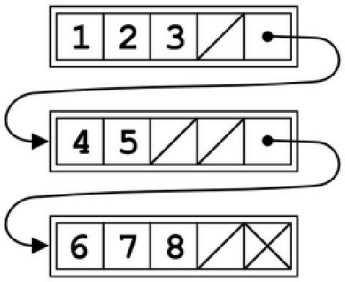
WIKIPEDIA

Unrolled linked list

In computer programming, an **unrolled linked list** is a variation on the linked list which stores multiple elements in each node. It can dramatically increase cache performance, while decreasing the memory overhead associated with storing list metadata such as references. It is related to the B-tree.



Unrolled linked list
In this model, the maximum number of elements is 4 for each node.

Contents

Overview

A typical unrolled linked list node looks like this:

```
record node {
    node next       // reference to next node in list
    int numElements // number of elements in this node, up to maxElements
    array elements  // an array of numElements elements,
                    //   with space allocated for maxElements elements
}
```

Each node holds up to a certain maximum number of elements, typically just large enough so that the node fills a single cache line or a small multiple thereof. A position in the list is indicated by both a reference to the node and a position in the elements array. It is also possible to include a *previous* pointer for an unrolled doubly linked list.

To insert a new element, we simply find the node the element should be in and insert the element into the `elements` array, incrementing `numElements`. If the array is already full, we first insert a new node either preceding or following the current one and move half of the elements in the current node into it.

To remove an element, we simply find the node it is in and delete it from the `elements` array, decrementing `numElements`. If this reduces the node to less than half-full, then we move elements from the next node to fill it back up above half. If this leaves the next node less than half full, then we move all its remaining elements into the current node, then bypass and delete it.

Performance

One of the primary benefits of unrolled linked lists is decreased storage requirements. All nodes (except at most one) are at least half-full. If many random inserts and deletes are done, the average node will be about three-quarters full, and if inserts and deletes are only done at the beginning and end, almost all nodes will be full. Assume that:

- m = `maxElements`, the maximum number of elements in each `elements` array;
- v = the overhead per node for references and element counts;
- s = the size of a single element.

Then, the space used for n elements varies between $(v/m + s)n$ and $(2v/m + s)n$. For comparison, ordinary linked lists require $(v + s)n$ space, although v may be smaller, and arrays, one of the most compact data structures, require sn space. Unrolled linked lists effectively spread the overhead v over a number of elements of the list. Thus, we see the most significant space gain when overhead is large, `maxElements` is large, or elements are small.

If the elements are particularly small, such as bits, the overhead can be as much as 64 times larger than the data on many machines. Moreover, many popular memory allocators will keep a small amount of metadata for each node allocated, increasing the effective overhead v. Both of these make unrolled linked lists more attractive.

Because unrolled linked list nodes each store a count next to the *next* field, retrieving the kth element of an unrolled linked list (indexing) can be done in $n/m + 1$ cache misses, up to a factor of m better than ordinary linked lists. Additionally, if the size of each element is small compared to the cache line size, the list can be traversed in order with fewer cache misses than ordinary linked lists. In either case, operation time still increases linearly with the size of the list.

See also

- CDR coding, another technique for decreasing overhead and improving cache locality in linked lists similar to unrolled linked lists.
- the VList, another array/singly-linked list hybrid designed for fast lookup
- the skip list, a similar variation on the linked list, offers fast lookup and hurts the advantages of linked lists (quick insert/deletion) less than an unrolled linked list
- the B-tree and T-tree, data structures that are similar to unrolled linked lists in the sense that each of them could be viewed as an "unrolled binary tree"
- XOR linked list, a doubly linked list that uses one XORed pointer per node instead of two ordinary pointers.
- Hashed array tree, where pointers to the chunks of data are held in a higher-level, separate array.

References

- Shao, Z.; Reppy, J. H.; Appel, A. W. (1994), "Unrolling lists" (http://dl.acm.org/ft_gateway.cfm?id=182453&type=pdf&CFID=80599027&CFTOKEN=68993242), *Conference record of the 1994 ACM Conference on Lisp and Functional Programming*: 185–191, doi:10.1145/182409.182453 (https://doi.org/10.1145%2F182409.182453), ISBN 0897916433

External links

- Implementation written in C (https://github.com/badgerman/quicklist)
- Another implementation written in Java (https://github.com/megatherion/Unrolled-linked-list)
- Open Data Structures—Section 3.3—SEList: A Space-Efficient Linked List (http://opendatastructures.org/ods-java/3_3_SEList_Space_Efficient_.html)

Retrieved from "https://en.wikipedia.org/w/index.php?title=Unrolled_linked_list&oldid=773251221"

This page was last edited on 1 April 2017, at 08:13 (UTC).

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.