

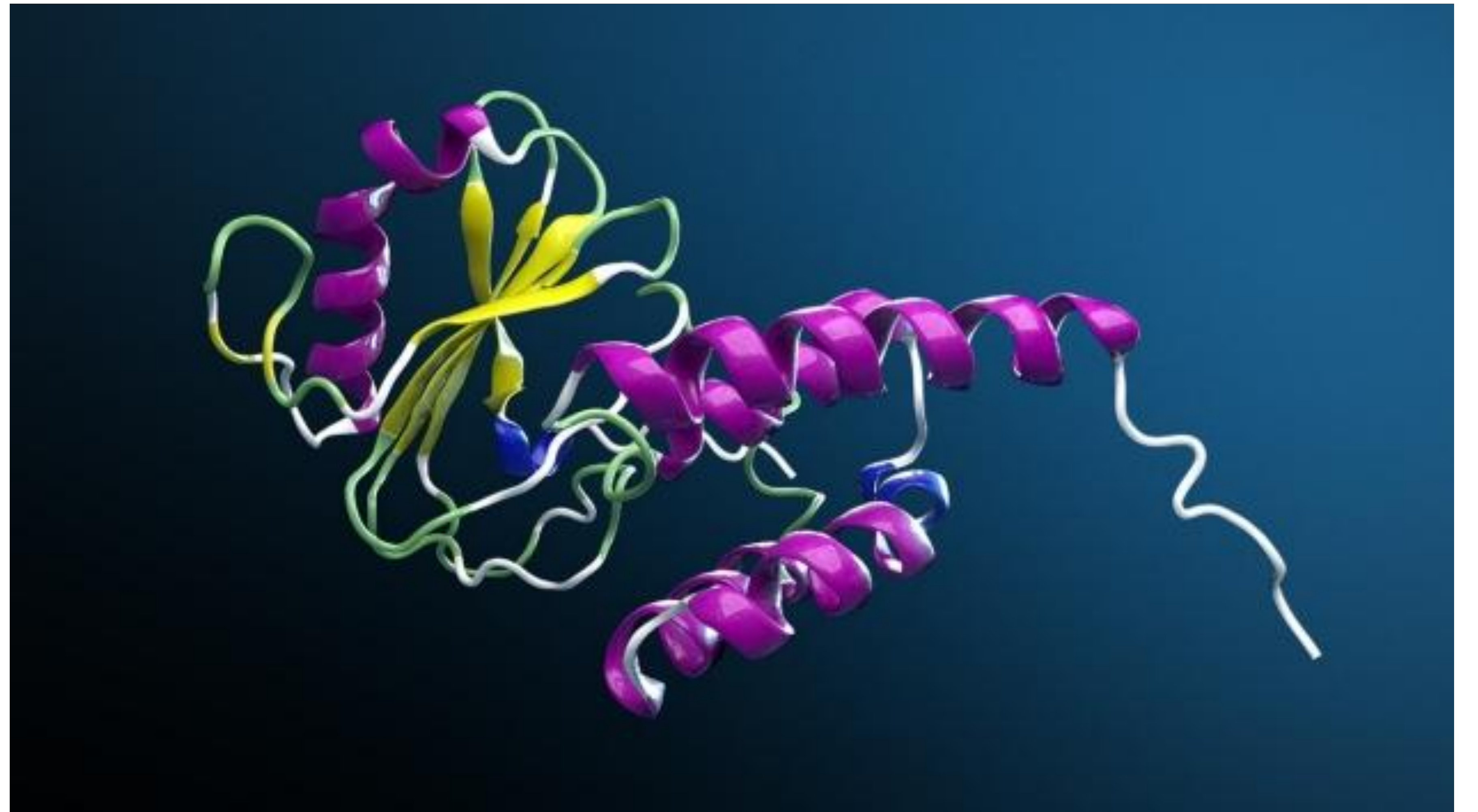


nvMolKit: Accelerating the RDKit on GPUs for high throughput computational chemistry workflows

Kevin Boyd

Who are we? Nvidia BioNemo

- <https://www.nvidia.com/en-us/clara/biopharma>
- Mission is to accelerate drug discovery
- We provide a variety of libraries and services
 - BioNemo Framework - open-source machine learning framework for training Deep Learning models
 - BioNemo NIMs: Optimized inference for protein folding, small molecule generation, and more
 - cuEquivariance - high performance primitives for equivariant neural networks
 - cuik-molmaker - fast molecular features for chemprop
 - Research efforts for generative and predictive models



Contributors to today's work

- Saeed Paliwal
- Eva Xue
- Alireza Moradzadeh
- Xuanguo Huang
- Nia Dickson
- Srimukh Veccham

Why target the RDKit

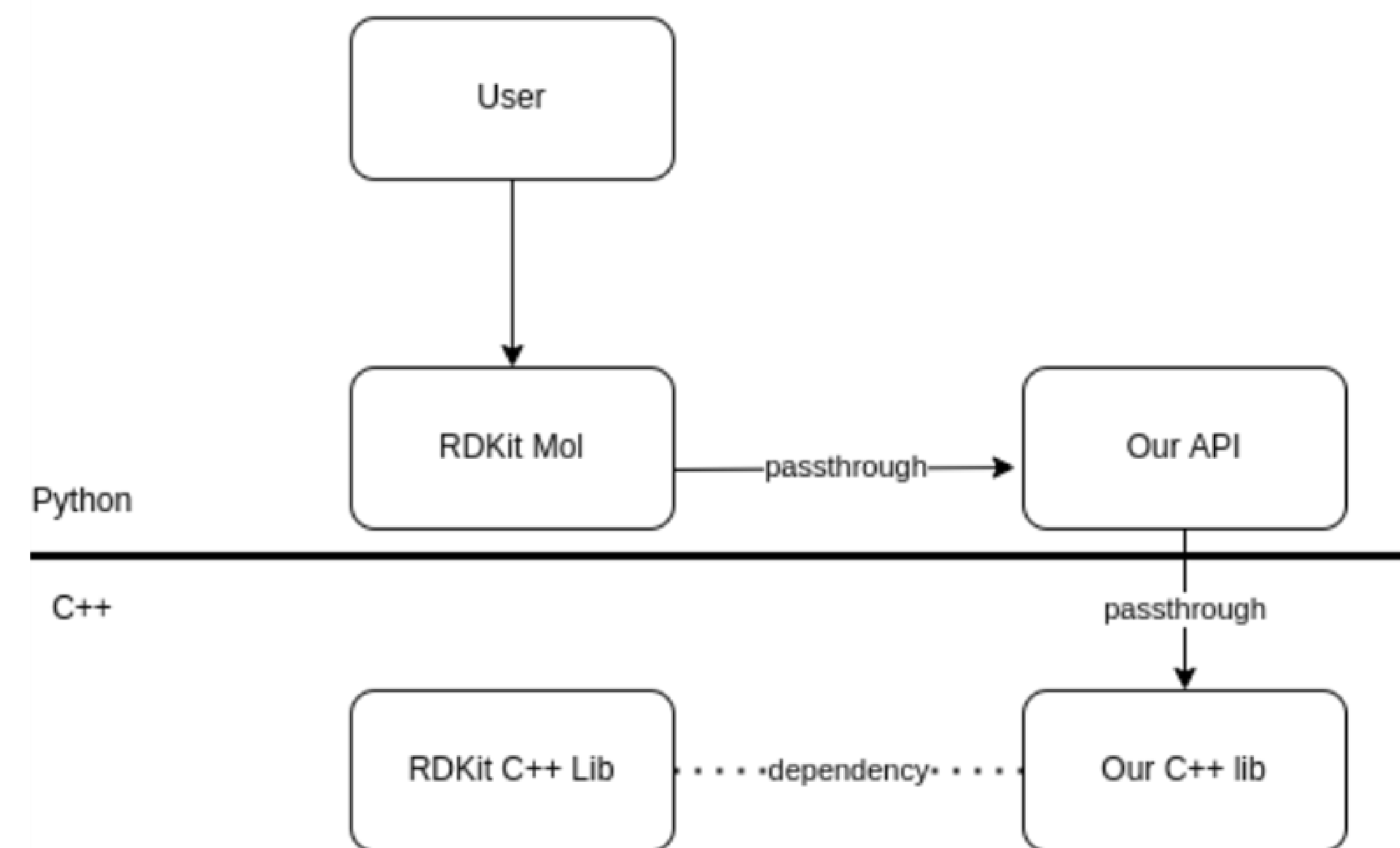
- It's used everywhere
- Some RDKit functions are used directly in high throughput workloads
- The RDKit is often a preprocessing step for ML models. Sometimes these steps can bottleneck ML workloads

Agenda

- nvMolKit introduction
- GPU programming basics
- GPU-accelerated similarity
- GPU-accelerated MMFF relaxation
- GPU-accelerated Conformer Generation
- CPU Accelerations

nvMolKit Overview

- CUDA / C++ library with python bindings
- Links to RDKit at C++ level, uses the same Boost Python bindings, so RDKit structures can be passed directly to nvMolKit
- Provides nearly drop-in replacements for RDKit APIs, with changes to allow batches of operations
- Free, open source, permissively licensed.
- <https://github.com/NVIDIA-Digital-Bio/nvMolKit>



```
confs_per_molecule = 100
params = ETKDGv3()
params.useRandomCoords = True

# RDKit version
for mol in mols:
    EmbedMultipleConfs(mol, numConfs=confs_per_molecule, params=params)

# nvMolKit version
nvMolKitEmbedMolecules(mols2, confsPerMolecule=confs_per_molecule, params=params)
```


GPU programming considerations

- Lots of compute capability - often need to provide batches of operations to run in parallel
- Very good at problems with high compute density
- Specialized hardware (tensor cores) for reduced precision matrix multiplies
- Code divergence hurts performance - some CUDA cores sit idle while others work
- Peak performance of FP16 dense matmuls is ~1 petaflop for an H100
- Non-matmul FP32 peak performance is ~60-70 teraflops



Accelerated Tanimoto Similarity (and friends)

Tanimoto Similarity




- Similarity metric
- Calculation based on
- Both CPUs and GPUs (__popc)

Some Applications of the Quadrat Method

Henry Allan Gleason

Bulletin of the Torrey Botanical Club, Vol. 47, No. 1 (Jan., 1920), pp. 21-33 (13 pages)

<https://doi.org/10.2307/2480223> · <https://www.jstor.org/stable/2480223>

		
2		
0	1	0
0	0	0
2		

LINGOS, FINITE STATE MACHINES, AND SIMILARITY SEARCHING

$$T^{AB} = \frac{\sum_i \min(n_i^A, n_i^B)}{\sum_i n_i^A + \sum_i n_i^B - \sum_i \min(n_i^A, n_i^B)}$$

where n_i^A is the count of the i th Lingo type, belonging to molecule A, and s is the number of Lingo types in molecule A (or B). For decomposition into Lingos of length q , the total number of Lingos in molecule A is given by

$$\sum_i n_i^A = |A| - q + 1$$

where $|A|$ is the length of the SMILES string of molecule A (similarly for B).

The computation of T^{AB} is the problem of matching the substrings that occur in molecules A and B. This is trivial to implement using naïve (brute force) character comparisons over all pairs of Lingos in molecules A and B, but the approach has a high computational overhead in which in the worst case the number of character comparisons is of order $\Theta(|A| \times |B|)$

Bajusz, D., Rácz, A. & Héberger, K. Why is Tanimoto index an appropriate choice for fingerprint-based similarity calculations?. *J Cheminform* **7**, 20 (2015).

<https://doi.org/10.1186/s13321-015-0069-3>

J. Chem. Inf. Comput. Sci. **1998**, 38, 983–996

Chemical Similarity Searching

Peter Willett*

Krebs Institute for Biomolecular Research and Department of Information Studies, University of Sheffield, Sheffield S10 2TN, U.K.

John M. Barnard and Geoffrey M. Downs

Barnard Chemical Information, 46 Uppergate Road, Sheffield S6 6BX, U.K.

Received February 27, 1998

ausgeprägt sind. Die ganze Vorbereitung des Materials mit Hilfe mathematisch-statistischer Methoden (70 Aufnahmen) erforderte eine mehr als einmonatige Arbeit. Diese anstrengende Arbeit war aber von grossem Nutzen und mit Hilfe dieser Methoden wurde das ganze Material zur weiteren Verarbeitung gut vorbereitet.

Ausgangspunkt der Arbeit war die Haupttabelle von Aufnahmen (Tab. No. 1.), wo alle Aufnahmen eingetragen sind, die auf den durchforschten Sandgebieten der Tiefebene von Záhorie gemacht wurden. In die Tabelle wurden nur unvollständige Aufnahmen nicht aufgenommen, die nicht ein vollständiges Bild der beschriebenen Fläche gaben, bzw. die nur für informative Zwecke verfertigt wurden (Aufnahmen No. 34, 35, 44 und 49). Die Reihenfolge der Aufnahmen in der Haupttabelle (Tab. No. 1) ist teils dieselbe, wie in der Bearbeitung in der Dissertation (10), teils zufällig. Die Hauptaufnahmetabelle diene als Unterlage für die Berechnung der Summen der Aufnahmeeinheiten der Arten immer in zwei miteinander verglichenen Aufnahmen. Den Begriff *Aufnahmeeinheit der Art* führt J. Motyka ein (7, 8). Es ist dies eigentlich kombinierter Wert von Abundanz und Dominanz, der beim gegenseitigen Vergleich zweier Aufnahmen bei den einzelnen Arten in Betracht gezogen wird. Die Summen der Aufnahmeeinheiten in den einzelnen Aufnahmen sind in Tab. No. 2 auf der Diagonale von der oberen linken zur unteren rechten Ecke angeführt. Es sind dies die Werte „a“ und „b“. Die Summen der Aehnlichkeiten der Aufnahmeeinheiten von Arten, die sich immer in zwei verglichenen Aufnahmen befinden, sind in Tab. No. 2 in der oberen rechten Hälfte angeführt. Sie stellen die „v“ Werte dar.

Zur Veranschaulichung kann ein einfaches Beispiel dienen, wie es in Tab. No. 3. angeführt ist.

Die Summen der Aehnlichkeiten sind untereinander nur schwer vergleichbar. Deshalb müssen sie auf den Aehnlichkeitskoeffizienten, der in Prozenten ausgedrückt ist, umgerechnet werden. Dies kann durch eine einfache Berechnung durchgeführt werden. Zum Unterschied von der ersten Bearbeitung (M. Ružička, 11) wurde in dieser Arbeit zur Berechnung des Aehnlichkeitskoeffizienten eine andere Formel verwendet, die nach persönlicher Mitteilung von J. Motyka die gegenseitigen Beziehungen der Aufnahmen besser zum Ausdruck bringt, weil sie nicht nur die Summe der Aehnlichkeit, sondern auch die Summe der Unterschiedlichkeit ausdrückt. Es ist dies die folgende Formel:

$$Kp = 100 \cdot \frac{v}{a + b - v} \quad (1)$$

¹ und ²: Der Verfasser spricht hiermit Prof. dr. J. Motyka, dr. A. Matuszkiewicz und Prof. dr. W. Matuszkiewicz seinen Dank aus für die Hilfe und die wertvollen Ratschläge bei der Ausgewählung und Anwendung dieser Methoden während seines Studienaufenthaltes in Polen.

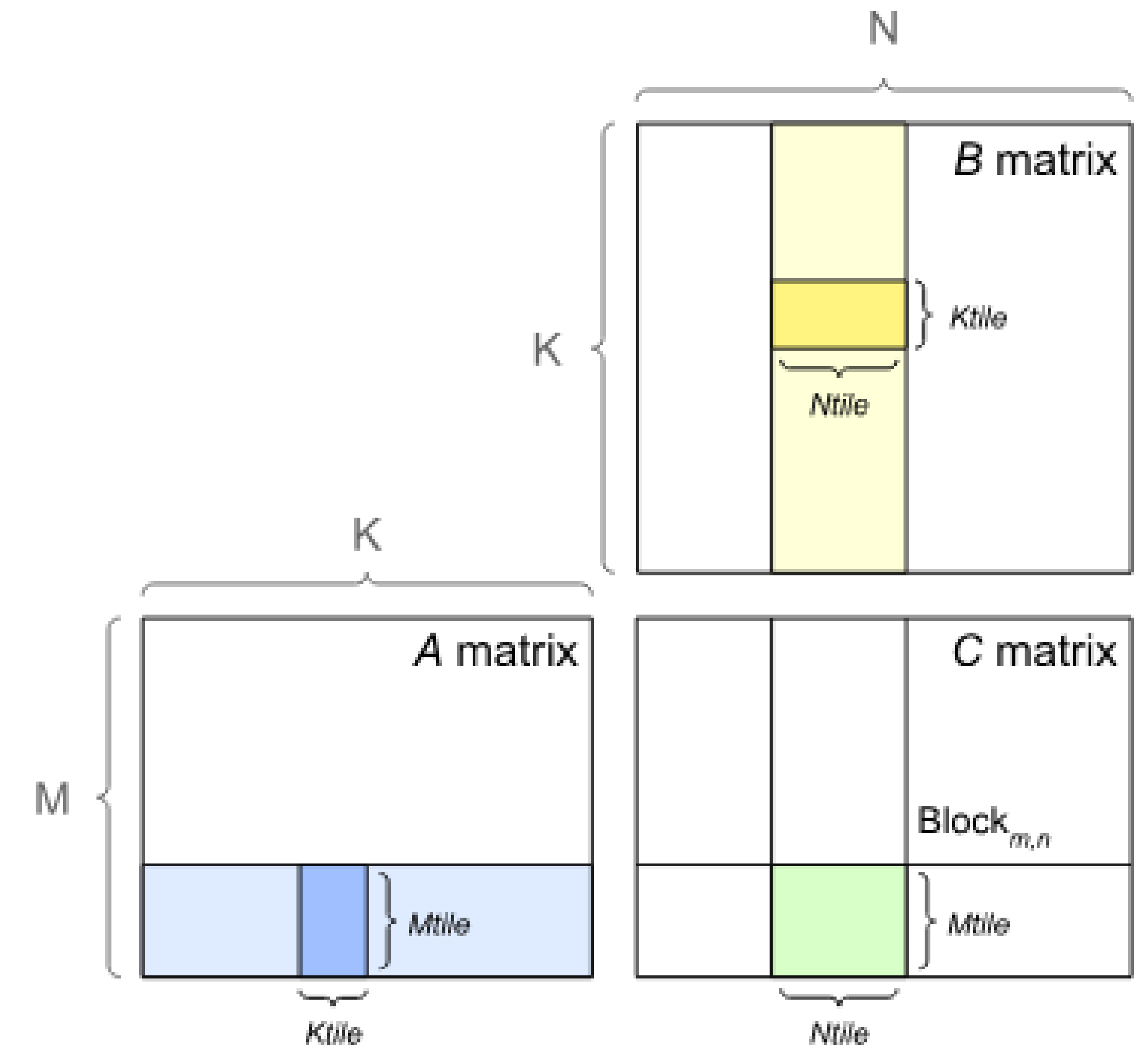
$$T(A, B) = \frac{c}{a + b - c}$$

- a : total number of unique
- b : total number of unique
- c : number of unique

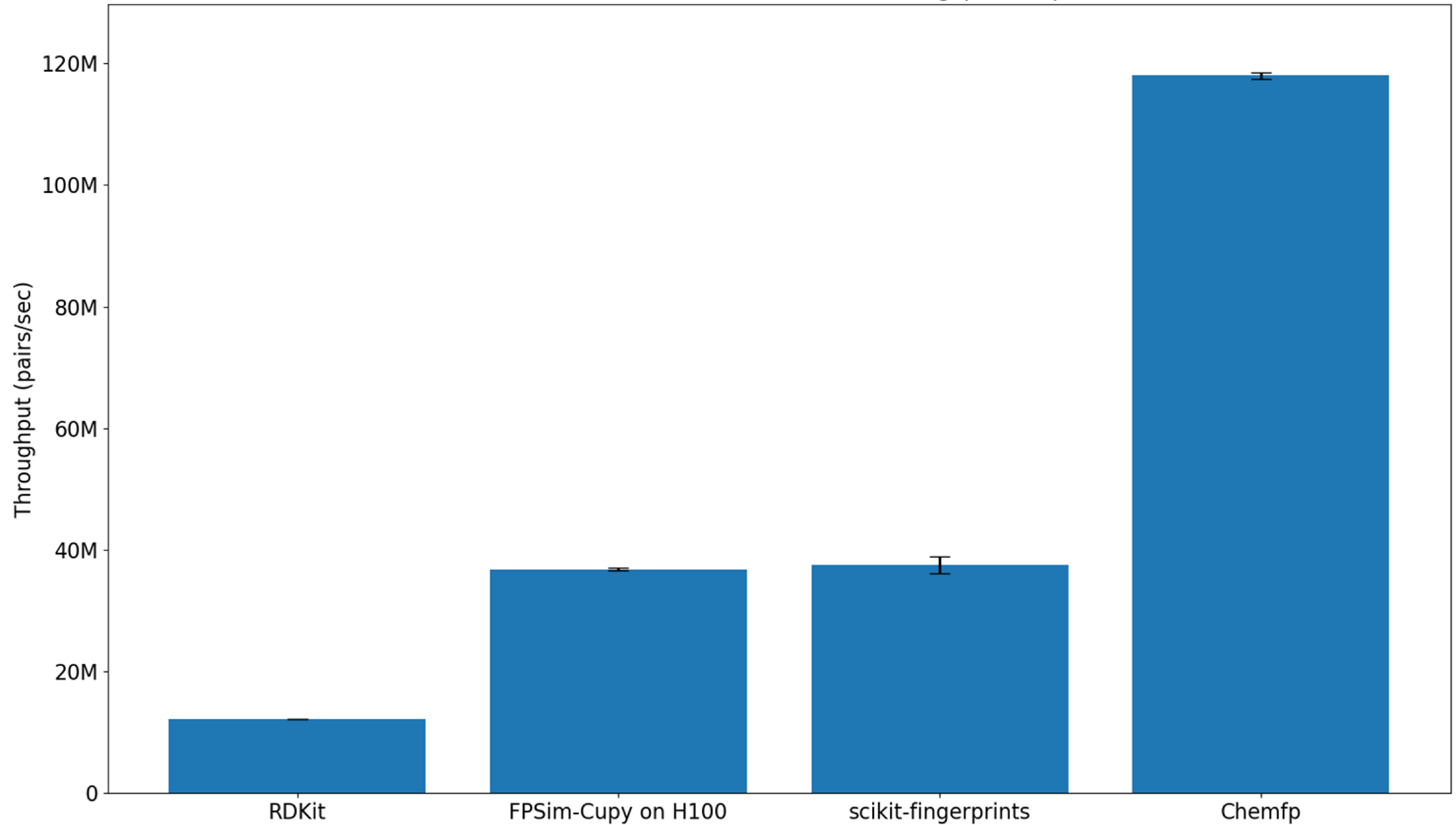
Batch N X M Similarity on the GPU

- Targeting the use case of computing similarity matrices
 - N queries against a database of M molecules
 - N * N self-similarity matrix for clustering
- Use memory reuse techniques common in GPU matrix multiply code
- Use specialized tensor core instructions that do bitwise AND/ORs with popcount accumulates, written in PTX assembly directly

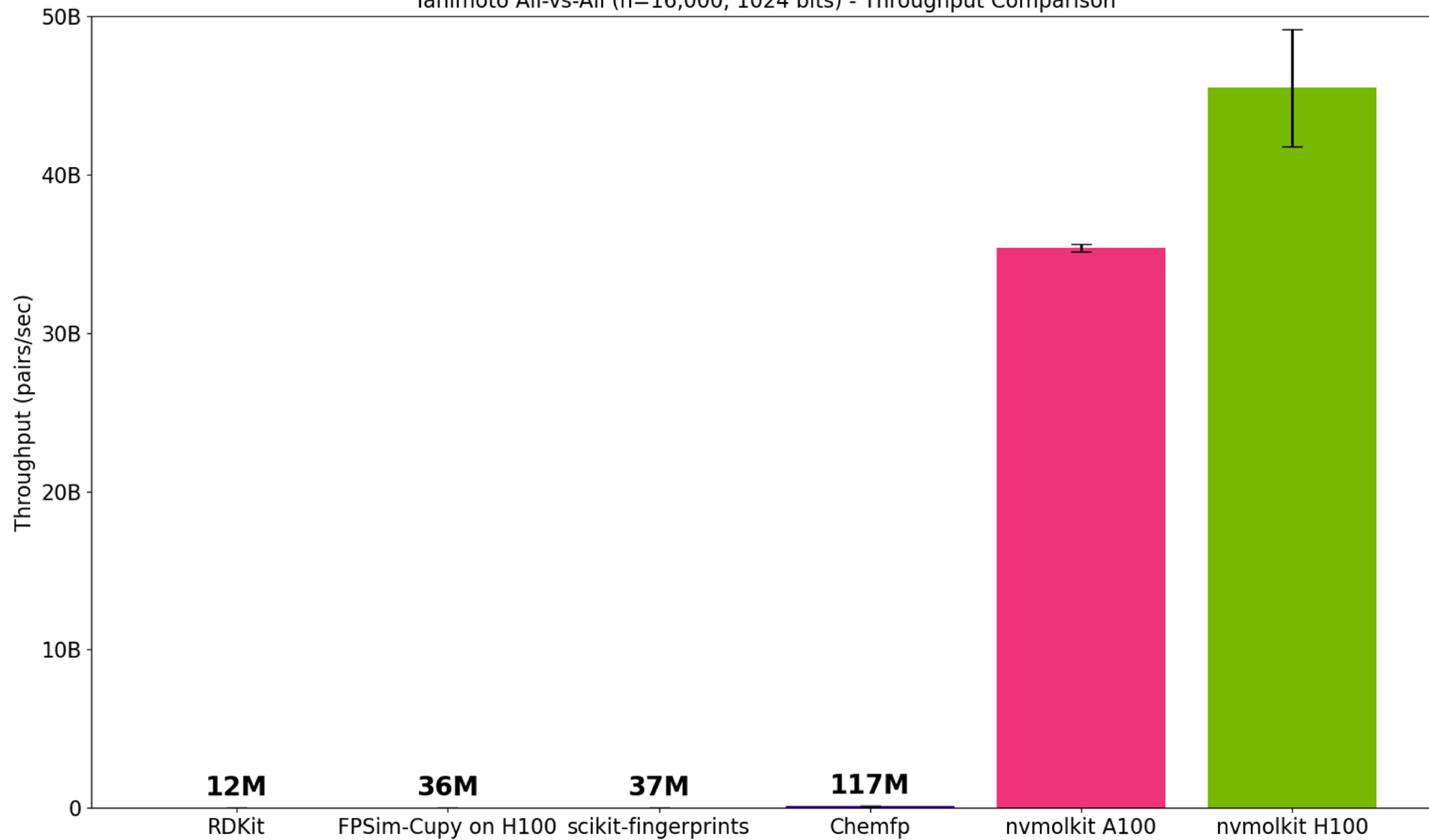
```
.reg .b32 %Ra<4>, %Rb<2>, %Rc<4>, %Rd<4>;  
mma.sync.aligned.m16n8k256.row.col.s32.b1.b1.s32.and.popc  
    {%Rd0, %Rd1, %Rd2, %Rd3},  
    {%Ra0, %Ra1, %Ra2, %Ra3},  
    {%Rb0, %Rb1},  
    {%Rc0, %Rc1, %Rc2, %Rc3};
```



Tanimoto All-vs-All (n=16,000, 1024 bits) - Throughput Comparison

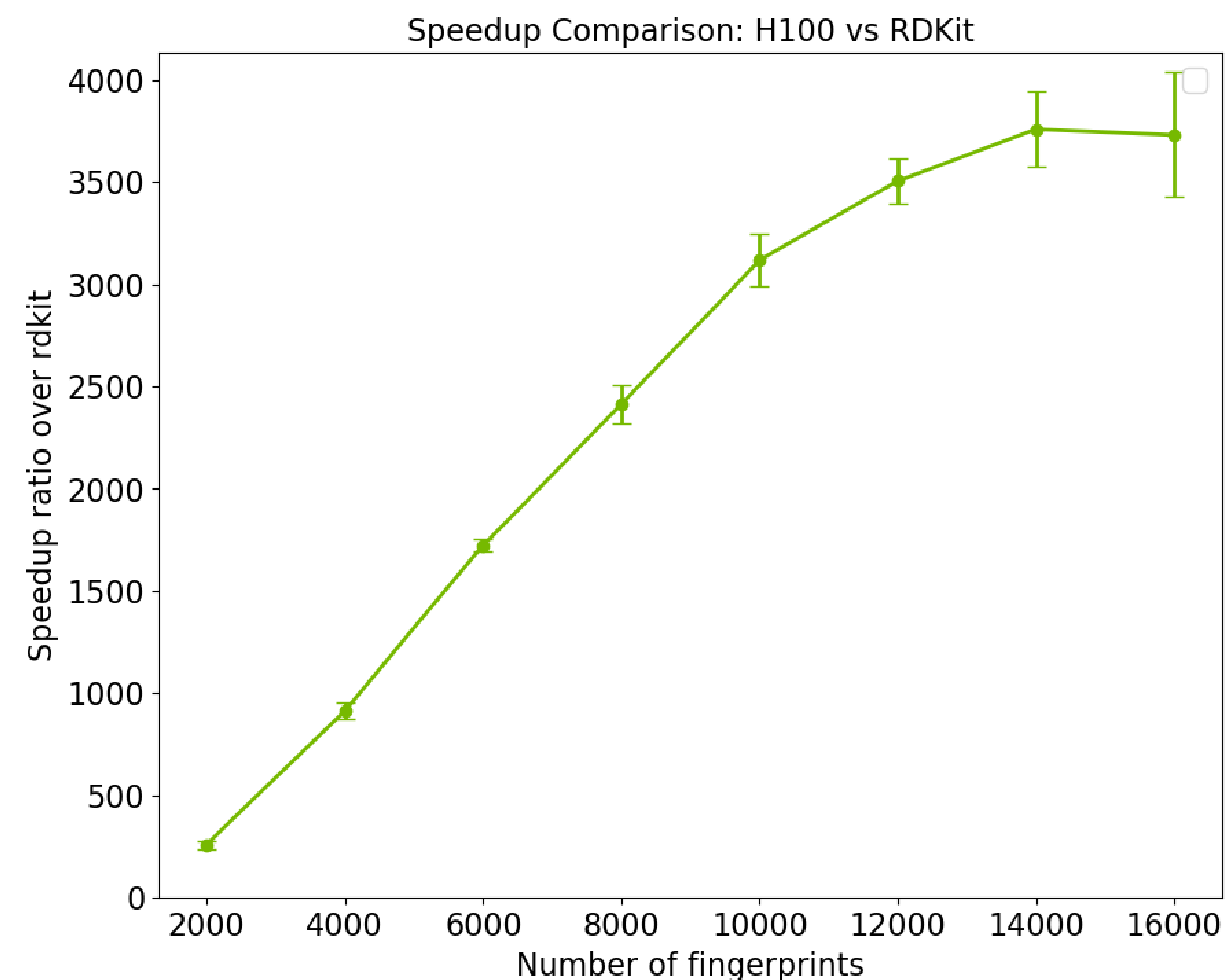


Tanimoto All-vs-All (n=16,000, 1024 bits) - Throughput Comparison



Similarity Notes and Caveats

- ChemFP benchmark is for free public version
- nvMolKit benchmarks are for GPU-resident data/outputs
- Need a lot of compute to get to maximum throughput
- An 80GB RAM GPU will max out at ~22K * 22K matrix, we haven't written code to segment the compute at this time
- Have not implemented some database-specific optimizations that other packages have
- Single GPU for now





MMFF relaxation

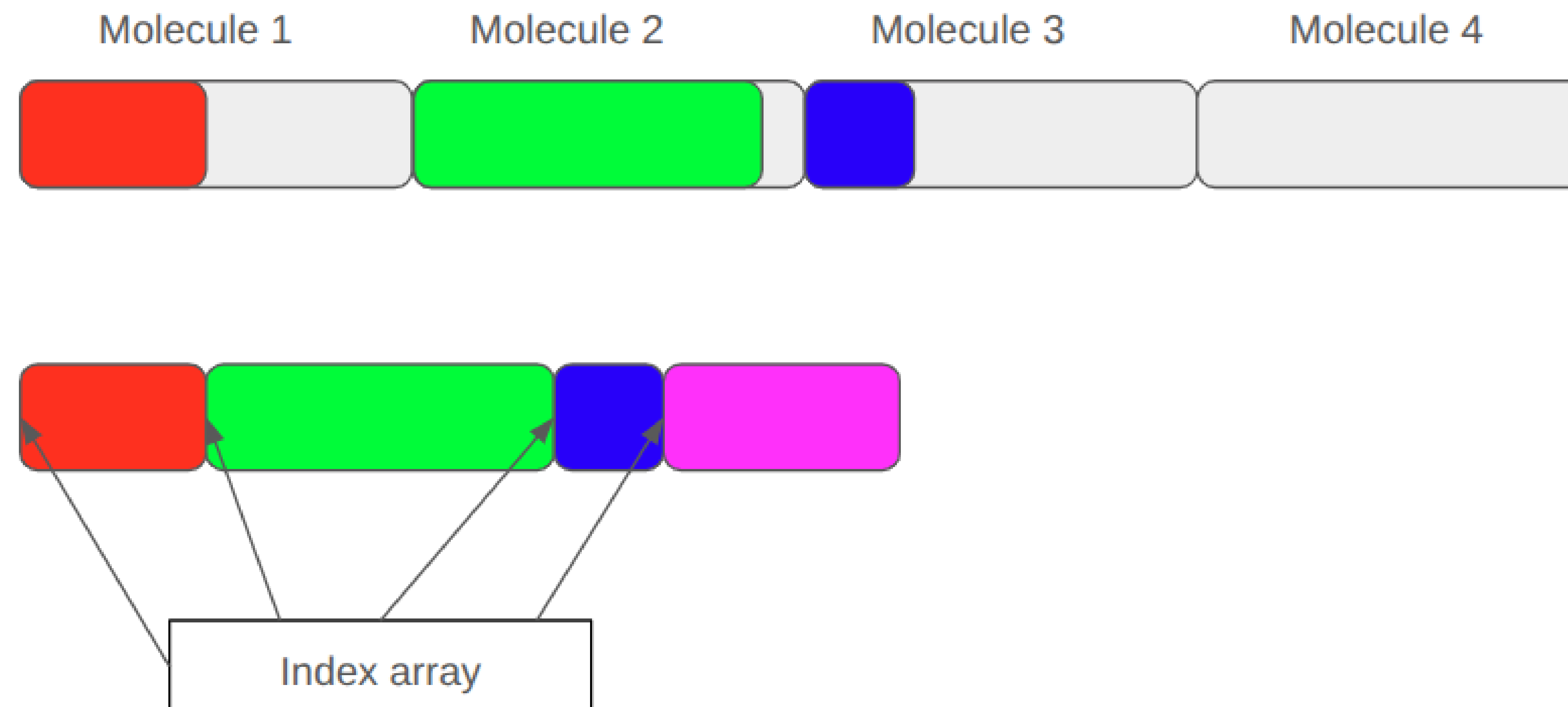
MMFF Relaxation

- Used to minimize conformers
- Typical forcefield with bond, angle, nonbonded terms
- Minimization is driven by RDKit's double-precision BFGS implementation

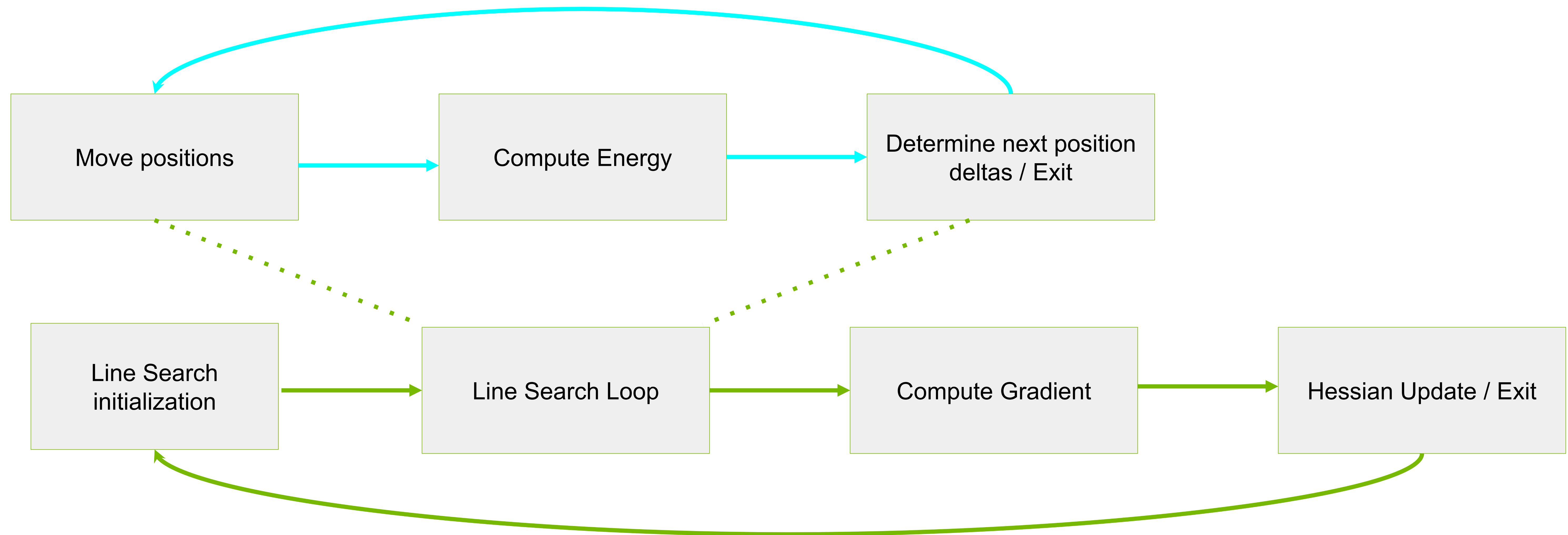
$$E_{\text{MMFF}} = \sum EB_{ij} + \sum EA_{ijk} + \sum EBA_{ijk} + \sum EOOP_{ijk;l} + \sum ET_{ijkl} + \sum E_{\text{vdW}}_{ij} + \sum EQ_{ij}$$

Our approach - batching is critical

- Molecules are typically too small to saturate GPUs
- Support multiple conformers per molecule and multiple molecules
- Need ~hundreds to saturate the GPU depending on size
- With variable sized molecules, use packed coordinates with indexing arrays (CSR-like)
- Similarly pack all metadata and forcefield parameters



Minimizer details and challenges



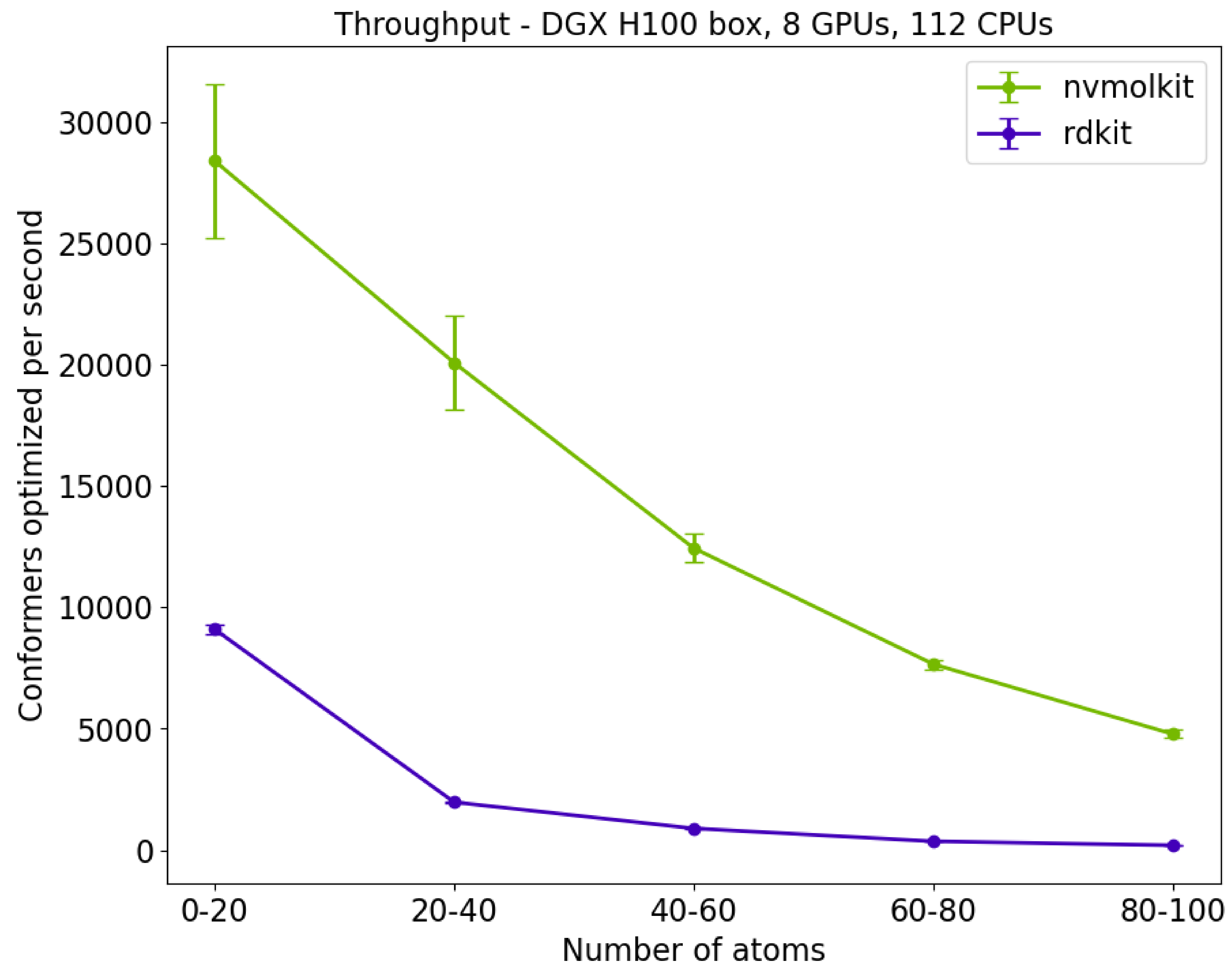
- Loops continue until all batch elements are finished
- Each loop exit check is an expensive CPU synchronization

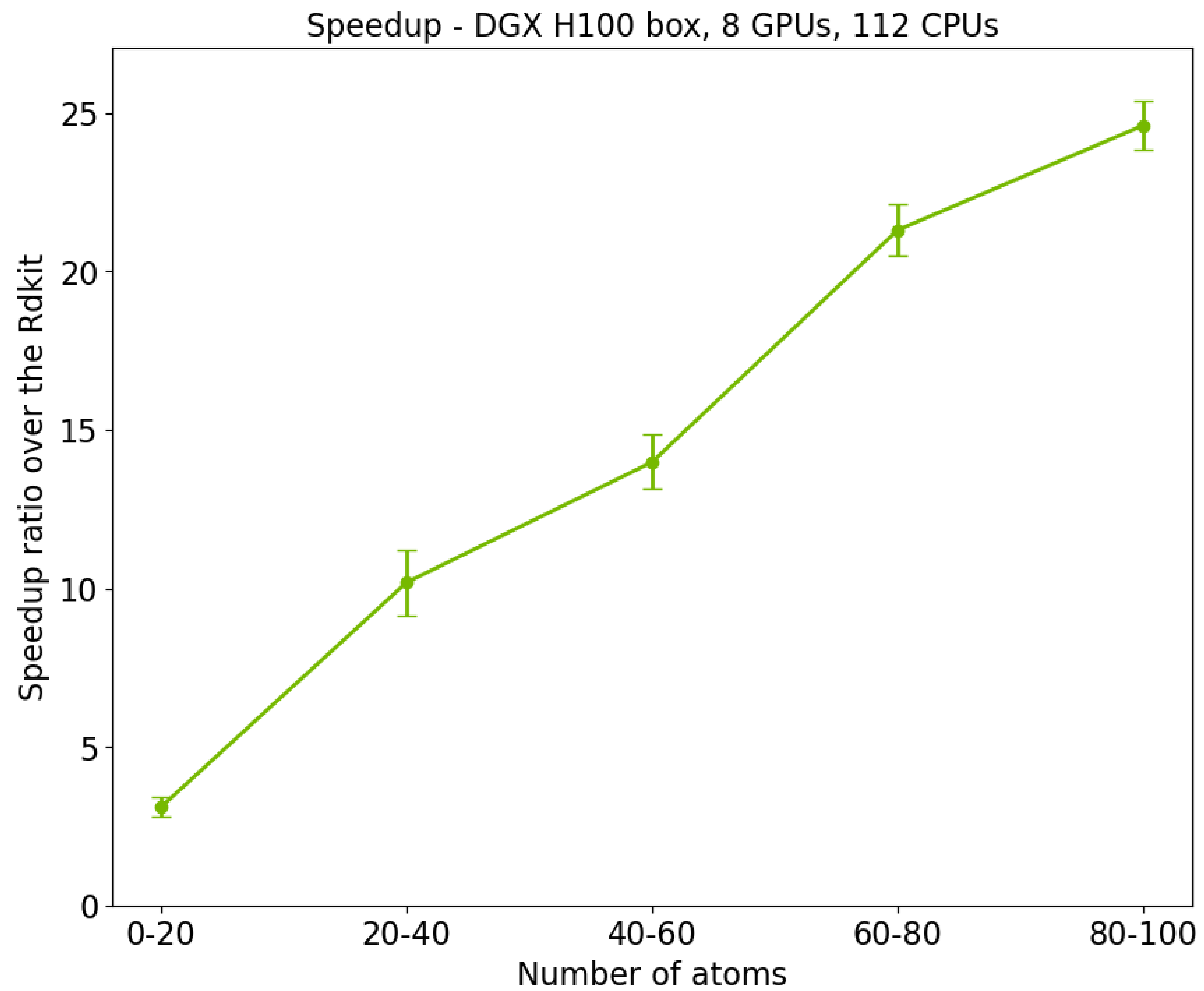
MMFF Minimization Implementation Details

- Mini-batching to reduce “done” molecules waiting for the rest of the batch
- Have multiple threads with multiple CUDA streams targeting the same GPU, running multiple batches at once
 - Hides latency of constant CPU syncing to check for loop iteration exit condition
- Prune out finished molecules each iteration and only dispatch kernels for unfinished molecules
- Hessian update is the most expensive operation. We’ve limited the initial implementation to 256 atoms per molecule to use shared memory and avoid global memory reads/writes

Multi-GPU benchmarks

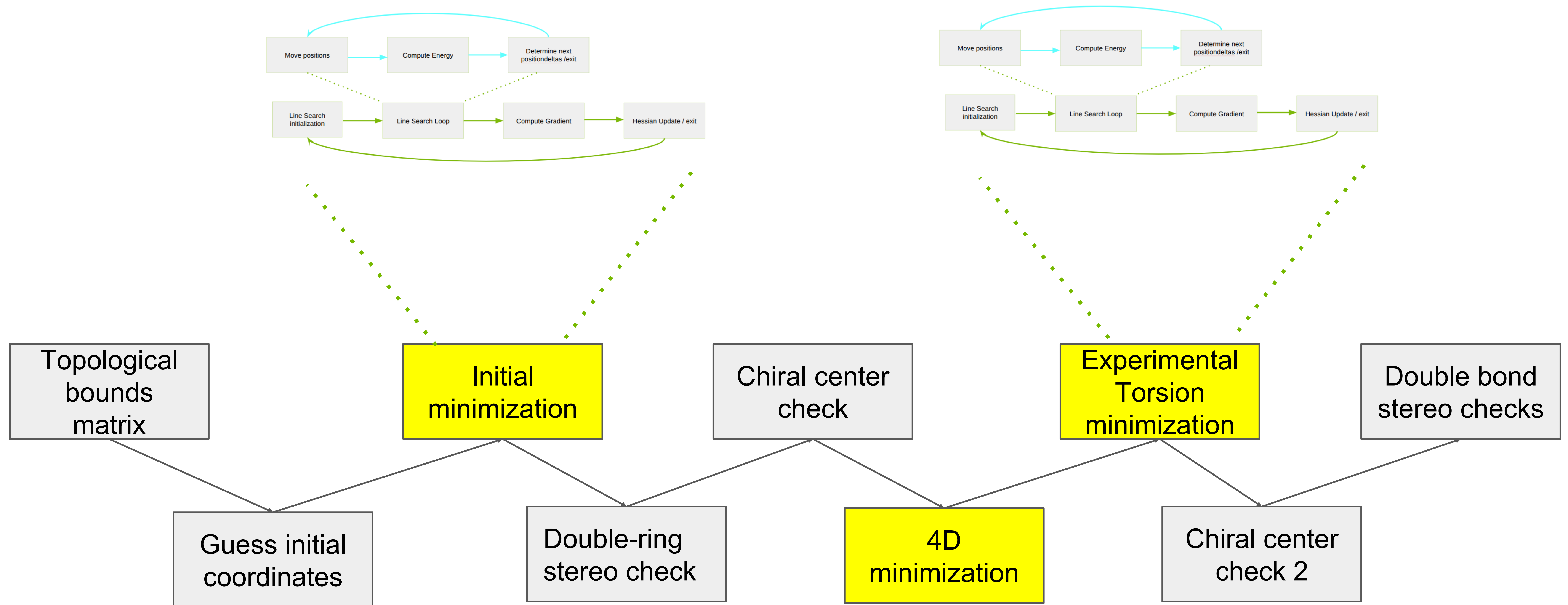
- nvMolKit uses CPUs for preprocessing and dispatching multiple streams to the same GPU
- Hard to find a perfect way to compare RDKit multithreaded performance with nvMolKits.
- The following benchmarks are for an H100 DGX system with
 - 8 H100 GPUs
 - 112 CPU cores
- Comparisons are made between RDKit multithreaded using all CPU cores, and nvMolKit using all CPU and GPU resources





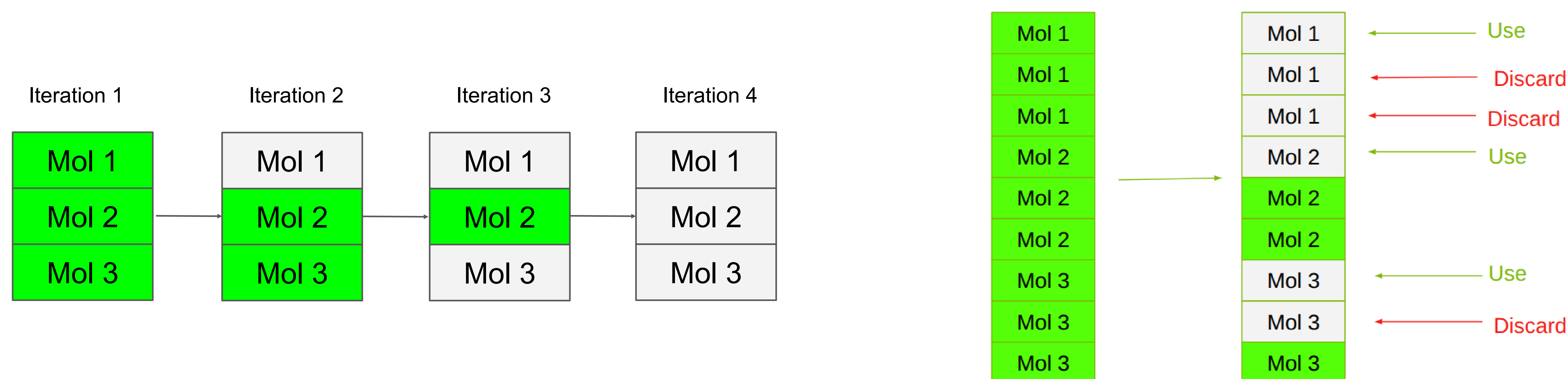
ETKDG Conformer Generation

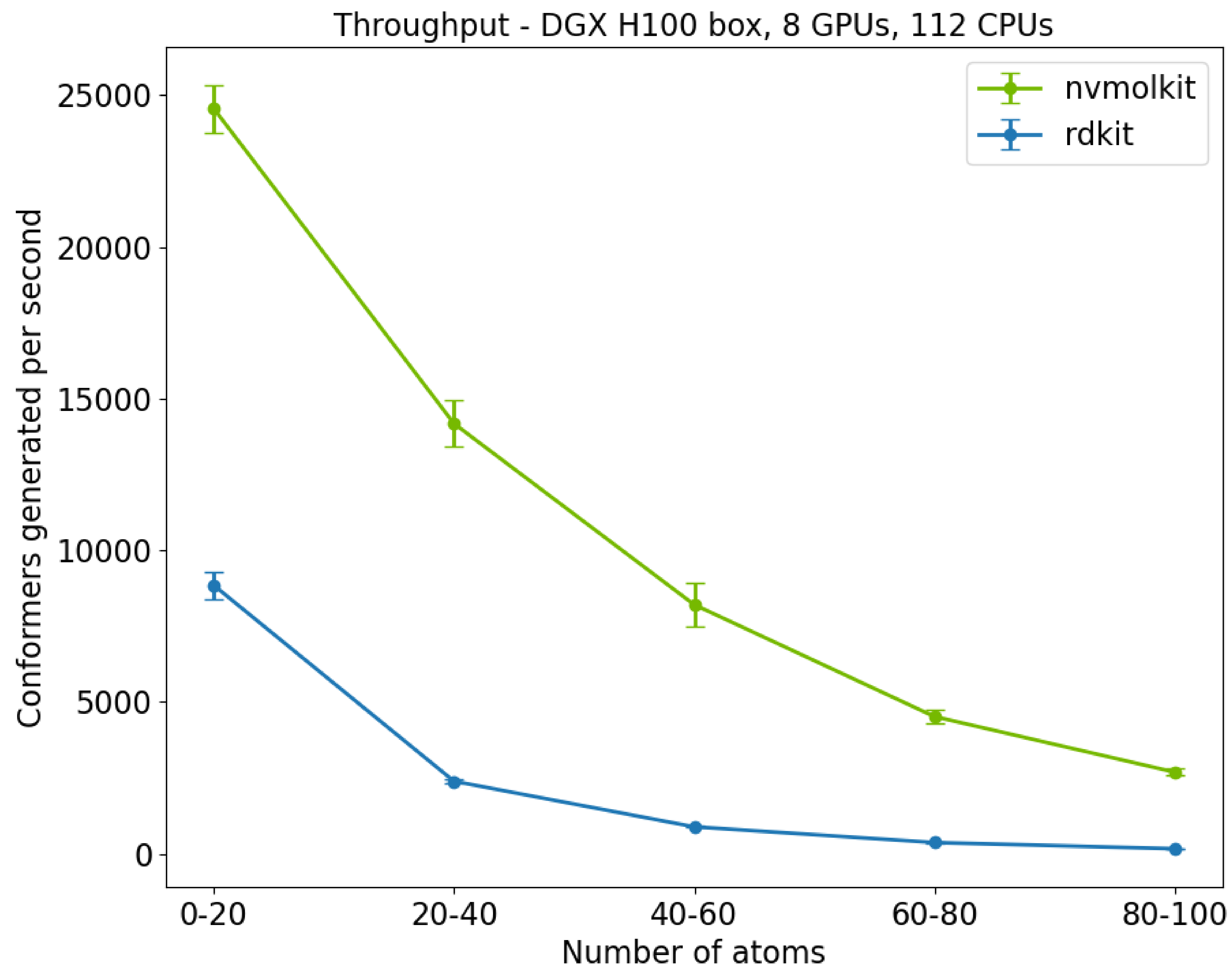
ETKDG details / outline



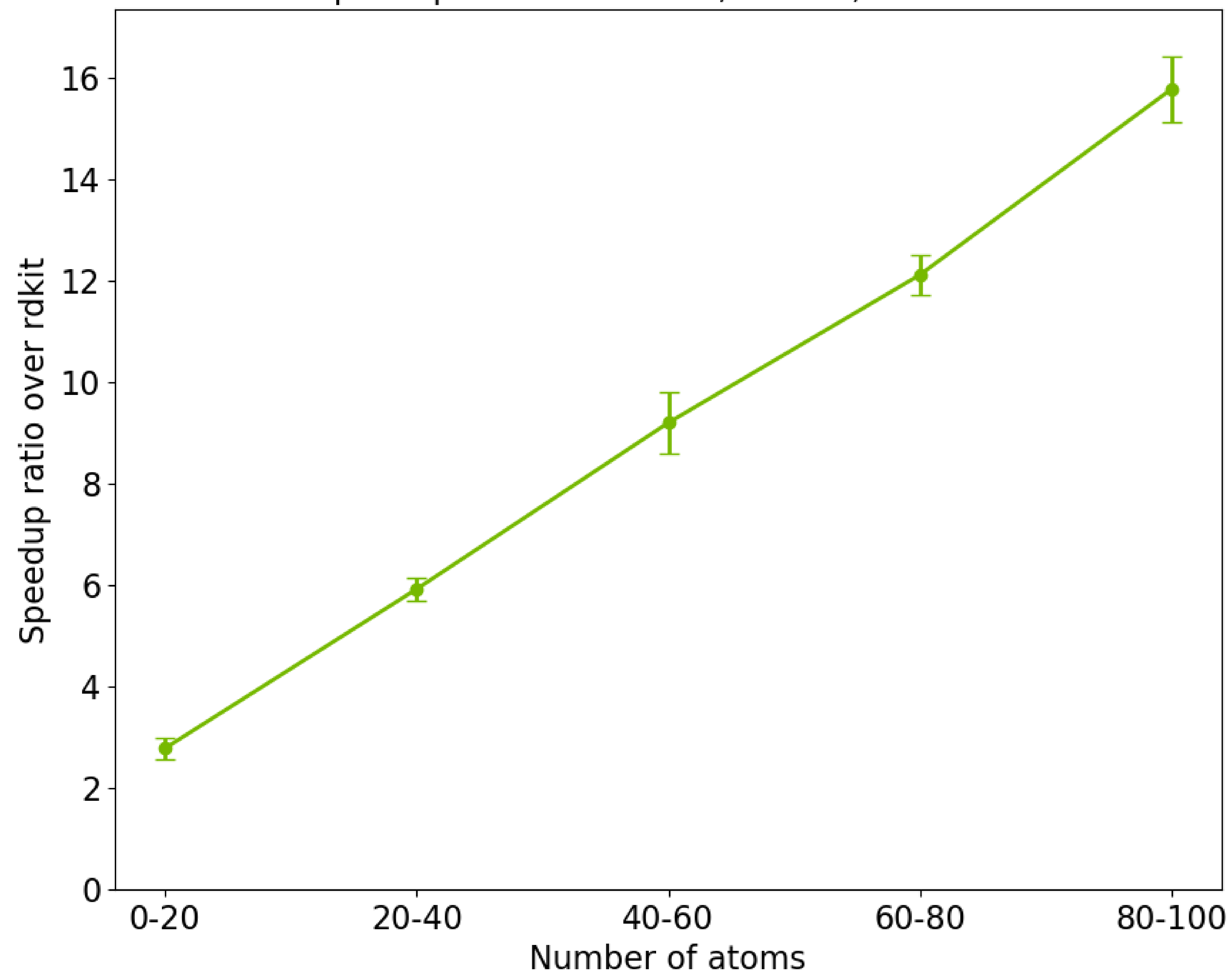
ETKDG challenges and opportunities

- We're now tracking a lot of statuses for each molecule in the batch:
 - Is the molecule active within the iteration?
 - If so, is it active within the minimization loop?
 - If so, is it active within the line search loop?
- Most molecules succeed in a few iterations. But the default max iterations for RDKit is hundreds of iterations
 - Very strong tail effect
 - Difficult for GPU to be efficient
- An optimization opportunity - unrolling the iteration loop and oversubscribing
 - Help saturate GPU for low workloads
 - Dynamically retarget problematic molecules once normal cases have finished





Speedup - DGX H100 box, 8 GPUs, 112 CPUs



ETKDG notes and caveats

- Only a few nonstandard options are supported. ETKDGv3() default params work with one exception
- Random coordinate generation only (useRandomCoords=True)
- 256 Atom size limit
- No support for conformer deduplication (yet)
- No support for coordMap (yet)



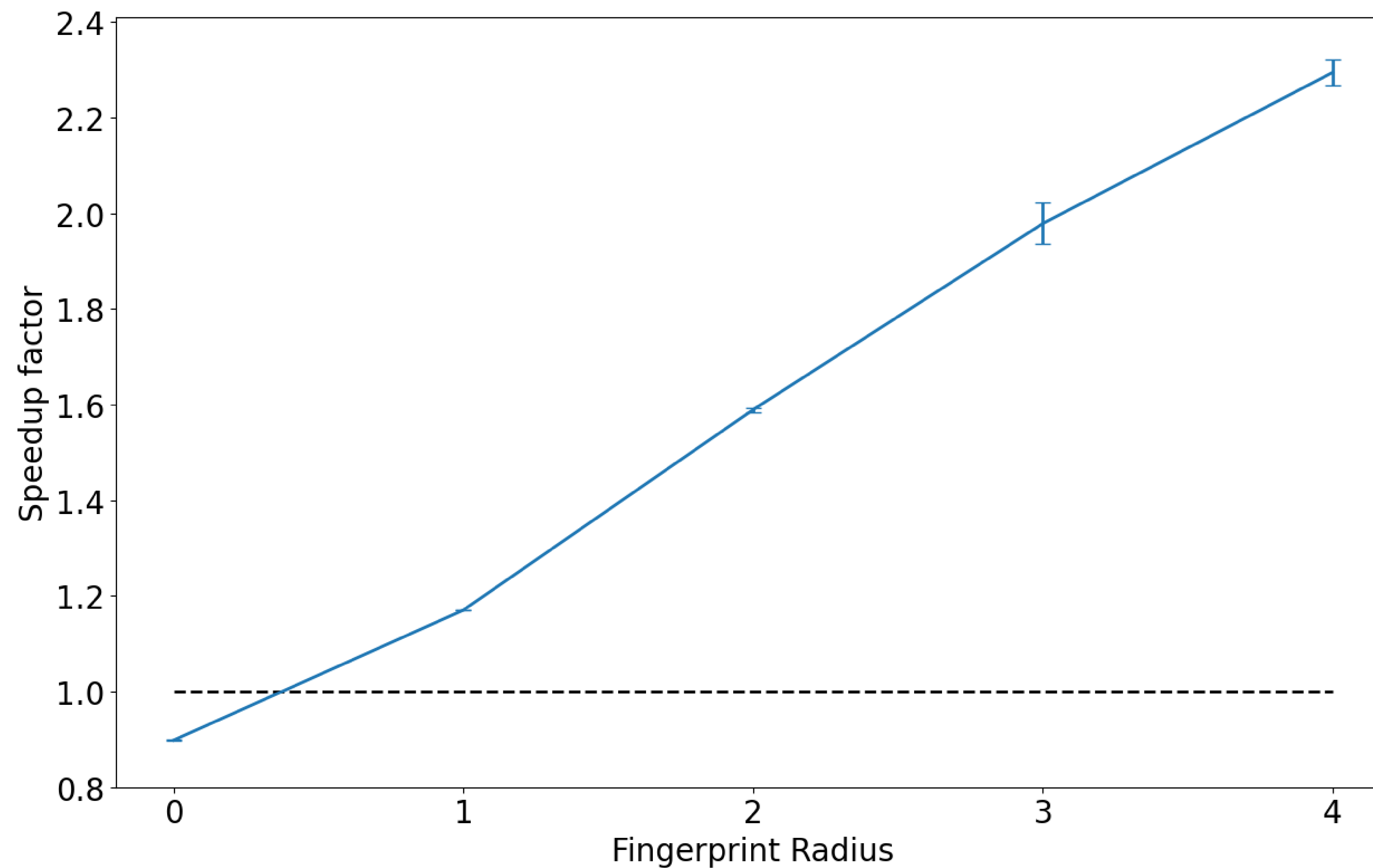
Coming Next

- Release 0.1.0 is out!
- Please try it out, find bugs, and request features!
- Our current priority list
 - Multi-GPU fingerprinting + Similarity
 - Conformer deduplication
 - Support other combinations of (ET)(K)(DG)
 - Removing size limits on minimizer + ETKDG
 - GPU Butina Clustering
 - conda-forge builds

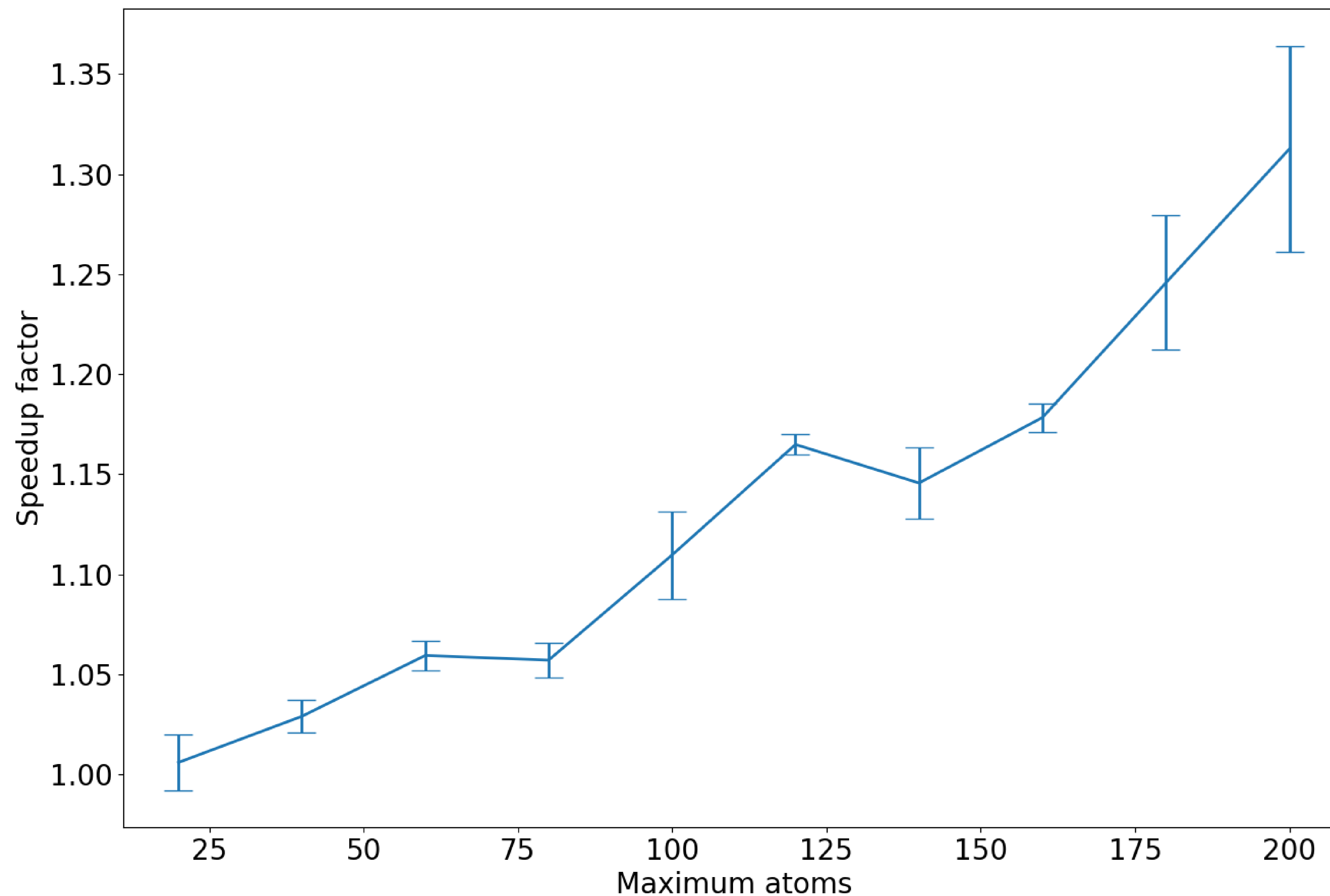


CPU speedups landed in the RDKit

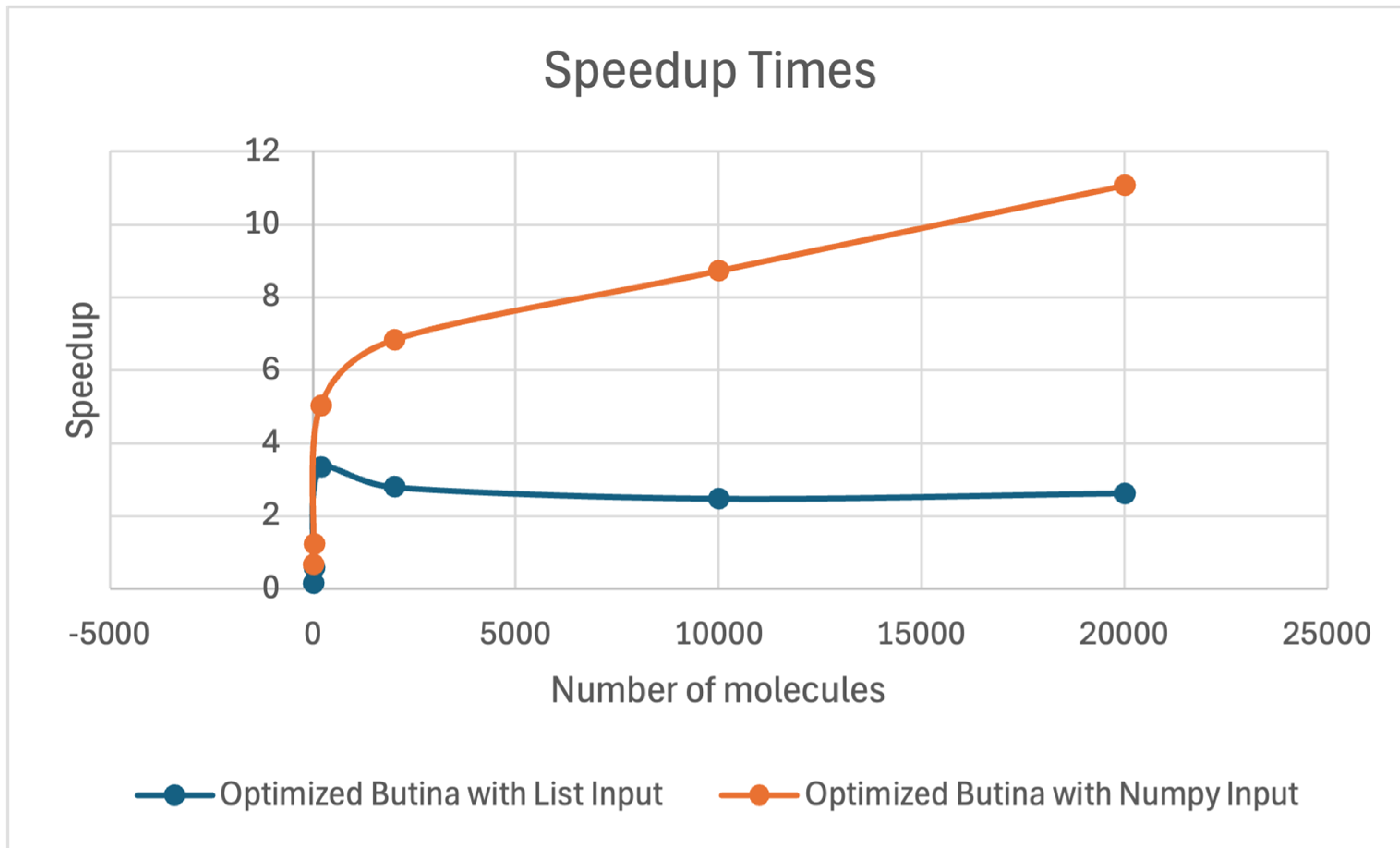
Morgan Fingerprint Improvements (RDKit 2025.3, PR ##7862)



Large molecule SMILES parse improvements (RDKit 2025.3, Issue #7888)



Butina Clustering Performance Improvements (PR #7892)





CPU Core Molecule Representation changes

Current Data Structure

What does an ROMol look like? (example: sucrose with 23 atoms & 24 bonds)

Each atom has:

- 1 Atom allocation
- 1 vector of edge list iterators and other atom indices
- 1 vector of name-value pairs:
 - 2-4 name strings (4 total unique)
 - 1-2 values that are separately allocated strings (3 total unique)

Each bond has:

- 1 Bond allocation
- 1 edge list node
- 1 vector of name-value pairs (no pairs, but still allocated space for 1-2 pairs)

Other data:

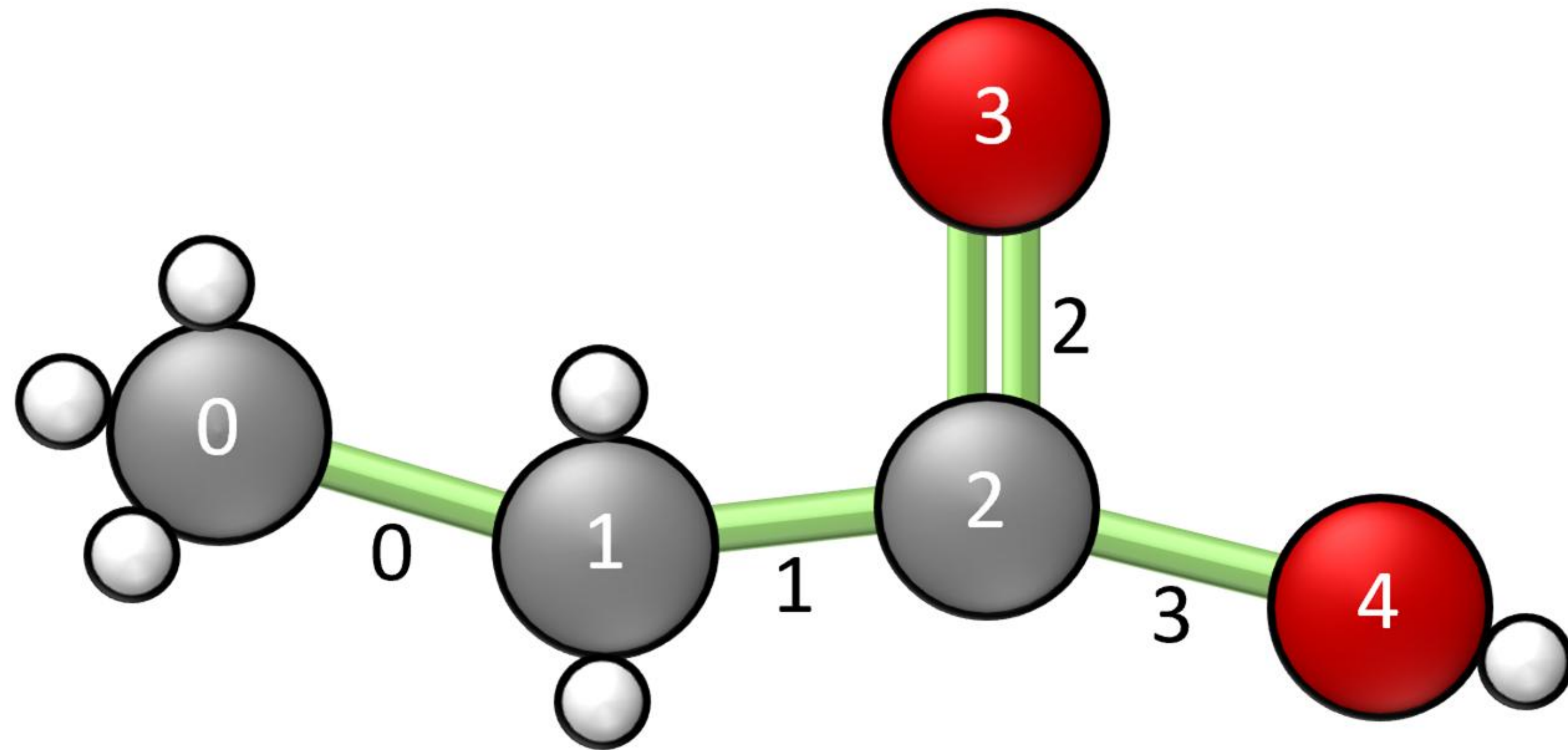
- 1 RingInfo with:
 - 1 vector containing 11 int vectors for rings of atoms
 - 1 vector containing 11 int vectors for rings of bonds
 - 1 vector containing 3 int vectors for atoms of rings
 - 1 vector containing 3 int vectors for bonds of rings
- 1 vector of name-value pairs:
 - 2 name strings
 - 1 value that is a separately allocated string
- 1 allocation of a single byte

Total for 1 RWMol
representing sucrose:

- 317 allocations
- At least 18.7KB

Compressed Sparse Row Format

An efficient way of representing the connectivity of a sparse graph



Dense Matrix

	0	1	2	3	4
0		0			
1	0		1		
2		1		2	3
3			2		
4			3		

Compressed Sparse Row (CSR)

	0	1	2	3	4	5		
row offsets	0	1	3	6	7	8		
	↓	↓	↘	↘	↘	↘		
	0	1	2	3	4	5	6	7
column indices	1	0	2	1	3	4	2	2
	0	1	2	3	4	5	6	7
values (bonds)	0	0	1	1	2	3	2	3

Current work in draft PR on RDKit

- [PR 8591](#)
- Condensed representations for:
 - Atom/Bond data
 - Properties
 - Conformers
 - Queries
- WIP - some cleanup left to do and a few remaining test failures
- Please check it out!

Thank you!

nvMolKit Links

Source Code: <https://github.com/NVIDIA-Digital-Bio/nvMolKit>

Documentation:
<https://nvidia-digital-bio.github.io/nvMolKit>

Blog Post:
<https://research.nvidia.com/labs/dbr/blog/nvMolKit/>

BioNemo Links

BioNemo Framework: <https://github.com/NVIDIA/bionemo-framework>

BioNemo NIMs: <https://www.nvidia.com/en-us/clara/biopharma/>

cuEquivariance: <https://github.com/NVIDIA/cuEquivariance>

cuik-molmaker: <https://github.com/NVIDIA-Digital-Bio/cuik-molmaker>

Research publications

- DualBind/ToxBench: <https://arxiv.org/abs/2507.08966>
- Megalodon: <https://arxiv.org/abs/2505.18392>
- LoQI: <https://chemrxiv.org/engage/chemrxiv/article-details/689f756ea94eede154ec6217>
- MolMIM: <https://arxiv.org/abs/2208.09016>



Appendix: Morgan Fingerprints

Throughput of 1024 bit Morgan fingerprints

