Serial Visualization Pipeline

Contour

Clip
Parallel Visualization Pipeline

Contour

Clip

Contour

Clip

Contour

Clip
<table>
<thead>
<tr>
<th></th>
<th>Jaguar – XT5</th>
<th>Exascale*</th>
<th>Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>224,256</td>
<td>100 million – 1 billion</td>
<td>~1,000×</td>
</tr>
<tr>
<td>Concurrency</td>
<td>224,256 way</td>
<td>10 billion way</td>
<td>~50,000×</td>
</tr>
<tr>
<td>Memory</td>
<td>300 Terabytes</td>
<td>128 Petabytes</td>
<td>~500×</td>
</tr>
</tbody>
</table>

## Exascale Projection

<table>
<thead>
<tr>
<th></th>
<th>Jaguar – XT5</th>
<th>Exascale*</th>
<th>Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>224,256</td>
<td>100 million – 1 billion</td>
<td>~1,000×</td>
</tr>
<tr>
<td>Concurrency</td>
<td>224,256 way</td>
<td>10 billion way</td>
<td>~50,000×</td>
</tr>
<tr>
<td>Memory</td>
<td>300 Terabytes</td>
<td>128 Petabytes</td>
<td>~500×</td>
</tr>
</tbody>
</table>

### MPI Only?

- **Vis object code + state:** 20MB
- **On Jaguar:** 20MB × 200,000 processes = 4TB
- **On Exascale:** 20MB × 10 billion processes = 200PB!

Exascale Projection

<table>
<thead>
<tr>
<th></th>
<th>Jaguar – XT5</th>
<th>Exascale*</th>
<th>Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>224,256</td>
<td>100 million – 1 billion</td>
<td>~1,000×</td>
</tr>
<tr>
<td>Concurrency</td>
<td>224,256 way</td>
<td>10 billion way</td>
<td>~50,000×</td>
</tr>
<tr>
<td>Memory</td>
<td>300 Terabytes</td>
<td>128 Petabytes</td>
<td>~500×</td>
</tr>
</tbody>
</table>

Visualization pipeline too heavyweight?

On Jaguar: 1 trillion cells → 5 million cells/core
On Exascale: 500 trillion cells → 50K cells/core

Hybrid Parallel Pipeline

Distributed Memory Parallelism

Contour

Shared Memory Parallel Processing

Clip
Threaded Programming is Hard

Example: Marching Cubes

Easy because cubes can be processed in parallel, right?

How do you resolve coincident points?
How do you capture topological connections?

How do you pack the results?
Revisiting the Pipeline

- Lightweight Object
- Serial Execution
- No explicit partitioning
- No access to larger structures
- No state
function ( in  , out  )
Functor

function ( in \ast, out \ast )
**Iteration Mechanism**

- **Engine**
  - foreach element

- **Functor**

**Conceptual Iteration**

**Reality:** Iterations can be scheduled in parallel.
Comparison

Engine
  foreach element
  Functor

Filter
  foreach element
Comparison

Engine
foreach element

Functor 1

Functor 2

Filter 1
foreach element

Filter 2
foreach element
Engine Types: Map

Example Usage: Vector Magnitude
Engine Type: Topological Reduce

Example Usages: Cell to Point, Normal Generation
Engine Types: Generate Geometry

Example Usages: Subdivide, Marching Cubes
Engine Types: Compress

Example Usage: Marching Cubes
Conclusion

• Why now? Why not before?
  – Rules of efficiency have changed.
• Concurrency: Coarse $\rightarrow$ Fine
• Execution cycles become free
• Minimizing DRAM I/O critical
• The current approach is unworkable
  – The incremental approach is unmanageable
• Designing for exascale requires lateral thinking