Cache-friendly programming

- Reality: Cache misses costlier than ever before!
  - A similar issue between main-memory and external memory

- Two major algorithmic strategies:
  - Cache-aware: Tune performance of the code to details of cache
  - Cache-oblivious: Algorithm uses cache well regardless of details

- Usually: Use a cache-oblivious algorithm, fine-tune for cache specifics

- Example: Matrix multiplication

\[
\begin{array}{cc}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{array}
\quad
\begin{array}{cc}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{array}
\]
Cache-friendly array layout

- Recursively defined cache-friendly layout for array multiplication:

$$\text{LAYOUT}(A_{11} \quad A_{12}) = \text{LAYOUT}(A_{21} \quad A_{22}) \quad \text{LAYOUT}(A_{11} \quad A_{12}) \ldots$$
Sorting an array of strings

- We could use Quicksort
  - but it is expensive to compare strings

- A better (standard) option is Radix sort (similar to Bucket sort)
  - “Group” strings according to first letter
    - “grouping” using pointers, of course
  - Sort groups recursively using remaining letters

- Can be viewed as a breadth-first exploration of a tree (actually, a trie):

\[
\begin{array}{c}
A & C & G & T \\
\end{array}
\]

This is slow because we:
- access 1\textsuperscript{st} letter of all strings
- access 2\textsuperscript{nd} letter of all strings
- etc.
Burstsort (Sinha and Zobel, 2004)

- A **burst trie** is a trie in which overflowing buckets “burst” into several smaller buckets

- Each string is processed in a linear, cache-friendly manner
  - this results in a depth-first traversal of the trie

- When the string alphabet is large (256 for ASCII, grouped DNA letters), the trie is short and fat
  - depth-first traversal more cache-friendly than breadth-first traversal
  - upper level nodes remain in cache due to temporal locality

- A problem with pointers:
  - when a bucket bursts, a new level needs to be created
  - all strings need to be accessed, which can lead to cache misses

- Alternative algorithm C-burstsort (2006), stores unexamined tails of strings in buckets, instead of pointers
MP6 issues, Final Exam