Bounding the Cache-Related Preemption Delay

Jan Reineke

October 15, 2015
Outline

1 Why Preemptive Scheduling?

2 CRPD Computation
   - Analysis of the Preempted Task
   - Analysis of the Preempting Task
   - Analysis of the Preempted and the Preempting Task
   - Simplifying or Eliminating the Problem
   - Policies other than LRU

3 Accounting for CRPD within Response-Time Analysis

4 Summary
Outline

1  Why Preemptive Scheduling?

2  CRPD Computation
   - Analysis of the Preempted Task
   - Analysis of the Preempting Task
   - Analysis of the Preempted and the Preempting Task
   - Simplifying or Eliminating the Problem
   - Policies other than LRU

3  Accounting for CRPD within Response-Time Analysis

4  Summary
Why use preemptive scheduling?

- Preemption often increases schedulability of task sets.
- Tasks with short deadlines are often unschedulable non-preemptively.

Example

Given: Two periodic tasks \( T_1 \) and \( T_2 \), with periods \( P_1 = 2, P_2 = 8 \), deadlines \( D_1 = P_1, D_2 = P_2 \), and execution times \( C_1 = 1, C_2 = 3 \).
Why use preemptive scheduling?

- Preemption often increases schedulability of task sets.
- Tasks with short deadlines are often unschedulable non-preemptively.

Example

Given: Two periodic tasks $T_1$ and $T_2$, with periods $P_1 = 2$, $P_2 = 8$, deadlines $D_1 = P_1$, $D_2 = P_2$, and execution times $C_1 = 1$, $C_2 = 3$. 
Preemption does not come for free!

- The preemining task “disturbs” the state of performance-enhancing features like caches and pipelines.
- Once the preempted task resumes its execution, the disturbance may cause additional cache misses.
- The additional execution time due to additional cache misses is known as the \textit{cache-related preemption delay}.

\begin{align*}
T_1 & \uparrow \\
T_2 & \uparrow \\
\text{\textcolor{gray}{\square}} = \text{CRPD} \\
\uparrow = \text{Task Activation}
\end{align*}
How to take preemption cost into account?

Where to account for preemption cost?

- Integrate into WCET Analysis: [Schneider, 2000]
  - Assume cache misses everywhere
  - Very pessimistic but easy to use with existing schedulability analyses

- WCET Analysis + CRPD Analysis: [Lee et al., 1996]
  - \( WCET_{\text{bound}} + n \cdot CRPD_{\text{bound}} \geq \) execution time of task with up to \( n \) preemptions
  - More precise but also requires new schedulability analyses taking into account the CRPD bounds
How to take preemption cost into account?

Where to account for preemption cost?

- Integrate into WCET Analysis: [Schneider, 2000]
  - Assume cache misses everywhere
  - Very pessimistic but easy to use with existing schedulability analyses

- WCET Analysis + CRPD Analysis: [Lee et al., 1996]
  - $WCET_{bound} + n \cdot CRPD_{bound} \geq$ execution time of task with up to $n$ preemptions
  - More precise but also requires new schedulability analyses taking into account the CRPD bounds

Focus of this talk: approaches to bound the CRPD
And a bit on: using these bounds within schedulability analyses
Outline

1 Why Preemptive Scheduling?

2 CRPD Computation
   - Analysis of the Preempted Task
   - Analysis of the Preempting Task
   - Analysis of the Preempted and the Preempting Task
   - Simplifying or Eliminating the Problem
   - Policies other than LRU

3 Accounting for CRPD within Response-Time Analysis

4 Summary
Preempted Task:
How many additional cache misses can a single preemption by any preemtping task cause in a given preempted task?
CRPD Analyses

- **Preempted Task:**
  How many additional cache misses can a single preemption by any preempting task cause in a given preempted task?

- **Preempting Task:**
  How many additional cache misses can a single preemption by a given preempting task cause in any preempted task?
Preempted Task:
How many additional cache misses can a single preemption by any preemtion task cause in a given preemption task?

Preempting Task:
How many additional cache misses can a single preemption by a given preemption task cause in any preemption task?

Preempted + Preempting Task
How many additional cache misses can a single preemption by a given preemption task cause in a given preemption task?
Outline

1 Why Preemptive Scheduling?

2 CRPD Computation
   • Analysis of the Preempted Task
   • Analysis of the Preempting Task
   • Analysis of the Preempted and the Preempting Task
   • Simpifying or Eliminating the Problem
   • Policies other than LRU

3 Accounting for CRPD within Response-Time Analysis

4 Summary
Analysis of the Preempted Task: Useful Cache Blocks (UCB)

Definition (Useful Cache Block, [Lee et al., 1996])

A memory block \( m \) at program point \( P \) is called a useful cache block, if

a) \( m \) may be cached at \( P \)

b) \( m \) may be reused at program point \( Q \) that may be reached from \( P \) with no eviction of \( m \) on this path.

\( \text{UCB} \supseteq \{A, B, C, D\} \)

\( \times = \text{hit} \)

\( \bigcirc = \text{miss} \)
Analysis of the Preempted Task: Useful Cache Blocks (UCB)

**Definition (Useful Cache Block, [Lee et al., 1996])**

A memory block $m$ at program point $P$ is called a useful cache block, if

a) $m$ may be cached at $P$

b) $m$ may be reused at program point $Q$ that may be reached from $P$ with no eviction of $m$ on this path.

\[ \text{UCB} = \{A, B, C\} \]

\[ \text{Cache State:} [A, B, C, D] \]

\[ X = \text{hit} \]
\[ O = \text{miss} \]
UCB Analysis

What may be cached?
What may be reused?
Forward May-Analysis!
Backward May-Analysis!

minimal distance \leq \text{associativity}?

Program point P

minimal distance
\leq \text{associativity}?
UCB Analysis

Combination of two LRU-may-analyses:

What may be cached?
Forward May-Analysis!

What may be reused?
Backward May-Analysis!

Minimal age

Minimal distance to reuse

\[ \text{Minimal age} + \text{Minimal distance to reuse} \leq \text{associativity} \]

\[ \implies \text{Memory block may be useful} \]
Improvement: Path Analysis

Some blocks are never useful at the same time:

\[ \leq \text{associativity} \]

\[ \geq y \]

\[ P \]

\[ \leq \text{associativity} \]

Literature:
[Tomiyama and Dutt, 2000, Negi et al., 2003, Staschulat and Ernst, 2007]
Improvement:
Avoid Accumulating Overestimations

Schedulability analyses rely on:

\[ WCET_{bound} + n \cdot CRPD_{bound} \geq \text{exec. time with up to } n \text{ preemptions} \]

\[ \overset{\uparrow}{\text{BRT} \cdot |UCB|} \]

# of possible preemptions

Yet, we usually have:

\[ WCET_{bound} \geq \text{execution time without preemptions} \]
\[ CRPD_{bound} \geq \text{additional execution time due to one preemption} \]

\[ \Rightarrow \text{Overestimation in both analyses adds up: Some cache misses are counted twice!} \]
Bounding the CRPD using UCBs for Fully-Associative Caches

- CRPD bound at program point $P$:

$$\text{CRPD}^{\text{LRU}}_{\text{UCB}}(P) = \text{BRT} \cdot \min(|\text{UCB}(P)|, k),$$

where $k =$ associativity and BRT $=$ Block Reload Time.

- CRPD bound independent of program point:

$$\text{CRPD}^{\text{LRU}}_{\text{UCB}} = \max_P \text{CRPD}^{\text{LRU}}_{\text{UCB}}(P)$$

Slightly more complicated for set-associative caches: sum up bounds of all cache sets.
Outline

1. Why Preemptive Scheduling?

2. CRPD Computation
   - Analysis of the Preempted Task
   - Analysis of the Preempting Task
   - Analysis of the Preempted and the Preempting Task
   - Simplifying or Eliminating the Problem
   - Policies other than LRU

3. Accounting for CRPD within Response-Time Analysis

4. Summary
Analysis of the Preempting Task: Evicting Cache Blocks

Definition (Evicting Cache Blocks (ECB), [Tomiyama and Dutt, 2000])

A memory block of the preempting task is called an *evicting cache block*, if it may be accessed during the execution of the preempting task.

\[ \text{Cache State: } [A, B, C, D] \quad \xrightarrow{X} \quad [X, Y, Z, D] \quad \xrightarrow{Y} \quad [X, Y, Z, D] \quad \xrightarrow{Z} \quad [X, Y, Z, D] \]

\[ \text{Cache State: } [A, B, C, D] \quad \xrightarrow{X} \quad [X, Y, Z, D] \quad \xrightarrow{Y} \quad [X, Y, Z, D] \quad \xrightarrow{Z} \quad [X, Y, Z, D] \]

\[ A \quad B \quad C \quad B \quad A \quad C \]

\[ \times = \text{hit} \]
\[ \circ = \text{miss} \]

\[ = \text{additional miss due to preemption (CRPD)} \]
Analysis of the Preempting Task: Evicting Cache Blocks

Definition (Evicting Cache Blocks (ECB), [Tomiyama and Dutt, 2000])

A memory block of the preemption task is called an *evicting cache block*, if it may be accessed during the execution of the preempting task.

![Diagram showing cache state transitions](image)

- **Cache State**: \([A, B, C, D]\)
- **Cache State**: \([X, Y, Z, D]\)

- \(\times\) = hit
- \(\bigcirc\) = miss
- \(\bullet\) = additional miss due to preemption (CRPD)

\[
\text{CRPD}_{\text{LRU}}^{\text{ECB}} \stackrel{?}{=} \text{BRT} \cdot \min(|\text{ECB}|, k)
\]

- \(k\) = associativity
- \(\text{BRT} = \text{Block Reload Time}\)
CRPD Computation for LRU using ECBs: Pitfall

\[ [b, a, 9, 8] \xrightarrow{8} [8, b, a, 9] \xrightarrow{9} [9, 8, b, a] \xrightarrow{a} [a, 9, 8, b] \xrightarrow{b} [b, a, 9, 8] \quad 0 \text{ misses} \]
CRPD Computation for LRU using ECBs: Pitfall

ECBs = \{e\}

\[ [b, a, 9, 8] \xrightarrow{8} [8, b, a, 9] \xrightarrow{9} [9, 8, b, a] \xrightarrow{a} [a, 9, 8, b] \xrightarrow{b} [b, a, 9, 8] \]

0 misses

\[ [e, b, a, 9] \xrightarrow{8^*} [8, e, b, a] \xrightarrow{9^*} [9, 8, e, b] \xrightarrow{a^*} [a, 9, 8, e] \xrightarrow{b^*} [b, a, 9, 8] \]

4 misses

- \(|UCB| = 4\)
- \(|ECB| = 1\)
- \(k = \text{associativity} = 4\)
- \(\text{number of additional misses} = 4\)
CRPD Computation for LRU using ECB: Sound but Imprecise

- ECB analysis only to determine whether the set is used at all by the preemtng task or not:

\[
\text{CRPD}_{\text{ECB}}^{\text{LRU}} = \begin{cases} 
0 & \text{if } \text{ECB} = \emptyset \\
\text{BRT} \cdot k & \text{otherwise}
\end{cases}
\]

- Cannot do better than that without knowledge of preempted task.
Outline

1 Why Preemptive Scheduling?

2 CRPD Computation
   - Analysis of the Preempted Task
   - Analysis of the Preempting Task
   - Analysis of the Preempted and the Preempting Task
   - Simplifying or Eliminating the Problem
   - Policies other than LRU

3 Accounting for CRPD within Response-Time Analysis

4 Summary
Analysis of Preempted and Preempting Task: “Shallow” Combination

Take the minimum of the $UCB$- and $ECB$-based estimations.

- CRPD bound for entire program:
  \[
  CRPD_{LRU}^{UCB+ECB} = \min(CRPD_{LRU}^{ECB}, CRPD_{LRU}^{UCB})
  \]

Becomes slightly more complicated for set-associative caches:
For a program point: sum of point-wise minima of all cache sets.

Literature:

- For direct-mapped caches: [Negi et al., 2003]
- For set-associative caches:
  [Tan and Mooney, 2004, Burguière et al., 2009]
Analysis of Preempted and Preempting Task: “Deeper” Combination

Without preemption:
\[ [a, 9, 8, 7] \xrightarrow{8} [8, a, 9, 7] \xrightarrow{9} [9, 8, a, 7] \xrightarrow{a} [a, 9, 8, 7] \]

With preemption:
\[ [e, a, 9, 8] \xrightarrow{8} [8, e, b, a] \xrightarrow{9} [9, 8, e, b] \xrightarrow{a} [a, 9, 8, e] \]

Some of the UCBs are guaranteed to remain useful under preemption!
Analysis of Preempted and Preempting Task: “Deeper” Combination

Without preemption:

$$[a, 9, 8, 7] \xrightarrow{8} [8, a, 9, 7] \xrightarrow{9} [9, 8, a, 7] \xrightarrow{a} [a, 9, 8, 7]$$

With preemption:

$$[e, a, 9, 8] \xrightarrow{8} [8, e, b, a] \xrightarrow{9} [9, 8, e, b] \xrightarrow{a} [a, 9, 8, e]$$

Some of the UCBs are guaranteed to remain useful under preemption!

- \( \text{CRPD}_{\text{UCB} \& \text{ECB}} = \min(\text{CRPD}_{\text{UCB}}, \text{CRPD}_{\text{ECB}}) = \min(3, 4) = 3 \)
- Yet: actual number of additional misses: 0
Analysis of Preempted and Preempting Task: “Deeper” Combination

Without preemption:
\[ [a, 9, 8, 7] \xrightarrow{8} [8, a, 9, 7] \xrightarrow{9} [9, 8, a, 7] \xrightarrow{a} [a, 9, 8, 7] \]

With preemption:
\[ [e, a, 9, 8] \xrightarrow{8} [8, e, b, a] \xrightarrow{9} [9, 8, e, b] \xrightarrow{a} [a, 9, 8, e] \]

Some of the UCBs are guaranteed to remain useful under preemption!

- CRPD_{UCB\&ECB} = \min(CRPD_{UCB}, CRPD_{ECB}) = \min(3, 4) = 3
- Yet: actual number of additional misses: 0

Why?
Analysis of Preempted and Preempting Task: “Deeper” Combination

- Without preemption:
  - \([a, 9, 8, 7] \xrightarrow{8} [8, a, 9, 7] \xrightarrow{9} [9, 8, a, 7] \xrightarrow{a} [a, 9, 8, 7]\)

- With preemption:
  - \([e, a, 9, 8] \xrightarrow{8} [8, e, b, a] \xrightarrow{9} [9, 8, e, b] \xrightarrow{a} [a, 9, 8, e]\)

Some of the UCBs are guaranteed to remain useful under preemption!

- CRPD_{UCB\&ECB} = \min(CRPD_{UCB}, CRPD_{ECB}) = \min(3, 4) = 3
- Yet: actual number of additional misses: 0

Why?

- Minimal number of ECBs to evict a UCB is 2, but \(|ECB| = 1\)
- A single ECB is not sufficient to evict any of the UCBs.
Combining UCB and ECB: Notion of Resilience

Determining the maximal number of ECBs, such that no additional cache miss may occur:

\[ m \in \text{UCB} \]

- \[ m \]
- \[ a_1 \] \[ [m, -, -, -, -] \]
- \[ a_2 \] \[ [m, a_1, a_2, a_1, m, -, -] \]
- \[ a_3 \] \[ [a_3, a_2, a_1, m, -, -] \]
- \[ m \] \[ [m, a_3, a_2, a_1, -] \]
Combining UCB and ECB: Notion of Resilience

Determining the maximal number of ECBs, such that no additional cache miss may occur:

\[ m \in UCB \]

\[ m \text{ is 4-resilient} \]

\[ \begin{array}{c}
  m \\
  a_1 \\
  a_2 \\
  a_3 \\
  m_3, a_2, a_1, m, -, -, -, - \\
  m, a_3, a_2, a_1, -, -, -, -
\end{array} \]
Resilience Analysis

Definition (Resilience)

A memory block $m$ is called $l$-resilient at program point $P$, if all possible next accesses to $m$

- that would be hits without preemption,
- would still be hits in case of a preemption at $P$ with $l$ accesses.
Resilience Analysis

Definition (Resilience)

A memory block \( m \) is called \( l \)-resilient at program point \( P \), if all possible next accesses to \( m \)
- that would be hits without preemption,
- would still be hits in case of a preemption at \( P \) with \( l \) accesses.

- No UCB is \( k \)-resilient, i.e., no UCB remains useful after a preemption with \( k \) (= associativity) many ECBs.
- Each \( (l+1) \)-resilient UCB is also \( l \)-resilient.
- Each UCB is at least 0-resilient.
Resilience Analysis

Definition (Resilience)

A memory block $m$ is called $l$-resilient at program point $P$, if all possible next accesses to $m$

- that would be hits without preemption,
- would still be hits in case of a preemption at $P$ with $l$ accesses.

$m \in UCB$

$m$ is 4-resilient

$ECB = \{e_1, e_2, e_3, e_4\}$
Resilience Analysis

Definition (Resilience)

A memory block $m$ is called $l$-resilient at program point $P$, if all possible next accesses to $m$
- that would be hits without preemption,
- would still be hits in case of a preemption at $P$ with $l$ accesses.

In general: if $|ECB| \leq l$ then the UCB is not evicted
Resilience Analysis

Definition (Resilience)

A memory block $m$ is called $l$-resilient at program point $P$, if all possible next accesses to $m$

- that would be hits without preemption,
- would still be hits in case of a preemption at $P$ with $l$ accesses.

0-resil. 3-resil. 0-resil.

no access to $m$ 3-resil. 0-resil.

$m$ is not useful 3-resil. 3-resil.

$m$

$m$
Bounding the CRPD using Resilience

CRPD (Combining UCB and ECB by using Resilience)

\[ \text{blocks contributing to CRPD} \]

\[ \left\{ \text{useful UCB} \big| \left\{ m | m \text{ is ECB}-\text{resilient} \right\} \right\} \]

\[ \text{remain useful} \]
Bounding the CRPD using Resilience

\[ CRPD \leq BRT \times \left| \left( UCB \setminus \{ m \mid m \text{ is ECB}-resilient \} \right) \right| \]

- \( UCB \): blocks contributing to CRPD
- \( \{ m \mid m \text{ is ECB}-resilient \} \): useful blocks
- \( \left| \cdot \right| \): count of elements
- \( \setminus \): set difference
- \( \leq \): less than or equal to
- \( BRT \): a threshold value

This equation bounds the CRPD by considering the resilience of ECB blocks and their usefulness.
Bounding the CRPD using Resilience: Example

ECBs = \{e\}

\[\begin{align*}
[c, b, a, x] &\xrightarrow{a}[a, c, b, x] &\xrightarrow{b}[b, a, c, x] &\xrightarrow{c}[c, b, a, x] \\
[e, c, b, a] &\xrightarrow{a}[a, e, c, b] &\xrightarrow{b}[b, a, e, c] &\xrightarrow{c}[c, b, a, e]
\end{align*}\]

no misses

no misses
Bounding the CRPD using Resilience: Example

- $|\text{ECB}| = 1$
- $a$, $b$, and $c$ are 1-resilient
- $\text{CRPD}_{\text{UCB} \& \text{ECB}}^{\text{res}} = BRT \times |\text{UCB} \setminus \{m \mid m \text{ is } |\text{ECB}|\text{-resilient}\}| = 0$

Graph:

- $0x0a \rightarrow [c, b, a, x] \rightarrow [a, c, b, x] \rightarrow [b, a, c, x] \rightarrow [c, b, a, x]$
- $\text{no misses}$

- $0x0b \rightarrow [e, c, b, a] \rightarrow [a, e, c, b] \rightarrow [b, a, e, c] \rightarrow [c, b, a, e]$
- $\text{no misses}$

ECBs $= \{e\}$
Bounding the CRPD using Resilience: Example

- $|ECB| = 1$
- $a$, $b$, and $c$ are 1-resilient
- $CRPD_{res}^{UCB\&ECB} = BRT \times |UCB\ \backslash \{m \mid m \text{ is } |ECB|-resilient\}| = 0$

Instead of: $CRPD_{UCB\&ECB} = \min(CRPD_{UCB}, CRPD_{ECB}) = 3 \times BRT$
Bounding the CRPD using Resilience: Example

- $|\text{ECB}| = 1$
- $a$, $b$, and $c$ are 1-resilient
  - $\text{CRPD}^{\text{res}}_{\text{UCB} \& \text{ECB}} = \text{BRT} \times |\text{UCB} \setminus \{m \mid m \text{ is } |\text{ECB}|\text{-resilient}\}| = 0$
- Instead of: $\text{CRPD}_{\text{UCB} \& \text{ECB}} = \min(\text{CRPD}_{\text{UCB}}, \text{CRPD}_{\text{ECB}}) = 3 \times \text{BRT}$
Outline

1. Why Preemptive Scheduling?

2. CRPD Computation
   - Analysis of the Preempted Task
   - Analysis of the Preempting Task
   - Analysis of the Preempted and the Preempting Task
   - Simplifying or Eliminating the Problem
   - Policies other than LRU

3. Accounting for CRPD within Response-Time Analysis

4. Summary
Deferred Preemption

- Restrict preemptions to a set of predefined *preemption points*.
- Introduces new problem: blocking time, i.e., time until next preemption point is reached.
  Intervals between preemption points ≡ critical sections.

\[
T_1 \quad \uparrow \quad \text{BT} \quad \uparrow \quad \text{BT}
\]

\[
T_2 \quad \uparrow \quad \text{Context Switch Costs} \quad \uparrow \quad \text{Task Activation} \quad \ast \quad \text{Preemption Point}
\]

Where to place preemptions points, s.t.
- CRPD is minimized, and
- *Maximum Blocking Time* is minimized?

Analysis to determine maximum blocking time for given set of preemption points: [Lee et al., 1998, Altmeyer et al., 2009]
Cache Partitioning

Additional cache misses are due to interference on the cache.

→ Cache Partitioning eliminates this interference.

- Change layout of instructions and data such that tasks map to disjoint cache sets
- Particularly difficult for large arrays

### Hardware-based Cache Partitioning [Kirk and Strosnider, 1990, Chiou, 1999]:
- Partition cache by cache sets and/or cache ways
- Increases hardware cost
- Renewed interest in multi-cores with shared caches
Outline

1. Why Preemptive Scheduling?

2. CRPD Computation
   - Analysis of the Preempted Task
   - Analysis of the Preempting Task
   - Analysis of the Preempted and the Preempting Task
   - Simplifying or Eliminating the Problem
   - Policies other than LRU

3. Accounting for CRPD within Response-Time Analysis

4. Summary
Do existing approaches work for FIFO, PLRU, etc.?

Plain answer: No!
Do existing approaches work for FIFO, PLRU, etc.?

Plain answer: No!

Counterexample for FIFO [Burguierre et al., 2009]:

\[
\text{ECBs} = \{ x \} \quad \begin{align*}
[b, a] \xrightarrow{a} [b, a] & \xrightarrow{e^*} [e, b] \xrightarrow{b} [e, b] & \xrightarrow{c^*} [c, e] \xrightarrow{e} [c, e] & \quad 2 \text{ misses} \\
[x, b] \xrightarrow{a^*} [a, x] & \xrightarrow{e^*} [e, a] \xrightarrow{b^*} [b, e] & \xrightarrow{c^*} [c, b] \xrightarrow{e^*} [e, c] & \quad 5 \text{ misses}
\end{align*}
\]

- \(|\text{UCB}(s)| = 2\)
- \(|\text{ECB}(s)| = 1\)
- Associativity \(k = 2\)
- But: number of additional misses \(= 3\)

Same result for PLRU.
Idea [Burguière et al., 2009]: Use Relative Competitiveness Results

Some relative competitiveness results:

- **PLRU**($n$) is $(1, 0)$-miss-competitive relative to LRU$(1 + \log_2 n)$.
- **FIFO**($n$) is $(\frac{n}{n-r+1}, r)$-miss-competitive relative to LRU$(r)$.

$\implies$ Performing WCET and CRPD analyses assuming LRU$(1 + \log_2 n)$ replacement should give correct bounds for PLRU$(n)$.

Can we also make use of non-$(1, 0)$-competitiveness?
Applying Relative Competitiveness: A sequence of memory accesses

- **Notation:**
  - $m =$ number of misses
  - $\bar{m} =$ number of misses in the case of preemption

- $m_{pre} = 4$
- $m_{post} = 2$
- $\bar{m}_{pre} = m_{pre} = 4$
- $\bar{m}_{post} = m_{post} + m_{CRPD} = 5$
Applying Relative Competitiveness: A sequence of memory accesses

- **Notation:**
  - $m =$ number of misses
  - $\bar{m} =$ number of misses in the case of preemption

- Assume $P(t)$ is $(k, c)$-miss-competitive rel. to LRU(s). Then:
  \[
  \bar{m}^{P(t)} = \bar{m}_{pre}^{P(t)} + \bar{m}_{post}^{P(t)}
  \]
Applying Relative Competitiveness: A sequence of memory accesses

- **Notation:**
  - $m = \text{number of misses}$
  - $m = \text{number of misses in the case of preemption}$

![Diagram showing memory accesses]

- $m_{\text{pre}} = 4$
- $m_{\text{post}} = 2$
- $m_{\text{pre}} = m_{\text{pre}} = 4$
- $m_{\text{post}} = m_{\text{post}} + m_{\text{CRPD}} = 5$

Assume $P(t)$ is $(k, c)$-miss-competitive rel. to $\text{LRU}(s)$. Then:

$$m_{P(t)} = m_{p_{\text{pre}}} + m_{P(t)}$$

$$\leq [k \cdot m_{\text{pre}} + c] + [k \cdot (m_{\text{post}} + m_{\text{LRU}(s)} + m_{\text{CRPD}}) + c]$$
Applying Relative Competitiveness: A sequence of memory accesses

- **Notation:**
  - $m = \text{number of misses}$
  - $\overline{m} = \text{number of misses in the case of preemption}$

  $m_{\text{pre}} = 4$ \hspace{2cm} $m_{\text{post}} = 2$

  $\overline{m}_{\text{pre}} = m_{\text{pre}} = 4$ \hspace{2cm} $\overline{m}_{\text{post}} = m_{\text{post}} + m_{\text{CRPD}} = 5$

- **Assume $P(t)$ is $(k, c)$-miss-competitive rel. to LRU$(s)$. Then:**

  \[
  \overline{m}^{P(t)} = \overline{m}_{\text{pre}}^{P(t)} + \overline{m}_{\text{post}}^{P(t)} \\
  \leq [k \cdot m_{\text{pre}}^{LRU(s)} + c] + [k \cdot (m_{\text{post}}^{LRU(s)} + m_{\text{CRPD}}^{LRU(s)}) + c] \\
  = [k \cdot m^{LRU(s)} + c] + [k \cdot m_{\text{CRPD}}^{LRU(s)} + c]
  \]
Applying Relative Competitiveness

Assume $P(t)$ is $(k, c)$-miss-competitive rel. to LRU$(s)$. Then:

$$m^{P(t)} \leq [k \cdot m^{LRU(s)} + c] + [k \cdot m^{LRU(s)}_{CRPD} + c]$$

- In WCET analysis:
  Take into account $k \cdot m^{LRU(s)} + c$ misses

- In CRPD analysis:
  Take into account $k \cdot m^{LRU(s)}_{CRPD} + c$ misses
Outline

1. Why Preemptive Scheduling?

2. CRPD Computation
   - Analysis of the Preempted Task
   - Analysis of the Preempting Task
   - Analysis of the Preempted and the Preempting Task
   - Simplifying or Eliminating the Problem
   - Policies other than LRU

3. Accounting for CRPD within Response-Time Analysis

4. Summary
Recap: Rate-Monotonic Schedulability Condition

The time-demand function $W_i(t)$ of the task $\tau_i$ is defined as follows:

$$W_i(t) = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j.$$  

Theorem

A system $\mathcal{T}$ of periodic, independent, preemptable tasks is schedulable on one processor by algorithm A if it holds that:

$$\forall \tau_i \in \mathcal{T} \exists t \text{ with } 0 < t \leq D_i \text{ and } W_i(t) \leq t$$

This condition is also necessary for synchronous, periodic task sets and also sporadic task sets.

Note that this holds for implicit-deadline and constrained-deadline task sets.
Recap: Response-Time Analysis for RM

Want to determine whether there is a $t$ such that $W_i(t) \leq t \leq D_i$.

Compute smallest such $t$ (if there is any) by fixed-point iteration:

$$n := 0$$
$$R_i^0 := C_i$$

**do**

$$n := n + 1$$

$$R_i^n := C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_j^{n-1}}{T_j} \right\rceil C_j$$

**while** $R_i^n < D_i$ and $R_i^{n-1} < R_i^n$

A set of tasks is schedulable if $R_i^n < D_i$ for every task $i$. 
How to incorporate CRPD?

Introduce additional term $\gamma_{i,j}$ in the time-demand function:

$$W_i(t) = C_i + \sum_{j=1}^{i-1} \left\lfloor \frac{t}{T_j} \right\rfloor (C_j + \gamma_{i,j}).$$

Fixed-point iteration can be appropriately adapted.
How to incorporate CRPD?

Introduce additional term $\gamma_{i,j}$ in the time-demand function:

$$W_i(t) = C_i + \sum_{j=1}^{i-1} \left\lfloor \frac{t}{T_j} \right\rfloor (C_j + \gamma_{i,j}) .$$

Fixed-point iteration can be appropriately adapted.

What is the meaning of $\gamma_{i,j}$?

Which value should it take?
Interpretation of $\gamma_{i,j}$

Without nested preemptions:

$$\gamma_{i,j} = \text{CRPD due to one preemption of task } i \text{ by task } j.$$ 

But what if there are nested preemptions?
First Interpretation of $\gamma_{i,j}$: “Effect of preemption task”

Under this interpretation, $\gamma_{i,j}$ needs to bound the effect of $j$’s execution on all tasks of lower priority up to task $i$. 
First Interpretation of $\gamma_{i,j}$:
“Effect of preempting task”

For *direct-mapped caches* there are three such approaches:

1. “ECB-Only”: $\gamma_{i,j} = BRT \cdot |ECB_j|$
2. “UCB-Union”: $\gamma_{i,j} = BRT \cdot |\bigcup_{k \in \text{aff}(i,j)} UCB_k|$, where $\text{aff}(i,j) = \text{hep}(i) \cap \text{lp}(j)$.
3. Combination of the above: $\gamma_{i,j} = BRT \cdot |\bigcup_{k \in \text{aff}(i,j)} UCB_k \cap ECB_j|$
Second Interpretation of $\gamma_{i,j}$: “Effect of immediately preempted task”

Under this interpretation, $\gamma_{i,j}$ needs to bound the effect of $j$’s execution and that of the execution of task’s preemption $j$ itself on the task that $j$ immediately preempted.
Second Interpretation of $\gamma_{i,j}$: “Effect of immediately preemted task”

For *direct-mapped caches* there are three such approaches:

1. “UCB-Only”: $\gamma_{i,j} = BRT \cdot \max_{k \in \text{aff}(i,j)} |UCB_k|$

2. “ECB-Union”: $\gamma_{i,j} = BRT \cdot \left| \bigcup_{h \in hp(j) \cup \{j\}} ECB_h \right|$. 

3. Combination of the above:

$$\gamma_{i,j} = BRT \cdot \max_{k \in \text{aff}(i,j)} \left| UCB_k \cap \bigcup_{h \in hp(j) \cup \{j\}} ECB_h \right|.$$
Some Open Problems

- Soundly and precisely handle set-associative caches in response-time analysis and use *resilience* information.
- Empirical analysis of sources of imprecision: is overestimation due to characterization of preempting tasks, of preempted task, or other imprecisions?
- What is a *sound penalty* for a preemption-induced cache reload?
  → main memory latency is *not*, even for simple in-order pipelines
  → on the other hand, often part of the latency can be hidden
Outline

1. Why Preemptive Scheduling?

2. CRPD Computation
   - Analysis of the Preempted Task
   - Analysis of the Preempting Task
   - Analysis of the Preempted and the Preempting Task
   - Simplifying or Eliminating the Problem
   - Policies other than LRU

3. Accounting for CRPD within Response-Time Analysis

4. Summary
Summary

- Preemptive Scheduling desirable, but not for free:
  → Need to bound CRPD
- For LRU, the CRPD can be bounded by analyzing
  ▶ the preempted task: UCB analysis
  ▶ the preempting task: ECB analysis
    ★ Sound approach rather imprecise
    ★ Need to couple more tightly with analysis of preempted task
  ▶ both, the preempted and the preempting task
    ★ “Shallow” combination
    ★ “Deeper” combination: Resilience analysis
- Approaches do not carry over to FIFO, PLRU, etc. immediately
  ▶ First approach: relative competitiveness
- Can be accounted for in response-time analysis in different ways
  ▶ Has so far only been investigated in detail for direct-mapped caches
In *Workshop on Software Technologies for Future Dependable Distributed Systems*.

In *Proceedings of 9th International Workshop on Worst-Case Execution Time (WCET) Analysis*.

PhD thesis.
Supervisor-Arvind, and Supervisor-Rudolph, Larry.

Smart (strategic memory allocation for real-time) cache design using the mips r3000.


Analysis of cache-related preemption delay in fixed-priority preemptive scheduling.


Limited preemptible scheduling to embrace cache memory in real-time systems.


Compiler support for software-based cache partitioning. 
*SIGPLAN Not.*, 30(11):125–133.

Accurate estimation of cache-related preemption delay. 
In *Proceedings of the 1st ACM international conference on Hardware/software codesign and system synthesis (CODES+ISSS’03)*, pages 201–206, New York, NY, USA. ACM.

Schneider, J. (2000). 
Cache and pipeline sensitive fixed priority scheduling for preemptive real-time systems. 

Scalable precision cache analysis for real-time software. 

Integrated intra- and inter-task cache analysis for preemptive multi-tasking real-time systems.

