Myths and Realities: The Performance Impact of Garbage Collection

Presented by: Tapasya Patki

February 17, 2011

Authors

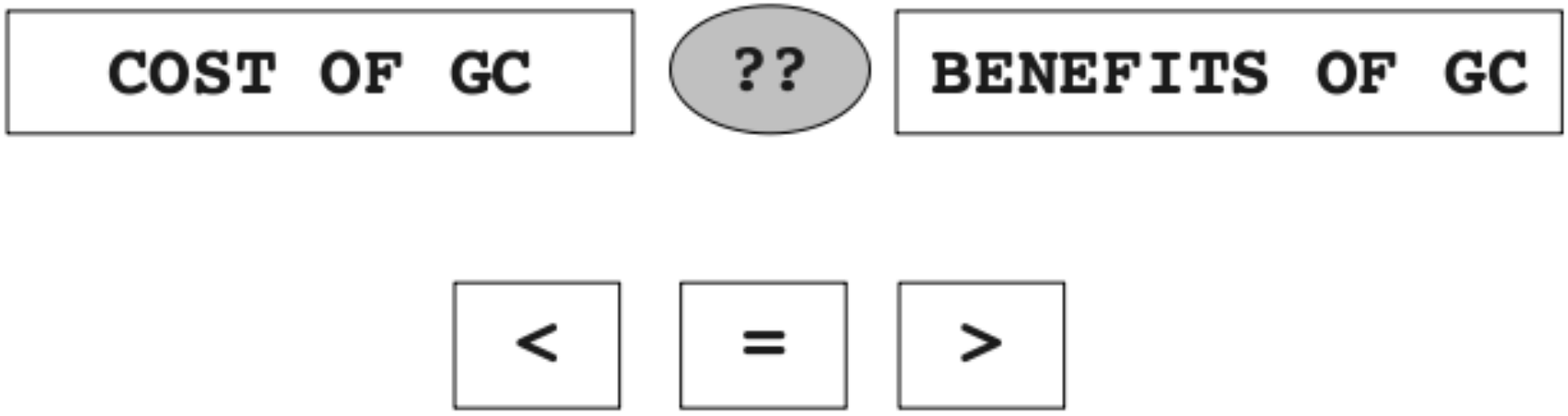
ANU

THE AUSTRALIAN NATIONAL UNIVERSITY

IBM

THE UNIVERSITY OF TEXAS AT AUSTIN

Motivation: Cost-Benefit Analysis



Motivation: To GC or not to GC

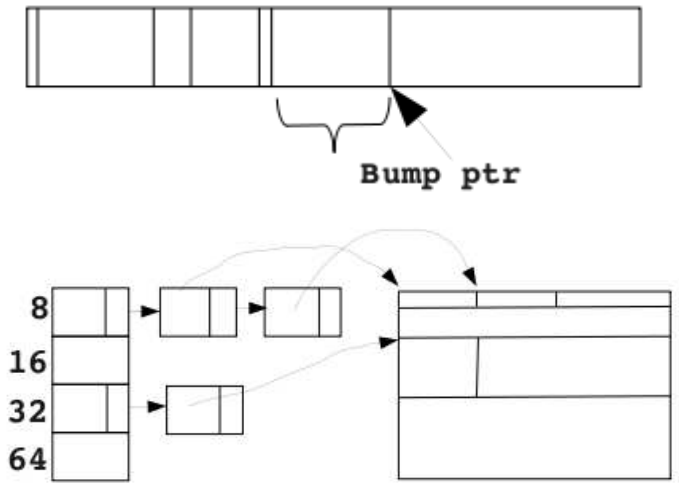
- Explicit Memory Management or Garbage Collection?
- Can GC ever improve application performance?
- Whole-heap collection or Generational collection?

Motivation: Some more questions

- Sensitivity to Heap Size
- Frequency of collection
- Cost of collection: Whole-heap or Nursery?
- Space tradeoff: SemiSpace or Mark-Sweep?
- Locality: L1, L2, TLB misses
- Influence of processor architecture on GC

Background: Allocation Strategies

- Contiguous
  - append new object by incrementing a bump pointer
  - speed of allocation, locality
- Free-list
  - k size-segregated lists (binning)
  - allocate new object in smallest size class
  - internal fragmentation, faster compaction



Background: Collection Strategies

- Tracing
  - transitive closure: `roots` $\cup$ `remembered-set`
  - reclaim by copying live objects
- Reference Counting
  - count number of references
  - reclaim objects with no references

Background: Whole-heap GC Algorithms

Algorithm	**Allocator**	**Collector**
SemiSpace	Contiguous	Tracing
MarkSweep	Free-list	Tracing
RefCount	Free-list	Reference Counting

Background: Generational GC Algorithms

Algorithm	**Nursery**	**Mature**
GenSS	SemiSpace	SemiSpace
GenMS	SemiSpace	MarkSweep
GenRC	SemiSpace	RefCount

- write-barrier: to record pointers from mature-space to nursery
- size of nursery
- compaction of nursery survivors

Methodology

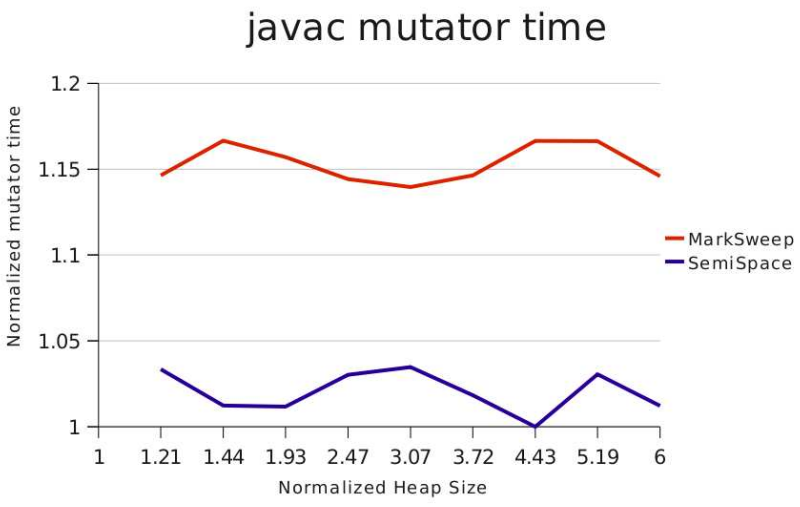
- MMTk in IBM Jikes RVM
  - Java-in-Java design
  - pseudo-adaptive compilation using application profiles (deterministic)
  - immortal space for itself (compiler, classloader, collector)

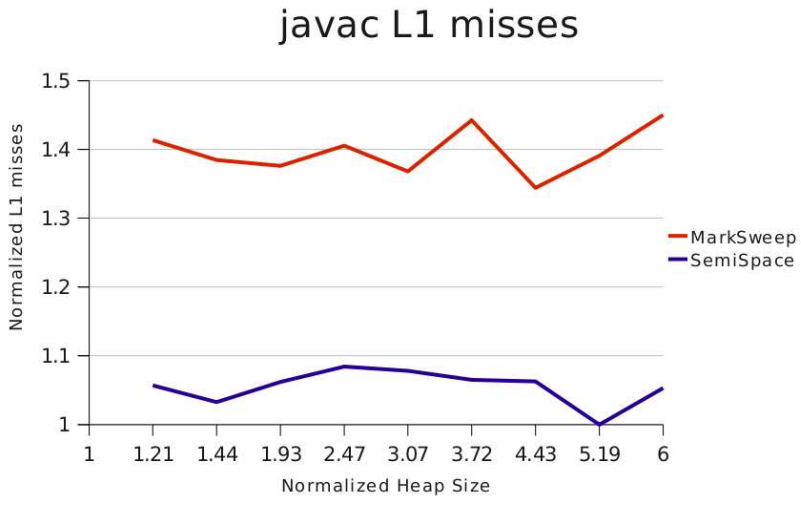
Arch	**CPU Freq**	**RAM**	**L1**	**L2**
Athlon	1.9Ghz	1GB	64K both	512K
P4	2.6GHz	1GB	8K D, 12K I	512K
PPC	1.6GHz	768MB	32K D, 64K I	512K

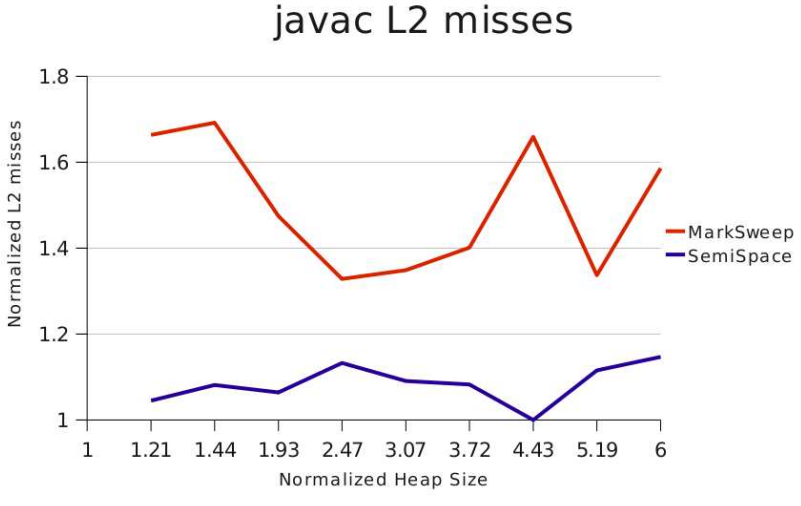
Benchmarks

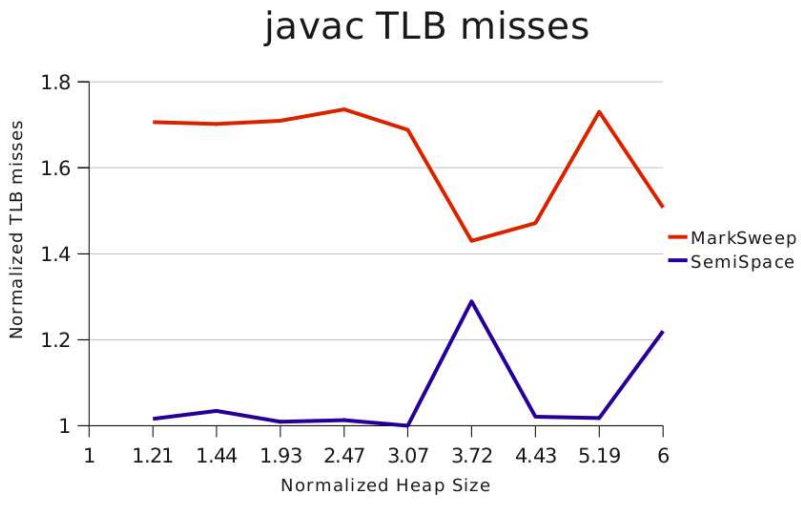
- Mutator Phase: application code, contains allocation sequence and write-barriers
- GC Phase
- SPEC JVM benchmarks

Results: javac

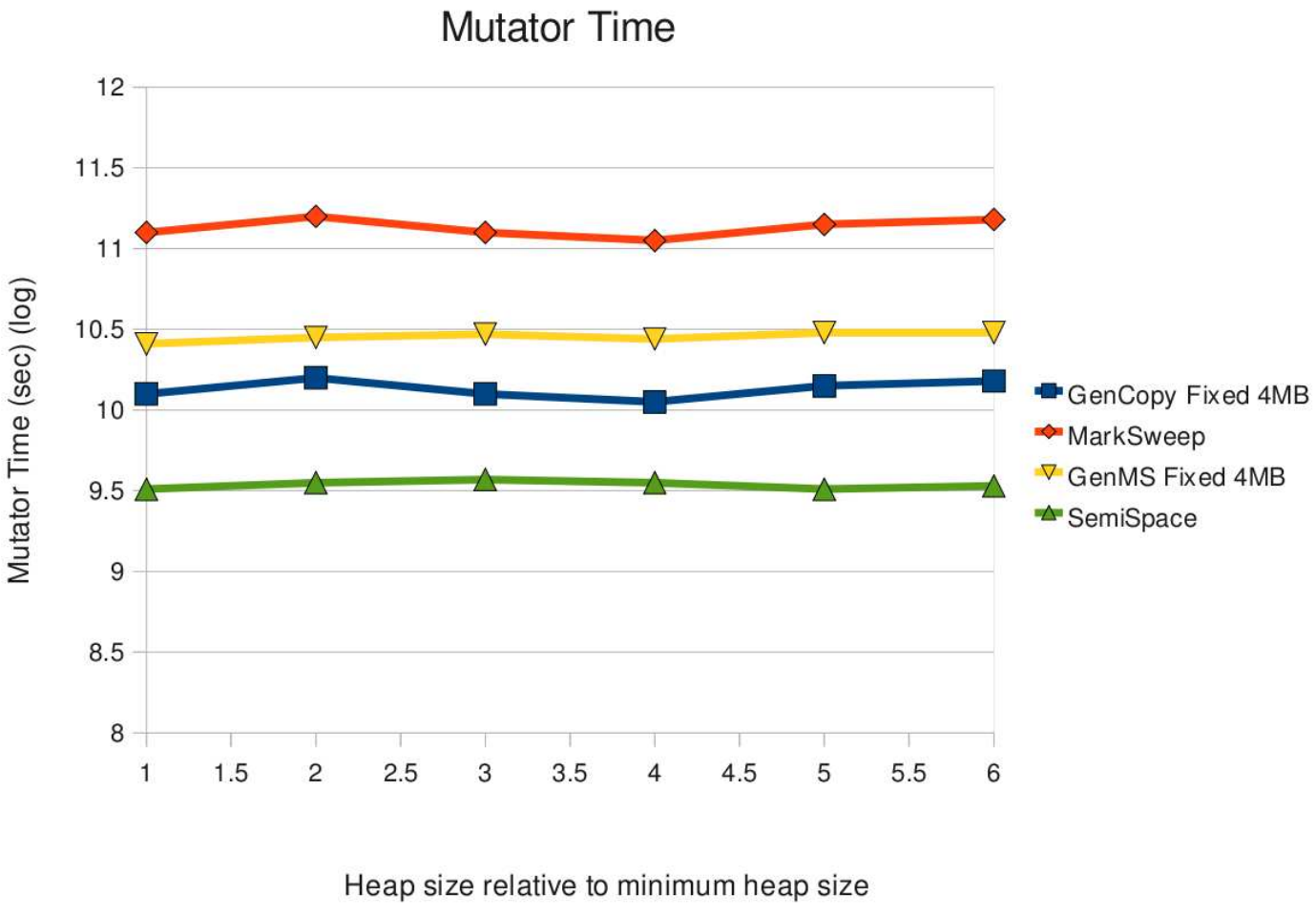








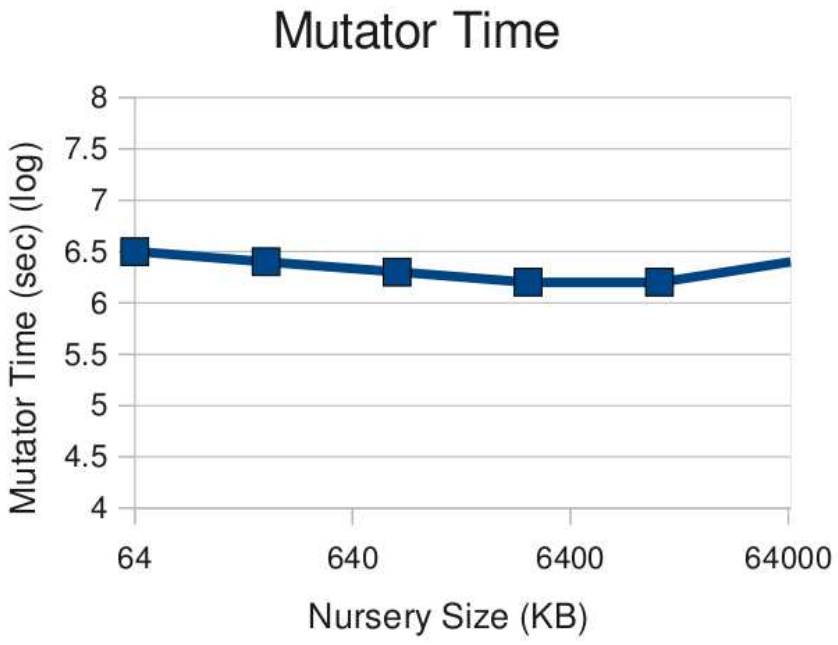
Results: Generational vs Whole-heap (Rough Sketch)

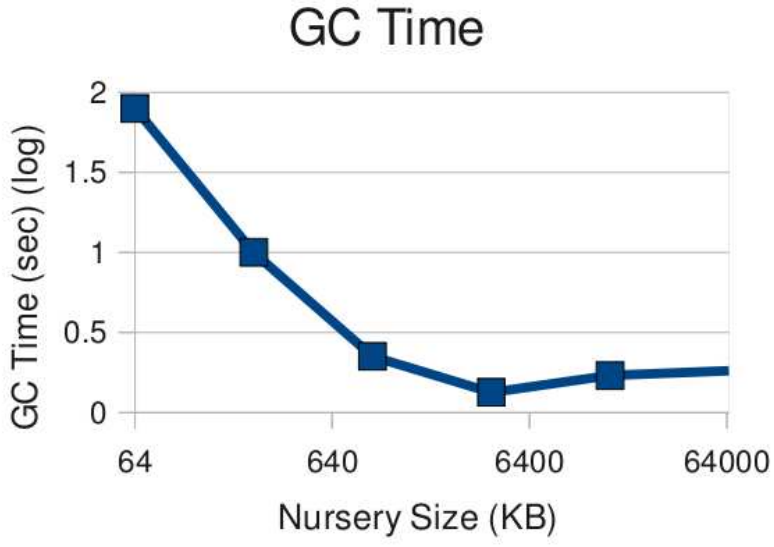


Results: Write Barrier

	% Overhead %
_202_jess	13.6
_228_jack	1.7
_205_raytrace	0.9
_227_mtrt	2.9
_213_javac	4.6
_201_compress	0
pseudojbb	3.1
_209_db	2.4
_222_mpegaudio	0
Geometric mean	3.2

Results: Nursery Size Trend for GenCopy and GenMS (Rough Sketch)

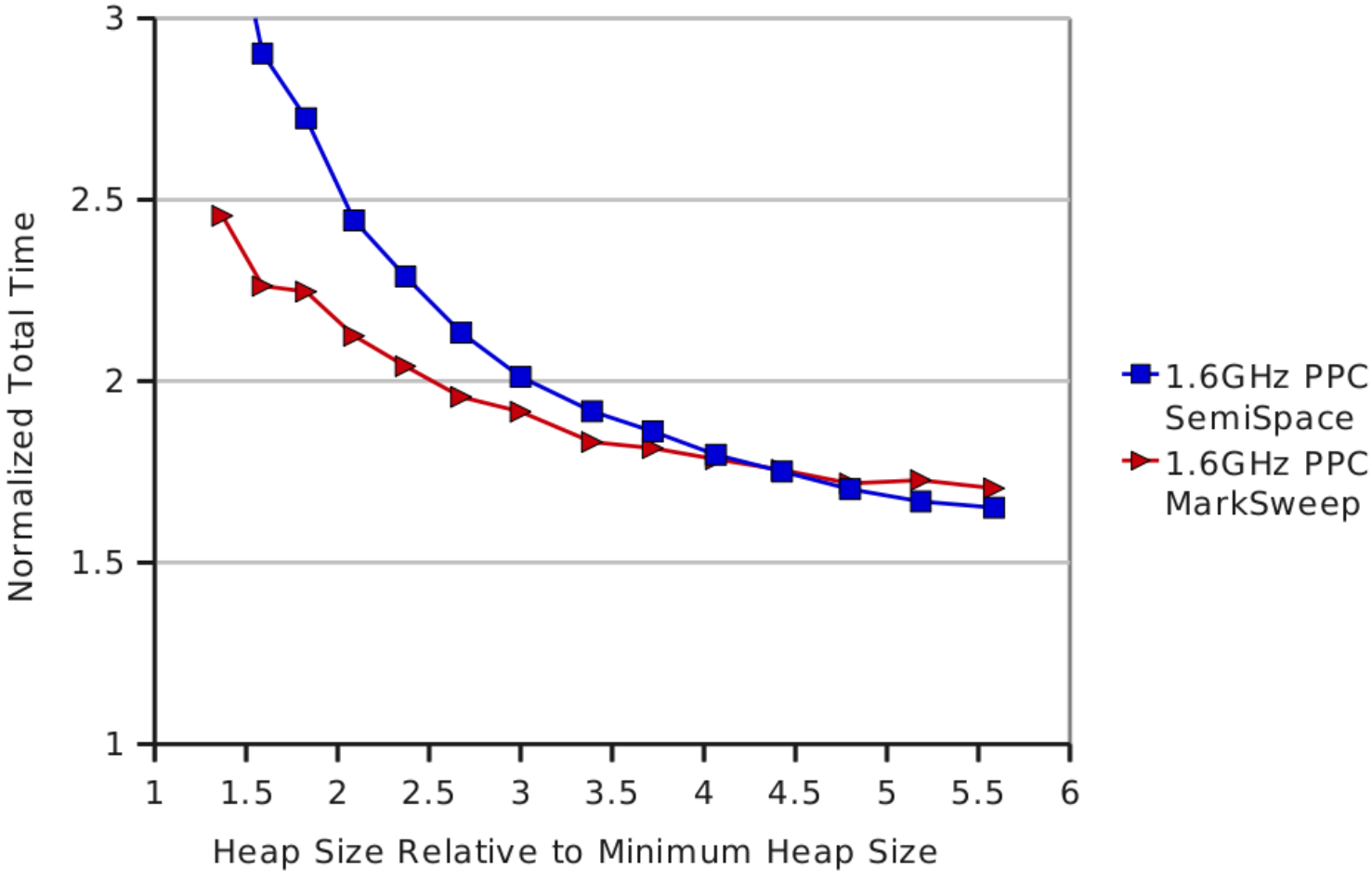




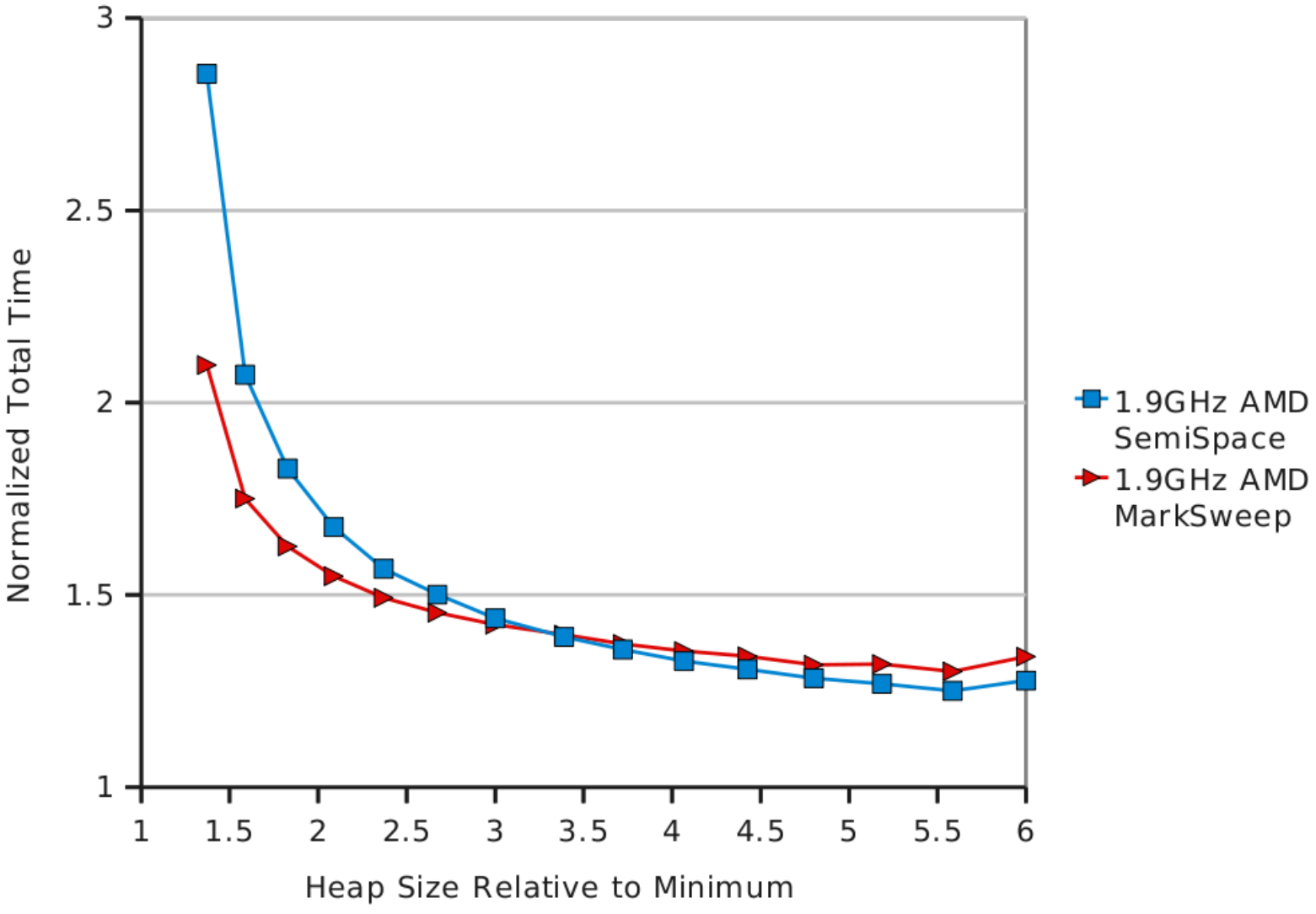
Results: Architecture Influences

- Architecture Influences
  - Crossover point: When SemiSpace outperforms MarkSweep
  - Limited space versus Locality tradeoff

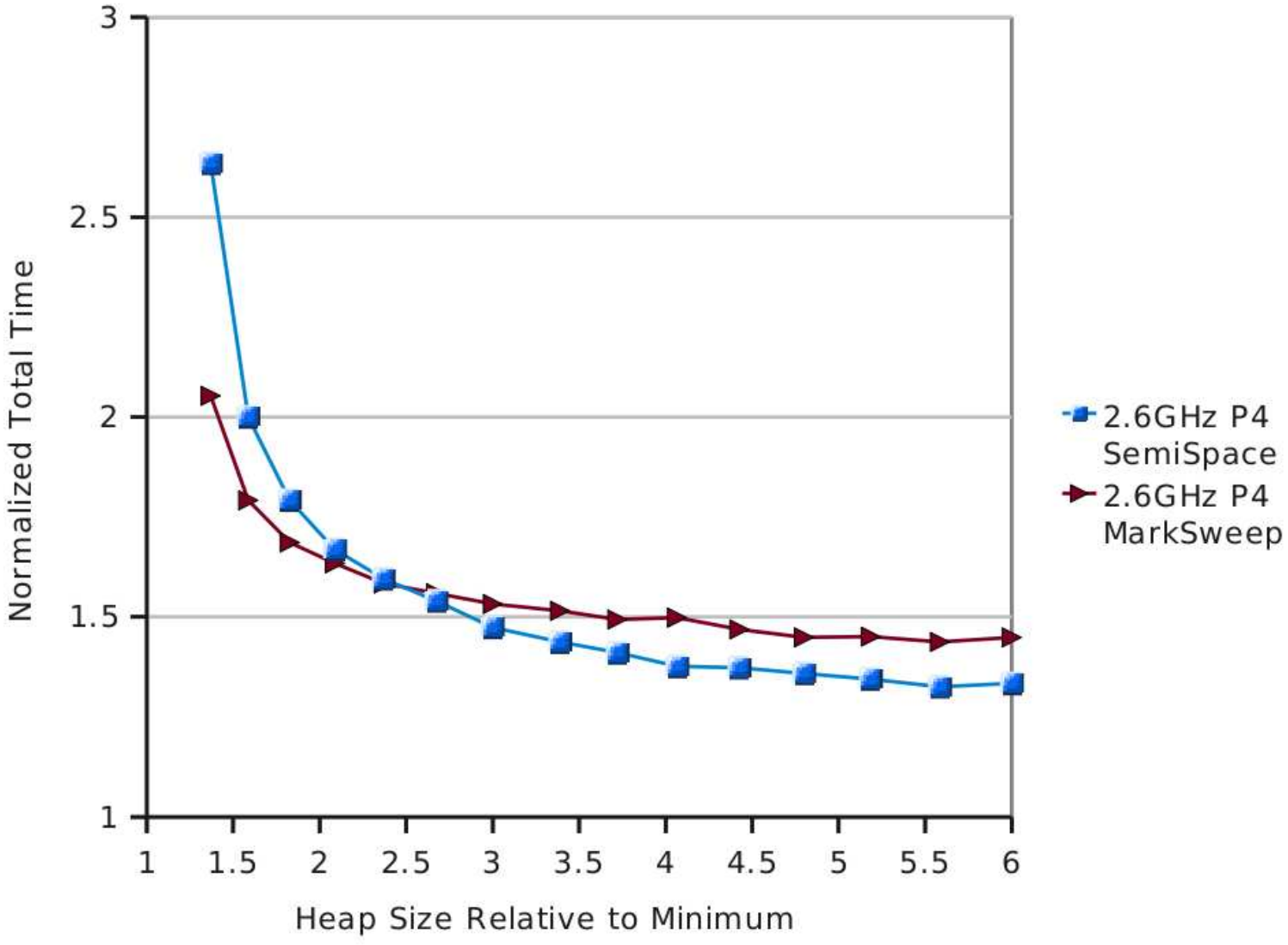
Results: Architecture Influences, PPC 1.6GHz



Results: Architecture Influences, Athlon 1.9GHz



Results: Architecture Influences, P4 2.6 GHz



Myths and Realities

- Contiguous is better than Free-list
  - Allocation is 11% faster, total improvement 1%
  - Locality improves mutator performance by 5-15% (SS vs MS)
- Tracing is usually better than Reference Counting
  - Only live objects are touched
  - Locality improves
  - RC can be useful for mature objects, when most of the heap consists of live objects
- Sensitivity to Heap Size
  - Determines collection frequency
  - Small heaps: MarkSweep, Modest to large heaps: SemiSpace

Myths and Realities

- Generational better than Whole-heap
  - Write-barrier over head is 1-14%, 3.2% average
  - Collection time benefits outweigh the write-barrier overhead
  - Locality of nursery: spatial
  - Locality of mature objects: temporal
- Nursery size
  - Fixed overhead of scanning roots (about 64KB)
  - Need not be matched to L2 cache size (512KB)
  - Should depend on fixed overhead (4-8MB)
  - If large, collection cost degrades performance (>8MB)

Impact on the GC Research Community

- Compare generational GC with explicit memory management
  - GC is competitive with explicit memory management
  - 5x memory, outperforms
  - 3x memory, 17% slower on average
- GC can do slightly better than explicit memory management in large heaps

Some more references

- www.cs.utexas.edu/users/mckinley/395Tmm/talks/May-4-MMTk.ppt
- M Hertz and E Berger, Quantifying the Performance of Garbage Collection vs. Explicit Memory Management, OOPSLA'05