Acceleration of the prefiltering and alignment phases in MMseqs2 using the Processing-in-Memory approach

Anna Zalesińska,
Mateusz Juszczak,
Ignacy Kuźniewski,
Tomasz Wiśniewski

Motivations

- Exploring a new hardware architecture – *Processing-in-Memory* (PIM).
- Identifying compatible computational model – Many-to-many sequence alignment.

- Application in a state-of-the-art pipeline
  – MMseqs2 acceleration as core element of structural prediction pipelines such as ColabFold.
- Exploratory Proof-of-Concept to evaluate physical hardware constraints.

Potential advantages of Processing-In-Memory Architecture

- MMseqs2 search is dominated by large database accesses rather than complex arithmetic
- Sequence comparisons expose massive parallelism across database shards
- K-mer lookup is based mostly on simple integer operations and irregular accesses
- A database-stationary PIM design can reduce repeated data movement

UPMEM PIM Architecture

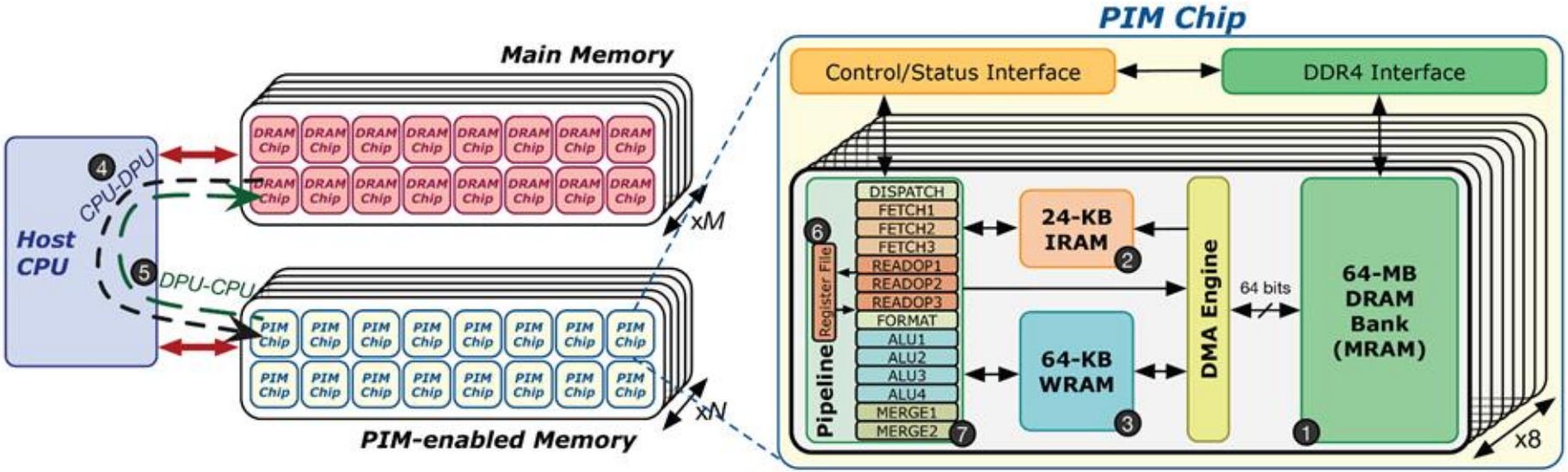
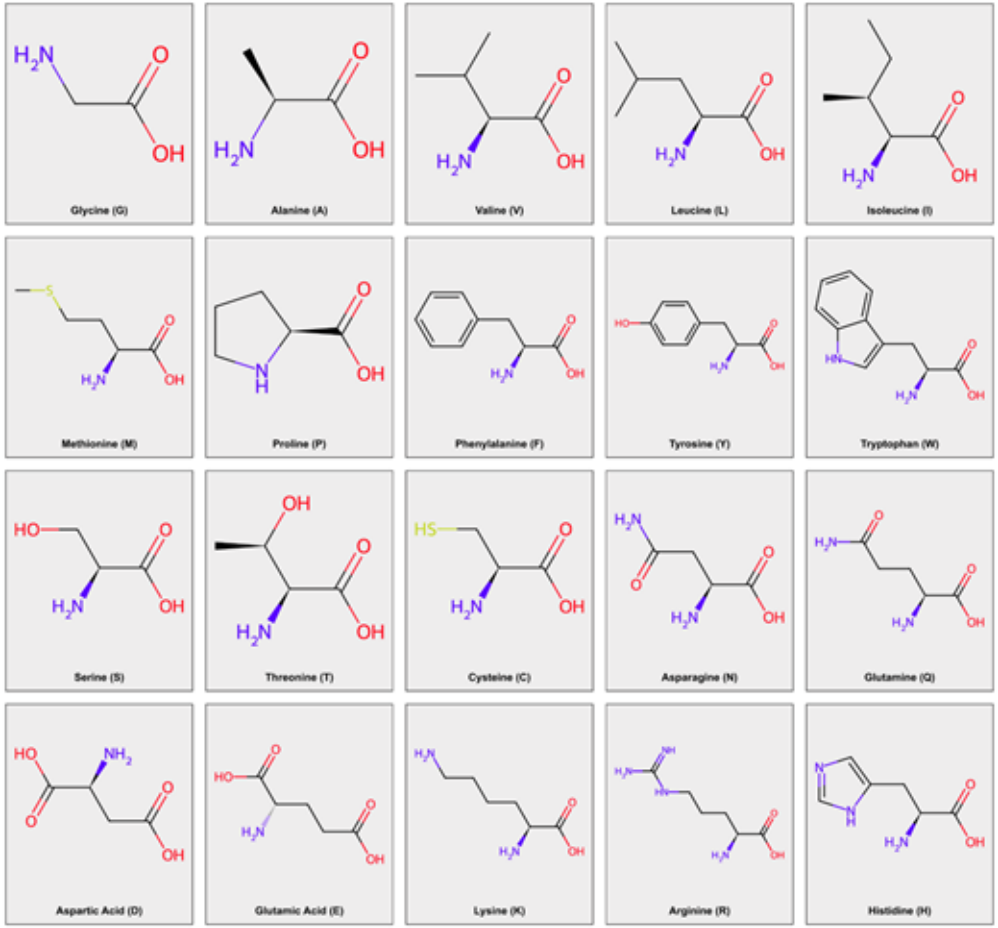


Figure 1: UPMEM-based PIM system with a host CPU, standard main memory, and PIM-enabled memory (left), and internal components of a UPMEM PIM chip (right) [198, 199].

Source: Gómez-Luna et al., *Benchmarking a New Paradigm...*, arXiv:2105.03814 (v7, 2022).

Sequence alignment

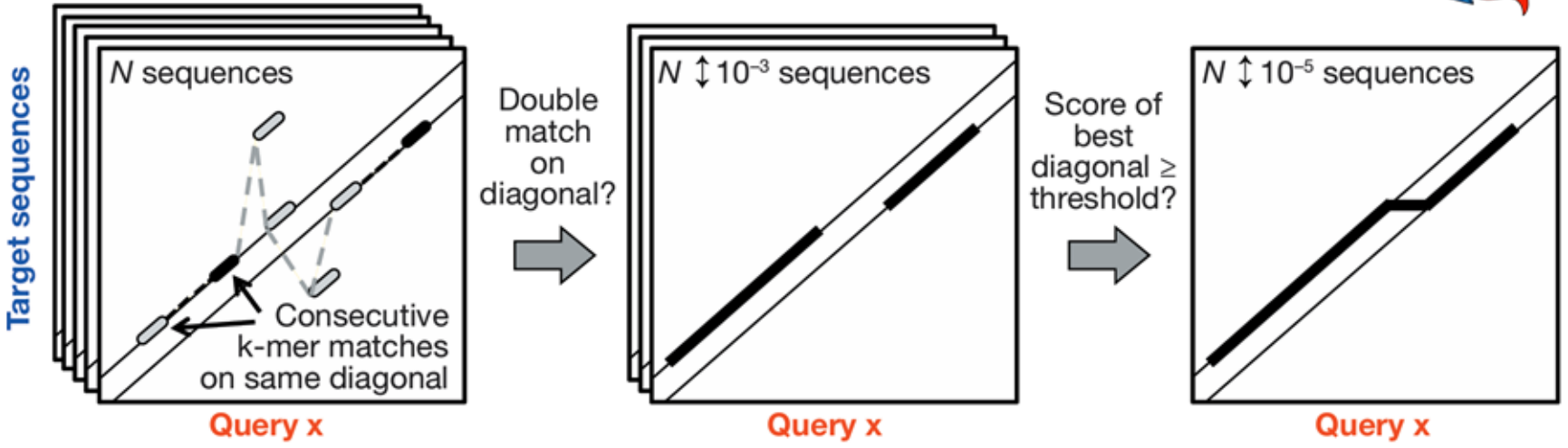


Aligned sequences

Query:	K	L	G	I	F	I	G	F	D	V
Target:	K	L	G	-	F	L	-	-	D	-

Search pipeline





Source: Steinegger & Söding (2017), *Nat Biotechnol*, Fig. 1, doi:10.1038/nbt.3988

The processing stages run independently
and communicate via the file system

Adapted Processing Stages

- k-mer matching
- ungapped alignment
- gap-affine alignment

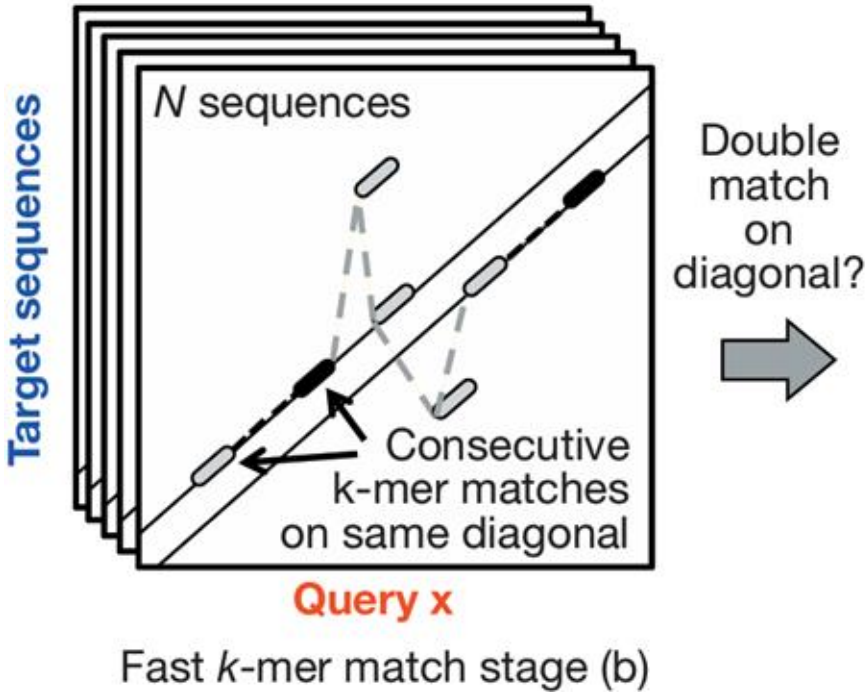
In addition, relative to the reference PIM implementation, we added support for Position-Specific Substitution Matrices (PSSMs).

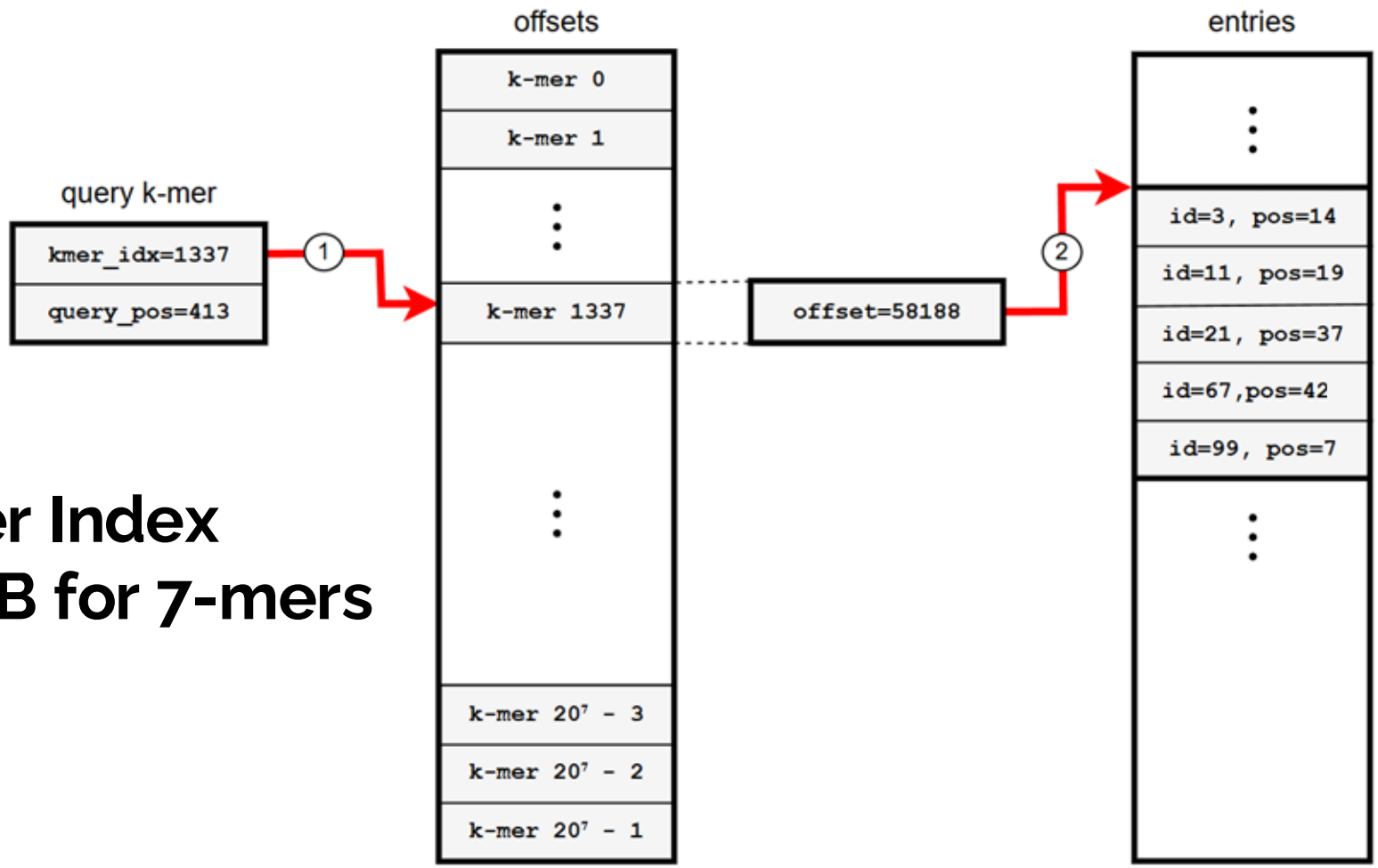
This addresses the limitation of treating all amino acid mismatches equally, even though different substitutions have very different biological plausibility.

Źródło: Steinegger & Söding (2017), *Nat Biotechnol*, Fig. 1, doi:10.1038/nbt.3988

K-mer matching stage

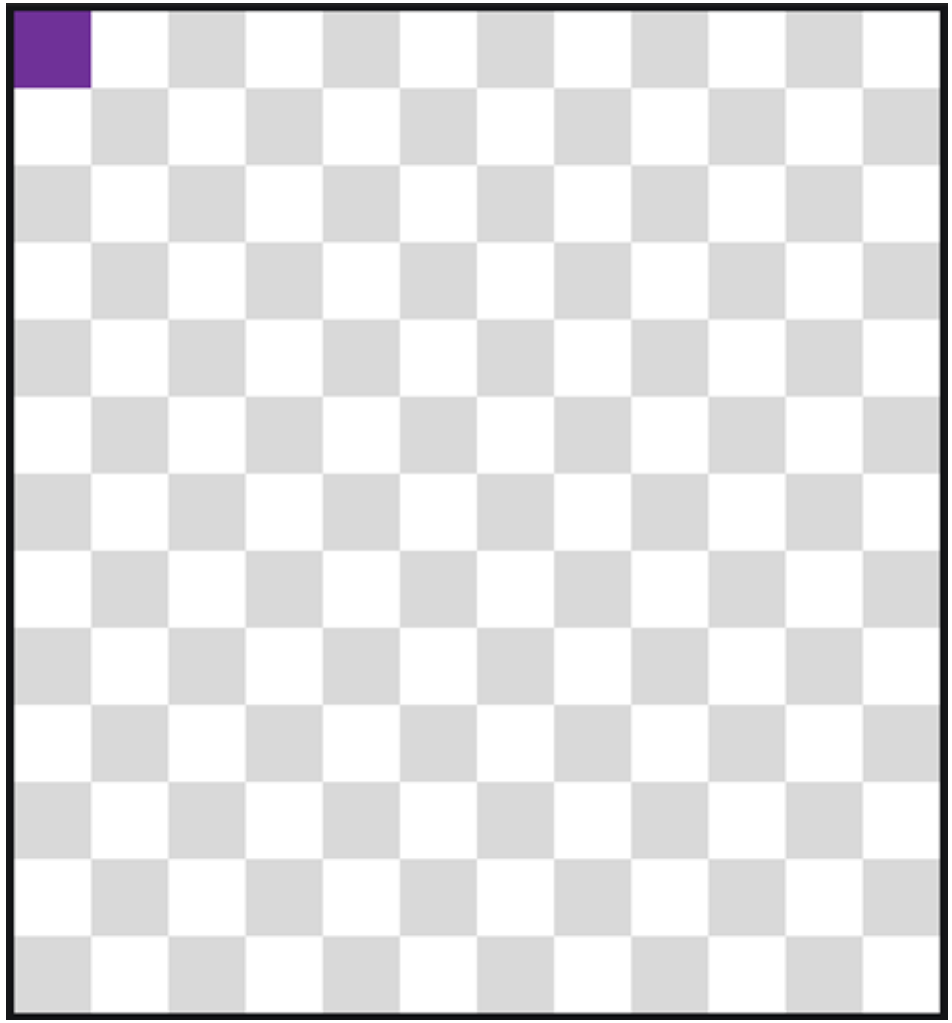
- **Goal:** Identification of pairs of sequences with detecting double k-mer matches on diagonal
- **Mechanizm:** Fast querying of the k-mer index and geometric verification of double matches using state tracking table

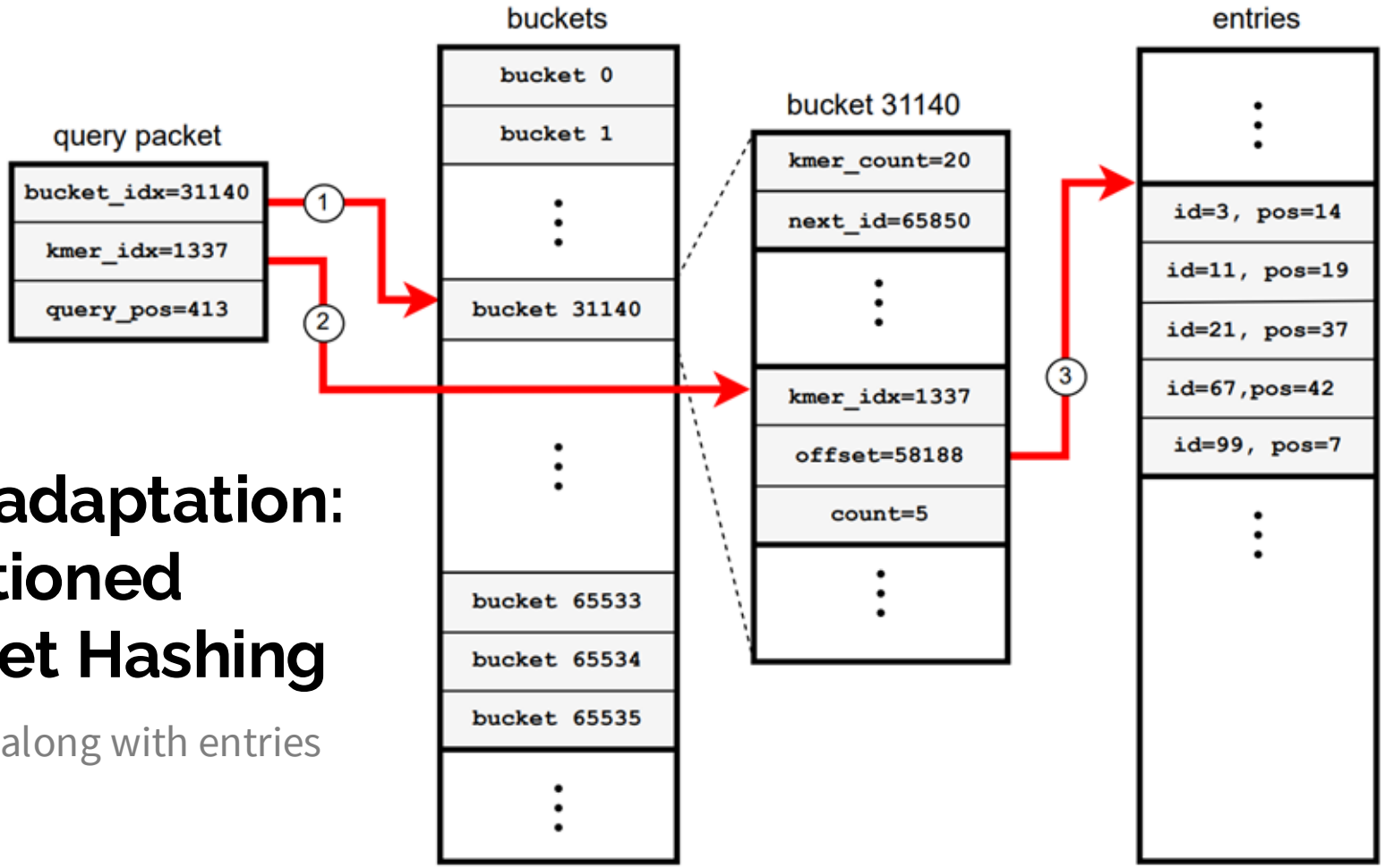




K-mer Index
~10GB for 7-mers

- **A single DPU can physically hold a maximum of ~0.65% of the total 7-mer space**
- DPU Memory: **64 MB**
- K-mer Entry Size: **8 Bytes**
  (4B for value + 4B for IDs)
- Absolute Max Capacity:
  64 MB / 8 B = ~8.3 Million entries
- Total Possible 7-mers:
  1.28 Billion combinations

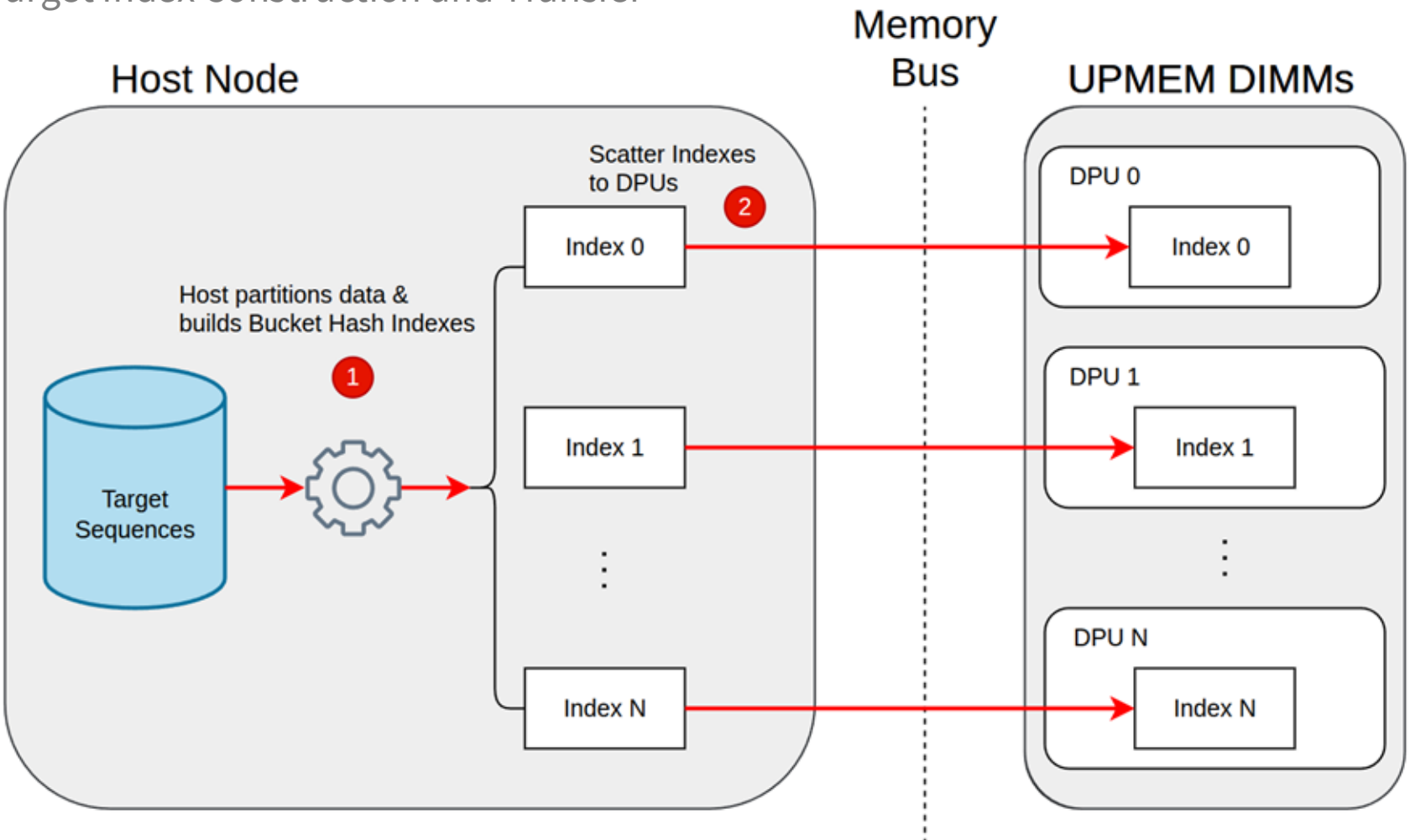




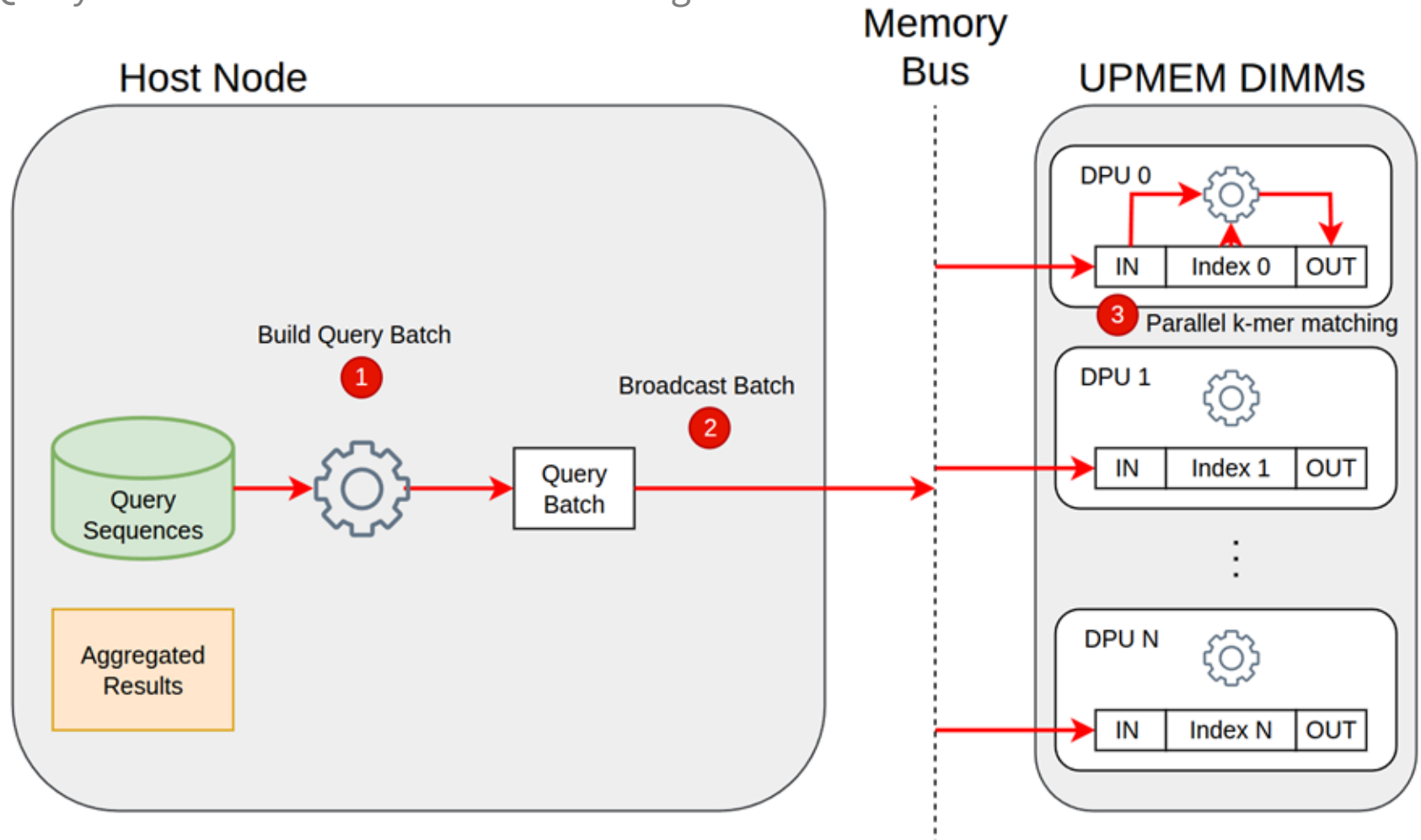
DPU adaptation: Partitioned Bucket Hashing

~ 44 MB along with entries

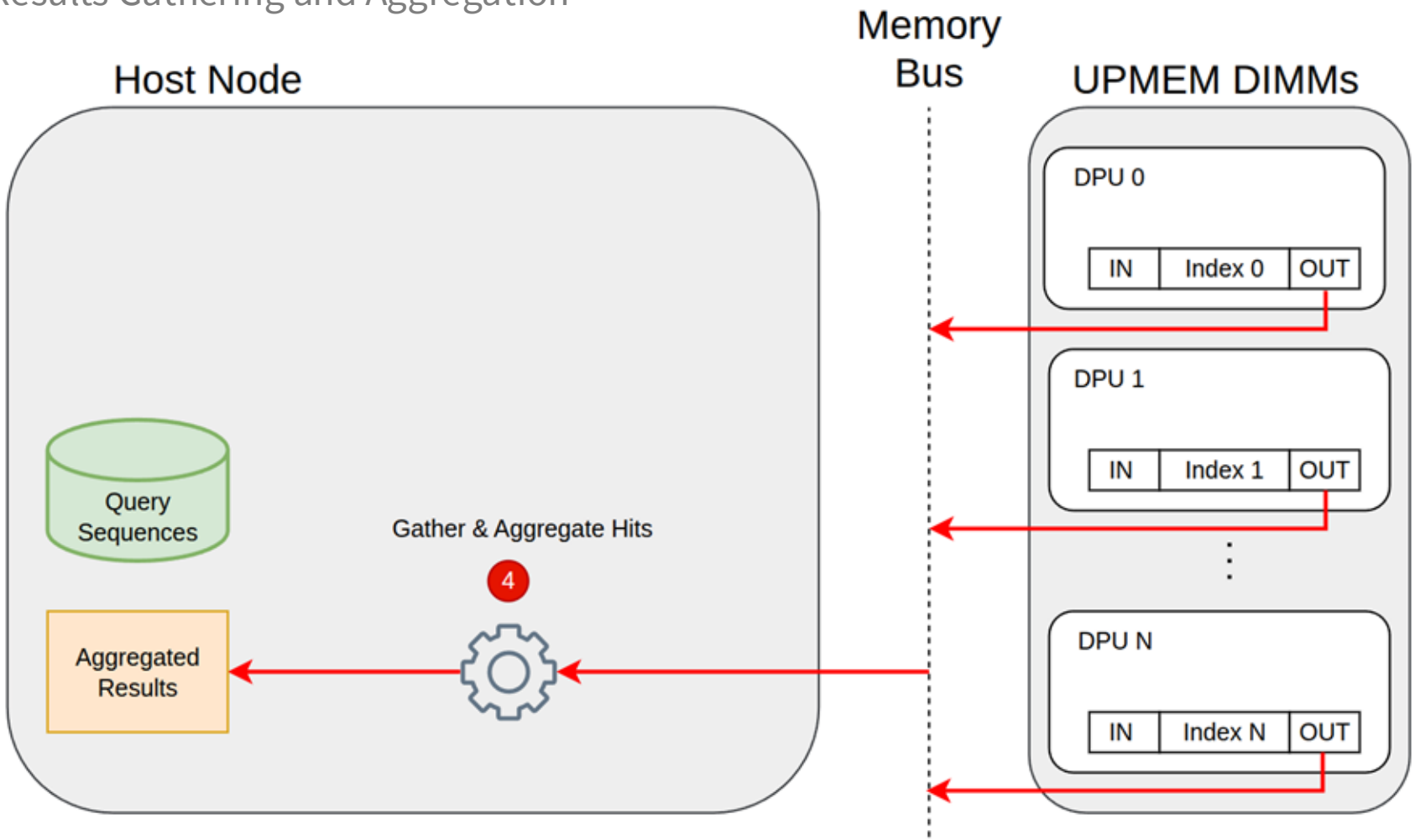
Target Index Construction and Transfer



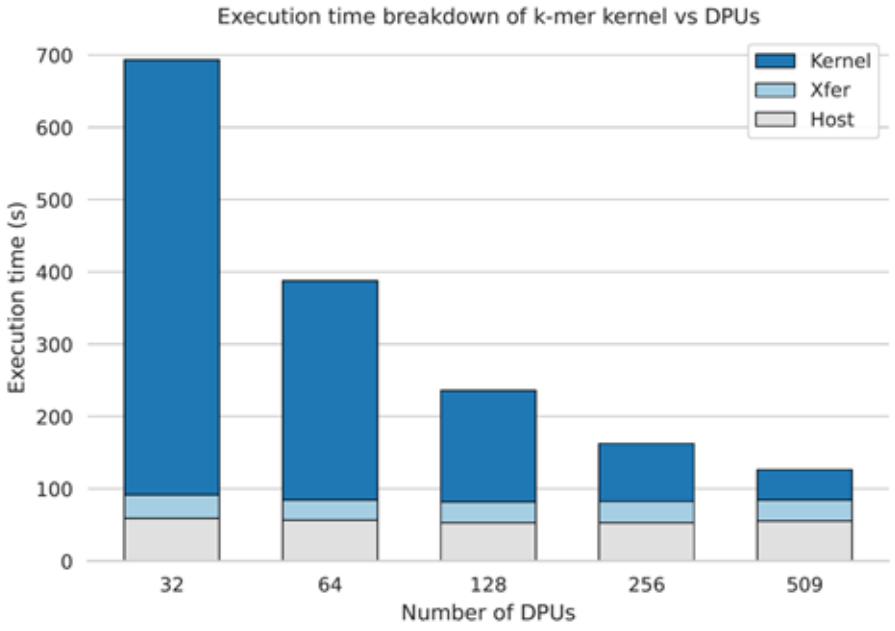
Query Batch Construction and Processing

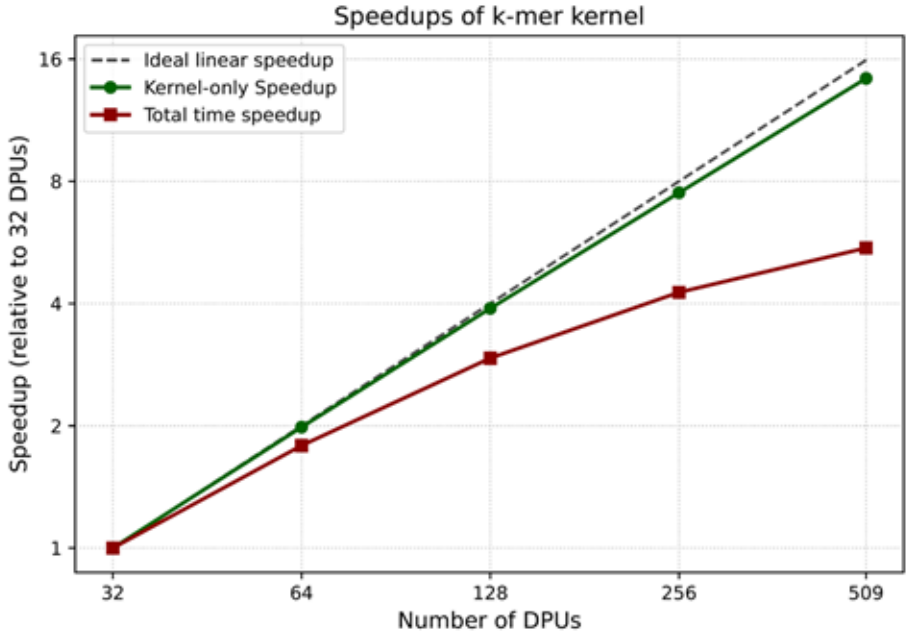


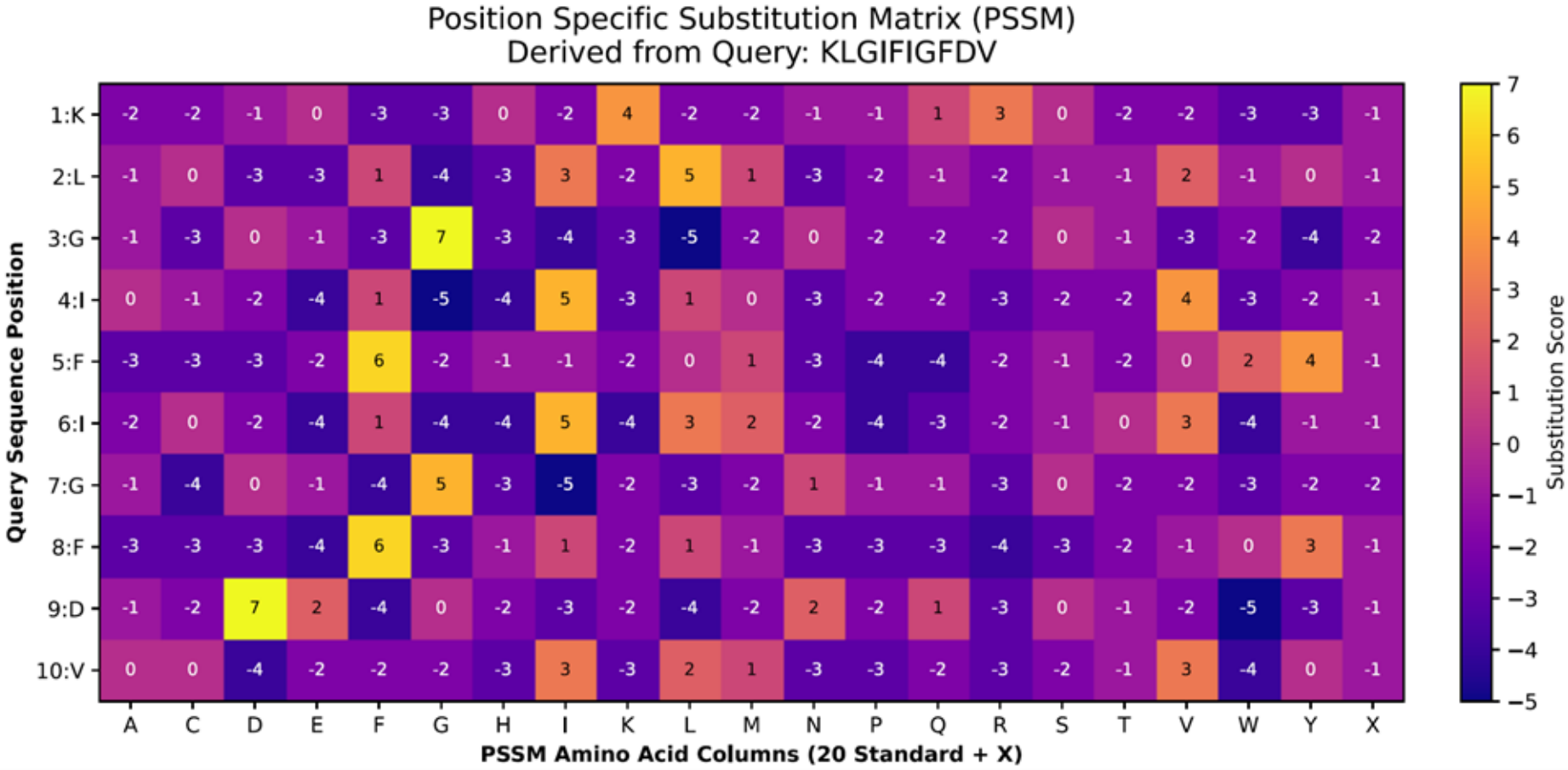
Results Gathering and Aggregation

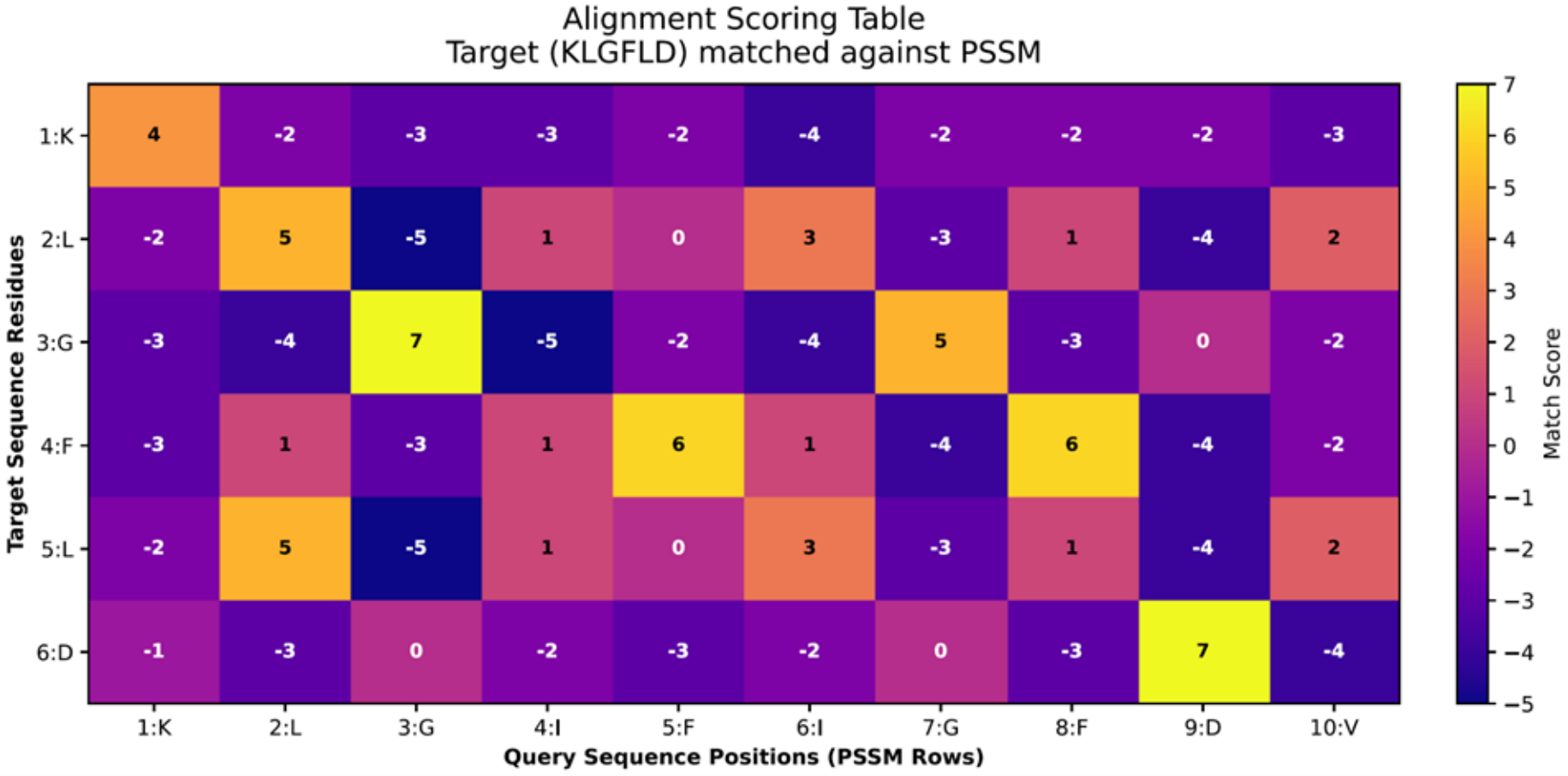


Scaling Results & The Data Transfer Bottleneck



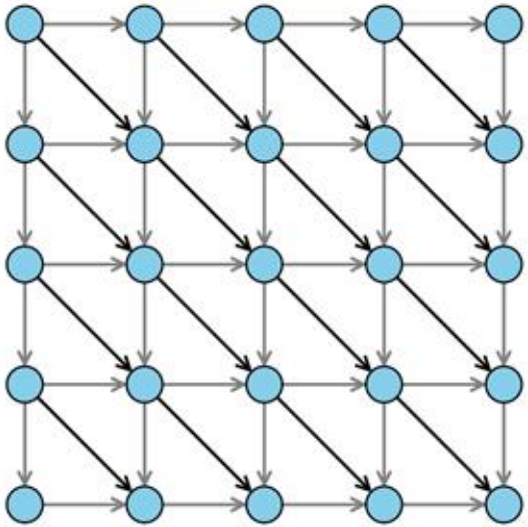



Position Specific Substitution Matrix (PSSM)
Derived from Query: KLGIFIGFDV
Query Sequence Position
1:K -2 -2 -1 0 -3 -3 0 -2 4 -2 -2 -1 -1 1 3 0 -2 -2 -3 -3 -1
2:L -1 0 -3 -3 1 -4 -3 3 -2 5 1 -3 -2 -1 -2 -1 -1 2 -1 0 -1
3:G -1 -3 0 -1 -3 7 -3 -4 -3 -5 -2 0 -2 -2 -2 0 -1 -3 -2 -4 -2
4:I 0 -1 -2 -4 1 -5 -4 5 -3 1 0 -3 -2 -2 -3 -2 -2 4 -3 -2 -1
5:F -3 -3 -3 -2 6 -2 -1 -1 -2 0 1 -3 -4 -4 -2 -1 -2 0 2 4 -1
6:I -2 0 -2 -4 1 -4 -4 5 -4 3 2 -2 -4 -3 -2 -1 0 3 -4 -1 -1
7:G -1 -4 0 -1 -4 5 -3 -5 -2 -3 -2 1 -1 -1 -3 0 -2 -2 -3 -2 -2
8:F -3 -3 -3 -4 6 -3 -1 1 -2 1 -1 -3 -3 -3 -4 -3 -2 -1 0 3 -1
9:D -1 -2 7 2 -4 0 -2 -3 -2 -4 -2 2 -2 1 -3 0 -1 -2 -5 -3 -1
10:V 0 0 -4 -2 -2 -2 -3 3 -3 2 1 -3 -3 -2 -3 -2 -1 3 -4 0 -1
A C D E F G H I K L M N P Q R S T V W Y X
PSSM Amino Acid Columns (20 Standard + X)
Substitution Score
7 6 5 4 3 2 1 0 −1 −2 −3 −4 −5


Alignment Scoring Table
Target (KLGFLD) matched against PSSM
Target Sequence Residues
1:K
2:L
3:G
4:F
5:L
6:D
4 -2 -3 -3 -2 -4 -2 -2 -2 -3
-2 5 -5 1 0 3 -3 1 -4 2
-3 -4 7 -5 -2 -4 5 -3 0 -2
-3 1 -3 1 6 1 -4 6 -4 -2
-2 5 -5 1 0 3 -3 1 -4 2
-1 -3 0 -2 -3 -2 0 -3 7 -4
1:K
2:L
3:G
4:I
5:F
6:I
7:G
8:F
9:D
10:V
Query Sequence Positions (PSSM Rows)
Match Score
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−5

Ungapped kernel

Gapped Alignment

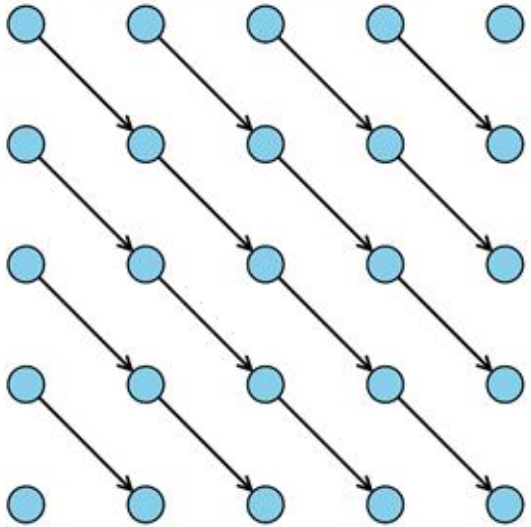


Fully Connected Mesh
(diagonal, horizontal and vertical dependencies)

$$M_{i,j} = \max \begin{cases} 0 & \text{(Local start)} \\ M_{i-1,j-1} + \text{PSSM}(i, B_j) & \text{(Match)} \\ M_{i-1,j} + d & \text{(Deletion)} \\ M_{i,j-1} + d & \text{(Insertion)} \end{cases}$$

M: Dynamic programming matrix
d: Gap penalty
$\text{PSSM}(i, B_j)$: Score at pos i for AA B_j

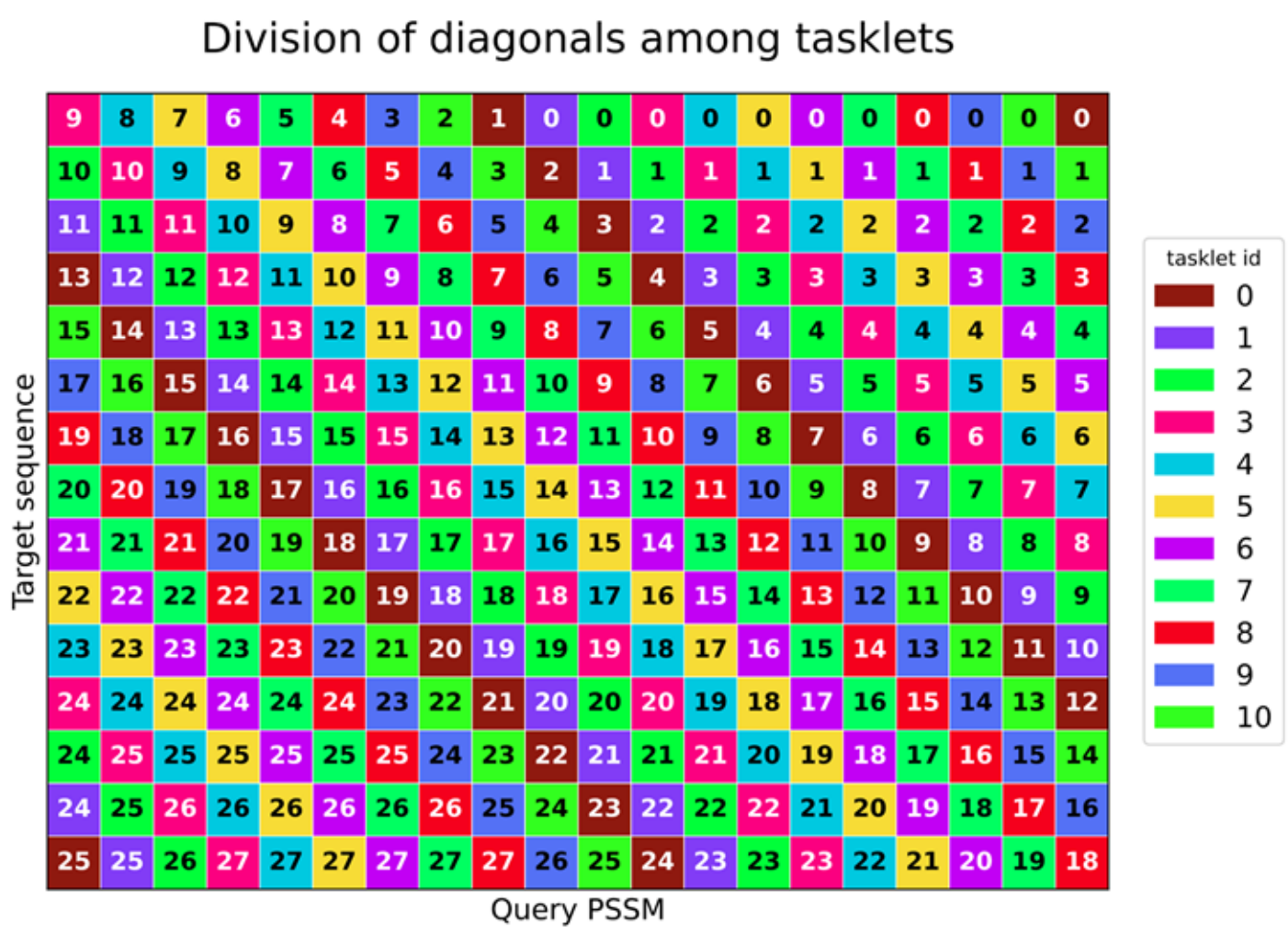
Ungapped Alignment

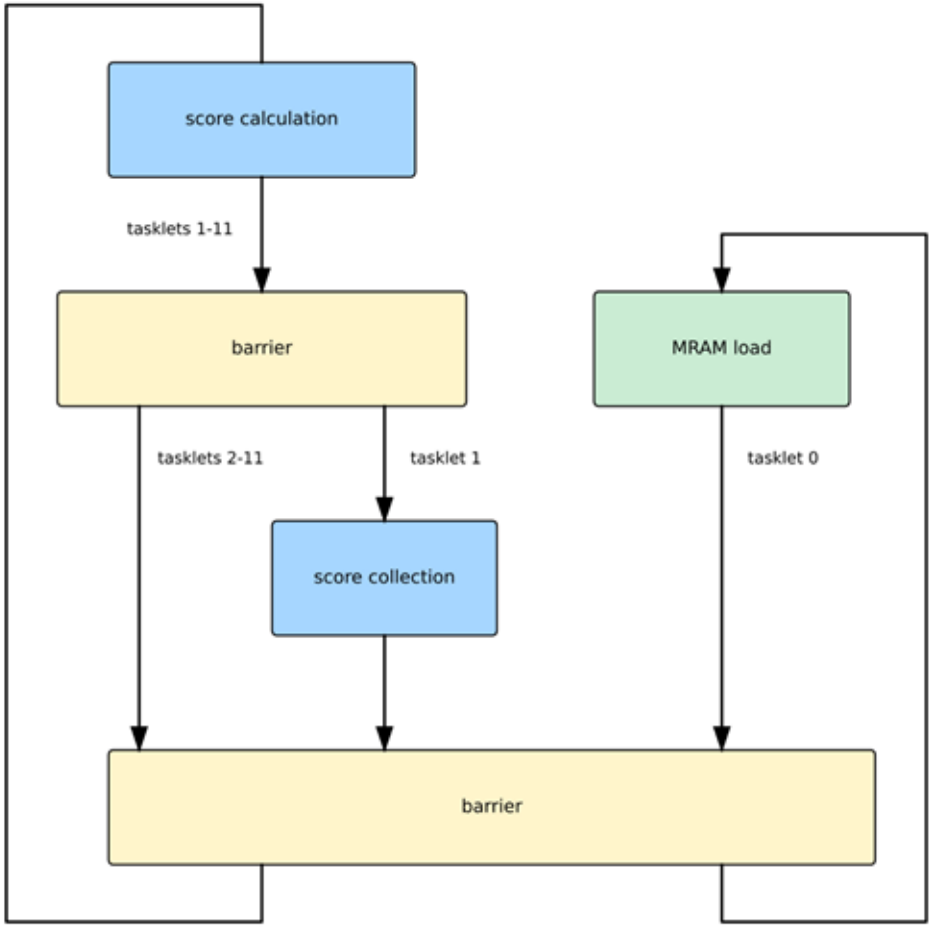


Isolated Diagonal Chains
(no communication between diagonals)

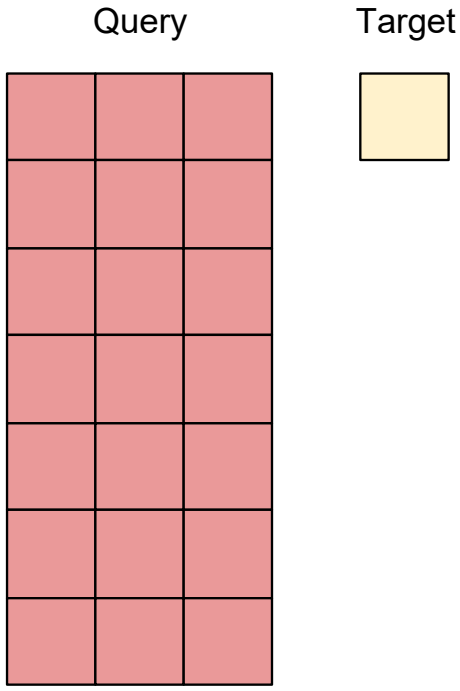
$$M_{i,j} = \max \begin{cases} 0 & \text{(Local start)} \\ M_{i-1,j-1} + \text{PSSM}(i, B_j) & \text{(Match)} \end{cases}$$

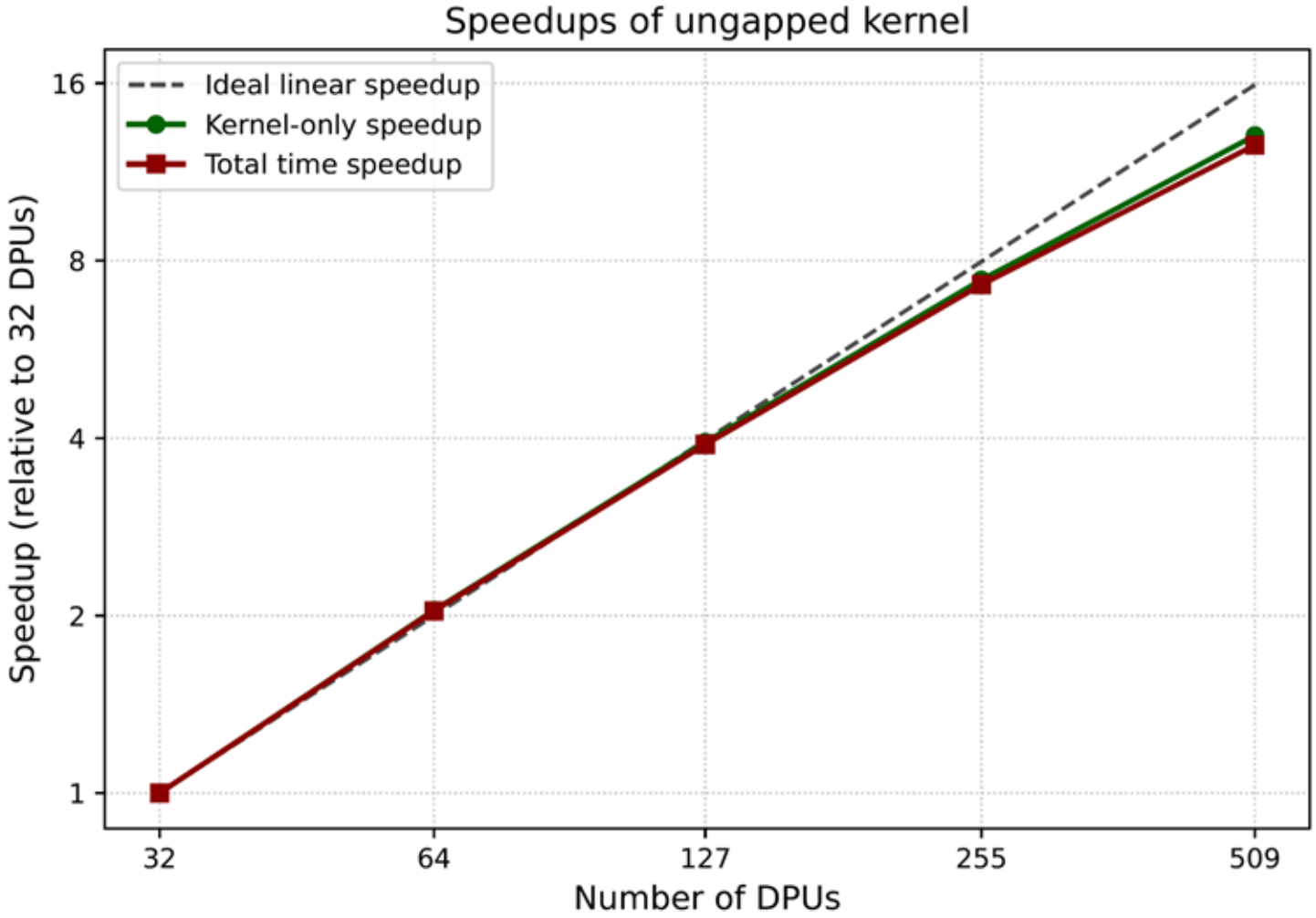
M: Dynamic programming matrix
$\text{PSSM}(i, B_j)$: Score at pos i for AA B_j


Division of diagonals among tasklets
Target sequence
Query PSSM
tasklet id
0
1
2
3
4
5
6
7
8
9
10

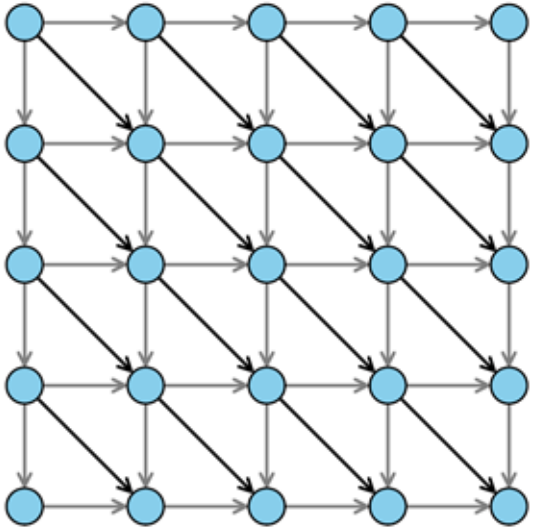



```
1  for (i = 0; i < num_queries; i += 1) {
2      for (j = 0; j < num_targets; j += 1) {
3          align(queries[i], targets[j])
4      }
5  }
```




Speedups of ungapped kernel
Ideal linear speedup
Kernel-only speedup
Total time speedup
Speedup (relative to 32 DPUs)
16
8
4
2
1
32
64
127
255
509
Number of DPUs

Gapped Alignment

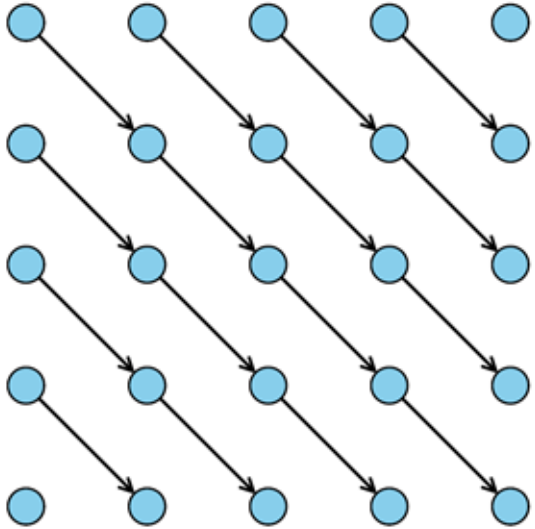


Fully Connected Mesh
(diagonal, horizontal and vertical dependencies)

$$M_{i,j} = \max \begin{cases} 0 & \text{(Local start)} \\ M_{i-1,j-1} + \text{PSSM}(i, B_j) & \text{(Match)} \\ M_{i-1,j} + d & \text{(Deletion)} \\ M_{i,j-1} + d & \text{(Insertion)} \end{cases}$$

M: Dynamic programming matrix
d: Gap penalty
$\text{PSSM}(i, B_j)$: Score at pos i for AA B_j

Ungapped Alignment



Isolated Diagonal Chains
(no communication between diagonals)

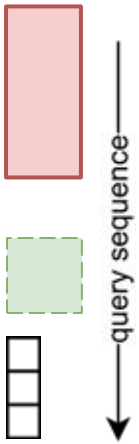
$$M_{i,j} = \max \begin{cases} 0 & \text{(Local start)} \\ M_{i-1,j-1} + \text{PSSM}(i, B_j) & \text{(Match)} \end{cases}$$

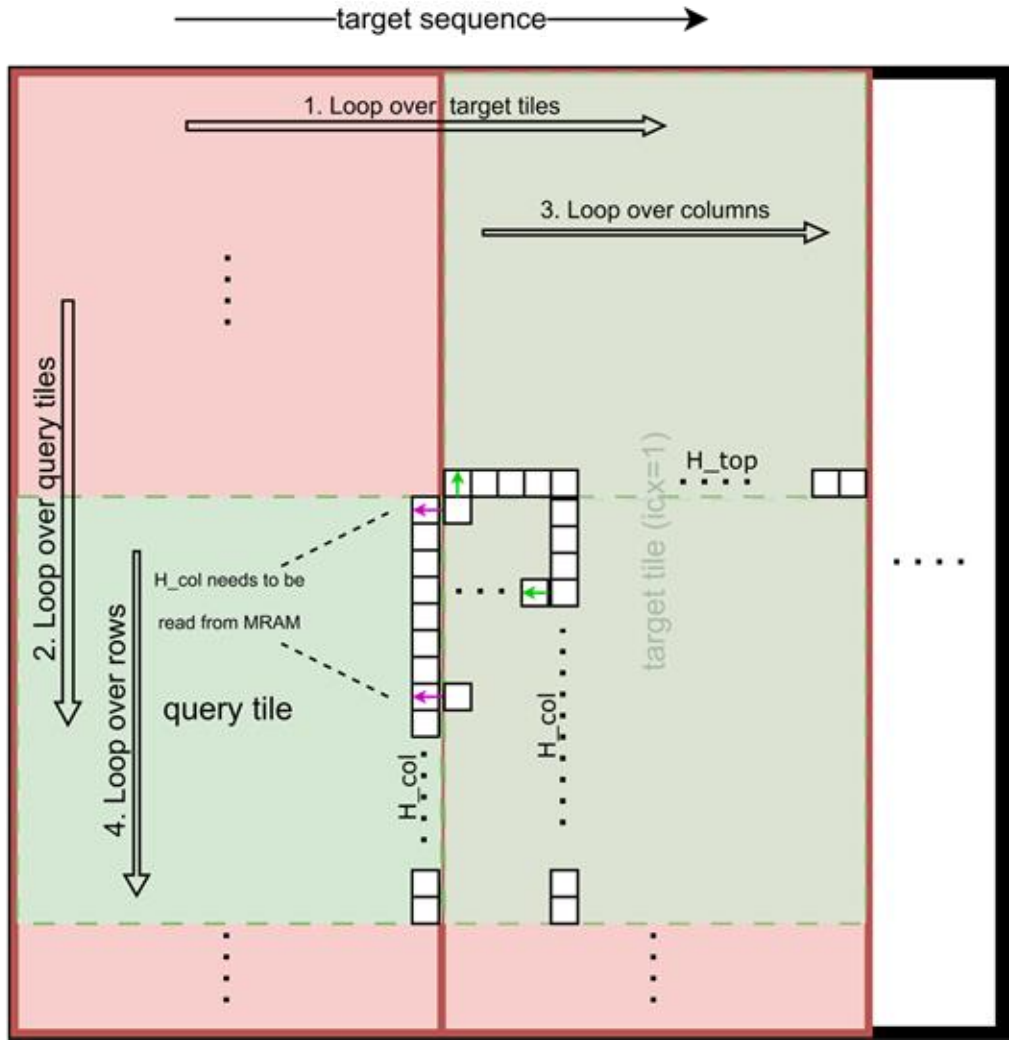
M: Dynamic programming matrix
$\text{PSSM}(i, B_j)$: Score at pos i for AA B_j

Gap-affine kernel

Four-Level Loop Structure:

- Outer loop over target tiles (64 x full height),
- Outer loop over query tiles (64 x 64),
- Inner loop over columns,
- Innermost loop over cells





Host side (CPU)

- Longest Processing Time (LPT) target-to-DPU scheduling,
- MRAM target replacement, up to one batch,
- Per-query replacement of the query and its associated PSSM,
- DPU state tracking:
  – processing,
  – result-ready status,
  – work reassignment
- End-of-query result aggregation and post-processing

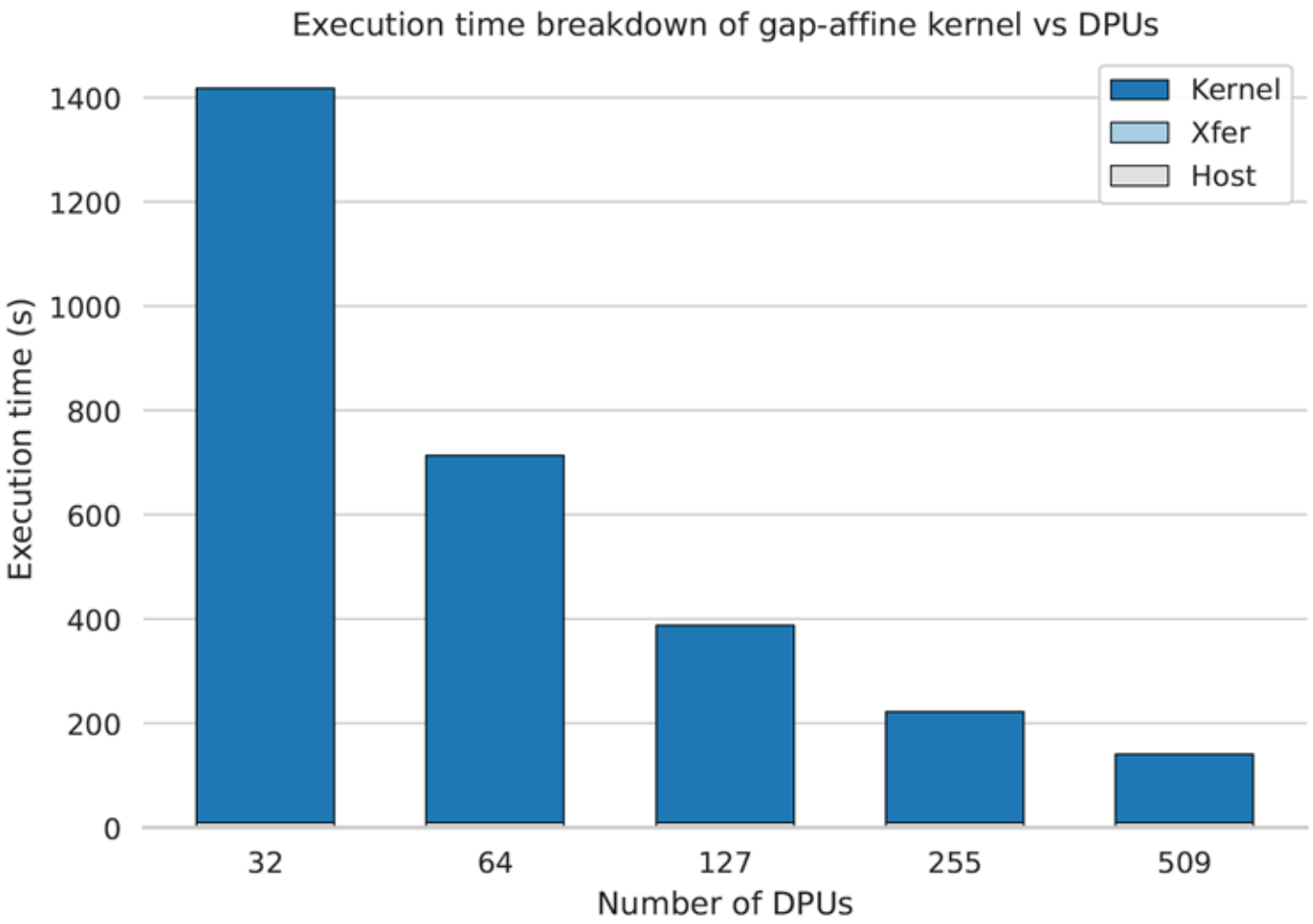
Kernel side

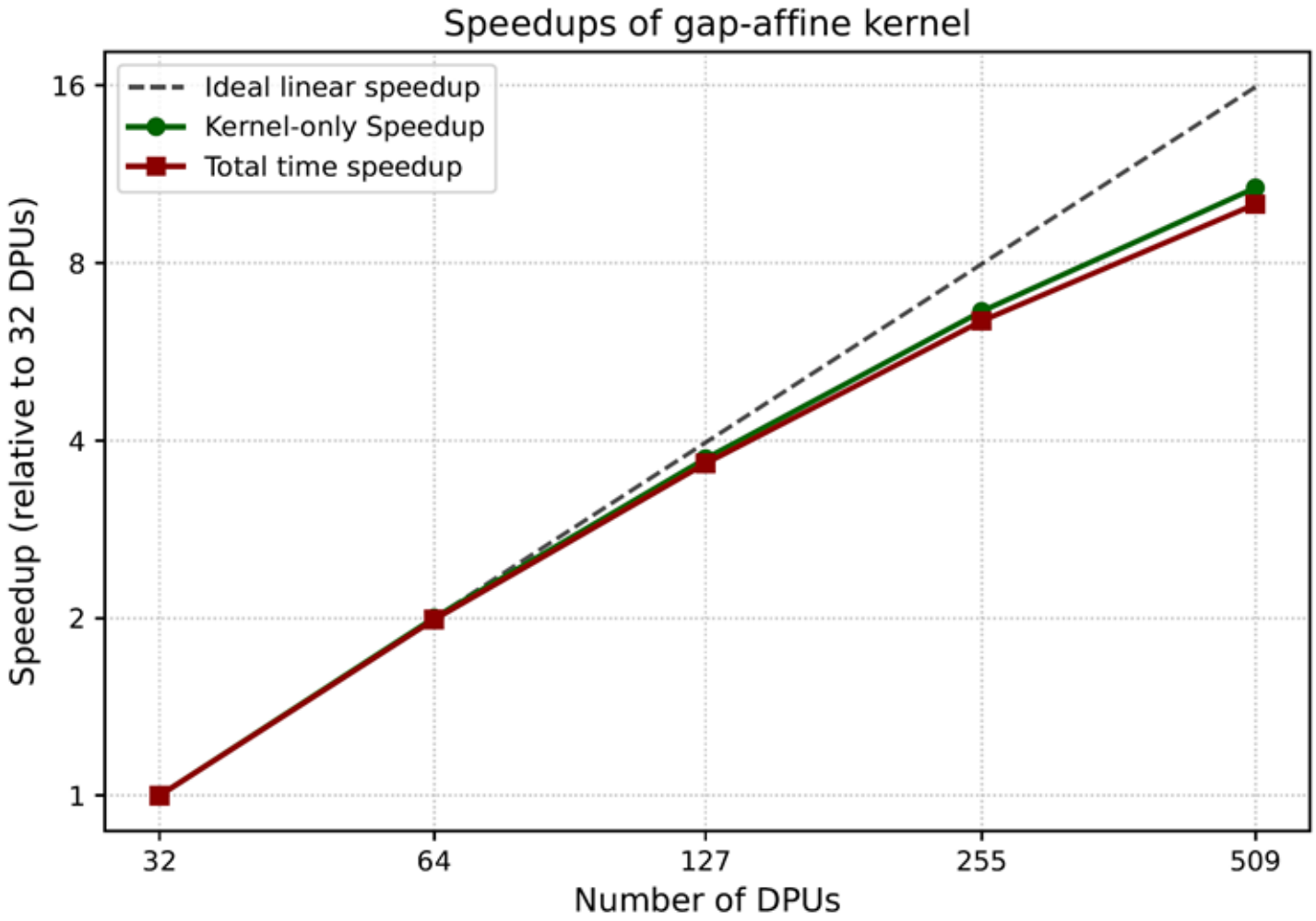
MRAM usage:

- Each target tile is read once from MRAM into WRAM, then fully consumed locally during the inner DP sweep ,
- The vertical DP boundary is not kept as a full matrix; instead, boundary vectors are reloaded from MRAM at tile boundaries and written back after processing each target tile,
- The PSSM is stored as a contiguous 1D buffer in row-major form and fetched tile-wise, enabling sequential access during the inner loop.



Instead of storing the full previous row, the inner loop carries only a small scalar state across iterations, mainly the diagonal and upper neighbors needed for the recurrence.


Execution time breakdown of gap-affine kernel vs DPUs
Kernel
Xfer
Host
Execution time (s)
1400
1200
1000
800
600
400
200
0
32
64
127
255
509
Number of DPUs

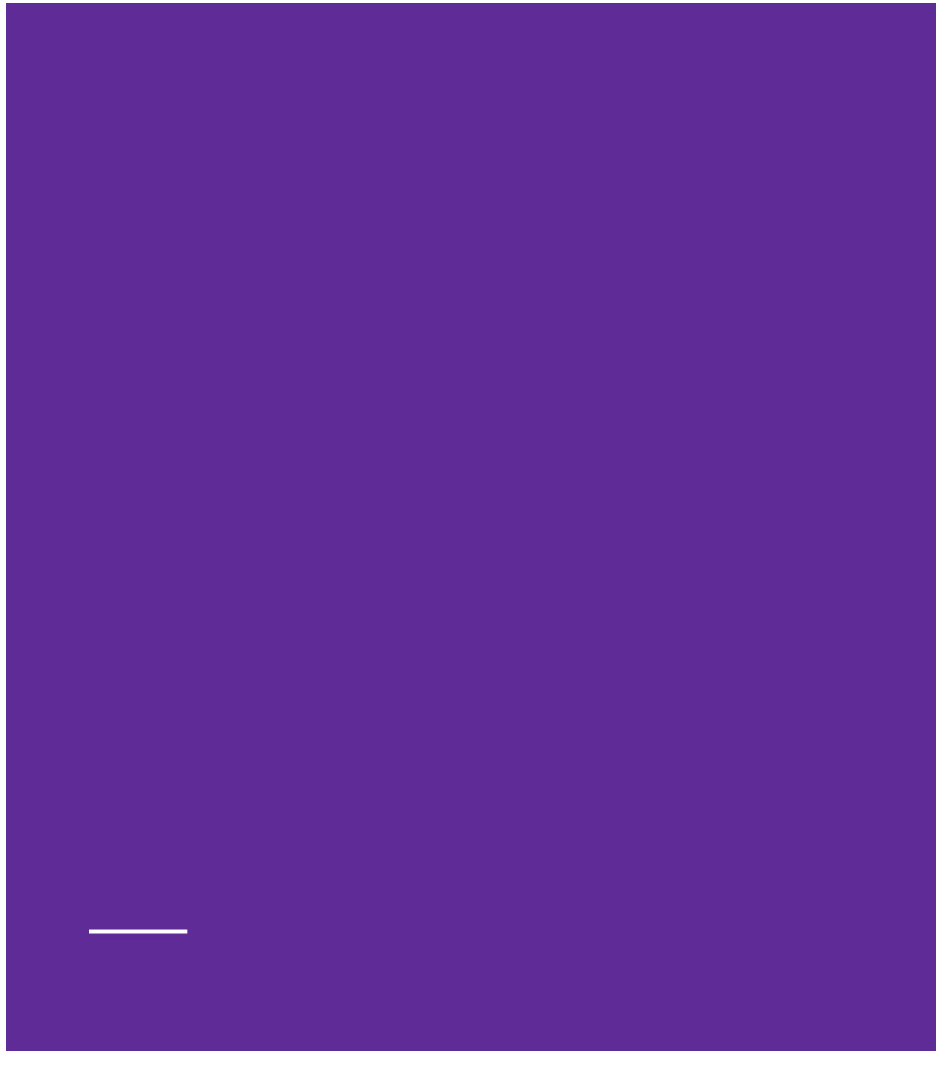

Speedups of gap-affine kernel
Ideal linear speedup
Kernel-only Speedup
Total time speedup
Speedup (relative to 32 DPUs)
16
8
4
2
1
32
64
127
255
509
Number of DPUs

Conclusions

- **Feasibility:** We demonstrated that multiple MMseqs2 stages can be mapped to a real PIM platform, including k-mer matching, ungapped alignment, and gap-affine alignment with PSSM support.
- **Scalability:** Kernel-side performance scales close to linearly with the number of DPUs, especially for the lighter stages, confirming that the underlying parallelism maps well to PIM.
- **Bottlenecks:** The main limitations are no longer in the kernels themselves, but in host-side preprocessing, data transfers, and memory pressure in the gap-affine stage.

Implication: Current PIM systems look most promising for k-mer-centric and database-stationary search. For state-heavy stages, insufficient WRAM capacity led to underutilization.

Thank you



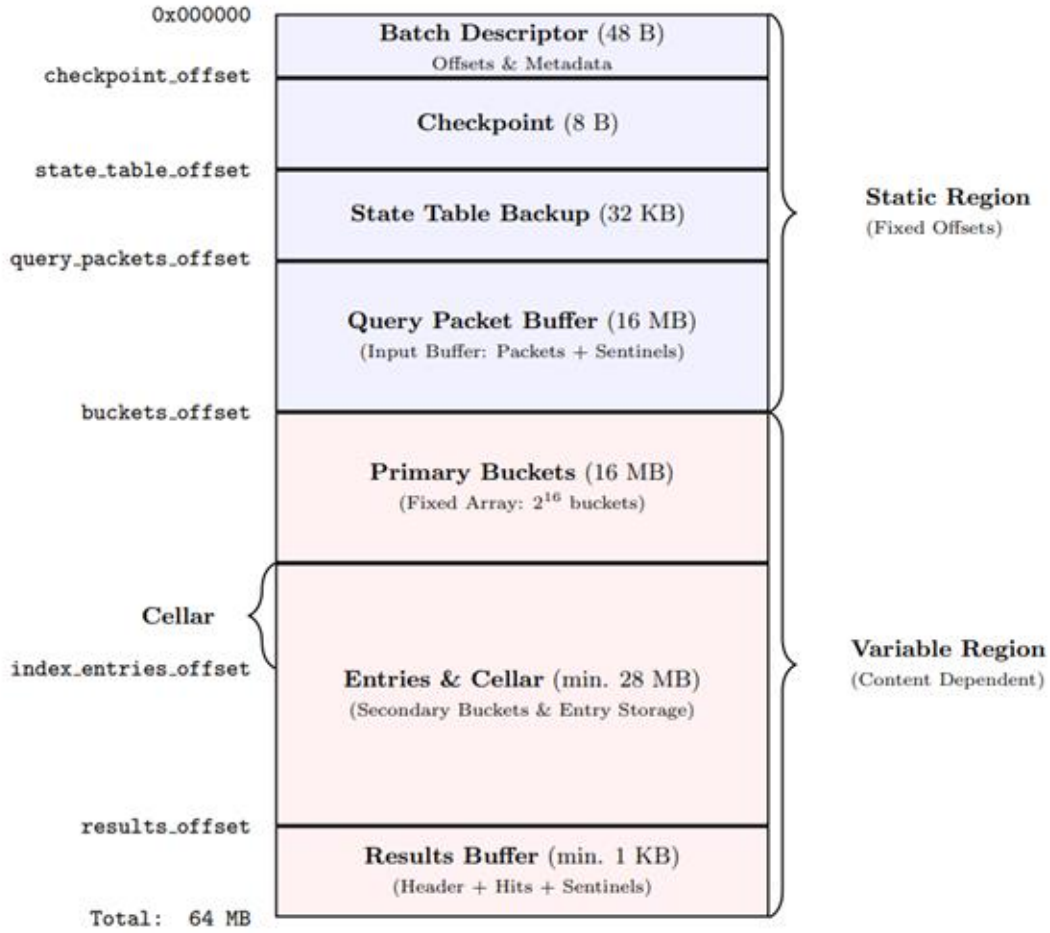
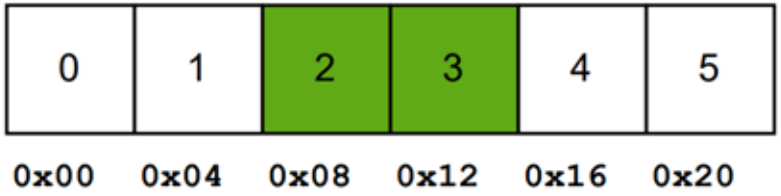
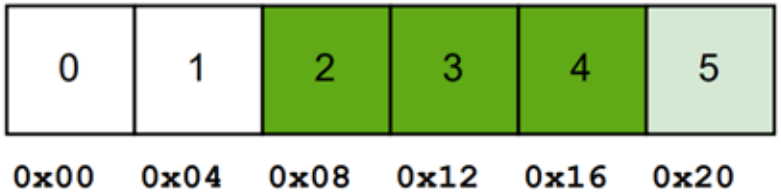


FIGURE 3.1: DPU MRAM Memory Layout. The layout is divided into a static region (red) and a variable region (blue)

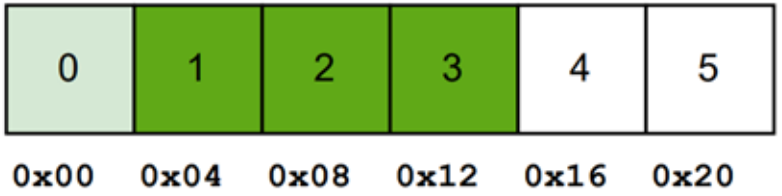
a) even offset, even count



b) even offset, odd count



c) odd offset, odd count



d) odd offset, even count

