Raymond Zhu

Introduction

The primary goal of this project was to implement a dictionary codec to compress data with relatively low cardinality and accelerate search/scan operations. The dictionary encoding process involves creating a dictionary of unique data items and replacing each item in the dataset with its corresponding dictionary ID. To enhance performance, the implementation leveraged multi-threading for faster encoding and SIMD (Single Instruction, Multiple Data) instructions to optimize query performance.

Encoding

The encoding process involves scanning a raw column of data and replacing each data item with an encoded identifier (ID) that points to its corresponding entry in the dictionary. Implementation must support multi-threaded encoding to ensure that the process is both efficient and scalable. Specifically, the implementation should read raw data from a file, construct a dictionary of unique entries, and generate an encoded column file that stores both the dictionary and the encoded data. Utilizing multi-threading should speed up this process and make it feasible for larger datasets.

Querying

The querying process enables users to interact with an existing encoded column file. Specifically, the implementation should support checking whether a particular data item exists in the column and be able to return the indices of all occurrences of the data item. It should also be able to return all unique matching data and their corresponding indices when given a prefix. This process should be optimized using SIMD instructions for efficient querying.

Vanilla Column Search/Scan

To evaluate the efficiency of the dictionary codec, the code also needed to implement a vanilla column search/scan, which does not involve dictionary encoding. This served as the baseline for performance comparison.

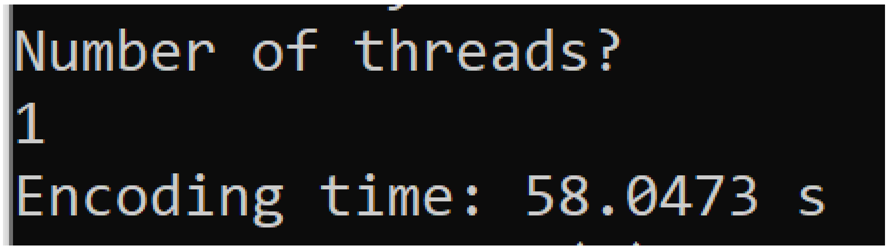
Design and Use of Multi-threading and SIMD

Multi-threading is utilized in the dictionary encoding process to split the workload and leverage multiple CPU cores for faster dictionary construction and encoding. Specifically, threads are used to independently scan different segments of the raw data and build partial dictionaries, which are then merged to create the final encoded column.

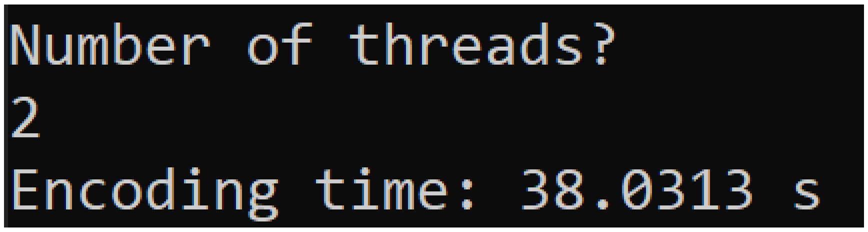
SIMD (Single Instruction, Multiple Data) instructions are applied to accelerate search/scan operations on encoded data. SIMD allows processing multiple data points in parallel, which enhances the efficiency of both exact queries and prefix searches.

Dictionary Encoding Results:

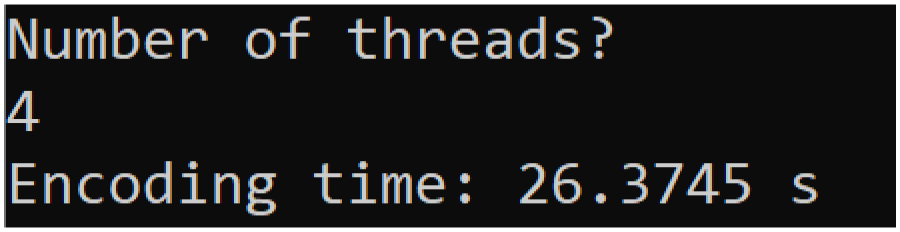
The following results represent the encoding time for different numbers of threads used during the dictionary encoding process. As can be seen from the data and the accompanying graph, there is a significant reduction in encoding time as the number of threads increases from 1 to 8, after which the performance gains begin to plateau.



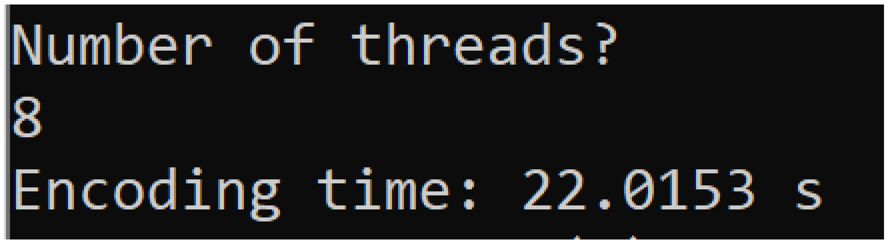
```
Number of threads?
1
Encoding time: 58.0473 s
```



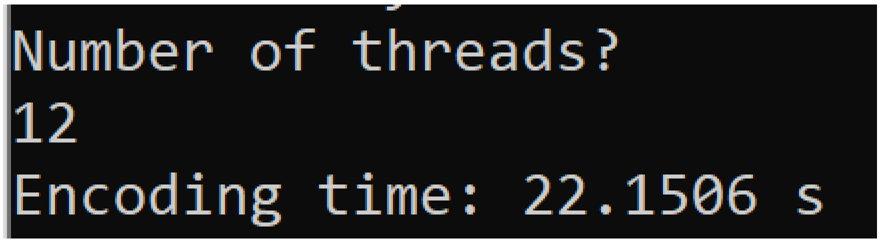
```
Number of threads?
2
Encoding time: 38.0313 s
```



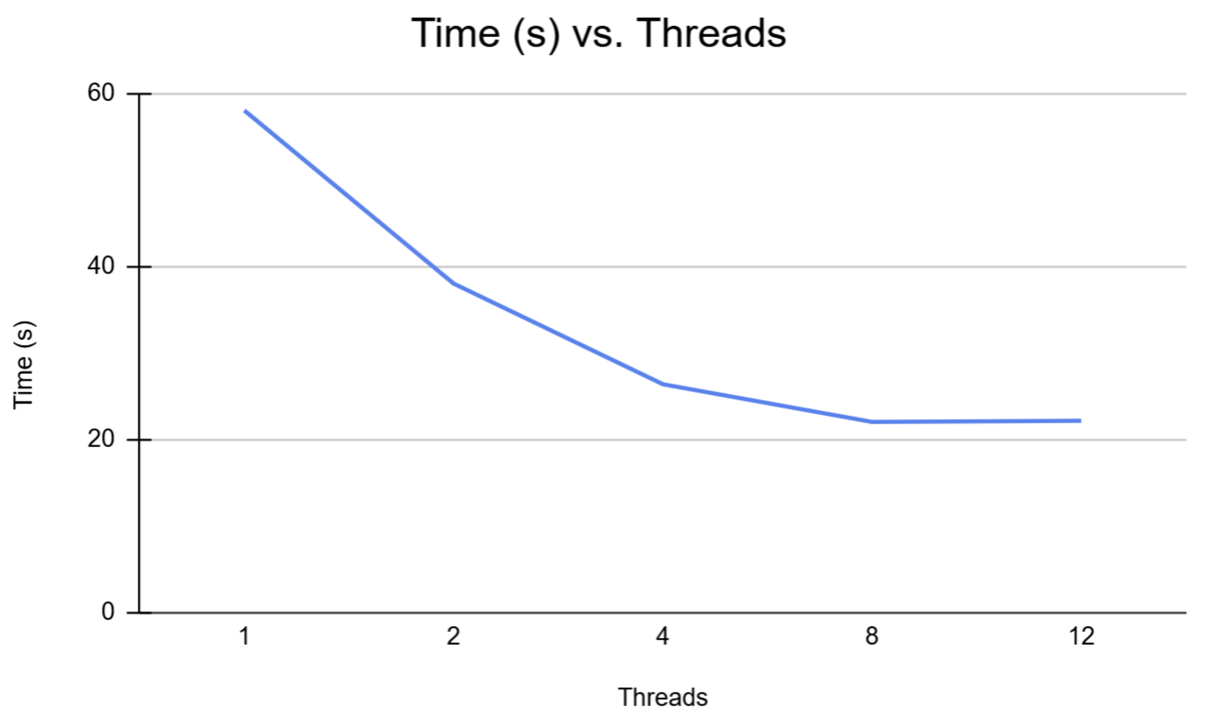
```
Number of threads?
4
Encoding time: 26.3745 s
```



```
Number of threads?
8
Encoding time: 22.0153 s
```

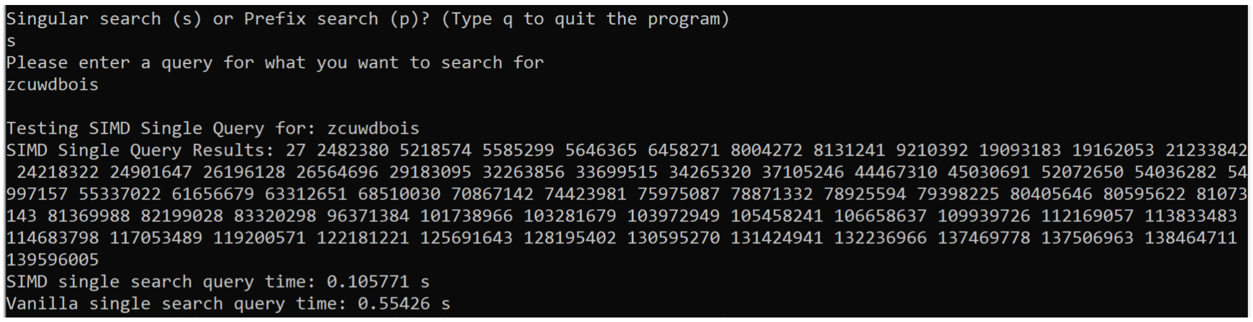


```
Number of threads?
12
Encoding time: 22.1506 s
```



Single Search Result Example:

Search for zcuwdbois



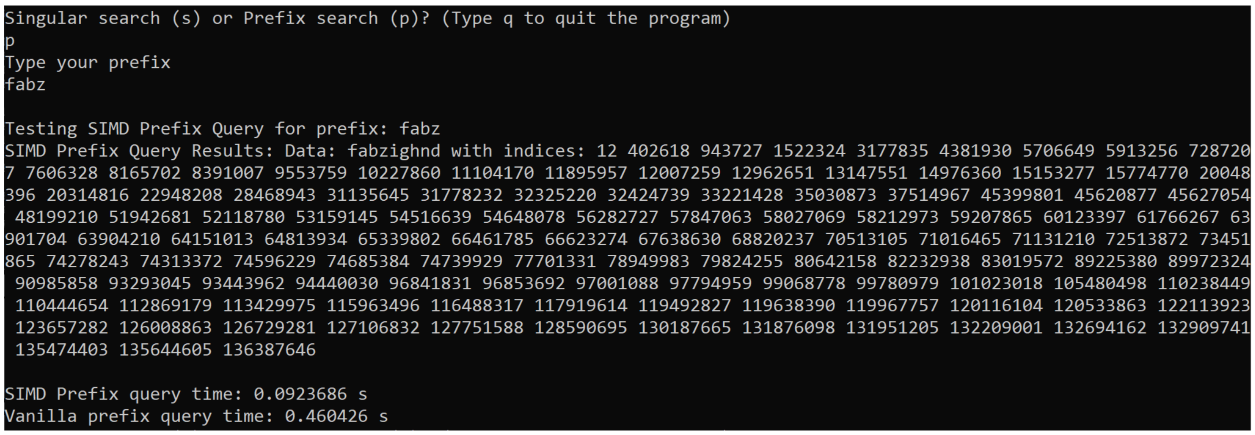
```
Singular search (s) or Prefix search (p)? (Type q to quit the program)
s
Please enter a query for what you want to search for
zcuwdbois

Testing SIMD Single Query for: zcuwdbois
SIMD Single Query Results: 27 2482380 5218574 5585299 5646365 6458271 8004272 8131241 9210392 19093183 19162053 21233842
 24218322 24901647 26196128 26564696 29183095 32263856 33699515 34265320 37105246 44467310 45030691 52072650 54036282 54
997157 55337022 61656679 63312651 68510030 70867142 74423981 75975087 78871332 78925594 79398225 80405646 80595622 81073
143 81369988 82199028 83320298 96371384 101738966 103281679 103972949 105458241 106658637 109939726 112169057 113833483
114683798 117053489 119200571 122181221 125691643 128195402 130595270 131424941 132236966 137469778 137506963 138464711
139596005
SIMD single search query time: 0.105771 s
Vanilla single search query time: 0.55426 s
```

The SIMD-enabled single search query for "zcuwdbois" took 0.105771 seconds, which is considerably faster than the vanilla implementation, which took 0.55426 seconds.

Prefix Search Result Example:

Search for fabz



```
Singular search (s) or Prefix search (p)? (Type q to quit the program)
p
Type your prefix
fabz

Testing SIMD Prefix Query for prefix: fabz
SIMD Prefix Query Results: Data: fabzighnd with indices: 12 402618 943727 1522324 3177835 4381930 5706649 5913256 728720
7 7606328 8165702 8391007 9553759 10227860 11104170 11895957 12007259 12962651 13147551 14976360 15153277 15774770 20048
396 20314816 22948208 28468943 31135645 31778232 32325220 32424739 33221428 35030873 37514967 45399801 45620877 45627054
 48199210 51942681 52118780 53159145 54516639 54648078 56282727 57847063 58027069 58212973 59207865 60123397 61766267 63
901704 63904210 64151013 64813934 65339802 66461785 66623274 67638630 68820237 70513105 71016465 71131210 72513872 73451
865 74278243 74313372 74596229 74685384 74739929 77701331 78949983 79824255 80642158 82232938 83019572 89225380 89972324
 90985858 93293045 93443962 94440030 96841831 96853692 97001088 97794959 99068778 99780979 101023018 105480498 110238449
 110444654 112869179 113429975 115963496 116488317 117919614 119492827 119638390 119967757 120116104 120533863 122113923
 123657282 126008863 126729281 127106832 127751588 128590695 130187665 131876098 131951205 132209001 132694162 132909741
 135474403 135644605 136387646

SIMD Prefix query time: 0.0923686 s
Vanilla prefix query time: 0.460426 s
```

The SIMD-enabled prefix search for "fabz" took 0.0933 seconds, compared to the vanilla prefix search which took 0.4604 seconds.

The performance improvements observed are indicative of the ability of SIMD instructions to perform multiple operations in parallel, thus significantly speeding up the scanning process. This is particularly beneficial for data-intensive queries, where each element in the data set can be processed simultaneously. Both single search and prefix search using SIMD outperform their vanilla counterparts, highlighting the benefits of SIMD for accelerating the querying operations in dictionary-encoded data. This makes SIMD a highly effective tool for reducing query time, especially in cases where real-time responsiveness is critical.

Conclusion

The dictionary codec implementation effectively demonstrated reduced encoding times with multi-threading, significant query performance improvements with SIMD, with gains evident in both single data item and prefix searches, and a clear advantage of dictionary encoding over vanilla scanning, particularly for large datasets. Future optimizations could include adaptive thread management for dynamic workload distribution or enhanced SIMD usage to support more complex query operations. This project underscores the efficacy of combining multi-threading and SIMD to optimize data encoding and querying, delivering meaningful improvements in both compression and access speed.