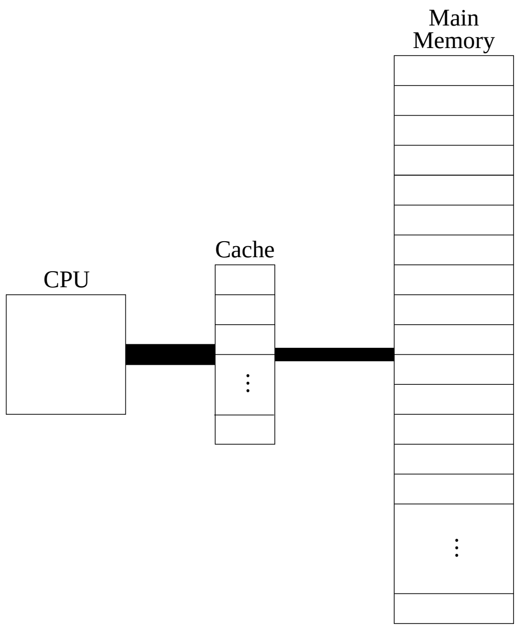
A Cache-Oblivious Heap

- Introduced by Arge et al. [1].
- Based on *distribution* of elements

References

[1] L. Arge, M. A. Bender, E. D. Demaine, B. Holland-Minkley, and J. I. Munro, Cache-oblivious priority queue and graph algorithm applications, *Proceedings of the 34th ACM Symposium on Theory of Computing*, ACM Press (2002), 268–276.

The Ideal-Cache Model



1. Automatic Replacement
2. Optimal Replacement
3. Tall Cache: $M = \Omega(B^2)$
4. Full Associativity

Often used bounds:

Scan(N) $= \Theta(\frac{N}{B})$

Sort(N) $= \Theta(\frac{N}{B} \log_{M/B} \frac{N}{B})$.

Techniques

Sequential Data Access This is obviously cache oblivious and Scan(N)

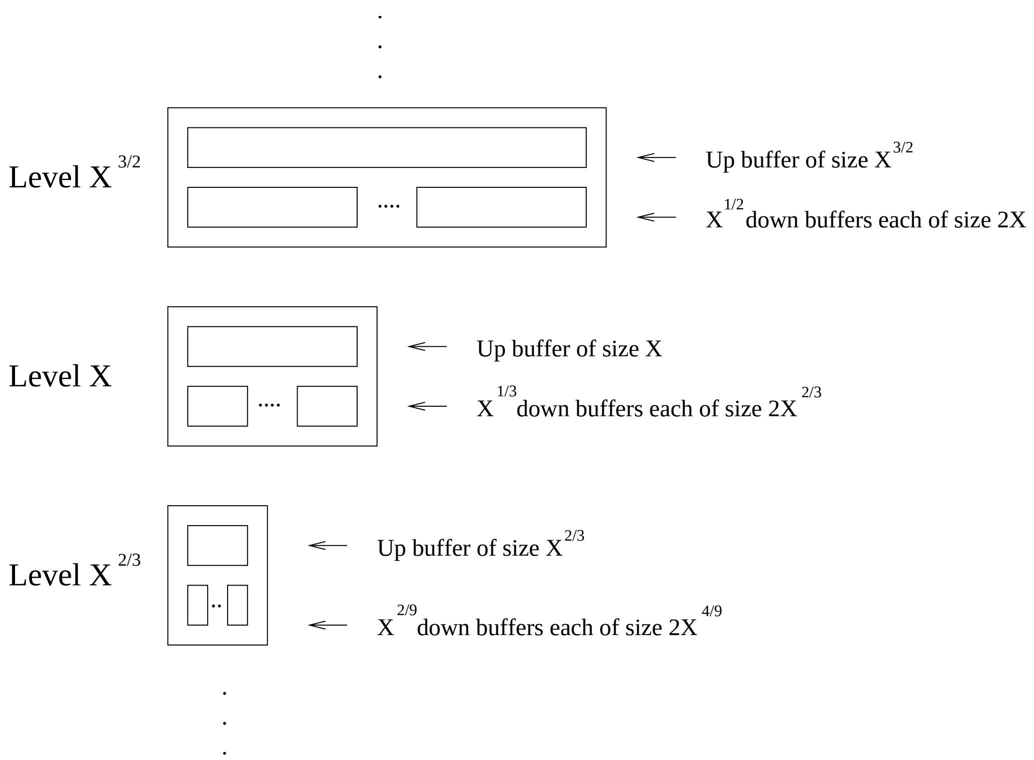
Divide-and-conquer At some point, the problem size will fit in a cache level and further division will be free (in regard to I/Os).

Recursive Layout A static data structure can be placed in memory such that for example ordinary binary tree searching becomes cache oblivious (van Emde Boas layout)

Lazy evaluation using buffers Use buffers of growing size. At some point these fit in a cache level. Elements move lazily between levels (only when levels are full or empty).

Distribution Heap

- Structure build of *levels*
- Smallest level is constant c in size
- Grows with a power of 3/2 for each level
- Levels are named after their size:
  $c, \ldots, X^{2/3}, X, X^{3/2}, X^{9/4}, \ldots, P.$
- The total number of elements in the structure is N and thus there are $\Theta(loglogN)$ levels
- A level consists of *up* and *down* buffers



Buffers

- Level X has $X^{1/3}$ downbuffers of size $2X^{2/3}$ and 1 up buffer of size X. In total $3X$.
- A down buffer on level X is then twice the size of the up buffer on level $X^{2/3}$.

Invariants:

1. At any level the elements are sorted *among* the down buffers, so that all elements in a down buffer are smaller than any element in the next down buffer.
2. Any element f in a down buffer on level X is smaller than any element g in the up buffer u^X on the same level.
3. Any element f in a down buffer on level X is smaller than any element g in a down buffer in the level above ($X^{3/2}$).

Furthermore, each down buffer on level X must contain at least $\frac{1}{2}X^{2/3}$ elements. This corresponds to keeping the buffers at least 1/4 full.

Space Usage

The original article claims $O(N)$ space, but if space is calculated as:

$$3 \sum_{i=0}^{b} c^{(3/2)^i} \leq 3c^{(3/2)^{b+1}}$$

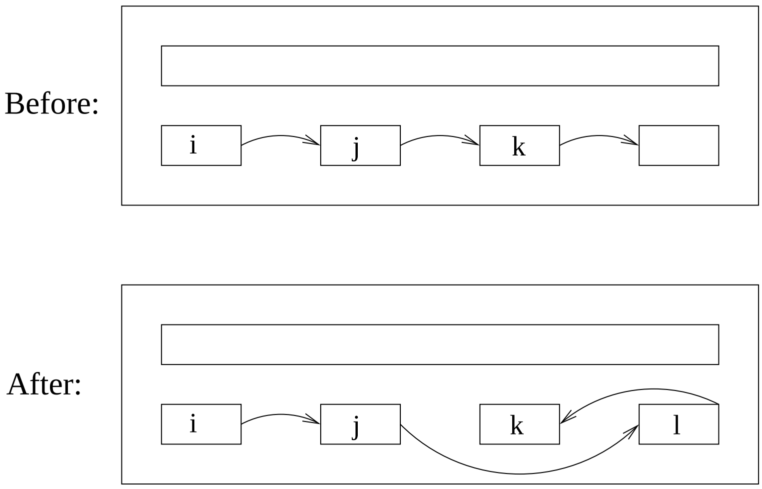
it is clear that the right side will dominate!

Fix: Divide up buffer into $X^{1/3}$ buffers of size $X^{2/3}$. A full level X uses $O(N)$ space. When one block on the next level $X^{3/2}$ is sized $(X^{3/2})^{2/3}$ this is still $O(N)$ space.

Basic Operation: push

A `push`-operation pushes the up buffer on level X into the down buffers on level $X^{3/2}$.

1. Sort the up buffer cache obliviously
2. Distribute the elements among the down buffers on level $X^{3/2}$
3. Split down buffers which runs full
4. Place remaining elements in up buffer on level $X^{3/2}$ and push recursively if needed



I/O Complexity of push

The cost of `push` between two levels X and $X^{3/2}$:

- The cost of sorting the up buffer is Sort(X)
- To distribute elements in down buffers is Scan(X) + $X^{1/2}$
- Scan(X) for FindMedian and Scan(X) for split, but only for every X elements. To split a down buffer is thus $O(1/B)$ pr. element amortized

The cost of a push is therefore Sort(X) + $X^{1/2}$

Basic Operation: Pull

A `pull`-operation fills the down buffers on level X with X elements from level $X^{3/2}$. Two cases:

1. The down buffers on level $X^{3/2}$ contains at least $\frac{3}{2}X$ elements which ensures that at least $\frac{1}{2}X$ elements are left after X elements have been removed.

2. The down buffers on level $X^{3/2}$ contains too few elements in which case a recursive pull from level $X^{9/4}$ is needed.

The operation is then to:

- Sort each of the first three down buffers on level $X^{3/2}$ (which contain at least $\frac{3}{2}X$ elements)
- Merge this with the up buffer on level X
- Fill the up buffer on level X with as many elements as before and distribute the remaining elements among the down buffers

I/O Complexity of pull

The two cases are analyzed separately:

1. The X elements are pulled by sorting the first three downbuffers on level $X^{3/2}$ and removing X elements by scanning. This is dominated by Sort(X).
2. Ignoring the cost of the recursive pull, the cost of inserting the elements from level $X^{9/4}$ on level $X^{3/2}$ is Sort($X^{3/2}$), but this can be amortized over the $X^{3/2}$ elements which have to be pulled before a new recursive pull is needed.

Distributing elements into the down buffers is done in Scan-complexity which is dominated by Sort. The total amortized bound for pull is therefore Sort(X)

Total I/O Complexity

The cost of push between level X and $X^{3/2}$ was Sort(X) + $X^{1/2}$. The cost of a pull between the same levels was Sort(X). What is the total cost?

The biggest level is P. After $P/2$ push/pull operations, the structure is *rebuild* leaving all up buffers empty and all down buffers half full. Therefore:

- At least X elements must be pushed to level X before a recursive push.
- At least X elements must be pulled from level X to level $X^{2/3}$ before a recursive pull.
- The size of $P^{2/3}$ is $O(N)$ (because it will always be half full).

The previous analysis of the cost between two levels can thus be summarized.

We want to get rid of the $X^{1/2}$ term in `pull`

1. $X \geq B^2$

   In this case $X^{1/2}$ is dominated by $sort(X)$. Show by solving: $X^{1/2} \leq O(\frac{X}{B} \log_{M/B} \frac{X}{B})$.

2. $B \leq X \leq B^2$

   Here $X^{1/2}$ might dominate. Fix by:

   - Place a partially filled memory block from each down buffer ($X^{1/2}$) and only transfer whole blocks.
   - By the tall cache assumption ($M = \Omega(B^2)$) the $X^{1/2}$ blocks fit in the cache (because $X^{1/2} \leq B$) as well as all other levels in this case, because there is only a constant number of these.
   - Similarly, this is done with all pivot-elements.

3. $X \leq B$

   The levels covered by this case have size less than $B^{3/2}$, so by the tall-cache assumption these levels can all be stored in memory.

The total amortized I/O cost pr. insert and extract operation is calculated by the sum of the cost of push and pull on all levels:

$$\sum_{i=c}^{P} O\left(\frac{1}{B}\log_{M/B}\left(\frac{i}{B}\right)\right),$$

which is dominated by the largest level P:

$$O\left(\frac{1}{B}\log_{M/B}\left(\frac{P}{B}\right)\right).$$

Arge et al. argues that since $P = O(N)$, this matches the optimal bound achievable for cache-oblivious priority queues:

$$O\left(\frac{1}{B}\log_{M/B}\left(\frac{N}{B}\right)\right).$$

We showed that $P \neq O(N)$ which could imply that the argument above does not hold. Fortunatly, the analysis above charges the cost of pull and push operations on a level to the level below, which means that the largest level is not part of the analysis. Level $P^{2/3}$ is indeed $O(N)$ which makes the argument valid.

Limited Address Space

- 32-bit computers can only address 4GB
- Many operating systems do not allow for *overcommitted* memory allocation

This is a problem for the *lazy* evaulation data structures with quickly growing size levels:

Space required for the Distribution Heap:

No. of levels	No. of integers	Memory required
1	27	108 Bytes
2	108	432 Bytes
3	531	≈2 KB
4	5.556	≈22 KB
5	211.215	≈844 KB
6	54.058.146	≈216 MB
7	76.043.050.000	≈304 GB

Space required for the The Funnel Heap

Link	No. of integers	Memory required
1	1040	≈4KB
2	28800	≈115KB
3	3457068	≈13MB
4	2522898684	≈10GB
5	12934608790536	≈51PB

Conclusion: Lazy evaluation using buffers might not seem to be such a good idea! Should grow much slower to be usable.