Cornell University
Center for Advanced Computing

Parallel Large-Scale Visualization

Aaron Birkland
Cornell Center for Advanced Computing

Data Analysis on Ranger
January 2012

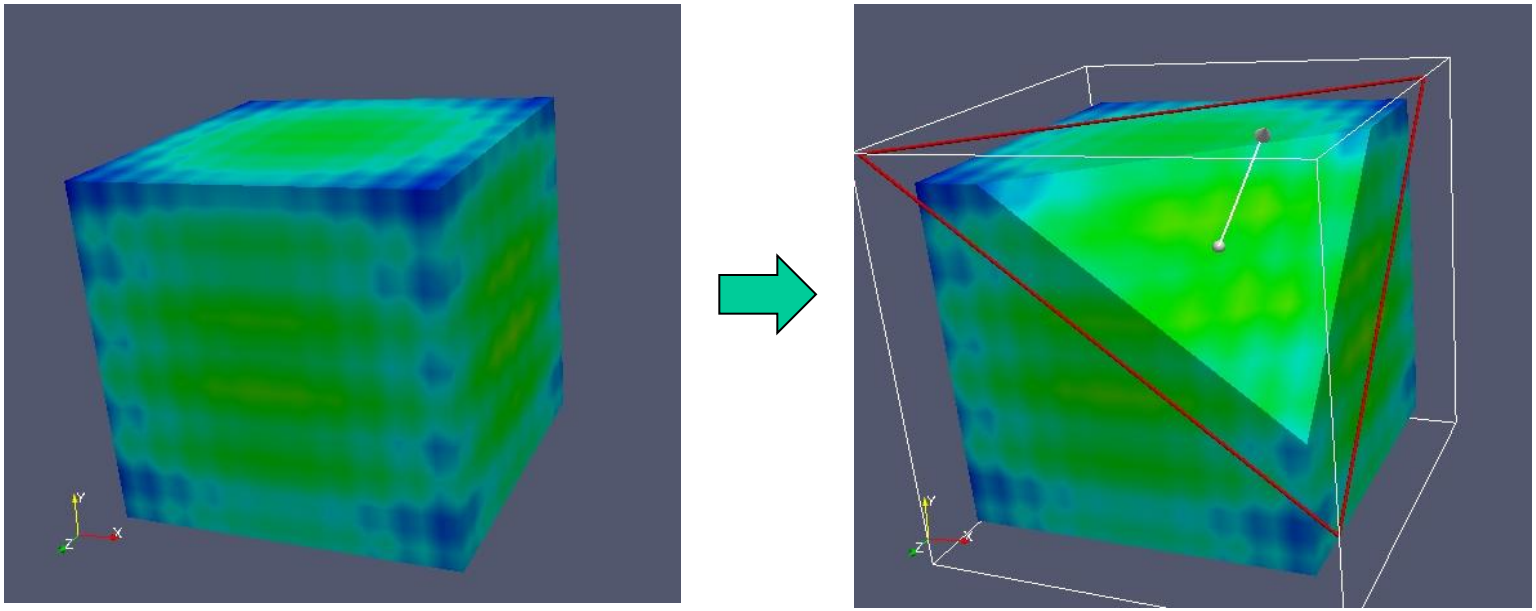
Parallel Visualization

- Why? Performance
  - Processing may be too slow on one CPU
    - Interactive visualization requires real-time frame rates
    - Use lots of CPUs
    - Shared-memory/multicore *or* distributed
  - Data may be too big for available node
    - Virtual memory works, but paging is slow
    - Use lots of nodes to increase physical memory size
    - Big shared-memory/multicore scaling is costly ($/CPU)

Increase interactivity or feasibility

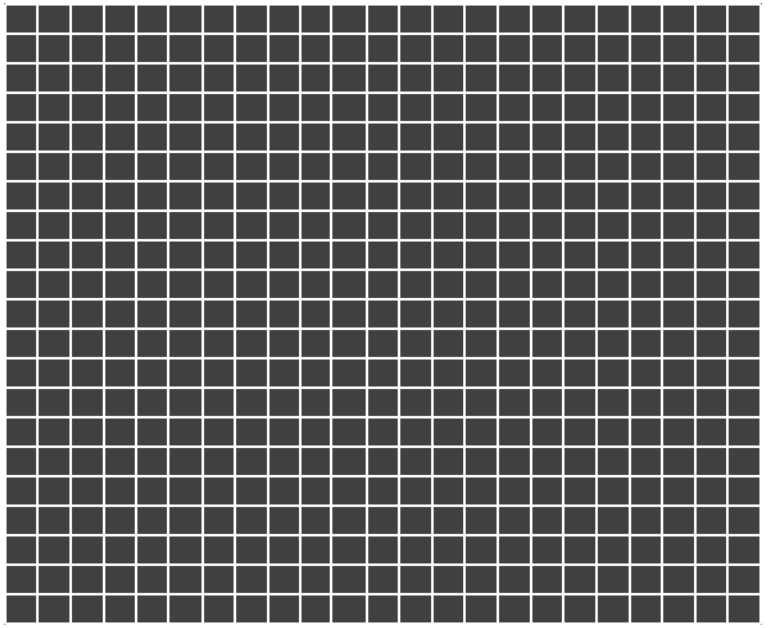
Memory Utilization

- Some visualization techniques cause memory use to skyrocket!



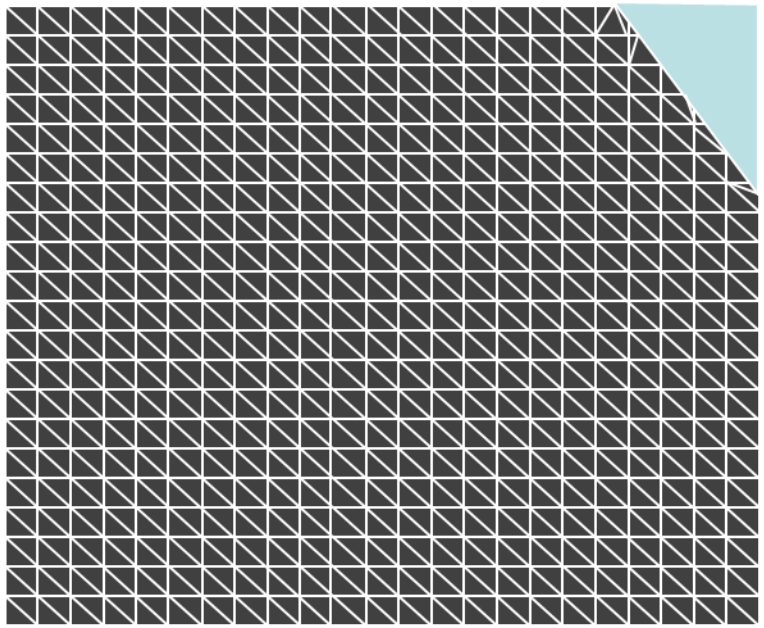
Memory Utilization: Regular Grids

- Specified by:
  - (x,y,z) origin
  - (nx, ny, nz) counts
  - Data array
- Requires very little memory



Memory Utilization: Regular Grids

- Chop off corner -> need an unstructured grid to represent data points
- Specified by
  - Explicit list of vertices
  - Explicit list of triangles
- Memory use can go up *many* times

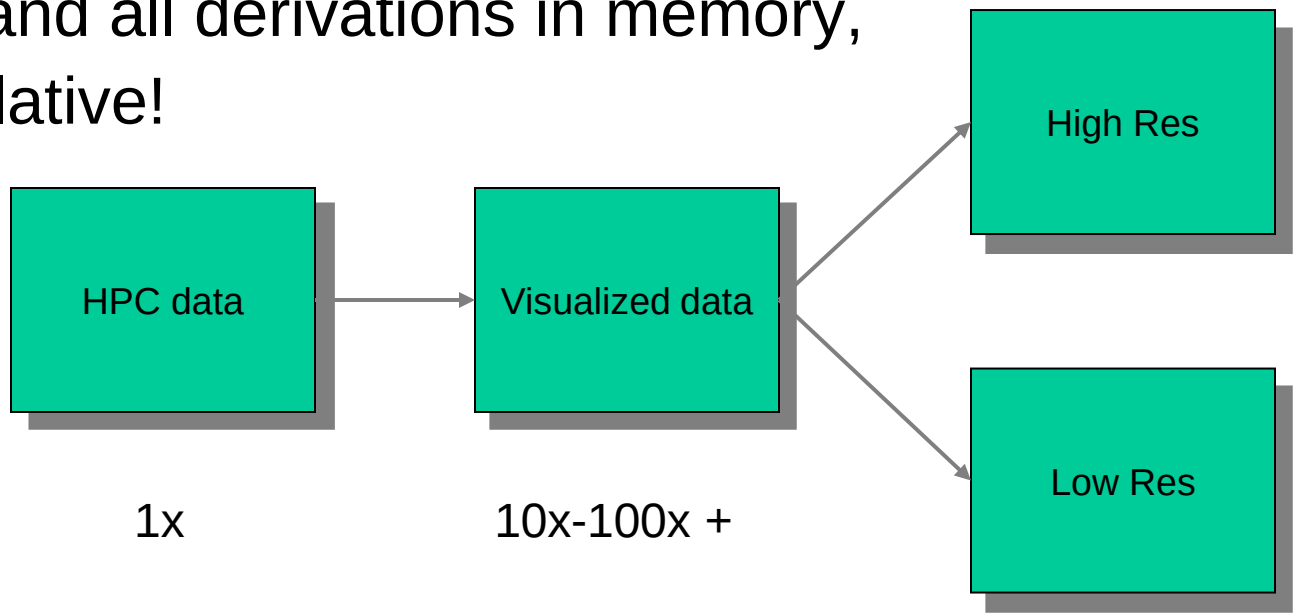


Memory Utilization: examples

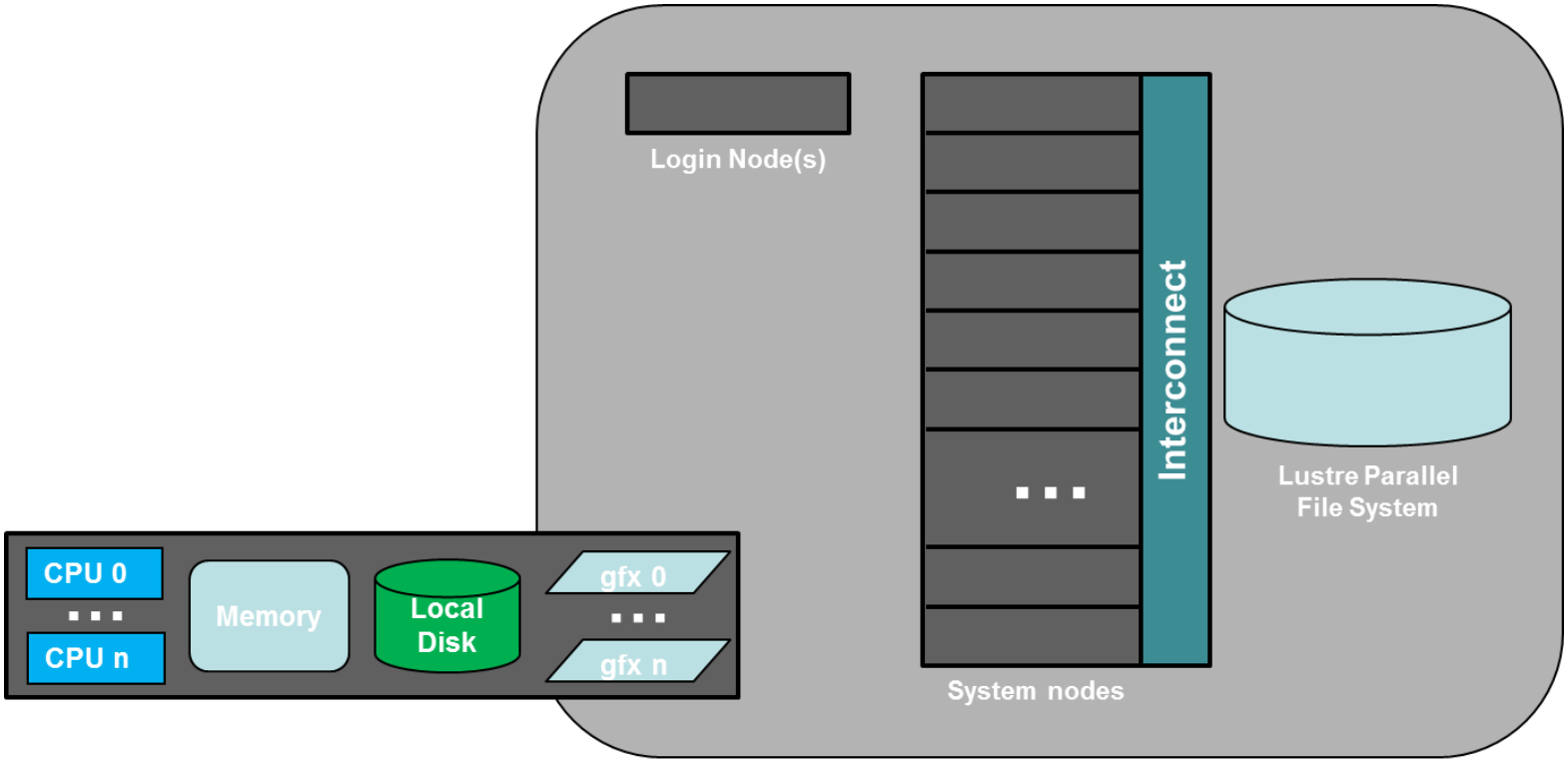
- Mummy.vtk:
  - Structured Grid
  - (128x128x128)
  - 2MB raw data
- Contour: 7MB
  - Polygonal Mesh
- Slice of Contour: .1MB
- Tetrahedralize: 520MB!!
  - Unstructured Grid
  - Data points -> Tetrahedrons

Visualization scales with HPC

- Large data produced by large simulations require large visualization machines and produce large visualization results
- Data and all derivations in memory, cumulative!



TACC Parallel Visualization Systems



Spur, Longhorn

TACC Parallel Visualization Systems

- Spur
  - 8 nodes, 1 TB total aggregate memory
  - 16 cores per node (128 total)
  - 4 GPUs per node (32 total)
  - DDR (dual data rate) Infiniband interconnect

- Longhorn
  - 256 nodes, 14.5 TB total aggregate memory!
  - 8 cores per node (2048 total)
  - 2 GPUs per node (512 total)
  - QDR (quad data rate) Infiniband interconnect

Longhorn Configuration

- 256 Dell Quad Core Intel Nehalem Nodes
  - 240 Nodes
    - Dual socket, quad core per socket: 8 cores/node
    - 48 GB shared memory/node (6 GB/core)
    - 73 GB Local Disk
    - 2 Nvidia GPUs/node (FX 5800 - 4GB RAM)
  - 16 Nodes
    - Dual socket, quad core per socket: 8 cores/node
    - 144 GB shared memory/node (18 GB/core)
    - 73 GB Local Disk
    - 2 Nvidia GPUs/node (FX 5800 – 4GB RAM)
- Direct Connection to Ranger's Lustre Parallel File System
- 10G Connection to 210 TB Local Lustre Parallel File System

Parallel Visualization: Task Parallelism

- Divide the overall workflow into tasks that can happen independently and, hence, concurrently
- Usually does not scale well in practice

Timesteps

Processes

	1	2	3	4	5
1		Read file 1	Isosurface 1	Cut Plane 1	
2			Read file 2	Streamlines 2	Render
3	Read file 3	Triangulate 3	Decimate 3	Glyph 3	

Parallel Visualization: Pipeline Parallelism

- Useful when processes have different/specialized resources
- Bottlenecks if one stage is particularly slow

Timesteps

Processes	1	2	3	4	5
1	Read file 1	Read file 2	Read File 3		
2		Isosurface 1	Isosurface 2	Isosurface 3	
3			Render 1	Render 2	Render 3

Parallel Algorithms: Data Parallelism

Data parallelism

Data set is partitioned among the processes and all processes execute same operations on the data.

Scales well as long as the data and operations can be decomposed.

Timesteps

Processes	1	2	3
1	Read partition 1	Isosurface partition 1	Render partition 1
2	Read partition 2	Isosurface partition 2	Render partition 2
3	Read partition 3	Isosurface partition 2	Render partition 3

Parallel Algorithms: What doesn't work

- Streamlines!
  - Not data-parallel
  - Partial streamlines must be passed from processor to processor as the streamline moves from partition to partition
  - No more parallelism available than the number of streamlines!
  - If >1 streamlines pass through the same partition, you may not even get that

Parallel Data Management

- *Data must be distributed* across parallel processes to take advantage of resources
- *Explicit* Parallel formats use separate files for partitions
- *Implicit* parallel formats have a structure where data partitions can be deduced from file structure
  - .vtk legacy, silo, raw
- *Non-parallel* formats need to be read serially and distributed in order to be used in parallel
  - Overhead!
  - Vtk xml formats (.vtu, .vti, etc)

Parallel Data Management

- Read the manual!
  - Vis software has varying support for file formats
  - True parallel I/O may not be implemented for some formats
  - Vis software will try to “hide” it’s failings
- Example: ParaView (from FAQ)
  - *Currently there are only a few readers that truly work in parallel: VTK files (not legacy), partitioned legacy VTK files, ParaView data files, HDF5 files, EnSight master server files, and raw (binary) files can be read in parallel. For demonstration purposes, ParaView will distribute pieces of a data set when the reader cannot. Unfortunatley, this is an inefficient process.*

Rendering

- Many graphics primitives spread out over nodes
- Rendering solutions
  - 1. Gather triangles onto one node, render there
    - Best when there's not a lot of data to render
  - 2. Render triangles in place, gather and Z-composite the results
    - Best when there *is* a lot of data to render
    - Overhead is *almost* independent of data size
- VisIt and ParaView both do it both ways
  - User controls threshold, but both apps aim for reasonable defaults