Scheduling of “On-Chip Shared Resources” on Multicores for Predictability

Wang Yi
Joint work with Pontus Ekberg, Guan Nan & Martin Stigge
Uppsala University

UPMARC
Uppsala Programming for Multicore Architectures Research Center
Multi-core Architecture

L2 Cache

L2 caches
Bandwidth
Cores

Shared Resources

Off-chip memory

Bandwidth
What are we doing now in Uppsala?

- **Cache partitioning/Cache-Aware-Scheduling** [EMSOFT 09]
  - for task-level timing analysis, Use existing techniques e.g. aiT/Absint
  - for system-level timing analysis, Use RTA techniques - response time analysis for multiprocessor scheduling e.g. [RTSS 09]

- **Scheduling of Bandwidth/Bus Access**
  - (in progress)

- **Multiprocessor Scheduling**
  - Mainly “theory work” but hope for “insightful results”
  - We have developed a fixed-priority algorithm for multiprocessor scheduling with Liu & Layland’s utilization bound – [submitted RTAS 2010]
UPMARC Research Areas
10 Millions $, 10 years @ Uppsala

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

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

Resource Management
- Efficiency: soft RT applications
- Predictability: hard RT applications
- Both hard & soft RT applications
Real-Time Control Systems

the software contains both Hard and Soft RT tasks

- Migration to multicore platform
- How to exploit performance gains, but maintain predictability?
Assumptions (from the industries)

- No specific “hardware support”
  - ... ... we can not simply “remove” the shared caches!
- Everything we do should be implementable in software e.g. cache coloring
  - easy to integrate in existing standard software e.g. RT-LINUX
- However, you can change your applications to make Efficient and Predictable uses of the shared hardware resources
Cache-Aware Scheduling [EMSOFT 09]
coloring the cache for predictability
An Experiment on a LINUX machine with 2 cores

(by Zhang Yi, North Eastern Univ, China)
Cache analysis on multicores

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

- core 1: Private L1 cache
- core 2: Private L1 cache
- core 3: Private L1 cache
- core 4: Private L1 cache
- Task 5

Shared L2 cache
Cache-Coloring: partitioning and isolation
Cache-Coloring: partitioning and isolation

- E.g. LINUX – Power5 (16 colors)
Cache-Coloring: partitioning and isolation

Task 1

Task 2

Task 3

Task 4
An Experiment on a LINUX machine with 2 cores

with Cache Coloring/Partitioning

(by Zhang Yi, North Eastern Univ, China)
What to do when \#tasks > \#cores?

Cache-Aware Scheduling and Analysis
Cache-Coloring: partitioning and isolation
Cache-Coloring: partitioning and isolation
Fixes-Priority Scheduling Algorithms

- (1) algorithm with blocking (FPb): a task can run if
  - it has the highest priority among waiting tasks
  - there is a free core and
  - there are enough free colors
  otherwise all in the queue must wait

- (2) algorithm without blocking: a task may run if
  - There is a free core and
  - It has the highest priority among those that have enough free colors

Problem with non-blocking sch: unbounded priority-inversion
• Can all task instances meet their deadlines?
• This problem can be solved as an LP problem
Problem Window Analysis (due to Baruah)

- Suppose that the task set is non-schedulable.
- Let $J_k$ be the first job missing its deadline. Its release time is $r_k$, and define $l_k = r_k + S_k$.
- $[r_k, l_k]$ of length $S_k$ is the problem window.
Problem Window Analysis (with cache)

- $J_k$ cannot run if at least $A - A_{k}^{\text{max}} + 1$ colors are busy

where $A_{k}^{\text{max}} = \max_{i \leq k} A_i$ (in the blocking-style scheduling, a job waits as long as any higher priority job is waiting)
Dividing the problem window

- **α-interval**: in which all cores are busy;
- **β-interval**: in which
  - at least one core is idle and
  - at least $A - A_k^{\text{max}} + 1$ cache partitions are “busy”
Resource usages within the windows

- $\alpha_i$ denotes $\tau_i$'s accumulated execution time during $\alpha$ - intervals.
- $\beta_i$ denotes $\tau_i$'s accumulated execution time during $\beta$ - intervals.
Towards a sufficient schedulability test

- Assume that $J_k$ is not schedulable. Then the sum of the accumulated lengths of the $\alpha$- and $\beta$- intervals is at least $S_k$

- So it must hold that

$$\sum_i \left( \frac{1}{M} \alpha_i + \frac{A_i}{A - A_{k}^{\text{max}} + 1} \beta_i \right) \geq S_k$$

- We can use an LP solver to detect, if $\alpha_i$ and $\beta_i$ can be chosen in a way to satisfy this condition. If this is not the case, then $J_k$ is schedulable.
As an LP problem

Maximize \[ \sum_i \left( \frac{1}{M} \alpha_i + \frac{A_i}{A - A_{k}^{\text{max}} + 1} \beta_i \right) \]

subject to:

\[ \forall j : \alpha_j + \beta_j \leq I_k^j \]

\[ \forall j : \alpha_j \leq \frac{1}{M} \sum_i \alpha_i \]

\[ \forall j : \beta_j \leq \frac{1}{A - A_{k}^{\text{max}} + 1} \sum_i A_i \beta_i \]
As an LP problem

Maximize \[ \sum_i \left( \frac{1}{M} \alpha_i + \frac{A_i}{A - A_k^{\text{max}} + 1} \beta_i \right) \]

subject to:

\[ \forall j : \alpha_j + \beta_j \leq I_k^j \]

\[ \forall j : \alpha_j \leq \frac{1}{M} \sum_i \alpha_i \]

\[ \forall j : \beta_j \leq \frac{1}{A - A_k^{\text{max}} + 1} \sum_i A_i \beta_i \]

Interference Constraint:

$I_k^j$ is the upper bound of the work done by $\tau_j$ in the problem window.
As an LP problem

\[
\text{Maximize } \sum_{i} \left( \frac{1}{M} \alpha_i + \frac{A_i}{A - A_{max}^k + 1} \beta_i \right)
\]

\text{subject to: } \forall j : \alpha_j + \beta_j \leq I_j^k

\forall j : \alpha_j \leq \frac{1}{M} \sum_i \alpha_i

\forall j : \beta_j \leq \frac{1}{A - A_{max}^k + 1} \sum_i A_i \beta_i

Core Constraint:
The work done by a task in the \( \alpha \)-intervals cannot be larger than the total length of the \( \alpha \)-intervals.
As an LP problem

Maximize \( \sum_i \left( \frac{1}{M} \alpha_i + \frac{A_i}{A - A_k^{\text{max}} + 1} \beta_i \right) \)

subject to:

\( \forall j : \alpha_j + \beta_j \leq I_k^j \)

\( \forall j : \alpha_j \leq \frac{1}{M} \sum_i \alpha_i \)

\( \forall j : \beta_j \leq \frac{1}{A - A_k^{\text{max}} + 1} \sum_i A_i \beta_i \)

**Cache Constraint:**
The work done by a task in the \( \beta \) - intervals can not be larger than (the upper bound of) the total length of the \( \beta \) - intervals.
A sufficient schedulability test for multicores

For each task $\tau_k$, let $\chi_k$ denote the solution of the LP problem

Maximize \[ \sum_i \left( \frac{1}{M} \alpha_i + \frac{A_i}{A - A_k^{\text{max}} + 1} \beta_i \right) \]

subject to:
\[ \forall j : \alpha_j + \beta_j \leq I_k^j \]
\[ \forall j : \alpha_j \leq \frac{1}{M} \sum_i \alpha_i \]
\[ \forall j : \beta_j \leq \frac{1}{A - A_k^{\text{max}} + 1} \sum_i A_i \beta_i \]

$\tau$ is schedulable by $\text{FP}_{\text{CA}}$, if for each task $\tau_k \in \tau$ it holds

$\chi_k < S_k$
Conclusions

- To achieve predictability of “hard real-time applications”
  - We need “Resource isolation”
    - e.g. cache partitioning/coloring, time slicing
  - The scheduler must ensure that tasks using the same colors, don’t run in parallel

- The schedulability problem can be solved using LP solvers
  - Scalable, may deal with task sets with thousands of tasks
  - Sufficient test (due to over-approximation)

- Future work (in progress):
  - Improving the analysis precision
  - Studying the influence of cache-coloring and allocation on system performance and timing behaviours
Scheduling and Analysis of Bus Access (recent work)
Multi-core Memory System

- Core 1: Private L1 cache
- Core 2: Private L1 cache
- Core 3: Private L1 cache
- Core 4: Private L1 cache

- Shared Memory
  - 1-2 cycles
  - 5-20 cycles
  - 100 – 200 cycles
Scheduling of Bus Access

Task: (1) Memory Read, (2) Local Computation in L1, (3) Memory Write

Task 1

Task 2

Task 3

Task 4

M: Shared Memory
System-level timing analysis

- #cores < #tasks
- What is the WCRT (worst case response time) for each task?
The behaviour will be repeated with a period: $T$
The execution of task would look like:

Read input or load context

Local computation

Write output or save context
The execution of task would look like:

- **Local scheduling**
- **Local computation**
- **Bus request**
- **Bus scheduling**
The Bus Scheduling problem to solve:

Bus request

Local computation

C

Bus request

Local scheduling

R

W

Bus scheduling

Bus scheduling

Make sure the various Schedulers work such that This point is not later than The deadline
Task graph/Arbitrary Acyclic Graph

guaranteeing End-to-End timing constraints
An example system

Problem (to solve): How to schedule the Bus access such that all task instances will be computed within their periods?
Preliminary results

- This can be easily modelled using UPPAAL
  - model the scheduling strategies e.g. TDMA, EDF, FCFS, RoundRobin … as a timed automaton
  - the R/W-operations, task releases as timed automata
  - schedulability analysis as a reachability problem
- The TIMES tool is even better for this purpose – schedulability analysis
- The analysis is surprisingly efficient
Thanks