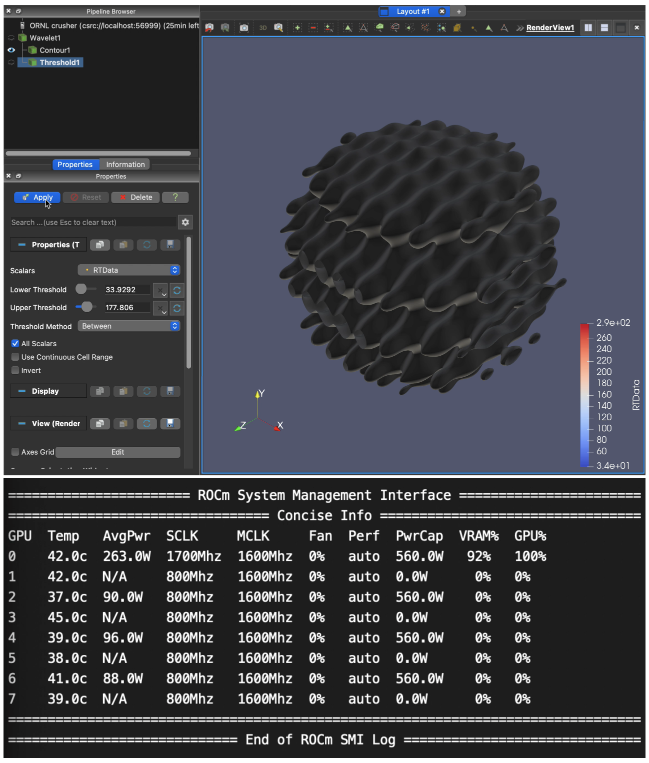
For the dataset conversion from VTK to VTK-m and back, we implemented several helper routines. These are zero-copy operations for most cases because, whenever possible, only the ownership of the pointers to the underlying resources is transferred. Even copies from host-to-device and device-to-host are minimized by using a VTK-m dataset wrapper in VTK called `vtkmDataSet`, which implements the `vtkDataSet` interface and only copies the data when required. Another commonly used ParaView functionality is computing the range of the various fields of a dataset. This has also been accelerated by using VTK-m, which speeds up the computation and avoids memory transfer from device to host.

Figure 3 shows an example of ParaView running with VTK-m accelerated filters on Crusher, which is an early access test bed for the Frontier system. As described in Section 3.1, VTK-m is using the Kokkos device adapter on this hardware. The bottom of the image shows the output of the `rocm-smi` command, which is used to verify and monitor the GPU usage by the filters.

VTK-m accelerated filter overrides are available in recent releases of ParaView and can be enabled during building. If enabled, the overrides can also be turned on or off at run time using the ParaView settings (Figure 4).

5.2 VisIt

VisIt is a scientific visualization and analysis tool that operates on mesh-based field data. Its functionality is grouped into four major categories: plots, operators,



ORNL crusher (csrc://localhost:56999) (25min left
Wavelet1
Contour1
Threshold1

Properties | Information

Apply | Reset | Delete | ?

Search ... (use Esc to clear text)

Properties (T

Scalars: RTData
Lower Threshold: 33.9292
Upper Threshold: 177.806
Threshold Method: Between
[✓] All Scalars
[ ] Use Continuous Cell Range
[ ] Invert

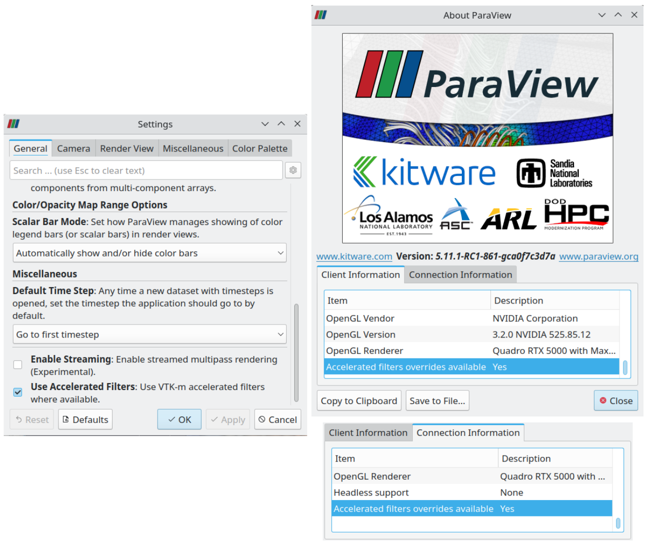
Display

View (Render

[ ] Axes Grid Edit

```
====================== ROCm System Management Interface ======================
============================== Concise Info ==================================
GPU  Temp    AvgPwr  SCLK     MCLK     Fan  Perf  PwrCap  VRAM%  GPU%
0    42.0c  263.0W  1700Mhz  1600Mhz  0%   auto  560.0W   92%   100%
1    42.0c  N/A     800Mhz   1600Mhz  0%   auto  0.0W      0%   0%
2    37.0c  90.0W   800Mhz   1600Mhz  0%   auto  560.0W    0%   0%
3    45.0c  N/A     800Mhz   1600Mhz  0%   auto  0.0W      0%   0%
4    39.0c  96.0W   800Mhz   1600Mhz  0%   auto  560.0W    0%   0%
5    38.0c  N/A     800Mhz   1600Mhz  0%   auto  0.0W      0%   0%
6    41.0c  88.0W   800Mhz   1600Mhz  0%   auto  560.0W    0%   0%
7    39.0c  N/A     800Mhz   1600Mhz  0%   auto  0.0W      0%   0%
==============================================================================
============================ End of ROCm SMI Log =============================
```

Figure 3. ParaView with integrated VTK-m accelerated filters running on Crusher, an early access test bed for the Frontier system. VTK-m is using the Kokkos device adapter. The output of the `rocm-smi` command is being used to verify and monitor GPU usage by the filters.



Search ... (use Esc to clear text)

components from multi-component arrays.

Color/Opacity Map Range Options

Scalar Bar Mode: Set how ParaView manages showing of color legend bars (or scalar bars) in render views.

Automatically show and/or hide color bars

Miscellaneous

Default Time Step: Any time a new dataset with timesteps is opened, set the timestep the application should go to by default.

Go to first timestep

[ ] **Enable Streaming**: Enable streamed multipass rendering (Experimental).

[✓] **Use Accelerated Filters**: Use VTK-m accelerated filters where available.

Reset | Defaults | OK | Apply | Cancel

www.kitware.com **Version:** ***5.11.1-RC1-861-gca0f7c3d7a*** www.paraview.org

Item	Description
OpenGL Vendor	NVIDIA Corporation
OpenGL Version	3.2.0 NVIDIA 525.85.12
OpenGL Renderer	Quadro RTX 5000 with Max...
Accelerated filters overrides available	Yes

Item	Description
OpenGL Renderer	Quadro RTX 5000 with ...
Headless support	None
Accelerated filters overrides available	Yes

Figure 4. Left: The `Use Accelerated Filters` ParaView setting can be used to activate or deactivate accelerated filter overrides at run time. This setting is shown regardless of whether or not ParaView was built with the override support. Right: The availability of the accelerated filter overrides on clients and servers can be found in the `About ParaView` dialog box.

expressions, and queries. All four of these capabilities are built on a filter infrastructure that operates on mesh-based fields. Plots are somewhat special in that they consist of a rendering capability that may include some built-in filter operations. To date, the VTK-m integration has consisted of modifying the filter infrastructure to use VTK-m filters when comparable VTK-m functionality exists.

Previously, VisIt's filters used VTK and VTK datasets. The filters were enhanced to support using both VTK and VTK-m. When VTK-m is enabled in VisIt and the filter supports VTK-m, the filter will use VTK-m. The internal dataset representation was also modified to support providing either a VTK dataset or a VTK-m dataset. When the filter is set to use VTK, it will request the data as a VTK dataset or convert the dataset to a VTK dataset if it is stored as VTK-m. Conversely, when the filter is set to use VTK-m, it will request the data as a VTK-m dataset and convert it if necessary. It will use zero-copy conversions whenever possible.

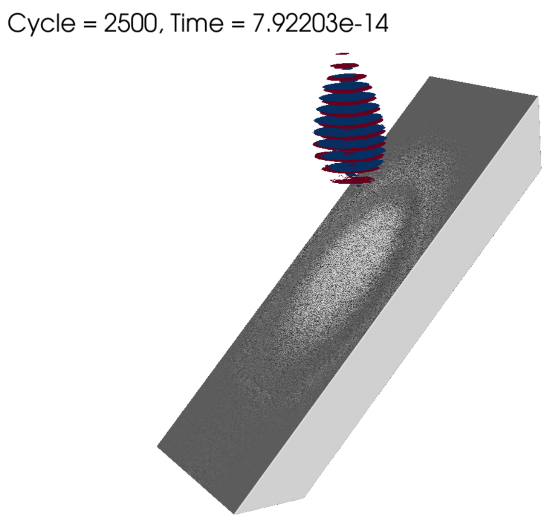
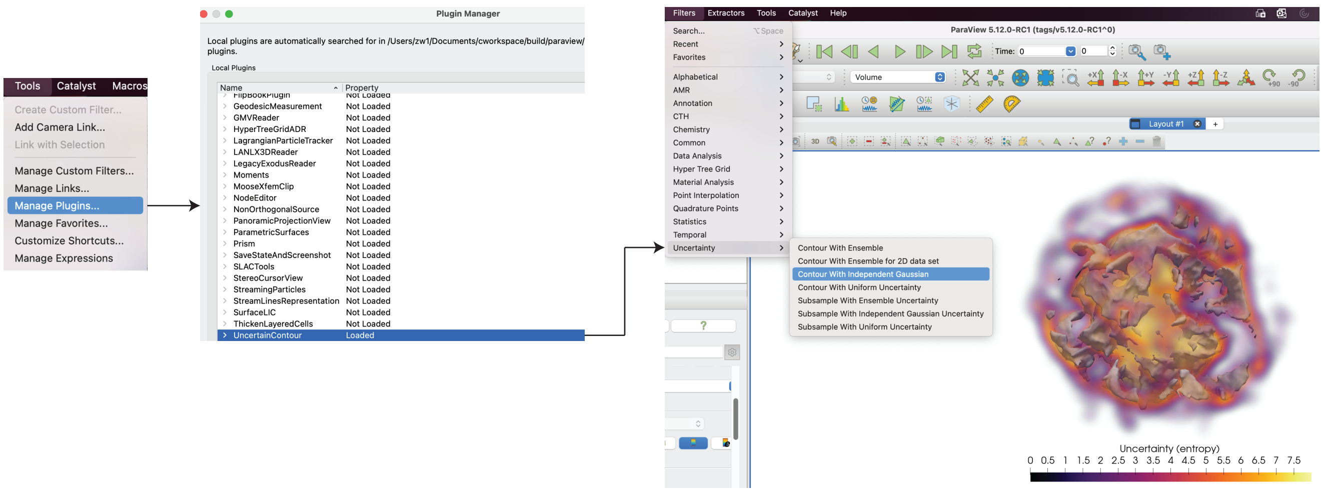


Figure 5. Visualization from a 70-billion cell WarpX simulation and Gordon Bell submission (Fedeli et al. 2022) visualized with 2,048 GCDs on Frontier using VisIt.

Figure 5 depicts a WarpX simulation visualized with VisIt and VTK-m. A laser from a laser wake-field electron accelerator is impacting a solid gas target. The red and blue surfaces represent the laser field and the gray surface represents the target. This data is from a simulation that was part of a series of simulations to help remove a major limitation of compact laser-based electron accelerators, which are promising candidates for next generation high-energy physics experiments and ultra-high dose rate FLASH radiotherapy (Fedeli et al. 2022). The image in Figure 5 was generated by VisIt running on Frontier and using 2,048 GCDs across 256 nodes. The surfaces were generated by using the VTK-m contour filter and were rendered in parallel by using Mesa 3D. VTK-m is using the Kokkos device adapter for AMD GPUs.

5.3 *Ascent*

Ascent is a lightweight, in-situ visualization and analysis library designed for running multiphysics simulations on HPC systems. As an in-situ library as opposed to a post-hoc visualization tool, Ascent shares execution resources with the simulation and can process the data as it is generated, thereby reducing I/O costs, although it must pause the simulation to do so. To minimize the encumbrance on the simulation and execution resources, Ascent's lightweight



Plugin Manager

Local plugins are automatically searched for in /Users/zw1/Documents/cworkspace/build/paraview/plugins.

Local Plugins

Name	Property
FlipBookPlugin	Not Loaded
GeodesicMeasurement	Not Loaded
GMVReader	Not Loaded
HyperTreeGridADR	Not Loaded
LagrangianParticleTracker	Not Loaded
LANLX3DReader	Not Loaded
LegacyExodusReader	Not Loaded
Moments	Not Loaded
MooseXfemClip	Not Loaded
NodeEditor	Not Loaded
NonOrthogonalSource	Not Loaded
PanoramicProjectionView	Not Loaded
ParametricSurfaces	Not Loaded
Prism	Not Loaded
SaveStateAndScreenshot	Not Loaded
SLACTools	Not Loaded
StereoCursorView	Not Loaded
StreamingParticles	Not Loaded
StreamLinesRepresentation	Not Loaded
SurfaceLIC	Not Loaded
ThickenLayeredCells	Not Loaded
UncertainContour	Loaded

Figure 6. Integration of the VTK-m isosurface uncertainty filter in ParaView using the plug-in approach for visualization of large-scale supernova simulations (Sandoval et al. 2021).

design ensures a smaller memory requirement. It is written using efficient distributed-memory and many-core libraries to guarantee performance and scalability on current and next-generation HPC platforms. Ascent has three main use cases: creating pictures, transforming data, and capturing data. Ascent is easy to use with only five API calls; supported in C, C++, Python, and Fortran; and provides an infrastructure to integrate custom analysis. The data interface between simulation code and Ascent is managed through the Conduit API (Harrison et al. 2022), which provides a simplified interface for passing data and describing structure.

Although optional, VTK-m is a dependency for Ascent because it is currently the only option for rendering low-order mesh data and provides filters for transforming and/or analyzing the simulation data (e.g., slice, histogram, contour). Ascent also leverages VTK-m's zero-copy capabilities as well as its ability to pass device-pointers; these features allow the simulation data to remain on the device and be passed directly to Ascent and then on to VTK-m without being transferred to host memory. VTK-m has been integrated into Ascent via a (previously external) library called VTK-h (VTK hybrid) that combines VTK-m's shared-memory, high-performance filters with MPI's efficient distributed-memory coordination.

Figures 7 and 8 show in-situ renderings of the WarpX simulation (see Section 6.1 later in this paper) generated by Ascent and executed on the OLCF's exascale supercomputer, Frontier. The simulation was executed at two resolutions: 578.8 million cells across 552 GCDs on 69 nodes and 4.63 billion cells across 4,416 GCDs on 552 nodes. Ascent used VTK-m filters to upscale the data along multiple axes, generate isosurfaces, and clip several fields before using VTK-m's raytracer and volume renderer to generate the final images. To guarantee performance, VTK-m uses the Kokkos device adapter for AMD GPUs.

5.4 *Alternative delivery mechanisms*

Integrating a new feature that was implemented with VTK-m filters into visualization software such as ParaView or VisIt can be a lengthy process. For example, making a VTK-m filter available in ParaView requires multiple steps, including implementing a VTK filter that wraps the VTK-m filter, completing the arduous process of contributing the change to the VTK project, and then repeating similar steps in ParaView itself. Such time-consuming software integration can hinder the availability of VTK-m filters inside visualization tools and limit the opportunities for VTK-m filters to increase the pace of scientific discovery.

Our ultimate goal is to make VTK-m filters practical for real use and to put tools in the hands of end users in a timely manner. An alternative approach to a full integration through the visualization software stack is to provide this functionality through a plug-in that is supported by tools such as ParaView and VisIt. For the plug-in approach, the VTK-m filter is still wrapped inside a VTK filter, but the time needed for the software integration and testing in VTK and ParaView can be bypassed.

Figure 6 illustrates the VTK-m isosurface uncertainty filter (Wang et al. 2023; Athawale et al. 2021) made available in ParaView via the plug-in method. The isosurface uncertainty filter is one of the major successes of the VTK-m library because it is the first production-level uncertainty visualization filter deployed for efficient large-data analysis. Using the plug-in method depicted in Figure 6, VTK-m filters can be easily combined with existing filters in ParaView for improved data comprehension.

6 Interfacing with applications

The ultimate goal of the VTK-m work for the ECP was to provide scientists with the tools needed to understand large amounts of data and make scientific discoveries. The previous sections of this paper describe the efforts for making these tools available. This section provides some examples of applying VTK-m and its companion enabling technologies to real-world science problems, which often happens through the visualization tools described in the previous section.

6.1 *Laser wakefield acceleration*

WarpX is a particle-in-cell simulation code and was awarded the 2022 ACM Gordon Bell Prize (Fedeli et al. 2022). As part of the ECP, WarpX was developed as a new application

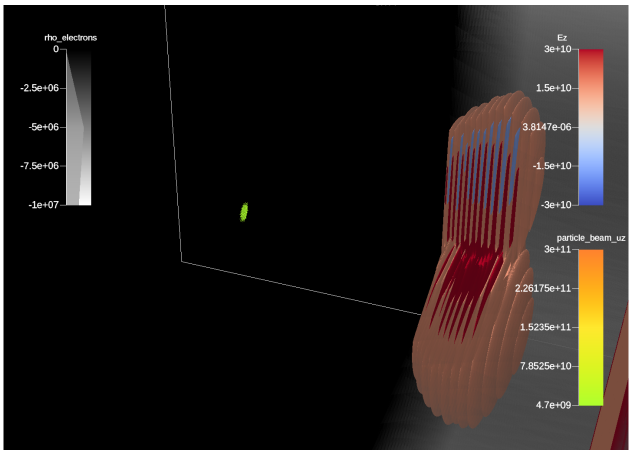


Figure 7. WarpX in-situ visualization of a laser-wakefield accelerator on 4,416 GCDs across 552 nodes of Frontier using Ascent and VTK-m. The image depicts an early time step of the simulation at high resolution.

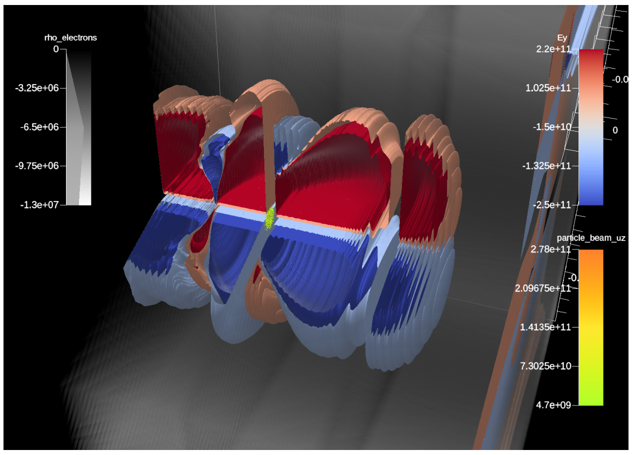


Figure 8. WarpX in-situ visualization of a laser-wakefield accelerator on 552 GCDs across 69 nodes of Frontier using Ascent and VTK-m. The image depicts a later time step of the simulation at low resolution.

to succeed its predecessor, Warp (Vay et al. 2012), with the goal of studying advanced particle acceleration in laser-driven plasma wakefields in an effort to advance future high-energy physics colliders (Albert et al. 2021). Beyond that, WarpX is used to describe kinetic physics in particle accelerators, laser-plasmas, fusion devices, inertial confinement fusion, and astrophysical plasmas and to model microelectronics (Yao et al. 2022).

Figures 7 and 8 are in-situ renderings of a *staged* laser wakefield accelerator in a boosted reference frame (Vay et al. 2011). An electron beam (orange-green) is accelerated to the right through multiple stages to high energies. In the plasma stages (gray), the strong traversal focusing fields are shown in red-blue.

In the staging approach, a particle beam is accelerated through multiple plasma elements. In each stage, an ultra-intense laser pulse excites a plasma wakefield. This depletes the laser pulse's energy and generates very strong electric fields in the plasma wake, which can be used to accelerate an injected electron beam. The acceleration itself can be 3–4 orders of magnitude more compact than relying on state-of-the-art radio-frequency accelerator elements. Besides increased beam energy, physicists study how to preserve beam properties essential for transport, focusing (e.g., emittance), and applications (e.g., charge and current).

WarpX features advanced techniques such as GPU-acceleration for three vendors, mesh-refinement capabilities, dynamic load balancing, and unique advanced numerical solvers. WarpX relies on multi-level parallelization: coarse parallelization uses block-structured domain decomposition with MPI using the AMReX library (Zhang et al. 2019), and compute acceleration leverages CUDA/HIP/SYCL or OpenMP so that the simulations can scale on large, massively parallel HPC systems.

If WarpX relied on only traditional post-processing workflows for the visualization of the dynamics of exascale simulations, the resulting multi-petabyte scale output per simulation would severely limit the available snapshots and/or level of detail to visualize. Addressing this need, WarpX interfaces with Ascent for in-situ visualization. For this, utility routines for specialized WarpX diagnostics for application-specific descriptions were implemented in AMReX. WarpX performs data preparation steps for diagnostics in situ, shares the respective AMReX memory buffers with zero-copy APIs through Conduit with Ascent, and renders with VTK-m in the same domain-decomposition and on the same compute device as the simulation itself.

Realistic visualization of particle trajectories (advection) in a plasma or particle accelerator requires high temporal fidelity in traditional workflows, and this fidelity can create significant data overhead. With VTK-m, an opportunity to significantly reduce data input for such workflows was identified by using a physics-motivated advection algorithm and the slowly changing nature of fields in a wakefield accelerator.

Plasma particles such as electrons and ions are inert and can be relativistic, which effectively changes their mass as they move. Traditional advection algorithms only used local properties of fields without accounting for a history or inert nature of a streamline. As in a particle-in-cell algorithm, the realistic track of a charged plasma particle can be integrated following the Lorentz-Force, which interpolates six local field components ($E_{x,y,z}$, $B_{x,y,z}$) and advances the particles' momentum (inertia) and position with an explicit iteration scheme (Boris 1970). The updated momentum is tracked over the path of a streamline to account for the evolving particle.

With this advection algorithm integrated in VTK-m, a snapshot of a simulation can be used to project the particles' physical position forward (and backward) for a meaningful time under the realistic assumption that fields are quasi-static (i.e., do not change much in time) besides translation along an axis. Figure 9 shows such particle trajectories of an off-axis injected electron beam in a wakefield calculated from a single snapshot and reproducing physical betatron oscillation.

6.2 Tokamak fusion reactor

Fusion energy research focuses on understanding the science needed to develop energy sources based on the controlled fusion of light atomic nuclei. One strategy to achieve

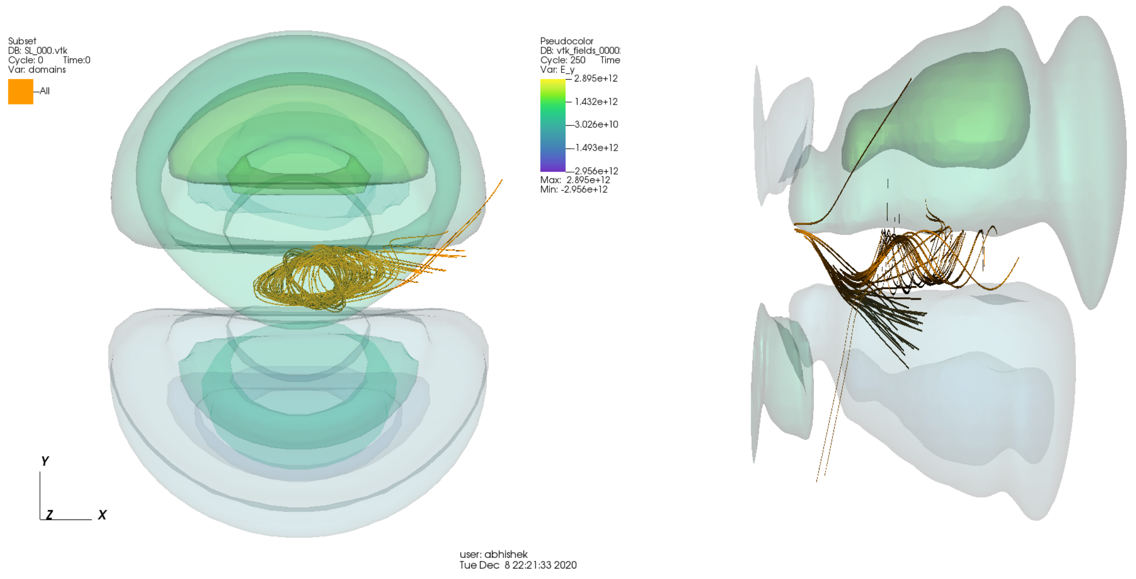


Figure 9. Side and front views of a laser wakefield with an injected electron bunch. Particles are advected from *a single snapshot* of the simulation in VTK-m.

fusion involves a device called a tokamak, which uses magnetic fields to confine a hot plasma in the shape of a torus. Significant efforts are currently underway to prepare for ITER, a large experimental fusion reactor under construction in France. The Whole Device Model Application (WDMApp) is a project in the ECP that aims to develop a high-fidelity model of magnetically confined fusion plasma in tokamaks. WDMApp is critical in the efforts to plan experiments on ITER and optimize the design of future next-step fusion facilities. These devices will operate in physics regimes not achieved by any current or past experiments, thereby making advanced and predictive numerical simulation the best tool for the task.

The behavior and evolution of the magnetic field in a tokamak is complex, and its control is critical for performance. For these reasons, analysis tools used for understanding the dynamic nature of the magnetic field are critical. The complexity of the 3D magnetic field lines makes analysis and visualization difficult. Because the field lines are periodic, this complexity can be reduced by using a Poincaré magnetic field-line puncture map (Sanderson et al. 2010). The Poincaré map is the intersection of a field line with a lower-dimensional subspace (called the Poincaré section). In our case, the Poincaré section is a 2D plane that is perpendicular to the axis of the tokamak. Given a set of magnetic field lines, the Poincaré map (i.e., the intersection of the magnetic field lines with the plane) provides a concise representation of the magnetic field and is easier to understand and analyze. Figure 10 shows some examples of Poincaré plots generated by VTK-m.

In practice, the Poincaré map is generated by creating many field lines and plotting each intersection, or puncture, with the plane. After a sufficient number of punctures has been collected, patterns in the map characterize the features in the magnetic field. The field lines are computed by modeling massless particles that follow magnetic field lines. These massless particles can follow magnetic field lines by advecting them in the direction of the magnetic field. The intersections generated from a single particle characterize the features of the magnetic surface at that position. The particles are advected using a differential equation solver such as the fourth order Runge-Kutta scheme.

Proper characterization of the magnetic field requires a large number of initial positions (typically tens of thousands), each of which typically results in between 1,000 and 3,000 intersections, with each intersection following a field line all the way through the tokamak's torus. Because of these numerous features, the computation of a Poincaré map can be very expensive. The WDMApp team has a Poincaré map code that runs on CPUs and takes several hours for the largest analysis run. The high cost of the analysis is due to two main factors: (1) the many particles and intersections required and (2) the complexity of the magnetic field calculation. In many applications of particle advection, the vector field is calculated at the nodes of each cell in the mesh. Linear interpolation within the cell is used to evaluate the magnetic field for the particle being advected. Because of the large number of advection steps required for each particle in the Poincaré map, small errors can rapidly accumulate. These errors are compounded because the evaluation of the magnetic field requires a complex set of calculations, and these calculations require high-order interpolation of several quantities.

Using VTK-m can significantly accelerate the computation of Poincaré maps by leveraging the parallelism of GPUs. Because the trajectory of each particle is completely independent, the task can be parallelized over each particle. Using this approach, a Poincaré map can be computed in under 3 minutes. The wall-clock time for a typical WDMApp simulation step is between 1.5 and 2 minutes, which means that Poincaré maps can be computed in situ in nearly real time. When WDMApp is run, the EFFIS workflow control system (Suchyta et al. 2022b) allocates an additional 1–2 nodes for the Poincaré map analysis. As a simulation step completes, EFFIS launches a Poincaré analysis task on GPUs in the node in a round-robin fashion. This allows

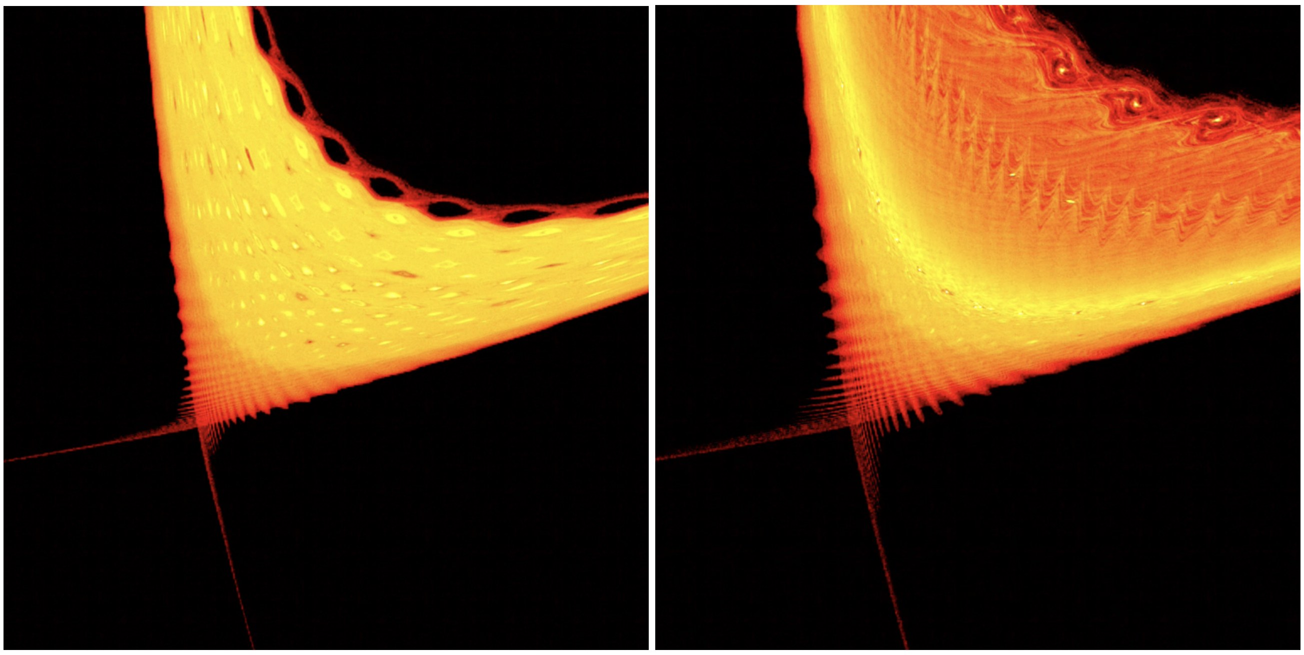


Figure 10. Poincaré maps from two different time steps from a simulation run at the Oak Ridge Leadership Computing Facility generated by VTK-m. The particles were placed near the edge of the tokamak where the plasma becomes very turbulent. The Poincaré map shows magnetic features in the plasma as the simulation progresses. Of particular interest are the long fingers that appear in the lower portion of the image. The evolving shape of these fingers over time provides valuable insight into the behavior of the magnetic field where the turbulence is extremely high.

asynchronous analysis to be performed on the additional nodes while the simulation is running. Because EFFIS was already being used as a code-integrating technology, it was straightforward to integrate VTK-m directly into this system. Figure 10 shows Poincaré maps from two different time steps of a simulation run at the OLCF. Real-time generation of Poincaré maps provides the WDMApp team with unprecedented capability for analysis of magnetic fields in fusion simulations.

6.3 Impact beyond ECP

In addition to the applications mentioned above, VTK-m still serves as the key infrastructure for accelerating data analysis and visualization in various scientific applications through in-situ visualization. This subsection lists several examples and illustrates how VTK-m is integrated into in-situ scientific workflows outside of the ECP.

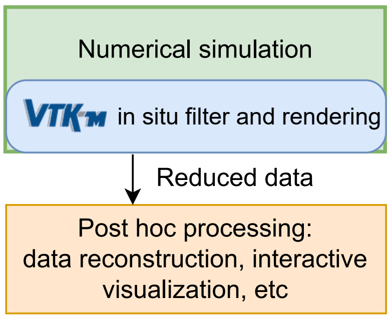

Numerical simulation
VTK-m in situ filter and rendering
Reduced data
Post hoc processing: data reconstruction, interactive visualization, etc

Figure 11. The in-situ reduction + post-hoc paradigm based on VTK-m.

Before being selected as a project in the ECP, VTK-m was a core component in the Visualization for the Extreme-Scale Scientific Computation Ecosystem (XVis) (Moreland et al. 2019) project. XVis focused on multiple ways to integrate in-situ visualization with the simulation, extract key information, and decrease the data size for post-hoc processing. The *in-situ reduction + post hoc* paradigm (Figure 11) is adopted in multiple scientific domains within and outside of the ECP, such as probability distribution function extraction of fields in combustion simulation (Ye et al. 2016) and binning mechanisms to reduce the data size of fusion simulation (Kress et al. 2018). We use two recent efforts as examples to illustrate how VTK-m facilitates scientific workflows beyond the ECP.

Nyx is a cosmological simulation code that aims to solve compressible hydrodynamics with N-body treatment of dark matter. Each simulation run may contain hundreds of time steps with multiple sets of simulation input parameters. The raw data size is usually hundreds of terabytes to several petabytes—a size that presents challenges when post-processing the data. VTK-m is used for in-situ analysis to extract the statistical properties of the down-sampled data to significantly reduce the size of raw data. The associated statistics model can be used to construct the data based on prior knowledge in post-processing with low data reconstruction error (Wang et al. 2019).

Eddy detection and tracking play key roles in analyzing the data generated by ocean simulations. Understanding the characteristics of eddies can help scientists explain the regional air-sea interactions. The VTK-m streamline filter can be used as an in-situ analysis to generate streamline data used for interactive post-hoc analysis (Han et al. 2022). With the help of VTK-m, the associated eddy analysis workflow can improve the interaction speed, reduce data storage, and meet the needs of real-time visual analysis interaction.

7 Conclusion

Although their use in HPC was novel a decade ago, GPUs and accelerators have become a staple of HPC hardware and are now used in over $1/3$ of the top 500 fastest supercomputers in the world.[*] Efficient use of these accelerators is critical for good performance on HPC systems, and VTK-m provides this functionality for scientific visualization.

Moreover, the ECP has been instrumental in making VTK-m available on today's GPU-reliant exascale machines. Many challenges were overcome with porting and integrating the software, and we are proud to have helped application scientists better analyze, explore, and understand their data. We anticipate the use of VTK-m in high-performance visualization software to be even more critical as HPC continues to evolve.

8 Acknowledgments

The work in this paper builds upon a foundation laid by many previous members of the VTK-m team as well as early adopters. We thank and acknowledge Cyrus Harrison, Nrushad Joshi, Brad King, Matthew Larsen, Manish Mathai, Robert Maynard, Jeremy Meredith, Chris Sewell, and Allison Vacanti. The authors also acknowledge the WarpX and AMReX open-source community members and WarpX PI Jean-Luc Vay for their roles in integrating VTK-m and WarpX technologies.

This research was supported by the ECP (17-SC-20-SC), a collaborative effort of the DOE's Office of Science and the National Nuclear Security Administration. This research used resources of the OLCF at ORNL, which is supported by the DOE's Office of Science under contract number DE-AC05-00OR22725. This research also used resources of the ALCF, a DOE Office of Science user facility at Argonne National Laboratory and is based on research supported by the DOE Office of Science-Advanced Scientific Computing Research Program under contract number DE-AC02-06CH11357. This work was conducted on a preproduction supercomputer with early versions of the Aurora software development kit.

References

Albert F, Couprie ME, Debus A, Downer MC, Faure J, Flacco A, Gizzi LA, Grismayer T, Huebl A, Joshi C, Labat M, Leemans WP, Maier AR, Mangles SPD, Mason P, Mathieu F, Muggli P, Nishiuchi M, Osterhoff J, Rajeev PP, Schramm U, Schreiber J, Thomas AGR, Vay JL, Vranic M and Zeil K (2021) 2020 roadmap on plasma accelerators. *New Journal of Physics* 23(3): 031101. DOI:10.1088/1367-2630/abcc62.

Atchley S et al. (2023) Frontier: Exploring exascale. In: *SC '23: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 52. DOI:10.1145/3581784.3607089.

Athawale TM, Sane S and Johnson CR (2021) Uncertainty visualization of the marching squares and marching cubes topology cases. In: *IEEE Visualization Conference (VIS)*. pp. 106–110. DOI:10.1109/VIS49827.2021.9623267.

Blelloch GE (1990) *Vector Models for Data-Parallel Computing*. MIT Press. ISBN 0-262-02313-X.

Boris JP (1970) Relativistic Plasma Simulation - Optimization of a Hybrid Code. In: *Numer. Simul. Plasmas*. Washington, D.C.: Naval Research Laboratory, pp. 3–67.

Carr HA, Rübel O, Weber GH and Ahrens JP (2021) Optimization and augmentation for data parallel contour trees. *IEEE Transactions on Visualization and Computer Graphics* 28(10): 3471–3485. DOI:10.1109/TVCG.2021.3064385.

Coplien JO (1995) Curiously recurring template patterns. *C++ Report* 7(2): 24–27.

Edwards HC, Trott CR and Sunderland D (2014) Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *Journal of Parallel and Distributed Computing* 74(12): 3202–3216. DOI:https://doi.org/10.1016/j.jpdc.2014.07.003.

Fedeli L, Huebl A, Boillod-Cerneux F, Clark T, Gott K, Hillairet C, Jaure S, Leblanc A, Lehe R, Myers A, Piechurski C, Sato M, Zaim N, Zhang W, Vay JL and Vincenti H (2022) Pushing the Frontier in the Design of Laser-Based Electron Accelerators with Groundbreaking Mesh-Refined Particle-In-Cell Simulations on Exascale-Class Supercomputers. In: *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. DOI:10.1109/SC41404.2022.00008. *Winning Paper, 2022 ACM Gordon Bell Prize*.

Han X, Yu X, Li G, Liu J, Zhao Y and Shan G (2022) Narrative in situ visual analysis for large-scale ocean eddy evolution. *IEEE Computer Graphics and Applications* 42(3): 65–73. DOI:10.1109/MCG.2022.3167044.

Harrison C, Larsen M, Ryujin BS, Kunen A, Capps A and Privitera J (2022) Conduit: A successful strategy for describing and sharing data in situ. In: *EEE/ACM International Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV)*. DOI:10.1109/ISAV56555.2022.00006.

Jeffers J, Reinders J and Sodani A (2016) *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition*. 2nd edition. Morgan Kaufmann. ISBN 978-0128091944.

Kress J, Choi J, Klasky S, Churchill M, Childs H and Pugmire D (2018) Binning based data reduction for vector field data of a particle-in-cell fusion simulation. In: *High Performance Computing*, *Lecture Notes in Computer Science*, volume 11203. Springer, pp. 215–229. DOI:10.1007/978-3-030-02465-9_15.

Larsen M, Labasan S, Navrátil P, Meredith J and Childs H (2015a) Volume rendering via data-parallel primitives. In: *Eurographics Symposium on Parallel Graphics and Visualization*. DOI:10.2312/pgv.20151155.

Larsen M, Meredith JS, Navratil PA and Childs H (2015b) Ray tracing within a data parallel framework. In: *IEEE Pacific Visualization Symposium (PacificVis)*. pp. 279–286. DOI:10.1109/PACIFICVIS.2015.7156388.

Lessley B, Binyahib R, Maynard R and Childs H (2016) External facelist calculation with data-parallel primitives.

[*] https://top500.org/lists/top500/2023/11/

In: *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. DOI:10.2312/pgv.20161178.

Lessley B, Moreland K, Larsen M and Childs H (2017) Techniques for data-parallel searching for duplicate elements. In: *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*. DOI:10.1109/LDAV.2017.8231845.

Li S, Marsaglia N, Chen V, Sewell CM, Clyne JP and Childs H (2017) Achieving portable performance for wavelet compression using data parallel primitives. In: *Proceedings of EuroGraphics Symposium on Parallel Graphics and Visualization (EGPGV)*. pp. 73–81. DOI:10.2312/pgv.20171095.

Lo LT, Sewell C and Ahrens J (2012) PISTON: A portable cross-platform framework for data-parallel visualization operators. In: *Eurographics Symposium on Parallel Graphics and Visualization*. DOI:10.2312/EGPGV/EGPGV12/011-020.

Maynard R, Moreland K, Ayachit U, Geveci B and Ma KL (2013) Optimizing threshold for extreme scale analysis. In: *Visualization and Data Analysis 2013, Proceedings of SPIE-IS&T Electronic Imaging*. DOI:10.1117/12.2007320.

Meredith JS, Ahern S, Pugmire D and Sisneros R (2012) EAVL: The extreme-scale analysis and visualization library. In: *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. pp. 21–30. DOI:10.2312/EGPGV/EGPGV12/021-030.

Meyers S (2005) *Effective C++: 55 Specific Ways to Improve Your Programs and Designs*. 3rd edition. Addison-Wesley Professional. ISBN 978-0-13-270206-5.

Moreland K, Maynard R, Pugmire D, Yenpure A, Vacanti A, Larsen M and Childs H (2021) Minimizing development costs for efficient many-core visualization using MCD^3. *Parallel Computing* 108(102834). DOI:10.1016/j.parco.2021.102834.

Moreland K, Pugmire D, Rogers D, Childs H, Ma KL and Geveci B (2019) XVis: Visualization for the extreme-scale scientific computation ecosystem, final report. Technical Report SAND 2019-9297, Sandia National Laboratories. DOI:10.2172/1762947.

Moreland K, Sewell C, Usher W, Lo LT, Meredith J, Pugmire D, Kress J, Schroots H, Ma KL, Childs H, Larsen M, Chen CM, Maynard R and Geveci B (2016) VTK-m: Accelerating the visualization toolkit for massively threaded architectures. *IEEE Computer Graphics and Applications* 36(3): 48–58. DOI:10.1109/MCG.2016.48.

Morozov D and Peterka T (2016) Block-parallel data analysis with DIY2. In: *Proceedings of the IEEE Large Data Analysis and Visualization Symposium (LDAV)*. DOI:10.1109/LDAV.2016.7874307.

Peterka T, Ross R, Kendall W, Gyulassy A, Pascucci V, Shen HW, Lee TY and Chaudhuri A (2011) Scalable parallel building blocks for custom data analysis. In: *Proceedings of Large Data Analysis and Visualization Symposium LDAV'11*. pp. 105–112. DOI:10.1109/LDAV.2011.6092324.

Pugmire D, Yenpure A, Kim M, Kress J, Maynard R, Childs H and Hentschel B (2018) Performance-portable particle advection with VTK-m. In: *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. pp. 45–55. DOI:10.2312/pgv.20181094.

Sanderson A, Chen G, Tricoche X, Pugmire D, Kruger S and Breslau J (2010) Analysis of recurrent patterns in toroidal magnetic fields. *IEEE Transactions on Visualization and Computer Graphics* 16(6): 1431–1440. DOI:10.1109/TVCG.2010.133.

Sandoval MA, Hix WR, Messer OEB, Lentz EJ and Harris JA (2021) Three-dimensional core-collapse supernova simulations with 160 isotopic species evolved to shock breakout. *The Astrophysical Journal* 921(2): 113. DOI:10.3847/1538-4357/ac1d49.

Sane S, Johnson CR and Childs H (2021a) Investigating in situ reduction via lagrangian representations for cosmology and seismology applications. In: *International Conference on Computational Science*. Springer, pp. 436–450. DOI:10.1007/978-3-030-77961-0_36.

Sane S, Yenpure A, Bujack R, Larsen M, Moreland K, Garth C, Johnson CR and Childs H (2021b) Scalable in situ computation of lagrangian representations via local flow maps. In: *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. DOI:10.2312/pgv.20211040.

Suchyta E, Choi JY, Ku SH, Pugmire D, Gainaru A, Huck K, Kube R, Scheinberg A, Suter F, Chang C et al. (2022a) Hybrid analysis of fusion data for online understanding of complex science on extreme scale computers. In: *IEEE International Conference on Cluster Computing (CLUSTER)*. pp. 218–229. DOI:10.1109/CLUSTER51413.2022.00035.

Suchyta E, Klasky S, Podhorszki N, Wolf M, Adesoji A, Chang C, Choi J, Davis PE, Dominski J, Ethier S, Foster I, Germaschewski K, Geveci B, Harris C, Huck KA, Liu Q, Logan J, Mehta K, Merlo G, Moore SV, Munson T, Parashar M, Pugmire D, Shephard MS, Smith CW, Subedi P, Wan L, Wang R and Zhang S (2022b) The exascale framework for high fidelity coupled simulations (EFFIS): Enabling whole device modeling in fusion science. *The International Journal of High Performance Computing Applications* 36(1): 106–128. DOI:10.1177/10943420211019119.

Trott CR, Lebrun-Grandié D, Arndt D, Ciesko J, Dang V, Ellingwood N, Gayatri R, Harvey E, Hollman DS, Ibanez D, Liber N, Madsen J, Miles J, Poliakoff D, Powell A, Rajamanickam S, Simberg M, Sunderland D, Turcksin B and Wilke J (2022) Kokkos 3: Programming model extensions for the exascale era. *IEEE Transactions on Parallel and Distributed Systems* 33(4): 805–817. DOI:10.1109/TPDS.2021.3097283.

Vay JL, Geddes CGR, Esarey E, Schroeder CB, Leemans WP, Cormier-Michel E and Grote DP (2011) Modeling of 10 GeV-1 TeV laser-plasma accelerators using Lorentz boosted simulations. *Physics of Plasmas* 18(12): 123103. DOI:10.1063/1.3663841.

Vay JL, Grote DP, Cohen RH and Friedman A (2012) Novel methods in the particle-in-cell accelerator code-framework warp. *Computational Science & Discovery* 5(1): 014019. DOI:10.1088/1749-4699/5/1/014019.

Wang KC, Xu J, Woodring J and Shen HW (2019) Statistical super resolution for data analysis and visualization of large scale cosmological simulations. In: *IEEE Pacific Visualization Symposium (PacificVis)*. pp. 303–312. DOI:10.1109/PacificVis.2019.00043.

Wang Z, Athawale TM, Moreland K, Chen J, Johnson CR and Pugmire D (2023) FunMC^2: A filter for uncertainty visualization of marching cubes on multi-core devices. In: *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. DOI:10.2312/pgv.20231081.

Yao Z, Jambunathan R, Zeng Y and Nonaka A (2022) A massively parallel time-domain coupled electrodynamics–micromagnetics solver. *The International Journal of High Performance Computing Applications* 36(2): 167–181. DOI: 10.1177/10943420211057906.

Ye YC, Neuroth T, Sauer F, Ma KL, Borghesi G, Konduri A, Kolla H and Chen J (2016) In situ generated probability distribution functions for interactive post hoc visualization and analysis. In: *2016 IEEE 6th Symposium on Large Data Analysis and Visualization (LDAV)*. pp. 65–74. DOI:10.1109/LDAV.2016.7874311.

Yenpure A, Childs H and Moreland K (2019) Efficient point merging using data parallel techniques. In: *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. DOI:10.2312/pgv.20191112.

Zhang W, Almgren A, Beckner V, Bell J, Blaschke J, Chan C, Day M, Friesen B, Gott K, Graves D, Katz MP, Myers A, Nguyen T, Nonaka A, Rosso M, Williams S and Zingale M (2019) AMReX: a framework for block-structured adaptive mesh refinement. *Journal of Open Source Software* 4(37): 1370. DOI:10.21105/joss.01370.