Revisiting HW Parallelism in Learned Indexes

Yandong Mao, et al. EuroSys'12

2024. 02. 28

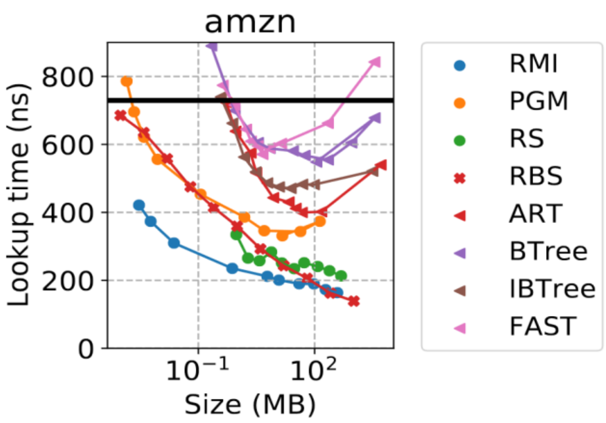
Presentation by Yeojin Oh, ZhuYongjie, Boseung Kim

yeojinoh@dankook.ac.kr, arashio1111@dankook.ac.kr, bskim1102@dankook.ac.kr

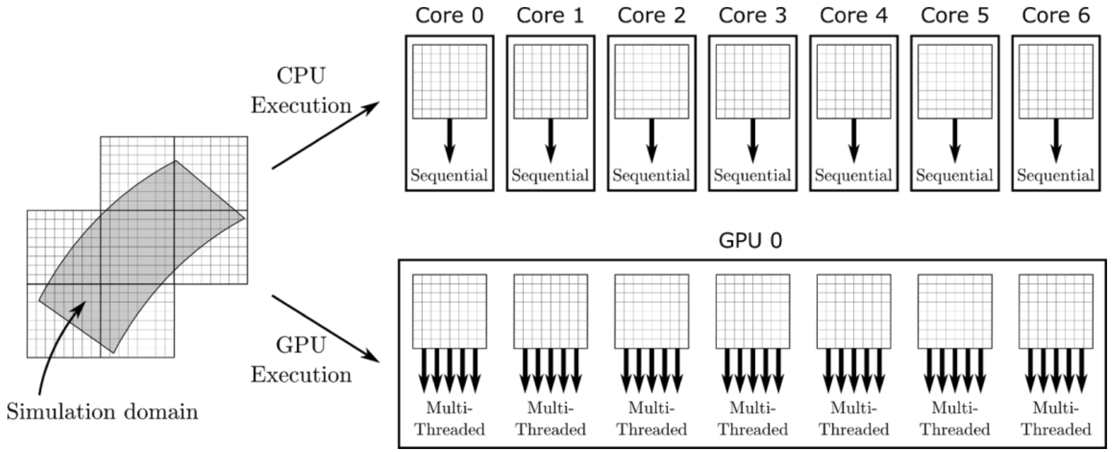
Contents

Motivation

- Learned Index는 인덱스에 ML을 적용한 새로운 인덱스 자료구조
  - Learned Index 는 ML 모델의 key-distribution을 학습하는 특성을 통해, 공간 대비 높은 탐색 성능을 보임
  - 하지만 ML 모델의 **HW parallelism이 가능한 특성은** 아직까지 충분히 활용하지 못 함



High lookup performance



Convolutional Neural Network (CNN)

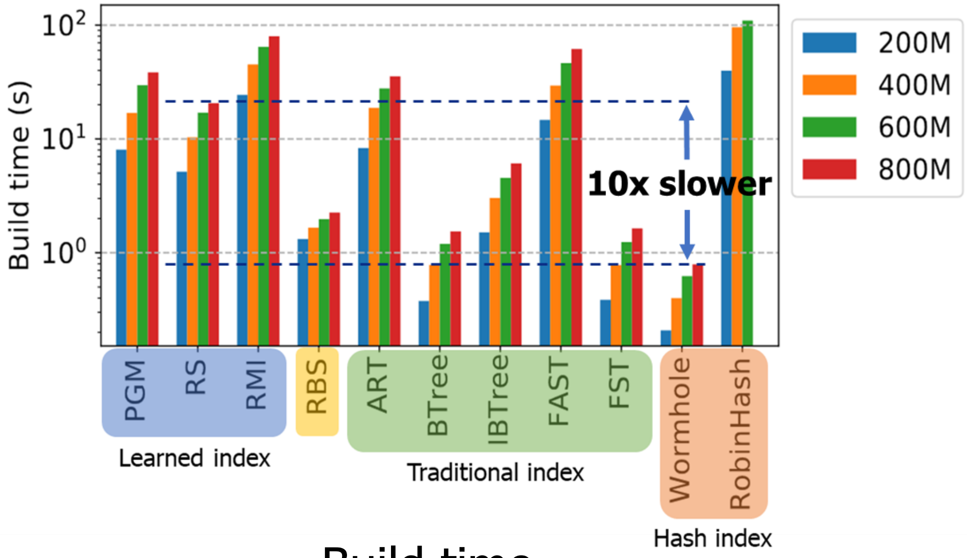
Motivation

- 하지만 learned index에는 ML모델의 HW parallelism 을 적용할 요소가 많음

Operation		Learned index construction (RMI/ALEX)			
		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

- 기존 Learned Index는 **parallel training** 특성을 활용하지 **앉음**
  - 이전에는 model training으로 인한 긴 build time을 해결하기 위해: 1) sampling, 2) light-weight model, 3) multi-threading 등의 기법을 사용함
  - SIMD, GPU를 통한 parallel training 을 사용하지 않았음



Build time

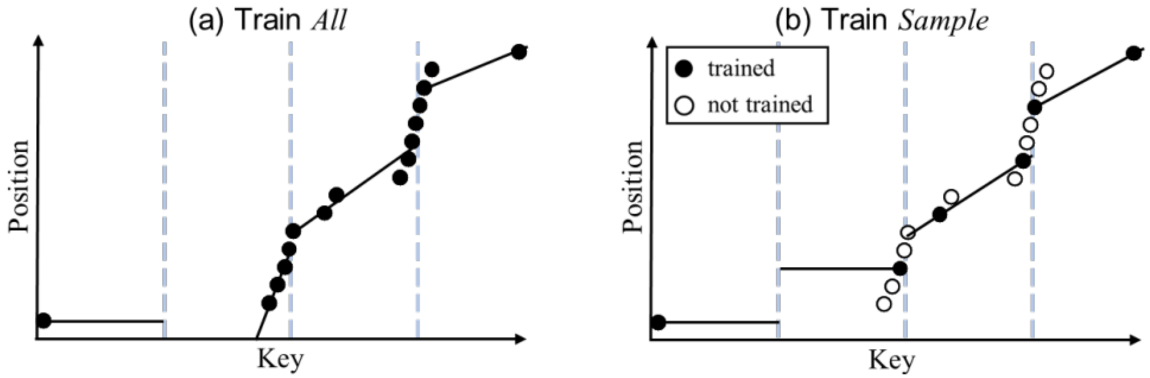
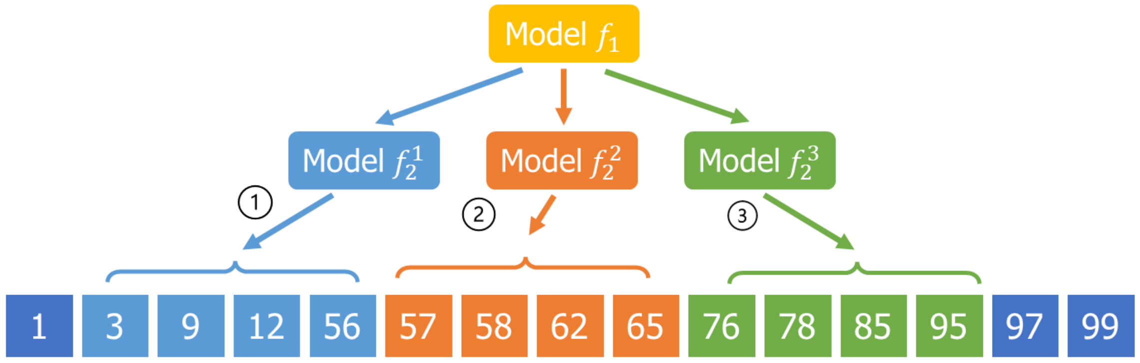


Figure 4: An example for sampling applied RMI.

Motivation

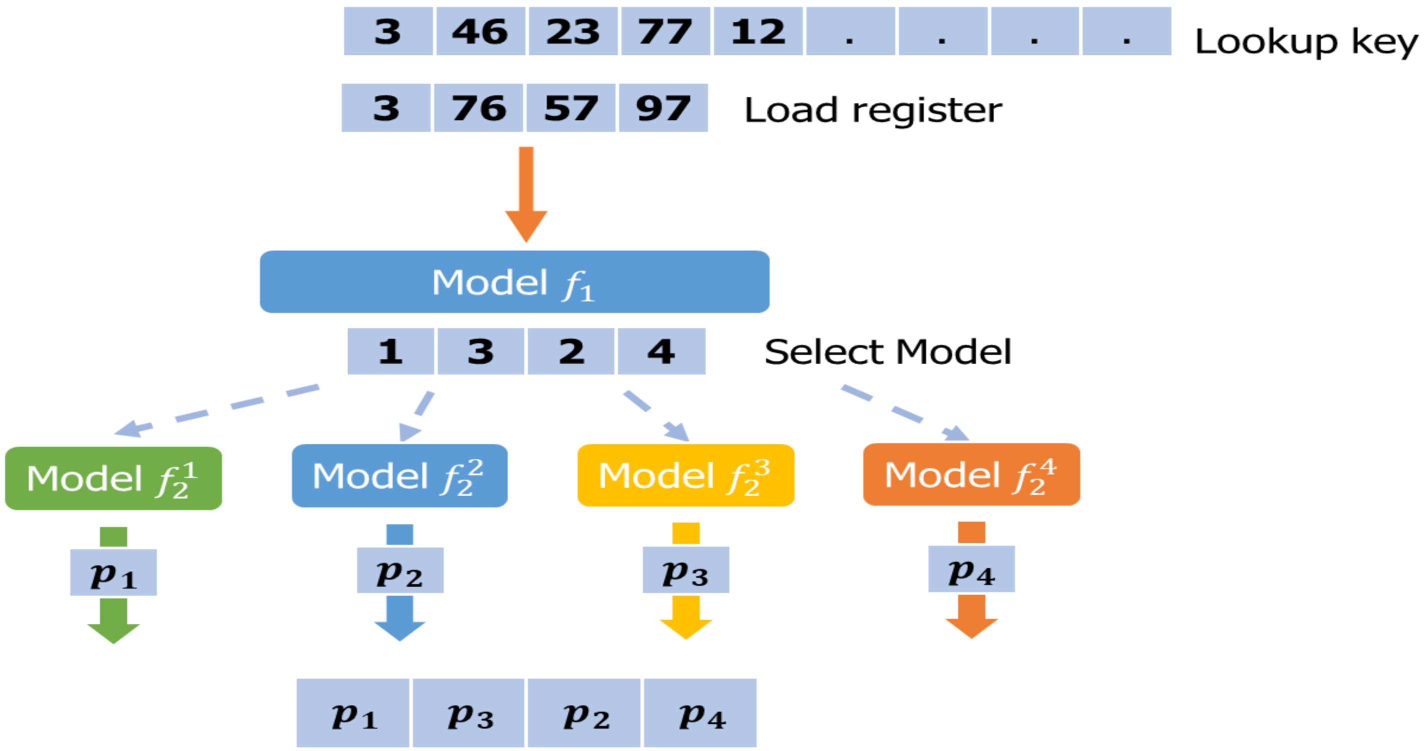
- 기존 learned index는 parallel inference를 활용하지 않음
  - Build시, model training 뿐 만 아닌, inferenc도 진행함.
    - Error-bound estimation of read-only learned index (e.g. RMI)
    - Model-based insertion of updatable learned index (e.g. ALEX, LIPP, SALI)

    → Parallel inference 를 통해 Index Construction time 감소 가능



Motivation

- 기존 Learned Index에 **parallel inference**를 활용한 경우도 **존재**
  - Prediction 시, 여러 개의 key를 batch + parallel하게 predict할 수 있음
    - SIMD로 parallel prediction한 논문이 존재 (XIndex-R)



Motivation

- 기존 Learned Index가 **parallel search**를 활용한 경우도 **존재**
  - Correction시, Binary/Exponential Search 또한 HW parallelism을 활용할 수 있음
    - LISA, FINDEX

SIMD in FINEdex

FINEdex / include / util.h

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

```c
static int linear_search(const int *arr, int n, int key) {
}

static int linear_search_avx (const int *arr, int n, int key) {
  __m256i vkey = _mm256_set1_epi32(key);
  __m256i cnt = _mm256_setzero_si256();
  for (int i = 0; i < n; i += 16) {
    __m256i mask0 = _mm256_cmpgt_epi32(vkey, _mm256_loadu_si256((__m256i *)&arr[i+0]));
    __m256i mask1 = _mm256_cmpgt_epi32(vkey, _mm256_loadu_si256((__m256i *)&arr[i+8]));
    __m256i sum = _mm256_add_epi32(mask0, mask1);
    cnt = _mm256_sub_epi32(cnt, sum);
  }
  __m128i xcnt = _mm_add_epi32(_mm256_extracti128_si256(cnt, 1), _mm256_castsi256_si128(cnt));
  xcnt = _mm_add_epi32(xcnt, _mm_shuffle_epi32(xcnt, SHUF(2, 3, 0, 1)));
  xcnt = _mm_add_epi32(xcnt, _mm_shuffle_epi32(xcnt, SHUF(1, 0, 3, 2)));
  return _mm_cvtsi128_si32(xcnt);
}
```

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
```

Design – Parallel Training

- Issue
  - Algorithm
  - Accuracy
  - Performance (Training Time)

Design

Algorithms for calculating variance

文A 4 languages

Article Talk | Read Edit View history Tools

From Wikipedia, the free encyclopedia

Algorithms for calculating variance play a major role in computational statistics. A key difficulty in the design of good algorithms for this problem is that formulas for the variance may involve sums of squares, which can lead to numerical instability as well as to arithmetic overflow when dealing with large values.

Naïve algorithm [edit]

A formula for calculating the variance of an entire population of size N is:

$$\sigma^2 = \overline{(x^2)} - \bar{x}^2 = \frac{\sum_{i=1}^N x_i^2 - (\sum_{i=1}^N x_i)^2/N}{N}.$$

Using Bessel's correction to calculate an unbiased estimate of the population variance from a finite sample of n observations, the formula is:

$$s^2 = \left(\frac{\sum_{i=1}^n x_i^2}{n} - \left(\frac{\sum_{i=1}^n x_i}{n}\right)^2\right) \cdot \frac{n}{n-1}.$$

Welford's online algorithm [edit]

It is often useful to be able to compute the variance in a single pass, inspecting each value x_i only once; for example, when the data is being collected without enough storage to keep all the values, or when costs of memory access dominate those of computation. For such an online algorithm, a recurrence relation is required between quantities from which the required statistics can be calculated in a numerically stable fashion.

The following formulas can be used to update the mean and (estimated) variance of the sequence, for an additional element x_n. Here, $\bar{x}_n = \frac{1}{n}\sum_{i=1}^n x_i$ denotes the sample mean of the first n samples $(x_1, \dots, x_n)$, $\sigma_n^2 = \frac{1}{n}\sum_{i=1}^n (x_i - \bar{x}_n)^2$ their biased sample variance, and $s_n^2 = \frac{1}{n-1}\sum_{i=1}^n (x_i - \bar{x}_n)^2$ their unbiased sample variance.

$$\bar{x}_n = \frac{(n-1)\,\bar{x}_{n-1} + x_n}{n} = \bar{x}_{n-1} + \frac{x_n - \bar{x}_{n-1}}{n}$$

$$\sigma_n^2 = \frac{(n-1)\,\sigma_{n-1}^2 + (x_n - \bar{x}_{n-1})(x_n - \bar{x}_n)}{n} = \sigma_{n-1}^2 + \frac{(x_n - \bar{x}_{n-1})(x_n - \bar{x}_n) - \sigma_{n-1}^2}{n}.$$

$$s_n^2 = \frac{n-2}{n-1}\,s_{n-1}^2 + \frac{(x_n - \bar{x}_{n-1})^2}{n} = s_{n-1}^2 + \frac{(x_n - \bar{x}_{n-1})^2}{n} - \frac{s_{n-1}^2}{n-1}, \quad n > 1$$

Design

Algorithms for calculating variance

文A 4 languages

Article Talk

Read Edit View history Tools

From Wikipedia, the free encyclopedia

Algorithms for calculating variance play a major role in computational statistics. A key difficulty in the design of good algorithms for this problem is that formulas for the variance may involve sums of squares, which can lead to numerical instability as well as to arithmetic overflow when dealing with large values.

WIKIPEDIA
The Free Encyclopedia

	RMI	Ours
알고리즘	Welford's online	Naïve algorithm
분산 계산	$\sum (Key - Average)^2$	$\sum Key^2$
Data overflow	X	O
Data Constraints	X	O
Accuracy	O	근사치 계산이 가능한 지? 아예 다른 값이 나오는 지?
Computing Time	Long	Short
SIMD 적용 가능여부	O	O

Design

- Parallel Welford's online algorithm

Welford's online algorithm [edit]

It is often useful to be able to compute the variance in a single pass, inspecting each value x_i only once; for example, when the data is being collected without enough storage to keep all the values, or when costs of memory access dominate those of computation. For such an online algorithm, a recurrence relation is required between quantities from which the required statistics can be calculated in a numerically stable fashion.

The following formulas can be used to update the mean and (estimated) variance of the sequence, for an additional element x_n. Here, $\bar{x}_n = \frac{1}{n}\sum_{i=1}^{n} x_i$ denotes the sample mean of the first n samples $(x_1, \dots, x_n)$, $\sigma_n^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x}_n)^2$ their biased sample variance, and $s_n^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x}_n)^2$ their unbiased sample variance.

$$\bar{x}_n = \frac{(n-1)\,\bar{x}_{n-1} + x_n}{n} = \bar{x}_{n-1} + \frac{x_n - \bar{x}_{n-1}}{n}$$

$$\sigma_n^2 = \frac{(n-1)\,\sigma_{n-1}^2 + (x_n - \bar{x}_{n-1})(x_n - \bar{x}_n)}{n} = \sigma_{n-1}^2 + \frac{(x_n - \bar{x}_{n-1})(x_n - \bar{x}_n) - \sigma_{n-1}^2}{n}.$$

$$s_n^2 = \frac{n-2}{n-1}\,s_{n-1}^2 + \frac{(x_n - \bar{x}_{n-1})^2}{n} = s_{n-1}^2 + \frac{(x_n - \bar{x}_{n-1})^2}{n} - \frac{s_{n-1}^2}{n-1}, \quad n > 1$$

Parallel algorithm [edit]

Chan et al.[10] note that Welford's online algorithm detailed above is a special case of an algorithm that works for combining arbitrary sets A and B:

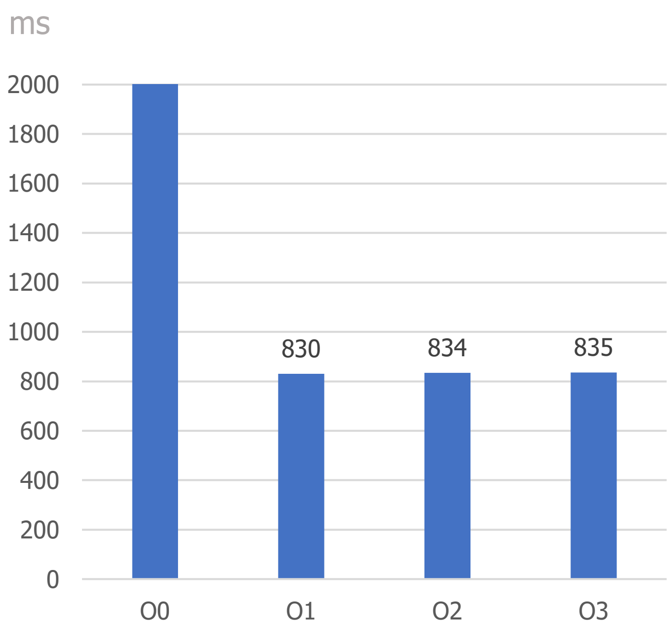
$$\begin{aligned} n_{AB} &= n_A + n_B \\ \delta &= \bar{x}_B - \bar{x}_A \\ \bar{x}_{AB} &= \bar{x}_A + \delta \cdot \frac{n_B}{n_{AB}} \\ M_{2,AB} &= M_{2,A} + M_{2,B} + \delta^2 \cdot \frac{n_A n_B}{n_{AB}} \end{aligned}.$$

This may be useful when, for example, multiple processing units may be assigned to discrete parts of the input.

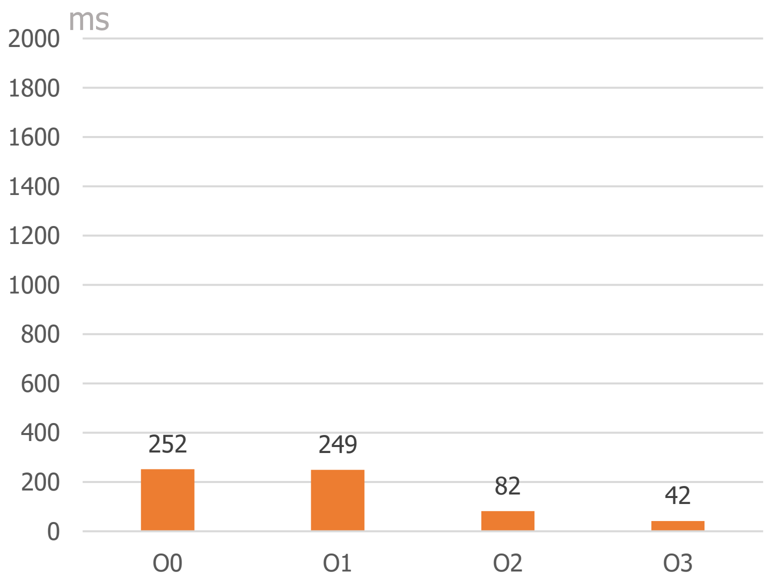
Chan's method for estimating the mean is numerically unstable when $n_A \approx n_B$ and both are large, because the numerical error in $\delta = \bar{x}_B - \bar{x}_A$ is not scaled down in the way that it is in the $n_B = 1$ case. In such cases, prefer $\bar{x}_{AB} = \frac{n_A \bar{x}_A + n_B \bar{x}_B}{n_{AB}}$.

Experiment – Parallel Training

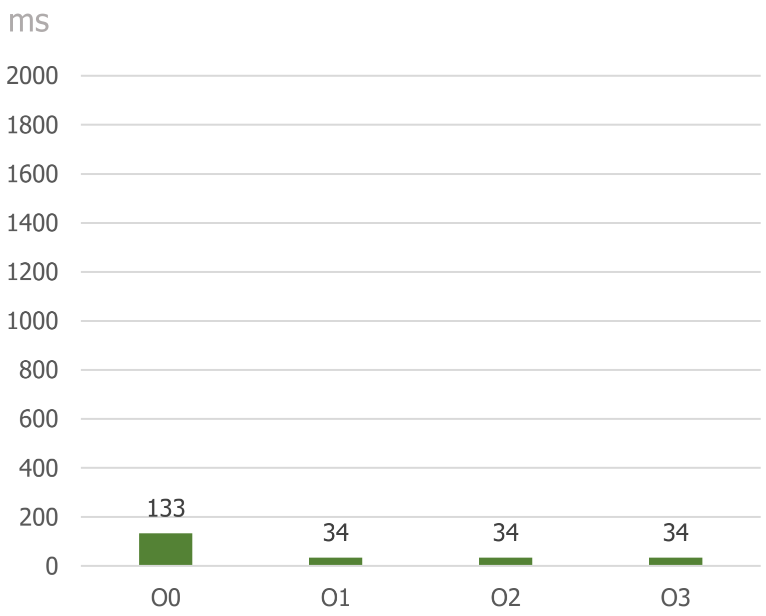
- Compare Training Algorithm with SISD/SIMD



(RMI) Welford’s online
+SISD



(Ours)Naïve Algorithm
+ SISD



(Ours) Naïve Algorith
m + SIMD

Future Work

1. Issues

    1) Naïve Algorithm Accuarcy

    - Approximate Computing이 가능한지, 혹은 아예 제대로 된 값을 구할 수 없는 것인지 확인
    - Experiment : RMI with Naïve algorithm on real-world dataset & workload

    2. Training Algorithm Performance Comparison

    - (RMI) Welford's online with SIMD vs (Ours) Naïve algorithm with SIMD

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