BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY
Office of Science

Engineering Fast Kernels for Rotation-Equivariant Chemical Foundation Models

Vivek Bharadwaj, DOE CSGF Program Review 2025

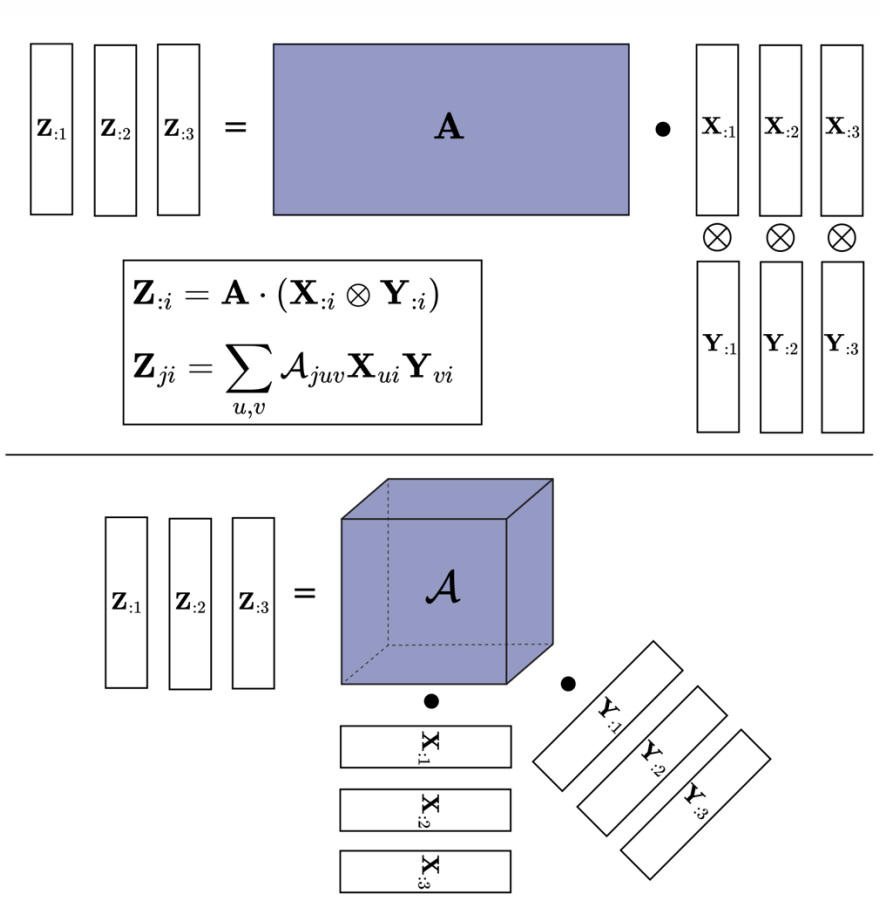
Sparse Tensors, Multilinear Maps, and Me

About me: computer science PhD student at UC Berkeley.

Interests: sparse linear algebra, multilinear maps and tensor kernels to accelerate scientific / ML workloads.

Some past projects:

- Algorithms for high-dimensional “tensor SVD”.
- Scaling bulk-synchronous linear algebra on large clusters.
- Distributed-memory graph algorithms.

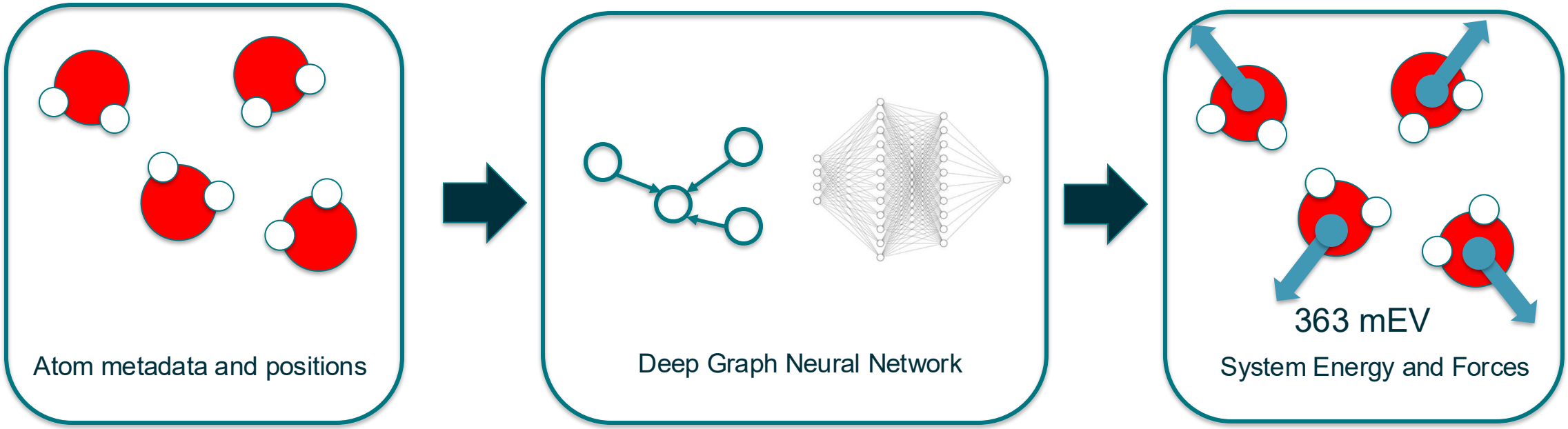


AM NOT: A chemist, computational or otherwise.

My Flight to the 2024 CSGF Program Review

Tristan Maxson (CSGF Y3) generously spent hours telling me about *machine learning interatomic potentials*.

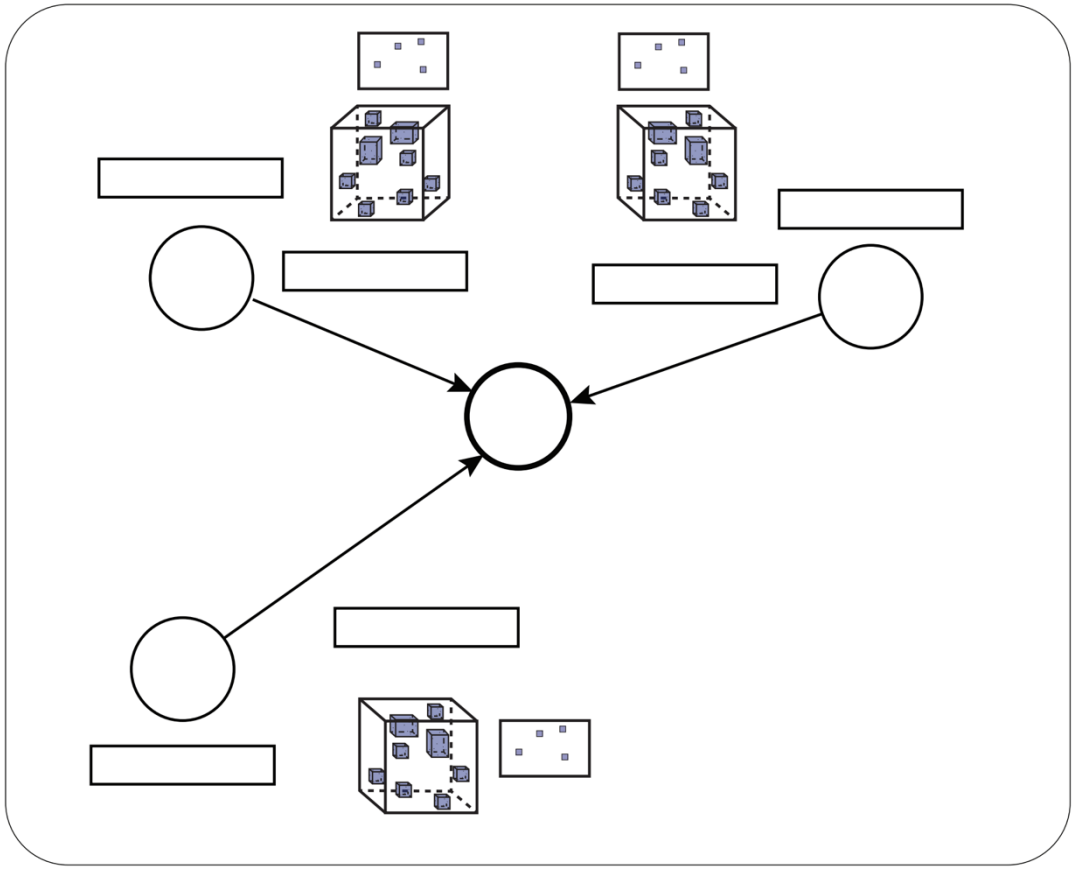
Interatomic Potentials via Graph Neural Networks



Goal: Use a message-passing graph neural network to predict atomic energies and forces.

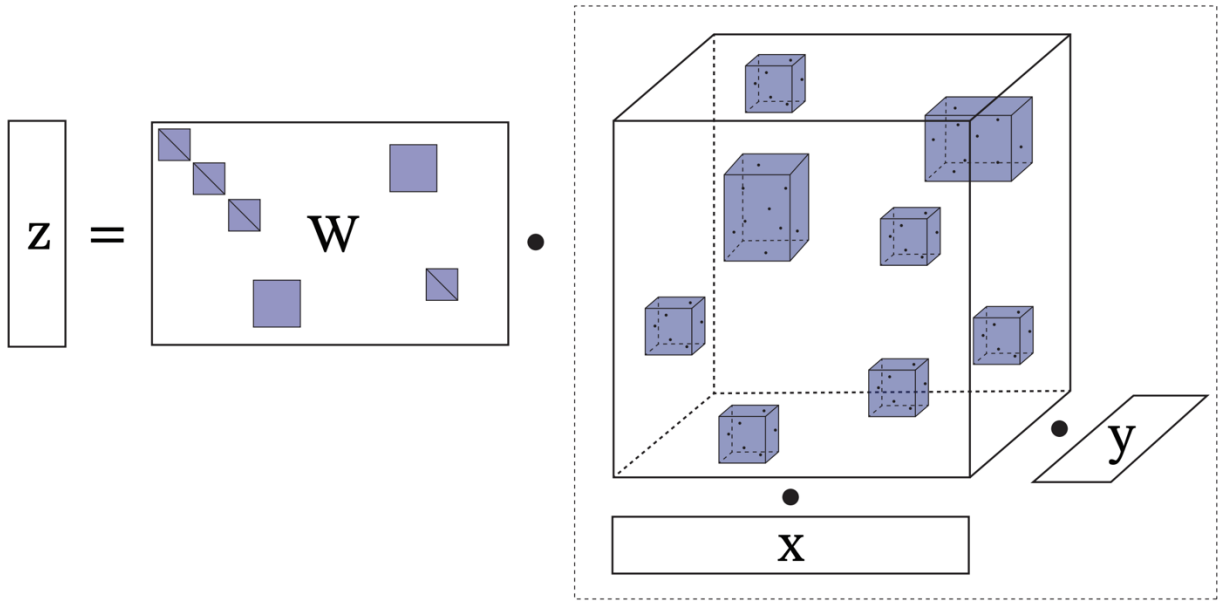
Constraint: Want our network to respect certain physical laws (e.g., symmetry, energy conservation).

Respecting Physical Symmetries



- Message-passing GNNs generate feature vectors for each node and edge.
- **O(3)-Equivariance:** If input coordinate system rotates, the output energy **stays the same** and the predicted forces **rotate compatibly**.
- Need to combine node / edge features in a highly-structured, prescriptive manner.

The Clebsch-Gordon Tensor Product



$$z = W \cdot P \cdot (x \otimes y) = W \sum_{i=1,j=1}^{m,n} x[i]\, y[j]\, \mathcal{P}[ij:]$$

Structure of the sparse tensor is known when GNN is initialized.

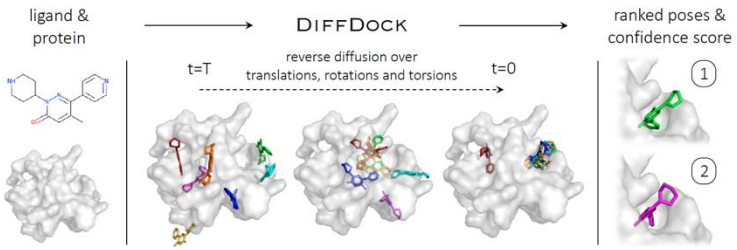
Typically, need to execute **millions** of these operations.

Problem: GPUs are optimized for dense GEMM!

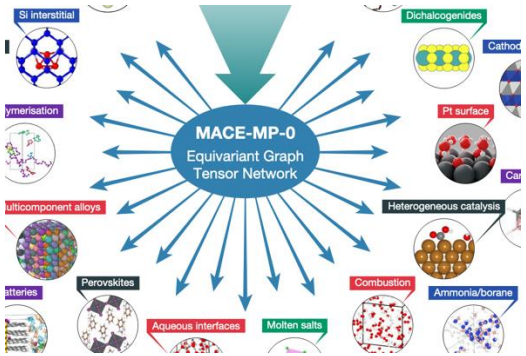
CG tensor product can run at <10% GPU utilization, consumes >80% of model runtime.

Some Large Equivariant Neural Network Projects

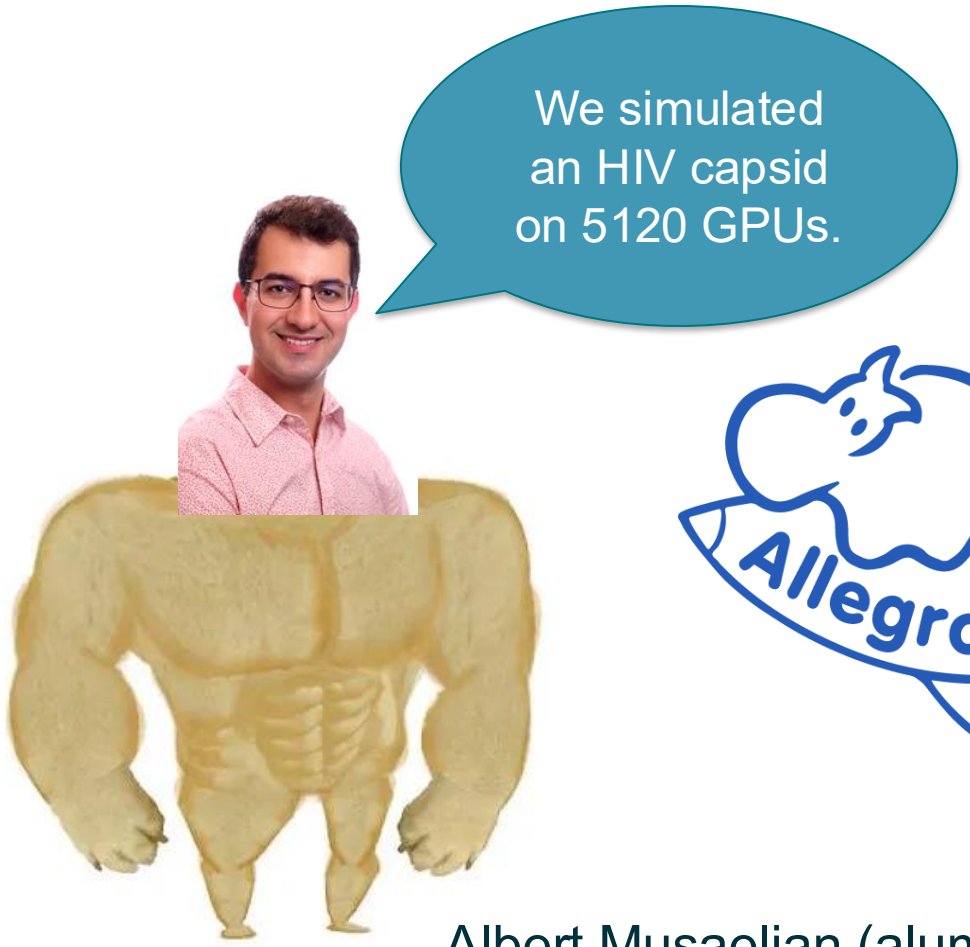
NequIP



Bowen Jing (Y4) & others



Samuel Blau (alum ’16) & others



Allegro

Albert Musaelian (alum’23) and colleagues nominated for the 2023 Gordon Bell prize.

OpenEquivariance: Turbocharging CG Performance

- We present OpenEquivariance: an open-source GPU kernel generator for the CG tensor product
- **OOM speedup over best open-source codes**, on-par w/ NVIDIA's **closed-source** package (but 2x speedup against their earlier version!)
- 5-6x speedup for MACE / Nequip, 10x+ smaller memory footprint. 76 Github stars and counting!

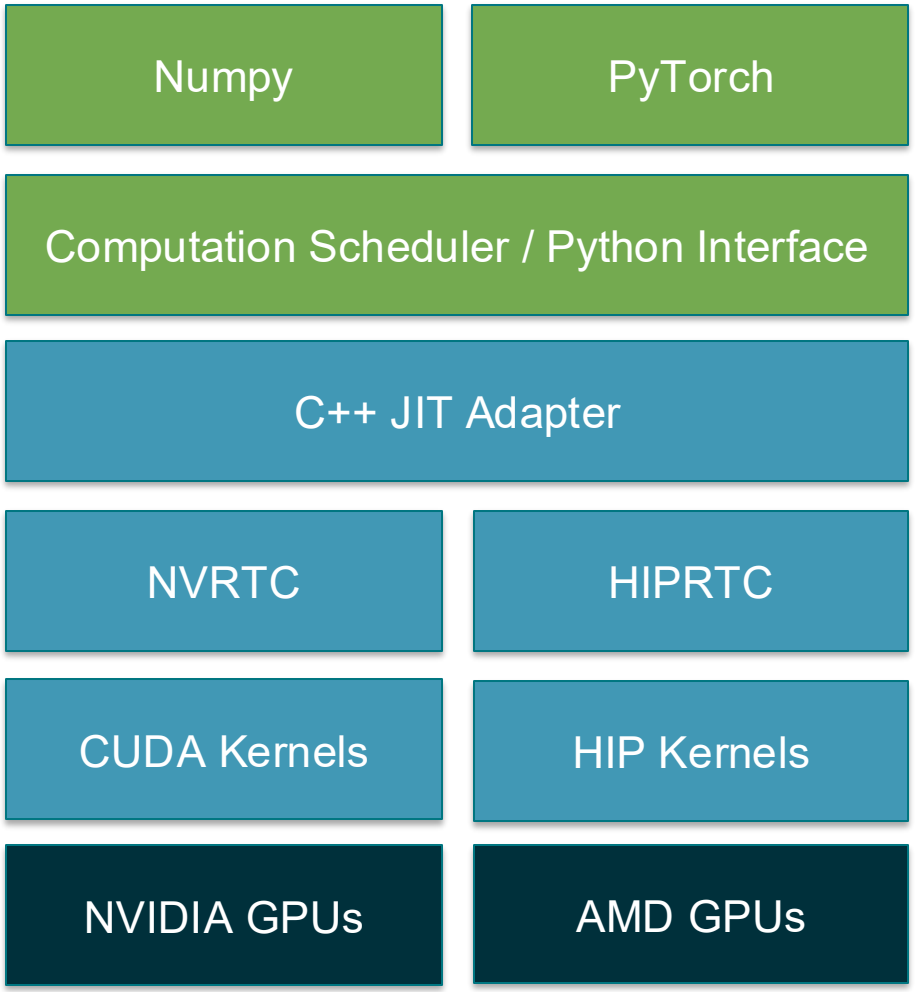

Me*


Austin Glover*


Aydın Buluç

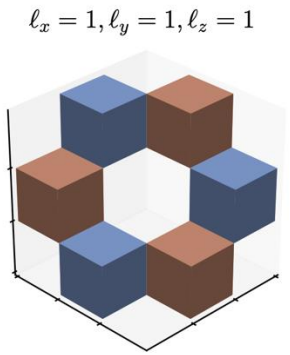

James Demmel

*=equal effort

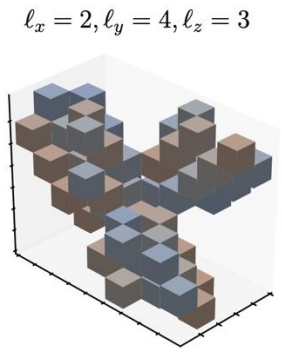


Subkernels of the CG Tensor Product

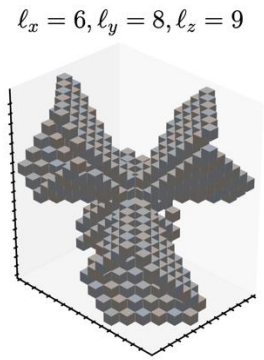
- Break the computation into segments, one for each nonzero block.
- Each segment contracts $\boldsymbol{x}$, $\boldsymbol{y}$ with a block, matrix-multiplies result by tile from $\boldsymbol{W}$.



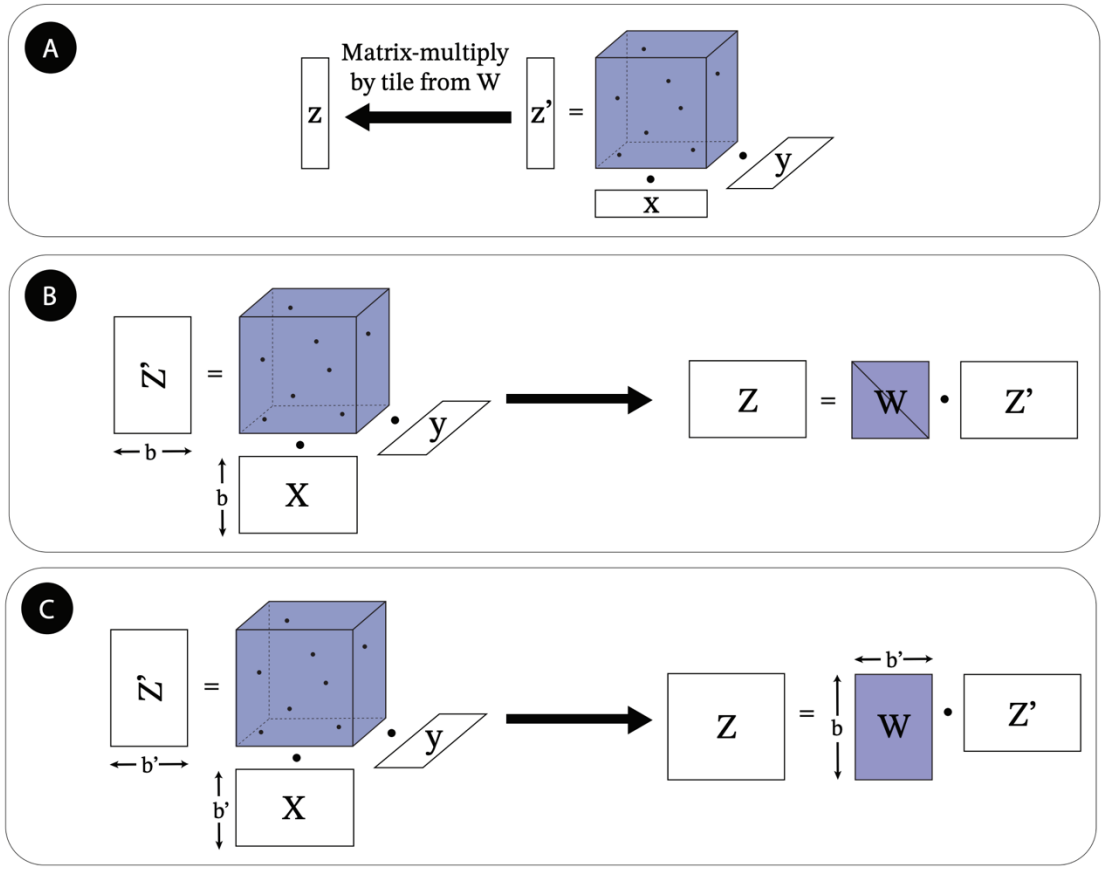
nonzero % = 0.22



nonzero % = 0.16

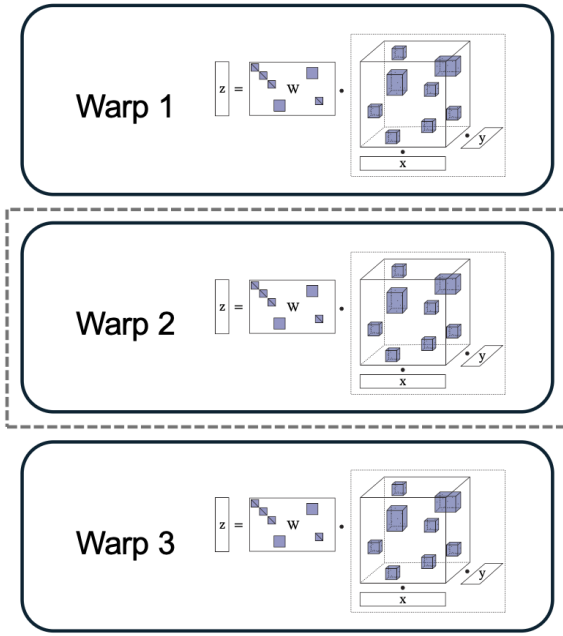


nonzero % = 0.08



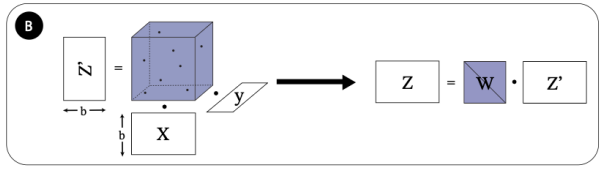
A Roadmap to Efficient CG Kernels

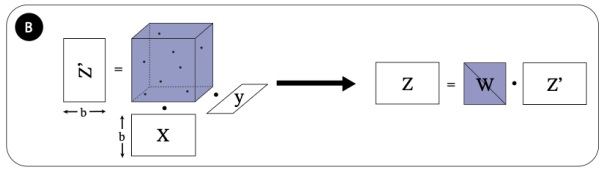
Assign each batch element to distinct GPU warp



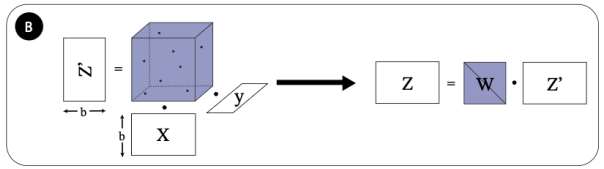
Schedule subkernels to manage SMEM usage, avoid memory traffic

Load subsegments of x, y, W into SMEM, reshape





Flush z to global memory, load next segment of x



Register-cache operands, use JIT to emit optimized instruction stream

Source

```
z[1] += 0x1.6a09e60000000p-2f * x[0] * y[0];
z[0] += -0x1.6a09e60000000p-2f * x[1] * y[0];
z[2] += 0x1.d363d00000000p-2f * x[1] * y[0];
z[1] += -0x1.d363d00000000p-2f * x[2] * y[0];
z[4] += -0x1.6a09e60000000p-1f * x[3] * y[0];
z[3] += 0x1.6a09e60000000p-1f * x[4] * y[0];
z[5] += -0x1.d363d00000000p-2f * x[4] * y[0];
z[4] += 0x1.d363d00000000p-2f * x[5] * y[0];
z[6] += -0x1.6a09e60000000p-2f * x[5] * y[0];
z[5] += 0x1.279a740000000p-1f * x[1] * y[1];
z[4] += 0x1.279a740000000p-2f * x[2] * y[1];
z[2] += -0x1.279a740000000p-2f * x[4] * y[1];
z[1] += -0x1.279a740000000p-1f * x[5] * y[1];
```

PTX

```
fma.rn.f32 %f4183, %f4126, %f4089, %f4182;
fma.rn.f32 %f4184, %f4152, %f4089, %f4183;
fma.rn.f32 %f4185, %f4112, %f4089, %f4184;
mul.f32 %f4186, %f4094, 0f3F08D677;
mul.f32 %f4187, %f4186, %f4096;
fma.rn.f32 %f4188, %f4187, %f4089, %f4146;
fma.rn.f32 %f4189, %f4100, %f4089, %f4188;
mul.f32 %f4190, %f4133, 0f3EA79762;
mul.f32 %f4191, %f4190, %f4096;
fma.rn.f32 %f4192, %f4191, %f4089, %f4171;
fma.rn.f32 %f4193, %f4137, %f4089, %f4192;
fma.rn.f32 %f4194, %f4135, %f4089, %f4158;
fma.rn.f32 %f4195, %f4137, %f4089, %f4194;
```

Forward Pass Warp-Level Algorithm

Algorithm Subkernel C Warp-Level Algorithm

Require: $\boldsymbol{X} \in \mathbb{R}^{b' \times (2\ell_x+1)}, \boldsymbol{y} \in \mathbb{R}^{(2\ell_y+1)}, \boldsymbol{W} \in \mathbb{R}^{b \times b'}$

Require: Sparse tensor $\mathcal{P}^{(\ell_x,\ell_y,\ell_z)}$ for subkernel

for $t = 1 \ldots b'$ **do** ▷ Parallel over threads

Load $\boldsymbol{x}_{\text{reg}} = \boldsymbol{X}[t,:], \boldsymbol{y}_{\text{reg}} = \boldsymbol{y}$

Initialize $\boldsymbol{z}_{\text{reg}} \in \mathbb{R}^{2\ell_z+1}$ to 0.

for $(i,j,k,v) \in \text{nz}(\mathcal{P})$ **do** ▷ Unroll via JIT

$\boldsymbol{z}_{\text{reg}}[k] \mathrel{+}= v \cdot \boldsymbol{x}_{\text{reg}}[i] \cdot \boldsymbol{y}_{\text{reg}}[j]$

Store $\boldsymbol{Z}'[t,:] = \boldsymbol{z}_{\text{reg}}$, compute $\boldsymbol{Z} \mathrel{+}= \boldsymbol{W} \cdot \boldsymbol{Z}'$.

Backward calculation is similar, but requires three update equations instead of one.

Optimizing Model Force Predictions

- The derivative of the predicted energy $E(\boldsymbol{R}, \boldsymbol{W})$ w.r.t. atomic positions $\boldsymbol{R}$ yields atomic forces.

$$\min_{\boldsymbol{W}} \mathcal{L}(\boldsymbol{R}, \boldsymbol{W}) = \min_{\boldsymbol{W}} \left\| \boldsymbol{F}_{\text{pr}}(\boldsymbol{R}, \boldsymbol{W}) - \boldsymbol{F}_{\text{gt}}(\boldsymbol{R}) \right\|_F^2$$

- To minimize the difference between predicted forces and a ground truth, we need a *second partial derivative of energy*:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}} &= 2 \cdot \text{vec}(\boldsymbol{F}_{\text{pr}}(\boldsymbol{R}, \boldsymbol{W}) - \boldsymbol{F}_{\text{gt}}(\boldsymbol{R}))^\top \frac{\partial \boldsymbol{F}_{\text{pr}}}{\partial \boldsymbol{W}} \\ &= -2 \cdot \text{vec}(\boldsymbol{F}_{\text{pr}}(\boldsymbol{R}, \boldsymbol{W}) - \boldsymbol{F}_{\text{gt}}(\boldsymbol{R}))^\top \boxed{\frac{\partial^2 E}{\partial \boldsymbol{R} \partial \boldsymbol{W}}} \end{aligned}$$

Novel Identities for Second Partial Derivatives

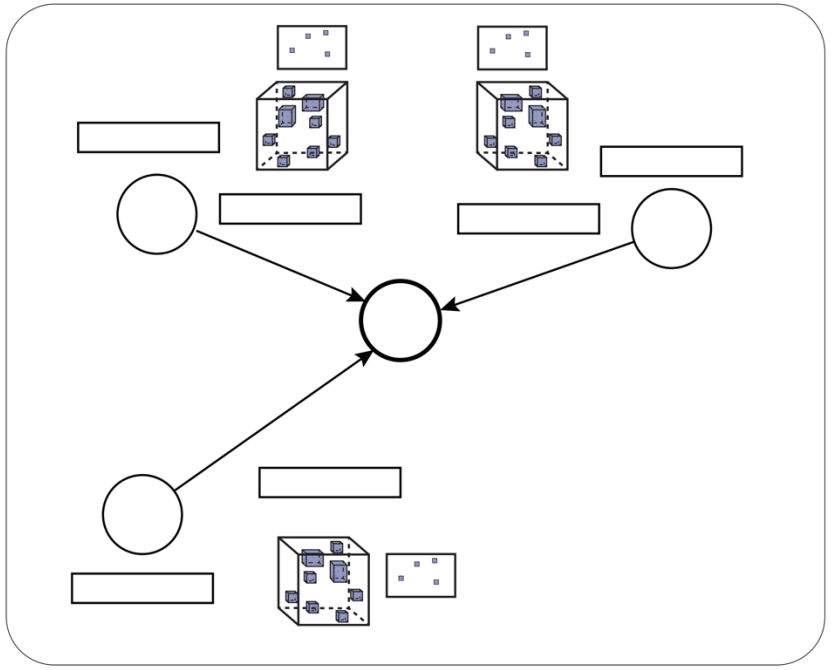
- PyTorch is happy to calculate a second partial derivative… if you supply a “double-backward” pass for the CG tensor product.
- **Key:** No additional kernel engineering required! Double-backward can be expressed as a linear combination of forward / backward calls.

$$
\begin{aligned}
\text{op1} &= \text{backward}(\partial\mathcal{L}/\partial\boldsymbol{a}, \partial\mathcal{L}/\partial\boldsymbol{b}, \boldsymbol{W}, \boldsymbol{g}_z) \\
\text{op2} &= \text{backward}(\boldsymbol{x}, \boldsymbol{y}, \partial\mathcal{L}/\partial\boldsymbol{C}, \boldsymbol{g}_z) \\
\text{op3} &= \text{TP}(\partial\mathcal{L}/\partial\boldsymbol{a}, \boldsymbol{y}, \boldsymbol{W}) \\
\text{op4} &= \text{backward}(\partial\mathcal{L}/\partial\boldsymbol{a}, \boldsymbol{y}, \boldsymbol{W}, \boldsymbol{g}_z) \\
\text{op5} &= \text{backward}(\boldsymbol{x}, \partial\mathcal{L}/\partial\boldsymbol{b}, \boldsymbol{W}, \boldsymbol{g}_z) \\
\text{op6} &= \text{TP}(\boldsymbol{x}, \partial\mathcal{L}/\partial\boldsymbol{b}, \boldsymbol{W}) \\
\text{op7} &= \text{TP}(\boldsymbol{x}, \boldsymbol{y}, \partial\mathcal{L}/\partial\boldsymbol{C}).
\end{aligned}
$$

$$
\begin{aligned}
\partial\mathcal{L}/\partial\boldsymbol{x} &= \text{op1}\,[1] + \text{op2}\,[1] \\
\partial\mathcal{L}/\partial\boldsymbol{y} &= \text{op1}\,[2] + \text{op2}\,[2] \\
\partial\mathcal{L}/\partial\boldsymbol{W} &= \text{op4}\,[3] + \text{op5}\,[3] \\
\partial\mathcal{L}/\partial\boldsymbol{g}_z &= \text{op3} + \text{op6} + \text{op7},
\end{aligned}
$$

Graph Convolution with Kernel Fusion

- Graph convolution has a well-studied SpMM memory access pattern.
- Fuse both CG tensor product and graph convolution to save compute AND memory!

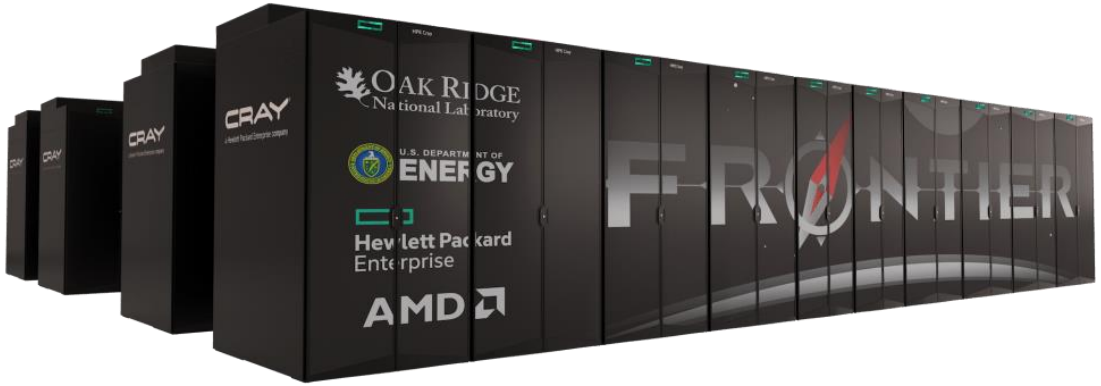


Algorithm Deterministic TP + Graph Convolution

Require: Graph $G = (V, E)$, $E[b] = (i_b, j_b)$
Require: Batch $\boldsymbol{x}_1, ..., \boldsymbol{x}_{|V|}, \boldsymbol{y}_1, ...\boldsymbol{y}_{|E|}, \boldsymbol{W}_1, ..., \boldsymbol{W}_{|E|}$
for $\text{segment}_i \in \text{schedule}$ **do**
  $(s, t) = E[k][0], E[k][1]$
  Set $\boldsymbol{z}_{\text{acc}} = 0$ ▷ Parallel over Warps
  for $b = 1... |E|$ **do**
    Execute segment subkernel sequence
    $\boldsymbol{z}_{\text{acc}} \mathrel{+}= \boldsymbol{z}_{\text{smem}}$
    if $b = |E|$ or $s < E[b+1][0]$ **then**
      if s is first vertex processed by warp **then**
        Send $\boldsymbol{z}_{\text{acc}}$ to fixup buffer.
      else
        Store z_{acc} to global memory.
      $\boldsymbol{z}_{\text{acc}} = 0$
  Execute fixup kernel.

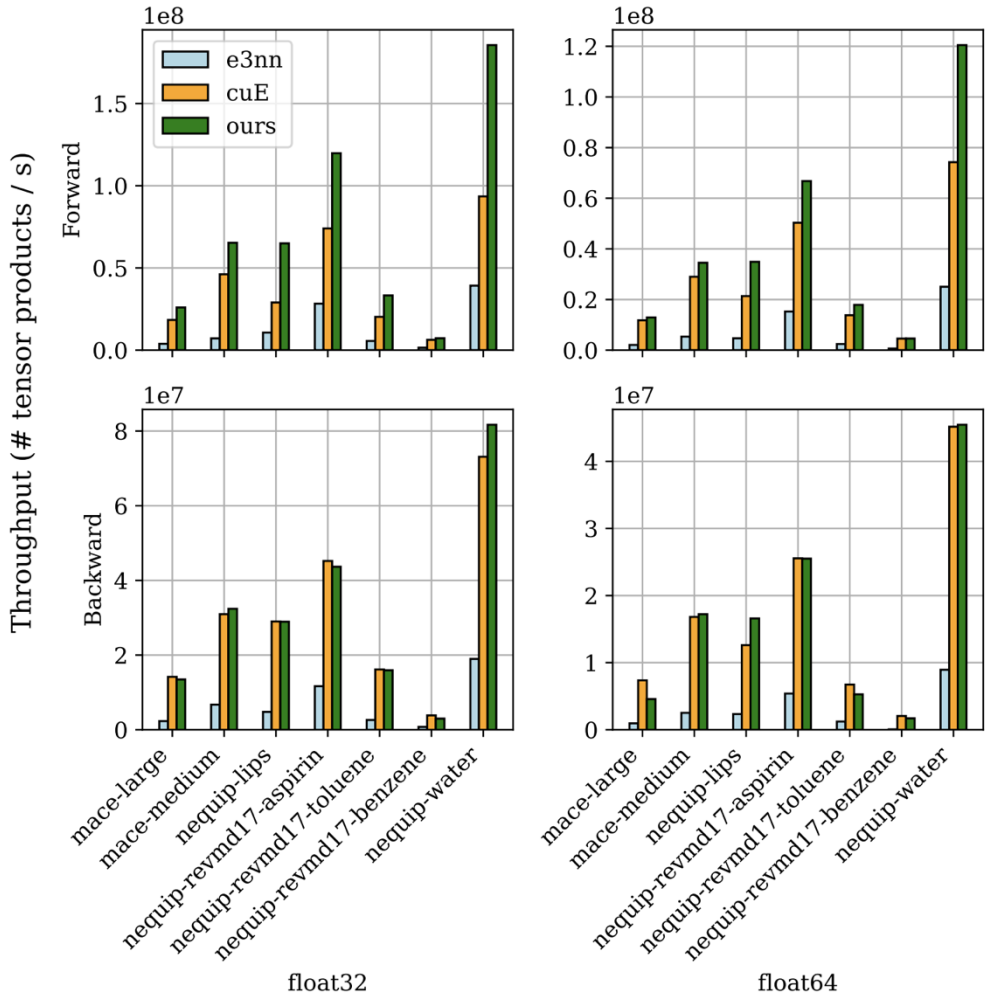
Experiments on DOE Flagship Systems

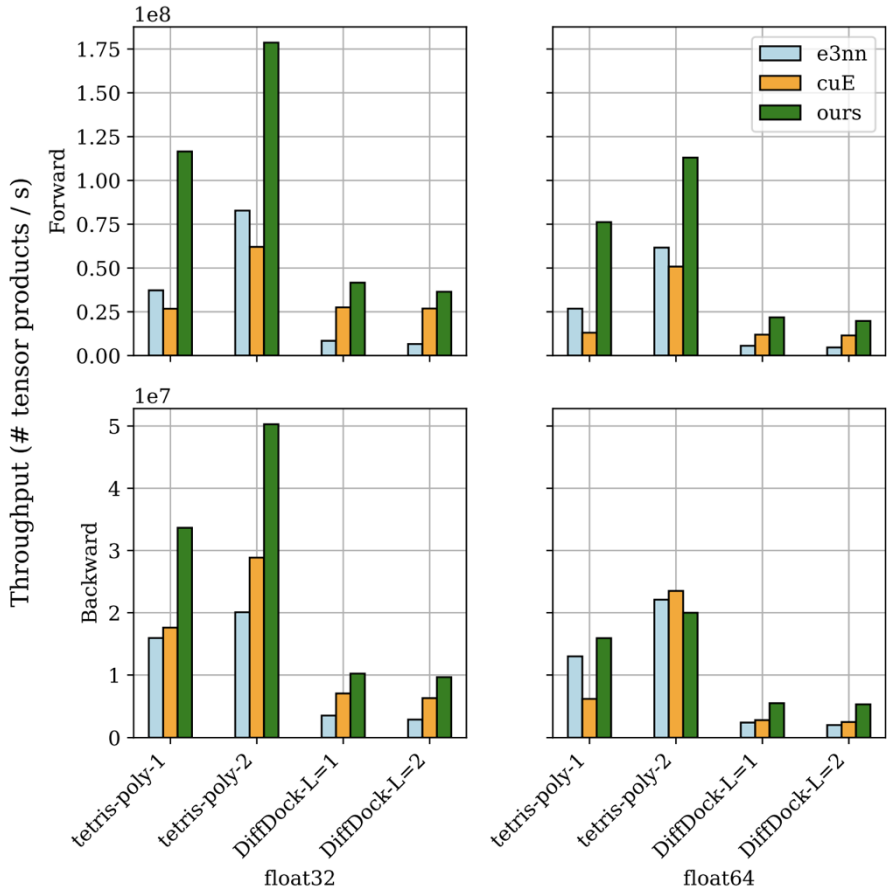
NERSC Perlmutter, A100 GPUs



OLCF Frontier, MI250x GPUs

A100 Tensor Product Throughput



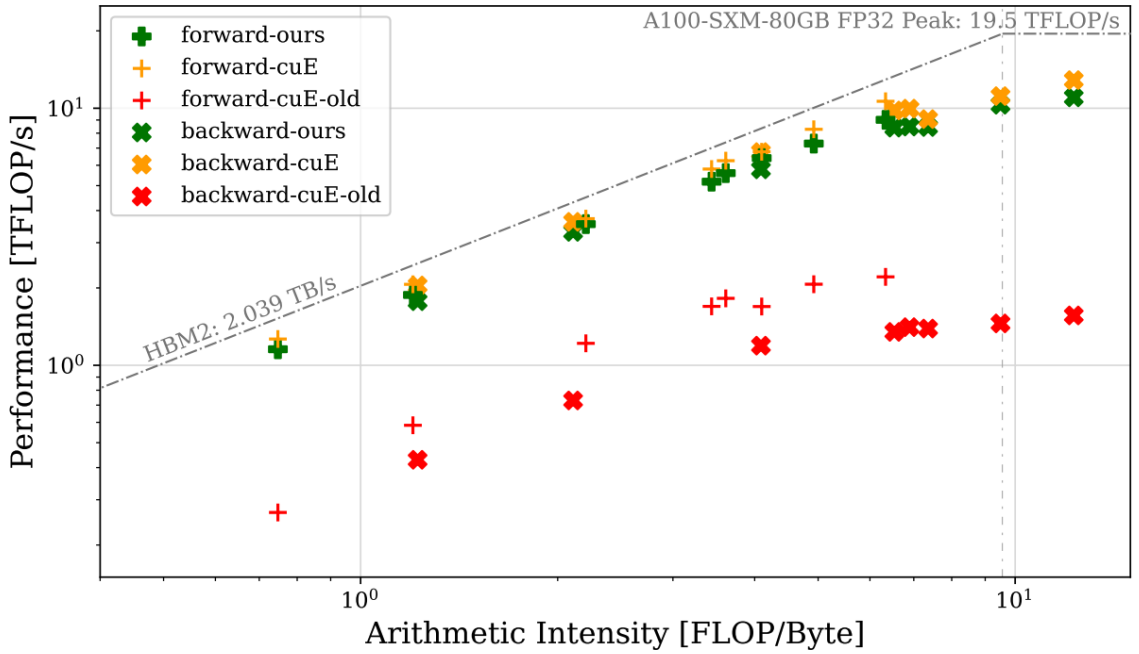


Benchmarks conducted with cuE v0.4.0

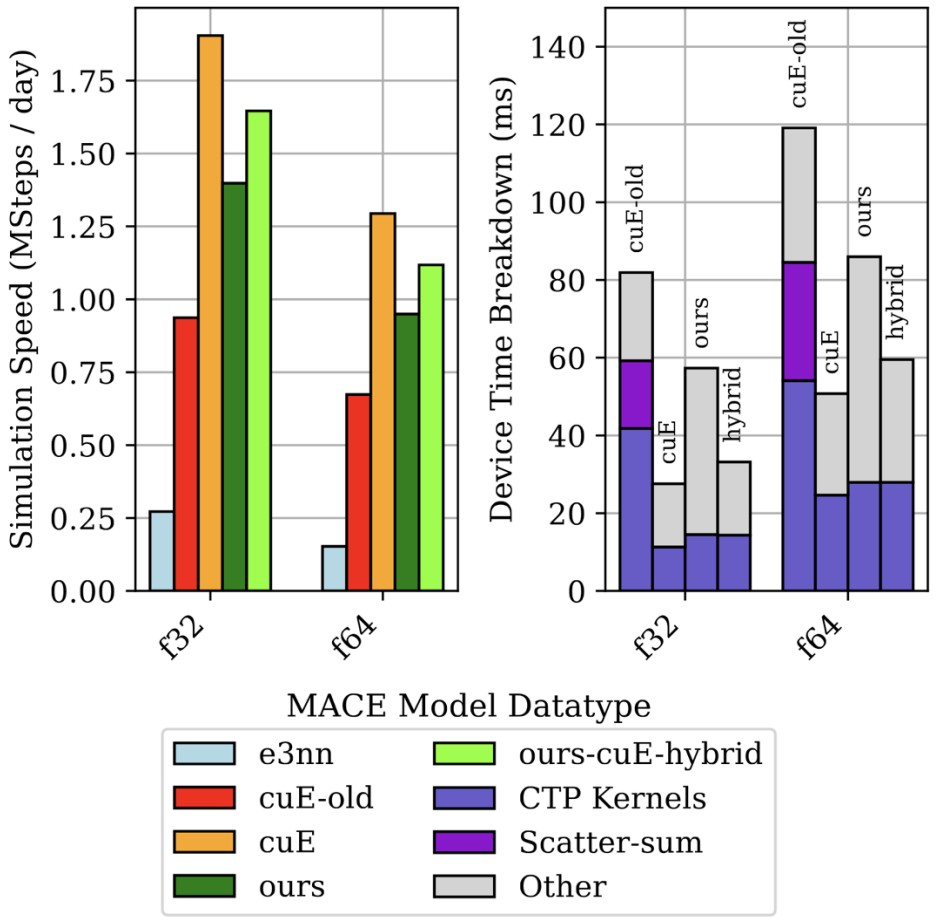
Cross-Platform Performance and Roofline Analysis

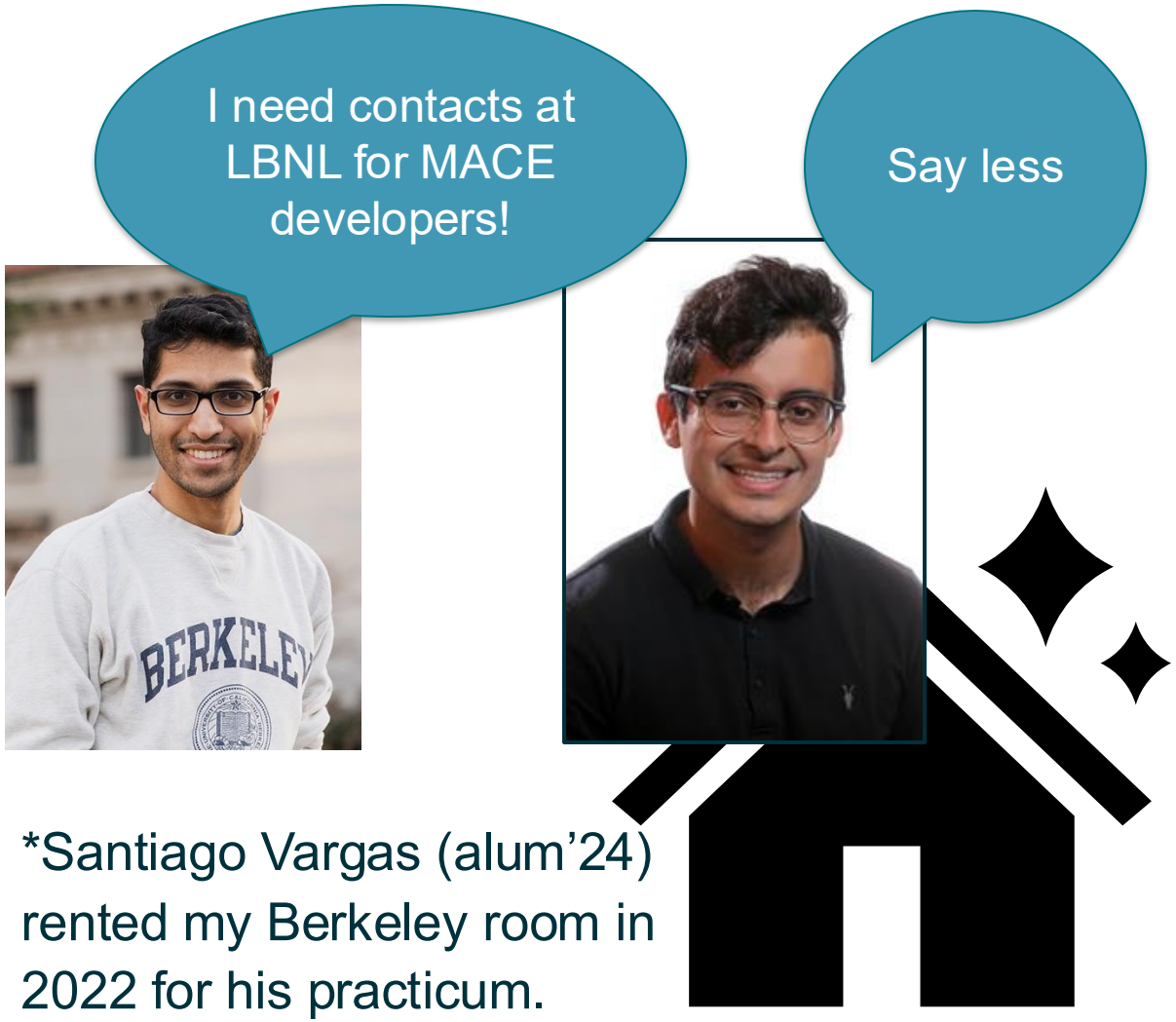
GPU	forward			backward		
	e3nn	cuE	ours	e3nn	cuE	ours
A100	13	2.8	**2.0**	21	**3.5**	3.7
A5000	29	4.2	**3.8**	42	**9.3**	11
MI250x	41	-	**3.0**	128	-	**15**

MACE-large isolated tensor product runtime (ms), batch size 50K FP32 unfused.



Accelerating MACE (Lawrence Berkeley Lab)





*Santiago Vargas (alum'24) rented my Berkeley room in 2022 for his practicum.

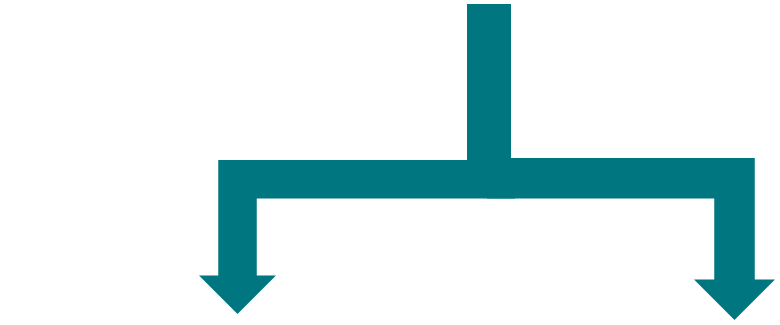
Conclusions and Further Work

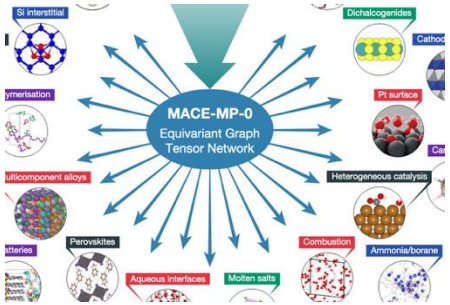
- OpenEquivariance is now **officially integrated** into Nequip and MACE.

- 5-6x E2E speedup, OOM memory reduction via:
  - Synchronization-light parallelization strategies.
  - Careful management of GPU shared memory.
  - Register caching / loop unrolling.
  - Kernel fusion, and more!

- Tentative future work: numerical precision tuning, JAX wrappers, AMD performance tuning.

```
pip install openequivariance
```





The Real Atoms were the Friends Along the Way

(that doesn’t scan) Thank you Lindsey, Kris, Michelle, Robyn, and the entire Krell staff!

Questions?
Scan the code for the package and paper.