Towards Parallel Learned Sorting

Ivan Carvalho
Supervisor: Dr. Ramon Lawrence

Can we do better than Quicksort?

- Performance of Sorting impacts many common operations
- Quicksort can sort an array in $\mathcal{O}(n \log n)$
  - The lower bound for comparison-based sorting is $\Omega(n \log n)$
- Quicksort is the default algorithm available in many languages/libaries (e.g. C++ STL)

Can we beat Quicksort and push the boundary of sorting performance?

IPS^{4}o

Yes, we can beat Quicksort (albeit it's not easy).

In-Place Parallel Super Scalar Sample Sort (IPS^{4}o)

- State-of-the-Art Sorting Algorithm
- Generalization of Quicksort: sorting with k pivots
- Implementation has many desirable properties

Desirable Properties of IPS^{4}o

- In-Place Partitioning

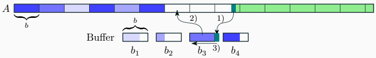


Figure 1: Example of in-place partitioning IPS^{4}o where each shade of blue represents elements assigned to a different bucket and green represents elements never visited. Instead of placing the elements directly to their position, IPS^{4}o places them into buffers and flushes the elements to the array when a buffer becomes full. After, defragmentation is executed to make the buckets contiguous. Reprinted from Axtmann et al., 2022.

- Branchless Decision Trees
  - Built from $\alpha k - 1$ samples (oversampling for k-pivot Quicksort)
  - Extremely efficient implementation of a decision tree

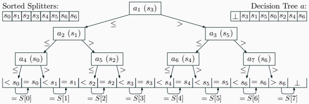


Figure 2: IPS^{4}o organizes the pivots in a decison tree structure for quickly finding the correct bucket for an element. Reprinted from Axtmann et al., 2022.

Desirable Properties of IPS^{4}o

- Parallelism
  - Custom task scheduler
  - Each task in IPS^{4}o has a parallel implementation
- Reusable Framework
  - Code can be reused to implement variants of the algorithm
  - In-place Parallel Super Scalar Radix Sort (IPS^{2}Ra)

Aside: Learned Indexes

Emerging field using machine learning to create highly efficient indexes that outperform traditional indexes (i.e. B-Trees)

Some indexes worth mentioning are:

- **Recursive Model Index**
- Piecewise Geometric Model Index
- RadixSpline
- **Updatable Learned Index with Precise Positions**

We will talk about ML-Enhanced Sorting. ML-Enhanced Sorting is intrinsically connected to Learned Indexes and reuses the models from Learned Indexes.

ML-Enhanced Sorting

New paradigm: Machine Learning Enhanced Sorting

Main idea: if there exists a model F that predicts the sorted position of a key x, we can sort the array in $\mathcal{O}(n)$ by moving each element to its correct location with $A[F(x)] = x$.

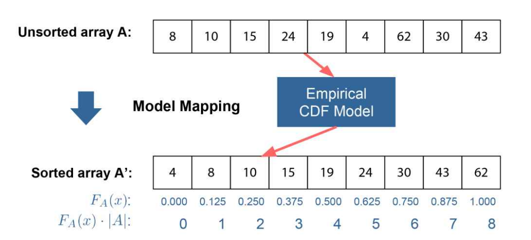


Figure 3: Ideal case of ML-enhanced with a perfect model. Reprinted from Kristo et al., 2020.

Challenges of ML-Enhanced Sorting

First challenge: $F(x)$ is not given. However, ML can overcome that: we sample data and learn $F(x)$.

But what kind of ML is useful to learn $F(x)$? One possible solution is to use Empirical Cumulative Distribute Functions (eCDF).

eCDF yields the probability $P(A \leq x)$ that an element is smaller than x, hence for an array with N:

$$pos = F(x) = \lfloor N \cdot P(A \leq x) \rfloor$$

Challenges of ML-Enhanced Sorting

There are more challenges even after we have $F(x)$:

- **Inversions**: Pair of elements with $a < b$ but $F(a) > F(b)$
- **Collisions**: Pair of elements with $F(a) = F(b)$
    - Collisions are exacerbated when there are many duplicates as it is guaranteed they will collide at $F(x)$

Learned Sort

Learned Sort 2.0: practical implementation of ML-enhanced sorting with outstanding performance

- Uses the Recursive Model Index to model the eCDF
- Performs two rounds of partitioning using the model similarly to IPS^{4}o
- Executes Insertion Sort to correct (very few) mistakes from the model

Limitations

- Cannot sort strings
- **No parallel implementation is available**

Contributions

We are the first work to interpret IPS^{4}o as a ML-enhanced algorithm, which has two consequences:

- First consequence is that models from the field of Learned Indexes can be used to create variants of IPS^{4}o
- Second consequence is that IPS^{4}o provides a framework to efficiently parallelize ML-enhanced sorting algorithms

We introduce the In-Place Parallel Learned Sorting (IPLS) algorithm to prove our point. We use a model from Learned Indexes using the IPS^{4}o framework to achieve parallel learned sorting.

Machine Learning Models for Sorting: Linear Models

Linear Models: partition based model.

Described by three parameters: the number of buckets k, the slope a and the constant term b

$$F(x) = \begin{cases} 0, & \text{if } \lfloor a \cdot x + b \rfloor < 0 \\ k - 1, & \text{if } \lfloor a \cdot x + b \rfloor \geq k \\ \lfloor a \cdot x + b \rfloor, & \text{otherwise} \end{cases}$$

Consequence of Linear Models: $x_i < x_j \rightarrow F(x_i) \leq F(x_j)$

Machine Learning Models for Sorting: Linear Models

How do we train a linear model for sorting?

Idea: minimize the maximum number of elements in a bucket.

Fastest Minimum Conflict Degree (FMCD):

- Model used by the Updatable Learned Index with Precise Positions
- Trains in $\mathcal{O}(S)$ for S samples
- Guarantees at most $S/3$ elements will be in the same partition
  - Reasonable to assume at most $N/3$ elements will be in the same bucket for the whole array as well
  - $\frac{N}{3}$ bound yields that on average $\mathcal{O}(\log N)$ recursive partition steps will be performed no matter what input is given

Proposed Algorithm

In-Place Parallel Learned Sorting (IPLS)

IPLS extends IPS^4o and has the goal to show that the framework can be used to implement a parallel version of ML-enhanced sorting:

- IPLS partitions the data in $k = 256$ buckets using linear models. It samples $\alpha k - 1$ random elements from the array with $\alpha = 0.2 \log N$.
- Then, it trains a linear model on the samples using the FMCD algorithm discussed earlier. The linear model $F(x)$ is then used to predict the bucket for each element.
- For $n \leq 2^{12}$, IPLS uses SkaSort as the base case (fast RadixSort)

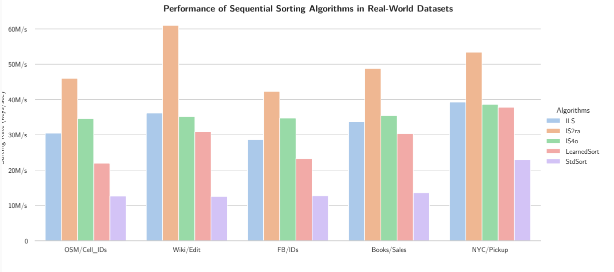
Experimental Evaluation

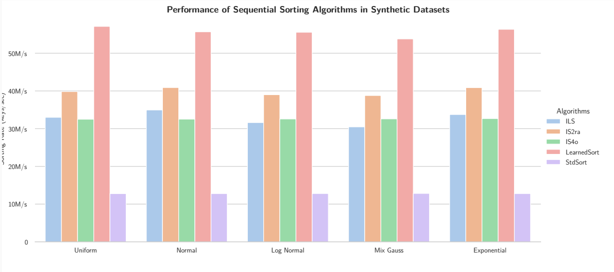
We compare the performance IPLS to other sorting algorithms on a **m5zn.metal** instance from AWS. The instance runs a Intel® Xeon® Platinum 8252C CPU @ 3.80GHz with 48 cores and 192 GB of RAM. The algorithms we compare against are:

- IPS^{4}o
- IPS^{2}Ra
- Learned Sort
- `std::sort`

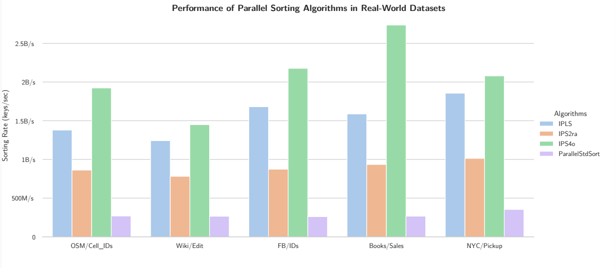
We compare both sequential and parallel settings. For the sequential case, we refer to the algorithms as IS^{4}o, IS^{2}Ra and ILS.

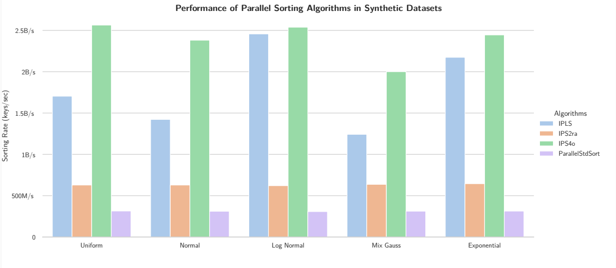
Results (Sequential)



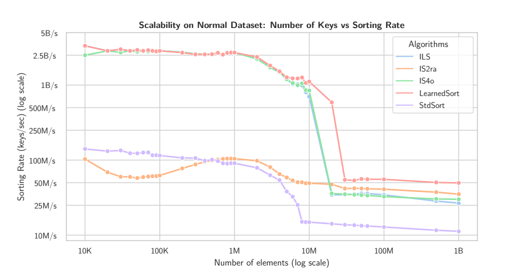


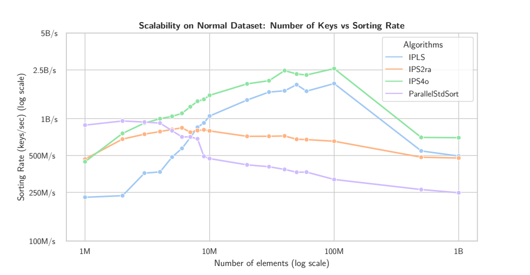
Results (Parallel)





Results (Scalability)





Analysis of Results

Sequential

- Learned Sort dominates on synthetic data and IS^{2}Ra dominates on real-world data
- ILS is competitive with IS^{4}o and wins in some datasets
- Scalability of ILS and IS^{4}o is almost identical

Parallel

- IPS^{4}o dominates on all datasets
- IPLS comes second and IPS^{2}Ra comes third
- Can interpret the results as which algorithms creates more independent subproblems
- Scalability preservers the same relative ordering for the algorithms, but the overhead of multithreading makes the parllel version slower than serial version for $n < 10^6$

Conclusion

1. IPS^{4}o provides a framework to implement parallel ML-enhanced sorting
2. We achieved parallel ML-enhanced sorting with linear models trained with FMCD
3. Advances in the field of Learned Indexes can also benefit sorting
    - Future models will benefit both applications and could potentially dethrone IPS^{4}o

Appendix

Machine Learning Models for Sorting

Sample Data $\rightarrow$ Sort Sample $\rightarrow$ Train Model $\rightarrow$ Predict on Keys

Machine Learning Models for Sorting: Concepts

Computing Budget: cost of executing all four phases from ML-enhanced sorting must be less than or equal to the the cost of executing Quicksort

CDF Based Models:

- Output is in $[0, 1)$
- Goal is to minimize the error of the CDF predictions
  - Mean-Squared Error is a common metric

Partition Based Models:

- Output is in $\{0, 1, 2, \ldots, k-1\}$
- Goal is related to minimizing either:
  - Average number of elements
  - Maximum number of elements

Machine Learning Models for Sorting: RMI

Recursive Model Index (RMI): eCDF based model.

A RMI has L levels, and each level i has M_i models that recursively select the model in the next level of the RMI. The output $F(x)$ of an RMI is a value in $[0, 1)$ that is an approximation for $P(A \leq x)$.

$$f_i(x) = f_i^{(\lfloor M_i f_{i-1}(x) \rfloor)}(x)$$

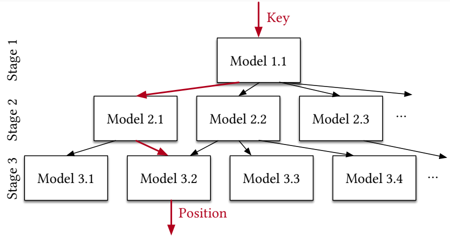


Figure 4: Generic RMI with multiple levels.

Machine Learning Models for Sorting: RMI

RMIs in practice are much simpler. They are limited to $L = 2$ levels, hence Learned Sort uses an RMI that is closer to:

$$F(x) = f_2^{(\lfloor M_2 f_1^{(1)}(x) \rfloor)}(x)$$

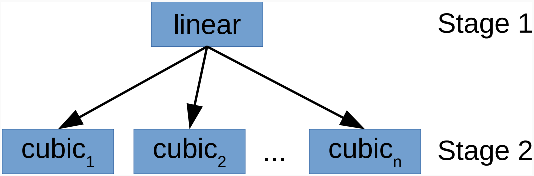


Figure 5: RMI used in practice with two levels.

Machine Learning Models for Sorting: Decison Trees

Decision Trees: partition based model.

Predicts the partition bucket $\mathcal{B}_i$ for each x_i with the property that if $x_i < x_j$, then $\mathcal{B}_i \leq \mathcal{B}_j$. Used by IPS^{4}o.

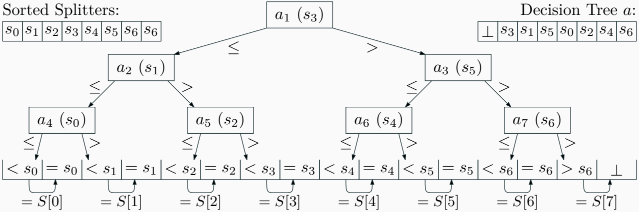


Figure 6: IPS^{4}o organizes the pivots in a decison tree structure for quickly finding the correct bucket for an element.