Data Centric Interactive Visualization of Very Large Data

Bruce D’Amora, Senior Technical Staff
Gordon Fossum, Advisory Engineer
IBM T.J. Watson Research/Data Centric Systems
Data Centric Domains

Key Domain Characteristics: Big Data, Complex Analytics, Scale and Time to Solution

Interactive, Real Time Visualization on Very Large Data in these domains is a key opportunity
Big Data Workflow Challenges

- **Performance Challenges** with Very Large Data Sets
  - Storage and I/O impact data location and access
  - Data movement bottlenecks workflow

- **Data may be subject to legal/contractual constraints**
  - Export laws, e.g. Oil and Gas Exploration in foreign countries
  - Privacy laws, e.g. Medical data

- **Data security** can be compromised by movement of data across networked systems
  - Central data location easier to secure

- **Remote users require access to visualization**
Current Strategies

- Current systems hampered by end-to-end data movement
- Visualization solutions designed primarily for workstation plus GPU platforms
- Visualization loosely coupled with simulation and analytics
Data Centric System Architecture

DCS “D0” layer: Dense Compute

DCS “D1” layer: Compute + Flash Storage

DCS “D2” layer: GPFS File Storage

Join the conversation at #OpenPOWERSummit
Big Data Capability – Workflow Enablement

Goal: Reduce total time to solution for data-centric multi-application workflow

Visualization generates 2D images for remote display
Remote users interact

Client Environment

Today:
Separate Simulation & Visualization Clusters
- Expensive Data Movement
- Visualization on x86

Opportunity:
Co-locate simulation, analytics, visualization
- Full workflow solutions via DCS systems & shared active storage
- Power+GPU connected by NVLINK
- OpenGL on POWER/GPU (Potential)

Join the conversation at #OpenPOWERSummit
Initial Prototype

IBM BlueGene/Q Server

HPC Applications

Volume Rendering Application

High Performance Compute Nodes (CN)

CN0  CN1  CN2  ....  CNn

Socket connections

High Performance I/O Nodes (ION)

Hybrid Scalable Solid-state Storage

Socket connections

IBM Power login server

Relay Server Application

Socket connection

Client Viewer

1. Volume Rendering application can run on any number of compute nodes

2. Compute nodes run a light version of Linux to minimize OS jitter

3. I/O is vectored from CN to ION. ION have PCIe attached persistent storage

4. Socket connections between CN and relay server are managed by ION

5. Relay server is the conduit between BlueGene volume rendering server and client.

6. Client viewer sends user input to and receives displayable images from relay server via multiple socket connections

Join the conversation at #OpenPOWERSummit
IBM Power 8 NVIDIA K40 System

Power 8/GPU Cluster

HPC Applications

Volume Rendering Application

High Performance Compute Nodes

Ubuntu Ubuntu Ubuntu Ubuntu...

Ubuntu LE Linux on IBM Power 8

8GB/s per FlashSystem 840 drawer
16-48TB per drawer

IBM CAPI-accelerated FlashSystem 840

High Performance I/O

IBM Power Login Node

NodeJS relay & http server

WebGL Client Browser

TCP socket connections

Fibre channel links to storage

Relay server broadcasts client state and volume renderer sends new images

Web socket connections

Client viewer sends user input to and receives displayable images from relay server via multiple socket connections

Join the conversation at #OpenPOWERSummit
Prototype Features

- Scalable rendering of volumetric data
- Dozens of gigabytes per GPU, lots of GPUs
- Four sets of cut planes, perpendicular to three world axes, and Lookat vector
- Adjustable alpha blending
  - Multiplier and exponent sliders provided
- Camera manipulation
  - Latitude and longitude of eye
  - Field of view angle
  - Re-centering object after cutplane
- Support for remote client visualization
Big Data Visualizer

**Vision**

3D arrays of data large enough that the number of data points exceeds the number of pixels available to display it by orders of magnitude.

**Implementation**

- Data can be index, index+alpha, or RGBA
- Each MPI Rank has its own subset of the data
- It extracts isoparametric surfaces, loading tiles with that data
- It uses raycasting to blend voxel colors, incorporating the previously computed surface data
- “MPI Accumulate Ranks” gather these tiles, sending them to the “top”
- Tiles are then shipped via socket to a (possibly remote) client
Isoparametric Surface

Vision
Huge data; rendered triangles would be far smaller than a pixel; why bother? Just find points of intersection along the x,y,z-aligned directions. Voxels that intersect surface form a very sparse subset of all voxels.

Implementation
• Precompute all intersections and store location and normal info.
• CUDA kernel call processes these intersections, adding colors to tiles.
Example of Isoparametric points

(Derived from the open-source female Visible Human dataset)
Raycasting

**Vision**
Previously computed opaque isoparametric points represent "starting color and depth" for a raycasted blend operation working back to front in each pixel for each tile that intersects the brick associated with this MPI Rank.

**Implementation**
For each intersecting tile, make a CUDA kernel call
Each thread is responsible for one pixel in that tile
If the thread's ray pierces the brick {
   Read samples from back to front, blending color
   // note "back" depends on possible point data in that pixel
}
Opaque point limits Raycasting

Intersects an opaque isoparametric point

No opaque points: raycast through entire brick

Eye point

2D tile on screen
Accumulation

Vision
Oct-tree hierarchy of MPI ranks
• Each leaf comprises a “compute rank”
• Each internal node comprises an “accumulate rank” with up to 8 children
• Children are guaranteed to be adjacent in the space of the large array

Implementation
• Tiles from compute ranks are sent to their immediate parent
• Each accumulate rank knows where the eye point is
• Children are sorted based on eye location
• Tiles are blended in correct order
• Result is sent to parent
• Root accumulate ranks (eight of them, to facilitate socket efficiency) send tiles via socket to client for visualization within client graphics framework
Oct-tree structure (shown as Quad-tree)

Root ranks

Second-level accumulate ranks

First-level accumulate ranks

Compute ranks

(some connections are not shown to keep chart simpler)
Example of blended image

(Derived from the open-source female Visible Human dataset)
Future Work

- Utilize CAPI accelerated FLASH IO to enable workflow data transfers
- Graph visualization
- Merged Scientific and Info Visualization
- Latency mitigation
- Dynamic reallocation of aggregation ranks to more efficiently scale with viewing and window changes
- Improving IO performance by rearranging data into tiles matching page buffer size