

# Performance Challenges of Modern Hardware Architectures for Real-Time Systems

Software Engineering Institute  
Carnegie Mellon University  
Pittsburgh, PA 15213

Jorgen Hansson, John Hudak and Peter  
Feiler, Dio de Niz



# Advanced Processors in Embedded Systems

---

## Goal

- Efficient use of advanced processor hardware architectures without sacrificing predictable execution times in embedded systems

## Objective

- Reduce worst-case execution time
  - Improve schedulability
  - Improve end-to-end latency through deadline reduction
- Avoid execution time variation to due cache/pipeline
  - No increase in actual execution time & latency variation
  - Maintain deterministic communication



# State of the Art – The Industry Perspective

---

Industry is increasingly deploying standardized hardware architectures featuring caching and pipelining.

Industry hesitates to enable caches/pipelines due to predictability impact.

Effects of caching and pipelining and their applicability to real-time and embedded systems have received little attention.

- Caching and pipelining strategies are primarily designed to improve throughput
- Caching and pipelining have primarily been studied in isolation
- Scratchpad memory provides predictability



# Advanced Processor Performance Challenges

---

## High-performance processors

- Multi-level caches
- Multi-level pipelines
- Multi-core processors
- MMU and cache sharing

## Embedded systems performance

- Predictability of worst-case execution time
- Interference through preemption
- Periodicity leads to dirty caches
- Disabled cache, pipeline leads to performance loss



# State of the Art – The Academic Perspective

---

Academia is developing predictive models and intricate caching schemes

- Associative cache
- N-way caching
- Prioritized caching
- Cache swapping
- Cache partitioning
- Cache locking
- Scratchpad
- Cache block sizes

Bottom up cache performance analysis



# Project Approach

---

## Learn from academic research results

- Identify key drivers for cache/pipeline performance
- Identify effective caching/pipelining schemes to manage predictability of WCET

## Utilize application architecture knowledge

- Embedded systems architectures with periodic and stochastic workloads
- Impact of specialized application architectures (ARINC653)
- Relevant application workload profiles

## Feasibility of an analytic framework

- Focus on improving worst-case execution time
- Aim for near-optimal efficiency with predictable WCET



# Validation of Analytic Framework

---

Analyze tradeoffs in performance with respect to cache characteristics (levels, locking, reload policies, etc), pipeline depth, and workload characteristics.

- Validate cache timing through experiments using a cache simulator.
  - Near-optimal cache partitioning with a single task, multiple tasks, task interaction
- Investigate and quantify the effects of pipeline depth coupled with various cache organizations by integration of a pipeline simulator.
  - Task characterization (I/O, data-access, CPU intensive) vs pipeline depth
- Create guidelines on the use of the timing models in model-based engineering.



# Project Team & Collaborators

---

## SEI

- Jorgen Hansson
- John Hudak
- Peter H. Feiler
- Dio de Niz

## Ford

- Bill Milam

## Nokia R&D

- Gopal Raghavan, John Shen (Previously Intel)

## CMU

- Raj Rajkumar (potential)



# Outline

---



Research survey and observations

**Initial focus on caches**

Value-based cache allocation

Cache strategy tradeoff

Validation by simulation

Hybrid cache use & other performance studies



# Academic Perspective Observations

---

Fancy caching schemes are hard to compare: Little conclusive results for general workloads

Realistic benchmark workloads: Many are chosen to show off specific caching schemes

Preemptive scheduling creates hell for cache miss prediction: Staschulat, Ernst (2005)

Prioritized caching performs better for periodic workloads: Tan, Mooney (2005)

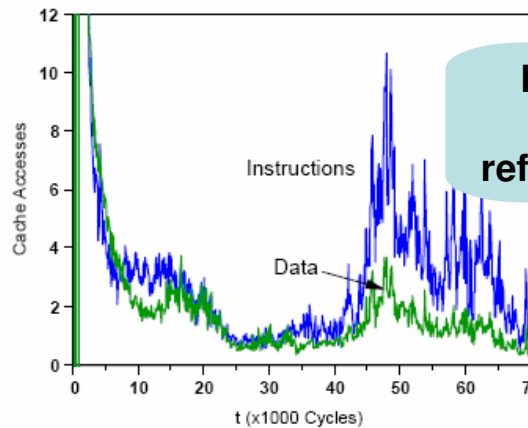
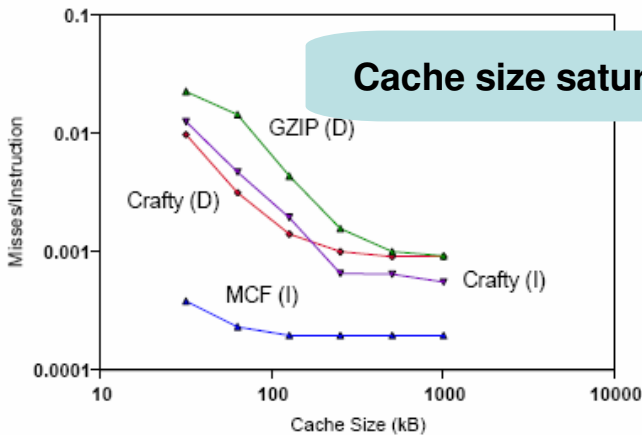
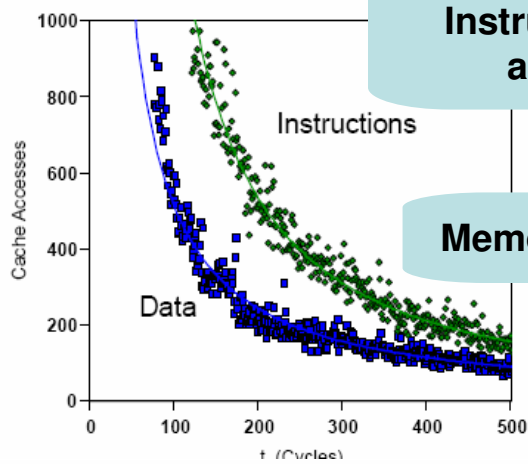
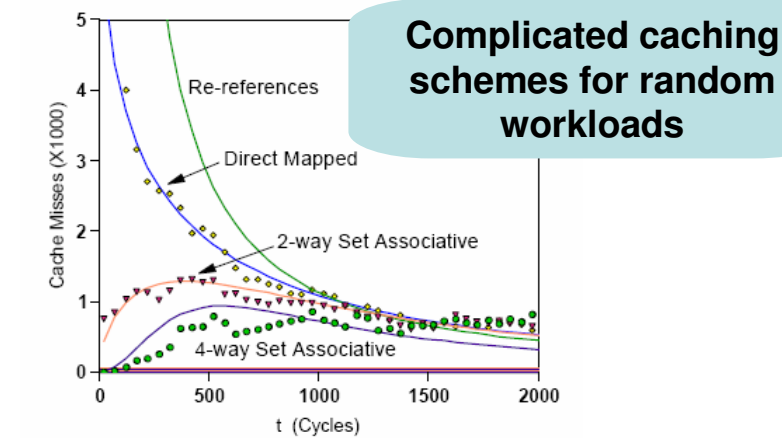
Benefits of software based cache partitioning: Mueller, et al – 1995

Scratch pad memory: predictable execution & minimized power consumption (2004-2006)



# Cache Misses Affect Execution Time Jitter

Cache Miss Behavior Is It 2? : A. Hartstein, IBM Watson

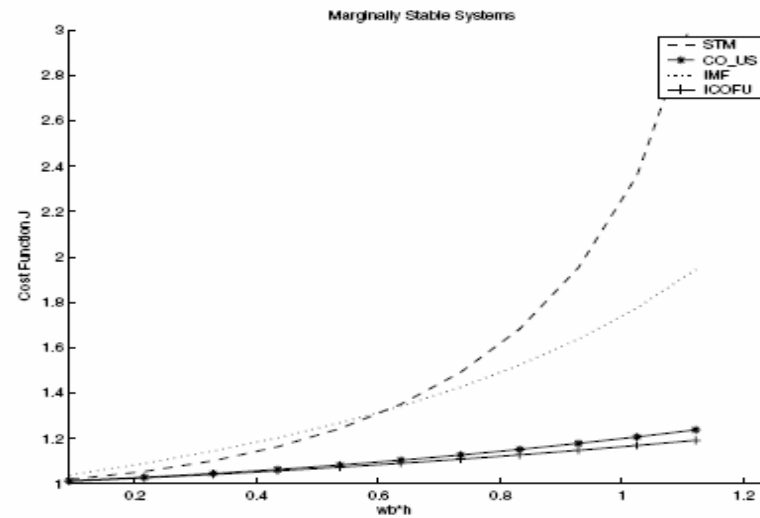
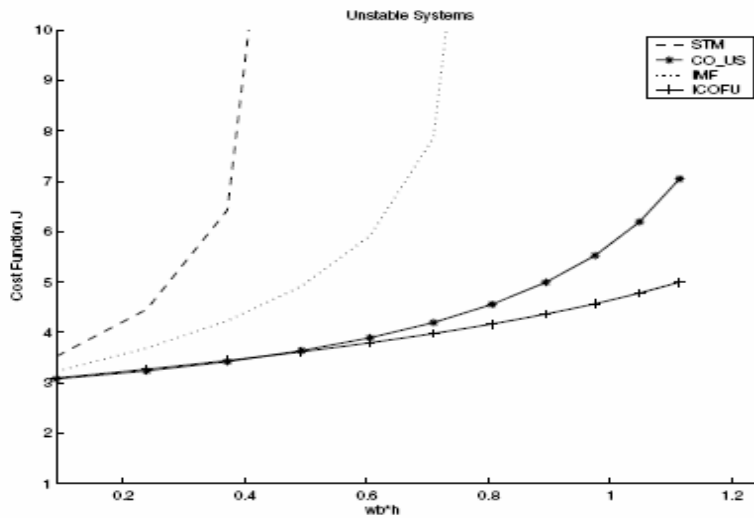
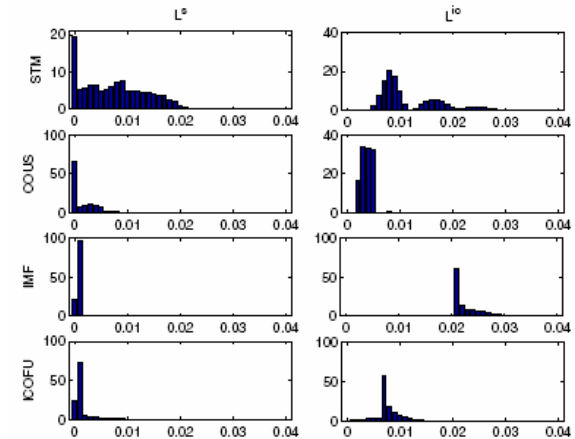


# Impact of Sampling Latency Jitter

## Impact of Scheduler Choice on Controller Stability

- A. Cervin, Lund U., CCACSD 2006

Root cause: sampling jitter due execution time jitter and non-deterministic communication



# Outline

---

- Research survey and observations
- Value-based cache allocation
- Cache strategy tradeoff
- Validation by simulation
- Hybrid cache use & other performance studies

Initial focus on caches



# Predictable Cache Use Strategy

---

## Static cache allocation

- Cache as fast memory (Scratch Pad Memory)
- Allocation based on memory access density
- Memory access patterns of periodic tasks & partitions

## Hybrid static/dynamic cache allocation

- Balance static and dynamic cache allocation
- Accommodate stochastic tasks without affecting critical tasks



# Value-based Static Cache Allocation

---

## Q-RAM applied to caches/scratch pads

- Original Q-RAM research by Rajkumar
- Utilized in graceful degradation (CMU/SEI collaboration)

## Maximize execution time reduction

## Memory access density as key performance driver

- Reduce access time most frequently accessed instructions & data

## Analysis and measurement of memory access density

- Benchmark task executions
- Utilize periodicity & instruction, local/global data statistics



# Q-RAM for Caches

---

Single resource single QoS dimension problem (SRSQP)

Cache as limited resource

Memory access density (MAD) as value measure

- Code loops, branching averages, periods of tasks are reflected
- Measurable by hardware & computable by analytical model

MAD-based allocation

- Allocate to fragments in decreasing density
- Use average access patterns for stochastic tasks



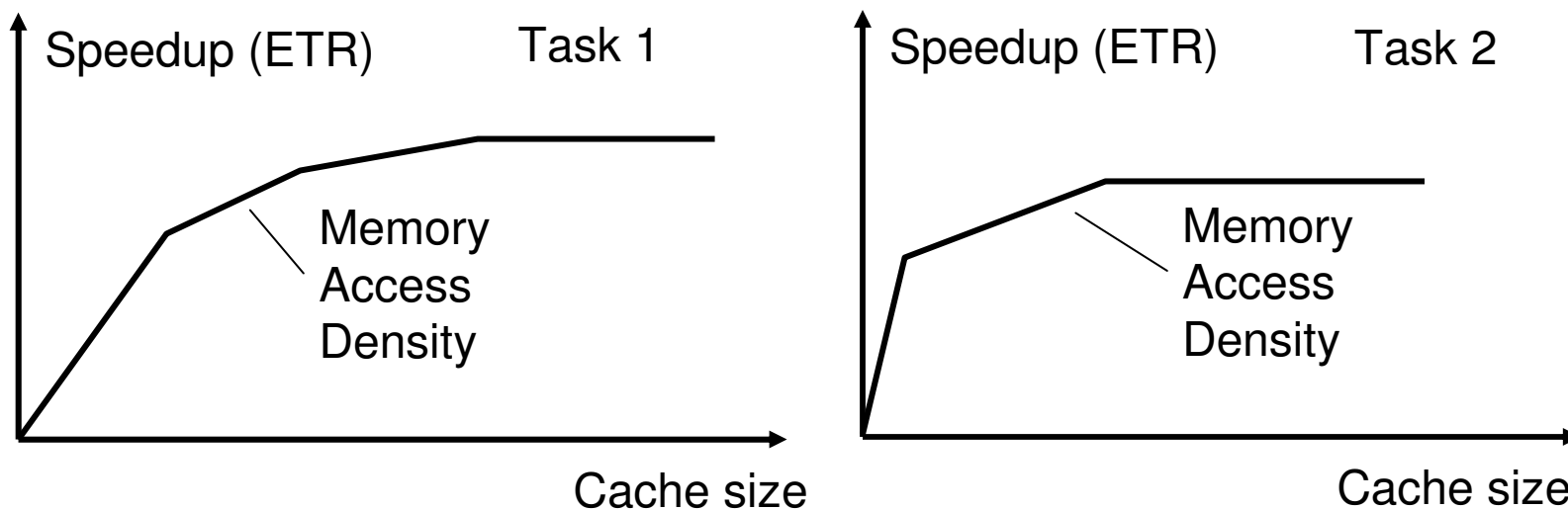
# Execution Time Reduction

ETR: execution time reduction

$$\text{Max ETR} = \sum FS_k * MAD_k * (\text{MAT} - \text{CAT})$$

Allocation in decreasing MAD order

Allocation across task access profiles



# Important Memory Access Measures

---

MAT: memory access time

CAT: cache access time

CaC: cache capacity

MAD: memory access density (accesses per sec)

AAP: application access profile

- Sum of fragments with different MAD
- FS: fragment size
- FAF: fragment access frequency

TAP: task access profile

- FAF per task dispatch
- TDF: task dispatch frequency (dispatches per sec:  $1/\text{period}$ )
- $\text{MAD} = \text{FAF} * \text{TDF}$



# Process/Partition Allocation

---

Cache per processor

Binding of process/partition to processor

PAP: process/partition access profile

PDF: process/partition dispatch frequency

$PAP = \sum TAP = \sum FS$  with same MAD

Common code/data:  $\sum$  MAD for same memory locations

Measured process/partition MAD

- Reflects process/partition level code/data sharing



# Cache & Processor Allocation - 1

---

## An iterative solution approach

- Cache allocation sizes from cache Q-RAM
- Balanced task set allocation based on Binpacker
- If not schedulable, reduce cache allocation based on cache Q-RAM and repeat
  - Bounded iteration
  - Not necessary if schedulable without cache

We have a Q-RAM implementation from previous projects

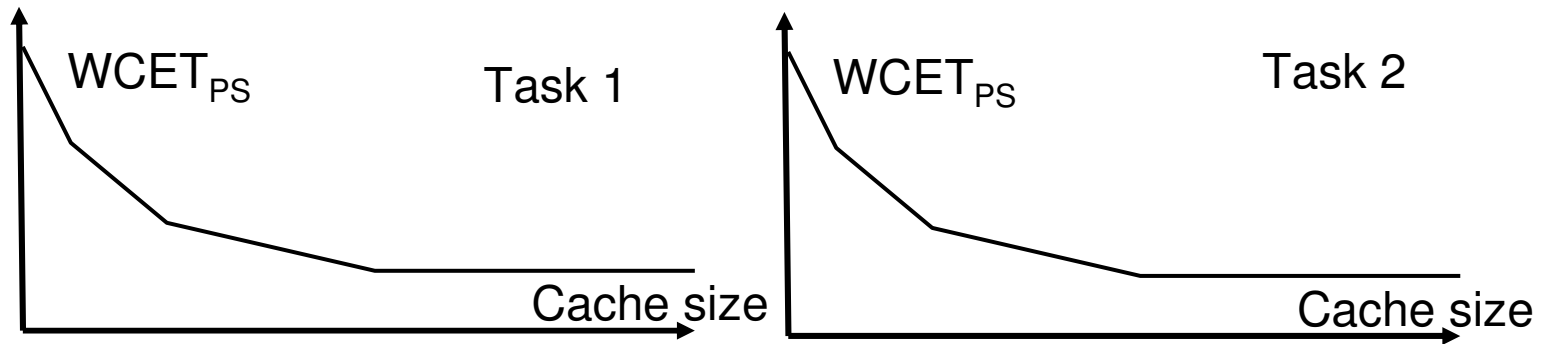
We have Binpacker as part of AADL OSATE toolset



# Cache & Processor Allocation - 2

Variant of Multi Resource Single QoS dimension problem (MRSQP)

- Processor & cache as two resources
- Constant utility function
- Codependent resource requirements
- $WCET_{PS}$  : WCET normalized to per second of real time



To adapt MRSQP to cache problem  
(cooperation with Raj Rajkumar)



# Outline

---

Research survey and observations

Initial focus on caches

Value-based cache allocation



Cache strategy tradeoff

Validation by simulation

Hybrid cache use & other performance studies



# Cache Overlay vs. Partitioning

---

Cache overlay (cache swap) used in partitioned systems

- Avoids cache dirtying side effect by other partitions

Cache overlay

- Cache load/unload cost per partition
- COC: Cache overlay cost per partition execution
- $COC = 2 * CAC * MAT * BlockXferReductionFactor$

Cache partitioning

- Partition size based on cache Q-RAM TAP or PAP sets
- Optimal system level cache partitioning



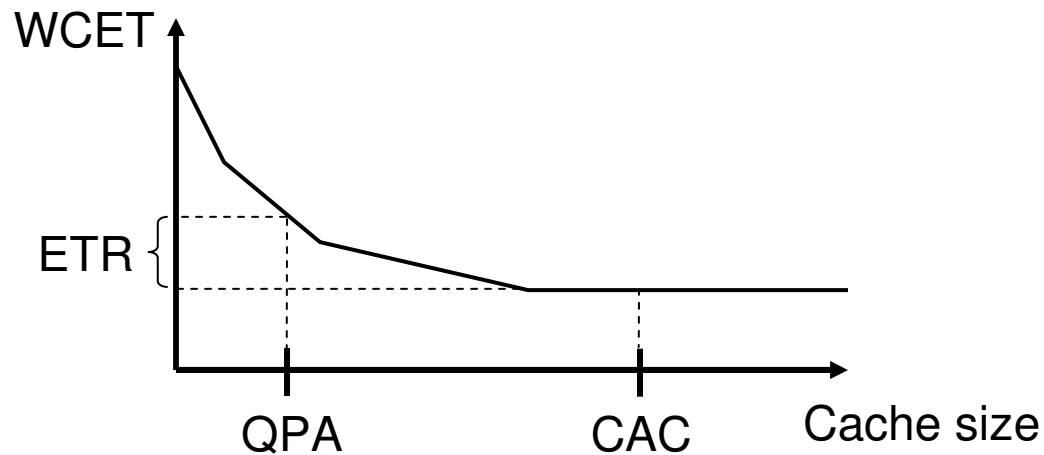
# Cache Overlay Gain

$PAP_{CAC}$  : Cache Q-RAM for partition CAC cache size

$PAP_{QPA}$  : Partition cache size based on system cache Q-RAM

$$ETR (PAP_{QPA} \rightarrow PAP_{CAC}) = \sum FS_k * MAD_k * (MAT - CAT)$$

- $k = index_{QPA} + 1 .. index_{CAC}$



# Cache Overlay Tradeoff Point

---

ETR vs. COC \* PDF

- Per second as time window
- Insensitive to partition and task execution rates

Comparison per partition

- Selective cache swap per partition

Comparison per processor

- $\sum$  partitions per processor
- Configurable processor cache swap

Comparison for whole system

- $\sum$  partitions
- Single processor cache swap configuration



# Outline

---

Research survey and observations

Initial focus on caches

Value-based cache allocation

Cache strategy tradeoff



Validation by simulation

Hybrid cache use & other performance studies



# Validation of Analytical Model

---

## Measurement of cache related measures

- Task working set sizes
- Memory access density profiles
- Cache size vs. execution time measurements

**Data from processor spec sheets for calculated values**

## Model validation

- Memory access density for typical workloads
- Cache behavior simulation based on typical embedded application profiles
- Performance comparison to existing cache strategies

**Collected initial samples  
Potential samples from Ford**

**Initial cache simulator runs with static allocation  
Investigating use of other cache simulators to validate fidelity of results**



# Outline

---

Research survey and observations

Initial focus on caches

Value-based cache allocation

Cache strategy tradeoff

Validation by simulation



Hybrid cache use & other performance studies



# Memory Access Categories

---

Instruction, local data, global data access ratio

Local data access patterns

- Register as fast memory
- Stack access patterns: top region of stack (LRU caching scheme)

Global data access patterns: does it pay to cache global data

- Cache gain as % of total time
- Random: low MAD
- Sequential: memory block
- Focused: write/read, higher MAD

**Previous work shows little total performance gain even on data intensive applications.**

**To be validated in context of this context.**



# Additional Cache Tradeoff Studies

---

Benefit of increased cache size

Multi-core with shared cache vs. cache per core

Hybrid cache vs. all static cache

Static cache based on averages vs. dynamic cache for stochastic tasks

Impact of processor speed, memory speed, cache speed



# Different Performance Improvement Objectives

---

Maximize overall spare capacity

Maximize per processor spare capacity

Reduce multi-path latency

Improve critical path performance

Improve system wide QoS





**Software Engineering Institute**

**Carnegie Mellon**



**Software Engineering Institute**

**Carnegie Mellon**

**Processor Performance**

April 2007

© 2007 Carnegie Mellon University