

# Memory Management for High-Performance Applications

**Emery Berger**

Univ. of Massachusetts, Amherst

November, 2000

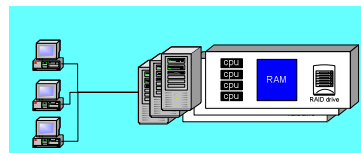
Note: Some material has been added for  
the classroom setting



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

## High-Performance Applications

- Web servers, search engines, scientific codes
- C or C++
- Run on one or a cluster of server boxes
- Needs support at every level



software
compiler
runtime system
operating system
hardware



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

## New Applications, Old Memory Managers

- Applications and hardware have changed
  - Multiprocessors now commonplace
  - Object-oriented, multithreaded
  - ⇒ Increased pressure on memory manager  
(malloc, free)



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

## Memory Management 101

- **Memory allocators:**
  - Accept requests for memory and return virtual address to a block of memory of the requested size
  - Accept requests to **free** memory, which allows virtual memory to be reused
- **The Heap:**
  - The pool of unused memory



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

## New Applications, Old Memory Managers (cont)

- Applications and hardware have changed
  - Multiprocessors now commonplace
  - Object-oriented, multithreaded
  - ⇒ Increased pressure on memory manager  
(malloc, free)
- But memory managers have *not* kept up
  - Inadequate support for modern applications



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

## Memory Allocators

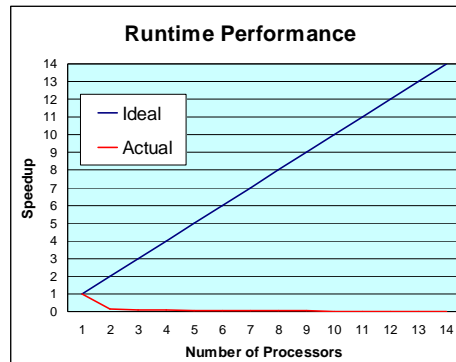
- Input:
  - Size of requested chunk of virtual memory
- Output:
  - Memory address of allocated virtual memory
- The Heap: (today's definition)
  - Structure for managing allocated and unallocated memory



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

## Current Memory Managers Limit Scalability

- As we add processors, program **slows down**
- Caused by *heap* contention



Larson server benchmark on 14-processor Sun



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

## The Problem

- Current memory managers **inadequate** for high-performance applications on modern architectures
- Limit scalability & application performance



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

## Implementing Memory Managers

- Memory managers must be
    - Space efficient
    - **Very fast**
  - Heavily-optimized C code
    - Hand-unrolled loops
    - Macros
    - Monolithic functions
- ⇒ Hard to write, reuse, or extend



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

## Real Code: DLmalloc 2.7.2

```
#define chunksize(p)      ((p)->size & ~(SIZE_BITS))
#define next_chunk(p) ((mchunkptr) ((char*)(p) + ((p)->size & ~PREV_INUSE)))
#define prev_chunk(p) ((mchunkptr) ((char*)(p) - ((p)->prev_size)))
#define chunk_at_offset(p, s) ((mchunkptr) (((char*)(p)) + (s)))
#define inuse(p)          (((mchunkptr)((char*)(p)) + ((p)->size & ~PREV_INUSE)) ->size) & PREV_INUSE
#define set_inuse(p)      ((mchunkptr)((char*)(p)) + ((p)->size & ~PREV_INUSE)) ->size |= PREV_INUSE
#define clear_inuse(p)    ((mchunkptr)((char*)(p)) + ((p)->size & ~PREV_INUSE)) ->size &= ~(PREV_INUSE)
#define inuse_bit_at_offset(p, s) (((mchunkptr)((char*)(p)) + (s)) ->size & PREV_INUSE)
#define set_inuse_bit_at_offset(p, s) (((mchunkptr)((char*)(p)) + (s)) ->size |= PREV_INUSE)
#define MALLOC_ZERO(charp, nbytes)
do {
    INTERNAL_SIZE_T* mzp = (INTERNAL_SIZE_T*)(charp);
    CHUNK_SIZE_T mctmp = (nbytes)/sizeof(INTERNAL_SIZE_T);
    long mcn;
    if (mctmp < 8) mcn = 0; else { mcn = (mctmp-1)/8; mctmp %= 8; }
    switch (mctmp) {
        case 0: for(;;) { *mzp++ = 0; }
        case 7: *mzp++ = 0;
        case 6: *mzp++ = 0;
        case 5: *mzp++ = 0;
        case 4: *mzp++ = 0;
        case 3: *mzp++ = 0;
        case 2: *mzp++ = 0;
        case 1: *mzp++ = 0; if (mcn <= 0) break; mcn--; }
    } while(0)
```



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science



## This Talk

- Building memory managers
  - *Heap Layers* framework
- **Problems with current memory managers**
  - Contention, false sharing, space
- **Solution: provably scalable memory manager**
  - *Hoard*
- Extended memory manager for servers
  - *Reap*



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science



## Problems with General-Purpose Memory Managers

- Previous work for multiprocessors
  - Concurrent single heap [Bigler *et al.* 85, Johnson 91, Iyengar 92]
    - Impractical
  - Multiple heaps [Larson 98, Gloger 99]
- Reduce contention **but** cause other problems:
  - P-fold or even **unbounded increase** in space
  - **Allocator-induced false sharing**





UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

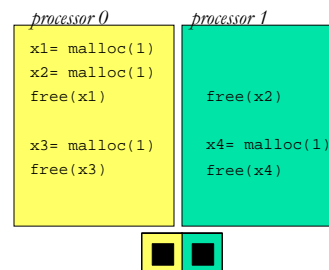
## Multiple Heap Allocator: Pure Private Heaps

- One heap per processor:

- malloc gets memory from its local heap
- free puts memory on its local heap

- STL, Cilk, *ad hoc*

Key:  = in use, processor 0  
 = free, on heap 1



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

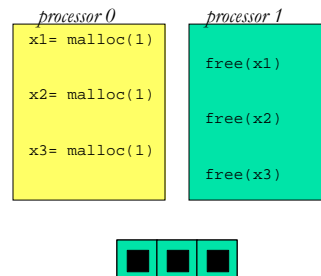
## Problem: Unbounded Memory Consumption

- Producer-consumer:

- Processor 0 allocates
- Processor 1 frees

- Unbounded memory consumption

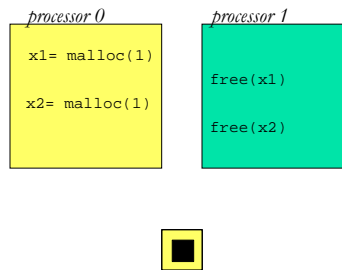
- Crash!



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

## Multiple Heap Allocator: Private Heaps with Ownership

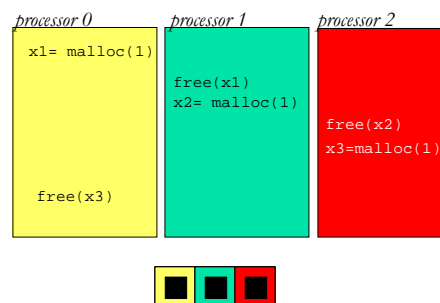
- `free` returns memory to *original* heap
- Bounded memory consumption
- No crash!
- “Ptmalloc” (Linux), LKmalloc



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

## Problem: P-fold Memory Blowup

- Occurs in practice
- Round-robin producer-consumer
  - processor  $i \bmod P$  allocates
  - processor  $(i+1) \bmod P$  frees
- Footprint = 1 (2GB), but space = 3 (6GB)
  - Exceeds 32-bit address space: Crash!



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

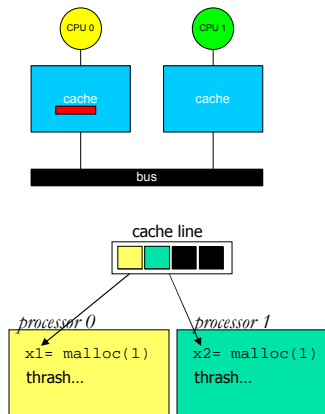


## Problem: Allocator-Induced False Sharing

### ■ False sharing

- Non-shared objects on same cache line
- Bane of parallel applications
- Extensively studied

- All these allocators cause false sharing!



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

## So What Do We Do Now?

- Where do we put free memory?
  - on central heap: ⇒ Heap contention
  - on our own heap: ⇒ Unbounded memory consumption  
(*pure private heaps*)
  - on the original heap: ⇒ P-fold blowup  
(*private heaps with ownership*)
- How do we avoid false sharing?



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science



## Overview

- Building memory managers
  - *Heap Layers* framework
- Problems with memory managers
  - Contention, space, false sharing
- **Solution: provably scalable allocator**
  - *Hoard* [ASPLOS 2000]
- Extended memory manager for servers
  - *Reap*



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science



## Hoard: Key Insights

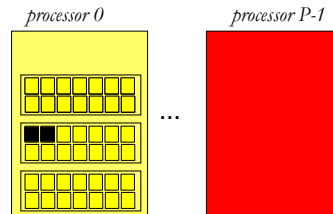
- Bound local memory consumption
  - Explicitly track utilization
  - Move free memory to a *global heap*
    - ⇒ Provably bounds memory consumption
- Manage memory in large chunks
  - ⇒ Avoids false sharing
  - ⇒ Reduces heap contention



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

## Overview of Hoard

- Manage memory in *heap blocks*
  - Page-sized
  - Avoids false sharing
- Allocate from local heap block
  - Avoids heap contention
- Low utilization ( $U < \frac{2}{3}A$ )
  - ⇒ Move heap block to global heap
    - Avoids space blowup



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

## Summary of Analytical Results

- Space consumption: near optimal worst-case
  - Optimal:  $O(n \log M/m)$   $\{P \ll n\}$  [Robson 77]
  - Hoard:  $O(n \log M/m + P)$
  - Private heaps with ownership:  $O(P n \log M/m)$
- Provably low synchronization

$n$  = memory *required*  
 $M$  = biggest object size  
 $m$  = smallest object size  
 $P$  = processors



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

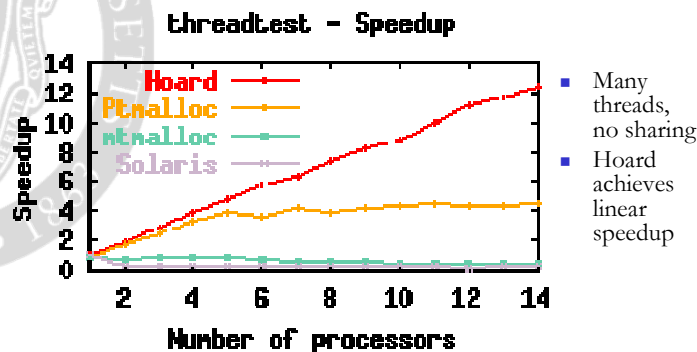
## Empirical Results

- Measure runtime on 14-processor Sun
  - Allocators
    - Solaris (system allocator)
    - *Ptmalloc* (GNU libc)
    - *mtmalloc* (Sun's "MT-hot" allocator)
  - Micro-benchmarks
    - *Threadtest*: no sharing
    - *Larson*: sharing (server-style)
    - *Cache-scratch*: mostly reads & writes (tests for false sharing)
- Real application experience similar



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

## Runtime Performance: *threadtest*



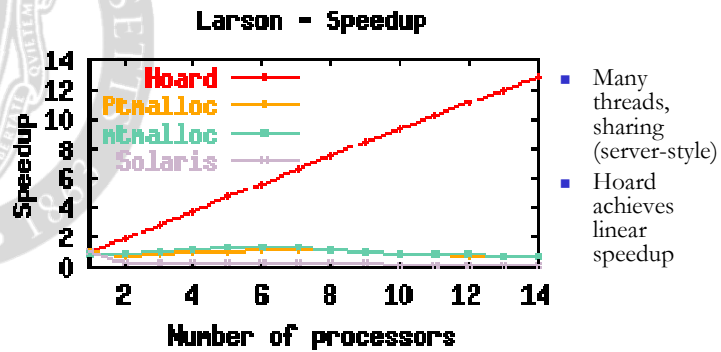
- Many threads, no sharing
- Hoard achieves linear speedup

$$\text{speedup}(x, P) = \frac{\text{runtime}(\text{Solaris allocator, one processor})}{\text{runtime}(x \text{ on } P \text{ processors})}$$



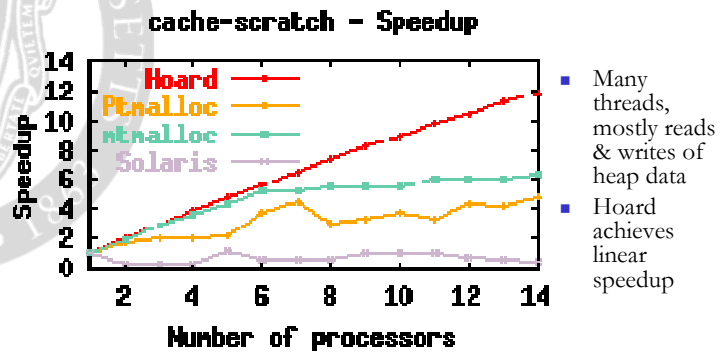
UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

## Runtime Performance: *Larson*



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

## Runtime Performance: *false sharing*



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

## Hoard in the “Real World”

- Open source code
  - [www.hoard.org](http://www.hoard.org)
  - 13,000 downloads
  - Solaris, Linux, Windows, IRIX, ...
- Widely used in industry
  - AOL, British Telecom, Novell, Philips
  - Reports: 2x-10x, “impressive” improvement in performance
  - Search server, telecom billing systems, scene rendering, real-time messaging middleware, text-to-speech engine, telephony, JVM

⇒ Scalable general-purpose memory manager



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

## Summary

- Building memory managers
  - *Heap Layers* framework [PLDI 2001]
- Problems with current memory managers
  - Contention, false sharing, space
- Solution: provably scalable memory manager
  - *Hoard* [ASPLOS-IX]
- Extended memory manager for servers
  - *Reap* [OOPSLA 2002]



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

## Lessons

- Hoard is really just a work queue
  - Instead of handing out work, it hands out virtual memory
- Performance considerations
  - Locality
  - Use private heaps
    - Reduces interaction among processors



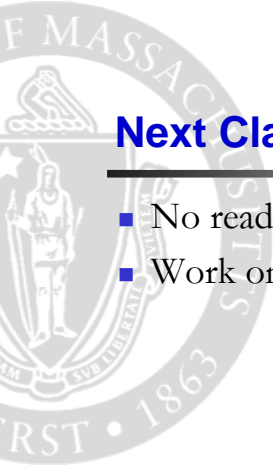
UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

## Lessons (cont)

- Performance considerations
  - Load balance
  - Use a global heap
    - Monitor utilization
    - Don't let any heap hoard all of the free memory
  - Granularity
  - Use page-sized blocks
    - Avoids false sharing
    - Reduces global heap interactions




UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science



## Next Class

---

- No reading
- Work on Assignment 4



UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science