Pretenuring for Java™

Presented by Maria Jump

Special Thanks to Steve Blackburn for the Power Point Presentation

Steve Blackburn, Sharad Singhai, Matthew Hertz,
Kathryn McKinley & Eliot Moss

Architecture and Language Implementation Laboratory
Department of Computer Science, University of Massachusetts, Amherst

University of Massachusetts Amherst
UMASS

THE UNIVERSITY OF TEXAS AT AUSTIN

Where are they now?

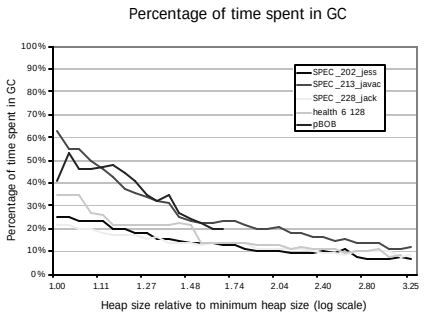
- Steven M. Blackburn
  - Senior Lecturer, Australia National University
- Sharad Singhai
  - Sandbridge Technology
- Matthew Hertz
  - Graduate Student, University of Mass., Amherst
- Kathryn S. McKinely
  - Associate Professor, University of Texas at Austin
- J. Eliot B. Moss
  - Associate Professor, University of Mass., Amherst

Generational GC

'nursery' 'older generation'

- Advantage:
  - Fast allocation (via 'bump pointer')
  - Cheap reclamation (en masse)
  - Non-fragmenting
  - Minimizes copying of older objects

GC Can Be Expensive



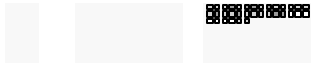
Problem

Long lived objects copied *at least* once

Pretenuring

Allocate long lived objects *directly* into older generation

- Identify long lived sites (via profiling) [CHL98]
- Modify to allocate into older generation



'nursery' 'older generation'

Pretenuring: our Goals

- Apply pretenuring to Java
- Generalize:
  - *different garbage collectors*
  - *program scales*
  - *different programs*
- Apply to different collectors
  - Appel-style generational collector
  - Older-first collector

Pretenuring for Java

Object Lifetime Homogeneity

Pretenuring requires allocation site lifetime homogeneity

- True for ML [CHL98]
- Not true for C (call chain is necessary) [BZ93]

What about Java?

GC-neutral Lifetime Statistics

(In the GC literature, 'time' is usually expressed in terms of allocation, not seconds)

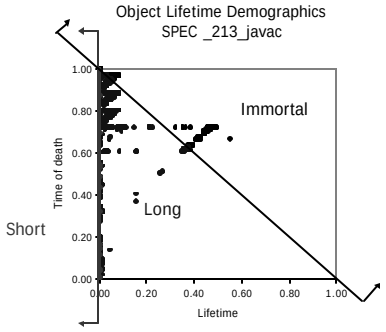
- **Lifetime**
  - Time from birth to death of an object
  - Expressed w.r.t. min heap size
- **Time of death**
  - Point in program when object dies
  - Expressed w.r.t. total allocation
- Use profiling
  - Trace 'birth' and 'death' events
  - Very frequent GCs (e.g. every 64K)

Discussion Question

- 64K granularity in GC Profiling
- What affects could profile granularity have on the pretenuring decision?

Three-way Classification

- Short
  - Lived less than fraction *s* of a heap
- Immortal
  - Lifetime is longer than 'deathtime' *(exploit fact that non-copied space is half cost of copied space)*
- Long
  - If neither short nor immortal



Mapping from Instance to Site

- Given 3 fractions: S_f, L_f, & I_f, *and a homogeneity factor,* H_f:
  - If $S_f + H_f > L_f + I_f$ site is *short*
  - Else if $S_f + L_f + H_f > I_f$ site is *long*
  - Otherwise site is *immortal*

Discussion Question

- What other methods could be used to decide to pretenure an allocation site?

Pretenuring Mechanics

- Generate advice file
  - `<class> <method> <offset> <[s|l|i]>`
- Supply advice to compiler
  - Env. variable or command line option
  - Compiler generates map
  - Consults map for each `new()`
  - Compiles in appropriate allocation code

CHL Pretenuring

- CHL pretenuring advice:
  - Profile application using *generational GC*
  - Any site where 80% of allocated instances 'survive' the nursery is long-lived
- Limitations
  - Can't easily combine advice
  - Collector-specific

Experimental Setting: Jikes RVM

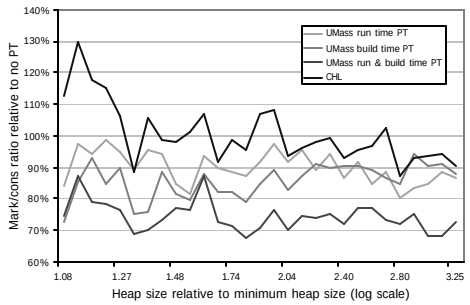
(formally known as Jalapeño)

- JVM written in Java *(see OOPSLA 99,00,01, PLDI 00,01)*
  - High performance
  - Aggressive optimizing compiler
  - Flexible GC toolkit
- 'Boot image' contains core classes *(class loader, compiler etc.)*
  - Opportunity for *application-neutral* pretenuring
  - (Additional to *application-specific* pretenuring)
- GCTk, a OO GC toolkit with run-time and build-time 'allocation advice' implemented

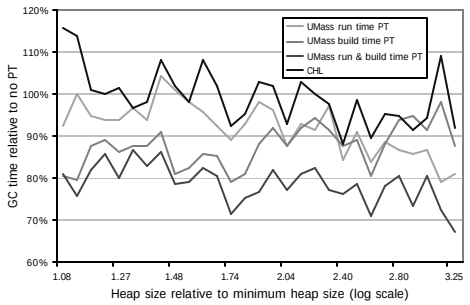
Results

- UMass build-time advice is 'true' advice
  - Advice for each application based on profile of remaining N-1 applications
  - Run-time advice is 'self' advice
- Used 5 benchmarks
  - 3 from SPEC JVM
  - 1 from Olden
  - IBM's pBOB
- Measurements across 32 heap sizes

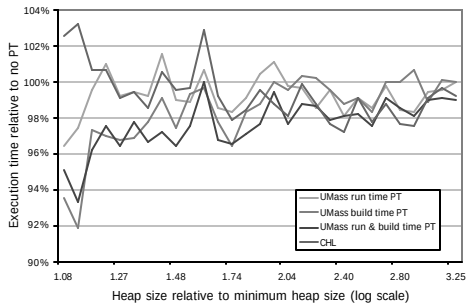
Mark/cons: geometric mean of 5 benchmarks



GC time: geometric mean of 5 benchmarks



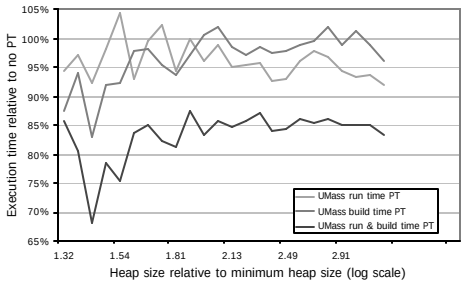
Execution time: geometric mean of 5 benchmarks



Pretenuring in the Older First GC Algorithm

- Older First [SMM99]
  - Efficient new copying GC algorithm
  - Different collection order to generational GC
  - Different heap layout to generational GC
    - We add a permanent space (for immortals)

Older First GC
Execution time: geometric mean of 5 benchmarks



Conclusions

- Java programs *are* suitable for pretenuring
- UMass pretenuring is general:
  - Exploits 'immortal' objects
  - Combinable (suitable for build-time)
  - Collector neutral
- Applied to Jalapeño/Jikes RVM
  - Significant performance improvements
  - Build-time pretenuring highly practical (significant benefits without application profiling)

UMASS

TEXAS

The End