

Lecture 14: Caching

Lecturer: *Przemysław Uznański*

Scribe: -

1 Cache-aware algorithms [KW02]

DAM model:

- CPU
- cache (with fast access) of size M , M/B blocks of size B
- memory/disk (with slow access) of size ∞

Cost is associated with number of memory accesses. Assume CPU cost is negligible, and actual cost comes from moving things to i from cache.

Example1: scanning N consecutive memory cells takes N/B memory transfers.

Example2: Accessing random N memory cells takes N memory transfers.

Example3: Binary search: $\log(N/B)$ (not really any significant gain)

1. B-trees, with branching factor $\Theta(B)$. Tree depth is $\log_B N$.
2. B^ϵ -trees: each node is a buffer of size B , with B^ϵ pivots. Insert amortizes and costs $\frac{\log_B N}{\epsilon B^{1-\epsilon}}$, queries cost $\frac{\log_B N}{\epsilon}$. Deletes by tombstones.
3. Sorting $\mathcal{O}(\frac{N}{B} \log_{M/B} \frac{N}{B})$ by M/B -way mergesort.

2 Cache-oblivious algorithms [BFF⁺07]

Design of cache-aware algorithms requires fine-tuning to parameters of the model. In modern systems we have many levels of caching...

The cache-oblivious model: do the algorithm that works well for (almost) any setting of parameters, as algorithm does not know B or M .

- Automatic block transfers triggered by word access with *offline optimal block replacement*.
- FIFO or LRU is 2-competitive given cache of $2 \times$ size.
- In fact it is OK to show that ANY caching strategy kind-of works.

Adapts to multi-level hierarchy.

Search trees: $\mathcal{O}(\log_B N)$. Static search tree - simulate B-tree on classic binary tree via memory placement. Take full binary tree on N nodes, cut it in half (height), so top is \sqrt{N} nodes (call it T) and bottom is \sqrt{N} trees ($T_1, \dots, T_{\sqrt{N}}$). Place in memory: place T , then $T_1, \dots, T_{\sqrt{N}}$, call recursively (van Emde Boas layout).

Analysis: cut in half until height piece size $\leq B$. So its also $\geq \sqrt{B}$. Height of a piece is between $\log B$ and $\frac{1}{2} \log B$. Number of pieces along path to root is $\leq \frac{\log N}{\frac{1}{2} \log B}$, and each piece is on at most 2 blocks.

COLA (Cache-Oblivious Lookahead Array)[BCR02]:

- $\log N$ levels
- i -th level contains 2^i elements, either completely full or completely empty
- each level is sorted

Insert: $\frac{\log N}{B}$ amortized. Naive searches: bin-search in each level, so $\log^2 N$. Refine by adding lookahead pointers: each fourth element from level i is preserved in level $i + 1$, with pointer. Then searching incurs $\log N$ cost.

References

- [BCR02] Michael A. Bender, Richard Cole, and Rajeev Raman. Exponential structures for efficient cache-oblivious algorithms. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan J. Eidenbenz, and Ricardo Conejo, editors, *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, volume 2380 of *Lecture Notes in Computer Science*, pages 195–207. Springer, 2002.
- [BFF⁺07] Michael A. Bender, Martin Farach-Colton, Jeremy T. Fineman, Yonatan R. Fogel, Bradley C. Kuszmaul, and Jelani Nelson. Cache-oblivious streaming b-trees. In Phillip B. Gibbons and Christian Scheideler, editors, *SPAA 2007: Proceedings of the 19th Annual ACM Symposium on Parallelism in Algorithms and Architectures, San Diego, California, USA, June 9-11, 2007*, pages 81–92. ACM, 2007.
- [KW02] Markus Kowarschik and Christian Weiß. An overview of cache optimization techniques and cache-aware numerical algorithms. In Ulrich Meyer, Peter Sanders, and Jop F. Sibeyn, editors, *Algorithms for Memory Hierarchies, Advanced Lectures [Dagstuhl Research Seminar, March 10-14, 2002]*, volume 2625 of *Lecture Notes in Computer Science*, pages 213–232. Springer, 2002.