Warp-centric K-Nearest Neighbor Graphs construction on GPU

Bruno H. Meyer, **Aurora Pozo**, **Wagner M. Nunan Zola**

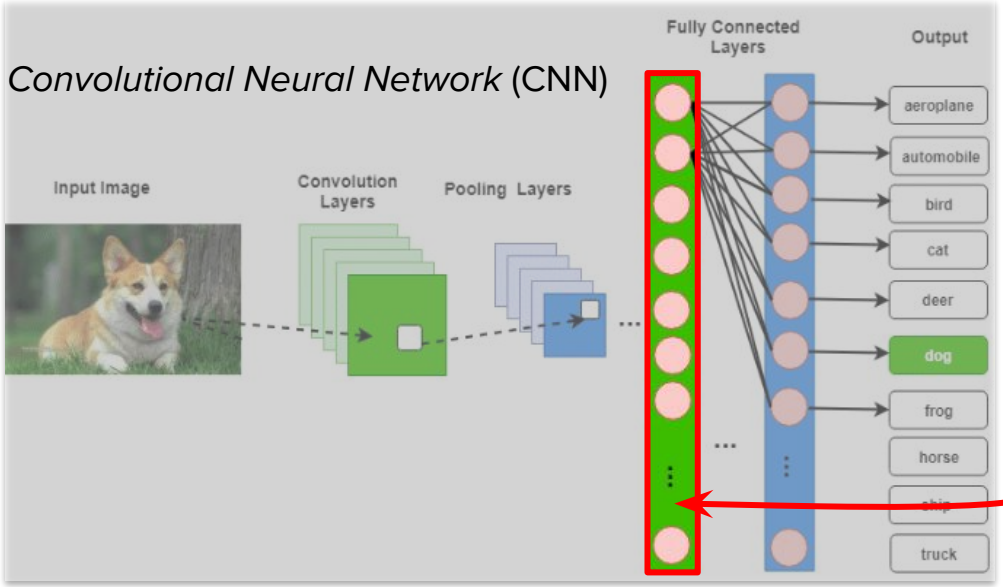
50th International Conference on Parallel Processing

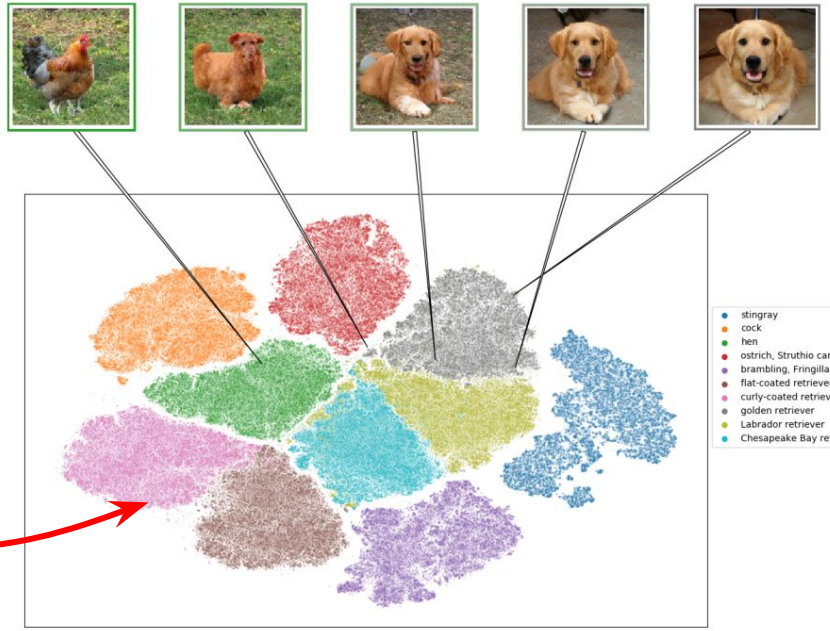
International Workshop on Deployment and Use of Accelerators (DUAC)

August 9-12, 2021

KNN Graphs (K-NNG) applications

- High-Dimensional points
- Required in other algorithms
- High computational cost





KNN Graph (K-NNG) and Approximate K-NNG

X : Points Set

$G(x) \mid x \in X$: Neighborhood of each point

$|G(x)| = K$

$$\max_{y \in G(x)}\left(||x-y||^2\right) \leq ||x-\bar{y}||^2 \mid \bar{y} \notin G(x)$$

Furthest Neighbor

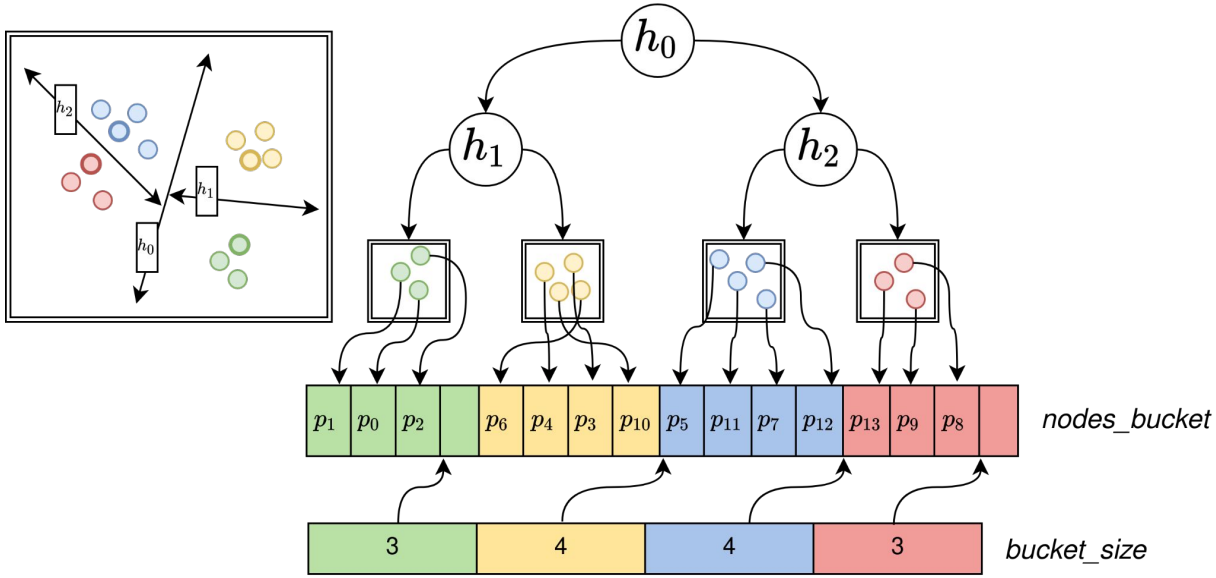
This can be unsatisfied for some neighbors in approximate KNN Graphs

Any point outside of G(x)

Objectives

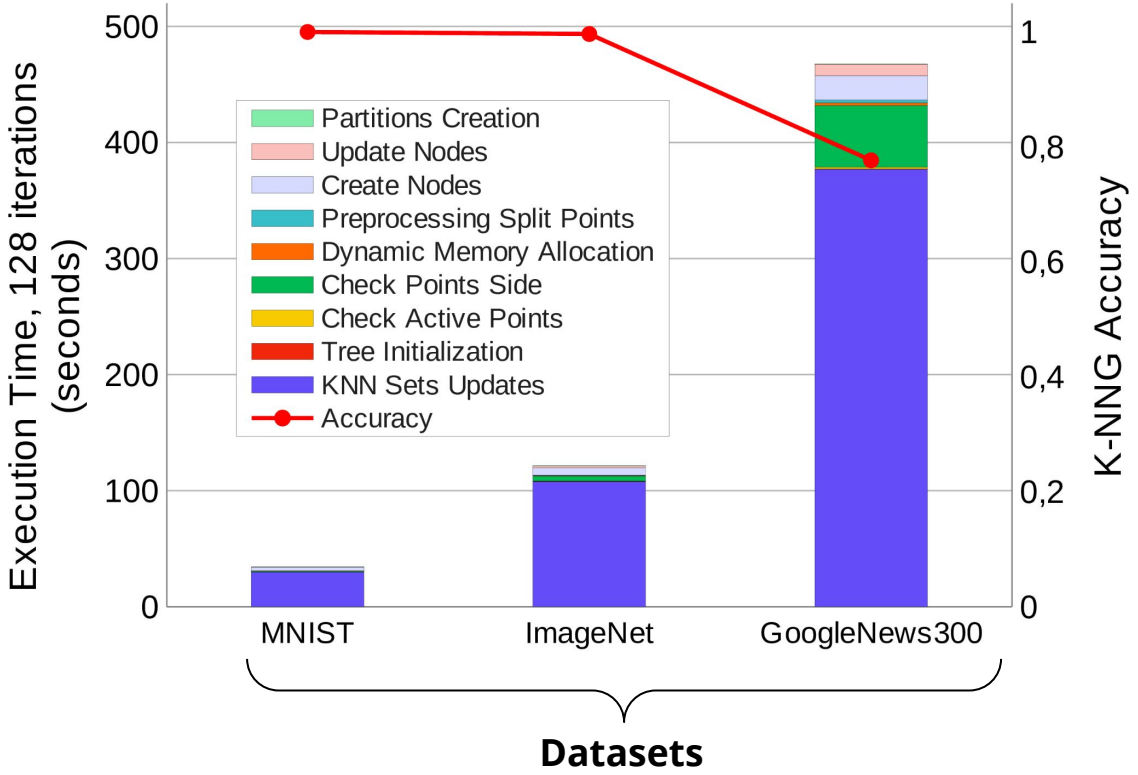
- Improve the RSFK (Random Sample Forest KNN) algorithm **in GPU**
    - Used to create **Approximate K-NNG**
- Identify bottlenecks
- Identify new approaches to optimize the algorithm

RSFK Tree Construction



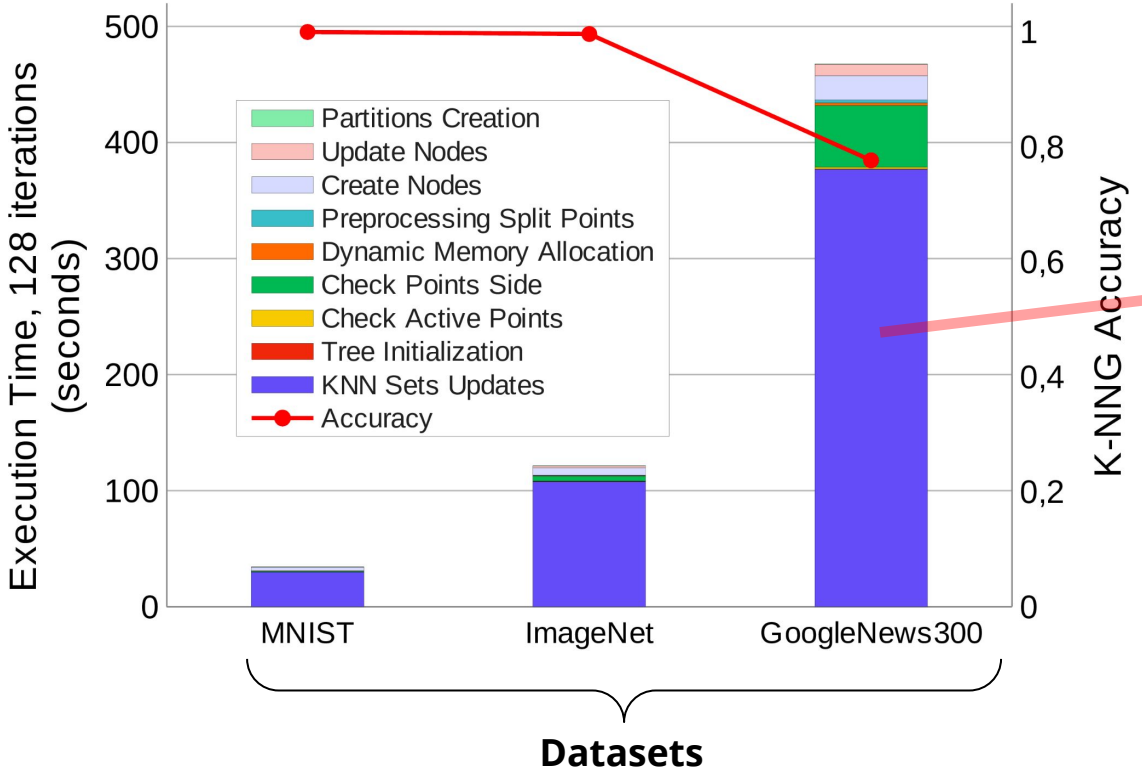
- One tree generated at a time
- Leaves control the trade-off between time and accuracy
- Executed only on GPU
- Methods to divide nodes:
  - Random Projection
  - **Random Sample (Ours)**

RSFK Bottleneck



- Measured the tree construction and K-NNG processing steps

RSFK Bottleneck



- Measured the tree construction and K-NNG processing steps
  - KNN Sets Updates is the main bottleneck
  - The larger the dataset, the more trees are needed to achieve good accuracy

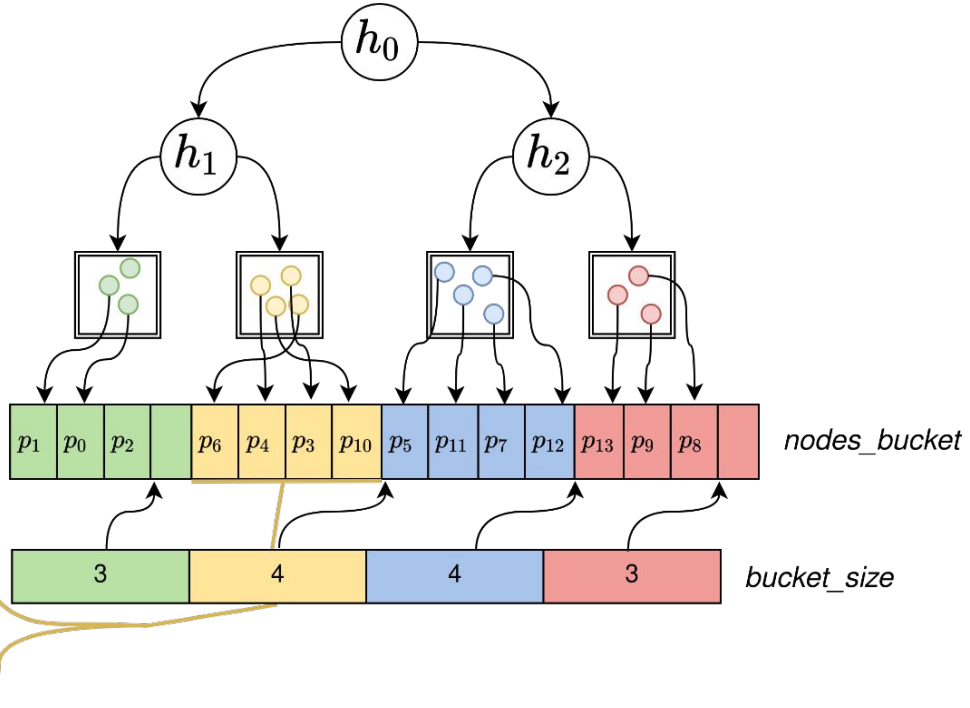
KNN Sets Update step

KNN Sets
(Index Matrix)

	K		
P_1	3	...	-1
P_2	3	...	6
P_3	-1	...	-1
P_4	6	...	-1
...	...	...	...
P_N	1	...	8

Virtual distance matrix

	p_6	p_4	p_3	p_{10}
p_6		Distance(p_6, p_4)	Distance(p_6, p_3)	Distance(p_1, p_{10})
p_4			Distance(p_4, p_3)	Distance(p_4, p_{10})
p_3				Distance(p_3, p_{10})
p_{10}				

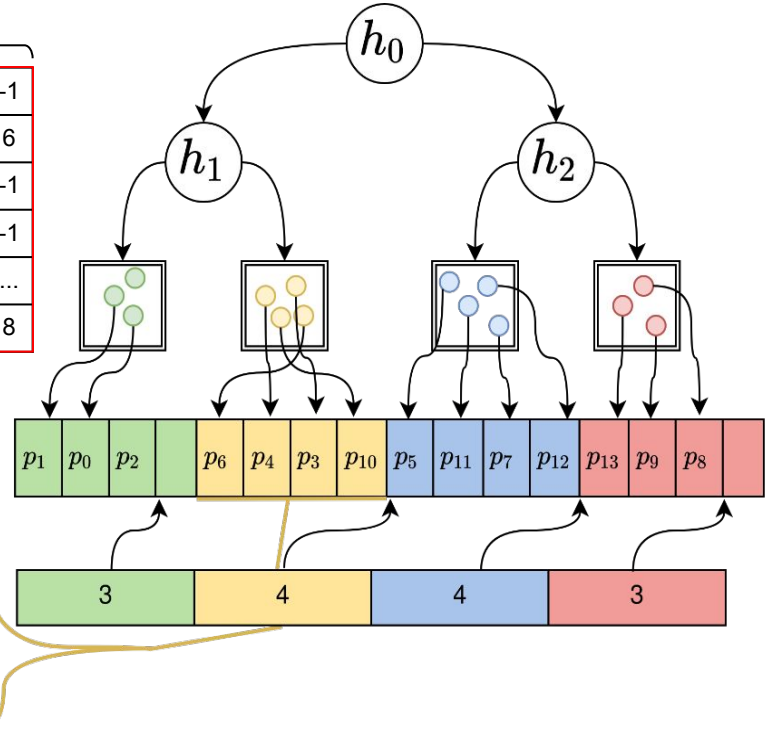


KNN Sets Update step

- ∀ element of the pair related to the Distance (p_A , p_B)

KNN Sets
(Index Matrix)

	K		
P_1	3	...	-1
P_2	3	...	6
P_3	-1	...	-1
P_4	6	...	-1
...	...	...	...
P_N	1	...	8



Virtual distance matrix

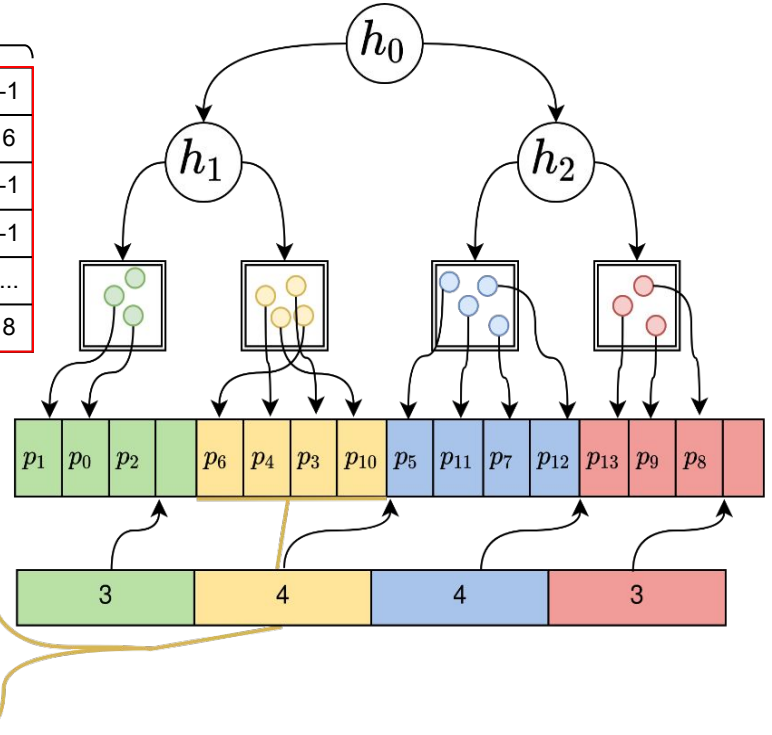
	p_6	p_4	p_3	p_{10}
p_6		Distance(p_6, p_4)	Distance(p_6, p_3)	Distance(p_1, p_{10})
p_4			Distance(p_4, p_3)	Distance(p_4, p_{10})
p_3				Distance(p_3, p_{10})
p_{10}				

KNN Sets Update step

- ∀ element of the pair related to the Distance (p_A, p_B)
  - Verify and update KNN Set of p_A if necessary

KNN Sets
(Index Matrix)

	K		
P_1	3	...	-1
P_2	3	...	6
P_3	-1	...	-1
P_4	6	...	-1
...	...	...	...
P_N	1	...	8



Virtual distance matrix

	p_6	p_4	p_3	p_{10}
p_6		Distance(p_6, p_4)	Distance(p_6, p_3)	Distance(p_1, p_{10})
p_4			Distance(p_4, p_3)	Distance(p_4, p_{10})
p_3				Distance(p_3, p_{10})
p_{10}				

KNN Sets Update step

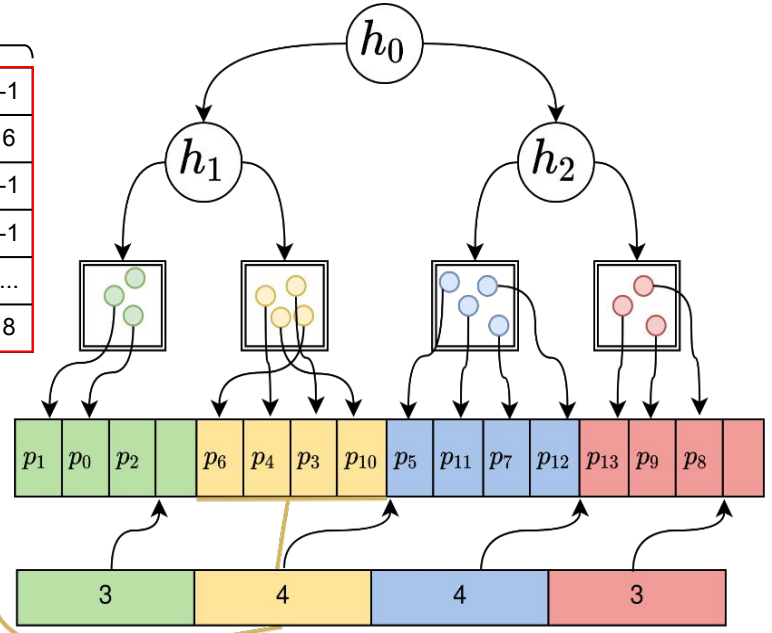
- ∀ element of the pair related to the Distance (p_A, p_B)
  - Verify and update KNN Set of p_A if necessary
  - Verify and update KNN Set of p_B if necessary

KNN Sets
(Index Matrix)

	K		
P_1	3	...	-1
P_2	3	...	6
P_3	-1	...	-1
P_4	6	...	-1
...	...	...	...
P_N	1	...	8

Virtual distance matrix

	p_6	p_4	p_3	p_{10}
p_6		Distance(p_6, p_4)	Distance(p_6, p_3)	Distance(p_1, p_{10})
p_4			Distance(p_4, p_3)	Distance(p_4, p_{10})
p_3				Distance(p_3, p_{10})
p_{10}				



Warp-centric GPU kernels

- GPU threads cooperate in groups:
  - Usually: all threads in the CTA (Cooperative Thread array) cooperate in a task

- Other possibilities are:
  - One task per thread
  - One task per warp (**warp-centric**)
    - Thread block has various active warps ⇒ many tasks in parallel
    - Warp threads cooperation:
      - All threads in the warp possibly cooperate using special warp primitives (SIMD approach)
      - Only one thread in the warp performs the task (SISD simulation)

- Algorithms can alternate these patterns in different phases
  - CTA cooperative → block cooperative → warp-centric

Example:
Reduction kernel
(transitions in phases)

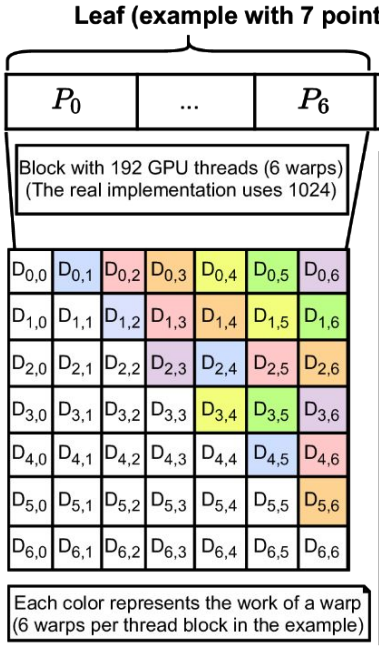
KNN Set Updates - Methods

- **A Warp-centric kernel:** All threads in a GPU warp cooperate in phases to:
  - Process a pair of KNN candidates
  - The computation of high dimensional distances
    - Better memory access pattern
    - Avoid warp divergence
  - Each thread block computes the KNN sets in a tree leaf
  - Avoid sorting operations and avoid maintaining heaps

- Three approaches considered:
  - **Atomic Approach**: based in atomic and lock operations
  - **Tiles Approach**: Divide the distances inside each leaf using a grid. Each tile of the grid is computed at time.
  - **Diagonal Approach**: Each diagonal above the main diagonal of the virtual distances matrix is computed at a time

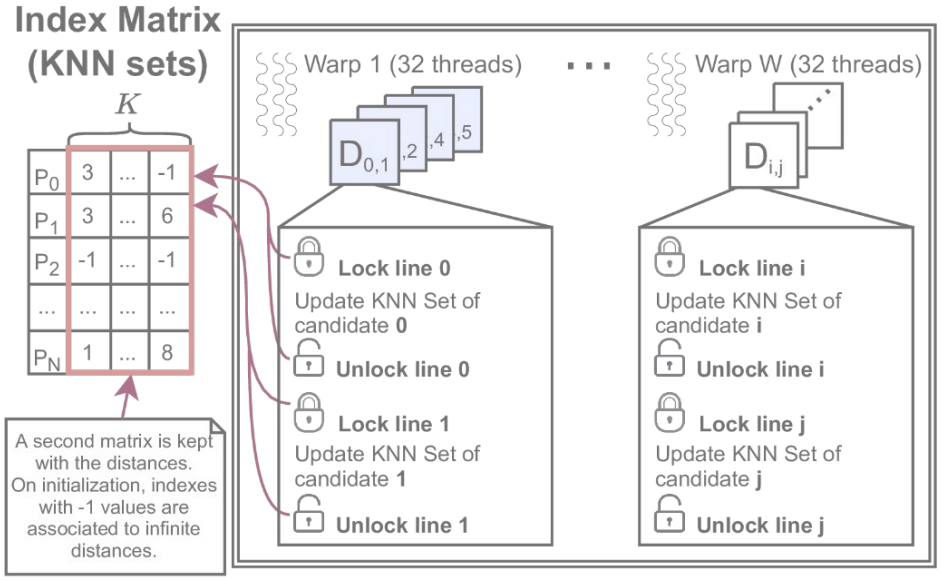
Avoid atomic and lock operations

Atomic Approach

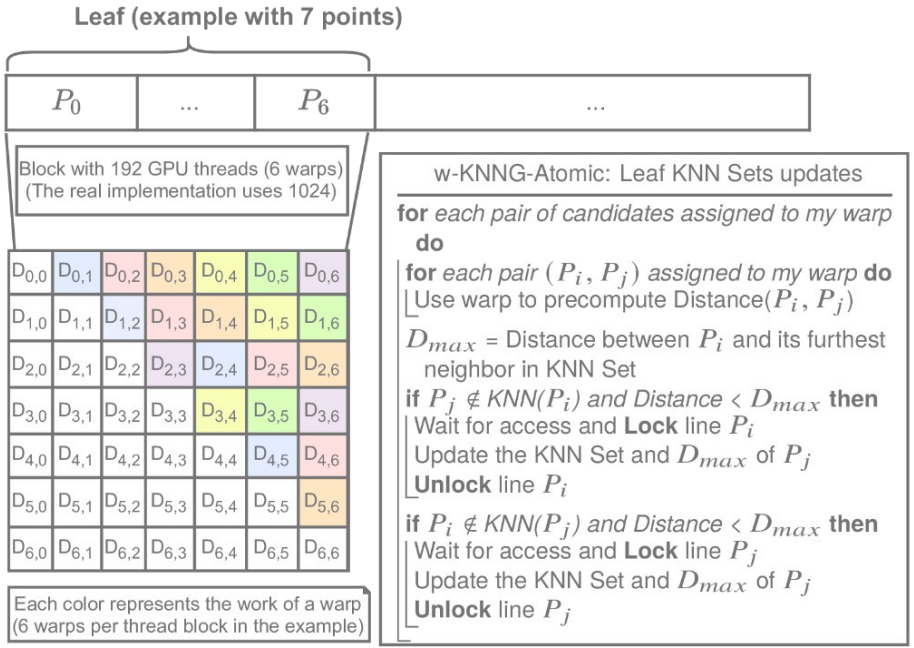


w-KNNG-Atomic: Leaf KNN Sets updates

for *each pair of candidates assigned to my warp* **do**
  for *each pair* (P_i, P_j) *assigned to my warp* **do**
    Use warp to precompute Distance(P_i, P_j)
  D_{max} = Distance between P_i and its furthest neighbor in KNN Set
  if $P_j \notin KNN(P_i)$ *and Distance* $< D_{max}$ **then**
    Wait for access and **Lock** line P_i
    Update the KNN Set and D_{max} of P_j
    Unlock line P_i
  if $P_i \notin KNN(P_j)$ *and Distance* $< D_{max}$ **then**
    Wait for access and **Lock** line P_j
    Update the KNN Set and D_{max} of P_j
    Unlock line P_j

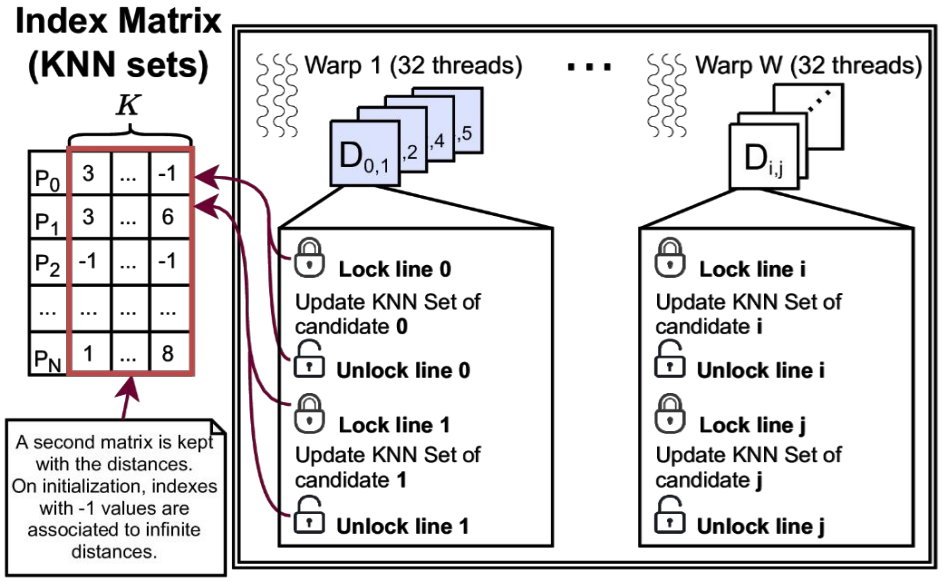


Atomic Approach

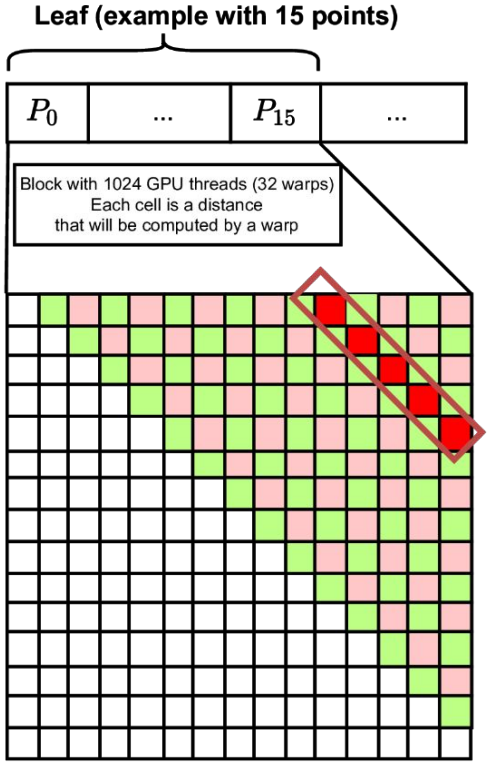


w-KNNG-Atomic: Leaf KNN Sets updates

for *each pair of candidates assigned to my warp* **do**
  for *each pair* (P_i, P_j) *assigned to my warp* **do**
    Use warp to precompute Distance(P_i, P_j)
    D_{max} = Distance between P_i and its furthest neighbor in KNN Set
    if $P_j \notin KNN(P_i)$ *and Distance* $< D_{max}$ **then**
      Wait for access and **Lock** line P_i
      Update the KNN Set and D_{max} of P_j
      Unlock line P_i
    if $P_i \notin KNN(P_j)$ *and Distance* $< D_{max}$ **then**
      Wait for access and **Lock** line P_j
      Update the KNN Set and D_{max} of P_j
      Unlock line P_j

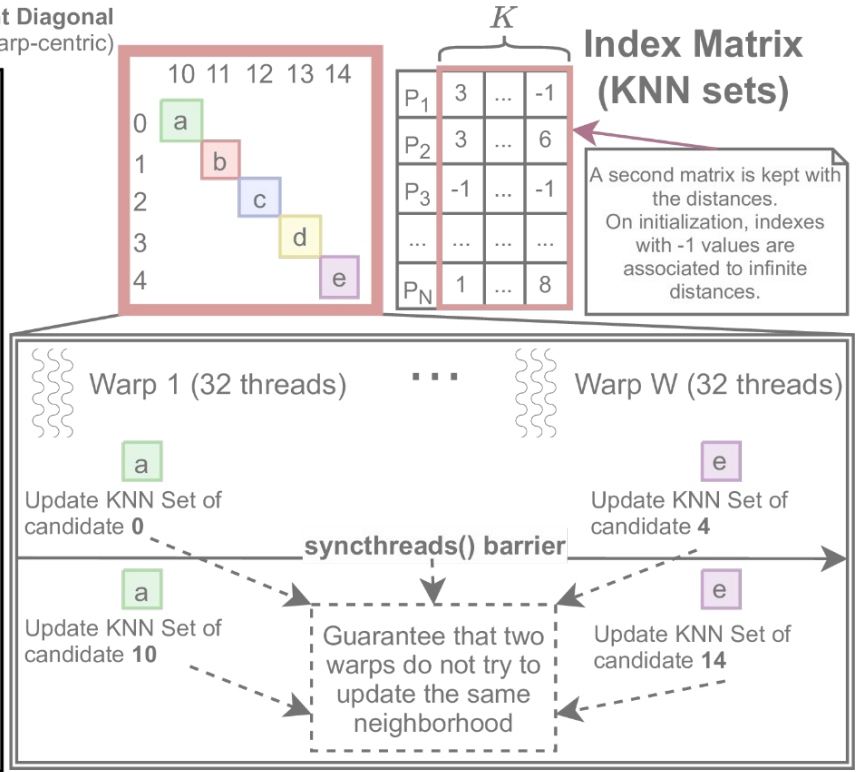


Diagonal Approach

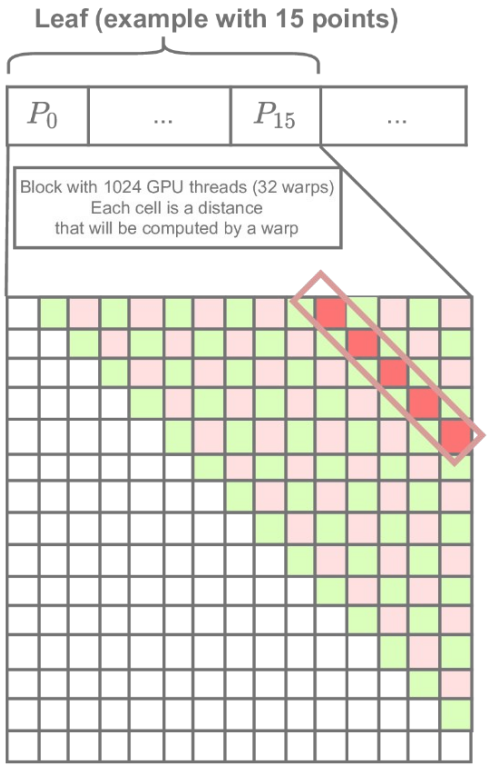


w-KNNG-Diagonal: Leaf KNN Sets updates

for *each Diagonal* **do**
  for *each pair* (P_i, P_j) *assigned to my warp* **do**
    Use warp to precompute Distance(P_i, P_j)
  syncthreads() // thread block barrier
  for *each pair* (P_i, P_j) *assigned to my warp* **do**
    Load Computed Distance(P_i, P_j)
    $P_i \leftarrow$ point related to the current line *i*
    D_{max} = Distance between P_i and its furthest neighbor in KNN Set
    if $P_j \notin KNN(P_i)$ *and Distance* $< D_{max}$ **then**
      Update the KNN Set and D_{max} of P_i
  syncthreads() // thread block barrier
  for *each pair* (P_i, P_j) *assigned to my warp* **do**
    Load Computed Distance(P_i, P_j)
    $P_j \leftarrow$ point related to the current column j
    D_{max} = Distance between P_j and its furthest neighbor in KNN Set
    if $P_i \notin KNN(P_j)$ *and Distance* $< D_{max}$ **then**
      Update the KNN Set and D_{max} of P_j
  syncthreads() // thread block barrier

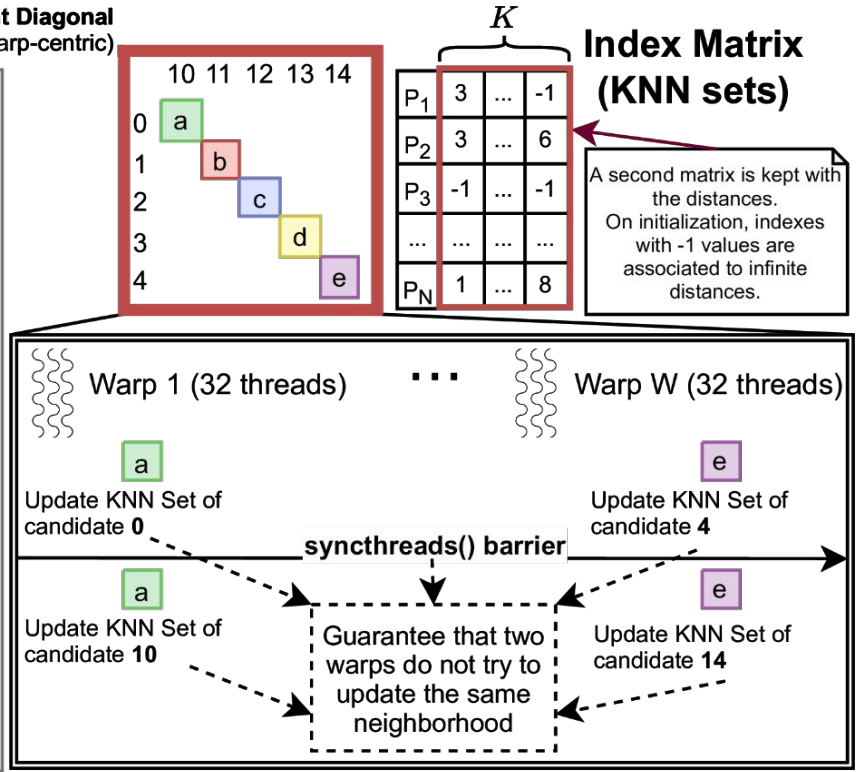


Diagonal Approach

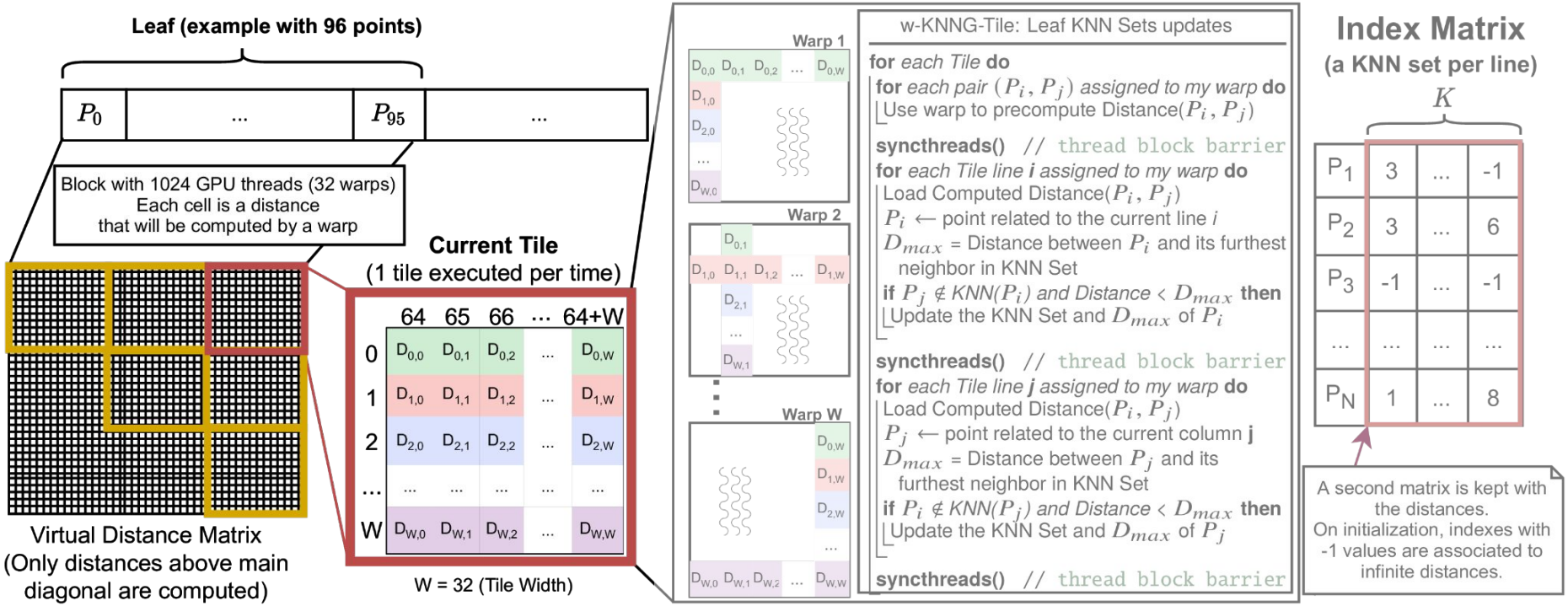


w-KNNG-Diagonal: Leaf KNN Sets updates

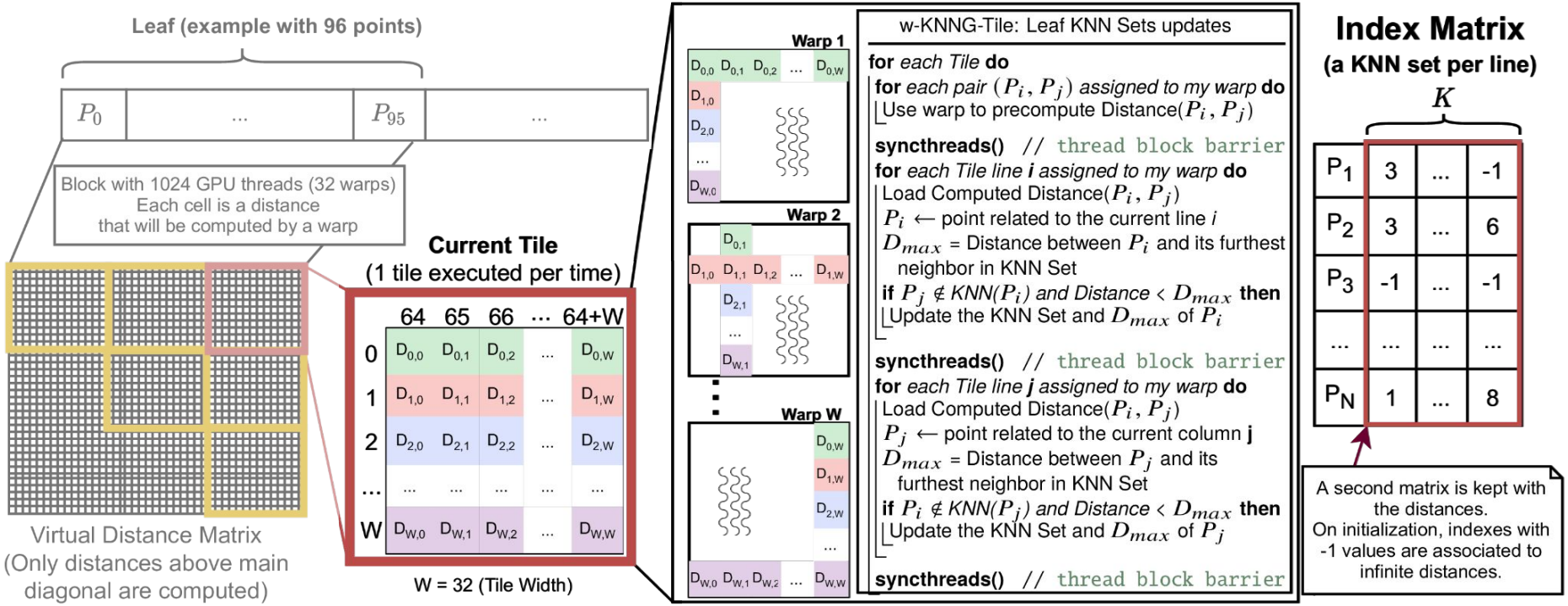
for *each Diagonal* **do**
  for *each pair* (P_i, P_j) *assigned to my warp* **do**
    Use warp to precompute Distance(P_i, P_j)
  syncthreads() // thread block barrier
  for *each pair* (P_i, P_j) *assigned to my warp* **do**
    Load Computed Distance(P_i, P_j)
    $P_i \leftarrow$ point related to the current line *i*
    D_{max} = Distance between P_i and its furthest neighbor in KNN Set
    if $P_j \notin KNN(P_i)$ *and Distance* $< D_{max}$ **then**
      Update the KNN Set and D_{max} of P_i
  syncthreads() // thread block barrier
  for *each pair* (P_i, P_j) *assigned to my warp* **do**
    Load Computed Distance(P_i, P_j)
    $P_j \leftarrow$ point related to the current column **j**
    D_{max} = Distance between P_j and its furthest neighbor in KNN Set
    if $P_i \notin KNN(P_j)$ *and Distance* $< D_{max}$ **then**
      Update the KNN Set and D_{max} of P_j
  syncthreads() // thread block barrier



Tiles Approach



Tiles Approach



Experiments

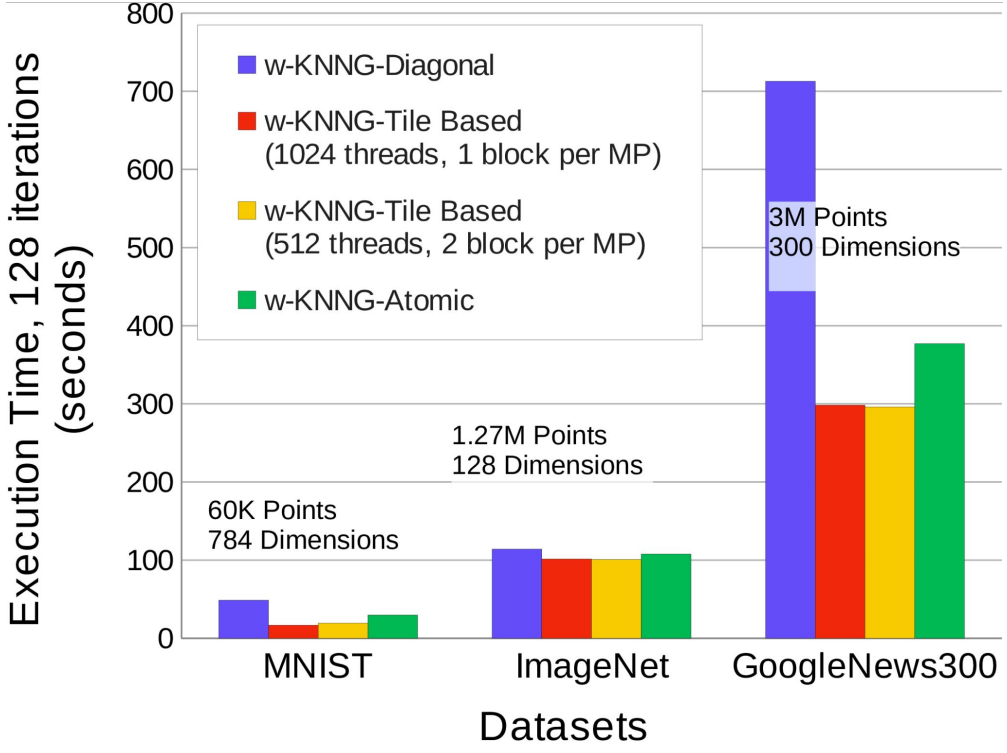
- Three datasets from real applications
  - **MNIST**: 70K Points, 784 Dimensions
  - **ImageNET**: 1.3M Points, 128 Dimensions
  - **GoogleNews300**: 3M Points, 300 Dimensions

- Artificial datasets
  - Generated using uniform distributions
  - Variation of different parameters: K, No. of points, No. of dimensions, Leaves sizes

- Comparison of methods:
  - **w-KNNG-Atomic** (Atomic Approach)
  - **w-KNNG-Diagonal** (Diagonal Approach)
  - **w-KNNG-Tile** (Tiles Approach)
  - Nearest Neighbor Exploration (**NNE**): Post-Processing method for RSFK
  - **FAISS Library** (popular library for similarity search with GPU)

- NVIDIA RTX 2070 GPU (Turing architecture)

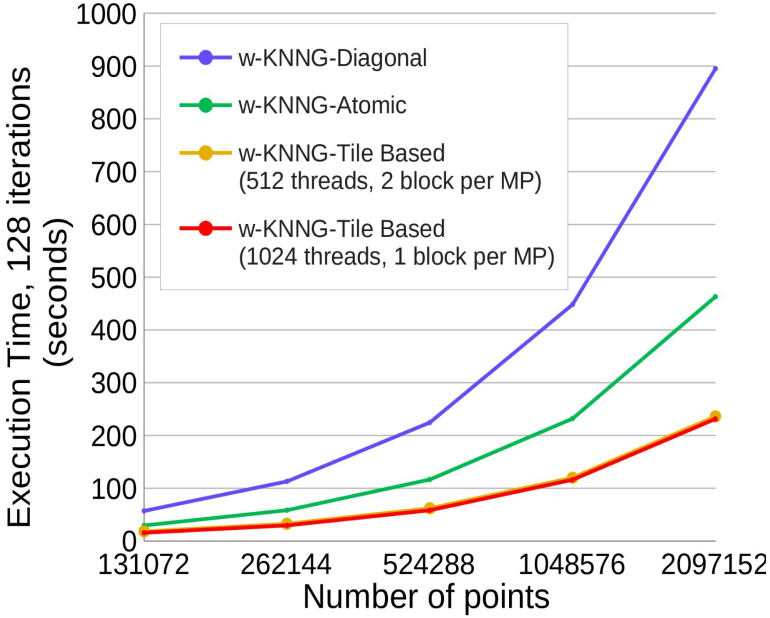
Results - Real Datasets

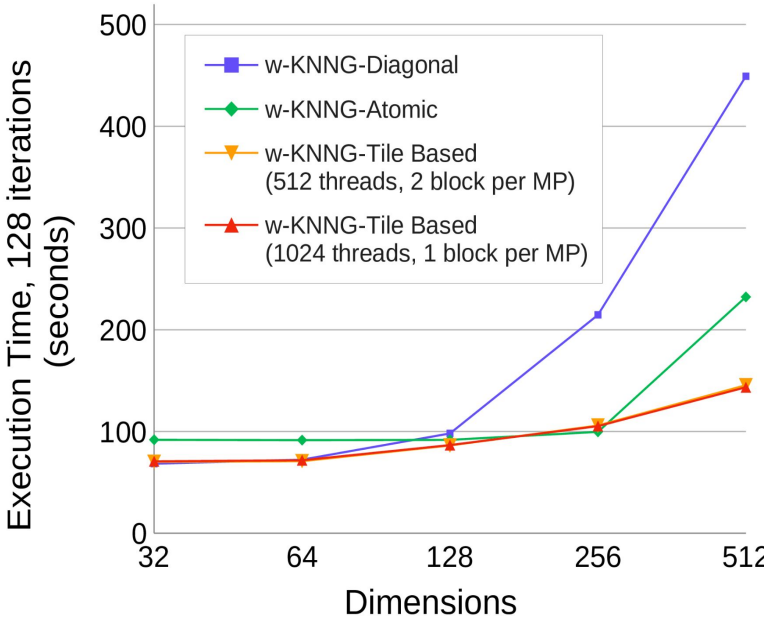


- Analysis of KNN Sets Updates step
- Tiles Approach was the faster method
- Size of tile have a small impact

Results - Scalability - No. of **Points** and **Dimensions**

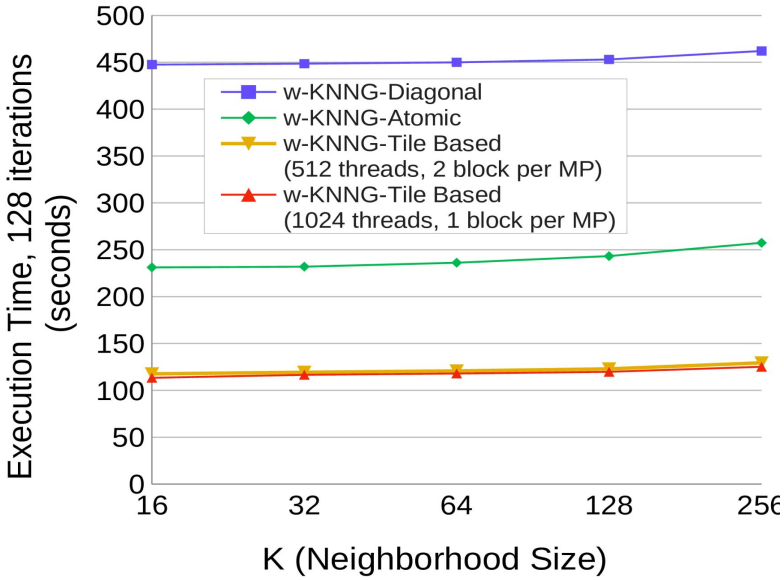
Execution time of KNN Sets Update step in the w-KNNG methods using artificially generated datasets. Each point in the charts represents an execution.

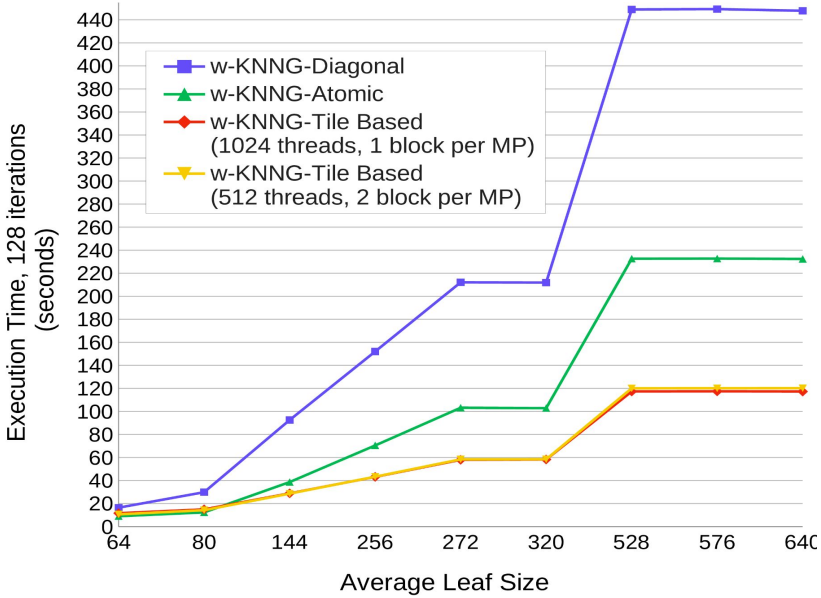




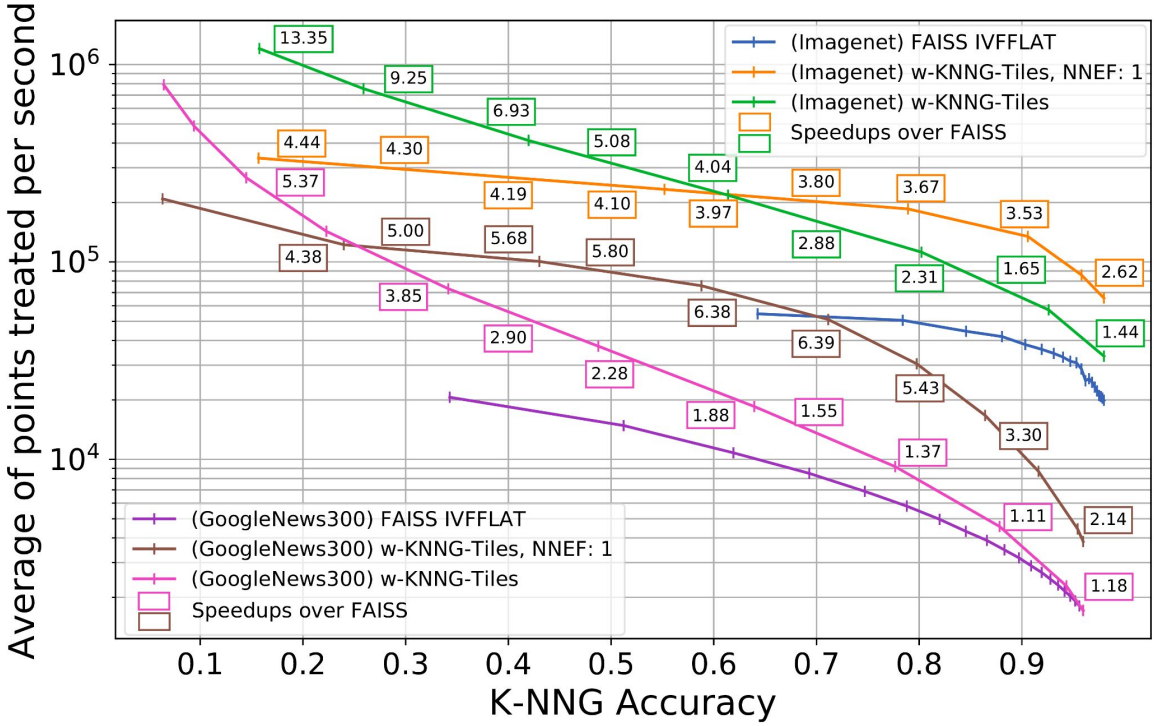
Results - Scalability - K and **Leaf Size**

Execution time of KNN Sets Update step in the w-KNNG methods using artificially generated datasets. Each point in the charts represents an execution.





Results - Comparison with FAISS in GPU



- Executions with different number of trees for RSFK and *nprobe* for FAISS (trade-off between time and quality)
- Tiles Approach was the best method
- Up to 639% speedup over FAISS library
- Use of Nearest Neighbor Exploring allows better speedups at higher values of accuracy

Conclusions

- We identified the bottleneck in the RSFK algorithm: Updates of KNN Sets
- Warp-centric method was mostly responsible for good performance in all kernels
  - For high dimensional data the high throughput of warp-centric calculations allows avoidance of sorting operations or maintaining heaps
- Problems caused by atomic and lock operations were mitigated by the w-KNNG Tiles and w-KNNG Diagonal kernels
- The proposed Tiles Approach was the best method
  - Better if used for high-dimensional datasets (more than 128 dimensions)
- Experiments suggest that the analyzed methods are compute-bound in some scenarios and memory-bound in others
  - Other analyses and profiling results in the main text

Future works

- Use multiple GPUs to execute RSFK
- Use hybrid methods to execute KNN Sets Updates
  - Choose kernel implementation based in the current step of the algorithm and data information
- Apply the proposed contribution to optimize other applications. Examples:
  - t-SNE (dimensionality reduction)
  - Spectral Clustering
- Verify the behavior of the algorithm in more GPU architectures

Contact

bruno.meyer@ufpr.br

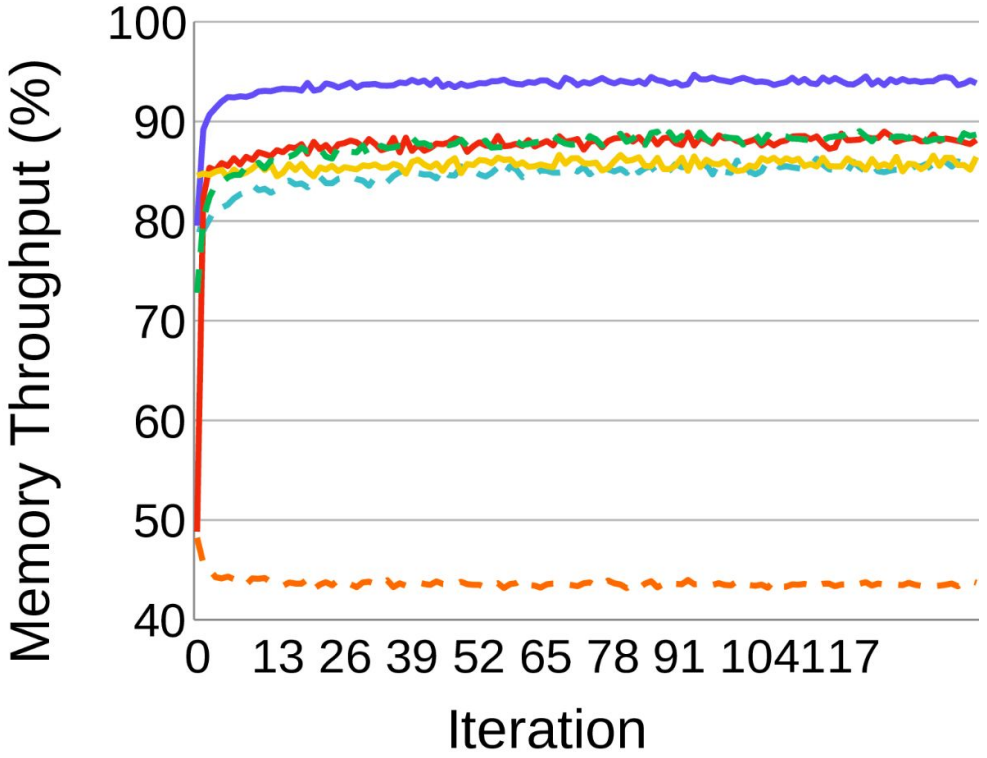
UFPR
UNIVERSIDADE FEDERAL DO PARANÁ

CNPq
Conselho Nacional de Desenvolvimento Científico e Tecnológico

Backup slides

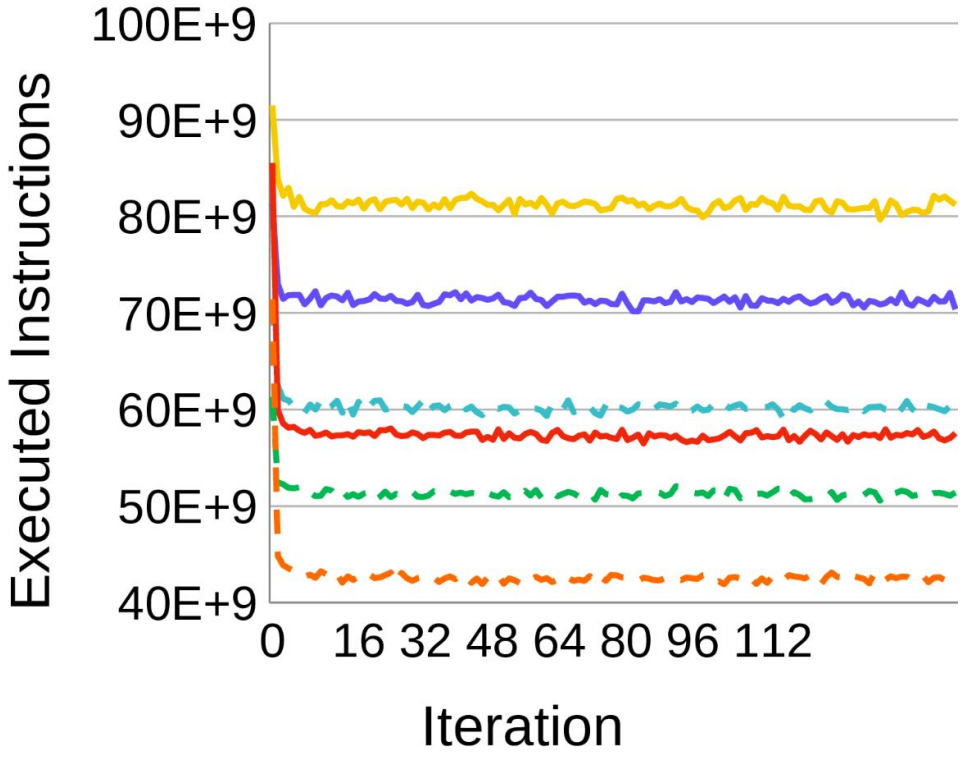
Backup slides

(a) Memory Throughput

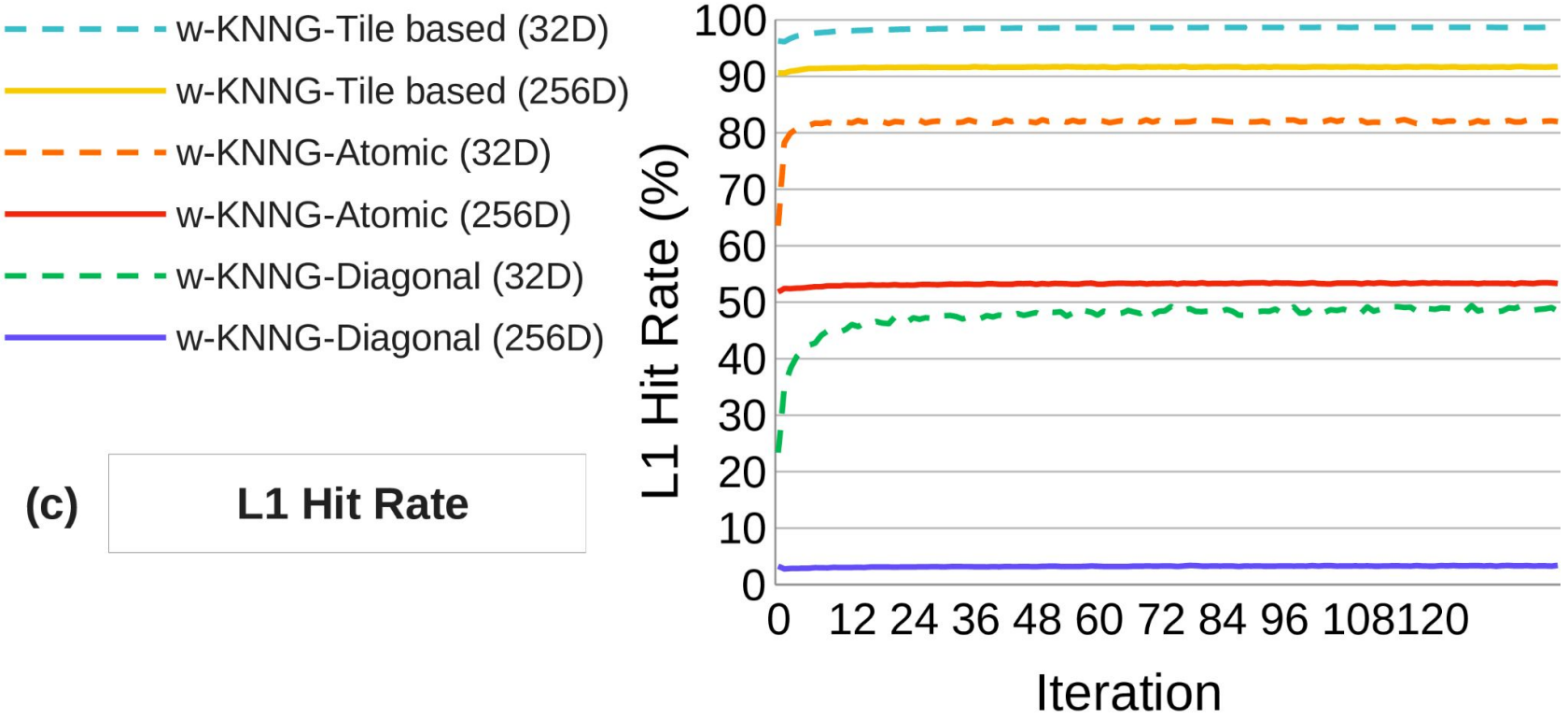


Backup slides

(b) Executed instructions

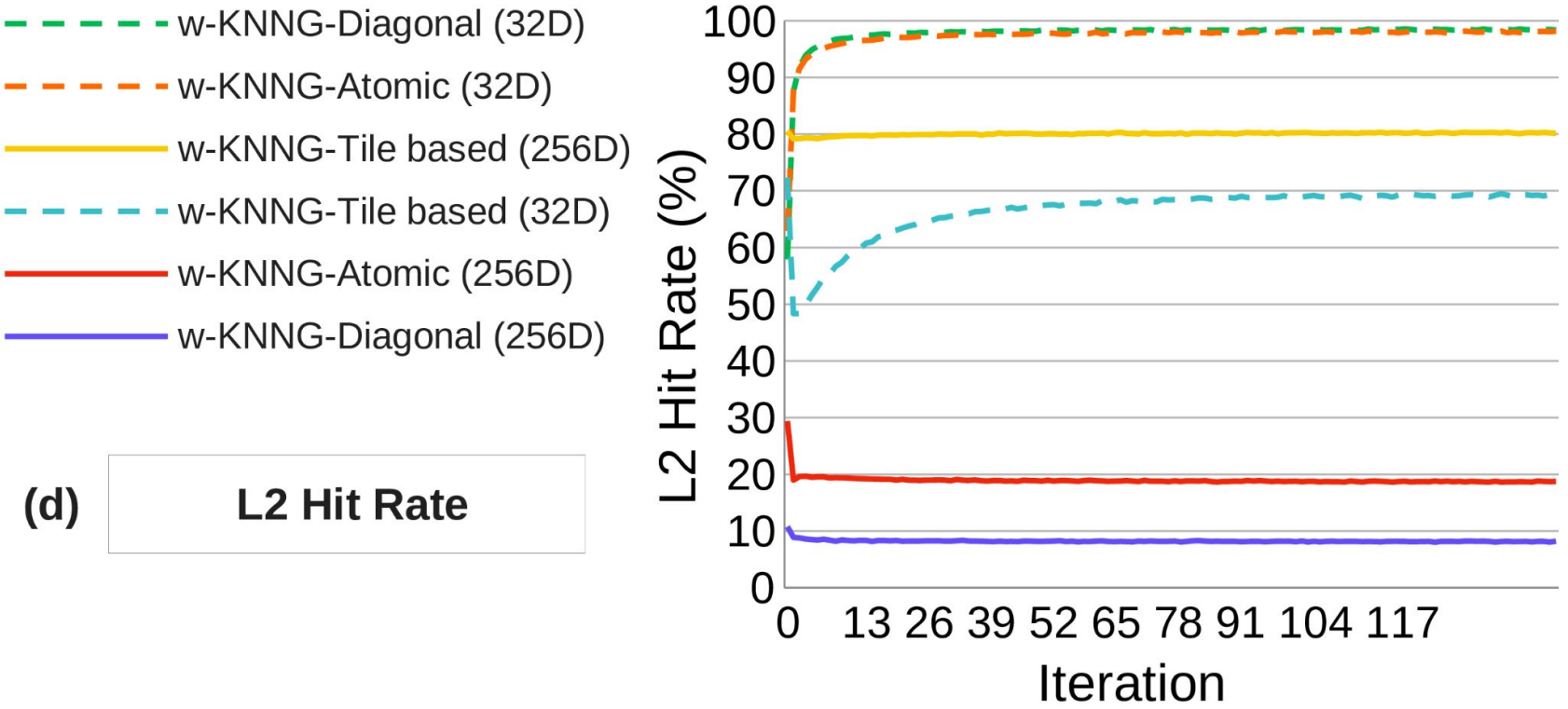


Backup slides



(c) **L1 Hit Rate**

Backup slides



(d) **L2 Hit Rate**

Backup slides