Cache-Oblivious String Dictionaries

Gerth Stølting Brodal
University of Aarhus

Rolf Fagerberg
University of Southern Denmark

SODA 2006, January 22-24, 2006, Miami, Florida
Computational Models
# Computational Models

<table>
<thead>
<tr>
<th>Parameters</th>
<th>RAM (von Neumann 1946)</th>
<th>I/O Model (Aggarwal and Vitter 1988)</th>
<th>Ideal Cache Model (Frigo et al. 1999)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>$N$</td>
<td>$N, M, B$</td>
<td>$N, M, B$</td>
</tr>
<tr>
<td>Properties</td>
<td>Simpel</td>
<td>Cache aware: $M$ and $B$ known</td>
<td>Cache oblivious: $M$ and $B$ unknown</td>
</tr>
<tr>
<td>Complexity</td>
<td># instructions</td>
<td># I/Os</td>
<td># I/Os</td>
</tr>
</tbody>
</table>

$N = \text{problem size}$

$M = \text{memory size}$

$B = \text{I/O block size}$
Ideal Cache Model — no parameters!

Frigo, Leiserson, Prokop, Ramachandran 1999

- Program with only one memory
- Analyze in the I/O model for
- Optimal off-line cache replacement strategy arbitrary $B$ and $M$

Advantages

- Optimal on arbitrary level $\Rightarrow$ optimal on all levels
- Portability, $B$ and $M$ not hard-wired into algorithm
- Dynamic changing $M$ (and $B$)
Cache-Oblivious Preliminaries
Cache-Oblivious Scanning

\[ O \left( \frac{N}{B} \right) \text{ I/Os} \]
Static Cache-Oblivious B-Tree

Recursive layout of binary tree \( \equiv \) van Emde Boas layout
Static Cache-Oblivious B-Tree

Searches perform $O(\log_B N)$ I/Os.
Static Cache-Oblivious B-Tree

Searches perform $O(\log_B N)$ I/Os.
Static Cache-Oblivious B-Tree

Searches perform $O(\log_B N)$ I/Os
Summary Cache-Oblivious Tools

<table>
<thead>
<tr>
<th>Operation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scanning</td>
<td>$O(N/B)$</td>
</tr>
<tr>
<td>B-tree searching</td>
<td>$O(\log_B N)$</td>
</tr>
<tr>
<td>Sorting*</td>
<td>$O\left(\frac{N}{B} \log_{M/B} \frac{N}{B}\right)$</td>
</tr>
</tbody>
</table>

* requires a tall cache assumption $M \geq B^{1+\epsilon}$

Frigo, Leiserson, Prokop, Ramachandran 1999
Brodal and Fagerberg 2002, 2003
String Dictionaries
## String Dictionaries

<table>
<thead>
<tr>
<th>Structure</th>
<th>RAM</th>
<th>I/O Model</th>
<th>Ideal Cache Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tries</td>
<td>$O(</td>
<td>P</td>
<td>)$</td>
</tr>
<tr>
<td>String B-trees</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
</tr>
</tbody>
</table>

### Query

- **RAM**: $O(|P|)$
- **I/O Model**: $O(|P|/B + \log_B n)$
- **Ideal Cache Model**: $O(|P|/B + \log_B n)$

### Space

- **RAM**: $O(N)$
- **I/O Model**: $O(N)$
- **Ideal Cache Model**: $O(N)$

### Dynamic vs. Static

- Dynamic
- Static, $M \geq B^{2+\varepsilon}$

### Definitions

- $n$: number of strings
- $N$: total length of strings
- $P$: query string
The Trouble Starts...

- Tries cannot be laid out in memory to support top-down searches in $O(\log_B N + |P|/B)$ I/Os

  Demaine et al. 2004

- Can construct suffix trees cache-obliviously using $O(\text{Sort}(N))$ I/Os, but cannot search in it efficiently...

  Farach et al. 2000

+ Cache-aware string B-trees support searches in a set of strings in $O(\log_B n + |P|/B)$ I/Os

  Ferragina and Grossi 1999
Tries vs Blind Tries

Trie

Blind trie
String Dictionary

Queries: Search blind trie + Verify one string
String Dictionary

Queries: Search blind trie + Verify one string
**Suffix Tree**

```
T  a a b a c d a c a b a b $
  1 2  ...  7  ...  13

P  a c a d a
```

Queries: **Search blind trie** + **Verify one suffix**
Suffix Tree

Queries: Search blind trie + Verify one suffix

Cache-Oblivious String Dictionaries
Tries

Queries: Search blind trie + Verify prefix of one path
Tries

Queries: Search blind trie + Verify prefix of one path

Cache-Oblivious String Dictionaries
Verifying a Prefix of a Path in a Tree
Verifying Paths in Giraffe Trees is Easy

Definition  A tree is a giraffe tree if all root-to-leaf paths share at least half of the nodes of the tree (long neck)
Verifying Paths in Giraffe Trees is Easy

**Definition**  A tree is a giraffe tree if all root-to-leaf paths share at least half of the nodes of the tree (long neck)

- A prefix of length $p$ of a path in a giraffe tree using a BFS layout can be traversed in $O(p/B)$ I/Os
Giraffe Cover of a Tree

- Constructed left-to-right using $O(N/B)$ I/Os and space $O(N)$
- A prefix of length $p$ of a path in a known giraffe in BFS layout can be traversed in $O(p/B)$ I/Os
Summary so far...

String dictionary search
Suffix tree search
Trie search

\[ \text{reduce to blind trie search} \]

Query: Blind trie search + \( O \left( 1 + \frac{|P|}{B} \right) \) I/Os
Cache-Oblivious (Blind) Tries
Cache-Oblivious (Blind) Tries

- Partition input trie $T$ into components (generalization of heavy paths)
- $T' = \text{collapse components in } T \text{ into high degree nodes and replace by weight balanced trees}$
- Apply van Emde Boas layout out to $T'$

Search: $O(\log_B n)$ I/O — ignoring searching inside components
Decomposition into Components

\[ D_v^0 = \{ u \in T_v \mid \text{rank}(u) = \text{rank}(v) \land \text{depth}(u) - \text{depth}(v) < 2^{2^0} \} \]

\[ D_v^i = \{ u \in T_v \mid \text{rank}(v) - \text{rank}(u) < \varepsilon 2^i \]

\[ \land 2^{2^{i-1}} \leq \text{depth}(u) - \text{depth}(v) < 2^{2^i} \} \]
Storing and Searching Components

- Store each layer $D^i_v$ separately
- Make a giraffe-decomposition of $D^i_v$
- For $D^i_v$ have a blind trie of size $O(2^{ε^2i})$ (using BFS layout) to select the right giraffe-tree
- Search: $D^i_v$ search the blind trie + search in one giraffe-tree
- Distribute $D^0_v, D^1_v, D^2_v, \ldots$ in the van Emde Boas layout of $T'$
- Analysis:
  - Search in blind trie for $D^{i+1}_v$ (lookahead) dominated by the matched characters in $D^i_v$ — requires $M \geq B^{2+δ}$
  - Space in van Emde Boas layout for a subtree of size $k$ becomes $O(k^3)$
There exists a cache-oblivious trie supporting prefix queries in

$$O(\log_B n + |P|/B)$$ I/Os,

where $P$ is the query string, and $n$ is the number of leaves in the trie.

It can be constructed in $O(\text{Sort}(N))$ time, where $N$ is the total number of characters in the input.

The space required is $O(N)$.

The structure assumes $M \geq B^{2+\delta}$. 
Final Remarks

- Lookahead in the query string is crucial (both cache-aware and cache-oblivious)

- A giraffe cover is a simple construction allowing topdown path traversals in a tree using $O(|P|/B)$ I/Os
The End