Practical Hardware Transactional vEB Trees

PPoPP'24

Mohammad Khalaji [1] Trevor Brown[1] Khuzaima Daudjee[1] Vitaly Aksenov[2]

1

2

Introduction

Sets and Maps

DS	Look-up	Insert	Delete	Predecessor	Successor
Hash Table	$O(1)$	$O(1)$	$O(1)$	$\boldsymbol{O(N)}$	$\boldsymbol{O(N)}$
BST	$O(\lg N)$	$O(\lg N)$	$O(\lg N)$	$\boldsymbol{O(\lg N)}$	$\boldsymbol{O(\lg N)}$
B-Tree	$O(\log_B N)$	$O(\log_B N)$	$O(\log_B N)$	$\boldsymbol{O(\log_B N)}$	$\boldsymbol{O(\log_B N)}$

DS	Look-up	Insert	Delete	Predecessor	Successor
Hash Table	$O(1)$	$O(1)$	$O(1)$	$\boldsymbol{O(N)}$	$\boldsymbol{O(N)}$
BST	$O(\lg N)$	$O(\lg N)$	$O(\lg N)$	$\boldsymbol{O(\lg N)}$	$\boldsymbol{O(\lg N)}$
B-Tree	$O(\log_B N)$	$O(\log_B N)$	$O(\log_B N)$	$\boldsymbol{O(\log_B N)}$	$\boldsymbol{O(\log_B N)}$
vEB Tree	$O(\lg \lg u)$	$O(\lg \lg u)$	$O(\lg \lg u)$	$\boldsymbol{O(\lg \lg u)}$	$\boldsymbol{O(\lg \lg u)}$

Sets and Maps

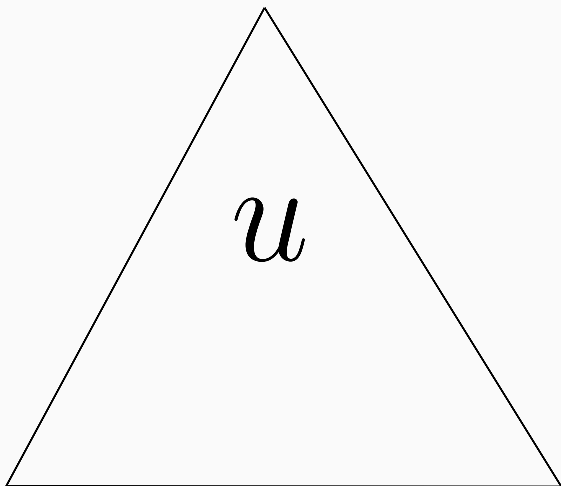
DS	Look-up	Insert	Delete	Predecessor	Successor
Hash Table	$O(1)$	$O(1)$	$O(1)$	$\boldsymbol{O(N)}$	$\boldsymbol{O(N)}$
BST	$O(\lg N)$	$O(\lg N)$	$O(\lg N)$	$\boldsymbol{O(\lg N)}$	$\boldsymbol{O(\lg N)}$
B-Tree	$O(\log_B N)$	$O(\log_B N)$	$O(\log_B N)$	$\boldsymbol{O(\log_B N)}$	$\boldsymbol{O(\log_B N)}$
vEB Tree	$O(\lg \lg u)$	$O(\lg \lg u)$	$O(\lg \lg u)$	$\boldsymbol{O(\lg \lg u)}$	$\boldsymbol{O(\lg \lg u)}$

u	$\lg \lg u$
65K	4
1M	4.3
1B	4.9
1T	5.3

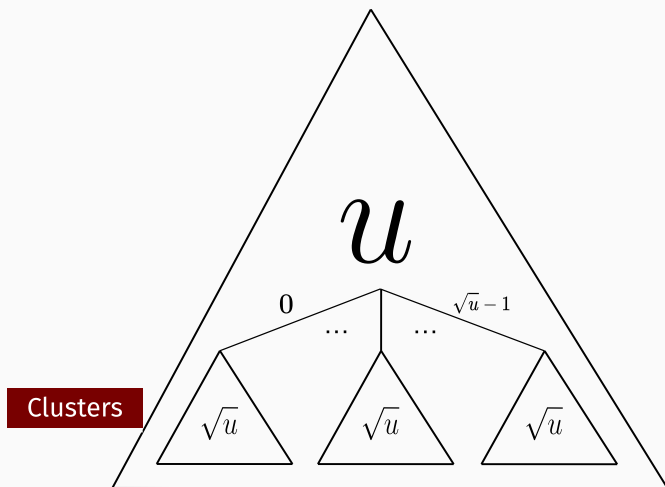
vEB Trees

- Introduced in 1977 by Peter van Emde Boas [4]
- Fixed universe size $u = 2^{\lg u}$: $\{0, \ldots, u-1\}$ $\rightarrow \lg u$ bit keys
- $O(\lg \lg u)$ complexity for all operations

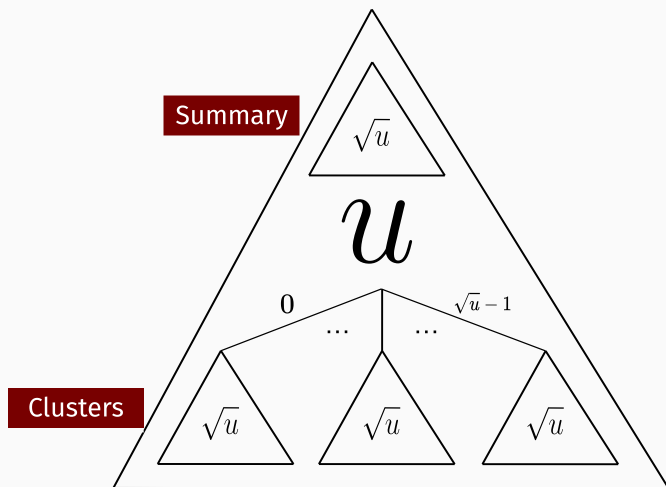
vEB Tree Structure



vEB Tree Structure

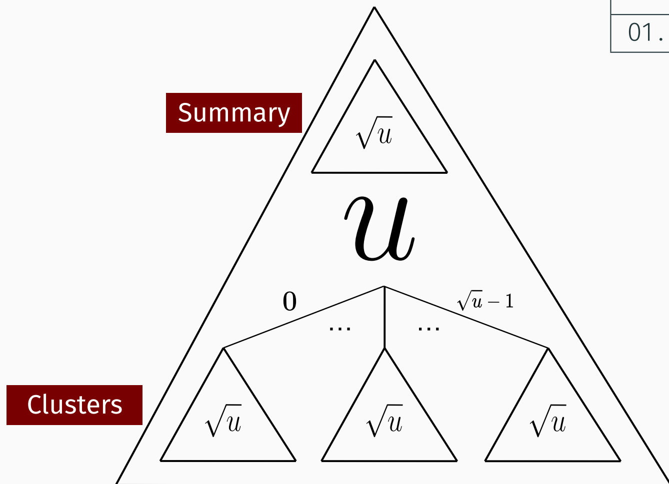


vEB Tree Structure

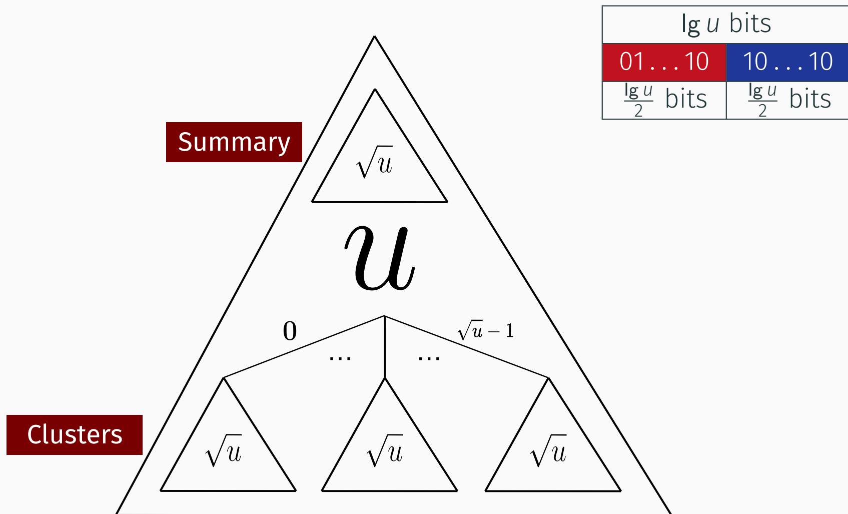


vEB Insert Example

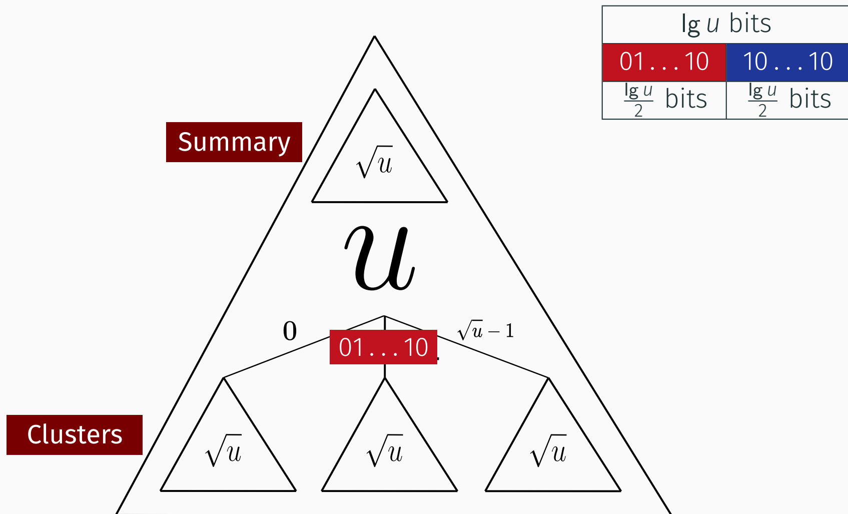
$\lg u$ bits
01 10



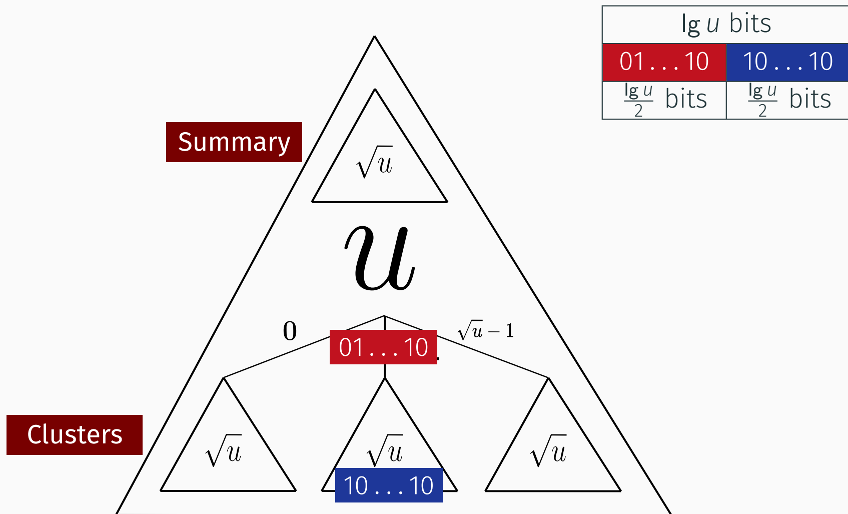
vEB Insert Example



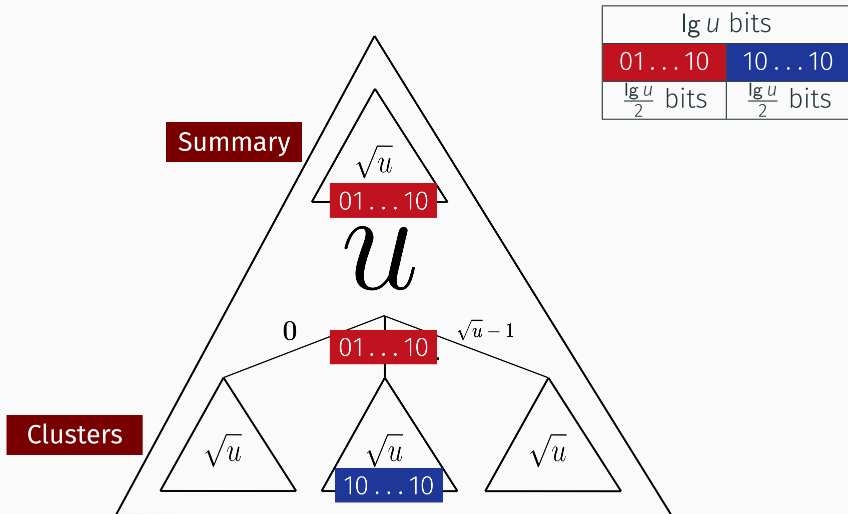
vEB Insert Example



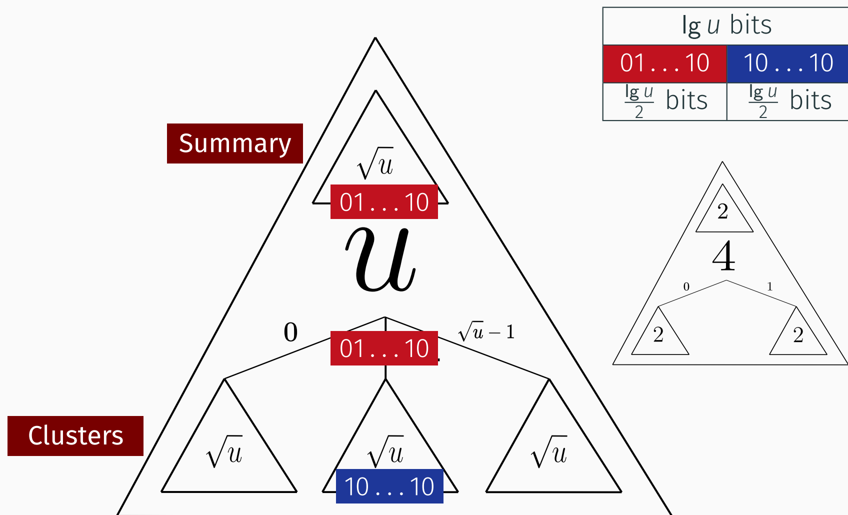
vEB Insert Example



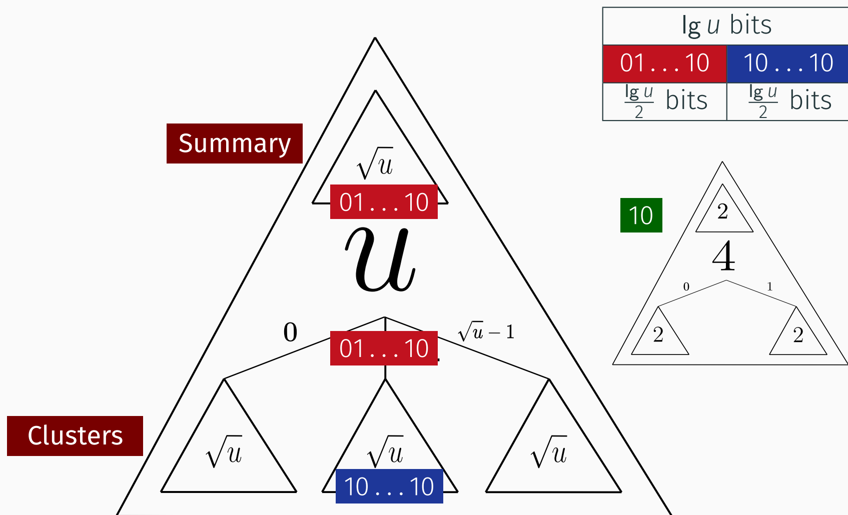
vEB Insert Example



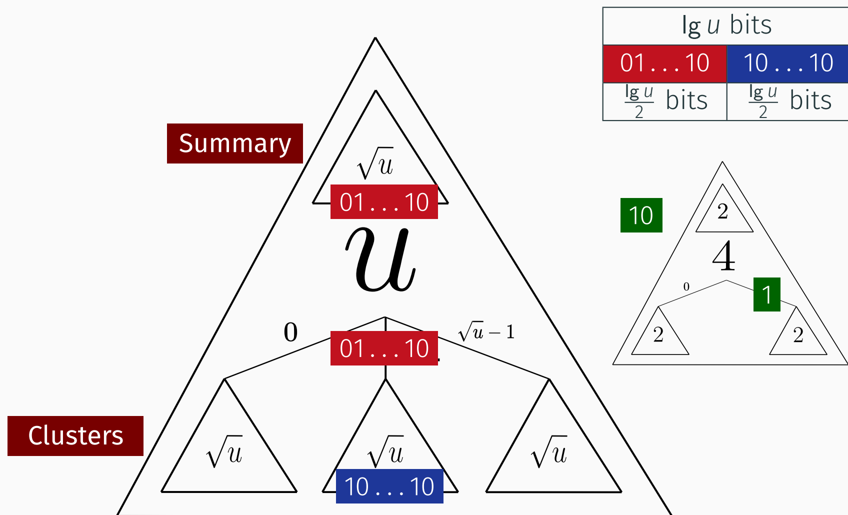
vEB Insert Example



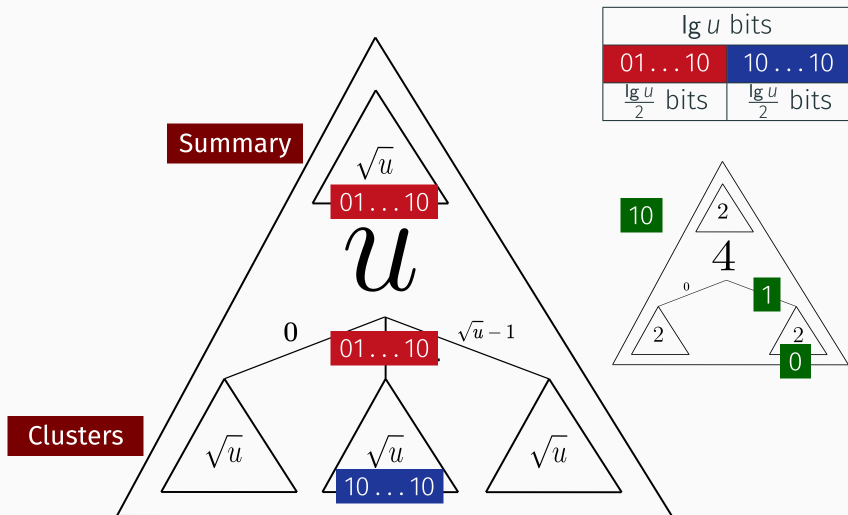
vEB Insert Example



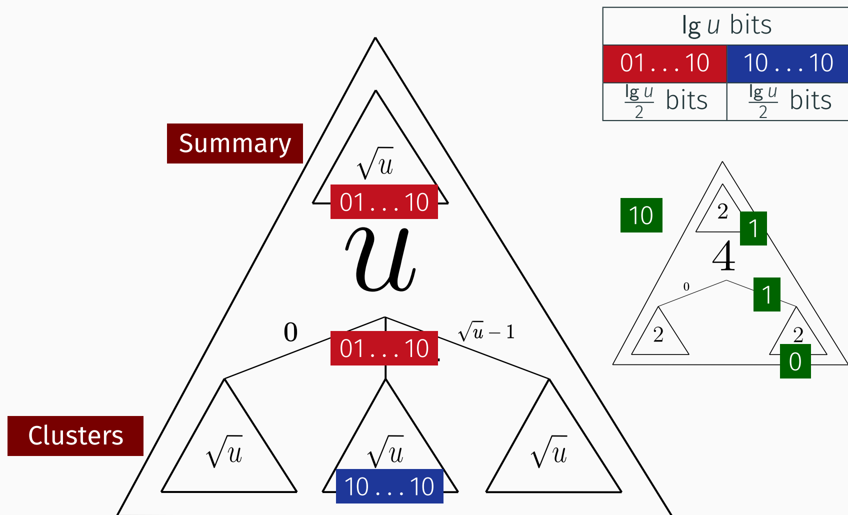
vEB Insert Example



vEB Insert Example



vEB Insert Example



Hardware Transactional Memory (HTM)

- Atomic blocks of code
    - Take effect as a single indivisible unit (Commit)
    - Or rolled back with no effect on shared memory (Abort)
- Hardware level →Efficient

Hardware Transactional Memory (HTM)

- Atomic blocks of code
    - Take effect as a single indivisible unit (Commit)
    - Or rolled back with no effect on shared memory (Abort)
- Hardware level → Efficient

```
int status = _xbegin();
```

Hardware Transactional Memory (HTM)

- Atomic blocks of code
  - Take effect as a single indivisible unit (Commit)
  - Or rolled back with no effect on shared memory (Abort)
- Hardware level → Efficient

```
int status = _xbegin();
if (status == _XBEGIN_STARTED) {
    Transactional code...
    _xend();
}
```

Hardware Transactional Memory (HTM)

- Atomic blocks of code
  - Take effect as a single indivisible unit (Commit)
  - Or rolled back with no effect on shared memory (Abort)
- Hardware level → Efficient

```
int status = _xbegin();
if (status == _XBEGIN_STARTED) {
    Transactional code...
    _xend();
}
else {
    Fallback code...
}
```

UNIVERSITY OF WATERLOO

Transactional Lock Elision (TLE)

Transactional Lock Elision (TLE)

```
if (_xbegin() == _XBEGIN_STARTED) {

    Transaction body...

    _xend();
}
```

Fast Path

Transactional Lock Elision (TLE)

Fast Path

Fallback Path

```
if (_xbegin() == _XBEGIN_STARTED) {

    Transaction body...

    _xend();
}
else {
    lock();
    Transaction body...
    unlock();
}
```

UNIVERSITY OF WATERLOO

Transactional Lock Elision (TLE)

Fast Path

Fallback Path

```
if (_xbegin() == _XBEGIN_STARTED) {
    if lock is already held
        abort();

    Transaction body...

    _xend();
}
else {
    lock();
    Transaction body...
    unlock();
}
```

Transactional Lock Elision (TLE)

Fast Path

Fallback Path

```
if (_xbegin() == _XBEGIN_STARTED) {
    if lock is already held
        abort();

    Transaction body... ← T2

    _xend();
}
else {
    lock(); ← T1
    Transaction body...
    unlock();
}
```

Transactional Lock Elision (TLE)

Fast Path

Fallback Path

```
if (_xbegin() == _XBEGIN_STARTED) {
    if lock is already held
        abort();

    Transaction body... ← T2

    _xend();
}
else {
    lock();
    Transaction body... ← T1
    unlock();
}
```

Transactional Lock Elision (TLE)

Fast Path

Fallback Path

```
if (_xbegin() == _XBEGIN_STARTED) {
    if lock is already held
        abort();

    Transaction body...

    _xend(); ← T2
}
else {
    lock();
    Transaction body... ← T1
    unlock();
}
```

Transactional Lock Elision (TLE)

Fast Path

Fallback Path

```
if (_xbegin() == _XBEGIN_STARTED) {
    if lock is already held
        abort();

    Transaction body...

    _xend();
}
else { ← T2
    lock();
    Transaction body... ← T1
    unlock();
}
```

UNIVERSITY OF WATERLOO

Methodology

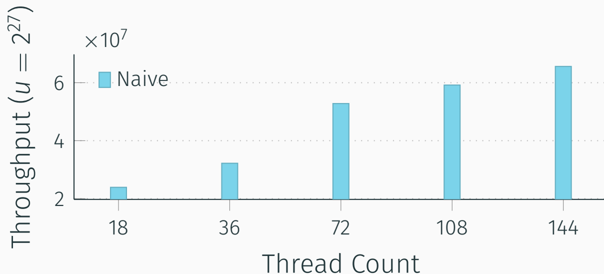
Overview

- Concurrent vEB trees with recursive summaries
- Full support for successor and predecessor queries
- Starting with a naive vEB set implementation
- Use simple building blocks to end up with a performant vEB map

First Step: Naive

Idea

Wrap sequential vEB operations in TLE blocks

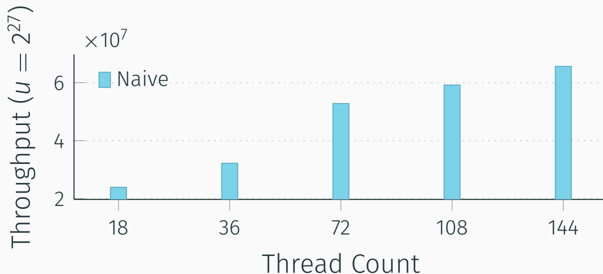


UNIVERSITY OF WATERLOO

First Step: Naive

Idea

Wrap sequential vEB operations in TLE blocks



Problem

Memory usage does not depend on the number of keys!

Improve Memory Footprint: Dynamic

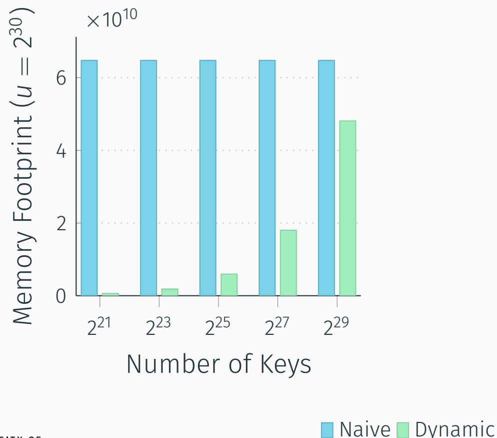
Can we save memory?

In the Naive variation, the entire tree is pre-allocated

Improve Memory Footprint: Dynamic

Can we save memory?

In the Naive variation, the entire tree is pre-allocated

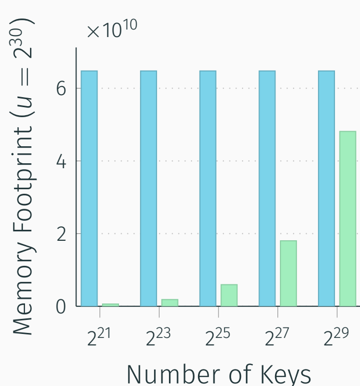


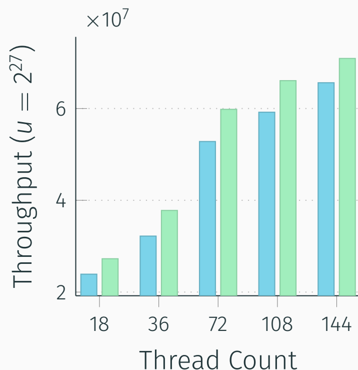
UNIVERSITY OF WATERLOO

Improve Memory Footprint: Dynamic

Can we save memory?

In the Naive variation, the entire tree is pre-allocated





Naive Dynamic

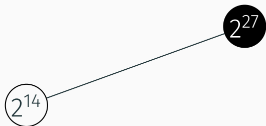
UNIVERSITY OF WATERLOO

Room For Improvement: Cut-Off

u Data

2^{27}

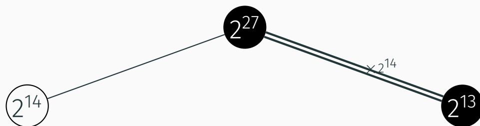
Room For Improvement: Cut-Off



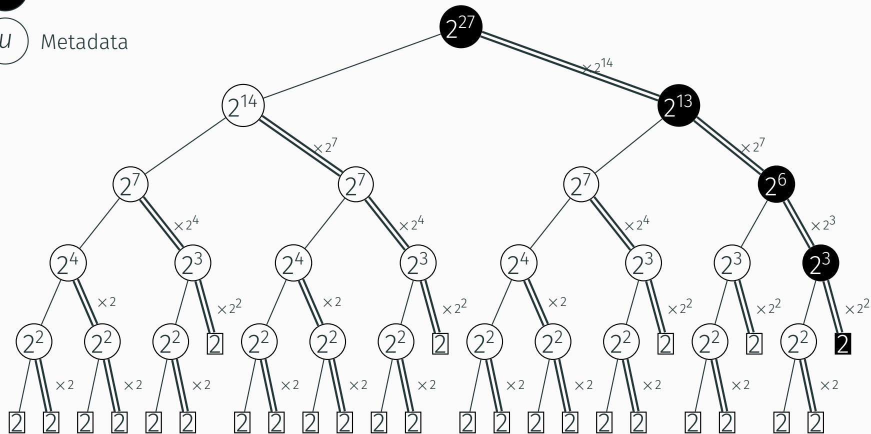
Room For Improvement: Cut-Off

u Data

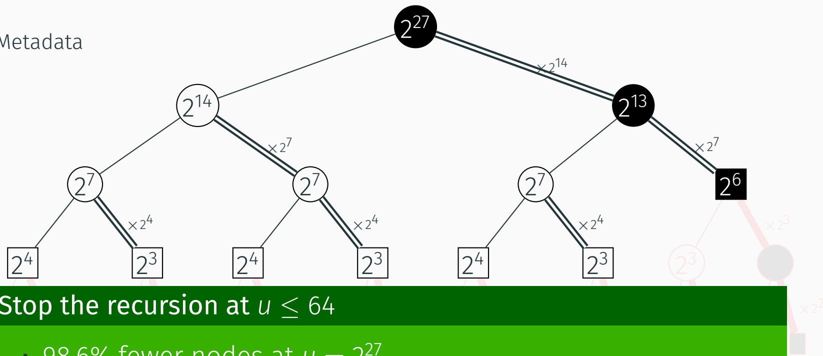
u Metadata



Room For Improvement: Cut-Off



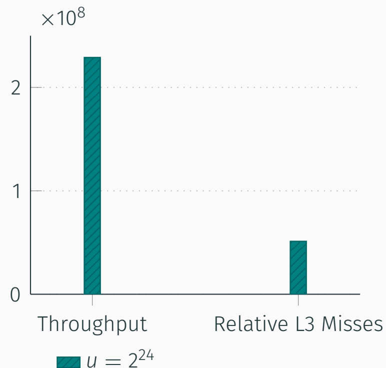
Room For Improvement: Cut-Off

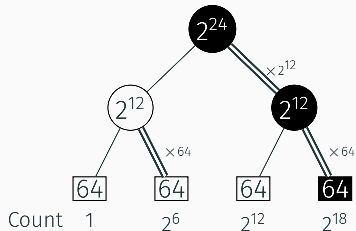


Stop the recursion at $u \leq 64$

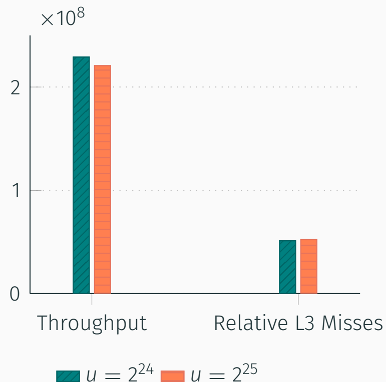
- 98.6% fewer nodes at $u = 2^{27}$
- One L3 cache miss vs. 7
- $3\times$ faster
- 75% less memory

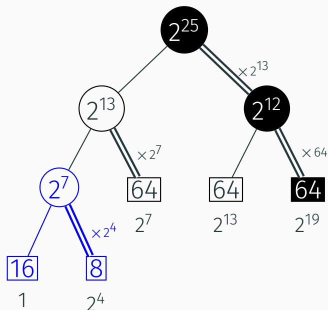
Problem with Cut-Off



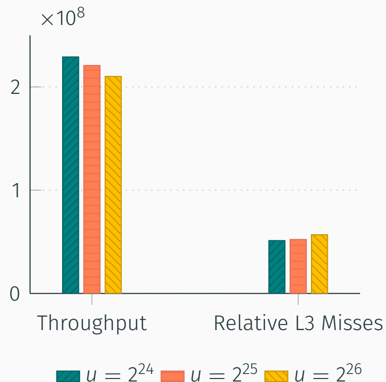


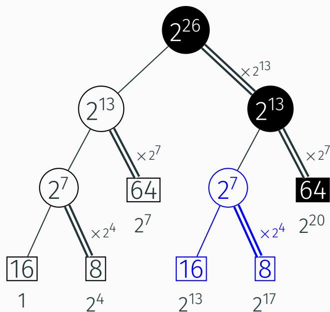
Problem with Cut-Off

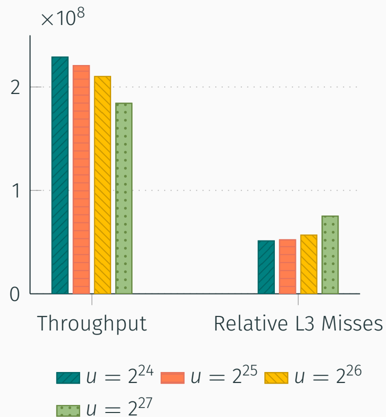


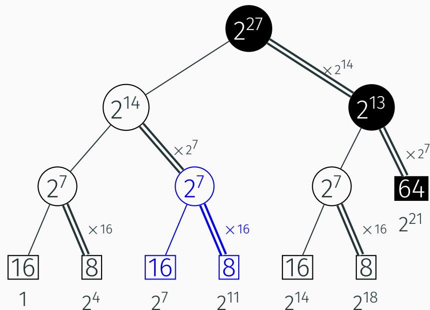


Problem with Cut-Off

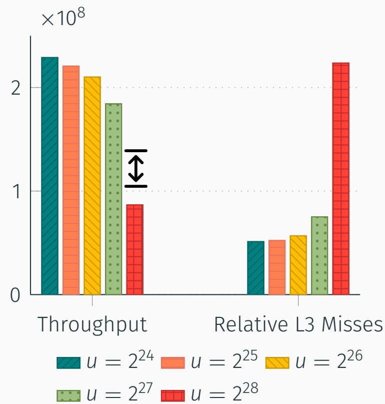




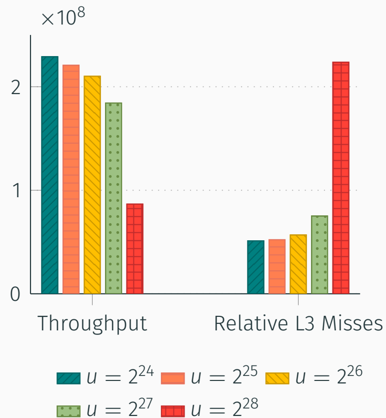

×10^8
2
1
0
Throughput
Relative L3 Misses
u = 2^24
u = 2^25
u = 2^26
u = 2^27

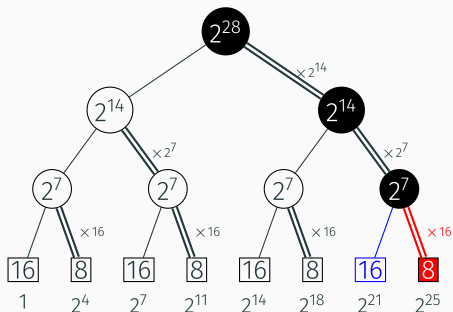

2^27
×2^14
2^14
2^13
×2^7
×2^7
2^7
2^7
2^7
64
2^21
×16
×16
×16
16
8
16
8
16
8
1
2^4
2^7
2^11
2^14
2^18

Problem with Cut-Off



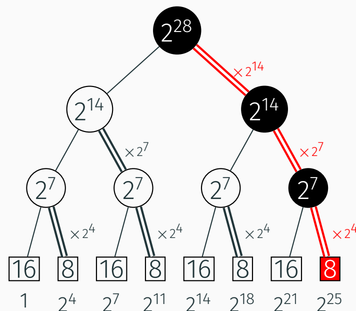
Problem with Cut-Off





Solution: New-Root

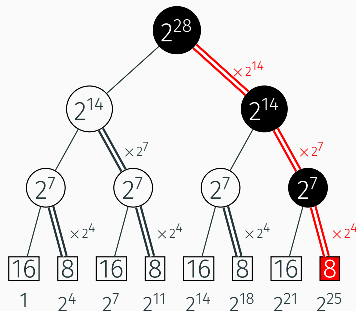
Before

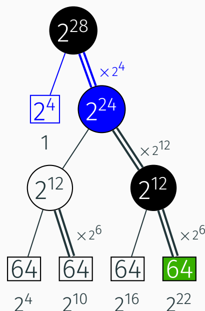


Universe Size	2^{24}	2^{25}	2^{26}	2^{27}	2^{28}	2^{29}
Data Leaf Count	2^{18}	2^{19}	2^{20}	2^{21}	2^{25}	2^{26}
Data Leaf Size	64	64	64	64	8	8

Solution: New-Root

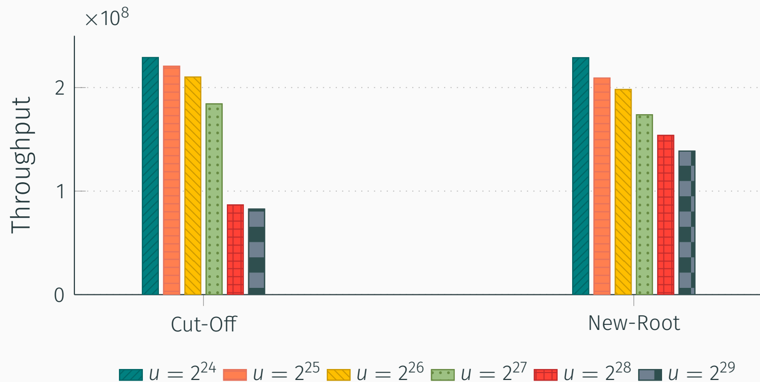
Before	After



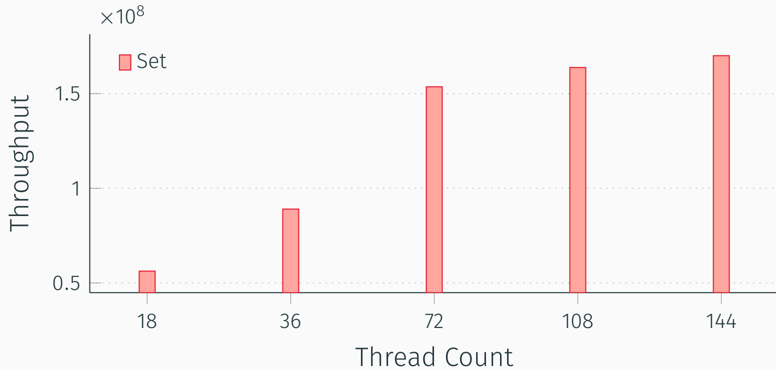


Universe Size	2^{24}	2^{25}	2^{26}	2^{27}	2^{28}	2^{29}
Data Leaf Count	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}
Data Leaf Size	64	64	64	64	64	64

Solution: New-Root



Moving On to Maps



Moving On to Maps

$\times 10^8$

Set
Map

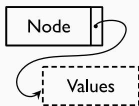
Throughput

1.5
1
0.5

18 36 72 108 144

Thread Count

Map



Moving On to Maps

$\times 10^8$

Set
Map
Embedded Map

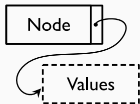
Throughput

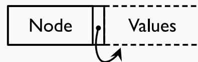
1.5
1
0.5

18
36
72
108
144

Thread Count

Map	Embedded Map





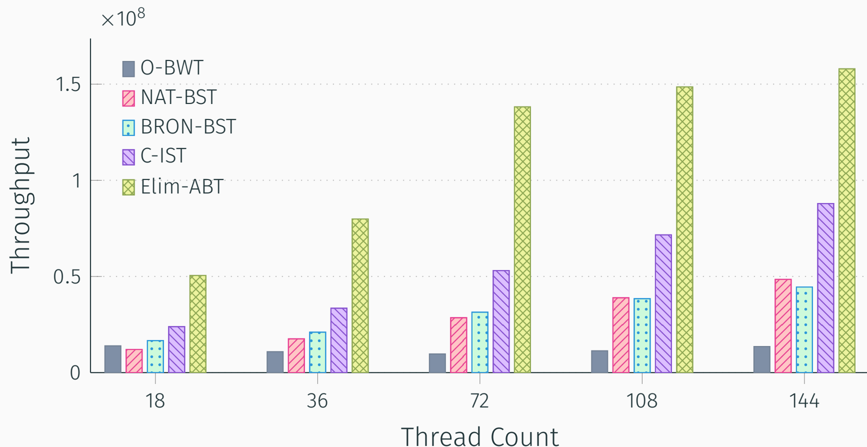
UNIVERSITY OF WATERLOO

Experimental Evaluation

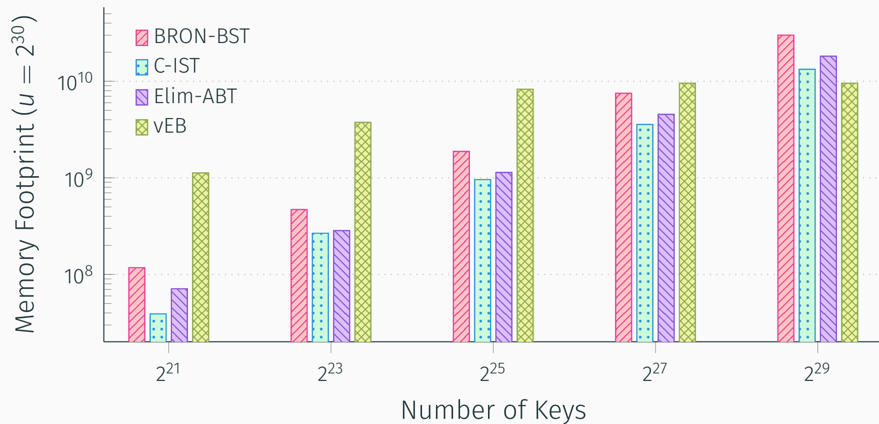
Experimental Setup

- Test harness: SetBench [2]
- 4 sockets × 18 cores per socket = 72 cores, 144 hardware threads
- Comparison points:
  - **Elim-ABT:** A concurrent (a,b)-tree [3]
  - **C-IST:** A doubly logarithmic interpolation search tree [2]
  - **BRON-BST:** An OCC BST [1]
  - **O-BWT:** The open-source implementation of BW-Tree [5]

Performance

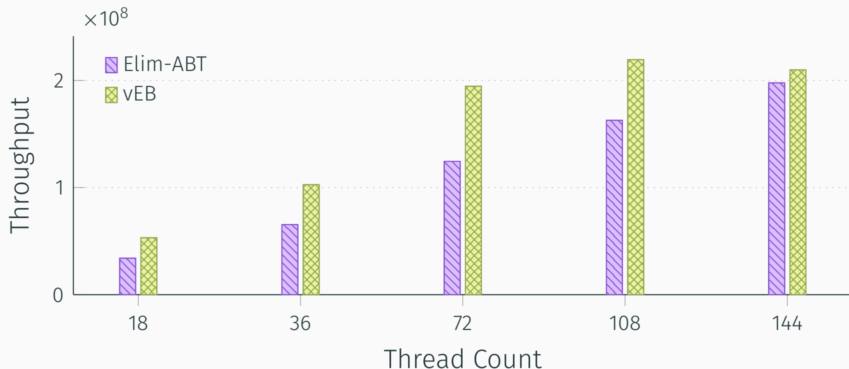


Memory Footprint



Successor Queries

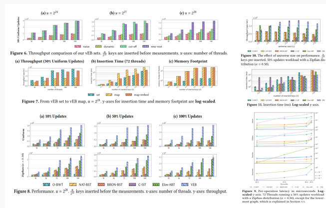
- The successor operation for Elim-ABT is non-linearizable
- Workload: 98% successor queries (Zipfian with $\alpha = 0.50$)



Summary

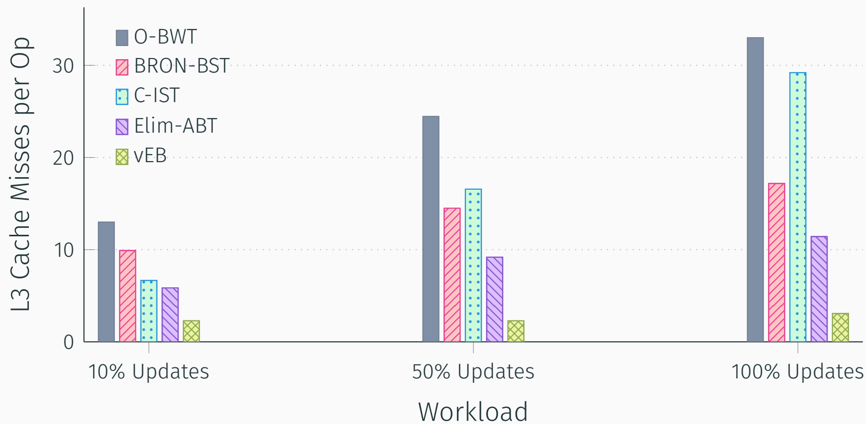
Summary

- Concurrent vEB trees utilizing HTM
- Low-overhead thread synchronization
- On average 5× faster
- More in the paper: https://bit.ly/htmveb
    - Zipfian workloads
    - Cache performance
    - Insertion time



Questions?

L3 Cache Misses



References i

[1] N. G. Bronson, J. Casper, H. Chafi, and K. Olukotun.
A practical concurrent binary search tree.
In *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP 2010, Bangalore, India, January 9-14, 2010*, pages 257–268. ACM, 2010.

[2] T. Brown, A. Prokopec, and D. Alistarh.
Non-blocking interpolation search trees with doubly-logarithmic running time.
In *PPoPP '20: 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, San Diego, California, USA, February 22-26, 2020*, pages 276–291. ACM, 2020.

References ii

[3] A. Srivastava and T. Brown.
Elimination (a, b)-trees with fast, durable updates.
In *PPoPP '22: 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Seoul, Republic of Korea, April 2 - 6, 2022*, pages 416–430. ACM, 2022.

[4] P. van Emde Boas, R. Kaas, and E. Zijlstra.
Design and implementation of an efficient priority queue.
Math. Syst. Theory, 10:99–127, 1977.

[5] Z. Wang, A. Pavlo, H. Lim, V. Leis, H. Zhang, M. Kaminsky, and D. G. Andersen.
Building a bw-tree takes more than just buzz words.
In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 473–488. ACM, 2018.