Dendro-GR: Massively Parallel Simulations of Binary Black Hole Intermediate-Mass-Ratio Inspirals

Milinda Fernando, Hari Sundar, Hyun Lim, Eric Hirschmann, David Neilsen

35th Jim Isenberg Pacific Coast Gravity Meeting, 2019, Utah State University Logan, Utah.
Introduction

- Dendro-GR: parallel adaptive framework for computational relativity
- Wavelet adaptive multi-resolution (WAMR)
- Symbolic code generation
- Targets binaries with intermediate mass ratios $50 \leq q \leq 100$
Numerical relativity & Structured adaptivity

- Conventional AMR uses nested boxes
- Boxes don’t naturally capture the geometry of binary black holes
- Computational Inefficiency

For efficient simulations, we need,
- unstructured grids
- supercomputers
Why block adaptivity is not enough?

\[ RG + zip + unzip \]

\[ RG \]

\[ \alpha = \frac{\text{number of octants}}{\text{regular grid octants}} \]

\[ q = 1 \]

\[ q \geq 10 \]
Octree based AMR

- Axis-aligned subdivision of space
- In each non-leaf node has children \( (2^{dim}) \)
- Provides high-levels of adaptivity while enabling simple and efficient data-structures, especially in parallel
**Octree Construction and Partitioning**

- Space Filling Curve (SFC) based partitioning
- SFCs are used to define ordering operator in 3D space
WAMR: Wavelet adaptive multi-resolution

\[ V_0 \]
\[ V_1 \]
\[ V_2 \]
\[ V_3 \]

Wavelet basis

Scaling function

Sparse Representation

\[ W_0 \]
\[ W_1 \]
\[ W_2 \]
\[ W_3 \]

Figures from Holmström (1996)
Dendro-GR

- Wavelet adaptive multiresolution
- Unstructured Octree Grid
- High levels of fine-grained parallelism
- Automatic code-generation via symbolic interface
- Portable and highly-scalable on modern supercomputers
Parallelism

- Distributed memory
- CPU
- Accelerator
- Shared memory
- Vectorization

- MPI
- CUDA, OpenCL
- CUDA, OpenMP
- SSE, AVX
Finite difference on unstructured grids

unzip

zip
Overview: How we solve BSSN equations

Note: BSSNKO equations are applied at the block level.
Automatic code-generation

Currently we support,

- CPUs (sequential and OpenMp)
- AVX (Vectorized code)
- GPUs
Heterogeneous Architectures

- GPUs are very good for SIMD
- CPUs handle inter-processor communication and boundary zones
- GPUs work on interior
- Computation and communication are interleaved

![Diagram of heterogeneous architectures]

- Ghost
- Boundary
- Interior

- Data exchange
- Computation

- $G_{rhs}$
- $G_{rhs}$
- $G_{rhs}$
- $G_{rhs}$
- $G_{rhs}$
- $G_{rhs}$
- $G_{rhs}$
- $G_{rhs}$
- $G_{rhs}$
- $G_{rhs}$
- $G_{rhs}$
- $G_{rhs}$
- $G_{rhs}$
- $G_{rhs}$
- $G_{rhs}$
- $G_{rhs}$
- $G_{rhs}$
- $G_{rhs}$
- $G_{rhs}$
- $G_{rhs}$
- $G_{rhs}$

- Time
Heterogeneous Architectures

- GPUs are very good for SIMD
- CPUs handle inter-processor communication and boundary zones
- GPUs work on interior
- Computation and communication are interleaved
Experiments: Nonlinear Sigma Model

\[ \partial_t^2 \phi - \nabla^2 \phi = -\frac{\sin 2\phi}{r^2} \]

```python
r = symbols('r')

# declare functions
chi = dendro.scalar("chi", "[pp]")
phi = dendro.scalar("phi", "[pp]")

phi_rhs = sum( d2(i, chi) for i in dendro.e_i )
- sin(2*chi)/r**2

chi_rhs = phi
```
Binaries with different mass-ratios

(a) $q = 1$

(b) $q = 10$

(c) $q = 100$

(d) $q = 1$

(e) $q = 10$

(f) $q = 100$
BBH $q = 1$
BBH $q = 10$
BBH $q = 10$
BBH $q = 10$
BBH $q = 10$
BBH $q = 10$
BBH $q = 10$
BBH $q = 10$
BBH $q = 10$
ET Comparison

**DENDRO-GR**

<table>
<thead>
<tr>
<th>dofs (along the diagonal) per core</th>
<th>total dofs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>768M</td>
</tr>
<tr>
<td></td>
<td>384M</td>
</tr>
<tr>
<td></td>
<td>192M</td>
</tr>
<tr>
<td></td>
<td>96M</td>
</tr>
<tr>
<td></td>
<td>48M</td>
</tr>
<tr>
<td></td>
<td>24M</td>
</tr>
<tr>
<td></td>
<td>12M</td>
</tr>
<tr>
<td>48 cores</td>
<td>35.46</td>
</tr>
<tr>
<td>96 cores</td>
<td>19.22</td>
</tr>
<tr>
<td>192 cores</td>
<td>11.23</td>
</tr>
<tr>
<td>384 cores</td>
<td>6.59</td>
</tr>
<tr>
<td>768 cores</td>
<td>3.35</td>
</tr>
<tr>
<td>1536 cores</td>
<td>1.75</td>
</tr>
<tr>
<td>3072 cores</td>
<td>1.04</td>
</tr>
<tr>
<td>250K</td>
<td>0.54</td>
</tr>
<tr>
<td>125K</td>
<td>0.39</td>
</tr>
<tr>
<td>62K</td>
<td>2.06</td>
</tr>
<tr>
<td>31K</td>
<td>1.92</td>
</tr>
<tr>
<td>15K</td>
<td>1.64</td>
</tr>
<tr>
<td>8K</td>
<td>1.29</td>
</tr>
<tr>
<td>4K</td>
<td>0.98</td>
</tr>
</tbody>
</table>

**EINSTEIN Toolkit**

<table>
<thead>
<tr>
<th>dofs (along the diagonal) per core</th>
<th>total dofs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>768M</td>
</tr>
<tr>
<td></td>
<td>384M</td>
</tr>
<tr>
<td></td>
<td>192M</td>
</tr>
<tr>
<td></td>
<td>96M</td>
</tr>
<tr>
<td></td>
<td>48M</td>
</tr>
<tr>
<td></td>
<td>24M</td>
</tr>
<tr>
<td></td>
<td>12M</td>
</tr>
<tr>
<td>48 cores</td>
<td>23.38</td>
</tr>
<tr>
<td>96 cores</td>
<td>12.44</td>
</tr>
<tr>
<td>192 cores</td>
<td>7.13</td>
</tr>
<tr>
<td>384 cores</td>
<td>4.68</td>
</tr>
<tr>
<td>768 cores</td>
<td>3.36</td>
</tr>
<tr>
<td>1536 cores</td>
<td>2.67</td>
</tr>
<tr>
<td>3072 cores</td>
<td>2.27</td>
</tr>
<tr>
<td>250K</td>
<td>23.65</td>
</tr>
<tr>
<td>125K</td>
<td>7.64</td>
</tr>
<tr>
<td>62K</td>
<td>4.87</td>
</tr>
<tr>
<td>31K</td>
<td>4.38</td>
</tr>
<tr>
<td>15K</td>
<td>2.92</td>
</tr>
<tr>
<td>8K</td>
<td>2.51</td>
</tr>
<tr>
<td>4K</td>
<td>2.52</td>
</tr>
<tr>
<td>2.51</td>
<td>3.45</td>
</tr>
<tr>
<td>2.16</td>
<td>2.92</td>
</tr>
<tr>
<td>2.08</td>
<td>2.31</td>
</tr>
<tr>
<td>2.04</td>
<td>2.92</td>
</tr>
<tr>
<td>2.08</td>
<td>2.92</td>
</tr>
<tr>
<td>2.04</td>
<td>2.92</td>
</tr>
</tbody>
</table>
Weak Scaling on Titan

![Graph showing weak scaling results for different problem sizes (p) on Titan, with time in microseconds (μs) on the y-axis and p on the x-axis. The graph is color-coded to represent different components: communication, unzip, rhs, derivatives, wavelets, total time, total dofs (zipped), and dofs per core. Each bar is divided into sections representing these components.]

- **communication (μs):** 1.65, 3.21, 3.94, 4.91, 5.75, 5.52, 5.73, 5.46, 6.23, 6.08, 5.63, 6.05, 6.32
- **unzip (μs):** 2.72, 1.95, 1.10, 0.73, 1.00, 0.88, 0.93, 0.83, 0.75, 0.86, 0.91, 0.97, 0.86
- **rhs (μs):** 1.65, 1.85, 1.87, 1.99, 1.85, 1.94, 2.14, 2.17, 1.96, 2.12, 1.64, 1.74, 1.82
- **derivatives (μs):** 0.50, 0.55, 0.57, 0.63, 0.57, 0.60, 0.82, 0.82, 0.63, 0.84, 0.54, 0.58, 0.61
- **wavelets (μs):** 0.18, 0.05, 0.06, 0.11, 0.14, 0.17, 0.18, 0.25, 0.30, 0.33, 0.52, 0.56, 0.40
- **Total time (μs):** 6.7, 7.6, 7.5, 8.4, 9.3, 9.1, 9.8, 9.5, 9.9, 10.2, 9.2, 9.9, 10.0
- **Total dofs (zipped):** 48.6M, 92.9M, 187.7M, 342.1M, 694.0M, 1.4B, 2.8B, 5.6B, 11.0B, 21.7B, 46.4B, 93.9B, 205.8B
- **dofs per core:** 1.52M, 1.45M, 1.47M, 1.34M, 1.36M, 1.40M, 1.35M, 1.36M, 1.34M, 1.33M, 1.42M, 1.43M, 1.57M
Strong Scaling on Titan

\[\text{time (s)}\]

<table>
<thead>
<tr>
<th>p</th>
<th>4K</th>
<th>8K</th>
<th>16K</th>
<th>32K</th>
<th>65K</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.31</td>
<td>8.76</td>
<td>3.86</td>
<td>1.85</td>
<td>0.99</td>
<td></td>
</tr>
<tr>
<td>3.62</td>
<td>1.92</td>
<td>1.59</td>
<td>1.00</td>
<td>0.66</td>
<td></td>
</tr>
<tr>
<td>5.68</td>
<td>3.12</td>
<td>1.64</td>
<td>0.88</td>
<td>0.45</td>
<td></td>
</tr>
<tr>
<td>2.93</td>
<td>1.69</td>
<td>1.15</td>
<td>0.67</td>
<td>0.37</td>
<td></td>
</tr>
<tr>
<td>30.5</td>
<td>15.5</td>
<td>8.2</td>
<td>4.4</td>
<td>2.5</td>
<td></td>
</tr>
</tbody>
</table>
Open source

- Dendro-GR is available on Github. (https://github.com/paralab/Dendro-GR)
- Dendro-GR builds with CMake. Requires MPI and GSL. CUDA optional for GPU support.
- Public version
  - Wave Equation
  - Maxwell Equations* (Baumgartes BSSN-like formulation)
  - BSSN Equations
- Support FEM in addition to FD
  - DG support coming soon
  - Extensively used for CFD
Dendro: Octree + Wavelet Adaptive Multiresolution (WAMR)
Scaling to $10^5$ cores with refinement
Conventional finite difference/finite volume numerical methods
Applications: Relativistic fluids and the BSSN equations
Currently testing binary black hole simulations
In Future, add the relativistic fluid module

Thank You!
Gravitational waves (GW) extraction using $\psi_4$

\[
\psi_4 = (-R_{ij} - KK_{ij} + K_{ik}K^k_j + i\epsilon^{kl}_i \nabla_k K_{lj}) \tilde{m}^i \tilde{m}^i
\]

$\psi_4$ can be expanded in spin$(s)=2$, spin weighted spherical harmonics.

- Note that, $\psi_4 = \mathcal{O}(1/r^2)$
- $\psi_4 = \sum_{l \geq 2, |m| \leq l} \psi_{4lm} - 2Y^{lm}$ where, $\psi_{4lm} = \int_{S^2} \psi_4 \times -2Y^{lm} d\Omega$
- Need to compute $\lim_{r \to \infty} r\psi_{4lm}$
GW extraction on adaptive octrees

- We use Lebedev quadrature to evaluate $\psi_4^{lm}$
- Use TreeSearch to find the octant which has Lebedev quadrature points.
- To evaluate, $\lim_{r \to \infty} r \psi_4^{lm}$ extract $\psi_4^{lm}$, at sequence of spheres, and solve following data fitting problems,
  
  $$ |\psi_4^{lm}(t_*, r_k)| \to \sum_{n=1}^{N} A_n(t_*)/r^n $$
  
  $$ \arg(\psi_4^{lm}(t_*, r_k)) \to \sum_{n=1}^{N} \phi_n(t_*)/r^n $$

Then $\lim_{r \to \infty} r \psi_4^{lm} = A_1 e^{i\phi_0}$, For more details, refer, Extraction of Gravitational Waves in Numerical Relativity by Bishop, N and Rezzolla, L. (https://arxiv.org/abs/1606.02532)