Biased Allocator for Generational Garbage Collector

Hyung-Kyu Choi, HyukWoo Park and Soo-Mook Moon

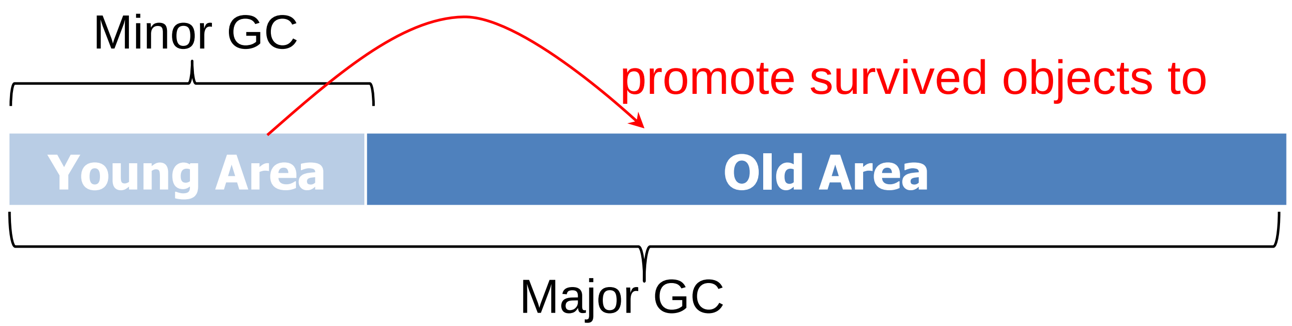
Virtual Machine & Optimization Lab.
Electrical and Computer Engineering
Seoul National University

Presenter: HyukWoo Park

Outline

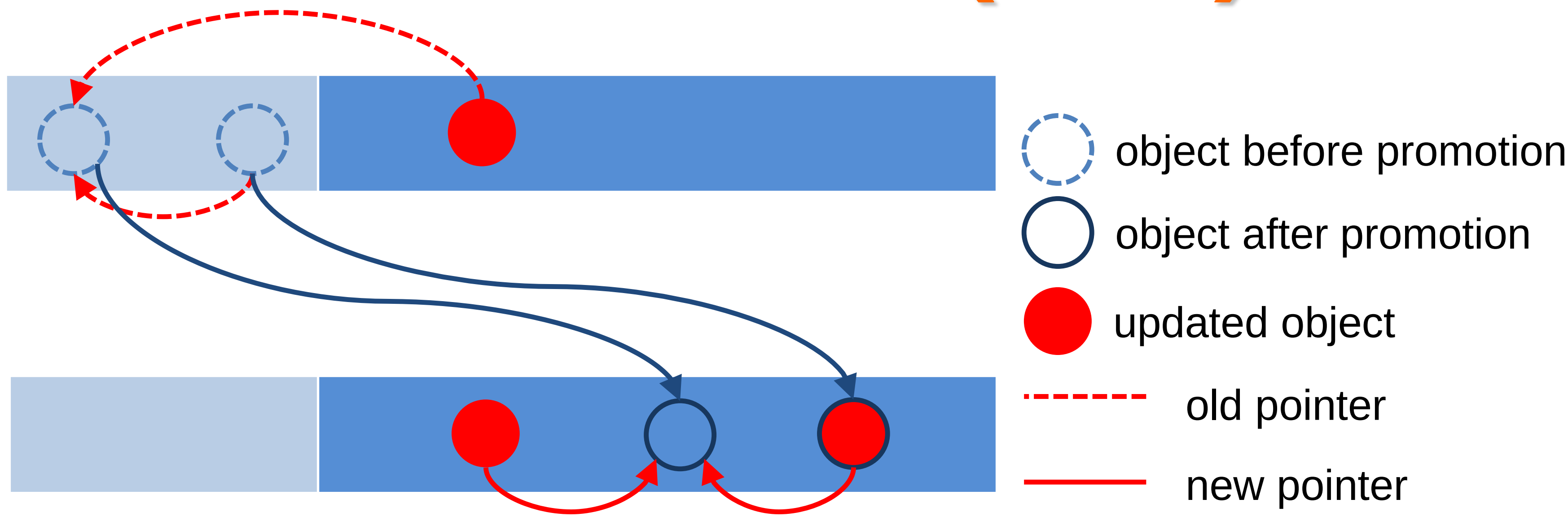
- Generational Garbage Collector
- Motivation
- Biased Object Allocator
- Evaluation
- Summary

Generational Garbage Collector



- Heap is divided into young area and old area
  - New objects are allocated from young area only
  - Minor GC for young area occur more frequently
    - GC-survived objects are promoted to old area
  - Major GC for young + old area occur once in a while
    - When minor GC fails to reclaim space or major GC is requested
- Reduces overhead of each GC, but sometimes makes young area overcrowded, causing more GCs

Generational GC (cont')



- Generational GC suffers from additional overhead when objects are promoted to old area
- Promotion overhead is consisted of
  - **Copying** objects from young area to old area
  - **Updating** address of pointers to moved objects

Motivation

- The overhead of promotion is unpredictable and can be heavy
  - Number of promoted objects varies
  - Number of pointers referring promoted objects varies
- Therefore overall overhead of generational GC can be reduced, if we can avoid the overhead of promotion.

- Let's allocate objects to old area instead of young area to avoid the promotion.

Biased Object Allocator

- Allocates some new objects directly to old area
  - Those objects likely to be long-lived
  - Keep young area from being overcrowded
  - Avoid the promotion overhead from young to old area

- How can we identify long-lived objects?
  - We analyze the code to predict lifetime of object and leave *hint* during ahead-of-time compilation

(1) Escape Analysis

- Escape analysis can identify local-scoped objects
  - whose live-range do not escape method boundary
    - J.-D. Choi et. al, Escape analysis for Java. In OOPSLA '99
    - They can be allocated to the stack, not the heap

```
public String foo (int a) {
        Integer x = new Integer(a);
        …
        return "interger " + x.toString();
}
```

live-range of object ↓

  - Those objects may be short-lived, so not allocated to old area

(2) Objects Allocated in Loops

- Most objects are allocated within loop
  - from the observation of specjvm98 benchmark
- Objects allocated within loop seems to be short-lived
  - They are likely to be temporary objects to compute something
- But we should select object carefully
  - Only object with size smaller than threshold are chosen
  - Aggressive biased allocation may suffer from side effects.

(3) Objects Assigned to Static Fields

- Previous research shows that objects assigned to static fields of classes tend to be immortal (long-lived)
  - M. Hirzel et al, Understanding the connectivity of heap objects. ISMM '02

- Let's allocate those object to old area

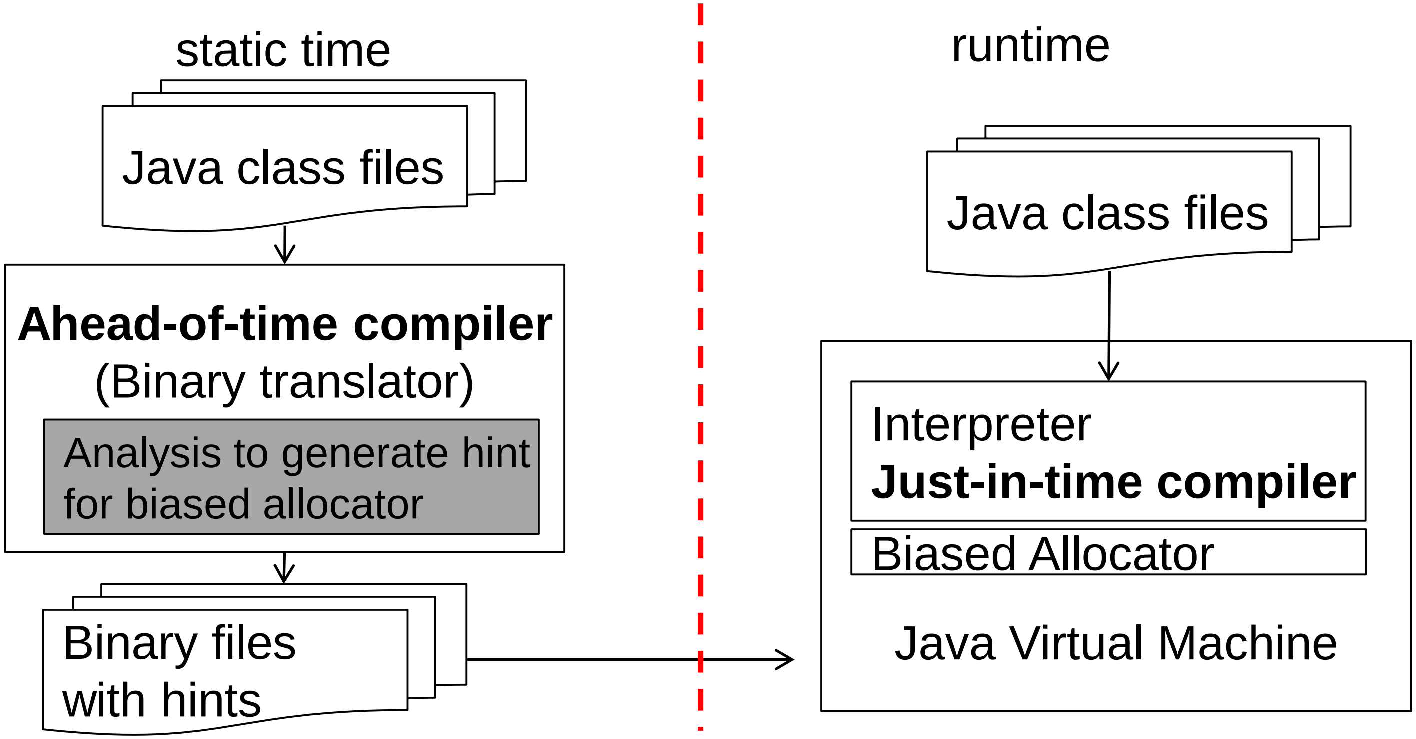
Generating Biased Hint

$object_{old}$	object which is expected to be long-lived
$object_{local}$	object identified by escape analysis
$object_{loop}$	object allocated within loop boundary
$object_{immortal}$	object which is assigned to static fields

$$object_{old} = \{(\text{all objects}) - object_{local} - object_{loop}\} + object_{immortal}$$

- Generate and leave a hint to a allocation site
  - where **$object_{old}$** are allocated

Implementation



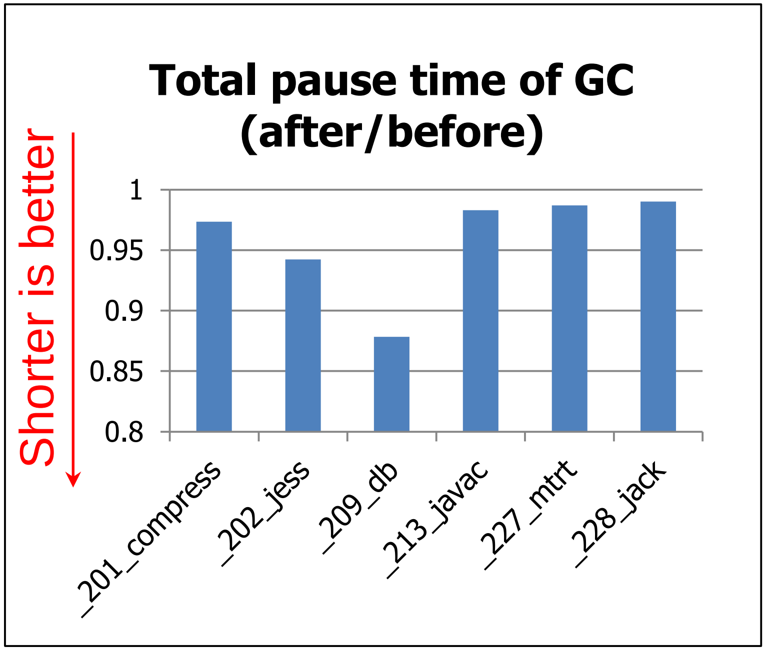
- Proposed analysis is implemented in ahead-of-time compiler (AOT)
  - To isolate analysis time from runtime
- Of course, analysis can be implemented in JITC as well.

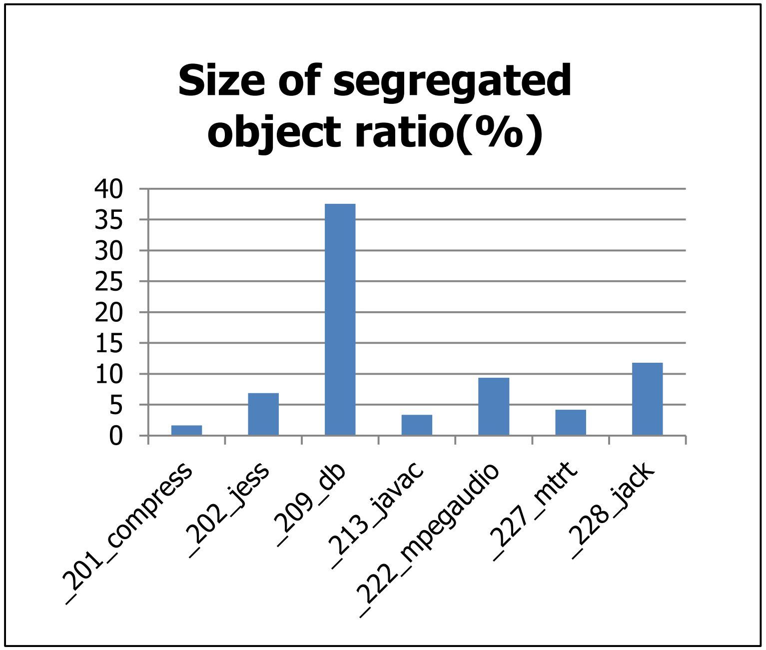
Evaluation Environment

- Digital TV platform
  - MIPS (AMD Xilleon)
  - 333MHz clock w/ 16K I-cache and 16K D-cache
  - 128MB main memory
  - Benchmark : specjvm98

- Oracle's phoneME Advanced MR2
  - with Just-in-time compilation (JIT)
  - with Ahead-of-time compilation (AOT)
  - 32MB of Java heap

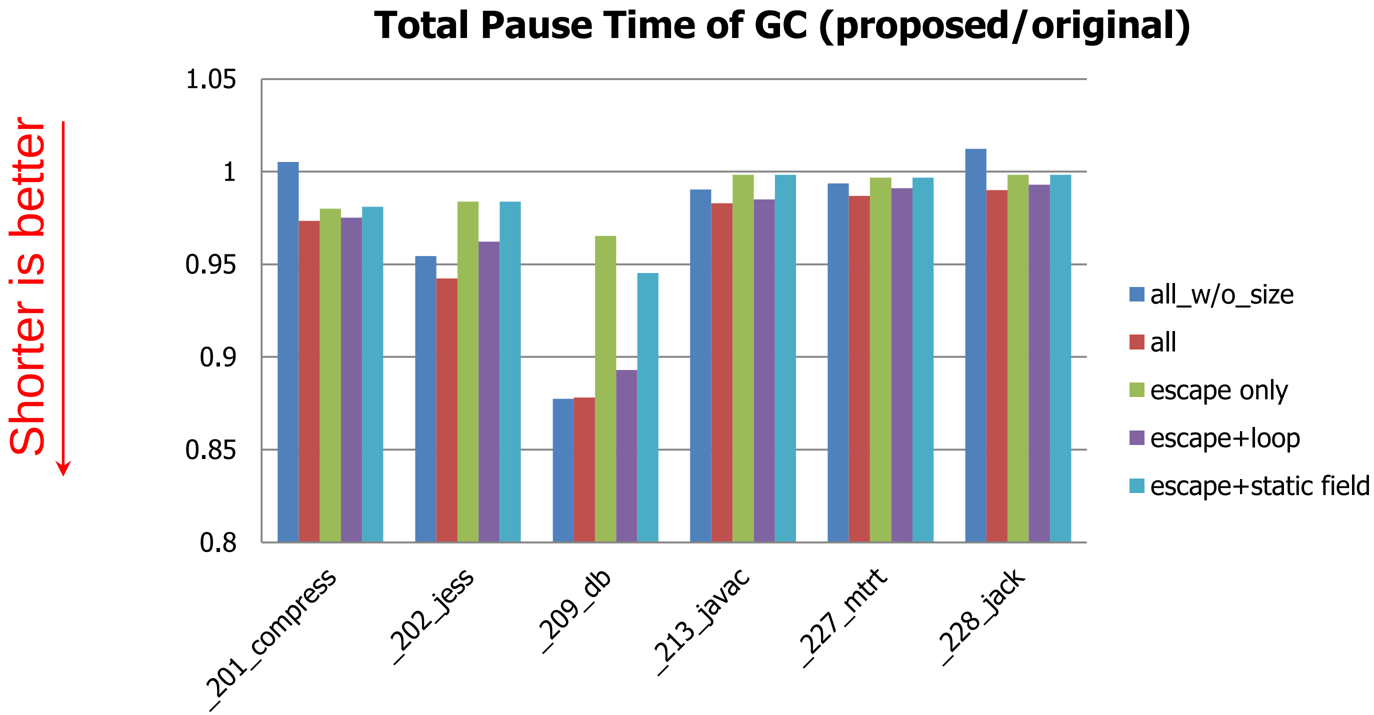
Total Pause Time of GC





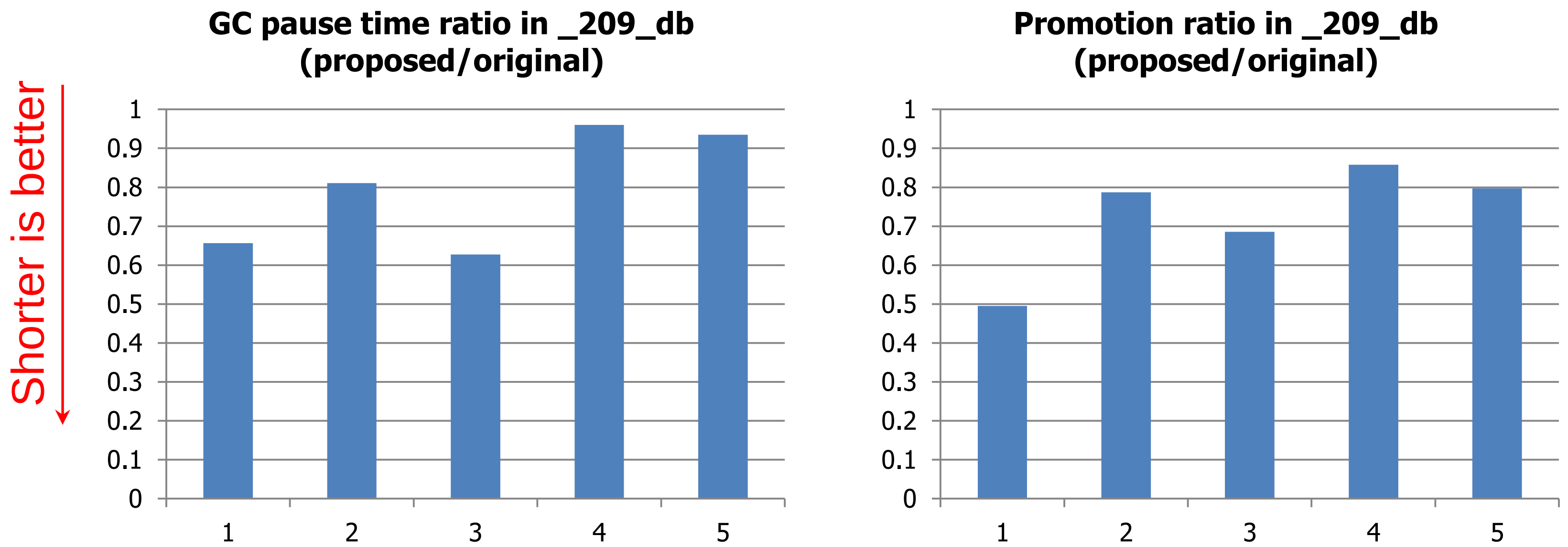
- GC overhead is reduced much in _209_db
  - Up to 37.6% amount of objects are biased in _209_db
- However total runtime is not improved much, because Java virtual machine spent relatively short time in garbage collection
  - Total runtime has been improved less than 1.4%

Effect of analyses



- Performance degrades with aggressive biased allocation
  - without considering size of objects
- Escape analysis and loop analysis are effective
- Static field analysis is not effective

Comparison of each GC



- Let's look into the first five GCs.
  - However we can't compare GC by one-to-one, because GC behavior has been changed after applying biased allocator
- Each GC invocation has shorter pause time with biased allocation.
- Promotion reduction and pause time reduction show correlation.
- The first GC invocation is delayed than original.

Summary

- Biased object allocation for generational GC
  - To reduce promotion overhead of generation GC
  - Allocates some new objects to old area with analyses
  - Three analyses has been used
    - Escape analysis
    - Loop analysis
    - Static field analysis
- Evaluation shows biased object allocator can reduce overhead of generational GC when used carefully

Thank you !