Concurrent GC Leveraging Transactional Memory

Phil McGachey
Ali-Reza Adl-Tabatabai
Richard L. Hudson
Vijay Menon
Bratin Saha
Tatiana Shpeisman

Introduction

- Moore’s Law leading to multi-cored chips
- Harder to exploit than raw CPU power
- Concurrent programming becomes common
- Need simplified programming models

Multi-Cored GC

- Stopping the world is unfeasible
  - Overhead of pausing operation
  - Non-parallel code in collector
  - Growing heap sizes
- Clear call for concurrent garbage collection

Concurrency Control

- More threads lead to more interactions
- Locks are already difficult to reason about
- Push for transactional memory
- Transactional Integrity
  - Strong atomicity - TM system responsible
  - Weak atomicity - Programmer responsible

Synergy

- Shared mechanisms
  - GC must observe modifications to objects
  - TM must detect conflicts
  - Barriers are required for both systems
- We leverage the overlap
  - Treat "to-space" objects as speculative

Our Goals

- Leverage strong atomicity infrastructure for GC
- Target desktop applications
  - Games, multimedia, VOIP
  - Not hard realtime
- Focus on pause time
  - Aim to keep 90% of pauses under 1ms

Implementing Atomicity

- STM with strong atomicity
  - No “non-transactional” memory accesses
- Version number for conflict resolution
  - Writes increment on commit
- Objects in transactions are write-locked
  - Lock can be anonymous

The GC Algorithm

- Don’t stop the world
  - Threads paused one at a time
  - Minimize work during each pause
- Copy a portion of the heap per GC cycle
- Designed to support parallelism
  - 1 GC thread per 10 application threads
  - Not necessary for current desktops

Mark Phase

- Pause threads one at a time
- Scan stack area
  - Runtime stacks
  - TM data structures
- Concurrently mark the heap
- Iterate until all reachable objects are marked
- Barrier prevents writes of unmarked pointers

Copy Phase

- Collect small region of heap
- Don’t pause the application
- Copy objects transactionally
- Read barrier follows forwarding pointers
- Write barrier updates pointers

Flip Phase

- Update pointers to forwarded objects
- Pause each thread individually
  - Scan stack area
  - Update forwarded pointers
- Concurrently flip the heap
- Same barriers as the copy phase

Pauses

- Phase changes
- Mark phase
  - Pause each thread to scan stack
  - Pause to guarantee no unmarked objects
- Flip phase
  - Pause to find unflipped pointers on stacks
  - Pause to guarantee no unflipped objects

Concurrent Copying

GC Thread

Application Thread

Concurrent Copying

GC Thread | Application Thread

Begin Copy

Concurrent Copying

GC Thread

Begin Copy

Copy Field A

Application Thread

Concurrent Copying

GC Thread

- Begin Copy
- Copy Field A
- Copy Field B

Application Thread

Concurrent Copying

GC Thread	Application Thread
Begin Copy	
Copy Field A	
Copy Field B	
Copy Field C	Write to Field A

Concurrent Copying

GC Thread	Application Thread
Begin Copy	
Copy Field A	
Copy Field B	
Copy Field C	Write to Field A
Write Forwarding Ptr	

Concurrent Copying

GC Thread	Application Thread
Begin Copy	
Copy Field A	
Copy Field B	
Copy Field C	Write to Field A
Write Forwarding Ptr	
	Read from Field A

Atomic Copying

- Copy operation must be atomic
- Wrap each object copy in a transaction
  - Strong atomicity avoids lost update
  - Prohibitively expensive
- Build on the STM infrastructure
- Favor application thread in conflicts

Transactional Copying

GC Thread

Store version #

Application Thread

Transactional Copying

GC Thread

- Store version #
- Copy Field A

Application Thread

Transactional Copying

GC Thread	Application Thread
Store version #	
Copy Field A	
Copy Field B	

Transactional Copying

GC Thread	Application Thread
Store version #	
Copy Field A	
Copy Field B	
Copy Field C	Write to Field A

Transactional Copying

GC Thread	Application Thread
Store version #	
Copy Field A	
Copy Field B	
Copy Field C	Write to Field A
Compare version #	

Transactional Copying

GC Thread	Application Thread
Store version #	
Copy Field A	
Copy Field B	
Copy Field C	Write to Field A
Compare version #	
	Read from Field A

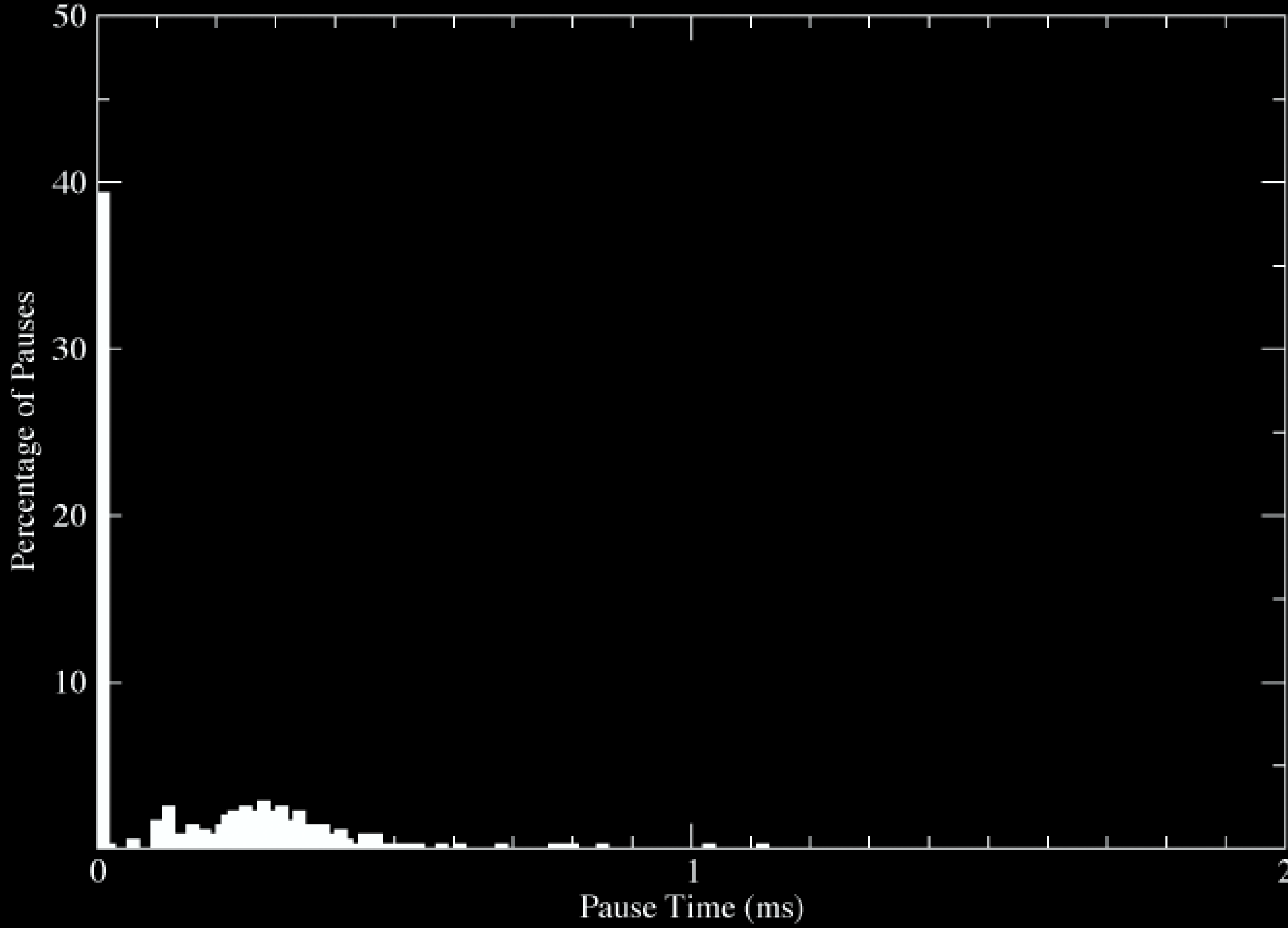
Barrier Synergy

- Strong atomicity barriers:
  - Logs reads and writes
  - Follows forwarding pointers
- Concurrent GC barriers
  - Prevents writes of unmarked pointers
  - Follow forwarding pointers

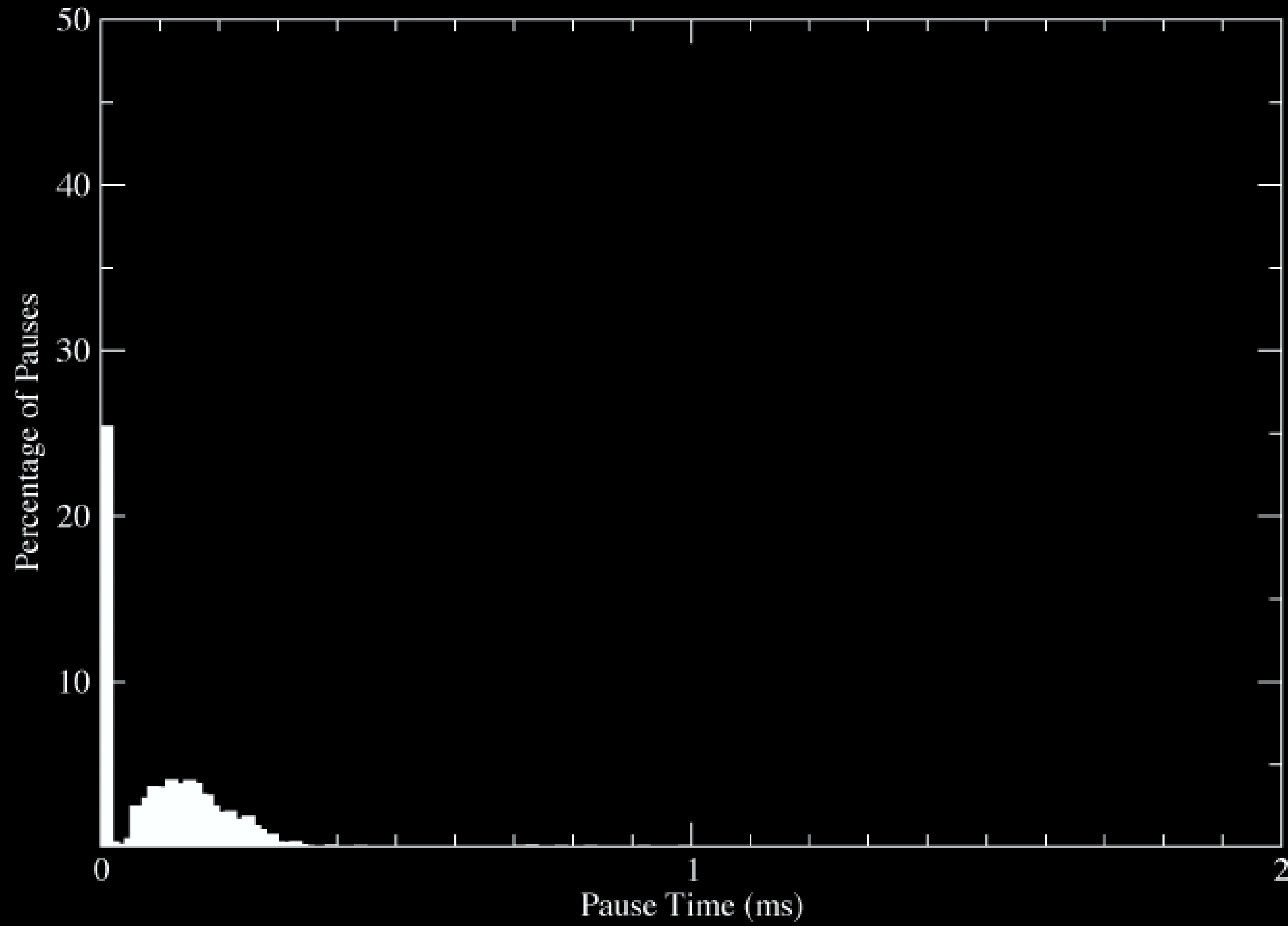
Experiments

- SPEC JVM98
- SPECjbb2000
- Atomicjbb
- AtomicTSP, AtomicOO7

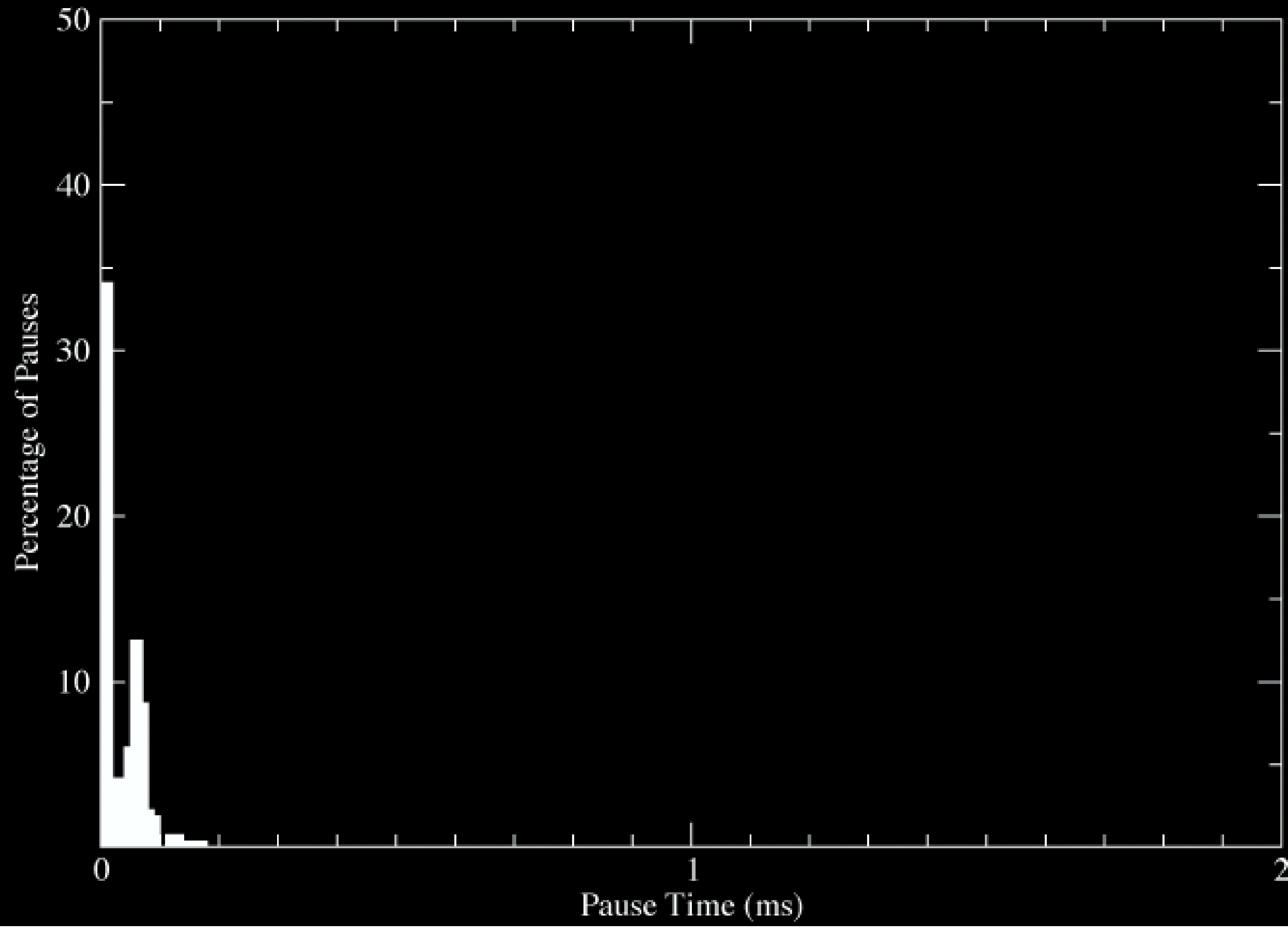
Javac



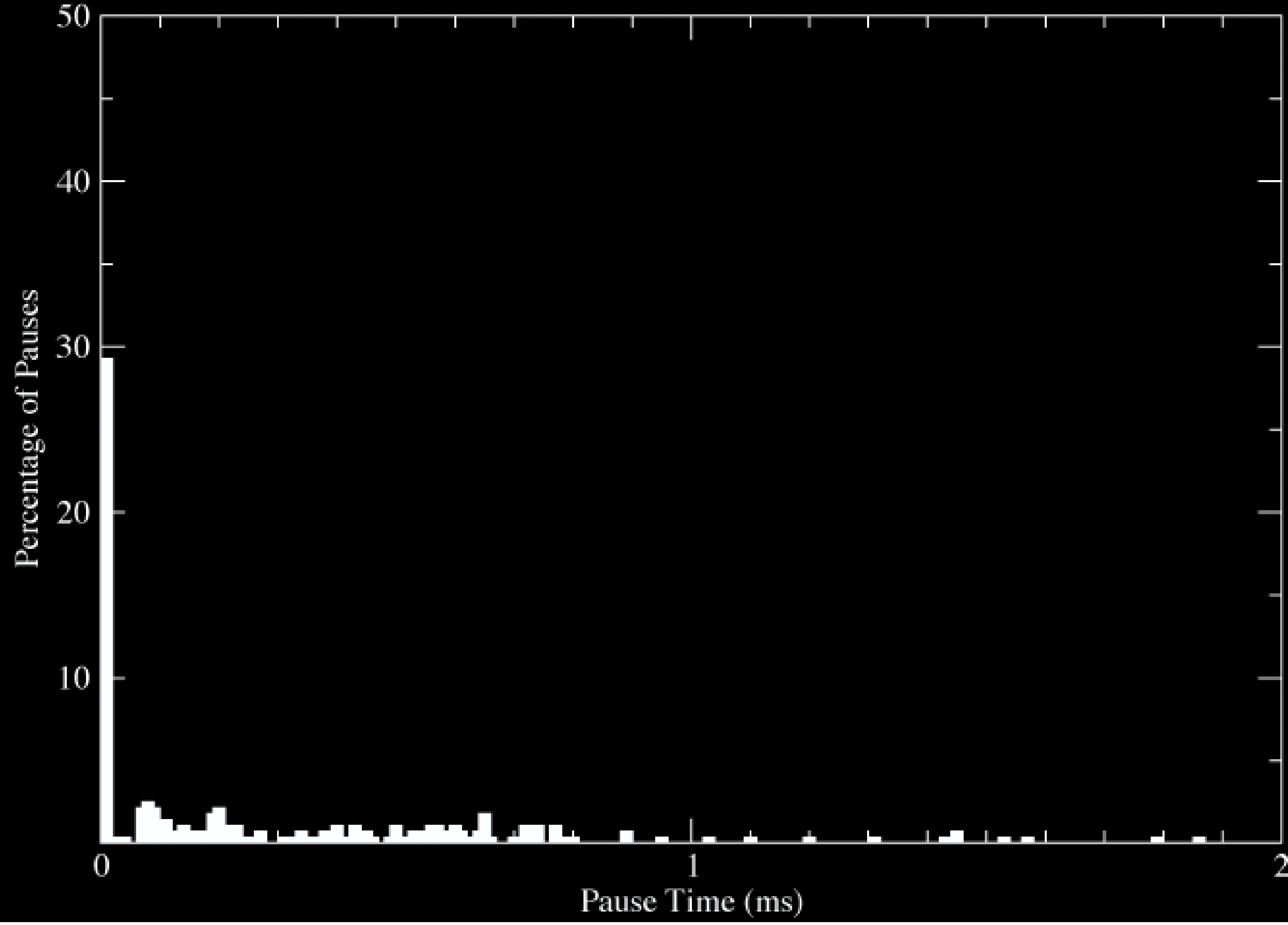
SPECjbb



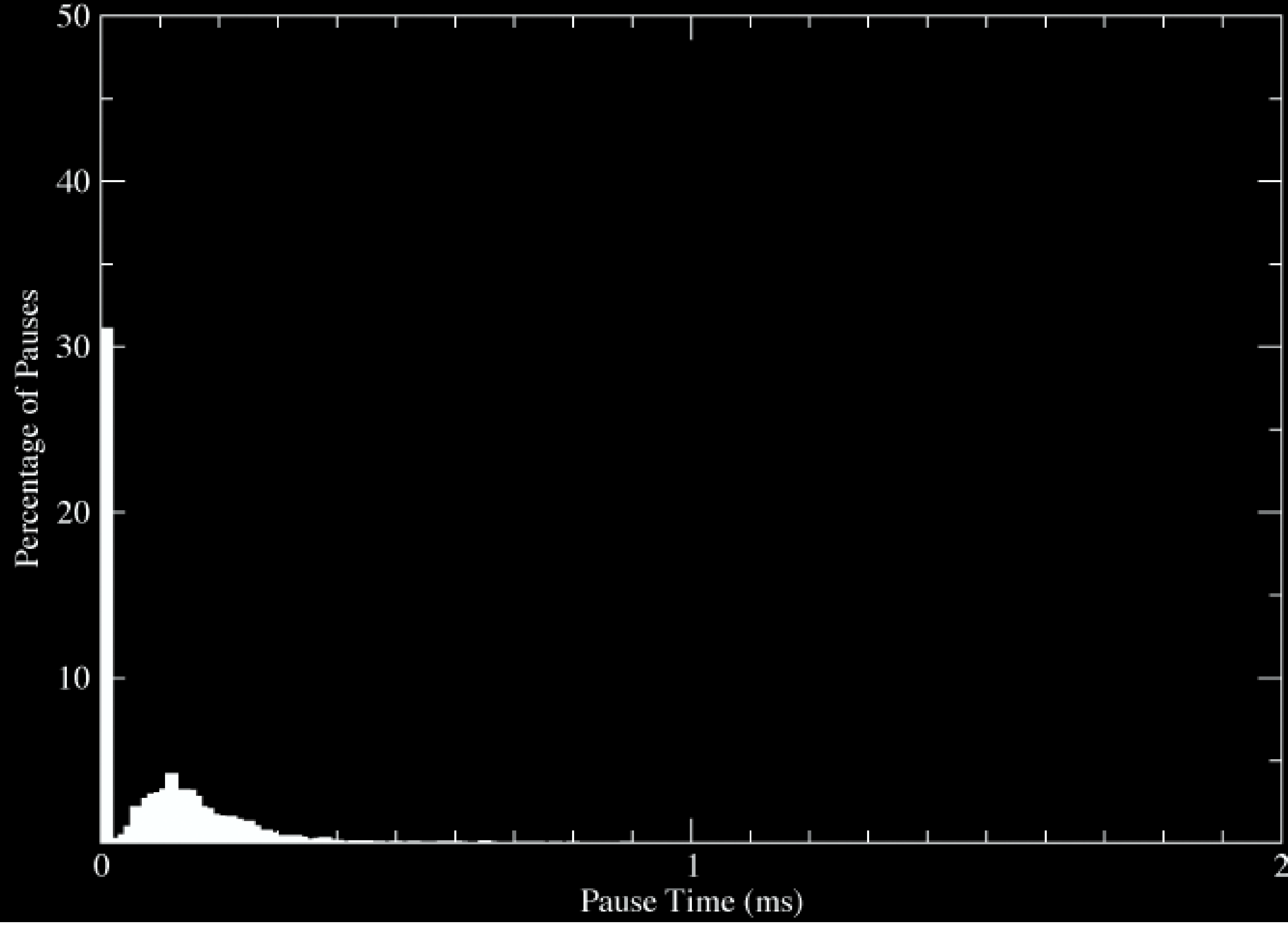
AtomicTSP



AtomicJBB



All workloads



[redacted]

Outliers

Benchmark	< 1ms	1..10 ms	10...100 ms	> 100 ms
201_compress	100.0%	0.00%	0.00%	0.00%
202_jess	100.0%	0.00%	0.00%	0.00%
209_db	100.0%	0.00%	0.00%	0.00%
213_javac	99.43%	0.57%	0.00%	0.00%
222_mpegaudio	100.0%	0.00%	0.00%	0.00%
227_mtrt	100.0%	0.00%	0.00%	0.00%
_228_jack	100.0%	0.00%	0.00%	0.00%
SPECjbb	99.72%	0.14%	0.14%	0.00%
AtomicOO7	100.0%	0.00%	0.00%	0.00%
AtomicTSP	100.0%	0.00%	0.00%	0.00%
Atomicjbb	85.00%	12.50%	2.14%	0.36%
Total	98.92%	0.85%	0.21%	0.02%
Target	≥ 90%	≤ 9%	≤ 0.9%	≤ 0.1%

Pauses per GC

Benchmark	Mark Phase	Flip Phase	Total
201_compress	2.0	2.0	4.0
202_jess	2.6	2.0	4.6
209_db	2.0	2.0	4.0
213_javac	2.7	2.0	4.7
222_mpegaudio	2.0	2.0	4.0
227_mtrt	3.7	2.9	6.6
_228_jack	2.1	2.0	4.1
SPECjbb	5.7	2.7	8.4
AtomicOO7	3.6	2.0	5.6
AtomicTSP	2.0	2.0	4.0
Atomicjbb	4.0	2.0	6.0
Average	2.9	2.1	5.1

Time In Each Stage

Benchmark	Mark Phase	Copy Phase	Flip Phase	Total
201_compress	0.19%	0.08%	0.26%	0.53%
202_jess	0.91%	0.37%	1.18%	2.45%
209_db	0.43%	0.14%	0.44%	1.00%
213_javac	0.82%	0.17%	0.82%	1.81%
222_mpegaudio	0.22%	0.08%	0.27%	0.57%
227_mtrt	1.81%	0.61%	2.02%	4.44%
_228_jack	0.77%	0.36%	0.58%	1.71%
SPECjbb	1.27%	0.27%	1.02%	2.56%
AtomicOO7	0.04%	0.01%	0.03%	0.08%
AtomicTSP	0.51%	0.00%	0.00%	0.51%
Atomicjbb	1.37%	0.52%	2.39%	4.28%
Average	0.76%	0.24%	0.82%	1.81%