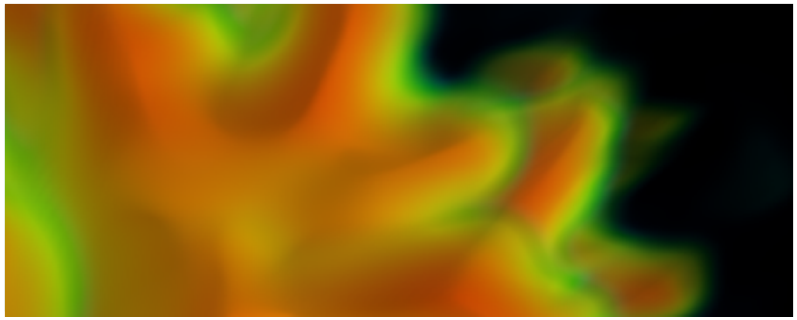


MPI-hybrid Parallelism for Volume Rendering on Large, Multi-core Systems

E. Wes Bethel, LBNL

Astronum 2010

17 June 2010

San Diego, CA, USA

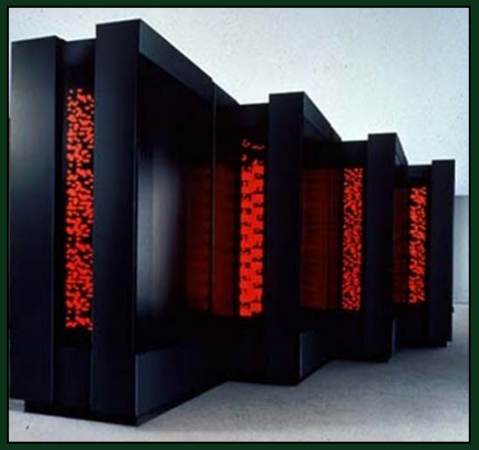
Hybrid Parallelism for Volume Rendering on Large, Multi-core Platforms

Overview

- Traditional approaches for implementing parallel visualization may not work well on future multi-core platforms: 100-1000 cores per chip.
- Hybrid-parallelism blends distributed- and shared-memory concepts.
- How well does hybrid-parallelism work for volume rendering at extreme concurrency?
- Experiment to compare performance shows favorable characteristics of hybrid-parallel, especially at very high concurrency.

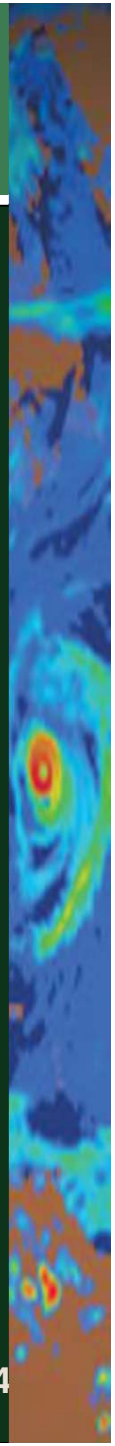
Parallelism

- Mid 1970s-present:
  - Vector machines: Cray 1 ... NEC SX
  - Vectorizing Fortran compilers help optimize a[i]=b[i]*x+c.
- Early 1990s-present:
  - The rise of the MPP based on the commodity microprocessor. Cray T3D, TM CM1, CM2, CM5, etc.
  - Message Passing Interface (MPI) becomes the gold standard for building/running parallel codes on MPPs.
  - Using MPI is like writing assembler: you have to do everything.
- Mid 2000s-present.
  - Rise of the multi-core CPU, GPU. AMD Opteron, Intel Nehalem, Sony Cell BE, NVIDIA G80, etc.
  - Large supercomputers comprised of lots of multi-core CPUs.
  - Shared memory programming on a chip: pthreads, OpenMP; data parallel languages (CUDA); global shared memory languages (UPC) and utilities (CAF).



State of Parallelism in Scientific Computing

- Most production codes written using MPI, vendor MPI implementations optimized for their architecture.
- HPC community wondering how well MPI will scale to high concurrency, particularly on 100-core CPUs.
- What to do?
  - Some alternatives: data parallel languages (CUDA), PGAS languages (UPC), global shared memory (CAF).
  - Various research projects explore different aspects of this space:
    - Chombo implementation in Titanium.
    - Autotuning work for multi-core platforms.
    - Distributed memory, multi-core (hybrid parallelism).



Related work in Hybrid Parallelism

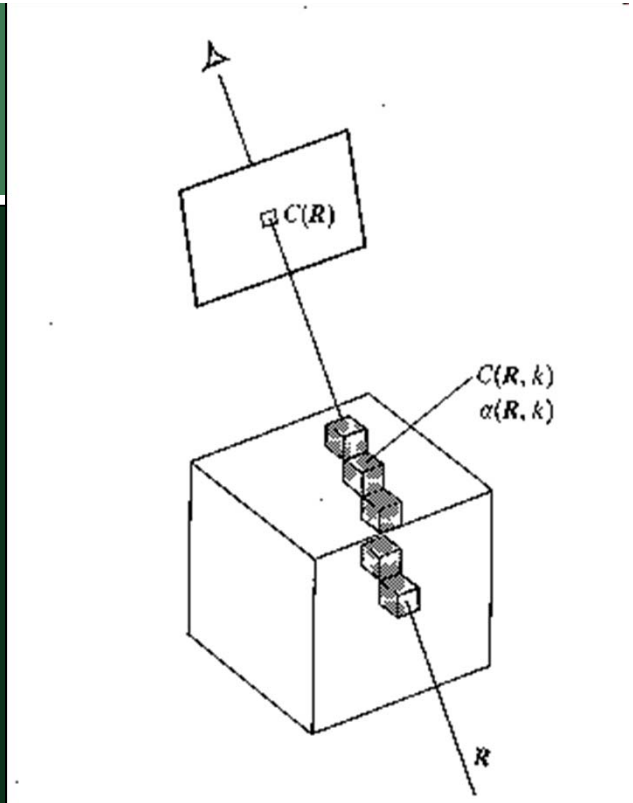
- Fundamental questions:
  - What is the right balance of distributed- vs. shared-memory parallelism? How does balance impact performance?
  - How to map algorithm onto a complex memory, communication hierarchy?
- Relatively new research area, not a great deal of published work.
- Studies focus on “solvers,” not vis/graphics.
  - Parallel visualization applications all use MPI, none “multi-core” aware.
- Conclusions of these previous works.
  - What is best? Answer: it depends.
  - Many factors influence performance/scalability:
    - Synchronization overhead.
    - Load balance (intra- and inter-chip).
    - Communication overhead and patterns.
    - Memory access patterns.
    - Fixed costs of initialization.
    - Number of runtime threads.

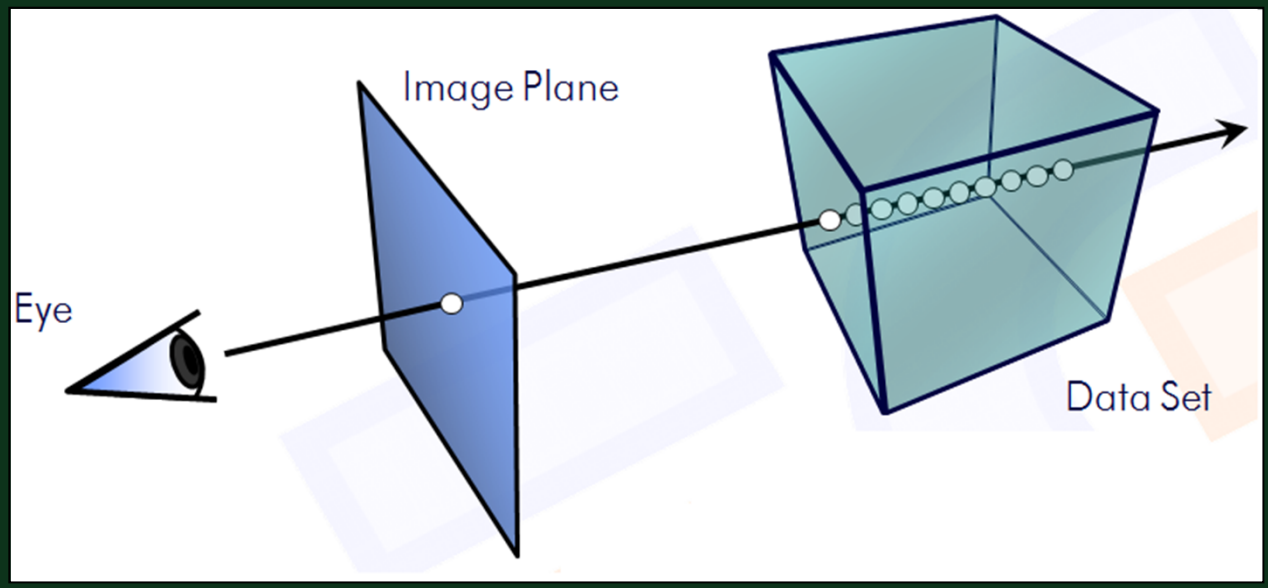
This Study

- First-ever study of hybrid parallelism on visualization: raycasting volume rendering.
  - Parallels similar work done for scientific computing.
- Hybrid-parallel implementation/architecture.
- Performance study.
  - Runs at 216K-way parallel: 6x larger than any published results.
  - Look at:
    - Costs of initialization.
    - Memory use comparison.
    - Scalability.
    - Absolute runtime.

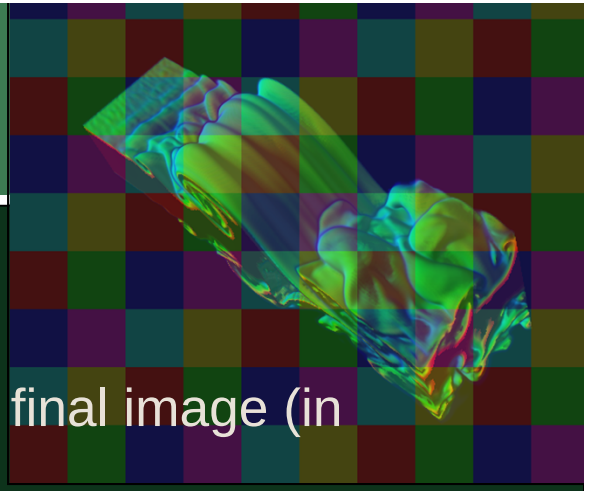
Algorithm Studied: Raycasting VR

- Overview of Levoy's method
  - For each pixel in image plane:
    - Find intersection of ray and volume
    - Sample data (RGBa) along ray, integrate samples to compute final image pixel color

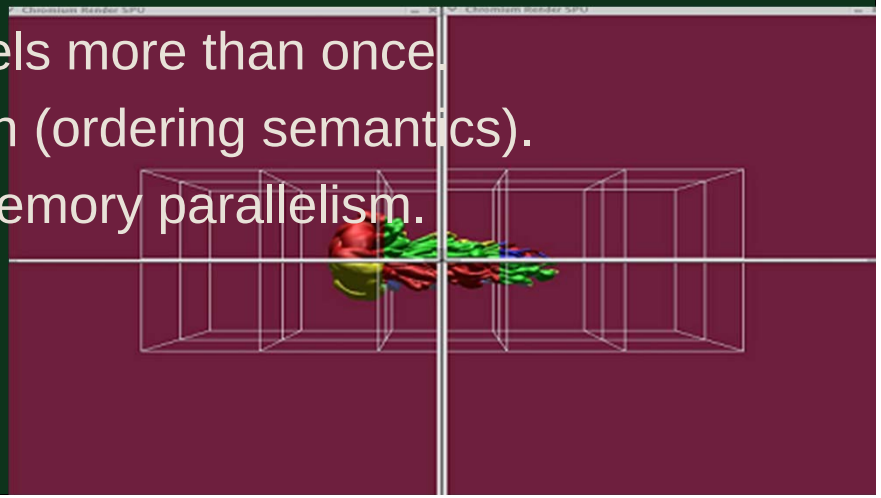




Parallelizing Volume Rendering

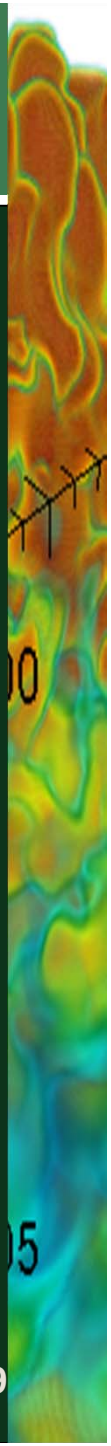


- Image-space decomposition.
  - Each process works on a disjoint subset of the final image (in parallel)
  - Processes may access source voxels more than once, will access a given output pixel only once.
  - Great for shared memory parallelism.
- Object-space decomposition.
  - Each process works on a disjoint subset of the input data (in parallel).
  - Processes may access output pixels more than once.
  - Output requires image composition (ordering semantics).
  - Typical approach for distributed memory parallelism.



Hybrid Parallel Volume Rendering

- Hybrid-parallelism a blend of shared- and distributed-memory parallelism.
- Distributed-memory parallelism:
  - Each socket assigned a spatially disjoint subset of source data, produces an image of its chunk.
  - All subimages composited together into final image.
  - MPI implementation.
- Shared-memory parallelism:
  - Inside a socket, threads use image-space partitioning, each thread responsible for a subset of the final image.
    - What is the best image tile size? (Autotuning presentation)
  - Implementations (2): pthreads, OpenMP.

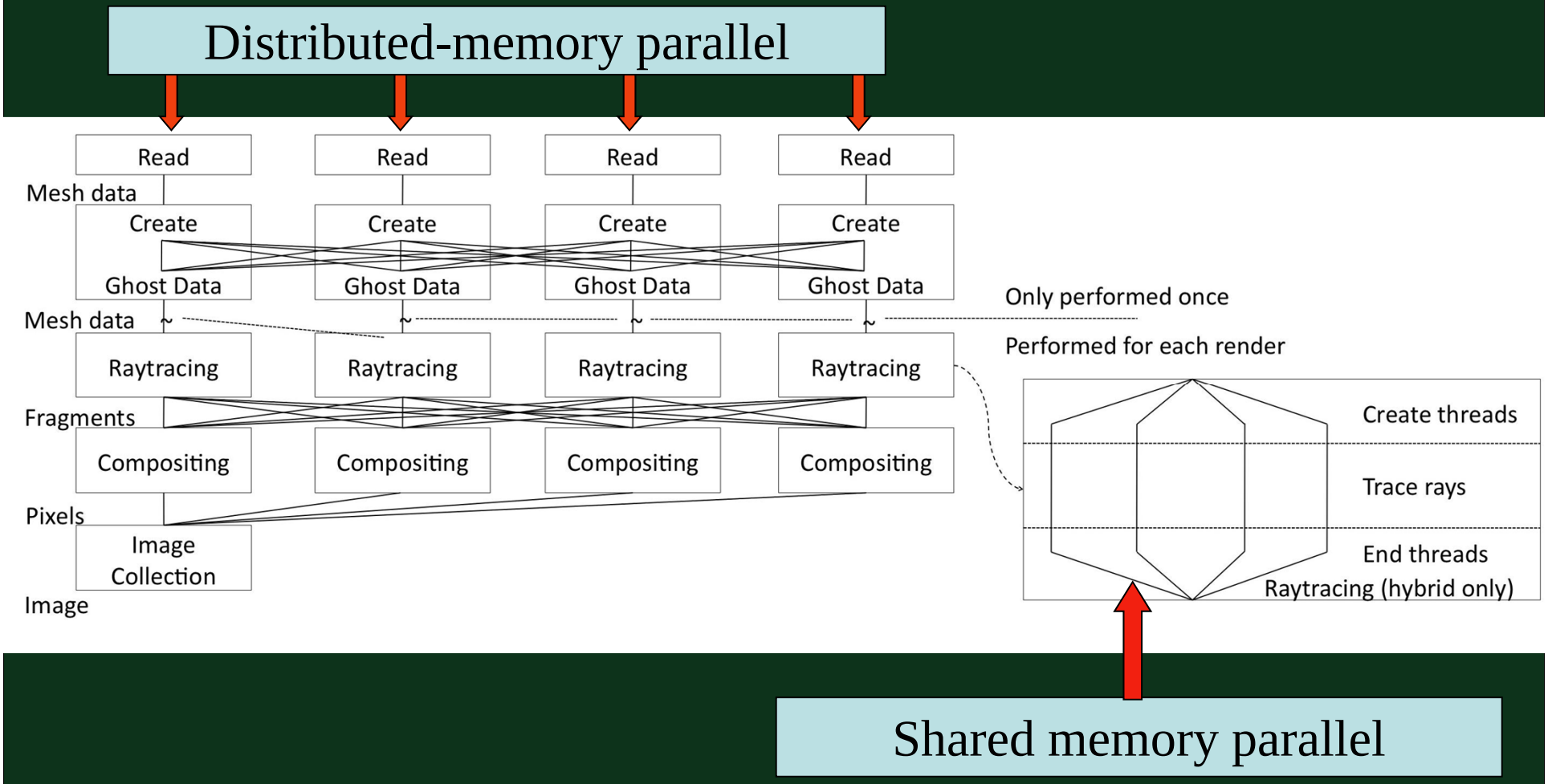


Hybrid Parallelism vs. Hybrid Volume Rendering

- Hybrid parallelism:
  - Refers to mixture of distributed- and shared-memory parallelism.
- Hybrid volume rendering:
  - Refers to mixture of object- and image-order techniques to do volume rendering.
  - Most contemporary parallel volume rendering projects are hybrid volume renderers:
    - Object order – divide data into disjoint chunks, each processor works on its chunk of data.
    - Image order – parallel compositing algorithm divides work over final image, each composites over its portion of the final image.
    - A two-stage algorithm, heavy communication load between stages.

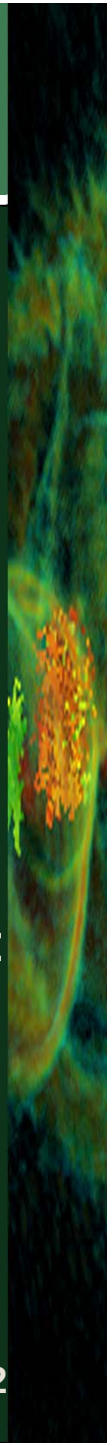
Hybrid Parallel Volume Rendering

- Our hybrid-parallel architecture:



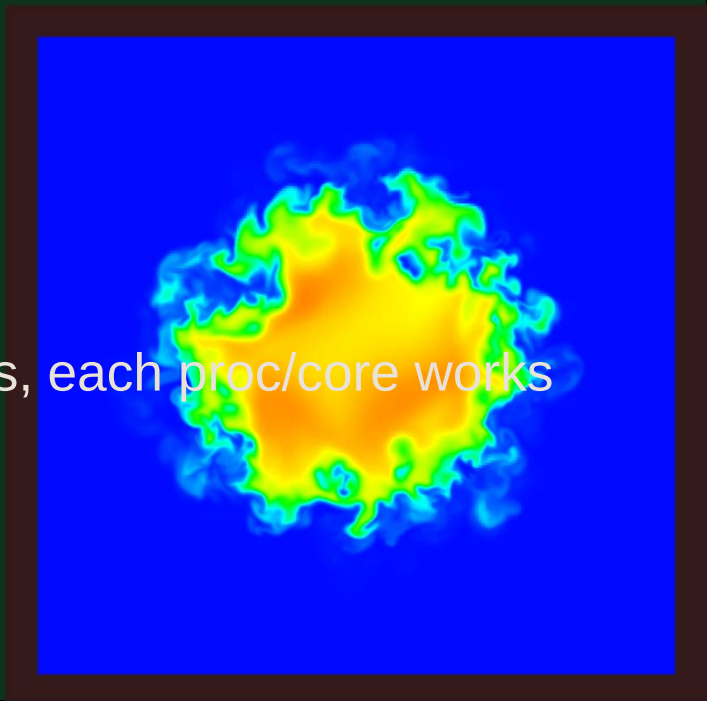
Our Experiment

- Thesis: hybrid-parallel will exhibit favorable performance, resource utilization characteristics compared to traditional approach.
- How/what to measure?
  - Memory footprint, communication traffic load, scalability characteristics, absolute runtime.
  - Across a wide range of concurrencies.
    - Remember: we're concerned about what happens at extreme concurrency.
  - Algorithm performance somewhat dependent upon viewpoint, data:
    - Vary viewpoints over a set that cut through data in different directions: will induce different memory access patterns.
- Strong scaling study: hold problem size constant, vary amount of resources.

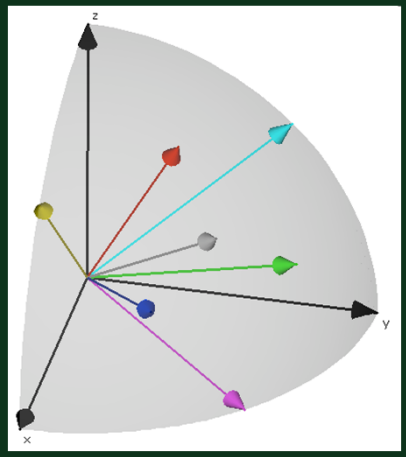


Experiment: Platform and Source Data

- Platform: JaguarPF, a Cray XT5 system at ORNL
  - 18,688 nodes, dual-socket, six-core AMD Opteron (224K cores)
- Source data:
  - Combustion simulation results, hydrogen flame (data courtesy J. Bell, CCSE, LBNL)
  - Effective AMR resolution: 1024^3, flattened to 512^3, runtime upscaled to 4608^3 (to avoid I/O costs).
  - 91B cells, ~3TB total memory footprint.
- Target image size: 4608^2 image.
  - Want approx 1:1 voxels to pixels.
- Strong scaling study:
  - As we increase the number of procs/cores, each proc/core works on a smaller-sized problem.
  - Time-to-solution should drop.



Experiment – The Unit Test



- Raycasting time: view/data dependent
  - Execute from 10 different prescribed views: forces with- and cross-grained memory access patterns.
  - Execute 10 times, result is average of all.
- Compositing
  - Five different ratios of compositing PEs to rendering PEs.
- Measure:
  - Memory footprint right after initialization.
  - Memory footprint for data blocks and halo exchange.
  - Absolute runtime and scalability of raycasting and compositing.
  - Communication load between RC and compositing.

Memory Use – Data Decomposition

- 16GB RAM per node
  - Sets lower bound on concurrency for this problem size: 1728-way parallel (no virtual memory!).
- Source data (1x), gradient field (3x)
- Want cubic decomposition.
  - 1x2x3 block configuration per socket for –only.
- -hybrid has ~6x data per socket than –only
  - Would prefer to run study on 8-core CPUs to maintain cubic shape

MPI-only		MPI-hybrid		
MPI PEs	Block Dimensions	MPI PEs	Block Dimensions	Memory Per Node
12^3=1728	$384 \times 384 \times 384$	288	$384 \times 768 \times 1152$	10368MB
24^3=13824	$192 \times 192 \times 192$	2304	$192 \times 384 \times 576$	1296MB
36^3=46656	$128 \times 128 \times 128$	7776	$128 \times 256 \times 384$	384MB
48^3=110592	$96 \times 96 \times 96$	18432	$96 \times 192 \times 288$	162MB
60^3=216000	$76 \times 76 \times 76$	36000	$76 \times 153 \times 230$	80.4MB / 81.6MB

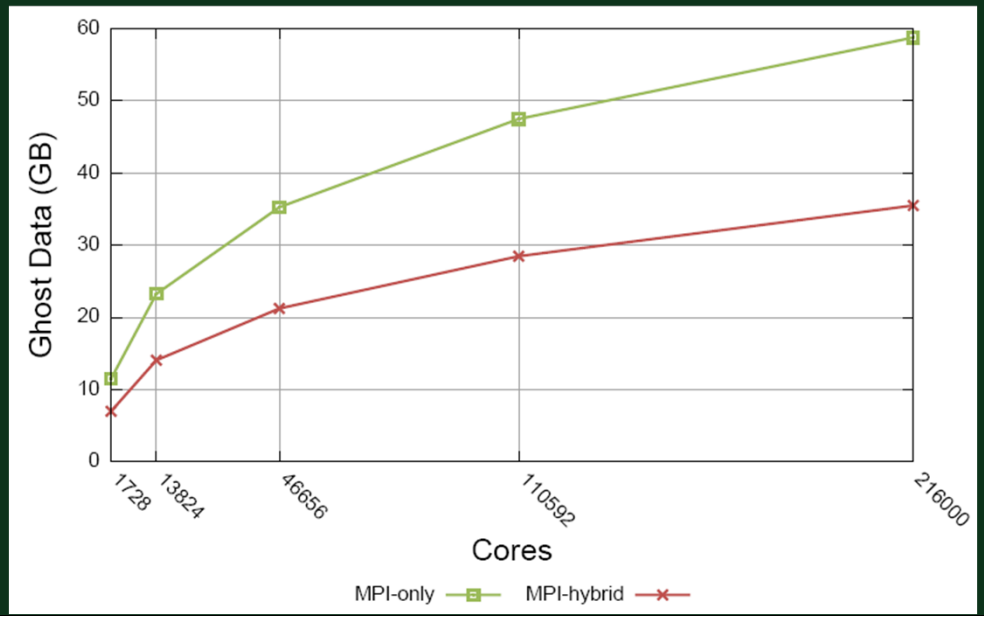
Memory Use – MPI_Init()

- Per PE memory:
  - About the same at 1728, over 2x at 216000.
- Aggregate memory use:
  - About 6x at 1728, about 12x at 216000.
  - At 216000, -only requires 2GB of memory for initialization per node!!!

Cores	Mode	MPI PEs	MPI Runtime Memory Usage		
			Per PE (MB)	Per Node (MB)	Aggregate (GB)
1728	MPI-hybrid	288	67	133	19
1728	MPI-only	1728	67	807	113
13824	MPI-hybrid	2304	67	134	151
13824	MPI-only	13824	71	857	965
46656	MPI-hybrid	7776	68	136	518
46656	MPI-only	46656	88	1055	4007
110592	MPI-hybrid	18432	73	146	1318
110592	MPI-only	110592	121	1453	13078
216000	MPI-hybrid	36000	82	165	2892
216000	MPI-only	216000	176	2106	37023

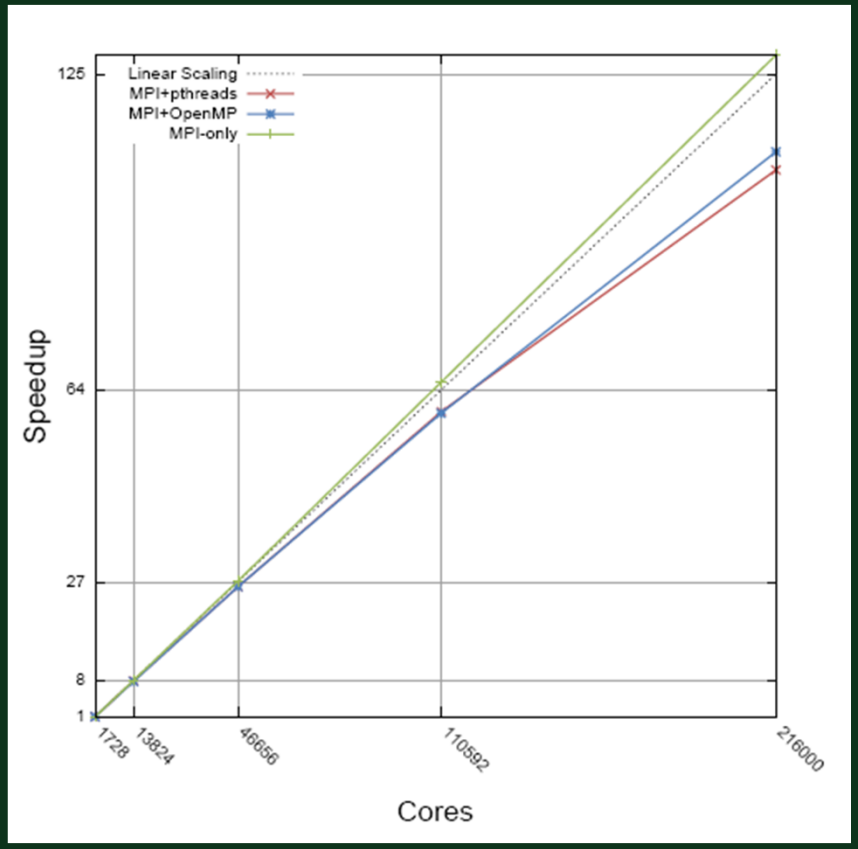
Memory Use – Ghost Zones

- Two layers of ghost cells required for this problem:
  - One for trilinear interpolation during ray integration loop.
  - Another for computing a gradient field (central differences) for shading.
- Hybrid approach uses fewer, but larger data blocks.
  - ~40% less memory required for ghost zones (smaller surface area)
  - Reduced communication costs



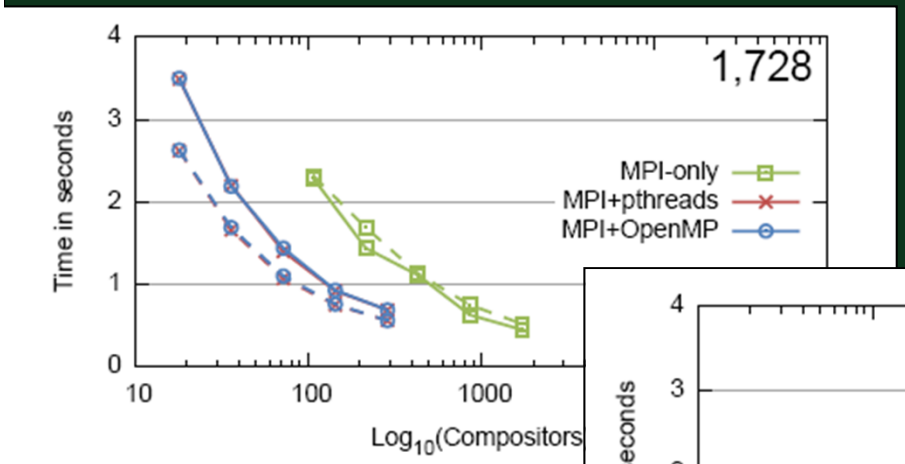
Scalability – Raycasting Phase

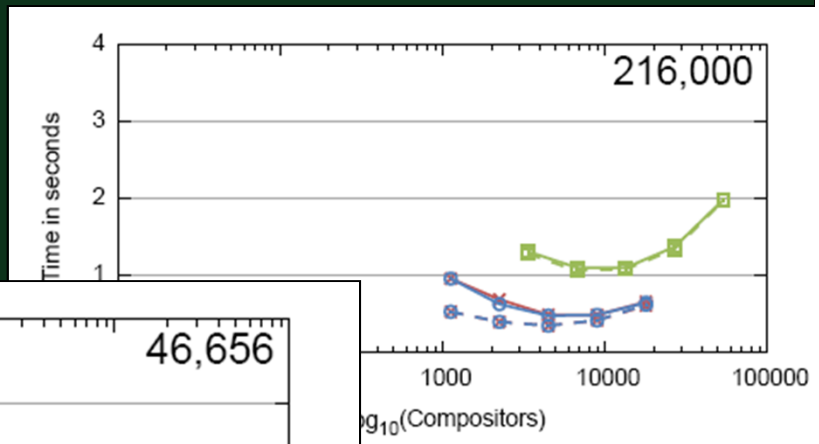
- Near linear scaling since no interprocess communication.
- -hybrid shows sublinear scaling due to oblong block shape.
- -only shows slightly better than linear due to reduced work caused by perspective foreshortening.

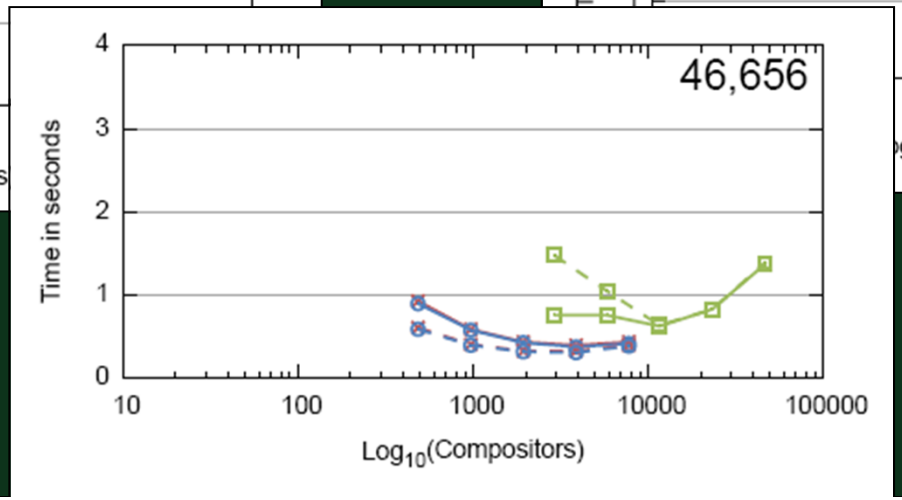


Scalability – Compositing

- How many compositors to use?
  - Previous work: 1K to 2K for 32K renderers (Peterka, 2009).
  - Our work: above ~46K renderers, 4K to 8K works better.
  - -hybrid cases always performs better: fewer messages.
  - Open question: why the critical point?

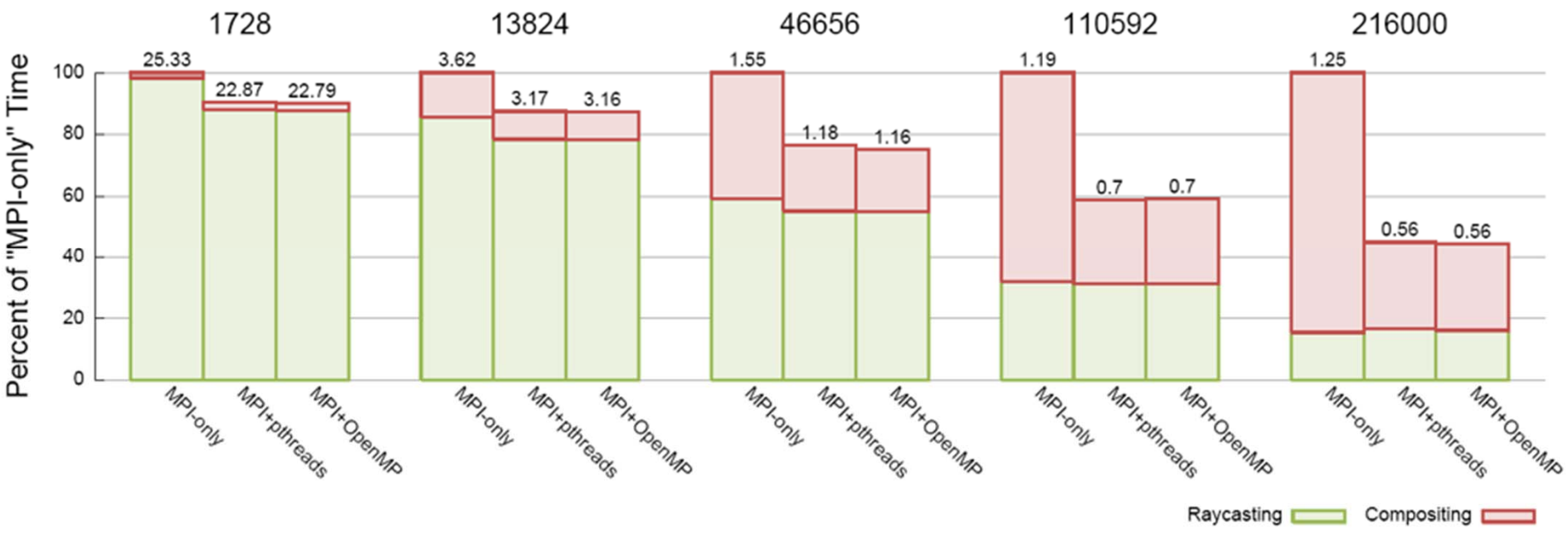






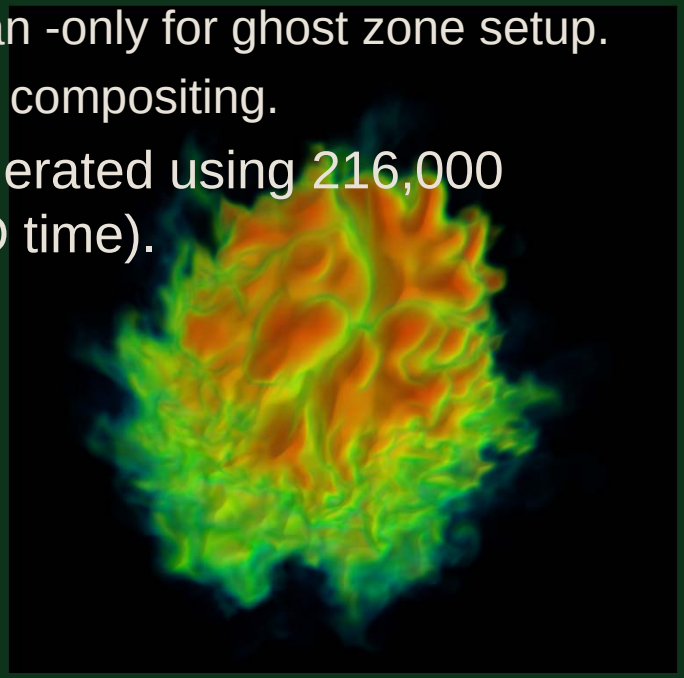
Absolute Runtime

- -hybrid outperforms –only at every concurrency level.
  - At 216K-way parallel, -hybrid is more than twice as fast as –only.
  - Compositing times begin to dominate: communication costs.



Summary of Results

- Absolute runtime: -hybrid twice as fast as –only at 216K-way parallel.
- Memory footprint: -only requires 12x more memory for MPI initialization then –hybrid
  - Factor of 6x due to 6x more MPI PEs.
  - Additional factor of 2x at high concurrency, likely a vendor MPI implementation (an N^2 effect).
- Communication traffic:
  - -hybrid performs 40% less communication than -only for ghost zone setup.
  - -only requires 6x the number of messages for compositing.
- Image: 4608^2 image of a ~4500^3 dataset generated using 216,000 cores on JaguarPF in ~0.5s (not counting I/O time).



Open Questions

- What about weak scaling?
- What about other visualization algorithms?
- What about more cores? E.g., GPUs?
- What about alternatives to pthreads/OpenMP?
- What if cores don't share memory?