Institute of Computing Technology, Chinese Academy of Sciences
中国科学院计算技术研究所

APoX: Accelerate Graph-Based Deep Point Cloud Analysis via Adaptive Graph Construction

Lei Dai[1,2], Shengwen Liang[1,2,3], Ying Wang[4,2,3,5], Huawei Li[1,2,6], Xiaowei Li[1,2,3]

[1] SKLP, Institute of Computing Technology, CAS, Beijing, China

[2] University of Chinese Academy of Sciences, Beijing, China

[3] Zhongguancun National Laboratory, Beijing, China

[4] CICS, Institute of Computing Technology, CAS, Beijing, China

[5] Zhejiang Laboratory, Zhejiang, China; [6] Peng Cheng Laboratory, Shenzhen, China

Email: {dailei19z, liangshengwen, wangying2009, lihuawei, lxw}@ict.ac.cn

2024.1.22

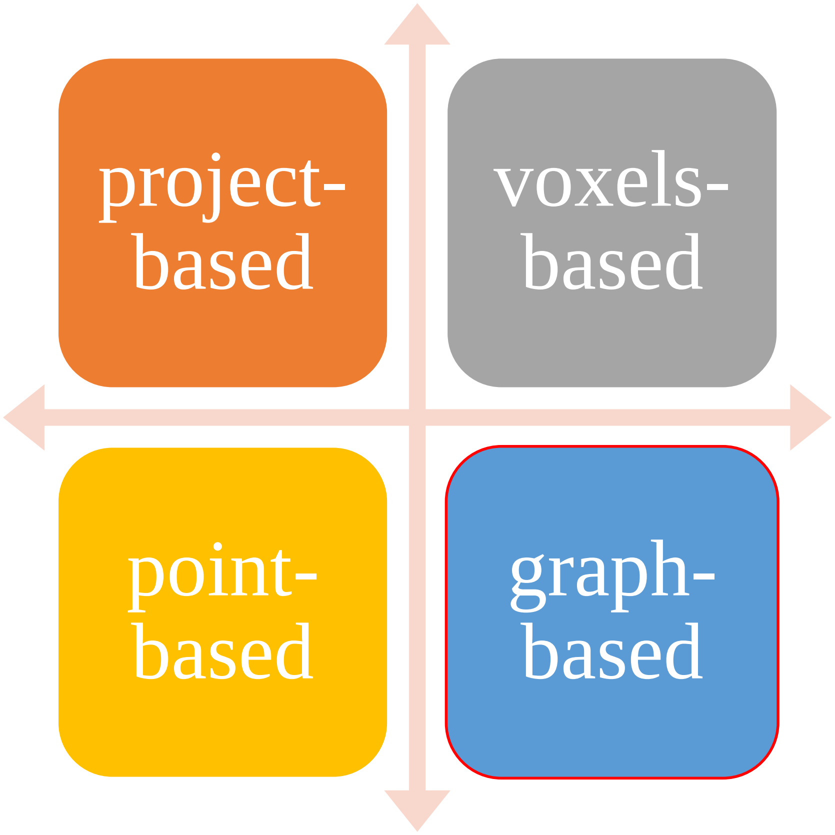
ASPDAC 2024

Outline

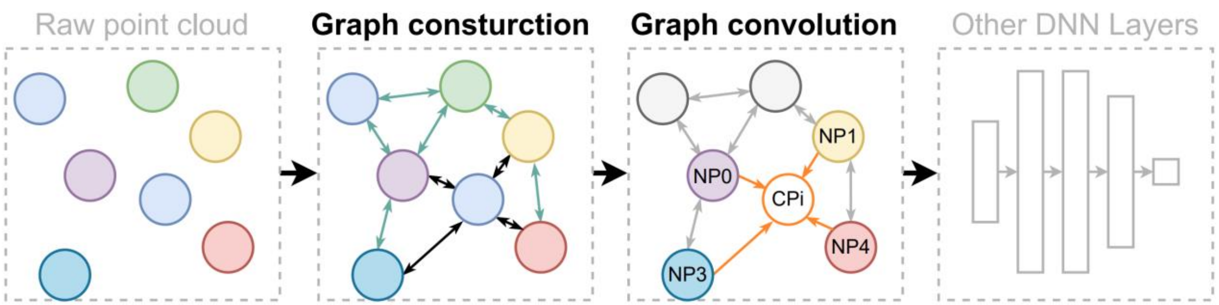
1. Backgrounds
2. Observation and Motivation
3. Architecture: APoX
4. Algorithm: ROAS
5. Evaluation

Deep Point Clouds Network

Classification of Deep Point Clouds



Graph-based Deep Point Cloud

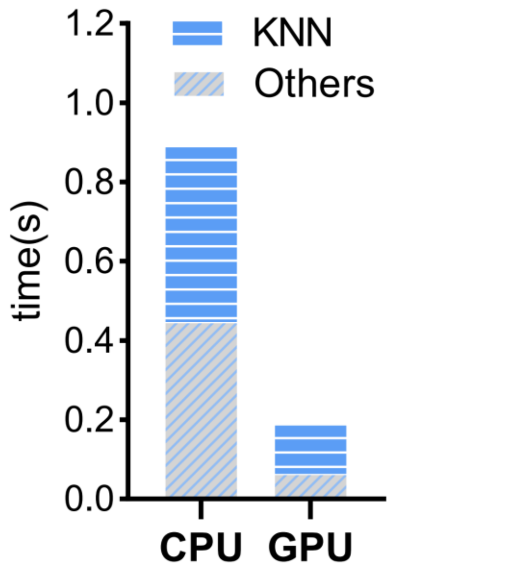


Methods of Graph Construction(GC)

- Exact method: BruteForce
- Tree based: KD-Tree
- Graph based: NN-Descent

Motivation

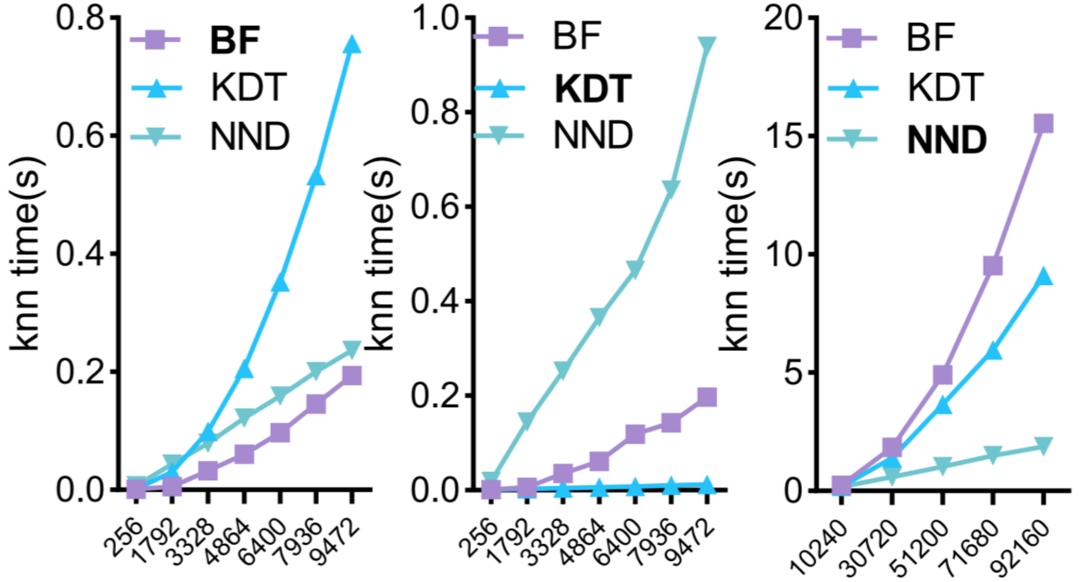
Graph Construction Time Ratio



(a) Inference time breakdown

- GC in CPU or GPU **bottleneck the execution by half** of the end-to-end execution time.

GC execution times under conditions



(b) K=16 D=160 (c) K=64 D=3 (d) K=12 D=16

- Different GC algorithms vary greatly in performance under different **number of neighbors (K), data dimensions (D), and size of point clouds(N)**

Analysis of GC algorithms

Theory analysis of algorithms

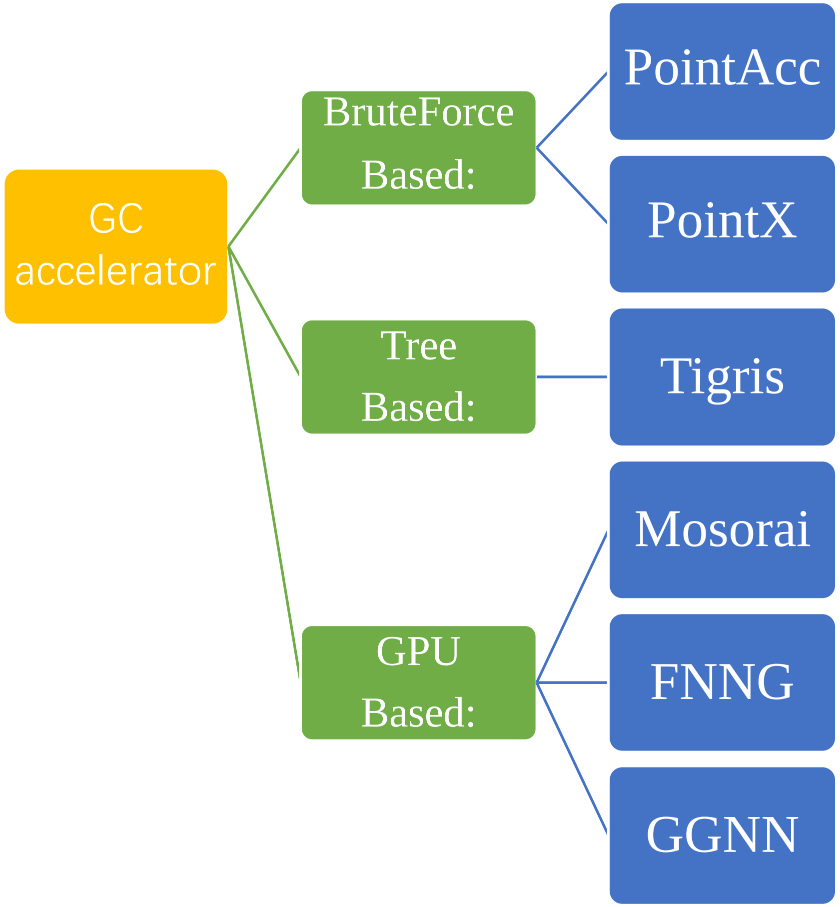
	Computational Complexity	Memory Access Pattern	DRAM Access Quantity
BrouteForce	$O(N*(N*D+NlogN))$	Sequential	$O(N*N*D)/Reuse$
NND-Based	$O(N^{1.14}*D*K)$	Semi-Random Indirect	$O(N^{1.14}*D*K)/Reuse$
KD-Tree-Based	$O(D*N*log_2 N)$	Sequential(Build)Indirect(Search)	$O(N*D*log_2 N)/Reuse$

N:point number of PC; D:dimension of PC; K:number of neighbors; $Reuse$:on-chip data reuse ratio(BF$>$ KD$>$NND).

Suitable scenarios of each algorithm

- **BruteForce:** **high dimension + middle point number**
- **NN-Descent:** **high dimension + high point number**
- **KD-Tree:** **low dimension**

GC in Existing Deep Point Cloud Accelerators



× Dedicated to one algorithm! Lack of input awareness!

➢ An accelerator supporting adaptive graph construction is required. But how?

Challenges:

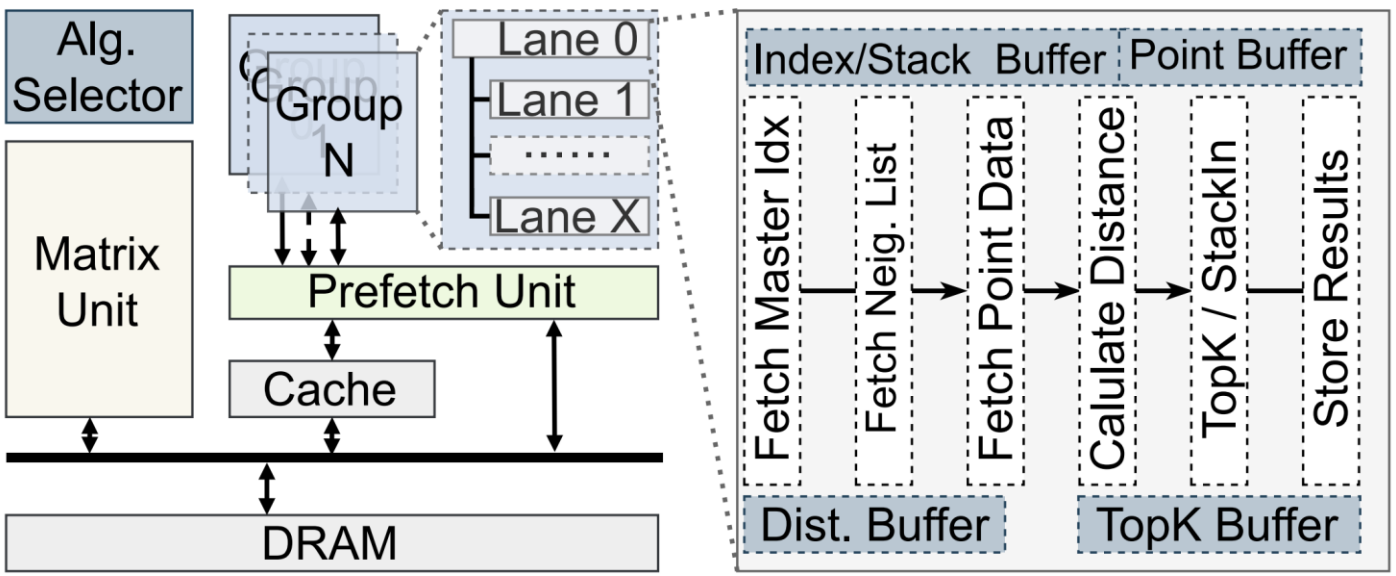
Q1: how to accurately **determine the most optimum GC approach** on the fly with minimum overhead?

Q2: how to build **a unified architecture** to support all three algorithms with distinct data structures, data flows, computations, and memory accesses?

Q3: how to **eliminate redundant** computations and redundant memory accesses?

=> The APoX

APoX Architecture Overview



Pipeline Stages

- Fetch Master Index
- Fetch Neighbor List
- Fetch Point
- Calculate Distances
- Update TopK /Push Stack
- Store Results

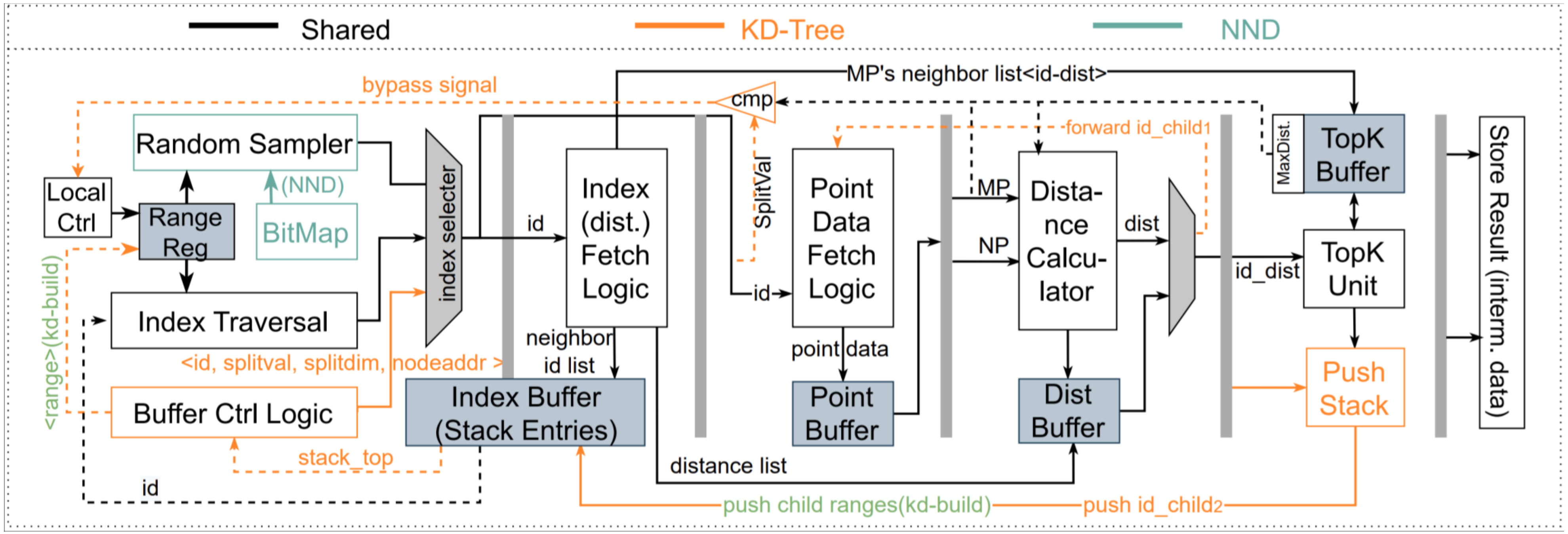
Design Targets

1. Supporting 3 GC approaches
2. Parallelism under different algorithms
3. Computation and buffer reusing
4. Choose the best approach on the fly
5. Redundancy elimination

Major Components

- Variable KNN Unit: Multi-groups and multi-lane in each group
- Matrix Unit: Handle Conv and Linear layers
- Alg. Selector: Select the best algorithm, configure VKU
- Cache and Prefetch Unit: Shared by multi lanes.

Variable KNN Unit



- Fetch Master Index
- Fetch Neighbor List
- Fetch Point
- Calculate Distances
- TopK /Push Stack
- Store Results

Resource Reuse

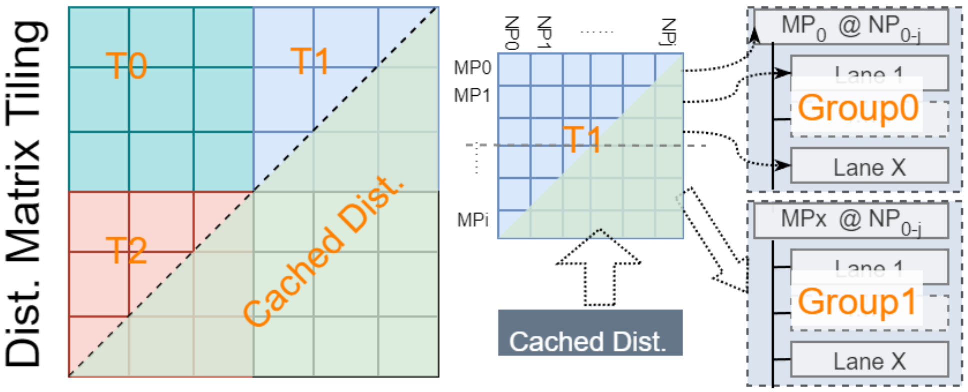
- ✓ TopK Buffer, Dist Buffer, Point Buffer are fully reused.
- ✓ Index Buffer stores both stack entries in KD-Tree and neighbor lists in others.
- ✓ Distance Calculator, TopK Unit are fully reused.

Redundancy Elimination

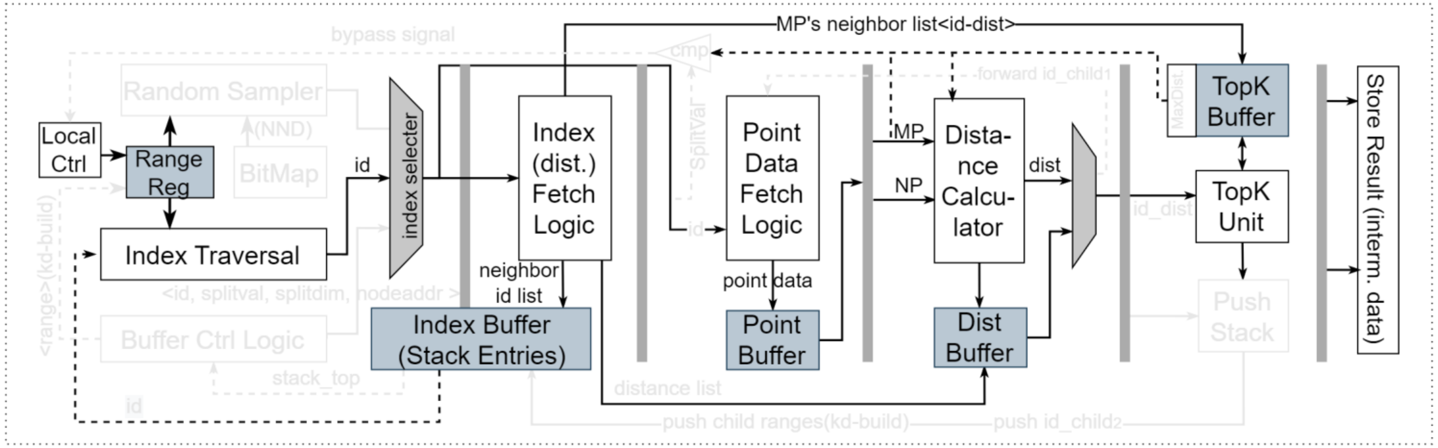
- ✓ Batch execution
- ✓ Early termination
- ✓ Caching reusable distance

Workflow of BruteForce

Mapping of BruteForce



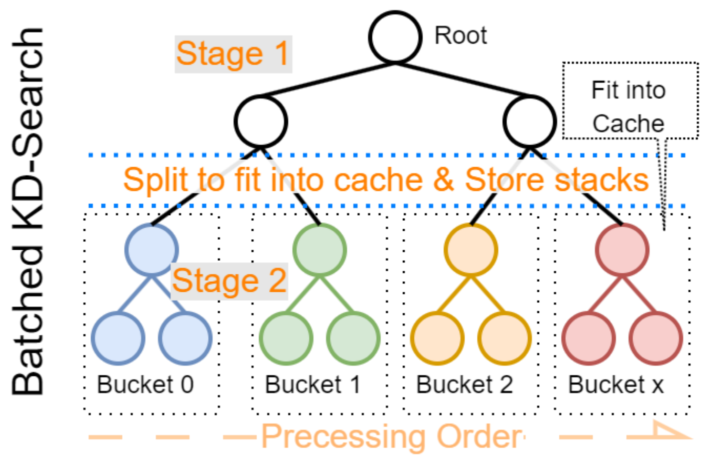
Dataflow of BruteForce



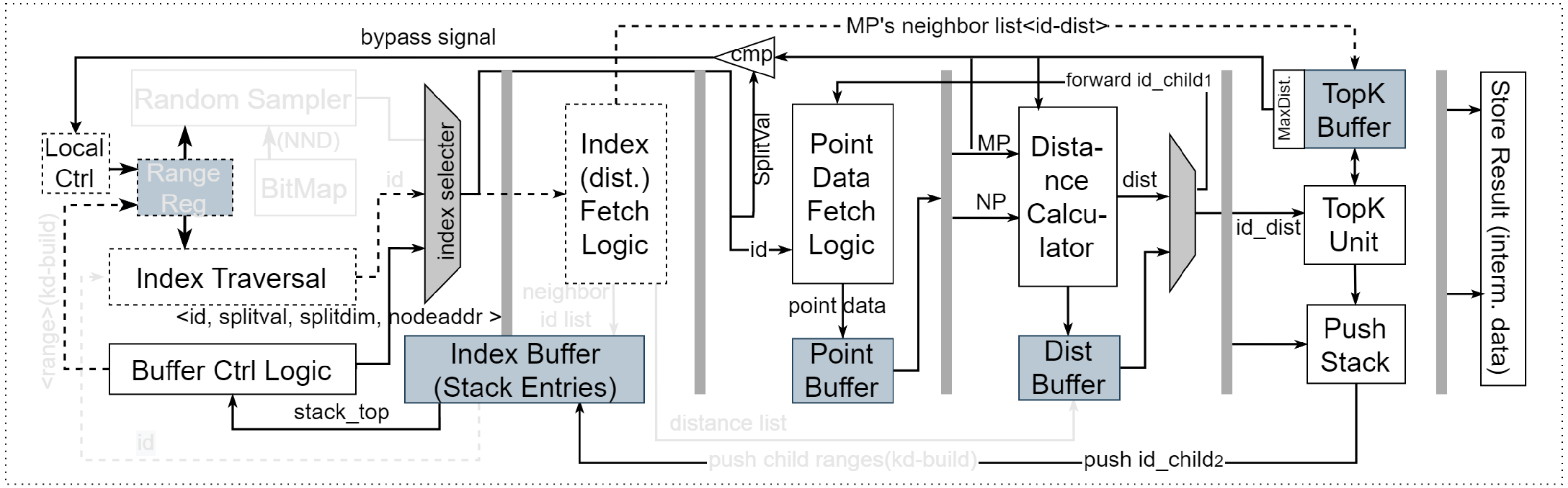
- ➢ The all-to-all distance calculation matrix is tiled
- ➢ Reusable distances are collected, cached and load when needed
- ➢ Each lane handles one MP in a tile, NP is broadcast to all lanes
- ➢ Basic resources are enabled

Workflow of KD-Tree

Mapping of KD-Tree



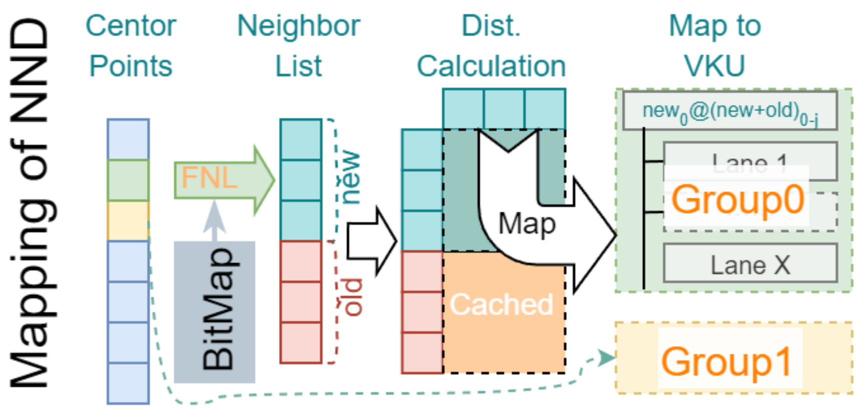
Dataflow of KD-Tree



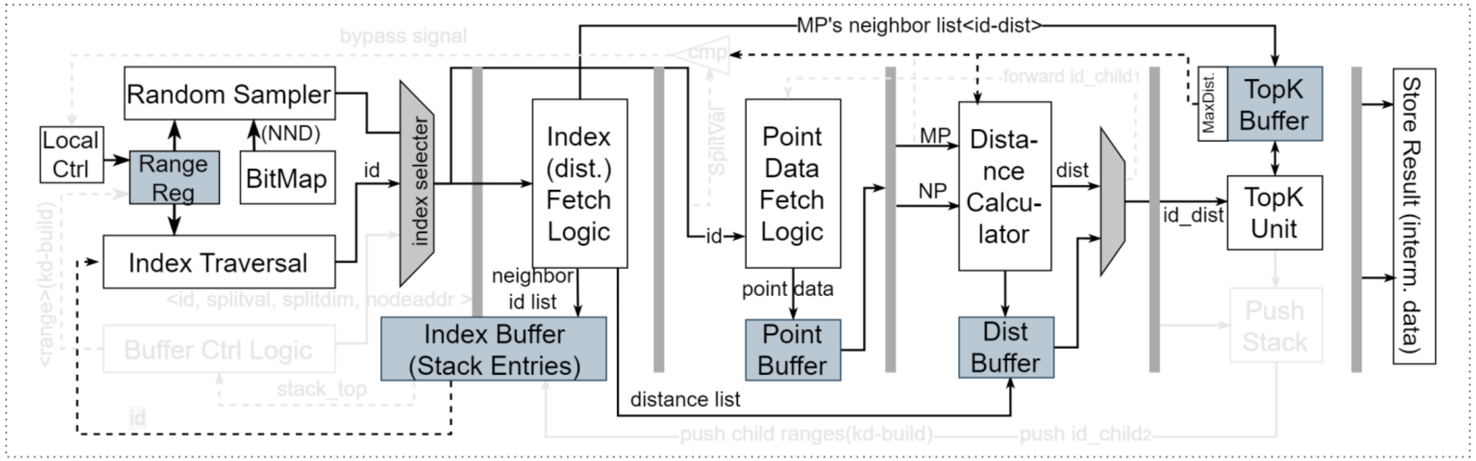
- ➢ The KD-Tree is split vertically to make sure subtrees can fit into on chip memory
- ➢ The stacks are split, stored, and read for batched execution
- ➢ Each lane handles a MP
- ➢ BufferCtrlLogic is configured to stack mode, PushStack logic is enbled, forward logic and bypass logic are enabled

Workflow of NN-Descent

Mapping of NN-Descent



Dataflow of NN-Descent



- ➢ The BitMap is enabled to mark new and old set in NN-Descent
- ➢ Processing of each center point is similar to BruteForce, but each matrix [new @(new+old)] is smaller.
- ➢ Each group handles a center point

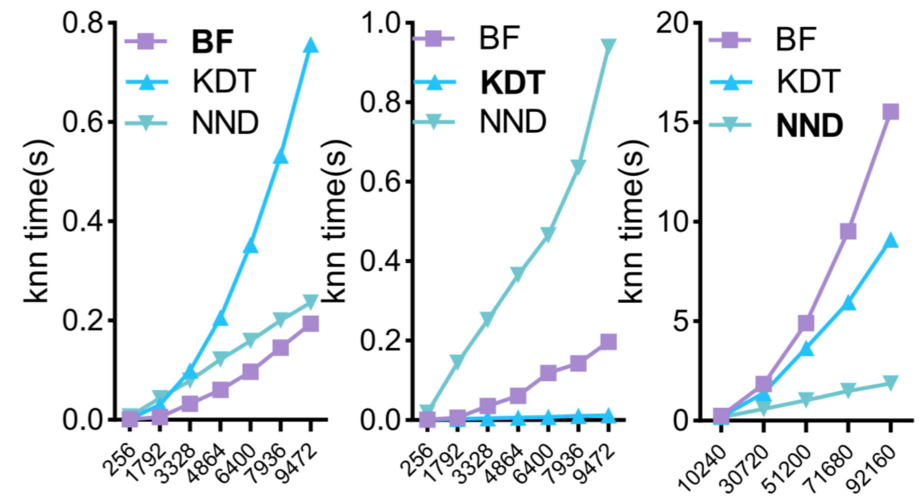
How to select the best GC approach?

Theory analysis of GC

	Computational Complexity	Memory Access Pattern	DRAM Access Quantity
BrouteForce	$O(N*(N*D+NlogN))$	Sequential	$O(N*N*D)/Reuse$
NND-Based	$O(N^{1.14}*D*K)$	Semi-Random Indirect	$O(N^{1.14}*D*K)/Reuse$
KD-Tree-Based	$O(D*N*log_2N)$	Sequential(Build)Indirect(Search)	$O(N*D*log_2N)/Reuse$

N:point number of PC; D:dimension of PC; K:number of neighbors; $Reuse$:on-chip data reuse ratio(BF> KD>NND).

GC execution time under conditions



down (b) K=16 D=160 (c) K=64 D=3 (d) K=12 D=16

GC approaches act differently under different conditions, but the trend of each GC can be fitted.

Algorithm Selector: ROAS

Algorithm Design

- Inputs:
  - N: Number of points
  - D: Dimension of points
  - K: The K value of TopK
- Output:
  - The best algorithm
- Design Idea
  - The running time of the three algorithms was collected offline
  - Three regression models are trained separately.
  - The algorithm with the lowest prediction time is taken at runtime

Algorithm 1: ROAS

```
Input: N, D, K
Output: optimal_alg
// Offline one time model fitting
Init(models =Regressor[3])
for alg in ALGs do
    for dim,k,size in target_ranges do
        runtime ← Collect_runtime(alg, dim, k, size)
        cond_time[dim,k,size].append(runtime)
    models[alg].Fit(cond_time)
save(models)
// Online select
pred_times ← Predict(models, N, D, K )
optimal_alg ← ALGs[arg_min(pred_times)]
return optimal_alg
```

Experimental Setup

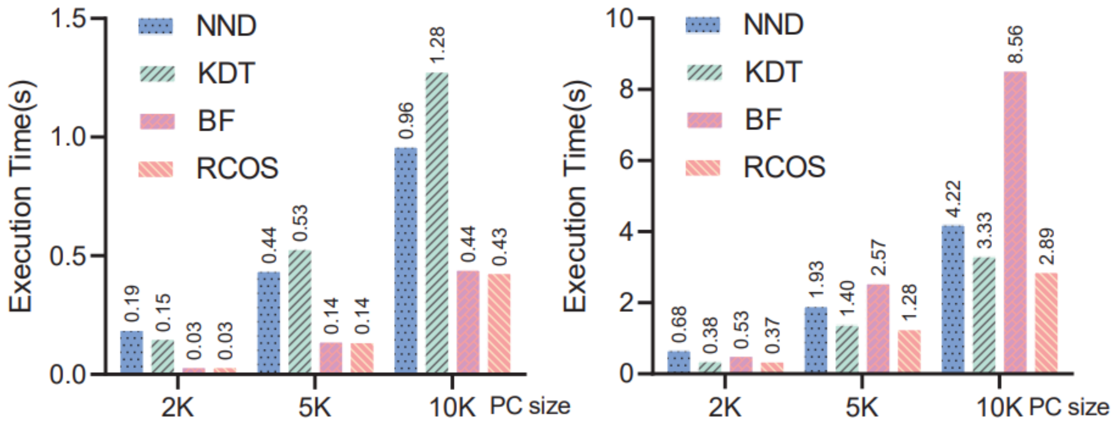
- Dataset: ModelNet40K
- Baselines:
  - CPU: dual Intel(R) Xeon(R) Silver 4214R CPU with 48 cores and 33MB cache
  - GPU: Nvidia Tesla P100 32GB
  - ASICs: PointAcc-edge, Tigris-modified
- Network:
  - EdgeConv with 64,64,128.256 dimension
  - Linear with 512, 512, 256 dimension
- Precision:
  - 16 bits Int in VKU
  - 16 bits Float in Matrix Unit
- Accuracy decrease < 1%

EVALUATED ASIC PLATFORMS

Chip	PointAcc-edge	Tigris-modified	ApoX
Technology	40nm	45nm	45nm
Frequency	1 Ghz	1Ghz	1 Ghz
Compute Unit	64	64	64
On-Chip Mem.	274KB	256KB Cache+ 230KB Buffer	256KB Cache+ 128 KB Buffer
Dram	DDR4-2133	DDR4-3200	DDR4-3200
Bandwidth	17GB/s	25.6GB/s	25.6GB/s
Matrix Unit Cores	16*16	16*16	16*16

Evaluations

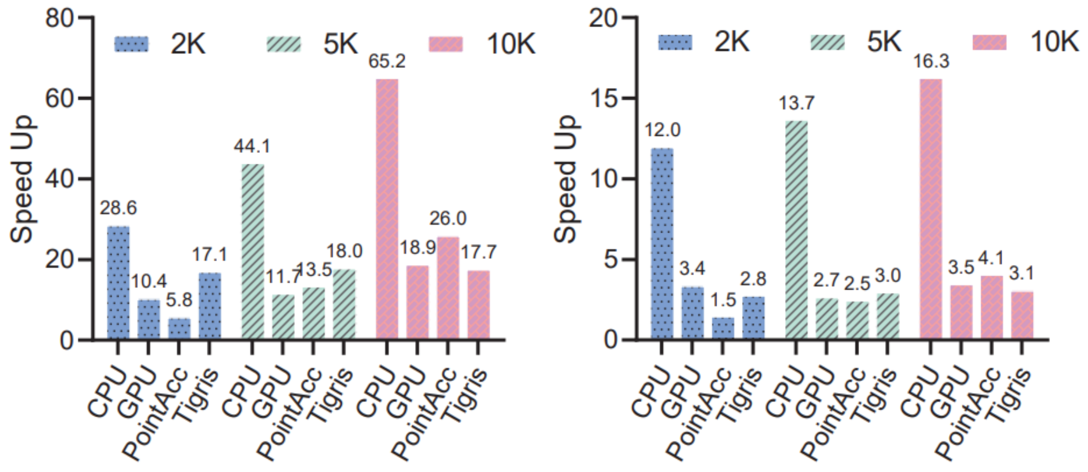
Execution times w.r.t size and Core numbers



(a) All Cores (b) Single Core

- ROAS can always achieve the best performance
- But the potential of algorithms, especially NND, is not be fully developed on the CPU platform

APoX speedup w.r.t baselines under different N

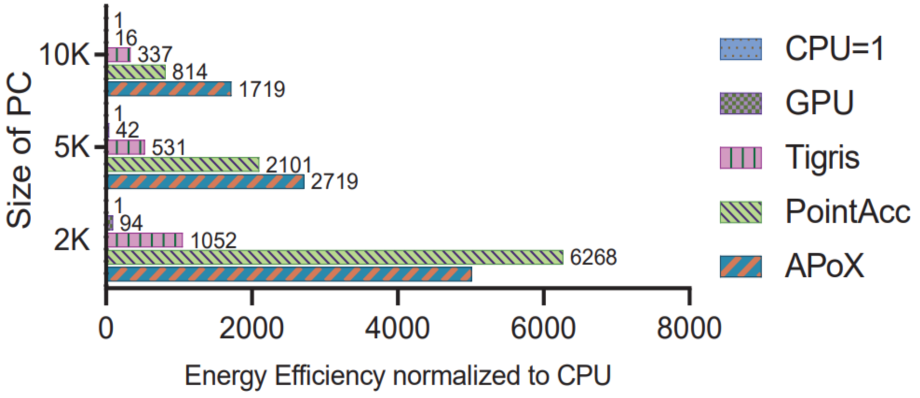


(a) KNN Speed Up (b) End to End Speed Up

- KNN speedup over CPU, GPU, PointAcc, Tigris:
  - Up to 65.2×, 18.9×, 26.0×, 17.7× (in 10K)
- End-to-end speedup over baselines:
  - Up to 16.3×, 3.5×, 4.1×, 3.1× (in 10K)
- The KNN speedup growths with the increase of point cloud size

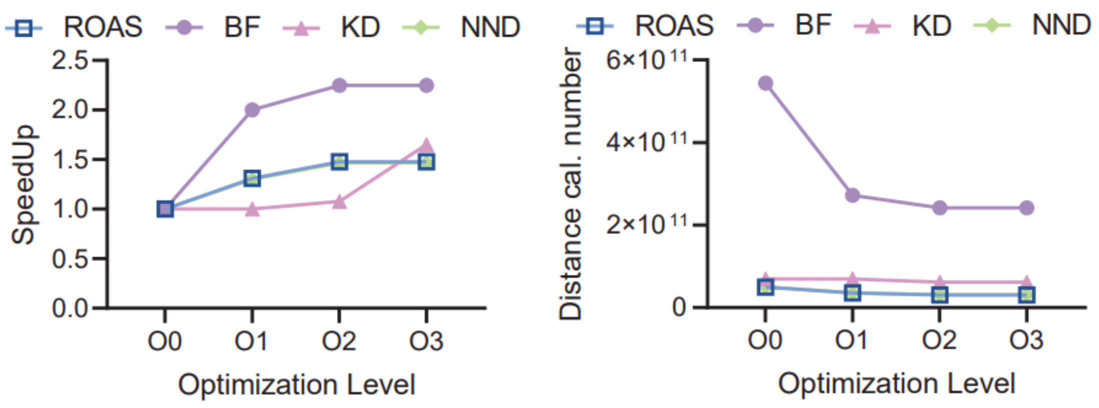
Evaluations

Energy efficiency



- APoX brings the best energy efficiency overall
- The larger N is, the more efficient w.r.t to ASIC baselines

Effect of optimizations [O0: None; O1:A; O2: A+B; O3: A+B+C]



(a)Speedup of each optimization

(b)Distance computation number

Optimizations:

- A. Cache reusable distance
- B. Early Stopping
- C. Batch Execution

- BF benefits from Caching reusable distance most for reduced distance computation
- KDT benefits from Batch Execution most for reduced memory access
- NND benefits averagely from Redundancy Reduction and Early Stopping

Conclusion

- We demonstrates **Graph Construction dominates the processing** of graph-based deep learning point cloud, and **performances of different graph construction varies** substantially under different point clouds.
- We summarize **BruteForce, K-D Trees, and NN-Descent into a unified computation paradigm** and established the **APoX** to support them concurrently with **3 optimizations.**
- The algorithms, **ROAS,** is developed to accurately recommend the optimal algorithm at runtime.
- Our evaluations validate that APoX achieves significant speedup and energy efficiency gains compared to CPUs, GPUs, PointAcc, and Tigris.

Q&A

Thanks For Your Listening!

APoX: Accelerate Graph-Based Deep Point Cloud Analysis via Adaptive Graph Construction

Lei Dai[1,2], Shengwen Liang[1,2,3], Ying Wang[4,2,3,5], Huawei Li[1,2,6], Xiaowei Li[1,2,3]
[1] SKLP, Institute of Computing Technology, CAS, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China
[3] Zhongguancun National Laboratory, Beijing, China
[4] CICS, Institute of Computing Technology, CAS, Beijing, China
[5] Zhejiang Laboratory, Zhejiang, China; [6] Peng Cheng Laboratory, Shenzhen, China
Email: {dailei19z, liangshengwen, wangying2009, lihuawei, lxw}@ict.ac.cn

2024.1.22