Lecture 24: Cache-oblivious algorithms II

- Search
  - binary
  - B-ary
  - cache-oblivious

- Sorting
  - mergesorts
  - cache-oblivious

**Why LRU block replacement strategy?**

\[
LRU_M \leq 2 \cdot OPT_{M/2} \quad \text{[Sleater and Tarjan 1985]}
\]

Proof.

- partition block access sequence into maximal phases of \( M/B \) distinct blocks
- LRU spends \( \leq M/B \) memory transfers/phase
- OPT must spend \( \geq M/2 \) memory transfers per phase: at best, starts phase with entire \( M/2 \) cache with needed items. But there are \( M/B \) blocks during phase. So \( \leq \) half free

**Search**

Preprocess \( n \) elements in comparison model to support predecessor search for \( x \).

**B-trees**

They support predecessor (and insert and delete) in \( O(\log_{B+1} N) \) memory transfers.

- each node occupies \( \Theta(1) \) blocks
- height= \( \Theta(\log_B N) \)
- need to know \( B \)
Binary search

Approximately, every iteration visits a different block until we are in \( x \)'s block. Thus, 
\[ MT(N) = \Theta(\log N - \log B) = \Theta(\log(N/B)). \] SLOW

van Emde Boas layout

[Prokop 1999]

- store \( N \) elements in complete BST
- carve BST at middle level of edges
- recursively layout the pieces and concatenate
- like block matrix multiplication, order of pieces doesn’t matter; just need each piece to be stored consecutively

Analysis of BST search in \( vEB \) layout:

- consider recursive level of refinement at which structure has \( \leq B \) nodes
- the height of the vEB tree is between \( \frac{1}{2} \log B \) and \( \log B \) \( \implies \) size is between \( \sqrt{B} \) and \( B \)
  \( \implies \) any root-to-node path (search path) visits \( \leq \frac{\log N}{\frac{1}{2} \log B} = 2 \log_B N \) trees that have size \( \leq B \)
- each tree of size \( \leq B \) occupies \( \leq 2 \) memory blocks

\[ \implies \leq 4 \log_B N = O(\log_B N) \text{ memory transfers} \]
• this generalizes to heights that are not powers of 2, B-trees of constant branching factor and dynamic B-trees: $O(\log_B N)$ insert/delete. [Bender, Demaine, Farach-Colton 2000]

Sorting

B-trees

$N$ inserts into (cache-oblivious) B-tree $\implies MT(N) = \Theta(N \log_B N)$ NOT OPTIMAL. By contrast, BST sort is optimal $O(N \log N)$

Binary mergesort

• binary mergesort is cache-oblivious.

• the merge is 3 parallel scans
  $\implies MT(N) = 2MT(N/2) + O(N/B + 1)$
  $MT(M) = O(M/B)$

• the recursion tree has $\log(N/M)$ levels, and each level contributes $O(N/B)$
  $\implies MT(N) = \frac{N}{B} \log \frac{N}{M}$. $\leftarrow \frac{B}{\log B}$ faster than the B-tree version discussed earlier!

$M/B$-way mergesort

• split array into $M/B$ equal subarrays

• recursively sort each

• merge via $M/B$ parallel scans (keeping one “current” block per list)

  $\implies MT(N) = \frac{M}{B} MT\left(\frac{N}{M/B}\right) + O(N/B + 1)$
  $MT(M) = O(M/B)$

  $\implies$ height becomes $\log_{M/B} \frac{N}{M} + 1$

  $= \log_{M/B} \frac{N}{B} \cdot \frac{B}{M} + 1$

  $= \log_{M/B} \frac{N}{B} - \log_{M/B} \frac{M}{B} + 1$

  $= \log_{M/B} \frac{N}{B}$
\[ MT(N) = O\left(\frac{N}{B} \log \frac{M}{B} \frac{N}{B}\right) \]

This is asymptotically optimal, in the comparison model.

**Cache-oblivious Sorting**

This requires the tall-cache assumption: \( M = \Omega(B^{1+\epsilon}) \) for some fixed \( \epsilon > 0 \), e.g., \( M = \Omega(B^2) \) or \( M/B = \Omega(B) \).

Then, \( \approx N^e \)-way mergesort with recursive ("funnel") merge works.

**Priority Queues**

- \( O(\frac{1}{B} \log \frac{M}{B} \frac{N}{B}) \) per insert or delete-min
- generalizes sorting
- external memory and cache-oblivious
- see 6.851