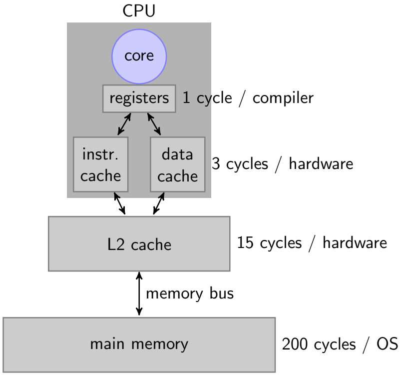
Interactions between WCET analysis and scheduling

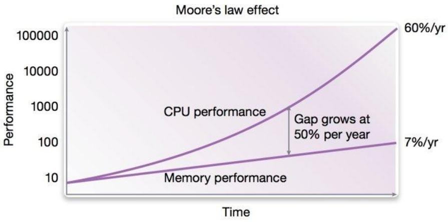
Guillaume Phavorin and Pascal Richard

LIAS/Université de Poitiers

April 4th, 2014

Contents


CPU
core
registers
1 cycle / compiler
instr. cache
data cache
3 cycles / hardware
L2 cache
15 cycles / hardware
memory bus
main memory
200 cycles / OS


Moore's law effect
100000
10000
1000
100
10
Performance
60%/yr
CPU performance
Gap grows at 50% per year
7%/yr
Memory performance
Time

Cache

Small and fast memory (compared to the main memory).
$\rightarrow$ to bridge the gap between the processor speed and the main memory access time.
$\rightarrow$ by storing:

- data that is frequently accessed (temporal locality),
- data that will (or may) be accessed next (spatial locality).

Instruction vs data caches, shared cache, cache hierarchy...

When a block is accessed:

- in cache: **cache hit** $\rightarrow$ low cost ($\approx$ 1 to 4 clock cycles),
- not in cache: **cache miss** $\rightarrow$ high cost ($\approx$ 8 to 32 cycles).

Cache organization

Cache:

- divided into **cache lines** of equal size:
  - number of contiguous bytes transferred from the main memory to the cache.
- that may be grouped into sets:
  - **direct-mapped**: 1 line = 1 set
    - a memory block can be mapped to only *one line*.
  - **fully-associative**: only one set containing all lines
    - a memory block can be mapped *everywhere* in the cache.
  - **set-associative**: lines equally divided into several sets
    - a memory block can be mapped only to *one set* BUT everywhere in it.

Eg: 8kB direct-mapped instruction cache with a 8 bytes line size and a 4 bytes instruction size (ARM7).

Replacement policy

Offline:

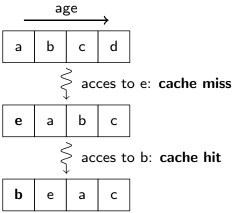
Belady's rule: the block whose next request is the furthest in the future is evicted. $\Rightarrow$ OPTIMAL.

Online:

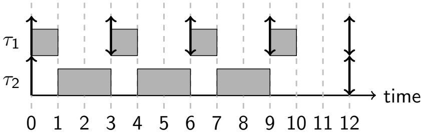
No optimal policy, as the access sequence is not known.

- LRU: Least Recently Used.

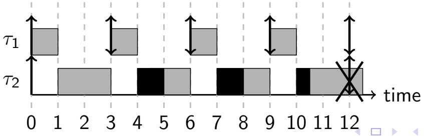
Example with a 4-way associative cache set



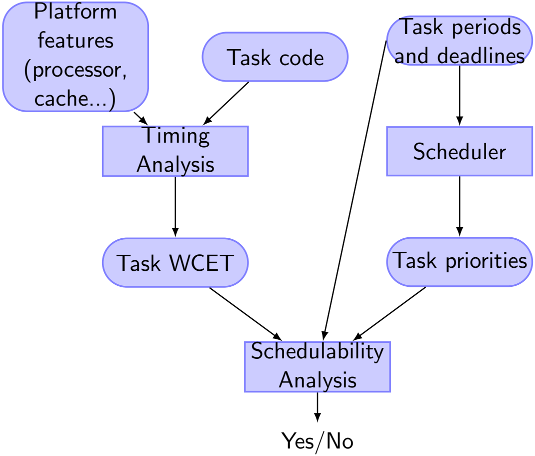
Cache-Related Preemption Delay (CRPD)



adding preemption costs



Classical approaches



"Magic" WCET

Goal:

- WCET accounting for all potential preemption delays.
    - preemption costs have no longer to be considered into the scheduling analysis.

Magic WCET: Approach 1

Easiest way to incorporate preemption delays into the WCET:

- **every access** $\rightarrow$ considered to be a **cache miss** (as if the cache was disabled)

But very pessimistic, and cache benefits are not taking any more into account.

Magic WCET: Approach 2

- taking cache benefits into account → tighter WCET,
- upper-bounding cache effects (**CRPD**: Cache-Related Preemption Delays) → to achieve predictability.

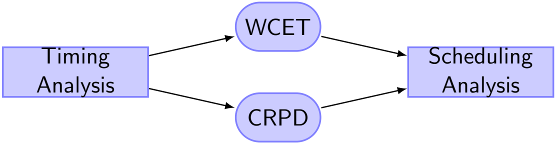
→ by conducting cache analyses:

1. for WCET: representation of cache contents to identify accesses that will be **"Always Hits"**.
2. to bound the impact of a preemption at a given program point.

$$\underbrace{WCET_{w/o\ preemption}}_{(1)} + n \cdot \underbrace{CRPD}_{(2)}$$

Problem: how to get n ? → very dependant on the chosen scheduling policy and the considered task system.

CRPD incorporated into the scheduling analysis



$$R_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot \left(C_j + \underbrace{\gamma_{i,j}}_{\textit{CRPD parameter}} \right)$$

$hp(i)$: tasks of higher priority than task τ_i.
$\gamma_{i,j}$: preemption cost due to each job of a higher priority preempting task τ_j executing within the worst-case response time of task τ_i.

Controlling preemption

Using well-known scheduling policies such as RM or EDF, schedulability improvement can be achieved by:

- limiting preemptions (*Buttazzo et al. 2013*),
- selecting the best possible preemption points in the program code, based on their overhead cost (*Bertogna et al. 2011*),
- ...

$\Rightarrow$ reduce CRPD.

But, scheduling decisions are independent from any cache-related parameter.

All previous strategies → use of "classical" scheduling policies (RM, EDF...):

- CRPD added to achieve better predictability,
- **but** scheduling decisions are independant from any cache-related parameter.

Would it not be better to take scheduling decisions to reduce CRPD?

- Taking delays due to the use of caches into account in the definition of scheduling algorithms.
  - Task model modified → addition of cache-related parameters:
    1. representing the sequence of accessed block,
    2. representing preemption cost.

CRPD-aware scheduling

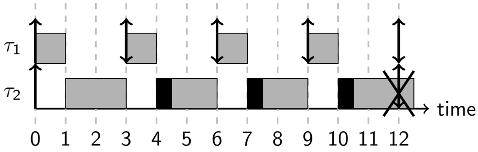
Scheduling decisions taken based on preemption costs $\rightarrow$ to minimize the general overhead.

Task defined by $\tau_i(C_i, D_i, T_i, \gamma)$

- C_i: WCET without preemption cost estimated when τ_i is executed fully non preemptively,
- γ: CRPD for one preemption $\rightarrow$ the same for all program points and all tasks.

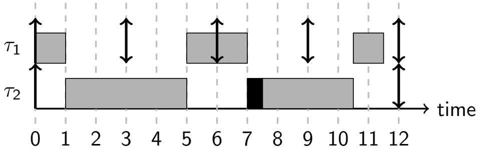
$\tau_1(1,3)$, $\tau_2(7,12)$, CRPD: $\gamma = 0.5$.

- Fixed-Job Priority Scheduling:



$\Rightarrow$ Fixed-Task and Fixed-Job Priority schedulers are **not optimal**.

- CRPD-aware scheduling:



Simplified scheduling with CRPD problem:

- INSTANCE:
  - a finite set of n tasks $\tau_i(C_i, D_i, T_i)$,
  - a positive number γ representing the Cache-Related Preemption Delay incurred by $\tau_i, 1 \leq i \leq n$ at every resume point after a preemption.
- QUESTION:
  - Is there a uniprocessor preemptive schedule meeting the deadlines?

$\Rightarrow$ the scheduling problem with CRPD is **NP-hard**.

Cache-aware scheduling

Scheduling with information about cache state and block reuse by the different tasks.

- eg: tasks using the same data or a common external library.

Job defined by $J_i(C_i, D_i, S_i)$:

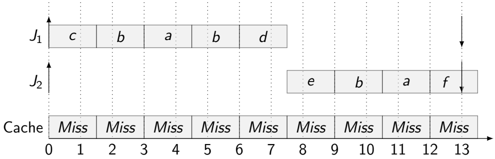
- C_i: WCET considering that all requested memory blocks are hits in the cache,
- D_i: relative deadline of the job,
- S_i: string denoting the sequence of memory blocks used during the job execution (no *if-then-else* structure).

Hypotheses:

- a single cache line,
- hit cost $= 0$, miss cost $= BRT$ (Block Reload Time),
- job preemption $\rightarrow$ only before requesting the next block,
- synchronous jobs.

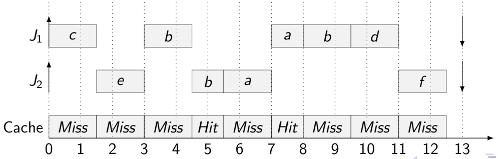
Cache size $= 1$, exec.(hit) $= 1$, exec.(miss) $= 1.5$, $S_1 = cbabd$, $S_2 = ebaf$.

- Fixed-Job Priority Scheduling ($prio(J_1) > prio(J_2)$):



$\Rightarrow$ Fixed-Task and Fixed-Job Priority schedulers are **not optimal**.

- Cache-aware scheduling:

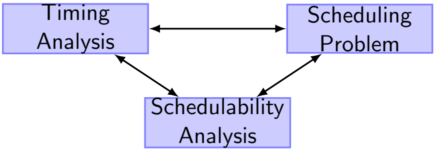


Simplified scheduling with cache memory problem:

- INSTANCE:
  - a finite alphabet $\Sigma \rightarrow$ representing all accessed blocks,
  - a finite set of n jobs $J_i(C_i, D, S_i)$ with a common deadline D,
- QUESTION:
  - Is there a uniprocessor preemptive schedule meeting the overall deadline D for every job J_i?

$\Rightarrow$ the scheduling problem with cache memory is **NP-hard**.

Conclusion



Improving WCET $\Rightarrow$ make scheduling more complex.
Many questions:

- which task(s) model(s)?
- which parameters for the timing analysis (WCET, CRPD...)?
- ...

Thank you for you attention!