In-Situ Interactive Visualization of HPC Simulations with Damaris

Joint work involving
Matthieu Dorier, Gabriel Antoniu, Dave Semeraro, Roberto Sisneros and Leigh Orf

Matthieu Dorier
KerData Team
Inria Rennes
ENS Cachan

June 13th 2012
Introduction: when offline visualization does not work anymore...

- **The old fashioned way**
  - Offline visualization:
    - Run your simulation for days
    - Write a bunch of files periodically, using HDF5, NetCDF, etc.
    - Move the files to an analysis cluster
    - Analyze your data
    - Find something scientifically relevant notice the simulation didn’t behave as expected
Introduction: when offline visualization does not work anymore...

- **Motivations**
  - I/O becoming a bottleneck: need to drastically reduce storage demands
  - Longer, more complex simulations: need to reduce the time-to-insight
  - Migrations of data to a visualization cluster become intractable
  - Visualization software also suffer from the I/O bottleneck
  - Need to adapt the output format from simulations to input readable from visualization
Towards inline visualization strategies

• **Loosely coupled strategy**
  - Visualization runs on a separate, remote set of resources
  - Partially or fully asynchronous
  - Solutions include staging areas, file format wrappers (HDF5 DSM, ADIOS, …)

• **Tightly coupled strategy**
  - Visualization is collocated with the simulation
  - Synchronous (time-partitioning): the simulation periodically stops
  - Solution by code instrumentation
  - Memory constrained
Four main goals

**User friendliness**
- 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 approach
Towards inline visualization strategies

<table>
<thead>
<tr>
<th>Coupling</th>
<th>Tight</th>
<th>Loose</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Impact on code</td>
<td>High</td>
<td>Low</td>
<td>Minimal</td>
</tr>
<tr>
<td>Adapatability</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interactivity</td>
<td>Yes</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>Instrumentation</td>
<td>High</td>
<td>Low</td>
<td>Minimal</td>
</tr>
<tr>
<td>Impact on run time</td>
<td>High</td>
<td>Low</td>
<td>Minimal</td>
</tr>
<tr>
<td>Resource usage</td>
<td>Good</td>
<td>Non-optimal</td>
<td>Better</td>
</tr>
</tbody>
</table>

- Researchers seldom accept tightly-coupled in-situ visualization
  - Because of development overhead, performance impact…
  - “Users are stupid, greedy, lazy slobs” [1]

- **Is there a solution achieving all these goals?**

Outline

• Introduction
• In-situ capabilities in diverse software
• Recall on the Damaris approach
• Using Damaris for in-situ visualization
• Conclusion
An overview of VisIt (LLNL)

- Provides the libsim library \(\Rightarrow\) simulation instrumentation
- Data is exposed through a set of callback functions (in C or Fortran)
  - About 10 to 20 lines of code per variable/object to expose
  - Need to re-write the simulation’s mainloop
- Works in a time-partitioning manner: simulation stops
  - If a user connected to the simulation \(\Rightarrow\) interactively answers its requests
- Can work on data in memory without any copy

```c
// This function is called to retrieve the mesh
visit_handle get_mesh_data(int domain, const char *name, void *cbdata) {
    visit_handle h = VISIT_INVALID_HANDLE;
    if(strcmp(name, "my_mesh") == 0) {
        if(VisIt_RectilinearMesh_alloc(&h) == VISIT_OKAY) {
            visit_handle hxc, hyc, hzc;
            VisIt_VariableData_alloc(&hxc);
            // ... idem for hyc and hzc
            VisIt_VariableData_setDataF(hxc, VISIT_OWNER_SIM, 1, NX, mesh_x);
            // ... idem for hyc and hzc
            VisIt_RectilinearMesh_setCoordsXYZ(h, hxc, hyc, hzc);
        }
        return h;
    }
}
```
Other visualization software

- **ParaView**
  - In-situ interface based on VTK, only available in C++
    - Not convenient for Fortran simulations
  - Fixed visualization pipeline, no interactivity
  - Possibility to write the visualization pipeline in Python
    - ParaView can generate a Python script from a user behavior
  - Possibility to redirect the output of the pipeline to a remote cluster (loosely coupled visualization capability)
- **EzViz**
  - Interface in C++ also, functionalities similar to ParaView
- ...
The case of Enzo and YT (UCSD,...)

- **Enzo**: AMR astrophysical simulation developed by UC San Diego and other labs
- Has its own visualization system: **YT**, written in **Python** on top of **Matplotlib**
- Enzo wraps its data into **NumPy structures**, and periodically loads a user-provided Python script

Enzo is an example of simulation for which
- A **specific visualization system** has been designed
- A **specific instrumentation** has been done (using the Python/C interface)
- Yet the Enzo developers admit that **time-partitioning is not appropriate** [1], as it interrupts the simulation
- Moreover, loading Python modules have an increasing impact at large scale [2]

---

[2] Personal experiments on JaguarPF, and private discussions with J. M. Favre
Outline

- Introduction
- In-situ capabilities in diverse software
- **Recall on the Damaris approach**
- Using Damaris for in-situ visualization
- Conclusion
Damaris at a glance

- **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)
  - XML external description of data
Damaris: efficiently leveraging shared-memory

- Two versions of the data
- Direct allocation in shared-memory
  - DC_alloc("varname",iteration)
- Overlap read-access on past iterations
- Avoids copy of data
Results on I/O with the CM1 application
Damaris achieves almost perfect scalability

Weak scalability factor \( S = N \frac{T_{\text{base}}}{T} \)

- \( N \): number of cores
- \( T_{\text{base}} \): time of an iteration on one core with write
- \( T \): time of an iteration + a write
Results on I/O with the CM1 application
Damaris hides the I/O jitter

**Graphs:**
- **BluePrint Power5, 1024 cores**
  - Average, max and min write time
- **Kraken Cray XT5**
  - Average and maximum write time
  - 28MB per process

**Axes:**
- **Duration of the write (sec)**
- **Number of cores**
- **Time to write (sec)**
- **Total amount of data (GB)**

**Lines:**
- **Collective-I/O**
- **File-per-process**
Results on I/O with the CM1 application
Damaris increases effective throughput

Average aggregate throughput from the writer processes

![Graph showing average aggregate throughput from the writer processes. The graph includes lines for File-per-process, Damaris, and Collective-I/O, with throughput values ranging from 0.0625 to 16 GB/s and number of cores ranging from 0 to 10,000.](image-url)
Results on I/O with the CM1 application

Time spent by Damaris writing data and time spent waiting

Damaris spares time? Let’s use it for visualization!
Outline

- Introduction
- In-situ capabilities in diverse software
- Recall on the Damaris approach
- Using Damaris for in-situ visualization
- Conclusion
Let’s take a representative example

// rectilinear grid coordinates
float mesh_x[NX];
float mesh_y[NY];
float mesh_z[NZ];
// temperature field
double temperature[NX][NY][NZ];
“Instrumenting” with Damaris

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

(Yes, that’s all)
Now describe your data in an XML file

```xml
<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 -->
</variable>
<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>
```

- Unified data description for different visualization software
- Damaris translates this description into the right function calls to any such software (right now: Python and VisIt)
- Damaris handles the interactivity by synchronizing and un-synchronizing dedicated cores
Using Damaris plugins system

- Plugins can be written in C, C++ or Python
- Very simple API from the simulation:
  - `DC_signal("event_name",iteration)`
- Different scopes of event: core, node, global
- Events are also exposed to VisIt: users can trigger an event himself from VisIt’s interface ➔ enhanced interactivity

```python
var = damaris.open("temperature")
for chunks in var.select( iteration = 1 )
    print numpy.average(chunks.data)
```

- Plugin system already used to implement an HDF5 persistency layer
- Connection between VisIt and Damaris was initially implemented as a plugin written by Roberto Sisneros (NCSA): very good feedback on usability
  - Now fully integrated within Damaris
- New challenge rising with Python: loading modules from many cores, doesn’t scale
Outline

• Introduction
• In-situ capabilities in diverse software
• Recall on the Damaris approach
• Using Damaris for in-situ visualization
• Conclusion
Conclusion: Using Damaris for in-situ visualization

<table>
<thead>
<tr>
<th>Coupling</th>
<th>Tight</th>
<th>Loose</th>
<th>Damaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>Impact on code</td>
<td>High</td>
<td>Low</td>
<td>Minimal</td>
</tr>
<tr>
<td>Adaptability</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interactivity</td>
<td>Yes</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>Instrumentation</td>
<td>High</td>
<td>Low</td>
<td>Minimal</td>
</tr>
<tr>
<td>Impact on run time</td>
<td>High</td>
<td>Low</td>
<td>Minimal</td>
</tr>
<tr>
<td>Resource usage</td>
<td>Good</td>
<td>Non-optimal</td>
<td>Better</td>
</tr>
</tbody>
</table>

- Impact on code: **1 line per object** (variable or event)
- Adaptability to **multiple visualization software** (Python, VisIt, ParaView, etc.)
  - Plugins, XML
- **Interactivity** through VisIt
- Impact on run time: simulation run time **independent of visualization**
- Resources usage:
  - preserves the “zero-copy” capability of VisIt thanks to **shared-memory**,
  - can **asynchronously** use GPU attached to Cray XK6 nodes
thank you!