From 13 Hours to 4.6: Profiling-Driven Kernel Fusion for TensorNet in Molecular Simulation

Manpreet Singh
Undergrad Student
Department of Computer Science, Thapar Institute of Engineering & Technology, India
Embedded LLM, Singapore
msingh10_be23@thapar.edu

Abstract

Machine learning force fields such as TensorNet accelerate molecular dynamics (MD) relative to ab initio methods, but their PyTorch inference is itself bottlenecked by memory-bound, kernel-launch-heavy operations. We profile TensorNet on an NVIDIA A100 and find that element-wise tensor operations and index-based message passing together account for 60.8% of inference time. We fuse these operations into single-launch Triton kernels, obtaining a 3.14× geometric-mean speedup on isolated operations (1K–64K atoms) and a 2.82× speedup on end-to-end TensorNet inference on the MD17/MD22 benchmarks. This reduces a 1M-step, 1 ns MD trajectory from 13 hours to 4.6 hours on an A100 with numerically verified physical accuracy, corresponding to an increase from roughly 1.8 ns/day to ∼5.2 ns/day on a single A100, within the throughput range required by production drug-discovery and materials-screening pipelines. We report this as an engineering contribution with an explicit and bounded scope: our end-to-end measurements cover small organic molecules (21–42 atoms), while our operation-level microbenchmarks establish that the same fused kernels scale favorably up to 64K atoms. We make no claim of end-to-end validation at protein scale. We also report regimes where PyTorch eager mode remains competitive, including atomic-contention-heavy scatters and sub-512-atom systems, and position the approach relative to compiler-based (`torch.compile`) and hand-written CUDA (cuEquivariance-style) alternatives.

1 Introduction and Motivation

Molecular dynamics is essential to drug discovery, protein engineering, and materials design, but ab initio accuracy costs $O(N^3)$ or worse [1]. Machine learning force fields (MLFFs) trade a small accuracy loss for orders-of-magnitude speedups over first-principles methods [1, 2]. TensorNet [3] is one such architecture, representing atomic environments with equivariant Cartesian tensors rather than spherical harmonics. Despite this efficiency, TensorNet inference in its reference PyTorch implementation [4] remains dominated by memory-bound operations, specifically index-based message aggregation and tensor decomposition, that map poorly onto PyTorch's one-kernel-per-operation execution model.

Contribution. We present a profiling-driven kernel-fusion study: we identify the operations that dominate TensorNet's inference time, fuse them into custom Triton [5] kernels, and quantify the resulting speedup and its limits. **This is an engineering, not an architectural, contribution:** we change no model weights, no equivariance structure, and no numerics beyond floating-point-safe kernel fusion, which we verify explicitly (§4.1). Throughout, every reported number compares against *PyTorch 2.7.1 eager-mode execution* (no `torch.compile`) on the same A100 accelerator; we make this baseline explicit because it materially changes the interpretation of the result, and we return to it in §4.2.

An extended version of this work was accepted at the EurIPS Workshop:Machine Learning for Simulations in Biology & Chemistry (SimBioChem) 2025. This submission is a short paper condensed from that longer work.

Massachusetts Institute of Technology - Molecular Machine Learning Conference 2026.

2 Related Work

MLFF architectures. SchNet [1] and PaiNN [2] popularized learned, equivariant message passing; TensorNet [3] instead uses Cartesian tensor decompositions, avoiding explicit spherical-harmonic machinery. **Compiler-level fusion.** `torch.compile` [6] and TorchScript [7] fuse operator graphs automatically, but we found (§3) that `torch.compile` does not effectively fuse TensorNet's graph-structured index operations, motivating hand-written kernels. **Hand-written GPU kernels.** The majority of custom-kernel acceleration work for interatomic potentials has targeted $\mathrm{SE}(3)$-*steerable convolutions* in NequIP/Allegro- and MACE-style architectures, e.g. cuEquivariance-style fused kernels [8], the high-performance Allegro/NequIP kernels of Tan et al. [9], the large-scale kernels of Musaelian et al. [10], and MACE itself [11, 12]. JAX-based differentiable MD [13] takes a compiler-first route in a different framework entirely. TensorNet's Cartesian-tensor formulation is structurally different from these spherical-harmonic pipelines, and to our knowledge has not previously received dedicated kernel-fusion treatment. Triton has separately shown 2–3$\times$ speedups for transformer attention [14] and sparse operations [15, 16], but not for equivariant molecular potentials. We view our contribution as narrow and complementary to this literature: a demonstration that the same profiling-driven fusion methodology transfers to TensorNet's specific operator pattern, not a claim of a new fusion technique.

3 Method

Profiling setup. We profile the TensorNet reference implementation in TorchMD-NET [4] using PyTorch's built-in profiler on a single A100 SXM4 GPU with 80 GB of memory, with a 4,096-atom system (cutoff radius 5.0 Å, average 32 neighbors/atom, 131,072 total edges, representative of a medium-sized protein). We confirmed `torch.compile` gives negligible improvement on this graph-structured workload, so all comparisons use eager-mode PyTorch as the baseline. Three operation categories account for 60.8% of execution time: element-wise tensor algebra (24.8%), index-based message aggregation (36.0%), and GEMMs (12.4%, already cuBLAS-optimized and left untouched). These operations are memory-bound due to the irregular access patterns induced by molecular graph connectivity; additional background on the GPU-level bottleneck and the motivation for fusion is provided in Appendix A.

Fused kernels. We target the two non-GEMM bottlenecks with five Triton kernel families: (i) vector-to-symmetric/skew-tensor construction, which PyTorch executes as 5 separate kernels (outer product, trace, symmetrize, subtract) and we fuse into 1; (ii) tensor decomposition (6→1 kernels); (iii) index-gather/multiply/scatter message passing (4→1 kernels); and (iv) a smooth radial cutoff fused directly into message passing (8→1 kernels), since cutoffs are evaluated billions of times over an MD trajectory. All kernels are tuned via grid search over block size (128–256 in 1D, 16–32 in 2D) and use atomic writes only where scatter conflicts are unavoidable.

4 Results

Hardware/software. A100 SXM4 GPU with 80 GB of memory, AMD EPYC 7763, PyTorch 2.7.1, CUDA 12.6, Triton 2.0. Each measurement is the median of 5 runs of 100 iterations (20 warmup).

Table 1 summarizes both regimes. Micro-benchmark speedup peaks at 4.89$\times$ for the fused cutoff+message-passing kernel (8 launches→1) and averages 3.14$\times$ across all operations and sizes; end-to-end TensorNet inference is 2.82$\times$ faster on average. The gap between the two (3.14$\times$ vs. 2.82$\times$) is explained entirely by the GEMM component (22.1% of time), which already runs through cuBLAS and is not a fusion target, plus residual Python dispatch overhead. For a 1M-step, 1 fs-timestep (1 ns) trajectory this converts to 13 h → 4.6 h on one A100. For context, while classical MD engines routinely exceed 1,000 ns/day, a practical target for accurate MLFFs in drug discovery screening is currently ∼1–10 ns/day for medium-sized proteins. Our optimized throughput of ∼5.2 ns/day (4.6 hours per ns) on a single A100 crosses the threshold for viable production screening, bringing ML-driven MD closer to the utility of classical force fields.

4.1 Physical accuracy

We validate every fused kernel against the PyTorch reference with `torch.allclose` ($\mathrm{rtol} = \mathrm{atol} = 10^{-4}$) over 1,000 inference steps on MD17/MD22: energies and forces match to $< 10^{-6}$ kcal/mol

Table 1: Speedup over PyTorch eager mode. Microbenchmarks report isolated operations at four system sizes (all fusion targets scale to 64K atoms, well beyond the molecules used for end-to-end validation). End-to-end results report the full TensorNet forward pass on MD17 (aspirin, 21 atoms) and MD22 (Ac-Ala3-NHMe, 42 atoms), the two datasets standard in the TensorNet/TorchMD-NET literature.

Micro-benchmark (op.)	1K	4K	16K	64K
Vector→SymTensor	3.77×	3.82×	3.88×	3.04×
Vector→SkewTensor	3.08×	3.06×	3.09×	2.33×
Tensor decomposition	2.20×	2.21×	2.09×	2.18×
Message passing	3.45×	3.51×	3.10×	2.61×
Fused cutoff + MP	4.02×	4.89×	3.45×	2.95×
Geo. mean	**3.21**×	**3.38**×	**3.05**×	**2.60**×

End-to-end (dataset, atoms)	Batch 1	Batch 32	Kernel launches
MD17 Aspirin (21)	2.54×	2.85×	−75% to −88%
MD22 Ac-Ala3-NHMe (42)	2.96×	2.85×	(per fused block)

and $< 10^{-5}$ kcal/mol/Å respectively, and tensor symmetry/tracelessness properties are preserved to $< 10^{-7}$. Fusion is a computational reorganization, not an approximation.

4.2 Baseline and scope, made explicit

Two scoping choices materially affect how these numbers should be read, and we state both plainly rather than leave them implicit. First, *all* speedups in this paper are relative to **PyTorch eager mode**, not `torch.compile`; we found `torch.compile` ineffective on TensorNet's graph-structured index operations (§3), so eager mode is the correct and, we believe, fair baseline, but it is a different (weaker) baseline than a compiled-graph comparison would be. Second, our *end-to-end* validation is limited to the small molecules (21–42 atoms) provided by the standard MD17/MD22 benchmarks; we have not run end-to-end inference or full MD trajectories on protein-sized systems. In standard ML-accelerated MD workflows, native engines like OpenMM act as the integrator but rely on PyTorch-based frameworks (e.g., TorchMD-NET) via custom wrappers to evaluate neural forces. Consequently, the memory-bound bottlenecks of PyTorch eager execution directly throttle the overall OpenMM simulation throughput. Our comparison is strictly scoped to the PyTorch-side inference (PyTorch-TensorNet vs. Triton-TensorNet) because accelerating this specific backend directly unblocks the hybrid MD pipeline. What we *can* say, and what our micro-benchmarks in Table 1 directly show, is that every individual fused kernel continues to outperform PyTorch up to 64K atoms, which is the operation-level evidence for scalability; we do not extrapolate this to an end-to-end protein-scale claim we have not measured. We similarly make no direct throughput comparison to specialized native-code MD engines (e.g. ACEMD, GROMACS, OpenMM's own kernels); our comparison is strictly PyTorch-TensorNet vs. Triton-TensorNet within the same TorchMD-NET codebase.

4.3 Where PyTorch still wins

Reconciling operation-level and end-to-end behavior: at the level of an *individual* scatter operation with heavy write conflicts (e.g. 4D index-add aggregating $[E, C, 3, 3] \to [N, C, 3, 3]$ tensors), atomic contention makes Triton up to 2.3× *slower* than PyTorch's hand-tuned atomics, and isolated operations below ∼512 atoms are dominated by kernel-launch overhead rather than compute. This does not contradict the 21–42-atom end-to-end speedups in Table 1: the end-to-end path *fuses many small operations together*, so the launches removed by fusion outweigh the per-kernel overhead even at small molecule sizes, whereas the isolated-operation regression is specific to operations run standalone. In production, we recommend a hybrid dispatch that routes only fusion-favorable operations to Triton and leaves high-contention scatters and simple reductions to PyTorch.

5 Conclusion

Profiling-driven Triton fusion of TensorNet's memory-bound operations yields a 3.14× micro-benchmark and 2.82× end-to-end inference speedup over PyTorch eager mode, with bitwise-verified physical accuracy and a 75–88% reduction in kernel launches, at no cost to model weights or equivariance. The methodology, consisting of profiling, fusing memory-bound operator groups, verifying numerics, and benchmarking against the correct baseline, is architecture-agnostic and, we believe, directly transferable to PaiNN- and MACE-style potentials, though each requires its own kernel implementation. We have deliberately scoped our claims to what we measured: eager-mode PyTorch as the baseline, small-molecule end-to-end validation with protein-scale evidence coming only from operation-level microbenchmarks, and an explicit account of the atomic-contention and small-system regimes where PyTorch remains preferable. The Triton kernels and accompanying benchmark scripts are available at `https://github.com/anonymous1234556-peer/TorchMD-triton`.

Acknowledgements

We acknowledge GPU infrastructure provided by Embedded LLM, Singapore, and thank the TorchMD-NET development team for their open-source implementation.

References

[1] Kristof T. Sch"utt, Pieter-Jan Kindermans, Huziel E. Sauceda, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert M"uller. SchNet: A continuous-filter convolutional neural network for modeling quantum interactions. In *Advances in Neural Information Processing Systems*, volume 30, pages 991–1001, 2017. 1, 2

[2] Kristof T. Sch"utt, Oliver T. Unke, and Michael Gastegger. Equivariant message passing for the prediction of tensorial properties and molecular spectra. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 9377–9388, 2021. 1, 2

[3] Guillem Simeon and Gianni De Fabritiis. TensorNet: Cartesian tensor representations for efficient learning of molecular potentials. In *Advances in Neural Information Processing Systems*, volume 36, pages 37334–37353, 2023. 1, 2

[4] Raul P. Pelaez, Guillem Simeon, Raimondas Galvelis, Antonio Mirarchi, Peter Eastman, Stefan Doerr, Philipp Th"olke, Thomas E. Markland, and Gianni De Fabritiis. TorchMD-Net 2.0: Fast neural network potentials for molecular simulations. *Journal of Chemical Theory and Computation*, 20(10):4076–4087, 2024. doi: 10.1021/acs.jctc.4c00253. 1, 2

[5] Philippe Tillet, Hsiang-Tsung Kung, and David D. Cox. Triton: An intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 10–19, 2019. doi: 10.1145/3315508.3329973. 1

[6] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, et al. PyTorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 929–947, 2024. doi: 10.1145/3620665.3640366. 2

[7] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas K"opf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, pages 8024–8035, 2019. 2

[8] 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. *Nature Communications*, 13: 2453, 2022. doi: 10.1038/s41467-022-29939-5. 2

[9] Chuin Wei Tan, Marc L. Descoteaux, Mit Kotak, Gabriel de Miranda Nascimento, Seán R. Kavanagh, Laura Zichi, Menghang Wang, Aadit Saluja, Yizhong R. Hu, Tess Smidt, Anders Johansson, William C. Witt, Boris Kozinsky, and Albert Musaelian. High-performance training and inference for deep equivariant interatomic potentials, 2025. 2

[10] Albert Musaelian, Simon Batzner, Anders Johansson, Lixin Sun, Cameron J. Owen, Mordechai Kornbluth, and Boris Kozinsky. Learning local equivariant representations for large-scale atomistic dynamics. *Nature Communications*, 14:579, 2023. doi: 10.1038/s41467-023-36329-y. 2

[11] Ilyes Batatia, Dávid Péter Kovács, Gregor N. C. Simm, Christoph Ortner, and Gábor Csányi. MACE: Higher order equivariant message passing neural networks for fast and accurate force fields. In *Advances in Neural Information Processing Systems*, volume 35, pages 11423–11436, 2022. 2

[12] Ilyes Batatia, Simon Batzner, Dávid Péter Kovács, Albert Musaelian, Gregor N. C. Simm, Ralf Drautz, Christoph Ortner, Boris Kozinsky, and Gábor Csányi. The design space of $E(3)$-equivariant atom-centered interatomic potentials, 2022. 2

[13] Samuel S. Schoenholz and Ekin Dogus Cubuk. JAX MD: A framework for differentiable physics. In *Advances in Neural Information Processing Systems*, volume 33, 2020. 2

[14] Tri Dao, Daniel Y. 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*, volume 35, 2022. 2

[15] Saman Ashkiani, Martin Farach-Colton, and John D. Owens. A dynamic hash table for the GPU. In *2018 IEEE International Parallel and Distributed Processing Symposium*, pages 419–429, 2018. doi: 10.1109/IPDPS.2018.00052. 2

[16] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2022. 2

A Primer on GPU Memory Bottlenecks in Message Passing

For folks in the molecular simulation field who may lack a background in GPU hardware operations, this section provides context on why message passing is computationally bottlenecked and how our approach addresses it.

In molecular graph neural networks, operations like `index_select` (gather) and `index_add` (scatter) are inherently memory-bound. GPUs achieve peak performance when they can read data in contiguous, predictable chunks (a process called memory coalescing). However, molecular graphs have irregular connectivity because atoms interact with changing neighbors as the simulation progresses. This irregular structure forces the GPU to make random, non-contiguous memory access patterns, which starves the Arithmetic Logic Units (ALUs) because they must spend a majority of cycles waiting for data to be fetched from memory.

Standard PyTorch executes these operations through multiple, fragmented kernel launches, each incurring memory read/write overhead. Triton mitigates this by using a block-based programming model. By fusing multiple operations into a single kernel, Triton keeps the necessary data in fast, on-chip SRAM for longer, significantly reducing the expensive round-trips to global memory and bypassing the bottleneck of fragmented kernel launches.