CISTER - Research Center in
Real-Time & Embedded Computing Systems

Trading Between Intra- and Inter-Task Cache Interference to Improve Schedulability

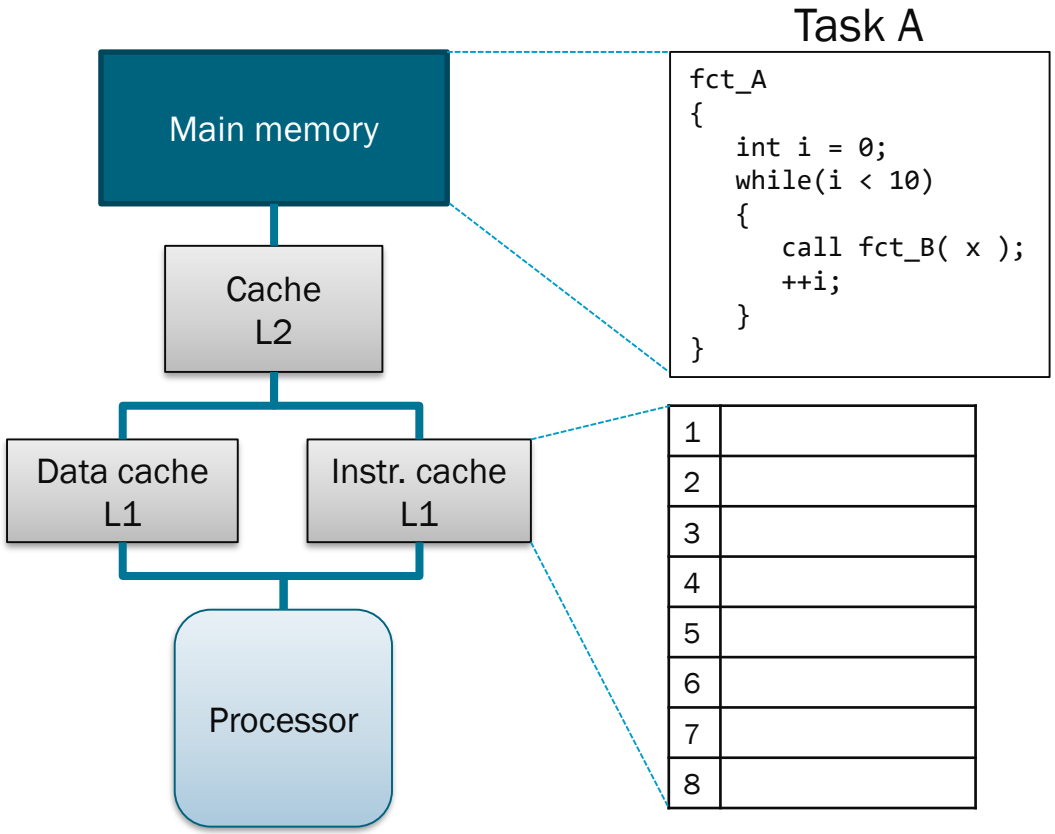
Syed Aftab Rashid, Geoffrey Nelissen,Eduardo Tovar

CISTER
Research Center in
Real-Time & Embedded
Computing Systems

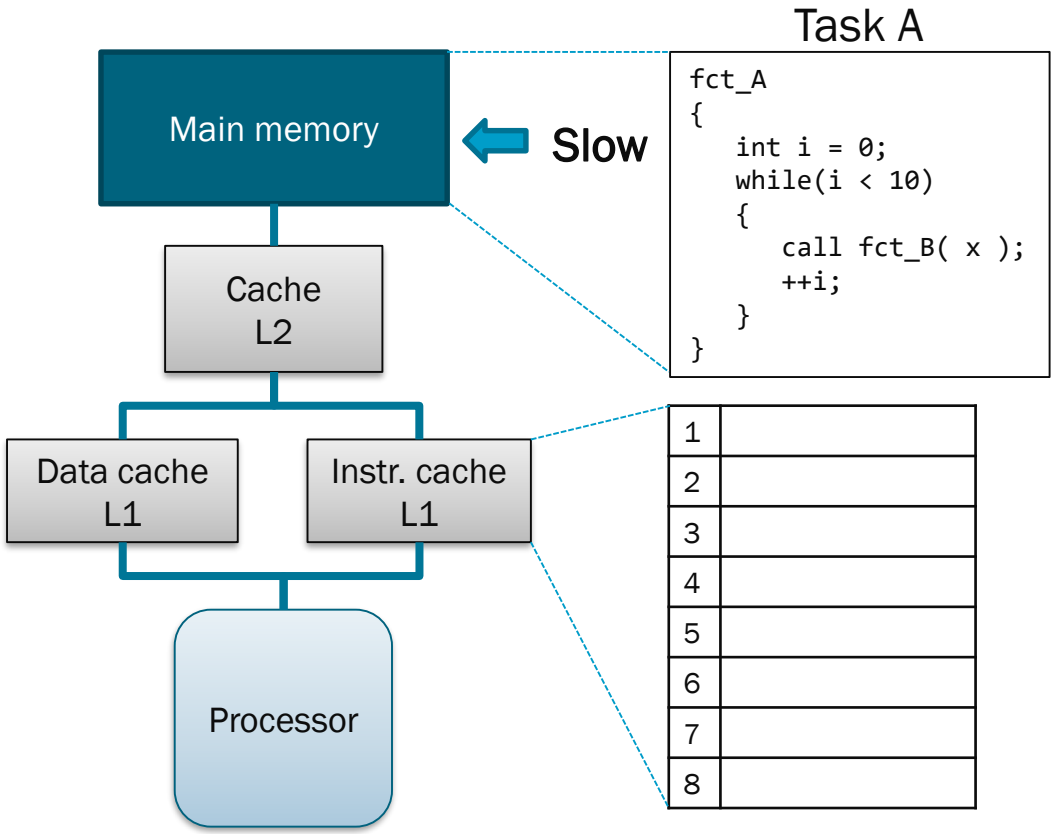
isep Instituto Superior de **Engenharia** do Porto

RTNS 2018

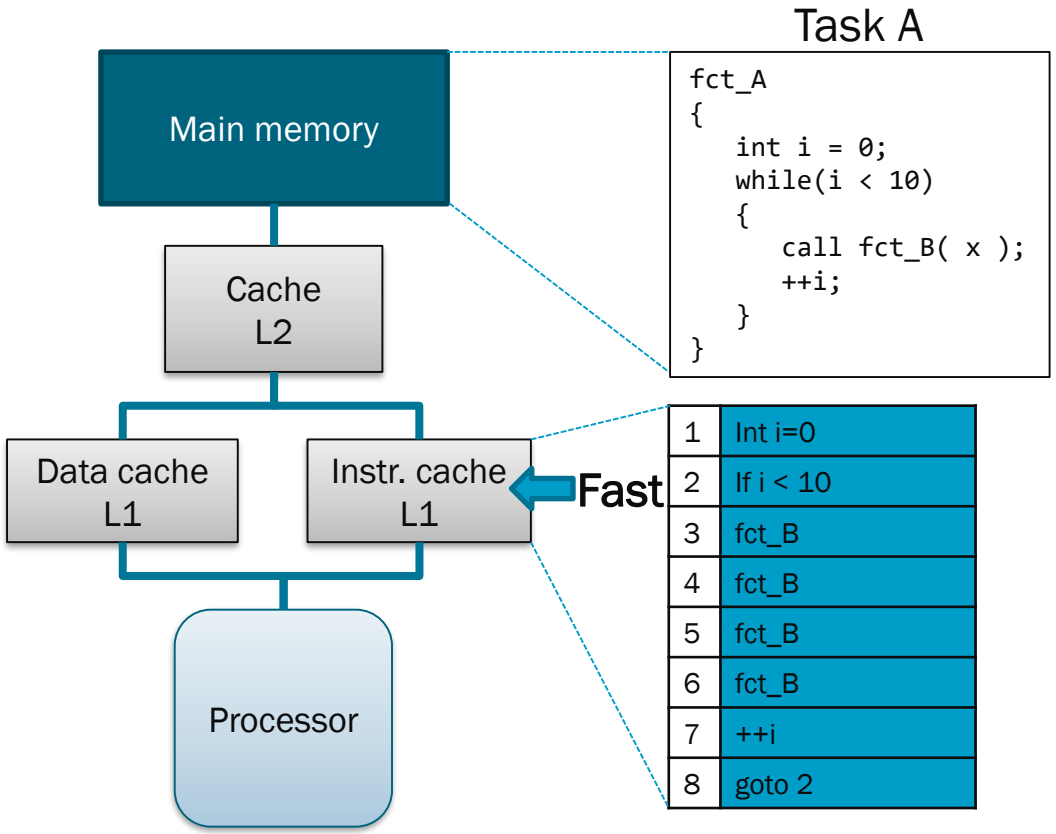
Caches Reduces Average Memory Access Time



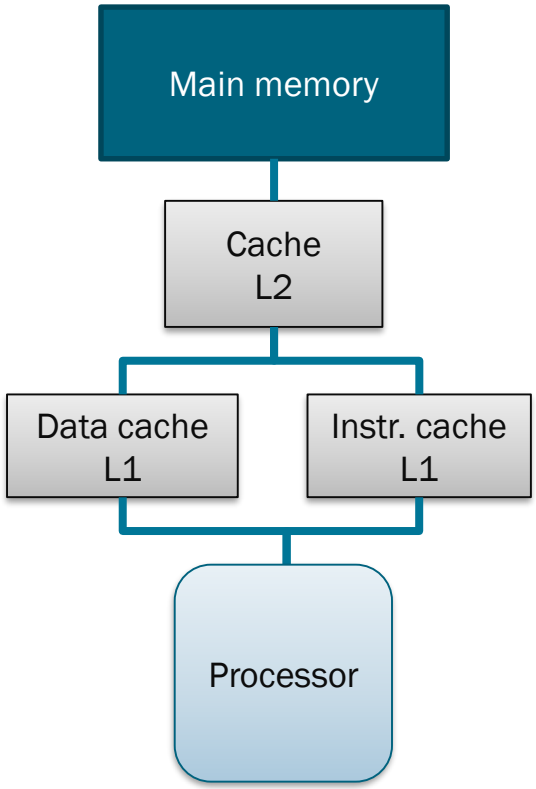
Caches Reduces Average Memory Access Time



Caches Reduces Average Memory Access Time



Caches Reduces Average Memory Access Time



The utilisation of caches reduces the worst-case execution time (WCET) of a task in isolation

Cache Memories Cause Time Variability due to Cache Interference

Cache Memories Cause Time Variability due to Cache Interference

- Intra-Task Cache Interference
- Inter-Task Cache Interference

What is Intra-task Cache Interference?

- Intra-task Cache Interference

Task A

```
fct_A
{
   int i = 0;
   while(i < 10)
   {
      call fct_B( x );
      ++i;
   }
}

fct_C
{
   int j = 0;
   while(j < 10)
   {
      call fct_D( y );
      ++j;
   }
}
```

Cache

1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	

What is Intra-task Cache Interference?

- Intra-task Cache Interference

Task A

```
fct_A
{
   int i = 0;
   while(i < 10)
   {
      call fct_B( x );
      ++i;
   }
}

fct_C
{
   int j = 0;
   while(j < 10)
   {
      call fct_D( y );
      ++j;
   }
}
```

Cache

1	Int i=0
2	If i < 10
3	fct_B
4	fct_B
5	fct_B
6	fct_B
7	++i
8	goto 2
9	
10	
11	
12	
13	
14	
15	
16	

What is Intra-task Cache Interference?

- Intra-task Cache Interference

Task A

```
fct_A
{
   int i = 0;
   while(i < 10)
   {
      call fct_B( x );
      ++i;
   }
}

fct_C
{
   int j = 0;
   while(j < 10)
   {
      call fct_D( y );
      ++j;
   }
}
```

Cache

1	Int i=0
2	If i < 10
3	fct_B
4	fct_B
5	fct_B
6	fct_B
7	++i
8	goto 2
9	Int j=0
10	If j < 10
11	fct_D
12	fct_D
13	fct_D
14	fct_D
15	++j
16	goto 10

What is Intra-task Cache Interference?

- Intra-task Cache Interference

Task A

```
fct_A
{
   int i = 0;
   while(i < 10)
   {
      call fct_B( x );
      ++i;
   }
}

fct_C
{
   int j = 0;
   while(j < 10)
   {
      call fct_D( y );
      ++j;
   }
}
```

Cache

1	Int i=0
2	If i < 10
3	fct_B
4	fct_B
5	fct_B
6	fct_B
7	++i
8	goto 2
9	Int j=0
10	If j < 10
11	fct_D
12	fct_D

What is Intra-task Cache Interference?

- Intra-task Cache Interference

Task A

```
fct_A
{
   int i = 0;
   while(i < 10)
   {
      call fct_B( x );
      ++i;
   }
}

fct_C
{
   int j = 0;
   while(j < 10)
   {
      call fct_D( y );
      ++j;
   }
}
```

Cache

1	Int i=0 / fct_D
2	If i < 10 / fct_D
3	fct_B / ++j
4	fct_B / goto 10
5	fct_B
6	fct_B
7	++i
8	goto 2
9	Int j=0
10	If j < 10
11	fct_D
12	fct_D

What is Intra-task Cache Interference?

- Intra-task Cache Interference

Task A

```
fct_A
{
   int i = 0;
   while(i < 10)
   {
      call fct_B( x );
      ++i;
   }
}

fct_C
{
   int j = 0;
   while(j < 10)
   {
      call fct_D( y );
      ++j;
   }
}
```

Cache

1	Int i=0 / fct_D
2	If i < 10 / fct_D
3	fct_B / ++j
4	fct_B / goto 10
5	fct_B
6	fct_B
7	++i
8	goto 2
9	Int j=0
10	If j < 10
11	fct_D
12	fct_D

Intra-task cache interference occurs if the **memory footprint** of a task is **larger** than the **allocated** cache space or when two memory **entries** of that task are **mapped** to the **same** space in cache.

Increase in **intra-task cache interference** may also result in **increasing** the **WCET** of tasks.

What is Inter-task Cache Interference?

- Inter-task Cache Interference

What is Inter-task Cache Interference?

- Inter-task Cache Interference
  - –Cache Related Preemption Delays (CRPD)
  - –Cache Persistence Reload Overhead (CPRO)

Inter-task Cache Interference due to CRPDs

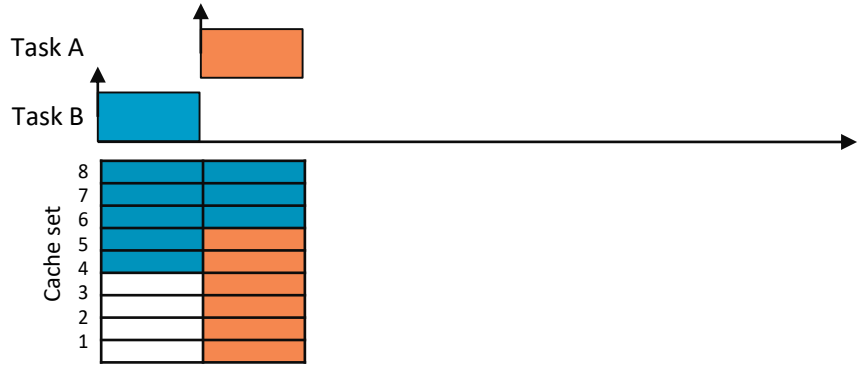
- Inter-task Cache Interference
  - Cache Related Preemption Delays (CRPD)

Inter-task Cache Interference due to CRPDs

- Inter-task Cache Interference
  - Cache Related Preemption Delays (CRPD)

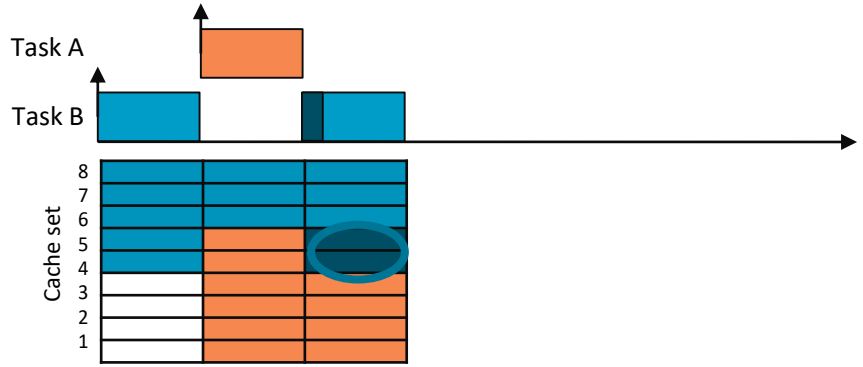
Inter-task Cache Interference due to CRPDs

- Inter-task Cache Interference
  - Cache Related Preemption Delays (CRPD)



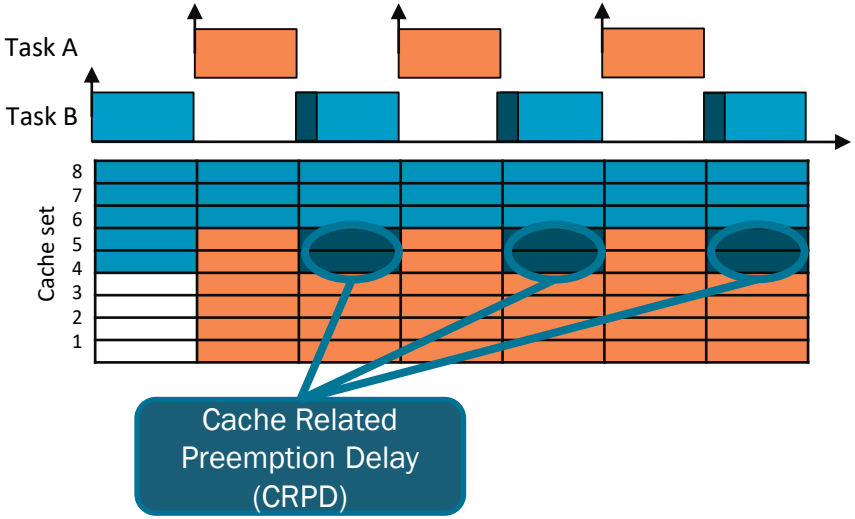
Inter-task Cache Interference due to CRPDs

- Inter-task Cache Interference
  - Cache Related Preemption Delays (CRPD)



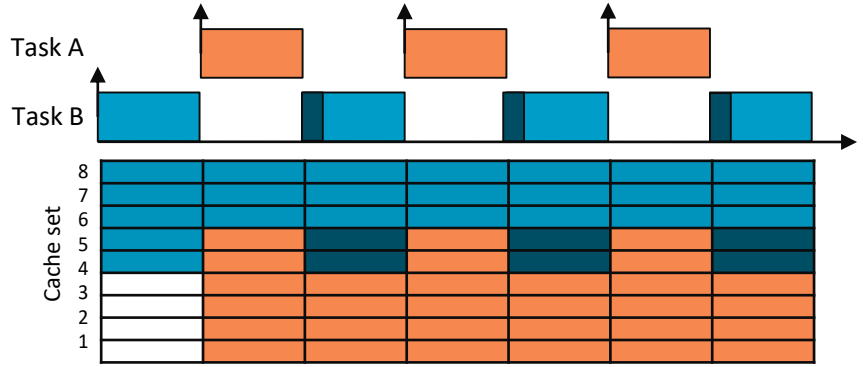
Inter-task Cache Interference due to CRPDs

- Inter-task Cache Interference
  - Cache Related Preemption Delays (CRPD)



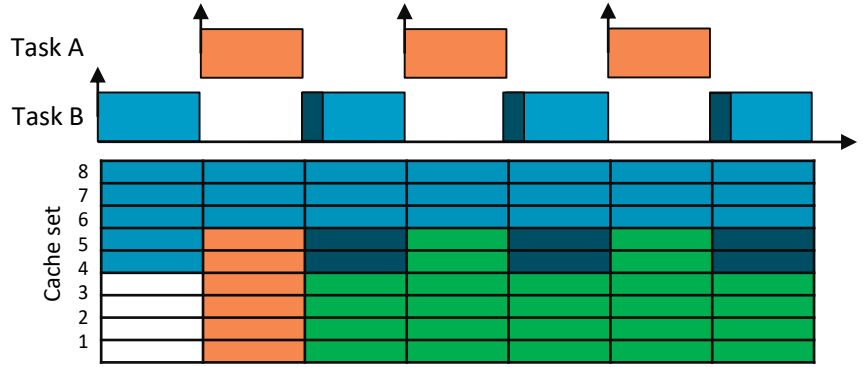
Inter-task Cache Interference due to CPROs

- Inter-task Cache Interference
  - Cache Persistence Reload Overhead (CPRO)



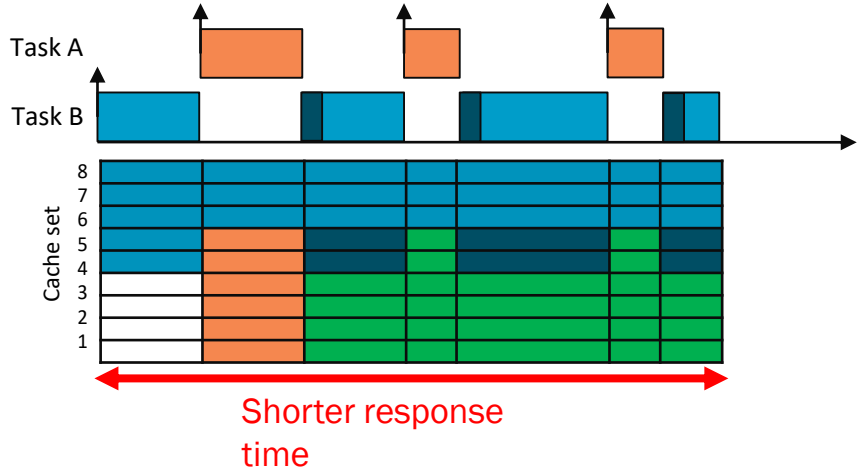
Inter-task Cache Interference due to CPROs

- Inter-task Cache Interference
  - Cache Persistence Reload Overhead (CPRO)



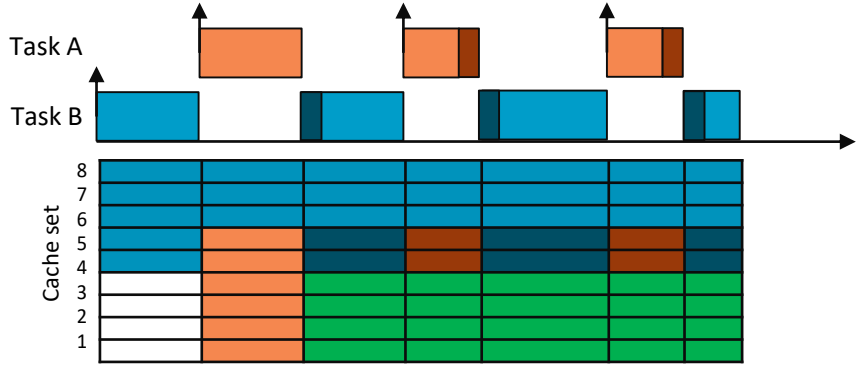
Inter-task Cache Interference due to CPROs

- Inter-task Cache Interference
  - Cache Persistence Reload Overhead (CPRO)



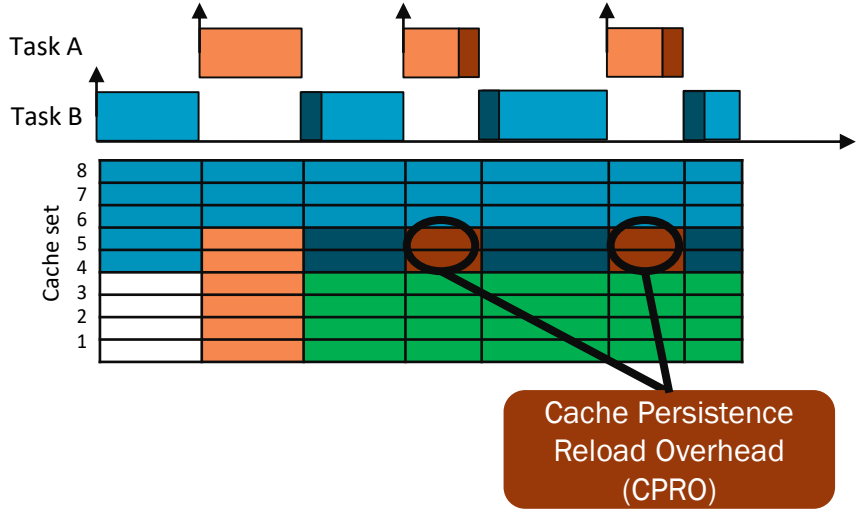
Inter-task Cache Interference due to CPROs

- Inter-task Cache Interference
  - Cache Persistence Reload Overhead (CPRO)



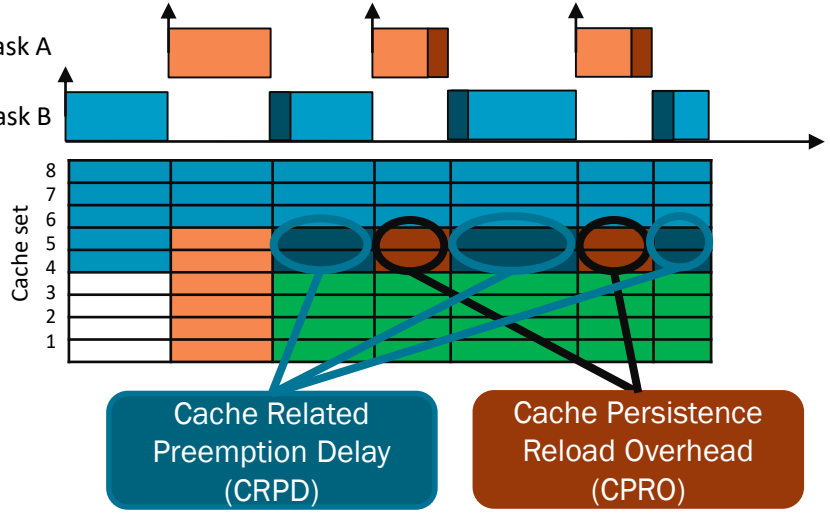
Inter-task Cache Interference due to CPROs

- Inter-task Cache Interference
  - Cache Persistence Reload Overhead (CPRO)



Cache Memories Cause Time Variability due to Intra- and inter-task cache interference

- Inter-task Cache Interference
  - –Cache Related Preemption Delay (CRPD)
  - –Cache Persistence Reload Overhead (CPRO)



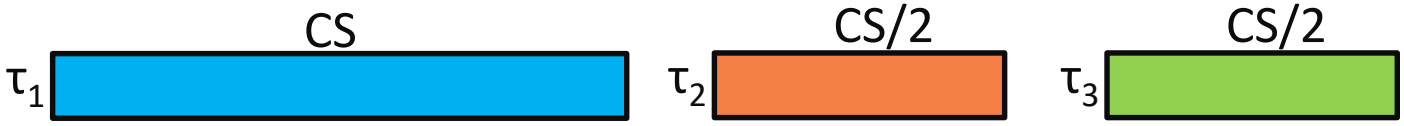
Both **CRPD** and **CPRO** may result in **increasing** the **WCET/WCRT** of tasks.

Intra- and Inter-Task Cache Interference depends on the Cache Allocation of Tasks

- Example: Task set $\Gamma=\{\tau_1, \tau_2, \tau_3\}$ and a cache of total size CS.

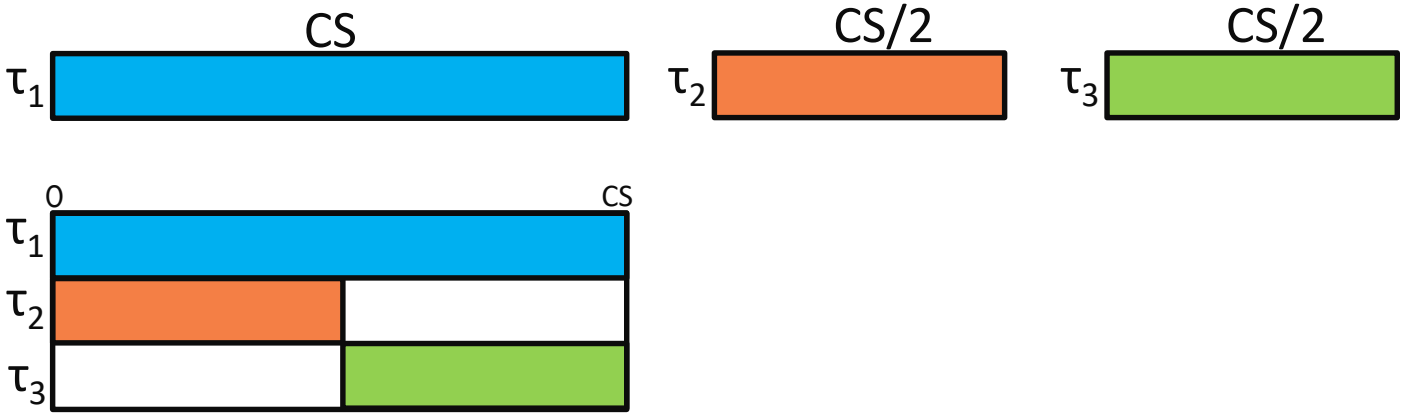
Intra- and Inter-Task Cache Interference depends on the Cache Allocation of Tasks

- Example: Task set $\Gamma=\{\tau_1, \tau_2, \tau_3\}$ and a cache of total size CS.



Intra- and Inter-Task Cache Interference depends on the Cache Allocation of Tasks

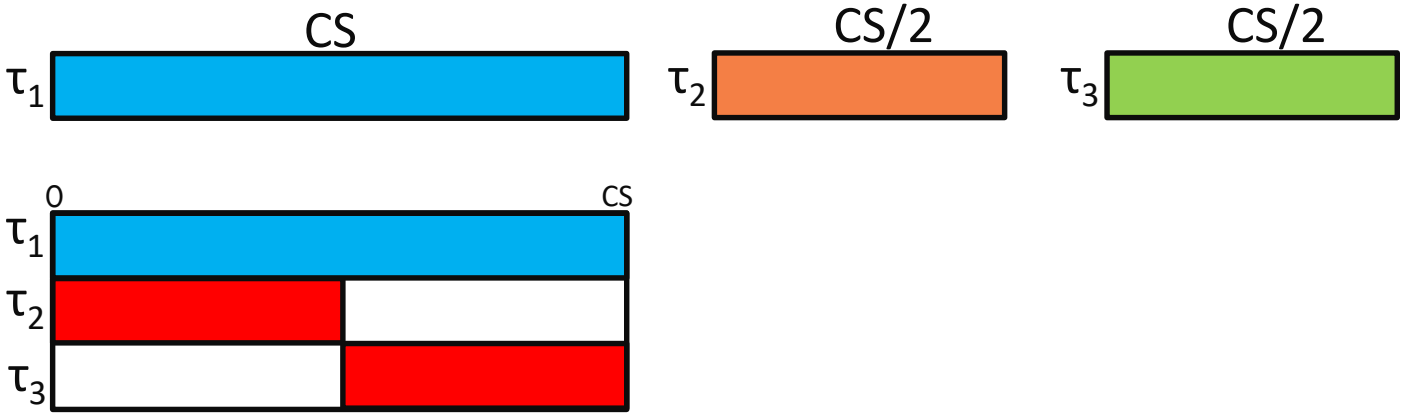
- Example: Task set $\Gamma=\{\tau_1, \tau_2, \tau_3\}$ and a cache of total size CS.



Sequential Cache Allocation

Intra- and Inter-Task Cache Interference depends on the Cache Allocation of Tasks

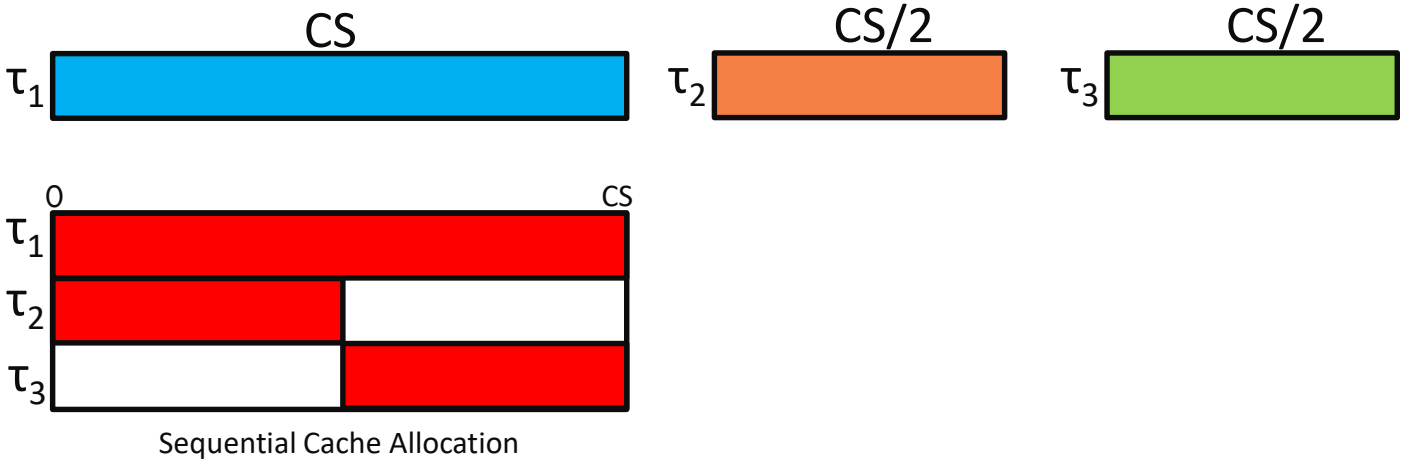
- Example: Task set $\Gamma=\{\tau_1, \tau_2, \tau_3\}$ and a cache of total size CS.



Sequential Cache Allocation

Intra- and Inter-Task Cache Interference depends on the Cache Allocation of Tasks

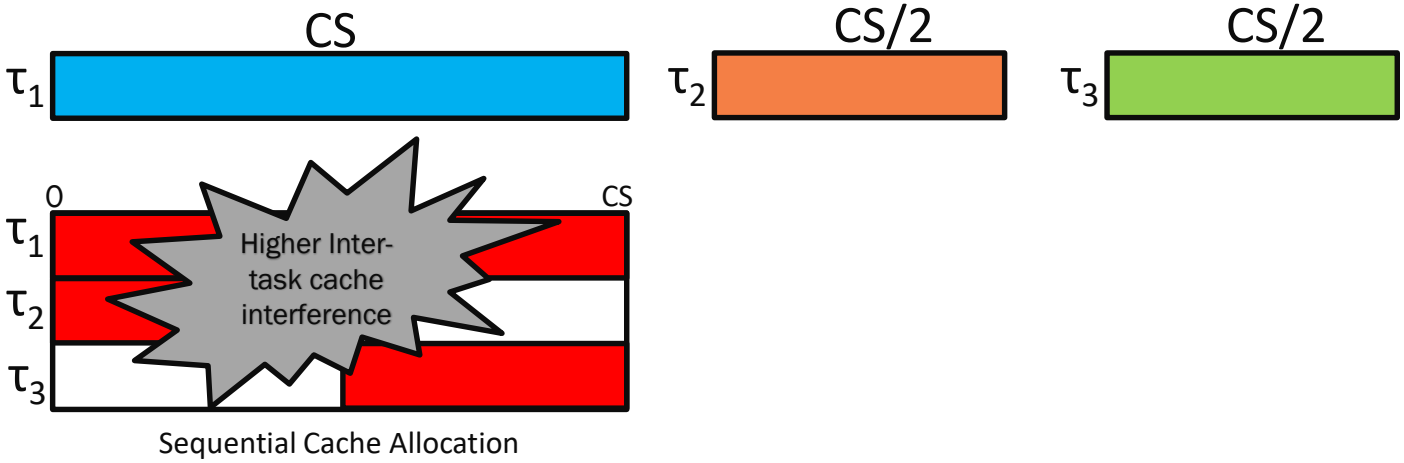
- Example: Task set $\Gamma=\{\tau_1, \tau_2, \tau_3\}$ and a cache of total size CS.



Sequential Cache Allocation

Intra- and Inter-Task Cache Interference depends on the Cache Allocation of Tasks

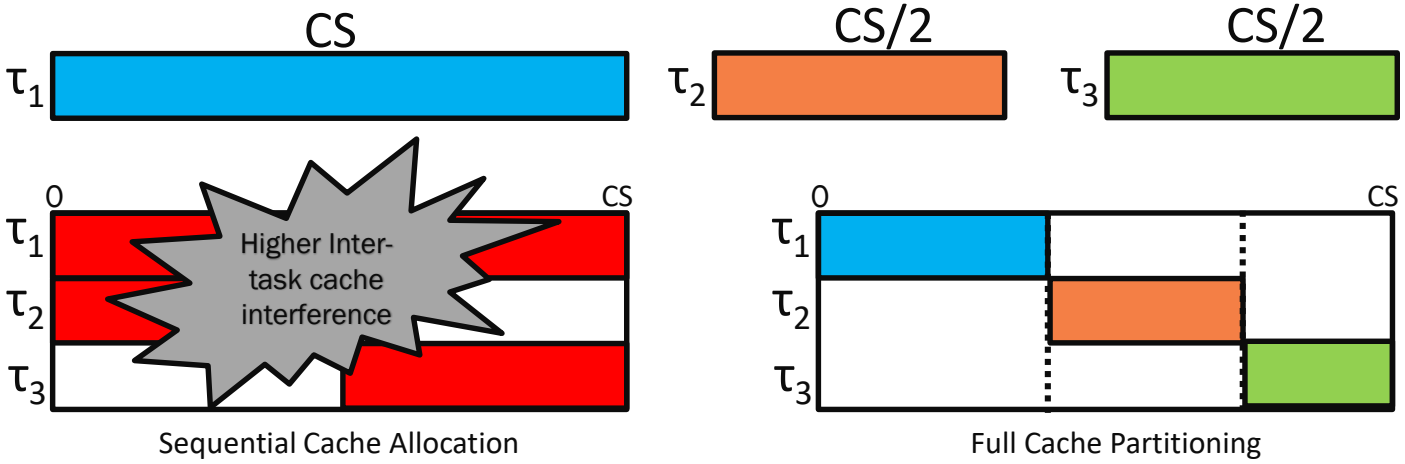
- Example: Task set $\Gamma=\{\tau_1, \tau_2, \tau_3\}$ and a cache of total size CS.



Sequential Cache Allocation

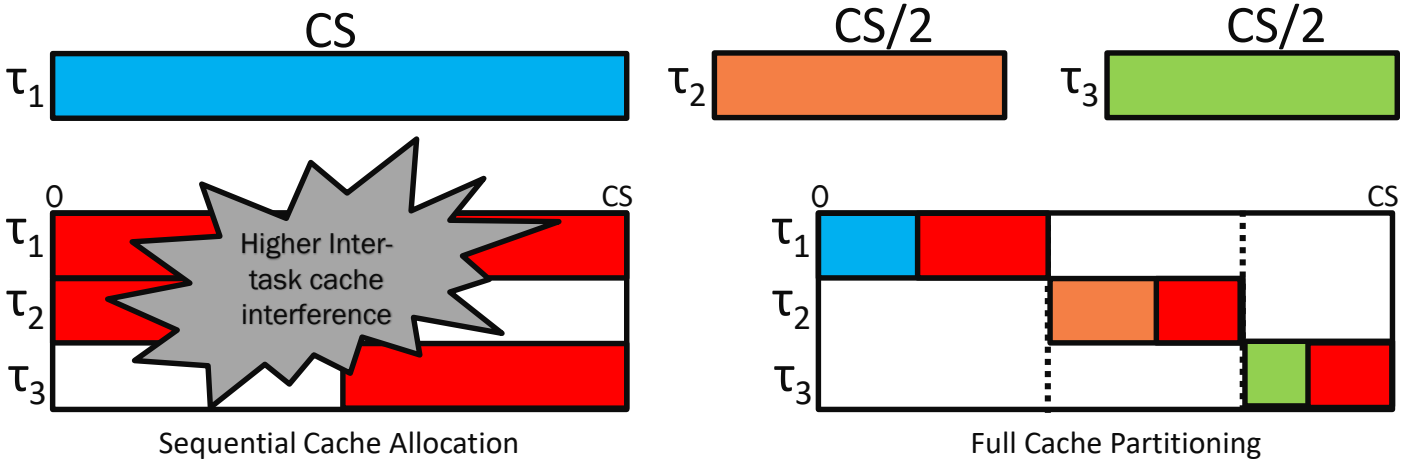
Intra- and Inter-Task Cache Interference depends on the Cache Allocation of Tasks

- Example: Task set $\Gamma=\{\tau_1, \tau_2, \tau_3\}$ and a cache of total size CS.



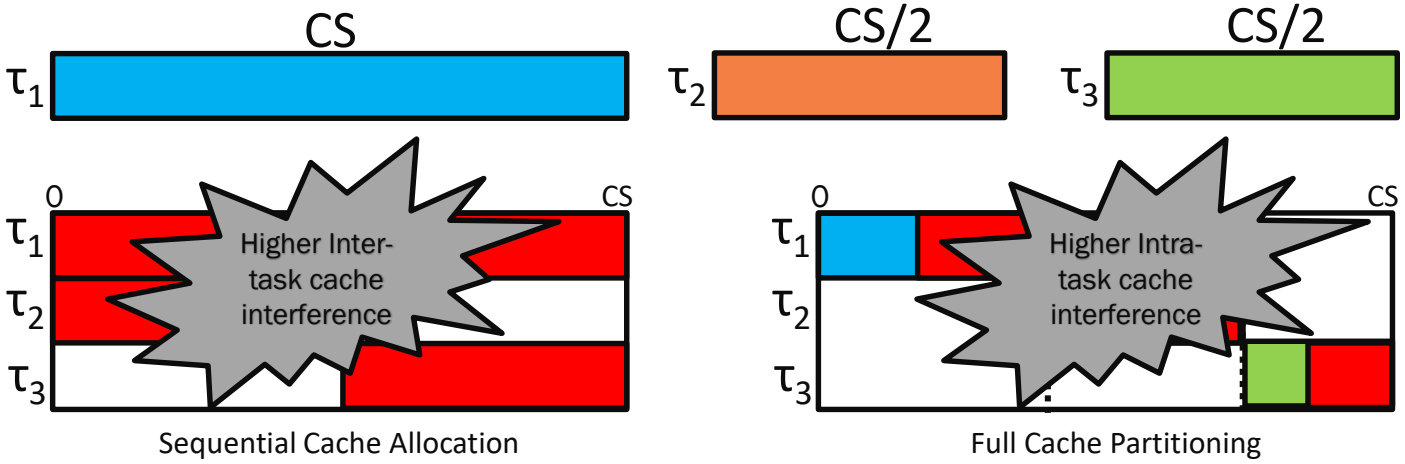
Intra- and Inter-Task Cache Interference depends on the Cache Allocation of Tasks

- Example: Task set $\Gamma=\{\tau_1, \tau_2, \tau_3\}$ and a cache of total size CS.



Intra- and Inter-Task Cache Interference depends on the Cache Allocation of Tasks

- Example: Task set $\Gamma=\{\tau_1, \tau_2, \tau_3\}$ and a cache of total size CS.

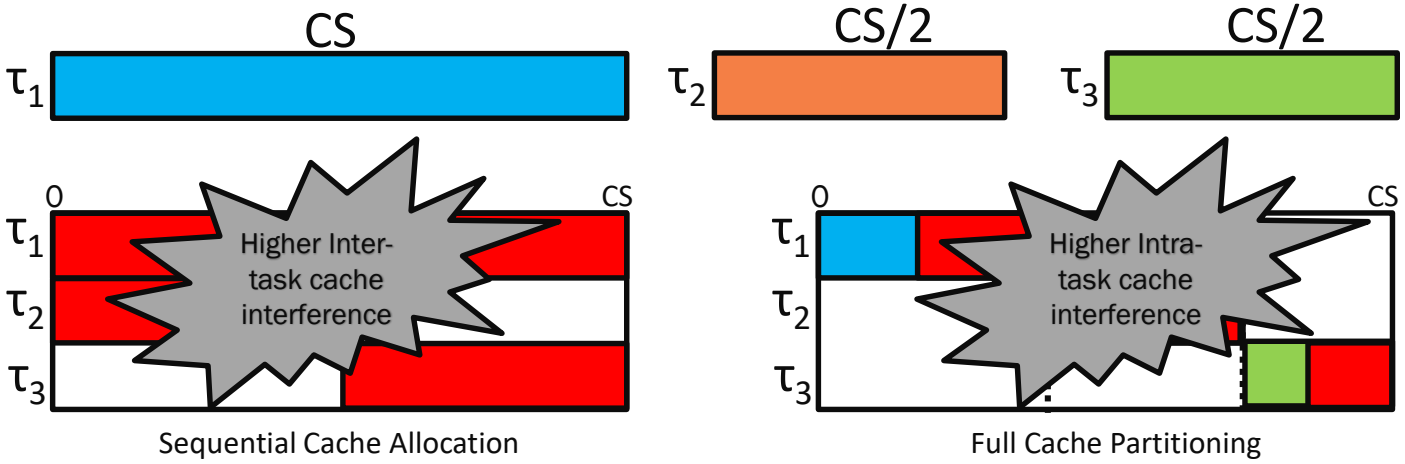


Sequential Cache Allocation

Full Cache Partitioning

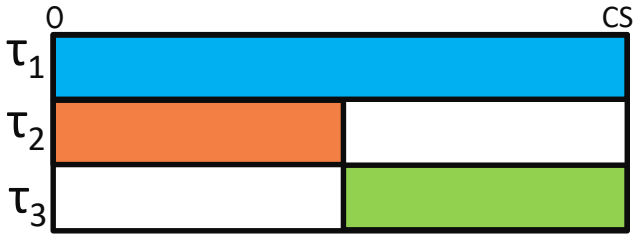
Intra- and Inter-Task Cache Interference depends on the Cache Allocation of Tasks

- Example: Task set $\Gamma=\{\tau_1, \tau_2, \tau_3\}$ and a cache of total size CS.



Sequential Cache Allocation

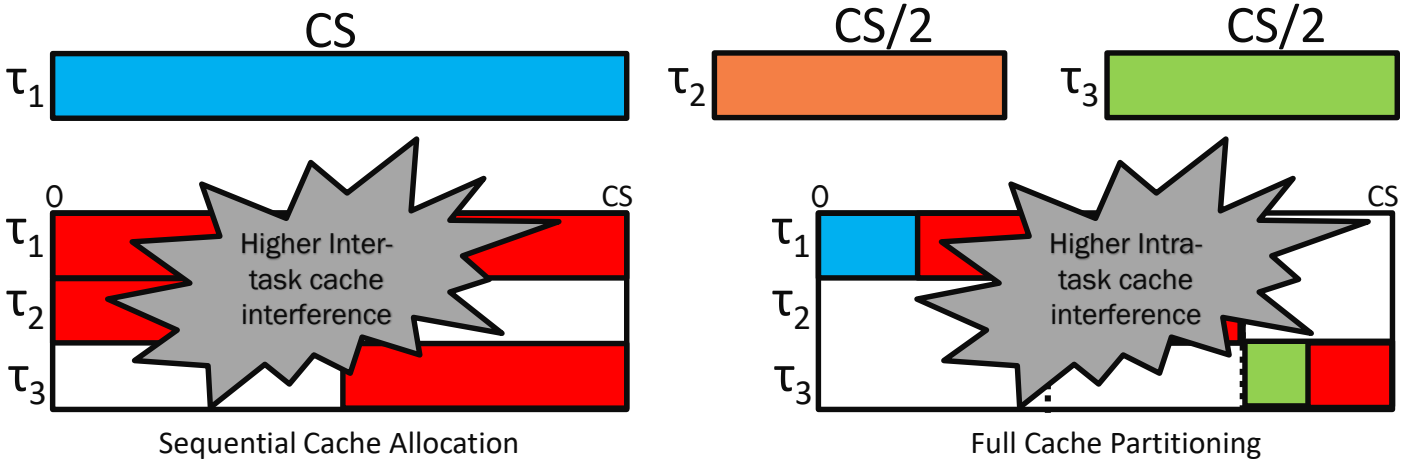
Full Cache Partitioning



Optimized Cache Allocation

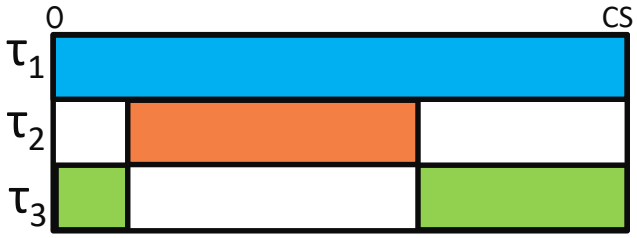
Intra- and Inter-Task Cache Interference depends on the Cache Allocation of Tasks

- Example: Task set $\Gamma=\{\tau_1, \tau_2, \tau_3\}$ and a cache of total size CS.



Sequential Cache Allocation

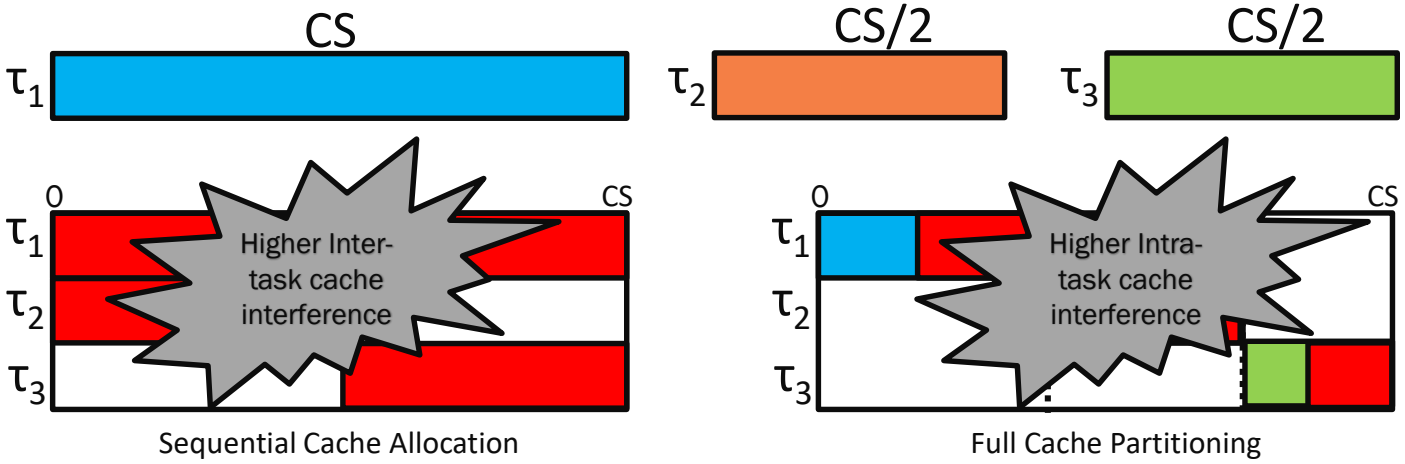
Full Cache Partitioning



Optimized Cache Allocation

Intra- and Inter-Task Cache Interference depends on the Cache Allocation of Tasks

- Example: Task set $\Gamma=\{\tau_1, \tau_2, \tau_3\}$ and a cache of total size CS.



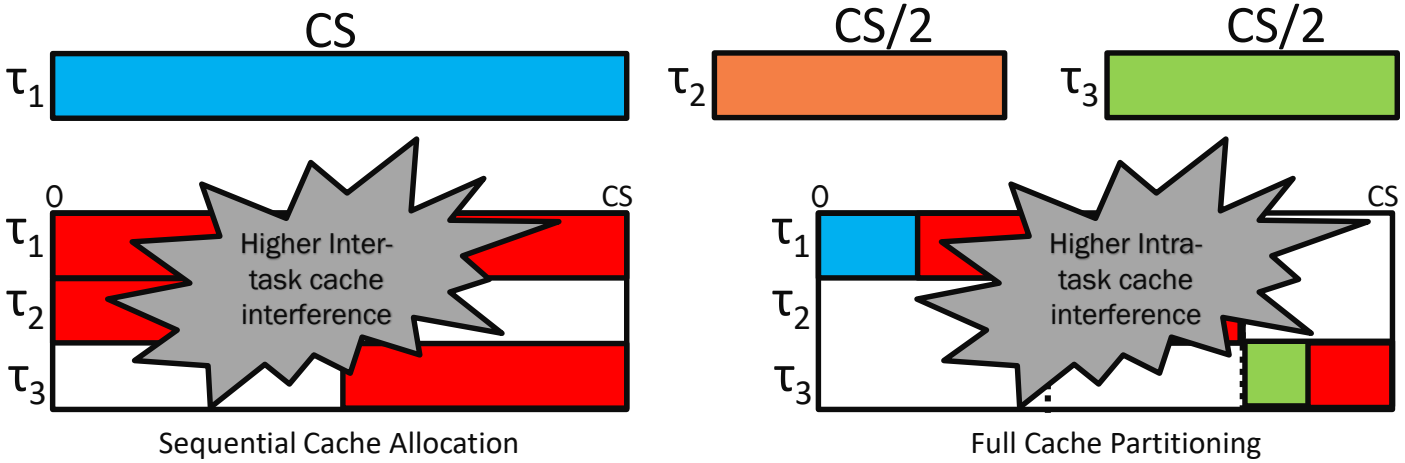
Sequential Cache Allocation

Full Cache Partitioning

Optimized Cache Allocation

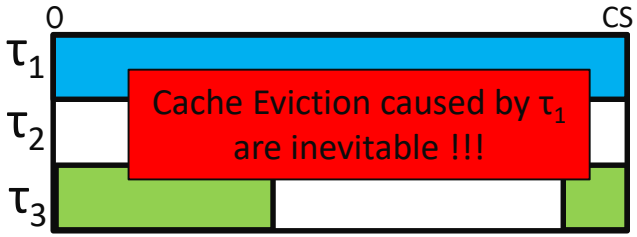
Intra- and Inter-Task Cache Interference depends on the Cache Allocation of Tasks

- Example: Task set $\Gamma=\{\tau_1, \tau_2, \tau_3\}$ and a cache of total size CS.



Sequential Cache Allocation

Full Cache Partitioning

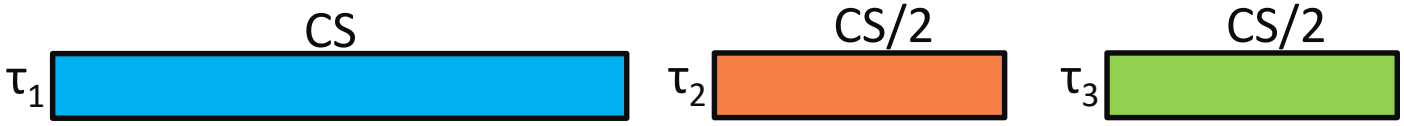


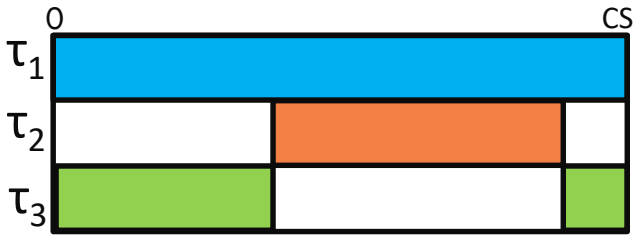
Optimized Cache Allocation

Trading between Intra- and Inter-task cache interference may improve schedulability

Trading between Intra- and Inter-task cache interference may improve schedulability

- Example: Task set $\Gamma=\{\tau_1, \tau_2, \tau_3\}$ and a cache of total size CS.

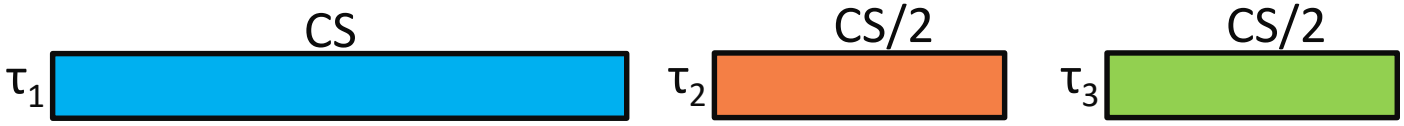


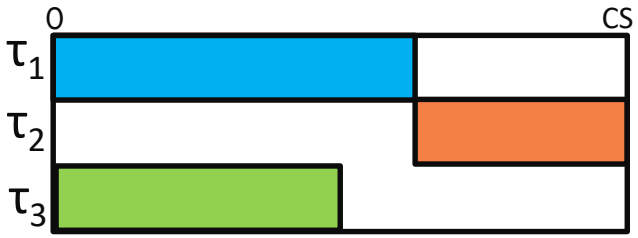


Optimized Cache Allocation

Trading between Intra- and Inter-task cache interference may improve schedulability

- Example: Task set $\Gamma=\{\tau_1, \tau_2, \tau_3\}$ and a cache of total size CS.

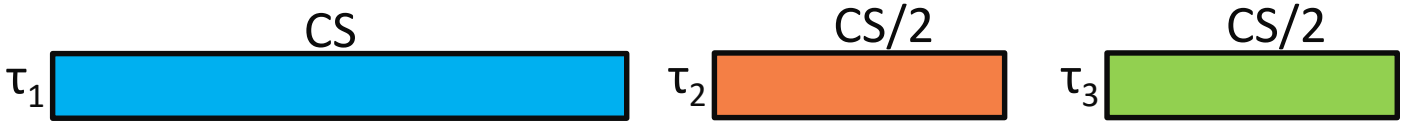


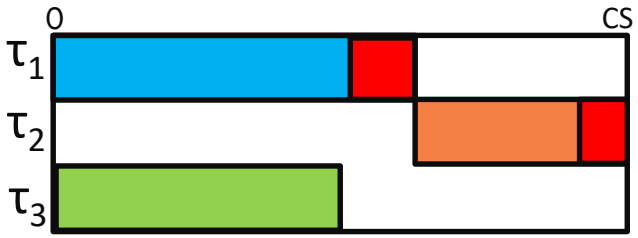


Optimized Cache Allocation

Trading between Intra- and Inter-task cache interference may improve schedulability

- Example: Task set $\Gamma=\{\tau_1, \tau_2, \tau_3\}$ and a cache of total size CS.

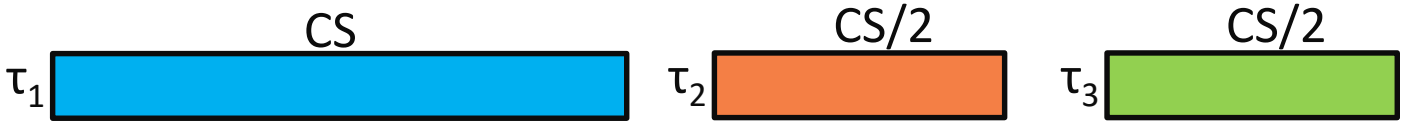


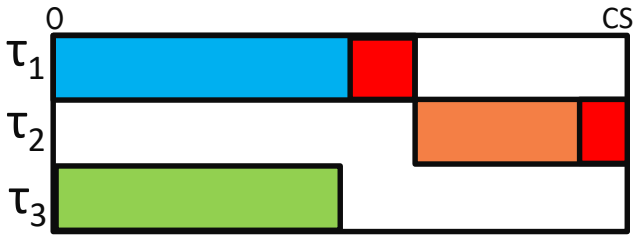


Optimized Cache Allocation

Trading between Intra- and Inter-task cache interference may improve schedulability

- Example: Task set $\Gamma=\{\tau_1, \tau_2, \tau_3\}$ and a cache of total size CS.





Optimized Cache Allocation

Task set **schedulability** may **improve** if the **reduction** in the **inter-task cache interference** between τ_1 and τ_2 **dominate** the **increase in the intra-task cache interference** of τ_1 and τ_2

Contributions

1. **Cache Coloring** approach to optimize task **layout** in memory
2. **Bounding** intra- and inter-task **cache interference** when using cache coloring
3. **Cache Interference**-Aware **WCRT** Analysis
4. **Simulated Annealing** Algorithm to optimize cache color **assignment** of tasks

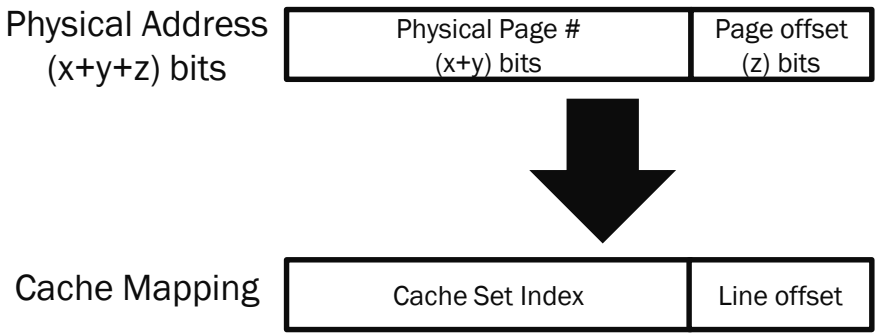
What is Cache Coloring?

Cache Coloring to Optimize Task layout in Memory

- **OS-level software** technique to **control** the **mapping** of tasks in cache.

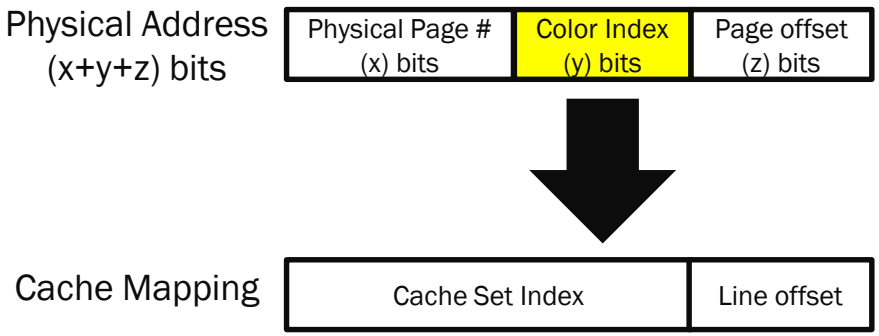
Cache Coloring to Optimize Task layout in Memory

- **OS-level software** technique to **control** the **mapping** of tasks in cache.
- **Lies** in the **mapping** between **physical address** and cache **entries**



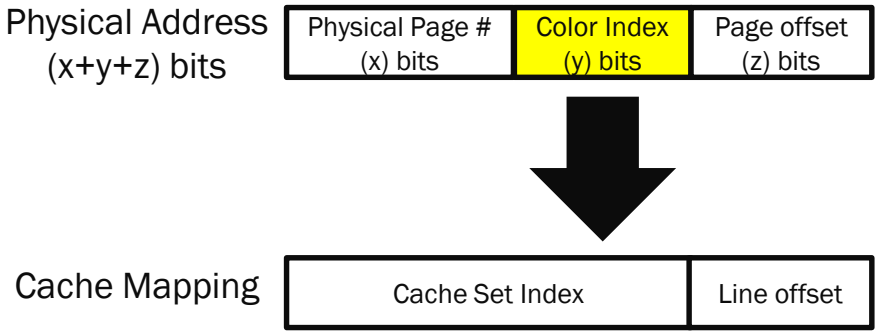
Cache Coloring to Optimize Task layout in Memory

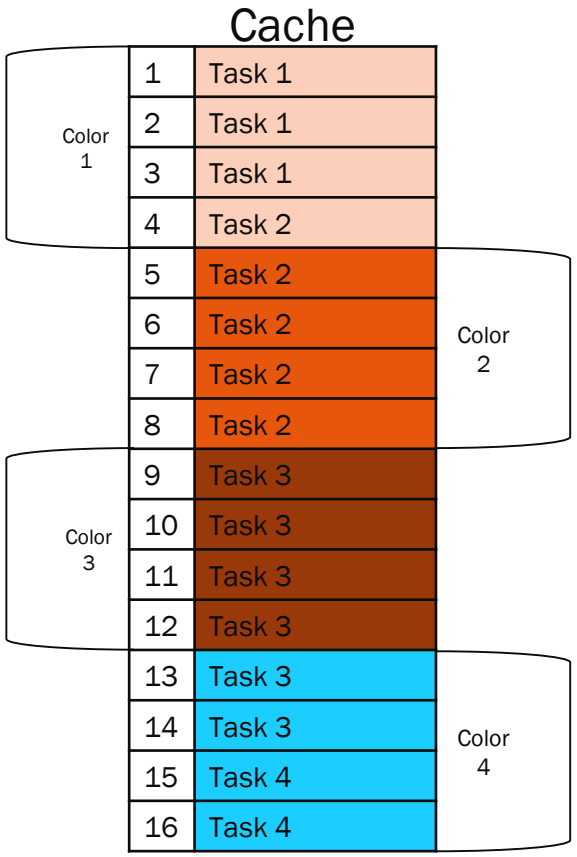
- **OS-level software** technique to **control** the **mapping** of tasks in cache.
- **Lies** in the **mapping** between **physical address** and cache **entries**



Cache Coloring to Optimize Task layout in Memory

- **OS-level software** technique to **control** the **mapping** of tasks in cache.
- **Lies** in the **mapping** between **physical address** and cache **entries**





Bounding Intra- and Inter-task Cache Interference

CISTER
Research Center in
Real-Time & Embedded
Computing Systems

isep Instituto Superior de Engenharia do Porto

RTNS 2018

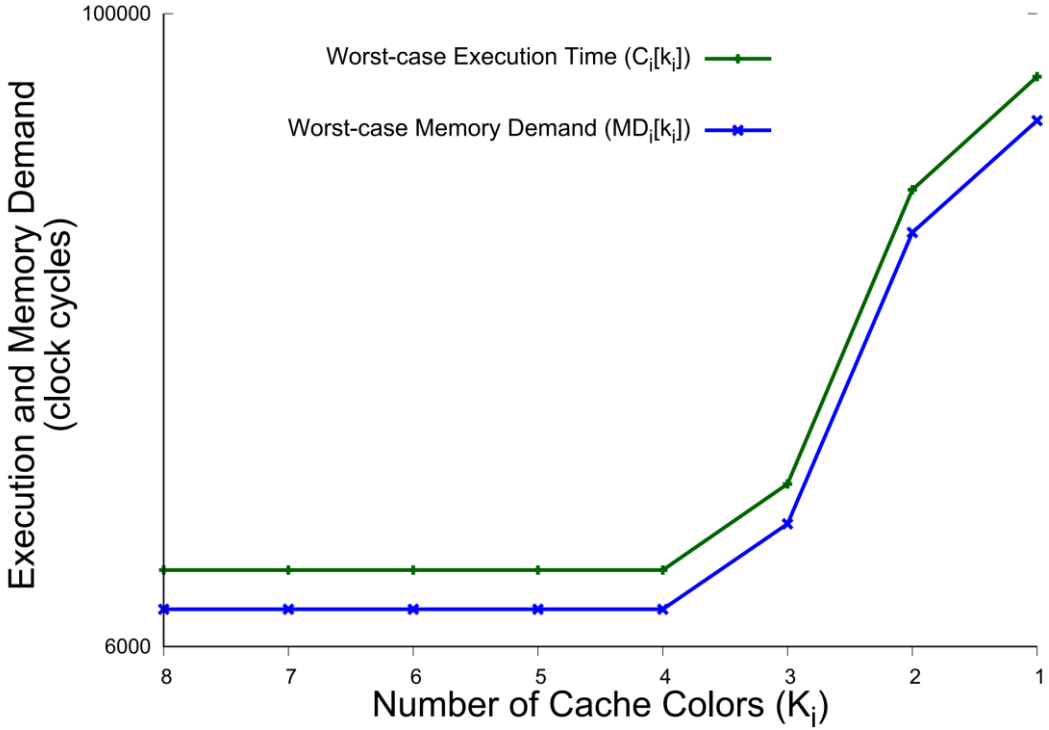
Bounding Intra-task Cache Interference

- **Intra-task** cache interference depends on the cache **space** or the **number** of cache colors **assigned** to a **task**.

Bounding Intra-task Cache Interference

- **Intra-task** cache interference depends on the cache **space** or the **number** of cache colors **assigned** to a **task**.

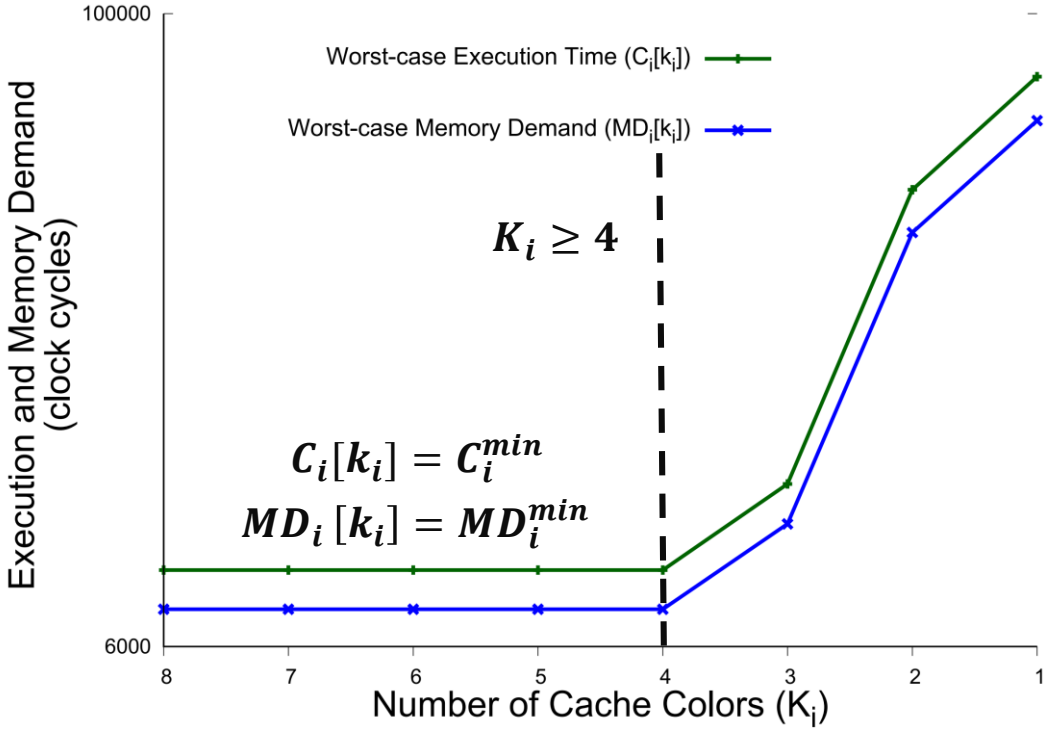
Variation in the **worst-case execution time** and the **worst-case memory demand** of the benchmark **fdct** of the Mälardalen benchmark suite



Bounding Intra-task Cache Interference

- **Intra-task** cache interference depends on the cache **space** or the **number** of cache colors **assigned** to a **task**.

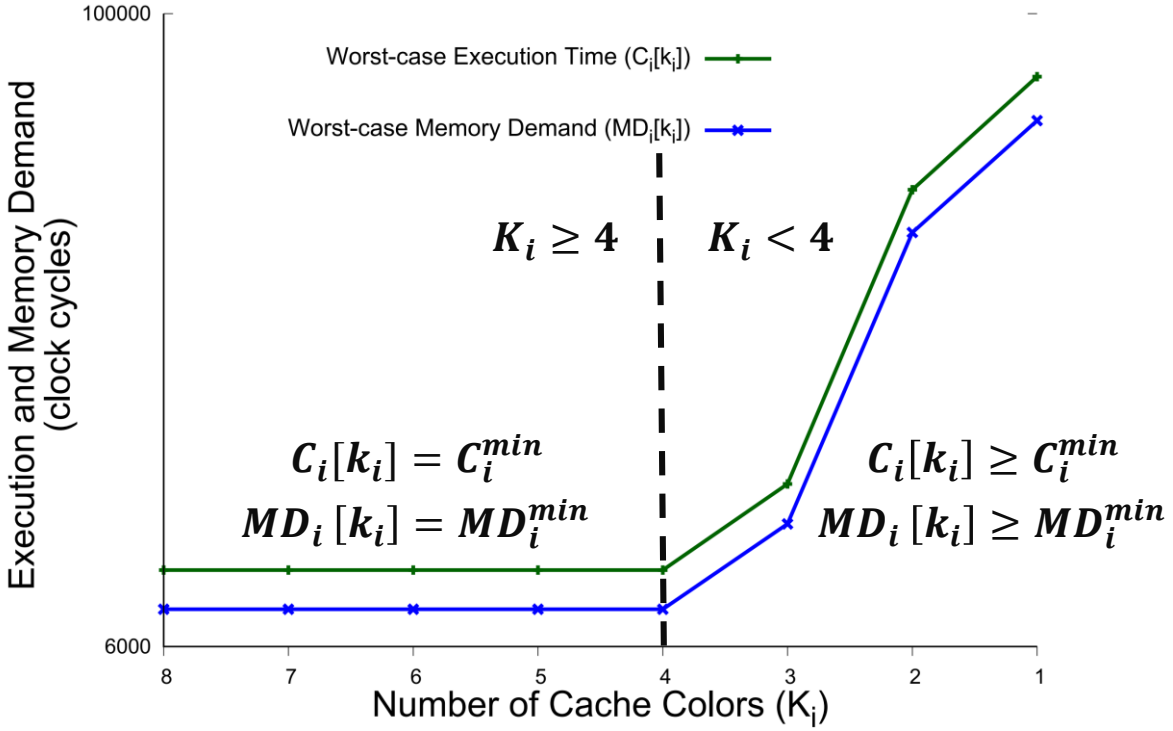
Variation in the **worst-case execution time** and the **worst-case memory demand** of the benchmark **fdct** of the Mälardalen benchmark suite



Bounding Intra-task Cache Interference

- **Intra-task** cache interference depends on the cache **space** or the **number** of cache colors **assigned** to a **task**.

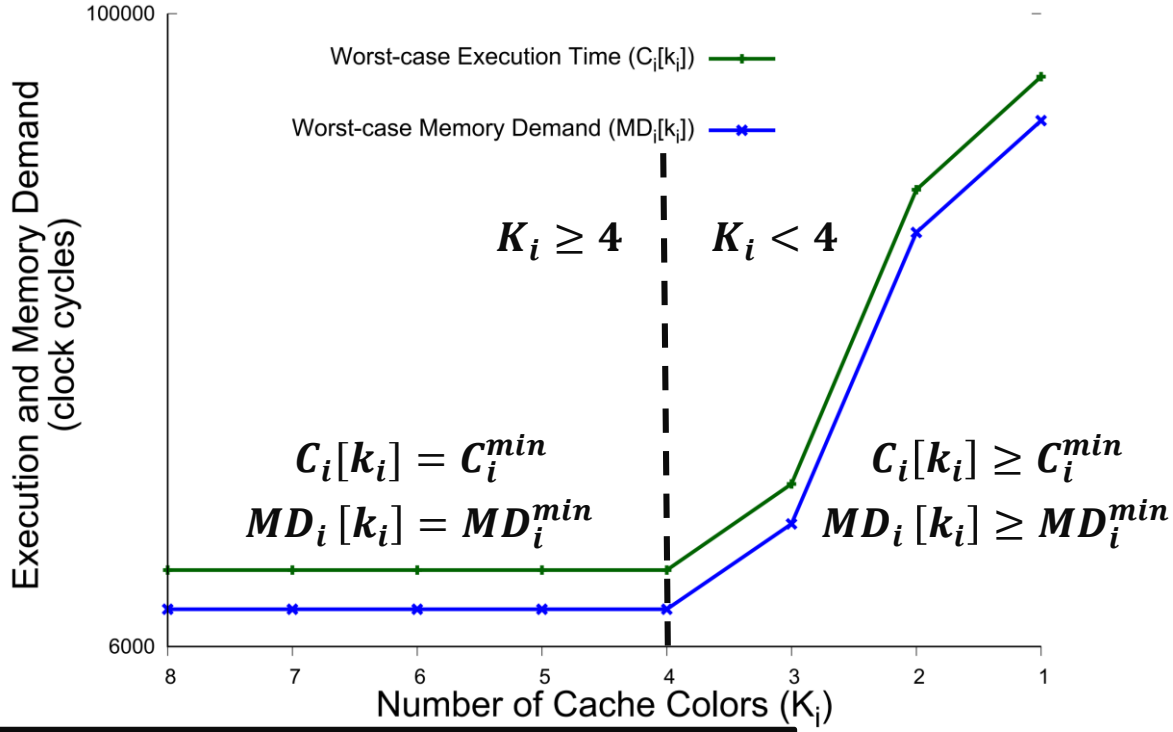
Variation in the **worst-case execution time** and the **worst-case memory demand** of the benchmark **fdct** of the Mälardalen benchmark suite



Bounding Intra-task Cache Interference

- **Intra-task** cache interference depends on the cache **space** or the **number** of cache colors **assigned** to a **task**.

Variation in the **worst-case execution time** and the **worst-case memory demand** of the benchmark **fdct** of the Mälardalen benchmark suite



$$CI_i^{intra,k_i} = MD_i[k_i] - MD_i^{min}$$

Bounding Inter-task Cache Interference

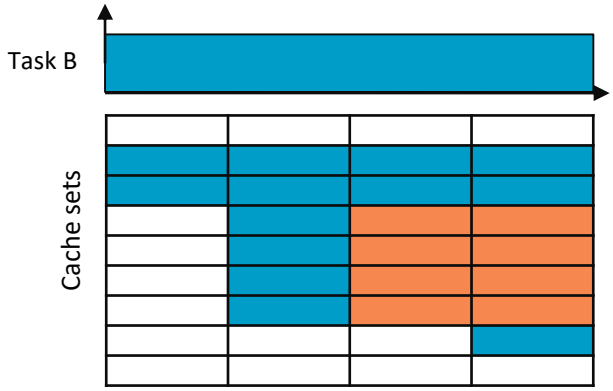
Bounding Inter-task Cache Interference

- Inter-task Cache Interference due to CRPDs
- Inter-task Cache Interference due to CPROs

How are CRPDs Calculated?
What are Useful Cache Blocks (UCBs)?

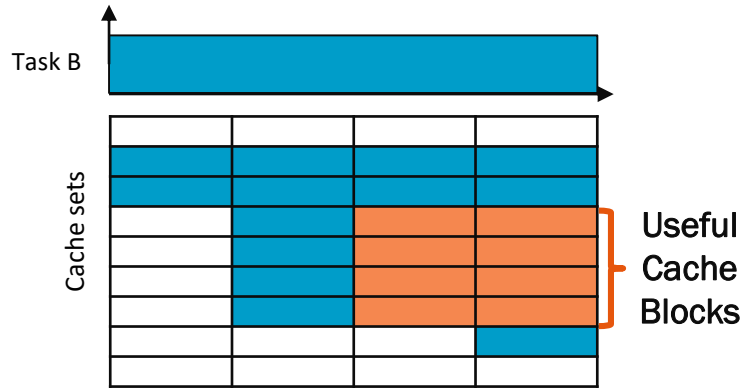
```
fct_A
{
   some code;
   int i = 0;
   while(i < 2)
   {
      call fct_B( x );
      ++i;
   }
   some code;
}
```

×2



How are CRPDs Calculated?
What are Useful Cache Blocks (UCBs)?

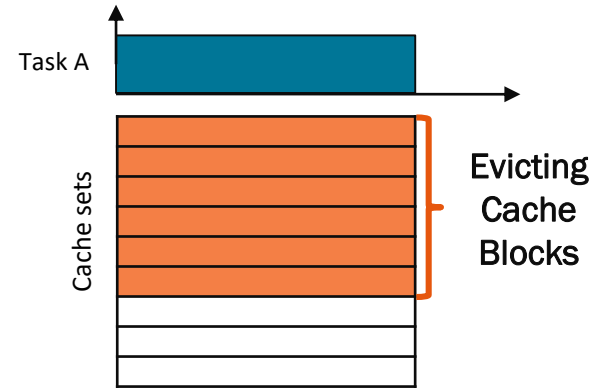
```
fct_A
{
   some code;
   int i = 0;
   while(i < 2)
   {
      call fct_B( x );
      ++i;
   }
   some code;
}
```



- **Useful Cache Blocks (UCB)** = cache blocks that are used more than once during the execution of a task without eviction

How are CRPDs Calculated?
What are Evicting Cache Blocks (ECBs)?

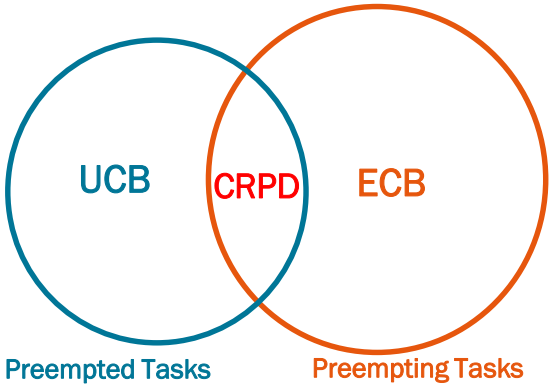
```
fct_C
{
    some code;
    .
    .
    .
    more code;
}
```



- **All** cache blocks used by a task during its execution are called Evicting Cache Blocks (ECB)

How are CRPDs Calculated? Use Both Preempting and Preempted Task

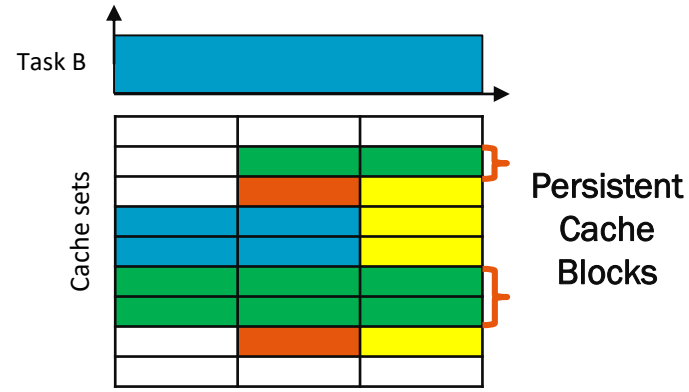
- Number of **UCBs** of the **preempted** task upper bound the CRPD it can **suffer**
- Number of **ECBs** of the **preempting** task upper bound the CRPD it can **cause**



CRPD = **intersection** between the **UCBs** of the **preempted task** and the **ECBs** of the **preempting tasks**

How are CPROs Calculated?
What are Persistent Cache Blocks (PCBs)?

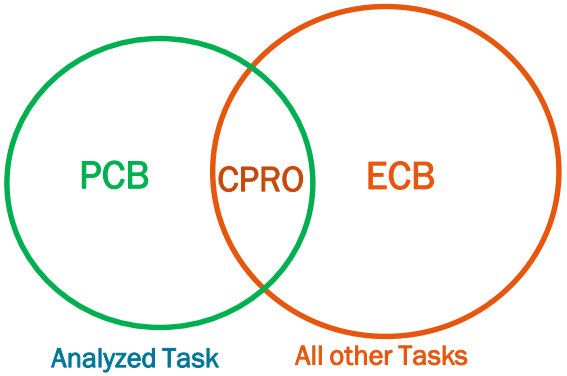
```
fct_C
{
    some code;
    .
    .
    .
    more code;
}
```



Cache blocks that are **once** loaded in the cache and will **never be evicted/invalidated** by the task itself when it executes in isolation are called Persistent Cache Blocks (PCBs)

How are CPROs Calculated?

- Number of **PCBs** of a task upper bounds the **CPRO** it can **suffer**.
- Number of **ECBs** of a task upper bounds the cache **evictions** it can **cause**
- **The intersection upper-bounds the CPRO**



CPRO= **intersection** between the **PCBs** of the **task** **under analysis** and the **ECBs** of all the other **tasks**

Problem with SOA CRPD/CPRO Calculation under Cache Coloring

- Several tasks **sharing** a cache color, actual set of **ECBs/UCBs/PCBs** of tasks may **not** be known.

Problem with SOA CRPD/CPRO Calculation under Cache Coloring

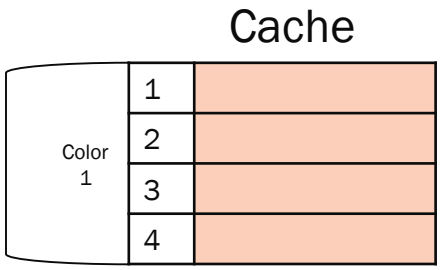
- Several tasks **sharing** a cache color, actual set of **ECBs/UCBs/PCBs** of tasks may **not** be known.

τ_1

For $k_1=1$, $|UCB_1|=2$

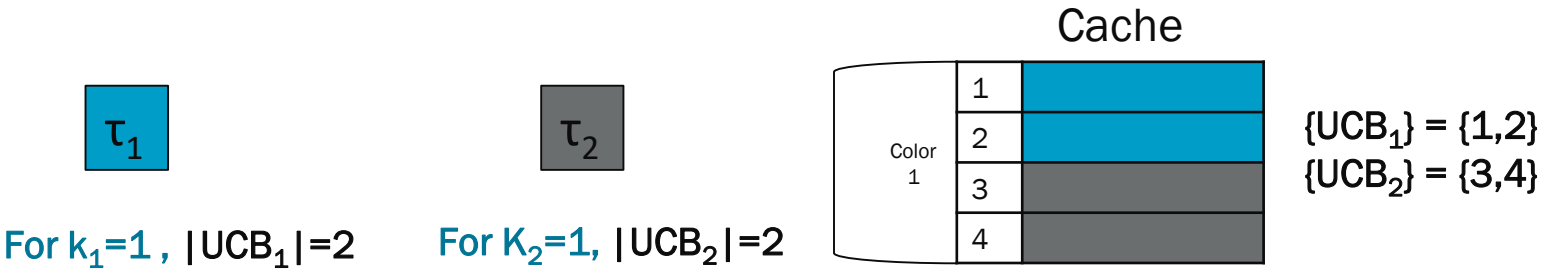
τ_2

For $K_2=1$, $|UCB_2|=2$



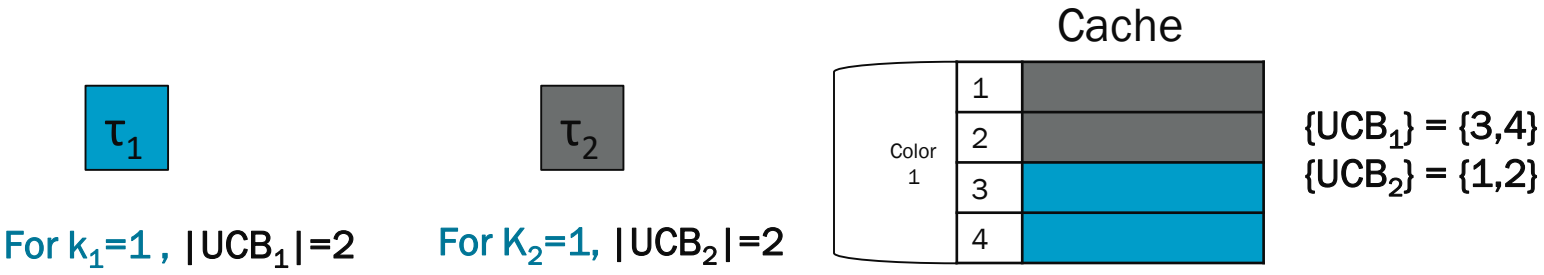
Problem with SOA CRPD/CPRO Calculation under Cache Coloring

- Several tasks **sharing** a cache color, actual set of **ECBs/UCBs/PCBs** of tasks may **not** be known.



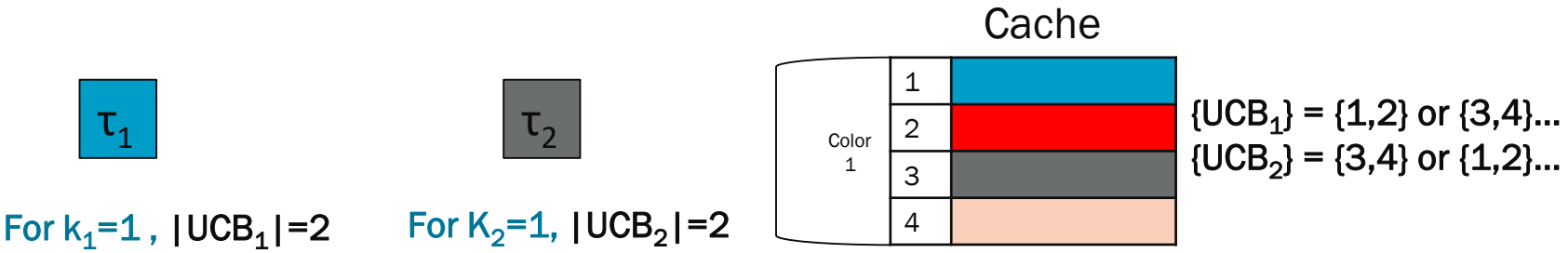
Problem with SOA CRPD/CPRO Calculation under Cache Coloring

- Several tasks **sharing** a cache color, actual set of **ECBs/UCBs/PCBs** of tasks may **not** be known.



Problem with SOA CRPD/CPRO Calculation under Cache Coloring

- Several tasks **sharing** a cache color, actual set of **ECBs/UCBs/PCBs** of tasks may **not** be known.



Problem with SOA CRPD/CPRO Calculation under Cache Coloring

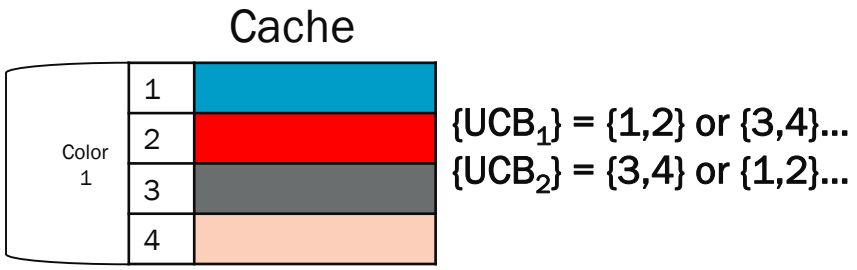
- Several tasks **sharing** a cache color, actual set of **ECBs/UCBs/PCBs** of tasks may **not** be known.

τ_1

For $k_1=1$, $|UCB_1|=2$

τ_2

For $K_2=1$, $|UCB_2|=2$



Different set of **ECBs/UCBs/PCBs** of tasks may lead to different **pessimistic/optimistic** value of **CRPD/CPRO**

Bounding Inter-task Cache Interference due to CRPDs

- Task τ_j may only **evict** cache content of τ_i if they use the **same** cache colors
- The **number** of UCBs/PCBs for a given **cache color assignment** are **known**.

Bounding Inter-task Cache Interference due to CRPDs

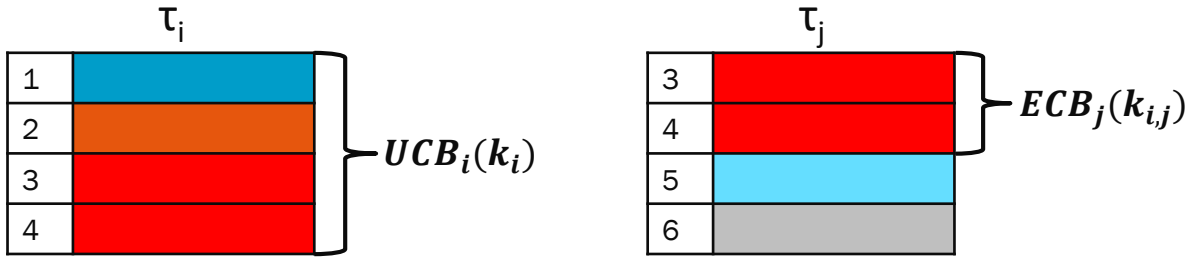
- Task τ_j may only **evict** cache content of τ_i if they use the **same** cache colors
- The **number** of UCBs/PCBs for a given **cache color assignment** are **known**.

Bounding Inter-task Cache Interference due to CRPDs

- Task τ_j may only **evict** cache content of τ_i if they use the **same** cache colors
- The **number** of UCBs/PCBs for a given **cache color assignment** are **known**.

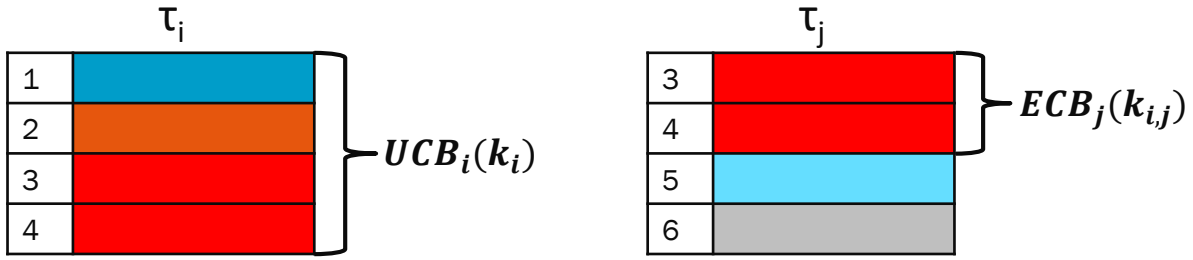
Bounding Inter-task Cache Interference due to CRPDs

- Task τ_j may only **evict** cache content of τ_i if they use the **same** cache colors
- The **number** of UCBs/PCBs for a given **cache color assignment** are **known**.



Bounding Inter-task Cache Interference due to CRPDs

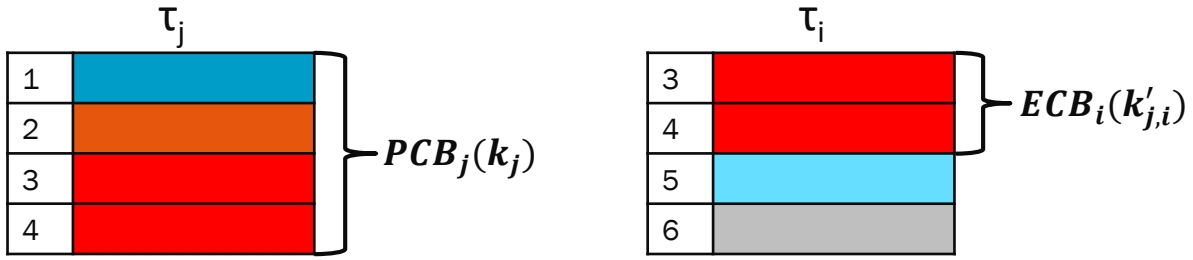
- Task τ_j may only **evict** cache content of τ_i if they use the **same** cache colors
- The **number** of UCBs/PCBs for a given **cache color assignment** are **known**.



$$CI_{i,j}^{inter,\gamma} = min\{UCB_i(k_i), ECB_j(k_{i,j})\}$$

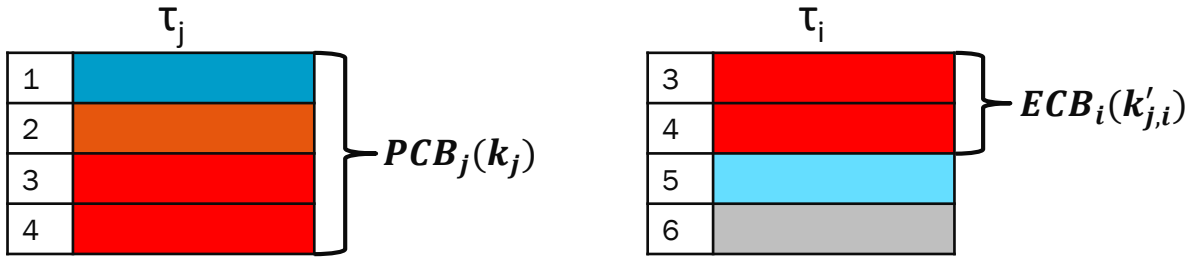
Bounding Inter-task Cache Interference due to CPROs

- Task τ_j may only **evict** cache content of τ_i if they use the **same** cache colors
- The **number** of UCBs/PCBs for a given **cache color assignment** are **known**.



Bounding Inter-task Cache Interference due to CPROs

- Task τ_j may only **evict** cache content of τ_i if they use the **same** cache colors
- The **number** of UCBs/PCBs for a given **cache color assignment** are **known**.



$$CI_{i,j}^{inter,\rho} = min\{PCB_j(k_j), ECB_i(k'_{j,i})\}$$

Cache Interference-Aware WCRT Analysis

CISTER
Research Center in
Real-Time & Embedded
Computing Systems

isep Instituto Superior de Engenharia do Porto

RTNS 2018

Cache Interference-Aware WCRT Analysis

Worst-case execution time of task τ_i in isolation assuming the cache space allocated to τ_i is equal to its size in main memory

Intra-task cache interference suffered by the higher priority task τ_j

$$R_i = \overbrace{C_i^{min}}^{} + \underbrace{CI_i^{intra,k_i}}_{} + \sum_{\forall j \in \text{hp}(i)} \left(\min \left\{ \left\lceil \frac{R_i}{T_j} \right\rceil \left(C_j^{min} + \overbrace{CI_j^{intra,k_j}}^{} \right) ; \left\lceil \frac{R_i}{T_j} \right\rceil PD_j \right. \right.$$

$$\left. \left. + \hat{MD}_j(R_i) + \underbrace{CI_{j,i}^{inter,\rho}(R_i)}_{} \right\} + \underbrace{CI_{i,j}^{inter,\gamma}(R_i)}_{} \right)$$

Intra-task cache interference suffered by task τ_i

Total Inter-task cache interference due to CPRO during the response time of task τ_i

Total Inter-task cache interference due to CRPD during the response time of task τ_i

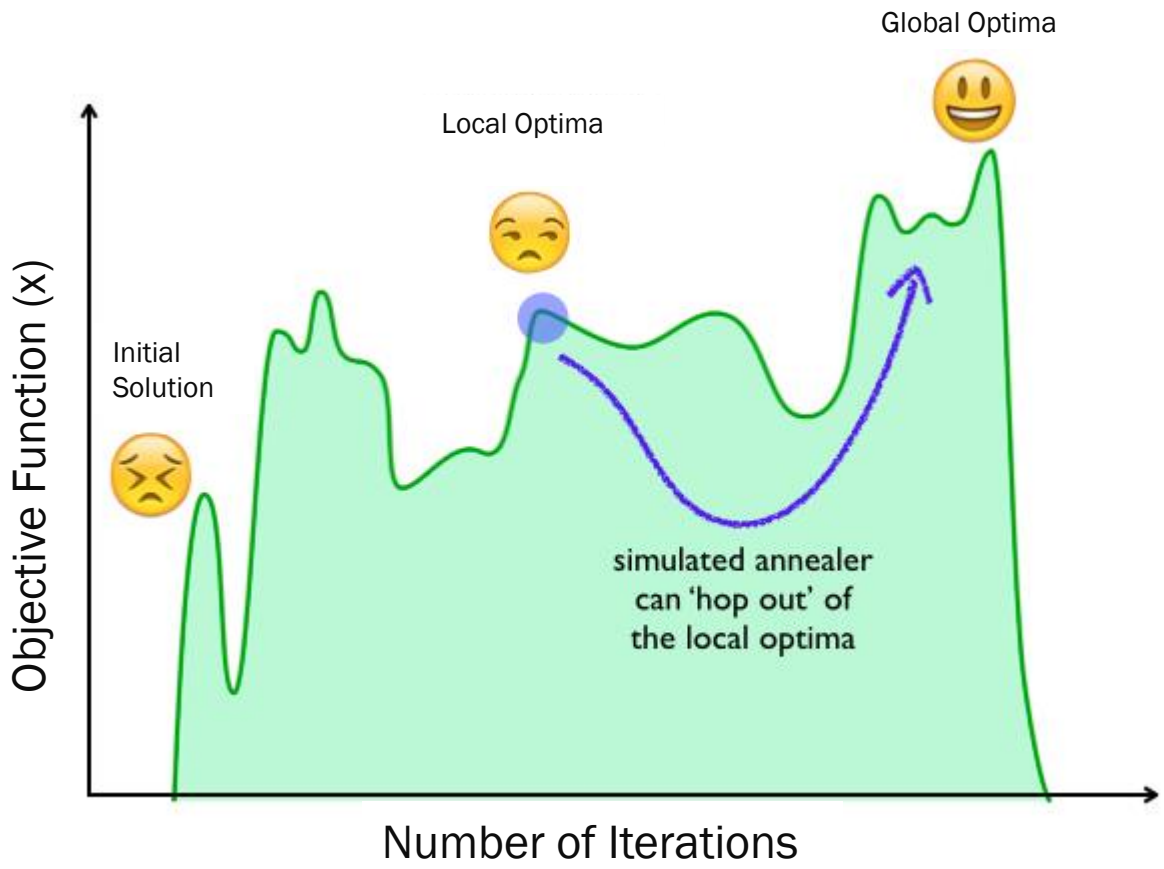
Simulated Annealing to Optimize Cache Color Assignment of Tasks

CISTER
Research Center in
Real-Time & Embedded
Computing Systems

isep Instituto Superior de Engenharia do Porto

RTNS 2018

Simulated Annealing to Optimize Cache Color Assignment of Tasks



Simulated Annealing to Optimize Cache Color Assignment of Tasks

- Algorithm

```
Generate(Ts);
Seq_CacheColorAssignment (Ts);
IF (Isscchedulable(Ts))
Return true;
Else
Call_Simulated_Annealing (Ts);
IF (Isscchedulable(Ts))
Return True;
Else
Return False;

Call_Simulated_Annealing (Ts)
{
While(current_temp>required_temp)
{
Select_Random(re_allocate(),re_size());
Check_newcolorassignment();
current_temp–;
}}
```

Simulated Annealing to Optimize Cache Color Assignment of Tasks

- Algorithm

```
Generate(Ts);
Seq_CacheColorAssignment (Ts);
IF (Isscchedulable(Ts))
Return true;
Else
Call_Simulated_Annealing (Ts);
IF (Isscchedulable(Ts))
Return True;
Else
Return False;

Call_Simulated_Annealing (Ts)
{
While(current_temp>required_temp)
{
Select_Random(re_allocate(),re_size());
Check_newcolorassignment();
current_temp–;
}}
```

Simulated Annealing to Optimize Cache Color Assignment of Tasks

- Algorithm

```
Generate(Ts);
Seq_CacheColorAssignment (Ts);
IF (Isscchedulable(Ts))
Return true;
Else
Call_Simulated_Annealing (Ts);
IF (Isscchedulable(Ts))
Return True;
Else
Return False;

Call_Simulated_Annealing (Ts)
{
While(current_temp>required_temp)
{
Select_Random(re_allocate(),re_size());
Check_newcolorassignment();
current_temp–;
}}
```

Simulated Annealing to Optimize Cache Color Assignment of Tasks

- Algorithm

```
Generate(Ts);
Seq_CacheColorAssignment (Ts);
IF (Isscchedulable(Ts))
Return true;
Else
Call_Simulated_Annealing (Ts);
IF (Isscchedulable(Ts))
Return True;
Else
Return False;

Call_Simulated_Annealing (Ts)
{
While(current_temp>required_temp)
{
Select_Random(re_allocate(),re_size());
Check_newcolorassignment();
current_temp–;
}}
```

Simulated Annealing to Optimize Cache Color Assignment of Tasks

- Algorithm

```
Generate(Ts);
Seq_CacheColorAssignment (Ts);
IF (Isscchedulable(Ts))
Return true;
Else
Call_Simulated_Annealing (Ts);
IF (Isscchedulable(Ts))
Return True;
Else
Return False;

Call_Simulated_Annealing (Ts)
{
While(current_temp>required_temp)
{
Select_Random(re_allocate(),re_size());
Check_newcolorassignment();
current_temp–;
}}
```

1. *Re_allocate ()*
2. *Re_size()*

Simulated Annealing to Optimize Cache Color Assignment of Tasks

- **Algorithm**

```
Generate(Ts);
Seq_CacheColorAssignment (Ts);
IF (Isscchedulable(Ts))
Return true;
Else
Call_Simulated_Annealing (Ts);
IF (Isscchedulable(Ts))
Return True;
Else
Return False;

Call_Simulated_Annealing (Ts)
{
While(current_temp>required_temp)
{
Select_Random(re_allocate(),re_size());
Check_newcolorassignment();
current_temp–;
}}
```

Cache

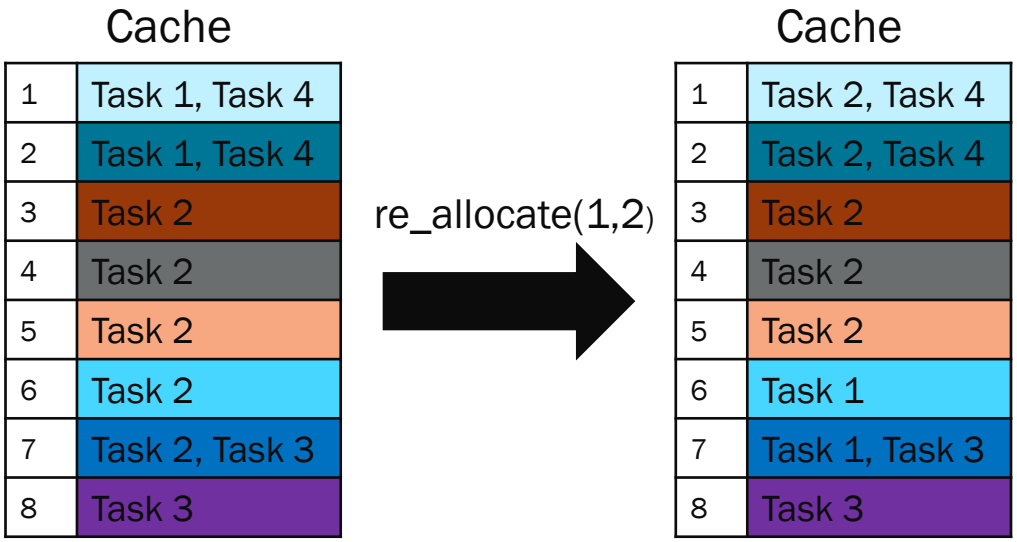
1	Task 1, Task 4
2	Task 1, Task 4
3	Task 2
4	Task 2
5	Task 2
6	Task 2
7	Task 2, Task 3
8	Task 3

Simulated Annealing to Optimize Cache Color Assignment of Tasks

- **Algorithm**

Generate(Ts);
Seq_CacheColorAssignment (Ts);
IF (Isscchedulable(Ts))
Return true;
Else
Call_Simulated_Annealing (Ts);
IF (Isscchedulable(Ts))
Return True;
Else
Return False;

Call_Simulated_Annealing (Ts)
{
While(current_temp>required_temp)
{
Select_Random(re_allocate(),re_size());
Check_newcolorassignment();
current_temp–;
}}

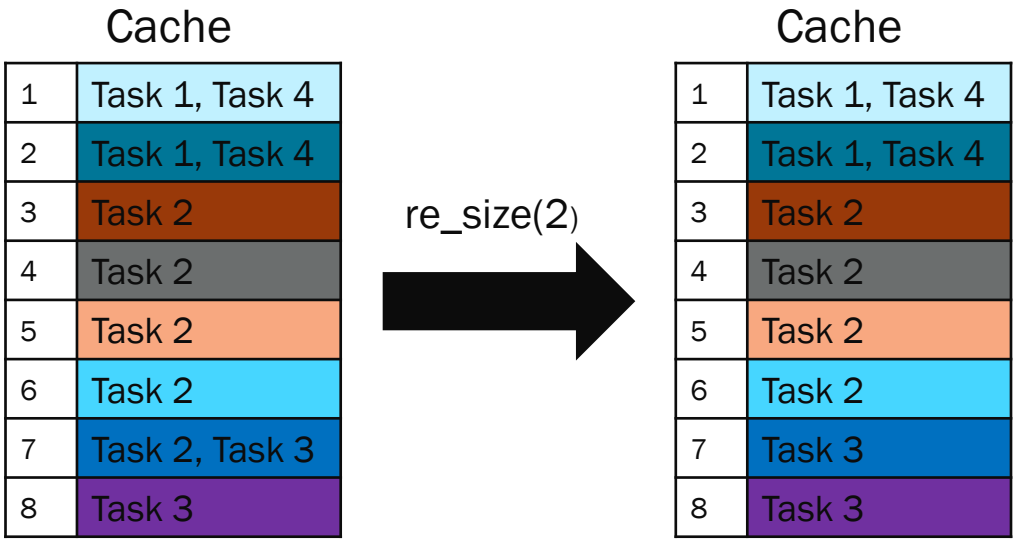


Simulated Annealing to Optimize Cache Color Assignment of Tasks

- **Algorithm**

```
Generate(Ts);
Seq_CacheColorAssignment (Ts);
IF (Isscchedulable(Ts))
Return true;
Else
Call_Simulated_Annealing (Ts);
IF (Isscchedulable(Ts))
Return True;
Else
Return False;

Call_Simulated_Annealing (Ts)
{
While(current_temp>required_temp)
{
Select_Random(re_allocate(),re_size());
Check_newcolorassignment();
current_temp–;
}}
```

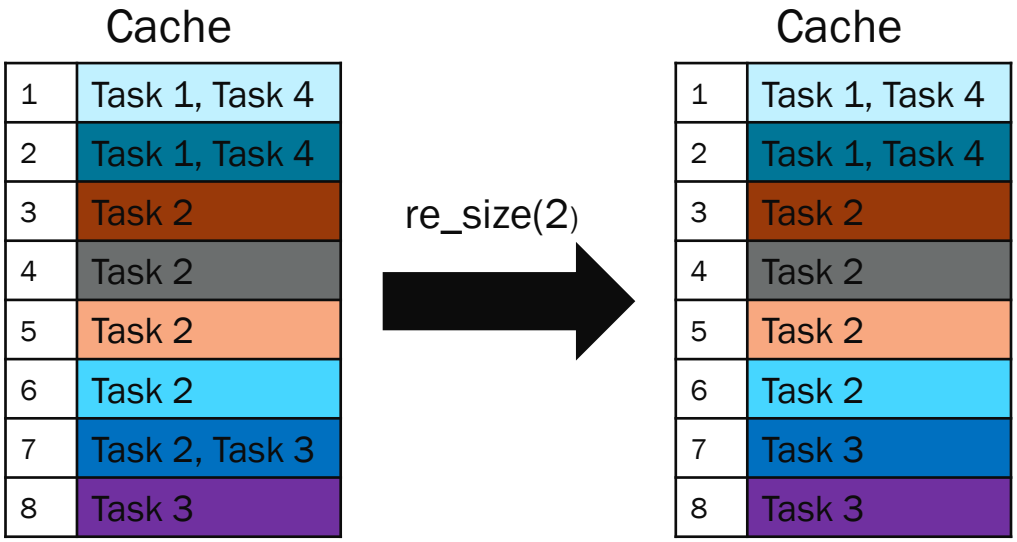


Simulated Annealing to Optimize Cache Color Assignment of Tasks

- **Algorithm**

Generate(Ts);
Seq_CacheColorAssignment (Ts);
IF (Isscchedulable(Ts))
Return true;
Else
Call_Simulated_Annealing (Ts);
IF (Isscchedulable(Ts))
Return True;
Else
Return False;

```
Call_Simulated_Annealing (Ts)
{
While(current_temp>required_temp)
{
Select_Random(re_allocate(),re_size());
Check_newcolorassignment();
current_temp–;
}}
```



Experimental Evaluation: Driving task parameters and Taskset generation

- Heptane static WCET analysis **tool** was used to derive task parameters

https://team.inria.fr/alf/software/heptane/

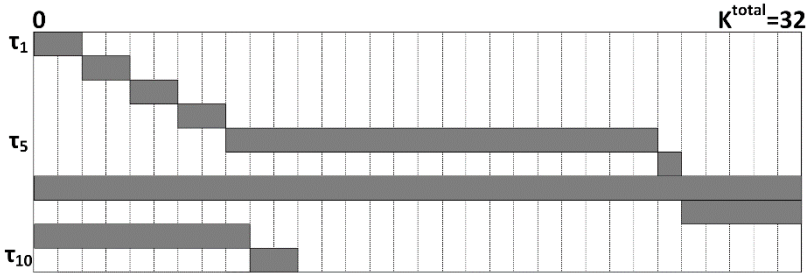
- Experiments were performed using **Mälardalen** benchmark suite.
- A **case study** experiment and **empirical** evaluation using large number of **task sets.**
- **Comparison** between the **proposed** and **SOA** approaches was performed by **varying** different parameters.

Experimental Evaluation: Case Study Experiment

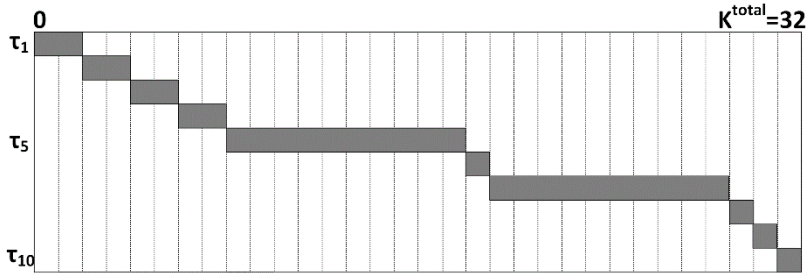
Name	$C_i[k_i]$	PD_i	$MD_i[k_i]$	k_i	T_i
minmax	2522	122	2400	2	14315
lcdnum	3440	984	2740	2	73143
cnt	10090	7191	3818	2	85816
ns	30149	28149	6172	2	169744
statemate	43344	10586	35257	18	636613
insertsort	7574	5974	2343	1	734873
nsichneu	316409	22009	294400	32	1889824
qurt	26141	9241	17713	5	2899034
fft	157880	123681	45816	9	6550339
bsort100	712289	710289	90893	2	267271122

Experimental Evaluation: Case Study Experiment

Name	$C_i[k_i]$	PD_i	$MD_i[k_i]$	k_i	T_i
minmax	2522	122	2400	2	14315
lcdnum	3440	984	2740	2	73143
cnt	10090	7191	3818	2	85816
ns	30149	28149	6172	2	169744
statemate	43344	10586	35257	18	636613
insertsort	7574	5974	2343	1	734873
nsichneu	316409	22009	294400	32	1889824
qurt	26141	9241	17713	5	2899034
fft	157880	123681	45816	9	6550339
bsort100	712289	710289	90893	2	267271122



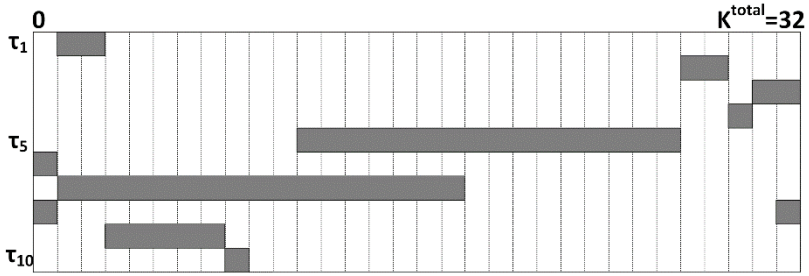
Sequential Cache Color Assignment



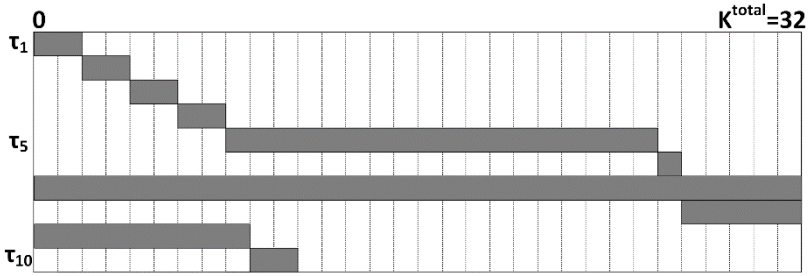
Full Cache Partitioning

Experimental Evaluation: Case Study Experiment

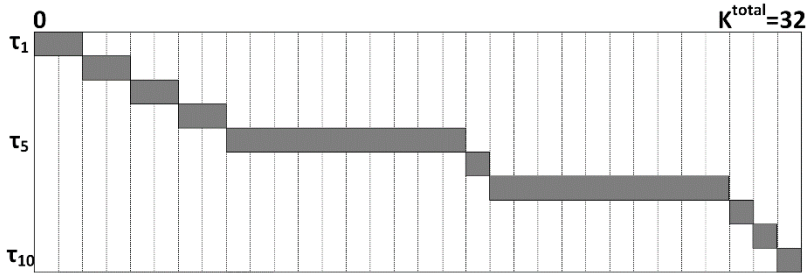
Name	$C_i[k_i]$	PD_i	$MD_i[k_i]$	k_i	T_i
minmax	2522	122	2400	2	14315
lcdnum	3440	984	2740	2	73143
cnt	10090	7191	3818	2	85816
ns	30149	28149	6172	2	169744
statemate	43344	10586	35257	18	636613
insertsort	7574	5974	2343	1	734873
nsichneu	316409	22009	294400	32	1889824
qurt	26141	9241	17713	5	2899034
fft	157880	123681	45816	9	6550339
bsort100	712289	710289	90893	2	267271122



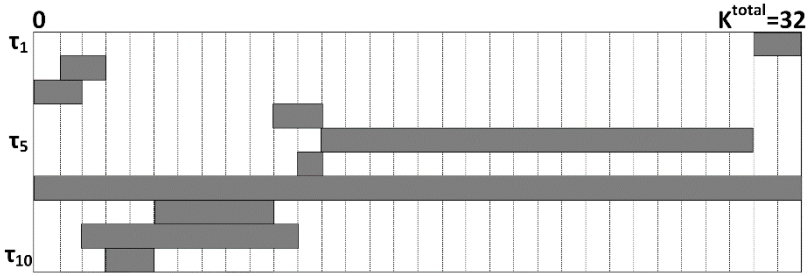
SA-based Cache Color Assignment with re-sizing



Sequential Cache Color Assignment



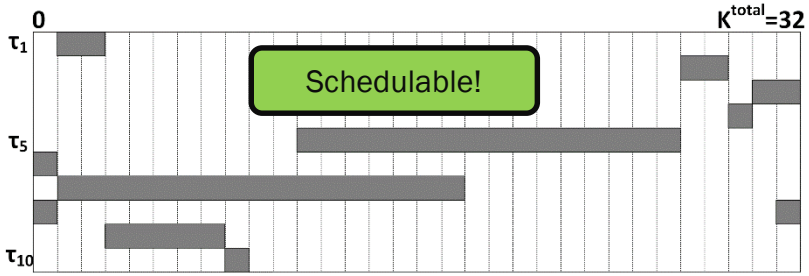
Full Cache Partitioning



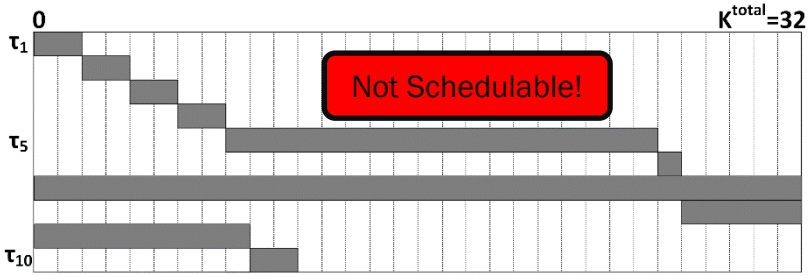
SA-based Cache Color Assignment NO re-sizing

Experimental Evaluation: Case Study Experiment

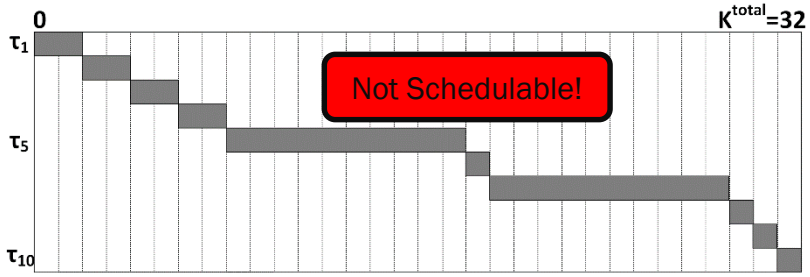
Name	$C_i[k_i]$	PD_i	$MD_i[k_i]$	k_i	T_i
minmax	2522	122	2400	2	14315
lcdnum	3440	984	2740	2	73143
cnt	10090	7191	3818	2	85816
ns	30149	28149	6172	2	169744
statemate	43344	10586	35257	18	636613
insertsort	7574	5974	2343	1	734873
nsichneu	316409	22009	294400	32	1889824
qurt	26141	9241	17713	5	2899034
fft	157880	123681	45816	9	6550339
bsort100	712289	710289	90893	2	267271122



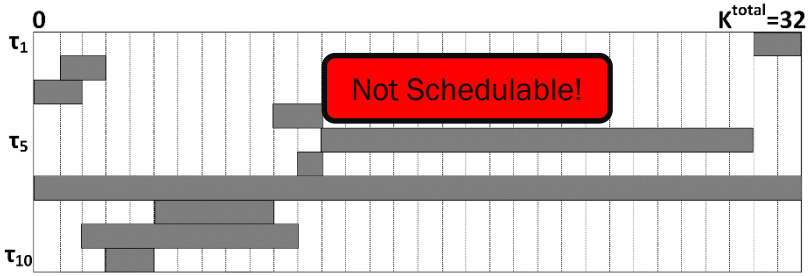
SA-based Cache Color Assignment with re-sizing



Sequential Cache Color Assignment

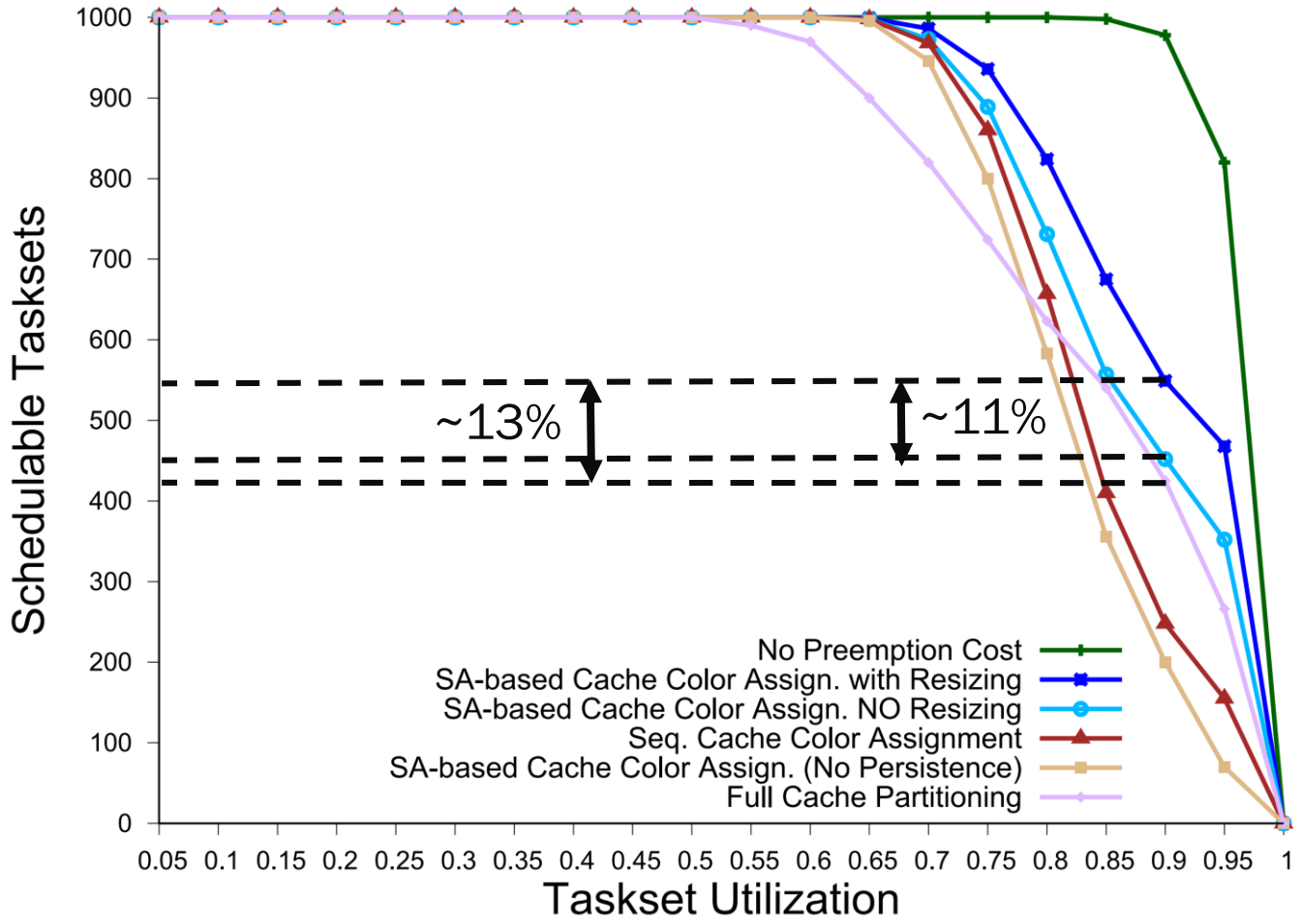


Full Cache Partitioning

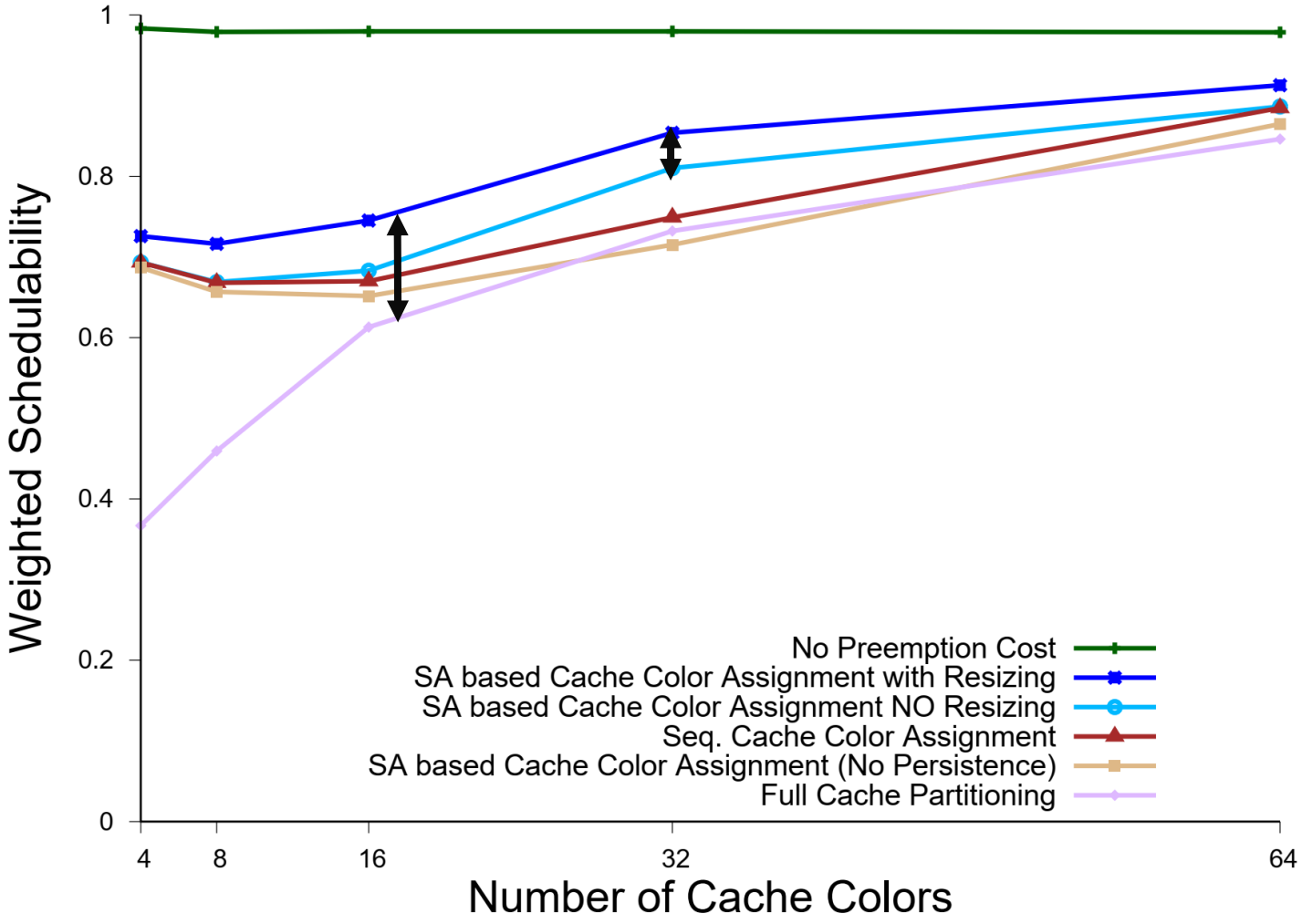


SA-based Cache Color Assignment NO re-sizing

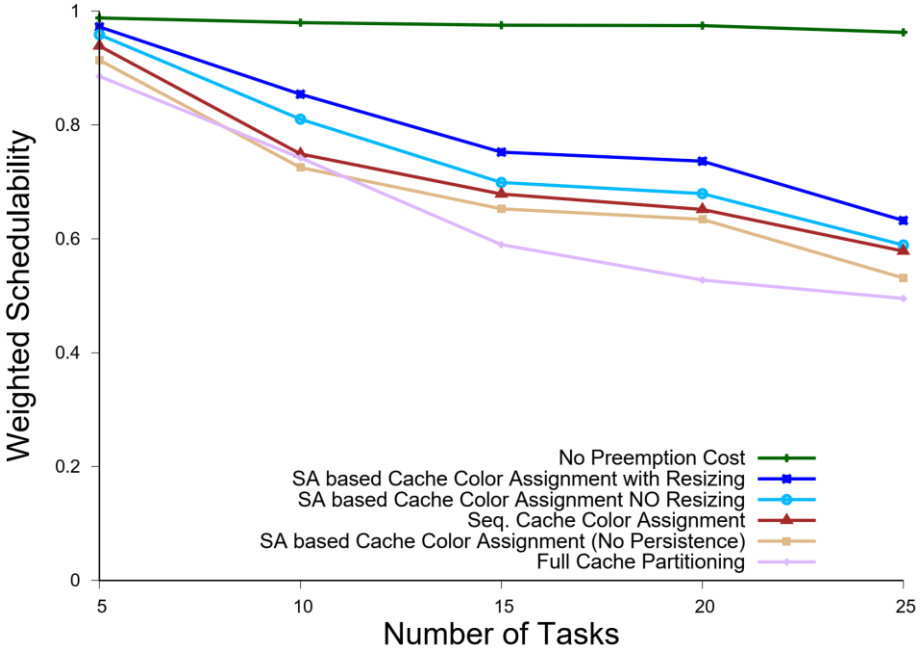
Experimental Evaluation: Varying Core Utilization

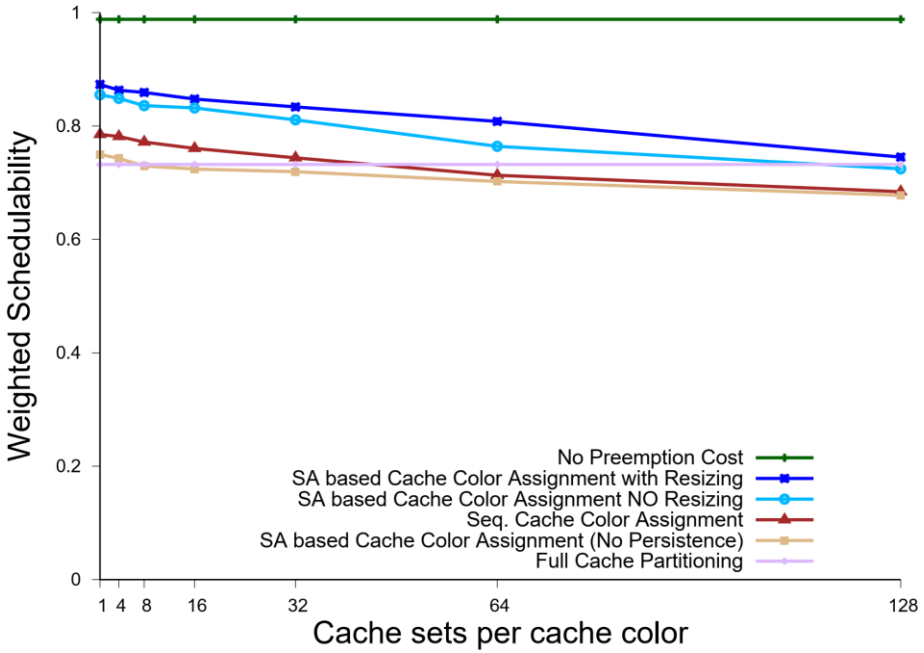


Experimental Evaluation: Varying Number of Cache Colors (or Cache size)



Experimental Evaluation: Varying Number of Tasks, Cache set per Cache Color





Conclusion and Future Work

Conclusion

- **Intra**- and **inter-task** cache interference is **interrelated** and **balancing** their **contribution** to tasks **WCRT** may result in **improving** task set **schedulability**.
- **Cache coloring** approach to **optimize** task **layout**.
- **Bounded** the **intra-** and **inter-task** cache interference **under** cache coloring.
- **Simulated Annealing** approach to optimize cache **color assignment** of tasks.
- Experimental **evaluation** showing the effectiveness our approach

Future Work

- Presented work assumed a **direct** mapped cache, in future we plan to **extend** it to N-way **set associative** caches.
- We also aim to extent this analysis to **shared** cache in **multicore** platforms.

Conclusion and Future Work

Conclusion

- **Intra**- and **inter-task** cache interference is **interrelated** and **balancing** their **contribution** to tasks **WCRT** may result in **improving** task set **schedulability**.
- **Cache coloring** approach to **optimize** task **layout**.
- **Bounded** the **intra-** and **inter-task** cache interference **under** cache coloring.
- **Simulated Annealing** approach to optimize cache **color assignment** of tasks.
- Experimental **evaluation** showing the effectiveness our approach

Future Work

- Presented work assumed a **direct** mapped cache, in future we plan to **extend** it to N-way **set associative** caches.
- We also aim to extent this analysis to **shared** cache in **multicore** platforms.

Thank You... ☺

CISTER
Research Center in
Real-Time & Embedded
Computing Systems

isep Instituto Superior de **Engenharia** do Porto

RTNS 2018