CSE 638: Advanced Algorithms

Lectures 20 & 21
( Cache-oblivious Priority Queue with Decrease-Keys )

Rezaul A. Chowdhury
Department of Computer Science
SUNY Stony Brook
Spring 2013
## Cache-Oblivious Buffer Heap

<table>
<thead>
<tr>
<th></th>
<th>Priority Queue</th>
<th>Amortized I/O Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Delete / Delete-Min</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Decrease-Key</strong></td>
</tr>
<tr>
<td>Cache-oblivious</td>
<td>Buffer Heap</td>
<td>(O\left(\frac{1}{B} \log_2 \frac{N}{B}\right))</td>
</tr>
<tr>
<td>Cache-aware</td>
<td>Tournament Tree</td>
<td>(O\left(\log_2 N\right))</td>
</tr>
<tr>
<td>Internal Memory</td>
<td>Binary Heap (worst-case)</td>
<td>(O\left(\log_2 N\right))</td>
</tr>
<tr>
<td>Internal Memory</td>
<td>Fibonacci Heap</td>
<td>(O(1))</td>
</tr>
</tbody>
</table>
Consists of $r = 1 + \lceil \log_2 N \rceil$ levels, where $N$ = total number of elements.

For $0 \leq i \leq r - 1$, level $i$ contains two buffers:

- **element buffer** $B_i$
  - contains elements of the form $(x, k_x)$, where $x$ is the element id, and $k_x$ is its key

- **update buffer** $U_i$
  - contains updates (*Delete*, *Decrease-Key* and *Sink*), each augmented with a time-stamp.
**Invariant 1:** \(|B_i| \leq 2^i\)

**Invariant 2:**

(a) No key in \(B_i\) is larger than any key in \(B_{i+1}\)

(b) For each element \(x\) in \(B_i\), all updates yet to be applied on \(x\) reside in \(U_0, U_1, \ldots, U_i\)

**Invariant 3:**

(a) Each \(B_i\) is kept sorted by element id

(b) Each \(U_i\) (except \(U_0\)) is kept (coarsely) sorted by element id and time-stamp
Cache-Oblivious Buffer Heap: Operations

The following operations are supported:

- **Delete**(x):
  Deletes the element x from the queue.

- **Delete-Min()**:
  Extracts an element with minimum key from queue.

- **Decrease-Key**(x, k_x): (weak Decrease-Key)
  If x already exists in the queue, replaces key k'_x of x with min(k_x, k'_x),
  otherwise inserts x with key k_x into the queue.

A new element x with key k_x can be inserted into queue by **Decrease-Key**(x, k_x).
**Cache-Oblivious Buffer Heap: Operations**

*Decrease-Key*(x, k_x) :
- Insert the operation into U_0 augmented with current time-stamp.

*Delete*(x) :
- Insert the operation into U_0 augmented with current time-stamp.

*Delete-Min*( ) :
- Two phases:
  - Descending Phase (Apply Updates)
  - Ascending Phase (Redistribute Elements)
Cache-Oblivious Buffer Heap: Delete-Min

Delete-Min() - Descending Phase (Apply Updates):

\[ B_0 \]

\[ B_1 \]

\[ B_2 \]

\[ B_{k-1} \]

\[ B_k \]

\[ U_0 \]

\[ U_1 \]

\[ U_2 \]

\[ U_{k-1} \]

\[ U_k \]

\[ U_{k+1} \]
Cache-Oblivious Buffer Heap: Delete-Min

Delete-Min( ) - Descending Phase (Apply Updates):

1. sort updates

\[ B_0 \]
\[ B_1 \]
\[ B_2 \]
\[ B_{k-1} \]
\[ B_k \]

\[ U_0 \]
\[ U_1 \]
\[ U_2 \]
\[ U_{k-1} \]
\[ U_k \]
\[ U_{k+1} \]
Delete-Min( ) - Descending Phase (Apply Updates):

2. apply updates:
   - Delete(x)
   - Decrease-Key(x, k_x)
Delete-Min() - Descending Phase (Apply Updates):

3. carry updates:
   - all updates that did not apply
   - applied Decrease-Keys as Deletes
Cache-Oblivious Buffer Heap: Delete-Min

\textbf{Delete-Min( ) - Descending Phase ( Apply Updates ) :}

1. sort updates:  
   - merge segments

\begin{itemize}
\item $B_0$
\item $B_1$
\item $B_2$
\item $B_{k-1}$
\item $B_k$
\end{itemize}

\begin{itemize}
\item $U_0$
\item $U_1$
\item $U_2$
\item $U_{k-1}$
\item $U_k$
\item $U_{k+1}$
\end{itemize}
Cache-Oblivious Buffer Heap: Delete-Min

Delete-Min() - Descending Phase (Apply Updates):

2. apply updates:
   - Delete(x)
   - Decrease-Key(x, k_x)
   - Sink(x, k_x)
Delete-Min() - Descending Phase (Apply Updates):

3. carry updates
Cache-Oblivious Buffer Heap: Delete-Min

**Delete-Min**() - **Descending Phase** (Apply Updates):
Delete-Min() - Descending Phase (Apply Updates):

- $B_0$
- $B_1$
- $B_2$
- $B_{k-1}$
- $B_k$

$U_0$
$U_1$
$U_2$
$U_{k-1}$
$U_k$
$U_{k+1}$
Cache-Oblivious Buffer Heap: Delete-Min

Delete-Min() - Descending Phase (Apply Updates):

- $B_0$
- $B_1$
- $B_2$
- $B_{k-1}$
- $B_k$

$U_0$
$U_1$
$U_2$
$U_{k-1}$
$U_k$
$U_{k+1}$
Delete-Min( ) - Ascending Phase (Redistribute Elements):

- Find $2^k$ elements with $2^k$ smallest keys
- Convert each remaining element $x$ with key $k_x$ to $\text{Sink}(x, k_x)$
Cache-Oblivious Buffer Heap: Delete-Min

Delete-Min() - Ascending Phase (Redistribute Elements):

- $B_0$ to $B_{k-1}$ are empty buffers.
- $B_k$ contains elements and will be redistributed.
- $U_0$ to $U_{k+1}$ are the union buffers.

Push the Sinks to $U_{k+1}$.
Delete-Min( ) - Ascending Phase (Redistribute Elements):

\[ B_0 \quad \square \]
\[ B_1 \quad \square \]
\[ B_2 \quad \square \]
\[ B_{k-1} \quad \square \]
\[ B_k \quad \bullet \bullet \bullet \bullet \bullet \bullet \bullet \bullet \bullet \bullet \bullet \]

\[ U_0 \quad \square \]
\[ U_1 \quad \square \]
\[ U_2 \quad \square \]
\[ U_{k-1} \quad \square \]
\[ U_k \quad \square \]
\[ U_{k+1} \quad \square \]
Cache-Oblivious Buffer Heap: Delete-Min

Delete-Min() - Ascending Phase (Redistribute Elements):

move all elements from $B_k$ to shallower levels leaving $B_k$ empty
Cache-Oblivious Buffer Heap: Delete-Min

Delete-Min() - Ascending Phase (Redistribute Elements):

\[
\begin{align*}
B_0 & \quad \boxed{} & U_0 \\
B_1 & \quad \boxed{} & U_1 \\
B_2 & \quad \boxed{} & U_2 \\
B_{k-1} & \quad \boxed{} & U_{k-1} \\
B_k & \quad \boxed{} & U_k \\
& \quad \boxed{} & U_{k+1}
\end{align*}
\]
Cache-Oblivious Buffer Heap: Delete-Min

Delete-Min() - Ascending Phase (Redistribute Elements):

\[ B_0 \quad U_0 \]
\[ B_1 \quad U_1 \]
\[ B_2 \quad U_2 \]
\[ B_{k-1} \quad U_{k-1} \]
\[ B_k \quad U_k \]
\[ U_{k+1} \]
Cache-Oblivious Buffer Heap: Delete-Min

Delete-Min() - Ascending Phase (Redistribute Elements):

$B_0$  

$B_1$  

$B_2$  

$B_{k-1}$  

$B_k$  

$U_0$  

$U_1$  

$U_2$  

$U_{k-1}$  

$U_k$  

$U_{k+1}$
Cache-Oblivious Buffer Heap: Delete-Min

Delete-Min( ) - Ascending Phase (Redistribute Elements):

\[ B_0 \quad B_1 \quad B_2 \quad \ldots \quad B_{k-1} \quad B_k \]

\[ U_0 \quad U_1 \quad U_2 \quad \ldots \quad U_{k-1} \quad U_k \quad U_{k+1} \]
Cache-Oblivious Buffer Heap: Delete-Min

Delete-Min() - Ascending Phase (Redistribute Elements):

- element with minimum key

\[ B_0 \]

\[ B_1 \]

\[ B_2 \]

\[ B_{k-1} \]

\[ B_k \]
Cache-Oblivious Buffer Heap: I/O Complexity

Potential Function: \[ \Phi(H) = \frac{1}{B} \left( 4r |U_0| + \sum_{i=1}^{r-1} (3r - i) |U_i| + \sum_{i=0}^{r-1} (i + 1) |B_i| \right) \]

**Lemma:** A **Buffer Heap** on \( N \) elements supports **Delete**, **Delete-Min** and **Decrease-Key** operations cache-obliviously in \( O\left(\frac{1}{B} \log_2 N\right) \) amortized I/Os each using \( O(N) \) space.
**Potential Function:** \( \Phi(H) = \frac{1}{B} \left( 4r |U_0| + \sum_{i=1}^{r-1} (3r - i)|U_i| + \sum_{i=0}^{r-1} (i + 1)|B_i| \right) \)

**Lemma:** A *Buffer Heap* on \( N \) elements supports *Delete*, *Delete-Min* and *Decrease-Key* operations cache-obliviously in \( O\left(\frac{1}{B \log_2 N}\right) \) amortized I/Os each using \( O(N) \) space.
Potential Function: \[ \Phi(H) = \frac{1}{B} \left( 4r |U_0| + \sum_{i=1}^{r-1} (3r - i) |U_i| + \sum_{i=0}^{r-1} (i + 1) |B_i| \right) \]

**Lemma:** A Buffer Heap on \( N \) elements supports *Delete*, *Delete-Min* and *Decrease-Key* operations cache-obliviously in \( O\left(\frac{1}{B} \log_2 N \right) \) amortized I/Os each using \( O(N) \) space.
Potential Function: \( \Phi(H) = \frac{1}{B} \left( 4r |U_0| + \sum_{i=1}^{r-1} (3r - i)|U_i| + \sum_{i=0}^{r-1} (i + 1)|B_i| \right) \)

Lemma: A Buffer Heap on \( N \) elements supports \textit{Delete}, \textit{Delete-Min} and \textit{Decrease-Key} operations cache-obliviously in \( O\left( \frac{1}{B} \log_2 N \right) \) amortized I/Os each using \( O(N) \) space.
Cache-Oblivious Buffer Heap: I/O Complexity

Potential Function: \[ \Phi(H) = \frac{1}{B} \left( 4r |U_0| + \sum_{i=1}^{r-1} (3r - i)|U_i| + \sum_{i=0}^{r-1} (i + 1)|B_i| \right) \]

Lemma: A Buffer Heap on \( N \) elements supports \textit{Delete}, \textit{Delete-Min} and \textit{Decrease-Key} operations cache-obliviously in \( \mathcal{O}\left(\frac{1}{B \log_2 N}\right) \) amortized I/Os each using \( \mathcal{O}(N) \) space.
Cache-Oblivious Buffer Heap: Layout

- All $B_i$’s are kept in a stack $S_B$.
- All $U_i$’s are kept in a stack $S_U$.
- An array $A_s$ is maintained in a stack $S_A$.
  for $0 \leq i \leq r - 1$ $A_s[i]$ contains:
  - $|B_i|$
  - number of segments in $U_i$
  - number of updates in each segment of $U_i$

In both stacks lower level buffers are placed above higher level buffers. The left to right order of the elements in any buffer are maintained top to bottom in the stack.
After each operation check whether $\sum |U_i| \geq \sum |B_i|$, and if so,

**Step 1:** Sort the elements in $S_B$ by element id and level number.

**Step 2:** Sort the updates in $S_U$ by element id and time-stamp.

**Step 3:** Scan $S_B$ and $S_u$ simultaneously, and apply the updates in $S_u$ on the elements of $S_B$.

**Step 4:** Reconstruct the data structure by filling the shallowest levels with the current elements in $S_B$, and emptying $S_u$. 

**Cache-Oblivious Buffer Heap: Reconstruction**
Cache-Oblivious Buffer Heap: I/O Complexity

**Lemma:** A BH supports *Delete*, *Delete-Min*, and *Decrease-Key* operations in \( O((1 / B) \log_2(N / B)) \) amortized I/Os each assuming a tall cache.

**Potential Method:**

Associates credit with the entire data structure instead of specific objects.

- \( D_0 \) = initial state of the data structure
- \( D_i \) = state of data structure after \( i \)-th operation, \( i = 1, 2, \ldots, n \)
- \( \Phi \) = a *potential function* mapping each \( D_i \) to a real number \( \Phi(D_i) \)
- \( c_i / a_i \) = actual / amortized cost of the \( i \)-th operation, \( i = 1, 2, \ldots, n \)

Then \( a_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) \Rightarrow \sum_{i=1}^{n} a_i = \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0) \)

- define \( \Phi \) so that \( \Phi(D_0) = 0 \) and \( \Phi(D_i) \geq 0 \) for all \( i \).

Then \( \sum_{i=1}^{n} c_i = \sum_{i=1}^{n} a_i - \Phi(D_n) \leq \sum_{i=1}^{n} a_i \)

Thus the total amortized cost is an *upper bound* on the total actual cost.
Cache-Oblivious Buffer Heap: I/O Complexity

**Lemma:** A BH supports *Delete, Delete-Min, and Decrease-Key* operations in \(O((1 / B) \log_2(N / B))\) amortized I/Os each assuming a tall cache.

**Proof:** We will use the *Potential Method*.

- Each *Decrease-Key* inserted into \(U_0\) will be treated as a pair of operations: \(<\text{Decrease-Key, Dummy}>\).

- Each component of the actual cost will have a \(\Theta(1 / B)\) factor associated with it which we will drop for simplicity.

- For \(0 \leq i \leq r - 1\), let \(u_i = |U_i|\), and \(b_i = |B_i|\).

- If \(H\) is the current state of BH, we define *potential* of \(H\) as:

\[
\Phi(H) = 4ru_0 + \sum_{i=1}^{r-1} (3r - i)u_i + \sum_{i=0}^{r-1} (i + 1)b_i
\]
Cache-Oblivious Buffer Heap: I/O Complexity

Amortized Cost of Reconstruction:

- Starts with $\sum_{i=0}^{r-1} u_i \geq \sum_{i=0}^{r-1} b_i$. Thus cost of steps 1 to 4 is $\leq 2r \sum_{i=0}^{r-1} u_i$.

- Total potential drop is $\geq 2r \sum_{i=0}^{r-1} u_i$.

- Amortized cost of reconstruction is $\leq 2r \sum_{i=0}^{r-1} u_i - 2r \sum_{i=0}^{r-1} u_i = 0$.

\[ \Phi(H) = 4ru_0 + \sum_{i=1}^{r-1} (3r - i)u_i + \sum_{i=0}^{r-1} (i + 1) b_i \]

After each operation check whether $\sum u_i \geq \sum b_i$, and if so,

**Step 1:** Sort the elements in $S_B$ by element id and level number.

**Step 2:** Sort the updates in $S_U$ by element id and time-stamp.

**Step 3:** Scan $S_B$ and $S_U$ simultaneously, and apply the updates in $S_U$ on the elements of $S_B$.

**Step 4:** Reconstruct the data structure by filling the shallowest levels with the current elements in $S_B$, and emptying $S_U$.

I/O cost of sorting $X$ elements = $O\left(\frac{X}{B} \log_2 X\right)$; $r = O(\log_2 N)$
**Cache-Oblivious Buffer Heap: I/O Complexity**

**Amortized Cost of **\textit{Reconstruction}**: 
\[ \Phi(H) = 4ru_0 + \sum_{i=1}^{r-1} (3r - i)u_i + \sum_{i=0}^{r-1} (i + 1)b_i \]

- Starts with \( \sum_{i=0}^{r-1} u_i \geq \sum_{i=0}^{r-1} b_i \). Thus cost of steps 1 to 4 is \( \leq 2r \sum_{i=0}^{r-1} u_i \).
- Total potential drop is \( \geq 2r \sum_{i=0}^{r-1} u_i \).
- Amortized cost of reconstruction is \( \leq 2r \sum_{i=0}^{r-1} u_i - 2r \sum_{i=0}^{r-1} u_i = 0 \).

**Amortized Cost of **\textit{Delete}**: 
- Actual cost = 1.
- Increase in potential = 4r.
- Amortized cost = 1 + 4r = \( O(\log_2 N) \).

\textit{Delete(x)}: 
Insert the operation into \( U_0 \) augmented with current time-stamp.

\[ \text{[Stack Push/Pop requires } O(1 / B) \text{ amortized I/Os each]} \]
Amortized Cost of **Reconstruction**:

\[ \Phi(H) = 4ru_0 + \sum_{i=1}^{r-1} (3r - i)u_i + \sum_{i=0}^{r-1} (i + 1)b_i \]

- Starts with \( \sum_{i=0}^{r-1} u_i \geq \sum_{i=0}^{r-1} b_i \). Thus cost of steps 1 to 4 is \( \leq 2r \sum_{i=0}^{r-1} u_i \).
- Total potential drop is \( \geq 2r \sum_{i=0}^{r-1} u_i \).
- Amortized cost of reconstruction is \( \leq 2r \sum_{i=0}^{r-1} u_i - 2r \sum_{i=0}^{r-1} u_i = 0 \).

Amortized Cost of **Delete**:

- Actual cost = 1.
- Increase in potential = 4.
- Amortized cost = 1 + 4 = \( O(\log_2 N) \).

**Decrease-Key**

- Actual cost = 2 \times 1.
- Increase in potential = 2 \times 4r.
- Amortized cost = 2 + 8r = \( O(\log_2 N) \).
Cache-Oblivious Buffer Heap: I/O Complexity

Amortized Cost of **Delete-Min**

**Actual Cost:**

- Cost of sorting $U_0$ is $\leq ru_0$.
- Cost of examining the updates in $U_0, U_1, ..., U_k$ is $\sum_{i=0}^{k-1} (k - i + 1) u_i$.
- Cost of examining the elements in $B_0, B_1, ..., B_{k-1}$ is $\sum_{i=0}^{k-1} b_i$.
  
  - Let $b_k$ and $b'_k$ be the number of elements in $B_k$ before and after the updates, respectively.
  - Then the total cost of examining after updates is $\max (b_k, b'_k)$.
  - Cost of accessing $A_s$ is $\leq r$.

Thus actual cost, $c \leq ru_0 + \sum_{i=0}^{k-1} (k - i + 1) u_i + \sum_{i=0}^{k-1} b_i + \max (b_k, b'_k) + r$
Cache-Oblivious Buffer Heap: I/O Complexity

Amortized Cost of **Delete-Min**: 

**Actual Cost:**

- Cost of sorting $U_0$ is $\leq ru_0$.

- Cost of examining the updates in $U_0$, $U_1$, ..., $U_k$.

- Cost of examining the elements in $B_0$, $B_1$, ..., $B_{k-1}$.

Let $b_k$ and $b'_k$ be the number of elements in the buffers before and after the updates, respectively.

Then the total cost of examining $B_k$ during updates and selection after updates is $\max(b_k, b'_k)$.

Cost of accessing $A_s$ is $\leq r$.

Thus actual cost, $c \leq ru_0 + \sum_{i=0}^{k} (k - i + 1)u_i + \sum_{i=0}^{k-1} b_i + \max(b_k, b'_k) + r$
Cache-Oblivious Buffer Heap: I/O Complexity

Amortized Cost of **Delete-Min**: 

\[ \Phi(H) = 4ru_0 + \sum_{i=0}^{r-1} (3r - i)u_i + \sum_{i=0}^{r-1} (i + 1)b_i \]

Potential Drop:

- Potential drop due to changes in update buffers is
  \[ \geq 4ru_0 + \sum_{i=1}^{k} (3r - i)u_i - (3r - k - 1)\sum_{i=0}^{k} u_i \]

- Potential drop due to changes in element buffers is
  \[ \geq \sum_{i=0}^{k-1} b_i + \max(b_k, b'_k) \]

Thus, total drop in potential is

\[ \geq ru_0 + \sum_{i=0}^{k} (k - i + 1)u_i + \sum_{i=0}^{k-1} b_i + \max(b_k, b'_k) \]

Therefore, amortized cost of **Delete-Min** is

\[ \hat{c} = c + \left( \Phi(H_{after}) - \Phi(H_{before}) \right) \leq r = O(\log_2 N) \]
Cache-Oblivious Buffer Heap: I/O Complexity

We assume $N \gg M = \Omega(B^{1+\varepsilon})$ for some $\varepsilon > 0$

$$\Rightarrow \log_2 N = O\left(\log_2 \frac{N}{B}\right)$$

Therefore, under the tall cache assumption,

Amortized I/O cost of each operation

$(Delete, Delete-Min$ and $Decrease-Key$) is $= O\left(\frac{1}{B} \log_2 \frac{N}{B}\right)$
Removing the “Tall Cache” Assumption

– Restrict the size of each update buffer: $|U_i| \leq 2^i$
– Now all buffers ($U_i$ and $B_i$) of the first $\log_2 B$ levels occupy only $O(B)$ blocks.
– No external I/O is required to access the first $\log_2 B$ levels.
– Thus amortized I/O cost of each operation is

$$= O\left( \frac{1}{B} (\log_2 N - \log_2 B) \right) = O\left( \frac{1}{B} \log_2 \frac{N}{B} \right)$$