Harnessing Memory Management to Optimize for Efficiency

Jennifer B. Sartor

Virtual Machine Summer School 2016
Multicore Challenge

Application

Managed language runtime environment

Operating System

Chip

$P$

$P$

$P$

$P$

LLC

memory (DRAM)
- Young objects die quickly

- Nursery
  - Traced for live objects
  - Copy to mature space
  - Reclaimed ‘en masse’
Problem: Bandwidth & Power Wall

Application

Managed language runtime environment

Zero initialization

Operating System

Chip

$  $  $  $

LLC  00000000

memory (DRAM)
Problem: Allocation Wall

Managed language runtime environment

Objects rapidly allocated and short-lived

Operating System

Application

Chip

$ P $ P $ P $ P

DEAD DEAD DEAD DEAD

LLC

memory (DRAM)
Why energy-efficient computing?

(1) 100 Billion kW.h per year in U.S. alone
(2) Saving 20% in efficiency = $2 billion

(energy.gov)

(1) More searches on mobile
(2) Battery life is a big concern

(Google)

End of Dennard scaling makes modern chips power-constrained

(Robert H. Dennard)
Heterogeneous Multicores

User Application

Virtual Machine

Operating System

BIG

BIG

little

little
Concurrent Garbage Collector

App.

Conc.
GC
STW
GC

a0
a1
a2
a3
g0
g1
g2
g3

Application

Conc. Collection

Release

Roots
If Collector Cannot Keep Up

- Application
- Roots
- Scan

- STW
- Collection
- Release

- App.
- Conc.
- GC
- STW
- GC

a0 → g0 → a1 → g1 → a2 → g2 → a3 → g3 → STW
Understanding the BIG core’s performance advantage
Understanding the BIG core’s performance advantage
Understanding the BIG core’s performance advantage
Understanding the BIG core’s performance advantage
Understanding the BIG core’s performance advantage
Sample at all DVFS states 😞

Estimate performance 😊

DVFS Performance Prediction

Sample at all DVFS states

Estimate performance

many applications here

compute bound

memory bound

frequency ➔

speedup ➔

performance ➔
Managed Multi-threaded Applications

Heterogeneity

Garbage Collection Service

Synchronization

Store Bursts
Sniper Simulator

- **Extensions to work with JVM**
  - Works with JIT compiler
  - Emulate system calls (futex & nanosleep)
  - JVM-simulator communication with new instruction

- **Simulates**
  - x86, cycle-level, parallel, high-speed
  - Multicore, heterogeneous
  - Different frequencies
  - McPat for power
Methodology

- Sniper simulator
- Jikes RVM 3.1.2 and DaCapo benchmarks
  - Collector
    - Generational Immix garbage collector
    - Concurrent mark-sweep snapshot algorithm
  - 2x minimum heap
  - Replay compilation, 2nd invocation
Cooperative Cache Scrubbing

Jennifer B. Sartor, Wim Heirman, Steve Blackburn*, Lieven Eeckhout, Kathryn S. McKinley^  

PACT 2014
Object allocation and management in managed languages and environments.
Problem: Bandwidth & Power Wall

Objects rapidly allocated and short-lived

Managed language runtime environment

Zero initialization

Operating System

Chip

Application

Objects rapidly allocated and short-lived

Managed language runtime environment

Zero initialization

Operating System

Chip

Application

Managed language runtime environment

Zero initialization

Managed language runtime environment
Cooperative Cache Scrubbing

Objects rapidly allocated and short-lived

Managed language runtime environment

Zero initialization

Application

Operating System

Chip

$P$

$P$

$P$

$P$

$P$

$P$

$P$

$P$

Dead

Dead

Dead

Dead

00000000

00000000

write

read

memory (DRAM)
Generational Garbage Collection

- Young objects die quickly
- Nursery
  - Traced for live objects
  - Copy to mature space
  - Reclaimed ‘en masse’
Dead Lines in LLC (8MB)

Cache lines (%)

0% 20% 40% 60% 80% 100%

antlr avrora bloat fop jython luindex lusearch lusearch.fix pmd sunflow xalan Mean

Nursery size:
4M 8M 16M
Dead Data Written Back?

Application

Managed language runtime environment

Operating System

Chip

$P$

$P$

$P$

$P$

$\text{LLC}$

memory (DRAM)
Useless Write Backs (8MB LLC)

Write backs (%)

- antlr
- avrora
- bloat
- fop
- jython
- luindex
- lusearch
- lusearch.fix
- pmd
- sunflow
- xalan
- Mean

Nursery size:
- 4M
- 8M
- 16M
Cooperative Cache Scrubbing

- Communicate managed language’s semantic information to hardware

- Caches
  - ‘Scrub’ dead lines
  - Zero lines without fetch

- Result
  - Better cache management
  - Avoid traffic to DRAM
  - Save DRAM energy
SW-HW Cooperative Scrubbing

- **Software**
  - Identify cache line-aligned dead/zero region
  - Generational Immix collector (stop-the-world)
    - After nursery collection, call scrub instruction on each line in entire range
    - Call zero instructions to zero region (32KB)

- **Hardware**
SW-HW Cooperative Scrubbing

- Software
- Hardware
  - Scrubbing (LLC)
    - `clinvalidate`: invalidates cache line
    - `clundirty`: clears dirty bit
    - `clclean`: clears dirty bit, moves line to LRU
  - Zeroing (L2)
    - `clzero`: zero cache line without fetch
  - Modifications to MESI cache coherence protocol
    - Back-propagation from LLC to L1/L2 cache levels
    - Local coherence transitions (no off-chip)
Methodology

- **Sniper simulator**
  - 4 cores, 8MB shared L3 (LLC), McPAT

- **Jikes RVM 3.1.2 and DaCapo benchmarks**
  - Generational Immix garbage collector
  - 4 application, 4 GC threads
Total DRAM Energy

![Bar Chart]

- Energy Reduction (%)
- 4M, 8M, 16M
- clinvalidate
- clundirity
- clclean
- clzero
- clclean+clzero

-22%
clclean+clzero Improvements

Bar chart showing improvements in various metrics such as DRAM Reads, DRAM Writes, Total DRAM Traffic, LLC misses, Execution time, Dynamic DRAM Energy, and Total DRAM Energy for 4MB, 8MB, and 16MB.
Related Work

- **Cooperative cache management**
  - ESKIMO by Isen & John, Micro 09
    - Useless reads and writes to DRAM by sequential C programs
    - Reduce energy
    - Require large map in hardware, extra cache bits
  - Wang et al., PACT 02/ ISCA 03; Sartor et al., 05
    - C & Fortran static analysis to give cache hints to evict or keep data

- **Zero initialization** [Yang et al., OOPSLA 11]
  - Studied costs in time, cache and traffic
  - Use non-temporal writes to DRAM, increase bandwidth
Conclusions

- Software-hardware cooperative cache scrubbing
  - Leverages region allocation semantics
  - Changes to MESI coherence protocol
  - New multicore architectural simulation methodology
  - Reductions
    - 59% traffic
    - 14% DRAM energy
    - 4.6% execution time

http://users.elis.ugent.be/~jsartor/