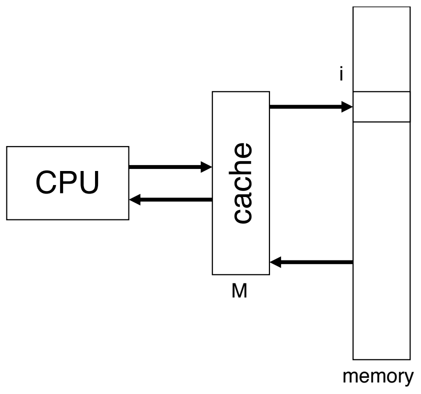
COMS E6998-9: Algorithms for Massive Data (Spring'19) Apr 4, 2019

Lecture 20: Large-Scale Models

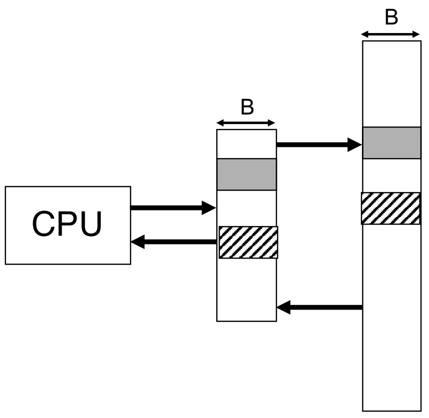
Instructor: *Alex Andoni* Scribes: *Biqing Qiu, Fan Wu*

1 Introduction

In this class, we are looking at machines with a cache. The CPU use cache to reduce the average cost to access data from the main memory. The cache retrieves and stores information from the main memory. It is much cheaper for the CPU to access the cache than the main memory. Given input of size n, and a cache of size M, we have that $M \ll n$.



Cache Line: A cache line is B consecutive addresses in the main memory. Each cache line is fetched as a whole from main memory to the cache. One cache thus has $\frac{M}{B}$ cache lines. We have that $B \ll M \ll n$, ie. $M < n^{2/3}, B < \sqrt{M}$.



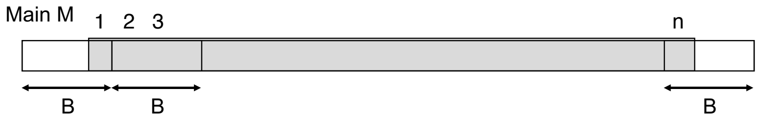
2 I/O Model

The I/O Model only counts the number of access to main memory as the cost of the algorithm.

Example 1. *Linear Scan.* Find x in an unsorted array A. I/O Cost $= \lceil \frac{n}{B} \rceil + 1$.

• Ceiling function is needed as n may not be divisible by B.

• $+1$ in case of misalignment of start of A and beginning of a cache line.



As the figure shows, in the worst case, the input starts from the end of an cache line and ends at the beginning of an cache line. So scan the whole input needs accessing $\lceil \frac{n}{B} \rceil + 1$ cache lines.

Example 2. *Random Access into A.* Let A be a hash table $h : [n] \to [n]$, with n hash table lookups $h(1), h(2), \ldots, h(n)$. I/O Cost: at each moment, $\leq \frac{M}{B}$ cache lines are already in the cache. Pr[cache line containing $h(i)$ is in cache] $\leq \frac{M/B}{n/B} = \frac{M}{n} \ll \frac{1}{2}$. $\mathbb{E}$[I/O cost] $\geq \frac{1}{2}n = \Omega(n)$.

Example 3. *Binary Search in Sorted Array A.* Classical runtime is $\Theta(\log n)$. Recall Binary Search:

Algorithm 1 Binary Search

```
function BS(l, r, y)
    if $l > r$ then return nil
    $m = \lfloor \frac{l+r}{2} \rfloor$
    if $A[m] = y$ then return found at m
    if $A[m] < y$ then return BS($m+1$,$r$,$y$)
    else return BS($l$, $m-1$, y)
```

Claim 1. *Runtime of binary search with cache is* $O(\log(\frac{n}{B}))$.

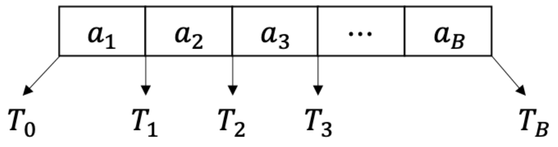
Proof. As long as $r - l \gg B$, each new access is in a new cache line. Start with $r - l = n$. Each recursive call halves distance between r and l. Number of recursive calls until $(r - l) < 3B$ is $\Theta(\log \frac{n}{3B})$. Once $(r - l) < 3B$, our range is in 4 cache lines. Hence, as long as $M > 4B$, runtime $= \Theta(\log \frac{n}{B} + 4) = \Theta(\log \frac{n}{B}) = \log n - \log B$. □

Remark. $B < \sqrt{n} \implies \log \frac{n}{B} \geq \frac{1}{2} \log n$. We have made very little progress!

3 B-Trees

We want to design a data structure on A to support faster queries.

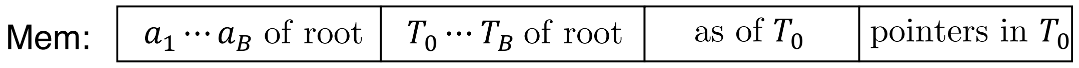
Definition. A B-tree is a tree such that each node is made of a block of B elements, $a_1, a_2, \ldots, a_B$, and each element a_i points to two subtrees T_{i-1} and T_i such that: 1) $a_1 \leq a_2 \leq \cdots \leq a_B$ and 2) $(\forall \in T_0) \leq a_1 \leq (\forall \in T_1) \leq a_2 \leq \cdots \leq a_B \leq (\forall \in T_B)$.



Remark. Binary trees are 1-trees.

Observation 1. Height of a balanced B-tree $h = \log_B(n)$.

Observation 2. Any node can be stored in 3 cache lines, if B-tree is stored in memory as a tree, ie., such that memory holds: $a_1, \ldots, a_B$ of root, $T_0, \ldots, T_B$ of root, a's of T_0, pointers in T_0, etc. This implies I/O cost $\leq 3h = O(\log_B(n)) = O(\frac{\log n}{\log B})$.



Since $B < \sqrt{n}$, runtime= $O(\frac{\log n}{\log B}) \sim$ linear time.

4 Other Algorithms

4.1 Sorting

We can achieve $O(\frac{n}{B} \log_{M/B}(\frac{n}{B}))$ runtime. Note that this depends on cache size M. Idea of algorithm: $\frac{M}{B}$-way merge sort. Store arrays $A_1, A_2, \ldots, A_k$ in the cache to be merged, which takes up about $\frac{M}{2B}$ cache lines, leaving some cache lines to store other algorithm variables.

4.2 Cache Oblivious Algorithms

Cache oblivious algorithms are I/O models that do not depend on M or B.

Motivations.

- Nice to not depend on B
- Able to optimize over multi-layer architectures.

Question. *Data structure with $O(lg\frac{n}{B})$ lookup?*

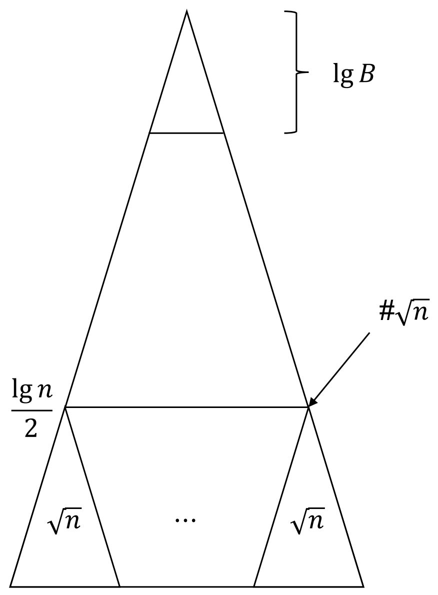
Heap? No.

Remember: heap is stored as a binary tree

First lgB levels are in the same cache line.

I/O cost $\geq \frac{lgn}{2} = lgn - lgB$

This is because from the height $\frac{lgn}{2}$, each level has at least $\sqrt{n}$ elements. $M < \sqrt{n}$, therefore start from this level, each access to the next level will enter into a new cache line.


$\lg B$
$\#\sqrt{n}$
$\frac{\lg n}{2}$
$\sqrt{n}$
$\cdots$
$\sqrt{n}$