MILC: Inverted List Compression in Memory

Yorrick Müller
Garching, 3rd December 2018

Introduction – Inverted Lists

- Inverted list := Series of sorted integers, no duplicates allowed
- Can be used as index
- Used in search engines, graph analytics and more
- Can be used in sort merge joins
- Compression reduces memory footprint and disk I/O

Common Approach

- Only store the difference between an element and its predecessor
- Use fewer bits to encode the difference
  - No point access possible without decompressing the whole list
  - Not well suited for SIMD

MILC – General Idea

Divide the data into blocks

For each block:

1) Save first value of the block uncompressed (= skip pointer)
2) Store only the difference between the remaining values and the skip pointer

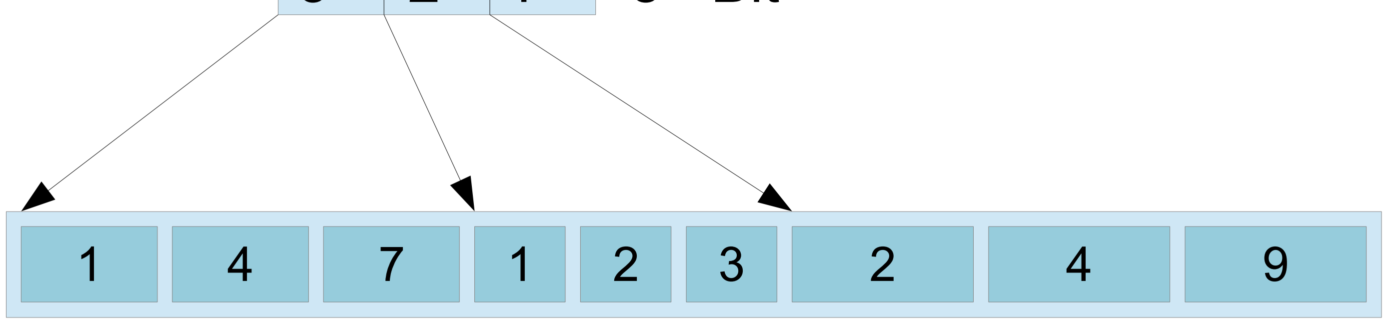
→ Allows point access without decompressing the whole sequence

MILC – General Idea

Uncompressed Values: 1, 2, 5, 8, 10, 11, 12, 13, 15, 17, 19, 24 (36 B)

Compressed: ~ 34 B

Skip Pointer	1	10	15	32 Bit
Address	0	9	15	32 Bit
#Elements	3	3	3	8 Bit
Bits used	3	2	4	8 Bit



MILC – First Optimization

Make the size of each block dynamic

Theorem:
A block with more than 160 elements can be stored more efficiently when split into smaller blocks

Approach:

- Use dynamic programming to solve an optimization problem
- Only block sizes from 1 to 160 need to be considered

=> Higher compression ratio

MILC – General Idea

Uncompressed Values: 1, 2, 5, 8, 10, 11, 12, 13, 15, 17, 19, 24 (36 B)

Compressed: ~ 25 B

Skip Pointer	1	10	32 Bit
Address	0	9	32 Bit
#Elements	3	7	8 Bit
Bits used	3	4	8 Bit

1	4	7	1	2	3	5	7	9	14

MILC – Second Optimization

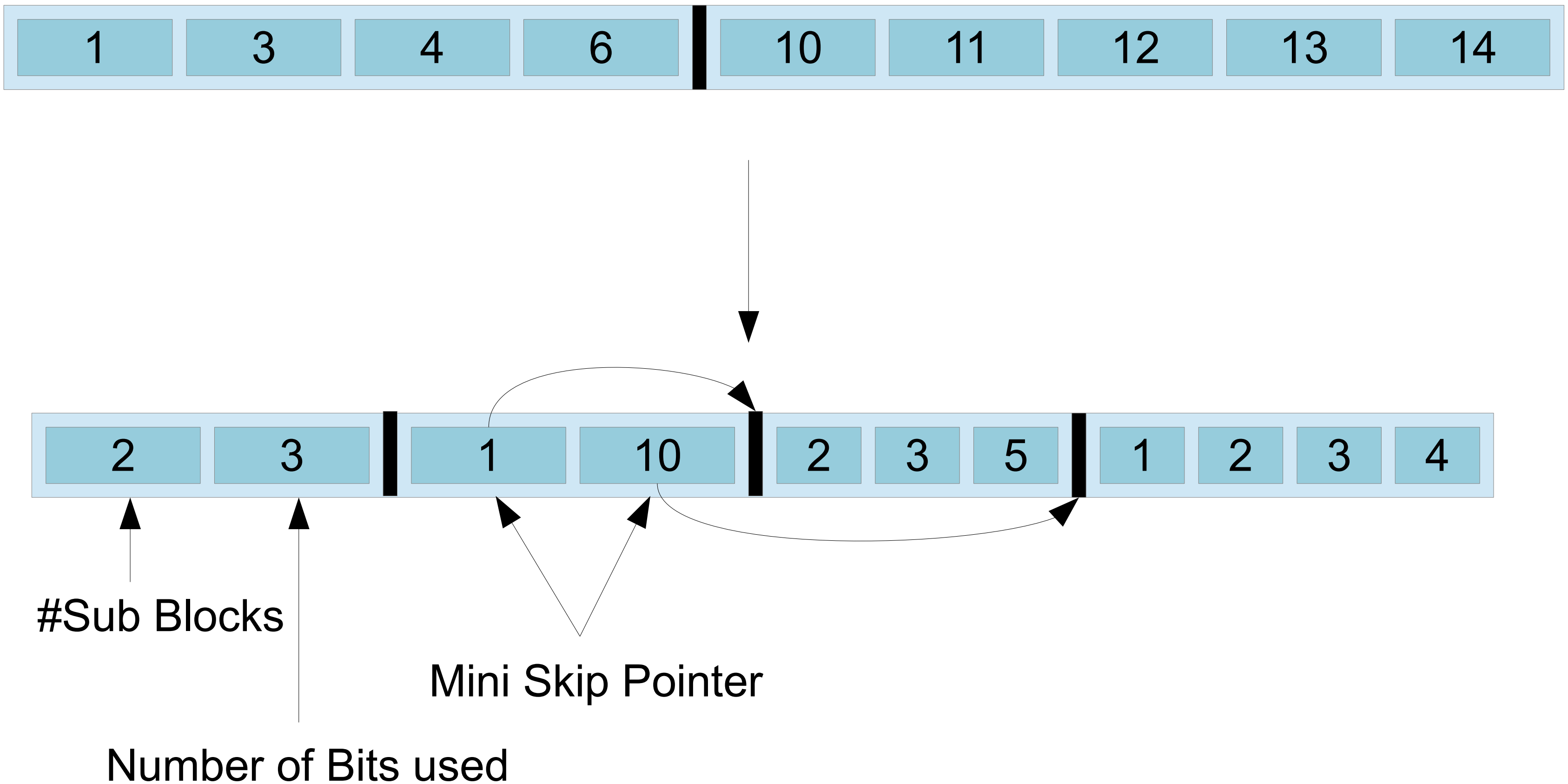
Subdivide blocks into sub blocks

Per block:

- Find optimal n
- Add a mini skip pointer every n elements
- Store only the difference of the remaining elements and their mini skip pointer

=> Higher compression ratio

MILC – Second Optimization



MILC – Third Optimization

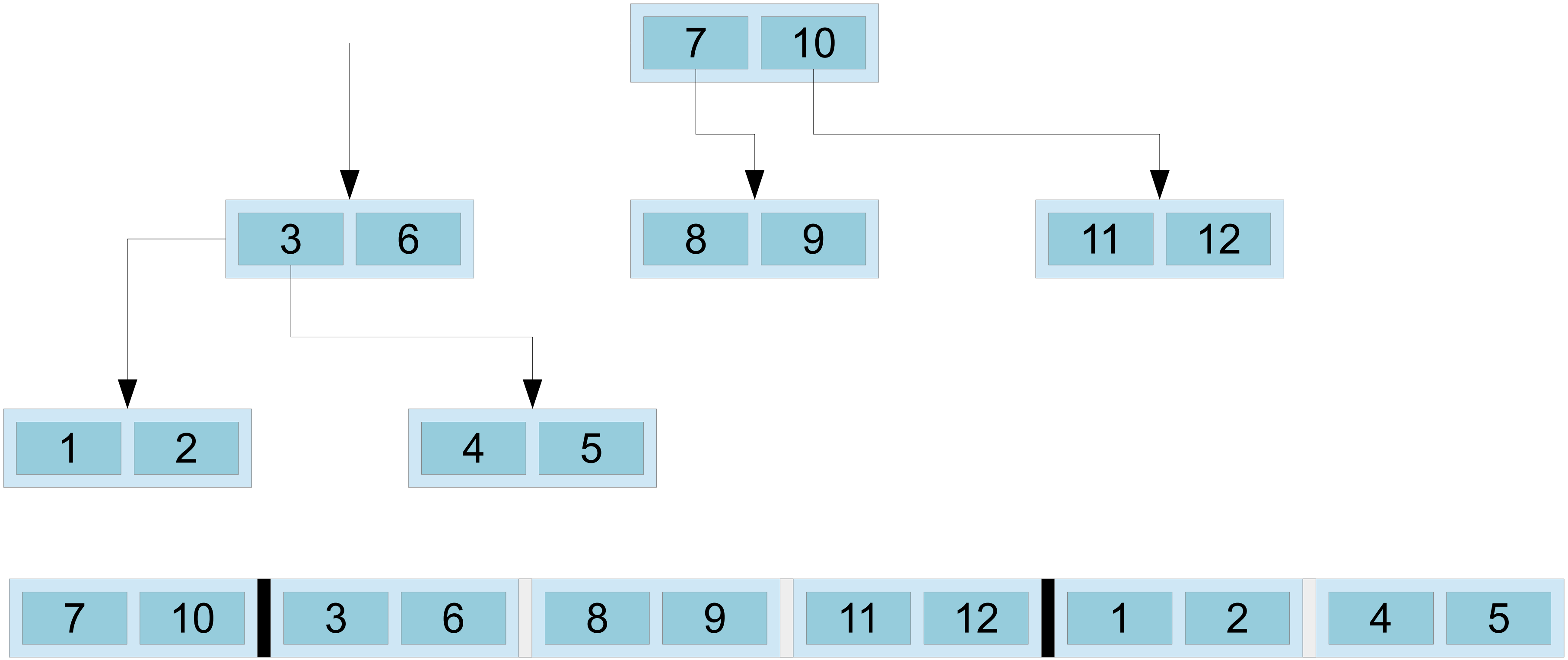
Arrange skip pointers in a linearized B-tree like fashion

Make each node 16 elements large s.t. each node is stored in exactly one cache line

→ Fewer cache lines accessed when searching for elements

MILC – Third Optimization

Reordering the values 1 through 12:



MILC – How do we Search in this Data Structure?

Search(x) :=

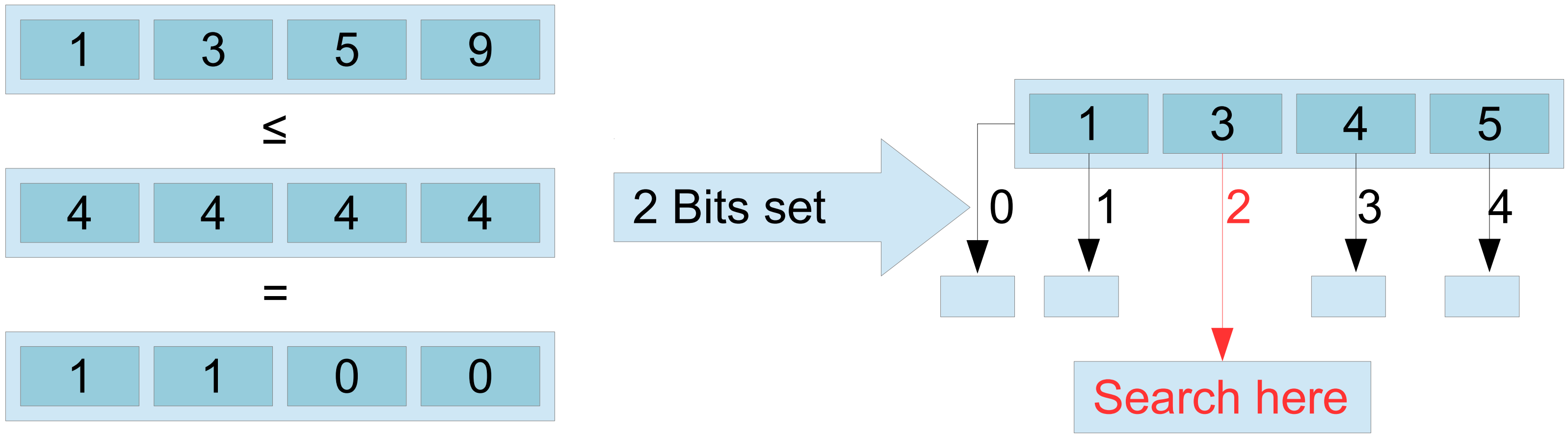
1) Search last element s <= x in the skip pointer tree
2) Find the last mini skip pointer q <= x - s in the block
3) Search for the value x - s - q in the corresponding sub block
4) The element is contained in the sequence iff it is found in any of
   those steps

MILC – Fourth Optimization

Search the tree nodes using SIMD (AVX2 in my case)

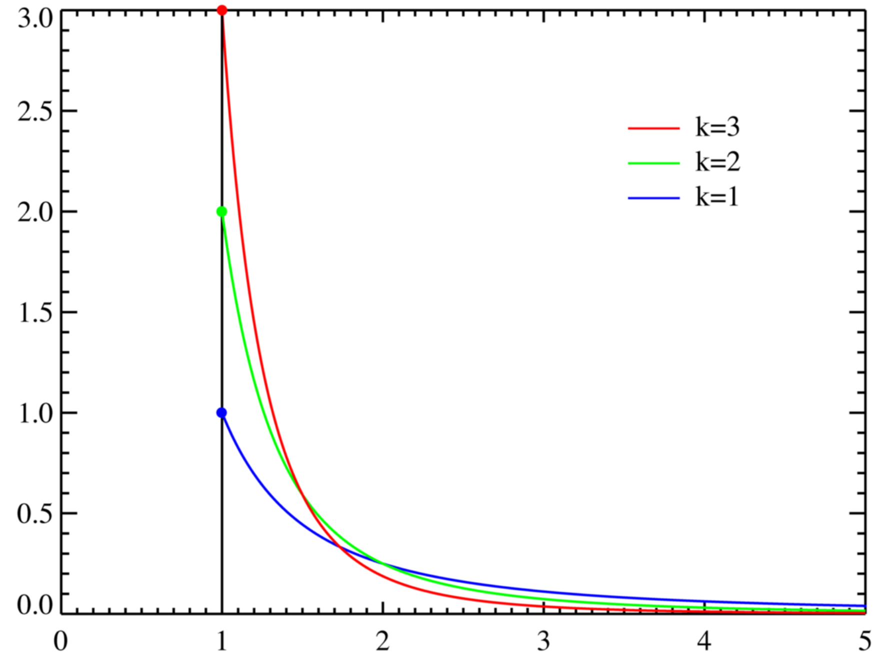
- Compare all keys in the node with the searched value
- Count the amount of set bits in the resulting bitmask

→ Number of bits set = index of the child node to descent to



Pareto Distribution

- Continuus counterpart to the Zipf distribution
- Low values more probable than high values
- P [X=x] $\in \Theta(1/x^2)$ for k=1, m=1 (given my discretization)
- $Pr[X < x] = 1 - (m/x)^k$ [2]



Src: https://upload.wikimedia.org/wikipedia/commons/d/d9/Pareto_distributionPDF.png

MILC – Test Setup

Inverted list generated by summing up the values generated by a discretized Pareto distribution

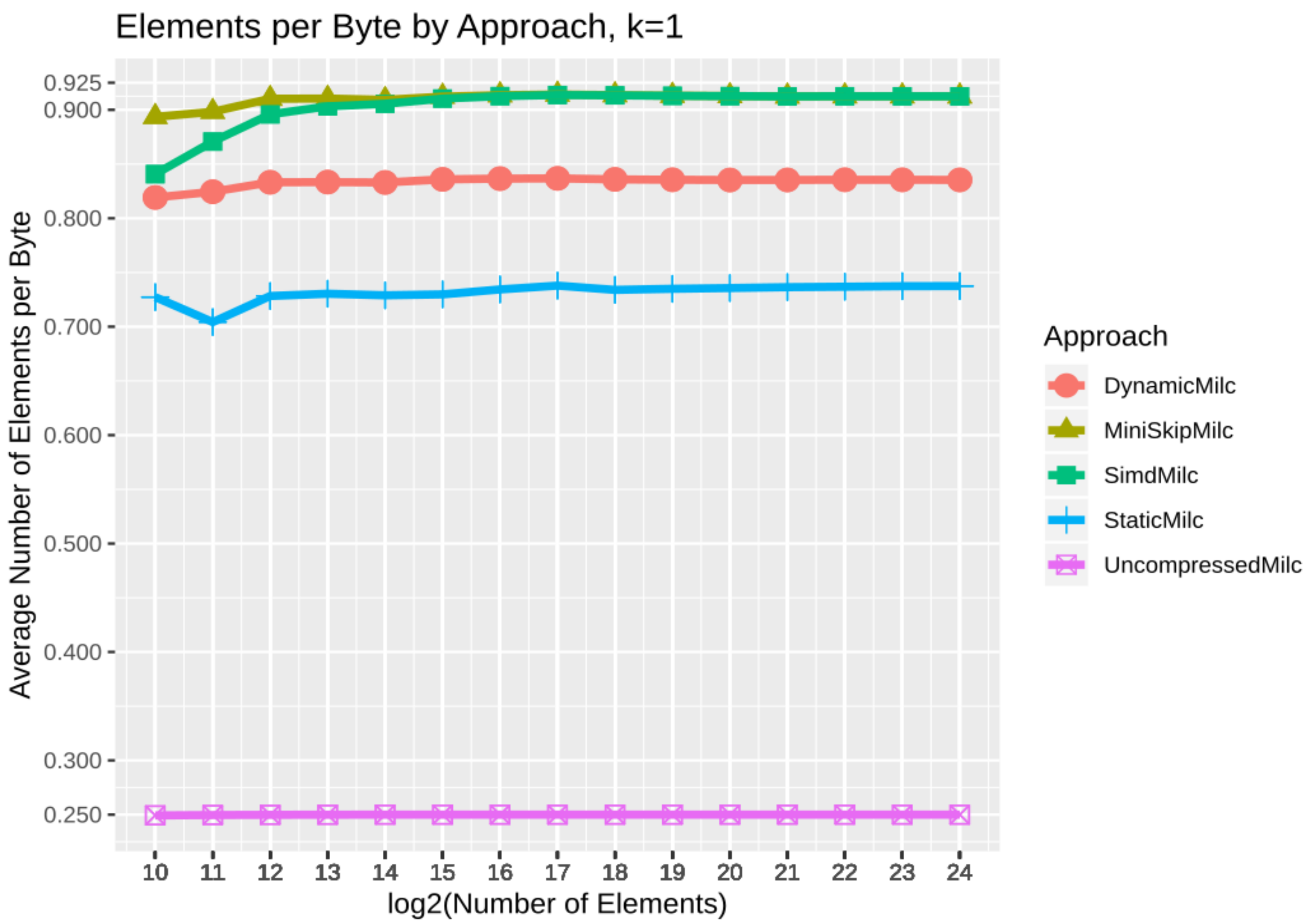
→ Most elements differ only slightly, some differ greatly

Generate a sequence of elements to lookup and then loop this sequence (→ don‘t measure the time it takes to generate random numbers)

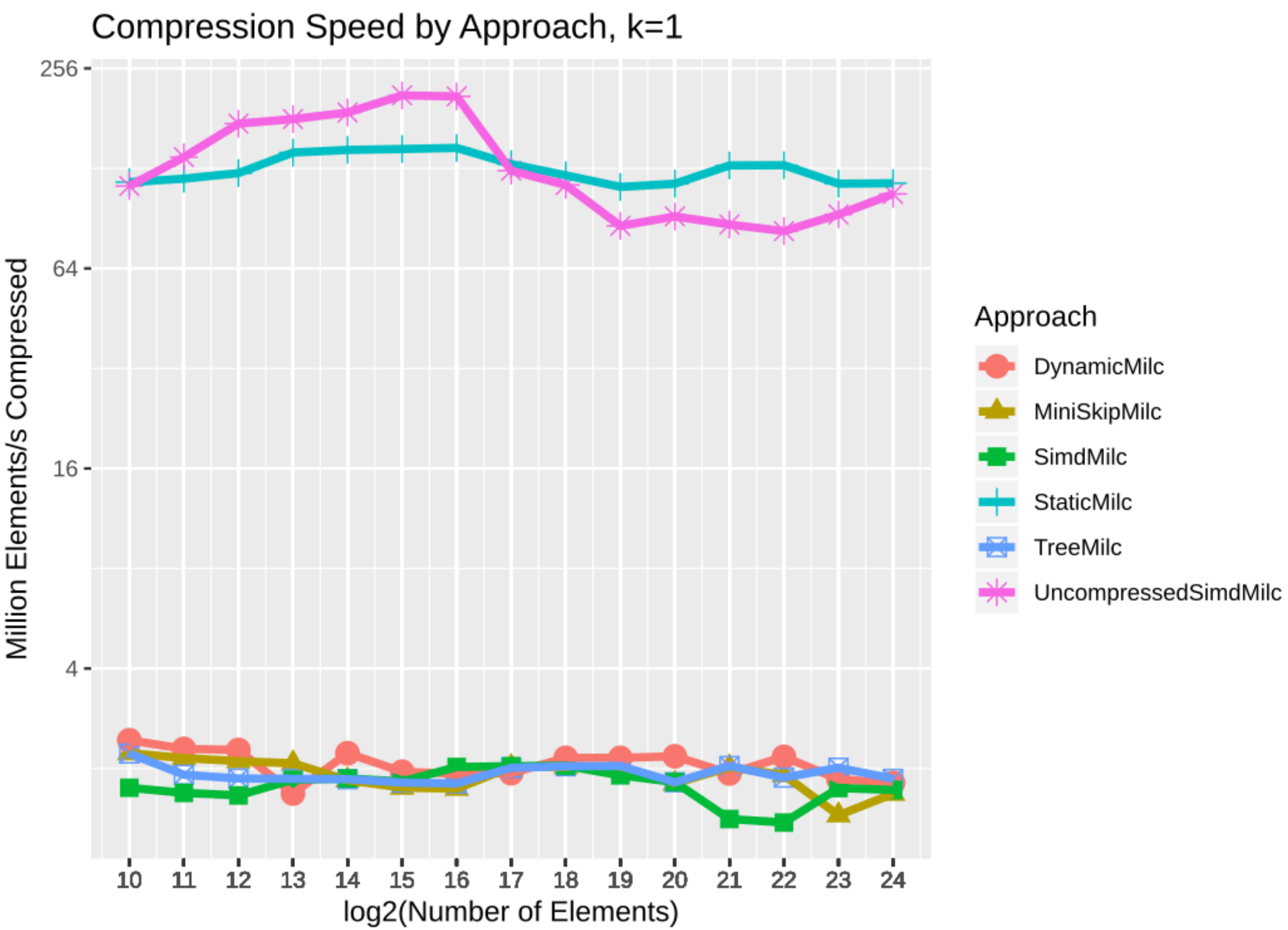
MILC – Approaches

UncompressedMilc	:= Uncompressed inverted list
StaticMilc	:= MILC with fixed block sizes
DynamicMilc	:= StaticMilc + dynamic block sizes
MiniSkipMilc	:= DynamicMilc + mini skip pointers
TreeMilc	:= MiniSkipMilc + reordered skip pointers
SimdMilc	:= TreeMilc + SIMD search
UncompressedSimdMilc	:= Uncompressed inverted list reordered and searched with SIMD

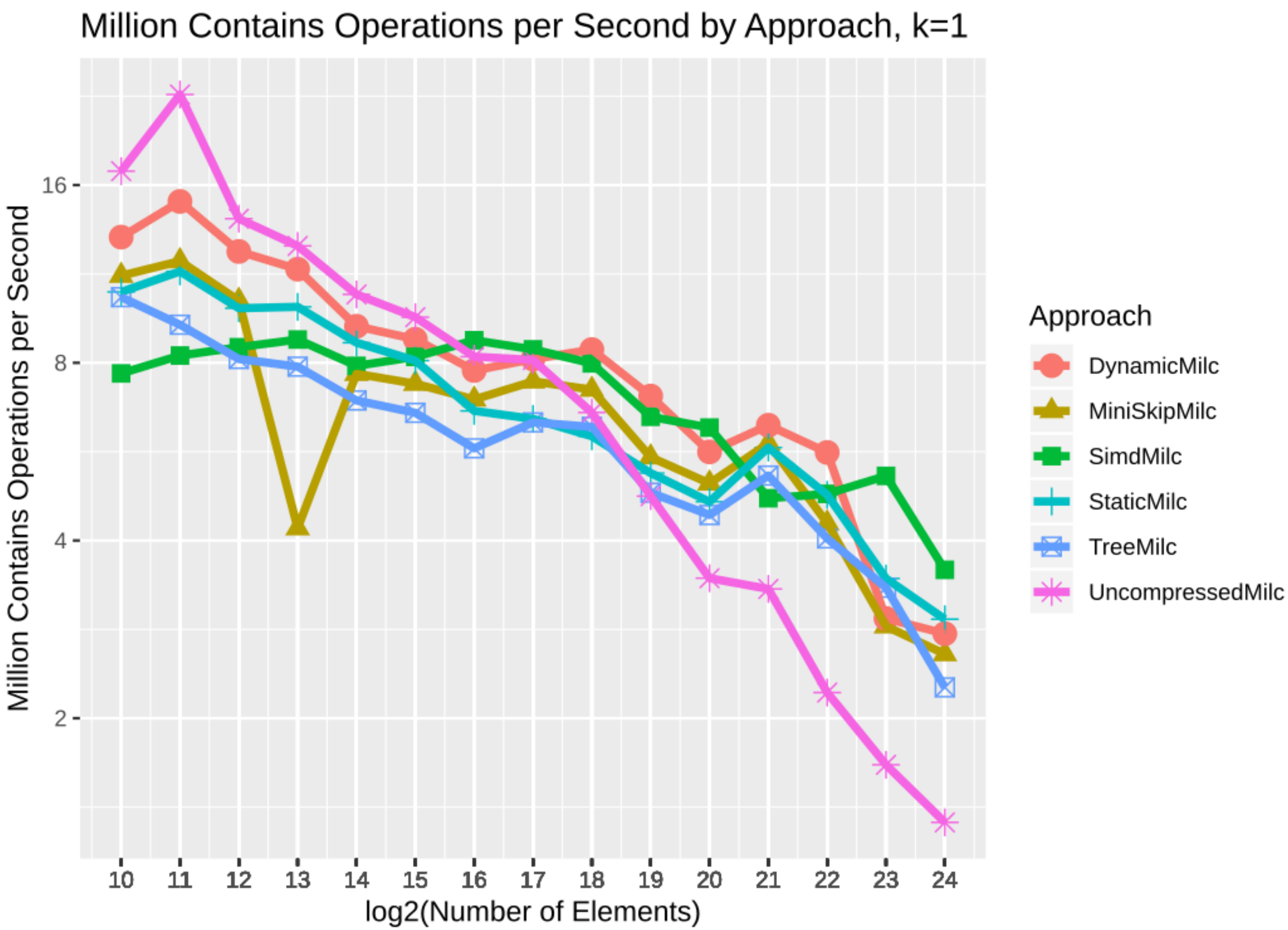
MILC – Compression Efficiency



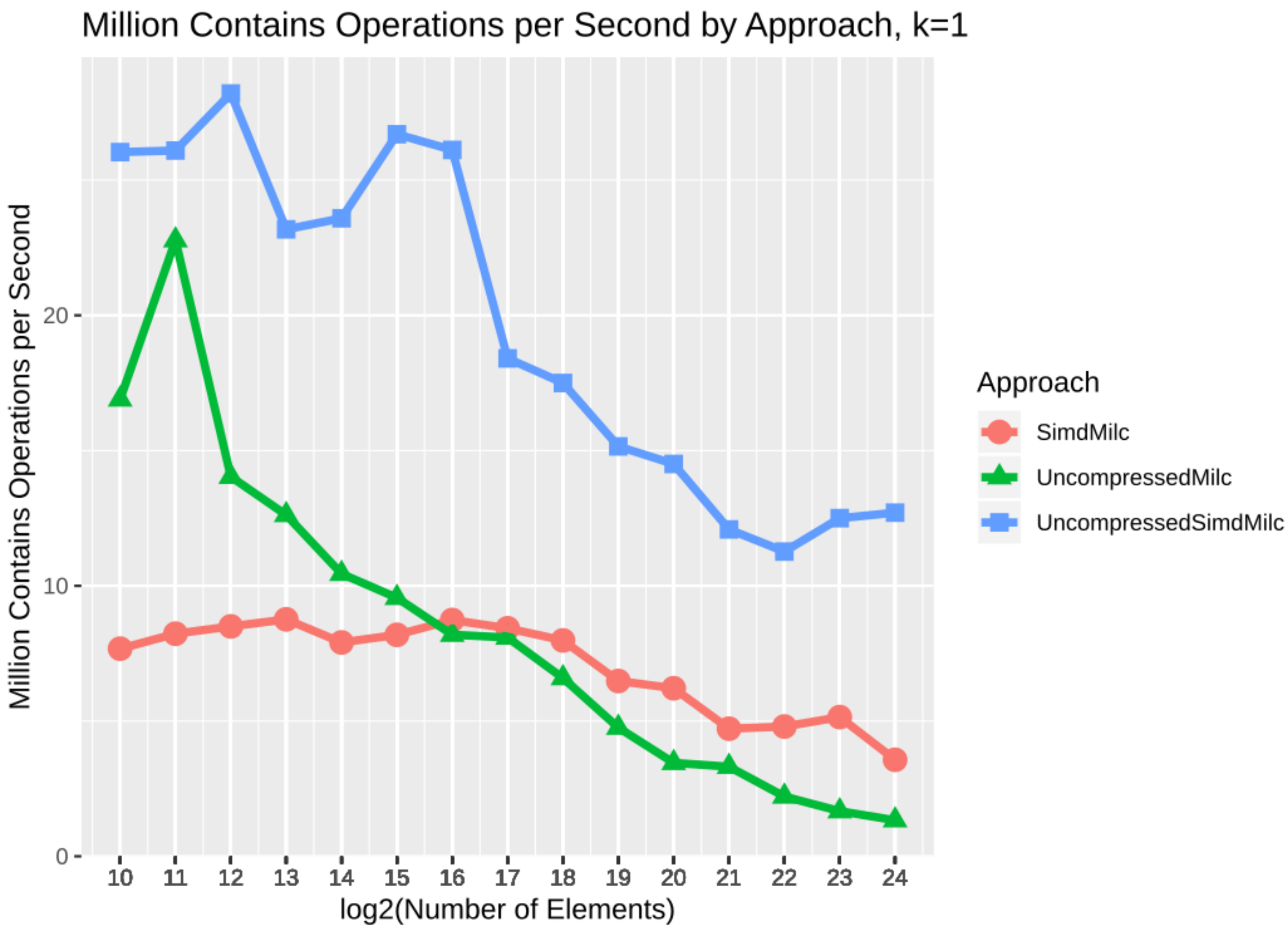
MILC – Compression Throughput



MILC – Contains Throughput



MILC – Perfomance Impact



Conclusion

- Gives good compression ratios on the test data (about 1:3.7 with k=1)
- Higher performance than an uncompressed sorted list on large datasets (given the tested distribution)
- SIMD-Tree optimization also very useful on uncompressed data

Thank you for your attention

Sources

[1] Wang, Jianguo, et al. "MILC: inverted list compression in memory." Proceedings of the VLDB Endowment 10.8 (2017): 853-864.

[2] Adamic, Lada A. "Zipf, power-laws, and pareto-a ranking tutorial." Information Dynamics Lab, HP Labs, Palo Alto, CA, http://ginger.hpl.hp.com/shl/papers/ranking/ranking.html (2000).