Immovable Objects Meet An Irresistible Force: Meshing Memory Management for Compaction without Relocation

Emery D. Berger Andrew McGregor

College of Information and Computer Sciences
University of Massachusetts Amherst
emery@cs.umass.edu, mcgregor@cs.umass.edu

Abstract

Because C and C++ objects cannot be relocated, memory allocators for those languages cannot compact objects and thus can suffer potentially catastrophic fragmentation. In a classic result, Robson showed that such allocators can consume up to $(O \log M/m)$, where M/m is the ratio of the largest and smallest object request sizes [8]. We present a counter-intuitive result: a memory management algorithm that can perform compaction without relocating objects. Our approach *meshes* together objects from separate pages whose virtual offsets do not overlap, compacting them physically onto a single page while maintaining their virtual addresses. Meshing depends on widely-available OS support and a randomized algorithm that provides provably effective compaction with high probability. We demonstrate analytically and empirically that meshing is practical and effective, and present MESH, a prototype meshing memory allocator.

1. Introduction

C/C++ addresses are available and frequently used. Can't move objects around.

Therefore, can't perform compaction.

Classical bounds: worst-case, tons of fragmentation [8].

Empirical studies suggest fragmentation is low (Wilson cite [5]) but the worst-case is still a problem. In real-time settings, can't rely on malloc and must either pre-allocate all memory into pools, or rely on compacting garbage collection.

OOPSLA'16, XXX.

ACM XXX. . . $15.00.
http://dx.doi.org/10.1145/2815400.2815409

Connect up to conservative garbage collection (cite Boehm); note limited ability to reduce fragmentation (perhaps in related work [9]).

We present a counter-intuitive result. *Meshing memory management*. Explain key insight (keep virtual addresses while compacting). We mesh together pages when the objects on the page don't have overlapping offsets. We re-map the virtual address to point to the merged physical page, and discard the old physical page.

Clearly, this can't always work. In the worst case, an adversarial program could fill every page with a single object at the same offset (e.g., at the start of the page), which would thwart any attempts at meshing.

We resolve this problem by using a randomized algorithm that makes such adversarial situations extremely unlikely. We show analytically that our algorithm guarantees that it can achieve significant compression when there is available space with high probability.

Demonstrate implementation.

Results.

Outline of rest of paper.

2. Analysis

Pick m random strings of length b with n ones. Let X^s be the number of copies of string s and note that $X^s \sim Bin(m, p)$ where $p = 1/\binom{b}{n}$. We say a pair of strings s^1 and s^2 are combinable if $s^1 \cdot s^2 = 0$. A set of strings is combinable if they are pairwise combinable.

2.1 Combining Pairs of Strings

Partition the set of all binary strings of length b with n ones into the sets

$$A = \{x \in \{0,1\}^b : |x| = n, x_1 = 1\}$$

and

$$B = \{x \in \{0,1\}^b : |x| = n, x_1 = 0\} .$$

In the case where $n = b/2$, the number of unpaired strings is

$$U = \sum_{s \in A} |X^s - X^{\bar{s}}|$$

since s is only combinable with its complement $\bar{s}$ when $n = b/2$.

Theorem 2.1.

$$D_m(p)/(2p) \leq E(U) \leq D_m(p)/p \,,$$

where

$$D_m(p) = 2(1-p)^{N-\lfloor mp \rfloor} p^{1+\lfloor mp \rfloor} (\lfloor mp \rfloor + 1) \binom{N}{\lfloor mp \rfloor + 1}$$

and note that $\sqrt{mp(1-p)}/\sqrt{2} \leq D_m(p) \leq \sqrt{mp(1-p)}$ *on the assumption that* $1/m \leq p \leq 1 - 1/m$.

Proof. Let $X \sim Bin(m, p)$. By the triangle inequality and linearity of expectation:

$$\begin{aligned} E[U] &= \sum_{s \in A} E[|X^s - X^{\bar{s}}|] \\ &\leq \sum_{s \in A} E[|X^s - mp|] + E[|X^{\bar{s}} - mp|] \\ &= 1/p \cdot E[|X - mp|] \end{aligned}$$

Define the event,

$$A = \{X^s \geq mp \geq X^{\bar{s}} \text{ or } X^s \leq mp \leq X^{\bar{s}}\}$$

By linearity of expectation:

$$\begin{aligned} E[U] &\geq \sum_{s \in A} P(A) E[|X^s - X^{\bar{s}}| \big| A] \\ &\geq 1/2 \cdot \sum_{s \in A} E[|X^s - mp|] + E[|X^{\bar{s}} - mp|] \\ &= 1/(2p) E[|X - mp|] \end{aligned}$$

The result follows since by a theorem of De Moivre, the absolute deviation of the binomial distribution is

$$E[|X - mp|] = D_m(p)$$

$$= 2(1-p)^{m-\lfloor mp \rfloor} p^{1+\lfloor mp \rfloor} (\lfloor mp \rfloor + 1) \binom{N}{\lfloor mp \rfloor + 1}$$

The bound on $D_m(p)$ is due to Blyth (1980) and Berend and Kontorovich (2013). □

Corollary 2.1. *For any* $n \leq b/2$,

$$\lim_{m \to \infty} E(U)/m = 0$$

i.e., the fraction of strings that are unpaired tends to zero when $m \to \infty$.

Proof. By the previous theorem, if $n = b/2$, $E(U)/m \leq D_m(p)/(mp) \leq \frac{\sqrt{(1-p)}}{\sqrt{mp}} \to 0$ as $m \to \infty$. Note that the bound also holds for $n \leq b/2$ because decreasing the weight of the strings only increases the probability that a pair of strings is combinable. □

2.1.1 Random Graph Approach

Consider the graph where nodes correspond to sampled strings where there is an edge between two nodes iff the strings are compatible. So each edge is present with probability $q = \binom{b-n}{n} / \binom{b}{n}$. Note that edges are pair-wise independent but not three-wise independent (e.g., there are fewer triangles than you would expect). However, the number of incident edges on a node is $Bin(m-1, q)$.

Let M be the size of the maximum in the graph. This number is the maximum number of mutually combinable pairs. Had the edges been fully independent then $E[M] \geq \frac{m}{2} \frac{c}{c+1}$ if $c = q(m-1)/2$ is constant.

Assuming mq is a small constant less than 1, the following lemma gives a good bound.

Lemma 2.1.

$$\begin{aligned} & \frac{m}{2} \cdot \max \Big(2 - 2(1-q)^{m-1} - (m-1)q, \\ & (m-1)q \left(1 - 2q + \frac{\binom{b-2n}{n}}{\binom{b}{n}} \right)^{m-2} \Big) \\ \leq \;& E(M) \\ \leq \;& \frac{m}{2} \cdot \min(1, (m-1)q) \,. \end{aligned}$$

E.g., for $b = 32, n = 14$ *and* $m = 5000$ *we get* $78.5 \leq E(M) \leq 81$.

Since $E(M)$ *is monotonically increasing in* q, *it follows that for* $q \geq 1 - 2^{-1/(m-2)}$ *we may deduce that,*

$$0.307 \times \frac{m}{2} = (1 - \ln(2)) \frac{m}{2} \leq E(M)$$

since $2 - 2(1-q)^{m-1} - (m-1)q$ *is maximized at* $q = 1 - 2^{-1/(m-2)}$.

It also follows from the upper bound,

$$P(M \geq 1) \leq \binom{m}{2} q \,.$$

E.g., for $b = 32, n = 14$, *we need* $m \geq 394$ *for* $P(M \geq 1) \geq 1/2$.

Proof. The upper bound follows from the fact that M is at most the number of edges in the graph. Hence, $E(M) \leq \binom{m}{2} q$.

Let I be the number of *isolated edges*; an edge uv is isolated if uv is the only edge in the graph incident on u or v. Let $I_e = 1$ if e is isolated, and $I_e = 0$ otherwise. Then

$$\begin{aligned} & E(I_{uv}) \\ = \;& P(uv \in G) P(uw, vw \notin G \; \forall w \in V \setminus \{u, v\} | uv \in G) \\ = \;& q \left(1 - 2q + \frac{\binom{b-2n}{n}}{\binom{b}{n}} \right)^{m-2} \,. \end{aligned}$$

The first lower bound then follows from $E(I) = \sum_e E(I_e)$.

Another lower bound follows from the following idea. Consider the graph formed by removing all but one incident edge from any node with two or more incident edges. The remaining graph is a matching. The expected size of this matching is at least (because some edge removals may have been counted twice):

$$\begin{aligned} & \binom{m}{2}q - \sum_v E(\max(deg(v)-1,0) \\ = & \binom{m}{2}q - m\sum_{d=1}^{m-1}\binom{m-1}{i}q^i(1-q)^{m-1-i}(i-1) \\ = & \binom{m}{2}q - m[(m-1)q - 1 + (1-q)^{m-1}] \\ = & m(1-(1-q)^{m-1}) - \binom{m}{2}q\,. \end{aligned}$$

□

Note that the expected number of triangles in such a graph is $\binom{m}{3}\binom{b-n}{n}/\binom{b}{n}\binom{b-2n}{n}/\binom{b}{n}$. For example, if $b = 32, n = 10, m = 1000$, we expect less than two triangles. Hence, with such parameters, it only makes sense to look for pairs of blocks that can be combined rather than sets of three or more blocks that can be combined together.

2.1.2 Random Walk Approach

Keep on adding strings up to m. If a new string combines with an existing one, remove both. Let S_t be number of strings at time t. Let $q = \binom{b-n}{n}/\binom{b}{n} = (1-n/b)(1-n/(b-1))\ldots(1-n/(b-n+1)) \geq (1-n^2/(b-n+1))$. Then,

$$P(S_{t+1} = S_t - 1) \geq q$$

$$P(S_{t+1} = S_t + 1) \leq 1 - q$$

If $q \leq 1/2$ then $E[S_t] = O(\sqrt{t})$ and $q \leq 1/2$ if $b \geq 2n^2 + n - 1$. As a consequence, if $q \leq 1/2$, we expect the number of unpaired strings to go to zero.

2.2 Combining Multiple Strings

Note that this is only possible if $n \leq b/3$.

Lemma 2.2. *Define a graph G where each string corresponds to a node and there is an edge between two nodes if the two strings are* not *combinable. Then, the optimal number of combined strings is the chromatic number of G.*

Unfortunately, the edges of G are not independent. Had they been independent, we would have been able to appeal to existing analysis to show that the expected value of the chromatic number is

$$\frac{m}{2\log_{1/q} m} \approx \frac{(1-q)m}{2\ln m}$$

where q is the probability of being combinable.

The dependencies between edges means (u,v) and (v,w) present implies (u,w) are less likely to be present.

However, the following lemma gives a weaker bound for our case:

Lemma 2.3. *It is possible to combine strings such that less than $(1+\epsilon)(m-1)(1-q)+1$ remain with probability at least $1 - me^{-\epsilon^2(m-1)(1-q)/3}$. For example, if $b = 10, n = 2, m = 10000$, then the final number of strings is at most 4156 with probability at least $1 - 3.92030 \times 10^{-13}$ by setting $\epsilon = 0.1$.*

Proof. The result follows because a greedy algorithm can color a graph in $\Delta + 1$ colors where Δ is the maximum degree of the graph. Pick an arbitrary node v and let the degree of v be D. Then $E[D] = (m-1)(1-q)$ and by the Chernoff bound

$$P(D \geq (1+\epsilon)(m-1)(1-q)] \leq e^{-\epsilon^2(m-1)(1-q)/3}\,.$$

Hence by the union bound, the probability that all degrees at less than $(1+\epsilon)(m-1)(1-q)$ is at most $me^{-\epsilon^2(m-1)(1-q)/3}$.

□

3. Related Work

BiBoP allocators.

Randomized memory management. Several previous memory managers have employed randomization, primarily for fault tolerance and security [**?** **?** **?** **?**]. DieHard uses randomized memory allocation to provide *probabilistic memory safety*, which enables (probabilistically) correct execution in the face of memory usage errors like heap buffer overflows and use-after-free [**?**]. Archipelago extends DieHard by placing each object in a randomly-selected page in a 64-bit address space, significantly extending its ability to tolerate memory errors [**?**]. Archipelago reduces its footprint by copying unused objects into a conventional heap and discards the backing physical page; it then uses virtual memory page protection to intercept reads and re-instantiates the physical page on demand. DieHarder extends DieHard to provide security against attacks that exploit memory errors [**?**], while Exterminator leverages a randomized memory allocator to perform statistical inference and automatically fix the application by generating "runtime patches" [**?**]. Stabilizer randomizes the placement of code, stack frames, and heap objects to enable statistically sound performance evaluation [**?**]; we adopt Stabilizer's shuffling heap approach. Unlike all of these, meshing uses randomization to enable it to effectively compact objects.

Virtual memory approaches. Appel and Li outline a number of approaches for using virtual memory in user programs [2]. Numerous memory managers have exploited virtual memory primitives for a variety of purposes [1, 3, 4, 6**?** **?** , 7]. The only memory manager that merges virtual pages is

Hound, a memory leak and "bloat" detector [**?**]. Hound uses periodic page protection to track accesses and thus identify stale objects. By allocating objects sequentially so they are age-segregated, it eventually isolates leaked objects and bloat on their own pages. To avoid mixing young and old objects, Hound does not reuse freed slots on a page until it becomes completely empty. Because this lack of reuse could lead to catastrophic memory consumption, Hound merges virtual pages onto physical pages as a backstop. Meshing adapts this approach and combines it with a randomized algorithm that lets it effectively merge pages regardless of allocation pattern. Unlike meshing, Hound's deterministic allocation strategy cannot guarantee successful compaction and is best-effort only.

References

[1] A. W. Appel, J. R. Ellis, and K. Li. Real-time concurrent collection on stock multiprocessors. In *Proceedings of the ACM SIGPLAN 1988 Conference on Programming Language Design and Implementation*, PLDI '88, pages 11–20, New York, NY, USA, 1988. ACM.

[2] A. W. Appel and K. Li. Virtual memory primitives for user programs. In *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS IV, pages 96–107, New York, NY, USA, 1991. ACM.

[3] H.-J. Boehm, A. J. Demers, and S. Shenker. Mostly parallel garbage collection. In *Proceedings of the ACM SIGPLAN 1991 Conference on Programming Language Design and Implementation*, PLDI '91, pages 157–164, New York, NY, USA, 1991. ACM.

[4] D. Dhurjati and V. Adve. Efficiently detecting all dangling pointer uses in production servers. pages 269–280, 2006.

[5] M. S. Johnstone and P. R. Wilson. The memory fragmentation problem: Solved? In *Proceedings of the 1st International Symposium on Memory Management*, ISMM '98, pages 26–36, New York, NY, USA, 1998. ACM.

[6] Microsoft Corporation. The structure of a page heap block. `https://msdn.microsoft.com/en-us/library/ms220938(v=vs.90).aspx`.

[7] B. Perens. Electric Fence. `http://linux.die.net/man/3/efence`.

[8] J. M. Robson. Worst case fragmentation of first fit and best fit storage allocation strategies. *The Computer Journal*, 20(3):242–244, 1977.

[9] G. Rodriguez-Rivera, M. Spertus, and C. Fiterman. A non-fragmenting non-moving, garbage collector. In *Proceedings of the 1st International Symposium on Memory Management*, ISMM '98, pages 79–85, New York, NY, USA, 1998. ACM.