Chapter 1

Literature Review

1.1 TLSF: a new dynamic memory allocator for real-time systems [4]

1.1.1 Summary

Proposes a new data structure, the Two Level segregated Free List. In Realtime Systems, scheduling algorithms require that performance be predictable and bounded. Traditional Segmented Free Lists can either cover a wide range of size classes or be very granular to reduce fragmentation. TLSF can do both and has constant access time.

1.1.2 Advantages

- Constant worst case execution time
- Low fragmentation
- Bit-level manipulation to find size class index

1.1.3 Disadvantages

- Two level list structure means potentially higher overhead
- Metadata stored directly in allocated memory (Knuth's boundary tag method)means lower cache locality

1.2 Memory allocation costs in large C and C++ programs [1]

1.2.1 Summary

Profiles various memory allocators for a variety of real malloc-heavy applications, by counting the number of instructions used for allocation and free operations.

1.2.2 Advantages

- Evaluates allocators on the basis of real rather than synthetic traces

1.2.3 Disadvantages

- Only instruction counts are being measured, this may be a heuristic for allocation speed but is not a measurement of eg. the speed of that allocated memory

1.3 The Garbage Collection Handbook: The Art of Automatic Memory Management [3]

1.3.1 Summary

Has a very comprehensive chapter on memory allocation, explains a wide variety of standard allocators and the principles behind them.

1.3.2 Advantages

- Good range of allocators presented.
- Most research papers seem to more or less summarise the contents of this chapter

1.3.3 Disadvantages

- Good starting point, but doesn't really dig into implementation details for advanced data structures.
- No concrete data, no comparison amongst allocators.

1.4 Inside Memory Management

1.4.1 Summary

Gives a simple example of how one might completely reimplement malloc using Linux Operating System calls (sbrk). Interesting but a little too in the weeds for this project.

1.5 Fundamental Algorithms

1.5.1 Summary

Similar to Garbage Collection Handbook, this is a highly influential volume. A little dated, but all modern allocators (specifically free list and buddy) have their roots in these algorithms.

1.6 The Memory Fragmentation Problem: Solved? [?]

1.6.1 Summary

Fragmentation (both internal and external) can have an effect on system performance. Both by wasting space and by scattering useful data. Previously, fragmentation was measured by synthetic trace analysis. In some cases great efforts were taken to ensure traces had statistically significant properties. This paper re-examines the implicit assumption: that synthetic trace analysis is an indicator of fragmentation in real programs, and finds this not to be true.

The paper also attempts to evaluate allocator policies independent of implementation and concludes that most fragmentation problems are actually poor implementations. The paper also examines which common features are shared by low fragmentation algorithms, and finds that immediate coalescing on free and reuse of most frequently freed blocks gives best performance.

1.6.2 Advantages

- Analyses real program traces
- Gives general recommendations, independent of implementation
- Documents experimental design clearly
- Mentions some standard programs on which allocators can be benchmarked.

1.6.3 Disadvantages

- Dated (but reproducible)
- Given that real fragmentation is actually lower than expected, are "power of two" systems as bad as the Garbage Collection Handbook makes out? This would have been an interesting point to investigate.

1.7 An algorithm with constant execution time for dynamic storage allocation. [6]

1.7.1 Summary

Proposes a one level variant of TLSF (this paper was published before TLSF was proposed), based around "power of two" size classes. Each size class holds blocks in the size range $\left[2^i, 2^{i+1}\right)$, making them flexible but still enforcing the condition that every block on the list must be able to fulfill a request. This means that memory requests do not require searching a free list.

1.7.2 Advantages

- One lookup table may be quicker than TLSF
- Allocation and free time is constant
- Memory immediately coalexced on release

1.7.3 Disadvantages

- Power of two size class means potentially more internal fragmentation, yet to be determined if this matters in practice

1.8 Fast Bitmap Fit: A CPU Cache Line friendly memory allocator for single object allocations [5]

1.8.1 Summary

Proposes encoding a small binary search tree directly into the bits of an integer. The proposed allocator can quickly search, allocate and free as the whole structure is cache-local. The allocator can also take a "hint" to attempt adjacent allocations.

1.8.2 Advantages

- Search structure is independent of underlying data, no scattering
- Search, allocation and freeing is fast, can all be done with bit-level instructions
- Search can take a hint

1.8.3 Disadvantages

- Less flexible, only a certain number of objects can be managed, objects all fixed size.

1.9 Improving the cache locality of memory allocation [2]

1.9.1 Summary

Previous research by Zorn just profiled allocators on the basis of instruction counts, but this neglects the possibility that an allocator may take more time to allocate, but allocate memory which is faster to use. This paper investigates the relationship between underlying allocator and execution time for a number of real programs. This sort of research into cache hit rates had only previously been undertaken for Garbage Collected languages, where its effect was obvious.

The paper concludes that implementation rate effects both cache hit rate and total execution time for a number of allocation-heavy programs, while noting that there isn't a single implementation which is universally best for every program.

1.9.2 Advantages

- Uses real program traces rather than synthetic

1.9.3 Disadvantages

- Strange process for simulating the cache. Although it introduces statistical variance, raw execution time might have been more realistic, especially since the benefit of cache hit rate is imrpoved execution time.

1.10 New methods for dynamic storage allocation (Fast Fits) [8]

1.10.1 Summary

Compares Free Lists with Buddy Allocators and proposes a new data structure which performs somewhere between the two. Fast Fits are based around a novel data structure called a cartesian tree. A cartesian tree is a two dimensional generalisation of a search tree where addresses are ascending from left to right and sizes are descending from top to bottom.

1.10.2 Advantages

- Significantly fewer nodes are searched in comparison to free lists
- As the root is the largest free size, requests which are too large to fulfill can be immediately rejected
- Paper seems speculative, might be interesting to investigate

1.10.3 Disadvantages

- In some cases the tree can be highly unbalanced and essentially linear
- The constraints of the cartesian tree mean that the search tree can't be rebalanced
- Appears to strawman list search. In best fit an entire list must be searched, but other allocators exist which don't do this, and they haven't been mentioned.
- Not the best cache locality in the search structure.

1.11 Self-adjusting binary search trees [7]

1.11.1 Summary

Introduces Splay Trees. A splay tree is a binary search tree where the most recent search result is "splayed" or rotated to the root. This gives the standard binary search tree performance for searching, with the added benefit that frequently accessed nodes will be quicker to retrieve.

1.11.2 Advantages

- Potentially fulfills both of the criteria for low fragmentation: immediate coalesce on free and reuse of most recently freed blocks

1.11.3 Disadvantages

- Good average behaviour, but worst behaviour is linear
- Not the best cache locality in the search structure.

Bibliography

[1] David Detlefs, Al Dosser, and Benjamin Zorn. Memory allocation costs in large c and c++ programs. *Softw. Pract. Exper.*, 24(6):527–542, jun 1994.

[2] Dirk Grunwald, Benjamin Zorn, and Robert Henderson. Improving the cache locality of memory allocation. In *Proceedings of the ACM SIGPLAN 1993 Conference on Programming Language Design and Implementation*, PLDI '93, page 177–186, New York, NY, USA, 1993. Association for Computing Machinery.

[3] Richard Jones, Antony Hosking, and Eliot Moss. *The Garbage Collection Handbook: The Art of Automatic Memory Management*. Chapman & Hall/CRC, 1st edition, 2011.

[4] M. Masmano, I. Ripoll, Alfons Crespo, and Jorge Real. Tlsf: A new dynamic memory allocator for real-time systems. volume 16, pages 79– 88, 01 2004.

[5] Dhruv Matani and Gaurav Menghani. Fast bitmap fit: A cpu cache line friendly memory allocator for single object allocations, 2021.

[6] T. Ogasawara. An algorithm with constant execution time for dynamic storage allocation. In *Proceedings Second International Workshop on Real-Time Computing Systems and Applications*, pages 21–25, 1995.

[7] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, jul 1985.

[8] C. J. Stephenson. New methods for dynamic storage allocation (fast fits). *SIGOPS Oper. Syst. Rev.*, 17(5):30–32, oct 1983.