Cache locality is not enough : High-Performance Nearest Neighbor Search with Product Quantization Fast Scan

Fabien André, Anne-Marie Kermarrec, Nicolas Le Scouarnec

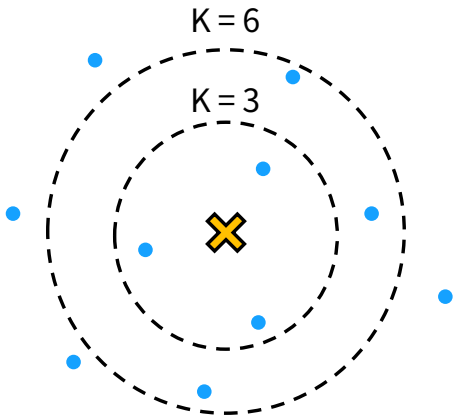
VLDB, 2015

Index

- Introduction
- Related works
  - IVFPQ
  - SIMD
- Method
- Experiments
- Conclusion

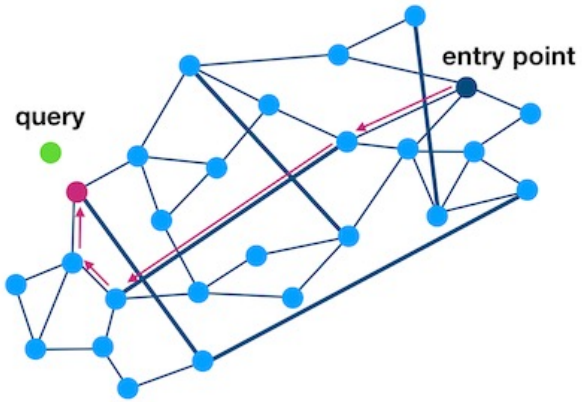
Introduction

- k-Nearest Neighbor search
  - 쿼리 지점과 가장 가까운 (유사한) 벡터 k 개를 찾는 방법



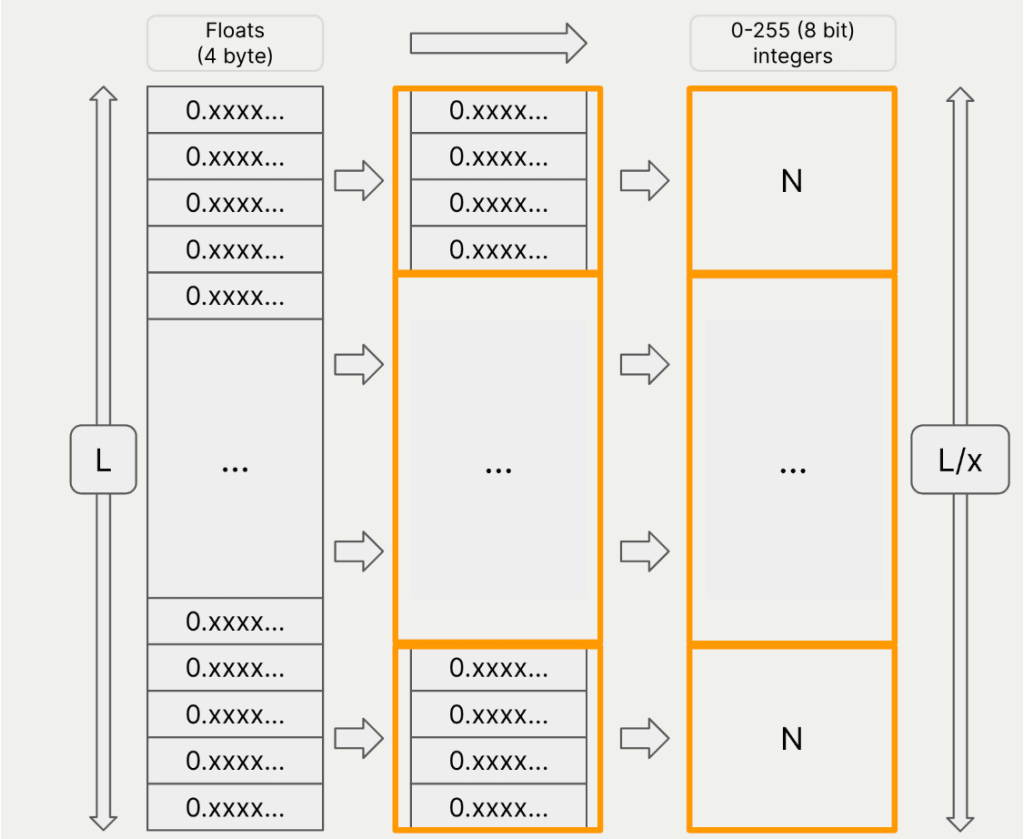
- Approximate k-Nearest Neighbor search
  - 정확도를 약간 포기하고, 속도를 높인 방법
  - 최근 ANN 연구 방법 중,

    Product quantization을 사용한 방법이 SOTA



Related works

- Product Quantization (PQ)
  - 고차원 벡터를 압축하는 방법들 중 하나
  - PQ의 장점?
    - 클러스터링으로 빠른 ANN 검색 가능하게 함
  - 다양한 방법과 결합하여 쓰임
    - IVFPQ, OPQ, DPQ…



Related works

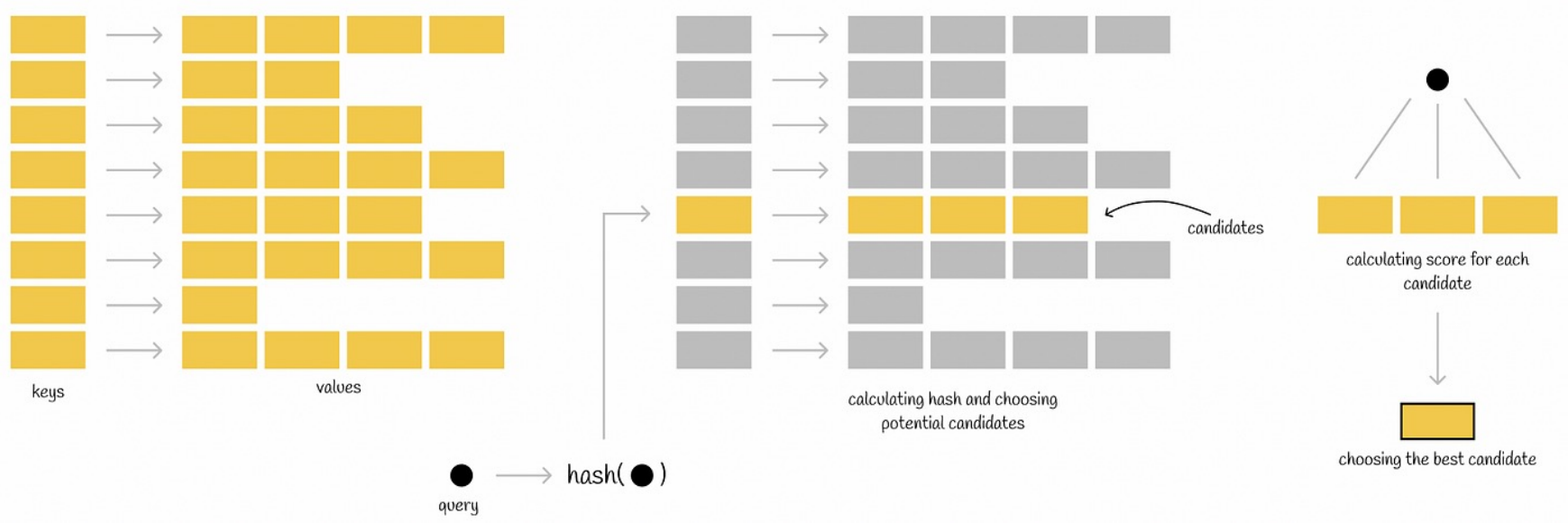
- IVFPQ
  - Inverted File + Product Quantization
  - ANN search에서 자주 쓰이는 방법
  - 본 논문에서는 IVFPQ 중 ADC 방법에 집중함

Related works

- IVFPQ Indexing 과정
  1. Clustering해서 IVF 생성
  2. 각 클러스터 내에서 잔차 구하기
  3. 구한 잔차 벡터를 Product Quantization
  4. IVF에 PQ화된 코드 넣기

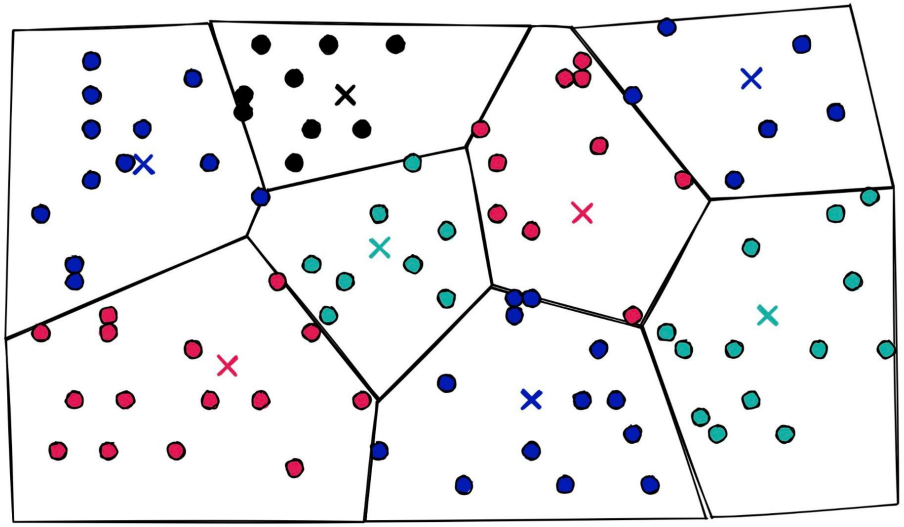
Related works

1. Clustering해서 IVF 생성
    - Inverted File (Inverted index)
        - 단어나 숫자와 같은 콘텐츠에서 테이블, 집합 등의 해당 위치로 매핑하는 데이터베이스 인덱스



Related works

1. Clustering해서 IVF 생성
    - 전체 벡터를 클러스터링하여 센트로이드를 key로 벡터의 인덱스가 담긴 IVF 생성
    - Search 시 탐색 범위를 줄이기 위한 필터링 장치가 됨



c0 → [p3, p10, p12, p6…]

c1 → [p3, p10, p12, p6…]

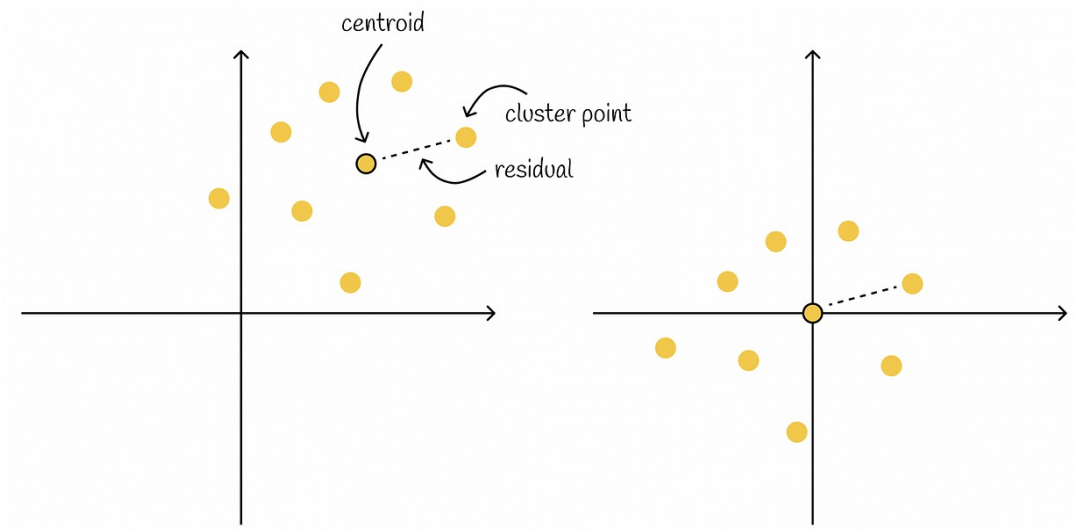
⋮

Cn-1 → [p3, p10, p12, p6…]

Related works

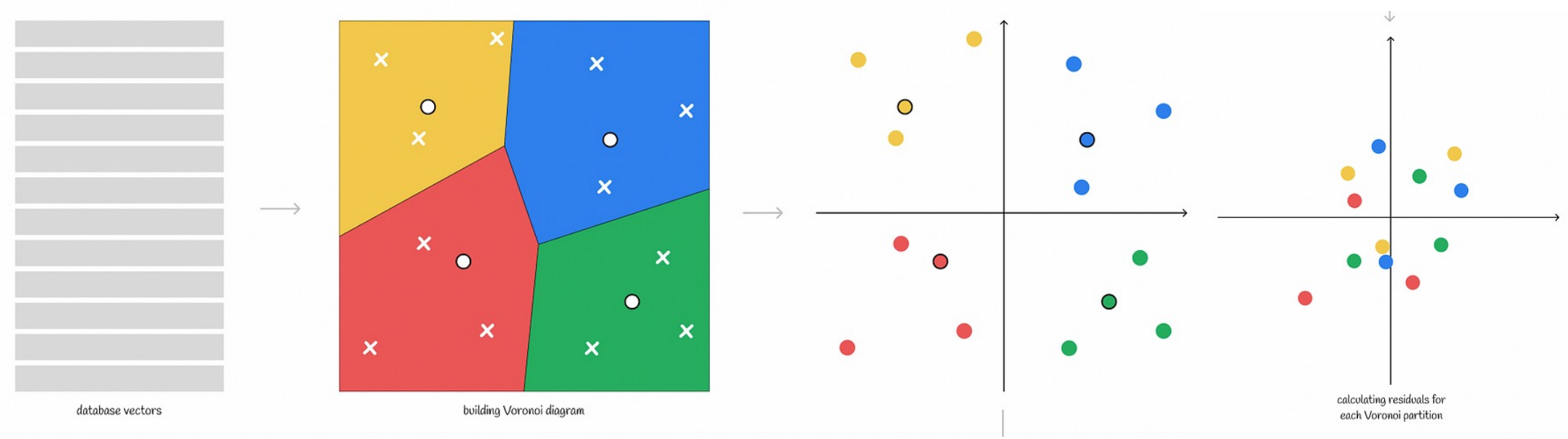
2. 각 클러스터 내에서 잔차 벡터 구하기

   - 잔차 : 중심으로부터 벡터의 오프셋
   - K-means를 한 클러스터의 센트로이드는 해당 클러스터에 속한 모든 점들의 평균
   - 모든 벡터에서 클러스터 평균값을 빼면 점들이 0을 중심으로 배치됨



Related works

2. 각 클러스터 내에서 잔차 벡터 구하기

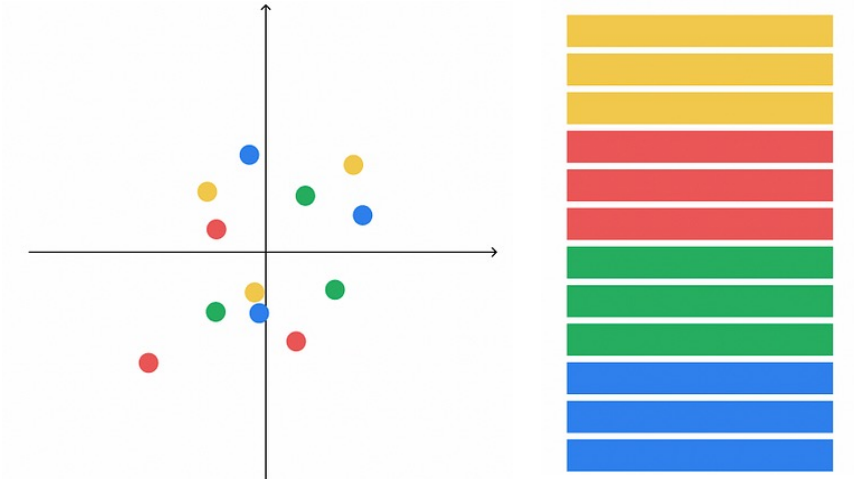


원본 벡터들 클러스터링 센트로이드별 벡터들 잔차 벡터

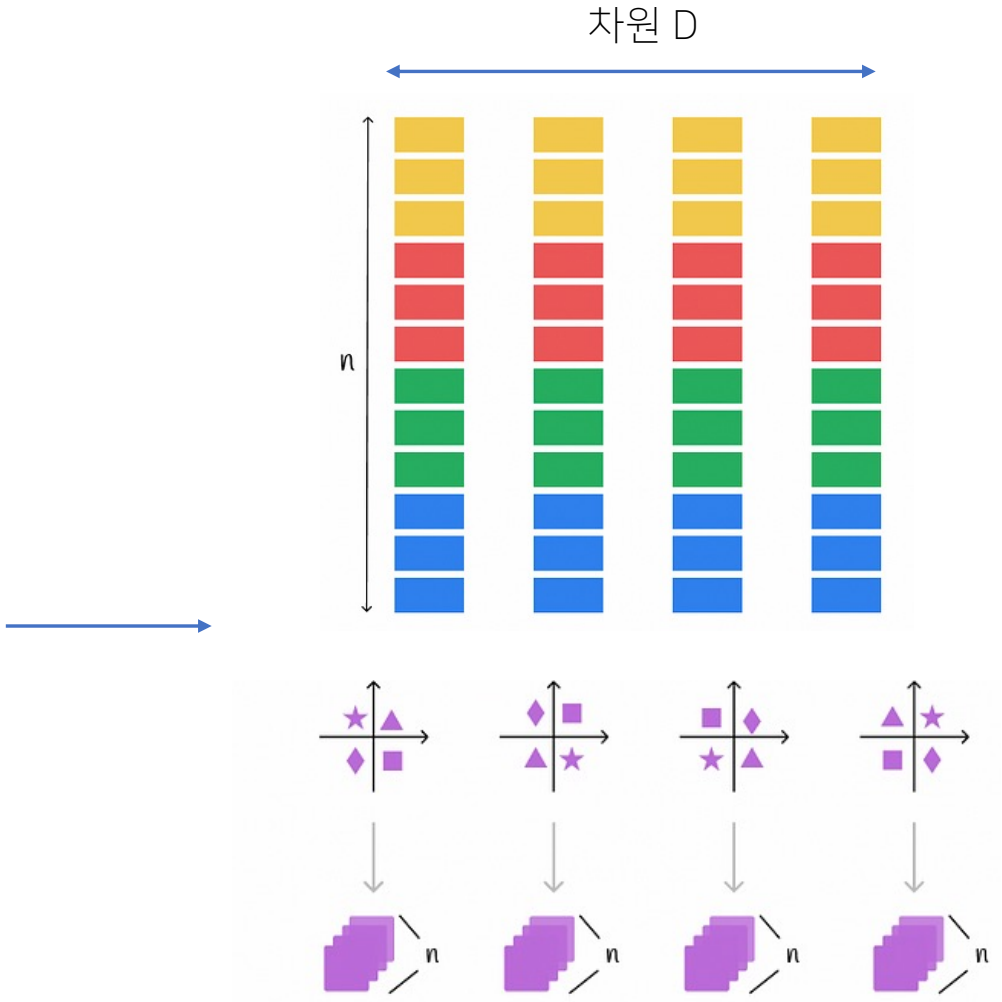
Related works

3. 구한 잔차 벡터를 Product Quantization

- 잔차 벡터들의 차원을 나눠 양자화



계산된
잔차 벡터



Related works

3. 구한 잔차 벡터를 Product Quantization

   - PQ를 통해 메모리 절약 및 빠른 액세스 가능

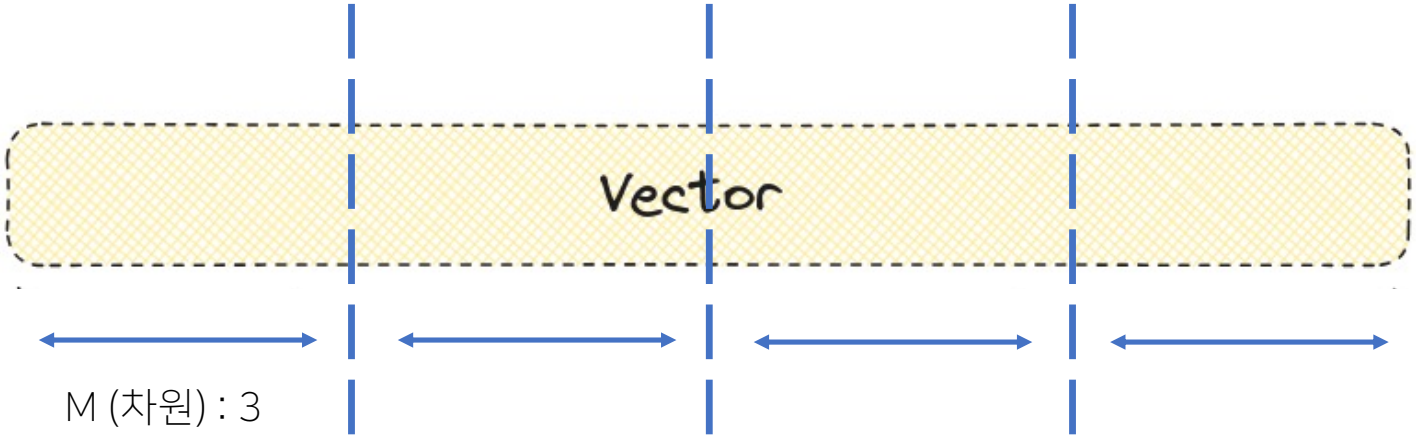
D (벡터의 길이)

Ex) D = 12

Related works

3. 구한 잔차 벡터를 Product Quantization

   - Subvector : 벡터를 몇 개로 나눠서 indexing할지 정하는 파라미터

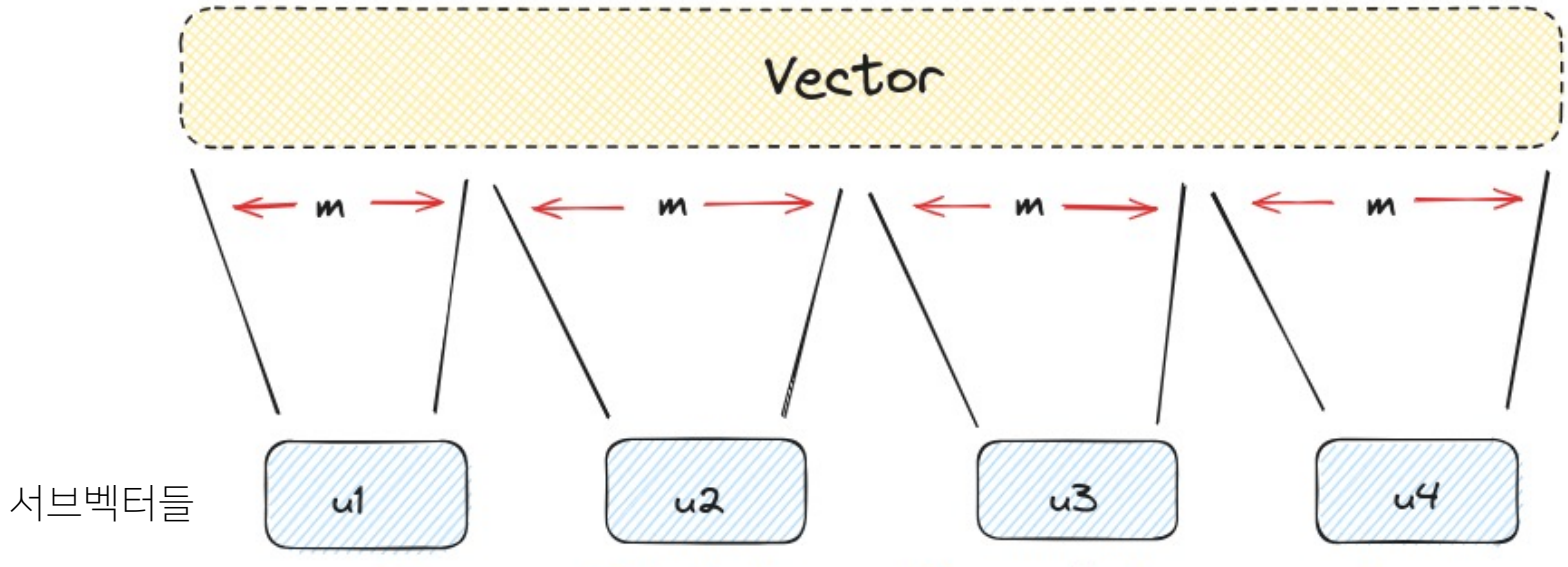


서브벡터 개수 : 4

Related works

3. 구한 잔차 벡터를 Product Quantization

   - Subvector 개수를 정하는 파라미터는 데이터의 차원에 나눠떨어져야함 (D / n_sub = D*)

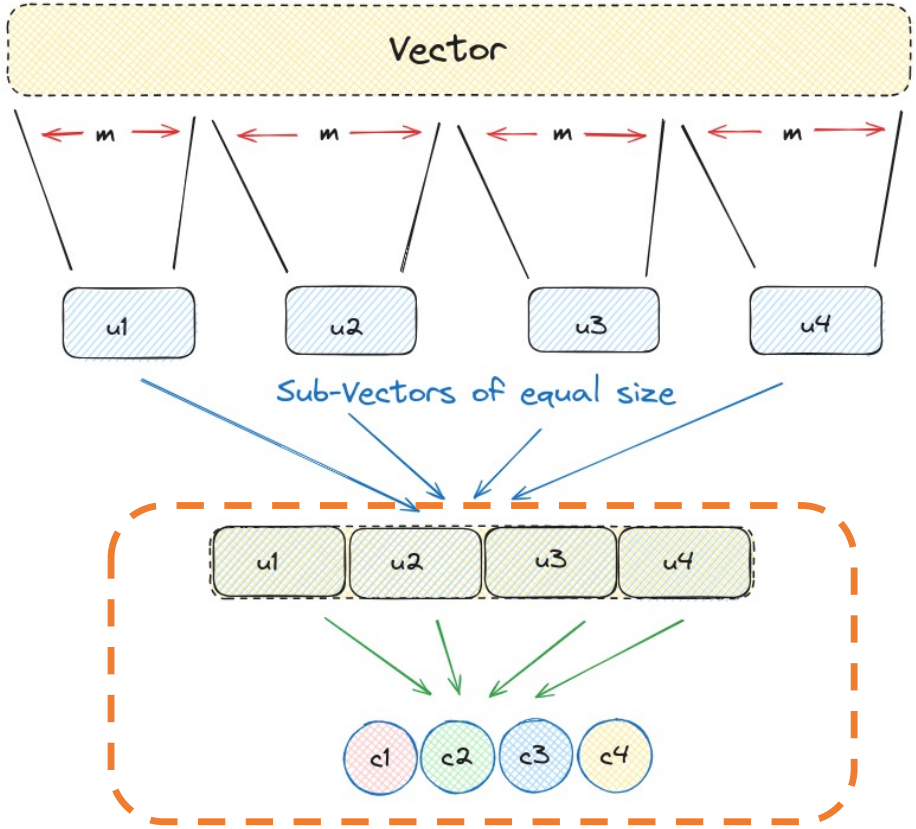


Related works

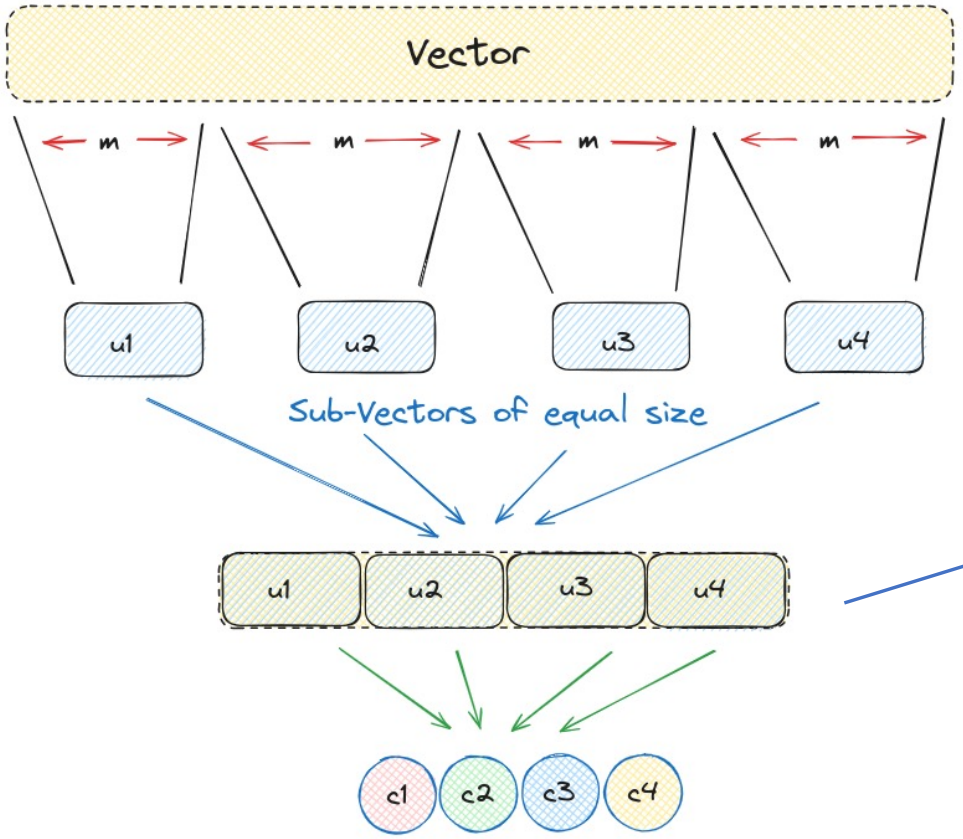
3. 구한 잔차 벡터를 Product Quantization

- 동일한 크기로 나눠진 서브벡터 각각에서 k-means 클러스터링
- 벡터를 이 때 생성된 centroid number로 표현 가능
- 센트로이드 개수는 파라미터

  Ex) n_centorids = 4

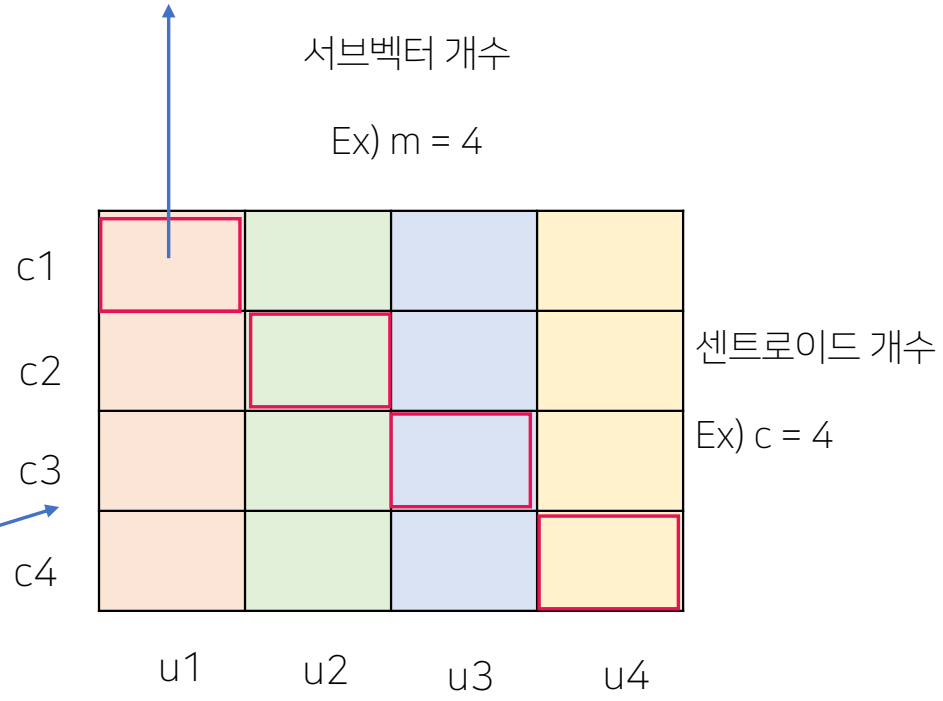


Related works



1, 2, 3, 4 로 벡터 표현 가능

12 (D) / 4 (n_centorids) = 3 (m)
3차원 크기의 벡터 값



서브벡터마다 코드북이 생성됨

Related works

4. IVF에 코드 넣기

   - 1000개의 벡터가 있었다면,

     1000개의 PQ화된 코드

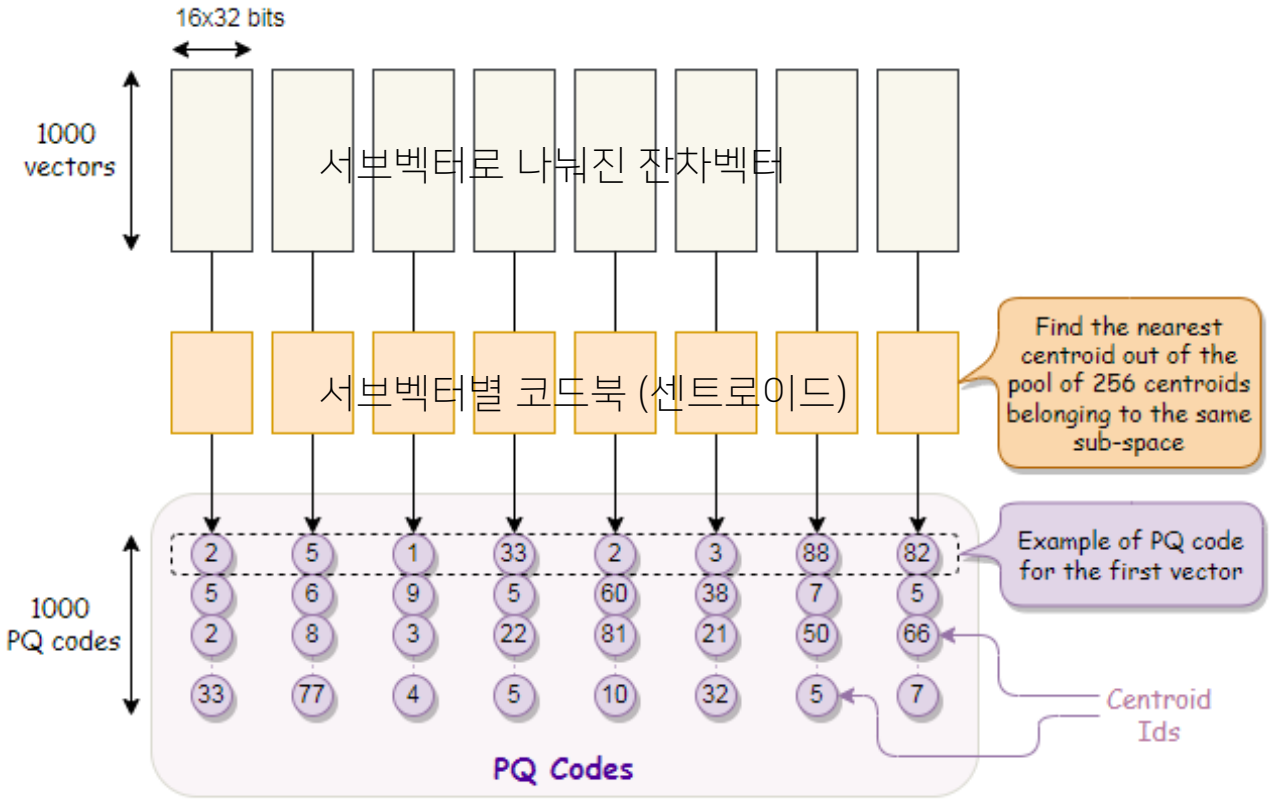
     (코드워드)가 생성됨

   - 해당 IVF에 코드워드 넣기

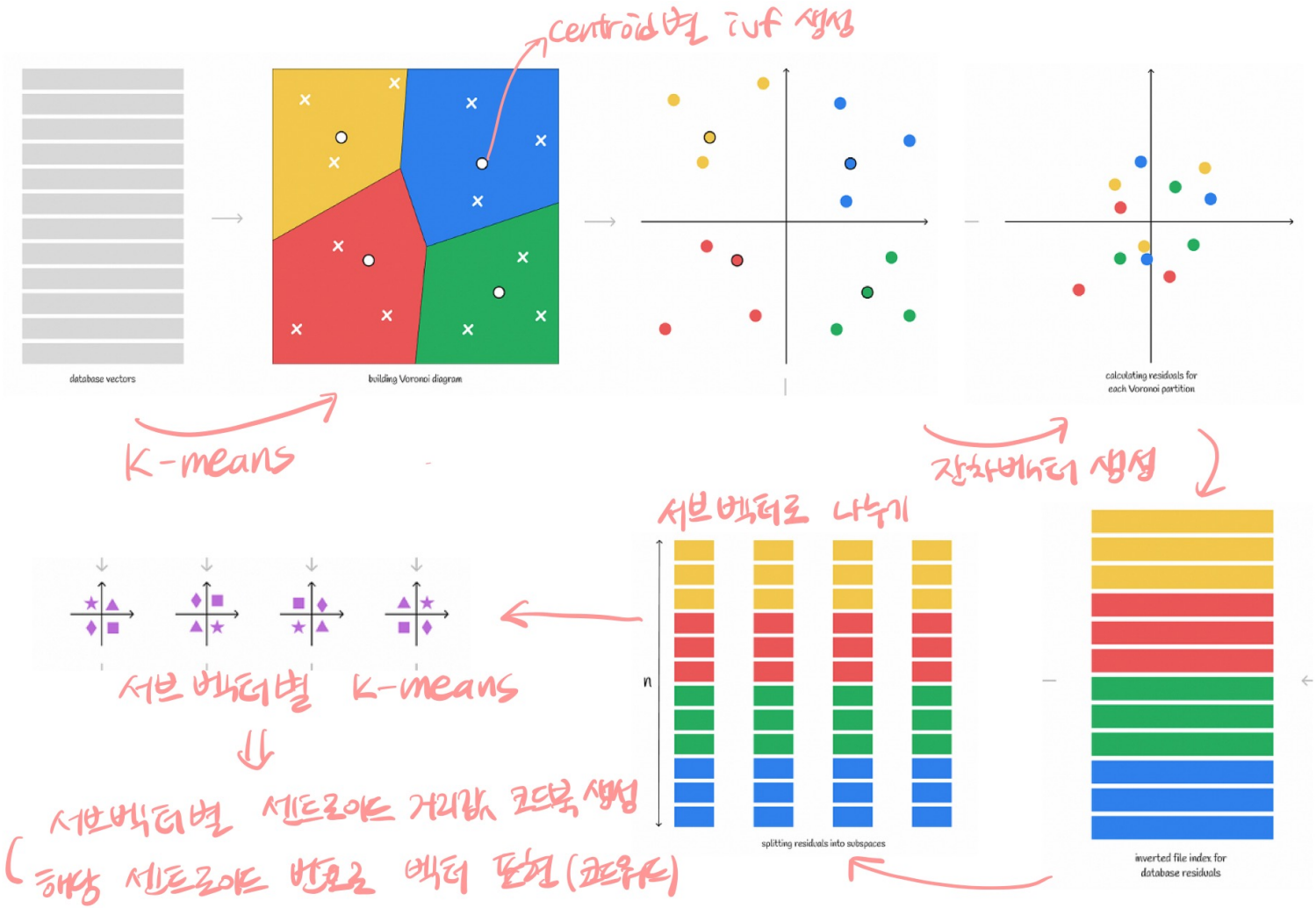
IVF c0 ⟶ [p3, p10, p12, p6…]

↓

2, 5, 1, 33, 2, 3, 88, 82



Related works



Related works

PQ의 효과

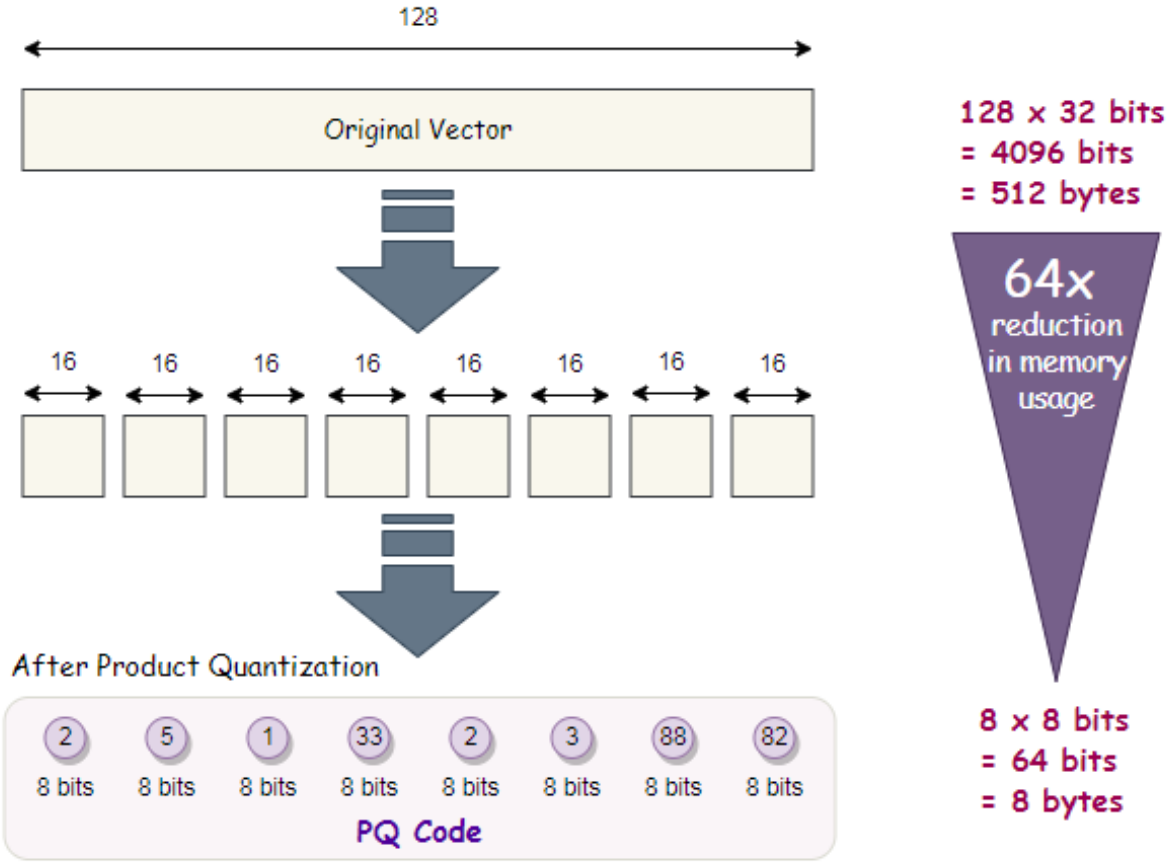
- 메모리 감소 효과
  - 생성된 코드북을 사용하여

    적은 숫자로도 벡터 표현 가능
  - 기존 차원 128,

    서브벡터 개수 8,

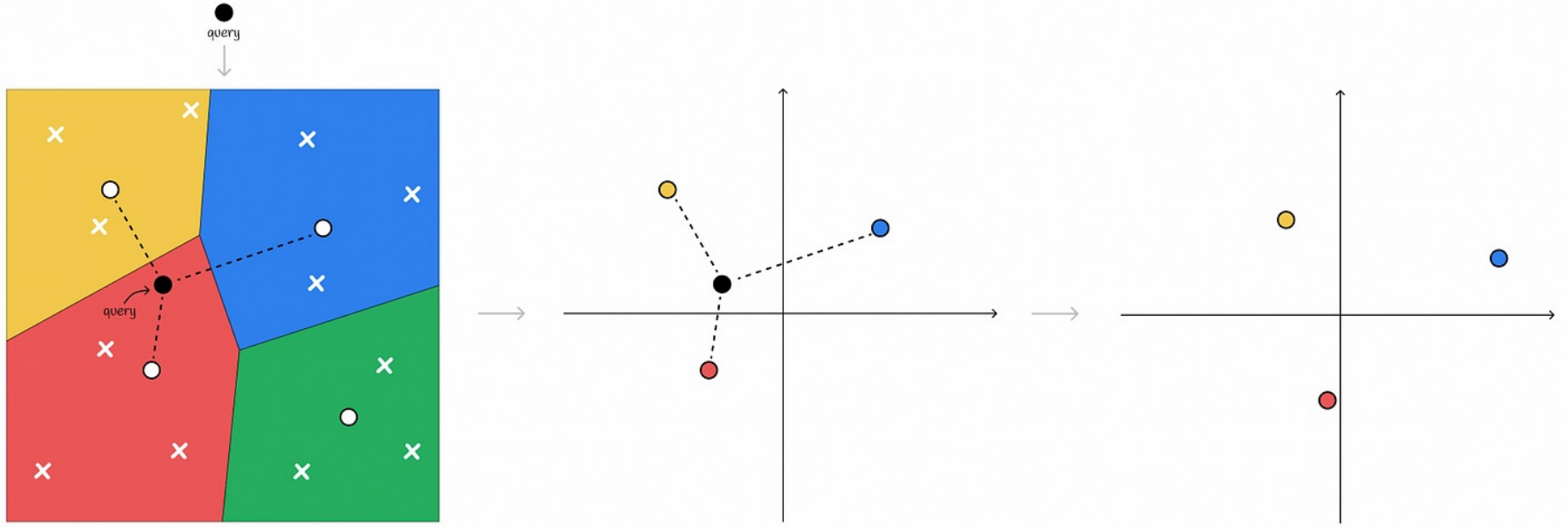
    centroid 256 일 때,

    512 -> 8 bytes



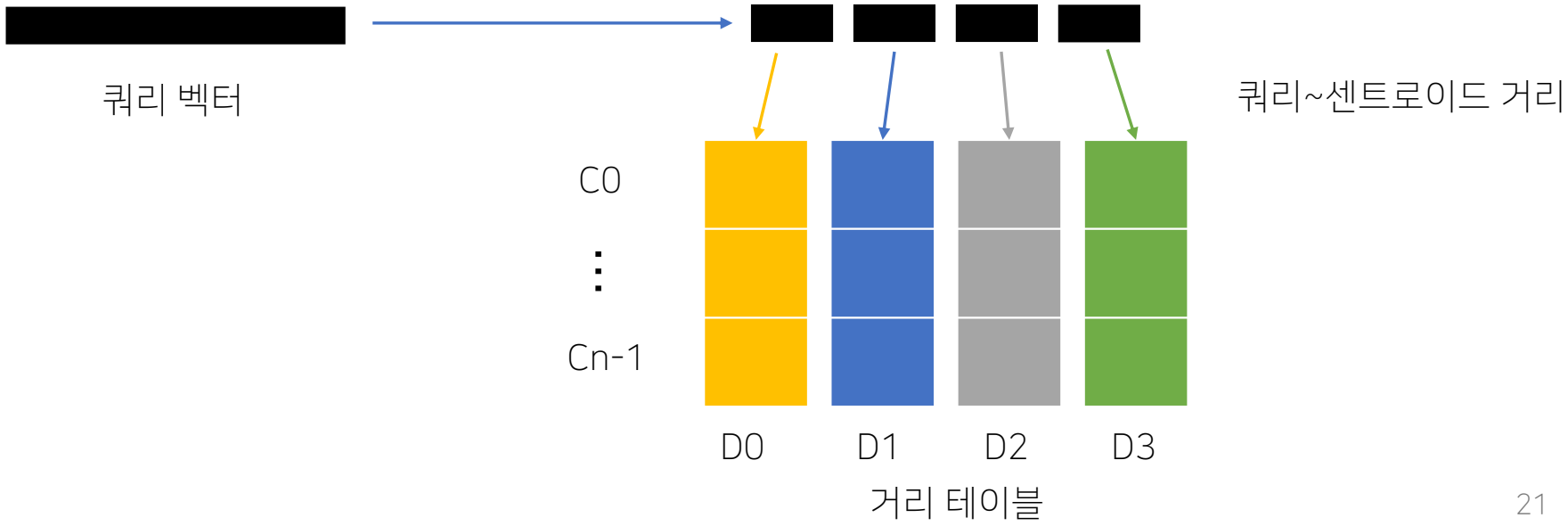
Related works

- IVFPQ Search
  - 쿼리 데이터가 들어오면, IVF 센트로이드와 거리 비교 후 가까운 IVF 선택
  - 선택된 영역에 속한 벡터들만 탐색 대상에 포함됨



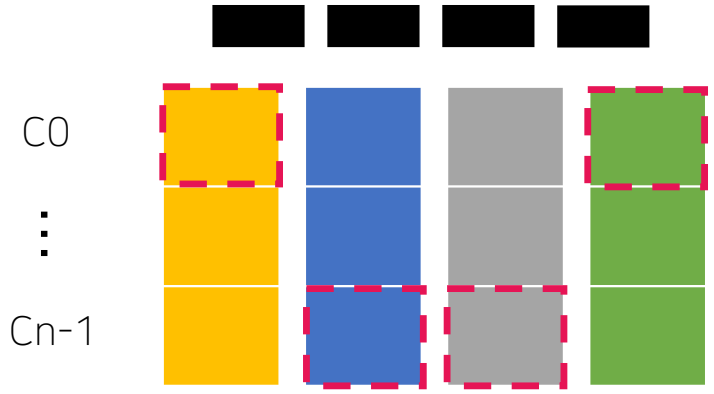
Related works

- IVFPQ Search
  - 쿼리 벡터를 서브 벡터로 나눠, 각 서브 벡터의 모든 센트로이드들(코드북)과 거리 비교 및 테이블 생성
  - 테이블 내 들어가는 각 요소 : 쿼리 벡터로부터 센트로이드 값까지 거리



Related works

- IVFPQ Search
  - 거리 테이블 (Lookup table) 생성 시, 해당 거리 값으로 쿼리 ~ PQ화된 벡터의 거리 계산 빠르게 가능
  - 선택된 IVF 내 모든 PQ화된 벡터들(코드워드)을 거리 테이블을 사용하여 거리 계산
  - 계산된 거리들을 heap에 넣어 k개 선별



거리 테이블

예시) 0 2 2 1

0.1 + 0.2 + 0.3 + 0.4 = 1.0

26 4 12 72

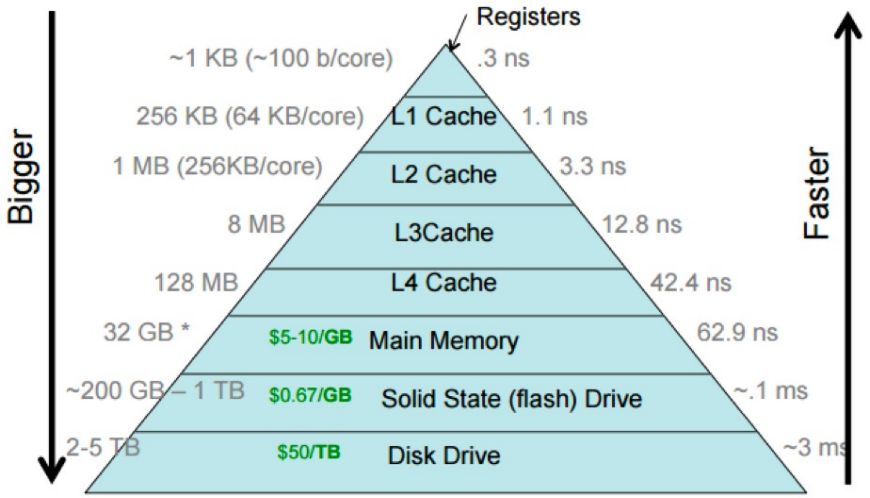
0.35 + 0.96 + 1.84 + 2.8 = 5.95

Related works

- 거리 계산 시 Memory Access
  - 서브벡터 별 거리테이블을 로드하기 위한 메모리 접근 (서브벡터 개수 회)

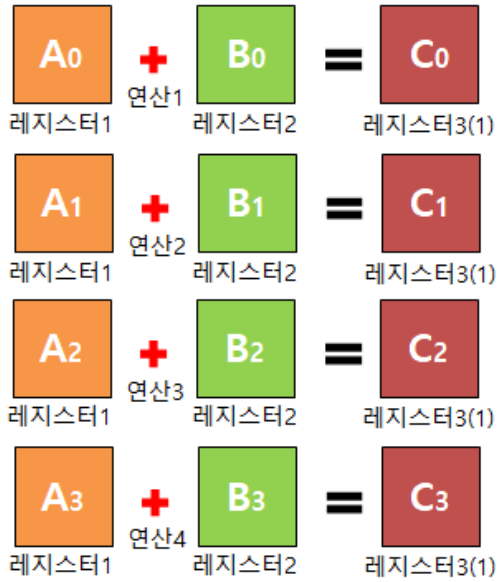
    ► L1 cache
  - 거리 테이블에서 계산할 값을 로드하기 위한 메모리 접근 (서브벡터 개수 회)

    ► 서브벡터 개수에 따라 L3 cache까지 갈 수 있음
  - 서브벡터 개수 회 덧셈

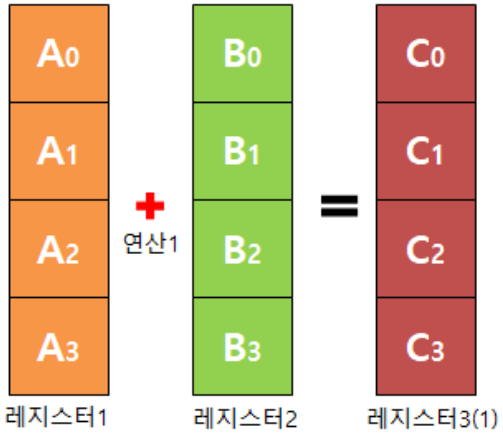


Related works

- SIMD (Single Instruction Multiple Data)
  - 한 번의 명령어로 여러 개의 데이터를 동시에 처리할 수 있도록 하는 병렬 컴퓨팅 기술



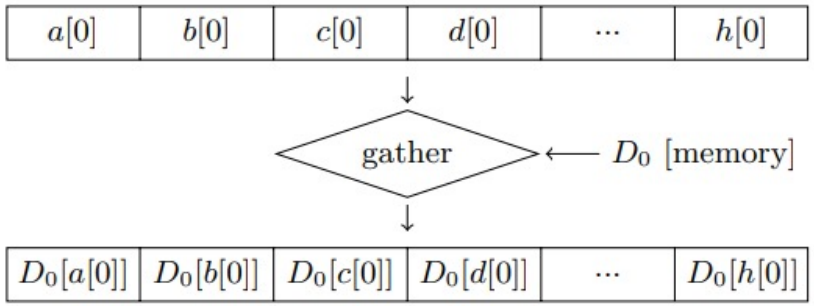
SISD 연산



SIMD 연산

Related works

- SIMD (Single Instruction Multiple Data)
  - SIMD gather 명령어 사용으로 거리 테이블 조회 및 계산에 사용되는 명령어 수와 CPU 사이클 수를 줄임
  - 거리 테이블에서 해당하는 거리값들을 가져오고 더하는 과정을 한번에 가능하게 함
  - 여러 종류의 SIMD 명령어 중 본 논문에서는 AVX 명령어 사용

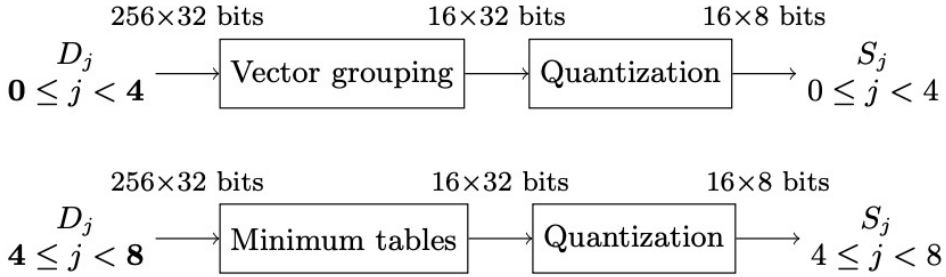


Method

- Motivation
  - 기존 방법으로 거리 계산 시 많은 캐시 접근으로 인한 병목 현상
    - ▶ 최대한 레지스터에 거리 테이블을 올려 캐시를 적게 접근
  - SIMD gather 명령어는 하나의 명령어지만, 로드하는 각 요소마다 1회의 메모리 접근 수행하여 지연 발생
    - ▶ 레지스터에 테이블을 올릴 수 있으며 낮은 지연시간을 가지는 pshufb로 대체
    - ▶ 다만, pshufb는 테이블의 크기를 각각 8비트의 16개 요소(16×8 비트, 128 비트)로 제한

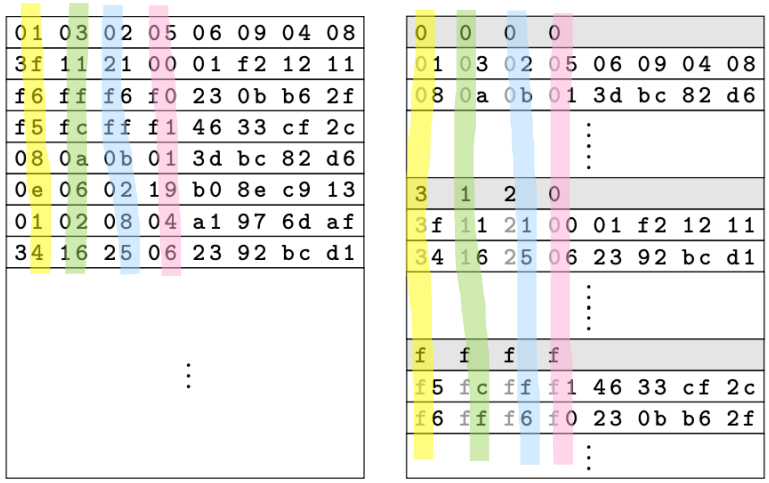
Method

- Overview
  - 목표 : 효율적인 메모리 접근과 SIMD 연산이 가능하도록 작은 테이블(하나당 16 x 8 bits)을 만들어야 함
  - (스캔된 벡터당 2회 미만의 L1 캐시 접근 수행)
  - 8개의 subvector, 256개의 센트로이드 PQ에 집중
  - 크게 두 단계로 나뉨
    - Vector grouping
    - Minimum tables

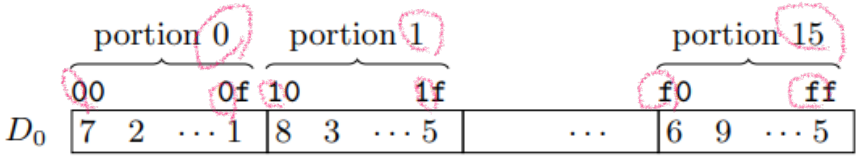


Method

- Vector grouping (256*32bits -> 16*32 bits)
  - 원래 PQ는 8개(서브벡터)의 거리 테이블 사용, 각 거리 테이블은 256개(센트로이드)의 32비트 요소로 구성됨
  - 작은 테이블을 쓰기 위해 D0…D3까지 4개의 거리 테이블은 벡터 그룹화 과정을 거침
  - 256까지의 구간을 16개로 나눠, 16개씩 같은 그룹에 들어가게 설정 (그룹 0의 경우 0-15까지)

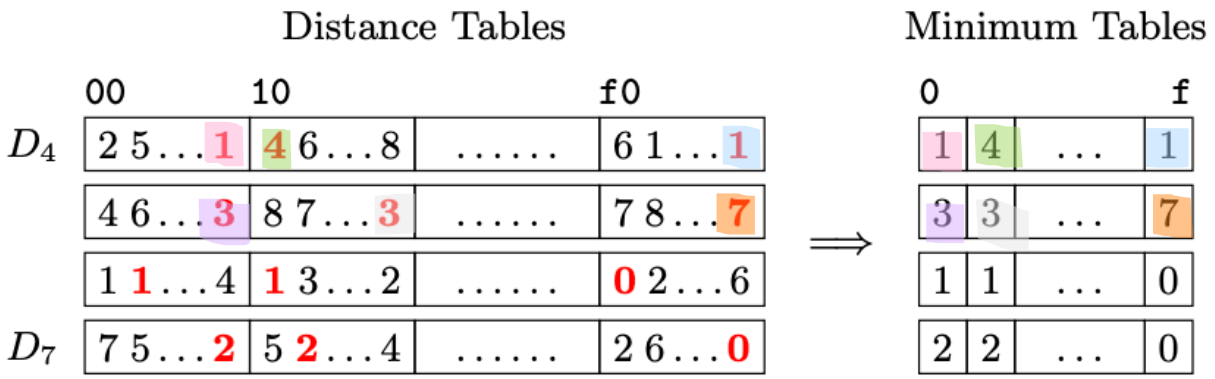


PQ화된 벡터들　　그룹화된 벡터들



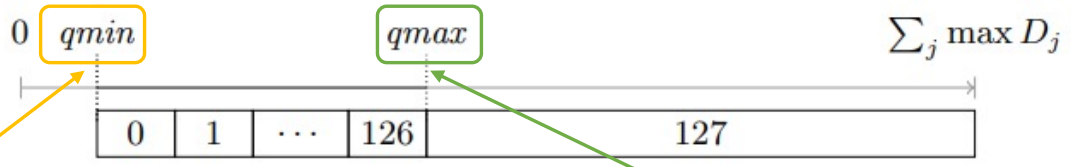
Method

- Minimum tables (256*32bits -> 16*32 bits)
  - D4…D7의 거리 테이블은 Minimum table을 통해 축소
  - 기존 거리 테이블들을 16개 구간으로 나눠, 구간 중 가장 작은 값을 가지게 함
  - 하한 값을 구하여, 탐색해야 할 벡터들을 pruning(가지치기) 하기 위함



Method

- Quantization Distance (16*32bits -> 16*8bits)
  - 원래 거리 테이블을 binning 해서 32bit -> 8bit로 small table에 올라가게 하기 위함
  - 부동 소수점 거리를 8비트 정수로 양자화
  - 127개의 구간으로 나누기, 각 구간의 크기는 (qmax−qmin)/127



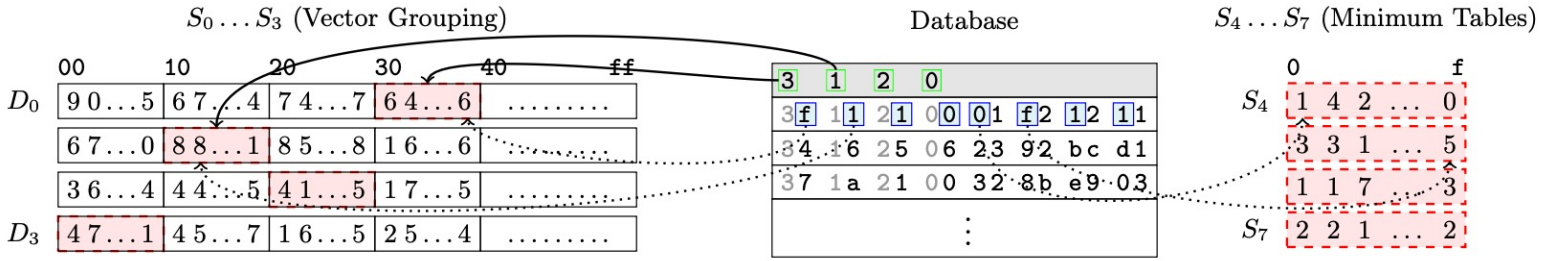
모든 거리 테이블에서 최소값

데이터베이스 파라미터% 벡터 중
쿼리 벡터의 임시 최근접 이웃의 거리
(보통 0.1-1% 사이)

Method

- Small table (16*8bits)
  - 거리 테이블을 레지스터에 올리기 위해 vector grouping 및 minimum table 사용
  - 검색해야 할 그룹들을 가져와 S0…S3까지는 Vector grouping,

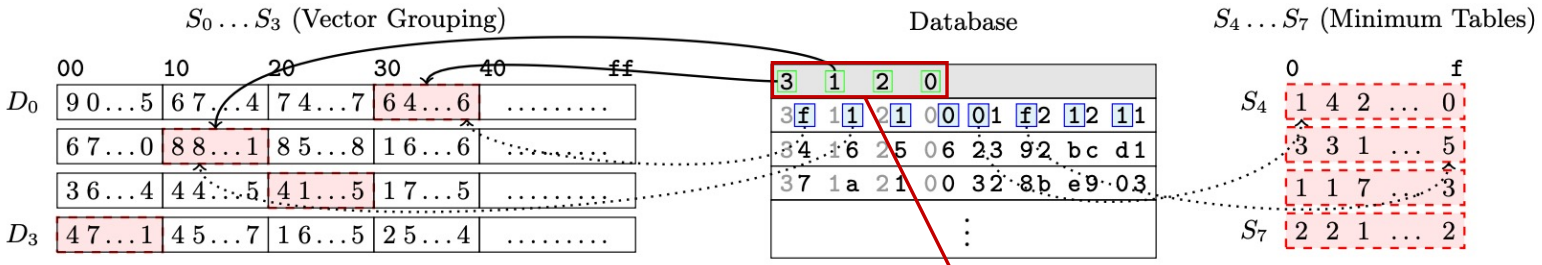
    S4…S7까지는 Minimum table로 Small table 형성
  - Minimum table은 값이 바뀌지 않음



Method

- Small table 생성 예시

구간별 최소값이기 때문에
그룹이 바뀌어도 달라지지 않음



계산해야 할 그룹에 따라 S0…S3 달라짐

Vector Grouping

Minimum Tables

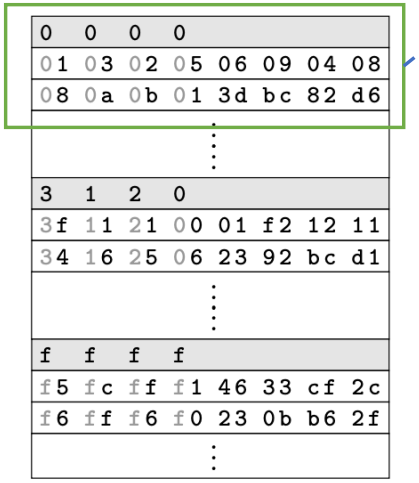
	0	1	2	…	f	
S0	6	4	12	…	6	30…3f
S1	8	8	0	…		10…1f
S2				…		20…2f
S3				…		00…1f
S4				…		
S5				…		
S6				…		
S7				…		

양자화된 거리
(0-127)

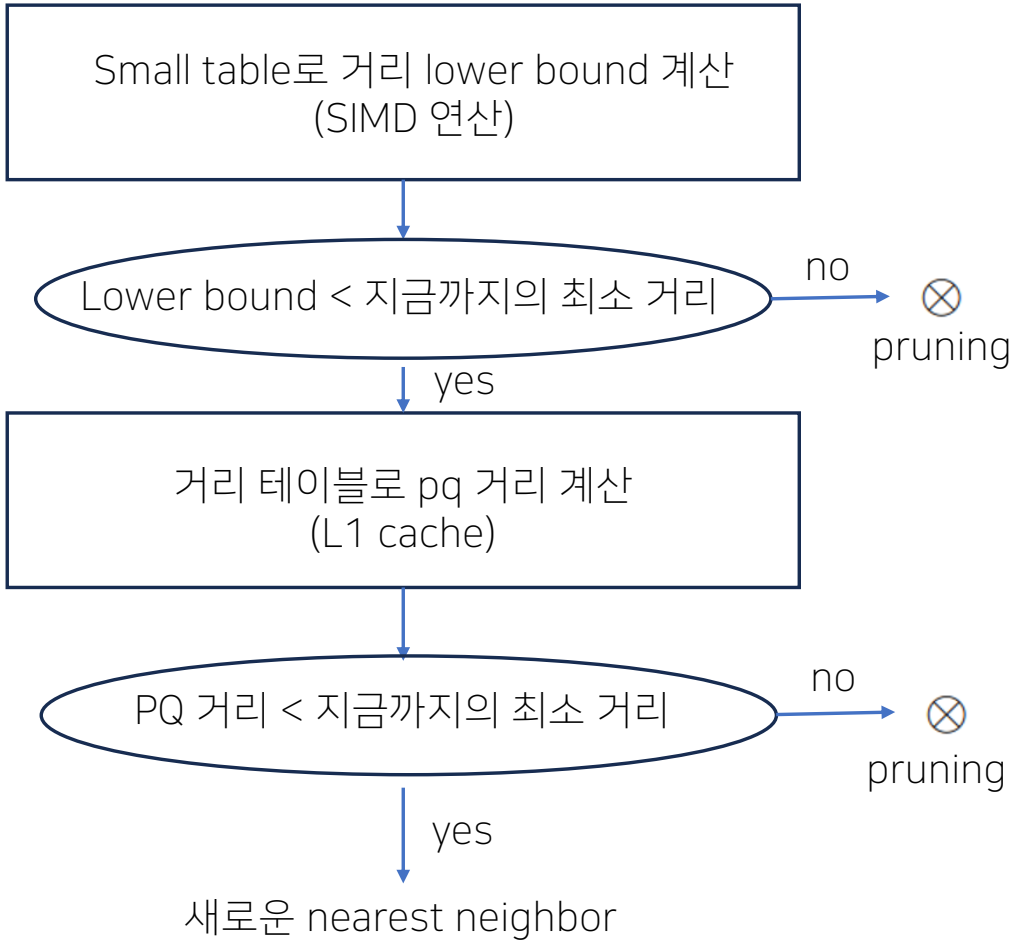
S 하나당 16 x 8 bits

Method

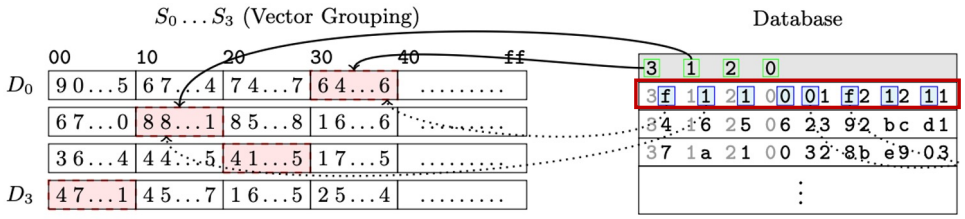
- Lookups in small tables (검색 시)
  - 그룹을 가져와서, small table 형성 및 계산



그룹화된 벡터들



Method



	0	1	2	…	f
S0	6	4	12	…	6
S1	8	8	0	…	
S2	4	1	56	…	5
S3	4	7	98	…	1
S4	1	4	2	…	0
S5	3	3	1	…	5
S6	1	1	7	…	3
S7	2	2	1	…	2

6 + 8 + 1 + 4 + 1 + 5 + 1 + 2 = 28

3f 11 21 00 01 f2 12 11

Small table로 거리 lower bound 계산
(SIMD 연산)

Lower bound < 현재 최근접 이웃 사이의 거리 — no → ⊗ pruning

yes

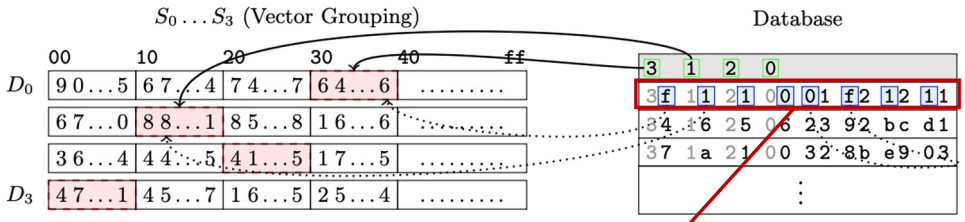
거리 테이블로 pq 거리 계산
(L1 cache)

PQ 거리 < 현재 최근접 이웃 사이의 거리 — no → ⊗ pruning

yes

새로운 nearest neighbor

Method



	00	10	20	···	ff
D0				···	
D1				···	
D2				···	
D3				···	
D4				···	
D5				···	
D6				···	
D7				···	

양자화 되지 않은 실제 거리

0.3 + 0.7 + 0.21 + 0.34 + 0.5 + 0.12 + 0.1 + 1.2 = 3.469

3f 11 21 00 01 f2 12 11

Small table로 거리 lower bound 계산
(SIMD 연산)

Lower bound < 현재 최근접 이웃 사이의 거리

no → ⊗ pruning

yes

거리 테이블로 pq 거리 계산
(L1 cache)

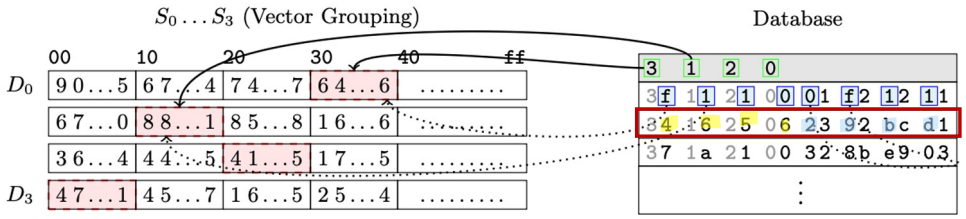
PQ 거리 < 현재 최근접 이웃 사이의 거리

no → ⊗ pruning

yes

새로운 nearest neighbor

Method



	⋯	4	5	6	⋯	f
S0	⋯	60	4	12	⋯	6
S1	⋯	8	8	7	⋯	
S2	⋯	4	1	56	⋯	5
S3	⋯	4	7	98	⋯	1
S4	⋯	1	4	2	⋯	0
S5	⋯	3	3	1	⋯	5
S6	⋯	1	1	7	⋯	3
S7	⋯	2	2	1	⋯	2

60 + 7 + 1 + 98 + 3 + 5 + 1 + 8 = 183

34 16 25 06 23 92 bc d1

Small table로 거리 lower bound 계산
(SIMD 연산)

Lower bound < 현재 최근접 이웃 사이의 거리
183 28

no → ⊗ pruning

yes

거리 테이블로 pq 거리 계산
(L1 cache)

PQ 거리 < 현재 최근접 이웃 사이의 거리
3.469

no → ⊗ pruning

yes

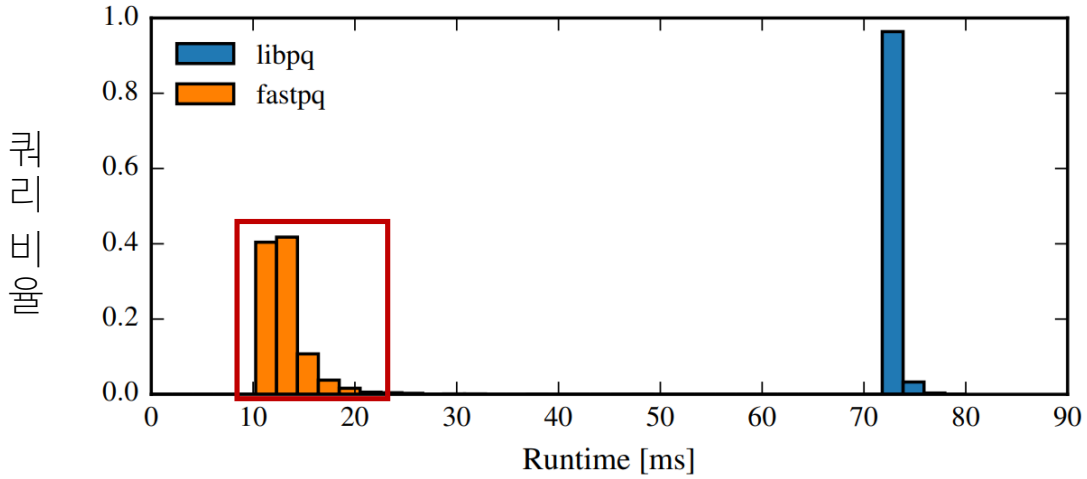
새로운 nearest neighbor

Experiments

- Setting
    - 데이터셋
        - SIFT 1B 사용
        - 128차원 벡터
    - 8개 파티션으로 인덱스 구축 (주 실험 파티션 크기 : 25M)
    - 쿼리는 가장 관련성이 높은 파티션으로 유도됨
    - 주로 fastpq (Fast Scan)와 libpq(PQ Scan) 비교 실험

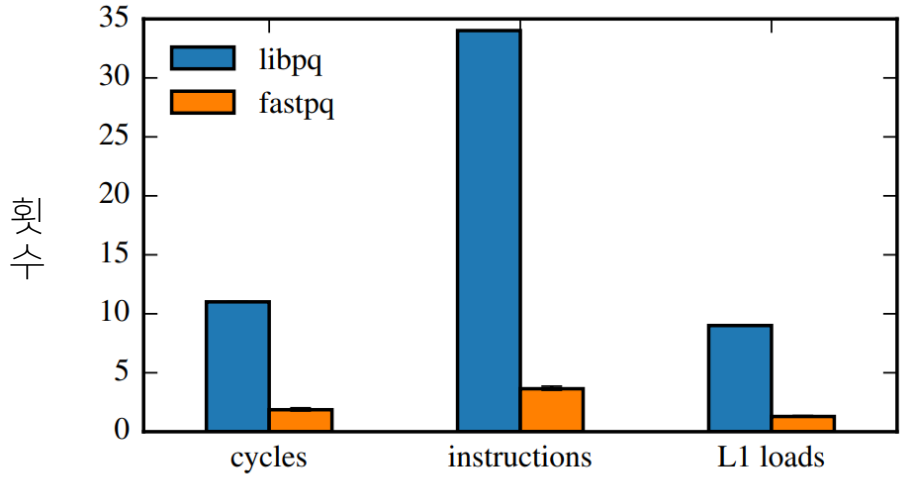
Experiments

- 응답 시간
  - 25M 벡터에서 2595개 쿼리의 최근접 이웃 쿼리 응답시간 분포 (top k = 100)
  - Fast scan이 기존 방법보다 4-6배 빠르게 응답



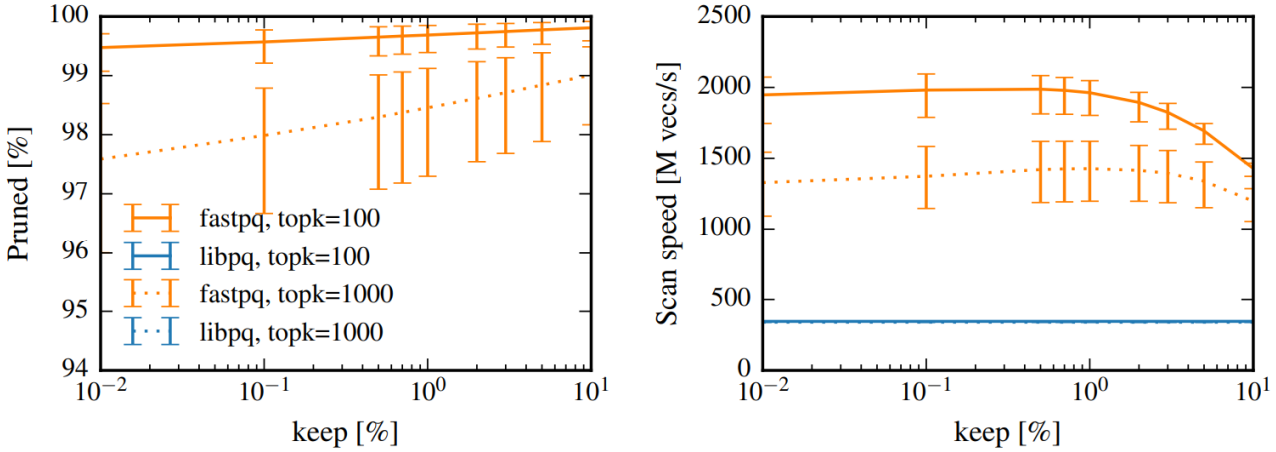
Experiments

- 성능 카운터
  - 83% 적은 cpu 사이클 사용 (cycle : cpu가 하나의 명령어를 처리하기 위한 동작의 주기)
  - 89% 적은 명령어 필요
  - 적은 L1 load 필요



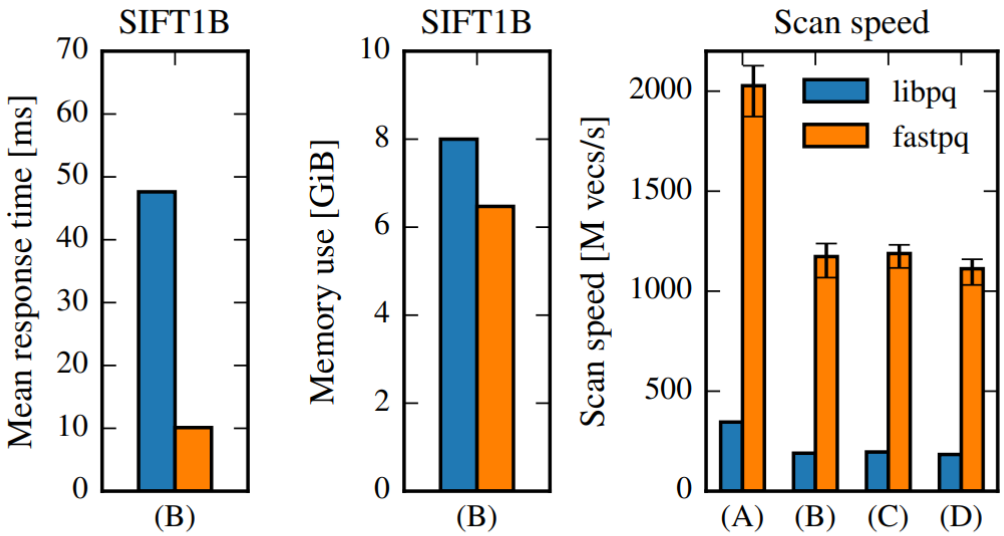
Experiments

- 파라미터 실험 – keep의 영향
  - Keep : qmax를 정하기 위해 볼 데이터의 비율
  - Keep을 늘릴수록 값이 정확해지므로 pruning은 잘 되지만, 일반 PQ Scan 방법으로 쿼리해야하므로 scan speed가 느려짐



Experiments

- 대규모 데이터에서의 성능
  - 10억개 벡터로 구성된 전체 데이터베이스에서 테스트
  - 10000개의 쿼리 수행
  - Fast Scan이 응답 시간이 더 짧고 메모리 사용량도 적음을 확인
  - 다양한 실험 환경에서도 Scan speed가 빠름 (A, B, C, D)



평균 응답 시간 메모리 사용량 스캔 속도

Conclusion

- Product Quantization 분야에서도 최적화에 집중한 논문
- SIMD 레지스터 사용으로 ANN 검색을 최적화함
- 최근 ANN 벤치마크에서 높은 성능을 보이는 메소드들 중에서 PQ가 많이 사용되고 있음
- 본 논문이 최적화 및 경량화 연구에 도움이 되고 있고, 다른 분야에도 적용될 수 있다고 예상함

감사합니다