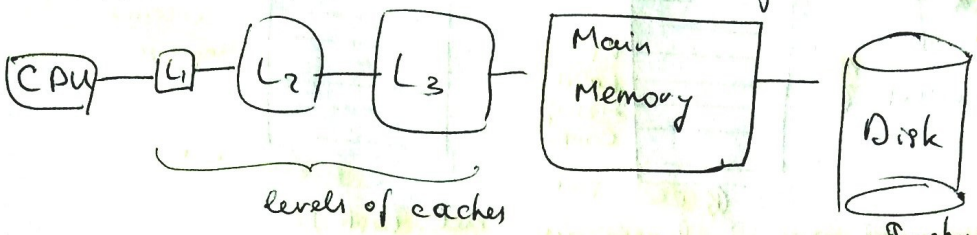
6.046
Lecture 24

Cache-oblivious algorithms

Modern Memory hierarchy

Random access model : access in memory at $O(1)$. Different Model.



1) slower [redacted] (higher latency)
2) and bigger space
3) bandwidth should be same

the larger space, the more time to access speed of light as fundamental limit.

Cost to Access

$$= \text{latency} + \frac{\text{amount of data}}{\text{bandwidth}}$$

latency: limited by speed of light

bandwidth: rate at which can get data out

Idea: as latency goes up, increase the amount of data, then the amortized cost to access an element goes down (fixed bandwidth)

$$\text{Amortized cost to access one element} = \frac{\text{latency}}{\text{amount}} + \frac{1}{\text{Bandwidth}}$$

latency amortized by amount getting it from disk into memory, slow to faster)

Spatial locality

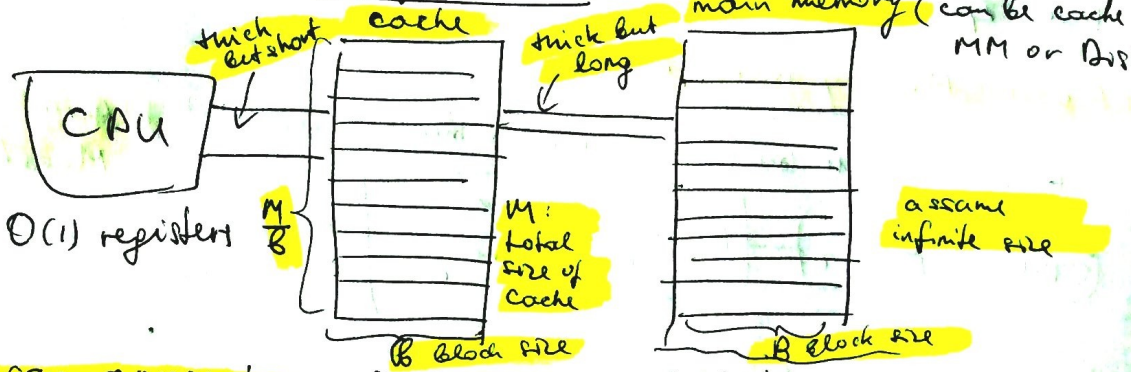
Want algorithms to use all elements on a block (after

Temporal locality

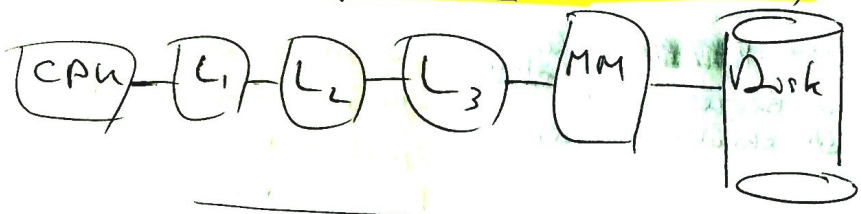
Want ideally to reuse Blocks (if algorithm is above linear will use elements in input more than once)

(cache-aware model)

Two-level memory model



can represent any 2 levels in (incl. Disk)



slower (higher latency)
more space
bandwidth same

Assume CPU can access cache instantaneously (free)

If CPU needs data, check cache — if there get it free,
else get entire block from main memory,
potentially need to remove data from cache and write it back to main memory to free up cache.

- accesses to cache free (but still measure computation time)
- count block memory transfers between cache and main memory

memory transfer: read or write a block from/into main memory

$MT(N)$ $(= MT_{B,M}(N))$

- N: size of problem (the only variable that can change)
- B: block size
- M: total size cache } fixed parameters

→ cannot measure as a function of block size or total cache size (fixed parameters)

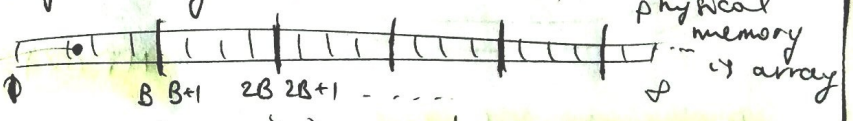
Lecture 24

Cache aware algorithm: B trees (know fixed cache parameters)

Cache - oblivious motivation:
1) want to design algorithms that perform well no matter the values of B and M
2) most "just memory" assuming algorithms are already cache - oblivious.
some will do well in the two - level memory model and some will not.

Cache - oblivious algorithm:

- algorithm does not know B & M
- accessing element automatically fetch block containing it
  & evicts block that will be used furthest in future
- Blocking of memory (a linear array)

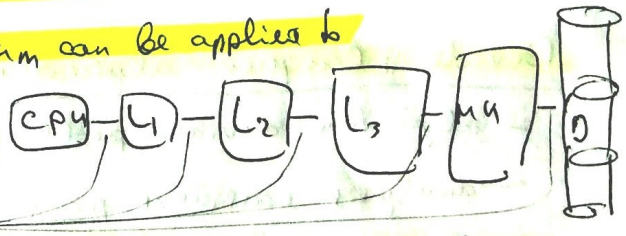


access ⇒ get entire block rounded down to prev. multiple of B, and rounded up to the next multiple of B

blocks of size B

Fact:
[if efficient on 2 levels & cache oblivious ⇒ efficient on ℓ levels]

Efficient cache oblivious algorithm can be applied to any adj pair of levels in (i.e. any memory transfer level):

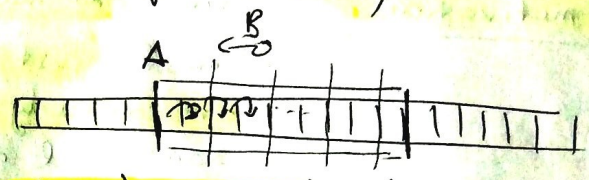


Because cache - oblivious algs do not tune to particular values of B & M.

Basic algorithms

Scanning (A, N) // visiting items in array in order e.g. sum array

for $i \leftarrow 1$ **to** N
do visit $A[i]$



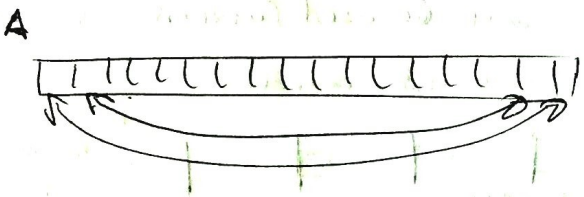
$MT(N) = O\left(\frac{N}{B} + 1\right)$

N could be $< B$, more precisely should be 2 since first and last block can be non-full

O(1) parallel scans

Reverse (A, N)

for $i \leftarrow 1$ **to** $\lfloor N/2 \rfloor$
do exchange $A[i] \leftrightarrow A[N-i+1]$



Assuming $\frac{M}{B} \geq 2$ (cache can store at least 2 blocks)

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

Binary Search (x)

if x equals to compared values early

$O(1)$

hope: $\log_B N$ without knowing B

$MT(N) = O\left(\lg\left(\frac{N}{B} + 1\right)\right) = O(\lg N - \lg B + 1)$ } bad

once search narrows down to 1 block

Divide & Conquer algorithms (incl. Binary search)

- algorithm divides problem down to $O(1)$ size
- analyses considers point at which:
  - problem fits in cache ($\leq M$)
  - problem fits in $O(1)$ blocks

change base case of recurrence

Order Statistics (median)

① conceptually partition array into $\frac{N}{5}$ 5-tuples

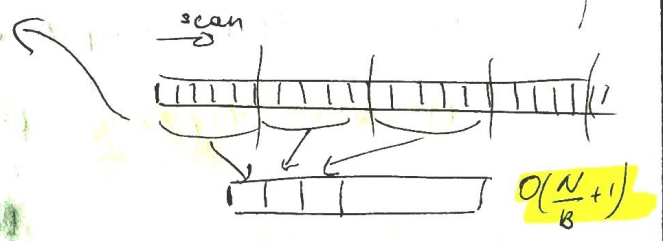
② compute median of each tuple

③ recursively compute median of these medians x } $MT(\frac{N}{5})$

④ partition around x

⑤ recurse on one side

$MT(\frac{3}{4}N)$



Analysis

$$MT(N) = MT\left(\frac{N}{5}\right) + MT\left(\frac{3}{4}N\right) + O\left(\frac{N}{B}+1\right)$$

$MT(1) \Leftarrow$ bad

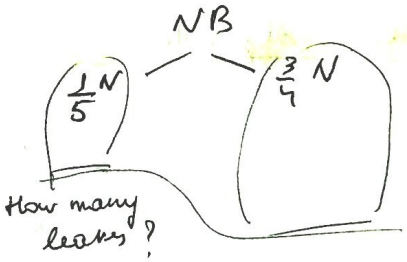
leaves: $L(N) = L\left(\frac{N}{5}\right) + L\left(\frac{3}{4}N\right)$

$L(1) = 1$

$N^{\alpha} = \left(\frac{N}{5}\right)^{\alpha} + \left(\frac{3}{4}N\right)^{\alpha}$

$1 = \left(\frac{1}{5}\right)^{\alpha} + \left(\frac{3}{4}\right)^{\alpha} \quad \alpha \approx 0.8398$

$\Rightarrow L(N) = \left(\frac{N}{5}\right)^{\alpha} + \left(\frac{3}{4}N\right)^{\alpha} = \omega\left(\frac{N}{B}\right)$ bad



$MT(B) = O(1)$

leaves $= \left(\frac{N}{B}\right)^{\alpha} = O\left(\frac{N}{B}\right)$

cost roughly geometric down the tree

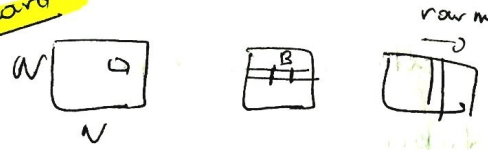
⇒ root dominates.

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

↑ can't do better because need to read all N.

Matrix Multiplication

standard



$C = A \cdot B$

$O(\frac{N}{B})$ mem. transfers to compute c_{ij}

$\Rightarrow MT(N) = O(\frac{N^3}{B})$

Memory layout

assume C stored in row-major

A row-major

B col-major

col j in B moves for each value in A's row segment

But want closer to $O(\frac{N^2}{B})$

under memory layout assumption we have
good spatial locality
bad temporal locality

assume one row of A in cache
then for each row in C
must transfer a row in A and entire B

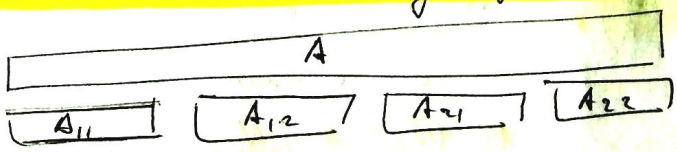
Block algorithm

$$\begin{matrix} \frac{n}{2} \\ n \\ \frac{n}{2} \end{matrix}\left(\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right) = \begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array} \cdot \begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array} =$$

$\frac{n}{2} \quad n \quad \frac{n}{2}$

$$= \begin{array}{c|c} A_{11}B_{11} & A_{11}B_{12} \\ \hline A_{21}B_{11} & A_{21}B_{12} \end{array} + \begin{array}{c|c} A_{12}B_{21} & A_{12}B_{22} \\ \hline A_{22}B_{21} & A_{22}B_{22} \end{array}$$

Store matrices recursively by block in memory.



same layout for C and B

recursively in memory in one array of size N

← powerful idea in cache-oblivious algorithms

Lecture 24

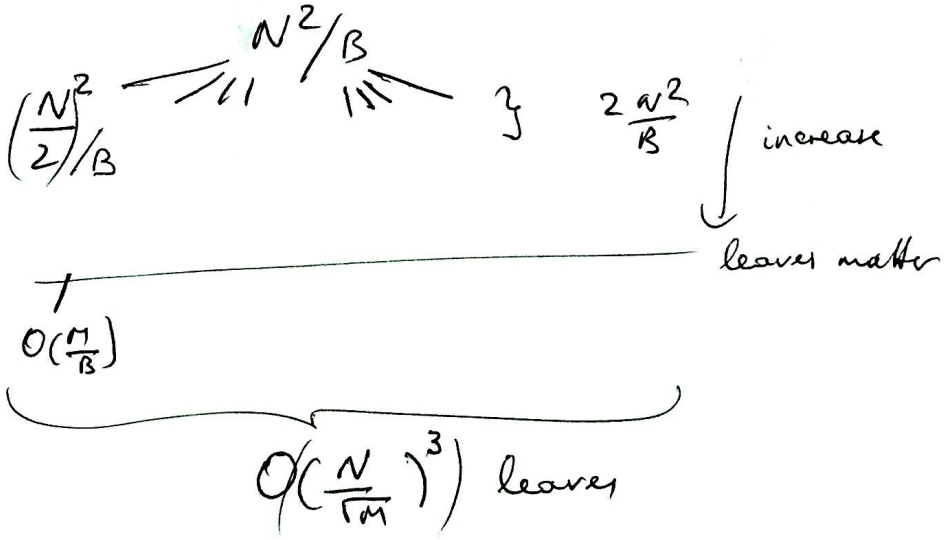
$$MT(N) = 8\,MT\left(\frac{N}{2}\right) + O\left(\frac{N^2}{B}\right)$$

relies on recursive memory layout

↖ scan same consecutive order for A, B, C

$MT(B) = O(1)$ ← base

$MT(c\sqrt{M}) = O\left(\frac{M}{B}\right)$



$$\Rightarrow MT(N) = O\left(\frac{N^3}{M^{3/2}} \cdot \frac{M}{B}\right) = O\left(\frac{N^3}{B\sqrt{M}}\right)$$

optimal! in the two-level memory model.