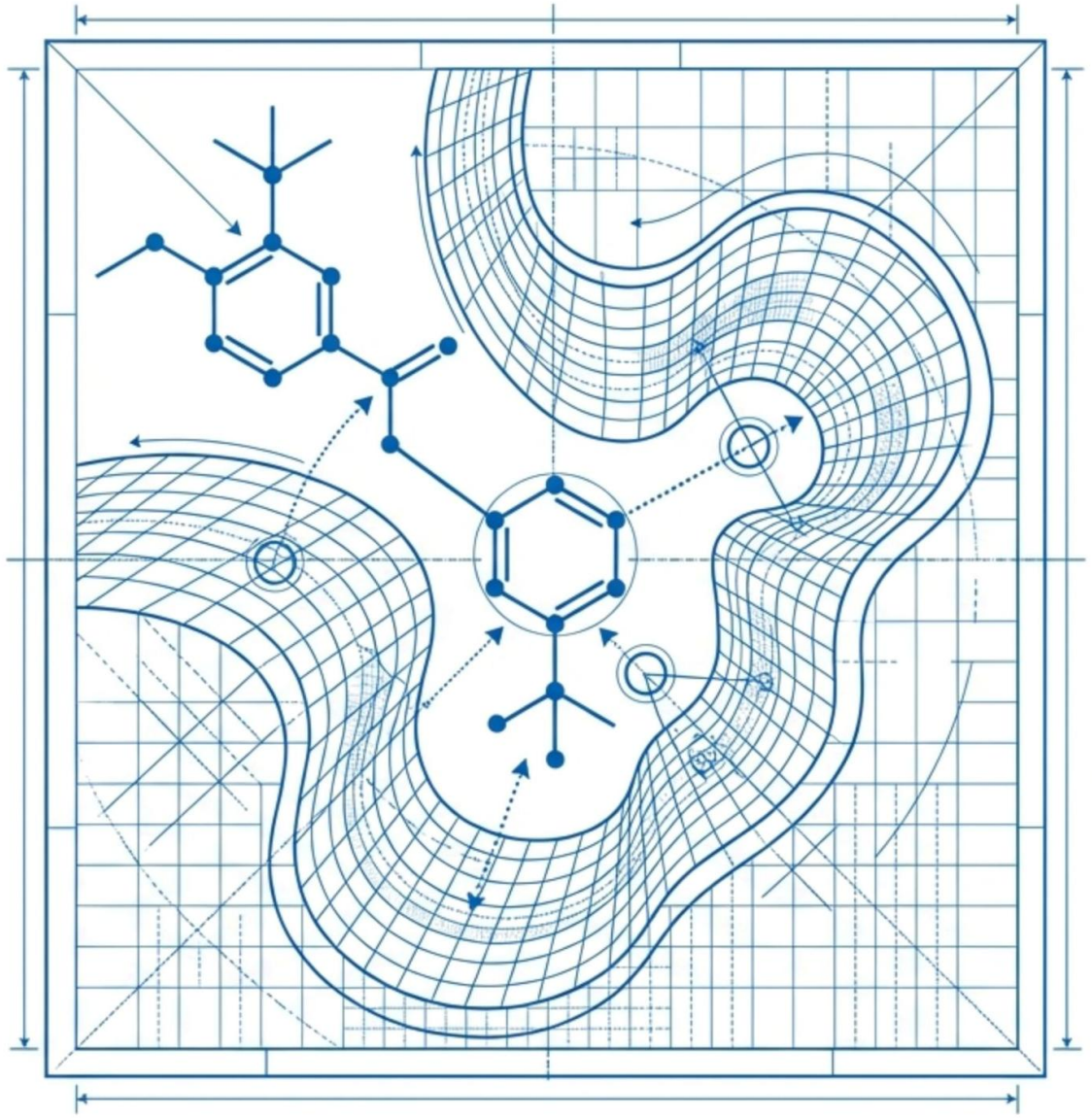
DOCK 3.8

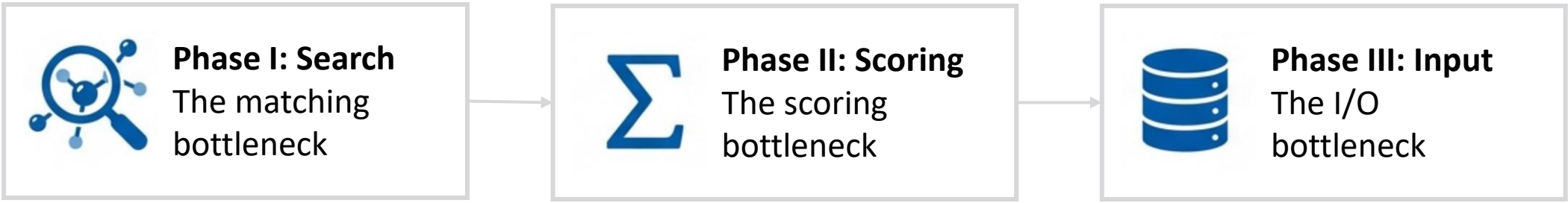
Review and optimization implementations

Algorithmic optimizations delivering >7x speedup with exact accuracy.

Phong Lam,
Carlsson Lab, Uppsala University



Identifying the Computational Bottlenecks Bottlenecks



Partly inspired from the Huang's group SWDOCK based on DOCK3.7

Matching optimization

Aims:

Propose a list of atom-pairs that satisfy, until enough matches.

$$|d_{ik} - d_{IK}| < \sigma,$$

For example:

Lig_spheres: [1, 2, 4, 5, 7]

Prot_spheres: [20, 21, 25, 29, 30]

→Combination of 4-points:

[1-20; 2-21; 4-25; 5-29];

[1-20; 2-21; 4-25; 7-30];

...

Optimizations:

- Reduce redundant matches
- Degeneracy check optimization

Outline:

```
search.f
└── fullmatch.f                [Called per (ligand_seed, receptor_seed) pair]
    ├── sphsfromdists.f        [Finds distance-compatible sphere pairs]
    │                           [Optimized: uses canonical ordering]
    │
    └── combinematches.f       [Generates all valid k-tuples of sphere pairs]
        ├── otherdistcheck.f   [Validates pairwise distances]
        └── degeneracycheck.f  [Removes duplicate orientations]
            └── set_rotation   [Computes rotation matrix]
```

Matching optimization

1. Canonical ordering

```fortran
do while (more_matches)
  do ligstart = 1, tcount
    do recstart = 1, spheres0%nsphr
      call fullmatch(...)
    enddo
  enddo
enddo
```

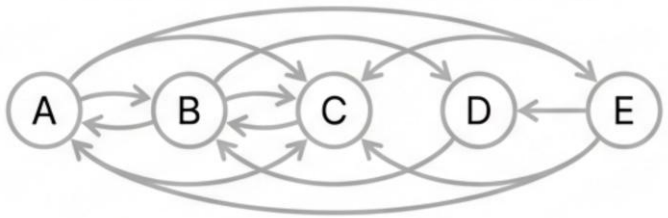
Original

```
do curligpt = 1, maxlig                    ! O(L) - ALL ligand spheres
  if (curligpt .ne. ligstart) then
    ligdist = disl(ligstart, curligpt)
    do currecpt = 1, maxrec                ! O(R)
      if (currecpt .ne. recstart) then
        distdiff = abs(disr(recstart, currecpt) - ligdist)
        if (distdiff .le. disttol) then
          ! Add to sphmatches
        endif
      endif
    enddo
  endif
enddo
```

Optimized

```
do curligpt = ligstart + 1, maxlig   ! ← KEY CHANGE: start from ligstart+1
    ligdist = disl(ligstart, curligpt)
    do currecpt = 1, maxrec
      if (currecpt .ne. recstart) then
        distdiff = abs(disr(recstart, currecpt) - ligdist)
        if (distdiff .le. disttol) then
          ! Add to sphmatches
        endif
      endif
    enddo
enddo
```

do curligpt = 1, maxlig



! Searches all other ligand spheres

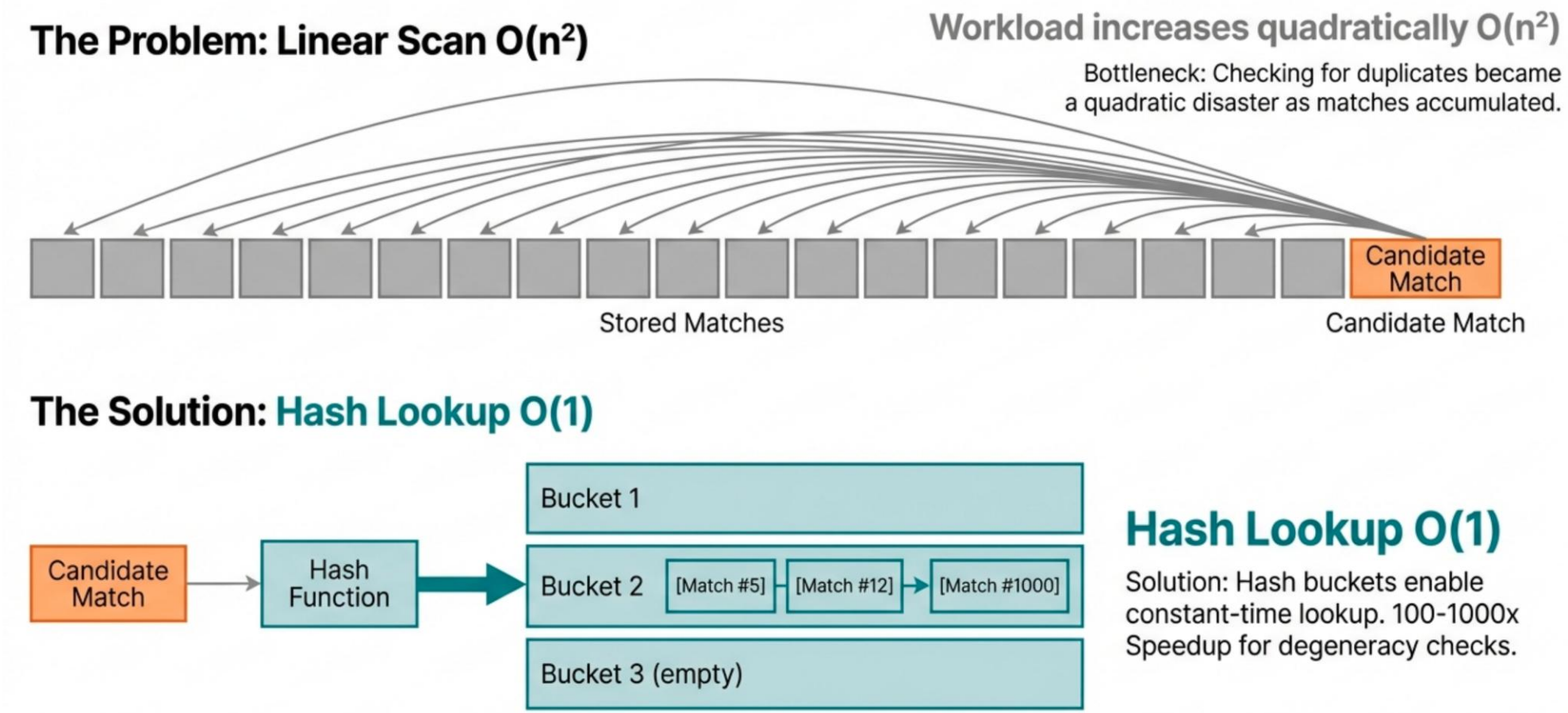
2X Reduction in search space

do curligpt = ligstart + 1, maxlig

! Only searches higher-indexed spheres

Matching optimization

2. Degeneracy check optimization



Matching optimization

2. Degeneracy check optimization

Original **$O(n \times k^2)$**

```fortran
! Original: degeneracycheck.f
do hashindex = 1, hashcount           ! O(n) - scan ALL previous hashes
  all_matched = .true.
  do tmpindex = 1, maxctr             ! O(k) - check each sphere
    match_found = .false.
    do centerindex = 1, maxctr        ! O(k) - compare with stored
      if (tmphash(tmpindex) .eq. hash(hashindex, centerindex)) then
        match_found = .true.
      endif
    enddo
    if (.not. match_found) then
      all_matched = .false.
      exit
    endif
  enddo
  if (all_matched) then
    degeneracycheck = .false.
    return
  endif
enddo
```

$O(n/B \times k^2) \sim O(k^2) = O(1)$ **Optimized**

```fortran
! Optimized: degeneracycheck.f
integer, parameter :: NBUCKETS = 65537   ! Prime number for good distribution
integer, save :: bucket_heads(NBUCKETS)
integer, save :: bucket_next(100000)

! Compute hash value for current match
hash_val = 0
do nodeindex = 1, nodetotal
  tmphash(nodeindex) = sphpairs(nodeindex, 1) +
                       sphpairs(nodeindex, 2) * maxpts
  hash_val = hash_val + tmphash(nodeindex) * (7919 ** nodeindex)
enddo

! Compute bucket index (1 to NBUCKETS)
bucket_idx = mod(abs(hash_val), NBUCKETS) + 1

! Search ONLY within this bucket - O(n/NBUCKETS) average
idx = bucket_heads(bucket_idx)
do while (idx .gt. 0)
  ! Compare with entries in this bucket only
  if (all_matched) then
    degeneracycheck = .false.
    return
  endif
  idx = bucket_next(idx)
enddo

! Not found - add to bucket's linked list head
bucket_next(hashcount) = bucket_heads(bucket_idx)
bucket_heads(bucket_idx) = hashcount
```

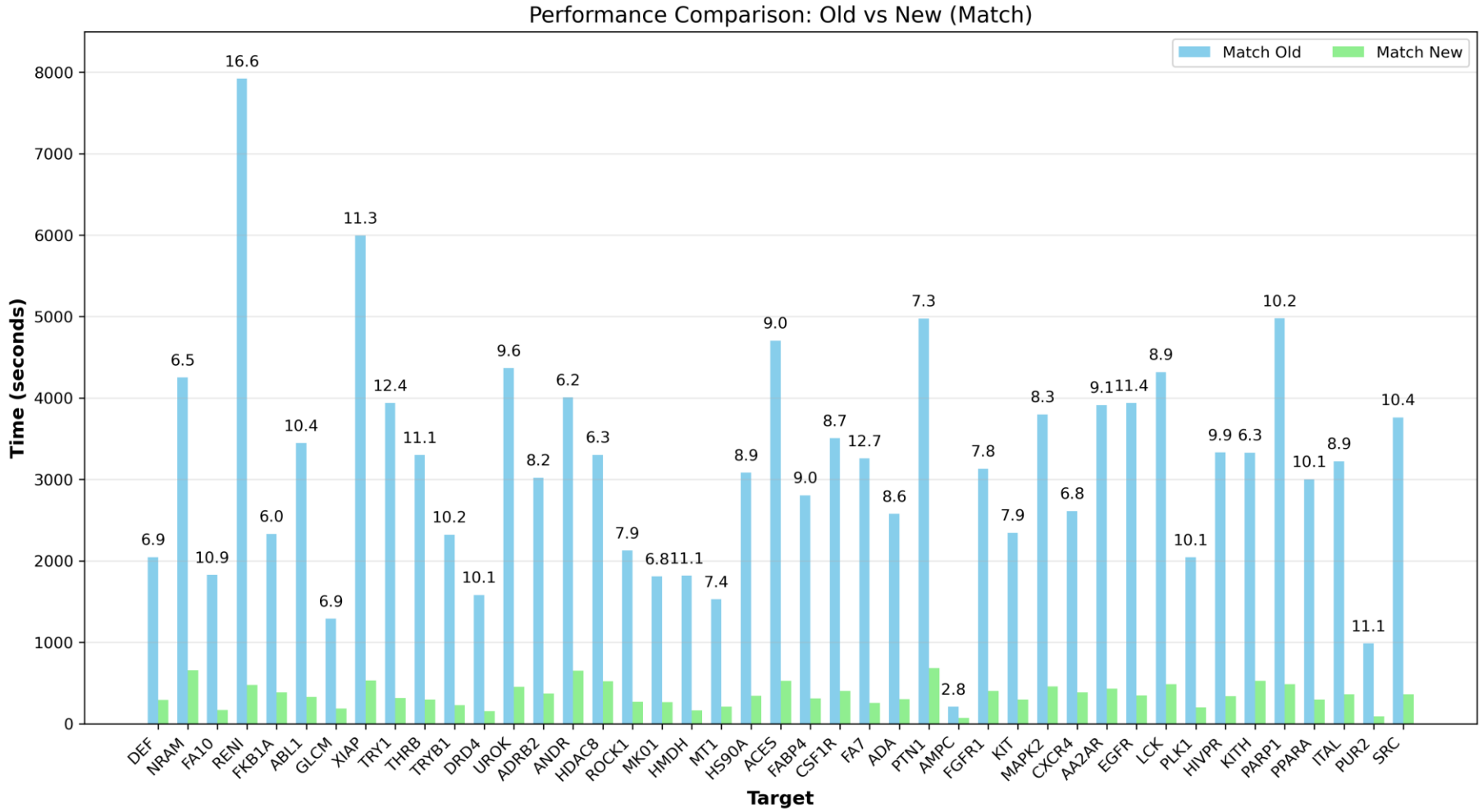
n = number of matches (common 1000)

k: min_node = max_node = 4

Matching optimization

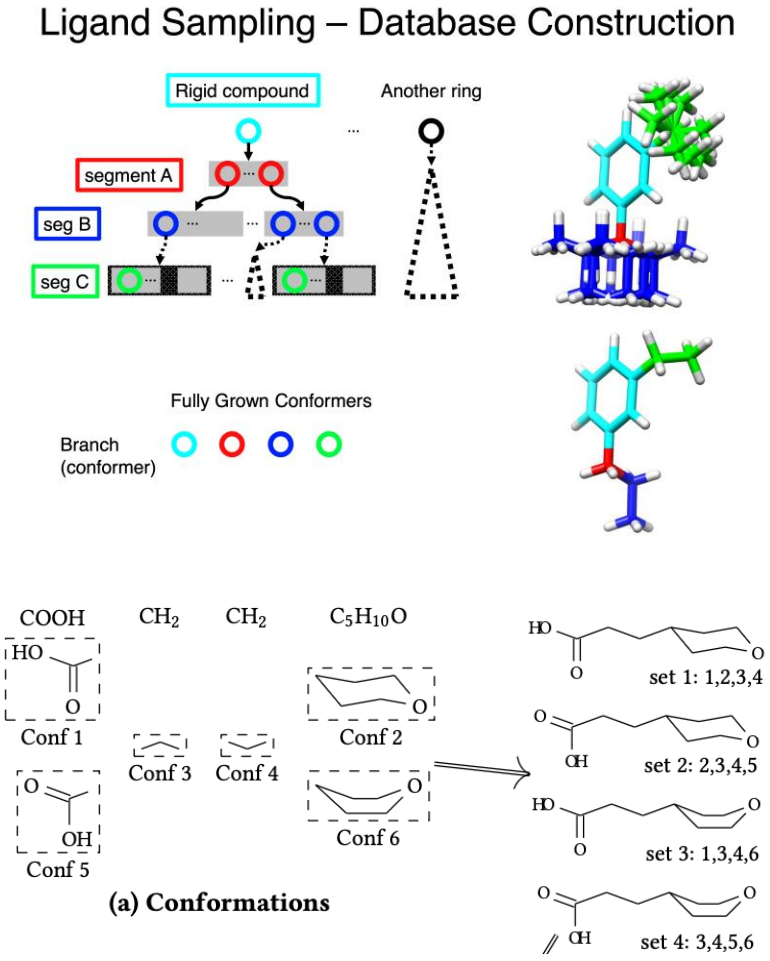
3. Speed-up gain

Average:
8.94X



Scoring optimization

- **DB2 recap:**
  - Rigid: reference part
  - Conf: group of atoms moving together
  - Set (conformer): group of conformers
- **Aims:**
  - Given a match → orient confs
    → $Score_{set} = \sum Score_{confs}$



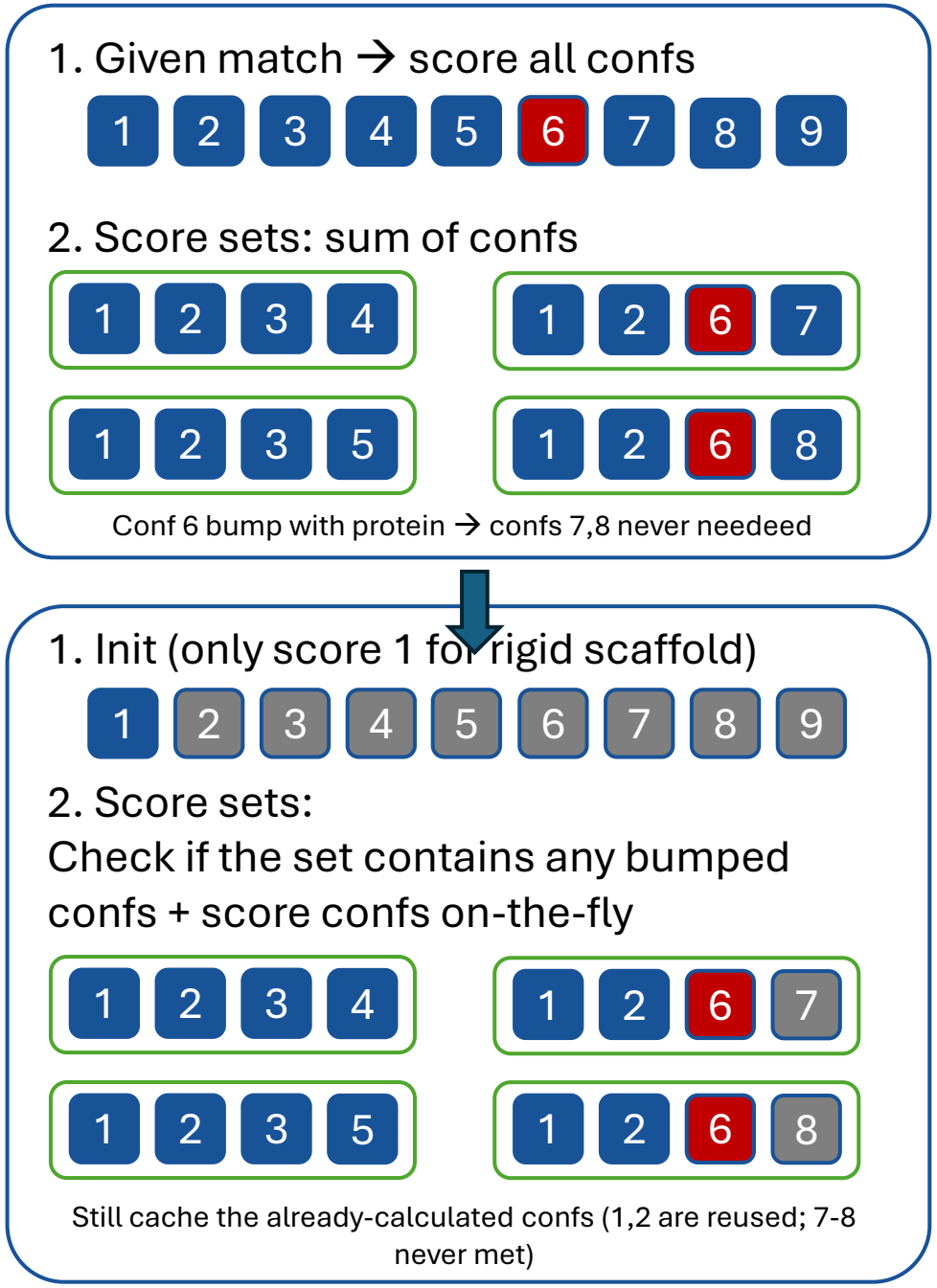
Scoring optimization

1. Lazy evaluation

- Pre-calculate all the confs may not be necessary
- If a set contain a bumped conf, we don't need to calculate the score of it

→ don't need to calculate score of the confs.

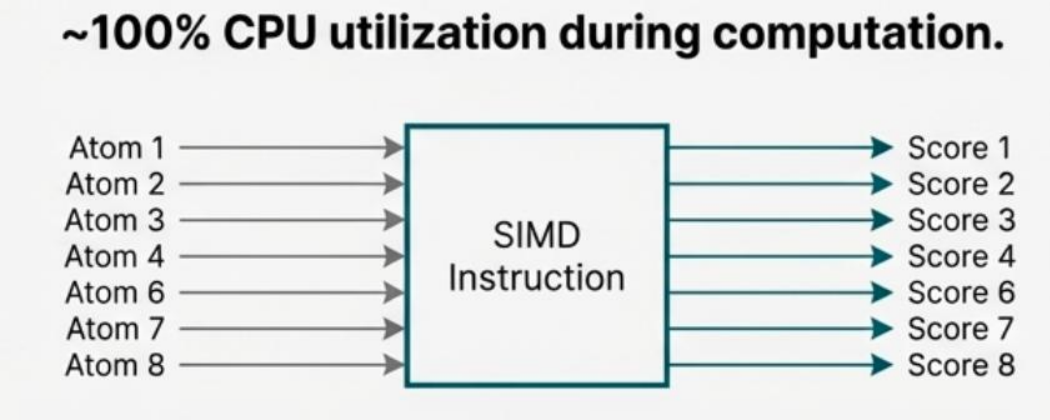
- Originally: only conduct in the "cluster" mode.
- Now: default behavior



Scoring optimization

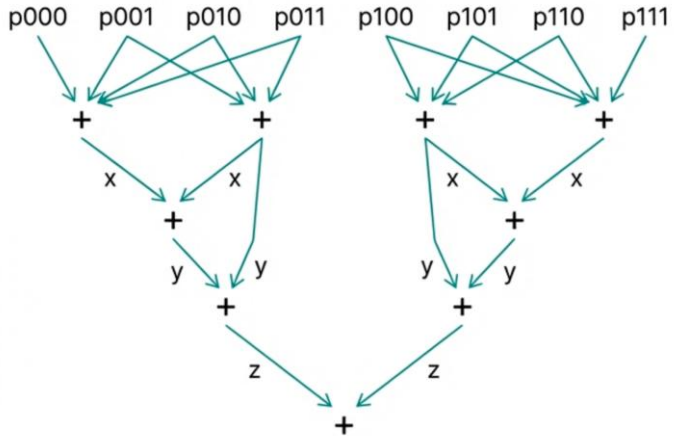
2. SIMD optimization + FMA

- Calculate 8 atoms at a time.
- Replace tri-linear interpolation to FMA format:



3. Early atom bump exit

- Originally: sum of vdW term for a conf >= max_bump
- Now: if any atom has >= max_bump; stop calculation.



Before Formula:

```
apt = a1*x*y*z + a2*x*y + ... + a8
```

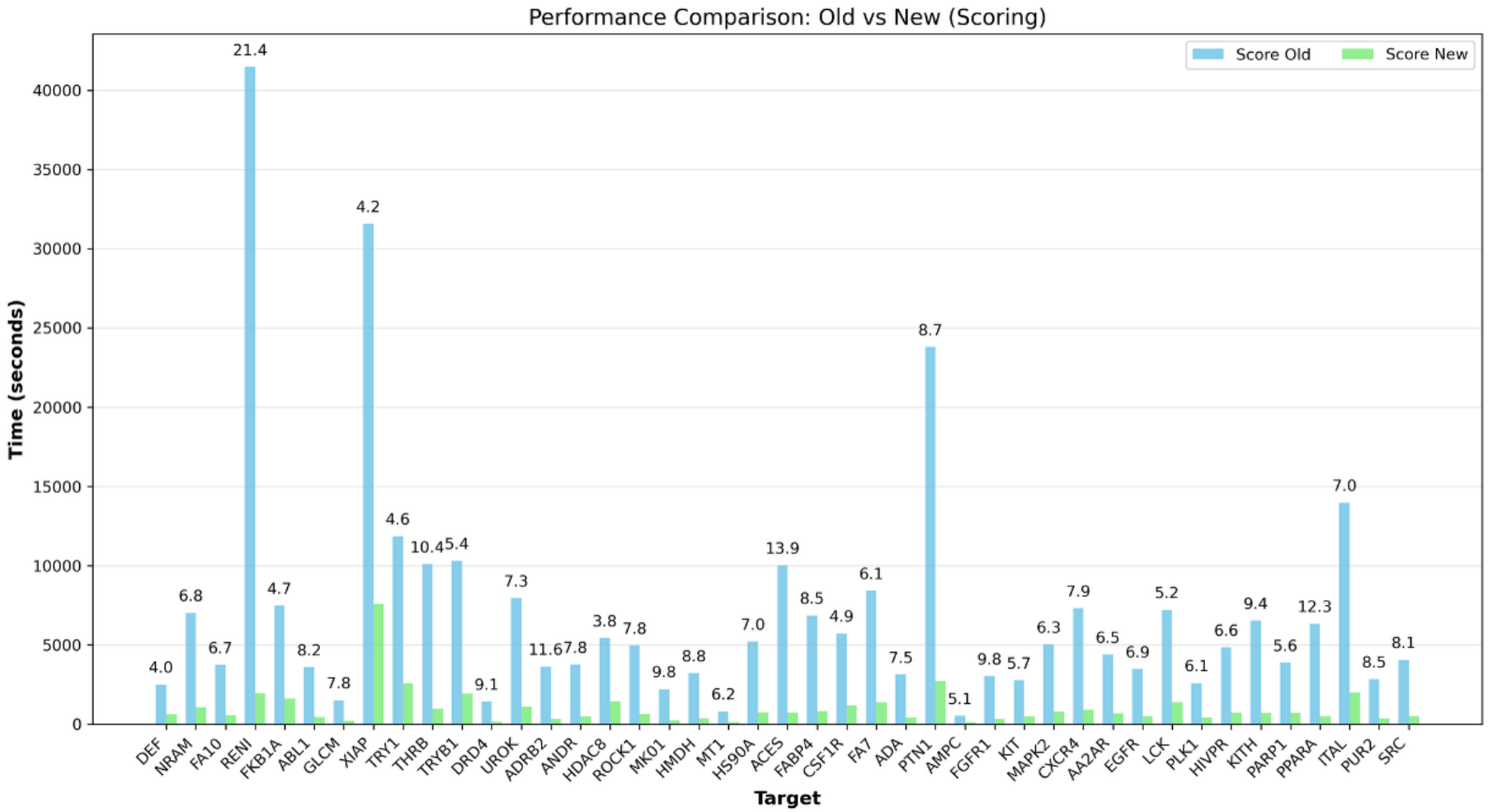
After Formula (Factorized):

```
apt = z * (y * (x * a1 + a4) + ..) + ...
```

Scoring optimization

4. Speed-up gain

Average:
7.04X



DB2 parsing optimization

- DB2 parsing in text format:
  - Type conversion required

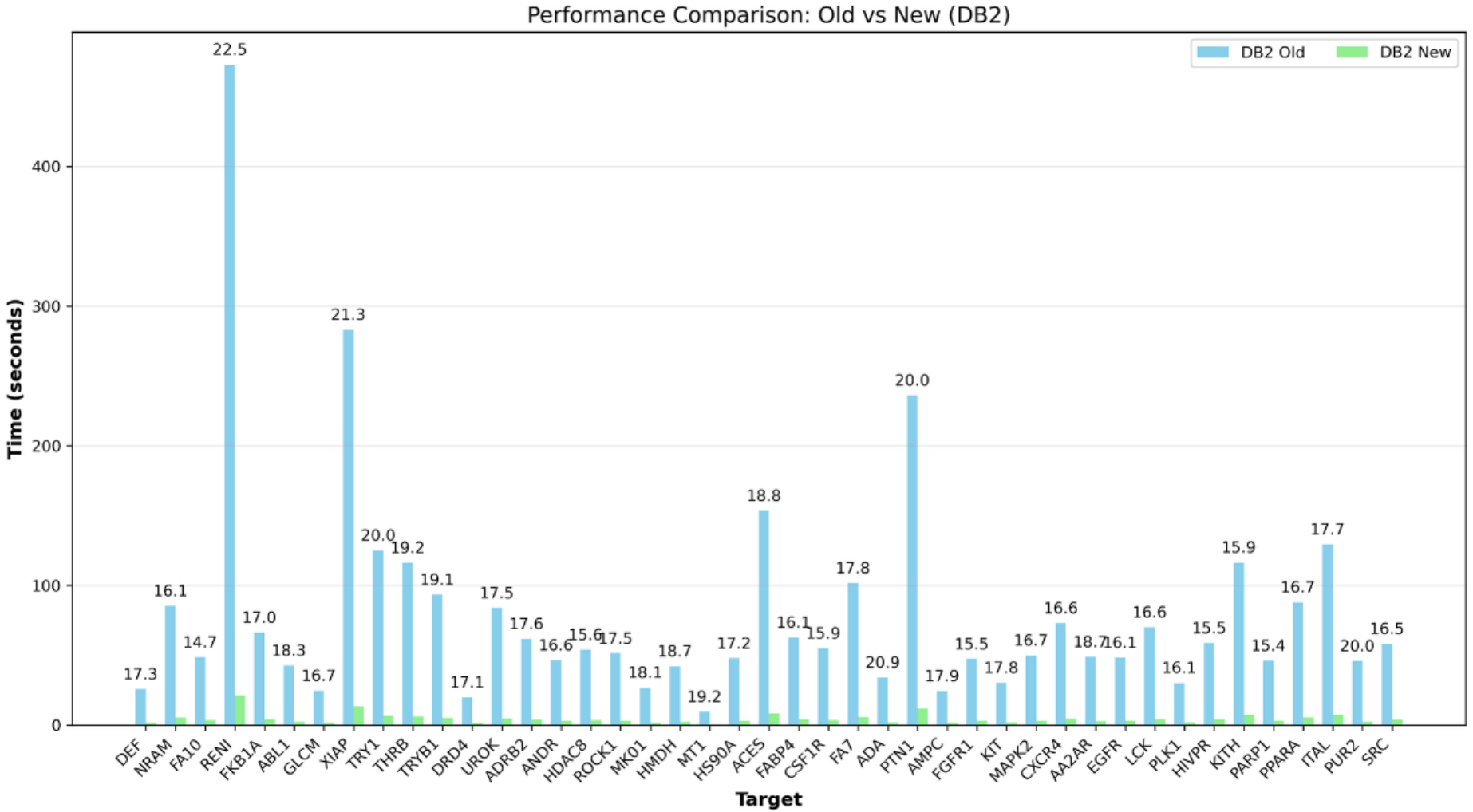
- DB2 parsing in the binary format offers:
  - Faster processing
  - Smaller storage file

- INDOCK: use_binary_db2: yes

DB2 parsing optimization

1. Speed-up gain

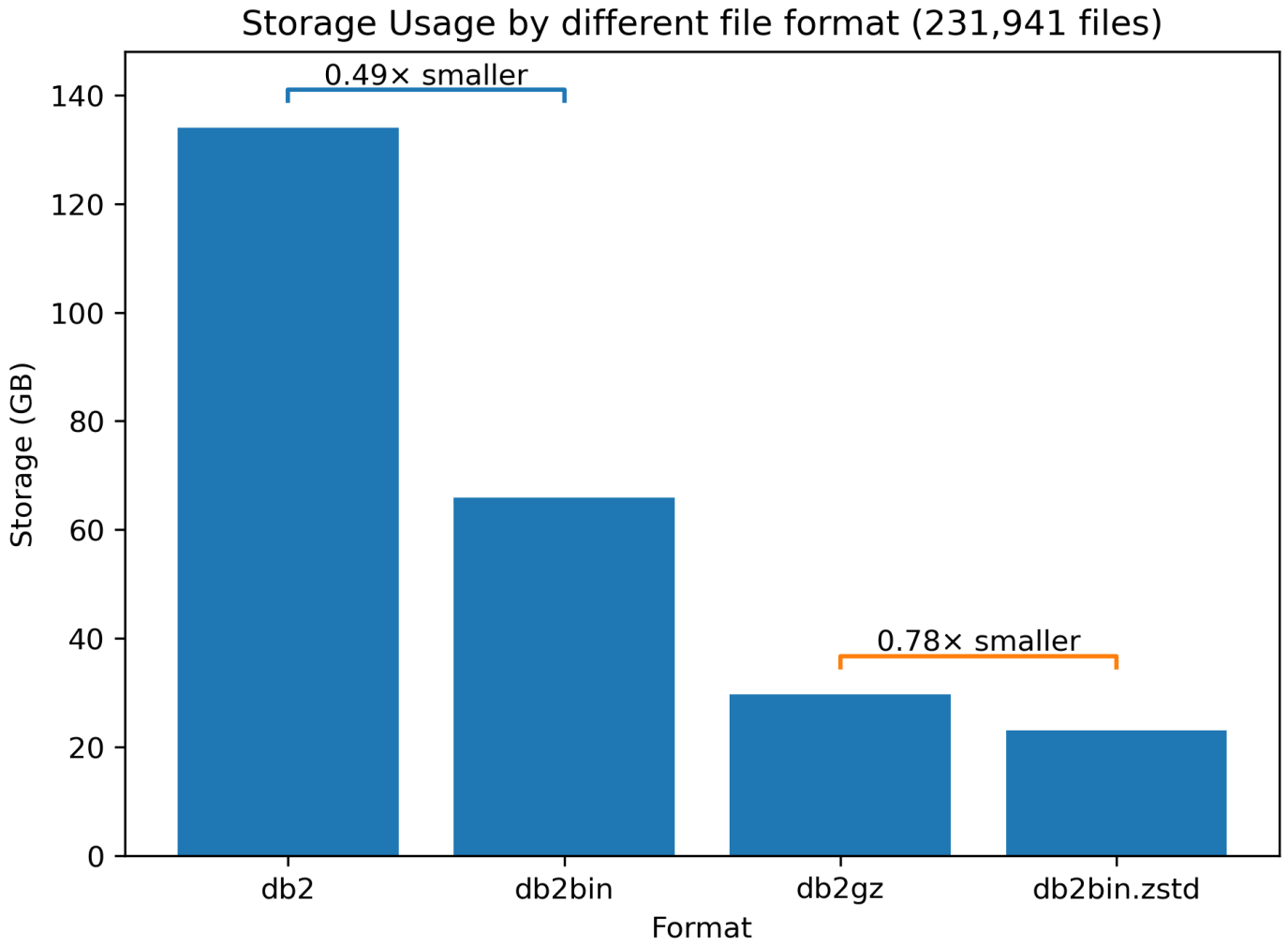
Average:
17.28X



DB2 parsing optimization

2. Size reduction gain

Random 50 tgz from ZINC22, leadlike



Input parsing optimization (optional)

A Python script to support the conversion of DB2 to db2bin.

Future plan: MolSanitizer supports db2bin

```python
#!/usr/bin/env python3
"""
db2_to_binary.py - Convert text DB2.gz files to binary DB2BIN.gz format

This script converts DOCK3 text-based DB2 files to the high-performance
binary format (db2bin.gz), which can be 5-20x faster to parse.

Usage:
    python3 db2_to_binary.py input.db2.gz output.db2bin.gz
    python3 db2_to_binary.py --dir /path/to/db2s/ /path/to/output/

Binary Format (db2bin.gz):
    - Header: magic "DB2BIN", version, counts, metadata
    - Arrays: contiguous binary arrays for fast I/O
    - All data is gzip compressed

Author: Phong Lam, Uppsala University
Date: January 2026
"""

import struct
import gzip
import argparse
from pathlib import Path


# Binary format constants
DB2BIN_MAGIC = b'DB2BIN'
DB2BIN_VERSION = 1


def parse_db2_text(input_path):
    """
    Parse a text DB2.gz file into structured data.

    Returns a dict with all the arrays needed for binary format.
    """
    data = {
        'refcod': '',
        'protcod': '',
        'smiles': '',
        'molname': '',
        'arbitrary': [],  # Lines 5+ of M section
        'total_atoms': 0,
        'total_bonds': 0,
        'total_coords': 0,
        'total_confs': 0,
        'total_sets': 0,
        'rigid_heavy_count': 0,
```

Validation – is it correct?

133,832 compounds

Matching integrity

One example

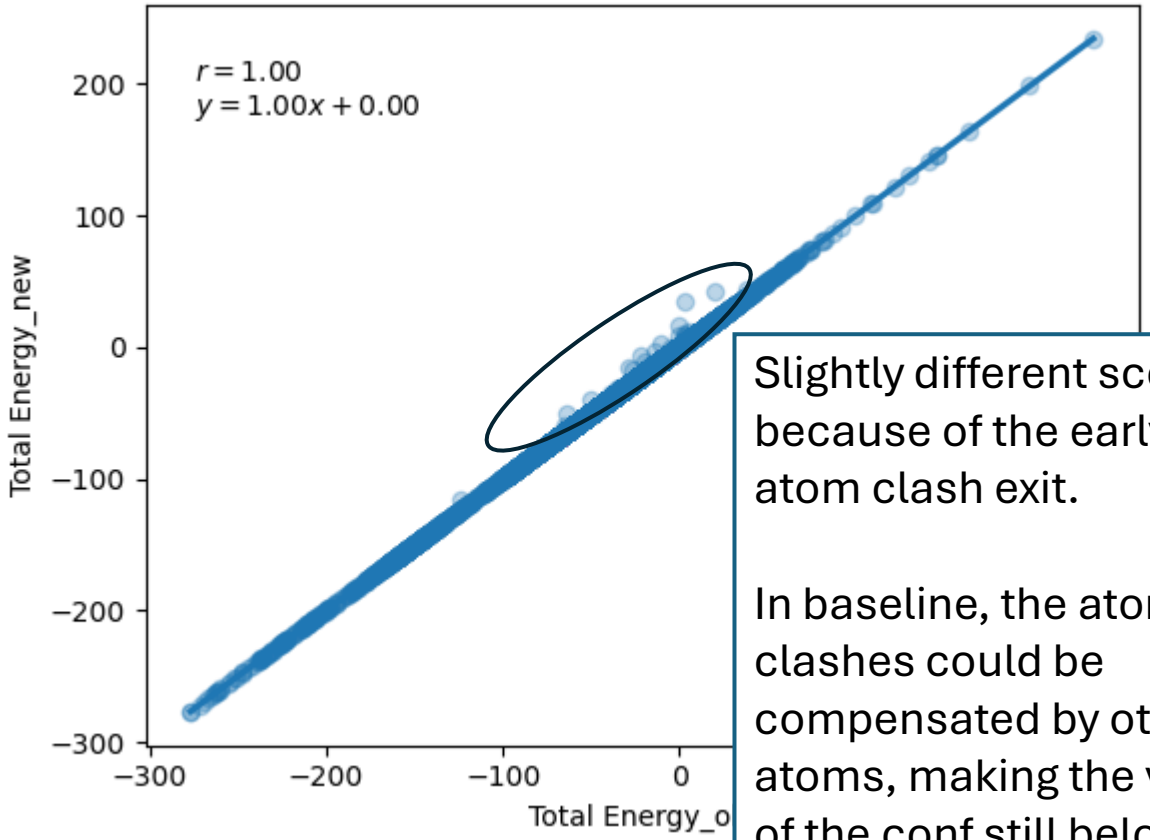
Original

```
total number of hierarchies:          9639
total number of orients (matches):      47693958
total number of conformations (sets):       1066163
total number of nodes (confs):       3110229
total number of complexes:                  5273779446
```

Optimized

```
total number of hierarchies:          9639
total number of orients (matches):      47693958
total number of conformations (sets):       1066163
total number of nodes (confs):       3110230
total number of complexes:                  5273779446
```

Scoring integrity

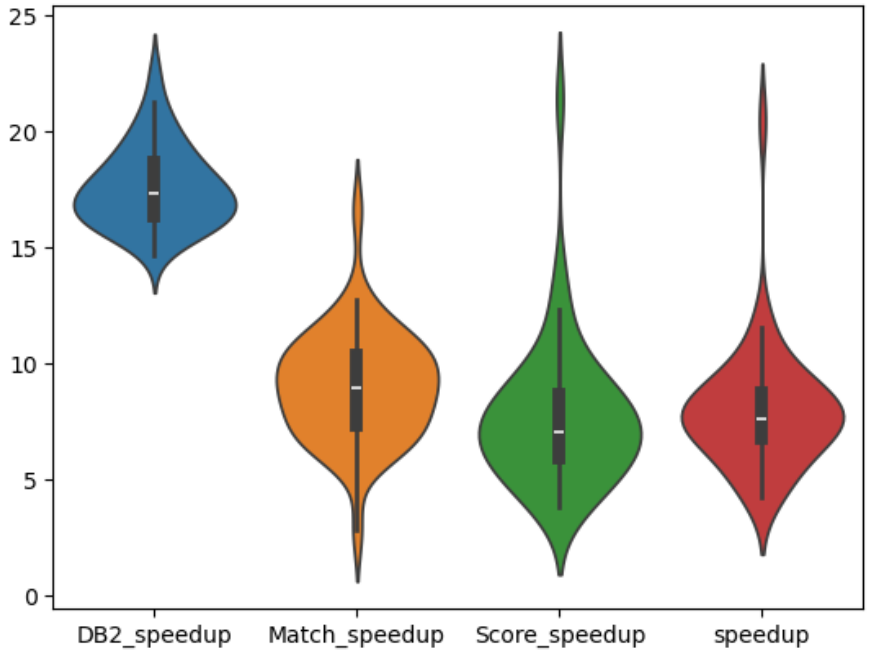


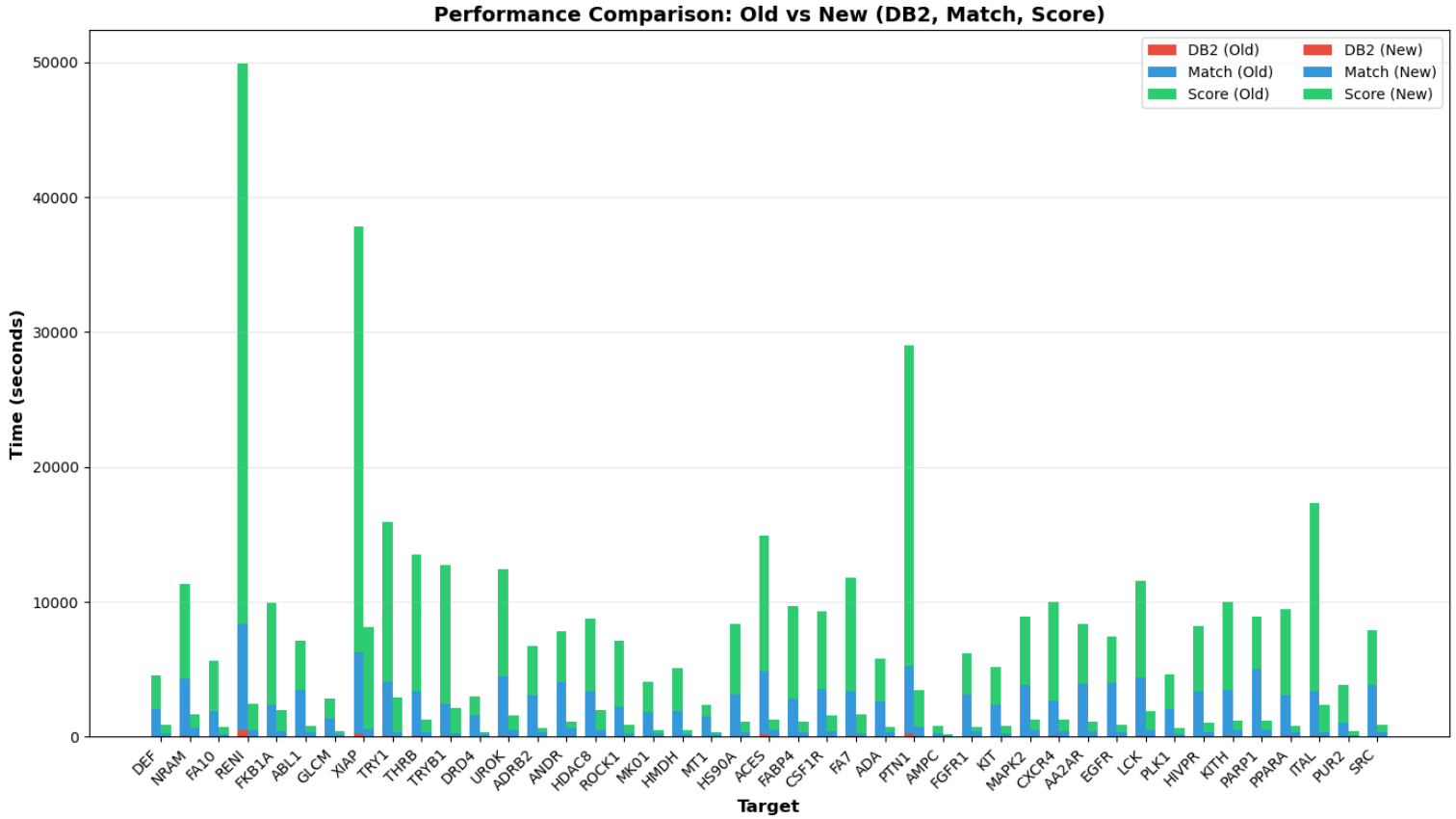
Slightly different score because of the early atom clash exit.

In baseline, the atom clashes could be compensated by other atoms, making the vdW of the conf still below max_bump

Summary

Whole-optimization speedup:
7.62X





Comparison to Niu Huang's implementation

- Based on DOCK3.8.5.
- Support original DB2 pipeline.
- Matching: still uses degeneracy check (optimized), while they used dist_tol_last to check for generated matches (10.1109/TPDS.2022.3194916), I tried to reproduce but at the end resulted in different number of matches → end-up optimizing the degeneracy check + canonical ordering.
- Scoring: they used SIMD for simultaneous 8-grid scoring, I used SIMD for 8-atom scoring. I used FMA as well as lazy evaluation. I did not require the re-organization of the DB2 format (conf and set sorting – 10.1145/3712285.3759833)