A new Notion of Useful Cache Block to Improve the Bounds of Cache-Related Preemption Delay

Sebastian Altmeyer, Claire Burguière

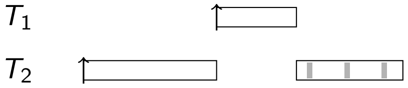
Saarland University

ECRTS 2009, Dublin

Cache-Related Preemption Delay (CRPD)

in case of preemption:
preempting task might evict cache-content of preempted task

CRPD $\cong$ cost of additional misses due to preemption



▨ = CRPD
$\uparrow$ = Task Activation

Useful Cache Blocks

Lee et al. 98:
A useful cache block (UCB) is a memory block that

a) may be cached at P, and
b) may be reused on at least one control flow path starting at program point P.

$\rightarrow$ #UCBs at P determines bound for CRPD at P
$\rightarrow$ global CRPD bound given by point with highest number of UCBs

UCB very common concept nowadays:

Originally for direct-mapped instruction/data caches, has been:

- extended to set-associative caches (Lee et al. 96[2], Staschulat and Ernst 07[7])
- improved by using different cache representation (Negi et al. 03[4],Staschulat and Ernst 04[5], 07[7])
- improved by analyzing preempting task (Negi et al. 03[4],Staschulat and Ernst 07[7])

and used within schedulability analysis in [1, 3, 6, 7, 8, 9] ....

always relying on the original notion of UCBs by Lee et al.!

UCBs in Schedulability Analysis

Schedulability analysis always computes:

$$WCET + \underbrace{n}_{\#\ \text{possible preemptions}} \times \underbrace{CRPD}_{\#\ UCB_{max} \times CRT}$$

However,
WCET analysis and CRPD analysis are treated separately

Hence,
overapprox. WCET bound and overapprox. CRPD bound accumulates

WCET Analysis

- overapproximates actual WCET
- overapproximates # cache-misses
- underapproximates # cache-hits

$\Rightarrow$ uses **underapproximation** of cache-content (**must-cache**)

$\Rightarrow$ only predicts cache-hit, if accessed block $\in$ **must-cache**

UCB Analysis

- overapproximates # additional misses
- additional miss $\subseteq$ hit during task execution
- overapproximates # cache-hits during execution

$\Rightarrow$ uses **overapproximation** of cache-content (**may-cache**)

$\Rightarrow$ predicts additional cache-miss,
if accessed block $\in$ **may-cache**

WCET Analysis vs. UCB Analysis

WCET Analysis:

- uses **underapproximation** of cache-content (**must-cache**)
- only predicts cache-hit, if accessed block $\in$ **must-cache**

UCB Analysis:

- uses **overapproximation** of cache-content (**may-cache**)
- predicts additional cache-miss, if accessed block $\in$ **may-cache**

Hence, some cache-misses counted twice!
(access m with $m \in$ (May-Cache $\setminus$ Must-Cache))

A new Notion of UCBs

A useful cache block (UCB) is a memory block that

a) may be reused on at least one control flow path starting at program point P, and
b) may be cached at P.

CRPD $\cong$ cost of additional misses due to preemption

A new Notion of UCBs: DC-UCBs

A **definitely-cached** *UCB (***DC-***UCB)* is a memory block that

a) may be reused on at least one control flow path starting at program point P, and
b) **must** be cached at P **and along the path to its reuse**.

CRPD $\cong$ cost of additional misses due to preemption
not taken into account by WCET analysis

Soundness of DC-UCB Approach

- DC-UCB $\subseteq$ UCB
- #DC-UCB might underapproximate actual CRPD

So, how to ensure that

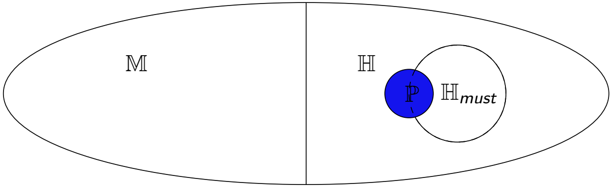
$$\text{WCET} + n \times (\#\text{DC-UCB}_{max} \times \text{CRT})$$

still bounds execution time under preemptions?

To prove:
(misses in WCET + DC-UCBs) $\supseteq$ (actual misses during execution)

Proof

$\mathbb{A}$:	all memory accesses during execution
$\mathbb{H}$:	all memory accesses leading to a hit
$\mathbb{M}$:	all memory accesses leading to a miss
$\mathbb{H}_{must}$:	all memory accesses classified as hits by must-cache ($\mathbb{H}_{must} \subseteq \mathbb{H}$)
$\mathbb{M}_{must}$:	all memory accesses classified as misses by must-cache ($\mathbb{M}_{must} = \mathbb{A} \setminus \mathbb{H}_{must}$)
$\mathbb{P}$:	additional misses due to preemption ($\mathbb{P} \subset \mathbb{H}$)



Proof

$\mathbb{A}$: all memory accesses during execution

$\mathbb{H}$: all memory accesses leading to a hit

$\mathbb{M}$: all memory accesses leading to a miss

$\mathbb{H}_{must}$: all memory accesses classified as hits by must-cache ($\mathbb{H}_{must} \subseteq \mathbb{H}$)

$\mathbb{M}_{must}$: all memory accesses classified as misses by must-cache ($\mathbb{M}_{must} = \mathbb{A} \setminus \mathbb{H}_{must}$)

$\mathbb{P}$: additional misses due to preemption ($\mathbb{P} \subset \mathbb{H}$)

(misses in WCET + DC-UCBs)

$$\mathbb{M}_{must} \cup (\mathbb{P} \cap \mathbb{H}_{must})$$

$$\supseteq$$

$$\mathbb{M} \cup \mathbb{P}$$

(actual misses during execution)

Proof

$\mathbb{A}$:	all memory accesses during execution
$\mathbb{H}$:	all memory accesses leading to a hit
$\mathbb{M}$:	all memory accesses leading to a miss
$\mathbb{H}_{must}$:	all memory accesses classified as hits by must-cache ($\mathbb{H}_{must} \subseteq \mathbb{H}$)
$\mathbb{M}_{must}$:	all memory accesses classified as misses by must-cache ($\mathbb{M}_{must} = A \setminus \mathbb{H}_{must}$)
$\mathbb{P}$:	additional misses due to preemption ($\mathbb{P} \subset \mathbb{H}$)

$$
\begin{aligned}
&\text{(misses in WCET + DC-UCBs)}\\
&\qquad \supseteq \mathbb{M}_{must} \cup (\mathbb{P} \cap \mathbb{H}_{must})\\
&\qquad = \mathbb{M}_{must} \cup (\mathbb{P} \cap (\mathbb{A} \setminus \mathbb{M}_{must}))\\
&\qquad = (\mathbb{M}_{must} \cup \mathbb{P}) \cap (\mathbb{M}_{must} \cup (\mathbb{A} \setminus \mathbb{M}_{must}))\\
&\qquad = (\mathbb{M}_{must} \cup \mathbb{P}) \cap (\mathbb{A})\\
&\qquad = \mathbb{M}_{must} \cup \mathbb{P}\\
&\qquad \supseteq \mathbb{M} \cup \mathbb{P}\\
&\text{(actual misses during execution)} \qquad \square
\end{aligned}
$$

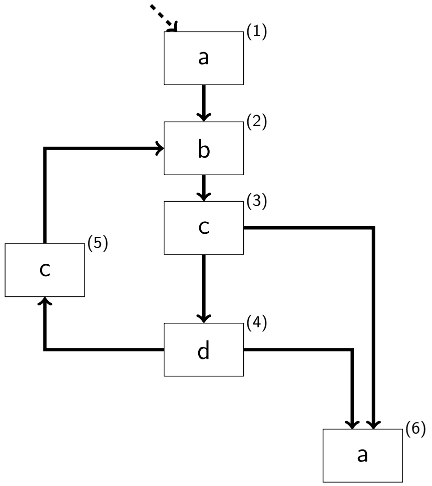
DC-UCB Analysis

implemented as backward program analysis
derives a set of DC-UCBs at each program point P

1. DC-UCBs(P) $= \{m\}$,
   if m accessed at P and $m \in \mathit{Must\_Cache}(P)$
2. forwards information to all predecessors of P
3. removes all memory blocks $m' \notin \mathit{Must\_Cache}(P)$
4. until fix-point, goto 2

for each P

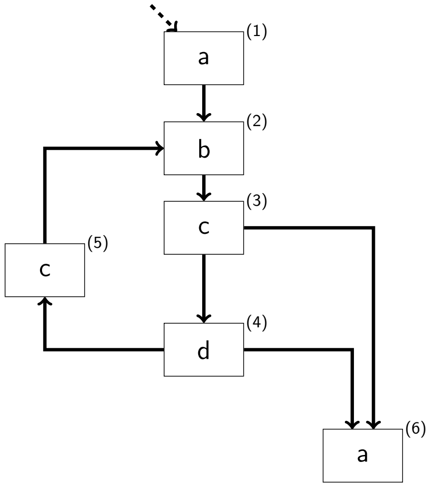
DC-UCB Analysis - Example



1. DC-UCB(P) := $\{m\}$
   if a) m accessed at P
   and b) $m \in Must\_Cache(P)$
2. forwards information to predecessors
3. DC-UCB(P) :=
   (DC-UCB(P) $\setminus$ $Must\_Cache(P)$)
4. unless fixpoint reached, goto 2

I	Must-Cache
1	$(\_, \_, \_, \_)$
2	$(a, \_, \_, \_)$
3	$(a, b, \_, \_)$
4	$(a, b, c, \_)$
5	(a, b, c, d)
6	$(a, \_, c, \_)$

DC-UCB Analysis - Example



➜ 1 DC-UCB(P) := $\{m\}$
 if a) m accessed at P
 and b) $m \in Must\_Cache(P)$

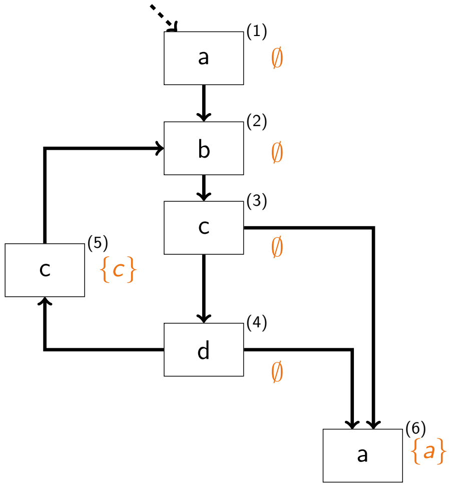
2 forwards information to predecessors

3 DC-UCB(P) :=
 (DC-UCB(P) $\setminus$ $Must\_Cache(P)$)

4 unless fixpoint reached, goto 2

I	Must-Cache
1	$(\_,\_,\_,\_)$
2	$(a,\_,\_,\_)$
3	$(a,b,\_,\_)$
4	$(a,b,c,\_)$
5	(a,b,c,d)
6	$(a,\_,c,\_)$

DC-UCB Analysis - Example



➔ 1 DC-UCB(P) := $\{m\}$
   if a) m accessed at P
   and b) $m \in Must\_Cache(P)$

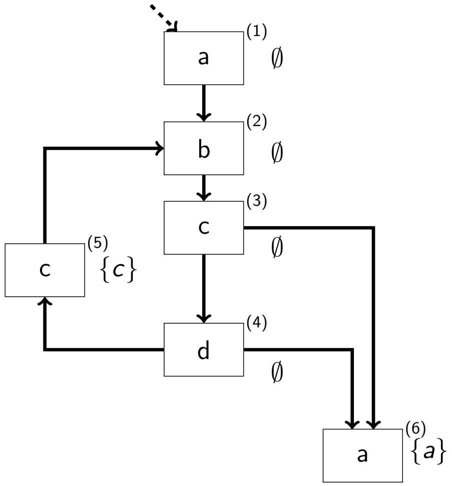
2 forwards information to predecessors

3 DC-UCB(P) :=
   (DC-UCB(P) $\setminus$ $Must\_Cache(P)$)

4 unless fixpoint reached, goto 2

I	Must-Cache
1	$(\_, \_, \_, \_)$
2	$(a, \_, \_, \_)$
3	$(a, b, \_, \_)$
4	$(a, b, c, \_)$
5	(a, b, c, d)
6	$(a, \_, c, \_)$

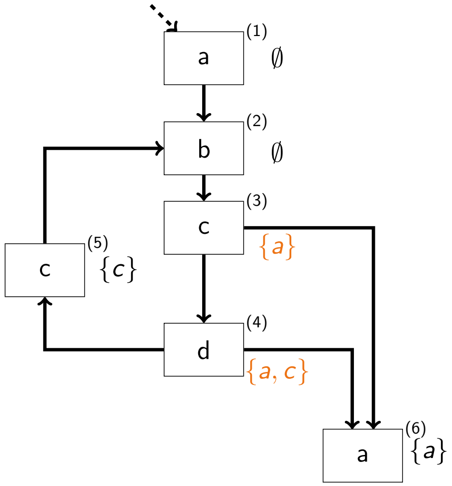
DC-UCB Analysis - Example



1. DC-UCB(P) := $\{m\}$
   if a) m accessed at P
   and b) $m \in Must\_Cache(P)$
2. ➔ forwards information to predecessors
3. DC-UCB(P) := (DC-UCB(P) $\setminus$ $Must\_Cache(P)$)
4. unless fixpoint reached, goto 2

l	Must-Cache
1	$(\_, \_, \_, \_)$
2	$(a, \_, \_, \_)$
3	$(a, b, \_, \_)$
4	$(a, b, c, \_)$
5	(a, b, c, d)
6	$(a, \_, c, \_)$

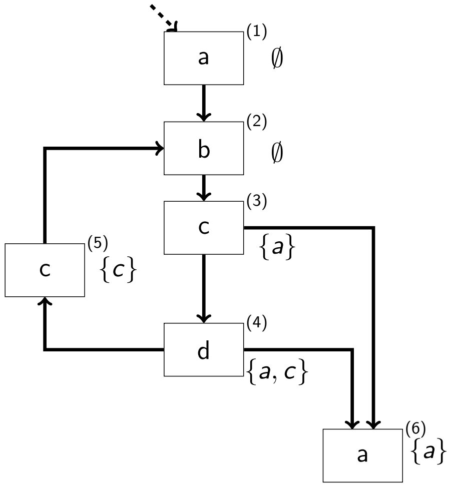
DC-UCB Analysis - Example



1. DC-UCB(P) := $\{m\}$
   if a) m accessed at P
   and b) $m \in Must\_Cache(P)$
2. ➔ forwards information to predecessors
3. DC-UCB(P) :=
   (DC-UCB(P) $\setminus$ $Must\_Cache(P)$)
4. unless fixpoint reached, goto 2

I	Must-Cache
1	$(\_, \_, \_, \_)$
2	$(a, \_, \_, \_)$
3	$(a, b, \_, \_)$
4	$(a, b, c, \_)$
5	(a, b, c, d)
6	$(a, \_, c, \_)$

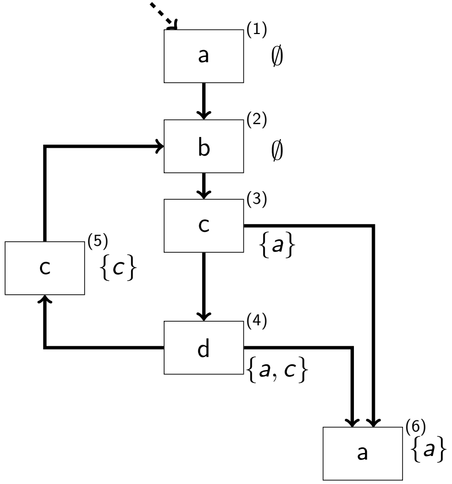
DC-UCB Analysis - Example



1. DC-UCB(P) := $\{m\}$
   if a) m accessed at P
   and b) $m \in Must\_Cache(P)$
2. forwards information to predecessors
3. ➔ DC-UCB(P) :=
   (DC-UCB(P) $\setminus$ $Must\_Cache(P)$)
4. unless fixpoint reached, goto 2

l	Must-Cache
1	$(\_, \_, \_, \_)$
2	$(a, \_, \_, \_)$
3	$(a, b, \_, \_)$
4	$(a, b, c, \_)$
5	(a, b, c, d)
6	$(a, \_, c, \_)$

DC-UCB Analysis - Example



1 DC-UCB(P) := $\{m\}$
   if a) m accessed at P
   and b) $m \in Must\_Cache(P)$

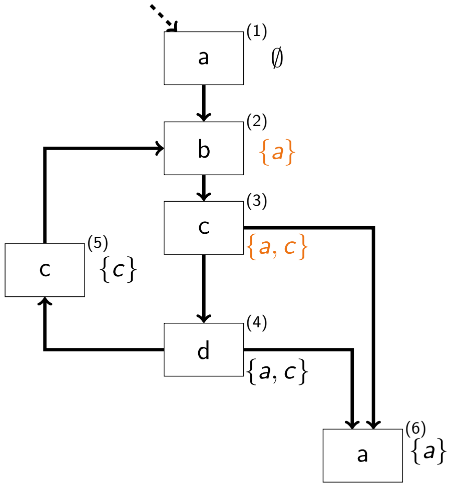
➔ 2 forwards information to predecessors

3 DC-UCB(P) :=
   (DC-UCB(P) $\setminus$ $Must\_Cache(P)$)

4 unless fixpoint reached, goto 2

I	Must-Cache
1	$(\_, \_, \_, \_)$
2	$(a, \_, \_, \_)$
3	$(a, b, \_, \_)$
4	$(a, b, c, \_)$
5	(a, b, c, d)
6	$(a, \_, c, \_)$

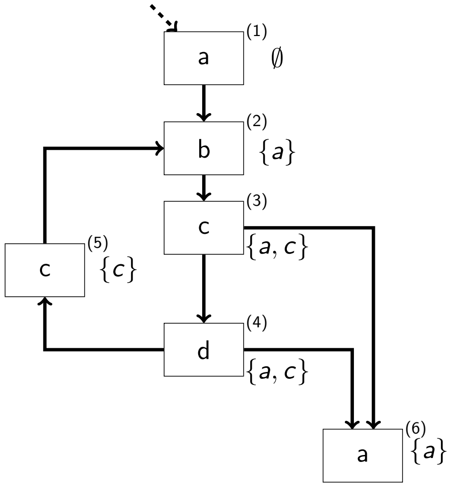
DC-UCB Analysis - Example



1. DC-UCB(P) := $\{m\}$
   if a) m accessed at P
   and b) $m \in Must\_Cache(P)$
2. forwards information to predecessors
3. DC-UCB(P) :=
   (DC-UCB(P) $\setminus$ $Must\_Cache(P)$)
4. unless fixpoint reached, goto 2

I	Must-Cache
1	$(\_, \_, \_, \_)$
2	$(a, \_, \_, \_)$
3	$(a, b, \_, \_)$
4	$(a, b, c, \_)$
5	(a, b, c, d)
6	$(a, \_, c, \_)$

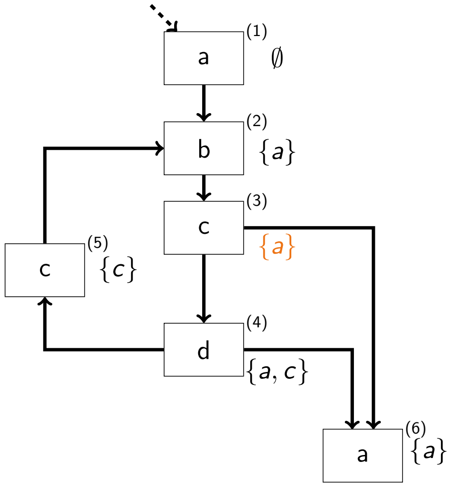
DC-UCB Analysis - Example



1. DC-UCB(P) := $\{m\}$
   if a) m accessed at P
   and b) $m \in Must\_Cache(P)$
2. forwards information to predecessors
3. ➔ DC-UCB(P) :=
   (DC-UCB(P) $\setminus$ $Must\_Cache(P)$)
4. unless fixpoint reached, goto 2

I	Must-Cache
1	$(\_, \_, \_, \_)$
2	$(a, \_, \_, \_)$
3	$(a, b, \_, \_)$
4	$(a, b, c, \_)$
5	(a, b, c, d)
6	$(a, \_, c, \_)$

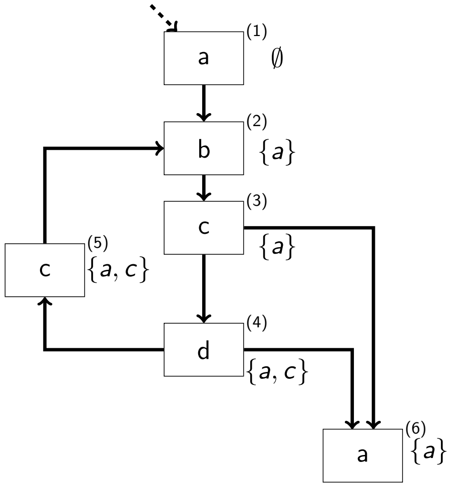
DC-UCB Analysis - Example



1. DC-UCB(P) := $\{m\}$
   if a) m accessed at P
   and b) $m \in Must\_Cache(P)$
2. forwards information to predecessors
3. ➜ DC-UCB(P) :=
   (DC-UCB(P) $\setminus$ $Must\_Cache(P)$)
4. unless fixpoint reached, goto 2

I	Must-Cache
1	$(\_, \_, \_, \_)$
2	$(a, \_, \_, \_)$
3	$(a, b, \_, \_)$
4	$(a, b, c, \_)$
5	(a, b, c, d)
6	$(a, \_, c, \_)$

DC-UCB Analysis - Example



1 DC-UCB(P) := $\{m\}$
if a) m accessed at P
and b) $m \in Must\_Cache(P)$

2 forwards information to predecessors

3 DC-UCB(P) :=
(DC-UCB(P) $\setminus$ $Must\_Cache(P)$)

➔ 4 unless fixpoint reached, goto 2

I	Must-Cache	DC-UCB
1	$(\_, \_, \_, \_)$	$\emptyset$
2	$(a, \_, \_, \_)$	$\{a\}$
3	$(a, b, \_, \_)$	$\{a\}$
4	$(a, b, c, \_)$	$\{a, c\}$
5	(a, b, c, d)	$\{a, c\}$
6	$(a, \_, c, \_)$	$\{a\}$

What about Data Caches and different Cache-Architectures?

DC-UCB analysis is independent of Cache-type:

1 DC-UCB(P) $:= \{m\}$
  if a) m accessed at P
  and b) $\underline{m \in Must\_Cache(P)}$

3 DC-UCB(P) $:= \underline{(\text{DC-UCB}(P) \setminus Must\_Cache(P))}$

only $Must\_Cache(P)$ influenced by cache-type!

$\longrightarrow$ DC-UCB analysis valid for cache-type T,
if $Must\_Cache$ for T exists

Problems:

Data caches under dynamically changing memory accesses (see Paper)

FIFO/PLRU unsuitable for (DC-)UCB analysis (see WCET-Workshop)

Evaluation

Setting:

- ARM 7 processor
- direct-mapped instruction cache (8kB, 1024 sets, 8 Byte linesize)
- testcases: Mälardalen WCET benchmark suite

Two scenarios:

1. UCBs compared to DC-UCBs (average # and maximal #)
2. UCBs vs. DC-UCBs in relation to WCET (assumed preemption each 10000 cycles, cache reload time 4 cycles)

1. UCB vs. DC-UCB

	average			upper bound		
	#UCB	#DC-UCB	Improv.	#UCB	#DC-UCB	Improv.
bs	13.6	1.4	52%	24	5	79%
bsort100	18.9	1.9	54%	35	8	77%
crc	115.0	2.5	84%	134	14	90%
fac	10.8	1.2	51%	19	4	79%
fibcall	5.1	1.6	41%	12	5	58%
fir	47.8	1.9	58%	79	9	89%
insertsort	7.8	2.1	31%	19	10	47%
loop3	3.7	1.5	39%	6	4	33%
matmult	27.6	5.6	56%	40	23	42%
minmax	1.8	1.1	9%	11	9	18%
ns	12.9	2.4	35%	31	13	58%
qsort-exam	127.1	1.9	78%	160	15	91%
qurt	340.8	1.4	76%	449	14	97%
select	102.0	1.8	73%	138	15	89%
sqrt	204.1	1.2	60%	361	14	96%

1. UCB vs. DC-UCB

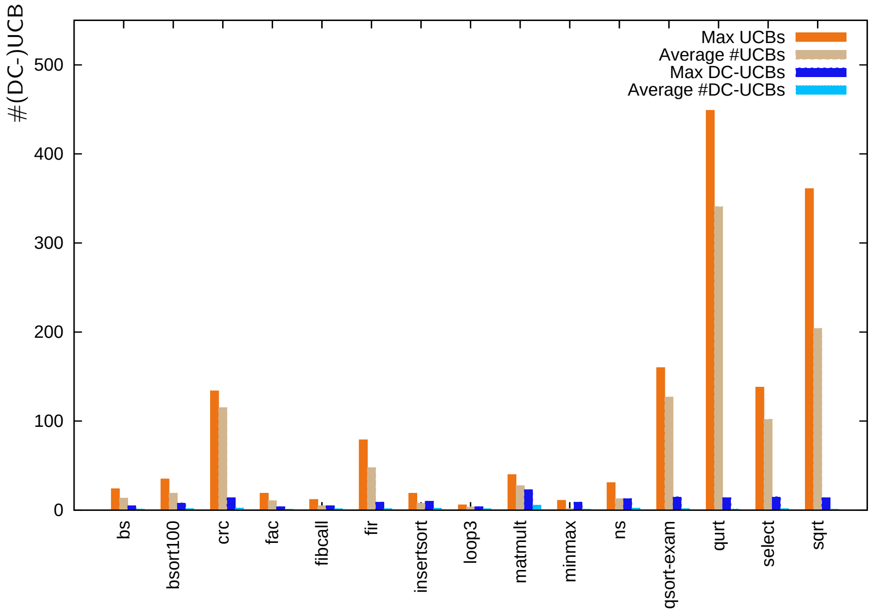
	average			upper bound		
	#UCB	#DC-UCB	Improv.	#UCB	#DC-UCB	Improv.
bs	13.6	1.4	52%	24	5	79%
bsort100	18.9	1.9	54%	35	8	77%
crc	115.0	2.5	84%	134	14	90%
fac	10.8	1.2	51%	19	4	79%
fibcall	5.1	1.6	41%	12	5	58%
fir	47.8	1.9	58%	79	9	89%
insertsort	7.8	2.1	31%	19	10	47%
loop3	3.7	1.5	39%	6	4	33%
matmult	27.6	5.6	56%	40	23	42%
minmax	1.8	1.1	9%	11	9	18%
ns	12.9	2.4	35%	31	13	58%
qsort-exam	127.1	1.9	78%	160	15	91%
qurt	340.8	1.4	76%	449	14	97%
select	102.0	1.8	73%	138	15	89%
sqrt	204.1	1.2	60%	361	14	96%

improvement UCB vs DC-UCB

1. UCB vs. DC-UCB

	average			upper bound		
	#UCB	#DC-UCB	Improv.	#UCB	#DC-UCB	Improv.
bs	13.6	1.4	52%	24	5	79%
bsort100	18.9	1.9	54%	35	8	77%
crc	115.0	2.5	84%	134	14	90%
fac	10.8	1.2	51%	19	4	79%
fibcall	5.1	1.6	41%	12	5	58%
fir	47.8	1.9	58%	79	9	89%
insertsort	7.8	2.1	31%	19	10	47%
loop3	3.7	1.5	39%	6	4	33%
matmult	27.6	5.6	56%	40	23	42%
minmax	1.8	1.1	9%	11	9	18%
ns	12.9	2.4	35%	31	13	58%
qsort-exam	127.1	1.9	78%	160	15	91%
qurt	340.8	1.4	76%	449	14	97%
select	102.0	1.8	73%	138	15	89%
sqrt	204.1	1.2	60%	361	14	96%

improvement UCB vs DC-UCB

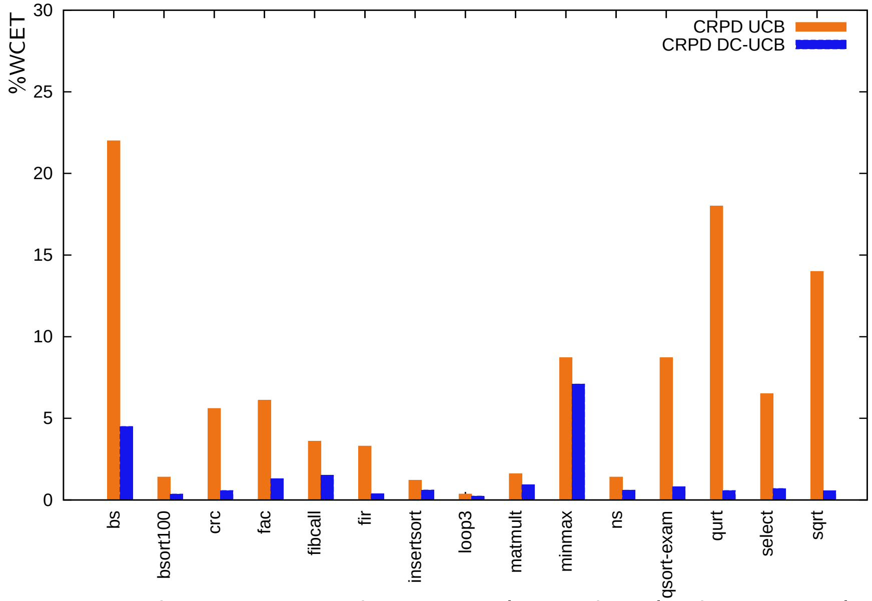


#(DC-)UCB
0
100
200
300
400
500
Max UCBs
Average #UCBs
Max DC-UCBs
Average #DC-UCBs
bs
bsort100
crc
fac
fibcall
fir
insertsort
loop3
matmult
minmax
ns
qsort-exam
qurt
select
sqrt

2. Relation to WCET

	WCET	CRPD UCB	CRPD DC-UCB
bs	445	96	20
bsort100	1567222	21980	5652
crc	290782	16320	1680
fac	1252	76	16
fibcall	1351	48	20
fir	29160	948	108
insertsort	6573	76	40
loop3	13449	48	32
matmult	742585	12000	6900
minmax	504	44	36
ns	43319	620	260
qsort-exam	22146	1920	180
qurt	214076	39512	1232
select	17088	1104	120
sqrt	39962	5776	224

assumed preemption each 10000 cycles, cache reload time 4 cycles



assumed preemption each 10000 cycles, cache reload time 4 cycles

Conclusions

DC-UCBs:

- improves on UCB by omitting double counted cache-misses
- improvement up to 90%, compared to UCBs
- only safe in combination with WCET analysis (no restriction)
- LRU/Data DC-UCB analysis comes for free

Thanks for your attention!

Bibliography:

J. V. Busquets-Mataix, J. J. Serrano, R. Ors, P. Gil, and A. Wellings.
Adding instruction cache effect to schedulability analysis of preemptive real-time systems.
In *Proceedings of the 2nd IEEE Real-Time Technology and Applications Symposium (RTAS'96)*, page 204, Washington, DC, USA, 1996. IEEE Computer Society.

C.-G. Lee, J. Hahn, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim.
Analysis of cache-related preemption delay in fixed-priority preemptive scheduling.
In *Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS'96)*, page 264, Washington, DC, USA, 1996. IEEE Computer Society.

C.-G. Lee, J. Hahn, Y.-M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim.
Analysis of cache-related preemption delay in fixed-priority preemptive scheduling.
IEEE Trans. Comput., 47(6):700–713, 1998.

H. S. Negi, T. Mitra, and A. Roychoudhury.
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, 2003. ACM.

J. Staschulat and R. Ernst.
Multiple process execution in cache related preemption delay analysis.
In *Proceedings of the 4th ACM international conference on Embedded software (EMSOFT'04)*, pages 278–286, New York, NY, USA, 2004. ACM.

J. Staschulat and R. Ernst.
Scalable precision cache analysis for preemptive scheduling.
In *Proceedings of the 2005 ACM conference on Languages, compilers, and tools for embedded systems (LCTES'05)*, pages 157–165, New York, NY, USA, 2005. ACM.

J. Staschulat and R. Ernst.
Scalable precision cache analysis for real-time software.

Relation: Cache Size / Code Size

Task	bs	bsort100	crc	fac	fibcall	fir	insertsort	loop3
#Instructions	69	123	288	48	47	209	81	1633
WCET	445	1567222	290782	1252	1351	29160	6573	13449
ratio 1KB	0.27	0.48	1.12	0.19	0.18	0.81	0.31	6.38
ratio 4KB	0.07	0.12	0.28	0.05	0.05	0.2	0.08	1.59
ratio 8KB	0.03	0.06	0.14	0.02	0.02	0.10	0.04	0.80

Task	matmul	minmax	ns	qsort-exam	qurt	select	sqrt
#Instructions	200	138	127	340	967	302	953
WCET	742585	504	43319	22146	214318	17 088	39962
ratio 1KB	0.78	0.54	0.50	1.33	3.7 8	1.18	3.73
ratio 4KB	0.2	0.13	0.12	0.33	0.94	0.29	0.93
ratio 8KB	0.10	0.07	0.06	0.17	0.4 7	0.15	0.47