Octopus
A scalable AMR toolit for astrophysics

Bryce Adelstein-Lelbach[1], Zach Byerly[3], Dominic Marcello[3]
PI: Hartmut Kaiser[1][2], Co-PI: Geoffrey Clayton[3]

[1]: Center for Computation and Technology
[2]: LSU Department of Computer Science
[3]: LSU Department of Physics

STE||AR
stellar.cct.lsu.edu

NSF
Overview of Research

• NSF STAR project: a cross-discipline collaboration between LSU computer scientists and astrophysicists.
  – Primary goal is to facilitate a highly realistic simulation of the merger of two white dwarfs.
  • The study of these binaries is important as they are possible progenitors of a number of astrophysically important objects, such as Type 1a supernovae.
Overview of Research

• Development of new adaptive mesh refinement (AMR) codes utilizing HPX, a framework for message-driven computation, instead of traditional HPC programming mechanisms (MPI, OpenMP, PGAS).
  – Existing unigrid codes are too slow.
    • 0.2 orbits/day running on 1,032 cores.
    • We want to be running hundreds if not thousands of orbits.
  – AMR codes can be many orders of magnitude faster ($10^4 – 10^6$).
    • Doing AMR with MPI is difficult and can face scalability problems, due to the inherently inhomogeneous nature of AMR.
Numerical Methods

• Our group uses 3D Eulerian hydrodynamics codes:
  – Explicit advection scheme
    • Kurganov and Tadmor, 2000, Journal of Computational Physics, 160, 241
  – Finite-volume method
  – Adaptive mesh refinement
    • Multigrid method (for solving Poisson’s equations): Martin and Cartwright, 1996
  – Angular momentum conservation
• Other references:
  – Dominic Marcello’s Ph.D. thesis
$q = 0.20d$

Source: Dominic Marcello, LSU Department of Physics
What’s HPX?

• A general purpose C++ runtime system for parallel and distributed applications of any scale.
• The HPX paradigm prefers:
  – **Asynchronous communication** to hide latencies and contention instead of avoiding them.
  – **Fine-grained parallelism** and an **active global address space** to enable dynamic and heuristic load balancing instead of statically partitioning work.
  – **Local, dependency-driven synchronization** instead of explicit global barriers.
  – Sending **work to data** instead of sending data to work.
Hiding Latency and Contention
(pull model)

Locality 1

Future object

Suspend thread A

Execute another thread

Resume thread A

Locality 2

Execute future:
Thread B

Result is being returned
Hiding Latency and Contention
(push model)

Locality 1
Thread A0 is finished
Execute another thread
Pass the result of thread B to thread A1

Locality 2
Thread B
Thread A1

Asynchronously start thread B, passing it thread A1 as an argument
Asynchronous vs Synchronous

A phone call is a form of synchronous communication.

Texting is a form of asynchronous communication. A text message is a future.
What’s Octopus?

• A general purpose HPX AMR framework.
  – Based heavily on ideas drawn from an existing LSU OpenMP AMR code.
    • Octree-based AMR.
  – Primarily designed for high-resolution, high-accuracy astrophysics hydrodynamics simulations.

• Octopus design:
  – Multi-tiered software architecture to maintain abstraction while supporting domain-specific physics.
  – Policy-driven genericity.
  – Powerful optimizations applied to the generic layers:
    • Timestep size prediction.
    • Time-based refinement.
    • Localization of dependencies
    • Eager computation of fluxes.
Universal Scalability Law

\[ S = \frac{N}{1 + \alpha(N - 1) + \beta N(N - 1)} \]

**S**: Speedup  
**\( \alpha \)**: Contention coefficient  
**\( \beta \)**: Synchronization delay coefficient  
**N**: Number of processors

Set \( \beta = 0 \), and you get Amdahl

\[ S = \frac{N}{1 + \alpha(N - 1)} \]

Maximum scaling point:

\[ N_{\text{max}} = \sqrt{(1 - \alpha)/\beta} \]
Scaling of OpenMP 3D Eulerian Code vs Octopus 3D Eulerian Code

- Measured scaling for 4 LOR with Octopus
- Modeled scaling for 4 LOR with Octopus
- Measured scaling for 5 LOR with Octopus
- Modeled scaling for 5 LOR with Octopus
- Measured scaling for 6 LOR with Octopus
- Modeled scaling for 6 LOR with Octopus
- Measured scaling for 7 LOR with Octopus
- Modeled scaling for 7 LOR with Octopus
- Measured scaling for 4 LOR with OpenMP
- Modeled scaling for 4 LOR with OpenMP
- Measured scaling for 5 LOR with OpenMP
- Modeled scaling for 5 LOR with OpenMP
- Measured scaling for 6 LOR with OpenMP
- Modeled scaling for 6 LOR with OpenMP

Speedup (scaled to 1-core relative to each code)

Cores (logscaled)
# USL Modeling

## Octopus

<table>
<thead>
<tr>
<th>LOR</th>
<th>$\alpha$</th>
<th>$\beta$</th>
<th>$N_{\text{max}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.04840</td>
<td>3.00E-04</td>
<td>56</td>
</tr>
<tr>
<td>5</td>
<td>0.03790</td>
<td>1.00E-04</td>
<td>98</td>
</tr>
<tr>
<td>6</td>
<td>0.04084</td>
<td>6.00E-05</td>
<td>126</td>
</tr>
<tr>
<td>7</td>
<td>0.05069</td>
<td>6.00E-06</td>
<td>397</td>
</tr>
</tbody>
</table>

## OpenMP

<table>
<thead>
<tr>
<th>LOR</th>
<th>$\alpha$</th>
<th>$\beta$</th>
<th>$N_{\text{max}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.40890</td>
<td>1.89E-02</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>0.37560</td>
<td>1.83E-02</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>0.37470</td>
<td>1.62E-02</td>
<td>6</td>
</tr>
</tbody>
</table>
Grain Size

Elapsed Time as a Function of Cores and Grain Size (LoR 2)
vega.cct.lsu.edu, ProLiant DL875 G6 (48 AMD Opteron 8431s)

Elapsed Time (averaged, seconds)

Elapsed Time as a Function of Cores and Grain Size (LoR 4)
vega.cct.lsu.edu, ProLiant DL875 G6 (48 AMD Opteron 8431s)

Elapsed Time (averaged, seconds)