HPC visualization of particle like data

CFD & Astrophysics School
Claudio Gheller, CSCS
cgheller@cscs.ch
November 2-4, 2016
How do we squeeze all the information from data?

• **Various tools** offer powerful instrument for automatically analyzing large volumes of data, for classification, association, clustering, etc. (e.g. data mining or statistical tools)

• In general, data analysis is characterized by accurate and sophisticated algorithms that often:
  - scale as $N^2$ or even $N^3$ (non-linear behavior)
  - are complex and computationally expensive
  - cannot be optimized/parallelized (not suitable for HPC system)

But an extremely accurate approach is not always necessary…
The power of visualization…

some problems require a overall data exploration approach, as that provided by visualization…

Visualization offers an intuitive and immediate insight into data

What takes hours for a CPU can take a glance for the human eye!!!

The visualization process plays a fundamental role in understanding data.
Particle simulations

- N-body and SPH method are widely used by the CFD and astrophysics communities
- These methods sample the fluid with points following their dynamics

The biggest N-body simulation so far: 2 trillion particles (Stadel et al. 2016)

SPH simulation of flow over a cylinder
Splotch is a ray-casting algorithm for effective visualization of point-like datasets, based on the solution of the radiative transport equation:

\[
\frac{dI(x)}{dx} = (E_p - A_p I(x))\rho_p(x)
\]

where the density is calculated smoothing the particle quantity on a “proper” neighborhood, by a Gaussian distribution function.
Splotch goals

HIGH QUALITY

3D VISUALIZATION
Visualization offers an intuitive and immediate insight into data

of POINT LIKE SCIENTIFIC DATA

of ANY SIZE
Point data are common in simulations... but not only...

Bigger computers, more and more sophisticated codes, larger instruments...
mean HUGE data...............  

Our approach: develop a specific algorithm for 3D points visualization and rely on supercomputers’ brute force...
Splotch at a glance

• Completely open source and self contained
  No dependencies
  Easy to compile
• Standard C++ based; portable to any platform/compiler
  Easily extensible
  Usable anywhere
  Scriptable
  Support for animations
• Can exploit (almost) any HPC system
  Reduced time to solution
  Big Data (any size?) can be processed

Download from:
https://github.com/splotchviz/splotch
How does Splotch work?

**Main steps:**
- Read data
- Set the view (rasterization step)
- Ray tracing (rendering step)
- Save the image
The rendering procedure: weights

\[
\frac{dI(x)}{dx} = (E_p - A_p I(x)) \rho_p(x)
\]

- Each particle influence a given volume, characterized by a smoothing radius \( R_p \) (just like for SPH)
- A smoothing kernel is defined to distribute a particle associated quantity on the volume. The kernel is currently a gaussian: \( \rho_p(x) \) is calculated
- The sigma of the gaussian is a fraction of \( R_p \)
- The contribution of the particle on a pixel is calculated
The rendering procedure: colors

\[
\frac{dI(x)}{dx} = (E_p - A_p I(x)) \rho_p(x)
\]

- $E_p$ and $A_p$ are the emission and absorption coefficients
- They give the color
- In the current implementation $A_p = f(E_p)$
- $E_p$ = available variable (density, temperature)
- Both color tables and RGB are supported (for the latter RT equation is solved for each component and three variables are loaded)
- Support to full physical motivated rendering is in progress
The rendering procedure: integration

\[
\frac{dI(x)}{dx} = (E_p - A_p I(x)) \rho_p(x)
\]

- Particles are sorted with distance from the camera
- Contribution from each particle are summed along the line of sight
- Integration order can be neglected in case of low optical depth
## The need for HPC. Use case, N-body simulations

<table>
<thead>
<tr>
<th>Pure gravity (N-body):</th>
<th>Size:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^{12}$ particles</td>
<td>$10^{12}$ elements</td>
</tr>
<tr>
<td>Three 3D coordinates + one variable</td>
<td>$4 \times 10^{12}$ elements</td>
</tr>
<tr>
<td>Each variable is a float number: 4 bytes</td>
<td><strong>14.5 TB of MEMORY!</strong></td>
</tr>
</tbody>
</table>

Not for a laptop!!! We need supercomputer

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Speed</th>
<th>Volume</th>
<th>Particles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Millennium I (WMAP1-GAD)</td>
<td>500 /h Mpc</td>
<td>10 billion particles</td>
<td></td>
</tr>
<tr>
<td>Millennium II (WMAP1 -GAD)</td>
<td>100/h Mpc</td>
<td>10 billion particles</td>
<td></td>
</tr>
<tr>
<td>Millenium XXL (WMAP1-GAD)</td>
<td>3 /h Gpc</td>
<td>303 billion particles</td>
<td></td>
</tr>
<tr>
<td>Bolshoi (WMAP7-ART)</td>
<td>250 /h Mpc</td>
<td>8 billion particles</td>
<td></td>
</tr>
<tr>
<td>Multidark (WMAP7-ART)</td>
<td>1 /h Gpc</td>
<td>8 billion particles</td>
<td></td>
</tr>
<tr>
<td>BigMD (WMAP7-GAD)</td>
<td>2.5/h Gpc</td>
<td>56.6 billion particles</td>
<td></td>
</tr>
<tr>
<td>MICE (WMAP5-GAD)</td>
<td>7 /h Gpc</td>
<td>8 billion particles</td>
<td></td>
</tr>
<tr>
<td>Horizon (FR) (WMAP3-RAMSES)</td>
<td>2 /h Gpc</td>
<td>68 billion particles</td>
<td></td>
</tr>
<tr>
<td>Horizon (KR) (WMAP5-GOTPM)</td>
<td>10.7 /h Gpc</td>
<td>372 billion</td>
<td></td>
</tr>
<tr>
<td>DEUS (FR) (WMAP7-RAMSES)</td>
<td>21 /h Gpc</td>
<td>550 billion particles</td>
<td></td>
</tr>
<tr>
<td>JUBILEE (WMAP7-CP3M)</td>
<td>6 /h Gpc</td>
<td>216 billion particles</td>
<td></td>
</tr>
</tbody>
</table>

Just run on Piz Daint
EUCLID 2 trillion particles run (largest ever)
HPC systems: the current (hybrid) scenario

- Multi CPU
- Distributed memory
- Message passing communication

- Multi core
- Shared memory
- Single address space

- Many cores
- Off-load memory management
Splotch: MPI implementation
Hybrid Implementation

- HPC System
- Cabinet
- Node/CPU
- Cores
- GPU
- OpenMP

Data in memory

Processes: 1, 2, 3, 4

Reduce

CSCS

ETH Zürich
GPU/KNL Implementation

Cabinet

Node/CPU

Cores

GPU

KNL

HPC System

CUDA

Processor N

Processor N+1

Processor N+2

Data file

Σ

ETH Zürich

CSCS
Splotch parallel algorithmic challenges

- Load balancing: each particle affects a different number of pixels (up to the whole image)
- Race conditions: parallel threads may want to write the same pixel at the same time

These challenges are particularly serious for many-cores accelerators, since hundreds or thousands of threads works concurrently:

Specific solutions have been implemented for them.
A few results: scalability
A few results: GPU benchmarks

Different parts of the code exploit differently the accelerator

The same algorithm can behave differently in different conditions
A few results: GPU vs. MIC-KNC (but old stuff...)

- **GPU outperforms Xeon Phi** very well in lower radii range.
- Phi performances decrease where a considerable portion of the image is unused.
Splotch: work in progress

1. Development of the physically motivated rendering
2. Improvement of the MPI parallel version
3. Development of a client-server version with web interface
4. Support to in-situ visualization

And now… A few examples (Splotch in action!)
Thank you for your attention.