A Scalable Concurrent `malloc(3)` Implementation for FreeBSD

Jason Evans <jasone@FreeBSD.org>

Overview

- What is `malloc(3)`?
- Previous allocators
- jemalloc algorithms and data structures
- Benchmarks
- Fragmentation
- Discussion

What is `malloc(3)`?

- C API for manual memory allocation/deallocation.
- Historically: malloc(), calloc(), realloc(), free().
- More recently: posix_memalign().
- Non-standard: valloc(), reallocf(), memalign().

API shortcomings

- No bounds checking (C problem).
- Size not externally available.
- No way to specify object use/lifetime.
- Lacking debugging facilities.
- In summary: very basic API.

Partial solutions

- Redzones catch some buffer overflows.
- malloc_usable_size(). (Ugly, but simple).
- Special allocation functions (batched allocation, like in newer dlmalloc).
- Arenas, pools, slabs, etc.
- Opinion: partial solutions just muddle things.

A few other implementations

- dlmalloc.
- ptmalloc.
- Hoard.
- phkmalloc.
- lkmalloc.
- libumem.
- Vam.

dlmalloc

- Region-based (boundary tags).
- Small objects intermixed (no segregation).
- Deallocation coalesces (delayed).
- Very tricky to tune, but the author has put in the time to do so.
- Some workloads cause severe fragmentation.

ptmalloc

- Based on dlmalloc.
- Used in GNU libc.
- Creates additional arenas on demand, helps with SMP scalability (degrades beyond 6-8 CPUs).
- Per-arena locking.

Hoard

- Multiple arenas.
- Pages contain only a single size class.
- Emptiness of arenas bounded to avoid “blowup”.

phkmalloc

- Previous FreeBSD allocator.
- Size classes are powers of two for small objects.
- Allocator metadata stored separately from application’s allocated objects (no interspersed free lists).

lkmalloc

- Region-based.
- Deallocation immediately coalesces.
- Multiple arenas. Thread IDs hashed --> arenas.
- Per-free list locking.

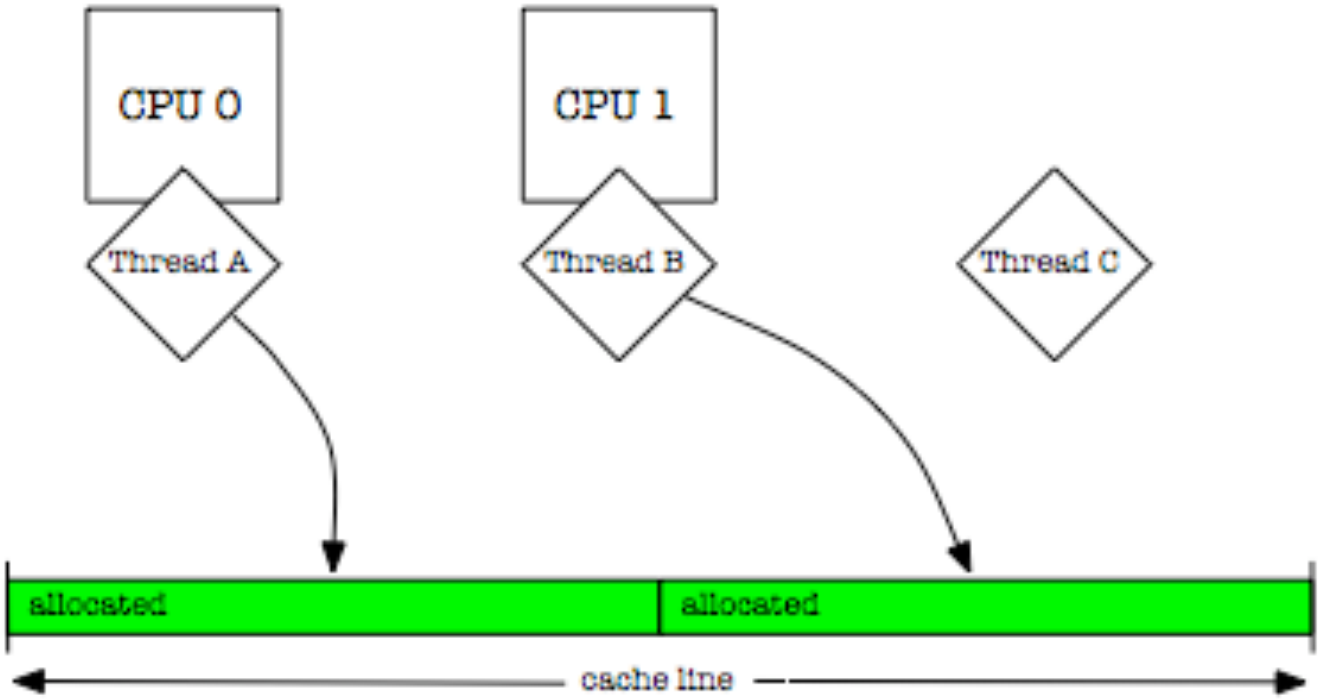
Problems jemalloc solves

- SMP scalability for multi-threaded programs (similar to lkmalloc).
- Bounded fragmentation for the cases that matter (similar to phkmalloc, vam).

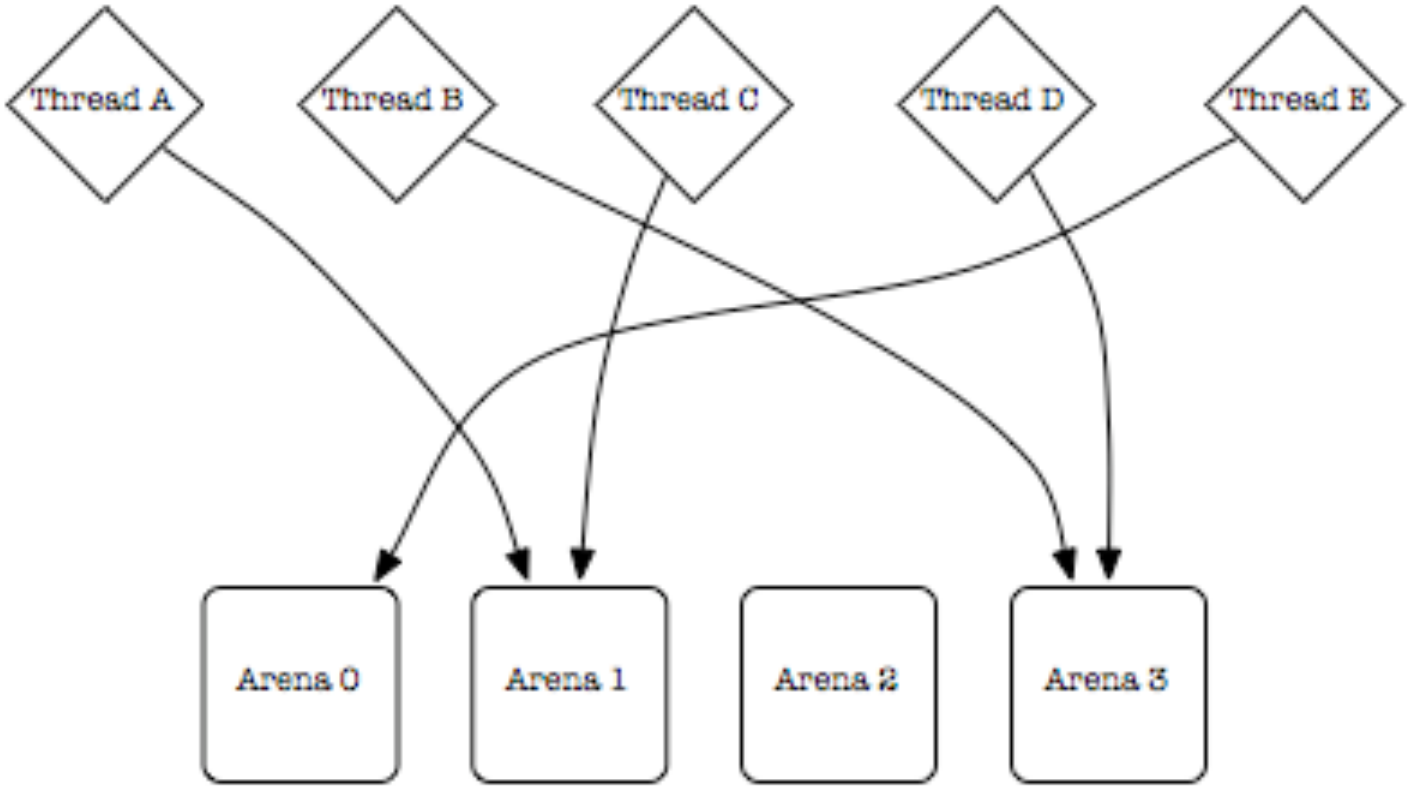
SMP scalability issues

- Mutual exclusion lock contention.
- Cache sloshing.
- False cache line sharing.

False cache line sharing



lkmalloc's thread ID hashing



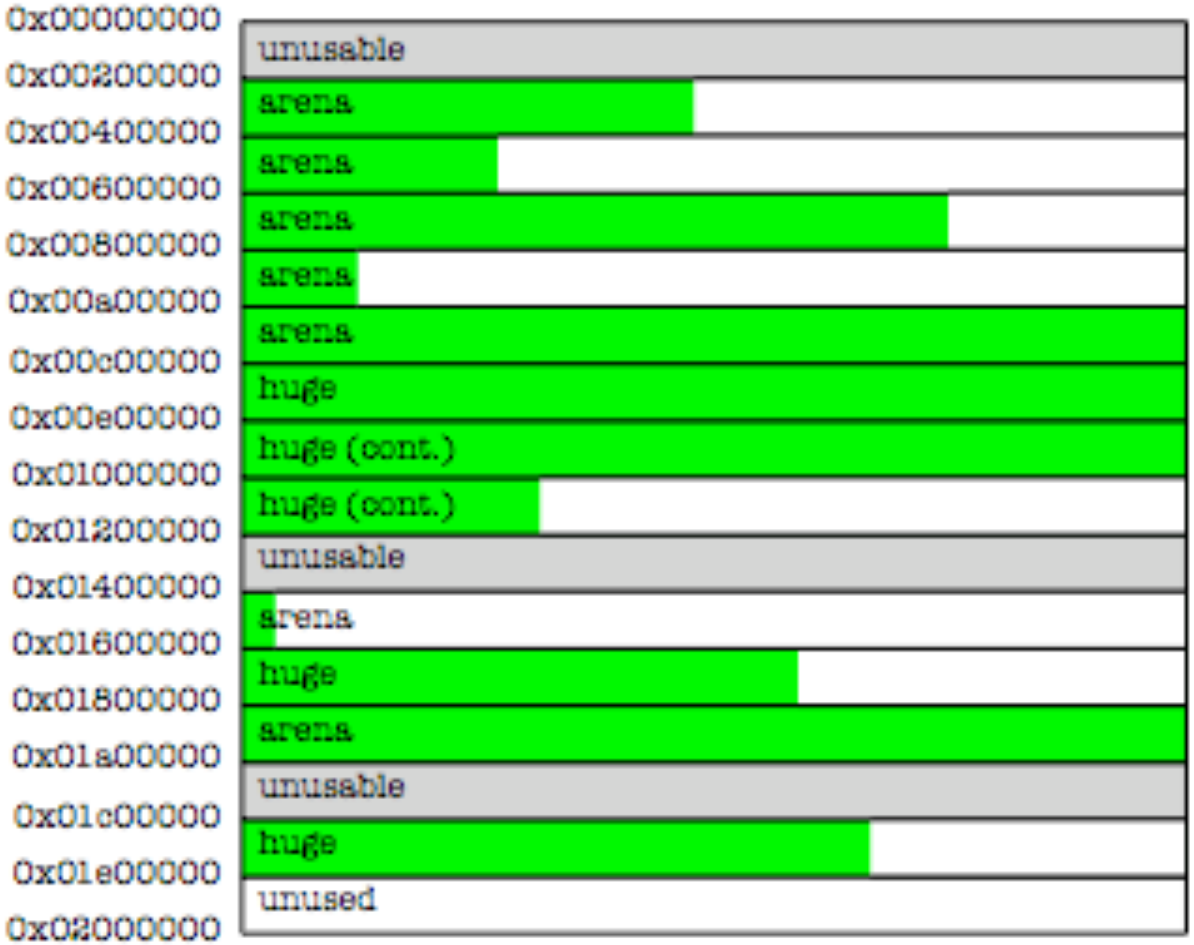
lkmalloc shortcomings

- Pointer hashing is very difficult to do well.
- False cache line sharing still a serious problem. (Boundary tags exacerbate the problem for user allocations.)

jemalloc overview

- Chunks, can be split into runs.
- Bitmaps track small objects in runs.
- Metadata stored separately from app's allocations (no interspersed free lists).
- Multiple arenas. TLS maps threads --> arenas. Arenas own chunks that are split into runs.
- Per-arena locking.

Chunks


0x00000000
0x00200000
0x00400000
0x00600000
0x00800000
0x00a00000
0x00c00000
0x00e00000
0x01000000
0x01200000
0x01400000
0x01600000
0x01800000
0x01a00000
0x01c00000
0x01e00000
0x02000000
unusable
arena
arena
arena
arena
arena
huge
huge (cont.)
huge (cont.)
unusable
arena
huge
arena
unusable
huge
unused

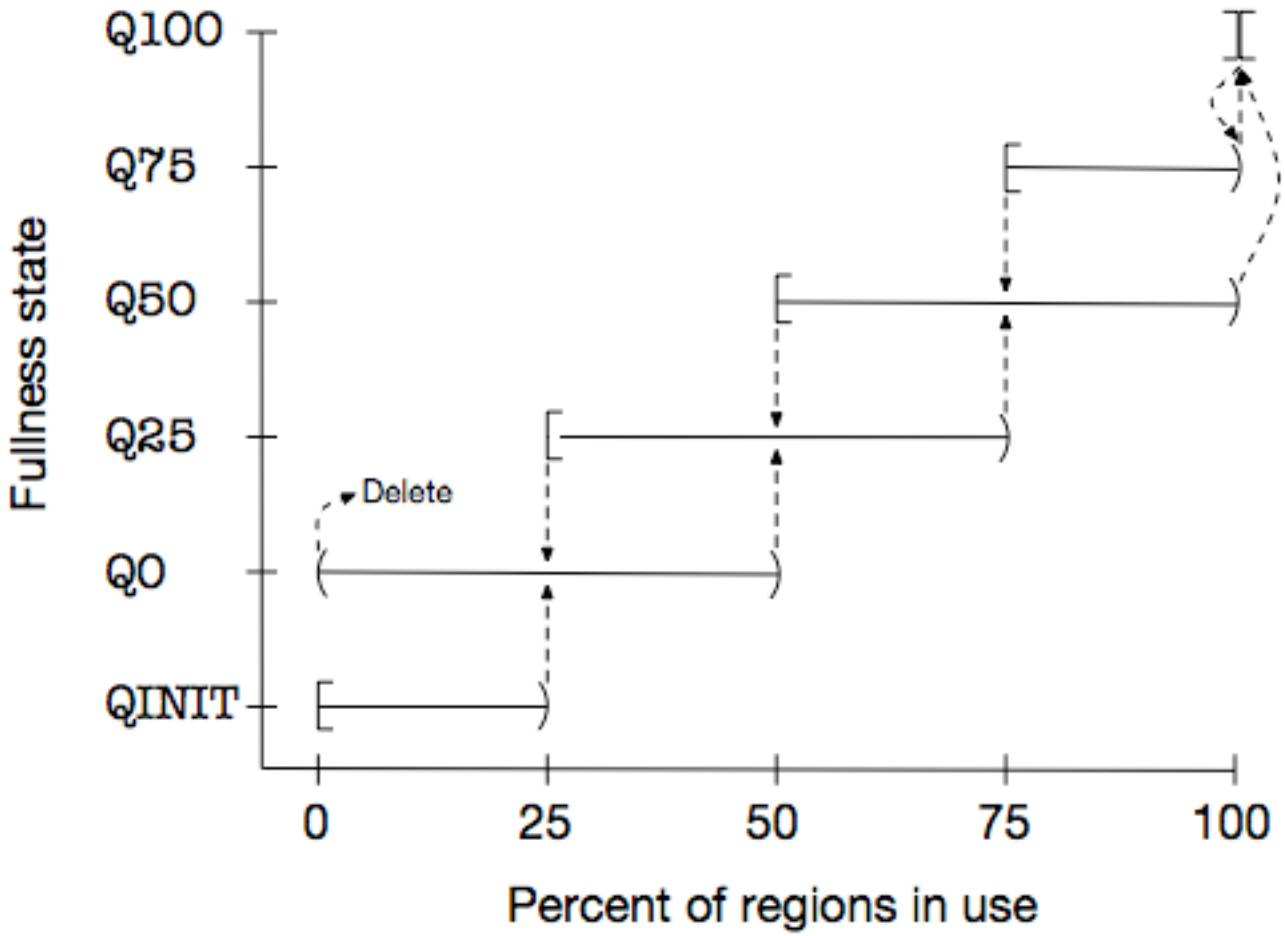
Small size classes

- Stored in runs, managed by per-run bitmaps.
- Address-ordered allocation.
- Tiny (2, 4, 8). Technically insufficiently aligned, not an issue in practice.
- Quantum-spaced (16, 32, 48, …, 480, 496, 512). (Reduce fragmentation.)
- Sub-page (1kB, 2kB).

Large/huge size classes

- Large (4kB, 8kB, 16kB, …, 256kB, 512kB, 1MB). Stored as runs (page-aligned).
- Huge (2MB, 4MB, 6MB, …). Stored as chunks.

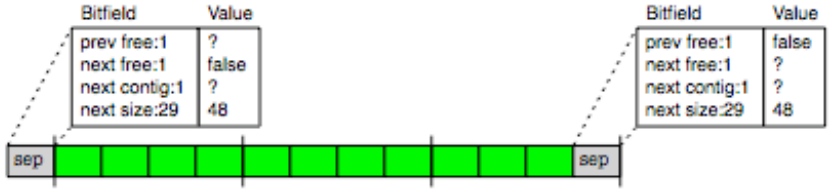
Keeping runs full/empty



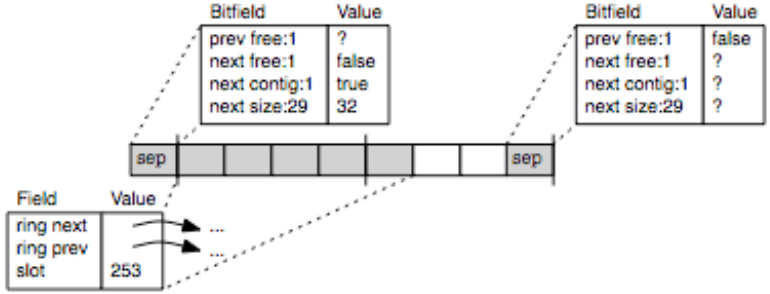
Problems with region-based jemalloc

- Complex.
- Fragmentation! Very sensitive to allocation patterns.
- Slab allocation missing.
- Object alignment not cache-line-friendly.

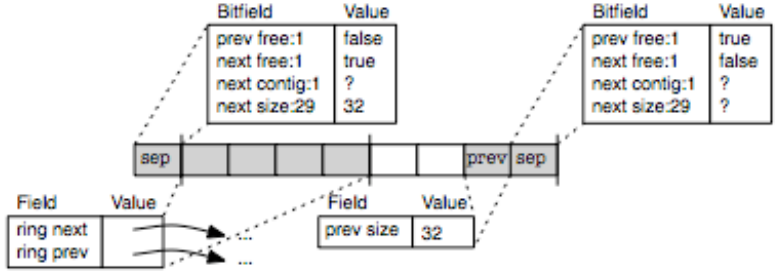
a) Allocated



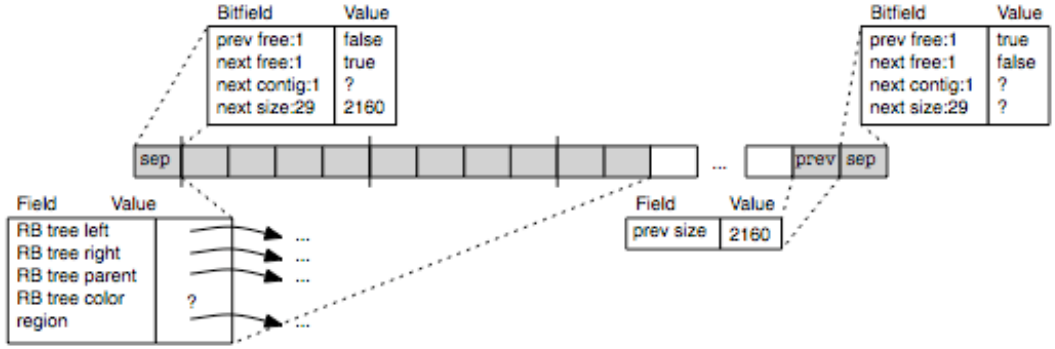
b) Delayed



c) Free (small)



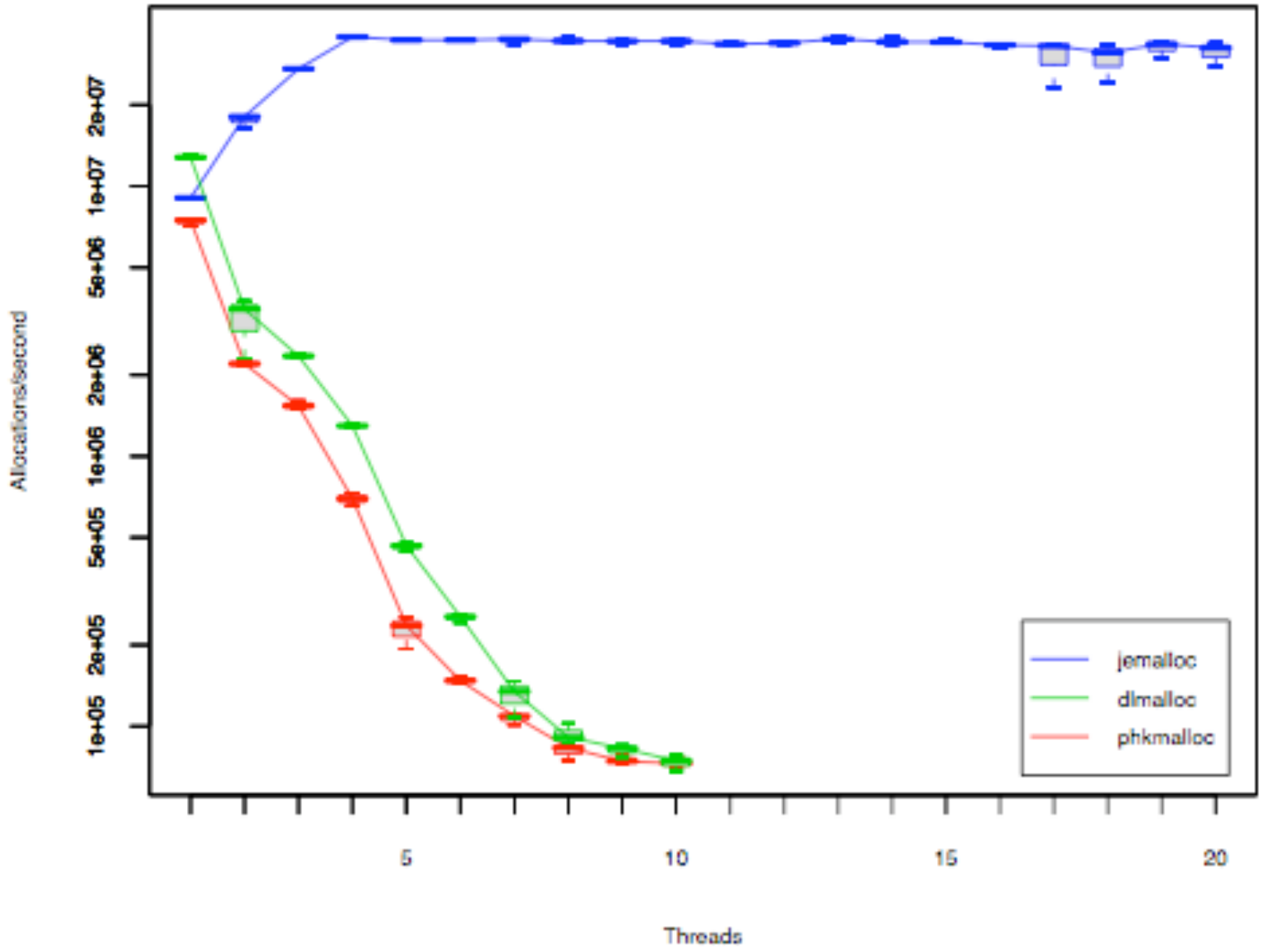
d) Free (large)



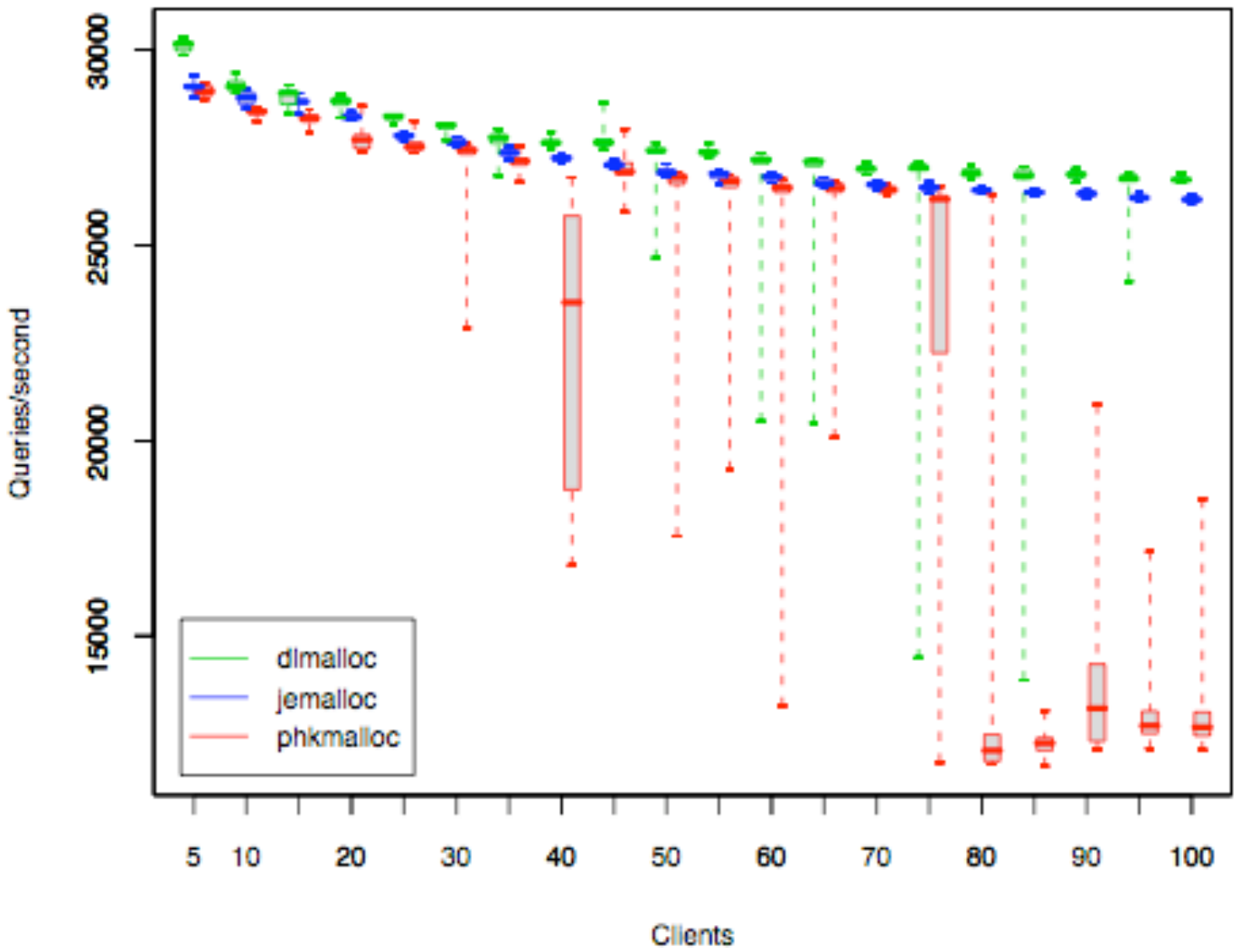
Benchmarks

- dlmalloc, phkmalloc, and jemalloc compared. Others would have been nice (ptmalloc, hoard, libumem).
- Multi-threaded: malloc-test, super-smack (select-key).
- Single-threaded: cca, cfrac, gs, sh6bench, smlng. (worldstone)

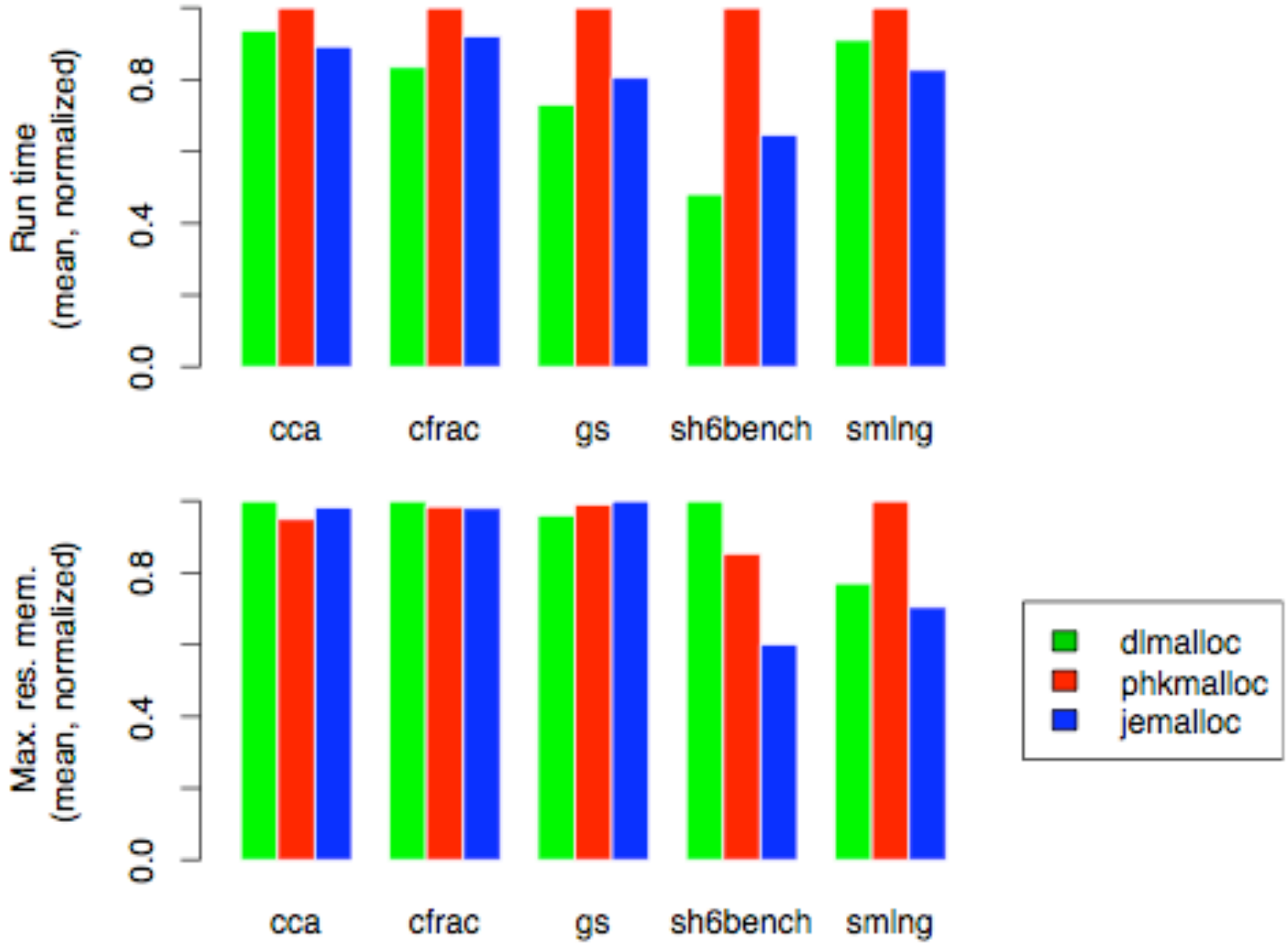
malloc-test


Allocations/second
1e+05
2e+05
5e+05
1e+06
2e+06
5e+06
1e+07
2e+07
5
10
15
20
Threads
jemalloc
dlmalloc
phkmalloc

super-smack


Queries/second
30000
25000
20000
15000
dlmalloc
jemalloc
phkmalloc
5
10
20
30
40
50
60
70
80
90
100
Clients

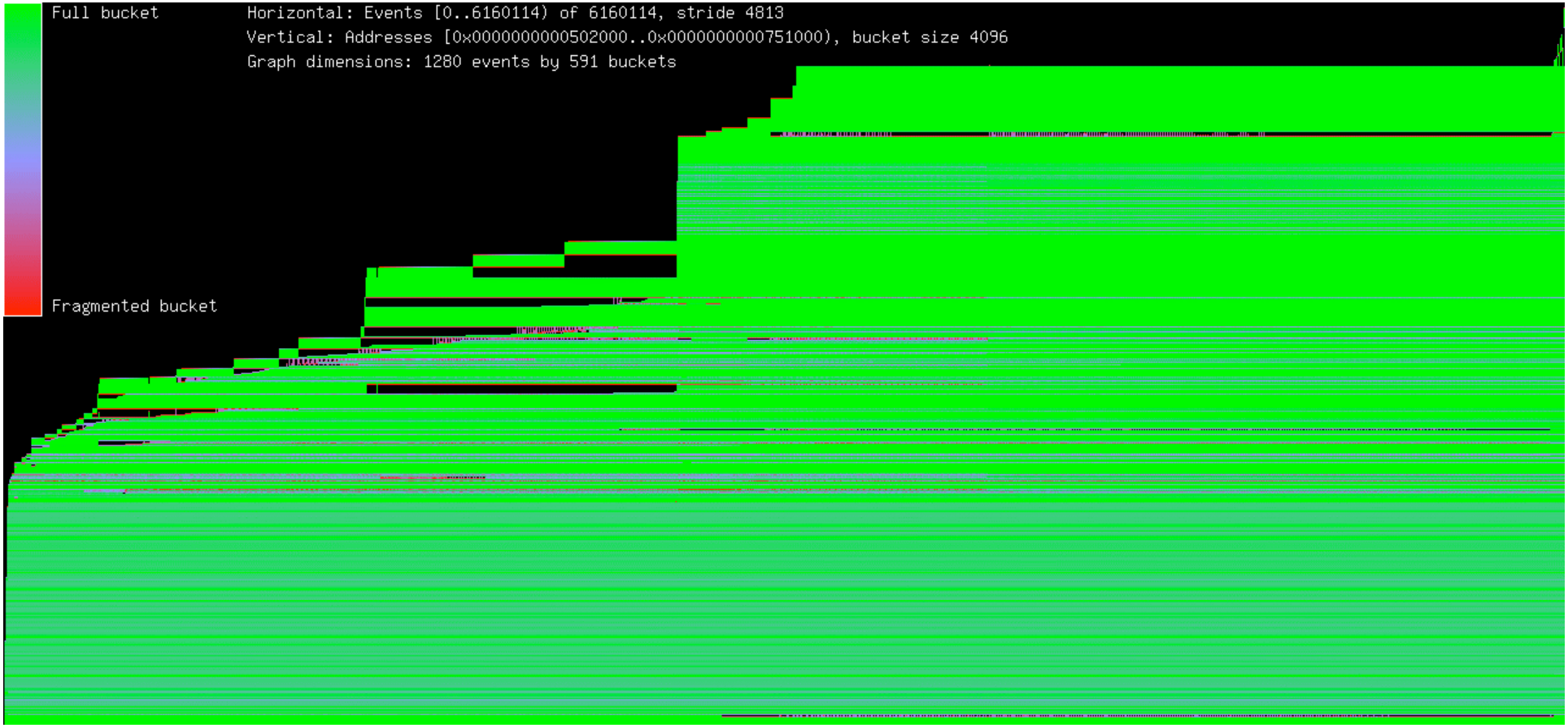
Single-threaded benchmarks



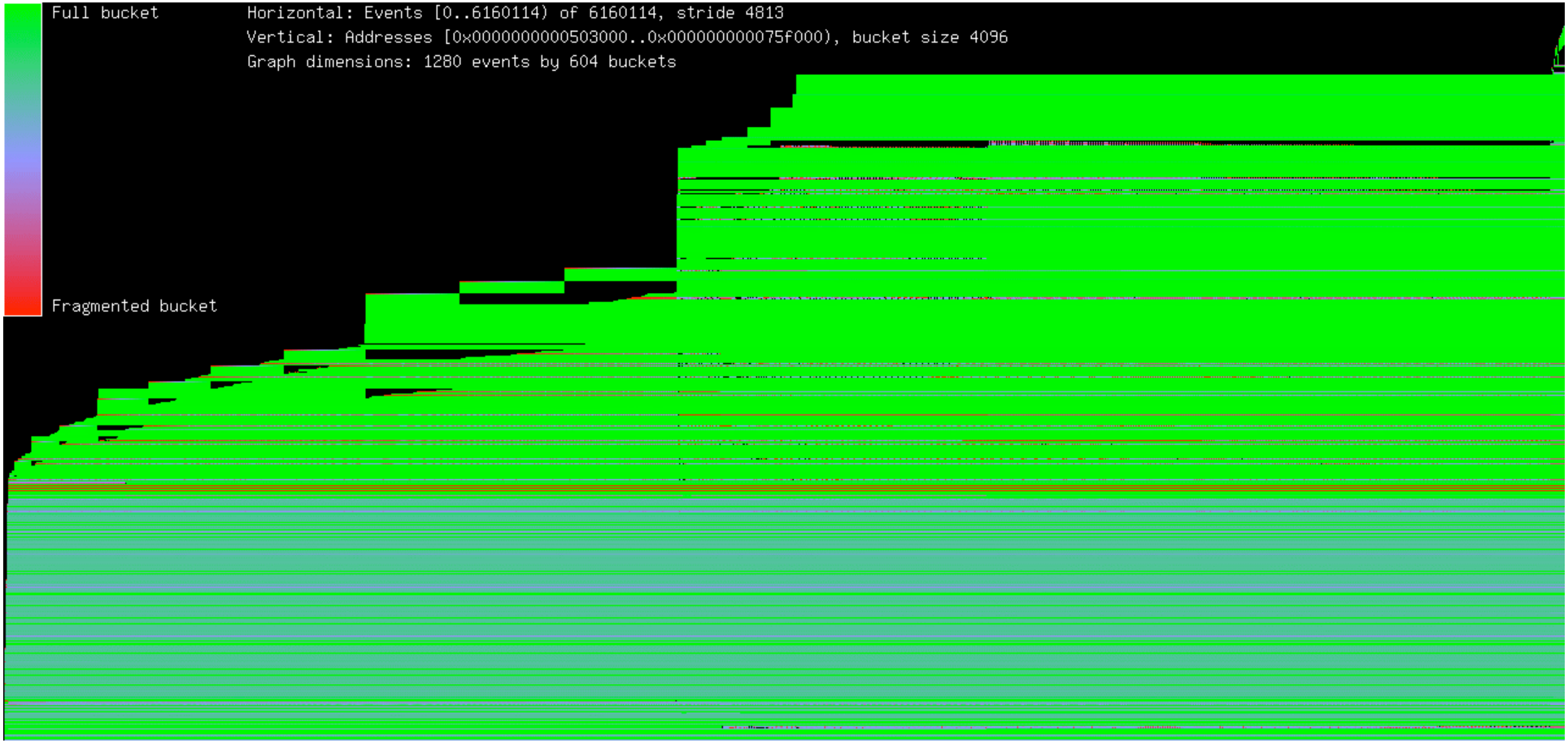
Fragmentation

- Quantitative comparison is difficult (requires narrow interpretation).
- Qualitative comparison is helpful, but also of limited usefulness.
- Different fragmentation patterns at various granularities (chunk, run, sub-run).

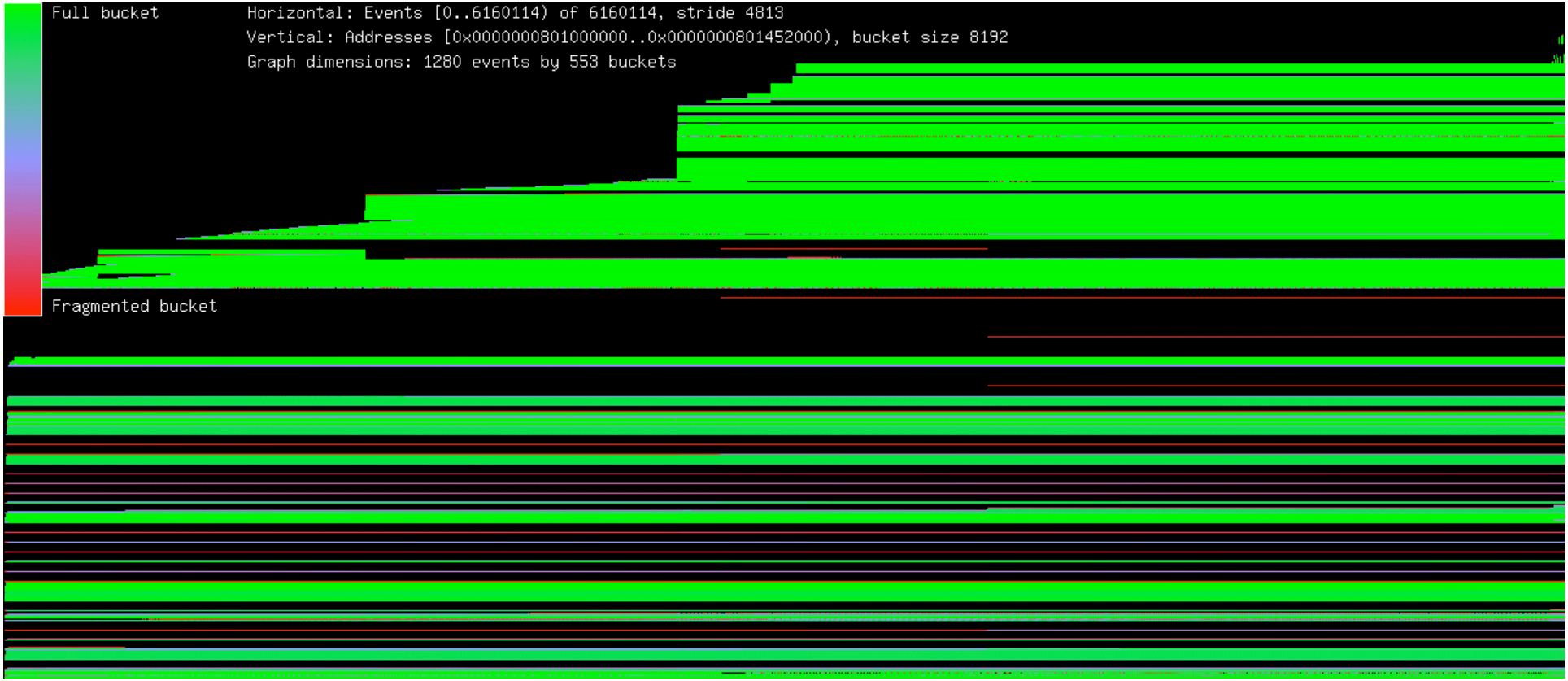
cca (dlmalloc)



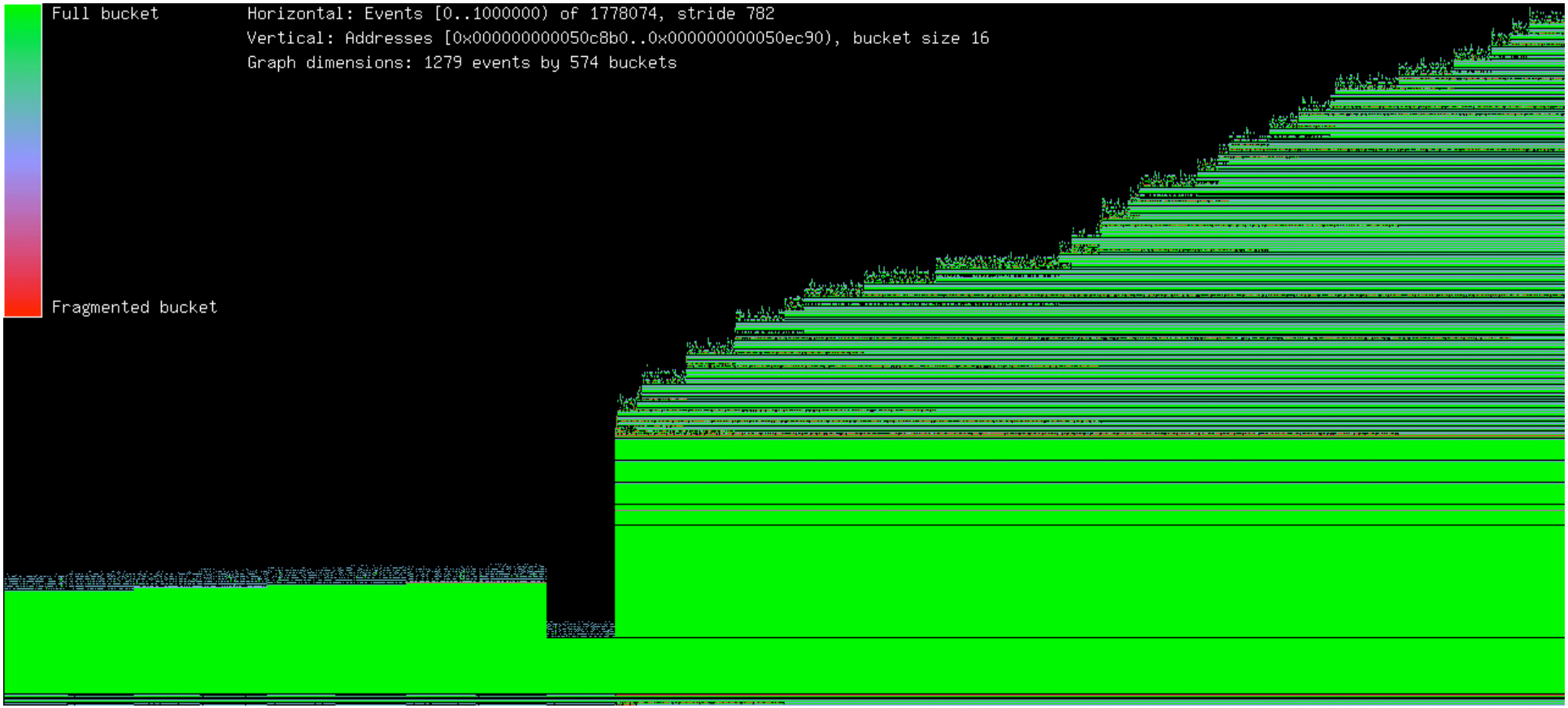
cca (phkmalloc)



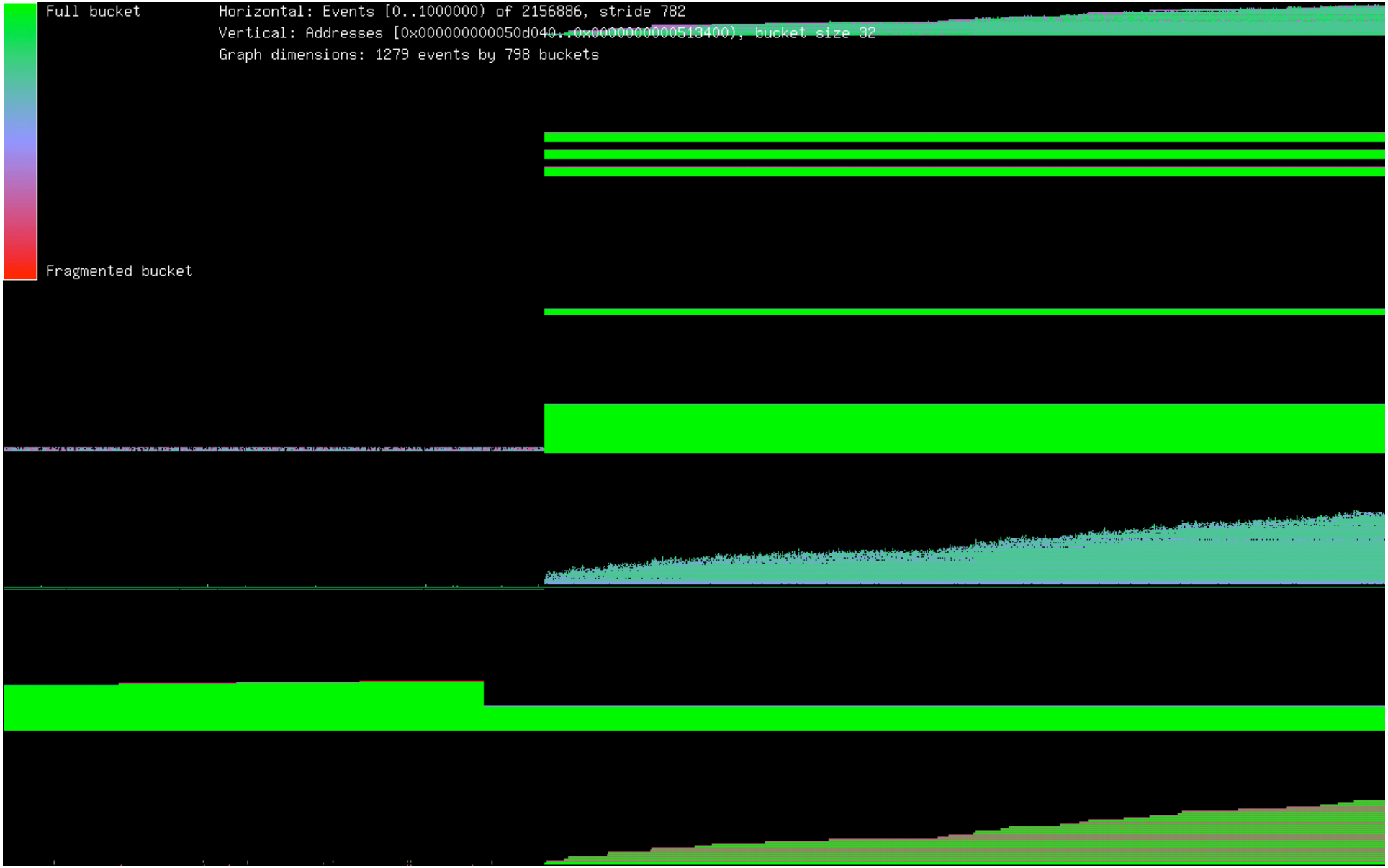
cca (jemalloc)



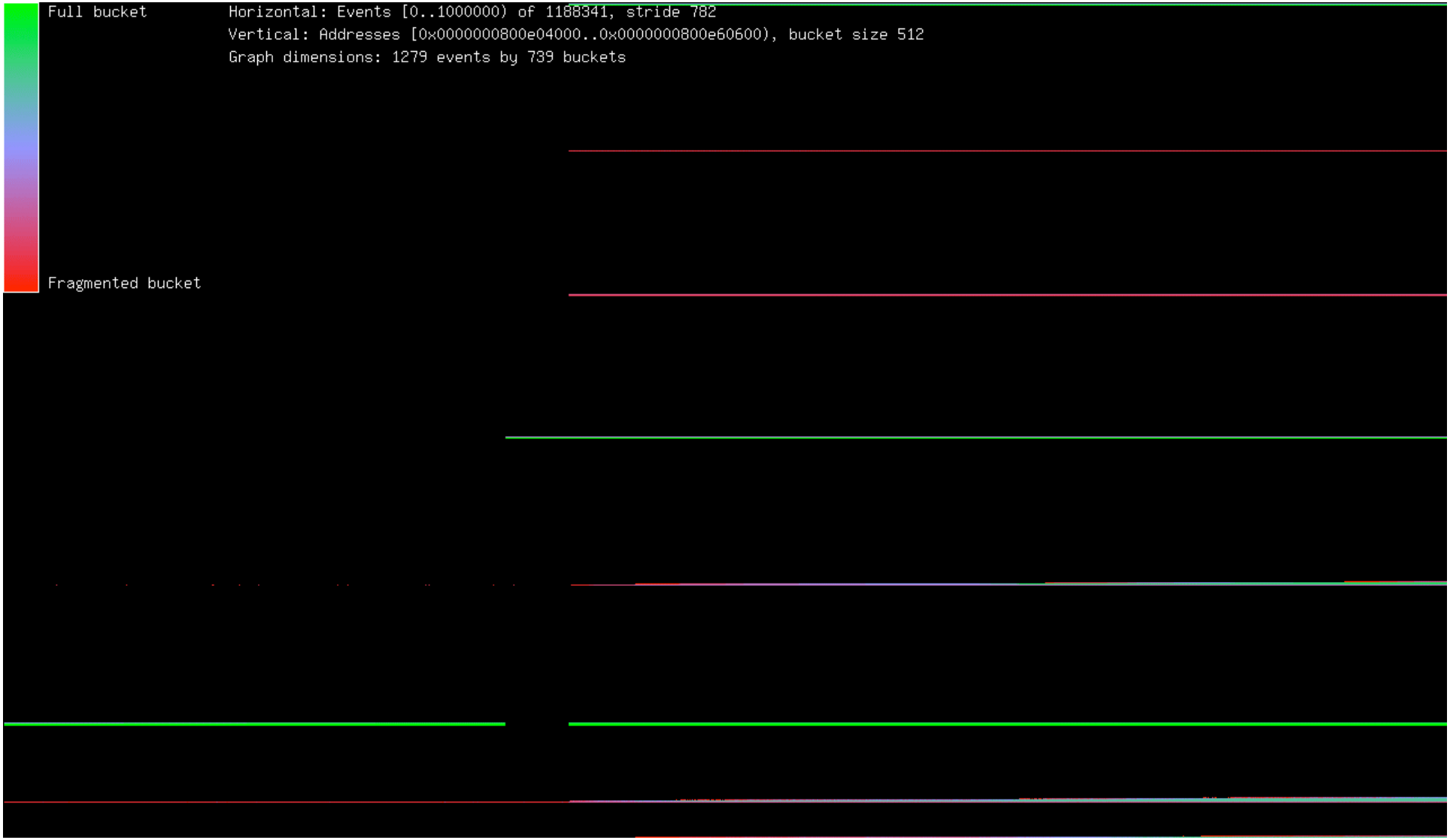
cfrac (dlmalloc)



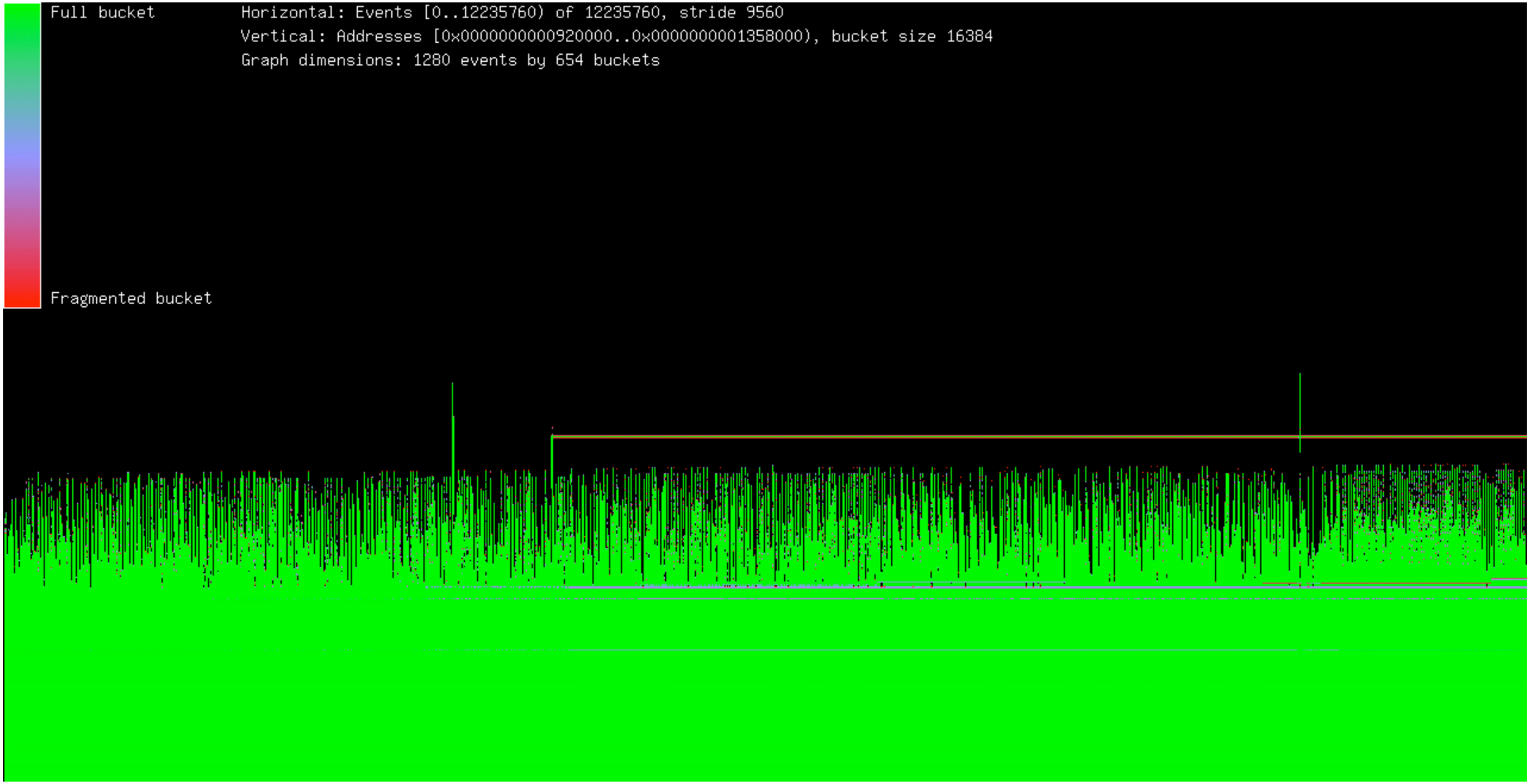
cfrac (phkmalloc)



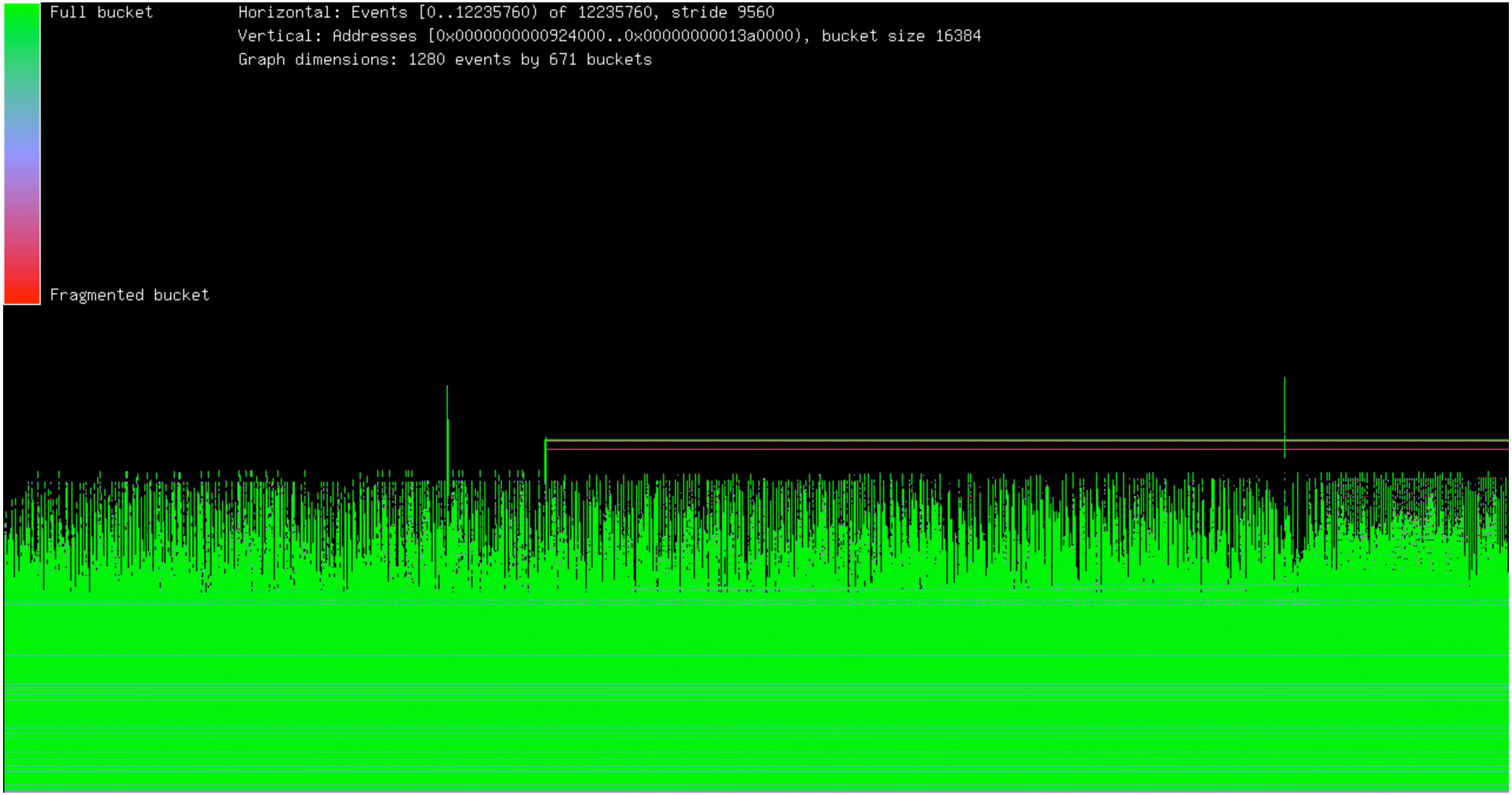
cfrac (jemalloc)



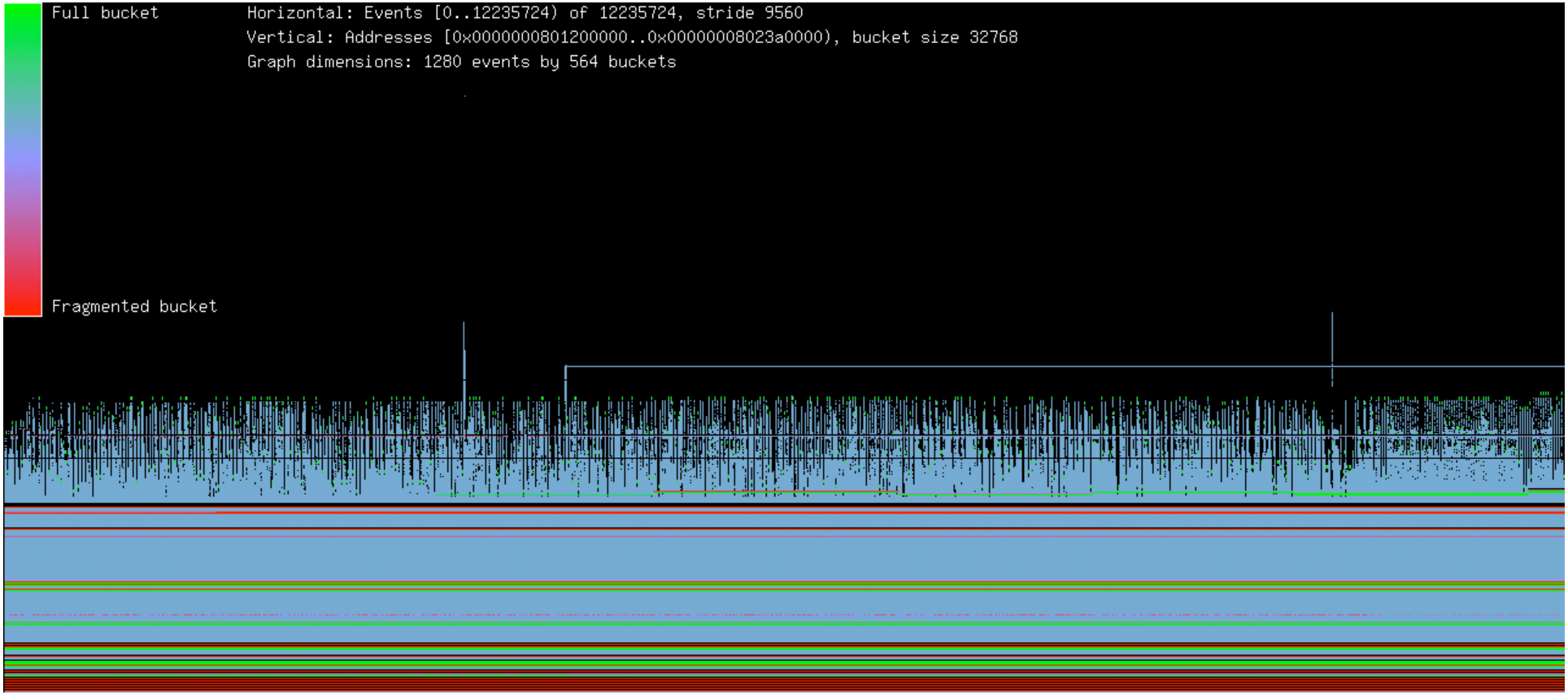
gs (dlmalloc)


Full bucket
Fragmented bucket
Horizontal: Events [0..12235760) of 12235760, stride 9560
Vertical: Addresses [0x0000000000920000..0x0000000001358000), bucket size 16384
Graph dimensions: 1280 events by 654 buckets

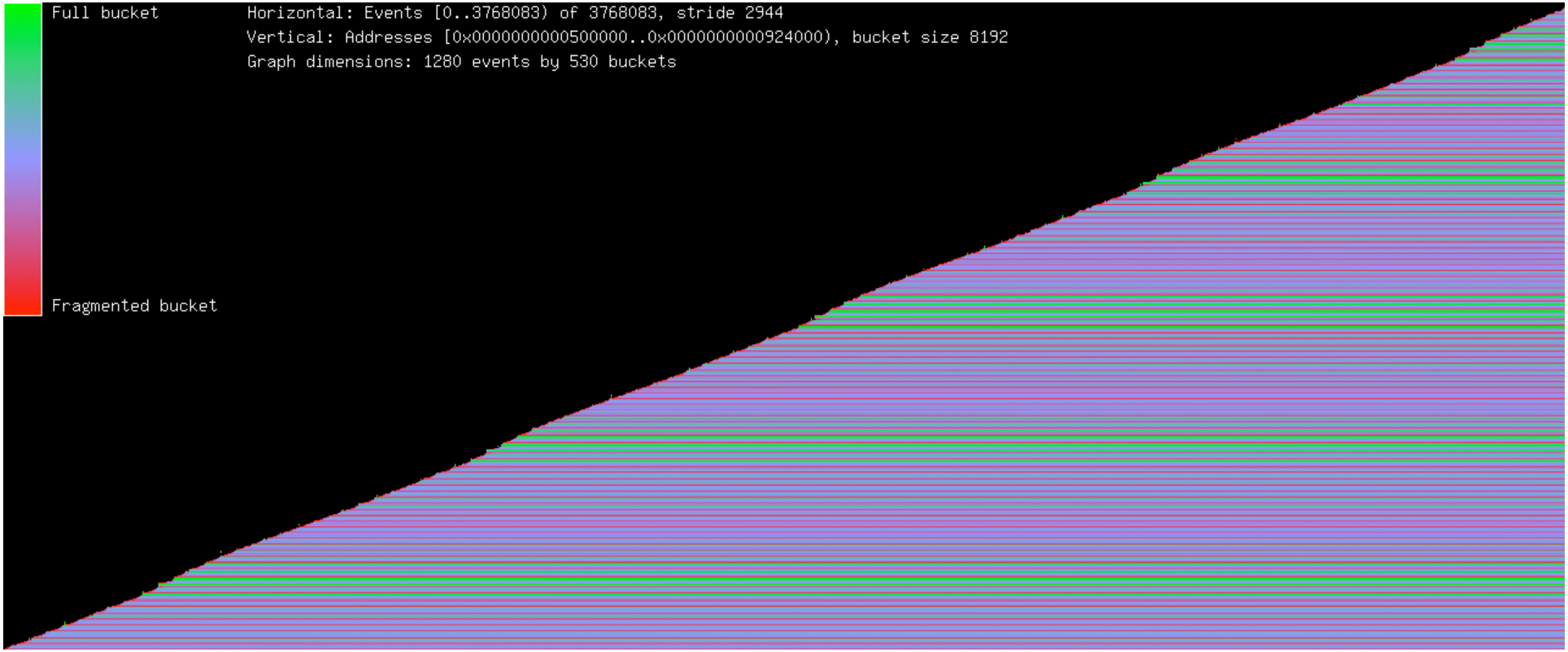
gs (phkmalloc)



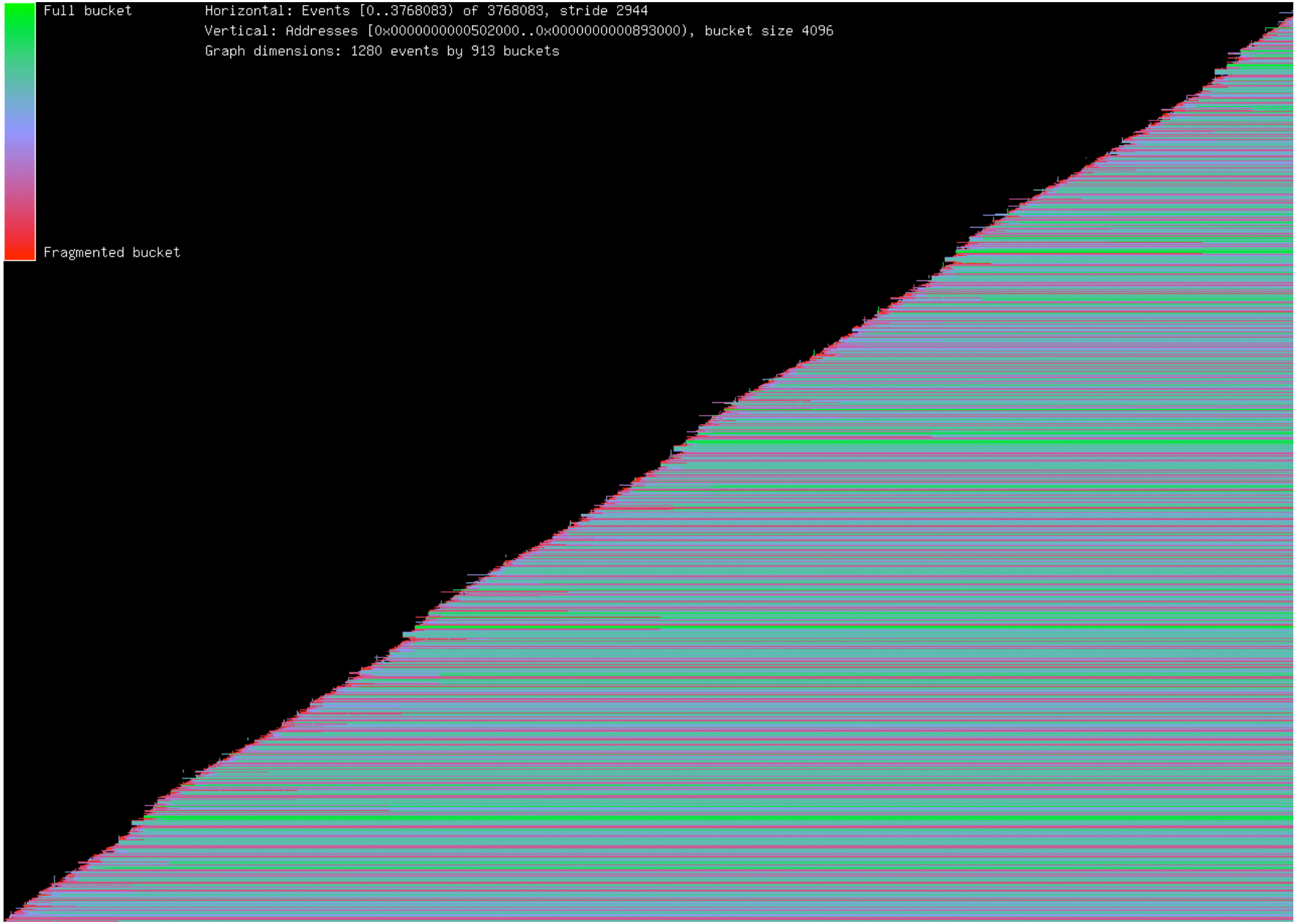
gs (jemalloc)


Full bucket
Horizontal: Events [0..12235724) of 12235724, stride 9560
Vertical: Addresses [0x0000000801200000..0x00000008023a0000), bucket size 32768
Graph dimensions: 1280 events by 564 buckets
Fragmented bucket

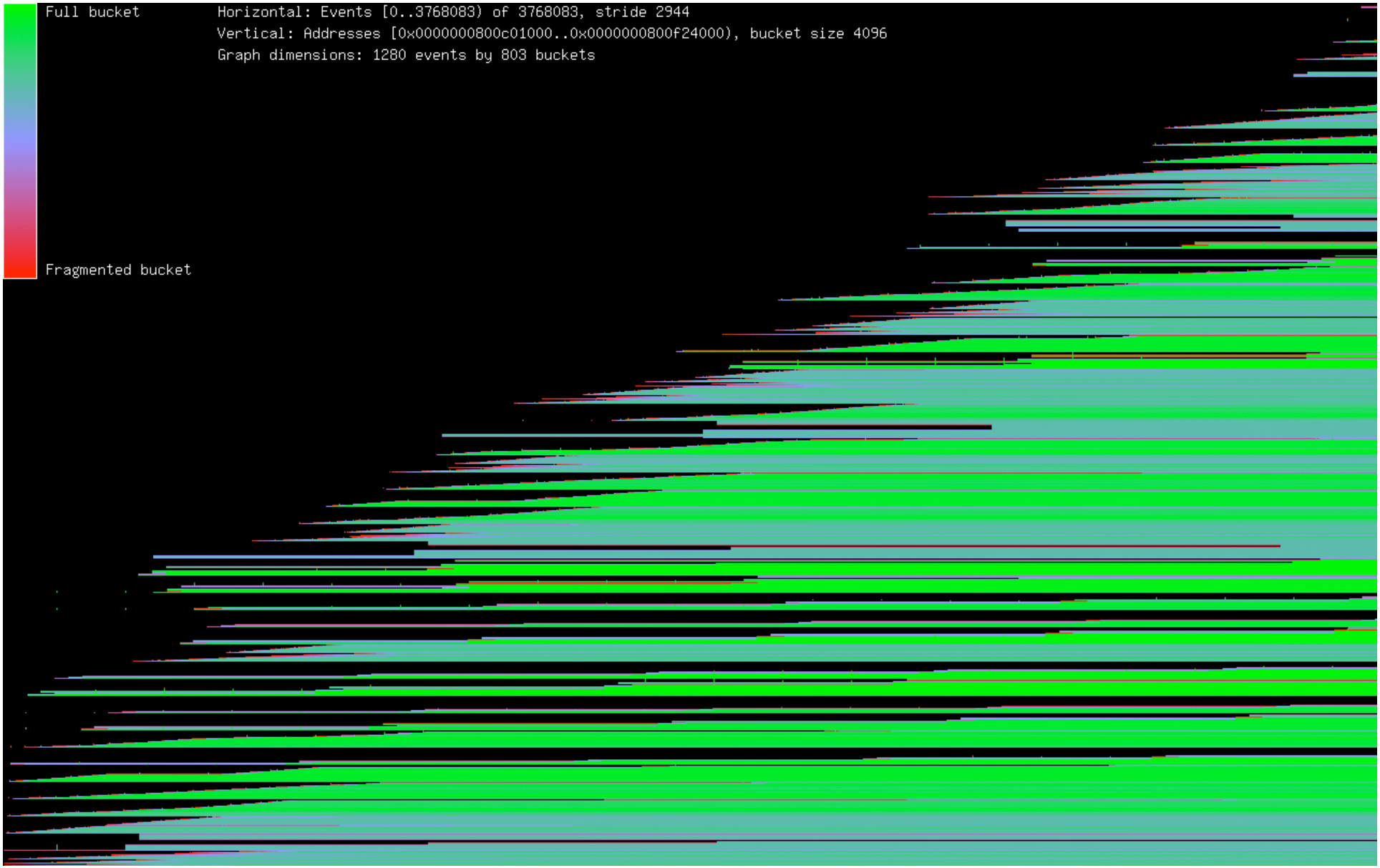
sh6bench (dlmalloc)



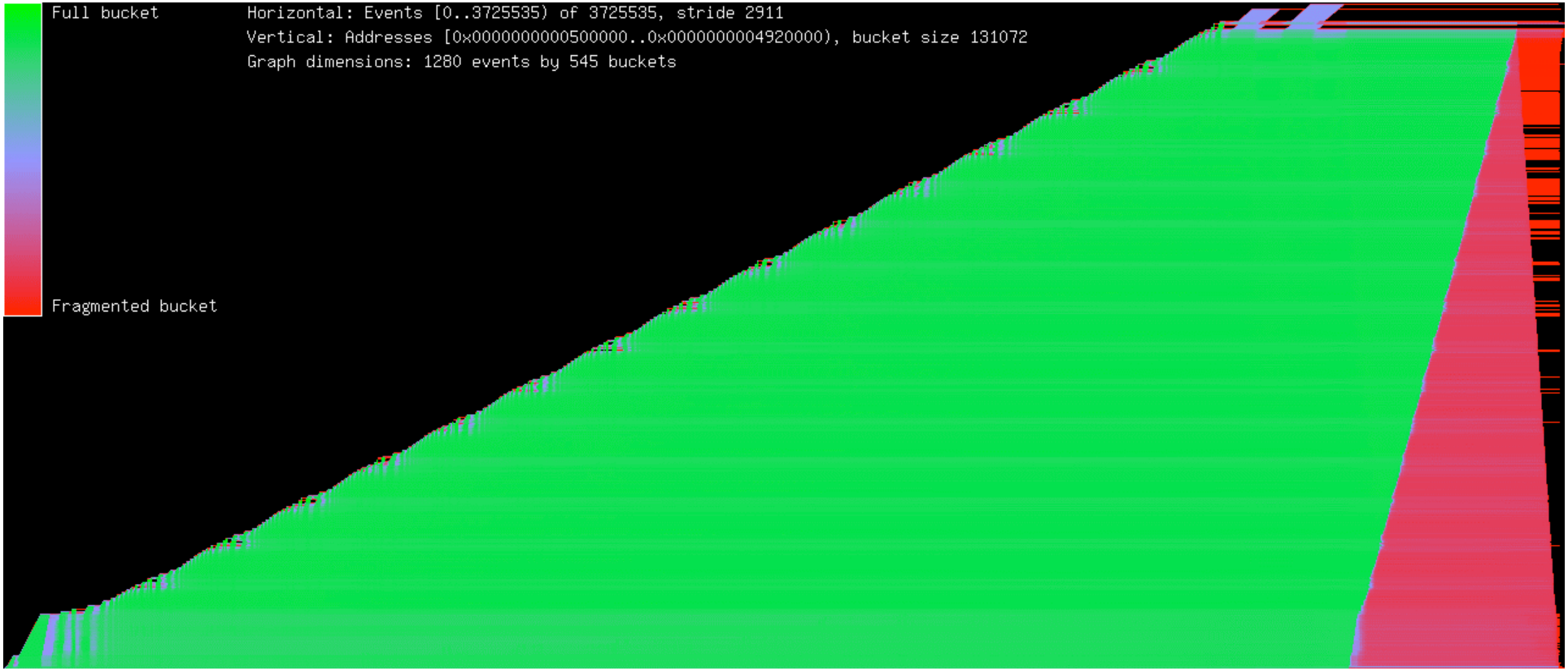
sh6bench (phkmalloc)



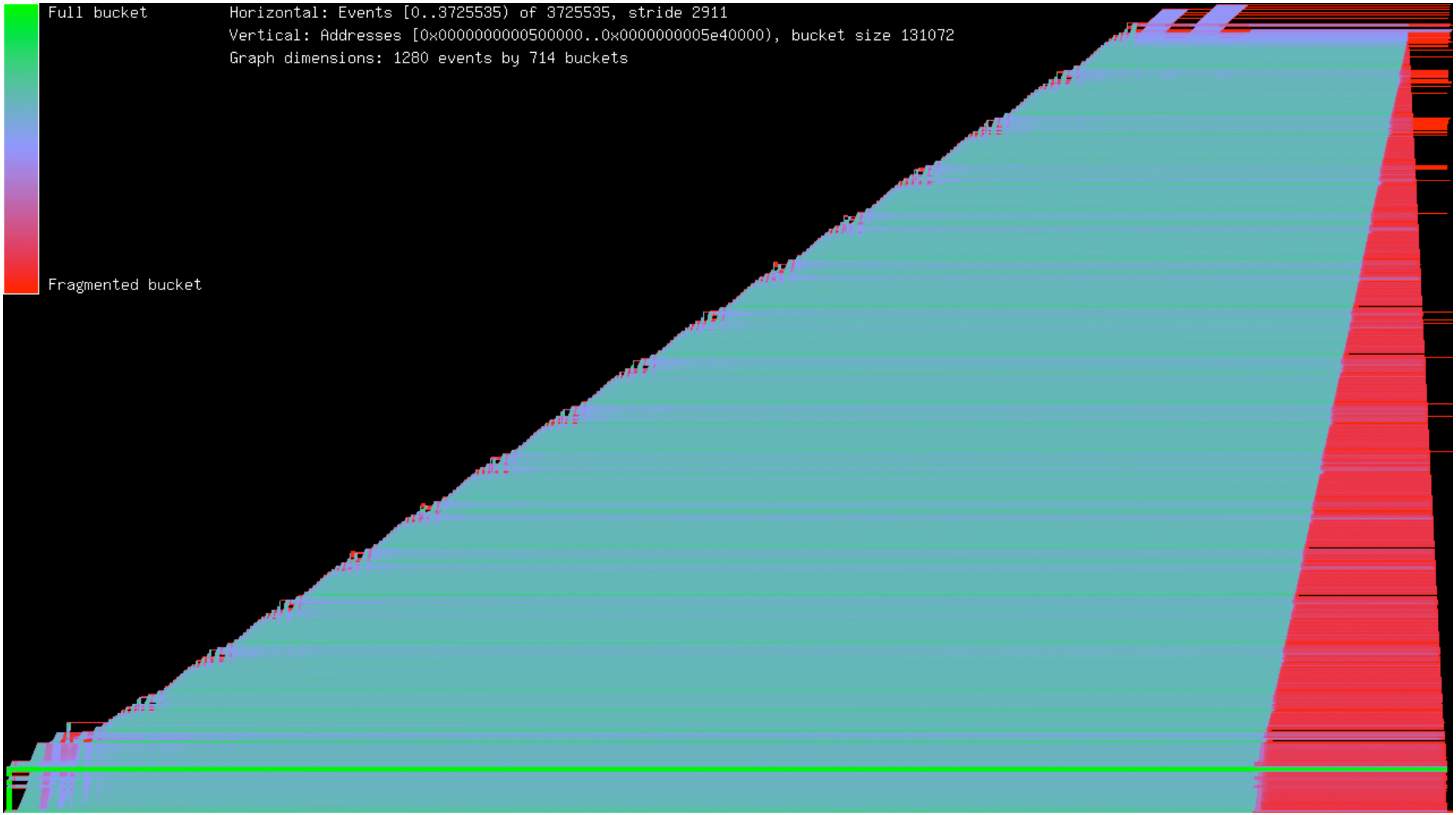
sh6bench (jemalloc)



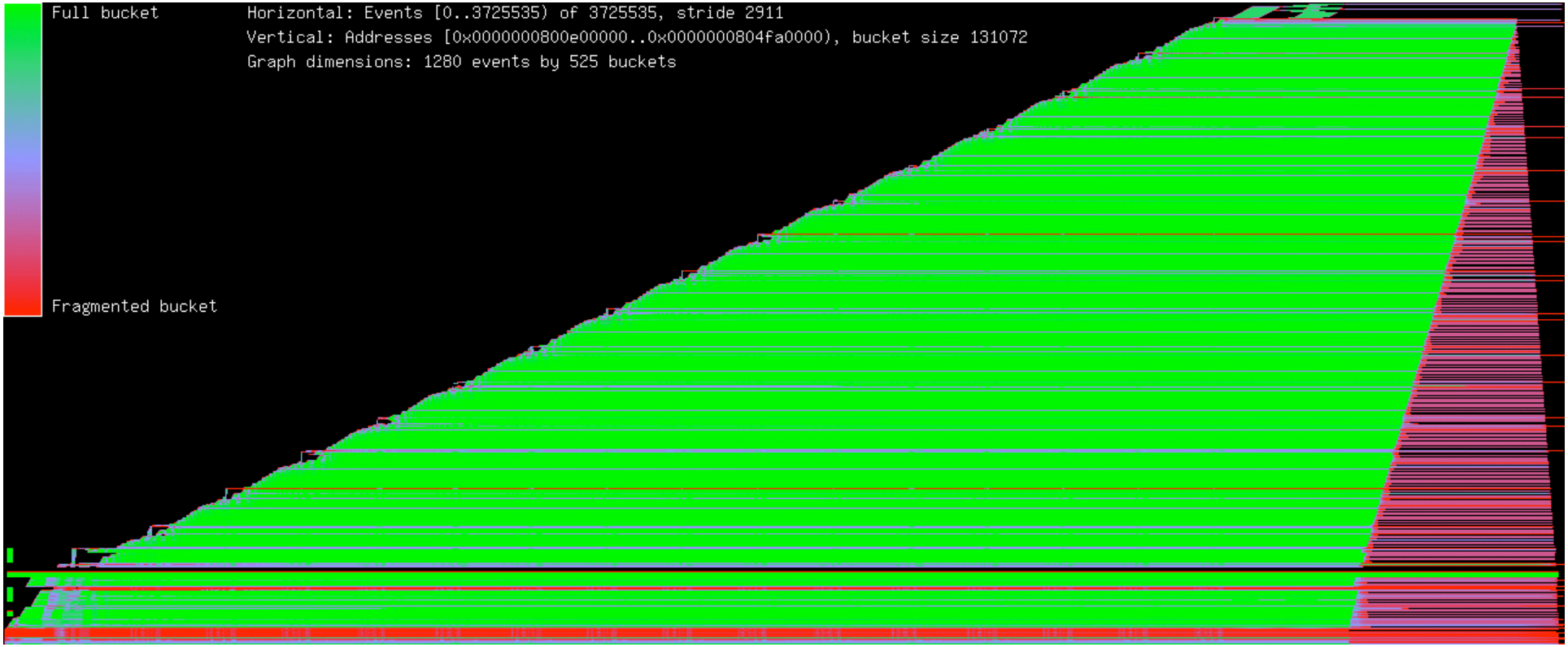
smlng (dlmalloc)



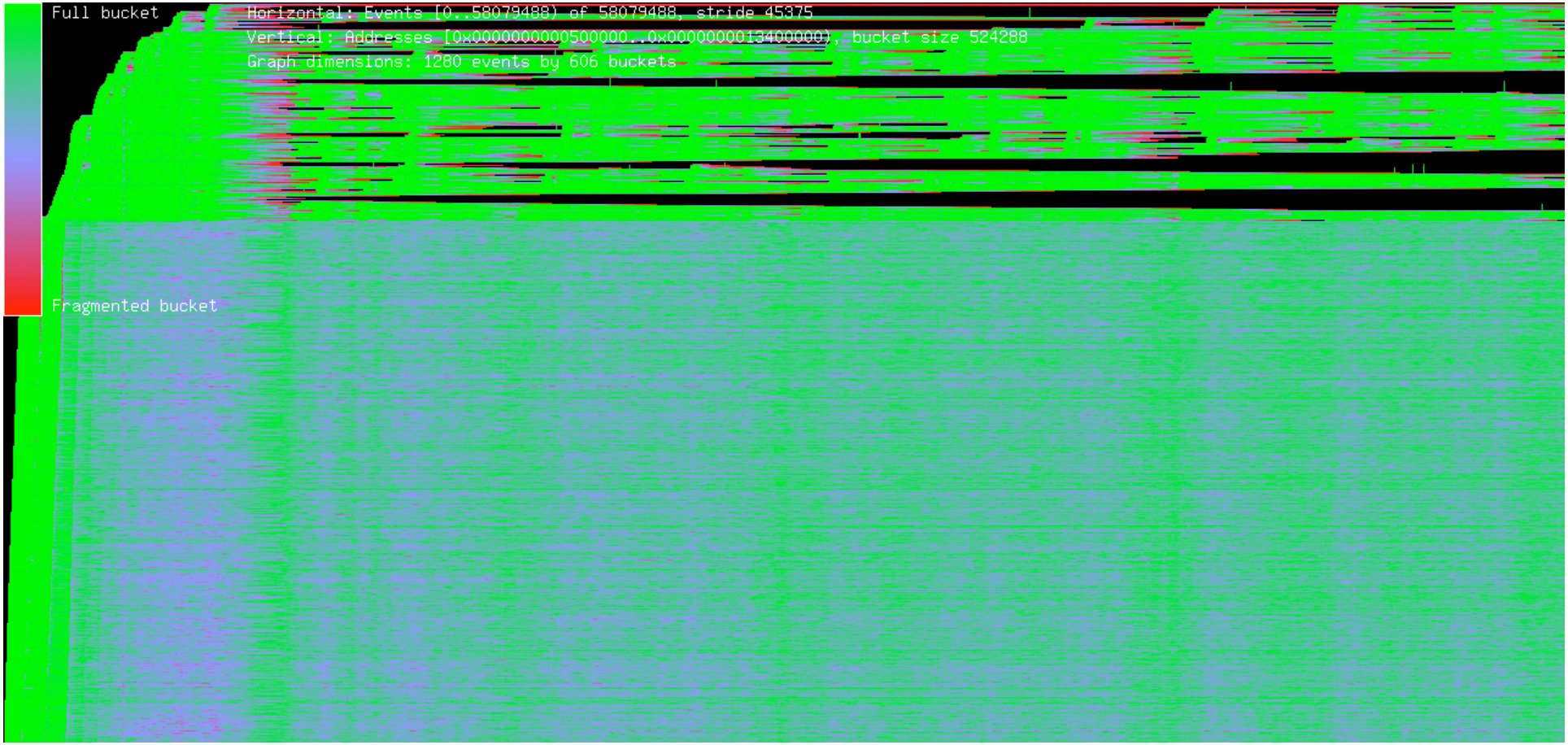
smlng (phkmalloc)



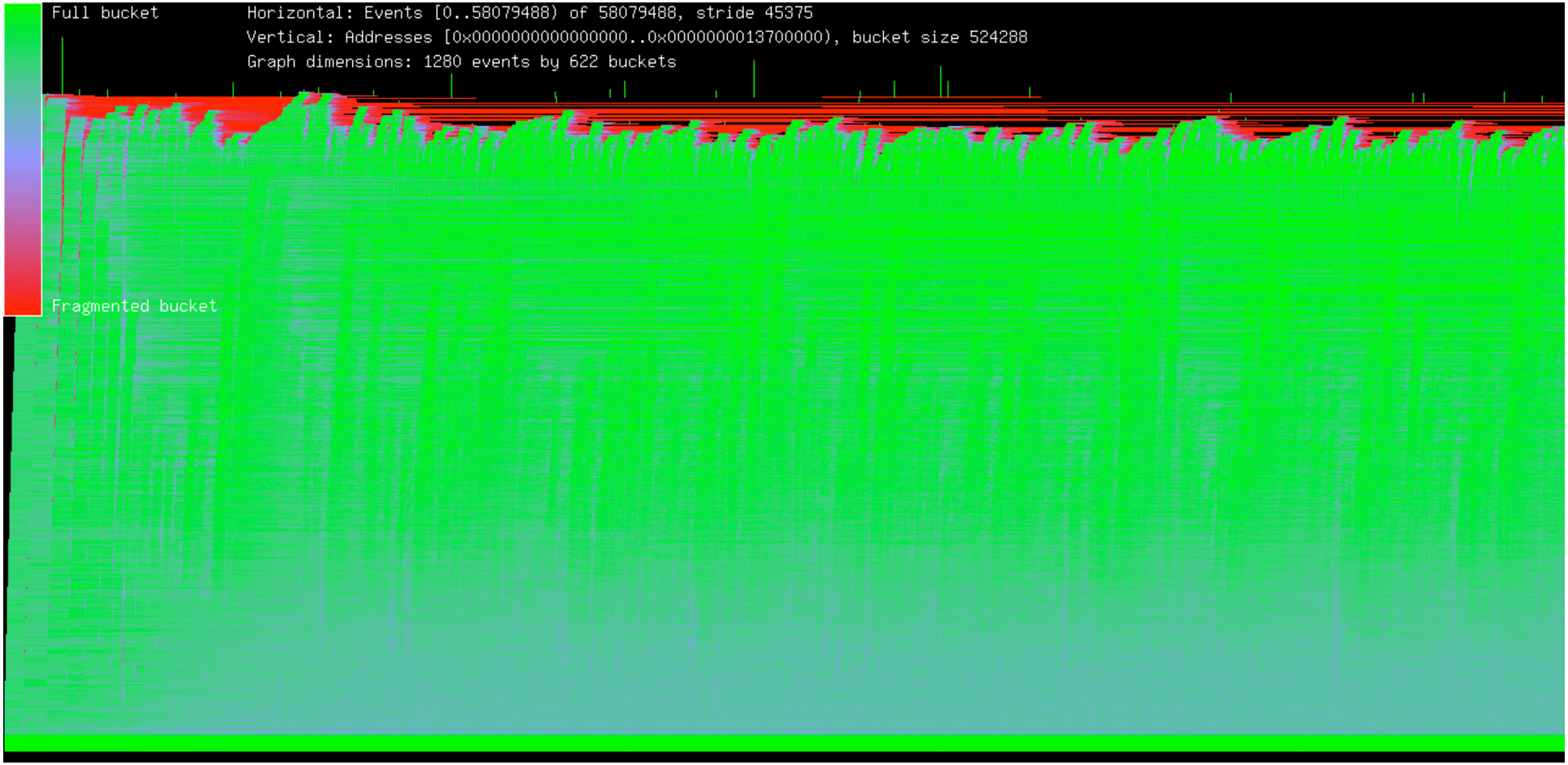
smlng (jemalloc)



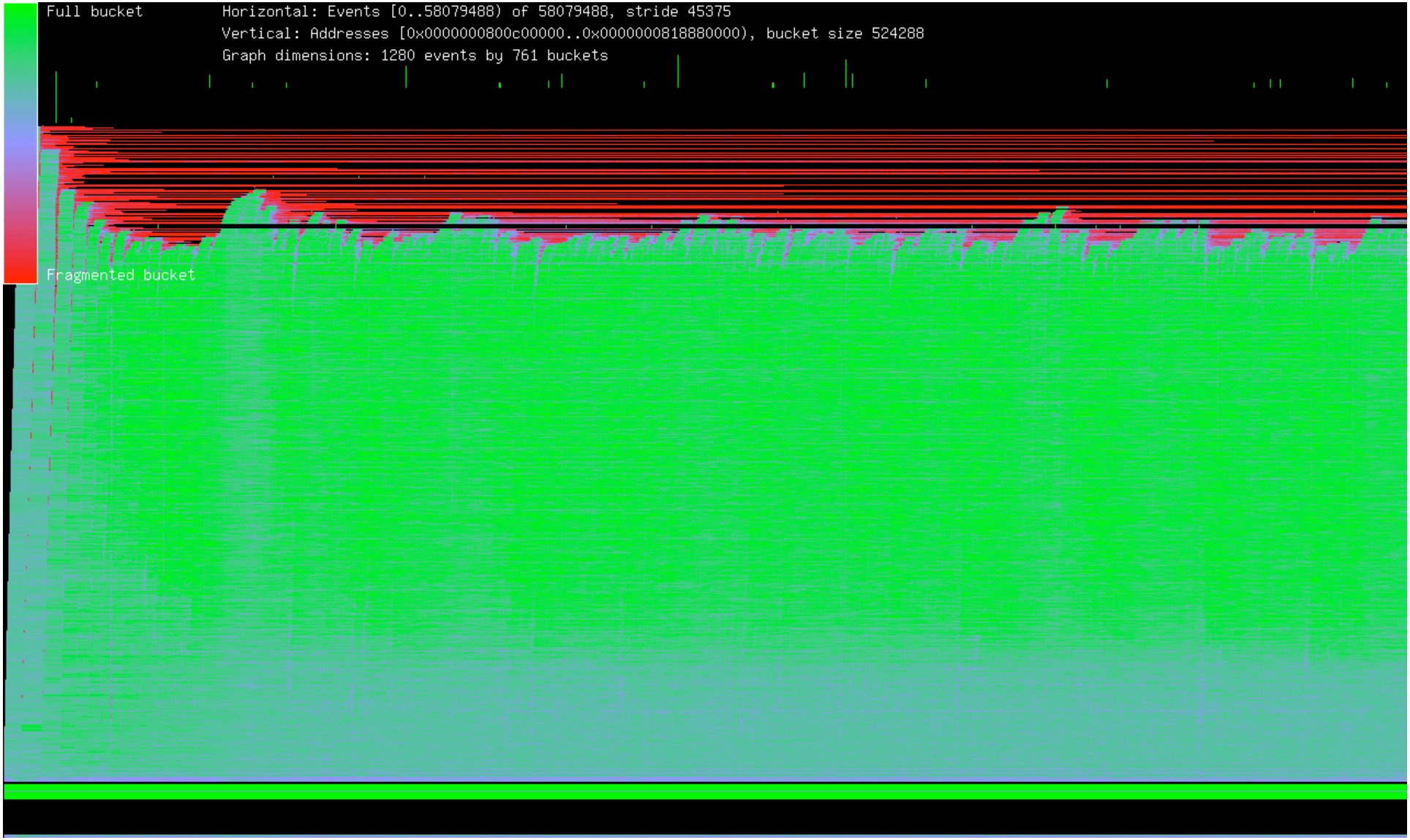
hummingbird (dlmalloc)



hummingbird (phkmalloc)

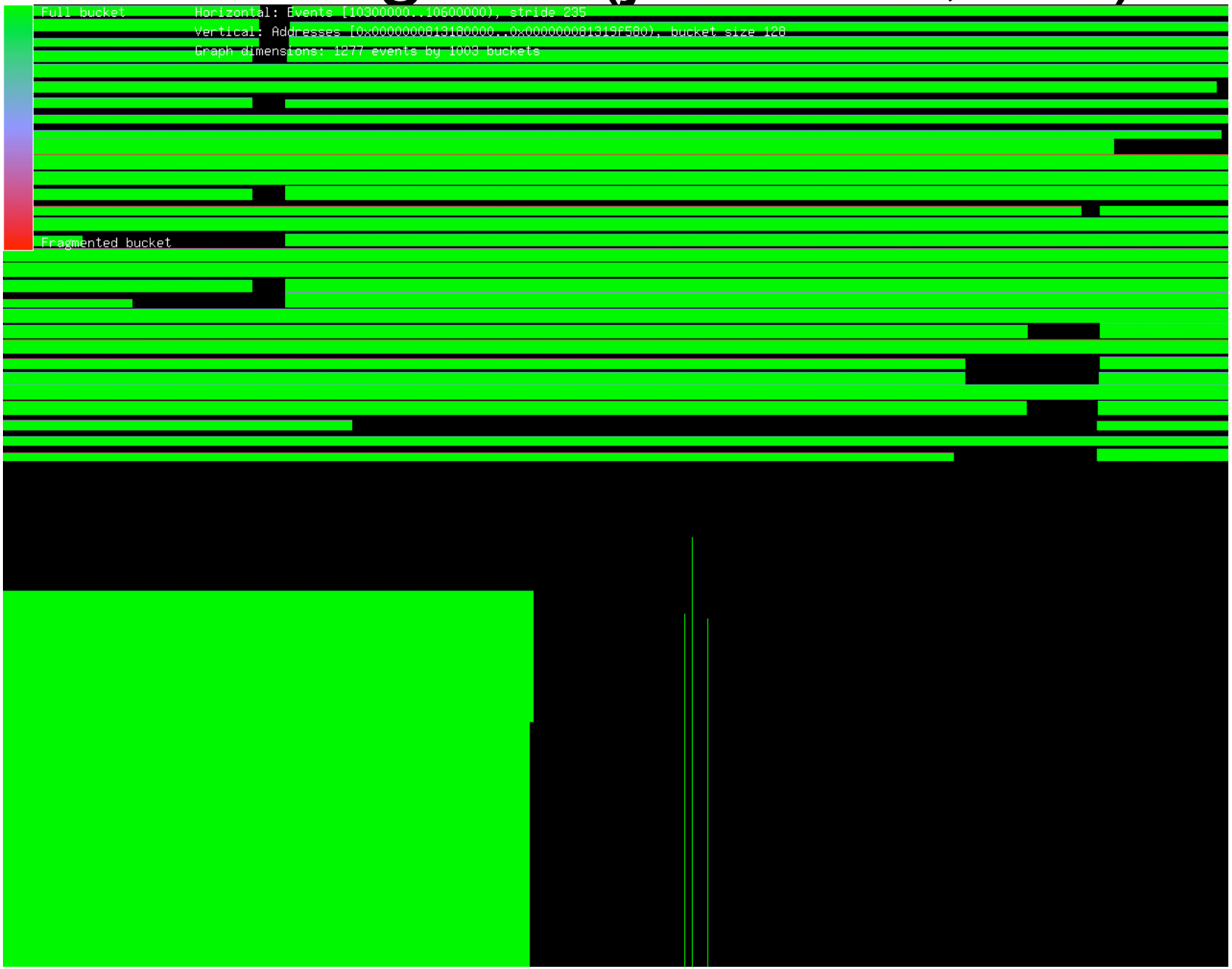


hummingbird (jemalloc, 1/3)



hummingbird (jemalloc, 2/3)

hummingbird (jemalloc, 3/3)



Disussion (performance)

- Microbenchmarks are particularly misleading for malloc.
- Tiny additions cause major performance loss (stats, division, etc.).
- Some apps do silly things (ex: incremental realloc()).
- What matters? Paging? Cache locality?

Discussion (features, 1/2)

- Should use multiple red-black trees for tracking of free runs, but sys/tree.h makes this prohibitively expensive.
- Debug features would be nice, but not in libc (valgrind!).
- Very (too?) configurable, via MALLOC_OPTIONS: {AHJKNPQSUVXZ}. {KNPQS} are new.

Discussion (features, 2/2)

- Allocator-specific APIs are a maintenance burden (config, stats, arenas).
- reallocf() shouldn't be in stdlib.h.
- Justifiable API?
  - void *malloc_np(size_t *size);
  - void *calloc_np(size_t *size);
  - void *memalign_np(size_t *size, size_t alignment);
  - void *realloc_np(void *ptr, size_t *size, size_t *oldsize);
  - size_t free_np(void *ptr);

Acknowledgements

- Testing:
  - Kris Kennaway (many bug reports, benchmarks)
  - FreeBSD community
- Financial:
  - FreeBSD Foundation (travel to BSDcan)
  - Mike Tancsa (hardware)
- Miscellaneous:
  - Robert Watson (remote machine access)
  - Peter Wemm (optimization)
  - Poul-Henning Kamp (review)
  - Aniruddha Bohra (hummingbird traces)
  - Rob Braun (instigator)

http://people.freebsd.org/~jasone/jemalloc/

Also, read the paper!