Kitware

28 Corporate Drive
Clifton Park, NY 12065
518.371.3971

XVIS: Visualization for the Extreme-Scale Scientific-Computation Ecosystem

Final Scientific/Technical Report

Prepared for: The U.S. Department of Energy, Office of Science

Grant Number: DE-SC0012386

Phase II Period of Performance: September 1, 2014 – August 31, 2017

Kitware, Inc.
28 Corporate Drive
Clifton Park, NY 12065
DUNS: 01-092-6207

Berk Geveci, Principal Investigator
(518) 881-4907
berk.geveci@kitware.com

This material is based upon work supported by the U.S. Department of Energy,

Office of Science under Award Number DE-SC0012386

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Abstract

The XVis project brings together the key elements of research to enable scientific discovery at extreme scale. Scientific computing will no longer be purely about how fast computations can be performed. Energy constraints, processor changes, and I/O limitations necessitate significant changes in both the software applications used in scientific computation and the ways in which scientists use them. Components for modeling, simulation, analysis, and visualization must work together in a computational ecosystem, rather than working independently as they have in the past. The XVis project brought together collaborators from predominant DOE projects for visualization on accelerators and combining their respective features into a new visualization toolkit called VTK-m.

The XVis team has been united in developing the VTK-m toolkit from starting together on a proof of concept, and building from that foundation to develop a comprehensive C++ library that contains all the necessary components to for designing and implementing visualization algorithms. To allow the XVis team to undertake such a large task we developed a robust infrastructure including revision control, code reviews, nightly, and per feature testing that facilitated developing multiple sections of VTK-m at the same time while maintaining stability and correctness for the project. This process has been incredibly successful, as it has allowed XVis developers to develop multiple components and algorithms at the same time without serious integration issues.

The transition to exascale has shifted the scientific-computation ecosystem to become significantly more heterogeneous. This transition from a largely homogeneous system is mainly being driven by the greater variance in accelerator architecture design compared to the x86 CPU, while at the same time we are seeing significantly reduced scaling of storage systems due to physical limitations. With VTK-m addressing these challenges, we have seen the general scientific visualization community begin to adopt VTK-m as the library to use for heterogeneous computation. Since the creation of VTK-m we now have VTK, VisIt, Sensi, ParaView Catalyst, and PyFR providing some form of VTK-m integration or support. The biggest indication of VTK-m adoption as the exascale visualization solution is the award of an ECP project to update VTK software using the VTK-m framework.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, under Award Number DE-SC0012386

Progress and Accomplishments

The VTK-m project is the cornerstone to the XVis project. As such, most of the development throughout the project was devoted to designing, building, and maintaining the VTK-m toolkit. By bringing together the existing work and developers of EAVL, Piston, and Dax the XVis team was able to leverage existing techniques and knowledge to build an incredible project. Kitware as a partner in the XVis project focused primarily on developing the VTK-m toolkit and worked on in situ integration. A summary of our major accomplishments is as follows.

Initial VTK-m Design

The VTK-m toolkit is a comprehensive C++ library that contains all the necessary components to for designing and implementing visualization algorithms. It is composed of four major components: control, execution, worklets, and filters. This separation of components allows for an API that is succinct, type safe, and efficient even when executing on multiple disparate architectures.

The control section holds the users representation of the arrays, data model, cell types, and the infrastructure for scheduling worklets in parallel. The execution environment is not exposed to the user but instead limited to worklet developers when developing new visualization and analysis algorithms. This separation of the control and execution representation allows VTK-m to decompose the control side representation into types that are best suited for the specific execution environment.

This separation of control and execution is made possible by providing different implementations for the device dependent components of the control environment, which we call the device adapter. This abstraction of the device adapter has allowed VTK-m to support different architectures including GPU's using CUDA, and multi-core CPUs and Xeon PHI's using Intel Threading Building Blocks.

The device adapter is more than just a way of allowing different representations of memory. It provides the fundamental building blocks of all VTK-m algorithms by providing not only device optimized data parallel primitives such as: sort, prefix-sum, and reduce; but also the common functionality that is needed for a fully featured development environment.

This includes but limited too optimized parallel looping for one and three dimensional iteration, integer atomic operations including set, get, and compare and exchange, runtime support for the given device adapter, and the ability for each device adapter to provide custom allocators, and load and stores operations.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, under Award Number DE-SC0012386

VTK-m array design is based around the ability of having a consistent interface that provides direct access of data no matter the type. This design is made possible by using C++ templates to build a consistent interface for concrete and implicit arrays. This generic array implementation allows for not only zero-copy support of other data layouts on the control side but also allows for VTK-m to convert array layouts between the control and execution environment allowing for even better performance. VTK-m array design also includes a dynamic array wrapper that facilitates handling data whose type is not known at compile time, allowing VTK-m to handle input from disks or runtime sources.

After implementing the device adapter and array architecture, we moved onto designing the worklet component. This component provides a mechanism for building and executing worklets in parallel on any device. The design is made flexible by having the execution, data types, and the worklet type being separate components that can mix together as desired. VTK-m currently supports three worklet patterns with the possibility of adding more as needed. The worklet patterns are:

- Ability to map from input array(s) to output array(s) where the indexing for each input and output is the same
- Ability to map between two arbitrary topological elements, such as cell to point. This allows worklets to have element-wide read access from source topological types and read/write element-wide access from destination types.
- Ability to reduce by key. Operates on an array of keys and one or more associated arrays of values. This allows worklets to have access to all the values that map to a unique key and output a fixed number of values per unique key.

As part of the worklet development we added generic cell shape mechanism and basic cell type operations such as parametric coordinates, world conversions, interpolation, and derivatives.

The worklet infrastructure has allowed the VTK-m project to develop a robust collection of filters within VTK-m. Filters are specific visualization algorithms which are implemented using VTK-m arrays, data sets, worklets and are capable of running on any of device. The filters that are currently integrated are:

- Cell Average
- CleanGrid
- ExternalFaces
- Point Average
- Point Elevation
- Marching Cubes
- Statistics
- Level of Detail
- Threshold

This material is based upon work supported by the U.S. Department of Energy, Office of Science, under Award Number DE-SC0012386

- Triangulate
- Tetrahedralize

The VTK-m filter infrastructure was designed with the understanding that writing new algorithms should be easy experience for developers, with VTK-m handling the complexity for targeting multiple devices. To this end the VTK-m filter design automatically builds the algorithm for all supported device adapters and than at runtime determines which device should be used. To make the system even more robust if execution fails due to a device specific issue, it will retry the algorithm on the next best device.

The VTK-m toolkit like any form of software is an iterative process, which requires feedback on correctness and reliability. To this end we setup a robust framework for VTK-m that combines per component, integration, and performance tests to verify VTK-m reliability and correctness. Combined with VTK-m feature branch workflow it allows each feature to be tested across all major compilers and operating systems before being introduced into the primary project. This continuous integration testing has improved developer productivity by improving faster turnaround time and a higher degree of certainty on the correctness of the project. To help on-board new developers and to make sure it was easy for developers to know what was happening the project also setup a developer wiki for design and implementation reviews, a doxygen website for API documentation, a VTK-m user's guide, and a mailing list for user and developer questions and communication.

The general scientific visualization community has met the development of the VTK-m toolkit with appreciation and interest. Not only have we seen contributions to VTK-m from the general community, we have also seen VTK-m being used by other projects. VTK-m has been adopted by VTK as part of the accelerator module, is part of the VisIt project, and has been used by PyFR team for in situ visualization. The most significant development is the award of an ECP project to update VTK software using the VTK-m framework. This demonstrates a large success of XVis: the VTK-m software is being adopted as a key component for our exascale visualization software.

Array Characterization

In general scientific visualization algorithms are based around topological information. More commonly the algorithms require at least two different types of topological information, for example cells and points, or faces and edges. This requirement means that either data has been duplicated on a per element basis, or a lookup table must be provided. These two approaches have serious draw backs, per element duplication for hexahedral point fields cause an eight fold increase in memory but provide consistent memory access patterns, look up tables remove the per field data duplication and memory tension but effectively require an additional random read to global memory.

VTK-m has decided to go with the lookup table approach for a couple of reasons. First this is the current

This material is based upon work supported by the U.S. Department of Energy, Office of Science, under Award Number DE-SC0012386

approach that visualization tools such as VTK use, the lookup table unlike the copy requires less book keeping for when field values change, and lastly the lookup table only needs to be generated once per topology and than can be uses for all fields. The challenge for VTK-m is how to mitigate the cost of the random global memory accesses.

The XVis team decided to tackle this problem on a per device adapter basis so that we had the ability to leverage any device specific functionality. In the case of NVIDIA GPU's the primary problem with random global memory reads is that they cause uncoalesced memory accesses, as global memory reads must be accessed in 32, 64, or 128 byte transactions. When a warp executes an instruction that uses global memory, the fetches are coalesced into the minimum number of transactions possible. In effect this means that while global memory reads are expensive on GPU's the cost can be alleviated if you can utilize the extra data loaded by the transactions. The approach that VTK-m has taken is to automatically use custom CUDA iterators that when possible replace global reads with linear texture reads. Texture caching is enabled by linear texture reads, this caching allows for reuse when threads access spatial near locations. These custom iterators have resulted in nearly a 10% performance increase when executing algorithms that require cell and point information.

In the case of the Intel x86 architecture backend the approach taken was not to improve the performance of the random global reads, but to see if the reads allowed us an opportunity to move to a better memory layout for the hardware. A typical array of vectors in VTK-m is stored as Array-of-Structures with each element being a Vec like structure; this type of layout in testing has inhibited the ability for compilers to generate optimized vector instructions. When doing the read if we also do a transformation to a Structure-of-Arrays layout the compiler is able to generate optimized vector instructions. This change does come with an increase copy overhead, which can be mitigated with the loop tiling work that has been investigated as part of the function characterization work.

Function Characterization

The worklet is the primary component of all visualization algorithms inside of VTK-m. Therefore it is critical that worklets and related infrastructure are efficient with minimal overhead and footprint. As part of the investigation of function characterization we looked at the following items: support for polymorphic runtime classes, efficient worklet dispatch, removal of unnecessary code generation, and optimized paths based on function and topological domains.

One of the challenges when developing VTK-m was how to have efficient polymorphic runtime classes that work across all device adapters. This feature is needed so that operations such as integrators or implicit functions can be compartmentalized and not cause the number of worklets to be compiled to scale exponentially. While this feature is needed the VTK-m team was adamant on finding a solution that would not sacrifice performance, the result was the application of a pattern very similar to C function

This material is based upon work supported by the U.S. Department of Energy, Office of Science, under Award Number DE-SC0012386

callbacks. The pattern was than generalized, and used to implement implicit functions that work across all device adapters inside VTK-m while maintaining performance.

During the work on runtime polymorphic classes the developer team discovered a secondary application for the pattern. C++ code bases that rely templates face consistently high compilation times, and generally high artifact size. These issues are caused by the inability to leverage incremental builds, and the inability for compilers to efficiently remove duplicate code. The VTK-m team was able to identify that certain device adapters would continuously generate the same runtime infrastructure code for each worklet invocation. By utilizing the callback pattern within the device adapter scheduling infrastructure we achieved multiple goals; the amount of code generation for the debug and tbb device adapter was reduced by nearly 40%, the project was able to reduce compilation times, and lastly it allowed a refactoring of the scheduling infrastructure which allowed for other team members to investigate performance improvements such as loop tiling withing VTK-m.

Unlike the array characterizations, general function characterizations can be fairly challenging to automatically determine. Due to the strong type system used by VTK-m it is able to leverage this information to do some automatic inspection and optimizations. Our work in this automatic inspection was focused in two areas; parallel primitives, and worklet scheduling. For the parallel primitives we have been able to extend the device adapters to use improved versions of sort, sort by key, reduce, and reduce by key based on the input data types. For the worklet scheduling we have been able to improve performance by having device adapters automatically use a custom scheduler tuned for topology worklets that execute on structured grids.

Expand Data Models

As part of the development of the VTK-m toolkit we developed a robust data model that includes advanced features to support modern architectures, in situ analysis, and model representations used by DOE simulations. Specifically, heterogeneous memory space support is made possible through the VTK-m array handle design, while at the same time allowing for zero-copy support. The VTK-m data model flexibility allows for it to support models that previously toolkits such as VTK could not, this includes mixed topology models that use multiple cell sets, models that need to be represented in multiple different coordinate systems simultaneously, and models that use hybrid design such as structured points but unstructured cells. Starting with VTK 8, VTK-m accelerated algorithms are offered as part of VTK. The flexible data model design and zero-copy support built into the core of VTK-m was critical for the efficient integration. This work not only shows that the VTK-m data model is capable of supporting all the types offered by VTK, but is also a major step in allowing end users to take advantage of VTK-m through existing toolkits and programs such as ParaView, and Catalyst.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, under Award Number DE-SC0012386

Flyweight In Situ

As part of the XVis project we have been investigating the usage of non-standard memory layouts for arrays and data structures within VTK. The initial implementation developed the vtkMappedDataSet and vtkMappedDataArray allowed for arbitrary memory layouts using runtime polymorphism. This approach while very flexible introduced significant overhead by relying heavily on virtual methods for all read and writes access. Based on the knowledge gained by implementing the mapped arrays and VTK-m arrays we developed vtkAOSDataArrayTemplate and SOADataArrayTemplate, this design uses template-based polymorphism to allow for read and write access that are close to raw pointer performance. This approach is similar to the VTK-m which allows for the memory layout to be a template parameter, with the differences being no support for heterogeneous memory systems, and the number of elements in vector types being a runtime option rather than compile time as with VTK-m. Since this updated design is based on the experiences of the VTK-m array development allows for constant value arrays, implicit point arrays, and other efficient data model concepts allowing VTK to have tight in situ coupling with simulations.

As part of the demonstration of our work on flyweight in situ we have integrated VTK-m into the Alpine and Sensi in situ frameworks. Both of these frameworks aim to provide general-purpose lightweight in situ capability. The Sensi frameworks offers instrumentation support for simulation codes for in situ processes and supports a wide range of frameworks including Catalyst, Libsim, and ADIOS. Sensei aims to facilitate the instrumentation of simulation codes for in situ processes while supporting a diverse set of frameworks including Catalyst, Libsim, ADIOS, and now VTK-m for a flyweight in situ visualization solution. As part of this integration we have worked with the Sensi community to add a contour analysis module, with the ability to easily add more VTK-m algorithms as Sensi analysis modules by Sensi users.

As part of our work on developing in situ integration with VTK-m we collaborated with NVIDIA to develop a VTK-m, VTK, and Catalyst prototype for the PyFR simulation. This prototype was demonstrated at Supercomputing 2015 running on 256 nodes of Titan. The prototype was developed to show that simulation, analysis, and visualization can occur completely on the GPU. This work was well received by the PyFR team and became the foundation of more recent large scale in situ runs on Titan which used over 5000 nodes and was part of PyFR Gordon Bell nominated paper "Towards Green Aviation with Python at Petascale".

This material is based upon work supported by the U.S. Department of Energy, Office of Science, under Award Number DE-SC0012386

Presentations and Publications

"External Facelist Calculation with Data-Parallel Primitives." Brenton Lessley, Roba Binyahib, Robert Maynard, and Hank Childs. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*, June 2016.

"VTK-m: Accelerating the Visualization Toolkit for Massively Threaded Architectures." Kenneth Moreland, Christopher Sewell, William Usher, Li-ta Lo, Jeremy Meredith, David Pugmire, James Kress, Hendrik Schroots, Kwan-Liu Ma, Hank Childs, Matthew Larsen, Chun-Ming Chen, Robert Maynard, and Berk Geveci. *IEEE Computer Graphics and Applications*, 36(3), May/June 2016.

"An Integrated Visualization System for Interactive Analysis of Large, Heterogeneous Cosmology Data," Annie Preston, Ramyar Ghods, Jinrong Xie, Franz Sauer, Nick Leaf, Kwan-Liu Ma, Esteban Rangel, Eve Kovacs, Katrin Heitmann, Salman Habib. In *Proceedings of IEEE PacificVis*, April 2016.

"ParaView Catalyst: Enabling In Situ Data Analysis and Visualization." Utkarsh Ayachit, Andrew Bauer, Berk Geveci, Patrick O'Leary, Kenneth Moreland, Nathan Fabian, Jeffrey Mauldin. In

"Visualization Toolkit: Improving Rendering and Compute on GPUs," Presentation, Robert Maynard, *GPU Technology Conference*, April 2016.

"Adapting the Visualization Toolkit for Many-Core Processors with the VTK-m Library." Presentation, Christopher Sewell and Robert Maynard, *GPU Technology Conference*, April 2016.

"VTK-m: Building a Visualization Toolkit for Massively Threaded Architectures," Invited presentation, *Ultrascale Visualization Workshop*, November 2015.

"Hands-on Lab: In-Situ Data Analysis and Visualization: ParaView, Catalyst and VTK-m," Marcus Hanwell and Robert Maynard, GTC Lab, March 2015.

"Visualization Toolkit: Faster, Better, Open Scientific Rendering and Compute," Robert Maynard and Marcus Hanwell, GTC Presentation, March 2015.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, under Award Number DE-SC0012386

Acknowledgement

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under contract number 14-017566. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, under Award Number DE-SC0012386