Latency Hiding in Tree Lookups using Out Of Order Execution

Lukas Karnowski

November 28, 2017

Table of contents

Table of contents

Latendy Hiding in Tree Lookups using Out Of Order Execution

What the . . . ?

Latency Hiding

Latency Hiding
in Tree Lookups

Latency Hiding
in Tree Lookups
using Out Of Order Execution

Table of contents

Adaptive Radix Tree

Adaptive **R**radix **T**ree

Adaptive Rradix Tree

Whats so special?

Adaptive Rradix Tree

Whats so special?

- Improved radix tree (or prefix tree)

Adaptive Rradix Tree

Whats so special?

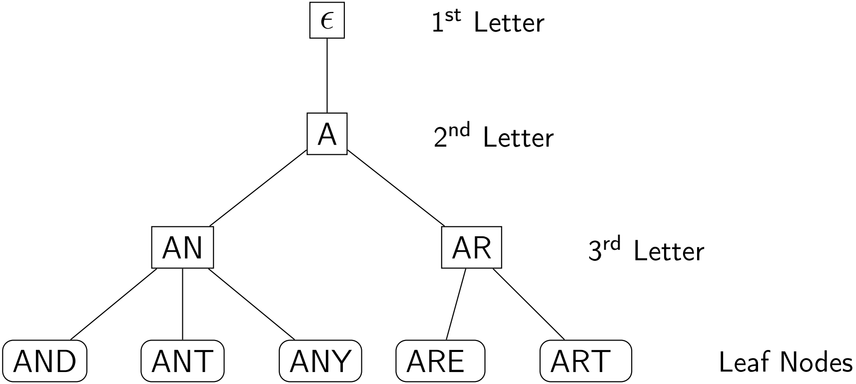
- Improved radix tree (or prefix tree)
- Dynamically adjusts node size

Adaptive Radix Tree

Whats so special?

- Improved radix tree (or prefix tree)
- Dynamically adjusts node size
- Can compress paths

Example radix tree



Different node types

- Node**4**
- Node**16**
- Node**48**
- Node**256**

Different node types

- Node**4**
- Node**16**
- Node**48**
- Node**256**

Example Node**4**:

Keys (1B each)				Pointer (8B each)			
0	13	42	255	Ptr to 0	Ptr to 13	Ptr to 42	Ptr to 255

Different node types

- Node**4**
- Node**16**
- Node**48**
- Node**256**

Example Node**4**:

Keys (1B each)				Pointer (8B each)			
0	13	42	255	Ptr to 0	Ptr to 13	Ptr to 42	Ptr to 255

Lookup using `findChild()`

Lookup algorithm

Lookup algorithm

```
lookup(node, key, depth):
  if node == NULL
    return NULL
  if isLeaf(node)
    if leafMatches(node, key, depth)
      return node
    return NULL
  // ...
  next = findChild(node, key[depth])
  return lookup(next, key, depth+1)
```

Lookup algorithm

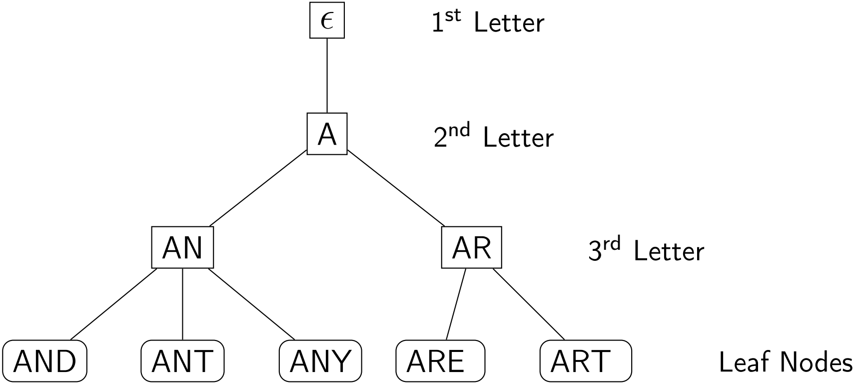


Table of contents

What is Out Of Order Execution?

$(a+b)+(c+d)$

What is Out Of Order Execution?

$(a+b)+(c+d)$

No dependency between **(a+b)** and **(c+d)**
$\rightarrow$ Can be calculated in parallel

What is Out Of Order Execution?

(a+b)+(c+d)

No dependency between **(a+b)** and **(c+d)**
$\rightarrow$ Can be calculated in parallel

Especially helpful for expensive operations, like **memory accesses**

Linked List Experiment

One list

Linked List data type:

```
struct Node {
  Node *next;
  std::uint8_t data[56];
};
```

Linked List Experiment

One list

Linked List data type:

```
struct Node {
  Node *next;
  std::uint8_t data[56];
};
```

Iteration:

```
for (Node *curr = list;
     curr != nullptr;
     curr = curr->next) {
  // Empty body
}
```

Linked List Experiment

One list

Linked List data type:

```
struct Node {
  Node *next;
  std::uint8_t data[56];
};
```

Iteration:

```
for (Node *curr = list;
     curr != nullptr;
     curr = curr->next) {
  // Empty body
}
```

In Assembler:

```
0x3590: mov  (%rax),%rax
0x3593: test %rax,%rax     ; depends on the first instr.
0x3596: jne  0x3590
```

Linked List Experiment

Two lists

Linked List Experiment

Two lists

```
for (Node *curr1 = list1, *curr2 = list2;
     curr1 != nullptr && curr2 != nullptr;
     curr1 = curr1->next, curr2 = curr2->next) {
  // Empty body
}
```

Linked List Experiment

Two lists

```
for (Node *curr1 = list1, *curr2 = list2;
      curr1 != nullptr && curr2 != nullptr;
      curr1 = curr1->next, curr2 = curr2->next) {
  // Empty body
}
```

In Assembler:

```
0x3600: mov    (%rax),%rax
0x3603: mov    (%rdx),%rdx  ; No dependency!
0x3606: test   %rax,%rax
0x3609: je     0x3610
0x360b: test   %rdx,%rdx
0x360e: jne    0x3600
0x3610: ...
```

Linked List Experiment

Results

Linked List Experiment

Results

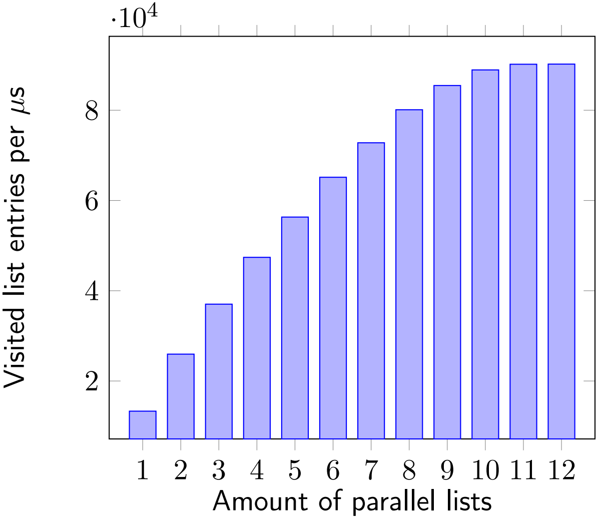


Table of contents

Basic idea

Basic idea

Perform multiple lookups at the same time

Basic idea

Perform multiple lookups at the same time

This technique is called **Group Prefetching**

Basic idea

Perform multiple lookups at the same time

This technique is called **Group Prefetching**

Keep track of every lookup

Tracking each state

How can we track the state of each lookup?

Tracking each state

How can we track the state of each lookup?

```
struct GPState {
  std::uint8_t key[8];
  Node *node;

  unsigned depth = 0;
  // ...
  bool finished = false;

  GPState() : node(nullptr) {}
  GPState(Node *node) : node(node) {}
};
```

The actual lookup algorithm

The actual lookup algorithm

```
void lookupGP(std::vector<GPState> &states) {
  while (/* not all finished */) {
    // Loop over every state
    for (auto &state : states) {
      if (state.finished)
        continue;

      // Perform the normal lookup algorithm step
      if (state.node == NULL || isLeaf(state.node)) {
        state.finished = true;
        continue;
      }
      state.node = *findChild(state.node,
                              state.key[state.depth]);
      state.depth++;
    }
  }
}
```

Benchmarking

Benchmarking

- TPC-H benchmark (see e.g. HyperDB Webinterface)

Benchmarking

- TPC-H benchmark (see e.g. HyperDB Webinterface)
- Joining `lineitem` with `orders`

Benchmarking

- TPC-H benchmark (see e.g. HyperDB Webinterface)
- Joining `lineitem` with `orders`
- `lineitem` has foreign key to `orders`

Benchmarking

- TPC-H benchmark (see e.g. HyperDB Webinterface)
- Joining `lineitem` with `orders`
- `lineitem` has foreign key to `orders`
- Creating an index on `orders`

Benchmarking

- TPC-H benchmark (see e.g. HyperDB Webinterface)
- Joining `lineitem` with `orders`
- `lineitem` has foreign key to `orders`
- Creating an index on `orders`
- Iterating the tuples in `lineitem` and performing a lookup in the ART for `orders` (with multiple keys using GP)

Benchmarking

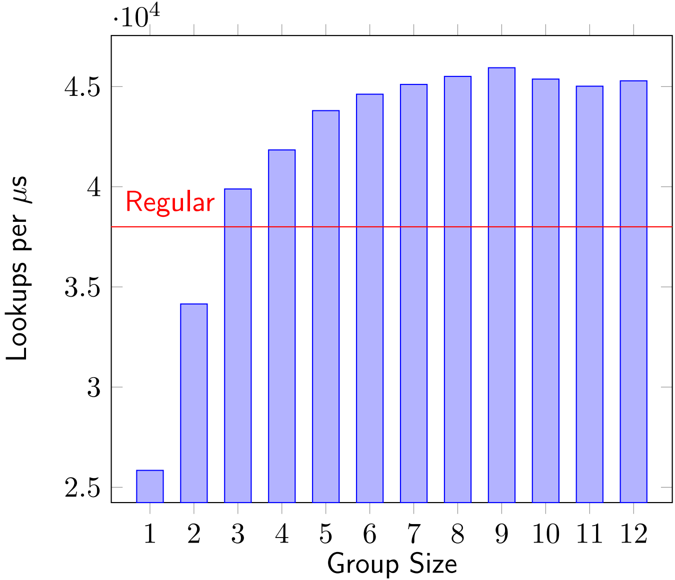
- TPC-H benchmark (see e.g. HyperDB Webinterface)
- Joining `lineitem` with `orders`
- `lineitem` has foreign key to `orders`
- Creating an index on `orders`
- Iterating the tuples in `lineitem` and performing a lookup in the ART for `orders` (with multiple keys using GP)
- Amount of parallel lookups is called **Group Size**

Benchmarking Results

Ordered

Benchmarking Results

Ordered

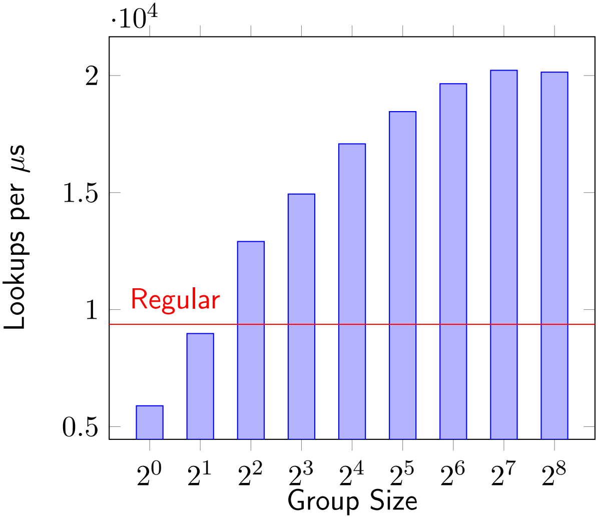


Benchmarking Results

Unordered

Benchmarking Results

Unordered



My reaction

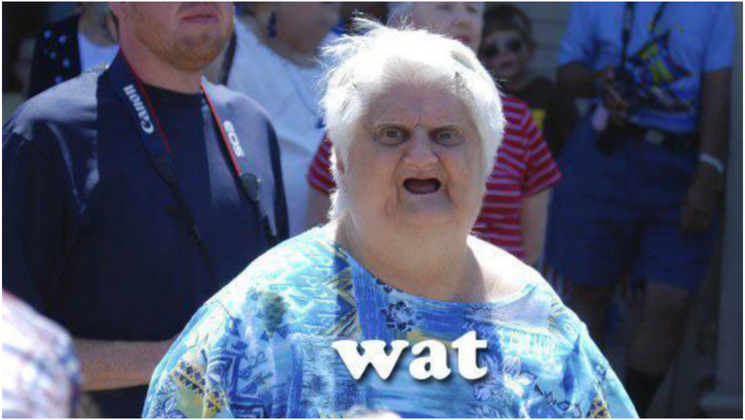


Table of contents

Lessons learned?

Group Prefetching . . .

Lessons learned?

Group Prefetching . . .

- . . . increases Performance, but not as much as seen in the Linked List experiment

Lessons learned?

Group Prefetching ...

- ... increases Performance, but not as much as seen in the Linked List experiment
- ... gives about 200% speed increase

Lessons learned?

Group Prefetching ...

- ... increases Performance, but not as much as seen in the Linked List experiment
- ... gives about 200% speed increase
- ... is always useful, when lookup keys are known in advance (e.g. during a Join)

Lessons learned?

Group Prefetching ...

- ... increases Performance, but not as much as seen in the Linked List experiment
- ... gives about 200% speed increase
- ... is always useful, when lookup keys are known in advance (e.g. during a Join)
- ... can be adjusted using the Group Size variable. Concrete value changes speed increase
  $\rightarrow$ perfect value depends on use case and hardware

Lessons learned?

Group Prefetching ...

- ... increases Performance, but not as much as seen in the Linked List experiment
- ... gives about 200% speed increase
- ... is always useful, when lookup keys are known in advance (e.g. during a Join)
- ... can be adjusted using the Group Size variable. Concrete value changes speed increase
  $\rightarrow$ perfect value depends on use case and hardware

Out Of Order Execution is quite cool

Q & A

Table of contents

- Viktor Leis, Alfons Kemper, Thomas Neumann. *The Adaptive Radix Tree: ARTful Indexing for Main-Memory Databases*. 2013.
- Onur Kocberber, Babak Falsafi, Boris Grot. *Asynchronous Memory Access Chaining*. 2015.
- Agner Fog. *Optimizing software in C++*. 2017.
- Andrew S. Tanenbaum, Todd Austin. *Structured Computer Organization*. Pearson, 2013.
- `http://i0.kym-cdn.com/entries/icons/mobile/000/001/007/WAT.jpg`