6.046	Lecture 23	May 12, 2015

TODAY: Cache-oblivious algorithms I (of 2)

- memory hierarchy
- external memory vs. cache oblivious models
- scanning
- divide & conquer
  - median finding
  - matrix multiplication
- LRU block replacement

So far we've viewed all word operations & all memory accesses as equal cost...

<u>Modern memory hierarchy</u>:

CPU (registers)	L1	L2	L3	L4 (Haswell)	Main Memory	Flash	Disk
~ 10K	100K	MBs	100MB		GBs–TB	100GB–TBs	TBs–PB
~ ns	10ns	10ns	100ns		μs	10–100 μs	10ms

⟶ bigger but slower <u>latency</u>:
distance travel & physical seek on disk

- bandwidth usually matched (RAID etc.)
- <u>blocking</u> to mitigate latency:
  - when fetching a word of data, get entire block containing it
  - <u>idea</u>: amortize latency over whole block
  - ⇒ amortized cost per word

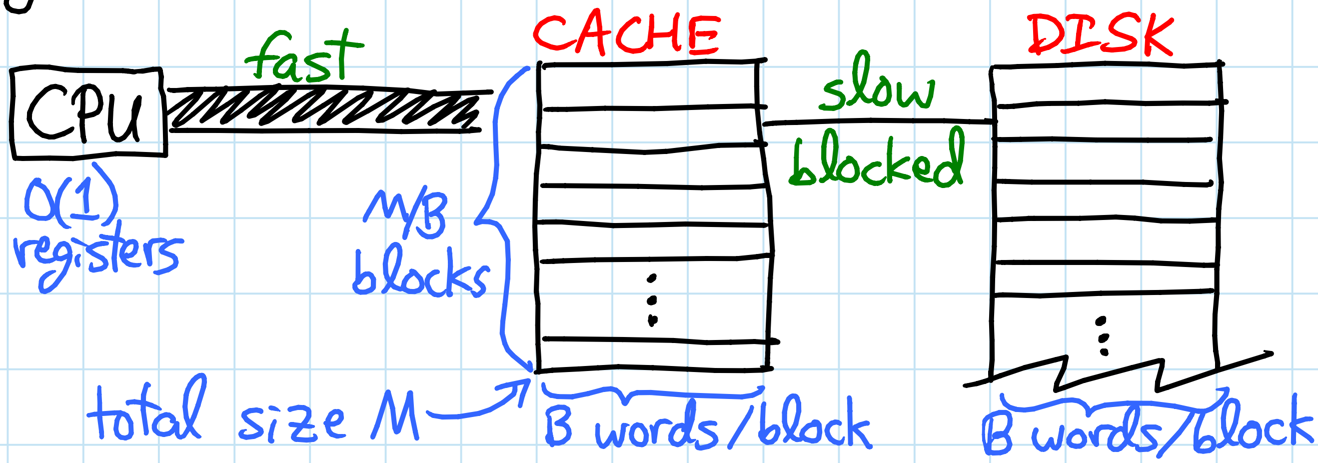
$$= \frac{\text{latency}}{\text{block size}} + \frac{1}{\text{bandwidth}}$$

set roughly equal via block size

  - to work, we need algorithms to use all elements in a block (<u>spatial locality</u>) & re-use blocks in cache (<u>temporal locality</u>)

External-memory model: [Aggarwal & Vitter 1988]

- just 2 levels:



- cache accesses free (just count computation)

$\Rightarrow$ count memory transfers between cache $\leftrightarrow$ disk
= # blocks read from/written to disk

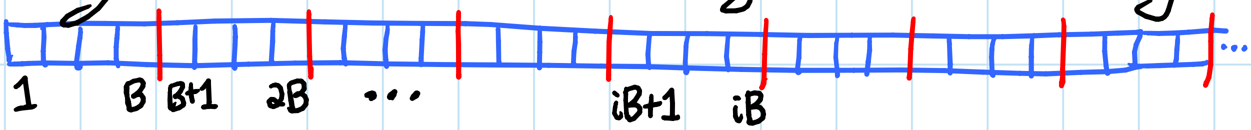
- algorithm explicitly reads & writes blocks

Cache-oblivious model:

[Frigo, Leiserson, Prokop, Ramachandran 1999]

→ FFTW → L in CLRS → M.Eng.

- algorithm doesn't know B or M (!)
- accessing a word in memory (blocked array:



automatically fetches entire block containing it & evicts (writes) least recently used (LRU) block from cache if full

(more like real caches)

⇒ every algorithm is a cache-oblivious algorithm

- new measurement & objective:

<u>minimize # memory transfers</u>

Why?

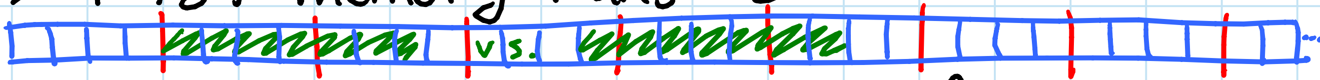
- cooler
- often possible
- "cleaner" algorithms & implementations
- automatic "tuning"
- optimize all levels of memory hierarchy (each with their own B & M)

Scanning:

Single scan: e.g. for i in range(N):

sum += A[i]

- assume array A stored contiguously in memory
- external memory: align A with block start
  - $\Rightarrow \lceil N/B \rceil$ memory transfers



- cache oblivious: can't control alignment
  - still $\le \lceil N/B \rceil + 1 = N/B + O(1)$

O(1) parallel scans: (assuming $M/B = \Omega(1)$)

- e.g. reversing A[0 : n]: [Bentley]

  for i in range($\lfloor N/2 \rfloor$):

  swap $A[i] \leftrightarrow A[N-i-1]$

- keep one block $\ni A[i]$ & one $\ni A[N-i-1]$

$\Rightarrow O(N/B + 1)$ memory transfers (assuming $M/B \geq 2$)

Divide & conquer approach: → cache oblivious

- algorithm divides problem down to $O(1)$ size
- analysis considers recursion at which
  - problem fits in cache i.e. $\le M$
  - problem fits in $O(1)$ blocks i.e. $O(B)$
- TODAY: one example of each

Median finding / order statistics:

- recall $O(N)$-time deterministic algorithm: [L2]
  1. view array as partitioned into columns of 5
     like blocks, but $O(1)$ size
  2. sort each column → median
  3. recursively find median of column medians
  4. partition array by x ($\le x$, $> x$)
  5. recurse on one side
- memory transfer analysis: $MT(N)$
  1. free
  2. scan $\Rightarrow O(N/B + 1)$
  3. $MT(N/5)$ ~ if we coalesce $N/5$ medians into a consecutive array (via 2 parallel scans)
  4. 3 parallel scans $\Rightarrow O(N/B + 1)$
  5. $MT(\frac{7}{10} N)$

$$\Rightarrow MT(N) = MT(N/5) + MT(\tfrac{7}{10} N) + O(N/B + 1)$$

- usual base case: $MT(O(1)) = O(1)$
  - $\Rightarrow MT(N) \geq$ # leaves $L(N)$ in recursion
  - $L(N) = L(N/5) + L(\frac{7}{10}N)$

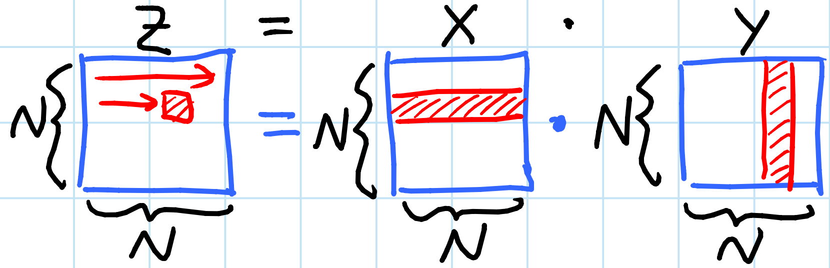
    $N^\alpha = (N/5)^\alpha + (\frac{7}{10}N)^\alpha$

    $1 = (1/5)^\alpha + (7/10)^\alpha$
  - $\Rightarrow \alpha \approx 0.83978$
  - $\Rightarrow MT(N) \geq N^{0.8} = \omega(N/B)$ if $B = \omega(B^{0.2})$

- stronger base case: $MT(O(B)) = O(1)$
  - $\Rightarrow$ # leaves $L(N) = (N/B)^\alpha = o(N/B)$
  - cost at each level of recursion tree decreases geometrically down

    (a little tricky to prove — better to use substitution method like L2)
  - $\Rightarrow$ dominated by root cost $O(N/B + 1)$
  - $\Rightarrow MT(N) = O(N/B + 1)$

Matrix multiplication:



Standard algorithm:

- ideal memory layout:
  - X stored in row-major order
  - Y stored in column-major order
  - Z stored in either, say row-major
- each z_{ij} costs $\Theta(N/B + 1)$
  - upper bound: 2 parallel scans
  - X row i gets re-used in all z_{i*} (assuming $M/B \geq 3$)
  - but Y column j gets read for every z_{ij} (assuming $M < N^2 = \text{size}(Y)$)
- $MT(N) = \Theta(N^3/B + N^2)$ — NOT OPTIMAL

Block algorithm:

$$\begin{bmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{bmatrix} = \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \cdot \begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{bmatrix} = \begin{bmatrix} X_{11}Y_{11} + X_{12}Y_{21} & X_{11}Y_{12} + X_{12}Y_{22} \\ X_{21}Y_{11} + X_{22}Y_{21} & X_{21}Y_{12} + X_{22}Y_{22} \end{bmatrix}$$

- store matrices in recursive block layout:

$$\boxed{X} = \boxed{X_{11}}\,\boxed{X_{12}}\,\boxed{X_{21}}\,\boxed{X_{22}}$$

recursive layouts

- order of blocks doesn't matter
- key: each block is stored consecutively

$$\Rightarrow MT(N) = \underbrace{8 \cdot MT(N/2)}_{\text{recursion}} + \underbrace{O(N^2/B + 1)}_{\text{addition is 3 parallel scans}}$$

- base cases: $MT(O(1)) = O(1)$
  $MT(O(B)) = O(1)$
  $MT(\underbrace{\sqrt{M/3}}) = O(M/B)$
  $\Rightarrow$ 3 $\sqrt{M/3} \times \sqrt{M/3}$ fit in cache
- recursion tree:

N^2/B — N^2/B

8

$\frac{1}{4} N^2/B$ — $2N^2/B$

$\vdots$

$\underbrace{O(M/B) \; O(M/B) \; \cdots}$ — $O\left(\frac{N^3}{M^{3/2}} \cdot \frac{M}{B}\right) = O\left(\frac{N^3}{B\sqrt{M}}\right)$

\# leaves $= 8^{\lg(N/\sqrt{M})} = O((N/\sqrt{M})^3)$

- geometrically increasing cost down tree (like Master Theorem)

$\Rightarrow$ dominated by leaf level

$\Rightarrow MT(N) = O\left(\frac{N^3}{B\sqrt{M}}\right)$ $\Leftarrow$ ASYMPTOTICALLY OPTIMAL

- generalizes to non-powers of 2 & non-square matrices
- similar algorithms & analyses for
  - Strassen's algorithm
  - FFT

Why LRU block replacement strategy?

$LRU_M \le 2 \cdot OPT_{M/2}$ RESOURCE AUGMENTATION

[Sleator & Tarjan 1985] (changing M)

Proof:

- partition block access sequence into maximal phases of M/B distinct blocks
- LRU spends $\le M/B$ memory transfers/phase
- OPT must spend $\ge \frac{M}{2}/B$ 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 $\le$ half free

ONLINE ALGORITHMS — comparing regular "online" algorithm (can't see the future) against offline/prescient optimal algorithm

- changing M by factor of 2 doesn't affect bounds like $O\left(\frac{N^2}{B\sqrt{M}}\right)$