Cache-Oblivious Dynamic Dictionaries with Optimal Update/Query Tradeoff

Gerth Stølting Brodal
Aarhus University
Center of
- Lars Arge, Professor, Centerleader
- Gerth S. Brodal, Associate Professor
- 5 Post Docs, 10 PhD students, 4 TAP
- Total budget for 5 years ca. 60 million DKR
I/O Efficient Algorithms

Streaming Algorithms

Cache Oblivious Algorithms

Algorithm Engineering
The problem...

- input size
- running time

Normal algorithm

I/O-efficient algorithm

bottleneck = memory size
Memory Hierarchies

Increasing access times and memory sizes

CPU

Processor

L1

L2

L3

RAM

Disk

Bottleneck
Memory Hierarchies vs. Running Time

Data Size vs. Running Time

L1, L2, L3, RAM

køretid vs. datastørrelse
# Memory Access Times

<table>
<thead>
<tr>
<th></th>
<th>Latency</th>
<th>Relative to CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>0.5 ns</td>
<td>1</td>
</tr>
<tr>
<td>L1 cache</td>
<td>0.5 ns</td>
<td>1-2</td>
</tr>
<tr>
<td>L2 cache</td>
<td>3 ns</td>
<td>2-7</td>
</tr>
<tr>
<td>DRAM</td>
<td>150 ns</td>
<td>80-200</td>
</tr>
<tr>
<td>TLB</td>
<td>500+ ns</td>
<td>200-2000</td>
</tr>
<tr>
<td>Disk</td>
<td>10 ms</td>
<td>$10^7$</td>
</tr>
</tbody>
</table>
“The difference in speed between modern CPU and disk technologies is analogous to the difference in speed in sharpening a pencil using a sharpener on one’s desk or by taking an airplane to the other side of the world and using a sharpener on someone else’s desk.” (D. Comer)
Disk Mechanics

- I/O is often bottleneck when handling massive datasets
- Disk access is $10^7$ times slower than main memory access!
- Disk systems try to amortize large access time transferring large contiguous blocks of data
- Need to store and access data to take advantage of blocks!
Internal Memory Index
B(oing)-Trees

One node = one disk block

Height $O(\log_B N)$
B-trees -
The Basic Searching Structure

- Searches
  Practice: 4-5 I/Os

- Repeated searching
  Practice: 1-2 I/Os

Search path
The Bad News...

Searching any external memory dictionary (incl. B-trees) requires worst-case

$$\Omega(\log_B N) \text{ I/Os}$$

Proof idea:

$$N \geq (N - B) / (B + 1)$$
B-trees

- **Searches** $O(\log_B N)$ I/Os
- **Updates** $O(\log_B N)$ I/Os
B-trees with Buffered Updates

- Searches cost
  
  \[ O(\log_d N) \text{ I/Os} \]
  
  \[ = O(\log_B N \cdot 1/\epsilon) \text{ I/Os} \]

- \( N \) updates cost
  
  \[ O(N \cdot \log_d N \cdot d / B) \text{ I/Os} \]
  
  \[ = O(N \cdot \log_B N \cdot 1/\epsilon B^{1-\epsilon}) \text{ I/Os} \]

Trade-off between search and update times – optimal!
B-trees with Buffered Updates

\[ \frac{N}{(M \cdot \left(\frac{M}{B}\right)^{\Theta(\delta)}} \right) \]

\[ \Theta(\log_{\delta} \frac{N}{M}) \]

\[ \frac{1}{\varepsilon} \log_B \frac{N}{M} \]

\[ \log_B \frac{N}{M} \]

\( \Theta(\log_{M/B} \frac{N}{M}) \)

\( \log^{1+\varepsilon} N \)

\( B/\log^3 N \)

\( B \log_B \frac{N}{M} \)
B-trees with Buffered Updates
Experimental Study

• 100.000.000 elements

• Search time basically unchanged with buffers

• 100 times faster updates

Hedegaard (2004)
Assumptions until now:

* $B$ is known + one level is the bottleneck

Increasing access times and memory sizes
increasing access times and memory sizes

Algorithm **does not know** $B$ and $M$

(assume optimal offline cache-replacement strategy)
I/O Efficient Scanning

```
sum = 0
for i = 1 to N do sum = sum + A[i]
```

\[ O\left(\frac{N}{B}\right) \] I/Os
B(oring)-Trees

One node = one disk block

Height $O(\log_B N)$
Recursive Search Tree Layout (Cache-Obliviousness)

Searches require $O(\log_B N)$ I/Os
(small subtrees have $\sqrt{B} \leq \text{size} \leq B$, a path traverses at most $2 \cdot \log_B N$ trees)
Experiments: Binary Tree Layout

DFS

BFS

inorder

van Emde Boas (in theory best)
Searches with Pointer Layout

- van Emde Boas layout wins, followed by the BFS layout
Searches with Implicit Layout

- BFS layout wins due to simplicity and caching of topmost levels
- van Emde Boas layout requires quite complex index computations
• van Emde Boas layout wins, followed by the BFS layout
Optimal Cache-Oblivious Static Index

= van Emde Boas Layout of Complete Binary Tree
Cache-Oblivious Dynamic Index?
Binary Search Trees
Dynamic

- **Embed** a dynamic tree of small height into a complete tree
- Static van Emde Boas layout
- Rebuild data structure whenever $N$ doubles or halves

Brodal, Fagerberg, Jacob (2002)
If an insertion causes non-small height then **rebuild** subtree at nearest ancestor with sufficient few descendants

- Insertions require amortized time $O(\log^2 N)$
Optimal Cache-Oblivious Dynamic Index

- Search $O(\log_B N)$ I/Os (optimal)
- Updates $O(\log_B N + (\log^2 N) / B)$ I/Os
Cache-Oblivious Index with Query-Update Trade-off ?
Cache-Oblivious Dynamic Dictionaries with Query-Update Trade-off

- Solution matching the I/O complexity of Buffered B-trees

- Searches $O(\log_B N \cdot 1/\varepsilon)$ I/Os

- $N$ updates $O(N \cdot \log_B N \cdot 1/\varepsilon B^{1-\varepsilon})$ I/Os
**xDict**

**Insert**
- Insert into smallest box
- When a box reaches capacity, **Flush** it and **Batch-Insert** into the next box

**Search**
- Search in each $x$-box
- $O(\log_B x)$ cost is dominated by largest box $O(\log_B N)$
**x-Box (capacity $x^2$)**

- **Size-$$x$$ input buffer**
- **Size-$$x^{3/2}$$ middle buffer**
- **Size-$$x^2$$ output buffer**

- **Upper level:** at most $x^{1/2}/4$ subboxes
- **Lower level:** at most $x/4$ subboxes

**Memory Layout**

- Subboxes stored contiguously in arbitrary order
- Unused (currently empty) subboxes are preallocated

Batch insert $x$ elements

Flush
External Memory Index

- Searches $O(\log_B N \cdot 1/\epsilon)$ I/Os

- $N$ updates $O(N \cdot \log_B N \cdot 1/\epsilon B^{1-\epsilon})$ I/Os