A Compacting Real-Time Memory Management System

Silviu S. Craciun, Christoph M. Kirsch, Hannes Payer, Ana Sokolova, Horst Stadler, Robert Staudinger

Hannes Payer
Computational Systems Group, University of Salzburg

July 1, 2008
Overview

Introduction

Compact-Fit

Experiments

Conclusion
Motivation

Traditional dynamic memory management systems are typically non-deterministic:

- unpredictable response times of memory operations
- unpredictable memory fragmentation

⇒ Dynamic memory management systems are typically not used in time-critical software components (hard real-time systems, device drivers . . . )
Predictable Memory Management System

Predictability in Time

The time a memory management operation takes is determined by the size of the object involved in the operation (allocation, deallocation, and dereference).

Predictability in Space

The number of actual allocations together with their sizes (not the order of invocations) determines how many more allocations of a given size will succeed before running out of memory.
Solution Space

- Time
  - predictable
  - unpredictable

- Space
  - predictable
  - unpredictable

Hannes Payer
Computational Systems Group, University of Salzburg
What We Want?

A memory management system predictable in time and space (component of the real-time operating system Tiptoe)

Properties:

- `malloc(n)` takes at most $O(n)$ time
- `free(n)` takes at most $O(n)$ time
- memory access (dereference) takes small constant time
- small and predictable memory fragmentation bound
Fragmentation Problem

fragmentation in a contiguous space $\Rightarrow$
compaction $\Rightarrow$ reference updates

35% free
not allocatable
Solution to Reference Updates

Application

Abstract Space Indirection Table

Concrete Space Memory
Solution to Reference Updates

Application

Abstract Space
Indirection Table

Concrete Space
Memory
Compaction

Trade-Off:

speed versus memory fragmentation

Requirement:

keep speed and memory fragmentation bounded and predictable

2 Extreme Non-Solutions:

- keep memory perfectly compact
- perform memory operations in constant time without considering memory fragmentation
Concrete Address Space

- concrete address space is divided into pages of equal size
Concrete Address Space

- concrete address space is divided into **pages** of equal size
- each page itself is divided into fixed-sized **page-blocks**
Concrete Address Space

- concrete address space is divided into **pages** of equal size
- each page itself is divided into fixed-sized **page-blocks**
- \( n \) predefined page-block sizes \( \Rightarrow \) \( n \) different **size-classes**
Concrete Address Space

- concrete address space is divided into **pages** of equal size
- each page itself is divided into fixed-sized **page-blocks**
- $n$ predefined page-block sizes $\Rightarrow n$ different **size-classes**

![Image of address space division with size-classes: green, red, and blue]
Concrete Address Space

- concrete address space is divided into **pages** of equal size
- each page itself is divided into fixed-sized **page-blocks**
- $n$ predefined page-block sizes $\Rightarrow n$ different **size-classes**
Concrete Address Space

- concrete address space is divided into pages of equal size
- each page itself is divided into fixed-sized page-blocks
- \( n \) predefined page-block sizes \( \Rightarrow \) \( n \) different size-classes

![Diagram showing concrete address space with different size-classes: green, red, and blue.](image)
Concrete Address Space

- concrete address space is divided into **pages** of equal size
- each page itself is divided into fixed-sized **page-blocks**
- \( n \) predefined page-block sizes \( \Rightarrow n \) different **size-classes**

![Concrete Address Space Diagram]

\[\text{size-class green}\]

\[\text{size-class red}\]

\[\text{size-class blue}\]

FREE size-class green
size-class red
size-class blue

Conclusion
Concrete Address Space

- Concrete address space is divided into **pages** of equal size
- Each page itself is divided into fixed-sized **page-blocks**
- $n$ predefined page-block sizes $\Rightarrow n$ different **size-classes**
Concrete Address Space

- concrete address space is divided into **pages** of equal size
- each page itself is divided into fixed-sized **page-blocks**
- $n$ predefined page-block sizes $\Rightarrow n$ different **size-classes**

![Diagram showing混凝土Address Space with different size-classes and connections between them.](image-url)
Deallocation May Involve Compaction

**Size-Class Compact Invariant:**

Each size-class can contain at most one not-full page.
Deallocation May Involve Compaction

Size-Class Compact Invariant:

Each size-class can contain at most one not-full page.

Hannes Payer
Computational Systems Group, University of Salzburg
Deallocation May Involve Compaction

Size-Class Compact Invariant:

Each size-class can contain at most one not-full page.
Deallocation May Involve Compaction

Size-Class Compact Invariant:

Each size-class can contain at most one not-full page.
Deallocation May Involve Compaction

Size-Class Compact Invariant:

Each size-class can contain at most one not-full page.
Deallocation May Involve Compaction

Size-Class Compact Invariant:

Each size-class can contain at most one not-full page.
Deallocation May Involve Compaction

Size-Class Compact Invariant:

Each size-class can contain at most one not-full page.
Deallocation May Involve Compaction

Size-Class Compact Invariant:

Each size-class can contain at most one not-full page.
Deallocation May Involve Compaction

Size-Class Compact Invariant:

Each size-class can contain at most one not-full page.
Compact-Fit Versions

- Compact-fit moving version (CFM)
  - concrete space = physical memory
  - allocated objects are *contiguous* in physical memory
  - compaction: leads to movements in physical memory

- Compact-fit non-moving version (CFNM)
  - concrete space = virtual memory (blocks)
  - allocated objects are *not contiguous* in physical memory, but are contiguous in virtual memory
  - compaction: reprogramming block table
Compact-Fit Moving Version Complexity

- `malloc(n)` takes $\Theta(1)$ time
- `free(n)` takes $\mathcal{O}(n)$ time because of compaction
- memory access (dereference) takes $\Theta(1)$ time because of abstract address space
- memory fragmentation is bounded and predictable
Compact-Fit Non-Moving Version Complexity

- `malloc(n)` takes $\Theta(n)$ time because of maintaining the virtual memory
- `free(n)` takes $\Theta(n)$ time because of maintaining the virtual memory and compaction
- Memory access (dereference) takes $\Theta(1)$ time because of abstract address space and virtual memory
- Memory fragmentation is bounded and predictable
### Partial Compaction

**Idea:**

Allow an arbitrary number $k$ of not-full pages within a size-class.

**Result:**

Each deallocation that happens when $number\_not\_full\_pages \leq k$ takes constant time, but fragmentation increases with $k$.

**Effect:**

This way we formalize, control, and implement the trade-off between temporal performance and memory fragmentation.
Partial Compaction

size-class red: k=2

size-class green

size-class blue
Partial Compaction

size-class red: \(k=2\)

size-class green

size-class blue
Partial Compaction

size-class red: k=2

size-class green

size-class blue
Partial Compaction

size-class red: \( k=2 \)

\[
\begin{array}{cccc}
\text{size-class red} & & & \\
& \text{size-class green} & & \\
\text{size-class red} & & & \\
& & \text{size-class blue} & \\
\end{array}
\]
Partial Compaction

size-class red: k=2

size-class green

size-class blue
Partial Compaction

size-class red: $k=2$

size-class green

size-class blue

Hannes Payer
Computational Systems Group, University of Salzburg
Partial Compaction

size-class red: $k=2$

size-class green

size-class blue

FREE
Partial Compaction

size-class red: $k=2$

size-class green

size-class blue
Related Work

Time

- **O(1)**
  - First-fit: free, deref
  - Best-fit: free, deref
  - DL: free, deref
  - TLSF: malloc, free, deref
  - Half-fit: malloc, free, deref

- **O(log n)**
  - Metronome

- **O(n)**
  - CFM: free
  - CFNM: malloc, free
  - Jamaica: malloc, free, deref

- **unbounded in n**
  - First-fit: malloc
  - Best-fit: malloc
  - DL: malloc
  - Jamaica: deref
  - CFM: deref
  - CFNM: deref

Space

- **unpredictable**
- **predictable**

Hannes Payer
Computational Systems Group, University of Salzburg
Incremental Allocation Benchmark
Incremental Free Benchmark
Incremental Free Partial Compaction

The figure compares the performance of different memory allocation algorithms, specifically focusing on Incremental Free Partial Compaction (IFCP). The X-axis represents the number of deallocation operations of increasing size, while the Y-axis shows the number of instructions. The graph includes lines for various allocation strategies:
- First-fit
- Best-fit
- TLSF
- Half-fit
- CFM
- DL
- CFNM

The results indicate that Incremental Free Partial Compaction (IFCP) effectively manages memory fragmentation, achieving a lower number of instructions compared to other strategies as the number of deallocation operations increases. This suggests that IFCP is particularly efficient in maintaining memory compactness during dynamic memory allocation and deallocation processes.
Fragmentation

![Graph showing fragmentation and compact-fitting algorithms]

- CFM 1
- CFM 2
- CFM 3,4
- CFM 5
- CFM 6,7,8,9
- TLSF

Hannes Payer
Computational Systems Group, University of Salzburg
Conclusion and Future Work

Contribution:

• Compact-fit is predictable in time and space
• moving and non-moving Compact-fit implementations

Future work:

• virtual machine implementation
• source-to-source translator
• concurrency and multi-processor support
• static program analysis can help to optimize the $k$ for the partial compaction strategy

http://tiptoe.cs.uni-salzburg.at/compact-fit