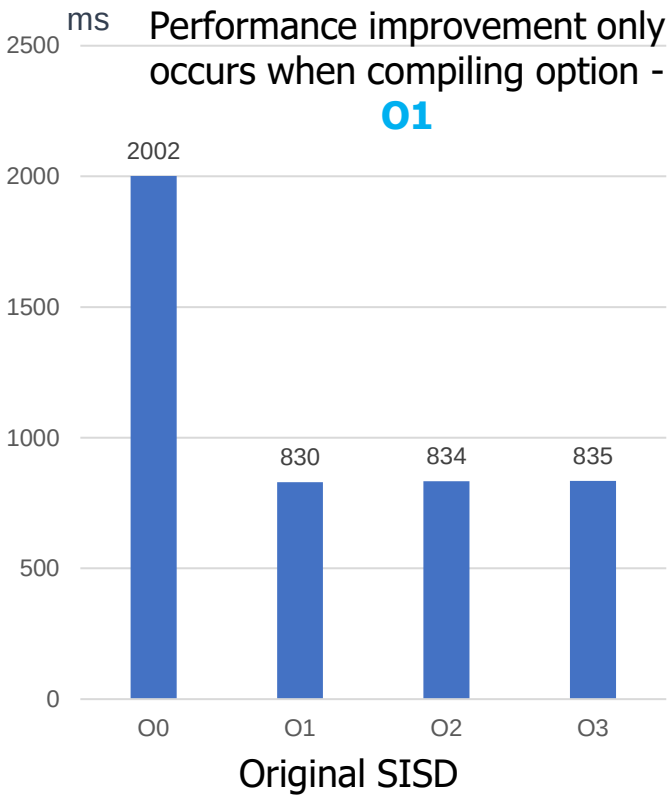
RMI + SIMD

Oh Yeojin, ZhuYongjie, Boseung Kim, Minguk Choi
{yeojinoh, arashio1111, bskim1102, mgchoi}@dankook.ac.kr

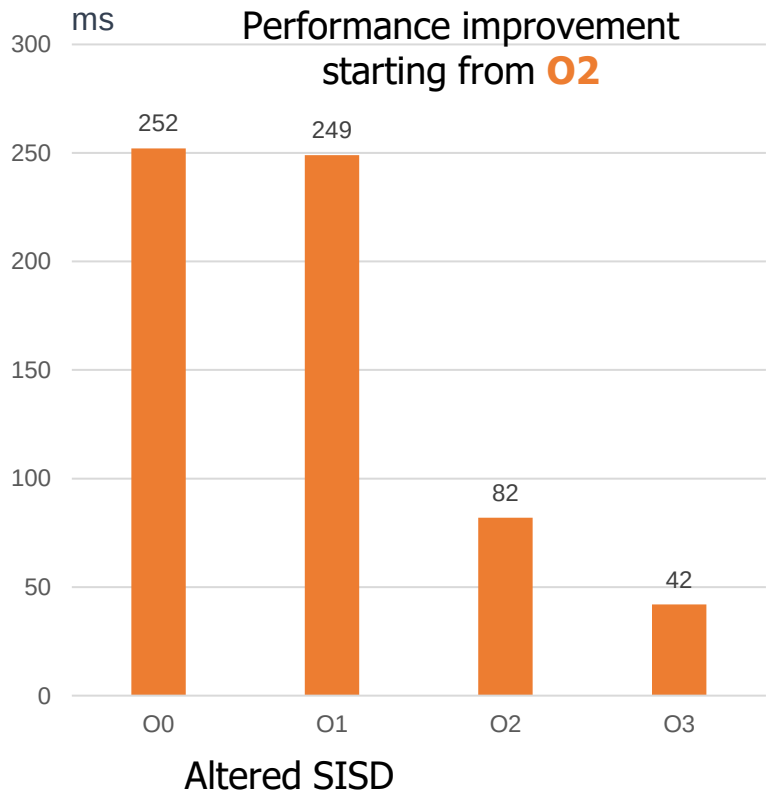
Contents

Model training with SIMD

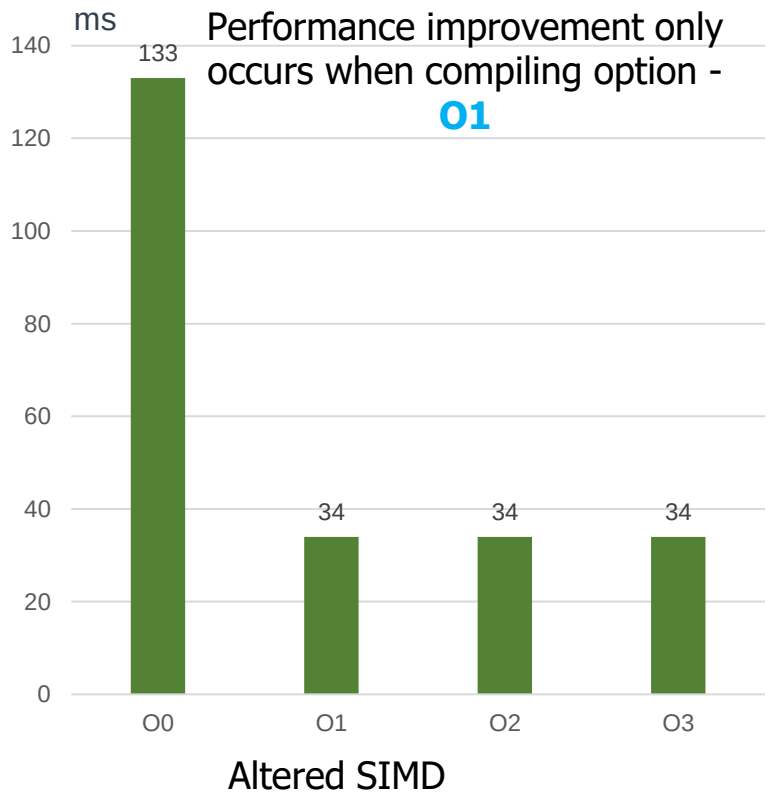
- Training Time (200M uint32_t keys between [1,200M+1])



Original SISD



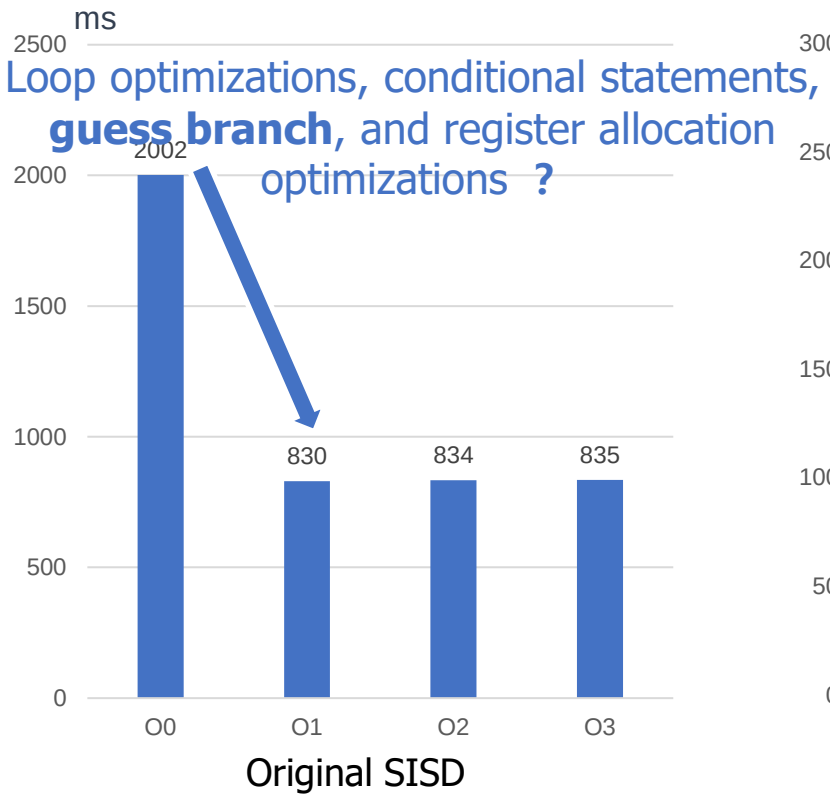
Altered SISD



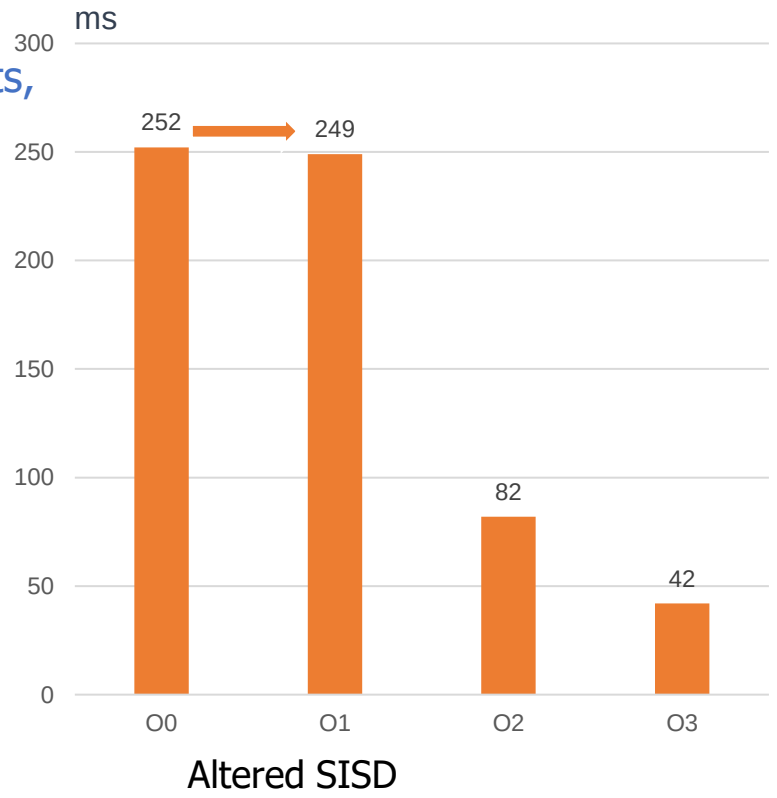
Altered SIMD

Model training with SIMD

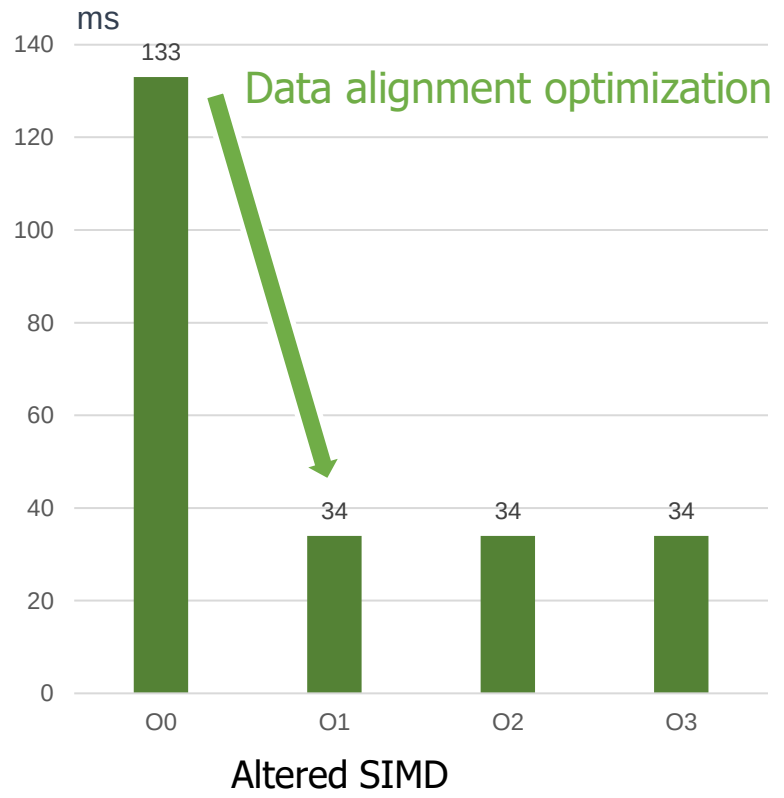
- Training Time (200M uint32_t keys between [1,200M+1])



Original SISD



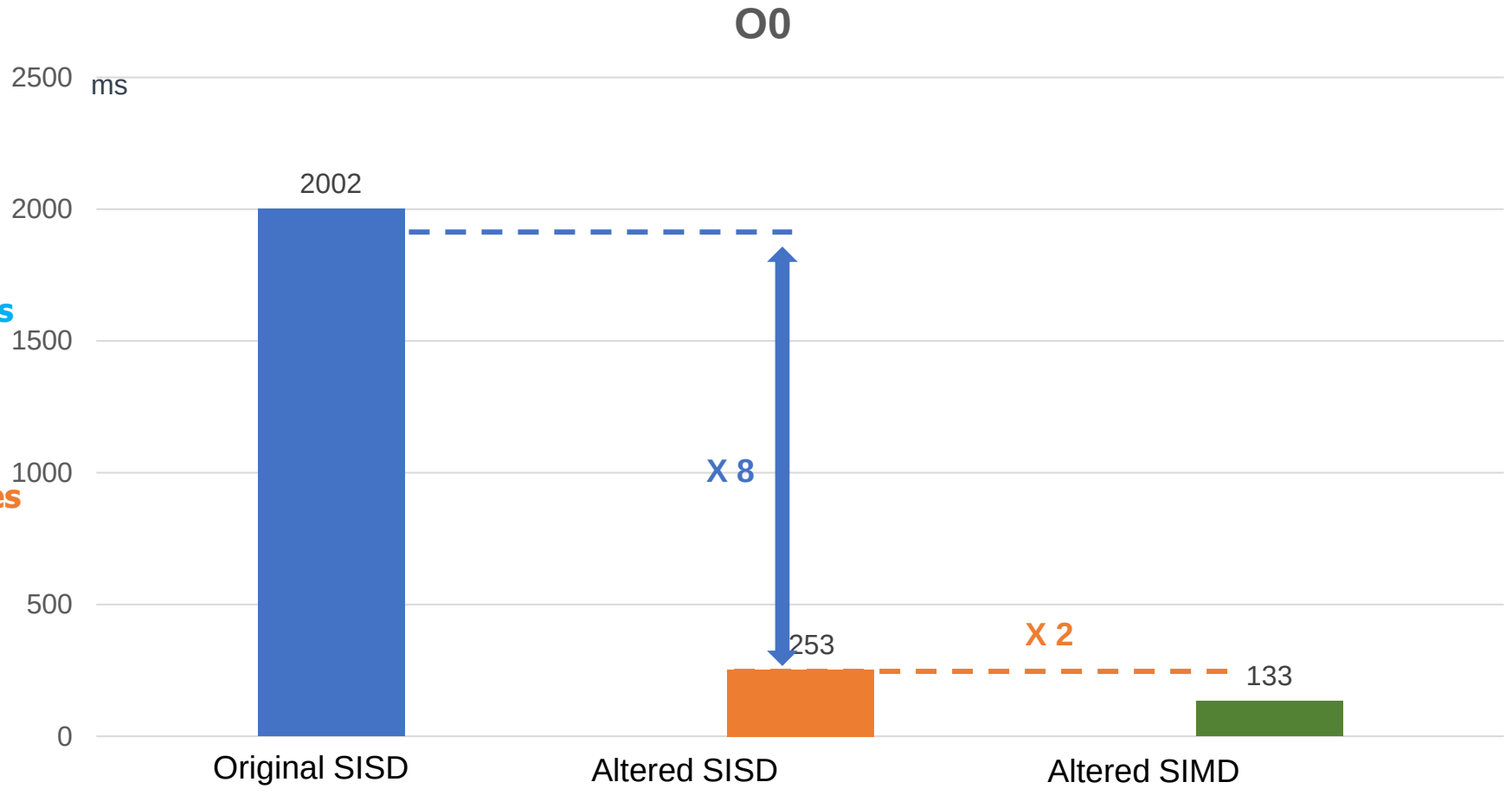
Altered SISD



Altered SIMD

Model training with SIMD

- Training Time (200M uint32_t keys between [1,200M+1])



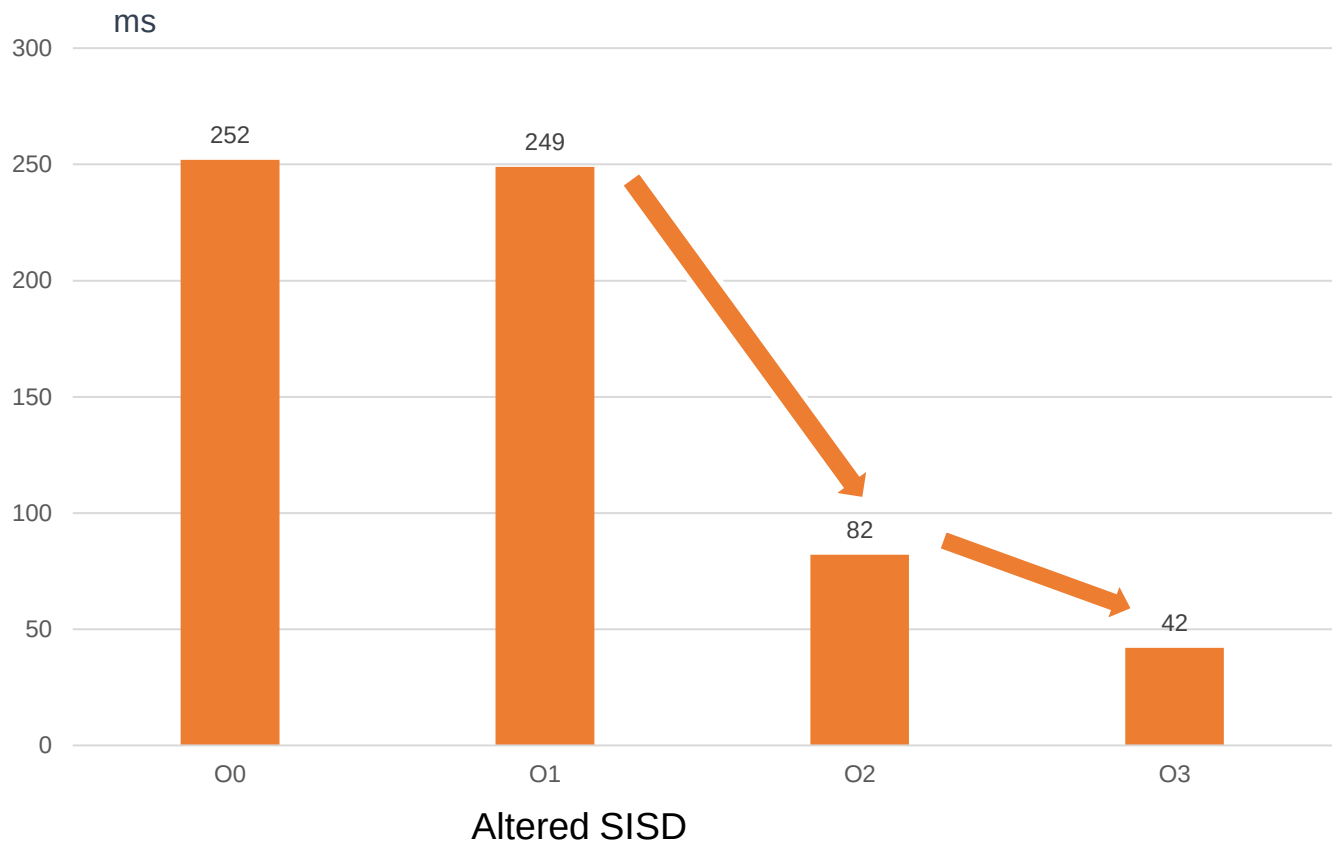
- The Altered SISD algorithm will have a performance improvement of about **8 times**
- The Altered SIMD algorithm will have a performance improvement of about **2 times**

Model training with SIMD

- Training Time (200M uint32_t keys between [1,200M+1])

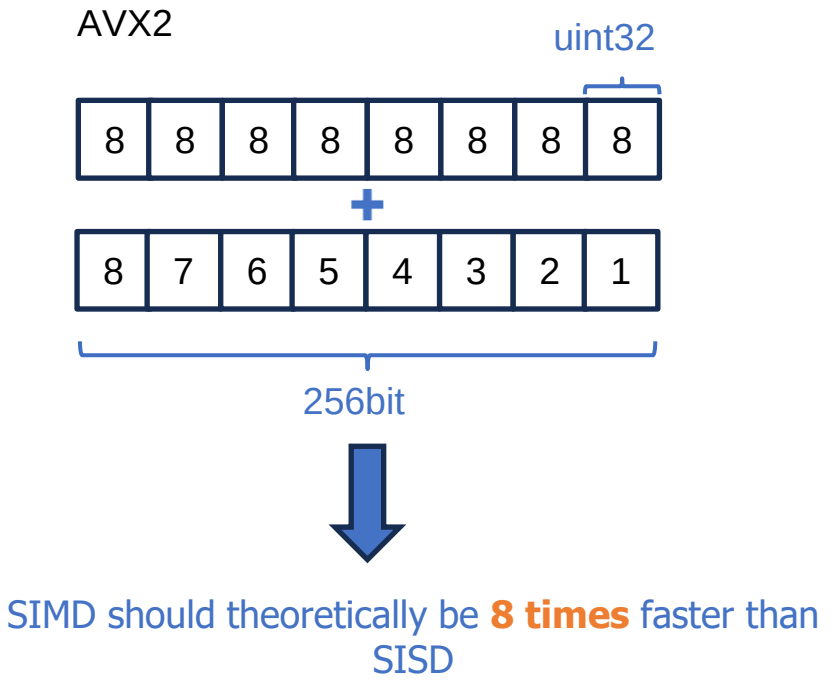
O1 -> O2 -> O3
Why does SISD performance improve?

Guess branch & Vector(SIMD)

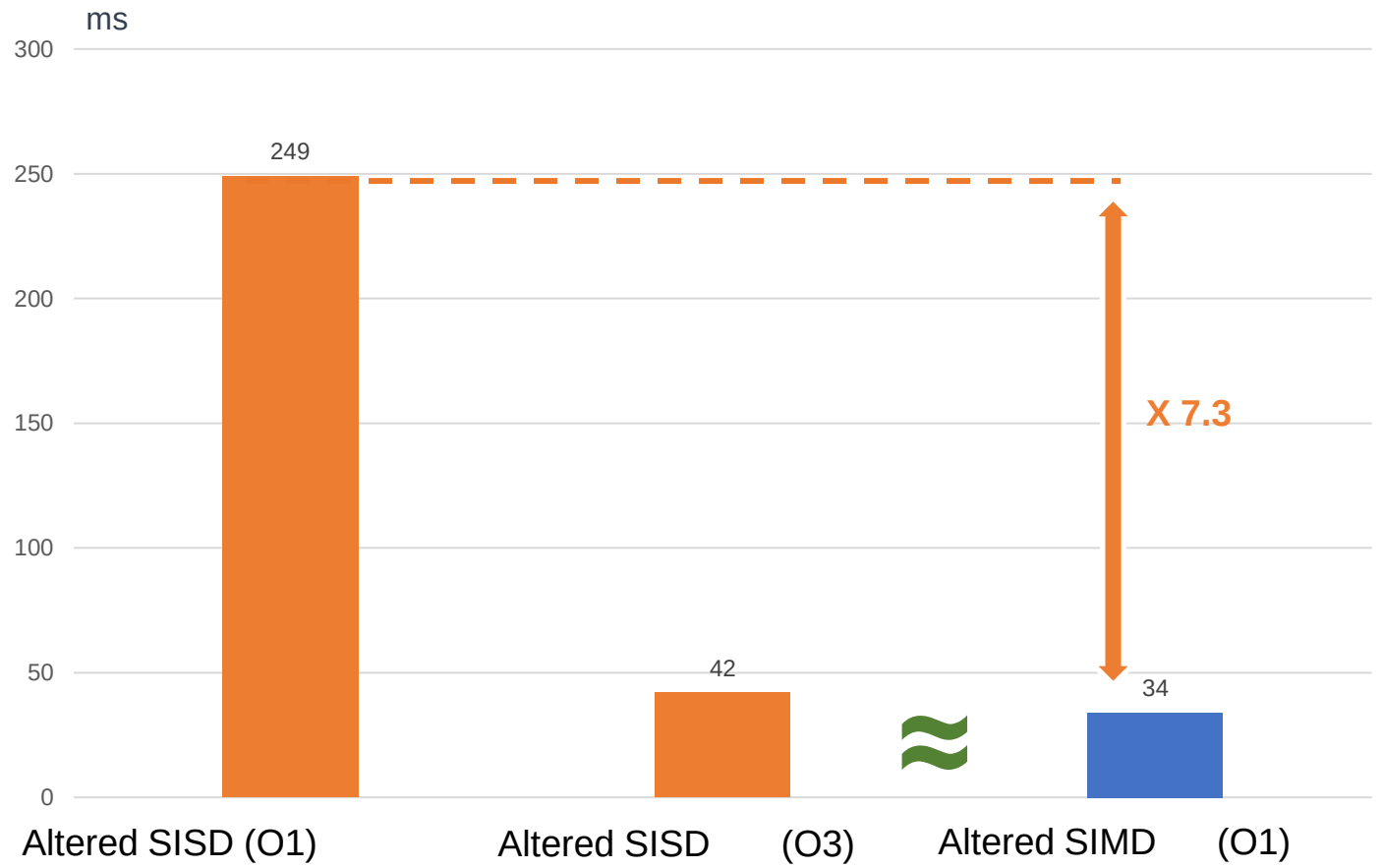


Model training with SIMD

- Training Time (200M uint32_t keys between [1,200M+1])



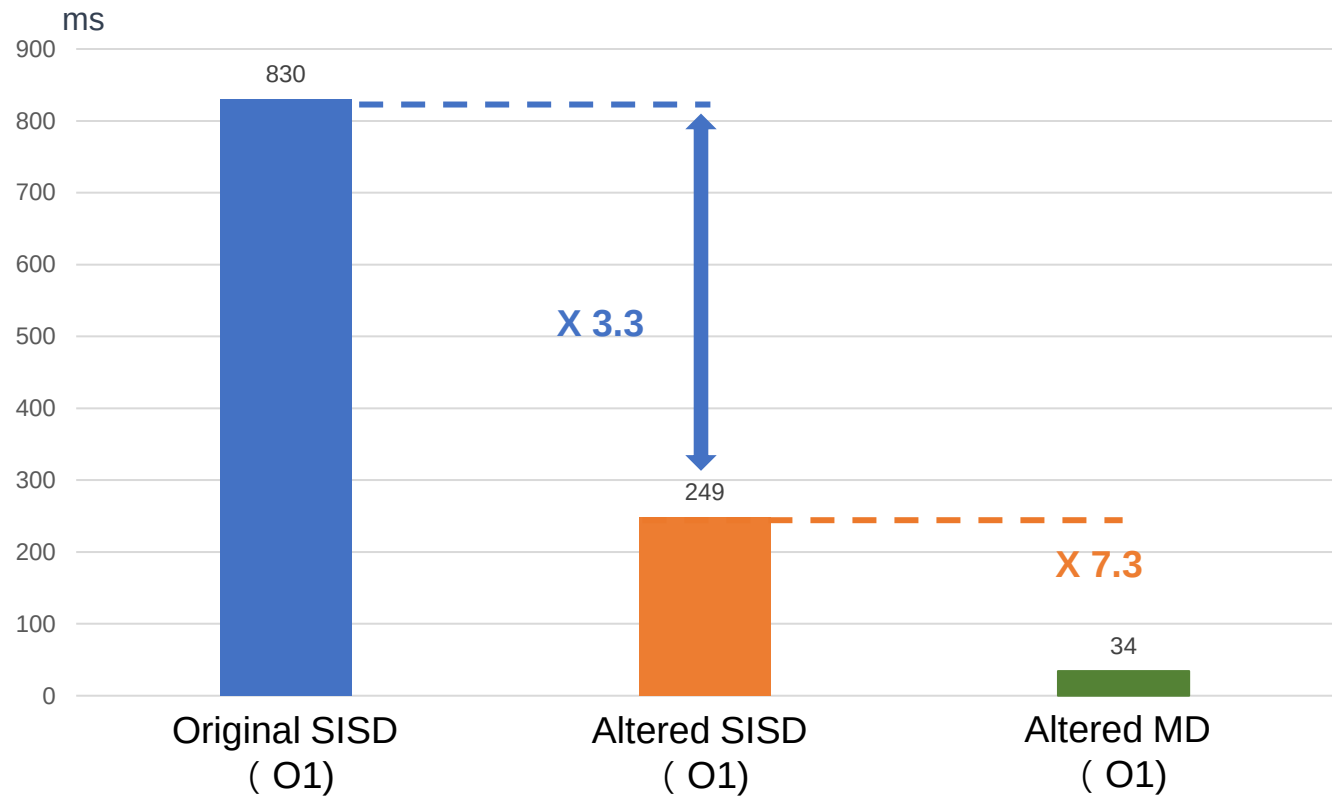
SIMD should theoretically be **8 times** faster than SISD



Model training with SIMD

- Training Time (200M uint32_t keys between [1,200M+1])
  - Analysis

- The new algorithm can provide more than **3 times** the performance. (O1)
- Provided over **24 times** performance after SIMD modification. (O1)



Future work

- Fatal Disadvantages

$$\hat{\alpha} = \frac{\sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i y_i}{n \sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}$$

$$\hat{\beta} = \frac{n \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n \sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}$$

$$\Rightarrow \sum_{i=1}^{n} y_i \quad \sum_{i=1}^{n} x_i^2 \quad \sum_{i=1}^{n} x_i \quad \sum_{i=1}^{n} x_i y_i$$

uint32 -> MAX = $4.294 * 10^9$

uint64 -> MAX = $1.844 * 10^{19}$

$\sum_{i=1}^{n} x_i^2$

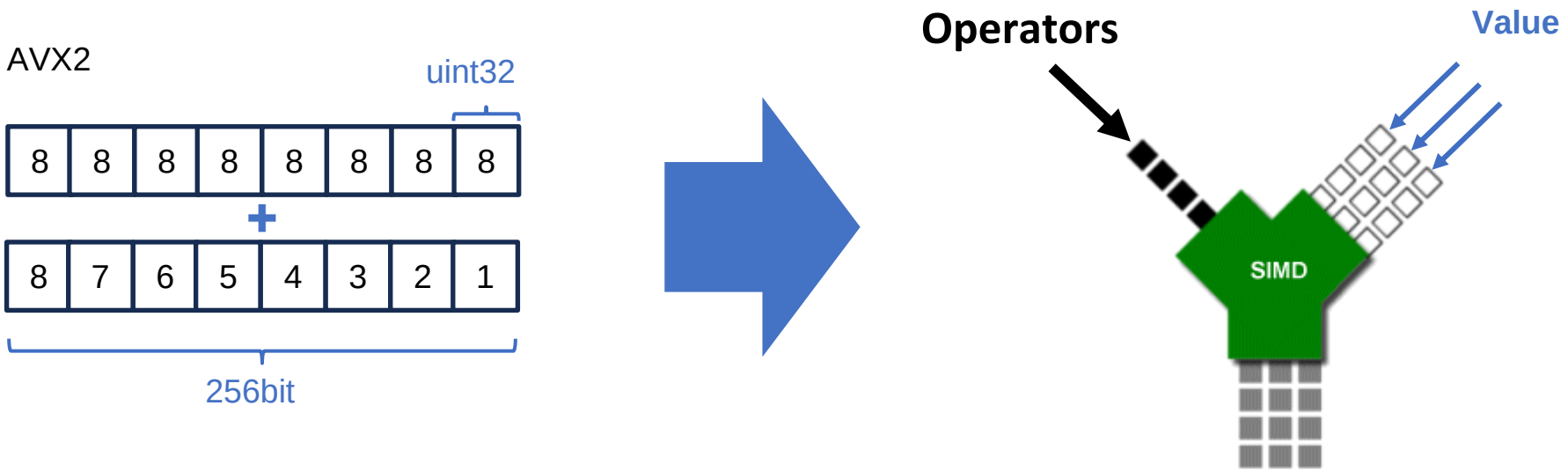
$\sum_{i=1}^{n} x_i y_i$

$2^{64} \times 2^{64}$

uint64

Future work

- Change our way of thinking



The SIMD we build is actually calculated repeatedly using known variables and operators

Future work

- Change our way of thinking

```
for (std::size_t i = 0; i != n; ++i) {
    auto x = *(first + i);
    std::size_t y = offset + i;

    double dx = x - mean_x;
    mean_x += dx / (i + 1);
    mean_y += (y - mean_y) / (i + 1);
    c += dx * (y - mean_y);

    double dx2 = x - mean_x;
    m2 += dx * dx2;
```

Involving too many iterative calculations

Excessive non independent data

Future work

- Change our way of thinking

```
for (std::size_t i = 0; i != n; ++i) {
    auto x = *(first + i);
    std::size_t y = offset + i;

    double dx = x - mean_x;
    mean_x += dx /  (i + 1);
    mean_y += (y - mean_y) / (i + 1);
    c += dx * (y - mean_y);

    double dx2 = x - mean_x;
    m2 += dx * dx2;
```

```
auto x = *(first + i);
std::size_t y = offset + i;

double dx = x - mean_x;
mean_x += dx /  (i + 1);
mean_y += (y - mean_y) / (i + 1);
c += dx * (y - mean_y);

double dx2 = x - mean_x;
m2 += dx * dx2;
```

```
auto x = *(first + i);
std::size_t y = offset + i;

double dx = x - mean_x;
mean_x += dx /  (i + 1);
mean_y += (y - mean_y) / (i + 1);
c += dx * (y - mean_y);

double dx2 = x - mean_x;
m2 += dx * dx2;
```

```
auto x = *(first + i);
std::size_t y = offset + i;

double dx = x - mean_x;
mean_x += dx /  (i + 1);
mean_y += (y - mean_y) / (i + 1);
c += dx * (y - mean_y);

double dx2 = x - mean_x;
m2 += dx * dx2;
```

So let them calculate side by side.

Revisiting HW parallelism in learned index

Oh Yeojin, ZhuYongjie, Boseung Kim, Minguk Choi
{yeojinoh, aeashio1111, bskim1102, mgchoi}@dankook.ac.kr

DANKOOK UNIVERSITY
Dankook University System Software Laboratory

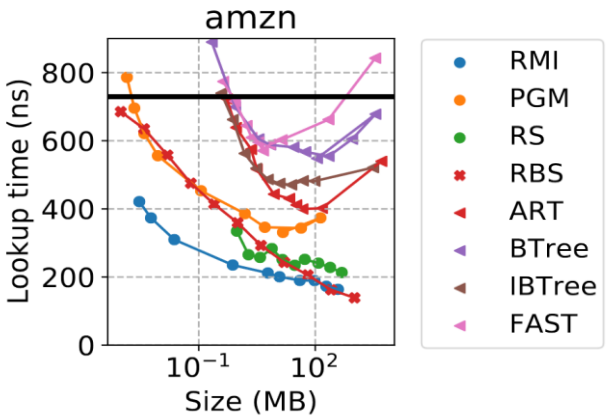
Design

- Goal
  - Learned Index에 ML 모델의 HW parallelism을 충분히 활용하자.
- Design
  - Parallel training + inference → index build time을 감소
    - Sampling과 orthogonal하므로, SIMD + Sampling 가능
  - Parallel inference + search algorithm → batch lookup에서의 index latency 감소
  - 일반적인 Index 사용환경을 고려하여, GPU가 아닌 CPU parallelism인 SIMD 활용

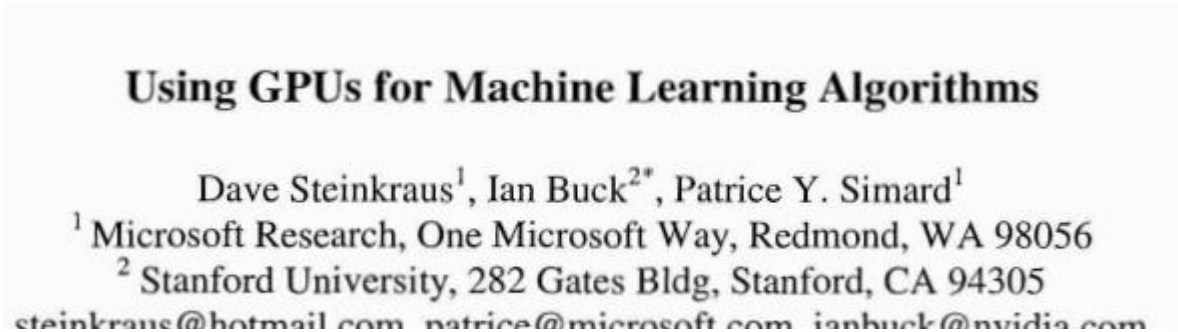
		Learned Index Construction (RMI/ALEX)			
Operations		Build		Lookup	
Point of View	Index	Training	Error bound Estimation Model-based Insertion	Prediction	Correction
	Model	Training	Inference	Inference	Correction
Optimization	Sampling	O	X	X	X
	Parallelism	O	O	O	O

Motivation

1. Learned Index는 인덱스에 ML를 적용한 새로운 인덱스 자료구조
   - Learned Index는 ML 모델의 key-distribution을 학습하는 특성을 통해, 공간 대비 높은 탐색 성능을 보임.
   - 하지만 **ML 모델의 HW parallelism이 가능한 특성**은 아직까지 충분히 활용하지 못 함



High lookup performance



Using GPUs for Machine Learning Algorithms

Dave Steinkraus[1], Ian Buck[2*], Patrice Y. Simard[1]
[1] Microsoft Research, One Microsoft Way, Redmond, WA 98056
[2] Stanford University, 282 Gates Bldg, Stanford, CA 94305

Efficient SIMD Implementation for Accelerating Convolutional Neural Network

Sung-Jin Lee
Dept. of Electronics Engineering
Hangdang-dong, Seongdong-gu
Seoul, Korea
82-2-2220-4701
leesky601@hanyang.ac.kr

Sang-Soo Park
Dept. of Electronics Engineering
Hangdang-dong, Seongdong-gu
Seoul, Korea
82-2-2220-4701
po092000@hanyang.ac.kr

Ki-Seok Chung
Dept. of Electronics Engineering
Hangdang-dong, Seongdong-gu
Seoul, Korea
82-2-2220-4701
kchung@hanyang.ac.kr

Motivation

		Learned Index Construction (RMI/ALEX)			
Operations		Build		Lookup	
Point of View	Index	Training	Error bound Estimation Model-based Insertion	Prediction	Correction
	Model	Training			

2. 기존 Learned Index는 **parallel training** 특성을 활용하지 않음.

- 이전에는 model training으로 인한 긴 build time을 해결하기 위해:
  - 1) sampling
  - 2) light-weight model
  - 3) multi-threading 등의 기법을 사용함.
- 하지만 현재 구현하는 SIMD + RMI는 data overflow, 기존 알고리즘을 단순히 SIMD 적용으로 novelty가 부족함

Motivation

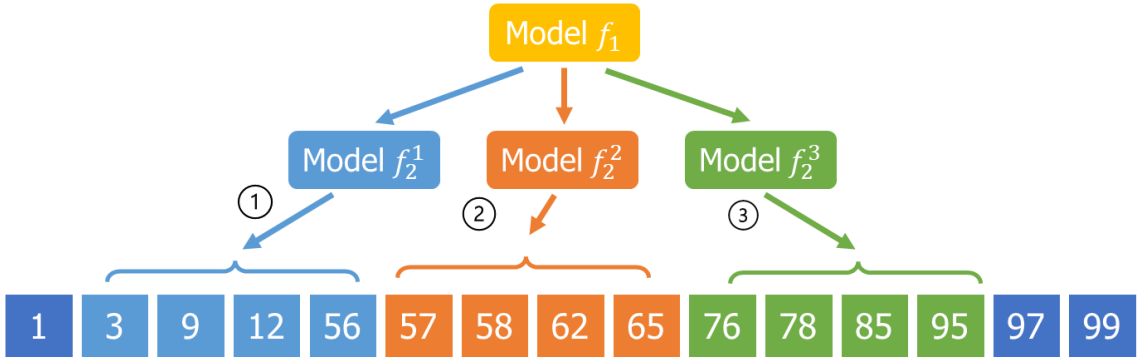
		Learned Index Construction (RMI/ALEX)			
Operations		Build		Lookup	
Point of View	Index	Training	Error bound Estimation Model-based Insertion	Prediction	Correction
	Model	Training	Inference	Inference	Correction

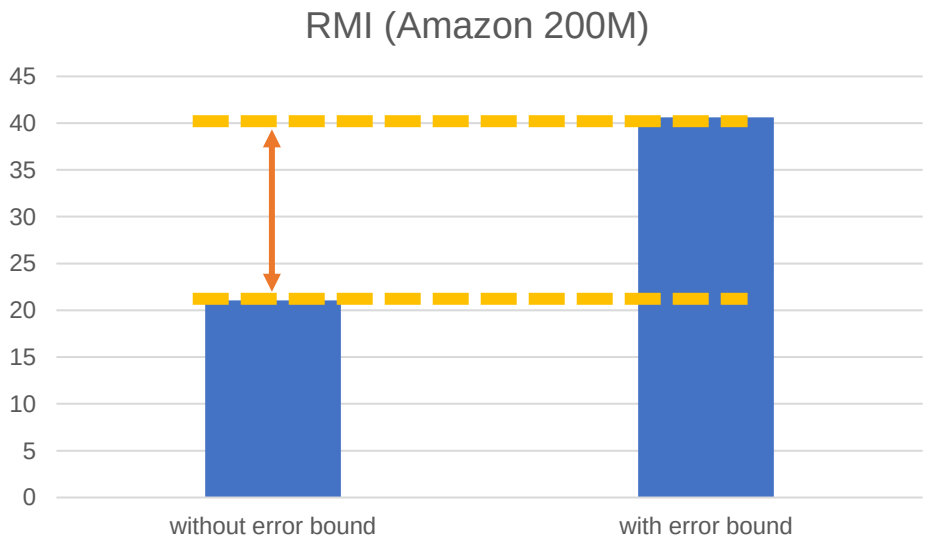
2. 기존 Learned Index는 기존 Learned Index는 **parallel training** 특성을 활용하지 않음.

(1) Build 시, model training 뿐만 inference도 진행함.

- Error-bound estimation of read-only learned index (e.g., RMI)
- Model-based insertion of updatable learned index (e.g., ALEX, LIPP, SALI)

→ parallel inference를 통해 Index Construction time 감소 가능





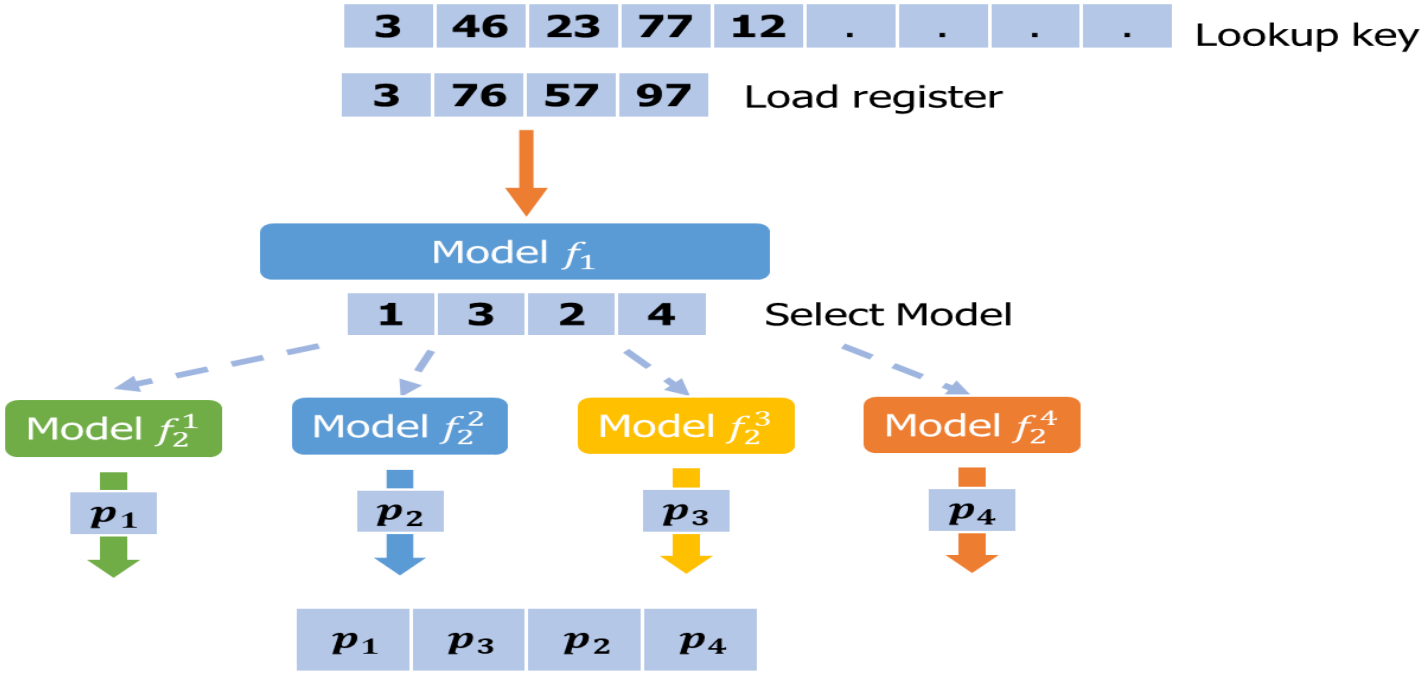
Motivation

		Learned Index Construction (RMI/ALEX)			
Operations		Build		Lookup	
Point of View	Index	Training	Error bound Estimation Model-based Insertion	Prediction	Correction
	Model	Training	Inference	Inference	Correction

3. 기존 Learned Index는 **parallel inference를 활용한 사례가 존재...**

(1) Prediction 시, 여러 개의 key를 batch + parallel하게 predict할 수 있음

- SIMD로 parallel prediction한 논문이 존재 (XIndex-R)
- GPU로 parallel prediction한 논문이 있으나, 일반적으로 Index를 위해 GPU라는 별도의 하드웨어를 사용하지 않음.
  - GPU를 사용하는 경우, Memory Bus로 인해 Single Lookup Latency 증가, Index Size 증가, Build Time 증가 (parallel training 적용 x)



Motivation

		Learned Index Construction (RMI/ALEX)			
Operations		Build		Lookup	
Point of View	Index	Training	Error bound Estimation Model-based Insertion	Prediction	Correction
	Model	Training	Inference	Inference	Correction

3. 기존 Learned Index는 **parallel inference를 활용한 사례가 존재...**

Prediction 시, 여러 개의 key를 batch + parallel하게 predict할 수 있음

- SIMD로 parallel prediction한 논문이 존재 (XIndex-R)
- GPU로 parallel prediction한 논문이 있으나, 일반적으로 Index를 위해 GPU라는 별도의 하드웨어를 사용하지 않음.
  - GPU를 사용하는 경우, Memory Bus로 인해 Single Lookup Latency 증가, Index Size 증가, Build Time 증가 (parallel training 적용 x)

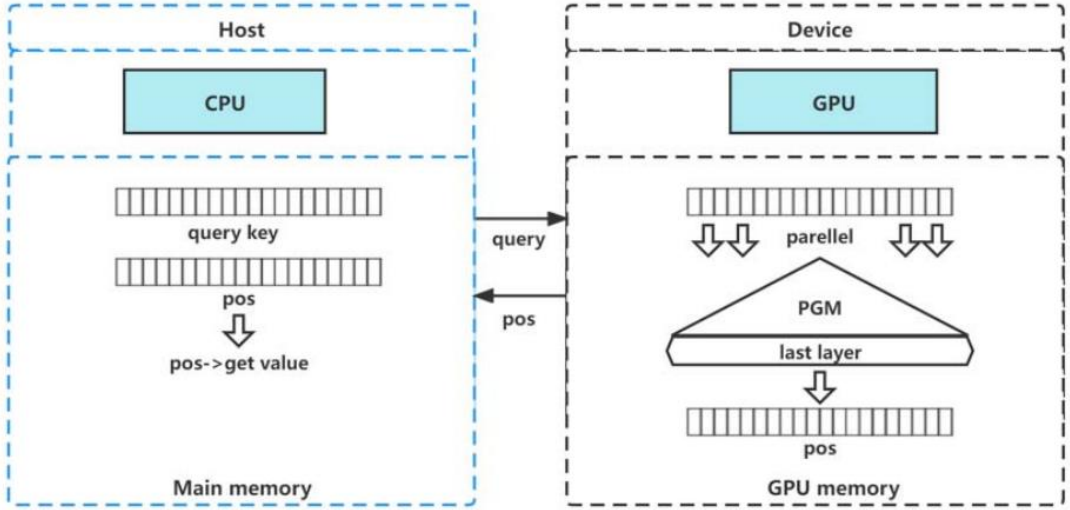


Fig. 2. GPU-PGM structure

SIMD in FINEdex

FINEdex / include / util.h

Code | Blame | 459 lines (403 loc) · 12.4 KB | Code 55% faster with GitHub Copilot

```
186  static int linear_search(const int *arr, int n, int key) {
194  }
195
196  static int linear_search_avx (const int *arr, int n, int key) {
197    __m256i vkey = _mm256_set1_epi32(key);
198    __m256i cnt = _mm256_setzero_si256();
199    for (int i = 0; i < n; i += 16) {
200      __m256i mask0 = _mm256_cmpgt_epi32(vkey, _mm256_loadu_si256((__m256i *)&arr[i+0]));
201      __m256i mask1 = _mm256_cmpgt_epi32(vkey, _mm256_loadu_si256((__m256i *)&arr[i+8]));
202      __m256i sum = _mm256_add_epi32(mask0, mask1);
203      cnt = _mm256_sub_epi32(cnt, sum);
204    }
205    __m128i xcnt = _mm_add_epi32(_mm256_extracti128_si256(cnt, 1), _mm256_castsi256_si128(cnt));
206    xcnt = _mm_add_epi32(xcnt, _mm_shuffle_epi32(xcnt, SHUF(2, 3, 0, 1)));
207    xcnt = _mm_add_epi32(xcnt, _mm_shuffle_epi32(xcnt, SHUF(1, 0, 3, 2)));
208    return _mm_cvtsi128_si32(xcnt);
209  }
210
```

Motivation

		Learned Index Construction (RMI/ALEX)			
Operations		Build		Lookup	
Point of View	Index	Training	Error bound Estimation Model-based Insertion	Prediction	Correction
	Model	Training	Inference	Inference	Correction

3. 기존 Learned Index는 **parallel inference를 활용한 사례가 존재...**

(2) Correction 시, Binary/Exponential Search 또한 HW parallelism을 활용할 수 있음

- LISA : However, once we have less than 8 elements left to search, we compare with all of them simultaneouly using SIMD instructions.
- FINDEX : Instead of using the binary searching in the prediction range and level bins, we optimize the search performance with SIMD instructions, i.e., Intel AVX2, which processes 256-bit data with one instruction.

Listing 2 Modified Hunt and Locate

```
hunt_n_locate(X, n, Y, m)
  lowbounds[m]
  for(int i=0; i<m; i++):
    if Y[i] <X[0] :
        lowbounds[i] =0
    else if Y[i] >X[n-1] :
        lowbounds[i] =n-1
    else:
        j=1
        start=0
        end=1
        while end<n && Y[i] >X[end] :
            start=end
            end<<=1
        // Binary search within bounds
        end=min(end, n-1)
        start=binary_search(
            max(start, 0), min(end, n-1),
            x, target)
        lowbounds[i] =start
    return lowbounds
```

Listing 3 Branchless binary search

```
binary_search(X, n, Y, m)
    lowbounds[m]
    for(int i=0; i<m; i++):
        lower=0
        upper=n
        // Search
        while((upper - lower) >1):
            mid=(lower+upper) / 2
            condition=Y[i] <X[mid]
            upper=bchoice(condition, mid, upper)
            lower=bchoice(!condition, mid, lower)
        // Bounds checking
        lower=bchoice(Y[i] <X[0] , 0, lower)
        lower=bchoice(Y[i] >X[n-1] , n-1, lower)
        lowbounds[i] =lower
    return lowbounds
```

Listing 4 A skiplist implementation of a binary search

```
skiplist_search(X, n, Y, m)
    skplst[n/8] , lowbounds[m]
    for(i=0; i<n/8; i++):
        skplst[i] =X[8 * i]
    for(i=0; i<m; i++):
        j=binary_search(skplst, Y[i])
        upper=n
        if j<n/8-1:
            upper=(j+1)*8
        lower=linear_search(X, 8*j, upper, Y[i])
        // Bounds checking
        lower=bchoice(Y[i] <X[0], 0, lower)
        lower=bchoice(Y[i] >X[n-1] , n-1, lower)
        lowbounds[i] =lower
    return lowbounds
```

Future Work

- 1. Learned Index에 Parallel Training/Inference/Search를 적용한 논문 찾기

- 2. RMI에 SIMD 구현
  - Parallel Training → Internal Model Training
  - Parallel Inference → Error-Bound Estimation, Batch Lookup
  - Parallel Search Algorithm → Correction Search Algorithm

- 3. SIMD 외에 Novelty를 갖을 수 있는 Design Points 생각해보기
  - 현재 Motivation/Design/Evaluation 이 너무 단순함.
    - SIMD 적용 X → SIMD 적용

  - 추가적인 SIMD 적용 points : 1) Updatable Index의 Model-Based Insertion, 2) Sampling + SIMD
    - 위 2가지 포인트들 또한 너무 단순함 → HW parallelism에서 벗어나야 하는가?

Thank you

2024. 02. 14

Presentation by Yeojin Oh, ZhuYongjie, Boseung Kim

yeojinoh@dankook.ac.kr, aeashio1111@dankook.ac.kr, bskim1102@dankook.ac.kr