8th workshop of the JLPC
November 19th, 2012

I/O and in-situ visualization: recent results with the Damaris approach

Joint work involving Matthieu Dorier, Gabriel Antoniu, Dave Semeraro, Roberto Sisneros, Tom Peterka

Matthieu Dorier
matthieu.dorier@irisa.fr
KerData Team
Inria Rennes, IRISA
ENS Cachan

Outline

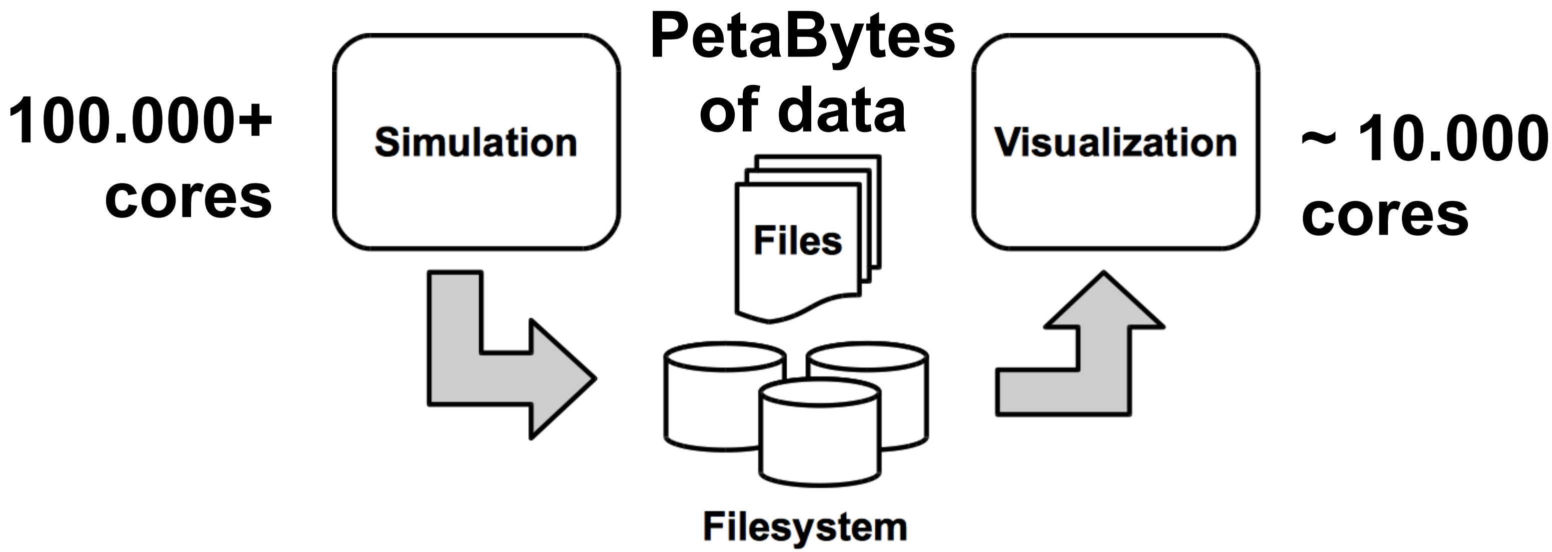
1. From I/O to in-situ visualization
2. Recall on the Damaris approach
3. Past results: achieving scalable I/O
4. Recent work: non-impacting in-situ visualization
5. Demonstration
6. Conclusion

1

From I/O to in-situ visualization

When parallel file systems don't scale anymore

The traditional I/O flow: offline data analysis



- Periodic data generation from the simulation
- Storage in a parallel file system (Lustre, PVFS, GPFS,…)
- Offline data analysis (on another cluster)

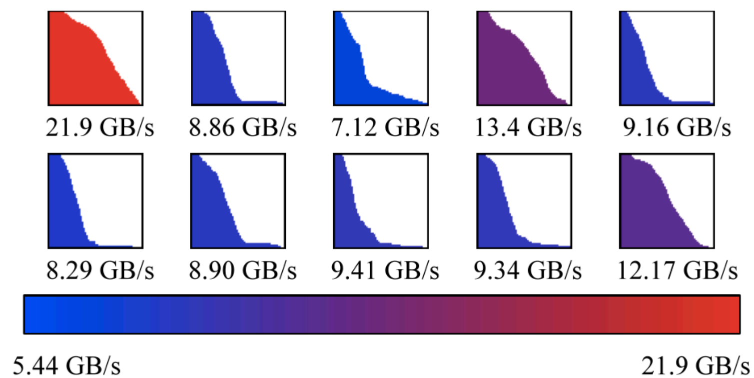
Periodic synchronous snapshots lead to I/O bursts and high variability (jitter)

Input Output

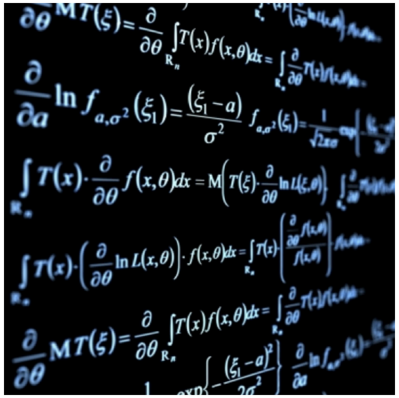
The “cardiogram” of a PVFS data server during a run of the CM1 simulation

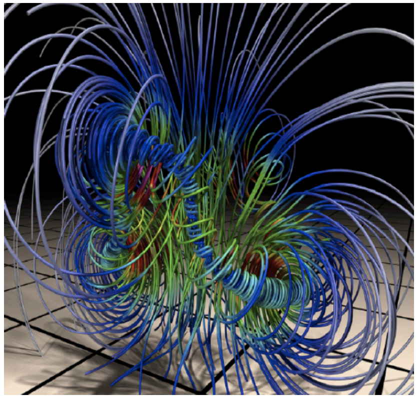
Visualizing throughput variability:

- Between cores
- Between iterations



Big Data challenge on post-petascale machines





- How to efficiently **store**, **move** data?
- How to **index**, **process**, **compress** these data?
- How to **analyze**, **visualize** and finally **understand** them?
- … but wait… why storing anyway?

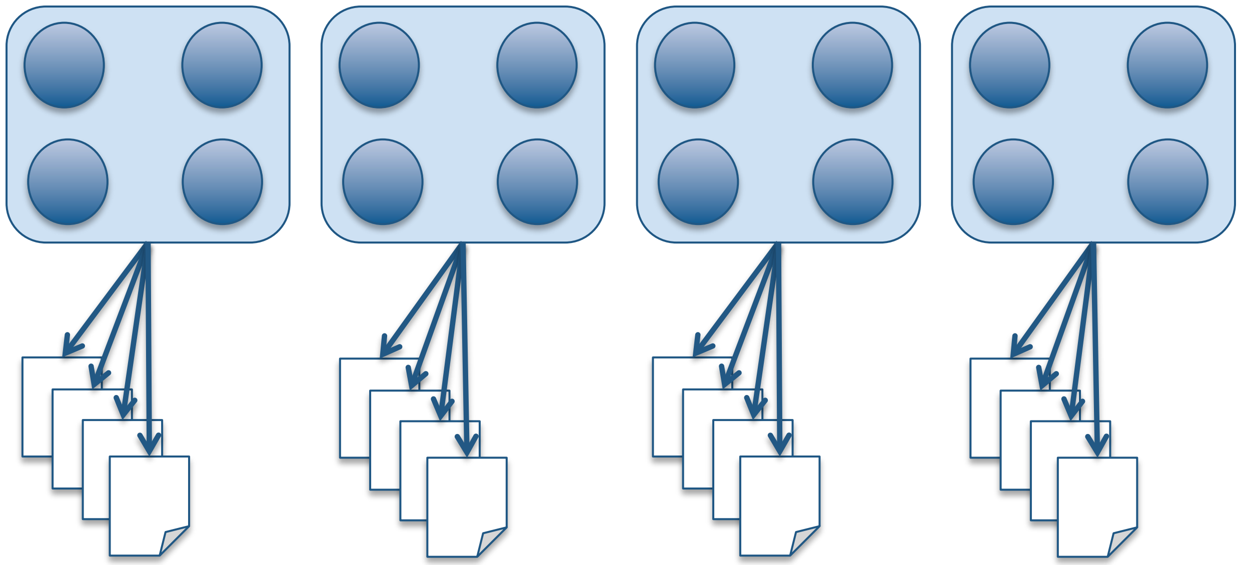
2

Recall on the Damaris approach

Dedicating cores to enable scalable asynchronous I/O

The Damaris approach: dedicated I/O cores

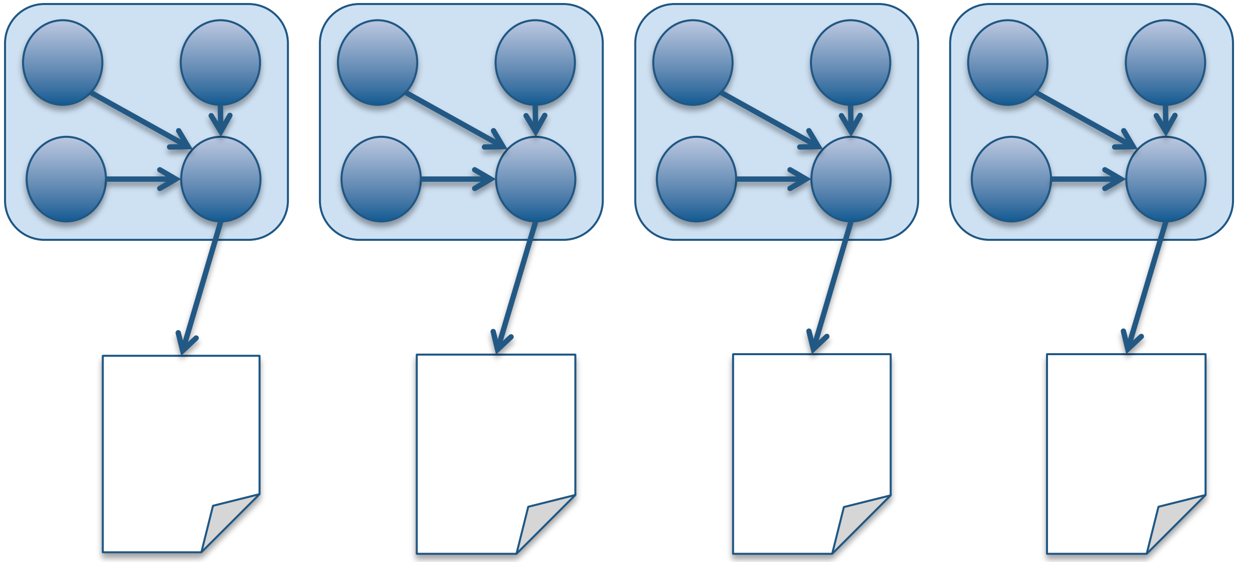
- **Next-generation supercomputers have multicore SMP nodes**



  - Network access contention at the level of a node

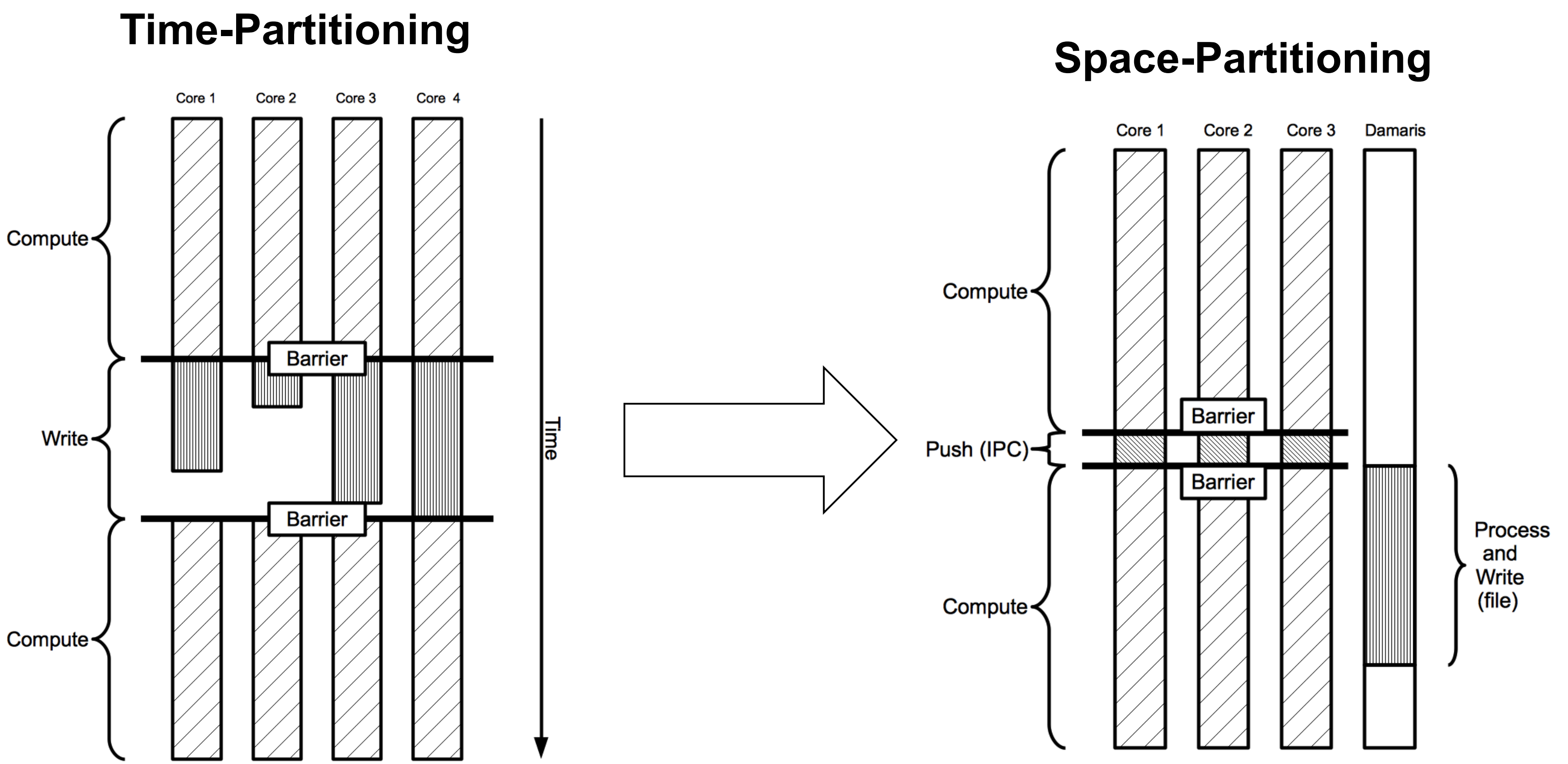
The Damaris approach: dedicated I/O cores

- **Next-generation supercomputers have multicore SMP nodes**



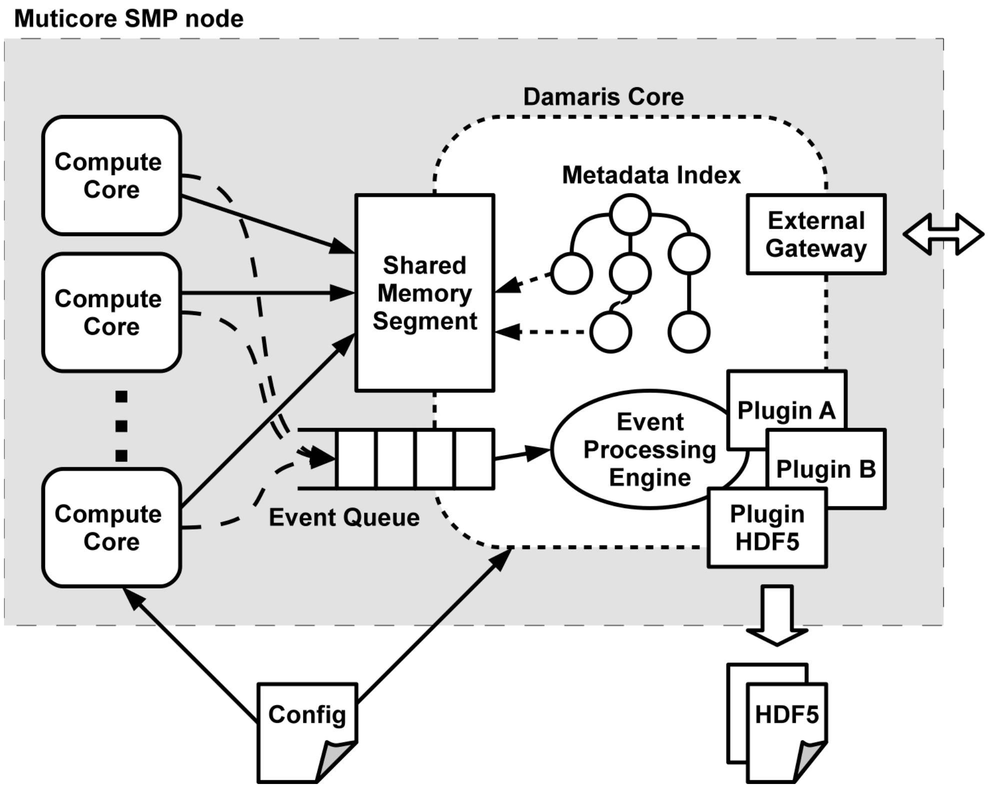
- Network access contention at the level of a node
- Possibility of efficient interactions through shared memory
- One core dedicated for gathering data
- This core only writes

Moving I/O to a dedicated core



Leave a core, go faster!

Damaris at a glance



- *Dedicated Adaptable Middleware for Application Resources Inline Steering*
- **Main idea:** dedicate one or a few cores in each SMP node for data management
- **Features:**
  - Shared-memory-based communications
  - Plugin system (C,C++, Python)
  - Connection to VisIt
  - XML external description of data

Damaris: current state of the software

- **Version 0.6.1** available at http://damaris.gforge.inria.fr/
  - Along with documentation, tutorials and examples
- Written in C++, uses
  - Boost for IPC, Xerces-C and XSD for XML parsing
- API for Fortran, C, C++

- **Tested on**
  - Grid'5000 (Linux Debian), Kraken (Cray XT5 - NICS), JaguarPF (Cray XK6 – Oak Ridge), JYC (Blue Waters testing system: Cray XE6 - NCSA), Intrepid (BlueGene/P – Argonne)
- **Tested with**
  - CM1 (climate), OLAM (climate), GTC (fusion), Nek5000 (CFD)

3

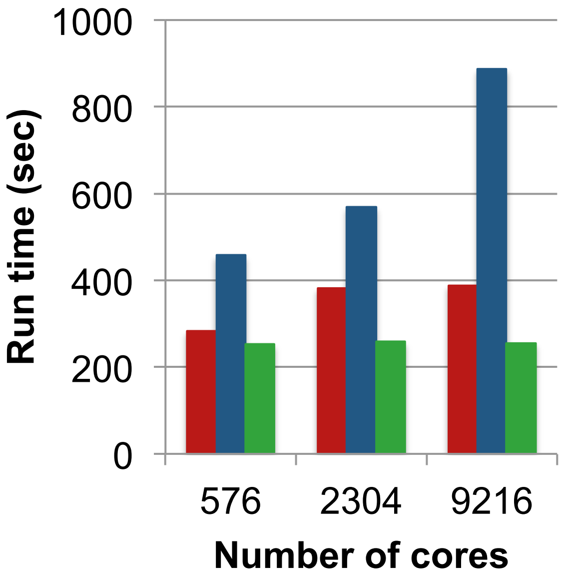
Past results: achieving scalable I/O

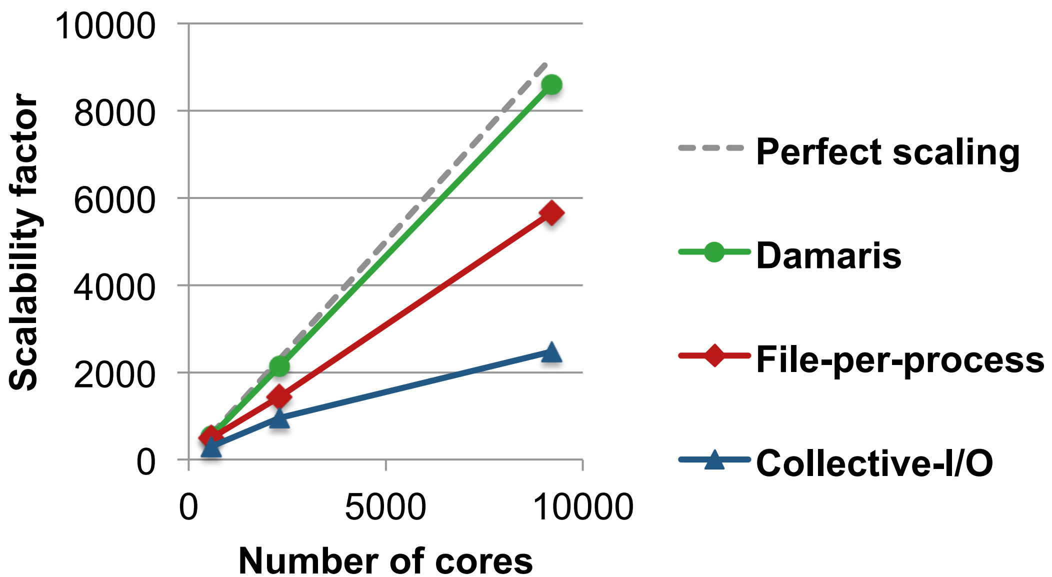
Running the CM1 simulation on Kraken, G5K and BluePrint with Damaris

- **The CM1 simulation**
  - Atmospheric simulation
  - One of the Blue Waters target applications
  - Uses HDF5 (file-per-process) and pHDF5 (for collective I/O)

- **Kraken**
  - Cray XT5 at NICS
  - 12 cores/node
  - 16 GB/node
  - Luster file system
- **Grid 5000**
  - 24 cores/node
  - 48 GB/node
  - PVFS file system
- **BluePrint**
  - Power5
  - 16 cores/node
  - 64 GB/node
  - GPFS file system

Results on I/O with the CM1 application
Damaris achieves almost perfect scalability





Weak scalability factor $S = N \frac{T_{base}}{T}$

N: number of cores
T_{base}: time of an iteration on one core w/o write
T: time of an iteration + a write

More results…

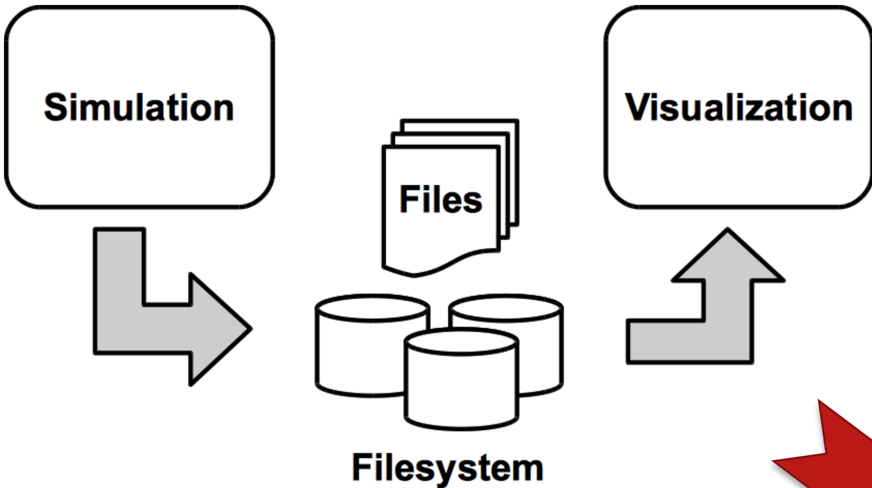
Damaris: How to Efficiently Leverage Multicore Parallelism to Achieve Scalable, Jitter-free I/O
Matthieu Dorier, Gabriel Antoniu, Franck Cappello, Marc Snir, Leigh Orf
Proceedings of IEEE CLUSTER 2012 (Beijing, China)

4

Recent work: non-impacting in-situ visualization

Getting insights from running simulations

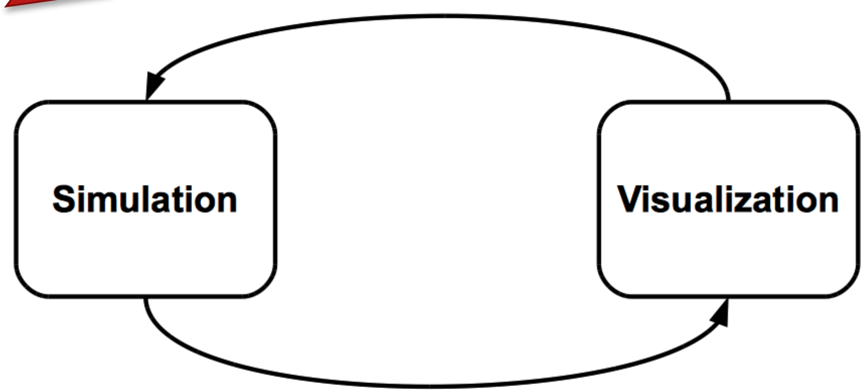
From offline to coupled visualization



- **Offline approach**
  - I/O performance issues in the simulation
  - I/O performance issues in visualization software
  - Too much data!!!

- **Coupled approach**
  - Direct insight in the simulation
  - Bypass the file system
  - Interactive

  BUT
  - Hardly accepted by users



In-situ approaches

- **In-situ = on the same node, collocated with the simulation**
- As opposed to remote visualization

- **Can be:**
  - Time-partitioning
  - Space-partitioning

- **Advantages:**
  - Can work from in-memory data without data movement
  - Can leverage GPUs when the simulation does not use them

- **Drawbacks:**
  - Code instrumentation
  - Impact on performance

Four main goals

Usability

Low impact on simulation code

Adaptability
(to different simulations and visualization scenarios)

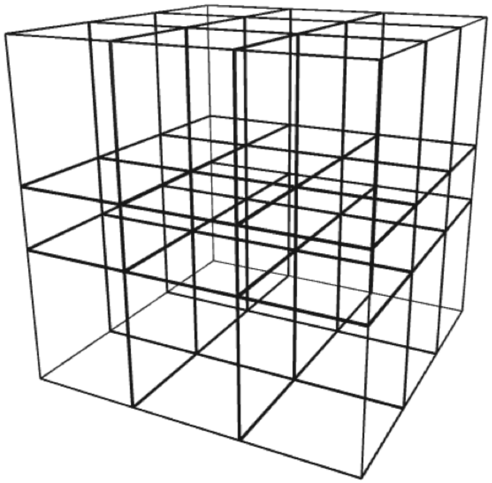
Performance

Low impact on simulation run time

Good resource utilization
(low memory footprint, use of GPU,…)

Driving the acceptance of any in-situ approach

In-situ visualization through Damaris: a glimpse at the interface



```
<parameter name="NX" type="int" value="4"/>
<layout name="px" type="float" dimensions="NX"/>
<variable name="mesh_x" layout="px">
<!-- idem for PTY and PTZ, py and pz, mesh_y and mesh_z -->

<layout name="data_layout" type="double" dimensions="NX,NY,NZ"/>
<variable name="temperature" layout="data_layout" mesh="my_mesh" />

<mesh type="rectilinear" name="my_mesh" topology="3">
   <coord name="mesh_x" unit="cm" label="width" />
   <coord name="mesh_y" unit="cm" label="depth" />
   <coord name="mesh_z" unit="cm" label="height" />
</mesh>
```

- **VisIt** provides the **libsimV2** library for in-situ visualization
- **Damaris** connects the simulation thanks to its **XML** description

```
DC_write("mesh_x",mesh_x);
DC_write("mesh_y",mesh_x);
DC_write("mesh_z",mesh_x);

DC_write("temperature",temperature);
```

- Low impact on code
- Use of dedicated cores
- High adaptability thanks to plugins
- Non-impacting interactivity

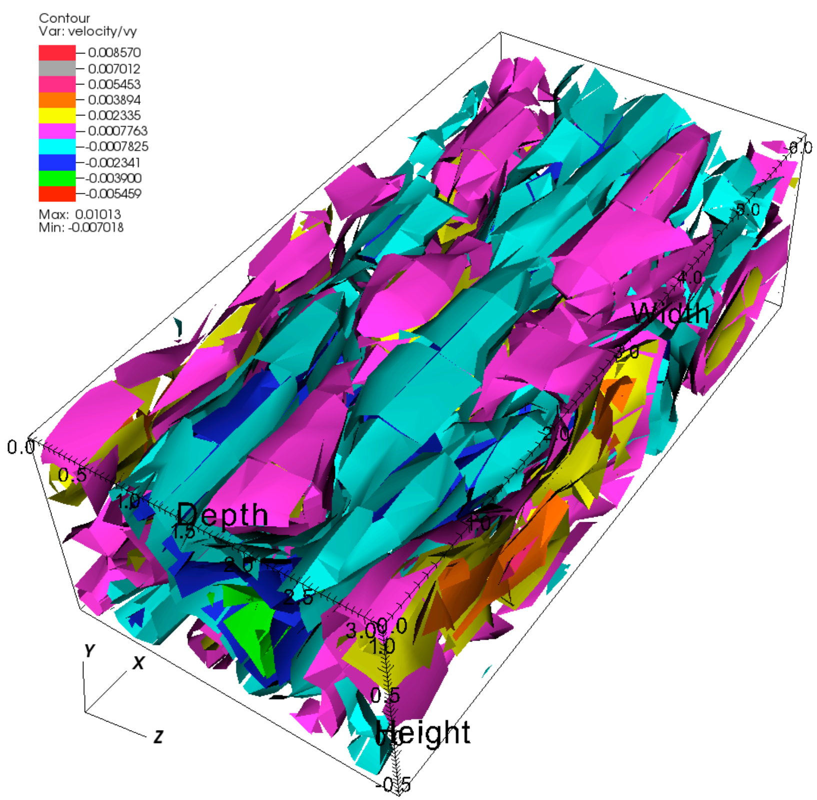
Damaris has a low impact on the code

	VisIt	Damaris
Curve.c	144 lines	6 lines
Mesh.c	167 lines	10 lines
Var.c	271 lines	12 lines
Life.c	305 lines	8 lines

Number of lines of code required to instrument sample simulations with VisIt and with Damaris

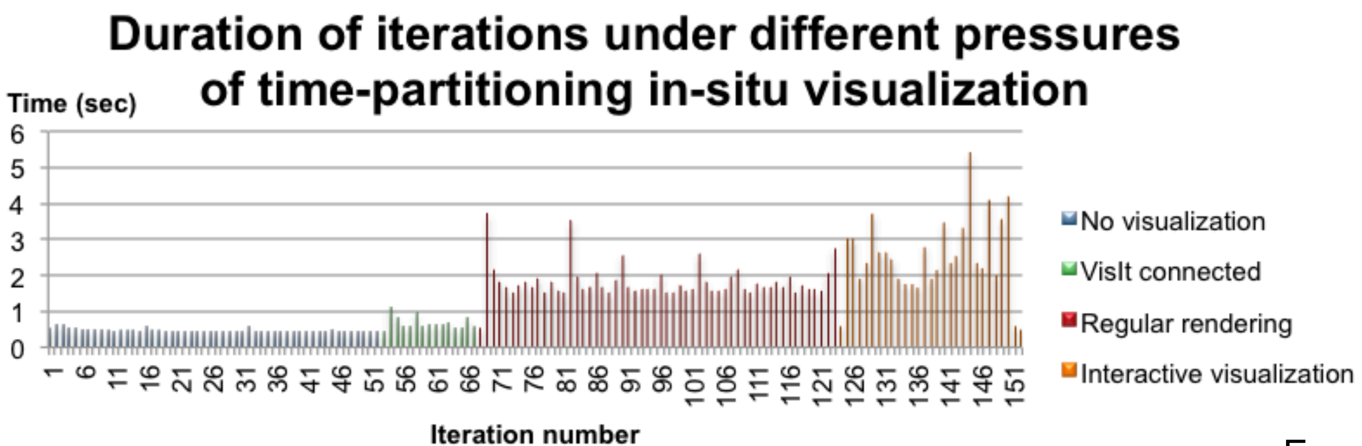
Nek5000	**VTK : 600 lines of C and Fortran**	**Damaris : 20 lines of Fortran, 60 of XML**

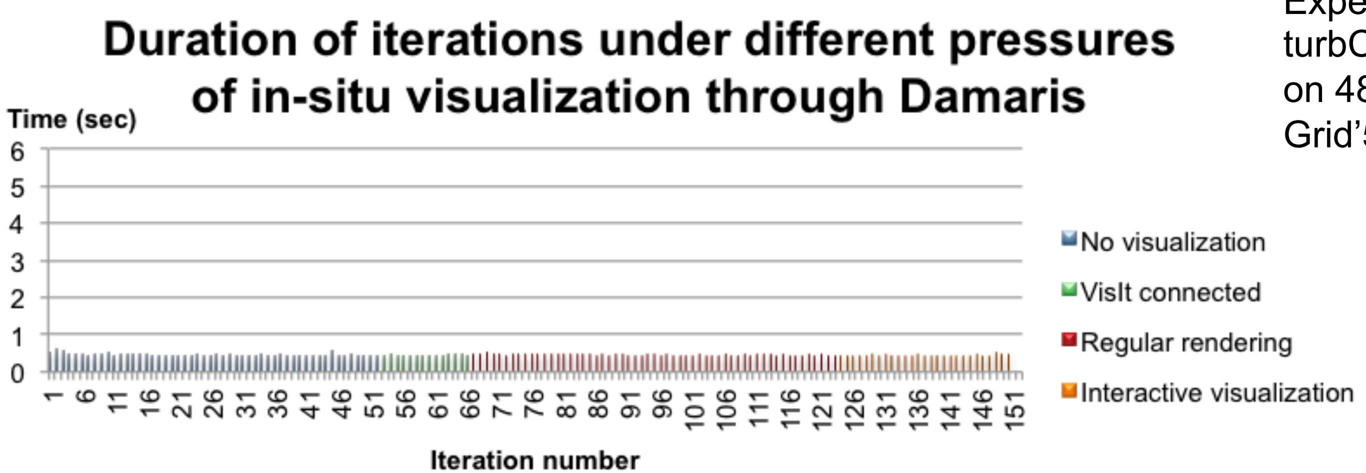
Results with the Nek5000 simulation



- **The Nek5000 simulation**
  - CFD solver
  - Based on spectral elements method
  - Developed at ANL
  - Written in Fortran 77 and MPI
  - Scales over 250,000 cores
- **Data in Nek5000**
  - Fixed set of elements constituting an unstructured mesh
  - Each element is a curvilinear mesh
- Damaris already knows how to handle curvilinear meshes and pass them to VisIt
- **Tested on up to 384 cores with Damaris so far**
- **Traditional approach does not even scale to this number!**

Damaris removes the variability inherent to in-situ visualization tasks





Experiments done with the turbChannel test-case in Nek5000, on 48 cores (2 nodes) of Grid'5000's Reims cluster.

5

Demonstration

Cool images coming

6

Conclusion

Conclusion

The Damaris approach

- Dedicated cores
- Shared memory
- Highly adaptable system thanks to plugins and XML
- Connection to VisIt

Past results

- Fully hides the I/O jitter and I/O-related costs
- 15x sustained write throughput (compared to collective I/O)
- Almost perfect application scalability
- Execution time divided by 3.5 compared to collective I/O

Recent work and results

- Efficient coupling of simulation and analysis tools
- Perfectly hides the run-time impact of in-situ visualization
- Minimal code instrumentation
- High adaptability

8th workshop of the JLPC
November 19th, 2012

Thank you!

I/O and in-situ visualization: recent results with the Damaris approach

Joint work involving Matthieu Dorier, Gabriel Antoniu, Dave Semeraro, Roberto Sisneros, Tom Peterka

Matthieu Dorier
matthieu.dorier@irisa.fr
KerData Team
Inria Rennes, IRISA
ENS Cachan