Reducing Active False Sharing in TCMalloc
What is False Sharing?
Why Is This Bad?

False sharing slows down multi-core processing
Different Forms of False Sharing

Active False Sharing

Passive False Sharing

Any others?
A Multi-Thread Allocator?

Without a multi-thread allocator:

One global lock per program that each thread running in that program shares

This is required so that allocations and deallocations don't conflict
TCMalloc

Performs (most) allocations for multiple threads lock free

This is achieved by having a Thread Cache of allocatable memory

Global locks are required for large objects or for expanding/reducing the thread cache by exchanging memory with the central heap
TCMalloc - Implementation

A thread cache is a list of free lists

Each free list corresponds to a different size-class

TCMalloc is lazily initialised, so the free lists begin empty and must be filled from the central heap as needed
Other Implementations

TCMalloc differs from Hoard and Scallop.
TCMalloc carves memory up at the central heap and passes around object size chunks.
Hoard and Scallop pass around raw memory and the thread carves the memory up upon allocation.
Benchmarks

Cache-thrash       -       Active False Sharing
Cache-scratch      -       Passive False Sharing
Larson            -       Server load approximation
Linux Scalability -       Scalability
SPOILERS!
Cache-thrash on Rat Architecture with Object Size 8

- glib
- Hoard
- TCMalloc
- edited-TCM
- scalloc

(time (s))

(thread count)
Linux-scalability on Rat Architecture with Object Size 8

- glib
- Hoard
- TCMalloc
- edited-TCM
- scalloc
Cache-Thrash and TCMalloc

TCMalloc hands object sized chunks around

Because of this it has no control on whether or not a cache-lines are divided between thread caches

With 8 threads and 8 byte objects an instrumented benchmark utilised only one cache-line
Contributing Factors

When a free list in a thread cache is small, expansion of the free list happens one object at a time.

When a free list is larger than what was needed, the excess allocatable memory is returned to the central heap.

No order is maintained resulting in a shuffling of objects in the central heap.
So... what to do?

Is desirable for objects to leave central heap as a cache-line sized chunk

I edited the code such that a cache-line sized group of objects would be retrieved
Does it work?

Yes.
No.
Kind of?
Future Work - Benchmarking

Do benchmarks more rigorously
Across more types of architectures
Identify Google's typical workload and test that
Future Work - Implementation

Need some mechanism which will ensure memory is passed around in cache-line blocks.

If central heap is cache-line contiguous, my edit ensures memory leaves central heap as contiguous memory.

Need a mechanism which returns memory to central heap as cache-lines.
Potential Solution

Have something that sits in between thread caches and central heap

When a thread-cache returns memory to the central heap it is first gathered in this intermediate stage

This stage collects objects until a full cache-line sized contiguous chunk is assembled and only then returns that chunk to central heap

There are obviously trade-offs with this approach but it may be worth it.
Or ...

Use Scallop for small objects and TCMalloc for large objects
References

Hoard: a scalable memory allocator for multithreaded applications; Berger, Emery D (2000)


TCMalloc documentation; Ghemawat S. and Menage P.

THANK YOU!

Questions... ?