Part 2

Multicore Real-Time Systems

-- Challenges & Solutions

Wang Yi
Uppsala University

VTSA Summer School
Luxembourg, Sept 2010

OUTLINE

- Multicore Challenges (Real-Time Applications?)
  * Why and what are multicore?
  * What we are doing in Uppsala: CoDeR-MP
  * The timing analysis problem

- Possible Solutions – Partition/Isolation
  * Dealing with Cache Contention [EHSOFT 2009]
  * Dealing with Bus Interference [RTSS 2010]
  * Dealing with Core Sharing [RTAS 2010]

What is multi-core, and why?

Multicore = Multiple hardware threads sharing the memory system

Free lunch is over, Erik Hagersten

The free lunch is over & Multicores are coming!

Erik Hagersten
Chief Architect at SUN (till 1999)
Professor of Computer Architecture, Uppsala
Theoretically you may get:

- Higher Performance
  - Increasing the cores -- unlimited computing power \( \infty \)!

- Lower Power Consumption
  - Increasing the cores, decreasing the frequency
    - Performance (IPC) = \( \text{Cores} \times F \)  \( \rightarrow \) \( 2 \times \text{Cores} \times F/2 \) \( \rightarrow \) \( \text{Cores} \times F \)
    - Power = \( C \times V^2 \times F \) \( \rightarrow \) \( 2 \times C \times (V/2)^2 \times F/2 \) \( \rightarrow \) \( C \times V^2/4 \times F \)
  
\( \Rightarrow \) Keep the "same performance" using \( \frac{1}{4} \) of the energy (by doubling the cores)

This sounds great for embedded & real-time applications!

UPMARC Research Areas

Applications & Algorithms
- Climate simulation
- PDE solvers
- Parallel algorithms for RT signal processing
- Parallelization of network protocols

Verification & Language Technology
- Erlang, language constructs/libraries, run-time systems
- Static analysis, Model-checking, testing, UPPAAL

Resource Management
- Efficiency: performance opt.
- Predictability: real-time applications

Objective (CoDeR-MP)

New techniques for
- High-performance software for soft RT applications &
- Predictable software for hard RT applications on multicore

Industry participation
- Control Software for Industrial Robots — ABB robotics
- Tracking with parallel particle filter — SAAB
Real-Time Tracking with parallel particle filter – SAAB

Parallelization
(Speed-up for PF algorithms)

Real-Time Control – ABB Robotics

OUTLINE

Multicore Challenges
- Why and what are multicores?
- What we are doing in Uppsala: CoDeR-MP
- The timing analysis problem

Possible Solutions – Partition/Isolation
- Dealing with Cache Contention [EMSOFT 2009]
- Dealing with Bus Interference [RTSS 2010]
- Dealing with Core Sharing [RTAS 2010]

Single-Processor Timing Analysis

Sequential Case (WCET analysis)

Concurrent Case (Schedulability analysis)

On single processor:

WCET = \#instructions + “cache miss penalty”

“Cache miss penalty” can be estimated “precisely”: by e.g. abstract interpretation – based on the history of executions.
On multicore processor:

\[ WCET = \#\text{instructions} + \text{"cache miss penalty"} + \ldots \]

"Cache miss penalty" can be much larger due to cache contentions from the other cores... and also bus delays.

WCET of a single task cannot be estimated in isolation.

---

An Example Architecture

Cache analysis on multicore

- L2 cache contents of task 1 may be over-written by task 2

---

An Experiment on a LINUX machine with 2 cores

(Wang Yi)

WCET (vary 10% – 50%)

---

Cache analysis on multicore

- L2 cache contents of task 1 may be over-written by task 2
The multicore challenge: **WCET analysis**

- Must explore all interleavings of "execution paths" on all cores
- Must represent "precise" timing information on each core (to keep track of the progress on each core and cache contents)

The multicore challenge: **Schedulability analysis**

- \#cores < \#tasks

---

**Cyclic dependence**

Multicore schedulability analysis

WCET analysis

**The “Impossible” Problem**

1. We must "schedule" the shared cache lines
2. We must "schedule" the shared memory bus
   - when cache misses occur
3. We must "schedule" the shared cores

---

**OUTLINE**

- **Multicore Challenges**
  - Why and what are multicores?
  - What we are doing in Uppsala: CoDeR-MP
  - The timing analysis problem

- **Possible Solutions — Partition/Isolation**
  - Dealing with Shared Caches [EMSOFT 2009]
  - Dealing with Bus Interference [RTSS 2010]
  - Dealing with Core Sharing [RTAS 2010]
Cache analysis on multicore

Cache-Coloring: partitioning and isolation

Cache-Coloring: partitioning and isolation

Cache-Coloring: partitioning and isolation

An Experiment on a LINUX machine with 2 cores
with Cache Coloring/Partitioning [ZhangYi et al]
What to do when \#tasks > \#cores?

Cache-Aware Scheduling and Analysis for Multicores [EMSOFT 2009]

Main message:
- “Isolation”: tasks of “same color” should not run at the same time
- The schedulability problem can be solved as an LP problem

OUTLINE

- Multicore Challenges
  - Why and what are multicores?
  - What we are doing in Uppsala: CoDeR-MP
  - The timing analysis problem
- Possible Solutions – Partition/Isolation
  - Dealing with Shared Caches [EMSOFT 2009]
  - Dealing with Bus Interference [RTSS 2010]
  - Dealing with Core Sharing [RTAS 2010]
Combining Abstract Interpretation and Model Checking for Multicore WCET Analysis [RTSS 2010]

**Basic Idea:**
Construct a timed model -- describing all possible timed traces of bus requests, that are possible from each core.

---

**Example (CFG with CHMC info from AI analysis)**

```
BB0  AM
  \-----\-----
   |   |   |
BB1  AH  BB3
  |   \-----
  \     |   |
BB2  AM  BB4
  |   \-----
  \     |   |
BB5  NC  BB2
```

**Private Cache Analysis by AI**
- **MUST analysis**, classify instructions that are predicted as AH
- **MAY analysis**, classify instructions that are predicted as AM
- **PERSISTENCE analysis**, classify instructions that are predicted as FM
- Everything else as Not “Classified (NC)”

---

**From CFG with CHMC to Timed Automata**
- **Modeling AH instructions**
  - If an instruction is AH, it never accesses the bus, so we only model the L1 Cache access time and the instruction execution time

```
\( c_{L1Hit} + \text{InstTime} \)
```

---

**From CFG with CHMC to Timed Automata**
- **Modeling AM instructions**
  - An AM instruction is guaranteed to access the shared bus, so we model bus access behavior and instruction execution

---

**Combining Static Analysis & Model-Checking**

1. Local cache analysis by abstract interpretation
2. Construct a timed automaton for each program to model the precise timing information on when to access the shared bus
3. Construct a timed automaton modeling the bus arbitration
4. Explore the TA models using UPPAAL to get the WCETs
From CFG with CHMC to Timed Automata

Modeling FM instructions

- For an FM instruction, one should distinguish between the first reference and the other references.

The upper path models the first reference to the instruction, which is a cache miss (access bus).

The lower path models the other references to the instruction, which are cache hits (do not access bus).

Modeling NC instructions

- So for NC instructions, we have to model both possibilities of cache misses and cache hits, and let the model checker to explore them.

The “cache miss” case for an NC instruction.

The “cache hit” case for an NC instruction.

Optimization by grouping

- To reduce state space by reducing the number of locations and edges, we grouping consecutive FM or AH instructions.

Given a sequence: \(<FM, AH, AH, FM, AH, AH>\)

Without grouping: 12 locations

With grouping: 6 locations (PostNode not included)

Example (CFG with CHMC info from AI analysis)

- AM
- NC
- AH
- BB0
- BB1
- BB2
- BB3
- BB4
- BB5

The Timed Automaton Describing "Bus Interference"

Modeling the Shared Bus

- Example: TDMA bus schedule

  The bus schedule is composed of consecutive segments
  Segments are divided into slots, where each slot is assigned to one core.
Modeling the TDMA Bus

- Timed automaton for the TDMA bus

### Slot switch

- Waiting for new requests
- Not enough time left for the request
- The request cannot be serviced in the current slot
- Check if there is a pending request
- Enough time and the right slot

Servicing a request

Putting All Together

- Now, we have
  - TA models for the programs running on all cores, describing all bus requests annotated with timing info, that are possible from the cores
  - TA model for a given bus arbitration protocol e.g. TDMA, FCFS, RR ...

**WCET estimation**

- Let the UPPAAL model checker explore the network of TA models
- The WCETs are extracted from the clock constraints within the UPPAAL model checker

**Scalability:** for TDMA, it scales very well; the analysis can be done separately for each program and the bus schedule.

Experiments and Evaluation

**WCET Benchmark programs (Maladalen)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>bs</td>
<td>Binary search algorithm for an array</td>
<td>78</td>
</tr>
<tr>
<td>edn</td>
<td>Finite Impulse Response (FIR) filter calculations</td>
<td>896</td>
</tr>
<tr>
<td>f dct</td>
<td>Fast Discrete Cosine Transform</td>
<td>647</td>
</tr>
<tr>
<td>insert</td>
<td>Insertion sort on a reversed array</td>
<td>106</td>
</tr>
<tr>
<td>j dct</td>
<td>Discrete Cosine Transformation on a pixel block</td>
<td>691</td>
</tr>
<tr>
<td>mainm</td>
<td>Matrix multiplication</td>
<td>247</td>
</tr>
</tbody>
</table>

Results for the TDMA Bus

- System configurations
  - Duo-core or 4-core systems
  - L1 Cache size = 2KB,
  - Cache associativity = 4
  - Cache line size = 8B
  - L1 hit latency = 1 cycle
  - Instruction execution = 1 cycle
  - Bus service time = 40 cycles
  - Two different slot sizes: 100 cycles, 200 cycles
Results for the TDMA Bus

- The WCET of each program can be calculated independently for the TDMA bus.

- The worst-case bus delay scenario:
  - A bus request arrives in the slot assigned to it, but finds that there are only 39 cycles left, which is just not enough to serve the request.
  - For slot size 100, worst-case delay = 39 + 100 + 40 = 179
  - For slot size 200, worst-case delay = 39 + 200 + 40 = 279

- Improvement:
  \[ \frac{\text{WCET}_{\text{AI+MC}}}{\text{WCET}_{\text{AI+MC}} - 1} \]
  - Describes how much our approach can tighten compared to assuming worst-case bus delay.

Results for the TDMA Bus

- Results for a duo-core system with slot size 100

<table>
<thead>
<tr>
<th>Programs</th>
<th>WCET</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>bs</td>
<td>8,282</td>
<td>14,644</td>
</tr>
<tr>
<td>edn</td>
<td>9,219,082</td>
<td>16,356,100</td>
</tr>
<tr>
<td>lct</td>
<td>268,882</td>
<td>479,946</td>
</tr>
<tr>
<td>insertot</td>
<td>21,041</td>
<td>29,702</td>
</tr>
<tr>
<td>jclkint</td>
<td>315,882</td>
<td>563,936</td>
</tr>
<tr>
<td>malmult</td>
<td>151,241</td>
<td>174,390</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>62%</td>
</tr>
</tbody>
</table>

Results for the TDMA Bus

- Results for a duo-core system with slot size 200

<table>
<thead>
<tr>
<th>Programs</th>
<th>WCET</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>bs</td>
<td>16,082</td>
<td>30,244</td>
</tr>
<tr>
<td>edn</td>
<td>18,428,441</td>
<td>34,946,900</td>
</tr>
<tr>
<td>lct</td>
<td>529,682</td>
<td>1,005,350</td>
</tr>
<tr>
<td>insertot</td>
<td>31,641</td>
<td>50,902</td>
</tr>
<tr>
<td>jclkint</td>
<td>624,482</td>
<td>1,102,740</td>
</tr>
<tr>
<td>malmult</td>
<td>179,241</td>
<td>231,790</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>75%</td>
</tr>
</tbody>
</table>

Results for the TDMA Bus

- Results for a 4-core system with slot size 100

<table>
<thead>
<tr>
<th>Programs</th>
<th>WCET</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>bs</td>
<td>16082</td>
<td>53644</td>
</tr>
<tr>
<td>edn</td>
<td>18404164</td>
<td>62519600</td>
</tr>
<tr>
<td>lct</td>
<td>529682</td>
<td>1793450</td>
</tr>
<tr>
<td>insertot</td>
<td>32082</td>
<td>82702</td>
</tr>
<tr>
<td>jclkint</td>
<td>628164</td>
<td>2110940</td>
</tr>
<tr>
<td>malmult</td>
<td>179241</td>
<td>117890</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>197%</td>
</tr>
</tbody>
</table>

Results for the FCFS Bus

- System configurations:
  - Duo-core system
  - L1 Cache size = 8KB
  - Cache line size = 8B
  - Cache associativity = 4
  - L1 cache hit latency = 1 cycle
  - Instruction execution time = 1 cycle
  - Bus service time = 40 cycles
Results for the FCFS Bus

Evaluation method
- Grouping the six benchmark programs into two task sets
  - \{bs, edn, fdct\} and \{insertsort, jfdctint, matmult\}
- Each task set is allocated on one core
- The tasks within the same task set are statically scheduled

<table>
<thead>
<tr>
<th>Schedules</th>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>bs, edn, fdct</td>
<td>matmult, insertsort, jfdctint</td>
</tr>
<tr>
<td>S2</td>
<td>bs, edn, edn</td>
<td>matmult, insertsort, jfdctint</td>
</tr>
<tr>
<td>S3</td>
<td>bs, edn, bs</td>
<td>matmult, insertsort, naswati</td>
</tr>
<tr>
<td>S4</td>
<td>bs, edn, bs</td>
<td>insertsort, jfdctint, naswati</td>
</tr>
<tr>
<td>S5</td>
<td>bs, edn, bs</td>
<td>jfdctint, matmult, naswati</td>
</tr>
<tr>
<td>S6</td>
<td>bs, edn, bs</td>
<td>jfdctint, naswati, naswati</td>
</tr>
<tr>
<td>S7</td>
<td>bs, edn, bs</td>
<td>jfdctint, naswati, naswati</td>
</tr>
<tr>
<td>S8</td>
<td>bs, edn, bs</td>
<td>jfdctint, naswati, naswati</td>
</tr>
</tbody>
</table>

Results for the FCFS Bus

The worst-case bus delay scenario
- A request \(req_i\) arrives when the bus is servicing a request from the other core which is issued immediately before \(req_i\)
- Given the above system configurations, the worst-case bus delay for the FCFS bus is 80 cycles (two times the bus service time)

<table>
<thead>
<tr>
<th>Programs</th>
<th>WCET (AI + MC)</th>
<th>WCET AI + Worst-Case</th>
<th>Maximal Impr.</th>
<th>Average Imp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>bs</td>
<td>3,802</td>
<td>4,319</td>
<td>82%</td>
<td>67%</td>
</tr>
<tr>
<td>edn</td>
<td>240,267</td>
<td>245,970</td>
<td>15%</td>
<td>12%</td>
</tr>
<tr>
<td>fdct</td>
<td>37,573</td>
<td>44,620</td>
<td>69%</td>
<td>46%</td>
</tr>
<tr>
<td>insertsort</td>
<td>14,968</td>
<td>15,763</td>
<td>23%</td>
<td>23%</td>
</tr>
<tr>
<td>jfdctint</td>
<td>40,153</td>
<td>48,056</td>
<td>45%</td>
<td>45%</td>
</tr>
<tr>
<td>matmult</td>
<td>138,406</td>
<td>140,117</td>
<td>4%</td>
<td>4%</td>
</tr>
</tbody>
</table>

Average improvement for all programs: 33%

Now, assume that we have a “safe WCET bound” for each task

Remember, we need to:
- “partition” the shared caches
- “partition” the shared memory bus

OUTLINE

Multicore Challenges
- Why and what are multicore?
- What we are doing in Uppsala: CoDeR-MP
- The timing analysis problem

Possible Solutions — Partition/Isolation
- Dealing with Shared Caches [EMSOFT 2009]
- Dealing with Bus Interference [RTSS 2010]
- Dealing with Core Sharing [RTAS 2010]
Dealing with Shared Cores

Multiprocessor Scheduling [a lot of excellent work done by Baruah et al]