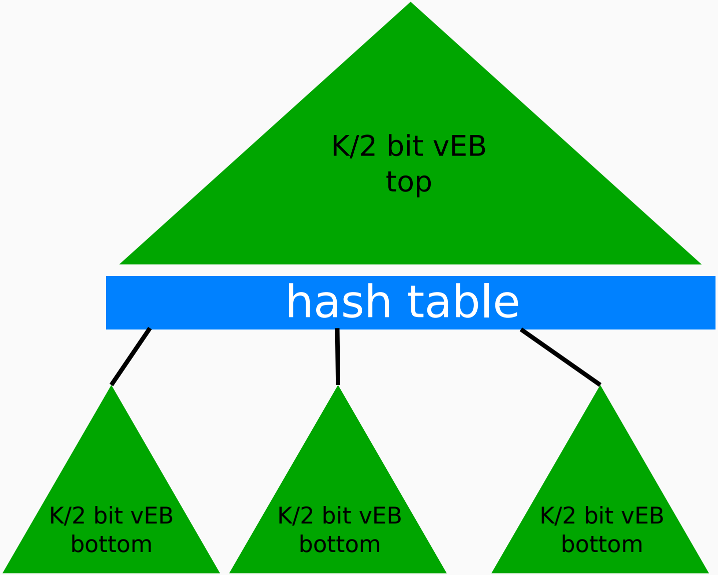
Van Emde-Boas Search Trees

Dominik Bez

Goals

- Implement van Emde-Boas tree as a fast std::set alternative
- Operations: insert, remove, locate/lower_bound in $O(\log \log n)$
    - locate$(x) = \min\{y \in VEB \mid y \geq x\}$
- Supporting unsigned/signed integers and floating point values
- Different variations
    - Generic van Emde-Boas tree (VEB)
    - Efficient specialized 32 bit implementation (VEB32)
    - Various concurrent versions of VEB32

Generic Van Emde-Boas Tree



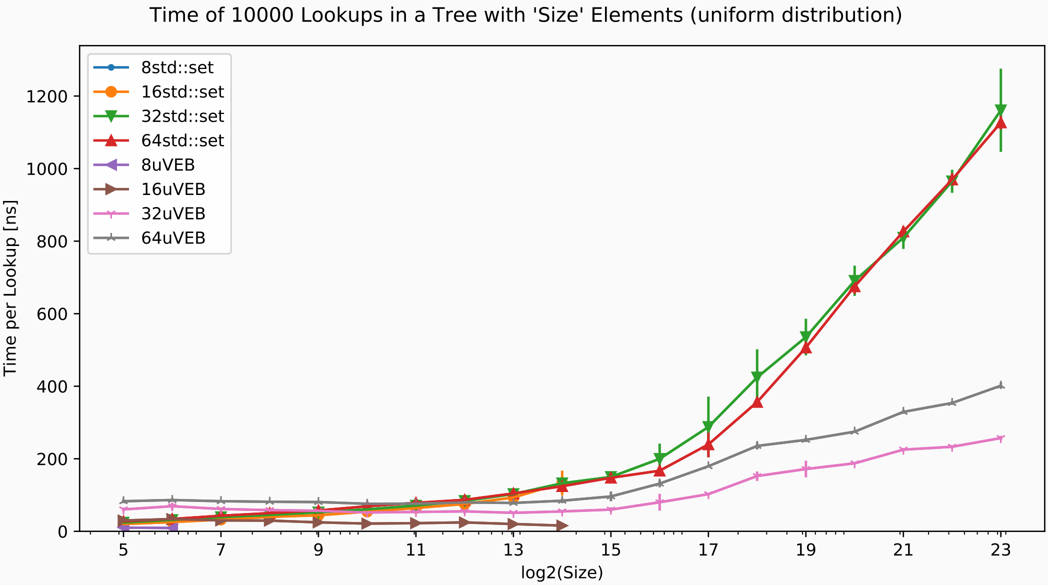
Generic Van Emde-Boas Tree - Differences to Algorithm Engineering Lecture

- Recursion ends for $K \leq 6$ ⇒ store a single 64 bit integer
  - Use bitwise and least/most significant bit operations
- Recursion also ends for a single value in bottom data structure
  - Least significant bit in pointer denotes this case
  - $K <$ sizeof(void*) ⇒ store data in the pointer itself
  - Otherwise, store the element directly on the heap

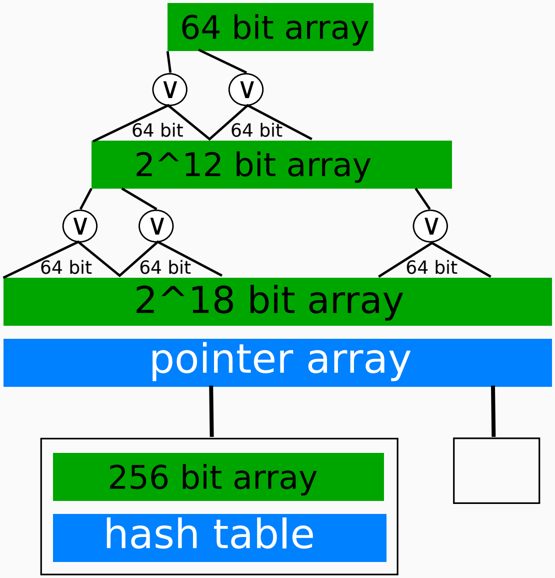
Generic Van Emde-Boas Tree - Recursion Base Case

```cpp
uint64_t storage;

void insert(const uint32_t value) noexcept {
  storage |= 1ULL << value;
}

void remove(const uint32_t value) noexcept {
  storage &= ~(1ULL << value);
}

uint32_t locate(const uint32_t value) const noexcept {
  return std::countr_zero(storage & ~((1ULL << value) - 1));
}
```

Generic Van Emde-Boas Tree - Evaluation



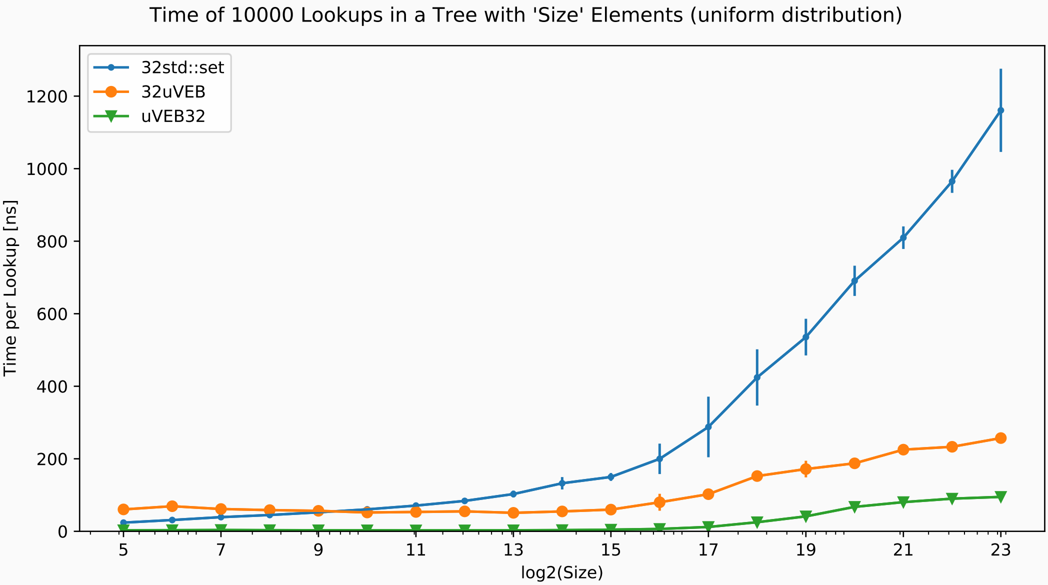
32 Bit Van Emde-Boas Tree



32 Bit Van Emde-Boas Tree - Differences to Algorithm Engineering Lecture

- Exploiting 64 bit processors
    - Top tree manages upper 18 bits instead of 16
    - Bottom trees manage lower 14 bits
        - Their top tree is flat and manages 8 bits
        - The remaining 6 bits are managed by a 64 bit integer again
        - ⇒ One level less than in the original paper
- Single values are stored directly in the pointer again
- Consumes at most 1 GB of memory compared to up to 100 GB

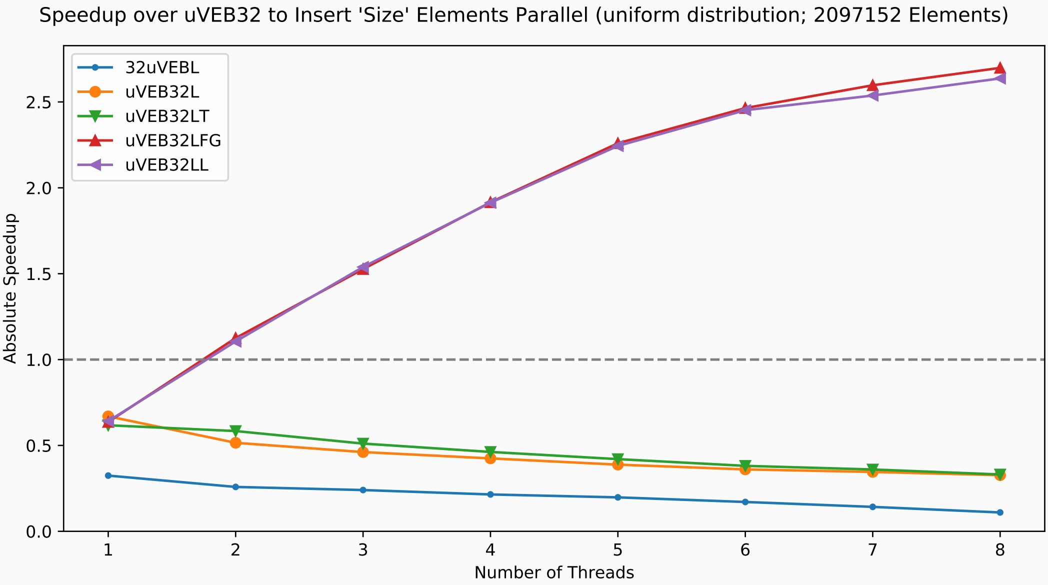
32 Bit Van Emde-Boas Tree - Evaluation



Concurrent Implementations

- Locked top
    - Top data structure completely locked (shared lock for locate)
    - Once the correct bottom data structure is found, it is locked instead
- Locked fine-grained
    - One mutex per 64 bottom data structures (one integer in top data structure)
- „Lockless“
    - Top data structure is lockless
    - Atomically swap pointer to bottom data structure with reserved value to reserve it (wait with spinlock)
- Remove always uniquely locks a shared mutex

Concurrent Implementations - Evaluation


Speedup over uVEB32 to Insert 'Size' Elements Parallel (uniform distribution; 2097152 Elements)
32uVEBL
uVEB32L
uVEB32LT
uVEB32LFG
uVEB32LL
Absolute Speedup
Number of Threads