SAND2011-3667C

Dax: Data Analysis at Extreme

Kenneth Moreland[1], Utkarsh Ayachit[2], Berk Geveci[2], Kwan-Liu Ma[3]

[1] Sandia National Laboratories, Albuquerque, NM 87185, USA
[2] Kitware, Inc., Clifton Park, NY 12065, USA
[3] Department of Computer Science, University of California at Davis, Davis, CA 95616, USA

E-mail: kmorel@sandia.gov

Abstract. Experts agree that the exascale machine will comprise processors that contain many cores, which in turn will necessitate a much higher degree of concurrency. Software will require a minimum of a 1000 times more concurrency. Most parallel analysis and visualization algorithms today work by partitioning data and running mostly serial algorithms concurrently on each data partition. Although this approach lends itself well to the concurrency of current high performance computing, it does not exhibit the appropriate pervasive parallelism required for exascale computing. The data partitions are too small and the overhead of the threads too large to make effective use of all the cores in an extreme scale machine. This paper introduces a new design philosophy for a visualization framework with the intention to encourage the design of algorithms that exhibit the pervasive parallelism necessary for extreme scale machines. We outline our design plans for Dax, a new visualization framework based on these design principles.

1. Introduction

Most of today's visualization libraries and applications are based off of what is known as the *visualization pipeline* [8,10]. The visualization pipeline is the key metaphor in many visualization development systems is also the internal mechanism or external interface for many end-user visualization applications.

Although the visualization pipeline lends itself well to the concurrency of current high performance computing [5, 11, 14, 15, 17], its structure prohibits the necessary extreme concurrency required for exascale computers. Large scale concurrency today is achieved by replicating the pipeline and partitioning the input data amongst processes [1]. However, extreme scale computers would require the data to be broken into billions of partitions. The overhead of capturing the connectivity information between these partitions as well as the overhead of executing these large computation units on such small partitions of data are too great to make such an approach practical.

To understand why, consider the sobering comparison between the Jaguar XT5 partition, a current petascale machine, and the projections for an exascale machine given in Table 1. These estimates are the envelope of those given by the International Exascale Software Project RoadMap [7] and the DOE Exascale Initiative Roadmap [2]. Because processor clock rates are not increasing, an exascale computer requires a thousand-fold increase in the number of cores. Furthermore, trends in processor design suggest that these cores must be hyper-threaded in order to keep them executing at full efficiency. In all, to drive a complete exascale machine will require between one and ten billion concurrently running threads.

Table 1. Comparison of characteristics between petascale and projected exascale machines.

	Jaguar – XT5	Exascale	Increase
Cores	224,256	100 million – 1 billion	400 – 5,000×
Threads	224,256 way	1 – 10 billion way	4,000 – 50,000×
Memory	300 Terabytes	10 – 128 Petabytes	30 – 500×

The overhead of partitioning a mesh at this fine a level and executing independent visualization pipelines on so many threads imposes too high of an overhead to be practical. Even if we somehow avoid the problem of running on the largest exascale machines, the problem of a fundamental change in processor architecture persists. The parallel visualization pipeline simply does not conform well to multi-core processors and many-core accelerators. In response several researchers are pursing the idea of a *hybrid parallel pipeline* [3, 4, 9]. The hybrid parallel pipeline breaks the problem into two hierarchical levels. The first level partitions the data amongst distributed memory nodes in the same way as the current parallel pipeline. In the second level we run a threaded shared memory algorithm to take advantage of a multi- or many-core processor.

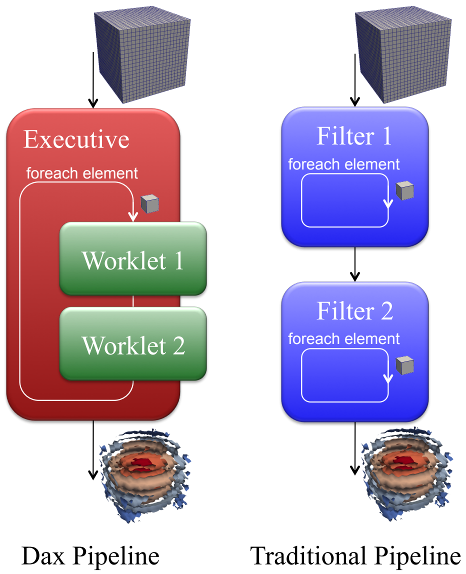
Although the current visualization pipeline does a good job in providing this first level of distributed memory concurrency, it provides no facilities whatsoever for this second layer of multi-threaded concurrency. This places the onus on each visualization pipeline filter developer. That is, each filter must be independently and painstakingly designed to exploit concurrency and optimized for whatever architecture is used. Even if this undertaking were to be performed, the concurrency is ultimately undermined at the connections of filters where execution threads must be synchronized and data combined.

This paper proposes an alternative framework to the visualization pipeline to perform data analysis at extreme scales. This proposed alternative framework provides the foundation for the development of the *Dax toolkit*. The Dax toolkit provides an infrastructure for developing visualization algorithms with pervasive parallelism. Dax does this by promoting best practices and encapsulating the management of threads, communication, and data management. Dax will provide a simplified development environment for GPU-based accelerator computing and pave the way for exascale computing architectures.

2. The Worklet as the Computational Building Block

The computational unit of our proposed Dax framework is a *worklet*, a single operation on a small piece of data. A worklet is simply a function that accepts as an argument a single element of a data set and returns some computation based on that element. Because worklets are light weight, lack state, and are limited to small data accesses, their execution can be scheduled on the massive amount of concurrent threads required by exascale computers and associated multi- and many-core processors.

The Dax unit called an *executive* applies worklets to a mesh by mapping the worklet to all elements in the mesh. The executive encapsulates the complexity of scheduling threads and communicating amongst them. It also encapsulates the idiosyncrasies of the underlying parallel hardware. At a high level, the basic concept is similar to that of MapReduce [6], except that Dax is designed specifically for scientific data rather than database queries. This distinction allows Dax to provide better metaphors for the data to ease development and also allows Dax to exploit the known topological connectivity of the data to more efficiently collect related data entries.



Like filters in a traditional visualization pipeline (such as the one implemented in VTK [16]), worklets have inputs and outputs and can be chained together. The figure at the right compares the workflow of each system. Although similar in nature, the Dax pipeline affords several advantages.

First, the Dax framework better hides the complexity of parallel programming. The iteration over the elements, which must be executed concurrently, is encapsulated in the executive. The same executive may be applied to an indefinite number of worklets. In contrast, in the traditional visualization pipeline this iteration mechanism is embedded and entangled in the visualization algorithm. For every new filter created, the parallel scheduling must be redesigned, reimplemented, and redebugged.

Second, the Dax framework simplifies the adaption of an algorithm implementation for multiple computing architectures by changing the executive implementation. For example, a special debugging executive could schedule worklets on a single thread so that they may be run in a debugger without having to worry about thread management. Because worklets are constrained to using local memory provided by the executive, a worklet executing correctly in a serial debugging executive will also be correct when executed in parallel. Also, so long as the worklet implementation can be ported to C compiler variants for CPU, GPU, or cell architectures, different executives can run the same implementation on each. Because the traditional visualization pipeline provides no such encapsulation for iteration, a new implementation is required to adapt a filter to new architectures.

Third, when chained together worklets provide better memory access than their filter counterparts. When a chain of worklets is executed, the executive can read a single element worth of data from memory and execute every worklet on that data before having to write anything back out to memory. This execution flow maximizes the execution to memory fetch ratio. Such optimization is becoming increasingly important on today's architectures where execution rates outstrip memory fetch rates and will be vital at exascale where power constraints limit the amount of memory accesses that may be done. In contrast, each filter in the traditional visualization pipeline must independently read every element from memory, process them, and write all the results back to memory before the execution of subsequent filters can start.

3. Modes of Execution

The obvious drawback of defining computation as an operation on a single element is that we loose the ability to express how these operations get applied to a mesh and how the results are interpreted and connected. To recover this expressiveness, the executive must support multiple modes of execution that determine how worklets are applied to a mesh. We can immediately identify several necessary modes of execution.

Field Map is a basic mapping of a worklet onto field values. A simple example is finding the magnitude of a vector field.

Cell Connectivity Map applies a worklet to each cell but also provides access to the cell's topology and interpolated fields (such as those defined on points). A gradient estimator worklet could use a cell connectivity map to derive the gradient of an interpolated field.

Topological Reduce applies a worklet to each point but also provides access to field data in adjacent cells. An averaging worklet could convert the previously mentioned cell gradients to an interpolated gradient field.

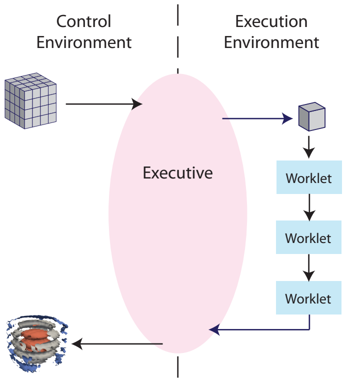
Generate Geometry applies a worklet to a cell much like a cell connectivity map except that the results are interpreted as new geometry. After executing the worklet, the generate geometry execution must identify coincident points and reconstruct the topological connections amongst geometry created by independent worklet threads. A marching cubes worklet would require a generate geometry execution mode.

This list is doubtlessly incomplete. To ensure flexibility, Dax must also allow developers to create new modes of execution for algorithms with new or unique communication needs.

4. Division of Control and Execution

In Dax, worklets must run in isolation. They cannot communicate with each other, and they cannot directly access any part of the working memory not explicitly passed through arguments. These constraints are necessary to ensure that a worklet cannot fail due to concurrent execution (by, for example, creating a deadlock or race condition). Worklets may also be scheduled to execute on a completely different computing architecture than the rest of program.

Consequently, the Dax toolkit is broken into two programming environments. Each environment has a different API and slightly different language semantics. The first environment is the execution environment, the environment in which worklets execute. The execution environment contains the API to define worklets and manipulate elements. This API uses a simple set of C functionality so that it is easily ported to compilers for multiple parallel architectures. The second environment is the control environment, which provides data and instantiates execution. This is the C++ API that is designed for users that want to use the Dax toolkit to analyze and visualize their data. These two environments are analogous to the device and host environments, respectively, of GPU programming languages like CUDA [13] and OpenCL [12]. The executive resides in between these two environments to manage communication and control flow between them as shown in the figure shown here.



5. Achieving Efficiency Through Constraints

One of the major design goals of the Dax toolkit is to encourage efficient algorithms by constraining the resources of the developer. Worklets are constrained in multiple ways: they can neither access data outside a limited range nor directly communicate with other worklets. Part of the motivation for these constraints is safety. With these constraints in place, race conditions and deadlocks are not possible, which greatly simplifies the design and implementation of algorithms. Another part of the motivation is efficiency. These constraints naturally enforce coding design that best utilize multiple threads and memory hierarchies.

Not every visualization algorithm can be adequately expressed with these constraints. They require communication patterns and execution scheduling not well characterized by a per-element execution. Consequently, some algorithms necessitate creating new execution modes. But by their nature execution modes are more difficult to design and more error prone to implement than worklets.

Developers will naturally prefer implementing worklets, which inherently run more efficiently than code requiring communication. When new execution modes are necessary, they will be designed to be as small and as simple as possible for ease of implementation and support. Since complicated communication tends to run less efficiently with high concurrency, the tendency to develop limited execution modes also contributes to more efficient code.

6. Challenges Ahead

Implementing the Dax toolkit poses several development challenges. Algorithms that manipulate fields by mapping worklets to elements are relatively straightforward (and in fact we already have some introductory code to do this).

Algorithms that build new topological features, such as clipping, slicing, and contouring, provide several challenges. Most such algorithms cannot determine the number of resulting elements a priori, so memory management is a concern. Another challenge is connecting elements independently generated by separate worklet instances. We plan to mitigate the problem by requiring worklets to define geometry based on features of the input. Thus, topological connections of the output can be inferred by the known connections of the input.

Acknowledgments

This work was supported in full by the DOE Office of Science, Advanced Scientific Computing Research, under award number 10-014707, program manager Lucy Nowell. Part of this work was performed by Sandia National Laboratories. Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration.

References

[1] Ahrens J, Law C, Schroeder W, Martin K and Papka M 2000 A parallel approach for efficiently visualizing extremely large, time-varying datasets Tech. Rep. #LAUR-00-1620 Los Alamos National Laboratory

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

[3] Camp D, Garth C, Childs H, Pugmire D and Joy K 2010 Streamline integration using mpi-hybrid parallelism on large multi-core architecture *IEEE Transactions on Visualization and Computer Graphics* DOI=10.1109/TVCG.2010.259

[4] Chen L and Fujishiro I 2008 Optimization strategies using hybrid MPI+OpenMP parallelization for large-scale data visualization on earth simulator *A Practical Programming Model for the Multi-Core Era* vol 4935 pp 112–124 DOI=10.1007/978-3-540-69303-1_10

[5] Childs H, Pugmire D, Ahern S, Whitlock B, Howison M, Prabhat, Weber G H and Bethel E W 2010 Extreme scaling of production visualization software on diverse architectures *IEEE Computer Graphics and Applications* **30** 22–31

[6] Dean J and Ghemawat S 2008 MapReduce: Simplified data processing on large clusters *Communications of the ACM* **51** 107–113

[7] Dongarra J, Beechman P *et al.* 2010 The international exascale software project roadmap Tech. Rep. ut-cs-10-652 University of Tennessee

[8] Haeberli P E 1988 ConMan: A visual programming language for interactive graphics *ACM SIGGRAPH Computer Graphics* **22** 103–111

[9] Howison M, Bethel E W and Childs H 2011 Hybrid parallelism for volume rendering on large, multi- and many-core systems *IEEE Transactions on Visualization and Computer Graphics* DOI=10.1109/TVCG.2011.24

[10] Lucas B, Abram G D, Collins N S, Epstein D A, Gresh D L and McAuliffe K P 1992 An architecture for a scientific visualization system *IEEE Visualization* pp 107–114

[11] Moreland K, Rogers D, Greenfield J, Geveci B, Marion P, Neundorf A and Eschenberg K 2008 Large scale visualization on the Cray XT3 using paraview *Cray User Group*

[12] Munshi A 2010 *The OpenCL Specification* Khronos OpenCL Working Group Version 1.1, Revision 36

[13] NVIDIA 2009 *NVIDIA CUDA Programming Guide, Version 2.3.1*

[14] Patchett J, Ahrens J, Ahern S and Pugmire D 2009 Parallel visualization and analysis with ParaView on a Cray Xt4 *Cray User Group*

[15] Pugmire D, Childs H and Ahern S 2008 Parallel analysis and visualization on cray compute node linux *Cray User Group*

[16] Schroeder W, Martin K and Lorensen B 2004 *The Visualization Toolkit: An Object Oriented Approach to 3D Graphics* 4th ed (Kitware Inc.) ISBN 1-930934-19-X

[17] White D 2005 Red storm capability visualization level II ASC milestone #1313 final report Tech. Rep. SAND2005-5989P Sandia National Laboratories