Petascale Visualization: Approaches and Initial Results

James Ahrens

Li-Ta Lo, Boonthanome Nouanesengsy,

John Patchett, Allen McPherson

Los Alamos National Laboratory

LA-UR- 08-07337

Questions about visualization in the petascale era

- **What are our options for running our visualization software?**
- **Can we run our visualization software on the supercomputer?**
- **Do we need to a visualization cluster to support the supercomputer?**

- **Define supercomputer and visualization options**
- **Current approach and performance**
- **New approach**
  - Ray-tracing for rendering

Trends in petascale supercomputing

- **Lots of compute cycles**
  - Multi-core revolution
- **Increasing latency from processor to memory, disk and network**
  - Many memory-only simulation results
- **Can compute significantly more data than can be saved to disk**
  - For example, on RR
    - To disk: 1 Gbyte/sec
    - Compute: 100 Gbytes on a triblade from Cells to Cell memory
- **Very expensive**

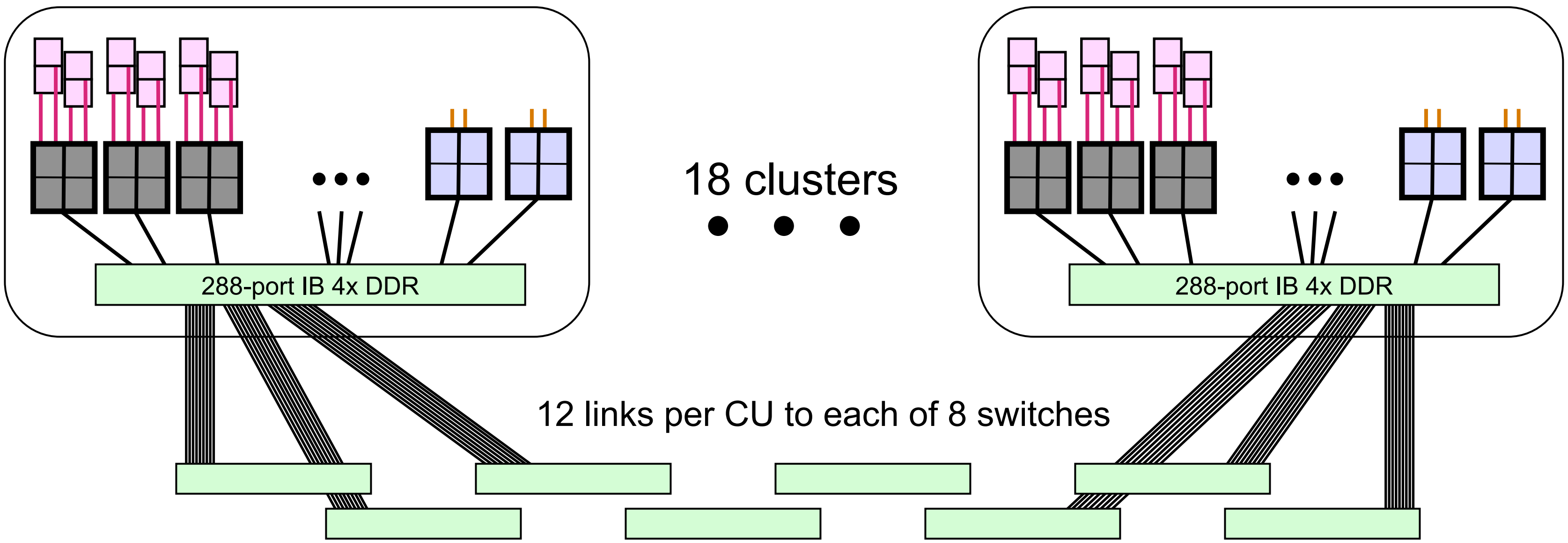
Supercomputing platforms

- **Definition of supercomputing platform**
  - Type of node
- **Co-processor architecture**
  - Example: Roadrunner
- **Multi-core processor**
  - Example: 16-way CPU (4 x 4 quad Opteron)

Roadrunner architectural overview

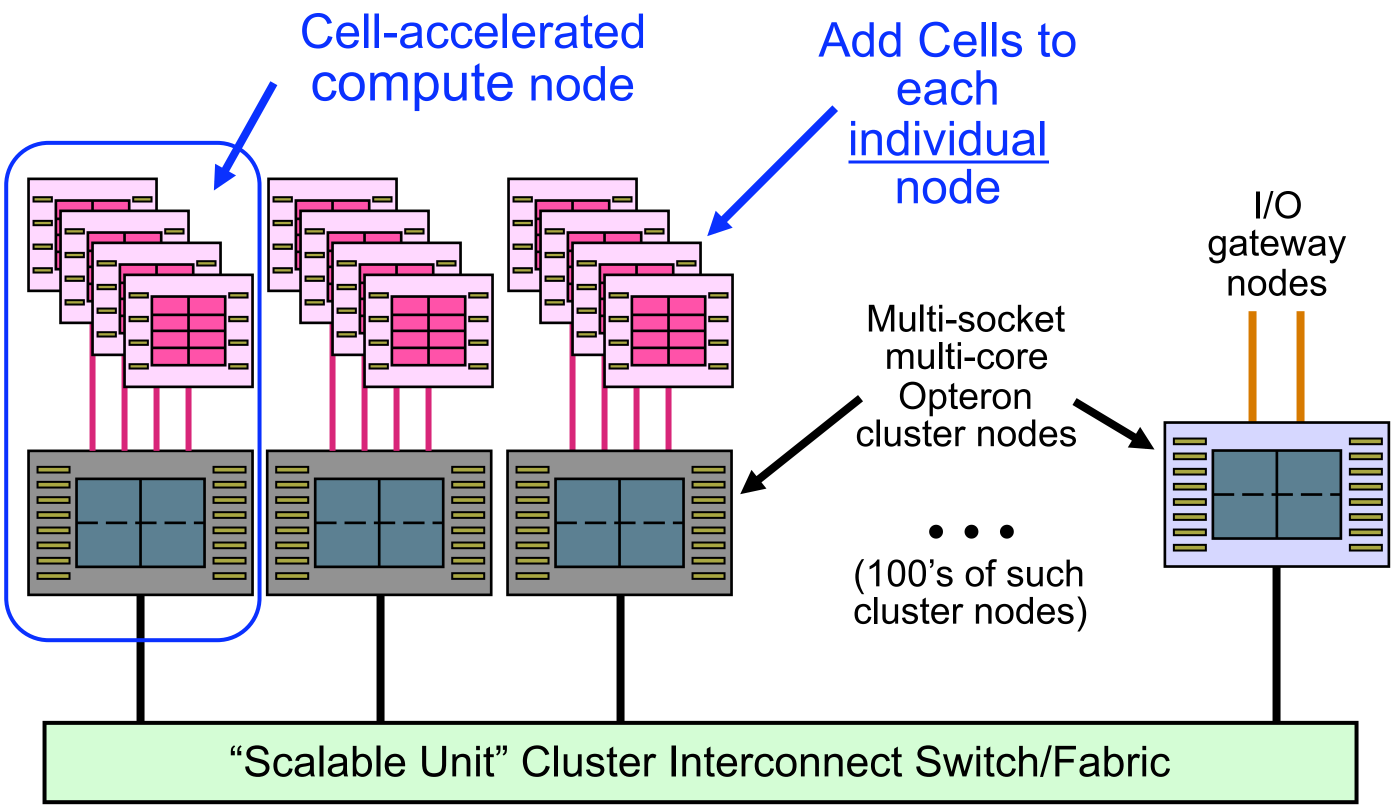
Connected Unit cluster
180 Triblade compute nodes w/ Cells
12 I/O nodes

6,480 dual-core Opterons ⇒ 23.3 Tflop/s (DP)
12,960 Cell eDP chips ⇒ 1.3 Pflop/s (DP)



Eight 2^{nd}-stage 288-port IB 4X DDR switches

Roadrunner is Cell-accelerated, not a cluster of Cells



Node-attached Cells is what makes Roadrunner different!

IBM Cell processors powers the Playstation

- **12960 Cell chips in Roadrunner!**
  - In Playstation – the Cell is used for physics processing – e.g. Little Big Planet
- **We plan to use the Cell for rendering…**

Can we efficiently run our visualization/rendering software on the supercomputer?

- **The data understanding process is composed of a number of activities:**
  - Analysis and statistics
  - Visualization
    - Map simulation data to a visual representation (i.e geometry)
  - Rendering
    - Map geometry to imagery on the screen
- **Already runs on the supercomputer**
  - Analysis, statistics and visualization
- <u>**Issue is rendering**</u>
- **Fast rendering for interactive exploration**
  - 5-10 fps minimum, 24-30 fps – HDTV, 60 fps - stereo
- **Typically provided by commodity graphics in a visualization cluster**

Related Work – Visualization hardware

- **SGIs (late 1998)**
  - SGI shared memory machine
  - “Blue Mountain ran Linpack, one of the computer industry's standard speed tests for big computers, at a fast 1.6 trillion operations per second (teraOps), giving it a claim to the coveted top spot on the TOP500 list, the supercomputer equivalent of the Indianapolis 500.”
  - Integrated Reality Engine graphics ($250K/each)
- **Commodity clusters (2004)**
  - Leverage commodity technology to replace SGI infrastructure
    - “Game” cards, PC-class nodes, Infiniband networks
- **What is next?**

Analysis of tradeoffs
Visualization/rendering on supercomputer or cluster

- **Visualization/rendering on the supercomputer**
  - Disadvantages
    - Cost to port rendering to the supercomputing platform
    - Allocate portion of supercomputer to analysis and visualization
  - Advantages
    - Scalable to supercomputer size
    - Access to “all” simulation results
- **Visualization/rendering on cluster**
  - Disadvantages
    - Cost of cluster and infrastructure to connect it
    - Less access to data – only data that is written to disk
  - Advantages
    - Independent resource devoted to visualization task
    - Very fast especially on smaller datasets

Standard parallel rendering solution
Sort-last parallel rendering of large data

- **Sort-last parallel rendering algorithms have two stages:**
  - 1. Rendering stage
    - The node renders its assigned geometry into a “distance/depth” buffer and image buffer
  - 2. Networking / compositing stage
    - These image buffers are composited together to create a complete result
- **Given there are two stages the performance is limited by the slower stage**
  - Assuming pipelining of the stages

Performance study

- **For real-world performance testing and to prepare for petascale visualization tasks…**
- **Incorporate rendering approaches into vtk/ParaView**
  - Vtk is open-source visualization library
  - Paraview (PV) is open-source parallel large-data visualization tool
- **Initially render on two types of nodes**
  - Multi-core node - 1, 2, 4, 8, 16 way
    - Mesa using multiple processes via parallel vtk
      - Data automatically partitioned and rendered by each process
      - On-node compositing to create final image
  - GPU
    - Standard OpenGL driver

Vtk/PV rendering performance – standard approach

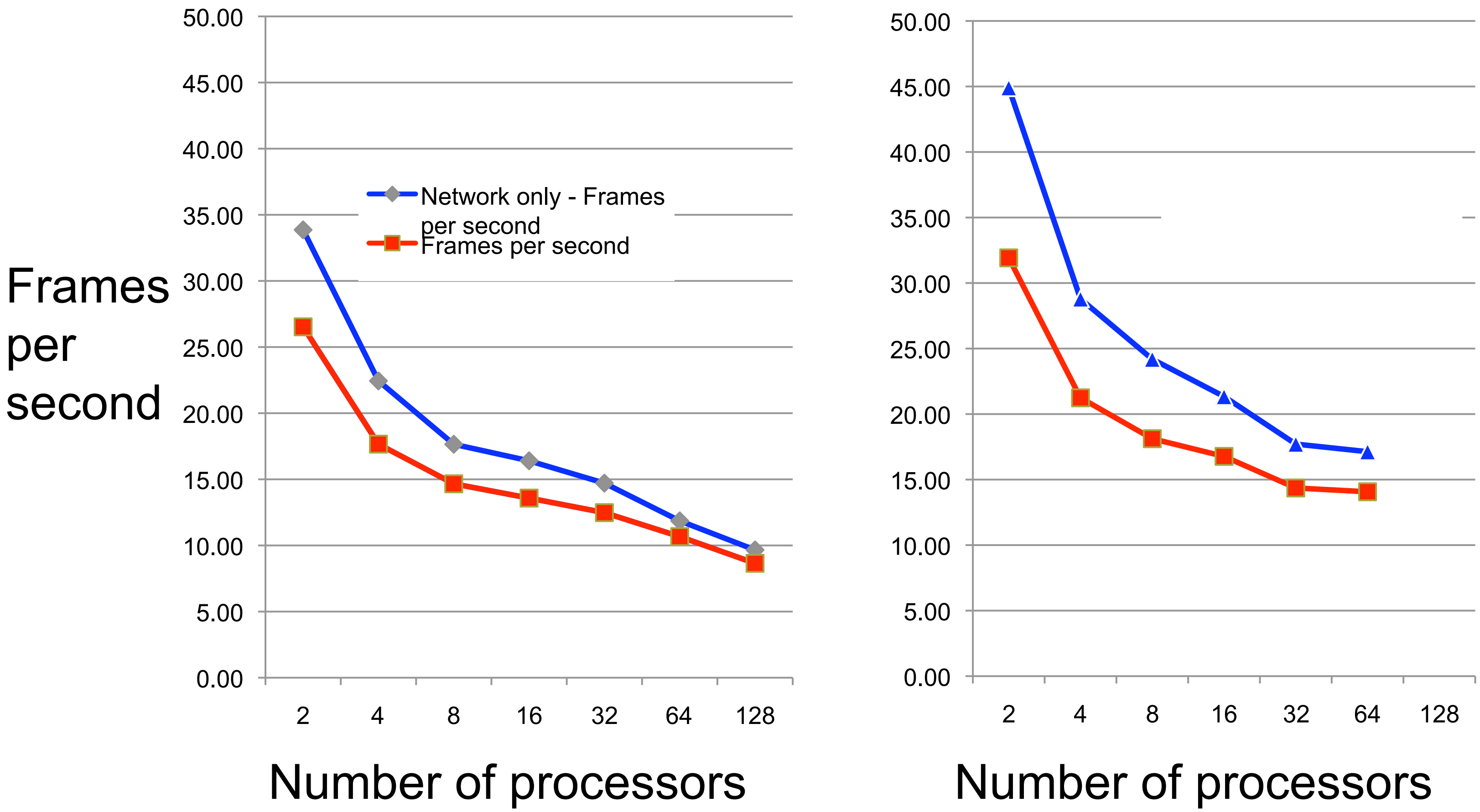
- **1 Million polygons rendering to a 1Kx1K image**

Rendering Type	Software	Architecture	Frames per second
Scan conversion	OpenGL	Nvidia Quadro FX 5600	18.6

1. Vtk GPU hardware rendering performance could be improved.

Rendering Type	Software	Architecture	Frames per second for # of cores				
			1	2	4	8	16
Scan conversion	Open GL Mesa	Multi-core (4 quad opt.)	0.7	1.2	2.0	3.2	4.6

Networking – IB-1, IB-2 compositing performance



Summary
Rendering and networking performance

- **10-15 frames per second on IB**

- **GPU-based**
  - 20 frames per second
- **CPU-based/supercomputer**
  - 5 frames per second with Mesa software rendering

- **This seems to suggest that visualization clusters are the right approach…**

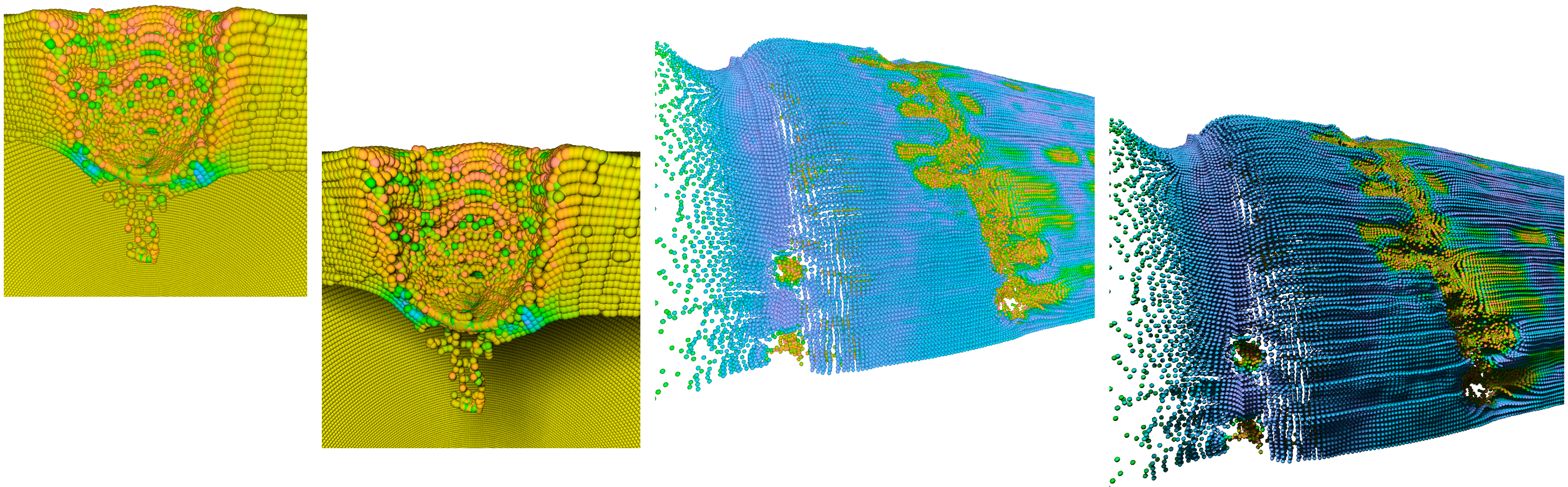
Another type of rendering

- **Scan conversion of polygons**
  - 1. OpenGL Software
    - Mesa - open-source
  - 2. OpenGL Hardware
    - Graphics cards – Nvidia

- **<u>Raytracing</u>**
  - Fast multi-core ready implementations
  - For RR - IBM's iRT software
    - Cell processor
  - . University of Utah – Manta software
    - Multi-core optimized, open-source

Why ray tracing?

- **Advanced rendering model**
  - More accurate lighting physics model
    - Shadows, reflections, refractions
  - Flexible software-based approach
  - Ability to integrate compute, analysis & rendering



Images courtesy Christiaan Gribble, Grove City College, PA (done while at Univ. of Utah)

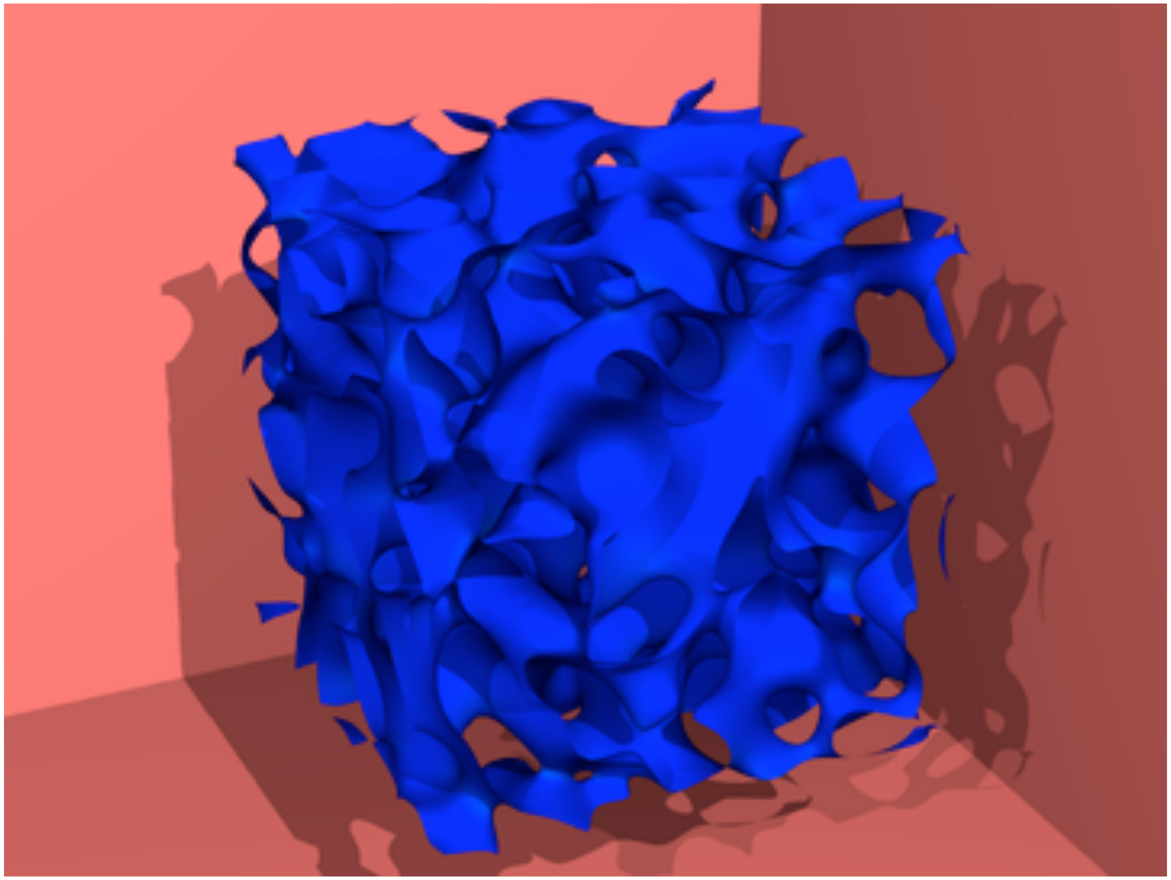
Using raytracing for rendering in vtk/PV

- **To be clear --**
- **Raytracing as a scan conversion/OpenGL replacement for parallel rendering**
  - Why? Optimized multi-core implementations available for ray-tracing
- **For this study, if there was an optimized multi-core OpenGL software we would use that:**
  - Aside - Tungsten Graphics is working on a Cell-based Mesa effort
    - Part of Gallium3D architecture
      - Their own rendering abstraction infrastructure

Using Manta raytracing for rendering in vtk/PV

- **Use vtk's rendering abstraction…**
- **Technical challenges**
  - Data representation
    - Mapping between representation of polygons in vtk and raytracer
      issues in 2D – points, lines
      - Scalar color mapping
  - Synchronization of control
    - Vtk runs in one thread and raytracer has many threads
    - Vtk and raytracer have their own event loop
      - Use callback mechanism in ray-tracer for synchronization

VtkManta serial rendering



Vtk/PV rendering performance

- **1 Million polygons rendering to a 1Kx1K image**

Rendering Type	Software	Architecture	Frames per second
Scan conversion	OpenGL	Nvidia Quadro FX 5600	18.6

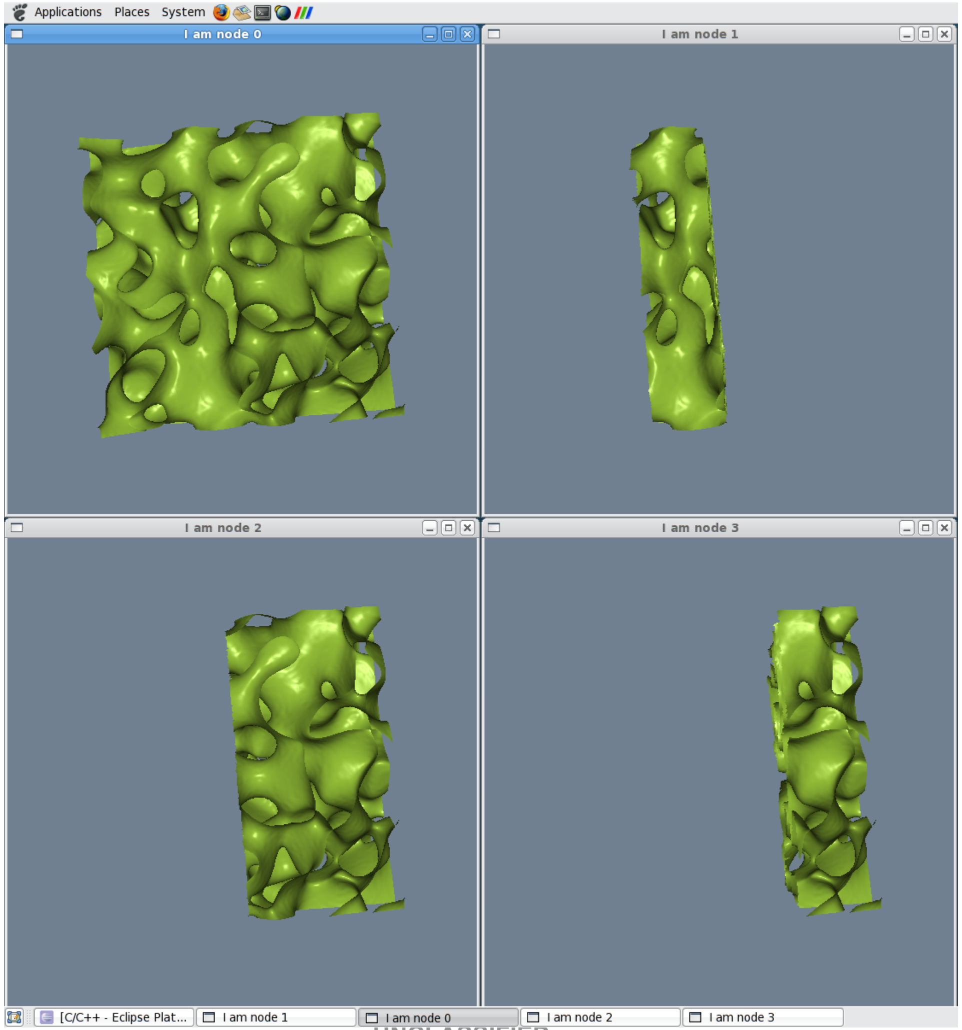
Rendering Type	Software	Architecture	Frames per second for # of cores				
			1	2	4	8	16
Scan conversion	Open GL Mesa	Multi-core (4 quad opt.)	0.7	1.2	2.0	3.2	4.6
Raytracing	Manta	Multi-core (4 quad opt.)	1.6	2.8	5.6	10.9	19.4

Parallel rendering with raytracing

- Render with raytracer on each node
- Need a depth value for sort-last compositing

- Raytracer calculates distance from eye to polygon
- At each pixel can use this value as a depth value for sort-last compositing

Parallel rendering with Manta in ParaView



UNCLASSIFIED

Summary
Rendering and networking performance

- **10-15 frames per second on IB**

- **GPU-based**
  - 20 frames per second
- **CPU-based / on supercomputer**
  - 20 frames per second with Manta raytracing (16-way multicore node)

- **This suggests trend is towards using the supercomputer and away from visualization cluster**

Future work

- iRT interface to vtk/PV in the works

- 1 Million polygons rendering to a 1Kx1K image - preliminary

Rendering Type	Software	Architecture	Frames per second
Raytracing	iRT	Cell blade (16 SPUs)	42

- Integration of IBM Cell-based ray-tracer into PV for visualization on RR platform

- Advanced ray-tracing

Conclusions

- **This preliminary study suggests that:**
  - Multi-core processors are starting to serve some of roles of traditional GPUs such as parallel rendering
  - Using fast software-based rendering methods may offer a path to utilizing our supercomputers for visualization