Engineering Fast Kernels for O(3)-Equivariant Deep Networks

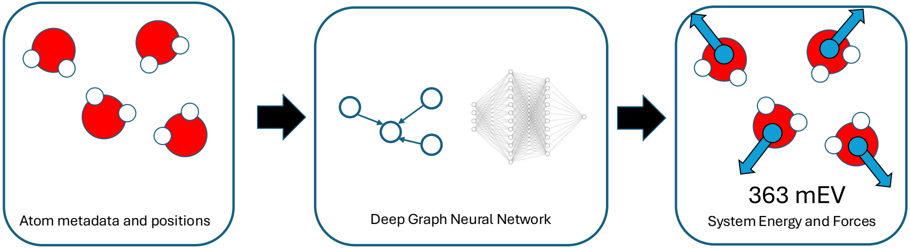
SIAM Computational Science and Engineering 2025

March 4, 2025

Vivek Bharadwaj *[1,2], Austin Glover *[1],
Aydın Buluç [2,1], James Demmel [1]

* denotes equal contribution
[1] EECS Department, UC Berkeley [2] AMCR, Lawrence Berkeley National Lab

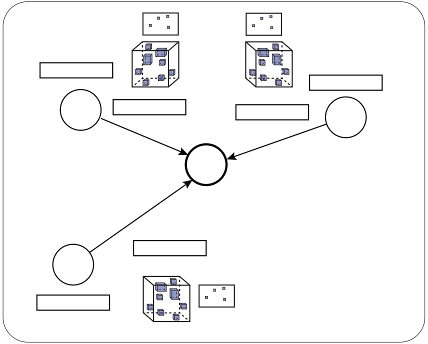
Geometric Deep Learning in Atomic Simulation



- Geometric deep learning achieves SOTA performance for fast interatomic potential calculation. Key primitive is the **message-passing graph neural network**.

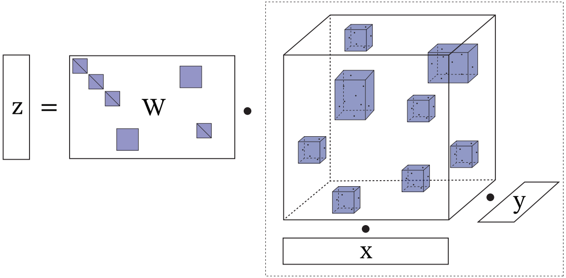
- Input: atom metadata and positions. Output: system energy, atomic forces.

Respecting Physical Symmetries



- Message-passing GNNs generate messages for each node and edge.
- O(3)-Equivariance: if the 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

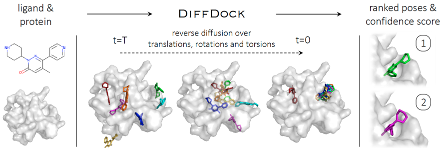


$$\boldsymbol{z} = \boldsymbol{W} \cdot \boldsymbol{P} \cdot (\boldsymbol{x} \otimes \boldsymbol{y}) = \boldsymbol{W} \sum_{i=1,j=1}^{m,n} \boldsymbol{x}[i]\, \boldsymbol{y}[j]\, \mathcal{P}[ij:]$$

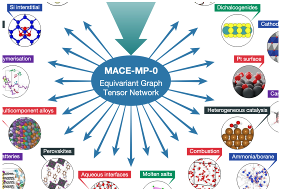
Kernel typically executes on a large batch of $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{W})$ inputs ($B \approx 10^5 - 10^7$).

Some Large Equivariant NN Projects

(a) Nequip [Bat+22]



(b) DiffDock [Cor+23]

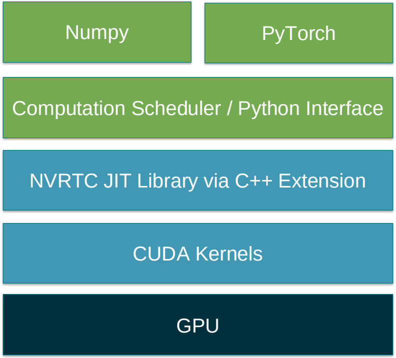


(c) MACE-MP0 [Bat+24]

Our Contributions [Bha+25]

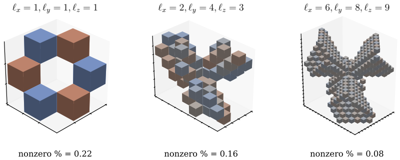
We introduce *OpenEquivariance*, a fast kernel generator for CG tensor products with up to **10x speedup** over the popular e3nn package and **2x** over NVIDIA cuE (joint work with Austin Glover).

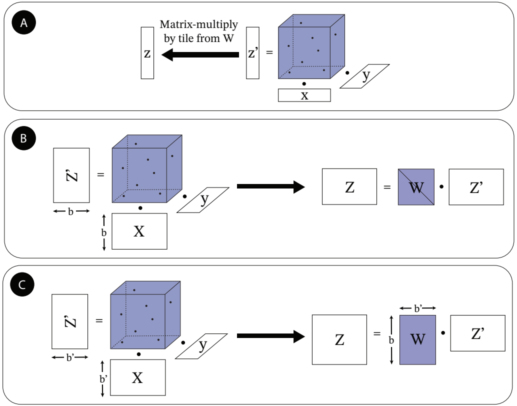
- Exploits sparse structure in the multilinear map tensor via JIT, register caching, loop unrolling.
- Provides FlashAttention-style [Dao+22] backward pass, novel identities for higher gradients.
- Fuses CG tensor product with graph convolution, saving order of magnitude in computation / memory on chemistry foundation models.



Subkernels of the CG Tensor Product

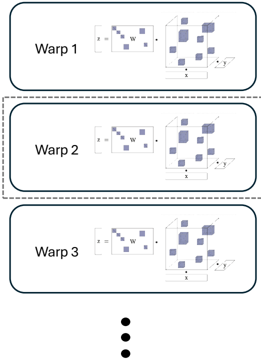
- Many nonzero blocks repeated in the sparse tensor (known at model compile-time).
- Operation splits into many smaller operations (subkernels)





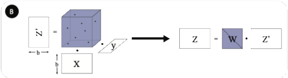
A Roadmap to Efficient CG Kernels

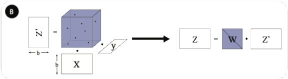
Assign each batch element to distinct GPU warp



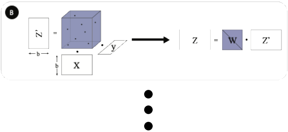
Schedule subkernels to manage SMEM usage, avoid memory traffic

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;
```

Warp-Level Forward 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 $\boldsymbol{Z} \mathrel{+}= \boldsymbol{W} \cdot \boldsymbol{Z}'$.

Gradient Calculations

Algorithm Subkernel C Warp-Level Backward

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: $\boldsymbol{G}_Z \in \mathbb{R}^{b \times (2\ell_z+1)}$, sparse tensor $\mathcal{P}^{(\ell_x,\ell_y,\ell_z)}$
  Threads collaboratively compute $\boldsymbol{G}'_Z = \boldsymbol{W}^\top \cdot \boldsymbol{G}_Z$

  for $t = 1...b'$ **do** ▷ Parallel over threads
    Load $\boldsymbol{x}_{\text{reg}} = \boldsymbol{X}[t,:]$, $\boldsymbol{y}_{\text{reg}} = \boldsymbol{y}$, $\boldsymbol{g}'_{z\text{reg}} = \boldsymbol{G}'_Z[t:]$
    Initialize $\boldsymbol{g}_{\text{x reg}}, \boldsymbol{g}_{\text{y reg}}, \boldsymbol{g}_{\text{w reg}}, \boldsymbol{z}_{\text{reg}}$ to 0.

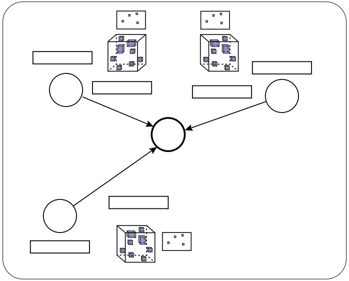
    for $(i,j,k,v) \in \text{nz}(\mathcal{P}^{(\ell_x,\ell_y,\ell_z)})$ **do** ▷ Unroll via JIT
      $\boldsymbol{g}_{\text{x reg}}[i] \mathrel{+}= v \cdot \boldsymbol{y}_{\text{reg}}[j] \cdot \boldsymbol{g}'_{\text{z reg}}[k]$
      $\boldsymbol{g}_{\text{y reg}}[j] \mathrel{+}= v \cdot \boldsymbol{x}_{\text{reg}}[i] \cdot \boldsymbol{g}'_{\text{z reg}}[k]$
      $\boldsymbol{z}_{\text{reg}}[k] \mathrel{+}= v \cdot \boldsymbol{x}_{\text{reg}}[i] \cdot \boldsymbol{y}_{\text{reg}}[j]$

    Store $\boldsymbol{g}_y = \text{warp-reduce}(\boldsymbol{g}_{\text{y reg}})$
    Store $\boldsymbol{G}_x[t,:] = \boldsymbol{g}_{\text{x reg}}$ and $\boldsymbol{Z}'[t,:] = \boldsymbol{z}_{\text{reg}}$
  Threads collaboratively compute $G_W = \boldsymbol{G}_Z \cdot (\boldsymbol{Z}')^\top$

- Given $\partial E/\partial \boldsymbol{z}$, want to compute $\partial E/\partial \boldsymbol{x}, \partial E/\partial \boldsymbol{y}, \partial E/\partial \boldsymbol{W}$.
- FA [Dao+22] Optimization: recompute forward pass to avoid memory traffic.
- Weight matrix handled through warp matrix multiplication.
- For training, we introduce novel identities for higher partial derivatives.

Kernel Fusion

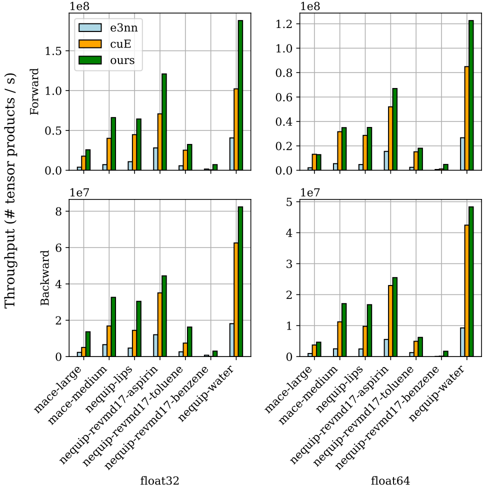
- Graph convolution has SpMM memory access pattern
- Fuse both kernels to save compute AND memory!



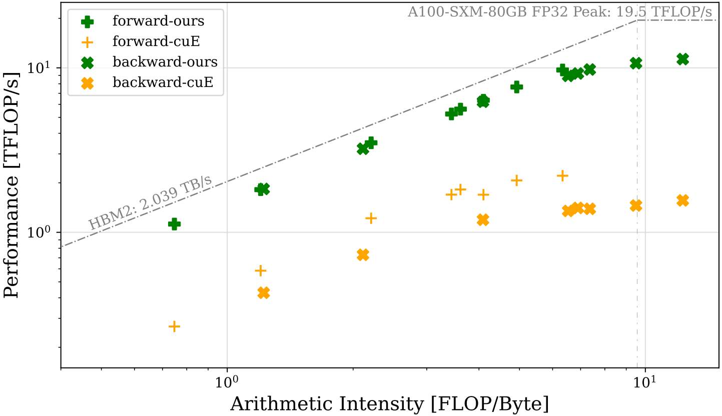
Algorithm Deterministic TP + Graph Convolution

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.

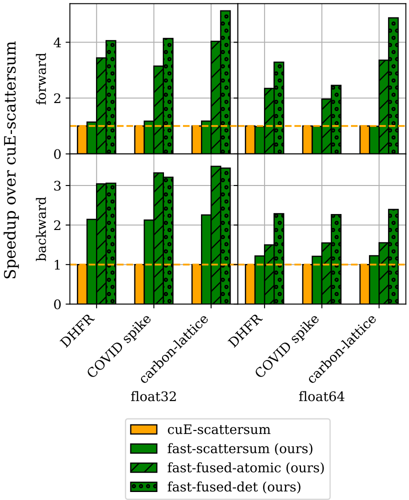
Throughput without Kernel Fusion



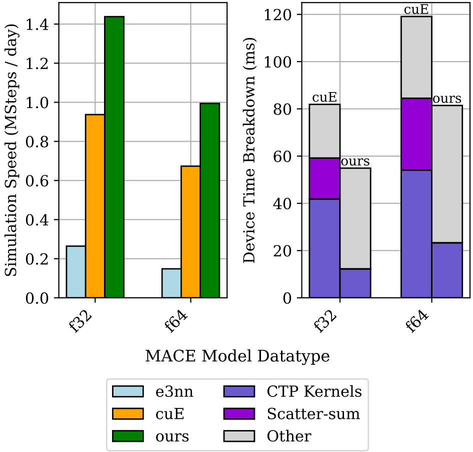
Roofline Analysis (F32)



Speedup with Kernel Fusion



Acceleration of MACE Foundation Model



Takeaways

- OpenEquivariance achieves SOTA performance on realistic input configurations, provides practical benefits for chemical foundation models.
- **Ongoing Work:** Applying our library to design compute / memory efficient ENNs for other scientific domains, e.g. high-energy physics.
- **Key Point:** Exploited structure in the tensor to engineer high performance kernels.

Thank you! Questions?

References I

[Bat+22] Simon Batzner, Albert Musaelian, Lixin Sun, Mario Geiger, Jonathan P. Mailoa, Mordechai Kornbluth, Nicola Molinari, Tess E. Smidt, and Boris Kozinsky. “E(3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials”. In: *Nature Communications* 13.1 (May 2022), p. 2453. ISSN: 2041-1723. DOI: 10.1038/s41467-022-29939-5. URL: https://doi.org/10.1038/s41467-022-29939-5.

[Bat+24] Ilyes Batatia et al. *A foundation model for atomistic materials chemistry*. 2024. arXiv: 2401.00096 [physics.chem-ph]. URL: https://arxiv.org/abs/2401.00096.

[Bha+25] Vivek Bharadwaj, Austin Glover, Aydın Buluç, and James Demmel. *An Efficient Sparse Kernel Generator for O(3)-Equivariant Deep Networks*. 2025. arXiv: 2501.13986 [cs.LG]. URL: https://arxiv.org/abs/2501.13986.

[Cor+23] Gabriele Corso, Hannes Stärk, Bowen Jing, Regina Barzilay, and Tommi S. Jaakkola. “DiffDock: Diffusion Steps, Twists, and Turns for Molecular Docking”. In: *The Eleventh International Conference on Learning Representations*. 2023.

References II

[Dao+22] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. Vol. 35. Curran Associates, Inc., 2022, pp. 16344–16359. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/67d57c32e20fd0a7a302cb81d36e40d5-Paper-Conference.pdf.