Experiments with Pure Parallelism

Hank Childs, Dave Pugmire, Sean Ahern,
Brad Whitlock, Mark Howison, Prabhat,
Gunther Weber, & Wes Bethel

April 13, 2010

The landscape: how tools process data

This technique is called “pure parallelism”

Pure parallelism

- Pure parallelism is data-level parallelism, but…
  - Multi-resolution can be data-level parallelism
  - Out-of-core can be data-level parallelism
- Pure parallelism: "brute force" … processing full resolution data using data-level parallelism
- Pros:
  - Easy to implement
- Cons:
  - Requires large I/O capabilities
  - Requires large amount of primary memory
  - → requires big machines

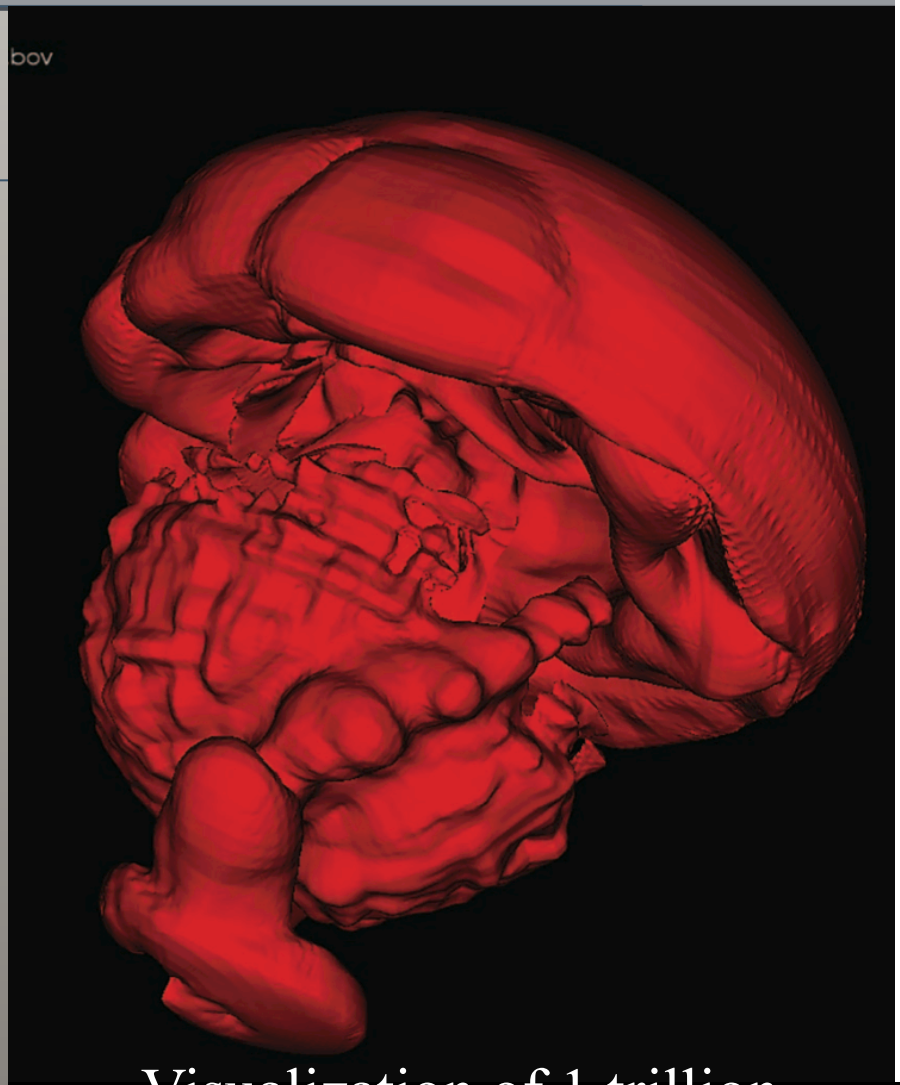
Research Questions

- Is it possible/feasible to run production-quality visual data analysis s/w on large machines and on large data sets?
  - Are the tools we use right now ready for tomorrow's data?
- What obstacles/bottlenecks do we encounter at massive data?

Experiment methodology

- Preprocess step: generate large data set
- Read it
- Contour
- Render @ 1024x1024

- Synthetic data:
  - Wanted to look at tomorrow's data; not available yet
  - Synthetic data should be reasonable surrogate for real data.



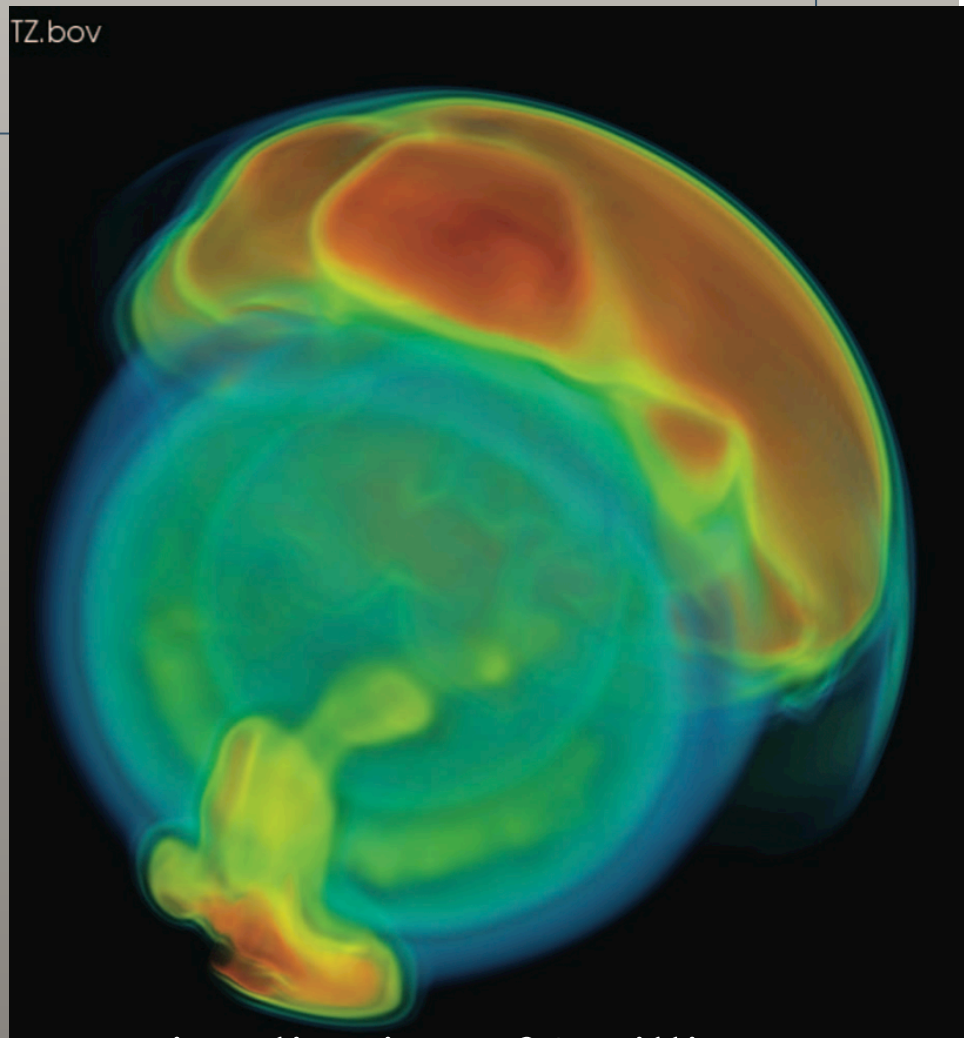
Visualization of 1 trillion cells, visualized with VisIt on Franklin using 16,000 cores.

Experiment methodology, continued

- Only used pure parallelism
  - This experiment was about testing the limits of pure parallelism
  - Purposely did not use in situ, multi-resolution, out-of-core, data subsetting
- Pure parallelism is what the production visualization tools use right now (*).

Volume rendering

- Ran into problems with volume rendering.
  - See Dave's talk.
- Problem eventually fixed, but not in time for study
  - Runs on these big machines are opportunistic and it's hard to get a second chance
  - Approximately five seconds per render
- Contouring exercises much of the infrastructure (read, process, render)



Visualization of 2 trillion cells, visualized with VisIt on JaguarPF using 32,000 cores.

Experiment methodology, continued

- Three basic variations
  - Vary over supercomputing environment
  - Vary over data generation
  - Vary over I/O pattern

Varying over supercomputer environment

- Goals:
  - Ensure results aren't tied to a single machine.
  - Understand differences from different architectures.
- Experiment details
  - 1 trillion cells per 16,000 cores
  - 10*NCores "Brick-of-float" files, gzipped
  - Upsampled data

Machine name	Machine type or OS	Total no. of cores	Memory per core (Gbytes)	System type	Clock speed	Peak flops	Top 500 rank (as of Nov. 2009)
JaguarPF	Cray	224,162	2.0	XT5	2.6 GHz	2.33 Pflops	1
Ranger	Sun Linux	62,976	2.0	Opteron Quad	2.0 GHz	503.8 Tflops	9
Dawn	Blue Gene/P	147,456	1.0	PowerPC	850.0 MHz	415.7 Tflops	11
Franklin	Cray	38,128	1.0	XT4	2.6 GHz	352 Tflops	15
Juno	Commodity (Linux)	18,402	2.0	Opteron Quad	2.2 GHz	131.6 Tflops	27
Purple	AIX (Advanced Interactive Executive)	12,208	3.5	Power5	1.9 GHz	92.8 Tflops	66

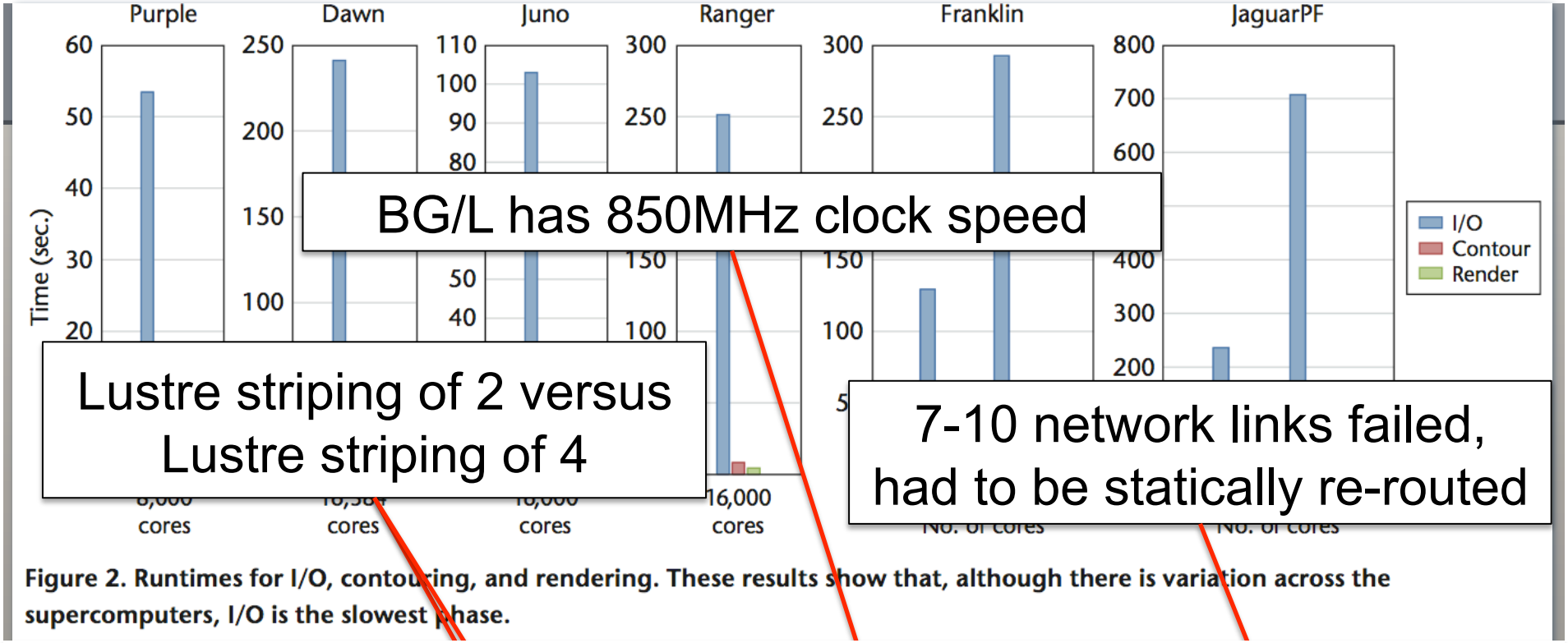
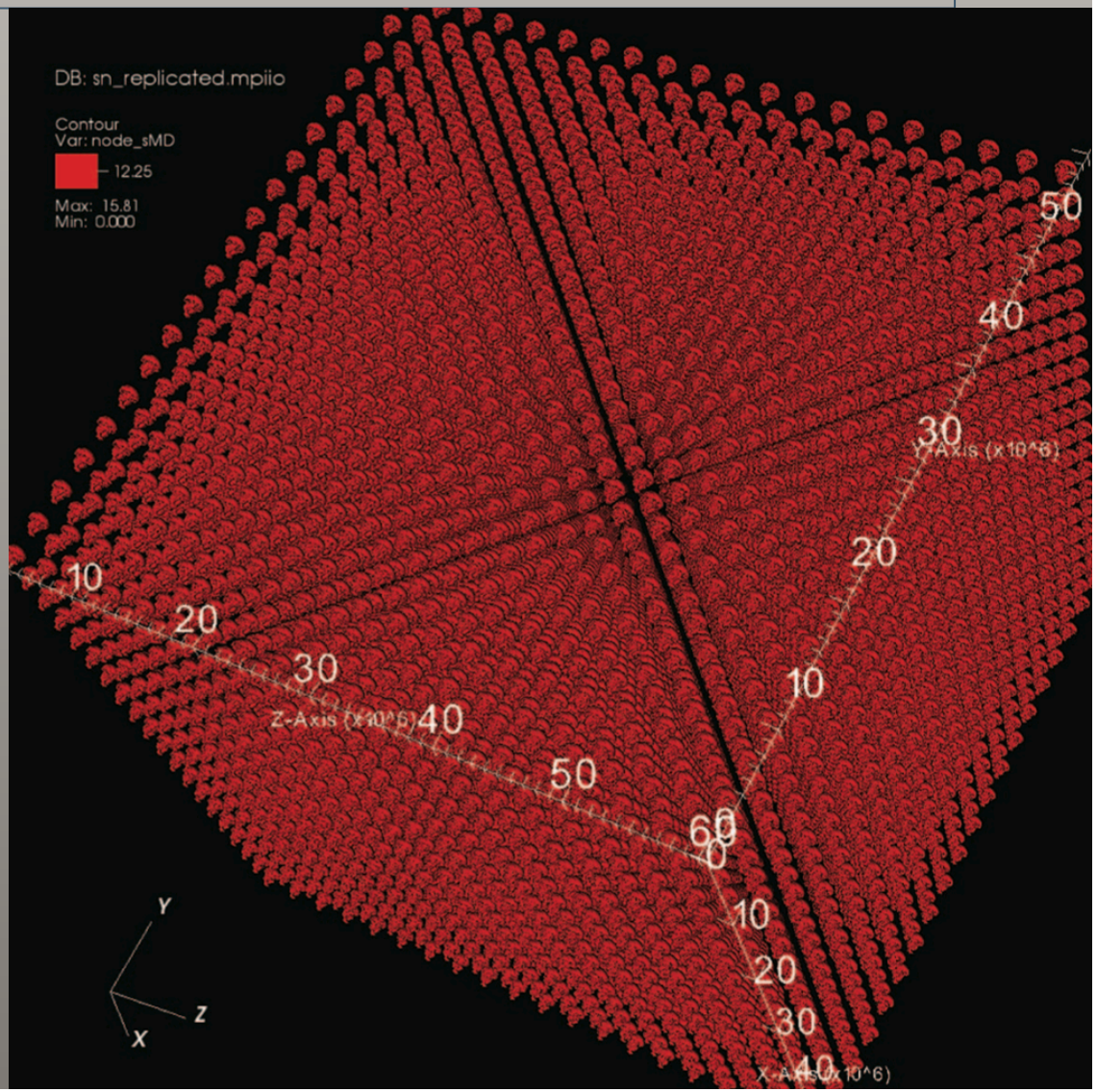


Figure 2. Runtimes for I/O, contouring, and rendering. These results show that, although there is variation across the supercomputers, I/O is the slowest phase.

Varying over data generation pattern

- Concern: does upsampling produce unrepresentatively smooth surfaces?
- Alternative: replication



Visualization of 1 trillion cells, visualized with VisIt on Franklin using 16,000 cores.

Results from data generation test

- Test on franklin, using 16,000 cores with unzipped data

Data generation	Total I/O time (sec.)	Contour time (sec.)	Total pipeline execution time (sec.)	Rendering time (sec.)
Upsampled	478.3	7.6	486.0	2.8
Replicated	493.0	7.6	500.7	4.9

Contouring time is the same because case where a triangle is generated is rare.

Rendering time is different because replicated pattern has more geometry.

Varying over I/O pattern

- Previous tests: uncoordinated I/O, doing 10 “fread”s per core.
- Can collective communication help?

I/O pattern	No. of cores	Data set size (TCells)	Total I/O time (sec.)	Data read (Gbytes)	Read bandwidth (Gbytes per second)
Collective	16,016	1	478.3	3,725.3	7.8
Noncollective	16,000	1	129.3	954.2	7.4

Franklin I/O maximum: 12GB/s

Pitfalls at scale

- Volume rendering (see Dave's talk)
- Startup time
  - Loading plugins overwhelmed file system
  - Took ~5 minutes
  - Solution #1: Read plugin information on MPI task 0 and broadcast. (90% speedup)
  - Solution #2: static linking
    - Still need to demonstrate at scale

Pitfalls at scale #2: All to one communication

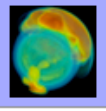
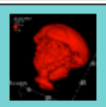
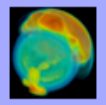
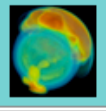
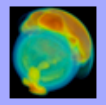
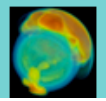
- Each MPI task needs to report high level information
  - Was there an error in execution for that task?
  - Data extents? Spatial Extents?
- Previous implementation:
  - Every MPI task sends a direct message to MPI task 0.
- New implementation (Miller, LLNL):
  - Tree communication

Pitfalls at scale #3: reproducible results

All-to-one?	No. of cores	Data set size (TCells)	Total I/O time (sec.)	Contour time (sec.)	Total pipeline execution time (sec.)	Pipeline minus contour & I/O (sec.)	Date run
Yes	16,384	1	88.0	32.2	368.7	248.5	June 2009
Yes	65,536	4	95.3	38.6	425.9	294.0	June 2009
No	16,384	1	240.9	32.4	277.6	4.3	Aug. 2009

Repeated debugging runs at scale are critical to resolving issues like these.

This study continued after the initial effort as a way to validate new machines.

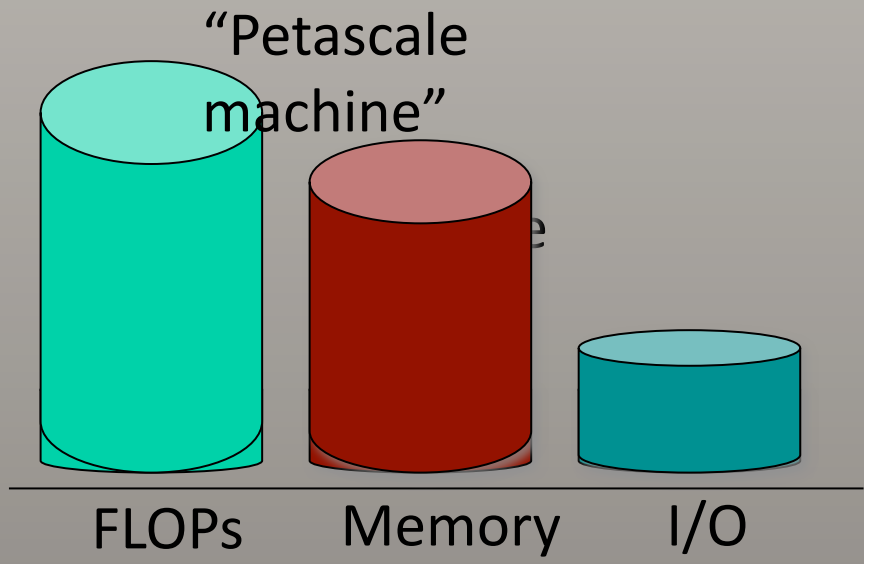
Date	Number of zones per timestep	Number of timesteps	Total Size Zones & Bytes	Mesh+Vars	Platform	Visioneers (Visualization-Engineers)	More Info...
October 2009	$20{,}001^3$ (8 Tz)	1		rect+1	12000 cpus of Graph (Linux)	Cyrus Harrison	More Info 
June 2009	$15{,}871^3$ (4 Tz)	1		rect+1	65536 cpus of Dawn (BlueGene/P) Dawn Details	Brad Whitlock	
May 2009	$12{,}596^3$ (2 Tz)	1		rect+1	32000 cpus of Jaguar (Cray)	David Pugmire Sean Ahern	More Info 
June 2009	$12{,}596^3$ (2 Tz)	1		rect+1	32000 cpus of Franklin (Cray)	Mark Howison Prabhat Hank Childs	More Info 
April 2009	$10{,}000^3$ (1 Tz)	1		rect+1	16000 cpus of Jaguar (Cray)	David Pugmire Sean Ahern	More Info 
May 2009	$10{,}000^3$ (1 Tz)	1		rect+1	16000 cpus of Ranger (Sun Linux)	Hank Childs	More Info 

Should more tools have been used?

- Could have performed this study with VisIt, ParaView, EnSight, etc.
- Successful test with VisIt validates pure parallelism.
- Of course, I/O is a big problem … but ParaView, EnSight, etc, are doing the same “fread”s.

Trends in I/O

- Pure parallelism is almost always >50% I/O and sometimes 98% I/O
- Amount of data to visualize is typically O(total mem)
- **Two big factors:**
  - ① **how much data you have to read**
  - ② **how fast you can read it**
- **→ Relative I/O (ratio of total memory and I/O) is key**



Anedoctal evidence: relative I/O really is getting slower.

Time to write memory to disk

Machine name	Main memory	I/O rate	
ASC purple	49.0TB	140GB/s	5.8min
BGL-init	32.0TB	24GB/s	22.2min
BGL-cur	69.0TB	30GB/s	38.3min
Petascale machine	??	??	>40min

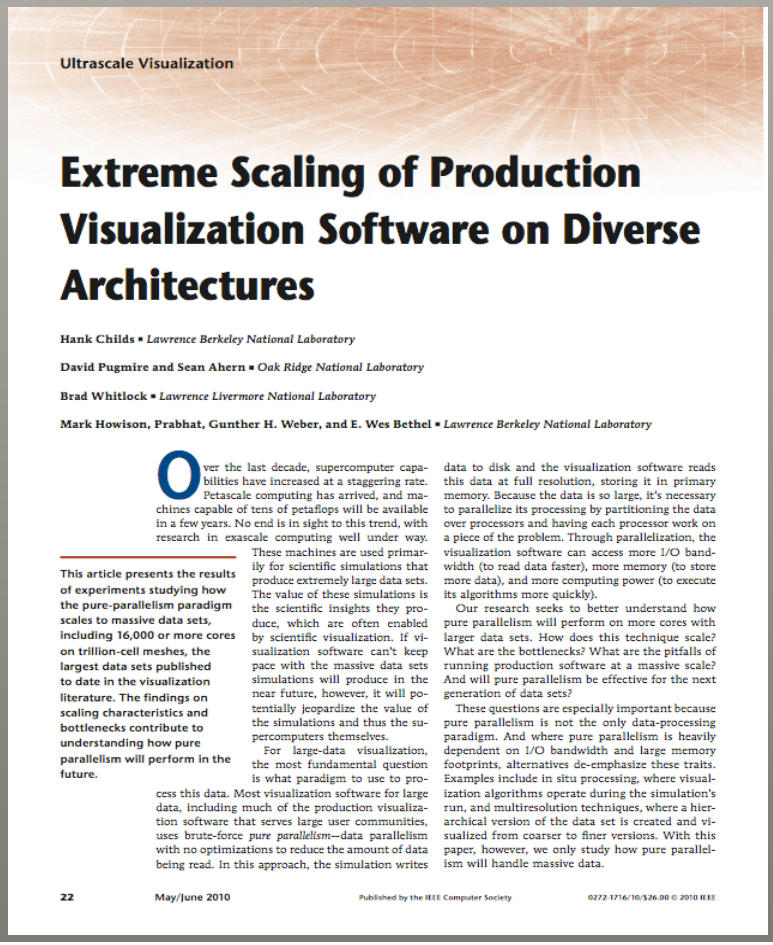
Why is relative I/O getting slower?

- "I/O doesn't pay the bills"
  - And I/O is becoming a dominant cost in the overall supercomputer procurement.
- Simulation codes aren't as exposed.
  - And will be more exposed with proposed future architectures.

We need to de-emphasize I/O in our visualization and analysis techniques.

Conclusions

- Pure parallelism works, but is only as good as the underlying I/O infrastructure
  - I/O future looks grim
  - Positive indicator for in situ processing
- Full results available in special issue of Computer Graphics & Applications, special issue on Ultrascale Visualization.



Ultrascale Visualization

Extreme Scaling of Production Visualization Software on Diverse Architectures

Hank Childs ▪ *Lawrence Berkeley National Laboratory*

David Pugmire and Sean Ahern ▪ *Oak Ridge National Laboratory*

Brad Whitlock ▪ *Lawrence Livermore National Laboratory*

Mark Howison, Prabhat, Gunther H. Weber, and E. Wes Bethel ▪ *Lawrence Berkeley National Laboratory*

This article presents the results of experiments studying how the pure-parallelism paradigm scales to massive data sets, including 16,000 or more cores on trillion-cell meshes, the largest data sets published to date in the visualization literature. The findings on scaling characteristics and bottlenecks contribute to understanding how pure parallelism will perform in the future.

Over the last decade, supercomputer capabilities have increased at a staggering rate. Petascale computing has arrived, and machines capable of tens of petaflops will be available in a few years. No end is in sight to this trend, with research in exascale computing well under way. These machines are used primarily for scientific simulations that produce extremely large data sets. The value of these simulations is the scientific insights they produce, which are often enabled by scientific visualization. If visualization software can't keep pace with the massive data sets simulations will produce in the near future, however, it will potentially jeopardize the value of the simulations and thus the supercomputers themselves.

For large-data visualization, the most fundamental question is what paradigm to use to process this data. Most visualization software for large data, including much of the production visualization software that serves large user communities, uses brute-force *pure parallelism*—data parallelism with no optimizations to reduce the amount of data being read. In this approach, the simulation writes data to disk and the visualization software reads this data at full resolution, storing it in primary memory. Because the data is so large, it's necessary to parallelize its processing by partitioning the data over processors and having each processor work on a piece of the problem. Through parallelization, the visualization software can access more I/O bandwidth (to read data faster), more memory (to store more data), and more computing power (to execute its algorithms more quickly).

Our research seeks to better understand how pure parallelism will perform on more cores with larger data sets. How does this technique scale? What are the bottlenecks? What are the pitfalls of running production software at a massive scale? And will pure parallelism be effective for the next generation of data sets?

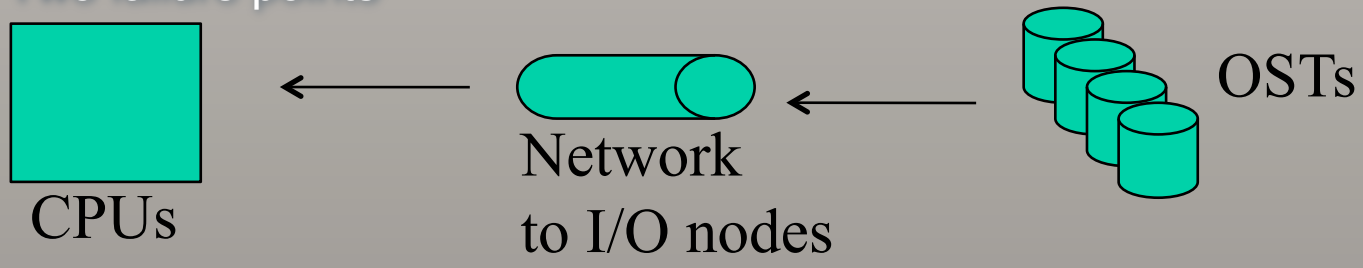
These questions are especially important because pure parallelism is not the only data-processing paradigm. And where pure parallelism is heavily dependent on I/O bandwidth and large memory footprints, alternatives de-emphasize these traits. Examples include in situ processing, where visualization algorithms operate during the simulation's run, and multiresolution techniques, where a hierarchical version of the data set is created and visualized from coarser to finer versions. With this paper, however, we only study how pure parallelism will handle massive data.

 Published by the IEEE Computer Society 0272-1716/10/$26.00 © 2010 IEEE

Backup slides

Is the I/O we saw on the hero runs indicative of what we will see on future machines?

- Two failure points



  - The number of I/O nodes will be dropping, especially with increasing numbers of cores per node, making the network to the I/O nodes the probable bottleneck
- Jaguar 32K / 2T cells took 729s. If BW disk is 5X faster that is still 145s.
- LLNL has a bunch of disk and we couldn't get below two minutes because of contention
- Even if BW gets enough disk, disk is very expensive and future machines will likely not get enough.
  - This is especially true if a FLASH solution takes hold.