Optimizing VisIt for Next Generation Architecture

Jian Huang
University of Tennessee

Hank Childs
University of Oregon

Volume rendering of 8 trillion cell data set.

Carbon dioxide from the ocean rising as plumes during Feb. 1900

Volume rendering of 8 trillion cell data set.

First production release
Project started
2000

Cross Mesh Field Evaluation
In situ visualization
2003

Molecular visualization
2004

SLIVR volume renderer, Parallel streamlines
2006

Data binning, Python filters, Scaled to 8 trillion cell data set
2008

Custom GUIs w/PySide, Double precision
2010

Scaled to 8 trillion cell data set
2012

Threading support
2014

VTK-m
IPCC Project Goals

- Demonstrate and push the envelope of scientific visualization software making effective use of the Intel® Xeon® Processors and Intel® Xeon Phi™ products.
  - Rendering: SWR / OSPRay
  - Data processing: parallel filtering using Intel Xeon Phi products.

- Demonstrate and promote the effective practices that contribute to the story of a viable HPC ecosystem on Intel Xeon Phi processors and coprocessors.
VisIt is an open source, turnkey application for data analysis and visualization of mesh-based data.

- Production end-user tool supporting scientific and engineering applications.
- Provides an infrastructure for parallel post-processing that scales from desktops to massive HPC clusters.
- Source released under a BSD style license.

Density Isovolume of a 3K^3 (27 billion cell) dataset
VisIt supports a wide range of use cases.

- Data Exploration
- Quantitative Analysis
- Visual Debugging
- Comparative Analysis
- Presentation Graphics
Examples of VisIt’s visualization capabilities.

- Streamlines
- Vector / Tensor Glyphs
- Pseudocolor Rendering
- Volume Rendering
- Molecular Visualization
- Parallel Coordinates
## Project Introduction

VisIt scales well on current HPC platforms.

<table>
<thead>
<tr>
<th>Machine</th>
<th>Architecture</th>
<th>Problem Size</th>
<th># of Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph</td>
<td>X86_64</td>
<td>20,001³ (8 T cells)</td>
<td>12K</td>
</tr>
<tr>
<td>Dawn</td>
<td>BG/P</td>
<td>15,871³ (4 T cells)</td>
<td>64K</td>
</tr>
<tr>
<td>Franklin</td>
<td>Cray XT4</td>
<td>12,596³ (2 T cells)</td>
<td>32K</td>
</tr>
<tr>
<td>JaguarPF</td>
<td>Cray XT5</td>
<td>12,596³ (2 T cells)</td>
<td>32K</td>
</tr>
<tr>
<td>Juno</td>
<td>X86_64</td>
<td>10,000³ (1 T cells)</td>
<td>16K</td>
</tr>
<tr>
<td>Franklin</td>
<td>Cray XT4</td>
<td>10,000³ (1 T cells)</td>
<td>16K</td>
</tr>
<tr>
<td>Ranger</td>
<td>Sun</td>
<td>10,000³ (1 T cells)</td>
<td>16K</td>
</tr>
<tr>
<td>Purple</td>
<td>IBM P5</td>
<td>8,000³ (0.5 T cells)</td>
<td>8K</td>
</tr>
</tbody>
</table>

### Scaling Studies of Isosurface Extraction and Volume Rendering (2009)

VisIt is also used daily by domain scientists.
VisIt’s infrastructure provides a flexible platform for custom workflows.

- **C++ Plugin Architecture**
  - Custom File formats, Plots, Operators
  - Interface for custom GUIs in Python, C++ and Java

- **Python Interfaces**
  - Python scripting and batch processing
  - Data analysis via Python Expressions and Queries.

- **Libsim library**
  - Enables coupling of simulation codes to VisIt for in situ visualization.
VisIt: What’s the Big Deal?

- Everything works at scale
- Robust, usable tool
- Features that span the “power of visualization”:
  - Data Exploration
  - Confirmation
  - Communication
- Features for different kinds of users:
  - Visualization Experts
  - Code Developers
  - Code Consumers

Healthy future: Vibrant Developer and User Communities
VisIt is a vibrant project with many participants.

- The VisIt project started in 2000 to support LLNL’s large scale ASC physics codes.
- The project grew beyond LLNL and ASC with research and development from DOE SciDAC and other efforts.
- VisIt is now supported by multiple organizations:
  - LLNL, LBNL, ORNL, UC Davis, Univ of Utah, Intelligent Light, …
- Over 75 person years of effort, 1.5+ million lines of code.
The VisIt team focuses on making a robust, usable product for end users.

- Regular releases (~6/year)
  - Executables for all major platforms
  - End-to-end build process script `build_visit`

- Customer Support and Training
  - visitusers.org, wiki for users and developers
  - Email lists: visit-users, visit-developers
  - Beginner and advanced tutorials
  - VisIt class with detailed exercises

- Documentation
  - “Getting data into VisIt” manual
  - Python interface manual
  - Users reference manual

- Modernization
  - Updating for plugin architecture based on VTK 7+, in-sync with Paraview
VisIt employs a parallelized client-server architecture.
VisIt-IPCC Timeline

- **Summer/Fall 2014:**
  - VisIt builds on Haswell and KNC; mini-app for VisIt showed good scaling on KNC, demonstrated at SC’14; VisIt itself shows poor scaling properties on KNC, however.

- **Spring/Summer/Fall 2015:**
  - Ospray+VisIt integrated and works on Haswell
  - Significant study on performance leading to: tuning of VisIt, memory (TBBmalloc), better affinity of threads to cores; good scaling properties for VisIt on KNC
  - Demonstrated at SC’15

- **Spring 2016:**
  - Initial open source release of VisIt + Intel ecosystem
VisIt-IPCC Timeline

- **Fall 2016:**
  - Ospray+VisIt, SWR+VisIt integrated and works on Broadwell and KNL; demonstrated by multiple users at SC’16
  - Good scaling properties for VisIt on KNL

- **Spring 2017:**
  - Prepared for transiting Ospray support in VisIt from VTK 6.0 to VTK7+, coordinating with VisIt developers leading the transition
  - Installed and deployed on Stampede 2 (KNL), no scaling test on Stampede 2 yet.

- **Fall 2017:**
  - Cori test/tuning completed
  - Extensive SWR testing
Rasterization

- Method for rendering surfaces
- Idea: iterate over geometric primitives (triangles) and deposit colors into image
- “Object order” method
  - iterate over objects (i.e., geometric primitives (triangles))
Ray Tracing

- Method for rendering surfaces
- Idea: iterate over pixels and determine geometry visible from that pixel
- “Image order” method
  - iterate over pixels
Rasterization vs Ray-tracing

- Rasterization: traditional technique for visualization
- Ray-tracing: potential for better pictures
  - May be faster in some cases (?)
- Recent research considering using ray-tracing over rasterization
  - Workloads are changing, with more geometry per node
- Evidence to date is incomplete
  - Victories for both techniques
- SC16 study by Larsen et al (Performance Model for In Situ Rendering) tackles this question
  - Slide credit to Matt Larsen
Ray Tracing Versus Rasterization

- 32 MPI ranks
- 100 images
  - One time initialization for ray-tracing is amortized
- Ray tracing
  - Wins when
    - Number of objects is large
    - Lower resolutions
CPU Ray Tracing Versus Rasterization

- 125 M Triangles
- 27 M Triangles
- 1 M Triangles
- 384^2 Resolution
- 4000^2 Resolution

Ray Tracing
- Render Time

Rasterization
- Render Time

Objects

\sqrt{\text{Image Size}}

Ratio

0.50
1.00
1.50

Resolution

Image Size

1 M Triangles

27 M Triangles

125 M Triangles

384^2 Resolution

4000^2 Resolution

2000
3000
4000

CPU1 Rasterization vs Ray-tracing
CPU Ray Tracing Versus Rasterization

Ray Tracing Wins

Rasterization Wins
CPU Ray Tracing Versus Rasterization

Ray Tracing
~10x Faster
At Lower Resolutions

Rasterization
~1.5x Faster
At High Resolutions
CPU Ray Tracing Versus Rasterization

Common Configurations
End Primer on Rendering

- **OpenSWR**: rasterization-based
  - Also integrated into Mesa, which is a software implementation of OpenGL

- **OSPRay**: ray tracing-based
  - Ray tracing also allows for many advanced lighting effects

- Both now easily accessible through enhancements to VTK
VisIt-IPCC Rendering Status

- Intel® Xeon® processors only:
  - Both VisIt Viewer + Engine work with SWR for surface & volume rendering
    - SWR pass rate: 94.44% (1697 total tests, pixdiff=8, avgdiff=5)
  - VisIt Engine works with OSPRay & SWR for surface and volume rendering

- Intel Xeon + KNL:
  - VisIt Viewer run on Xeon
  - VisIt Engine run on Xeon Phi (both SWR & OSPRay)
  - Can provide surface and volume rendering
VisIt-IPCC Release & Build

- **Software**
  - SWR/Mesa version 17.1.0
  - VisIt 3.0Beta (VTK 8.0.0)

- **Hardware**
  - Xeon + Xeon Phi (KNL)
  - Installed on KNL deployment: Cori (NERSC, Sep 2017)

- **Open-Source Release**
  - VisIt source for Intel Xeon and Intel Xeon Phi:
    http://visit.ilight.com/svn/visit/branches/bonnell/VTK-6.3-port/src/
The Big Plan: OSPRay

- VisIt 3.0 going to VTK8
- Excellent integration with OSPRay
- Expected ~Jan 2018
- Will remove need for special coding paths
The Big Plan: SWR

- VisIt updating to Mesa17
- Mesa17: SWR “just works” (and really well)
Recent rendering results

- All results on Cori using SWR
- Using Mesa 17.0.5
  - 17.3 should be even better for KNL
# VisIt+Cori+SWR (Haswell)

<table>
<thead>
<tr>
<th>Dataset Size</th>
<th>Output Size</th>
<th>Type</th>
<th>Arch</th>
<th>Time (Pass 2)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>9,714,856</td>
<td>1024 x 1024 px</td>
<td>LLVMPipe</td>
<td>Haswell</td>
<td>1.40930</td>
<td></td>
</tr>
<tr>
<td>9,714,856</td>
<td>1024 x 1024 px</td>
<td>SWR</td>
<td>Haswell</td>
<td>0.13606</td>
<td>10.358x</td>
</tr>
<tr>
<td>23,300,836</td>
<td>1024 x 1024 px</td>
<td>LLVMPipe</td>
<td>Haswell</td>
<td>3.34459</td>
<td></td>
</tr>
<tr>
<td>23,300,836</td>
<td>1024 x 1024 px</td>
<td>SWR</td>
<td>Haswell</td>
<td>0.28652</td>
<td>11.673x</td>
</tr>
<tr>
<td>42,845,096</td>
<td>1024 x 1024 px</td>
<td>LLVMPipe</td>
<td>Haswell</td>
<td>5.99737</td>
<td></td>
</tr>
<tr>
<td>42,845,096</td>
<td>1024 x 1024 px</td>
<td>SWR</td>
<td>Haswell</td>
<td>0.52170</td>
<td>11.496x</td>
</tr>
</tbody>
</table>
### VisIt+Cori+SWR (KNL)

<table>
<thead>
<tr>
<th>ID</th>
<th>Resolution</th>
<th>Tool</th>
<th>Node</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>9,714,856</td>
<td>1024 x 1024 px</td>
<td>LLVMPipe</td>
<td>KNL</td>
<td>6.16488</td>
</tr>
<tr>
<td>9,714,856</td>
<td>1024 x 1024 px</td>
<td>SWR</td>
<td>KNL</td>
<td>0.27565</td>
</tr>
<tr>
<td>23,300,836</td>
<td>1024 x 1024 px</td>
<td>LLVMPipe</td>
<td>KNL</td>
<td>14.62035</td>
</tr>
<tr>
<td>23,300,836</td>
<td>1024 x 1024 px</td>
<td>SWR</td>
<td>KNL</td>
<td>0.55981</td>
</tr>
<tr>
<td>42,845,096</td>
<td>1024 x 1024 px</td>
<td>LLVMPipe</td>
<td>KNL</td>
<td>26.60455</td>
</tr>
<tr>
<td>42,845,096</td>
<td>1024 x 1024 px</td>
<td>SWR</td>
<td>KNL</td>
<td>0.98717</td>
</tr>
</tbody>
</table>
VisIt-IPCC Data Processing on KNL

- VisIt isosurface algorithm: scales well up to 60 cores
  - speedups continue with more cores, but sublinear

- Other scalable algorithms
  - isovolume, box, vertex normals, etc.

- New challenge:
  - Parallelism on Phi exposes serial code in VisIt that becomes bottleneck
    - Examples: extents calculations, reading data, preparing message from client to server
  - This code must all be parallelized for overall scalability
Doing 68M cells in <1s using 68 threads.
Isosurfacing on Cori

Weak scaling on NERSC Cori KNLs, 50^3 mesh per thread (red = w/TBBmalloc, blue = w/o TBB malloc)
SC’16 Demo

- Dataset: a simulation of tornado
  - 490 x 490 x 280 spatial resolution
  - 220 time steps

- Application scenario: simulation-time visualization

- Goal: KNL / Xeon filtering + rendering

- Platform: one dual-socket Haswell Xeon workstation, and one KNL DAP
SC’16 Demo

- Interactive Rendering
  - Running in VisIt Viewer on Xeon
  - Volume rendering using OSPRay
  - 1~2 fps, viewport size: 1000 x 1000
SC’17 Demo

- Tornado data set
- SWR rendering
- VisIt 3.0Beta+VTK8+ SWR 17.0.5
Priorities – current project year

- Provide and support downloadable executables and source (with robust build documentations) as the software continue to evolve

- Upgrade VisIt’s dependency on VTK from VTK6.1 to VTK8
  - key architectural changes with rendering to maintain more robust plugins

- Enable and support production use of OSPRay/SWR enhanced VisIt on Cori

- Expand the collaboration with DOE’s Cinema and develop compelling use cases of VisIt-enhanced Cinema project
INTRODUCTION TO PERSISTENT MEMORY PROGRAMMING

Usha Upadhyayula
Eduardo Berrocal
Notices and Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

Intel, the Intel logo, Xeon, and others are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.

© Intel Corporation.
Introduction to Persistent Memory

• What is Persistent Memory?
  • Byte Addressable
  • Persistent Like Storage
  • Load /Store Access
    • No page caching
  • Cache Coherent
  • Memory-like Performance

• Why Does it Matter now?
  • Adds a new tier between DRAM and Block Storage (SSD/HDD)
  • Larger Capacity, Higher Endurance, Consistent low latency
  • Ability to do in-place persistence
    • No Paging, No Context Switching, No Interrupts, No Kernel Code Running
  • Ability to do DMA & RDMA

• What’s the impact on the Applications?
  • Need Ways to Enable Access to New High Performance Tier
  • May need to re-architect to unlock the new features and performance
What does Re-architecting to Byte-Addressable Persistent Memory Mean

• Applications Need to
  • Ensure Data Persistence
    • Stores are not guaranteed persistence until flushed out of CPU caches
  • Assure Data Consistency
    • Maintain Transactional Consistency of Persistent Data structures
      • Preventing torn updates using Transactions
  • Use Persistent Memory Aware Allocator
    • Prevent Persistent Memory Leaks
  • Detect and Handle Persistent Memory Errors
NVML: A Suite of Open Source Libraries

- **libpmemlog**: Interface to create a persistent memory resident log file
- **libpmemobj**: Interface for persistent memory allocation, transactions and general facilities
- **libpmemblk**: Interface to create arrays of pmem resident blocks, of same size, atomically updated

Support for volatile memory usage:
- **memkind**: Low level support for local persistent memory
- **libpmem**: Low level support for remote access to persistent memory
- **librpmem**: Low level support for remote access to persistent memory

Link to Open Source:
- [http://pmem.io/nvml/](http://pmem.io/nvml/)

Link to Intel® Developer Zone:
Evolving Persistent Memory Usages in HPC

• System’s point of view
  • Augmented capacity (great for memory-bound codes)
    • **6TB** for a 2 socket system
    • Speed close to that of DRAM
    • Cheaper than DRAM
  • Use case:
    • **ISO3DFD***: solve 3D acoustic isotropic wave equation
    • Used for Oil & Gas Exploration
    • Check-point hundreds of MBs each time step; reuse it in future time steps
Evolving Persistent Memory Usages in HPC

• Middleware’s point of view
  • Adding capabilities – such as Fault Tolerance – transparently to users
    • Example: PMEM-aware MPI* (one sided and MPI-IO)
      • One sided extension: Define persistent “windows” for automatic check pointing
      • MPI-IO: 2 modes: PMEM_IO_AWARE_FS, PMEM_IO_DISTRIBUTED_CACHE
    • Improving hierarchical check-pointing by using PMEM as “level 0” (super fast check-pointing)
      • 1 check-point for every time step
Evolving Persistent Memory Usages in HPC

• Application’s point of view
  • Expanding application’s features and interactivity
    • Enables **Interactive In-Situ Visualization** with Large Memory capacities
      • Early exploration with ParaView* is underway
      • VisIt* with LibSim* and other applications may be considered in the future
    • Multiple time-steps can be kept in memory eliminating round trips into File system
  • Improve restart times on Application crash/power fails