December 2020

DYNAMIC MEMORY MANAGEMENT WITH REDUCED FRAGMENTATION USING THE BEST-FIT APPROACH

HP INC

Dynamic memory management with reduced fragmentation using the best-fit approach

This disclosure relates to the field of Dynamic memory management in general. Disclosed idea makes use of the Best fit approach which makes use of the balanced trees with nodes sorted based on key values corresponding to free memory portion sizes. Also disclosed is the method to efficiently coalesce the freed memory.

This idea addresses the disadvantages of sequential search mechanism of finding the available free space and first-fit approach of memory management in the current flat memory-based allocators that are based on [1] approach. The current mechanism for dynamic memory management in use in most of the systems follows a sequential search for all the operations, this leads to a worst-case time complexity of O(N) and it follows the first-fit approach to allocate the first available free space for any request which leads to fragmentation issues.

The idea is based on the prerequisite that the basic block size of each memory unit is 32 bytes or above. Description of this disclosure is divided as follows:

1. Balanced tree approach: To follow best fit approach with time complexity O (log N)
2. Handling Duplicate free nodes of the same size
3. Block field coalescing: Performing memory compaction with negligible memory overhead and in O (1) time complexity

Each of above methods are explained in detail below:

1. For the Balanced Tree Approach, AVL tree [2] as depicted below in Fig. 1 is employed as the main data structure to manage the dynamic memory. AVL is a self-balancing binary tree with O (log N) time complexity for retrieval and insertion operations.
   In our idea proposed:
   - ❑ Each node in the AVL tree represents only the free available chunks of the memory managed.
   - ❑ Nodes of the tree are itself allocated in the beginning of each free available chunk
   - ❑ In the last 4 bytes of the free available chunk we maintain the size of the chunk.
   - ❑ Each node in the AVL tree maintains the following information:
     - Key: Size of the current free node
     - Link to left and right child nodes
     - Link to the parent node
     - Link to the depth node – used for duplicate handling

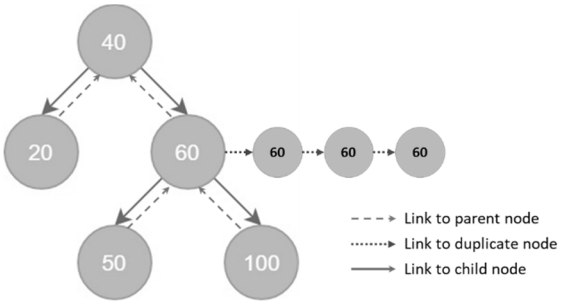

Fig. 1: Structure of the AVL tree used

The flowchart below describes the steps followed during allocation:

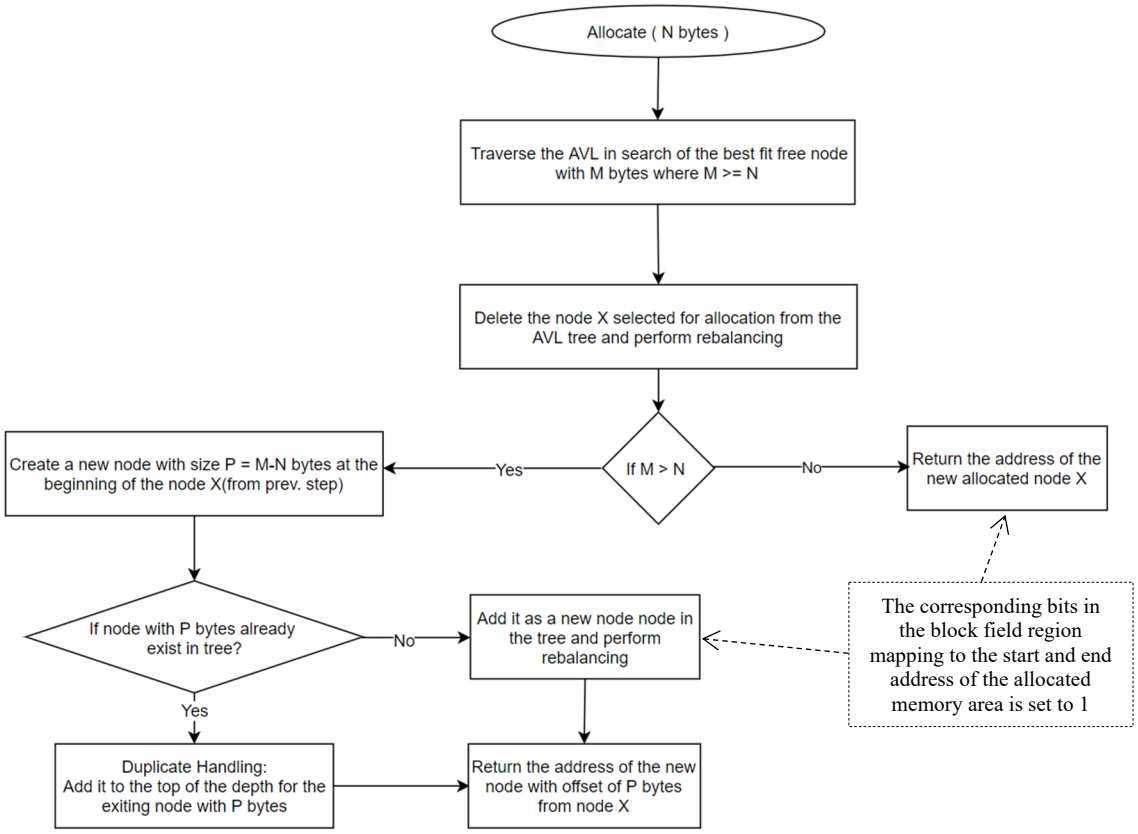

Fig. 2a: Allocation mechanism for the proposed idea

To service a request N (Fig. 2a) includes a best-fit mechanism of searching for a free memory portion that has a size of at least N with a worst-case time complexity of O (log n) and fragmenting the additional unused free memory after allocating N and inserting it as a new node. In this manner internal fragmentation can be reduced.
An example for the above process is depicted in Fig. 2b below:

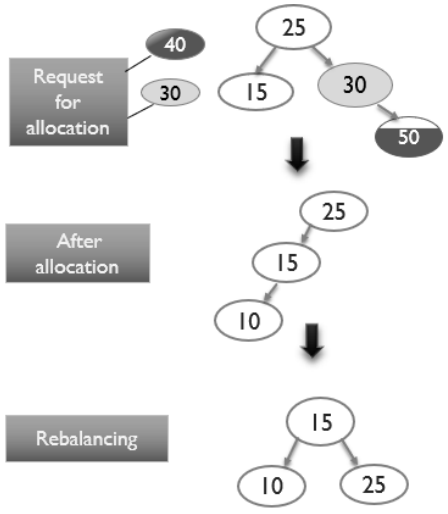

Fig. 2b: Allocation process

2. For Duplicate free node Handling, the idea proposes to maintain a link from the existing node to a new node with the same key size in the form of a linked list thereby avoiding additional duplicate nodes in the tree. The duplicate node is always added at the head of the linked list thus incurring no extra cost in terms of time for traversing the linked list.
   Duplicate handling is depicted as part of the Fig. 2a and Fig. 2b.

3. The following flowchart describes the Deallocation and Block Field coalescing mechanism

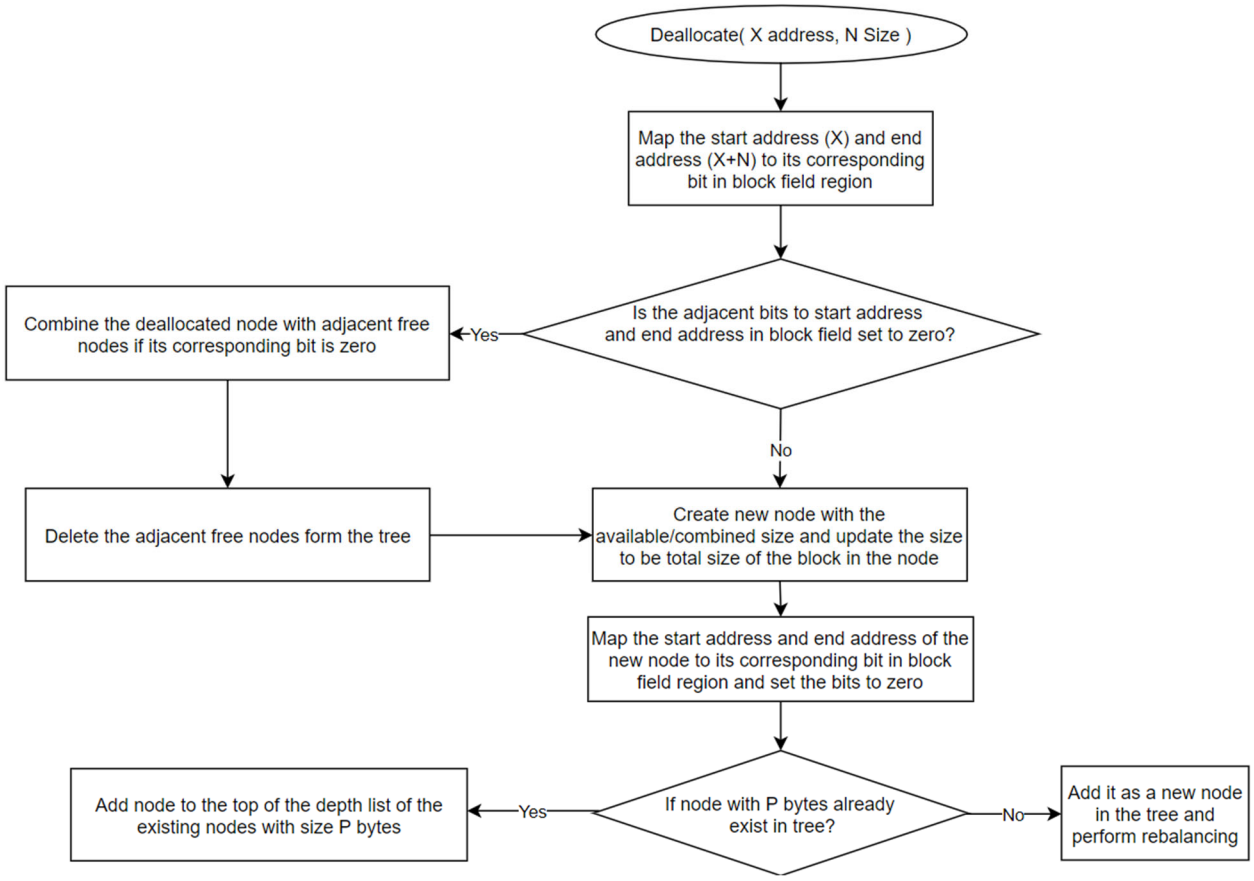

Fig. 3a: Deallocation and coalescing mechanism for the proposed idea

As shown in Fig. 3a free block of memory is inserted as a new node in the tree which involves locating the point in the tree to insert the new node based on its key value thereby maintaining the proper sorted ordering of the tree nodes.

Block Field Coalescing is performed after every free operation and done with the help of the block field region. The block field region is used to generally track and manage memory herein where each entry on the block field region represents every allocated memory portion and a free memory portion. It can be achieved by mapping only the starting address and ending address of the free block onto the block field region and mark the corresponding bits as 0 instead of marking all the bits mapping the entire memory to reduce the overhead. Then the adjacent bits of the starting block and the ending block which corresponding to the adjacent blocks of memory to the current free block in the dynamic memory region is checked for 0. If the adjacent bit is 0, we can ensure that the free blocks are existing adjacent to one another and thus coalescing is performed under such situations. Thus, the problem of

coalescing is addressed only by using a 0.16% of the dynamic memory to maintain the block fields. As example for the block field is depicted the Fig. 3b below.

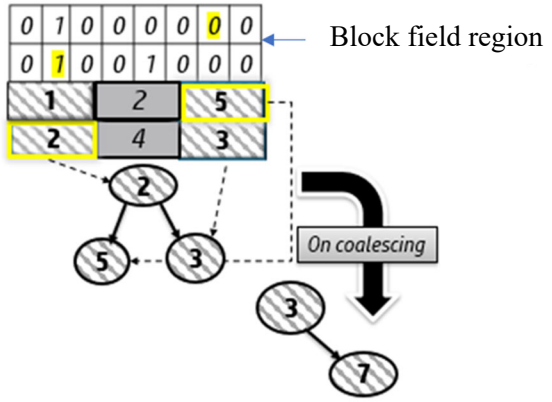

Fig. 3b Design flow for free with coalescing

Comparison of this idea with the existing approach is done henceforth to discuss the advantages and disadvantages. The existing approach [1] is based on a Doubly linked list data structure (Fig. 4) for memory management where every node represents the free blocks of memory and the nodes are arranged in the ascending order of their memory addresses.

Disadvantages:

- O(N) worst-case time complexity for all the operations occurs in [1] when the required size is present in the end of the linked list as it follows a sequential time complexity.
- Fragmentation issues are caused by [1] as it follows first-fit mechanism where the first available free block is allocated for every request

Advantages:

- The space occupied for meta data is O (0).
- Coalescing is done each time a new node is added into the linked list by checking if the adjacent nodes are adjacent blocks of free memory.

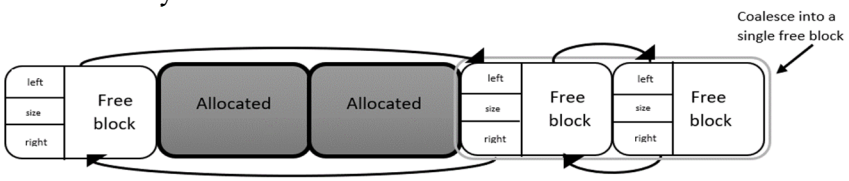

Fig. 4 Existing Linked-list based approach

The idea proposed has addressed the above disadvantages at the same time retaining all the advantages.

References

[1] Doug Lea, A Memory Allocator, 2000, http://gee.cs.oswego.edu/dl/html/malloc.html
[2] Sedgewick Robert, "Balanced Trees". Algorithms. (1983). page. 199. ISBN 0-201-06672-6.

Disclosed by Gurunandan Gurumurthy and Hrudayasri Vatsalya K, HP Inc.