Hard Real-Time Garbage Collection for a Java Chip Multi-Processor

Wolfgang Puffitsch

wpuffits@mail.tuwien.ac.at

TU WIEN

TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

JTRES '11, September 26-28, 2011

Hard Real-Time Garbage Collection

- GC increases productivity
- GC improves safety
- Real-time GC matured for uniprocessors
- CMPs still challenging
    - True parallelism
    - Synchronization more expensive

GC Phases

- Start GC cycle
- Scan local and static variables for references
- Trace objects and defragment
- Reclaim unvisited objects

Challenges

- Scan local variables
    - Stack (and registers)
    - No barriers wanted
    - Minimal disruption for application
- Eliminate fragmentation
    - Cannot allow fragmentation
    - Fixed block layout has overheads
    - Relocate objects without disruption application

System

- Java Optimized Processor (JOP)
- Memory accesses
    - Time-division multiple access
    - Round-robin
- Caching
    - Method and stack cache
    - Data caches being worked on
    - Not in this paper
- Scheduling
    - Partitioned (threads pinned to one core)
    - Fixed-priority (rate/deadline monotonic)

Locking

- Low-level locking
    - Single global hardware lock
    - Round-robin
    - Similar cost as compare-and-swap
- High-level locking
    - Per-object locks
    - FIFO queuing
    - Spin at top priority
    - Similar to MSRP

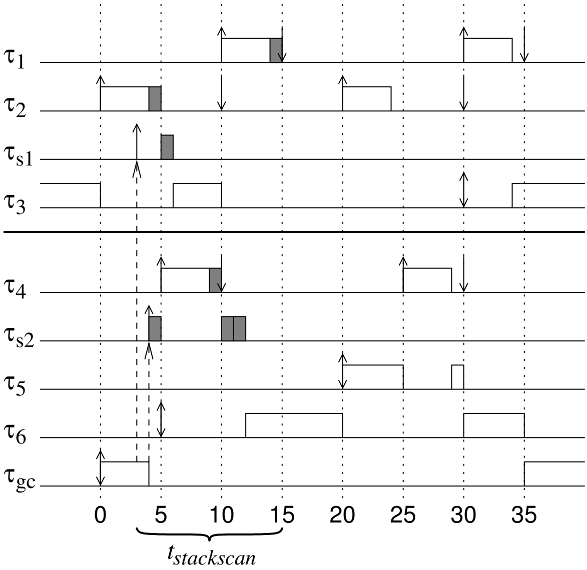
GC Algorithm

- Copying collector
  - Copies between to- and from-space
- Time-based
- Incremental
- Concurrent
- Not parallel
  - Memory bound task
  - Increase bandwidth in arbiter if needed
- Handle-based object layout
  - One level of indirection for field accesses

Stack Scanning

- Collect references in local variables
- Basic idea: scan stacks at end of job
    - `waitForNextPeriod()`
    - Stack is shallow ⇒ low overhead
    - Instant is known ⇒ no disruption
    - Need to wait until tasks have finished a period
- End-of-job for high-frequency tasks
- Event handler scans lower-frequency tasks

Stack Scanning Example



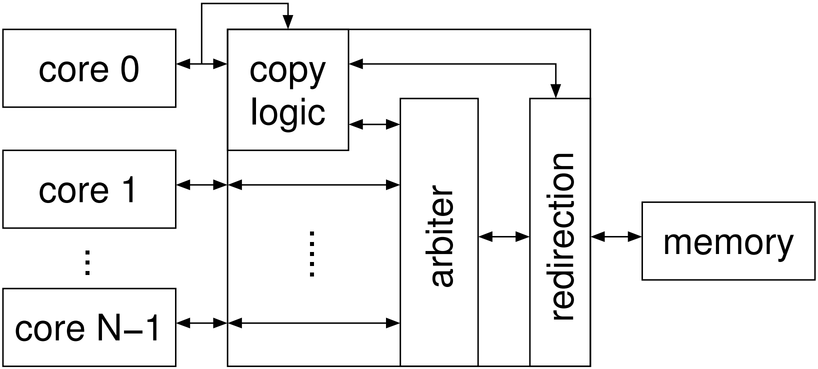
Stack Scanning Bounds

- σ ... self-scanning tasks
- ρ ... stack scanning events
- $t_{stackscan} \leq \max(\max_{\tau_i \in \sigma}(T_i + R_i), \max_{\tau_i \in \rho} R_i)$

Copying Support in Hardware

- Preemptible, transparent, consistent
- Redirect accesses to object being copied
- Must not disturb other accesses
- Redirection for *all* cores

Copying Hardware Block Diagram



Implementation

- JOP CMP, 8 cores, TDMA
- 3 cycles per individual memory access
- 26 cycles worst-case latency
- Pipeline memory arbiter
  - Sacrifice one cycle latency: 26 → 27 cycles
  - Relax critical path: 93.5 → ca. 100 MHz
  - Higher frequency even without copy unit
  - Negligible overhead for copy unit: 350 of 45k LCs

jPapabench

- Control unmanned aerial vehicle
- Complex real-time benchmark
- Other benchmarks too complex or too simple
    - Memory allocation, multiple threads
- Manual partitioning
- Some tasks scan their own stack
- Event handlers to scan other stacks

Partitioning

	Core	0	1	2	3	4	5	6	7
Priority	High				F1	F2			
		A3			F3	F4	A1	A2	
		SE	SE	SE			SE	SE	SE
		A4	S1	S2			A7	A6	A5
	Low	GC		S3					

Analysis

- Reasonable WCET for most tasks
- Soft-float problematic, but limited
- WCET of GC overly pessimistic
  - Annotations not expressive enough
  - Copying relatively cheap

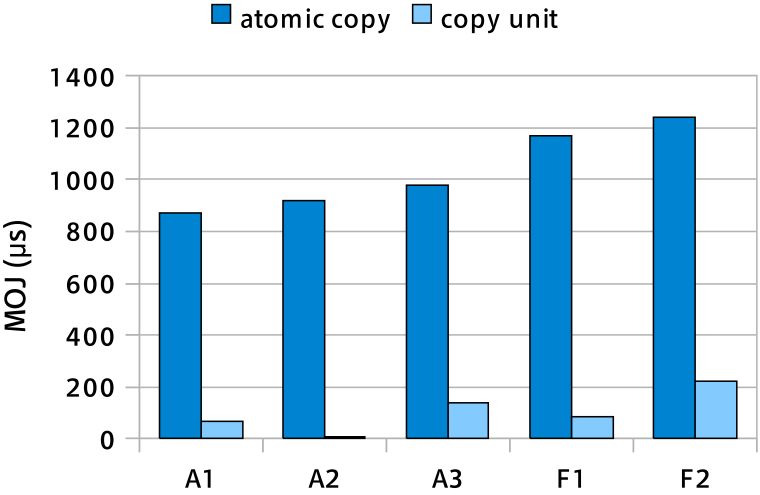
Measurements

- Measured response time (MORT) and release jitter (MOJ)
- Varied offsets, multiple runs
- Low jitter only for high-priority tasks
- 200 μs due to preemption and locking

Detailed Measurement Results

#	atomic copy		copy unit	
	MORT (μs)	MOJ (μs)	MORT (μs)	MOJ (μs)
A1	1 826	870	533	65
A2	3 904	921	2 622	9
A3	989	982	145	139
A4	3 536	3 529	2 174	2 168
A5	22 835	381	24 793	12
A6	3 935	3 639	3 502	3 123
A7	3 449	1 832	2 461	964
F1	1 188	1 171	103	86
F2	1 261	1 239	246	224
F3	1 605	1 588	760	743
F4	4 225	1 863	2 407	764
S1	—	—	—	—
S2	39 900	414	38 524	76
S3	44 511	39 616	43 012	37 979

Jitter Measurement Results



Conclusion

- Stack scanning with both little overhead and reasonable timing bounds
- Hardware support for preemptible, transparent copying on CMP
- Considerably increased scheduling quality

Thank you for your attention!