A Multicore Real-Time Mixed-Criticality Framework for Avionics

Jim Anderson

University of North Carolina at Chapel Hill

Work Supported by AFOSR and AFRL
Driving Problem

• **Advanced UAVs.**
  » Must automate pilot function using “AI-type” software.
  » Causes certification difficulties.
  » Workload is dynamic and computationally intensive.
  » Suggests the use of multicore.

**Our focus:** *What should the OS look like?*
  
  – Particular emphasis: *Real-time* constraints.

» Have worked some with **Northrop Grumman** on this.
Multicore in Avionics: The Good

- Enables computationally-intensive workloads to be supported.
- Enables SWAP reductions.
- Multicore is now the standard platform.
Multicore in Avionics: The Bad & Ugly

- Interactions across cores through shared hardware (caches, buses, etc.) are difficult to predict.

  **Approach 1:** Accept this fact. Use resulting slack for less-critical computing. “Multicore processors are big slack generators.”

- Approach 2: Manage shared caches more predictably so that more accurate execution time predictions can be made.
What is “Less Critical”?  

- We assume tasks are assigned to CRITICALITY levels, like in DO-178B:
  - **Level A:** Catastrophic.
    - Failure may cause a crash.
  - **Level B:** Hazardous.
    - Failure has a large negative impact on safety or performance…
  - **Level C:** Major.
    - Failure is significant, but…
  - **Level D:** Minor.
    - Failure is noticeable, but…
  - **Level E:** No Effect.
    - Failure has no impact.

![Diagram showing more conservative design vs. less conservative design]
Outline

• Background.
  » Real-time scheduling basics.
  » Mixed-criticality scheduling.

• MC$^2$: Proposed mixed-criticality architecture.

• $\text{MAN}^\text{RT}$: Proposed framework for managing shared caches.

• Future research plans.
Task Model Assumed in this Talk

- Set \( \{ T = (T.e, T.p) \} \) of periodic tasks.
  - \( T.e \) = T’s worst-case per-job execution cost.
  - \( T.p \) = T’s period & relative deadline.
  - \( T.u \) = T.e/T. p = T’s utilization.

\[ \begin{align*}
T &= (2, 5) \\
U &= (9, 15)
\end{align*} \]
Task Model Assumed in this Talk

- Set \( \{T = (T.e, T.p)\} \) of periodic tasks.
  - \( T.e \) = T’s worst-case per-job execution cost.
  - \( T.p \) = T’s period & relative deadline.
  - \( T.u \) = \( T.e / T.p \) = T’s utilization.

\[ T = (2,5) \]
\[ U = (9,15) \]
Task Model Assumed in this Talk

- Set \( \{T = (T.e, T.p)\} \) of periodic tasks.
  - \( T.e \) = T’s worst-case per-job execution cost.
  - \( T.p \) = T’s period & relative deadline.
  - \( T.u \) = \( T.e/T.p \) = T’s utilization.
Task Model Assumed in this Talk

- Set \( \{T = (T.e, T.p)\} \) of **periodic tasks**.
  - \( T.e = T \)'s worst-case per-job execution cost.
  - \( T.p = T \)'s period & relative deadline.
  - \( T.u = T.e/T.p = T \)'s utilization.

\[ T = (2,5) \]
\[ U = (9,15) \]

---

**One Core Here**

- Job release
- Job deadline
Multiprocessor Real-Time Scheduling

Two Approaches:

Partitioning

Steps:
1. Assign tasks to processors (bin packing).
2. Schedule tasks on each processor using uniprocessor algorithms.

Global Scheduling

Important Differences:
- One task queue.
- Tasks may migrate among the processors.
Hard vs. Soft Real-Time

- **HRT**: No deadline is missed.
- **SRT**: Deadline tardiness is bounded.

  » A wide variety of global scheduling algorithms are capable of ensuring bounded tardiness with no utilization loss.
Scheduling vs. Schedulability

What’s “Utilization Loss”? 

- W.r.t. scheduling, we actually care about two kinds of algorithms:
  - Scheduling algorithm (of course).
    - Example: Earliest-deadline-first (EDF): Jobs with earlier deadlines have higher priority.
  - Schedulability test.
    - Test for EDF
      - Test for \( \tau \) with EDF:
        - Utilization loss occurs when test requires utilizations to be restricted to get a “yes” answer.
        - deadlines: the higher priority.
        - no timing requirement will be violated if \( \tau \) is scheduled with EDF
        - a timing requirement will (or may) be violated …
Ensuring Bounded Tardiness

Under partitioning & most global algorithms, overall utilization must be capped to avoid deadline misses. Due to connections to bin-packing.

Exception: Global "Pfair" algorithms do not require caps. Such algorithms schedule jobs one quantum at a time. May therefore preempt and migrate jobs frequently. Perhaps less of a concern on a multicore platform.

Under most global algorithms, if utilization is not capped, deadline tardiness is bounded. Sufficient for soft real-time systems.

Example Global-EDF schedule…

On Processor 1

On Processor 2

T = (2,3)

U = (2,3)

V = (2,3)
Ensuring Bounded Tardiness

Example Global-EDF schedule…

**Tardiness** is at most one quatum.

\[
\begin{align*}
T &= (2,3) \\
U &= (2,3) \\
V &= (2,3)
\end{align*}
\]
Mixed-Criticality Scheduling
Proposed by Vestal [2007]

• Each task has an execution cost specified at each criticality level (A-E).
  » Costs at higher levels are (typically) larger.

• Example:
  \[ T.e_A = 20, \ T.e_B = 12, \ T.e_C = 5, \ldots \]

• Rationale: Will use more pessimistic analysis at high levels, more optimistic at low levels.
Mixed-Criticality Scheduling
Proposed by Vestal [2007]

Some “weirdness” here: Not just one system anymore, but five: the level-A system, level-B,…

- Costs at higher levels are (typically) larger.
- The task system is correct at level $X$ iff all level-$X$ tasks meet their timing requirements assuming all tasks have level-$X$ execution costs.
Outline

• Background.
  » Real-time scheduling basics.
  » Mixed-criticality scheduling.

• MC²: Proposed mixed-criticality architecture.

• $\text{MAN}^{\text{RT}}$: Proposed framework for managing shared caches.

• Future research plans.
MC²: Mixed-Criticality on Multicore
Our Proposed Mixed-Criticality Architecture (Joint with NGC)

- We assume five criticality levels, A-E, like in DO-178B.
- We statically prioritize higher levels over lower ones.
- We assume:
  - Levels A & B require HRT guarantees.
  - Levels C & D require SRT guarantees.
  - Level E is non-RT.
- Right now, we’re assuming a static system.
- We’re currently working on enabling dynamic changes at levels C-E.
Scheduling in MC\(^2\)

- **Level A (HRT):** Partitioned, cyclic executive (table-driven).
  - CEs are the de facto standard for highly critical workloads.
- **Level B (HRT):** Partitioned EDF (or RM).
  - PEDF (PRM) is a good HRT scheduler.
- **Levels C & D (SRT):** Global EDF.
  - GEDF is a good SRT scheduler.
- **Level E (Best Effort):** BE Scheduler.
MC²

Level A
- Core 1: CE
- Core 2: CE
- Core 3: CE
- Core 4: CE

Level B
- Core 1: EDF
- Core 2: EDF
- Core 3: EDF
- Core 4: EDF

Level C
- Core 1: G-EDF
- Core 2: G-EDF
- Core 3: G-EDF
- Core 4: G-EDF

Level D
- Core 1: G-EDF
- Core 2: G-EDF
- Core 3: G-EDF
- Core 4: G-EDF

Level E
- Core 1: Best Effort
- Core 2: Best Effort
- Core 3: Best Effort
- Core 4: Best Effort

(higher (static) priority)

(lower (static) priority)
Reclaiming Spare Capacity

• MC\(^2\) does this in **two ways**:
  
  » **At design time:** This occurs when we validate level-X RT guarantees.
    - This requires assuming **worst**-case level-X execution costs.
  
  » **At run time:** MC\(^2\) uses a technique called “slack shifting” to re-allocate available slack.
    - This exploits the fact that **actual** costs (at whatever level) may be less than worst-case costs.
MC² Papers
(All papers available at http://www.cs.unc.edu/~anderson/papers.html)

  » Focus is on **schedulability**: How to check timing constraints at each level and “shift” slack?

  » Focus is on **RTOS design**: How to reduce the impact of RTOS-related overheads on high-criticality tasks due to low-criticality tasks?
Outline

• Background.
  » Real-time scheduling basics.
  » Mixed-criticality scheduling.

• MC\(^2\): Proposed mixed-criticality architecture.

• $\text{MAN}^{RT}$: Proposed framework for managing shared caches.

• Future research plans.
$\text{MAN}^{RT}$: Basic Idea

- Builds upon an idea called page coloring.
  - Pages of physical memory are “colored” based on where their contents map in the cache.
  - Accesses to differently colored pages cannot cause cache conflicts.

- New twist: Require tasks to “lock” their needed colors using a real-time locking protocol.
  - Exploits recent work at UNC on optimal real-time multiprocessor locking protocols.
Assume, for the purpose of illustration, that we have a small, direct-mapped cache...

- 512 KB cache size.
- 64 byte cache line size.
- Results in 8192 cache lines.
Page Coloring Review

- 8192 cache lines of 64 bytes each.
- 4 KB physical page.
- There are 64 cache lines per page.
Page Coloring Review

- Page 0 maps to cache lines {0, …, 63}.
- Page 1 maps to cache lines {64, …, 127}.
- The mapping wraps after 128 pages.

This mapping is based upon bits of the physical memory address.
The mapping wraps after 128 pages.

The 128 disjoint sets of pages are assigned 128 colors.

Accesses to differently-colored pages can’t cause cache conflicts.
In practice, set associative (not direct mapped) caches are used.

In a 2-way set associative cache, there are 64 colors instead of 128 colors.

Two entire pages of the same color may be loaded without cache conflicts.
• In an $n$-way set associative cache colored this way, $n$ pages of the same color can be loaded without conflict.
Colors as Shared Resources

• Dedicating a page color to each real-time task has been used to implement cache partitioning.
  » However, this limits the number of real-time tasks that can run concurrently.
  » Since each task is assigned only a subset of the cache, thrashing can result.

• **Our idea:** Use multiprocessor real-time locking protocols to enable tasks to “lock” their needed colors.
Cache Coloring Synchronization Problem

• Each color is a shared resource with n replicas (for an n-way set associative cache).
• Before accessing a physical page, a task must “acquire” or “lock” a replica of the corresponding color.
Solving this Synchronization Problem

- Essentially need to support “nested” accesses of replicated resources on a multiprocessor.

- Protocols in these papers do this with asymptotically optimal worst-case blocking:

Example Job:
- lock a replica of red;
  - access a red page;
- lock a replica of the color blue;
  - access a blue page;
- unlock the blue replica;
- unlock the red replica
We call the resulting framework $\text{MAN}^{\text{RT}}$:

» Cache ($\text{MAN}$) management for Real-Time systems.

Just finished a first prototype assuming:

» where coloring is w.r.t. an 8MB 16-way L3 cache (the last level cache).
Impact on Worst-Case Execution Times (WCETs)

- Assessed by recording observed WCETs for benchmark task systems under:
  1. $\text{MAN}^{\text{RT}}$;
  2. page coloring but no color locking;
  3. no cache management.

- The following graphs show scaling factors $x/y$ where $x$ is WCET under (2) or (3) and $y$ is WCET under (1).
Example WCET Graph

WCET vs. Cache Footprint (WSS Fixed at 128KB)
Example WCET Graph

**WCET vs. Cache Footprint** (WSS Fixed at 128KB)

The amount of the cache a *task* may access.
Example WCET Graph

WCET vs. Cache Footprint (WSS Fixed at 128KB)

The amount of the cache a task may access.

The amount of the cache a job of a task may access.
Example WCET Graph

WCET vs. Cache Footprint (WSS Fixed at 128KB)
Example WCET Graph

WCET vs. Cache Footprint (WSS Fixed at 128KB)
Example WCET Graph

WCET vs. Cache Footprint (WSS Fixed at 128KB)

No Cache Management.

Scaling Factor

Cache Footprint (KB)

Scaling Factor

Footprint (KB)
Example WCET Graph

WCET vs. Cache Footprint (WSS Fixed at 128KB)

No Cache Management.

Coloring Only.

Cache Footprint (KB)

Scaling Factor

Scaling Factor

Footprint (KB)
Example WCET Graph

WCET vs. Cache Footprint (WSS Fixed at 128KB)

With no cache management, WCETs can be 8X higher than $\text{MAN}^{\text{RT}}$. 

[Graph showing scaling factor vs. cache footprint with annotations]

Scaling Factor

Cache Footprint (KB)
Example WCET Graph

WCET vs. Cache Footprint (WSS Fixed at 128KB)

With coloring only, WCETs can be 6.5X higher.
Example WCET Graph

WCET vs. Cache Footprint (WSS Fixed at 128KB)
Example WCET Graph

WCET vs. WSS (Cache Footprint Fixed at 5MB)
Example WCET Graph

**WCET vs. WSS** (Cache Footprint Fixed at 5MB)
Example WCET Graph

WCET vs. WSS (Cache Footprint Fixed at 5MB)

No Cache Management.

Coloring Only.

WSS (KB)

Scaling Factor

Scaling Factor
Example WCET Graph

WCET vs. WSS (Cache Footprint Fixed at 5MB)

Scaling factors decrease with increasing WSS from the 8X and 6.5X factors seen earlier.
Example WCET Graph

WCET vs. WSS (Cache Footprint Fixed at 5MB)
Impact on Schedulability

• We experimentally measured two factors:
  » Average deadline miss ratio
    = percentage of jobs that miss deadlines;
  » Average relative tardiness
    = tardiness/period.
Example Schedulability Graph

Avg. Miss Ratio vs. WSS (Cache Footprint Fixed at 3.5MB)
Example Schedulability Graph

Avg. Miss Ratio vs. WSS (Cache Footprint Fixed at 3.5MB)
Example Schedulability Graph

Avg. Miss Ratio vs. WSS (Cache Footprint Fixed at 3.5MB)

No Cache Management.

Coloring Only.

$\text{MAN}^{RT}$. 

WSS (KB)
Example Schedulability Graph

Avg. Miss Ratio vs. WSS (Cache Footprint Fixed at 3.5MB)

Significant deadline miss ratios.
Example Schedulability Graph

Avg. Miss Ratio vs. WSS (Cache Footprint Fixed at 3.5MB)

- Significant deadline miss ratios.
- Almost no misses.

WSS (KB) vs. Average Miss Ratio
Example Schedulability Graph
Avg. Miss Ratio vs. WSS (Cache Footprint Fixed at 3.5MB)
Example Schedulability Graph

Avg. Rel. Tardiness vs. WSS (Cache Footprint Fixed at 3.5MB)
Example Schedulability Graph

Avg. Rel. Tardiness vs. WSS (Cache Footprint Fixed at 3.5MB)
Example Schedulability Graph

Avg. Rel. Tardiness vs. WSS (Cache Footprint Fixed at 3.5MB)

No Cache Management.

Coloring Only.

$\text{MAN}^{\text{RT}}$. 

WSS (KB)

Avg. Relative Tardiness

Average Relative Tardiness vs. WSS (KB)
Example Schedulability Graph

Avg. Rel. Tardiness vs. WSS (Cache Footprint Fixed at 3.5MB)

Significant tardiness.
Example Schedulability Graph

Avg. Rel. Tardiness vs. WSS (Cache Footprint Fixed at 3.5MB)

Significant tardiness.

Almost no tardiness.

Avg. Relative Tardiness vs. WSS (KB)
Example Schedulability Graph

Avg. Rel. Tardiness vs. WSS (Cache Footprint Fixed at 3.5MB)
Additional Schedulability Study

• For certification, *analytically* assessed schedulability is probably more important.

• To evaluate this, we:
  
  » randomly generated a number of HRT task systems with *varying total utilization* for a 4-core machine;

  » assessed schedulability *analytically* assuming either $\text{MAN}^{RT}$ or no $\text{MAN}^{RT}$ but a WCET scaling factor.

    – Earlier experiments suggest such factors could be 8 or higher in practice.
Example Schedulability Graph
Cache Footprint Fixed at 2.56 MB

cache footprint: 2560KB, Quantum: 25ms
Example Schedulability Graph
Cache Footprint Fixed at 2.56 MB

Plots indicate the fraction of generated task systems deemed (by analysis) to be “schedulable.”
Example Schedulability Graph
Cache Footprint Fixed at 2.56 MB

cache footprint: 2560KB, Quantum: 25ms
Example Schedulability Graph
Cache Footprint Fixed at 2.56 MB

cache footprint: 2560KB, Quantum: 25ms

Schedulability

Total HRT Utilization
Example Schedulability Graph
Cache Footprint Fixed at 2.56 MB

Scaling factor of 1.0.

Schedulability

cache footprint: 2560KB, Quantum: 25ms

Total HRT Utilization
Example Schedulability Graph

Cache Footprint Fixed at 2.56 MB

Scaling factor of 1.5.

Total HRT Utilization

Schedulability

cache footprint: 2560KB, Quantum: 25ms

System Utilization

0.0 0.5 1.0 1.5 2.0

0.0 0.2 0.4 0.6 0.8 1.0
Example Schedulability Graph
Cache Footprint Fixed at 2.56 MB

Scaling factor of 2.0.

Total HRT Utilization
Example Schedulability Graph
Cache Footprint Fixed at 2.56 MB

$\text{MAN}^{RT}$ (same as 2.0 in this case).

Cache footprint: 2560KB, Quantum:

Total HRT Utilization

Schedulability
Example Schedulability Graph
Cache Footprint Fixed at 2.56 MB

Scaling factors of 4.0 to 8.0 (in line with our earlier measurements).
With a total HRT utilization of ~1.25, $\text{MAN}^{RT}$ successfully scheduled almost all systems, but only a tiny fraction could be scheduled assuming a (small) scaling factor of 3.0.
Example Schedulability Graph
Cache Footprint Fixed at 2.56 MB

cache footprint: 2560KB, Quantum: 25ms
Outline

• Background.
  » Real-time scheduling basics.
  » Mixed-criticality scheduling.

• $\text{MC}^2$: Proposed mixed-criticality architecture.

• $\text{MAN}^\text{RT}$: Proposed framework for managing shared caches.

• Future research plans.
Future Research Goals

- Extend $\text{MAN}^{RT}$ by considering:
  - color-assignment schemes;
  - its use in non-partitioned systems;
  - its impact w.r.t. tool-produced WCETs;
  - its impact when controlling all physical pages (so far, we’ve only looked at non-shared task data pages);
  - its use when systems may change dynamically.
Future Research Goals (Cont’d)

• Integrate $\text{MAN}^{\text{RT}}$ within $\text{MC}^2$:
  » may want to be more “parsimonious” w.r.t. “high-criticality colors,” more optimistic w.r.t. “low-criticality colors”;
  » an appropriate “factoring” between the RTOS and middleware is needed.

• Improve our synchronization-related analysis.

• Experiment with realistic workloads.
  » Some of you could really help us here!
URLs

- All of our code can be found at:

- As mentioned earlier, all referenced papers can be found at:
Thanks!

• Questions?