Identifying Sources of Unpredictability in COTS-based Multicore systems

Dakshina Dasari, Benny Akesson, Vincent Nelis, Muhammad Ali Awan and Stefan M. Petters
Outline of the talk

- Multicores and Real-time systems
- Sources of Unpredictability
- Caches, Buses, Memory
- Hardware prefetching
- System Management Interrupts, translation look-aside buffers
- Discussions
A real-time system is a system that reacts to events in the environment by performing predefined actions within specified time intervals.

Correctness of results depends on value and its time of delivery.

"Time is of the essence"
Real time applications

Expected to exhibit the required behavior within **time bounds**.

Essential: **upper bound** on the execution times of all tasks known at design time

Commonly called the **Worst-Case Execution Time (WCET)** and depends on the environment in which the application is executed

Platform: Earlier deployed on uniprocessors

Increasing demands of applications → Deployed on multicores
COTS-based Multicores for hard real time systems: Boon or bane?

- Increased computational power
- SWaP (Lesser Size, Weight, Low Power consumption)
- Natural fault containment for applications
- Decrease in number of computational nodes and wires
- Cost efficient
- Faster time to market
- Integrated functionality → Lesser number of components that can fail (e.g. connectors)

COTS: Commercially-available Off The Shelf
COTS-based Multicores for hard real time systems: Boon or bane?

Disadvantages

- Shared low level hardware resources
  - Shared Caches, inter communication channel, shared memory
- No spatial and temporal isolation
  - Variability in execution times depending on the coscheduled tasks
- Complex features make timing analysis challenging
- Increased unpredictability makes certification a nightmare!
The hard truths:

- Multicores are not yet hard-real time ready
- Safety critical systems form a small market segment.
- Not enough incentives to drive the industry to build “predictable-by-design” multicore systems
- Although around since 2004, no multicore systems are fully certifiable
- The current solution: Disable all the cores except one!!!!
Peripherals

Core

L1 cache

Shared

L2 cache

Memory Controller

Bus Controller Unit

Core

L1 cache

Processor Chip

Shared Front side bus

Memory

Peripherals

Shared Memory
## Unpredictability in Caches

<table>
<thead>
<tr>
<th>Feature</th>
<th>Types</th>
<th>Timing Analyzability</th>
<th>Modern Multicores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replacement Mechanism</td>
<td>LRU, PLRU, Random, FIFO</td>
<td>LRU</td>
<td>Generally PLRU</td>
</tr>
<tr>
<td>Write policy</td>
<td>Write back, write through</td>
<td>Write through</td>
<td>Generally Write back</td>
</tr>
<tr>
<td>Associativity</td>
<td>High</td>
<td>Difficult to analyze</td>
<td>High</td>
</tr>
<tr>
<td>Cache ownership</td>
<td>Private /Shared Caches</td>
<td>Shared caches complex to analyze</td>
<td>Very few have private caches</td>
</tr>
</tbody>
</table>
Avoid Sharing: Cache Partitioning

Per task or Per Core? Hardware /Software? Static /Dynamic

Shared Front Side bus

Shared Memory
Challenges for Bus Contention Delay
Analysis of a task

- Based on memory traffic (cache misses) from tasks on other cores
- Bus Requests not tagged with task priorities
- Reordering of request servicing (Higher priority tasks are stalled longer)
- Arbitration policy decides the availability of the bus
- Prefetching adds to the bus traffic
Bus Arbitration policies: TDMA

- Time Division Multiple Access Bus
  - Real time friendly (Predictable, Composable)
  - Non work-conserving
  - But generally not used in COTS (Commercially available off the shelf) systems

Assignment of bus time slots
## Unpredictability in Buses

<table>
<thead>
<tr>
<th>Feature</th>
<th>Types</th>
<th>Prefered for Timing Analyzability</th>
<th>Present in Modern Multicores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arbitration Mechanism</td>
<td>TDMA, Round Robin, Priority Based, FIFO</td>
<td>TDMA good for Real Time systems</td>
<td>Some weighted round robin/propriety mechanisms</td>
</tr>
<tr>
<td>Transaction ordering</td>
<td>Simple Inorder, pipelining, split transaction</td>
<td>Easier with simple Inorder</td>
<td>Split transaction, request reordering Based on internal priorities</td>
</tr>
<tr>
<td>Hardware prefetching</td>
<td>Enabled/Disabled</td>
<td>Disabled prefetching, reduces speculations</td>
<td>Enabled by default</td>
</tr>
</tbody>
</table>
Prefetching

- Prefetch instructions: Software
  - Compile-time analysis, schedule fetch instructions within user program
- Hardware based prefetching
  - Intel:
    - Adjacent cache line prefetcher
    - Hardware Prefetching: Detecting a stride in the array access pattern
- Disadvantages of OS transparent prefetching
  - Cache pollution: Replaces key cache blocks
  - Uses bus bandwidth
  - Delays important requests
  - Timing unknown to user and causes variations in program behavior
Some interesting research ideas

- Predictable execution model
  - Compatible and predictable execution phases -- prefetch all the required data in your time slot and then start executing (assumes TDMA)
  - Controlled data acquisition and replication models proposed
- Profiling task behavior to analyze requests patterns for single task, multiple tasks and at the core levels and analyze the interference on the bus
- Table driven bus arbiters -- different bus access schedules based on applications
- Budgeting the bus bandwidth -- Memory centric scheduling
- And more in the paper
Shared Memory Contention

Tasks

Core

L1 cache

Core

L1 cache

Core

L1 cache

Core

L1 cache

Shared L2 cache

Bus Controller Unit

Processor Chip

Requests

Shared Front Side bus

Memory Controller

Shared Memory
Memory organization

- Column address
- Row buffer
- I/O (read/write)
- Row address
- Activate (open)
- Precharge (close)
- Banks
- Data
- Column address
- I/O
Unpredictability in memory accesses

- DRAM is the center of memory hierarchy:
  - High density and high capacity
  - Low cost but slow access (compared to SRAM)
- DRAM access generally considered to be uniform.
- Non-uniform access latencies exist within DRAM
- The Row-buffer *caches the last accessed row in DRAM*
Page policy: Open Page

- Row address
- Column address
- Data
- I/O
- Read
- Write
- Activate (open)
- Precharge (close)

Keep the current row open
Good if high locality of reference
Page policy: Close Page

- Row address
- Column address
- Row buffer
- Activate (open)
- Precharge (close)
- Read
- Write
- I/O
- Data

Close the row after access
Good for random references
Unpredictability in DRAM Access

Page Policies
- Open page policy
- Close page policy
- Choice of policies -> variability in access times

Request reordering
- Prioritize requests that are present in an open row
- Prioritize reads over writes

DRAM refreshes
- Periodic refresh (every 7.5 microseconds)
- Irregular refreshes (in idle cycles)
- Increase the stall
Effect of system management interrupts

- SMI: System management interrupt
- SMM: System management mode

1. Executing real time task code
2. Enter SMM (execute handler)
3. Detect chipset errors
4. Handle system errors
5. CPU overheating
6. Fan control
7. Resume real time task code

(S100s of cycles)

Non maskable, OS transparent

X cycles

Y cycles

SMM: System management mode
SMI: System management interrupt
Thermal and Power saving strategies

- Processors like Intel employ “sleep states” by varying the power supplied to different devices.
- Wake up latencies vary with different sleep state levels.
- Deeper sleep states – Higher wake up latencies (More Idle devices are powered down).
- If a real-time task is now scheduled—it incurs the wake up latency.
- Power saving strategies cannot be disabled—may cause harm in the long run.
- Adaptive Thermal Monitoring
  - Reduce the temperature when threshold is breached by reducing the frequency and voltage adaptively.

**Many of these features are enabled by default and must be accounted for in the analysis or be disabled with caution.**
Variability in the Execution times

D1: Cache related Preemption delay
D2: Bus contention delay
D3: Delay in the shared memory subsystem
Dx: Other parameters
Discussions

- Multicores are here to stay and will be eventually adopted by safety critical systems
- Computational capabilities undisputable but must be assessed for predictability for real time systems
- Unpredictability at each level of system design in current hardware.
- Task memory profiles also an inherent characteristic (apart from the $C_i, T_i, D_i$)
- Holistic end to end analysis must consider all the possible sources of unpredictability to deploy safe and robust systems
- Academic research generally assume simplified models, validated on unrealistic simulators which do not capture the various sources of unpredictability — Pessimistic results and solutions cannot be applied to the industry
- Need to study the hardware in detail to achieve tighter and acceptable bounds

So much more to be done!
Additional details in paper

• Research study in the related areas
  • Different approaches to solve these issues
  • Limitations of current approaches and suggestions