Large-Scale Isosurfacing on a Distrubuted GPU Cluster

Byungil Jeong, Greg Abram,
Greg Johnson and Paul Navrátil

Motivation

- Large, high-resolution datasets require significant memory to visualize
- Current cluster-based machines provide memory across many nodes and many GPUs

- Need *distributed* isosurfacer to match machine topology
- Need *fast* isosurfacer for interactivity

Project Goals

- Optimize the marching cubes isosurfacing algorithm provided with the CUDA SDK
- Build framework for interactive visualization of large data on distributed GPU clusters.
  - MPI-based communication across nodes
  - Post-isosurface image composition
- Expand framework to other GPU-based visualization techniques

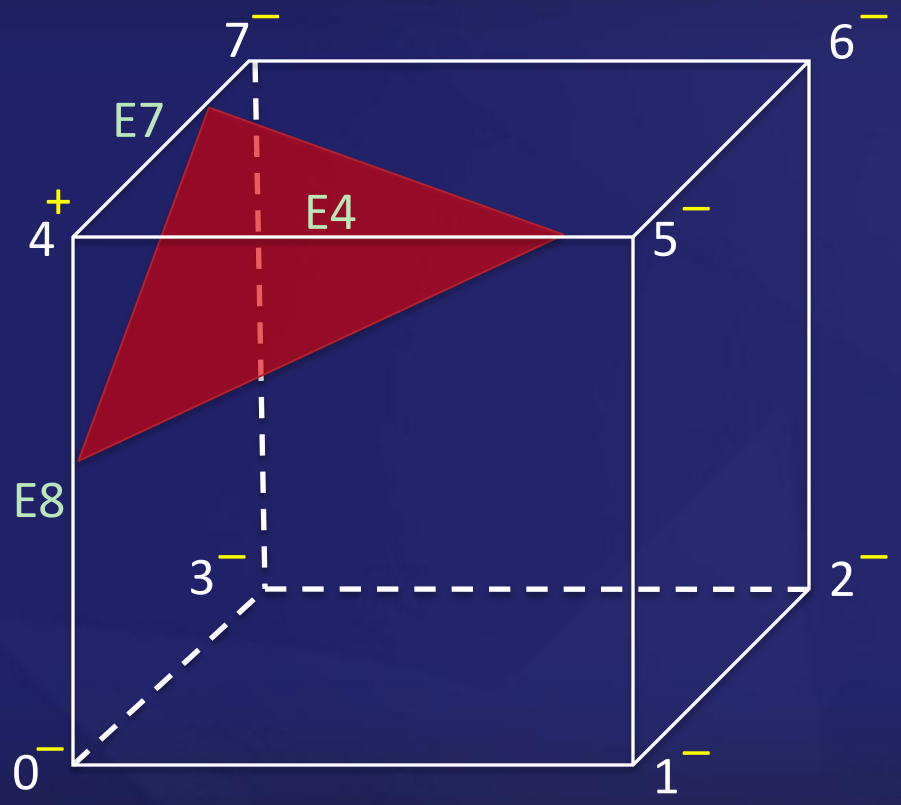
Outline

- Background
- CUDA SDK implementation
- Single-kernel implementation
- Future work
- Conclusions

Background

- Isosurface extraction:
  - Identify constant-value surfaces with a given volume
  - Creates sharp boundaries for given values (similar to an impulse-shaped transfer function)
  - Produces geometry that can be reused
- Marching Cubes algorithm
  - Independent operation for each voxel
  - Suitable for parallel processing and GPGPU
  - Scalable for large-scale volume data

Marching Cube Algorithm

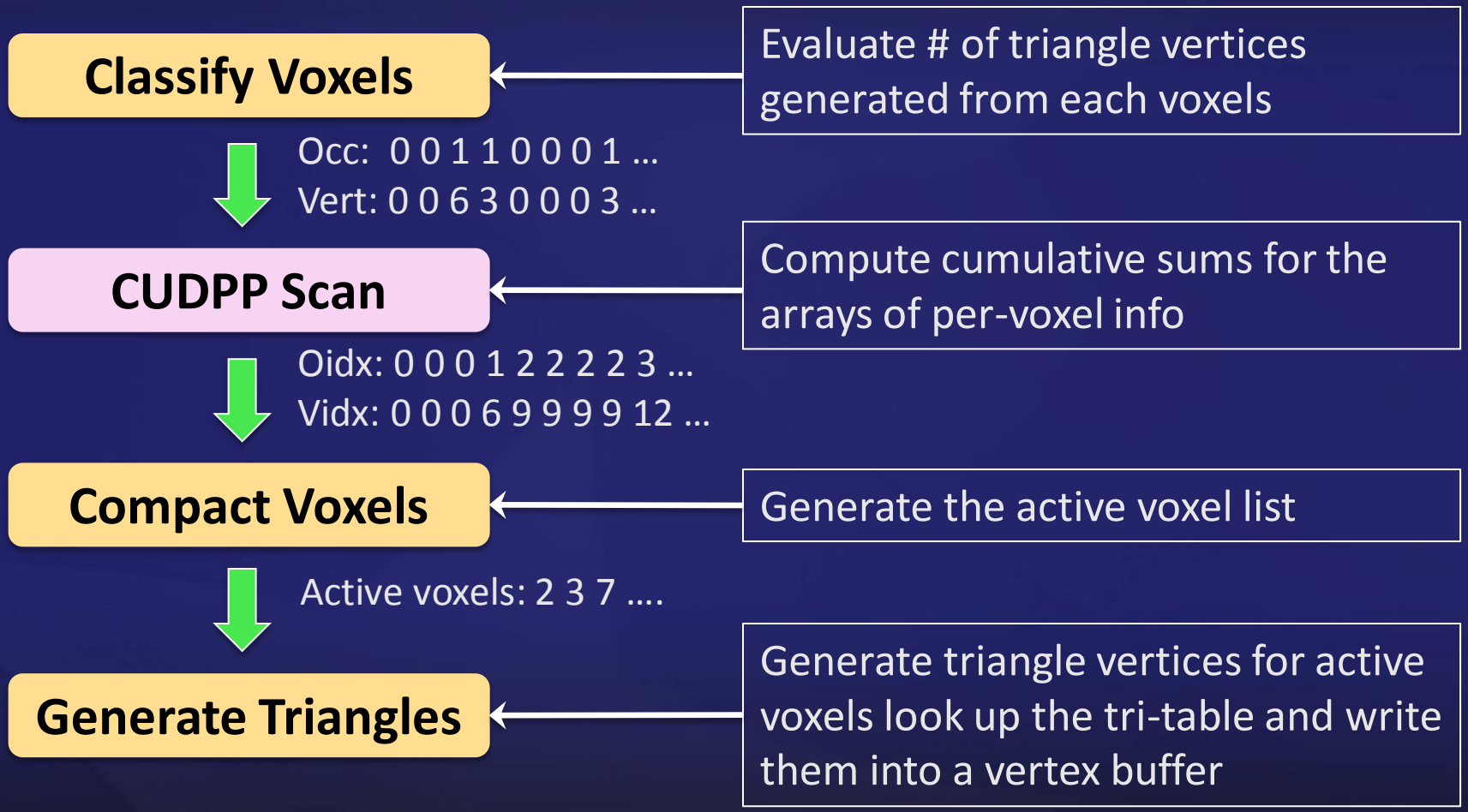


Tri-Table

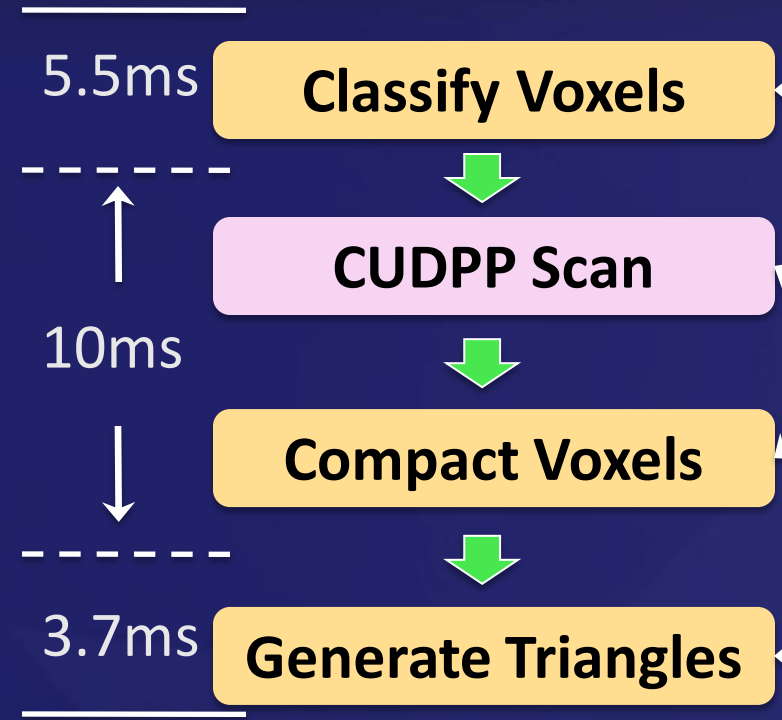
Cube Index	Edges
0x00	X
0x01	E0, E3, E8, X
:	:
:	:
0xEF	E4, E7, E8, X
0xFF	X

+ Data Value is bigger than iso-value, the corresponding bit of cube index = 0
– Data Value is smaller than iso-value, the corresponding bit of cube index = 1

Marching Cubes (CUDA SDK impl)



Motivation of Enhancement



Threads for adjacent voxels don't share volume data (duplicated read)

- Global memory size of O(total voxel #)
- Limit volume size to 256^3/GPU (4GB)
- About 50% of execution time

- Recalculate cube indices
- Calculate triangle vertices for all 12 edges and store them in shared memory

- Redesign this as a single kernel ?
- Is there a way to reduce the device memory usage?

Single Kernel Approach

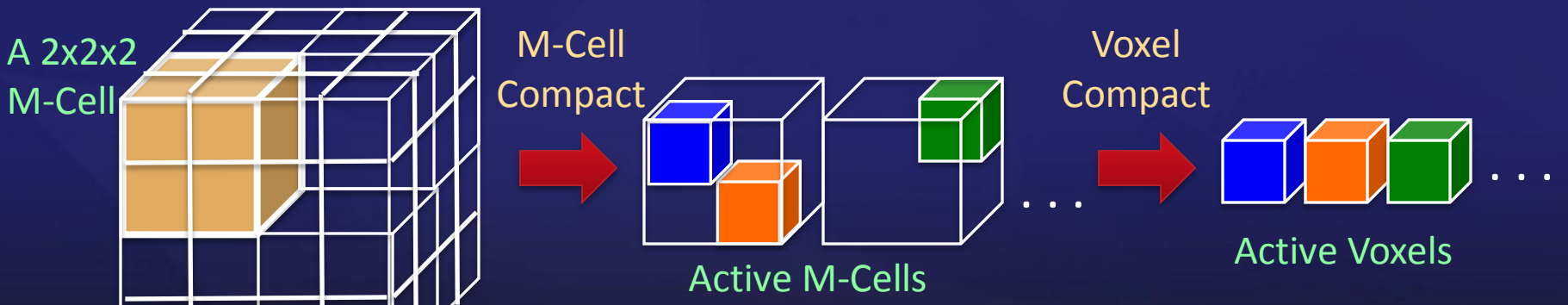
- Triangles generated in parallel but written into a single vertex buffer
  - Each thread keeps separate triangle count, cannot see other thread counts
  - Can use atomicAdd() to keep global triangle count,
    but can result in expensive synchronization
  - Keep atomicAdd location in shared memory to minimize latency

- Cannot know the exact vertex buffer size before runtime
  - Can conservatively pre-allocate buffer space for each voxel
    Can far exceed actual memory requirements, about double for our tests

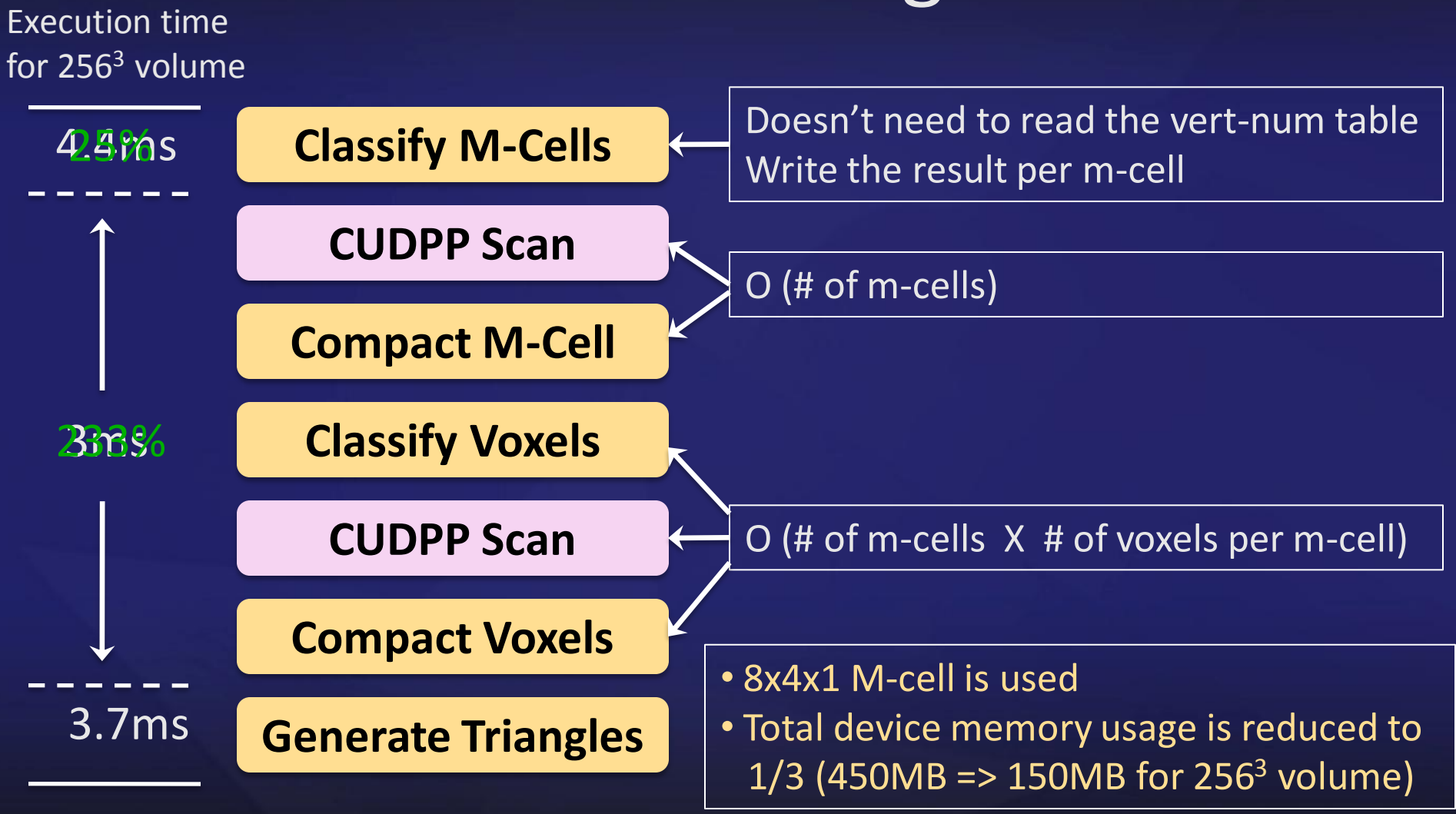
- Achieved similar performance to the CUDA SDK version

Meta-Cell Filtering Approach

- Filter out groups of inactive voxels (inactive meta-cells)
- Operate only on voxels that contain an isosurface
- Significantly reduces the sizes of arrays for CUDPP scan
  - O(# of voxels) => O(# of meta-cells) or O(total # of voxels in active meta-cells)
- Reduce the execution time of voxel classification and CUDPP scans.
  - Meta-cell filtering time < this execution time reduction?

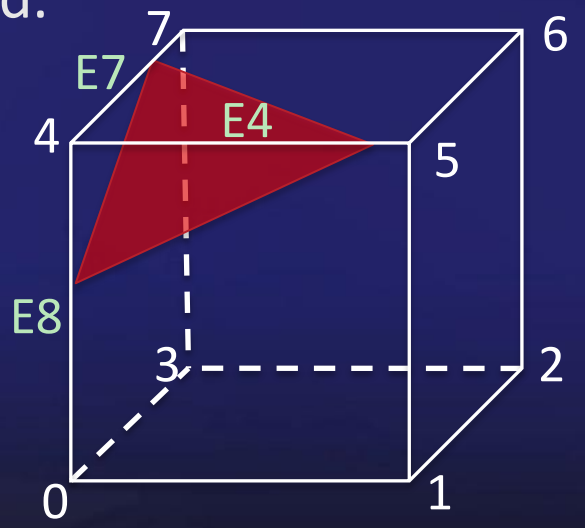


Meta-Cell Filtering Result

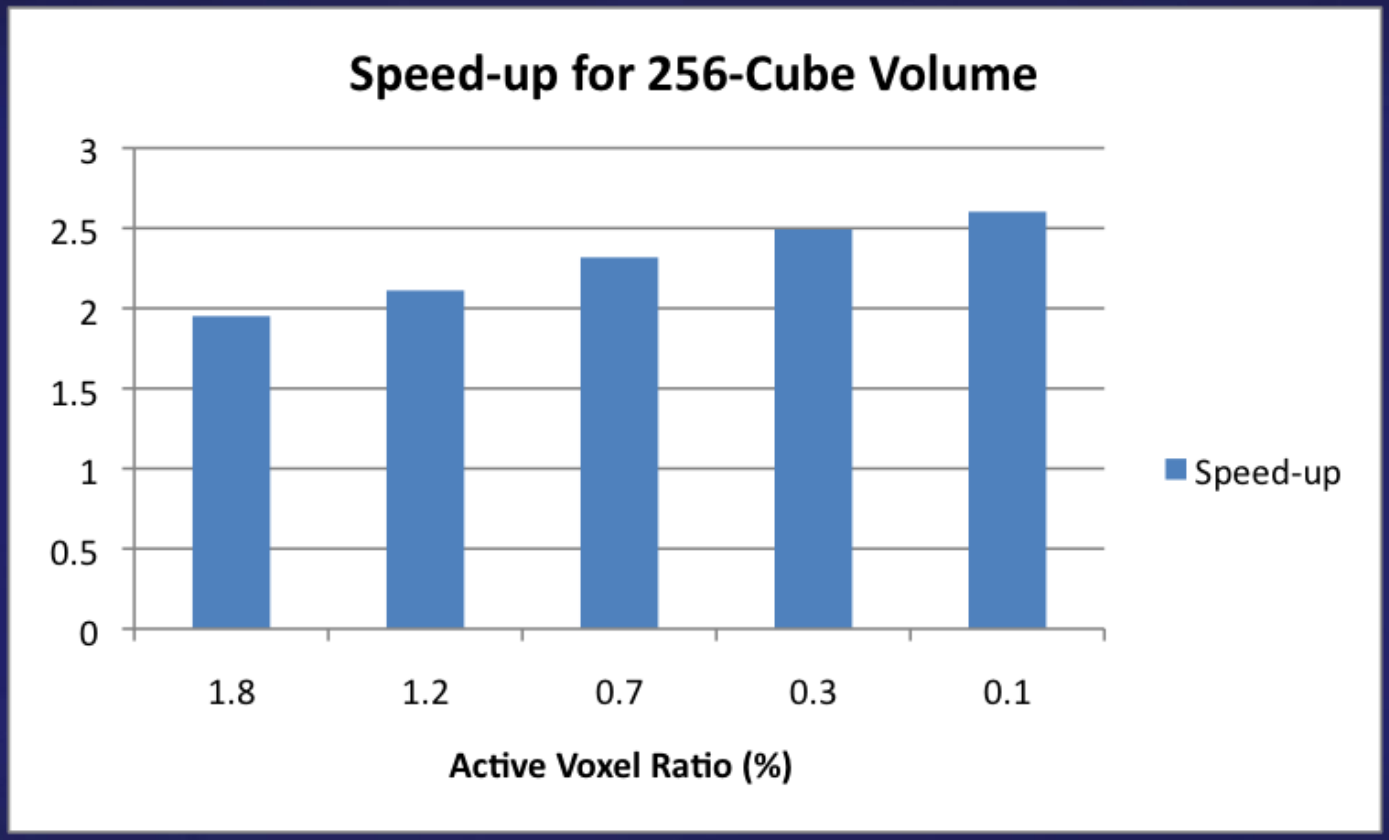


Enhancing GenerateTriangle Kernel

- Higher register count than other kernels
  - Lower occupancy
  - The CUDA SDK version uses shared memory to store the interpolated vertices on all 12 edges of a voxel.
  - Each voxel generates 1 – 5 triangles.
  - Many interpolated vertices are not used.
- Interpolate the vertices actually needed for triangle generation.
- Use shared memory (instead of registers) for reading volume data.
- Execution time is reduced to 2.8ms (30% speed up).

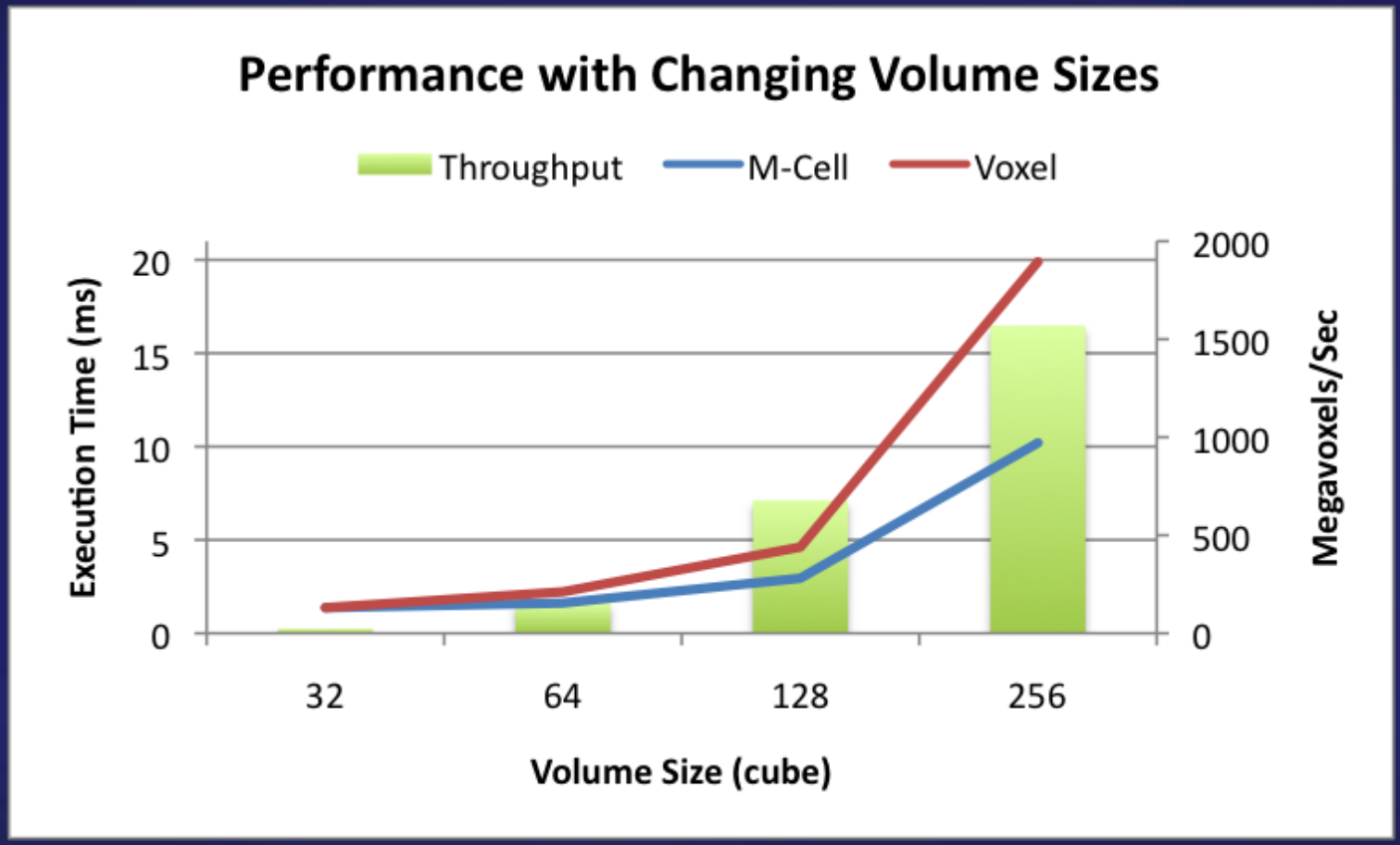


Performance Evaluation on A Single GPU



Speed up = (execution time of SDK version) / (execution time of enhanced version)
Active Voxel Ratio = (# of Active Voxels x 100) / Total # of Voxels

Performance Evaluation on A Single GPU

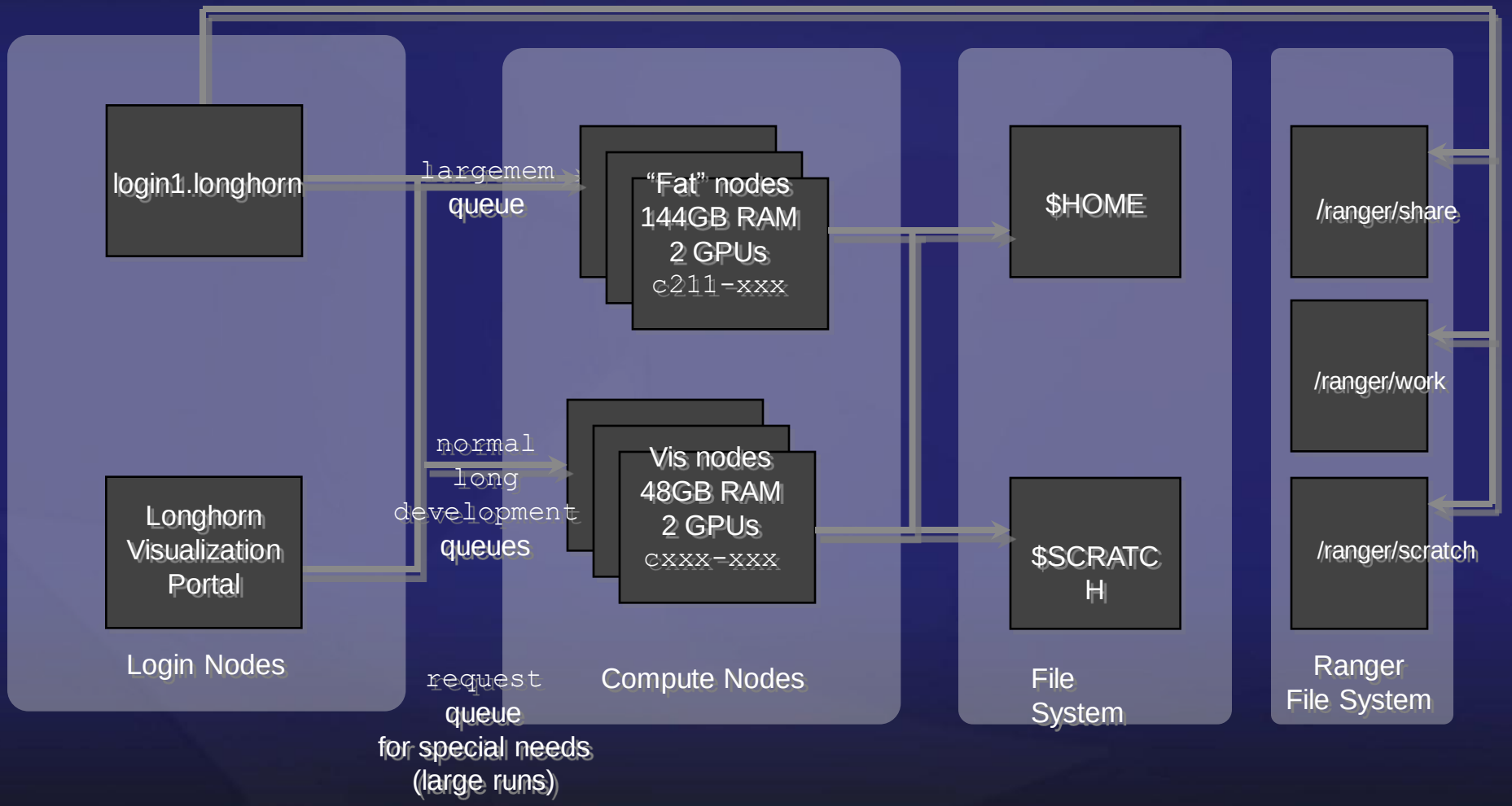


Voxel: execution time of SDK version, M-Cell : execution time of enhanced version
Throughput: # of voxels processed per second (enhanced version)

Longhorn Configuration

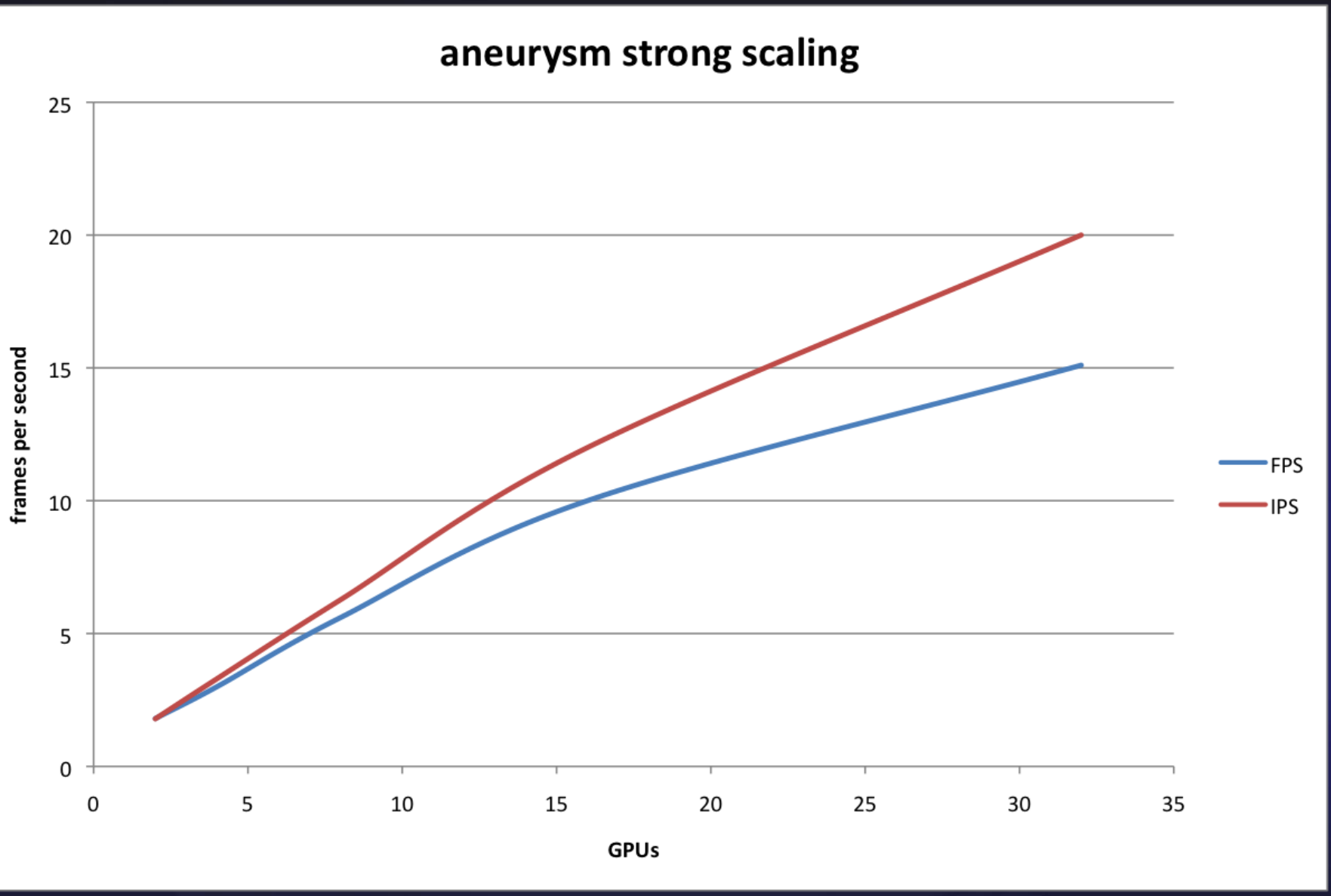
- 256 Dell Dual-Socket Quad Core Intel Nehalem Nodes
  - 240 Nodes
    - Dual socket, quad core per socket: 8 cores/node
    - 48 GB shared memory/node (6 GB/core)
    - 73 GB Local Disk
    - 2 Nvidia GPUs/node (FX 5800 - 4GB RAM)
  - 16 Nodes
    - Dual socket, quad core per socket: 8 cores/node
 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

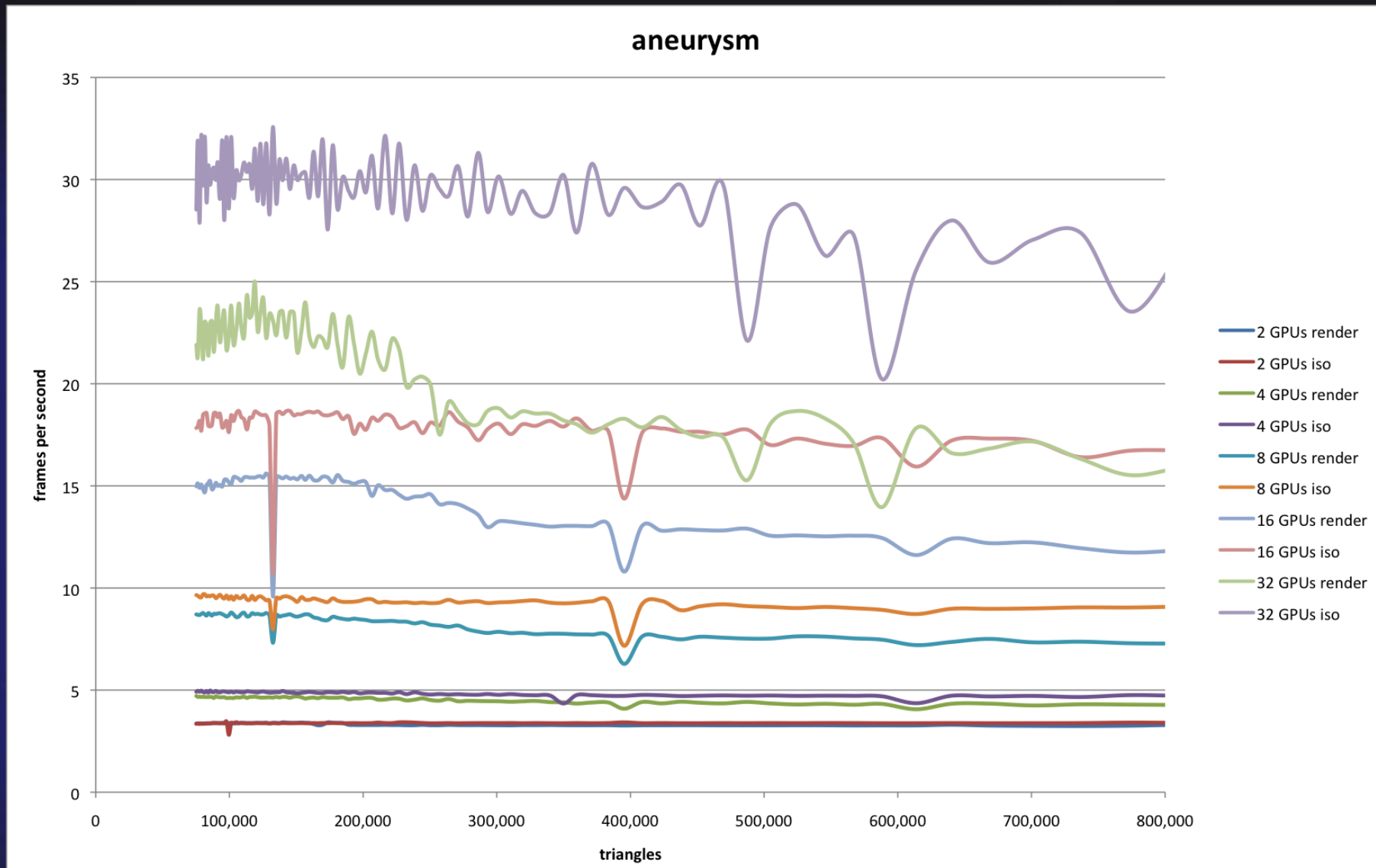
Longhorn topology

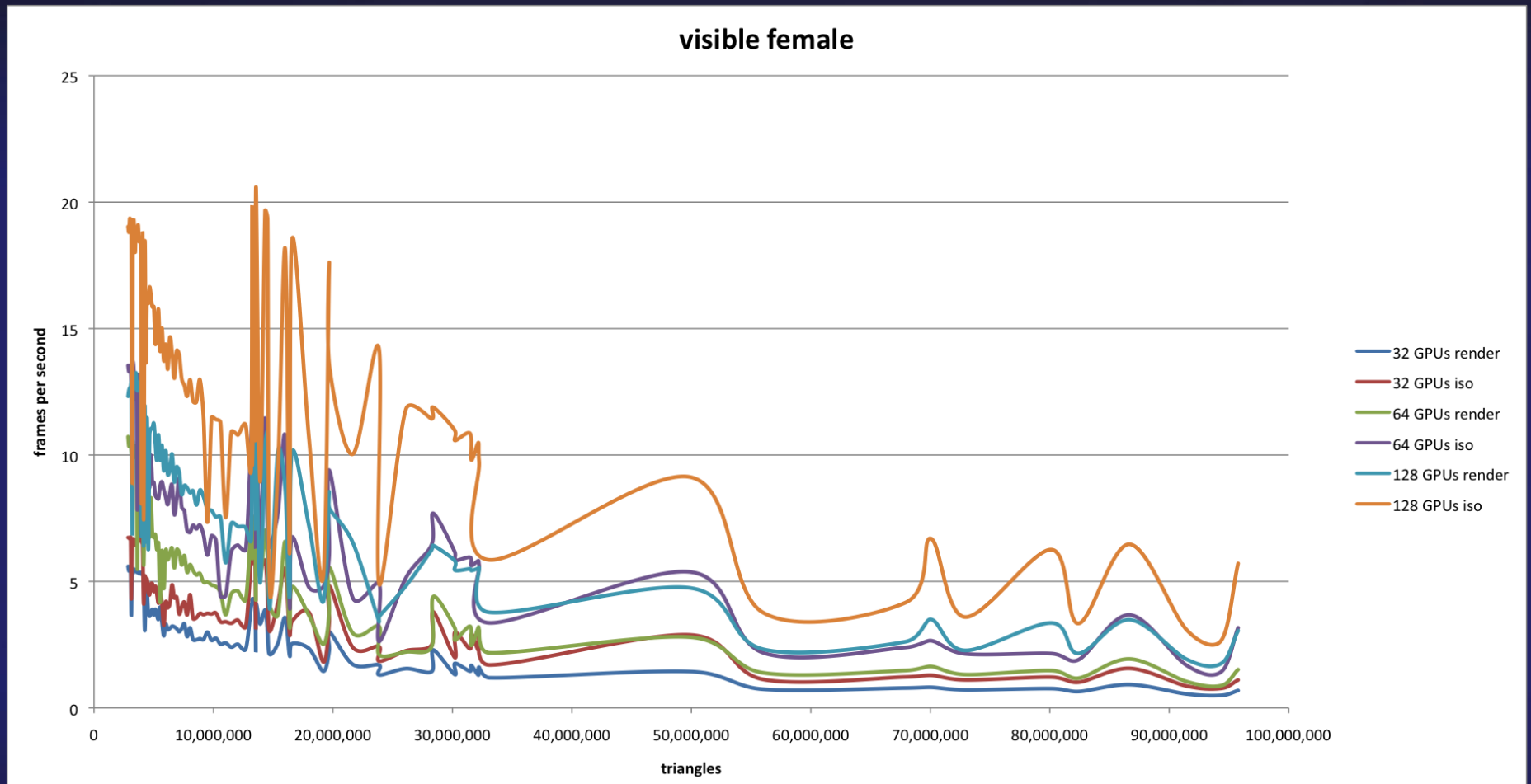


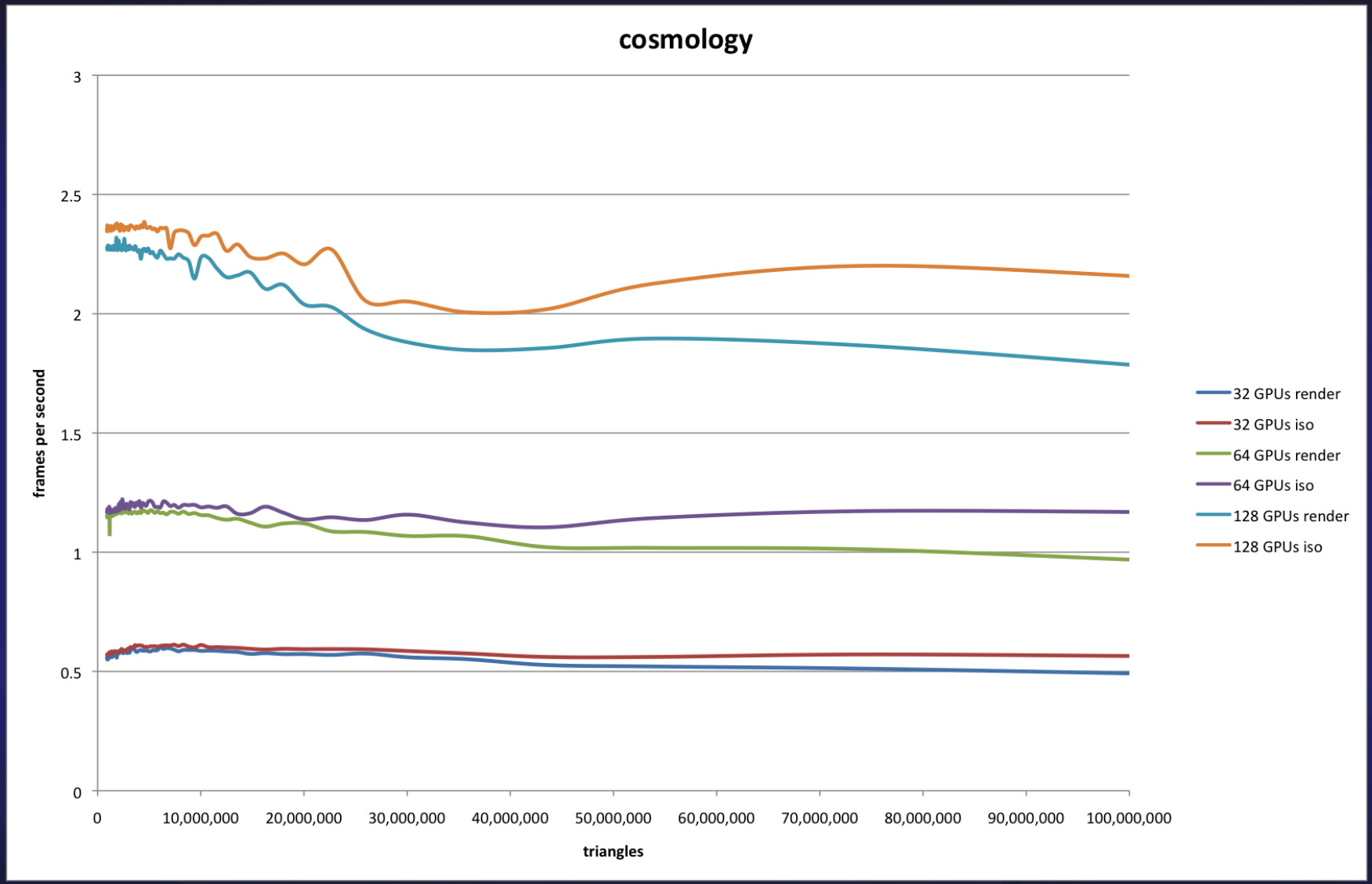
Datasets

- Aneurysm
  - CT scan of brain aneurysm
  - 512^3 broken into 64 blocks
- Visible Female
  - CT scan of female cadaver
  - 512×512×1792 broken into 28 blocks
- Cosmology
  - Simulation of dark matter density
  - 2048^3 broken into 256 blocks


aneurysm strong scaling
frames per second
25
20
15
10
5
0
0
5
10
15
20
25
30
35
GPUs
FPS
IPS


aneurysm
35
30
25
20
15
10
5
0
frames per second
0
100,000
200,000
300,000
400,000
500,000
600,000
700,000
800,000
triangles
2 GPUs render
2 GPUs iso
4 GPUs render
4 GPUs iso
8 GPUs render
8 GPUs iso
16 GPUs render
16 GPUs iso
32 GPUs render
32 GPUs iso


visible female
frames per second
25
20
15
10
5
0
0
10,000,000
20,000,000
30,000,000
40,000,000
50,000,000
60,000,000
70,000,000
80,000,000
90,000,000
100,000,000
triangles
32 GPUs render
32 GPUs iso
64 GPUs render
64 GPUs iso
128 GPUs render
128 GPUs iso


cosmology
frames per second
3
2.5
2
1.5
1
0.5
0
0
10,000,000
20,000,000
30,000,000
40,000,000
50,000,000
60,000,000
70,000,000
80,000,000
90,000,000
100,000,000
triangles
32 GPUs render
32 GPUs iso
64 GPUs render
64 GPUs iso
128 GPUs render
128 GPUs iso

Related Work

- GPU-based isosurfacers:
  - [TSD07] Real-Time Isosurface Extraction Using the GPU Programmable Geometry Pipeline
  - [WC09] Isosurface Extracton and View-Dependent Filtering from Time-Varying Fields Using Persistent Time-Octree (PTOT)
  - [LCD09] Fast Isosurface Rendering on a GPU by Cell Rasterization
  - [MSC10] Load-Balanced Isosurfacing on Multi-GPU Clusters

Future Work

- Performance evaluation of the enhanced isosurfacer for real datasets with various active voxel ratio and distribution.
- Integrate the enhanced isosurfacer with our parallel rendering framework:
  - SDK isosurfacer with the framework: 2.26 sec using 16 GPUs, 4.57 sec using 8 GPUs for $2K^3$ volume
  - Overlapping volume data download and isosurfacing kernels
- Consider the whole visualization workflow
  - Longhorn file system's peak throughput: 5.8GB/sec
  - Enhanced isosurfacer's single GPU throughput: 6GB/sec
  - Load only active meta-cells from the file system using pre-processed metadata
  - In-situ Visualization: directly fetch data from simulation to visualization

Conclusion

- Longhorn is the world's largest GPU-accelerated visualization cluster enabling extremely large-scale scientific visualization and data analysis.

- We are investigating GPU-based approaches to interactively visualize extremely large-scale data using large-scale GPU clusters such as Longhorn.

Access to Longhorn

- NSF TeraGrid allocation
  - Project lead should have academic affiliation
  - Start-up requests are generally accepted, larger Research requests are peer reviewed.
  - https://portal.teragrid.org/allocations-overview
- TACC STAR membership
  - TACC industrial affiliates program
  - http://www.tacc.utexas.edu/partnerships/industrial-partners/
  - Contact Melyssa Fratkin mfratkin@tacc.utexas.edu

More Information

- http://www.tacc.utexas.edu/resources/visualization/
- http://services.tacc.utexas.edu/index.php/longhorn-user-guide
- https://portal.longhorn.tacc.utexas.edu/

- bijeong@tacc.utexas.edu
  pnav@tacc.utexas.edu

Thank You!