Dendro-GR : A GPU-Accelerated AMR Solver for Gravitational Wave Propagation

Milinda Fernando

Collaborators: David Neilsen, Hari Sundar, Eric Hirschmann, Yosef Zlochower, Omar Ghattas, George Biros, William Black, David Van Komen, Andrew Carroll

North American Einstein Toolkit Workshop 2024, June 6th 2024.
Baton Rouge, LA

The University of Texas at Austin
Oden Institute for Computational Engineering and Sciences

Motivation

- Computing resources have grown exponentially
  - heterogeneous
  - increased complexity to explore fined-grained parallelism
- Numerical relativity
  - GW data analysis $\Rightarrow$ GW wave templates generated by **Numerical relativity** + other approximate models
  - **NR is computationally expensive (weeks to months) and thousands of waveforms are needed to tune/verify low-fidelity models**
- Can we **efficiently** use GPUs for GR simulations ?
  - Adaptive refinement (unstructured memory accesses)
  - Memory-bound computations

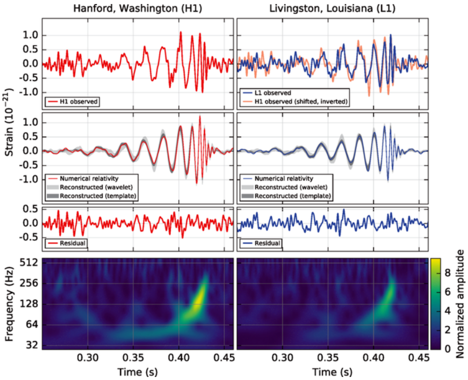
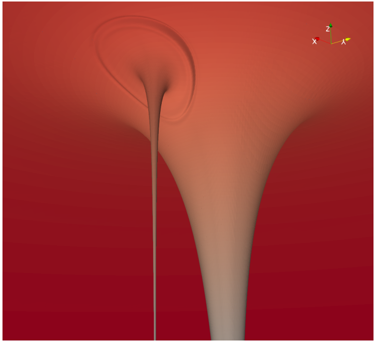


Image : Abbott, B.P., Abbott, R., Abbott, T.D., Abernathy, M.R., Acernese, F., Ackley, K., Adams, C., Adams, T., Addesso, P., Adhikari, R.X. and Adya, V.B., 2016. Observation of gravitational waves from a binary black hole merger. Physical review letters, 116(6), p.061102

Challenges in numerical relativity

- Complexity of the underlying equations (i.e., 24+ DOFs per grid point)
- Long time horizon simulation
- Memory bound computation
- AMR for black hole singularities with increasing mass ratios

mass-ratio $q = m_1/m_2$	Δx_{min} (BH1)	Δx_{min} (BH2)	time (M)	timesteps
1	8.33e-03	8.33e-03	650	7.8e4
4	3.33e-03	1.33e-02	700	2.1e5
16	9.80e-04	1.57e-02	1 400	1.4e6
64	2.56e-04	1.64e-02	6 000	2.3e7



We use BSSN formulation

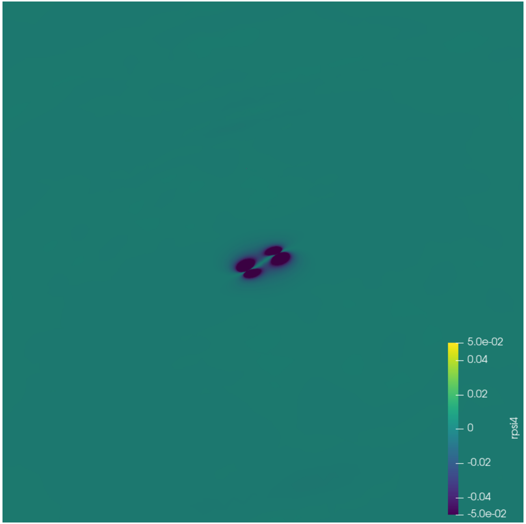
- 24 coupled nonlinear hyperbolic PDEs
- 4 elliptical constraint equations (free evolution with constraint monitoring)

$$
\begin{aligned}
\partial_t \alpha &= \mathcal{L}_\beta \alpha - 2\alpha K, \\
\partial_t \beta^i &= \beta^j \, \partial_j \beta^i + \frac{3}{4} f(\alpha) B^i, \\
\partial_t B^i &= \partial_t \tilde{\Gamma}^i - \eta B^i + \beta^j \, \partial_j B^i - \beta^j \, \partial_j \tilde{\Gamma}^i, \\
\partial_t \tilde{\gamma}_{ij} &= \mathcal{L}_\beta \tilde{\gamma}_{ij} - 2\alpha \tilde{A}_{ij}, \\
\partial_t \chi &= \mathcal{L}_\beta \chi + \frac{2}{3} \chi \left(\alpha K - \partial_a \beta^a \right), \\
\partial_t \tilde{A}_{ij} &= \mathcal{L}_\beta \tilde{A}_{ij} + \chi \left(-D_i D_j \alpha + \alpha R_{ij} \right)^{TF} + \\
&\quad \alpha \left(K \tilde{A}_{ij} - 2 \tilde{A}_{ik} \tilde{A}^k_{\ j} \right),
\end{aligned}
$$

$$
\begin{aligned}
\partial_t K &= \beta^k \partial_k K - D^i D_i \alpha + \\
&\quad \alpha \left(\tilde{A}_{ij} \tilde{A}^{ij} + \frac{1}{3} K^2 \right), \\
\partial_t \tilde{\Gamma}^i &= \tilde{\gamma}^{jk} \partial_j \partial_k \beta^i + \frac{1}{3} \tilde{\gamma}^{ij} \partial_j \partial_k \beta^k + \beta^j \partial_j \tilde{\Gamma}^i - \\
&\quad \tilde{\Gamma}^j \partial_j \beta^i + \frac{2}{3} \tilde{\Gamma}^i \partial_j \beta^j - 2 \tilde{A}^{ij} \partial_j \alpha + \\
&\quad 2\alpha \left(\tilde{\Gamma}^i{}_{jk} \tilde{A}^{jk} - \frac{3}{2\chi} \tilde{A}^{ij} \partial_j \chi - \frac{2}{3} \tilde{\gamma}^{ij} \partial_j K \right)
\end{aligned}
$$

BSSN PDEs: Discretization

- We extend our work on Dendro-GR[1]
- We are simulating binary black hole inspirals, merger, and GWs emitted
- We use adaptive octrees for discretization of the spatial domain
- **Space**: 6^{th} order finite differences
- **Time**: explicit RK4

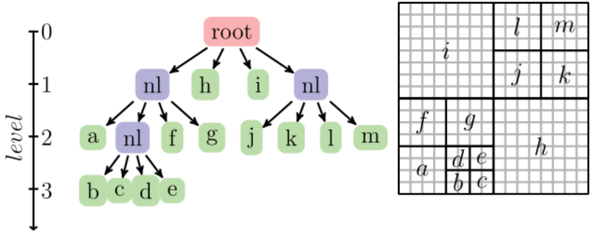


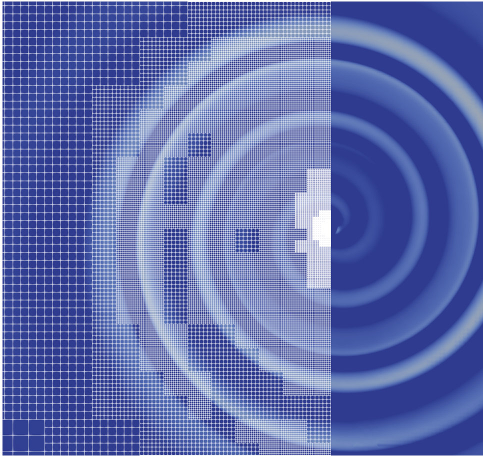
[1] Fernando, M., Neilsen, D., Lim, H., Hirschmann, E. and Sundar, H., 2019. Massively Parallel Simulations of Binary Black Hole Intermediate-Mass-Ratio Inspirals. SIAM Journal on Scientific Computing, 41(2), pp.C97-C138.

Dendro-GR framework

- **Octree**-based adaptive discretization
- **Localized adaptivity**: Spatial refinement is entirely governed by Wavelet transform of underlying fields (user-specified)
- **SymPyGR**: Symbolic code generation framework to support hardware-specific code generation for NR
- **Load-balancing & Partitioning**: SFC-based partitioning scheme for balancing work and communication costs
- Supports for **CPUs** and **GPUs** evolution (directly extends to any dynamical system with FD + explicit time-stepping)
- **Scalability**: Have shown good scalability across 262K cores on TACC's Frontera
- **Open source**
  - Dendro-5.01: Octree-based PDE solver with FD, FE discretization (`https://github.com/paralab/Dendro-5.01`)
  - Dendro-GR: Computational relativity framework (`https://github.com/paralab/Dendro-GR`)

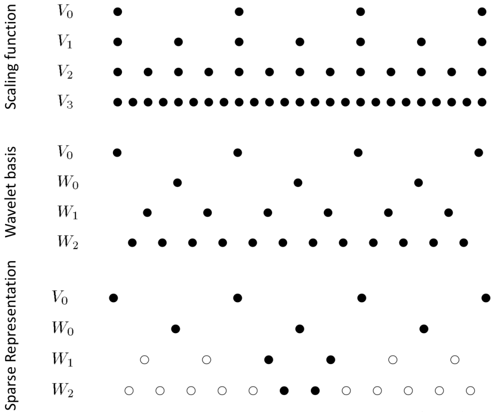
Octrees

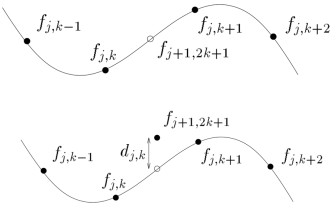


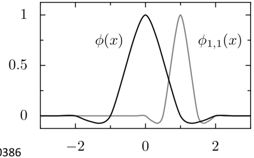


- Axis-aligned subdivision of space
- In each non-leaf node has children (2^{dim})
- Provides high-levels of adaptivity while enabling simple and efficient data-structures,especially in parallel

Wavelet Adaptive Multiresolution

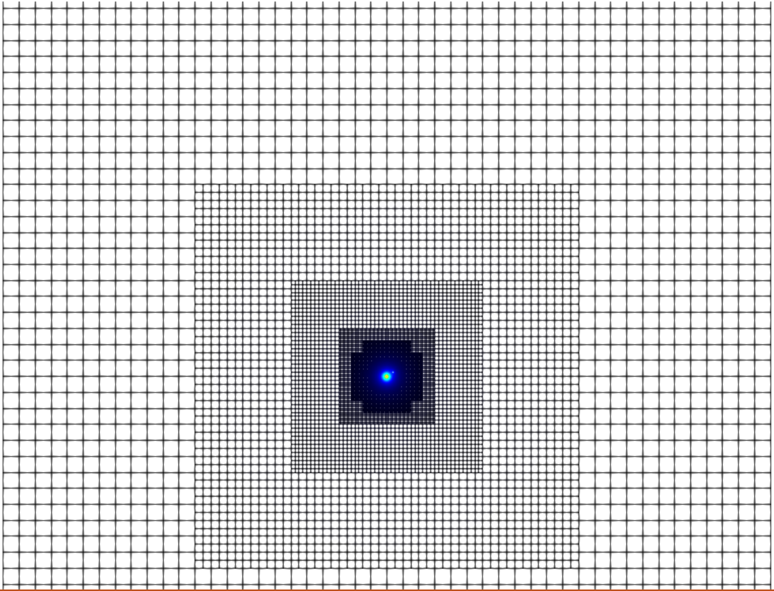




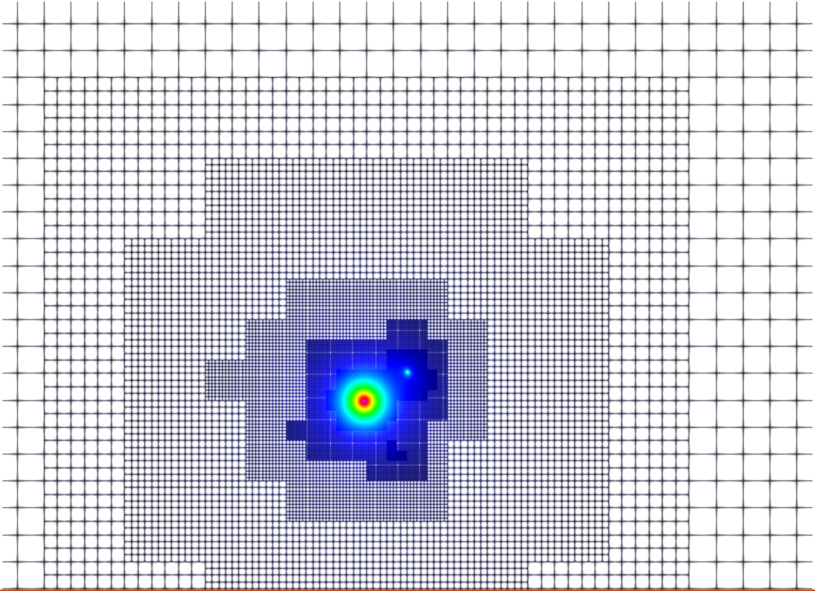


Figures from Holmström (1996) and arXiv:1512.00386

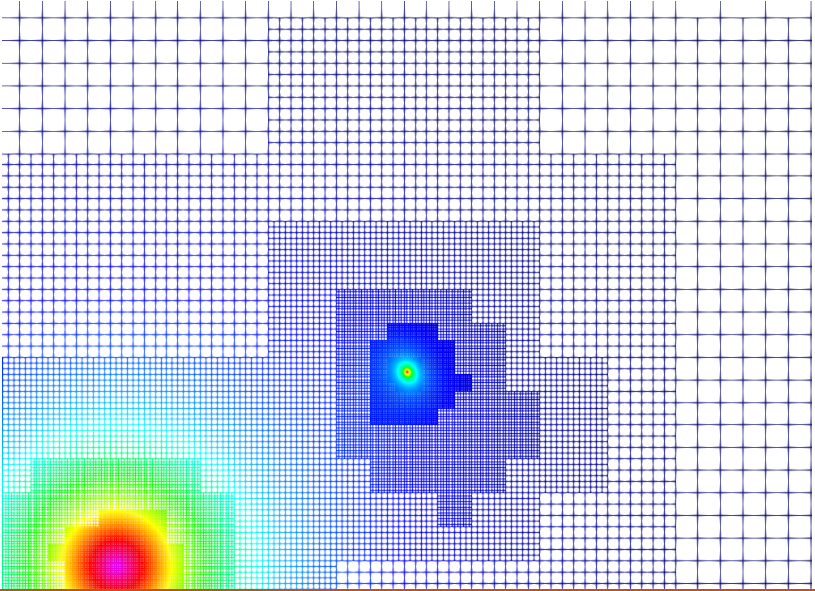
Wavelet based AMR for BBH $q = 16$



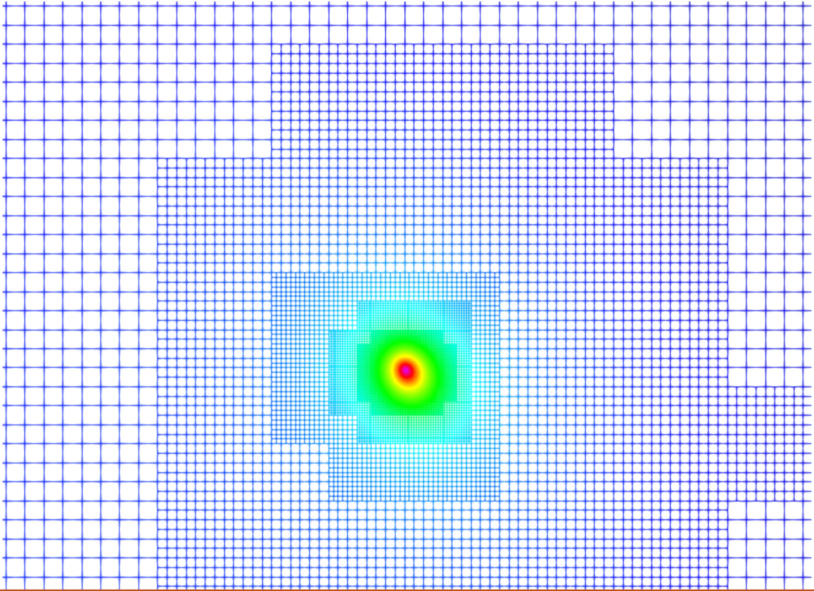
Wavelet based AMR for BBH $q = 16$



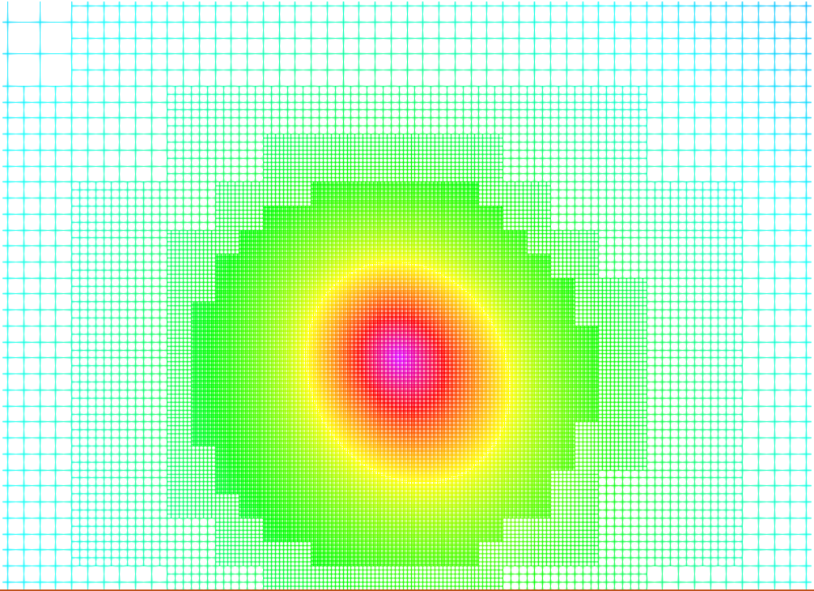
Wavelet based AMR for BBH $q = 16$



Wavelet based AMR for BBH $q = 16$

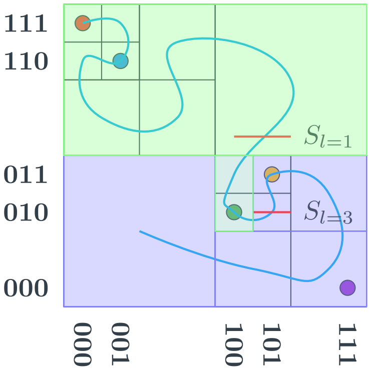


Wavelet based AMR for BBH $q = 16$



Distributed memory octree construction & partitioning

$\tau \leftarrow \Gamma$
for $p_i \in \tau$ **do**
    $\tau_c \leftarrow bucket(p_i)$
$reorder(\tau_c, SFC)$
for τ_c of τ **do**
    if $|\tau_c| > 1$ **then**
        $recurse(\tau_c)$
return

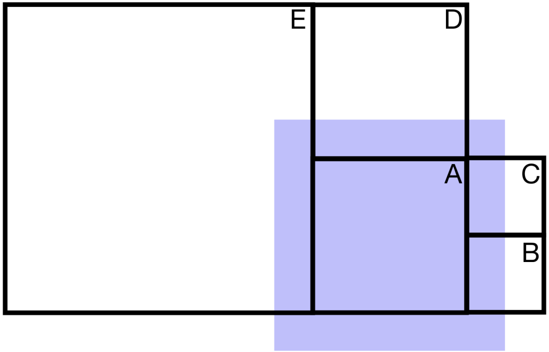


$T(n) = \mathcal{O}(nk)$ where $k \leq log_2(n)$

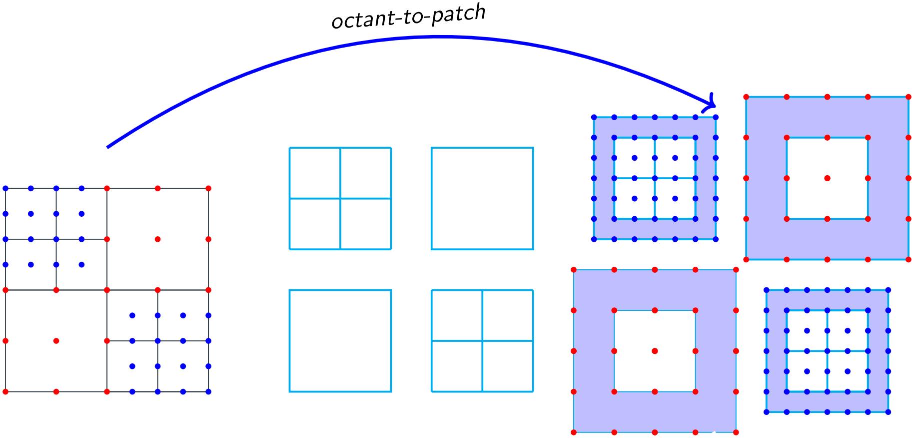
- Fast and efficient partitioning algorithms are essential for AMR applications
- We use SFC-based partitioning (i.e., reduces to SFC based sorting problem)

Octant vs. Grid patch : FD computations on octrees

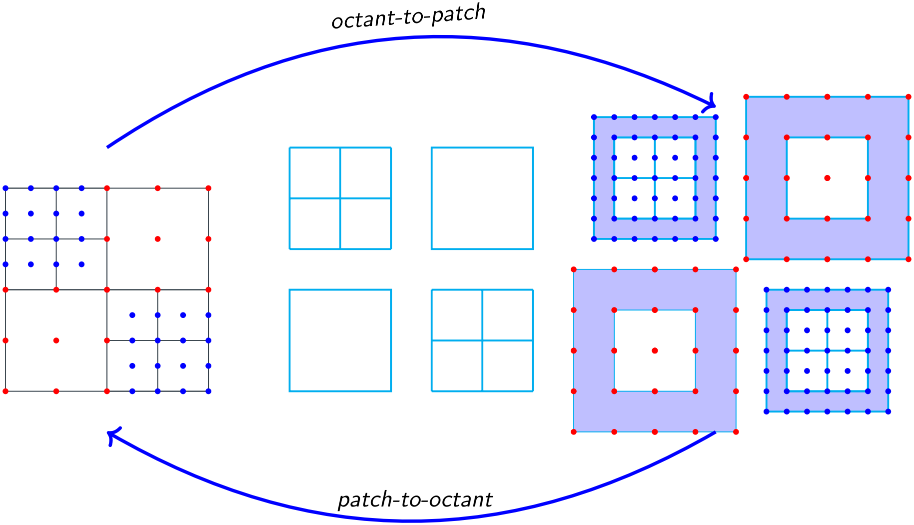
- **Octant**: uniformly spaced r^3 grid points
- **Patch**: octant with padding points, $(r+2k)^3$
- For all simulations presented we used $r = 7$ and $k = 3$



FD computations on Octrees



FD computations on Octrees



Overview: Evolution

- **Host**: Mesh generation, partitioning, octree related data structures
- **Device**: Time integration is entirely handled by the device

---

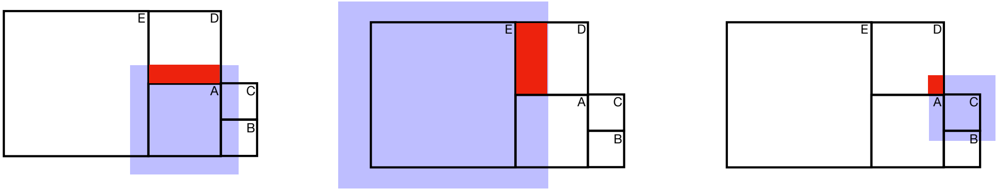
Algorithm Overview: Time evolution

Require: u state at $t = t_0$, T: time horizon, Δt timestep size, f_r: re-grid frequency
Ensure: u state at $t = T$

```
N ← (T − t_o)/ΔT
for each i ∈ [0 : N : fr] do                              ▷ 0 to N with f_r increments
    M ← construct_grid(u)                                 ▷ on the host
    v ← host_to_device(u)
    for each f_r timesteps do                             ▷ on the device
        v ← halo_exchange(v)                              ▷ synchronize partitions
        v̂ ← octant-to-patch(v)                            ▷ compute octant patches
        ŵ ← RHS(v̂, t)                                     ▷ evaluate RHS
        w ← patch-to-octant(ŵ)                            ▷ revert back to octants
        v ← AXPY(w, v, Δt)                                ▷ evolve state v = v + Δtw
    u ← device_to_host(v)
return u
```

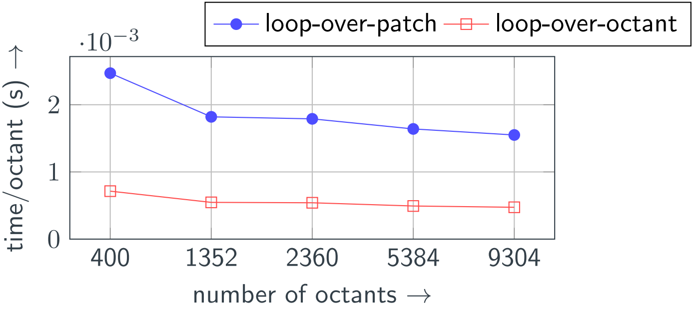
Octant to patch: Scatter vs. Gather

- **loop-over-patches**: For each patch **gather** information from neighboring octants
  - scattered reads
  - required interpolations are duplicated between neighboring patches
- **loop-over-octant**: For each octant **scatter** information for neighboring patches
  - uniform reads, and octant information is reused between multiple patches
  - no redundant interpolations

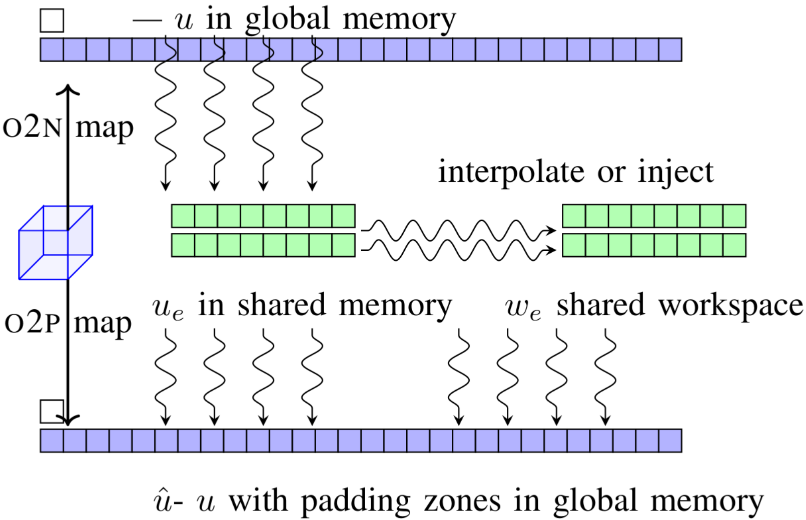


Octant to patch: Scatter vs. Gather

- **loop-over-patches**: For each patch **gather** information from neighboring octants
  - scattered reads
  - required interpolations are duplicated between neighboring patches
- **loop-over-octant**: For each octant **scatter** information for neighboring patches
  - uniform reads, and octant information is reused between multiple patches
  - no redundant interpolations



Octant-to-Patch: Data layout (on the GPU)



SymPyGR: Symbolic Code Generation

$$
\begin{aligned}
\partial_t \alpha &= \mathcal{L}_\beta \alpha - 2\alpha K, \\
\partial_t \beta^i &= \lambda_2 \beta^j \, \partial_j \beta^i + \frac{3}{4} f(\alpha) B^i, \\
\partial_t B^i &= \partial_t \tilde{\Gamma}^i - \eta B^i + \lambda_3 \beta^j \, \partial_j B^i - \lambda_4 \beta^j \, \partial_j \tilde{\Gamma}^i, \\
\partial_t \tilde{\gamma}_{ij} &= \mathcal{L}_\beta \tilde{\gamma}_{ij} - 2\alpha \tilde{A}_{ij}, \\
\partial_t \chi &= \mathcal{L}_\beta \chi + \frac{2}{3} \chi \left(\alpha K - \partial_a \beta^a \right), \\
\partial_t \tilde{A}_{ij} &= \mathcal{L}_\beta \tilde{A}_{ij} + \chi \left(-D_i D_j \alpha + \alpha R_{ij} \right)^{TF} \\
&\quad + \alpha \left(K \tilde{A}_{ij} - 2 \tilde{A}_{ik} \tilde{A}^k_j \right), \\
\partial_t K &= \beta^k \partial_k K - D^i D_i \alpha \\
&\quad + \alpha \left(\tilde{A}_{ij} \tilde{A}^{ij} + \frac{1}{3} K^2 \right), \\
\partial_t \tilde{\Gamma}^i &= \tilde{\gamma}^{jk} \partial_j \partial_k \beta^i + \frac{1}{3} \tilde{\gamma}^{ij} \partial_j \partial_k \beta^k + \beta^j \partial_j \tilde{\Gamma}^i \\
&\quad - \tilde{\Gamma}^j \partial_j \beta^i + \frac{2}{3} \tilde{\Gamma}^i \partial_j \beta^j - 2 \tilde{A}^{ij} \partial_j \alpha + \\
&\quad 2\alpha \left(\tilde{\Gamma}^i{}_{jk} \tilde{A}^{jk} - \frac{3}{2\chi} \tilde{A}^{ij} \partial_j \chi - \frac{2}{3} \tilde{\gamma}^{ij} \partial_j K \right)
\end{aligned}
$$

```python
from DENDRO_sym import *

a_rhs = Dendro.Lie(b, a) - 2*a*K

b_rhs = [3/4 * f(a) * B[i] +
 l2*vec_j_del_j(b, b[i]) for i in e_i]
        l2*vec_j_del_j(b, b[i])
        for i in e_i]

B_rhs = [Gt_rhs[i] - eta * B[i] +
        l3 * vec_j_del_j(b, B[i]) -
        l4 * vec_j_del_j(b, Gt[i])
        for i in e_i]

gt_rhs =  Dendro.Lie(b, gt) - 2*a*At

chi_rhs = Dendro.Lie(b, chi) +
          2/3*chi*(a*K - del_j(b))

At_rhs = Dendro.Lie(b, At) + chi *
        Dendro.TF(-DiDj(a) +
                  a*Dendro.Ricci) +
        a*(K*At -2*At_ikAtKj)

K_rhs = vec_k_del_k(K) - DIDi(a) +
        a*(1/3*K*K + A_ij_A_IJ(At))
```

SymPyGR: Symbolic Code Generation

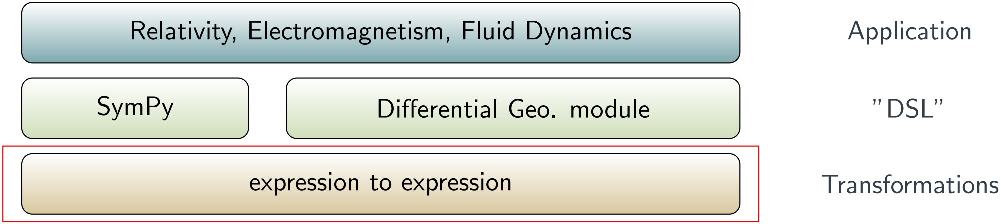
Relativity, Electromagnetism, Fluid Dynamics

Application

SymPyGR: Symbolic Code Generation

Relativity, Electromagnetism, Fluid Dynamics		Application
SymPy	Differential Geo. module	"DSL"

SymPyGR: Symbolic Code Generation

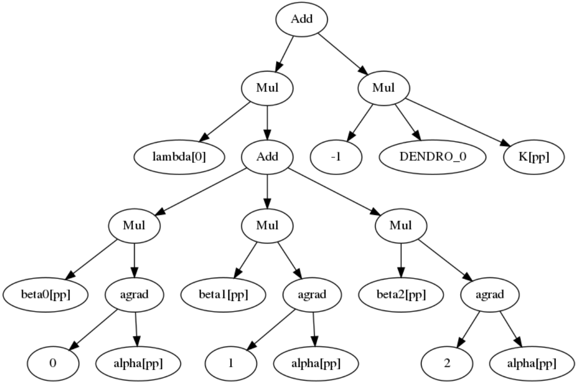


SymPyGR: Symbolic Code Generation

Relativity, Electromagnetism, Fluid Dynamics	Application
SymPy / Differential Geo. module	"DSL"
expression to expression	Transformations
C, C++, AVX, OpenMp, CUDA	Code generator

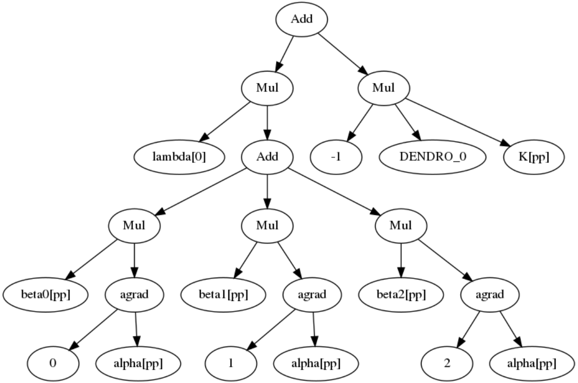
Architecture-specific code generation

Expression tree



Architecture-specific code generation

Expression tree

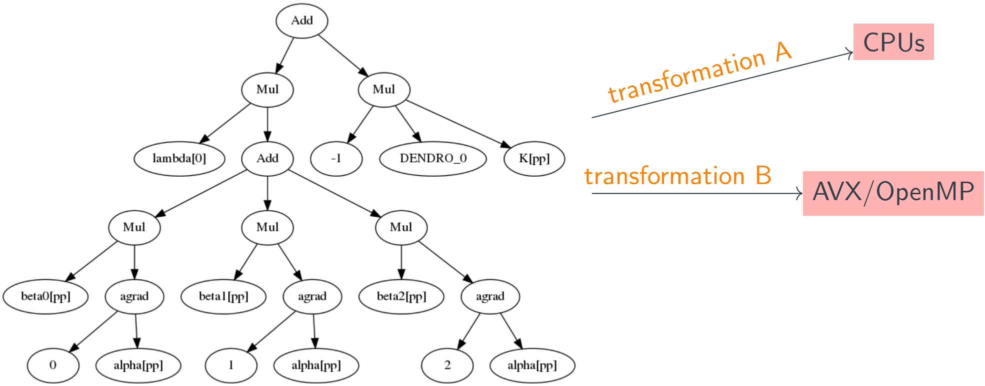


transformation A

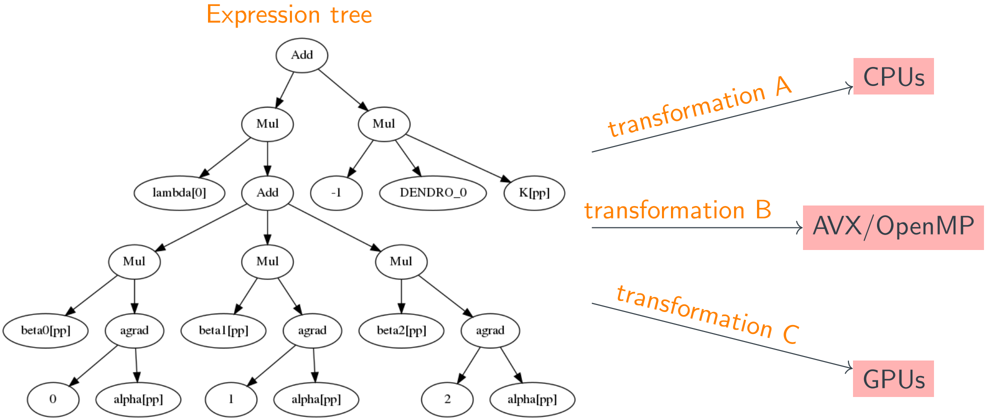
CPUs

Architecture-specific code generation

Expression tree



Architecture-specific code generation



Right-hand-side (RHS) evaluation

- RHS evaluation has two main components
  - $\mathcal{D}$: derivatives (210 FD evaluations per grid point)
  - $\mathcal{A}$: algebraic
- Example $\partial_t \alpha = \sum_{i=0}^{2} \beta^i \partial_i \alpha - 2\alpha K$,
  - $\mathcal{D}$: $\partial_0 \alpha, \partial_1 \alpha, \partial_2 \alpha$
  - $\mathcal{A}$: $\sum_{i=0}^{2} \beta^i \partial_i \alpha - 2\alpha K$

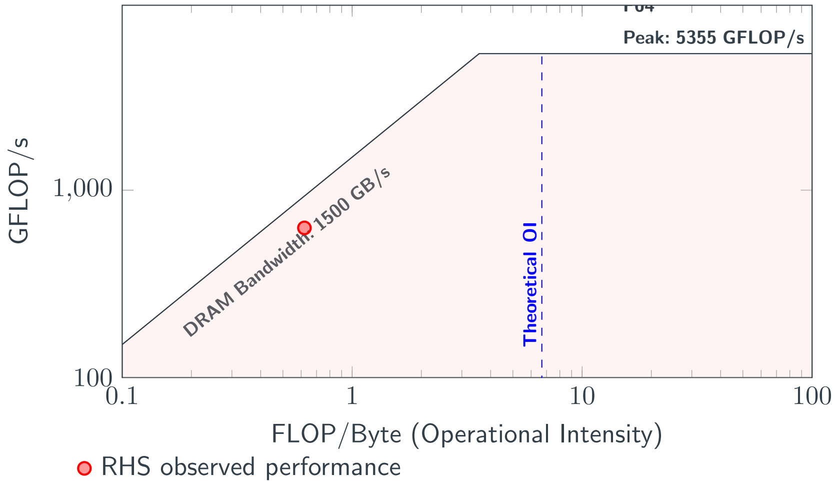
Right-hand-side (RHS) evaluation

- **Unfused** evaluation $\mathcal{D}$ and $\mathcal{A}$: stage 1 $\rightarrow$ compute and store all the derivatives, stage 2 $\rightarrow$ algebraic evaluation
- **Fused** evaluation $\mathcal{D} + \mathcal{A}$ combined
- Infinite cache model

$$Q_{D+A} = \frac{r^3(33(2d^2-1)+177(2d-1)+O_A)}{8(24(r+2k)^3+24r^3)} \approx 6.68 \tag{1}$$

$$Q_A = \frac{r^3(O_A)}{8(24\times 2+210)r^3} \approx 1.94 \tag{2}$$

Right-hand-side (RHS) evaluation



Why sub-optimal RHS performance ?

- High register pressure
- 24 input/output, 210 intermediate derivatives + stencil dependencies
- What can we do to improve performance ?

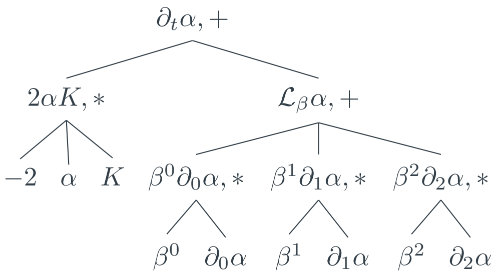
Right-hand-side (RHS) evaluation

- **SymPyGR + CSE** : Convert expressions to SymPy and then use SymPy common sub expression elimination (CSE) ($\approx$ 900 thread local variables)algorithm[2]
- **Binary-reduce**: Rewrite expressions as binary reductions to reduce expression length ($\approx$ 675 thread local variables)
- **Staging + CSE**: Reorder the derivative computations right before the corresponding RHS evaluations
  - If α derivatives are evaluated, try to compute the RHS that require derivatives of α

[2] Fernando, M., Neilsen, D., Lim, H., Hirschmann, E. and Sundar, H., 2019. Massively Parallel Simulations of Binary Black Hole Intermediate-Mass-Ratio Inspirals. SIAM Journal on Scientific Computing, 41(2), pp.C97-C138.

Right-hand-side (RHS) evaluation : Binary-reduce approach

- Use SymPyGR to generate computational graph $G = (V, E)$ (for BSSN $|V| = 2516, |E| = 6708$)
- For a specified traversal order, RHS is computed as with binary reductions
- **As a heuristic we use topological sort of the line graph of G**



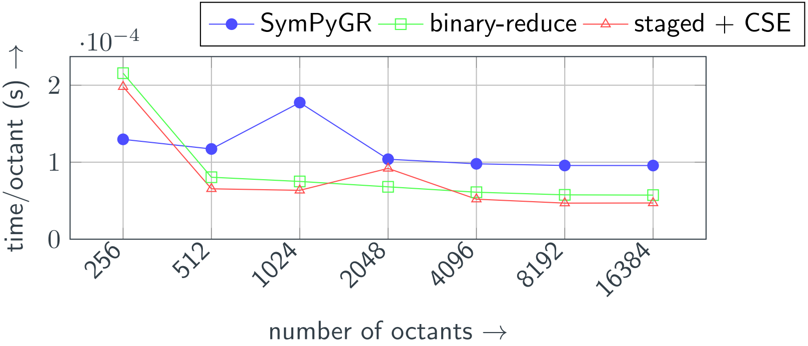
Algorithm $visit\_node(v)$

```
Require: G = (V, E), v ∈ V, B− local memory
  v.DONE ← true
  for u ∈ v.descendants do
      store(v, u, B)                          ▷ Store in local memory
      reduce(u, v)
      remove edge (u, v) from G
      if degree(u) is 0 then
          evict (u, B)
  if v is a final expr then
      store_to_global(v)
      if degree(v) is 0 then
          evict(v,B)
  return
```

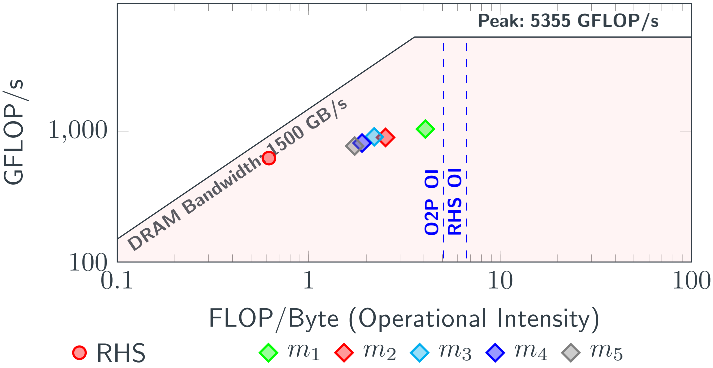
Right-hand-side (RHS) evaluation

- We enforced 56 registers per thread with `--ptxas-options=-O3`

RHS variation	ptx-spill stores (bytes)	ptx-spill loads (bytes)	average speedup w.r.t. SymPyGR
SymPyGR + CSE	15892	33288	1.00x
binary-reduce	10176	22012	1.55x
staging + CSE	8876	22028	1.76x



Roofline performance analysis

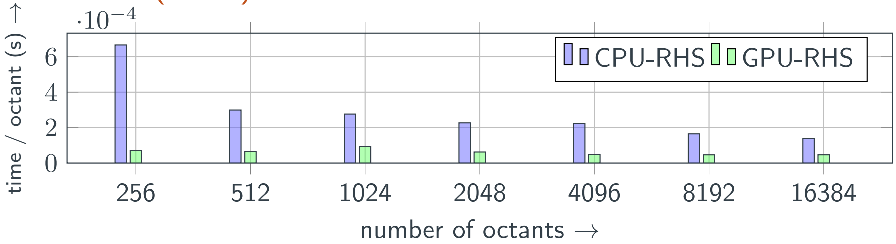


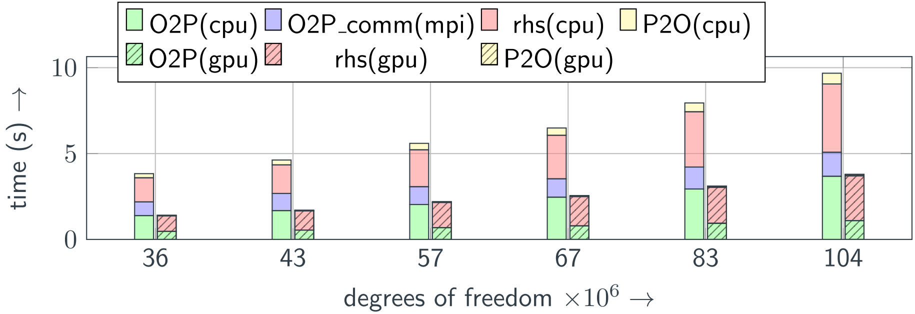
- For octant to patch computation we can write, $Q_{O2P} \leq \frac{8\times 3(2r-1)r^3}{8(2r^2+2r^3+12rk^2+6r^2k+8k^3)} \approx 5.07$
- m_1 mesh is the most non-uniform, m_5 mesh is the most uniform.

Experimental setup

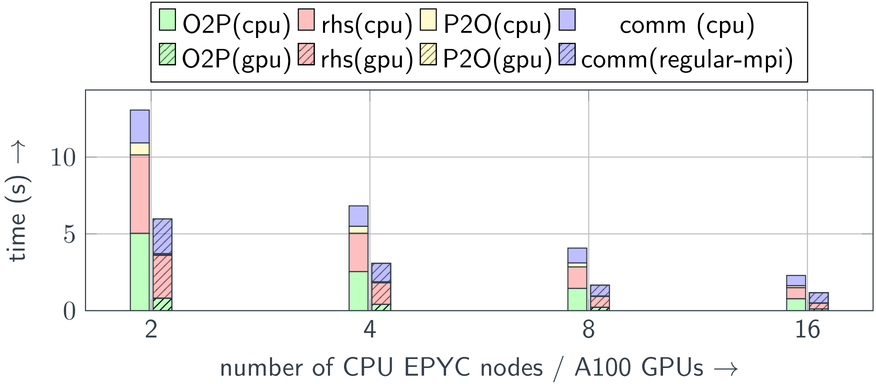
- **Frontera**: has 8K Intel Cascade Lake nodes
- **Lonestar 6**: has 16 nodes with dual NVIDIA A100 and dual AMD EPYC 7763 64-Core CPU ("Milan")
- All GPU-CPU comparisons were done on a single NVIDIA A100 GPU compared with a single CPU node (i.e., dual AMD EPYC with 64x2 cores)
- CPU only weak scalability study was performed in Frontera, GPU scalability studies were performed in Lonestar 6

Dual AMD EPYC (64x2) CPUs vs. Nvidia A100 GPU



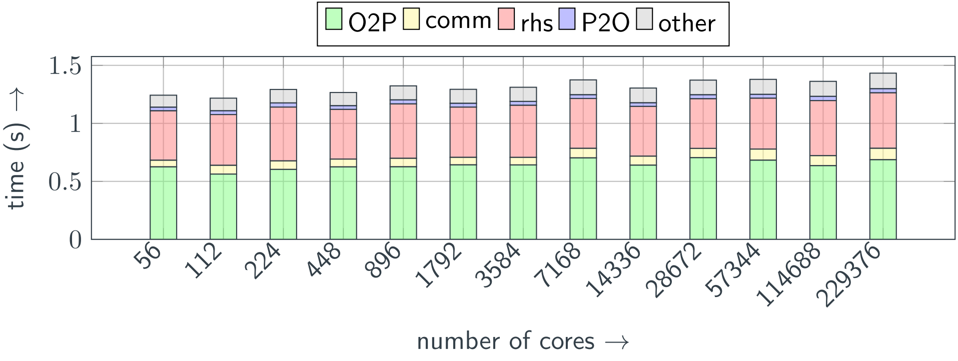


Strong scalability (CPU & GPU)



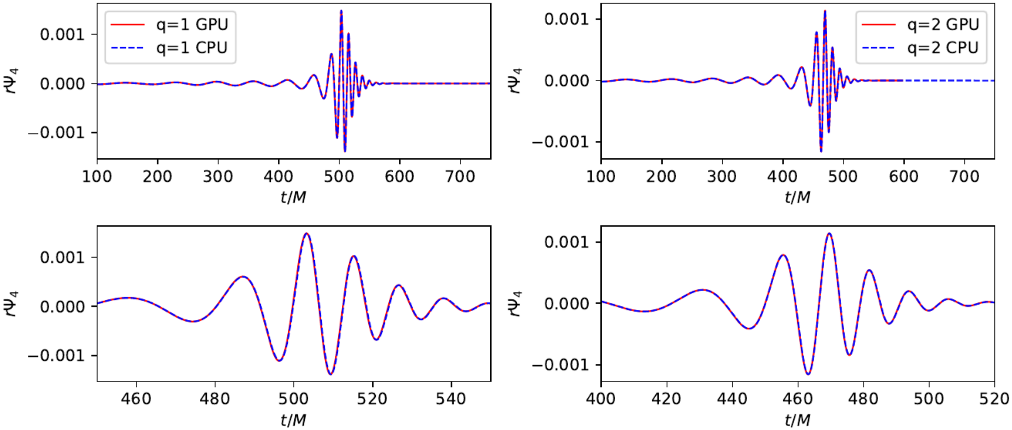
- Problem size 257M DOFs
- Time is shown for 5 RK4 timesteps
- **Parallel efficiencies** : GPU → 0.97, 0.89, and 0.64 , and CPU → 0.93, 0.79, 0.66

Weak scalability



- Largest problem size 118B DOFs
- Weak scalability study conducted with ≈ 500K DOFs per core

Accuracy & Validation: Dendro-GR vs. LazEv



- Dendro-GR CPU GW waveforms are verified with the LazEv code
- Dendro-GR GPU code is verified with the CPU waveforms

Overall time to solution

Mass ratio $q = m_1/m_2$	$\Delta x_{\min}$ (BH1)	$\Delta x_{\min}$ (BH2)	GPUs NVIDIA A100	T	timesteps	Wall time (hrs)
1	1.62e-2	1.62e-2	4	748M	183K	87
2	8.13e-3	3.25e-2	4	600M	252K	96
4	4.06e-3	3.25e-2	4	602M	506K	129
8	2.03e-3	3.25e-2	8	1400M	4M	388

- All the runs conducted on Lonestar 6, with GPU accelerated solver.
- For all the runs we start with initial coordinate seperation of 8M.

Conclusions

- First attempt to deploy GPUs for binary black hole simulations
- GPU acceleration gives performance gains of 2.0-2.5x speedup (Dual AMD EPYC 64x2 CPU cores vs 1 Nvidia A100 GPU)
- Performance on sparse adaptive computations on GPUs is challenging, but doable.
- Register spilling in RHS computation degrades performance
- You can find the Dendro-GR code at https://github.com/paralab/Dendro-GR

Acknowledgements

- Funding soources NASA-80NSSC20K0528, NSF: PHY-2207615, PHY-1912930

Further reading

- Fernando, M., Neilsen, D., Zlochower, Y., Hirschmann, E.W. and Sundar, H., 2023. Massively parallel simulations of binary black holes with adaptive wavelet multiresolution. Physical Review D, 107(6), p.064035.
- **Fernando, M., Neilsen, D., Hirschmann, E., Zlochower, Y., Sundar, H., Ghattas, O. and Biros, G., 2022, November. A GPU-accelerated AMR solver for gravitational wave propagation. In 2022 SC22: International Conference for High Performance Computing, Networking, Storage and Analysis (SC) (pp. 1078-1092). IEEE Computer Society.**
- Milinda Fernando, David Neilsen, Hyun Lim, Eric Hirschmann, Hari Sundar, "Massively Parallel Simulations of Binary Black Hole Intermediate-Mass-Ratio Inspirals" SIAM Journal on Scientific Computing 2019. 'https://doi.org/10.1137/18M1196972'
- Milinda Fernando, David Neilsen, Eric Hirschmann, Hari Sundar, "A scalable framework for Adaptive Computational General Relativity on Heterogeneous Clusters", (ACM International Conference on Supercomputing, ICS'19)
- Fernando, M. and Sundar, H., 2022. Scalable Local Timestepping on Octree Grids. SIAM Journal on Scientific Computing, 44(2), pp.C156-C183.

Thank You!

Questions ?