Multicore ARM Processors for Safety Critical Avionics

Gary Gilliland
DDC-I Technical Marketing Manger

Gary Gilliland

- Technical Marketing Manager at DDC-I
- 25+ years experience in embedded design, avionics and RTOS
- Electrical Engineering Degree from University of Texas

DDC-I Inc.

- Leading provider of mission/safety-critical software solutions for 30 years.
- Headquarters in Phoenix, AZ
  - World-wide presence
- Primary market: Certifiable avionics software

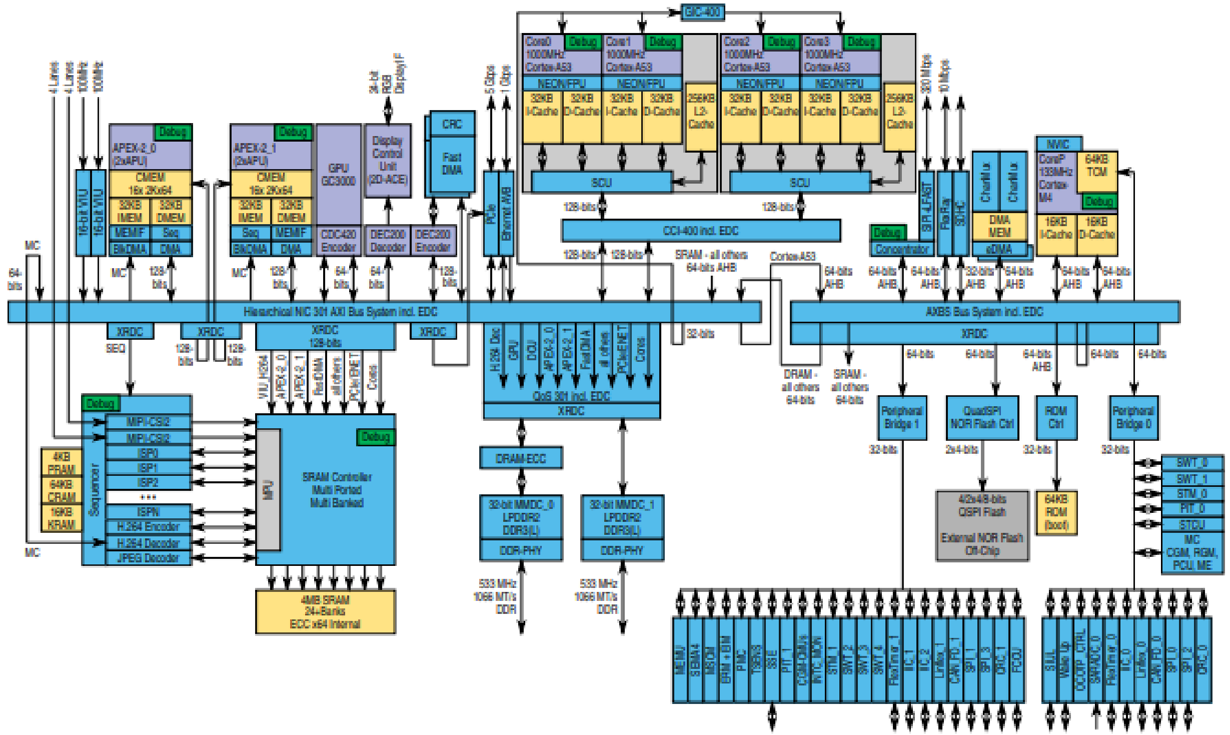
ARM Integration and Contention

ARM Value

- Highly integrated
- High performance
- Low power

Deos SafeMC for ARMv8-A Architecture processors from multiple manufactures.

- NXP
  - i.MX 8
  - S32V234
  - Layerscape
- Xilinx Zynq Ultrascale+



What happens to execution of concurrently running tasks on Cores 2-4, if tasks on Core 1 are not “well behaved” ?

Can Execution on Cores 2-4 have bounded WCETs for Safety Critical Tasks?

What would make MC Cert easier?

- If you found a genie in a lamp.
  - Private cache per core
  - Memory controller per core
  - Private memory per core

Is that good enough ?
Is that really why you went to multicore?

Multicore Guidance CAST-32A

- Software Planning
  - How many processors, what OS architectures and how they manage the cores.
- Planning and configuration of MCP
  - Document MCP settings to satisfy requirements
  - Document MCP settings contingency plans
  - Document resource partitioning and how you plan to mitigate contention issues.
- Interference Channels and Resource Usage
  - Identified the interference channels and chosen means of mitigation of the interference.
- Software Verification
  - Verify all the hosted software components function correctly and have sufficient time to complete their execution in the final configuration.
  - Verify that the data and control couplings between all the individual software components hosted on the same core or on different cores.
- Error Detection and Handling, and Safety Nets
- Reporting of Compliance with the Objectives of this Document

https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/cast_papers/media/cast-32A.pdf

Specifically not in CAST-32A

- Dynamically re-allocated of threads to different cores by the operating system.
- Hyperthreading. The idea of using the hyperthreading technology opens the door to contention issues inside the processor that you have no way of knowing about let alone controlling.

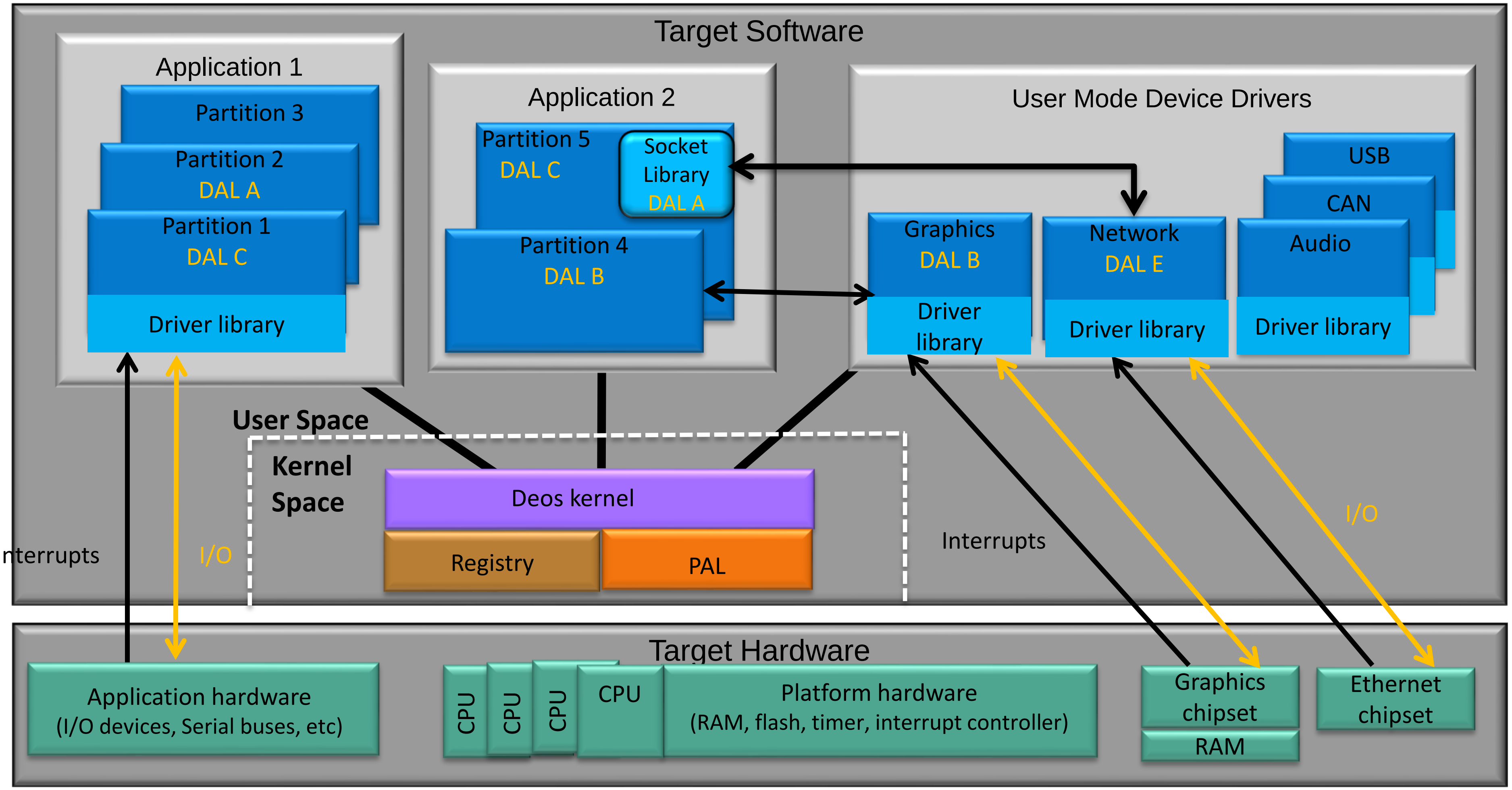
Multicore Processor Objectives

OBJECTIVES	DALs	DESCRIPTION	DDC-I COMMENT
MCP_Planning_1:	A, B, & C		
MCP_Resource_Usage_1:	A, B, & C		
MCP_Resource_Usage_2:	A & B		
MCP_Planning_2	A, B, & C		
MCP_Resource_Usage_3:	A & B		
MCP_Resource_Usage_4:	A & B		
MCP_Software_1:	A, B, & C		
MCP_Software_2:	A, B, & C		
MCP_Error_Handling_1:	A & B		
MCP_Accomplishment_Summary_1:	A, B, & C		

MCP Objectives Sample for Deos

OBJECTIVES	DALs	DESCRIPTION	DDC-I COMMENT
MCP_Planning_1:	A, B, & C	The applicant's software plans or other deliverable documents:	
		1) Identify the specific MCP processor, including the unique identifier from the manufacturer,	Deos supports many different multicore processors (MCP), this is product specific to be addressed by target system developer.
		2) Identify the number of active cores,	Deos supports the ability to select the number of which cores of an MCP to use, this determination is product specific to be addressed by target system developer.
		3) ………………… 4) …………………	
		5) Identify whether or not the MCP device will be used in an IMA platform to host software applications from more than one system,	Deos provides support for the development of IMA systems with multiple levels of safety. The desire to take advantage of these features is product specific to be addressed by target system developer
		6) Identify whether or not the MCP Platform will provide Robust Resource and / or Time Partitioning as defined in this document,	The MCP Platform will provide Robust Resource and Time Partitioning as defined in CAST-32A. The Deos product line provides Robust Resource Partitioning and Robust Time Partitioning by giving the target system developer interference channel solutions that range from elimination of the interference channel to a definitive bound on the interference channel utilizing features like Safe Scheduling, Cache Partitioning, and bounding memory transactions.
MCP_Resource_Usage_1:	A, B, & C	The applicant has determined and documented the MCP configuration settings that will enable the hardware and the software hosted on the MCP to satisfy the functional, performance and timing requirements of the system.	DDC-I provides a detailed users guides for the functionality and configuration of Deos. The target system developer is responsible for using these Users Guides to ensure correct configuration as well as CBIT check of configuration, if applicable

Deos High-Level Architecture



… loosely-coupled, modular application software partitions.

Safety Critical Multi-Core

Safety Critical Multicore Concerns:

1. Bound & control interference patterns
   - A. Minimize contention for shared resources (e.g., cache & memory)
   - B. Coordinate behaviors amongst cores
2. Getting good value from adding secondary cores

   Example concern: WCE will increase due to multicore interference patterns

Deos Multicore Solutions: SafeMC

1. Reduce interference patterns and reduce WCETs
   - A. Memory pooling & cache partitioning
   - B. Safe-scheduling
2. Performance enhancing features
   - A. Slack scheduling, including Window Activation for multicore

     Recovers and applies additional slack resulting from higher WCETs
   - B. Enable deterministic interrupting devices

Cache In Deterministic Systems

- The greatest performance factor for modern processors
- Growing in size and number of levels (e.g., L1, L2, and L3)
- Left uncontrolled, cache will cause performance variability (e.g., cache thrashing which increases the gap between best and worst case execution time (WCET))

Studies show that cache variability must be resolved in deterministic multicore systems

Cache Performance Variability

Cache variability is a significant issue for deterministic systems, that must be solved. Fixes include:

1. Cache flushing (e.g., flush cache between applications)
   - Good: Reduces performance variability
   - Bad: Forces cache flush overheads at an application context switch
2. Disabling of cache
   - Good: Eliminates cache performance variability
   - Bad: Huge performance penalty – forces the processor to a low level of performance. Also impractical for multicore processors.
3. Cache Partitioning – Several option with various results
   - Deos cache partitioning (patented)
   - Cache locking (not available on ARM)

Cache Partitioning via Cache Locking

Some processors allow selective cache locking:

- Lock lines, ways, or entire cache
- Core based partitions

Issues:

- Processor specific & not portable
- Requires application specific code in kernel and/or driver
  - Application manages cache
- Requires kernel/application linkage – Slow and high DAL
- Latest Arm processors don’t support.

Cache Partitioning with Deos

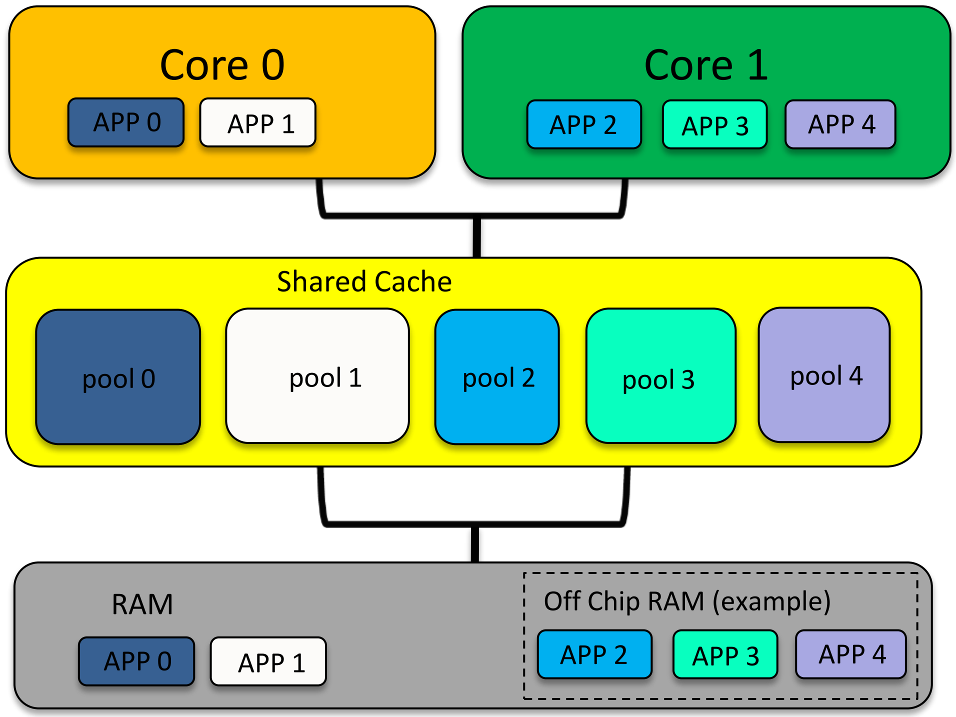
- Partitions Cache per Application
  - Best performance (reduces WCE) by eliminating shared cache thrashing across applications
  - Applications don't have to manage cache
    - No H/W cache locking instructions used.
- Portable
  - Processor Agnostic (does not require H/W "hooks")
  - Memory pools and partitions defined in XML configuration file, NOT in source code
  - No re-verification required platform to platform
- Applicable to both Mono-core & Multicore (x86, PPC and ARM)
- Patented Mechanism

Memory Pools & Cache Partitioning

Partitions memory segmented into "pools"

Cache segmented into "partitions" per application

RAM addresses Segmented per application



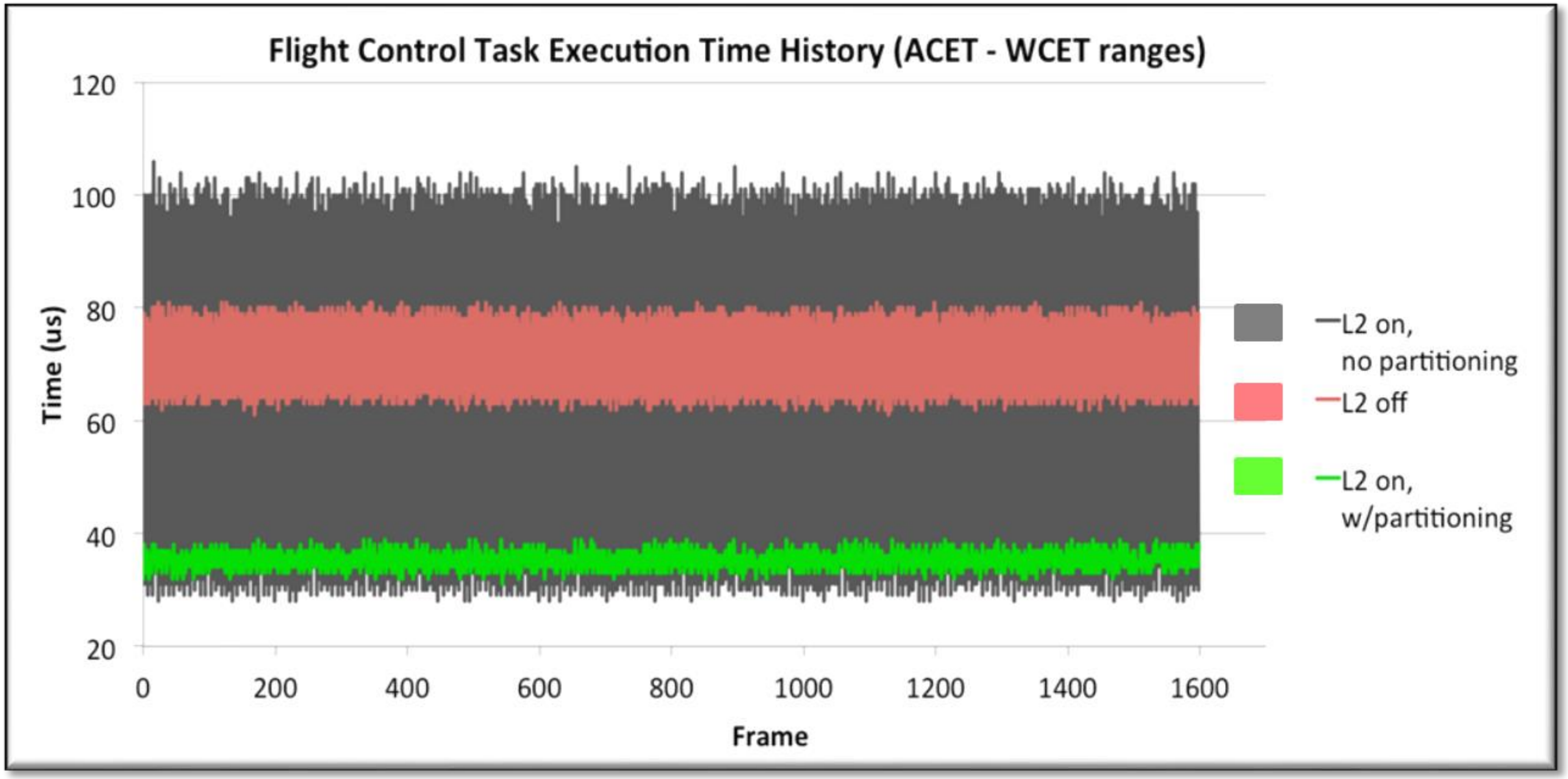
Memory Pooling - Enables physical memory segmentation (a key advantage for microcontrollers)

- Reduces cache thrashing
- XML Configuration (portable)
- Cache contention is bounded

- Partition per application per core.
- No application specific code
- No cache locking instructions used.

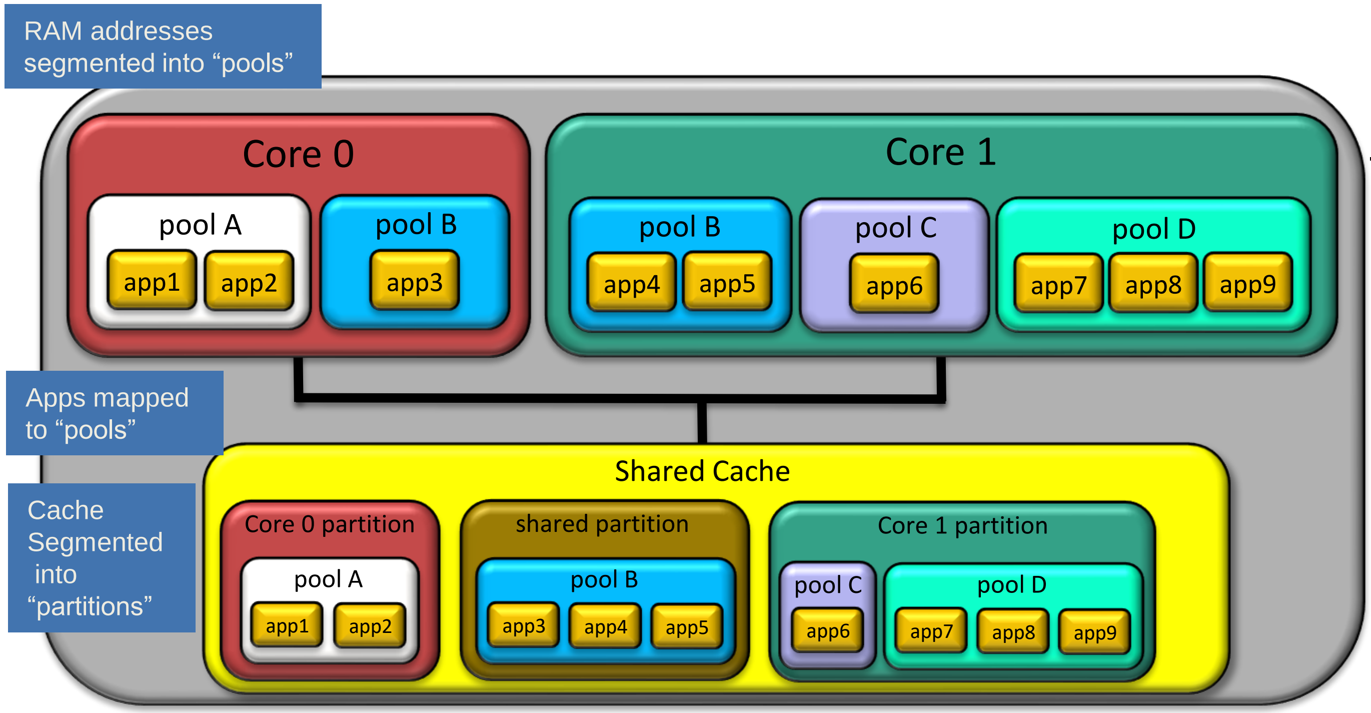
… optimizes application ACET/WCET behaviors & bounds WCET behavior.

Cache Partitioning – Bounding WCETs



- Bounds & controls cache interference patterns
- Can dramatically improve WCET performance

Memory Pools & Cache Partitioning



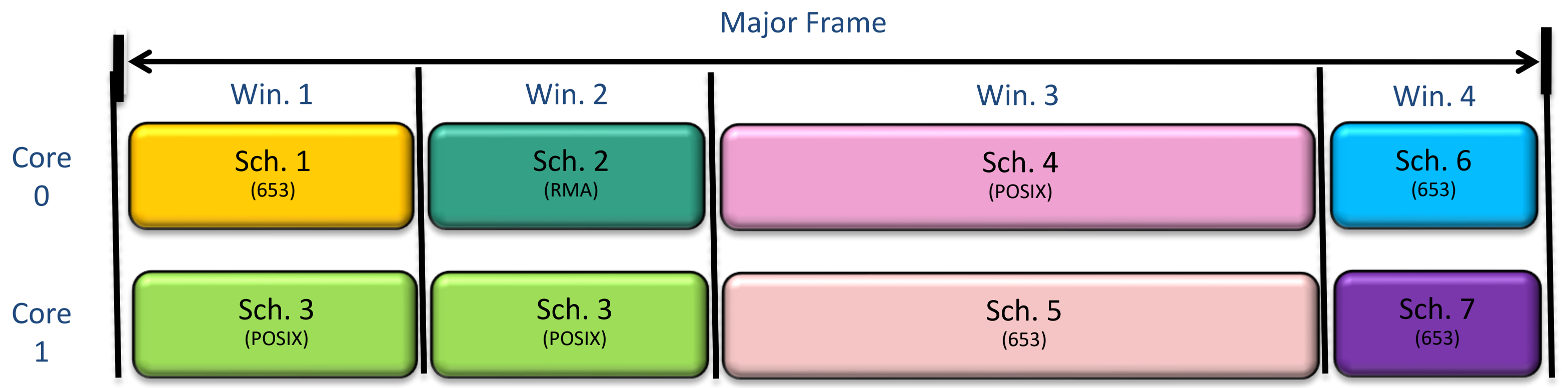
– With Deos' cache partitioning:
  - Min to no cross-core contention
  - Min to no cross-pool contention
  - Partitions can be shared across cores
  - Potential interference patterns are known
  - Cache contention is bounded & minimized

Patent Protected Methodology

US8069308B2, https://www.google.com/patents/US8069308?dq=US8069308&hl=en&sa=X&ved=0ahUKEwi7qqmC0OjTAhWJSiYKHVBBDFQQ6AEIJzAA
US20150205724A1, https://www.google.com/patents/US20150205724?dq=US20150205724+A1&hl=en&sa=X&ved=0ahUKEwj7pvbgz-jTAhVFQiYKHZP-BhcQ6AEIKTAA
US20090204764A1, https://www.google.com/patents/US20090204764?dq=US+20090204764+A1&hl=en&sa=X&ved=0ahUKEwjQvOT1z-jTAhUE4CYKHSJRCV0Q6AEIJzAA
EP2090987B1, https://www.google.com/patents/EP2090987B1?cl=en&dq=EP2090987B1&hl=en&sa=X&ved=0ahUKEwiT5aSN0OjTAhVBOyYKHZZEA68Q6AEIJzAA
EP3109765A1, https://www.google.com/patents/EP3109765A1?cl=en&dq=EP3109765A1&hl=en&sa=X&ved=0ahUKEwiN5cu10OjTAhVI0iYKHYQVBAMQ6AEIJzAA
EP2090987A1 https://www.google.com/patents/EP2090987B1?cl=en&dq=EP2090987A1&hl=en&sa=X&ved=0ahUKEwj_65nC0OjTAhUG7CYKHQl4DgEQ6AEIJzAA

… optimizes application ACET/WCET behaviors & bounds WCET behavior.

Deos Safe Scheduling for Multicore



- Bounds, controls & minimizes cross-core contention
  - Major frame partitioned into “windows”
  - Window boundaries align across cores

Bounded Multiprocessing

- Multiple scheduler/API types available
- Fine Grain locking for resource protection
- Allows for a mix of safety apps, or safety & non-safety apps

… optimizes application ACET/WCET behaviors and bounds WCET behavior.

Fine Grain Locking

- In the kernel all locking is done in a single core space only, therefore, no cross core blocking is possible.
  - No cross core locks (No resources used for all cores)
  - No single lock for scheduling (each core has a scheduler)
  - No single lock for all kernel interface objects (each object created has its own lock)
- Cross core blocking is only possible if a developer designs it to happen
  - Threads on different cores share a kernel interface object (semaphore, event, mailbox, etc.)
  - Thread creates another thread and schedules it on a different core
  - Threads of different cores share a memory pool
  - In these cases affects limited to the cores in question and not the others.

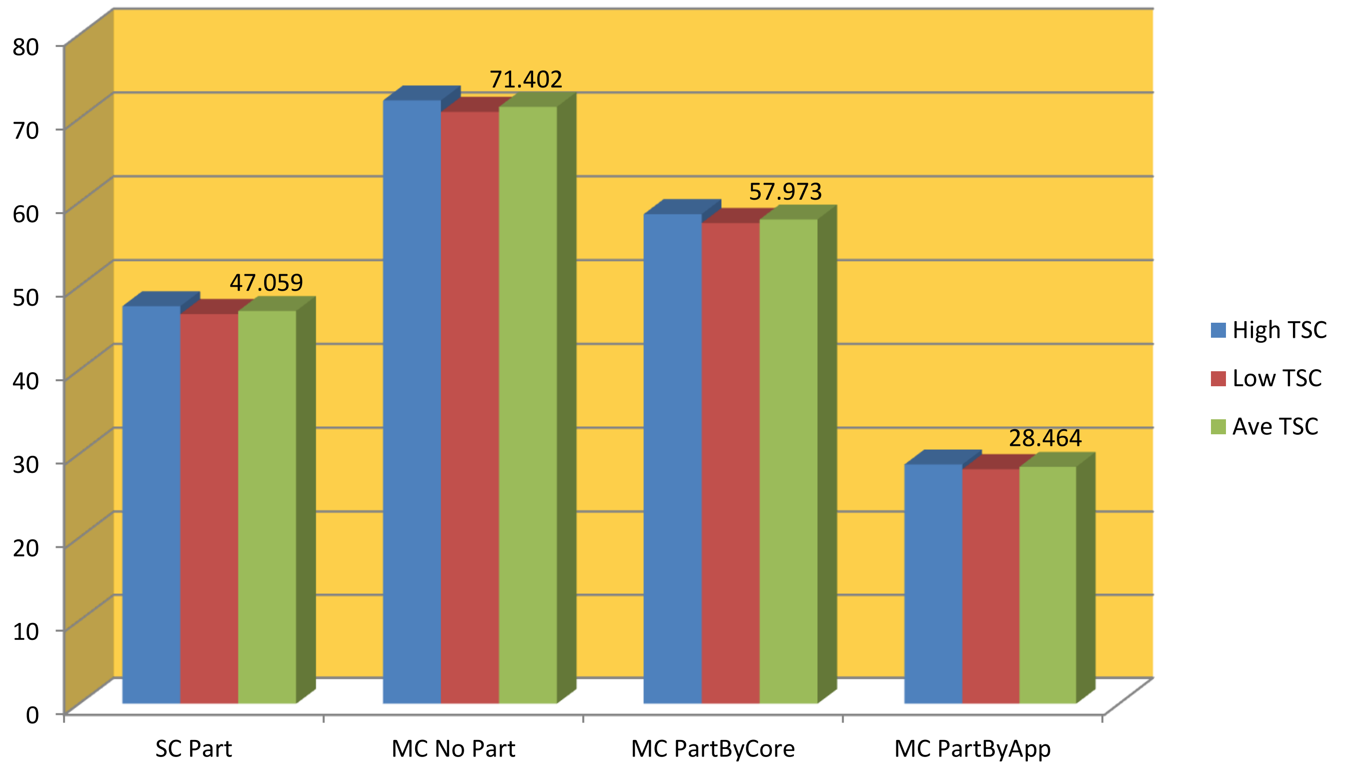
Memory Throttling

- Built-in Performance Counters
  - Certain processors have hardware capability to count processor level events.
  - Setup CPU to send interrupt when a particular performance counter threshold is reached.
  - For example, last level cache miss is a selectable event to be counted.
  - When threshold is hit for a particular partition, a decision can be made to on how to deal with the offending partition.

Multicore Test Setup

- This example consists of two Deos processes
- Worker process is made up of two threads
  - Writer - job is to fill a 600 KByte RAM buffer
  - Checksummer - computes a checksum over this buffer
  - Execution order is coordinated such that Checksummer's execution always closely follows Writer's.
  - Repeat 100 times and measure max, min, average
- Trasher process has a single thread, whose role is to disrupt cache and generate interfering memory bus activity.
- Goal to show how cooperation is the application design value for Multicore.

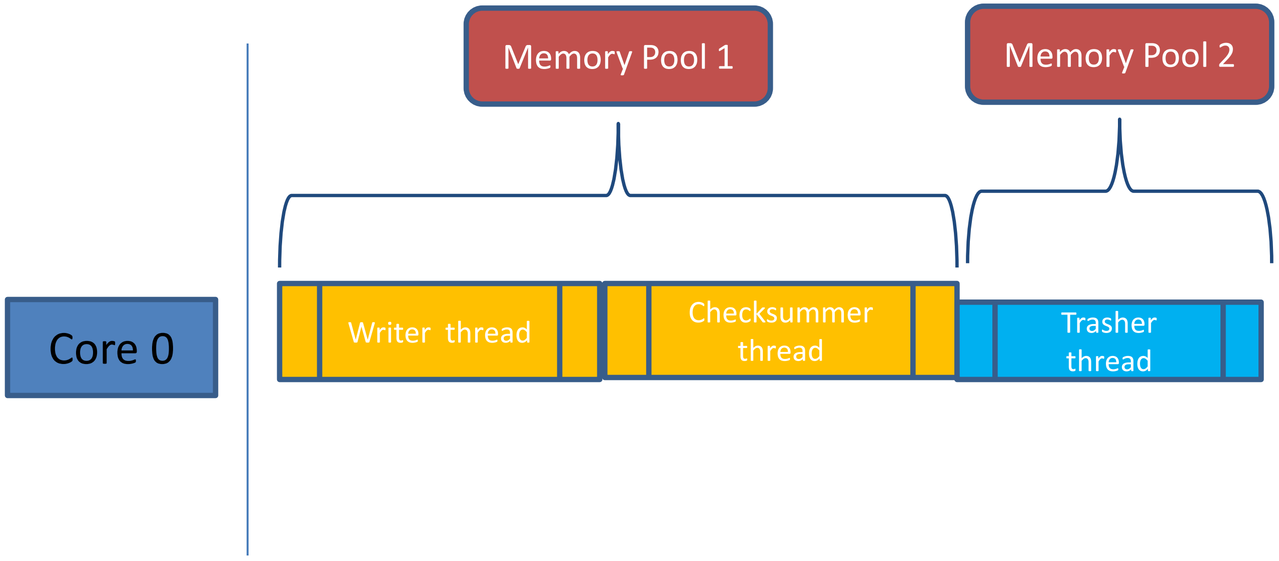
Cache Partitioning Results



TSC – Time Stamp Counter (msec)

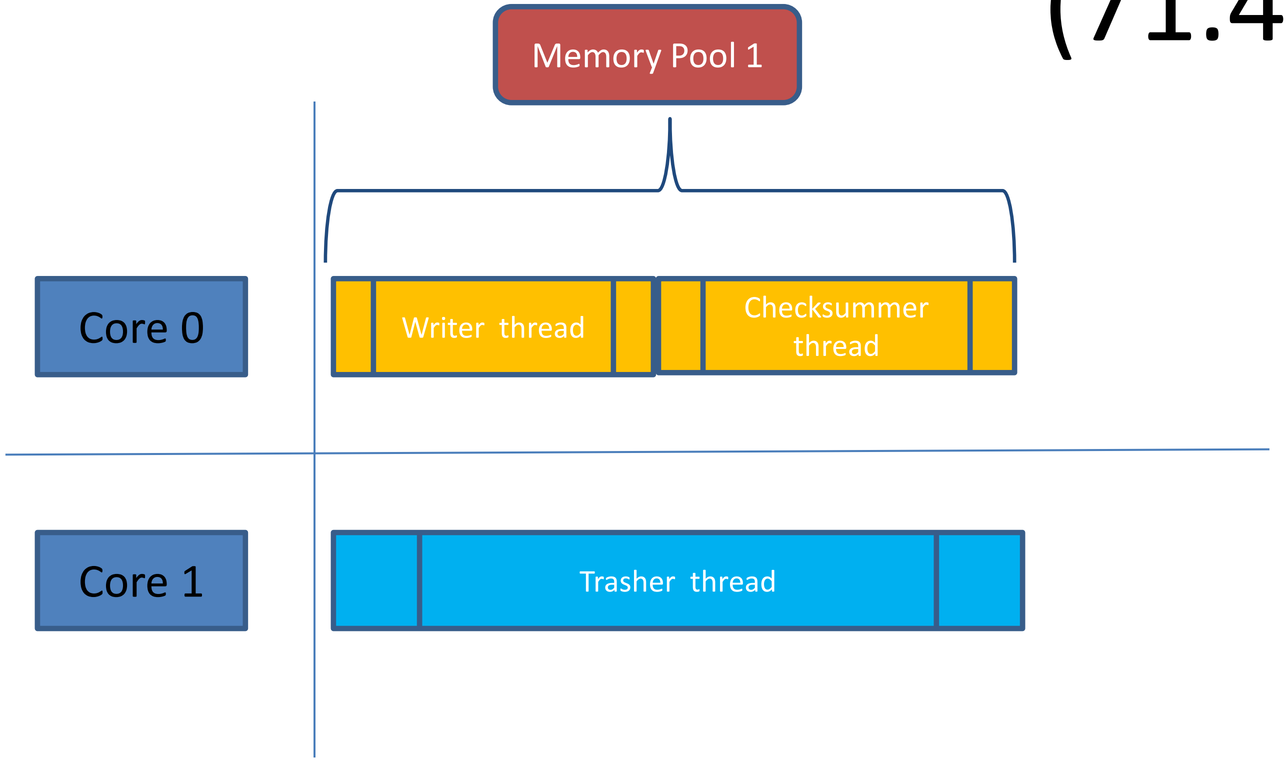
- Single Core with cache partitioning
- Multi-core with no cache partitioning
- Multi-core with cache partitioning by Core
- Multi-core with cache partitioning by Application

SC Cache Partitioning (47msec)



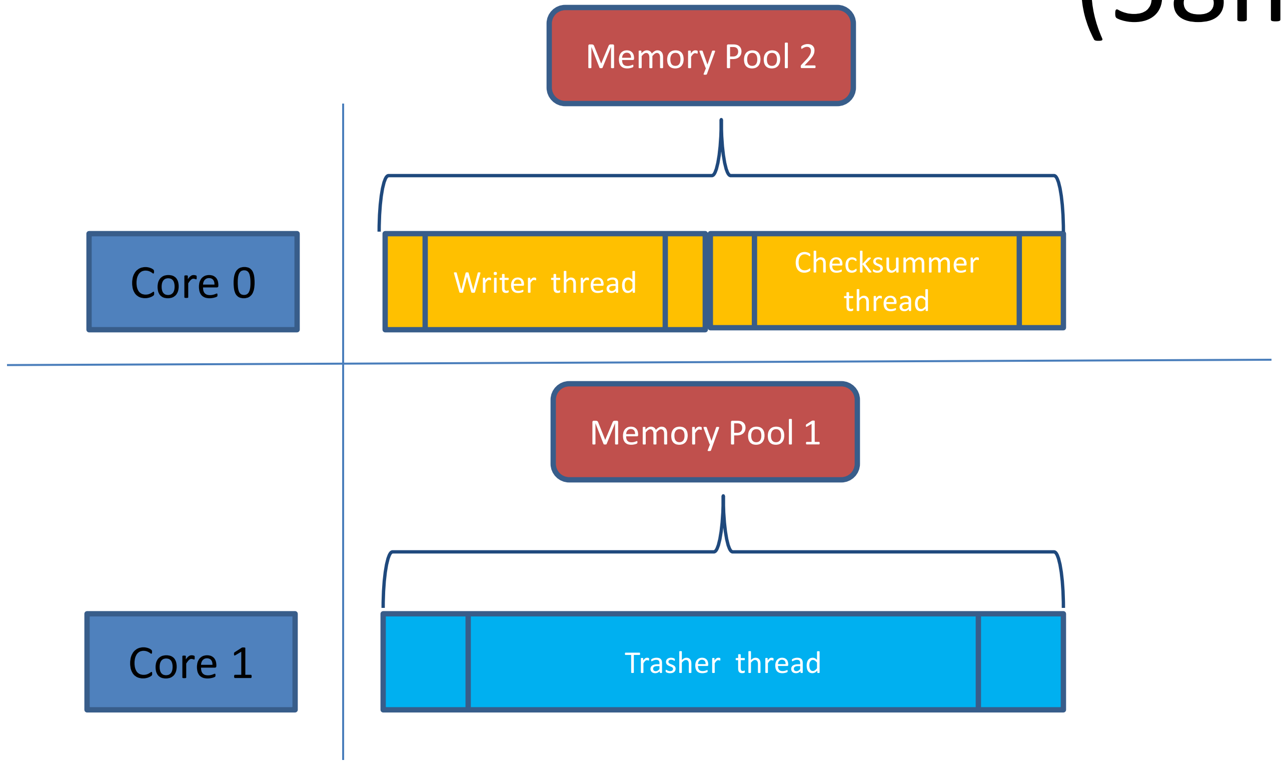
- Writer and Checksummer threads on same core so they are waiting on each other
- Trasher is on same core so trasher can not run or interfere with writer.
- Trasher in different memory pool so doesn’t affect cache.

Multicore No Cache Partitioning (71.4ms)



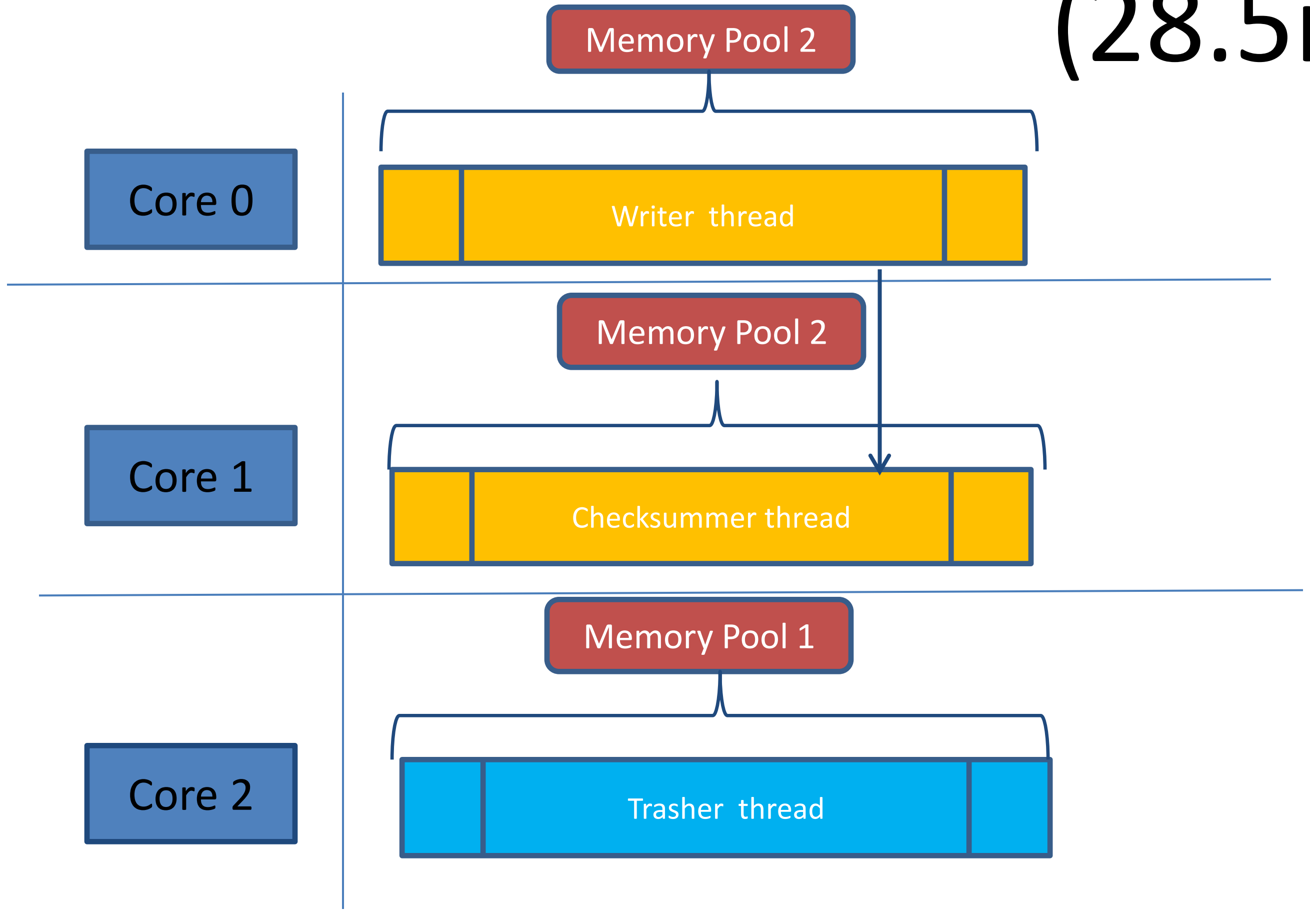
- Writer and Checksummer threads on same core at the same priority so checksummer waits till writer is complete.
- Trasher is on different core so trasher creating memory bus contention (MBC)
- Trasher in same memory pool causes it to be a slower.(cache is always dirty)

Multicore Cache Partitioning by Core (58msec)



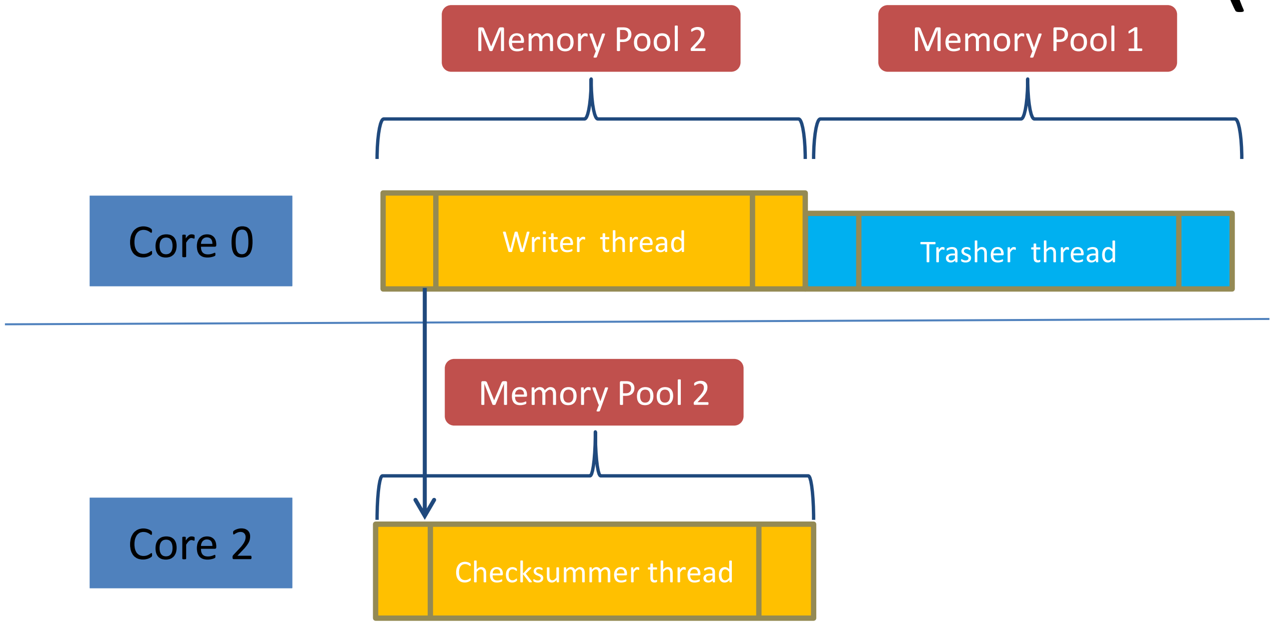
- Writer and Checksummer threads on same core so they have to wait on each other
- Trasher is on different core running more often so trasher creating MBC but not trashing test memory.

Multicore Cache Partitioning by App (28.5msec)



- Writer and Checksummer threads on different core so they can coordinate. (Never waiting on each other)
- Trasher is on different core so trasher creating MBC

Multicore Cache Partitioning by App (23msec)



- Writer and Checksummer threads on different core so they can coordinate. (Never waiting on each other.)
- Trasher is on same core as writer so trasher can not run or interfere with writer.

Lessons Learned

- Multicore focus is typically on contention when it should also be on cooperation. You are running multiple cores to get more work done.
- Results show
  - Best Case Timing is when threads are on multiple cores coordinating and contention is minimized.
  - Worst Case Timing is when there is poor or no cache control and poor or no application coordination.

Thank you!

Contact Information

Gary Gilliland

ggilliland@ddci.com

www.ddci.com