

Reducing Interrupt Latency by Using the Cache

By

Dejan Bucar

The Work Was Performed at
Enea OSE Systems

Supervisors: Peter Sandström (Enea), Jan Lindblad (Enea), Vlad Vlassov (KTH)

Introduction

- Objective
 - Long interrupt latency
 - Use the cache to decrease it
- Overview of the research
 - Lock the interrupt handler
 - Measure the interrupt latency
 - Measure the system performance

Introduction

- Expectations
 - The interrupt latency should be decreased
 - The rest of the system performance should run slower
- Structure of the presentation
 - Overview of real time operating systems, caches and interrupts.
 - Case studies
 - Conclusions

Real Time Operating Systems

- The correctness depends on *when* the result is produced
- A fast system is not real time!
 - And a real time system is not fast...
- Two types of real time systems
 - Hard: all deadlines must be kept
 - Soft: missed deadlines are accepted

Real Time Operating Systems

- Used for critical applications
 - Control systems, in-flight computers
 - Medical Equipment
 - Embedded systems
 - Communications systems

Real Time Operating Systems

- Scheduling
 - Priority based
 - The highest priority process get to run
 - Deadline base
 - Earliest deadline first
 - Rate monotonic scheduling

Real Time Operating Systems

- Problems with hard timing constraints
 - Hardware
 - DMA
 - Cache
 - Interrupts
 - Memory system
 - Programming languages C, C++, Java, Pascal

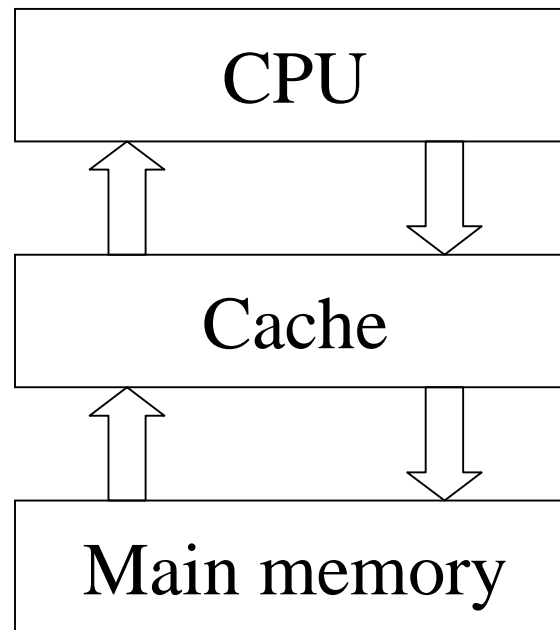
Real Time Operating Systems

- Conclusions
 - Hard real time systems hard to design using off-the shelf products
 - Soft real time systems more economic, use priority based scheduling

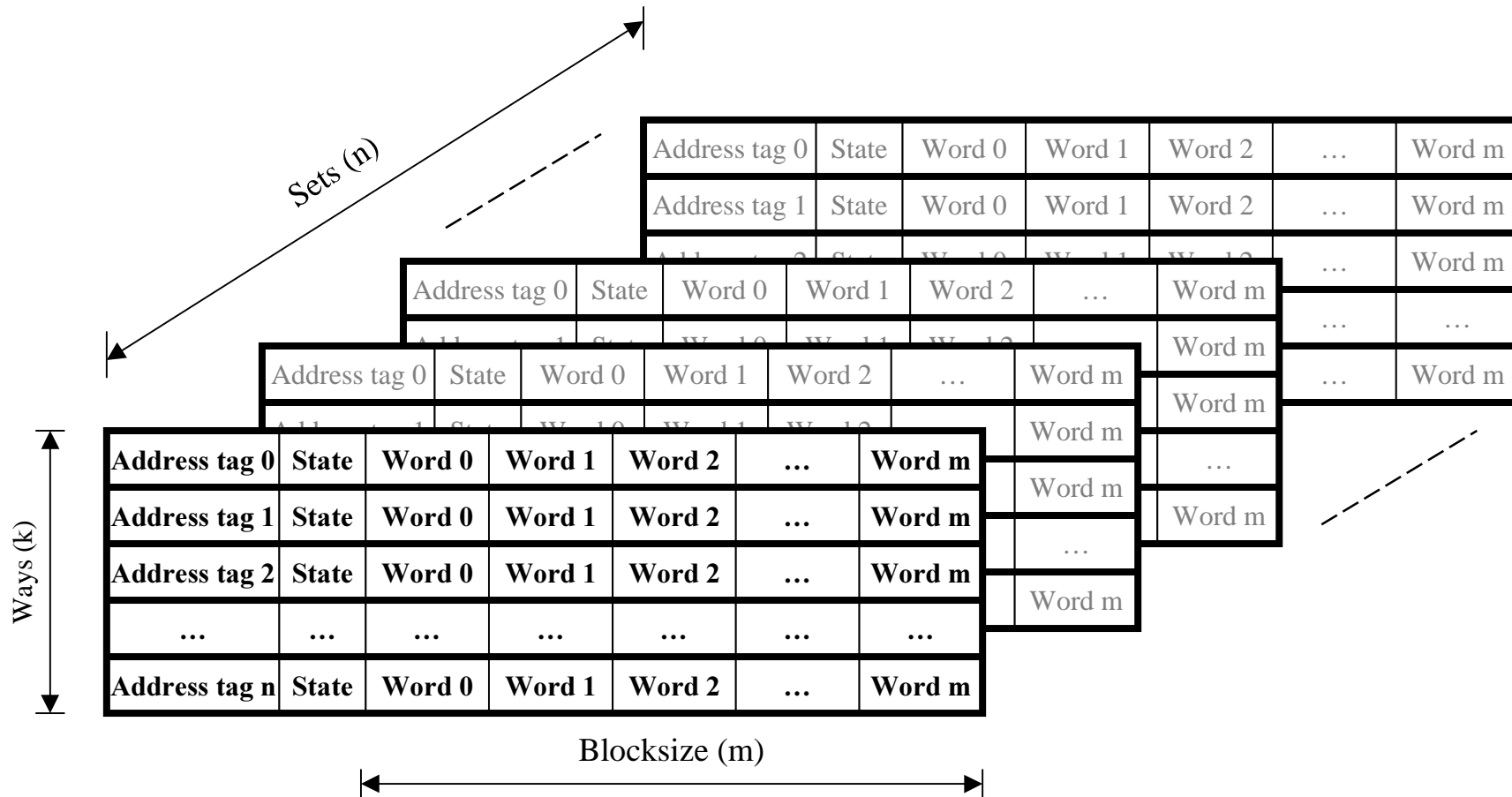
OSE (delta-kernel)

- Priority based scheduling
 - Prioritized processes
 - Background processes
 - Interrupt processes
- Runs on PowerPC, ARM, MIPS

Cache: What is it?



Cache: Overview



Cache: Parameters

- Set-associativity
- Blocksize
- Ways
- Size (= set-associativity · Ways · Blocksize · 32)
- Replacement-algorithms (LRU, FIFO, etc.)

Cache Aware Programming

- Static optimization
 - Done before or at compile-time
- Dynamic optimization
 - Profile execution and recompile

Cache Locking

- Permanently "lock" instructions or data into the cache
- Increases performance of locked code
- Makes the cache smaller

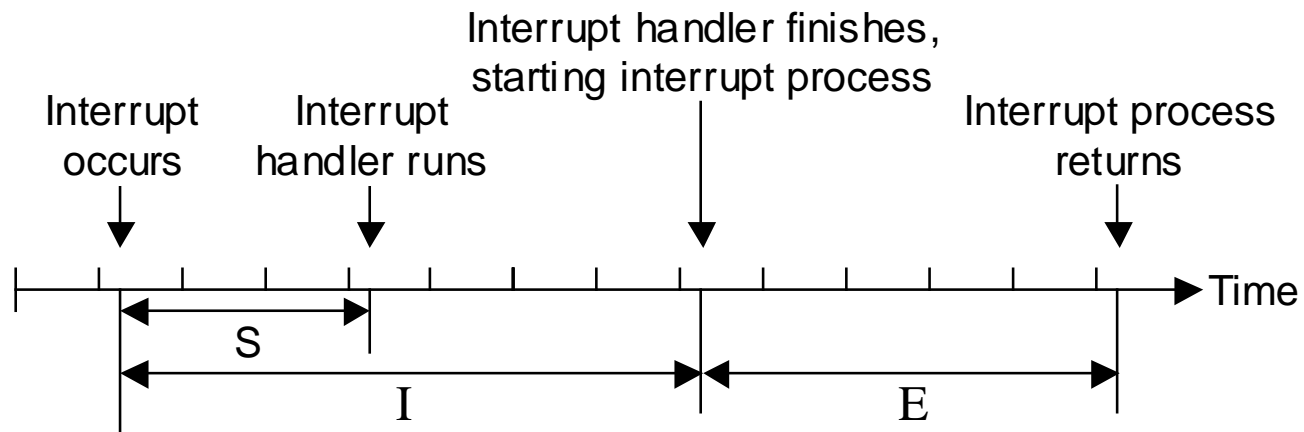
Interrupts

- Used by external devices to signal new data
- Problem in real time systems
 - Can occur at any time
 - Can be signalled at any rate

Interrupts

- Solutions in real time systems
 - Hard real time systems: disable them, use polling instead
 - Soft real time systems: make them as short as possible

Interrupt Latency



- I = interrupt latency
- E = interrupt process execution time
- S = time to start the interrupt handler

Interrupts in OSE

- Higher priority than other processes
- Should be as short as possible
- Interrupt processes are written by users

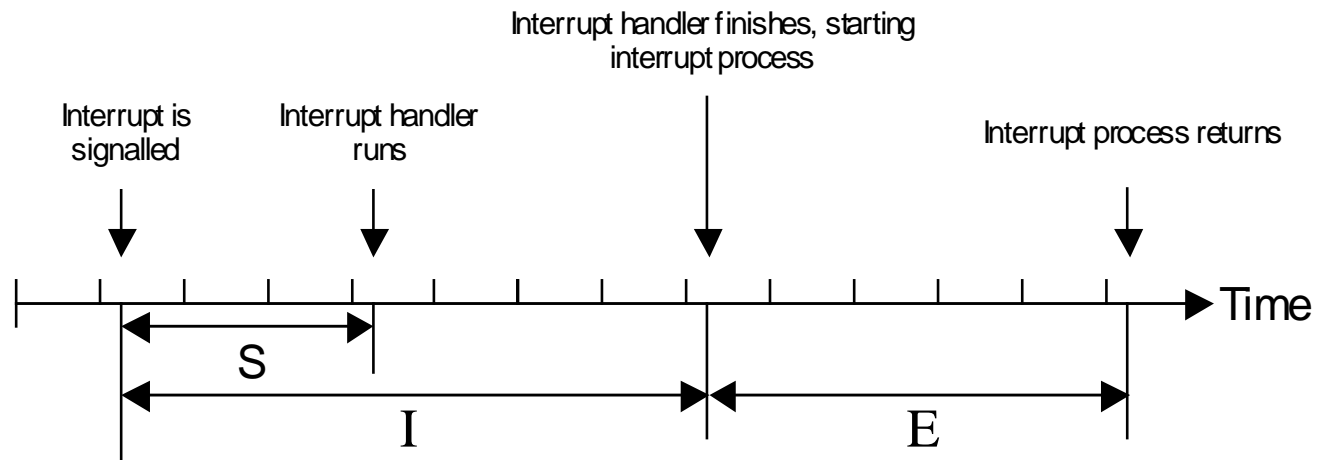
Reducing Interrupt Latency

- Lock cache handler into the cache
- Reduce time spent in interrupt disabled regions
- Reduce number of interrupt disabled regions

Lock interrupt handler into Cache

- Identify what to lock
- Measure the interrupt latency before and after locking
 - Maximum, minimum, average and jitter
- Measure the system performance

Interrupt Latency



- I = interrupt latency
- E = interrupt process execution time
- S = time to start the interrupt handler

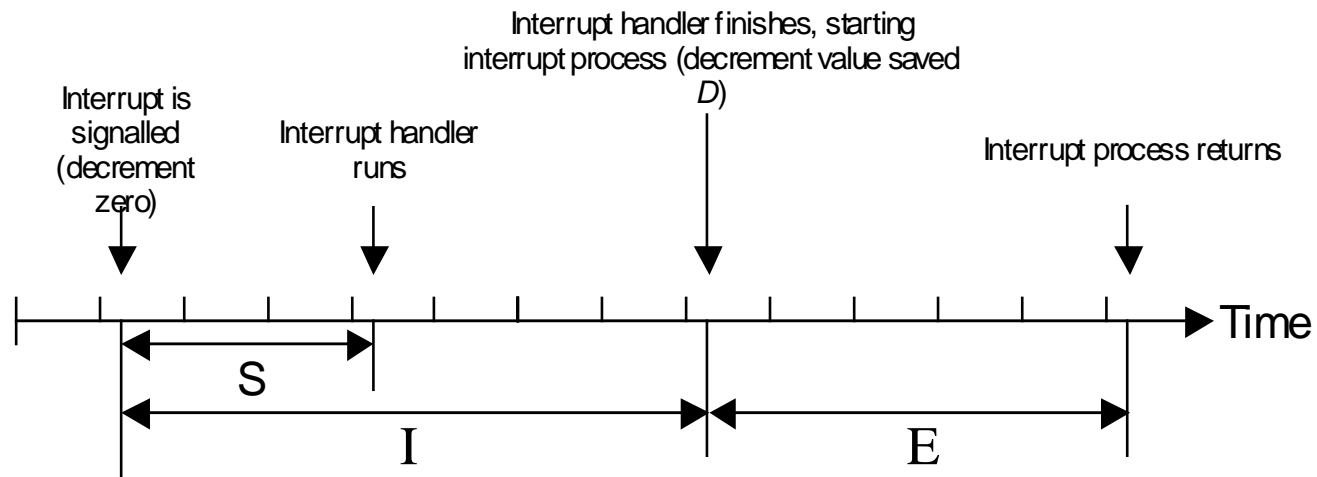
Locking the interrupt handler reduces I - S

Performance Evaluation

Measuring the interrupt latency (PowerPC)

- Load the decrement register with an value
- When decrement reaches zero an interrupt is thrown
 - The decrement register keeps counting down going from 0 to 0xFFFFFFFF
- When the interrupt process is started save the value of the decrement register

Interrupt Latency



- I = interrupt latency = 0 - D
- E = interrupt process execution time
- S = time to start the interrupt handler

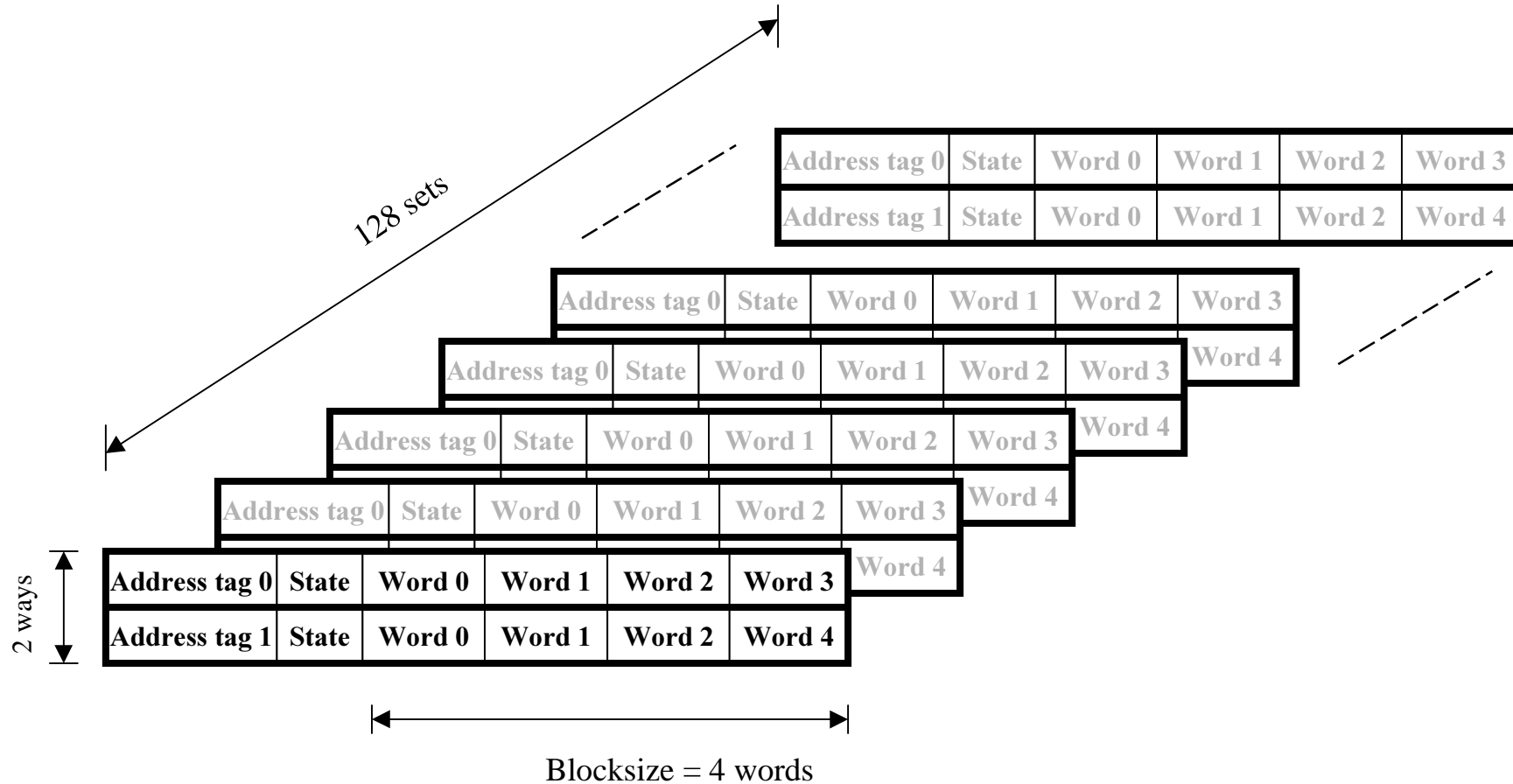
Performance Evaluation

- Measuring performance
 - Hard to measure performance
 - Dhrystone - simulates an average load

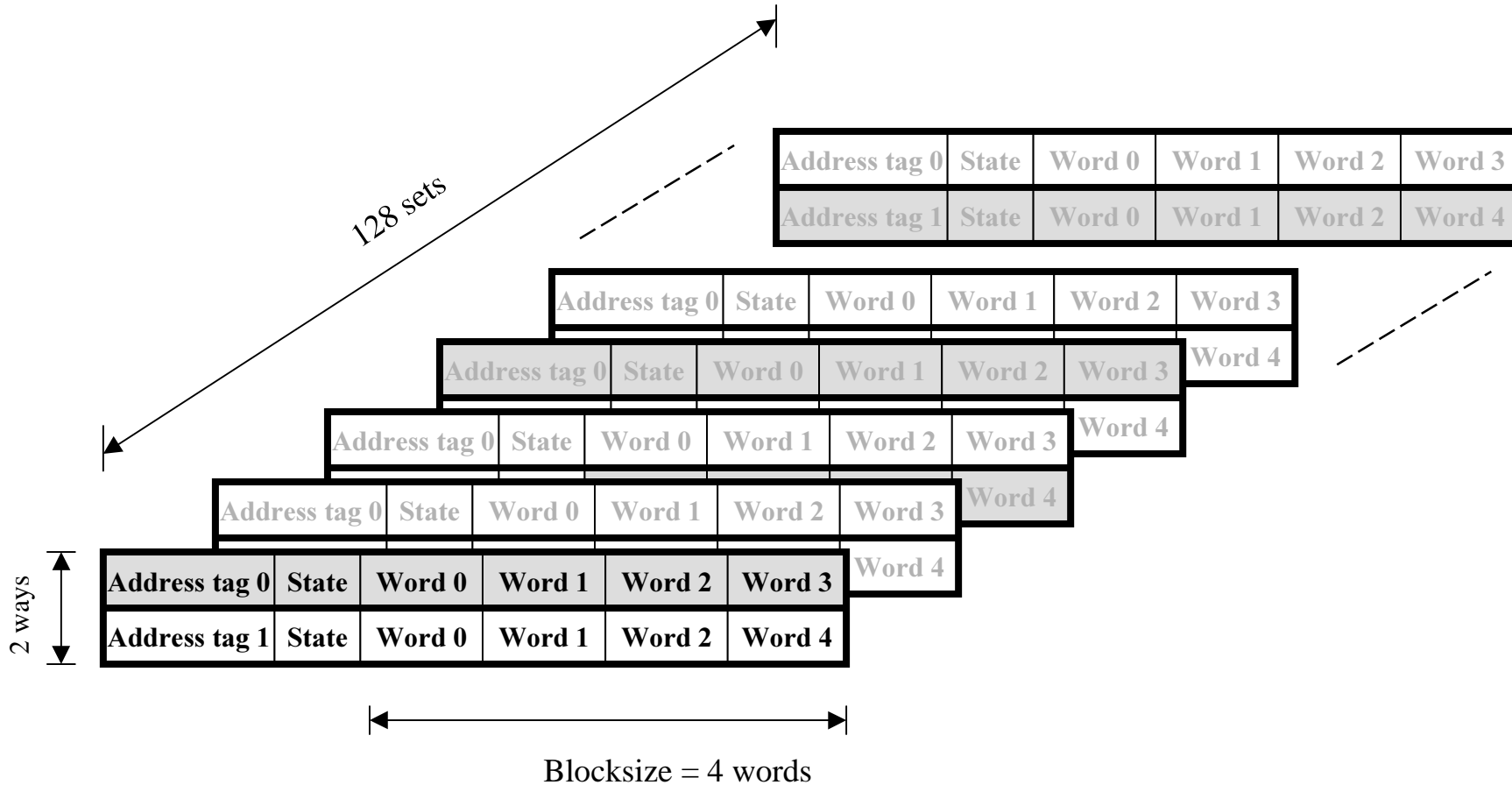
MPC860

- 4-Kbytes instruction cache
- 4-Kbytes data cache
- Trivial locking mechanism
 - Easy to implement
 - Individual blocks can be locked

MPC860: Cache

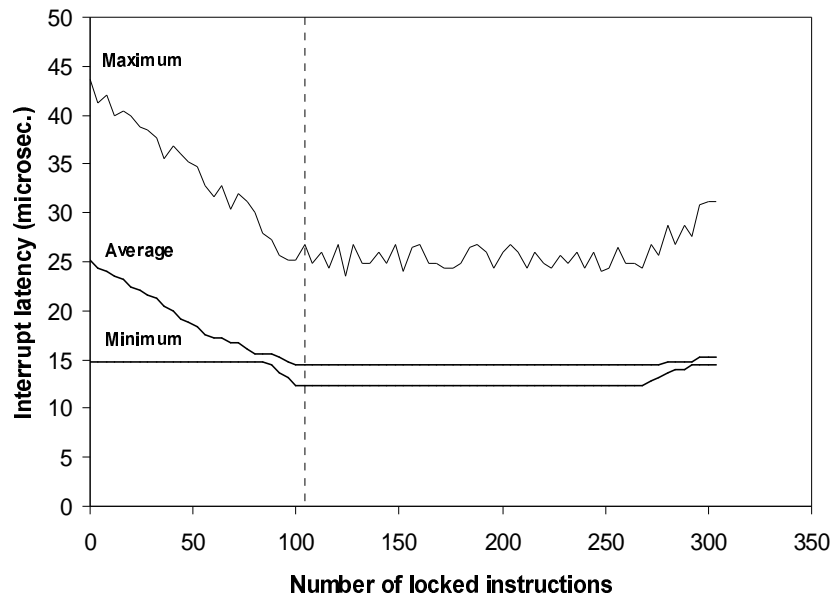


MPC860: Cache locking

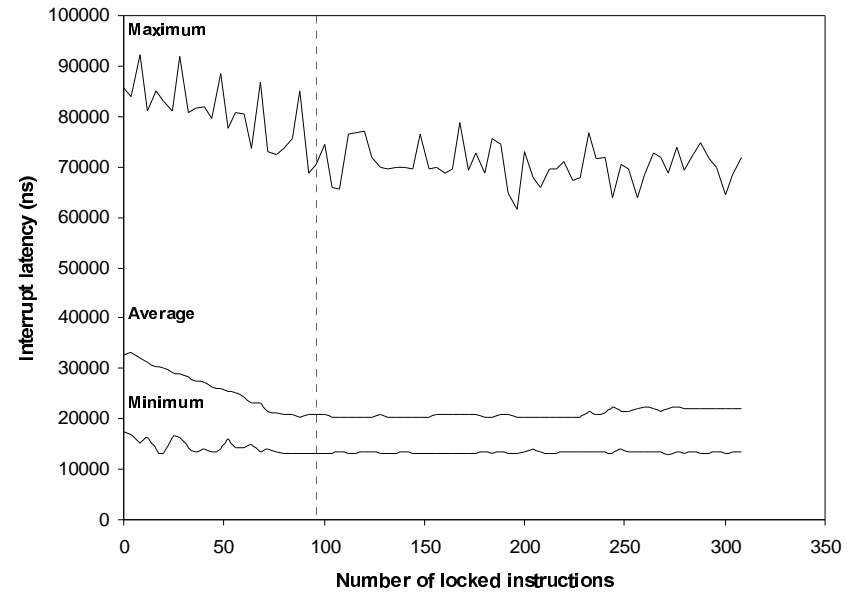


MPC860: Interrupt latency

Lightly loaded system

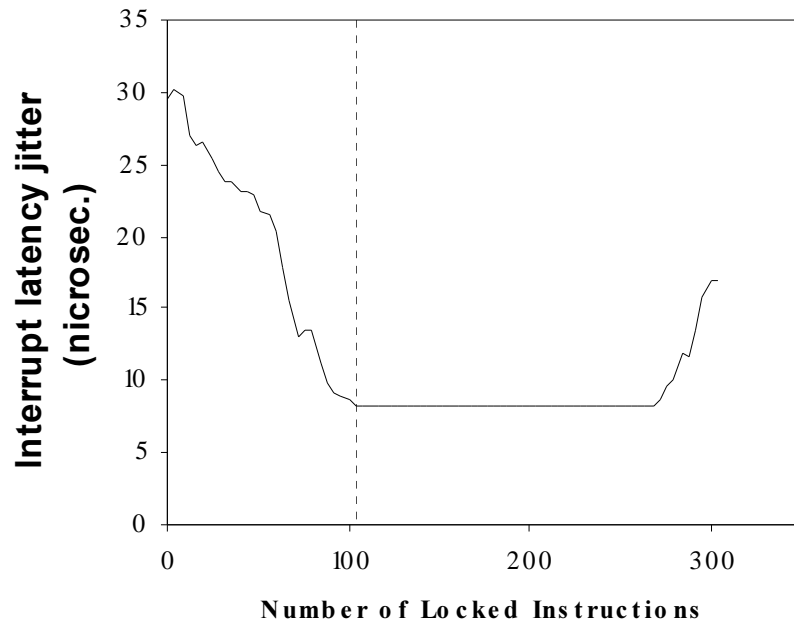


Heavy loaded system

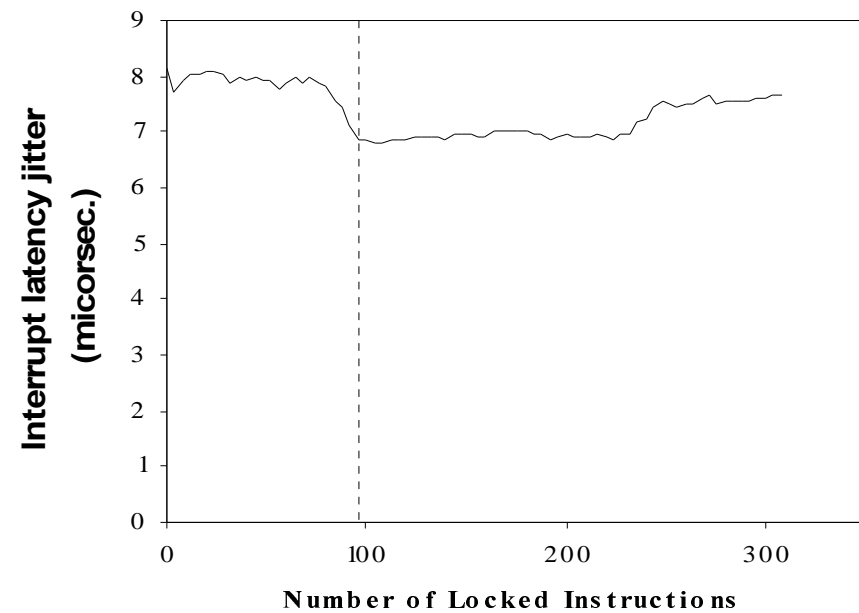


MPC860: Jitter

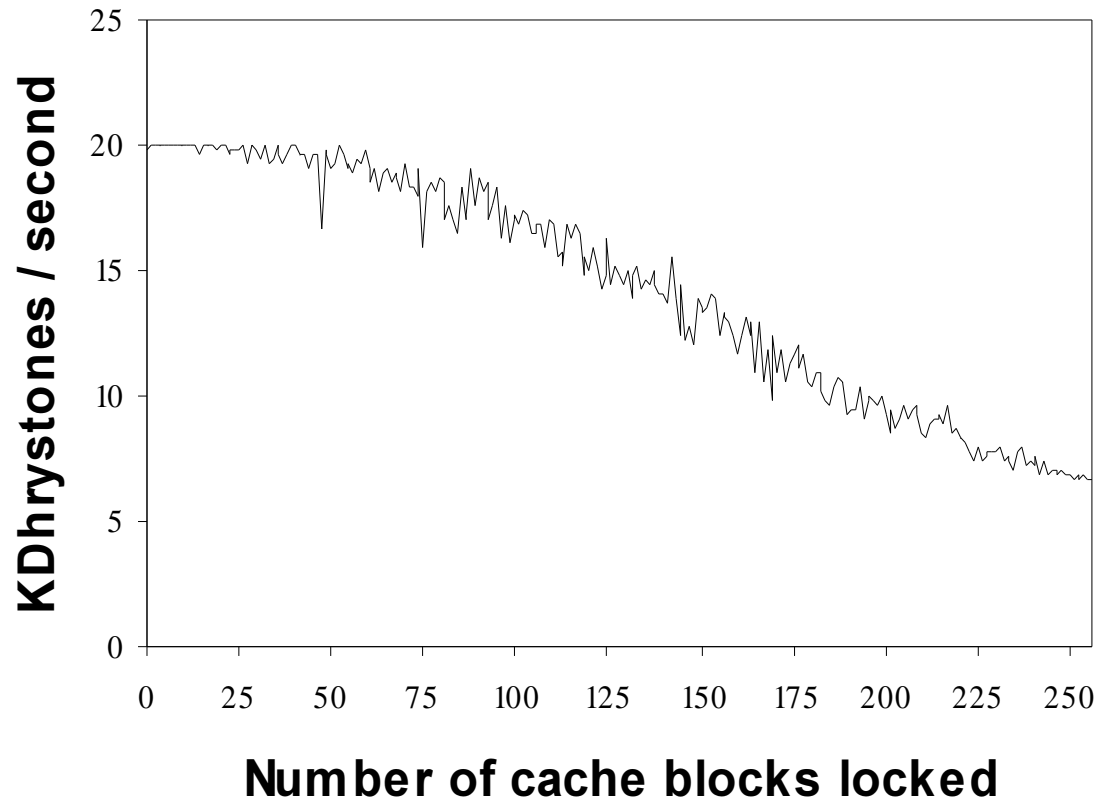
Lightly loaded system



Heavy loaded system



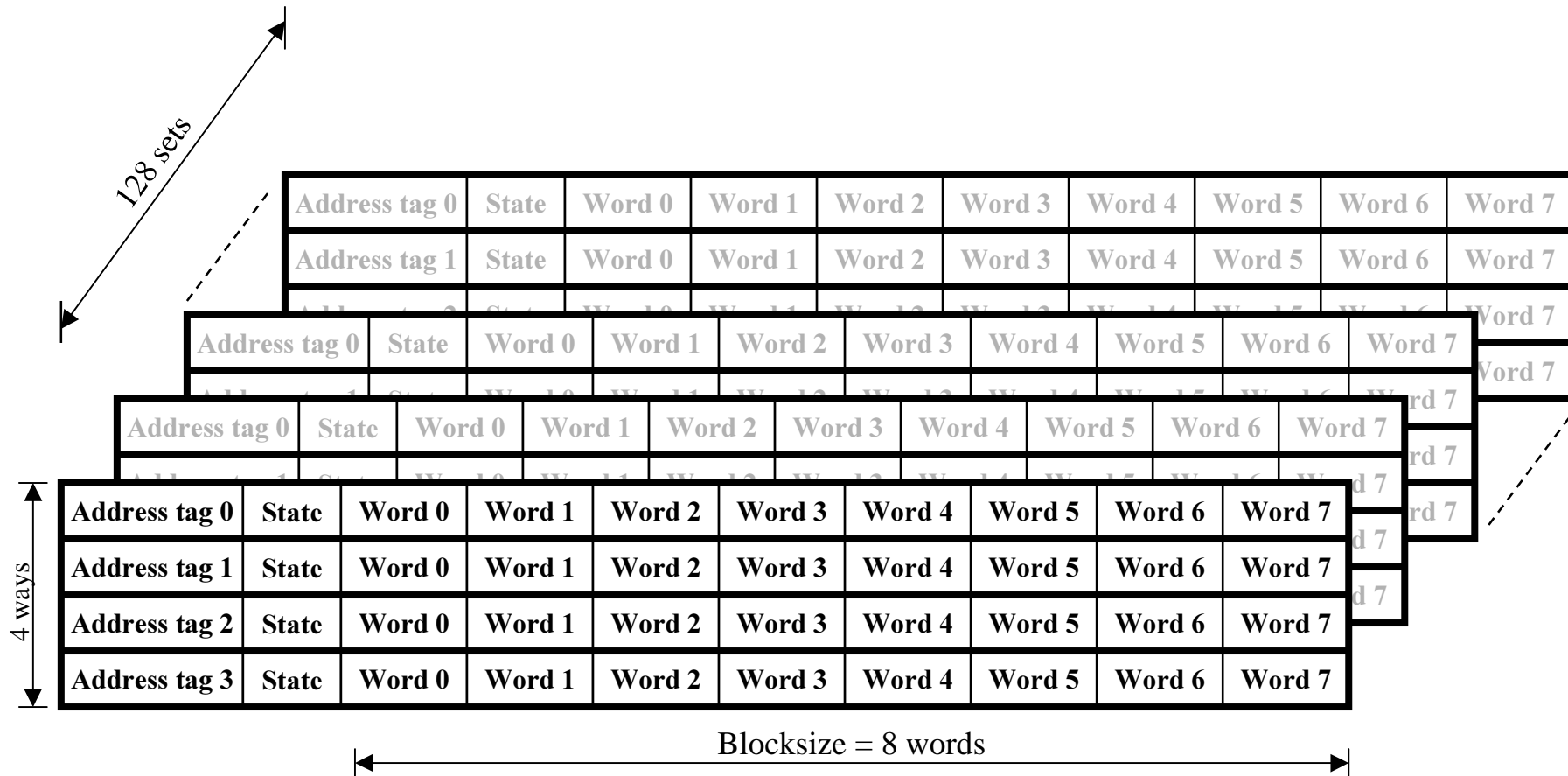
MPC860: Smaller cache



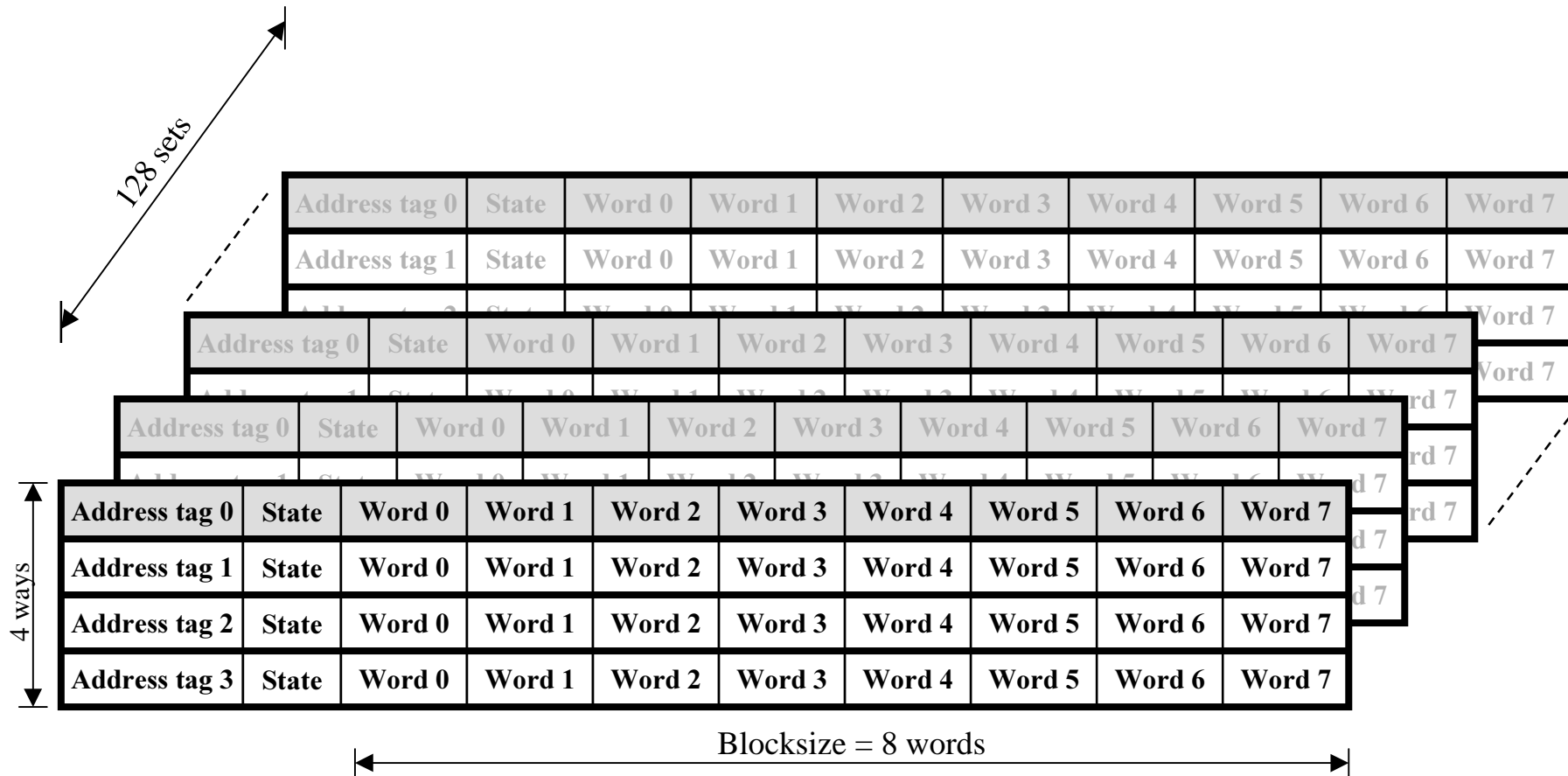
MPC8260

- 16-Kbytes of instruction cache
- 16-Kbytes of data cache
- Non-trivial cache locking
 - No explicit instructions to load instructions
 - Pre-fetch loading must be used
 - Only ways can be locked, not blocks

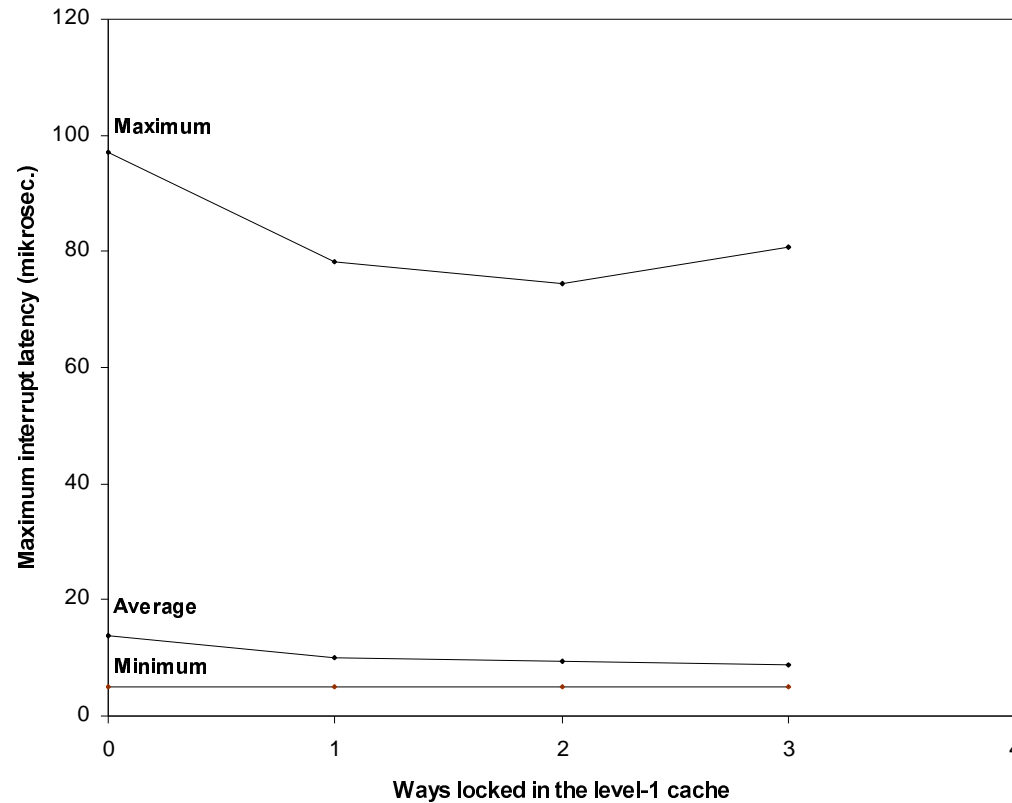
MPC8260: Cache



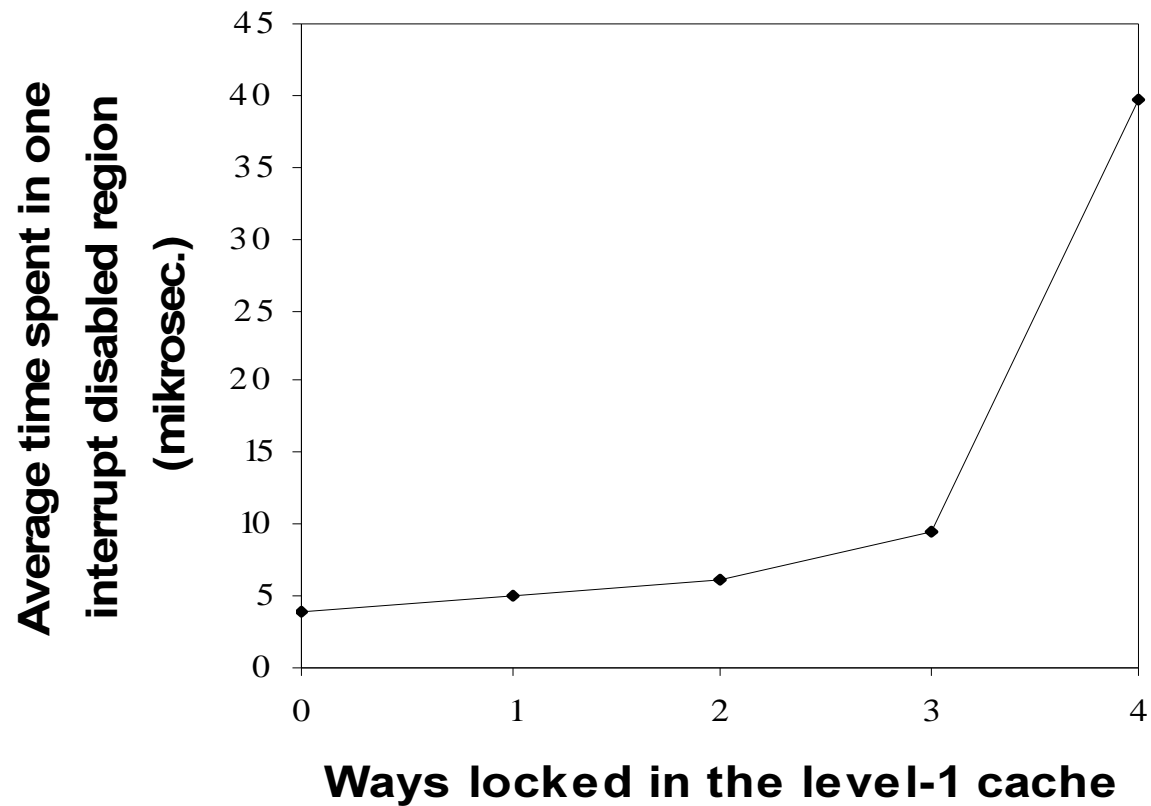
MPC8260: Cache locking



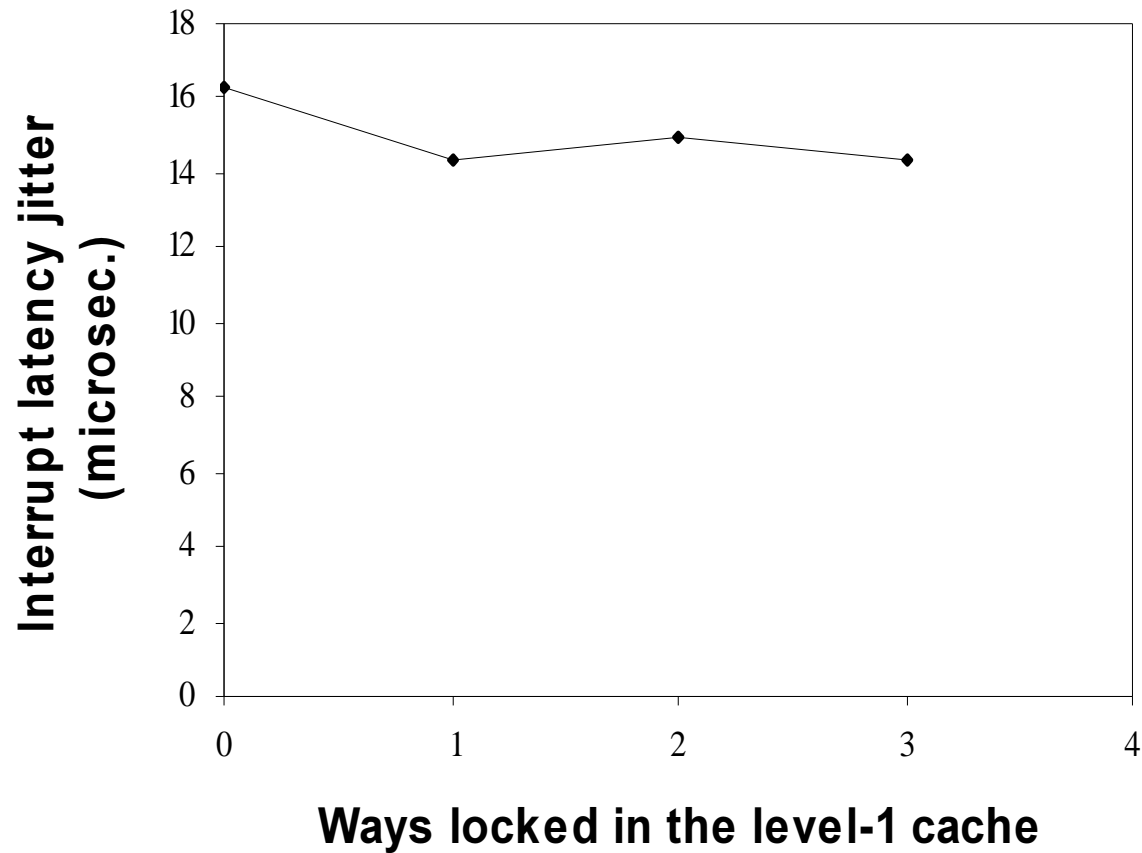
MPC8260: Interrupt latency



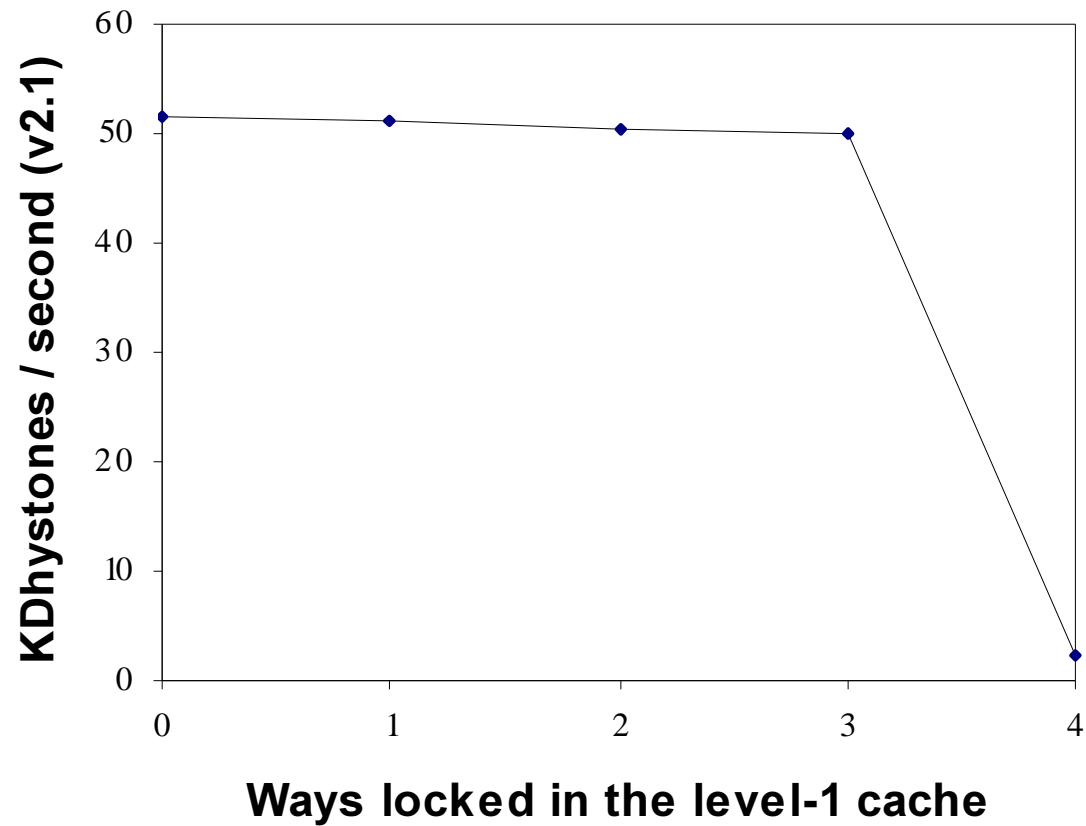
MPC8260: Disabled Interrupts



MPC8260: Jitter



MPC8260: Smaller cache



Conclusions

- Locking decreases the interrupt latency
- Rest of the system slows down
- Depends on the system
 - Number of interrupts signalled
 - The load in the system
 - Critical timing constraints

Future Work

- Evaluation of real time languages
 - Can they be used with existing hardware
- Reducing time spent in interrupt disabled regions
 - Investigate lock-free algorithms
 - Optimize code, keep regions small

Questions!