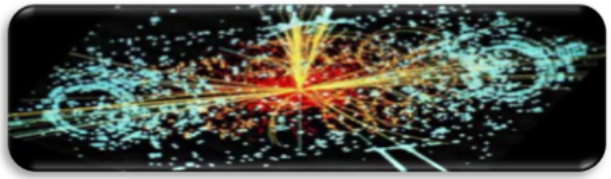
Concurrency-Optimized I/O For Visualizing HPC Simulations: An Approach Using Dedicated I/O Cores

Matthieu Dorier, Franck Cappello, Marc Snir,
Bogdan Nicolae, Gabriel Antoniu

4th workshop of the Joint Laboratory for Petascale Computing, Urbana, November 2010

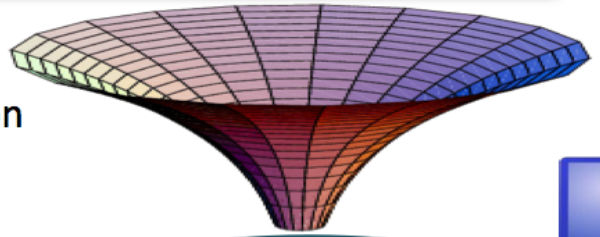
NCSA – University of Illinois at Urbana-Champaign
INRIA Saclay – Grand-Large Team
INRIA Rennes– KerData Team
ENS de Cachan - Brittany

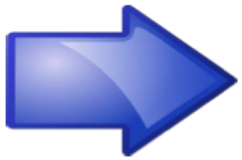
Context: I/O efficient visualization for post-petascale HPC simulations

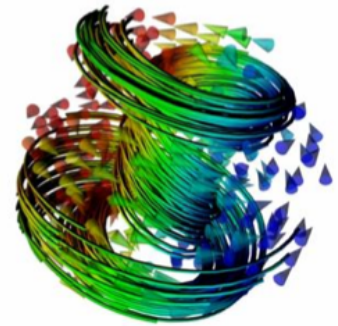


Blue Waters Simulations

The Problem:
Poor I/O concurrency control
Simulation/Backup/Visualisation

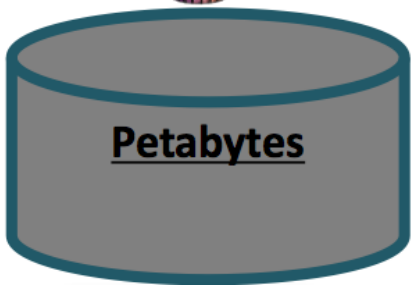




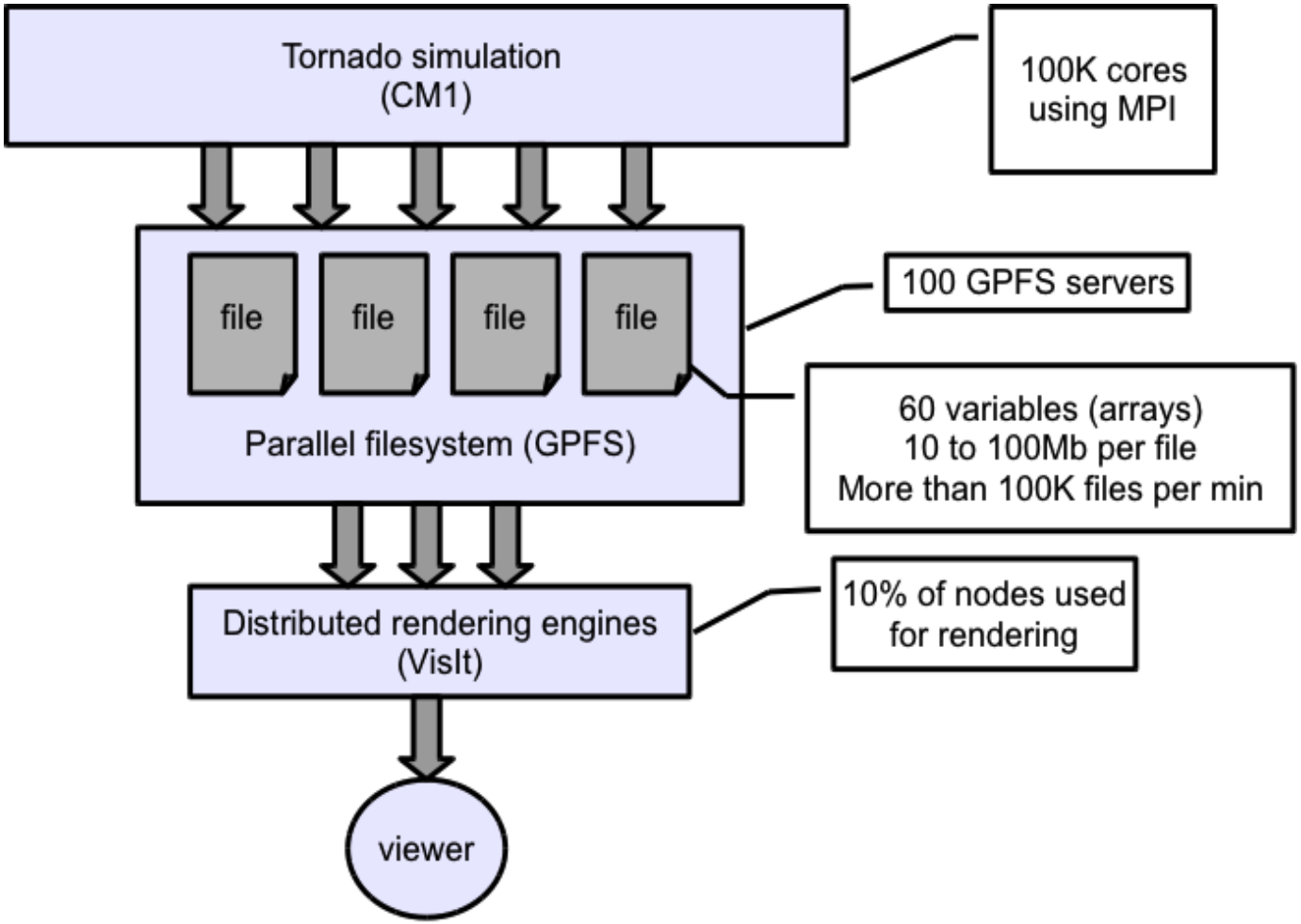


Visualization

The Challenge:
Efficient concurrent I/O
Efficiently visualize and store data without impacting the simulation

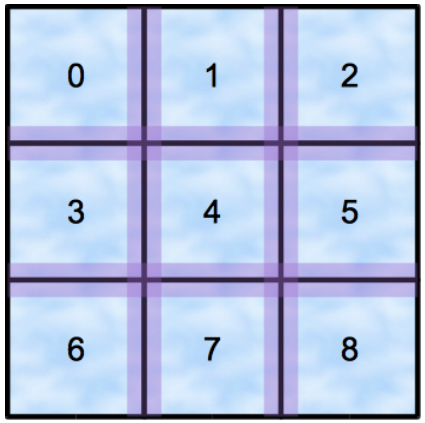


Case study: CM1 simulation/visualization pipeline

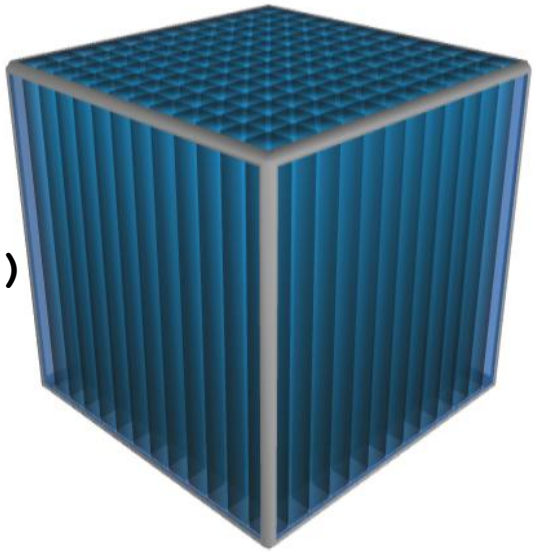


The CM1 parallel simulation

- Cloud Model 1st Generation (CM1)
- Sky discretization along a 3D grid: uses an approximation of the finite element method
- Layout: two dimensional grid of subdomains
- MPI processes are mapped onto this layout
- Each MPI process solves equations on its subdomain, then exchange ghost data with its neighborhood




```
Initialize, time t = 0
While t < time_max
     Solve equations
     Exchange ghost data (MPI)
     If t % K == 0 then
          Write file (HDF5)
     Endif
     t++
Endwhile
```

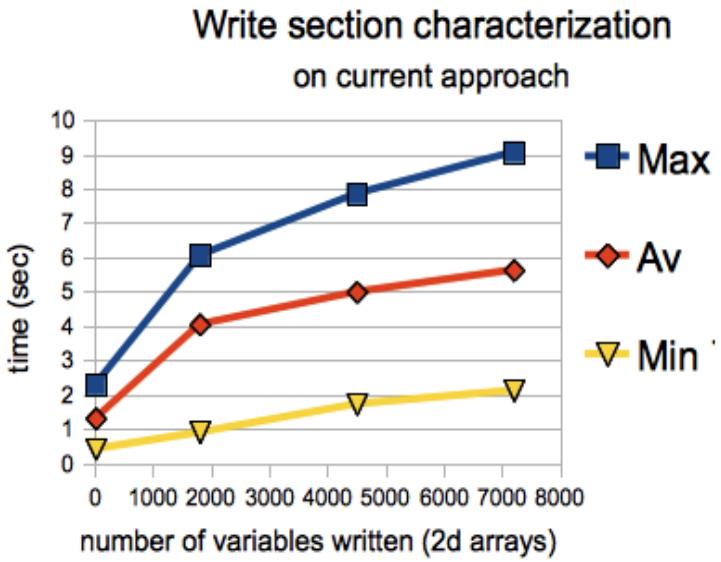


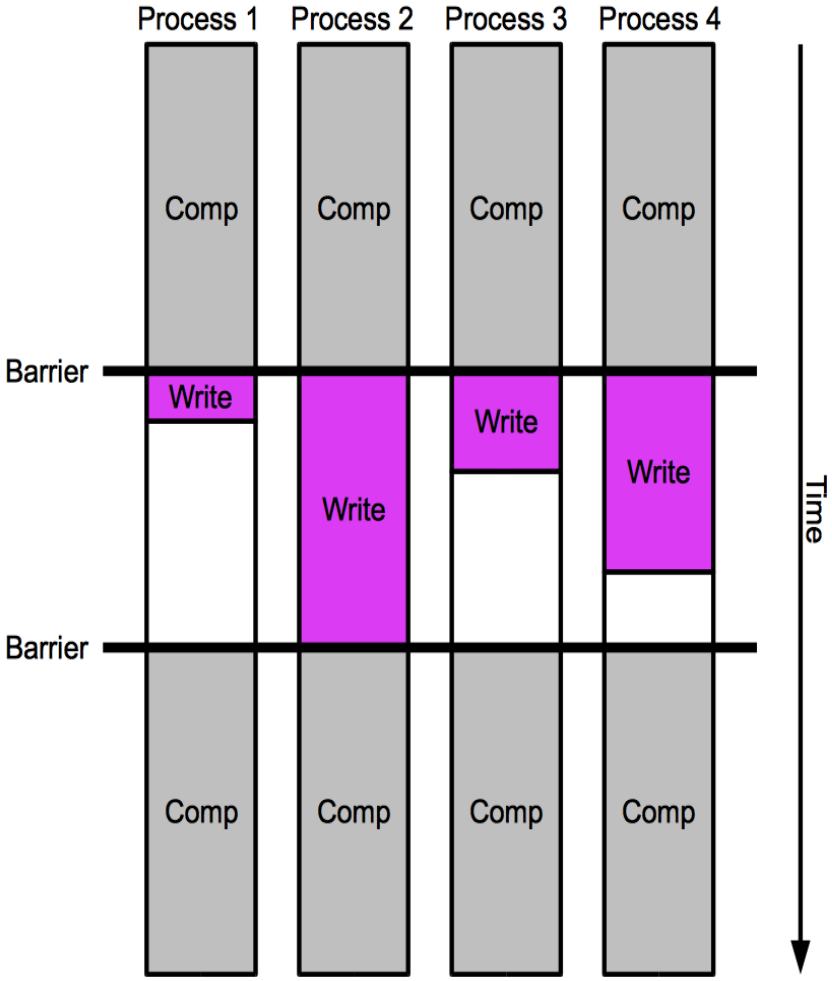
Output description

- **Output format:**
  - HDF5 (Hierarchical Data Format), providing dataset organization and description through metadata
  - Gzip compression (level 4)
  - One file per core per backup
  - One backup every K steps (K configurable)
- **In a BlueWaters deployement:**
  - 100K files per backup, written at the same time
  - Backup every 100 steps (depending on the configuration, 100 steps take 25 to 400 seconds)
  - Only 100 GPFS servers to handle all requests → bottleneck!

General behavior

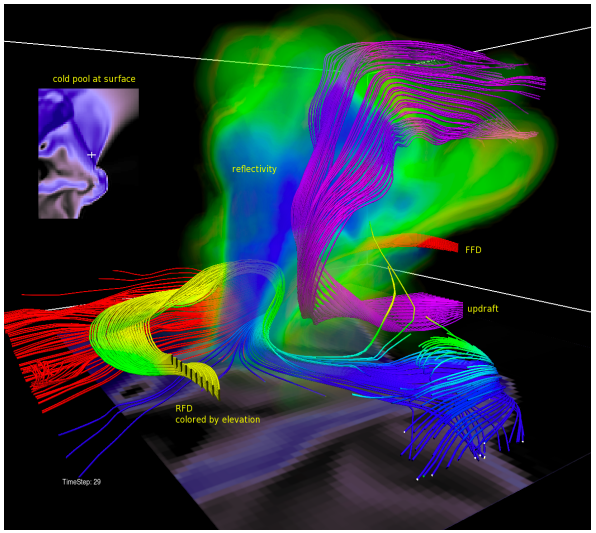
- All 100K processes have to wait for the slowest to complete its write

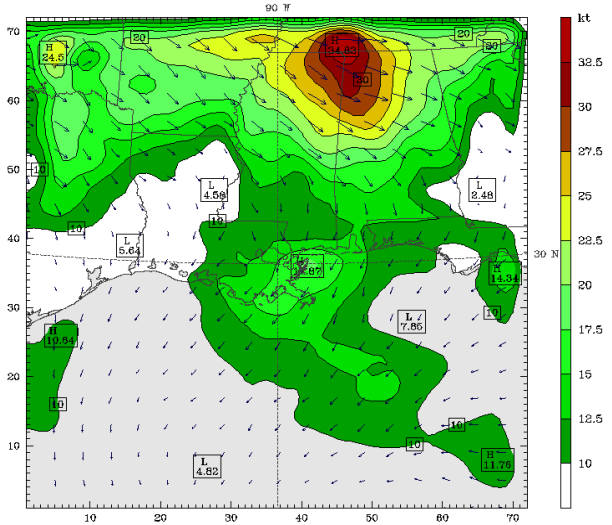




The visualization: VisIt

- Developed at LLNL, Department of Energy (DOE) Advanced Simulation and Computing Initiative (ASCI)
- Relies on VTK and distributed rendering

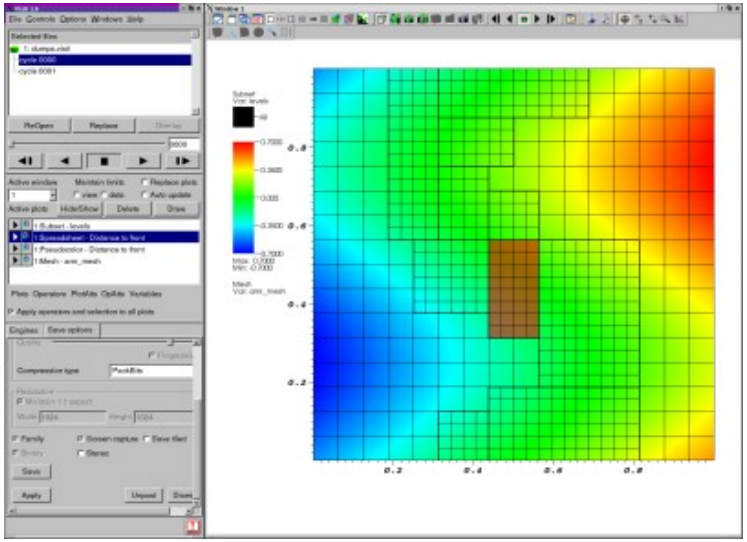




Challenge: how to efficiently cope with various read patterns?

- VisIt parallel rendering engine deployed on 10% of computational resources (rule of thumb)
- A wide variety of purposes → a wide variety of patterns

E.g.: How can 10.000 rendering engines gather and render only horizontal slices of a subset of variables (arrays) stored in more than 100.000 files and accessed through 100 GPFS servers?



Need to filter the data at some point...

This problem is not specific

- **Other simulation applications write one file/core/step and have a similar behavior, such as:**
  - The GTC Fusion Modeling Code (Gyrokinetic Toroidal Code), studying micro-turbulence in magnetic confinement fusion, using plasma theory
  - The Pixie3D code, 3D extension of MHD (Magnetic-Hydro-Dynamics) core
- **The « too-many-files » problem and the problem of post-processing and visualizing them already subject to studies**

Issues and possible solutions

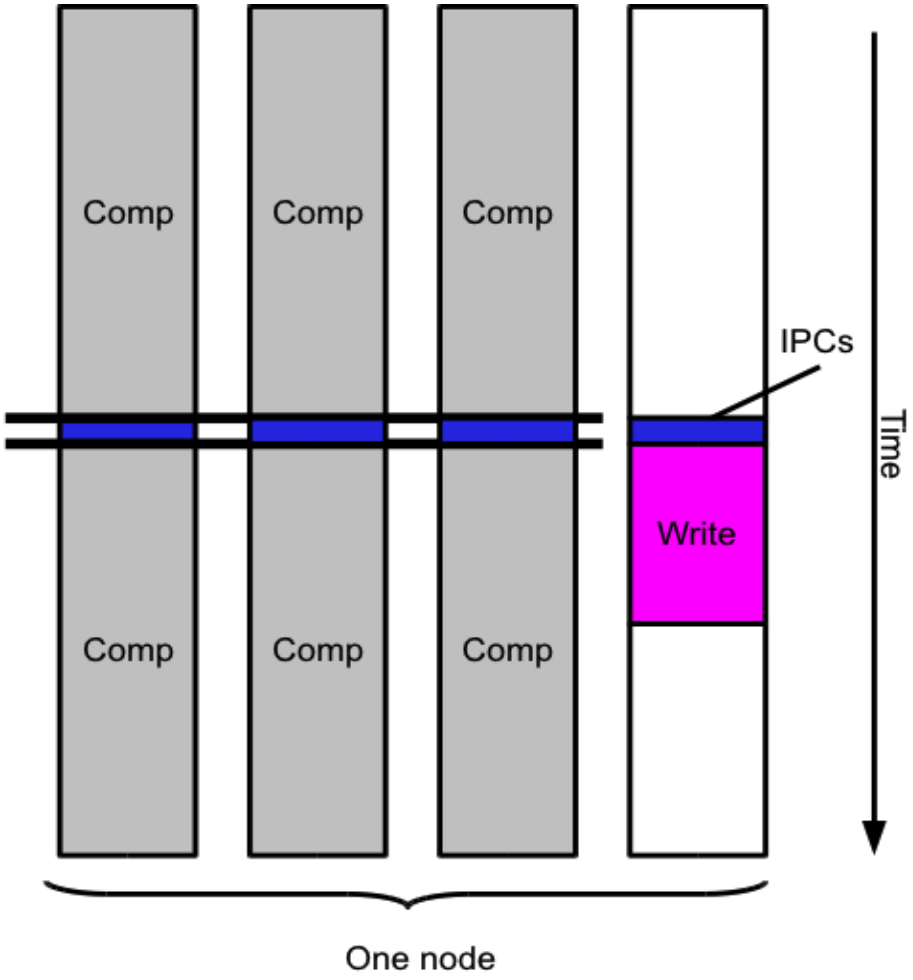
- **Current approach involves:**
  - Inefficient concurrent writing (waiting for the slowest process)
  - Too many files at every step bottleneck in GPFS
  - A read patterns too different from the write patterns
- **Possible solutions:**
  - Using PHDF5 to write in a collective manner, but
    - Still slow compared to individual writes
    - Does not allow any compression
  - Using filters
    - In the simulation → slows down the simulation
    - In the visualization tools → slows down the visualization

What we are looking for

- Optimize simulation/visualization I/O concurrency
  - Reduce overhead in write sections
  - Reduce the overall number of files
  - Adapt the output layout to the way it will be read by the visualization
  - Increase the compression level
  - Leverage the capabilities of HDF5

Proposal: use dedicated I/O cores

- One core per node handles a shared buffer
- The computation cores write in the shared buffer (possible, as CM1 only uses 2% of RAM out of 128GB/node)
- The write phase of the dedicated I/O core overlaps the next computation phase

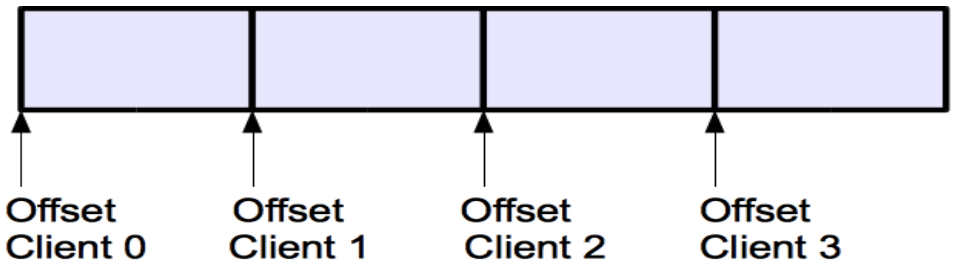


Potential usage of the spare time

- The I/O core may:
  - Filter data
  - Reformat data for more efficient visualization
  - Redistribute data
  - Better compress data
  - Directly handle VisIt requests (inline visualization)
    - Avoid read pressure on GPFS

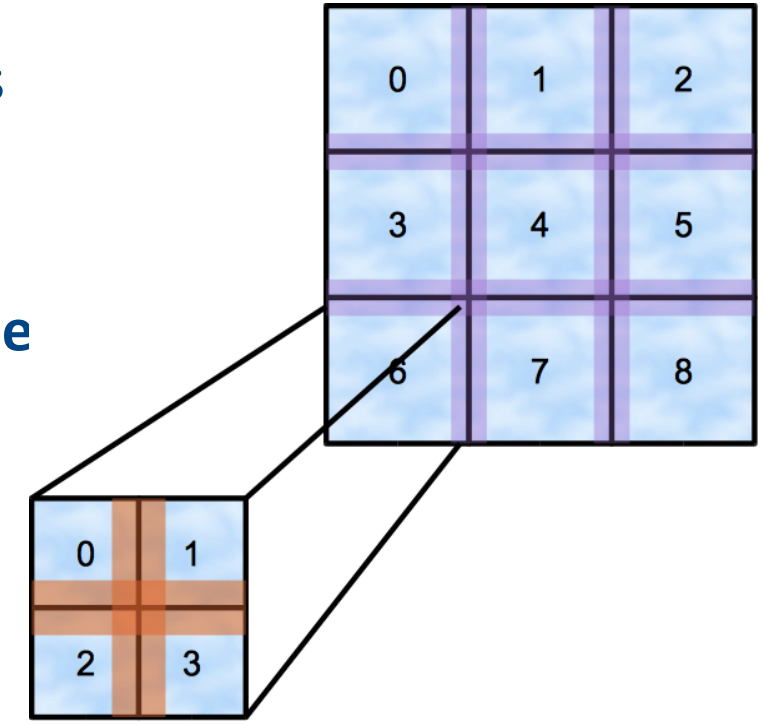
Implementation

- A dedicated MPI process in each node
- IPCs are used to handle communication between each computation core and the I/O core
  - A shared memory segment is opened by the I/O core when the application starts
  - Each computation core writes at a specific offset in the segment (no sharing, no synchronization)



Implementation

- **Goal:** gather data into larger arrays → we want cores to handle contiguous subdomains
- **Some modifications required in the simulation**
  - A new hierarchical layout built from the knowledge of ‘‘node ID'' and ‘‘core ID’’
  - A precise organization of processes within cores and nodes



Experimental settings

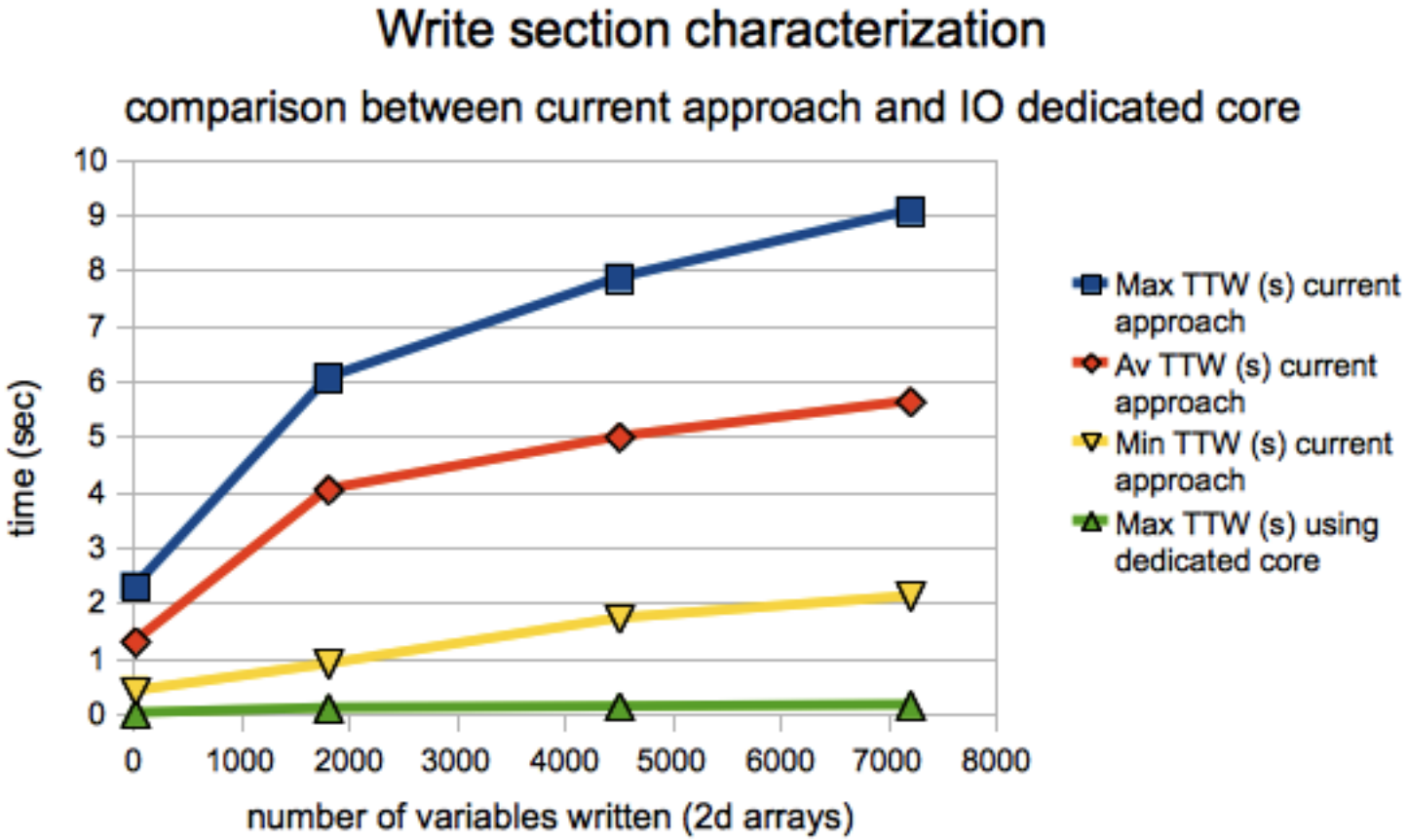
- On BluePrint:
  - 64 nodes, 16 cores/node
  - 2 GPFS servers
- Domain:
  - 960x960x300 pts
- Output example (for 15 variables enabled, uncompressed):
  - Backup every 50 steps (i.e. every 3min)
  - 15 MB/core/backup
  - 1024 files/backup (with current approach)
  - Total: about 16 GB/backup

- On BlueWaters:
  - 3200 nodes, 32 cores/node
  - 100 GPFS servers
- Domain:
  - 12800x12800x1000 pts
- Output example (for 15 variables enabled, uncompressed):
  - Backup every 100 steps (i.e. every 2min)
  - 96 MB/core/backup
  - 102400 files/backup (with current approach)
  - Total: about 10 TB/backup

Data layout

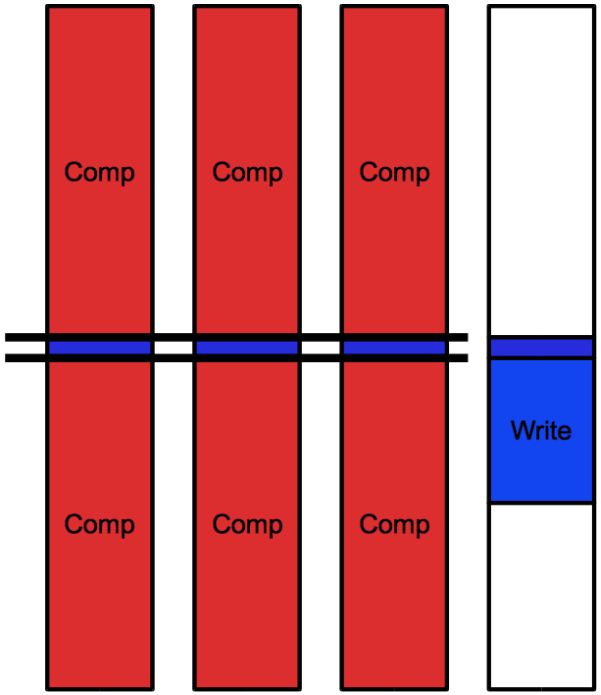
- **Current approach**
  - 16 cores/node participating in computation
  - 32x32 grid layout
  - 30x30x300 pts per subdomain
- **Proposed approach**
  - 15 cores/node participating in computation, 1 IO-core/node
  - 8x8 nodes layout, 5x3 cores layout
    → 40x24 grid layout
  - 24x40x300 pts per subdomain

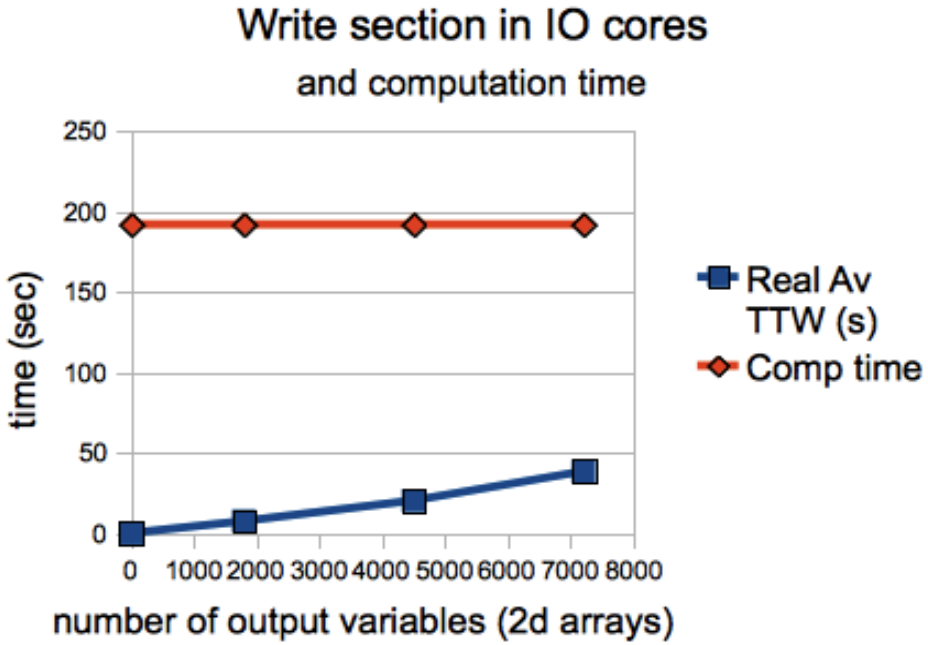
Comparison of the two approaches: seen from computational cores



Theoretical limits

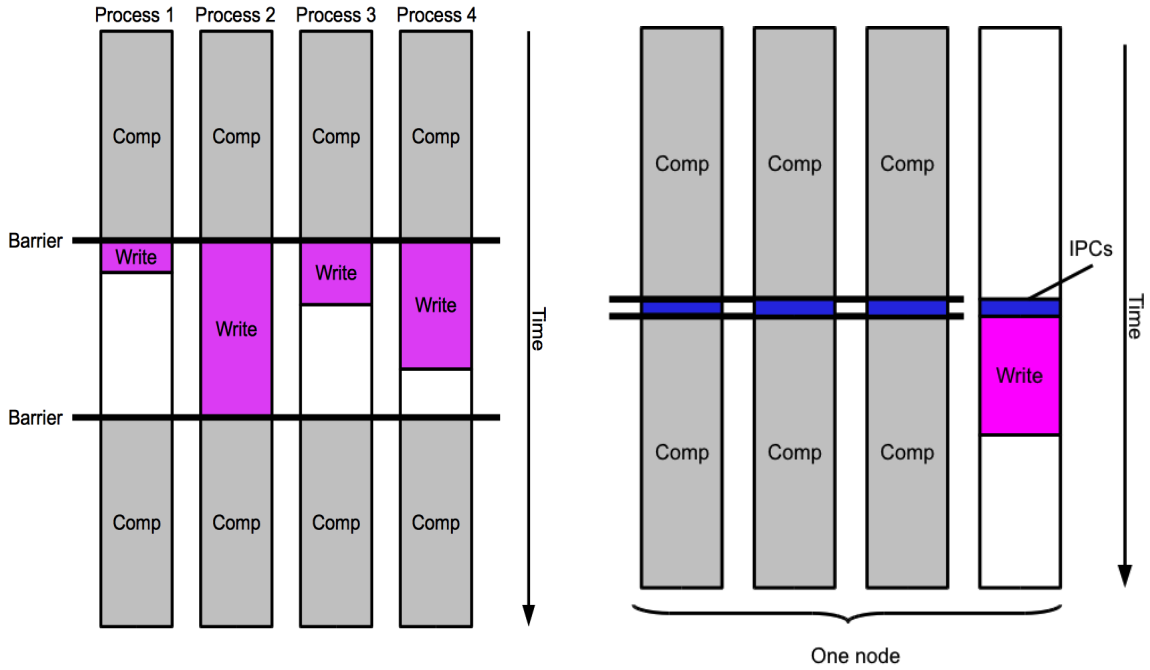
- **First limit:** write time must be less than computation time, in order to overlap the I/O without slowing down the application.





Theoretical limits

- **Second limit:** the gain of overlapping I/O and computation must be worth removing one core from computation



- **Achieved in 1hour: 814 steps with the current approach, 866 with the new one**

Summary: using dedicated I/O cores

- The proposed solution consists in
  - Removing one core per node from computation
  - Using this core to gather, filter and flush the data
- Proof-of-concept implementation
  - Uses MPI processes
  - Opens shared memory segments to handle the IPCs

Contribution at this stage

an approach which...

- Reduces overhead in I/O phases
- Spares time for filtering, compressing and other post-processing
- Globally achieves higher performance on BluePrint

Considering the experiments conducted on BluePrint and the theoretical analysis, we think this approach will also achieve higher performance on BlueWaters

Next steps: evaluation

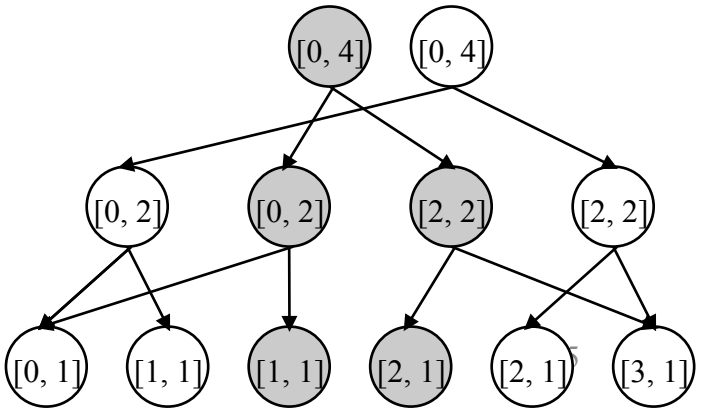
- Very short term: more extensive experiments
  - Hybrid programming models
  - Larger scales
  - Could be used with other back-ends, such as PHDF5
  - Great results with CM1, need to evaluate genericity
    - GTC (nuclear fusion simulation)

Next steps: leverage the I/O cores

- Goal: use the I/O core spare time to reduce visualization I/O
  - The I/O core can stay connected to VisIt for inline visualization
    - Avoid reading GPFS files, directly read from I/O cores
  - Generate summaries (metadata) for specific visualization patterns
    - Some visualization requests could only rely on metadata
    - Reduce concurrent accesses to the shared segment
  - Adaptive metadata management depending on the desired visual output

Next steps: concurrency-optimized metadata management

- Idea: leverage the BlobSeer approach
  - A concurrency-optimized, versioning-based data management scheme (KerData team, INRIA Rennes – Bretagne Atlantique)
    - Relies on concurrency-optimized metadata management
    - Writes generates new data chunks and metadata, no overwriting
    - Multiple versions of the data stay available and can be used for advanced post-processing or visualization
    - Concurrent reads and writes without sync



Next steps: concurrency-optimized metadata management

- Leveraging BlobSeer: what is needed?
  - Adapt BlobSeer's metadata layer to the needs of visualization
  - Efficient GPFS back-end for BlobSeer
- What can we expect?
  - Enhanced support for inline visualization
    - Only from metadata, with no sync
    - From the I/O cores, with no sync
  - Longer term: even less GPFS files overall
    - Initial approach: one file/core/step
    - Our approach: one file/node/step
    - Future: one file/multiple nodes/step

Thank you