Clément Béra ~ Smalltalk, Tips 'n Tricks

Free chunk management in the Cog VM

07 - *Thursday* - *Jun 2018*

Posted by Clement Bera in Cog, Spur

≈ Leave a comment

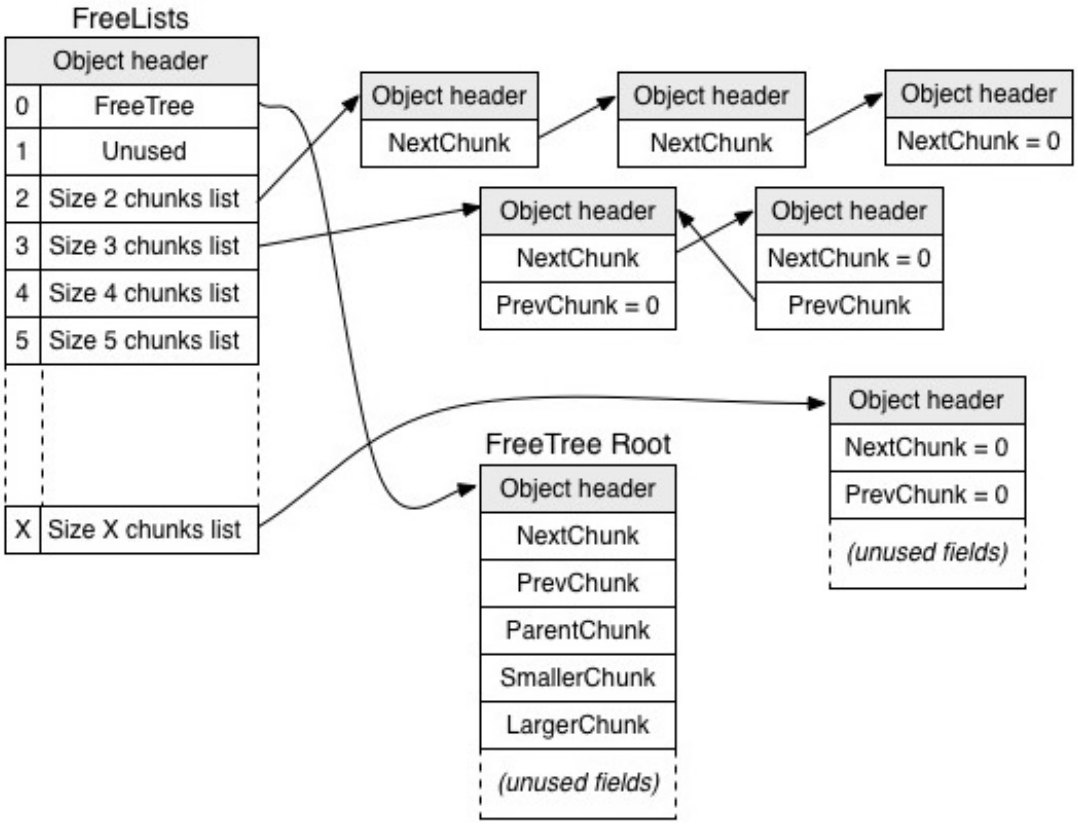
[UPDATE 5 oct: updated figure]

Hi all,

Recently I have been working on a new compactor algorithm for the Cog VM. The full GC is right now a stop the world Mark-Compact algorithm. In heaps of 1-2Gb, Between 30 and 50% of the full GC pause time is currently spent in the compaction phase. The rest of the pause is mostly due to the marking phase, but also some other things (managing mmap, etc.). I am currently trying to decrease the compaction time while using only in the common case an extra memory region (defaulted to 16Mb) whichever heap size is used. The new compaction algorithm is a hybrid Mark-Sweep-Compact, which compacts only part of the heap at each full GC based on the current occupation of the memory regions.

But! This post is not about the new compaction algorithm. It is about the recent changes I am introducing in the free chunks management to support efficient free chunks merges.

Free chunks are all referenced from a freelists object, known by the VM, as shown on the following figure. Since small free chunks are the most common, the freelists object includes a fixed number of fields to optimize free chunks allocation / deallocation of small free chunks. The exact number is based on the numFreeLists VM setting, X on the figure, currently 32 or 64 fields depending on word size.



Slots 1 to X for the freelists objects refers to the first free chunk of the given size, in allocationUnits (currently 64 bits). Since in Spur every object has at least a single field to support the forwarding object scheme, each free chunk has at least 1 field. This first field of each free chunk is abused to organize the chunks as a linked list. The first free chunk of size 1 points to the next one, etc.

With the linked list design, allocation is very efficient, but detaching a free chunk from the list may cost a lot (the VM may need to iterate over the whole list). Because of my new compaction algorithm, I needed to be able to detach efficiently free chunks to merge unmarked objects and free chunks to larger chunks during the sweep phase.

To solve this issue, I made today the free chunk linked list a double linked list. To do so, I needed 2 fields per objects. In 32 bits, that's not a problem since the allocation unit is 64 bits and each object requires at least a field for the forwarding object scheme. In 64 bits, I had to keep free chunks of size 1 in a single linked list, and detaching a free chunk of this specific size still requires to iterate over the whole linked list. The figure shows the 64 bits case, where free chunks of size 1 are organized as a single linked list, and larger free chunk as double linked lists.

Free chunks larger that X are managed differently, they're managed through a simplified version of the AVL tree with naive rebalancing which in practice is very efficient. For large free chunks, the first 5 fields are abused (we assume X >= 5). Each node in the tree is a double linked list with the free chunks of the specific size, re-using the first two fields as for the small chunks. The next three fields are used to represent the binary tree connections, parent is the parent in the tree (0 for the root), smallerIndex is the first child (a chunk with a smaller size) and largerIndex the second child (a chunk with a larger size).

With the double linked list scheme, detaching a free chunk does not require any more to walk linked list of the tree, the VM can just check the prev/parent relationships and remove the chunk. This sped-up drastically the sweep phase (most of the time was in detachFreeObject:) but it addition it improved the current production compactor which was also trying to merge free chunks in some cases.

I hope you enjoyed the post.