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...

Modern memory hierarchy:

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

  ➞ bigger but slower latency:
  - distance travel & physical seek on disk
  - bandwidth usually matched (RAID etc.)
  - blocking to mitigate latency:
    - when fetching a word of data, get entire block containing it
    - idea: amortize latency over whole block

  ➞ amortized cost per word
  \[
  \text{latency} + \frac{1}{\text{bandwidth}}
  \]
  - set roughly equal via block size

  ➞ to work, we need algorithms to use all elements in a block (spatial locality) & re-use blocks in cache (temporal locality)
External-memory model: [Aggarwal & Vitter 1988]
- just 2 levels:

- cache accesses free (just count computation)
- count memory transfers between cache ↔ disk
  = # blocks read from/written to disk
- algorithm explicitly reads & writes blocks
Cache-oblivious model:

- algorithm doesn't know B or M (!)
- accessing a word in memory (blocked array:
  
  \[ 1 \quad B \quad B+1 \quad 2B \quad \cdots \quad iB+1 \quad iB \]

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

\[ \Rightarrow \text{every algorithm is a cache-oblivious algorithm} \]
- new measurement & objective:
  minimize \# memory transfers

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
  \[ \Rightarrow \lceil \frac{N}{B} \rceil \text{ memory transfers} \]
- Cache oblivious: can't control alignment
  \[ \Rightarrow \frac{N}{B} + 1 = \frac{N}{B} + O(1) \]

**O(1) parallel scans**: (assuming \( \frac{N}{B} = \Omega(1) \))
- E.g. reversing `A[0:n]`:
  - for `i` in `range(\lceil \frac{N}{2} \rceil)`:
    - `swap A[i] \leftrightarrow A[N-i-1]`
- Keep one block in `A[i]` & one in `A[N-i-1]`
  \[ \Rightarrow O(\frac{N}{B} + 1) \text{ memory transfers (assuming } \frac{N}{B} \geq 2) \]
**Divide & conquer approach:** \( \rightarrow \) cache oblivious
- algorithm divides problem down to \( O(1) \) size
- analysis considers recursion at which
  - problem fits in cache \( i.e. \leq M \)
  - 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 \( \rightarrow \) median
  3. recursively find median of column medians
  4. partition array by \( x \) \( (\leq x, > x) \)
  5. recurse on one side
- memory transfer analysis: \( MT(N) \)
  1. free
  2. scan \( \Rightarrow O(N/B + 1) \)
  3. \( MT(N/5) \sim \) if we **coalesce** \( N/5 \) medians
     into a consecutive array
     (via 2 parallel scans)
  4. 3 parallel scans \( \Rightarrow O(N/B + 1) \)
  5. \( MT(\frac{7}{10} N) \)

\( \Rightarrow MT(N) = MT(N/5) + MT(\frac{7}{10} N) + O(N/B + 1) \)
usual base case: \( MT(O(1)) = O(1) \)
\[ \Rightarrow MT(N) \geq \# \text{ leaves } L(N) \text{ in recursion} \]
\[ L(N) = L\left(\frac{N}{5}\right) + L\left(\frac{7}{10} N\right) \]
\[ N^\alpha = \left(\frac{N}{5}\right)^\alpha + \left(\frac{7}{10} N\right)^\alpha \]
\[ 1 = \left(\frac{1}{5}\right)^\alpha + \left(\frac{7}{10}\right)^\alpha \]
\[ \Rightarrow \alpha \approx 0.83978 \]
\[ \Rightarrow MT(N) \geq N^{0.8} = \omega\left(\frac{N}{B}\right) \text{ if } B = \omega(B^{0.2}) \]

stronger base case: \( MT(O(B)) = O(1) \)
\[ \Rightarrow \# \text{ leaves } L(N) = \left(\frac{N}{B}\right)^\alpha = o\left(\frac{N}{B}\right) \]
- cost at each level of recursion tree decreases geometrically down
- (a little tricky to prove — better to use substitution method like L2)
\[ \Rightarrow \text{dominated by root cost } O\left(\frac{N}{B} + 1\right) \]
\[ \Rightarrow MT(N) = O\left(\frac{N}{B} + 1\right) \]
Matrix multiplication:

\[ N \{ \begin{bmatrix} X \end{bmatrix} \cdot N \{ \begin{bmatrix} Y \end{bmatrix} = N \{ \begin{bmatrix} Z \end{bmatrix} \} \]

**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 \( \Theta(NYB + 1) \)

- upper bound: 2 parallel scans

- \( X \) row \( i \) gets re-used in all \( z_{ik} \)
  (assuming \( NYB \geq 3 \))

- 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:**

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

- store matrices in recursive block layout:

- order of blocks doesn't matter

- key: each block is stored consecutively

\[ MT(N) = 8 \cdot MT(N/2) + O(N^2/B + 1) \]

**recursion** **addition** is 3 parallel scans
- base cases: \[ MT(O(1)) = O(1) \]
\[ MT(O(B)) = O(1) \]
\[ MT(O(\sqrt{m/3})) = O(\sqrt{m/B}) \]
\[
\Rightarrow 3 \sqrt{m/3} \times \sqrt{m/3} \text{ fit in cache}
\]
- recursion tree:

\[
\begin{array}{c}
N^2/B \\
8 \\
\frac{1}{4} N^2/B \\
\vdots \\
O(\sqrt{m/B}) \quad O(\sqrt{m/B}) \quad \cdots \\
\end{array}
\]

\[
\# \text{leaves} = 8^{\log_2 N/(2m^2)} = O\left(\frac{N}{\sqrt{m}}\right)^3 = O\left(\frac{N^3}{B\sqrt{m}}\right)
\]

- geometrically increasing cost down tree
  (like Master Theorem)

\[
\Rightarrow \text{dominated by leaf level}
\]

\[
\Rightarrow MT(N) = O\left(\frac{N^3}{B\sqrt{m}}\right) \quad \leftarrow \text{ASYMPTOTICALLY OPTIMAL}
\]

- generalizes to non-powers of 2
  & non-square matrices
- similar algorithms & analyses for
  - Strassen's algorithm
  - FFT
Why LRU block replacement strategy?

\[ \text{LRU}_m \leq 2 \cdot \text{OPT}_{m/2} \]

[Sleator & Tarjan 1985]

**RESOURCE AUGMENTATION**
(changing \( M \))

**Proof:**
- partition block access sequence into maximal phases of \( M/B \) distinct blocks
- LRU spends \( \leq M/B \) memory transfers/phase
- OPT must spend \( \geq \frac{M}{2}/B \) 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 \( \leq \) 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 \( O\left(\frac{N^3}{B\sqrt{M}}\right)\)