Evaluation of Cache Partitioning for Hard Real-Time Systems

S. Altmeyer*, R. Douma*, W. Lunniss†, R. Davis†

ECRTS 2014 - Madrid

* University of Amsterdam

† University of York

Cache Partitioning

Context: pre-emptively scheduled multi-task systems with caches

Idea: each task is assigned a dedicated part of the cache

Pros:

- predictable system
- less dynamic behaviour
- standard WCET analysis applicable

Cons:

- higher (non-preempted) execution times
- implementation overhead

Commonly accepted assumption:

(i) Cache partitioning increases predictability

(ii) Predictability is good for real-time systems

$\Rightarrow$ Cache partitioning is good!

We want to know:

- **How useful is cache partitioning?**
- **When to use cache partitioning?**
- **Which system/task parameters influence usefulness?**

Differences to related work:

1. based on analytical approaches
   (as used for safety-critical systems)
   - instead of measurements-based evaluation [3]
     or simplified execution time models [4]
2. using optimal* cache partitioning (wrt. schedulability)
   - instead of focussing on implementation details [8, 10, 11]
3. comparison with state-of-the-art pre-emption cost analysis
   - instead of comparisons with systems without caches [12]

* Terms and conditions may apply

Optimal Cache Partitioning Algorithm

Optimality with respect to schedulability
(if there exists a cache partitioning so that the task set is schedulable, the algorithm will find it)

Under the following **conditions**:

1. sustainable schedulability test [1]
   (a task set deemed schedulable, will remain so if parameters improve)
2. monotonic execution times
   (the execution times does not increase when the size of the assigned partition increases)

Optimal Cache Partitioning Algorithm

A **cache partitioning** P is a tuple of integers denoting for each task τ_i, the size p_i of its allocated partition:

$$P = (p_1, p_2 \ldots, p_n) : \underbrace{\mathbb{N} \times \ldots \times \mathbb{N}}_{n}$$

Simple **backtracking** algorithm:

$(\_, \_, \_, \_)$

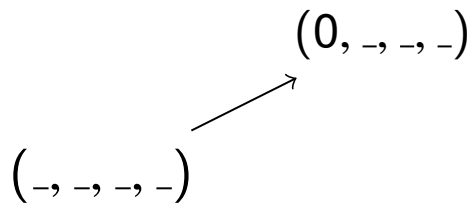
assuming 4 tasks and 4 cache sets

Optimal Cache Partitioning Algorithm

A **cache partitioning** P is a tuple of integers denoting for each task τ_i, the size p_i of its allocated partition:

$$P = (p_1, p_2 \ldots, p_n) : \underbrace{\mathbb{N} \times \ldots \times \mathbb{N}}_{n}$$

Simple **backtracking** algorithm:



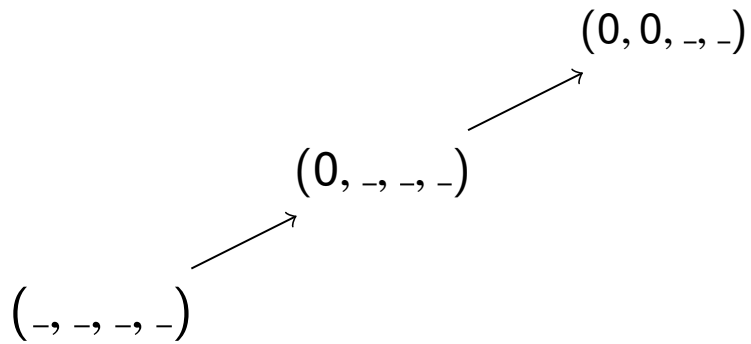
assuming 4 tasks and 4 cache sets

Optimal Cache Partitioning Algorithm

A **cache partitioning** P is a tuple of integers denoting for each task τ_i, the size p_i of its allocated partition:

$$P = (p_1, p_2 \ldots, p_n) : \underbrace{\mathbb{N} \times \ldots \times \mathbb{N}}_{n}$$

Simple **backtracking** algorithm:



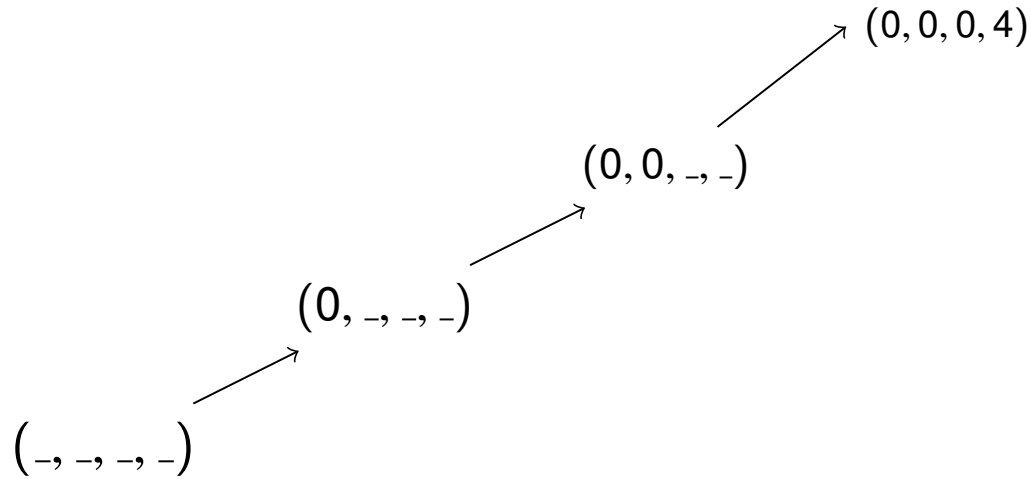
assuming 4 tasks and 4 cache sets

Optimal Cache Partitioning Algorithm

A **cache partitioning** P is a tuple of integers denoting for each task τ_i, the size p_i of its allocated partition:

$$P = (p_1, p_2 \ldots, p_n) : \underbrace{\mathbb{N} \times \ldots \times \mathbb{N}}_{n}$$

Simple **backtracking** algorithm:



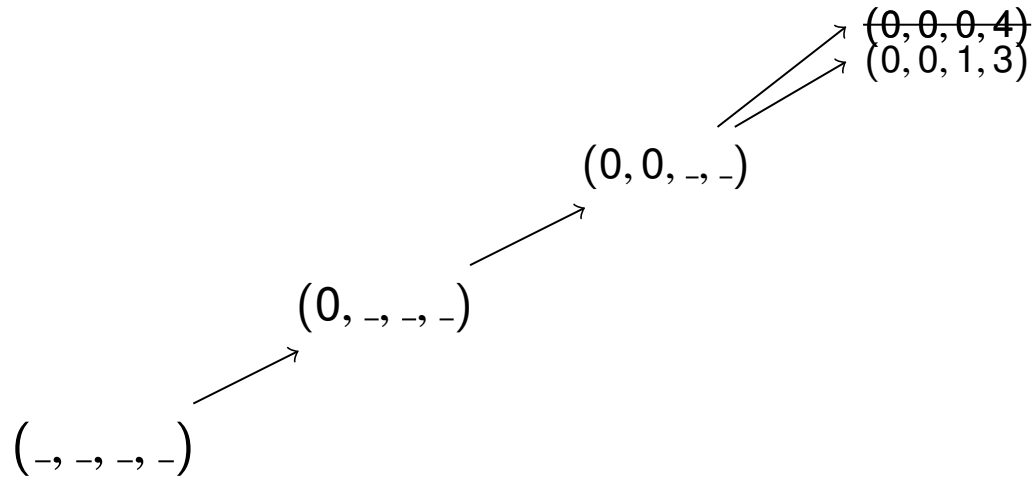
assuming 4 tasks and 4 cache sets

Optimal Cache Partitioning Algorithm

A **cache partitioning** P is a tuple of integers denoting for each task τ_i, the size p_i of its allocated partition:

$$P = (p_1, p_2 \ldots, p_n) : \underbrace{\mathbb{N} \times \ldots \times \mathbb{N}}_{n}$$

Simple **backtracking** algorithm:



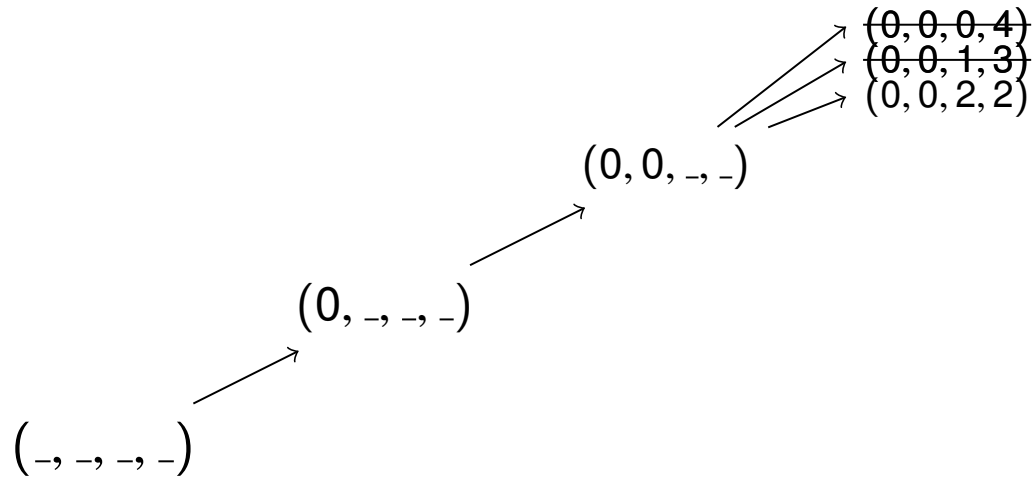
assuming 4 tasks and 4 cache sets

Optimal Cache Partitioning Algorithm

A **cache partitioning** P is a tuple of integers denoting for each task τ_i, the size p_i of its allocated partition:

$$P = (p_1, p_2 \ldots, p_n) : \underbrace{\mathbb{N} \times \ldots \times \mathbb{N}}_{n}$$

Simple **backtracking** algorithm:



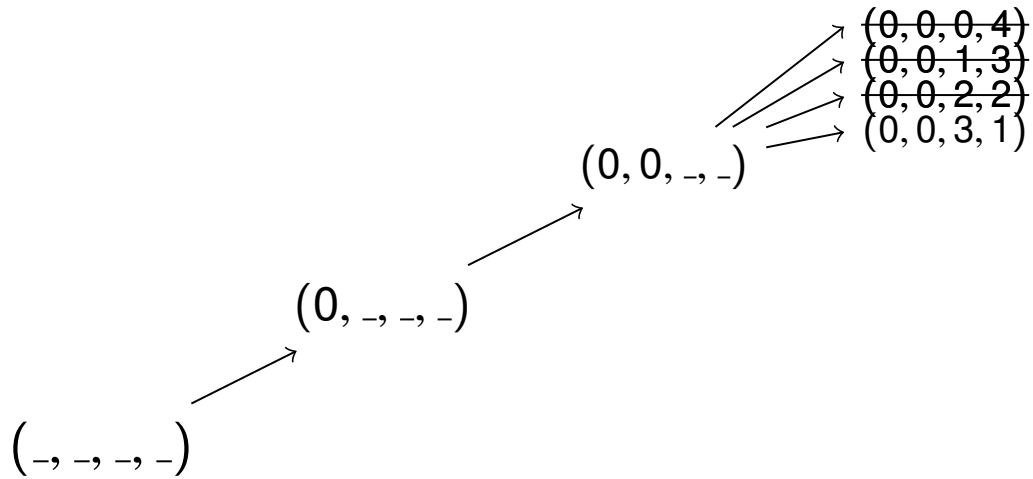
assuming 4 tasks and 4 cache sets

Optimal Cache Partitioning Algorithm

A **cache partitioning** P is a tuple of integers denoting for each task τ_i, the size p_i of its allocated partition:

$$P = (p_1, p_2 \ldots, p_n) : \underbrace{\mathbb{N} \times \ldots \times \mathbb{N}}_{n}$$

Simple **backtracking** algorithm:



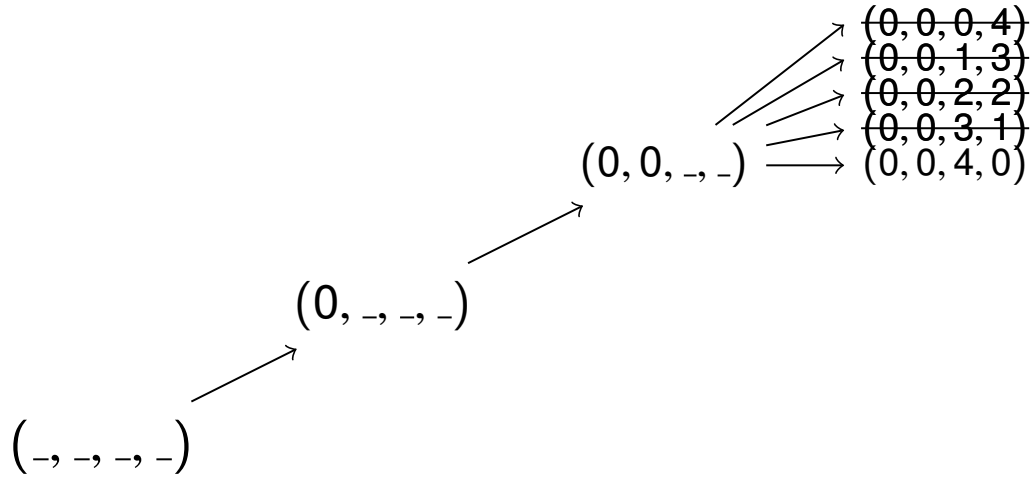
assuming 4 tasks and 4 cache sets

Optimal Cache Partitioning Algorithm

A **cache partitioning** P is a tuple of integers denoting for each task τ_i, the size p_i of its allocated partition:

$$P = (p_1, p_2 \ldots, p_n) : \underbrace{\mathbb{N} \times \ldots \times \mathbb{N}}_{n}$$

Simple **backtracking** algorithm:



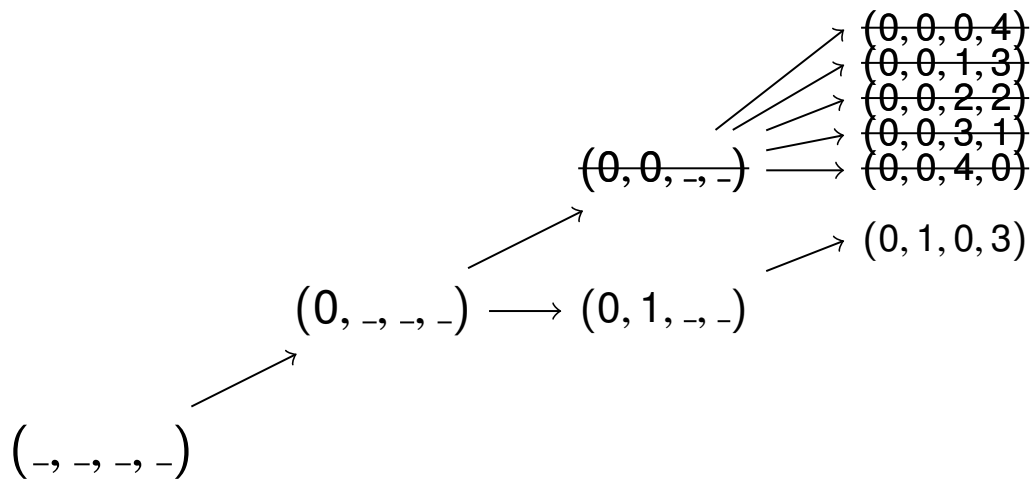
assuming 4 tasks and 4 cache sets

Optimal Cache Partitioning Algorithm

A **cache partitioning** P is a tuple of integers denoting for each task τ_i, the size p_i of its allocated partition:

$$P = (p_1, p_2 \ldots, p_n) : \underbrace{\mathbb{N} \times \ldots \times \mathbb{N}}_{n}$$

Simple **backtracking** algorithm:



assuming 4 tasks and 4 cache sets

Backtracking + Early Exit

Exit Condition 1 (pessimistic assumption)

each unspecified cache partition is given an equal share of the remaining cache
if schedulable under condition 1**: return true**

Exit Condition 2 (optimistic assumption)

each unspecified cache partition is given all the remaining cache
if not schedulable under condition 2**: prune search space**

(same worst-case behaviour, but strongly improved average case)

Requirement 1: Sustainable Schedulability Test

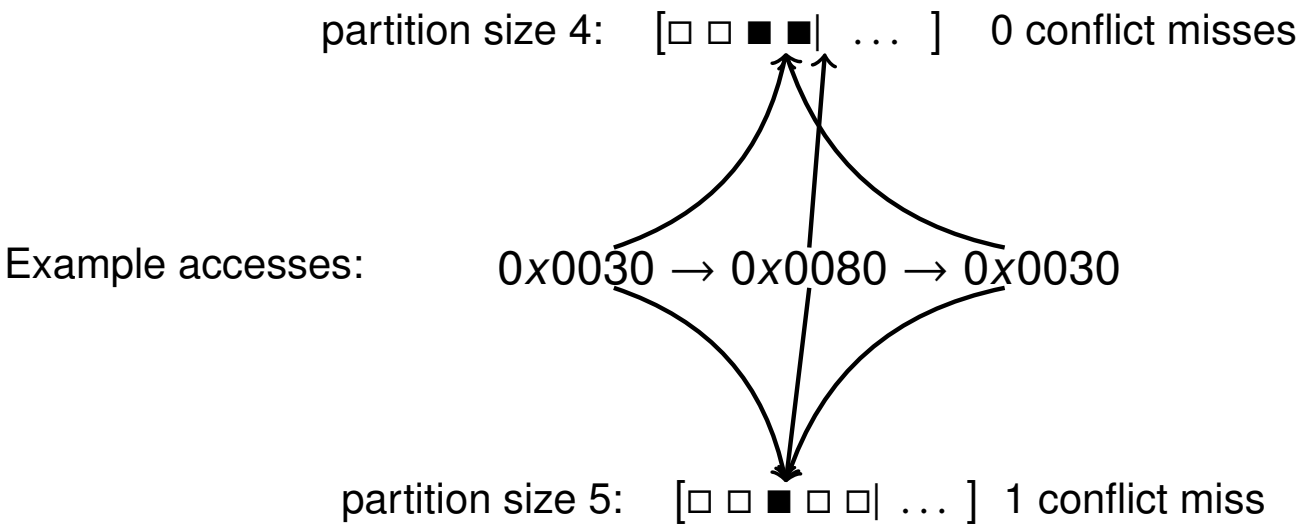
A schedulability test is **sustainable** if any taskset that was deemed schedulable by the test remains so if the parameters “improve” (e.g., if the execution times decrease or periods increase) [1]

Standard schedulability tests (RTA/DBF) are sustainable ✓.

Requirement 2: monotonic WCET

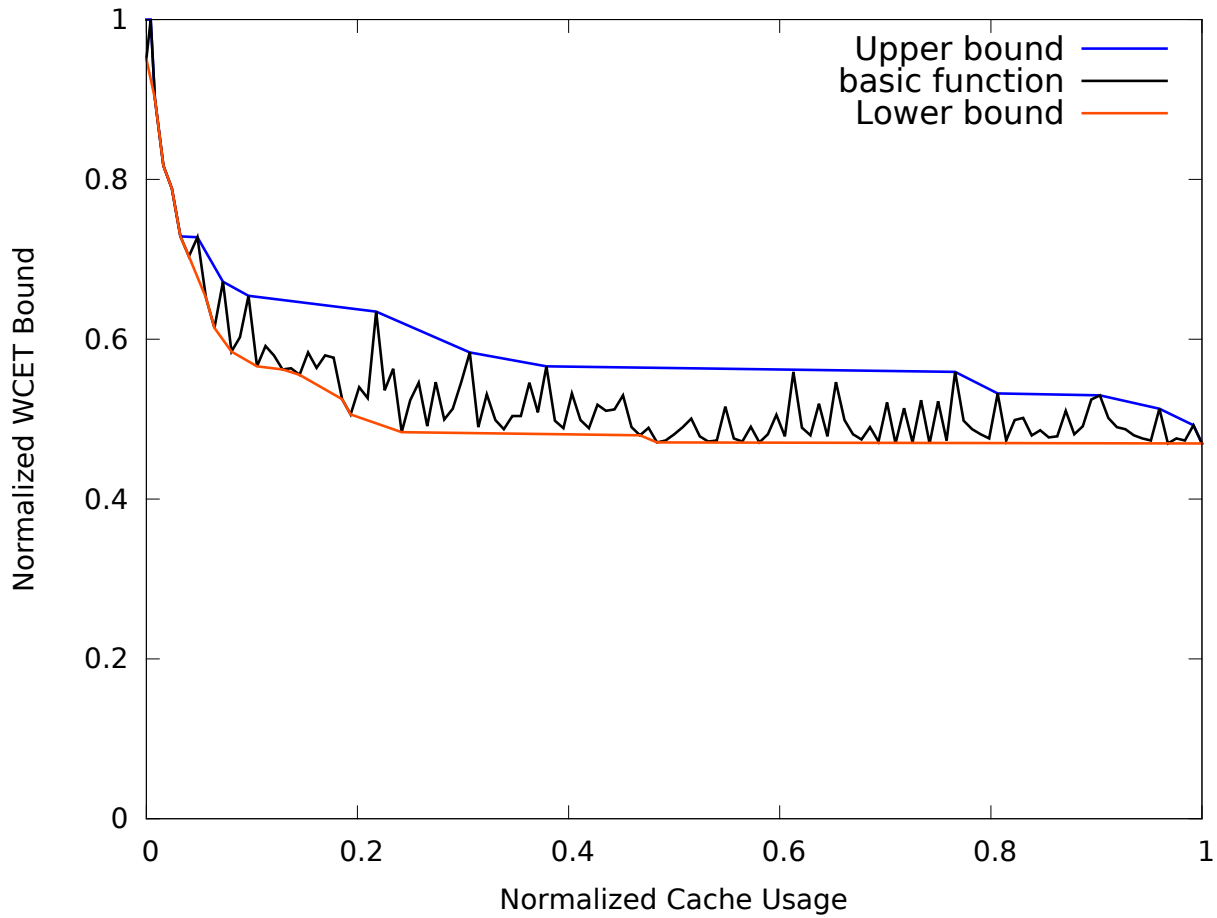
WCETs are not monotonically non-increasing with the size of the cache partition ✗

(despite assumption of monotonic execution times in the related work)



Requirement 2: monotonic WCET

Solution: use a monotonic upper bound ✓



But how large is the error that we introduce? We'll see later...

Unconstrained cache usage

i.e. each task can use the complete cache

1. **Analysis of pre-emption costs** by analyzing the effect of the *pre-empting* task (evicting cache blocks ECBs) on the *pre-empted* task (useful cache blocks UCBs)

2. **Pre-emption cost aware schedulability test**

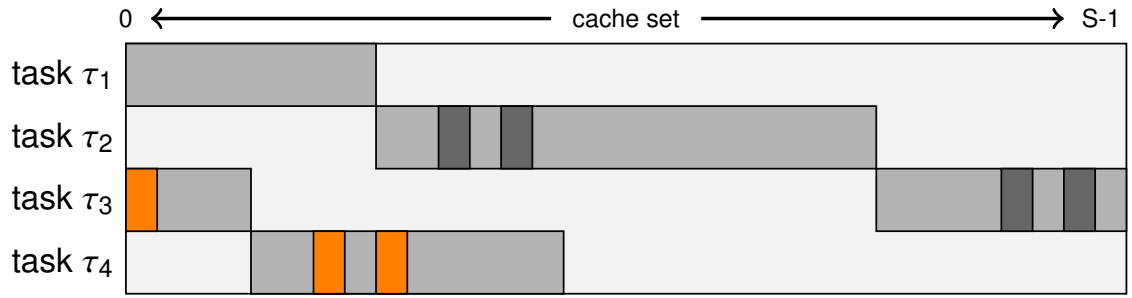
$$R_i = C_i + \sum_{\forall j \in \mathrm{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil (C_j + \gamma_{i,j})$$

where $\gamma_{i,j}$ denotes the preemption cost

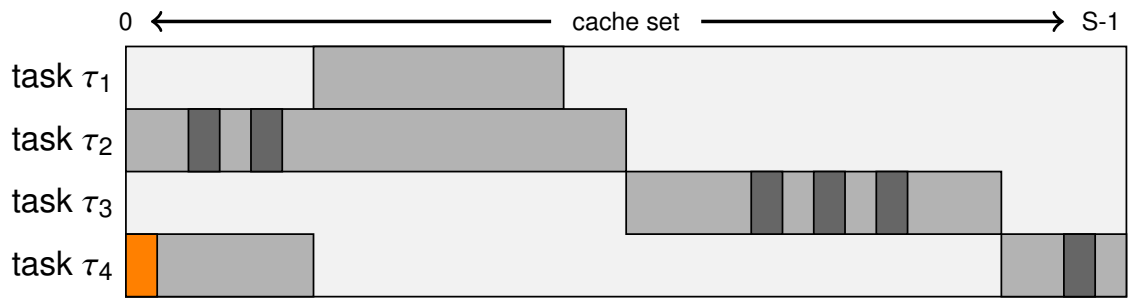
Influence of the Cache Layout [7]

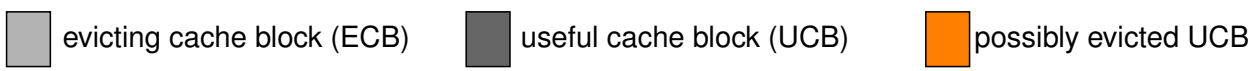
Pre-emption costs strongly depend on the cache layout
i.e, the mapping of memory blocks to cache sets:

Sequential Layout:



Optimized Layout:





Evaluation

We want to know:

- How useful is cache partitioning?
- When to use cache partitioning?
- Which system/task parameters influence usefulness?

We compare:

- no partitioning - optimal cache layout
- no partitioning - sequential cache layout
- optimal cache partitioning
- naive cache partitioning

Benchmarks:

Case Study 1: PapaBench [9]

- short control tasks
- high pre-emption costs w.r.t. WCET

Case Study 2: Mälardalen benchmark suite (+more) [6]

- longer, computationally intensive tasks
- low pre-emption costs w.r.t. WCET

Architecture and Task Selection

Taskset:

- randomly select 10 tasks (from the set of benchmarks)
- task utilizations generated using UUnifast [2]
- $C_i = U_i \cdot T_i$, and $D_i = T_i$ and no jitter, i.e, $J_i = 0$
- fixed priority scheduling with Deadline Monotonic priority order

Architecture:

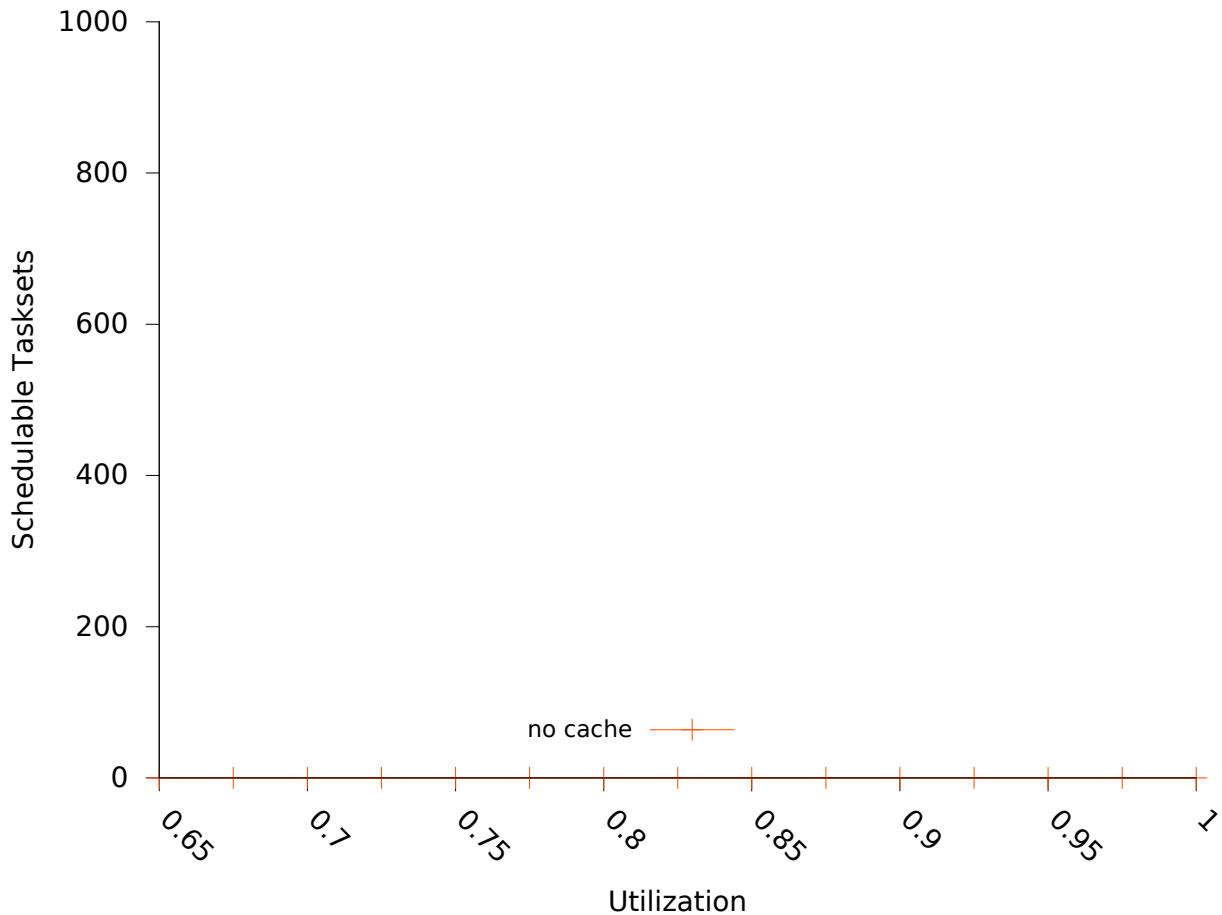
- ARM7 processor with clock speed of 100 MHz
- direct-mapped instruction cache with perfect data cache (4kB, 16 Bytes line size, 256 cache sets)
- a block reload time $= 8\mu s$
- WCET bounds derived by aiT Timing analyzer [5]
- no implementation overhead for cache partitioning

PapaBench Benchmarks

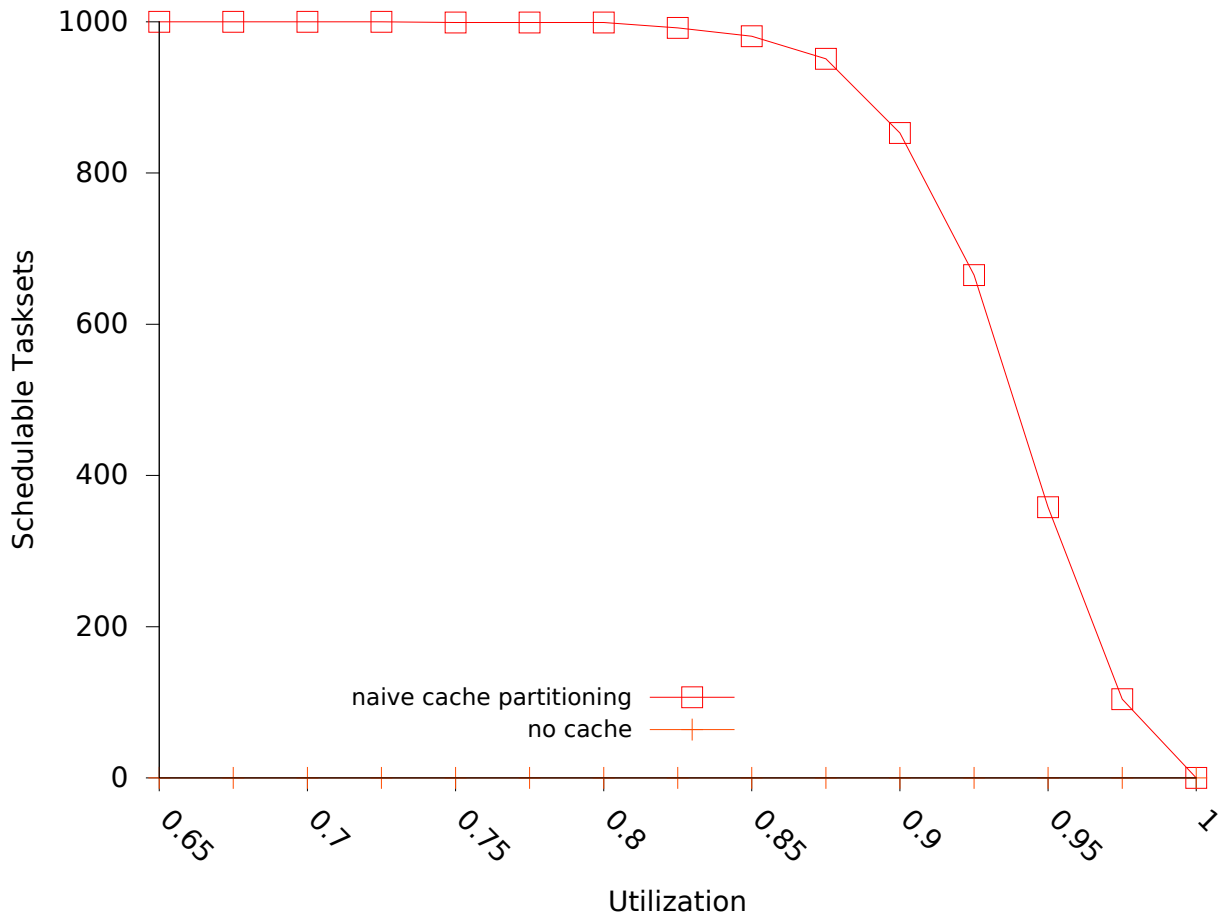
	Description	UCBs	ECBs	WCET
I4	interrupt-modem	2	10	$303\mu s$
I5	interrupt-spi-1	1	10	$251\mu s$
I6	interrupt-spi-2	1	4	$151\mu s$
I7	interrupt-gps	3	26	$283\mu s$
T5	altitude-control	20	66	$1478\mu s$
T6	climb-control	1	210	$5429\mu s$
T7	link-fbw-send	1	10	$233\mu s$
T8	navigation	1	256	$44,42ms$
T9	radio-control	0	256	$15,6ms$
T10	receive-gps-data	22	194	$5987\mu s$
T11	reporting	2	256	$12,22ms$
T12	stabilization	11	194	$5681\mu s$

(Instruction cache with perfect data cache)

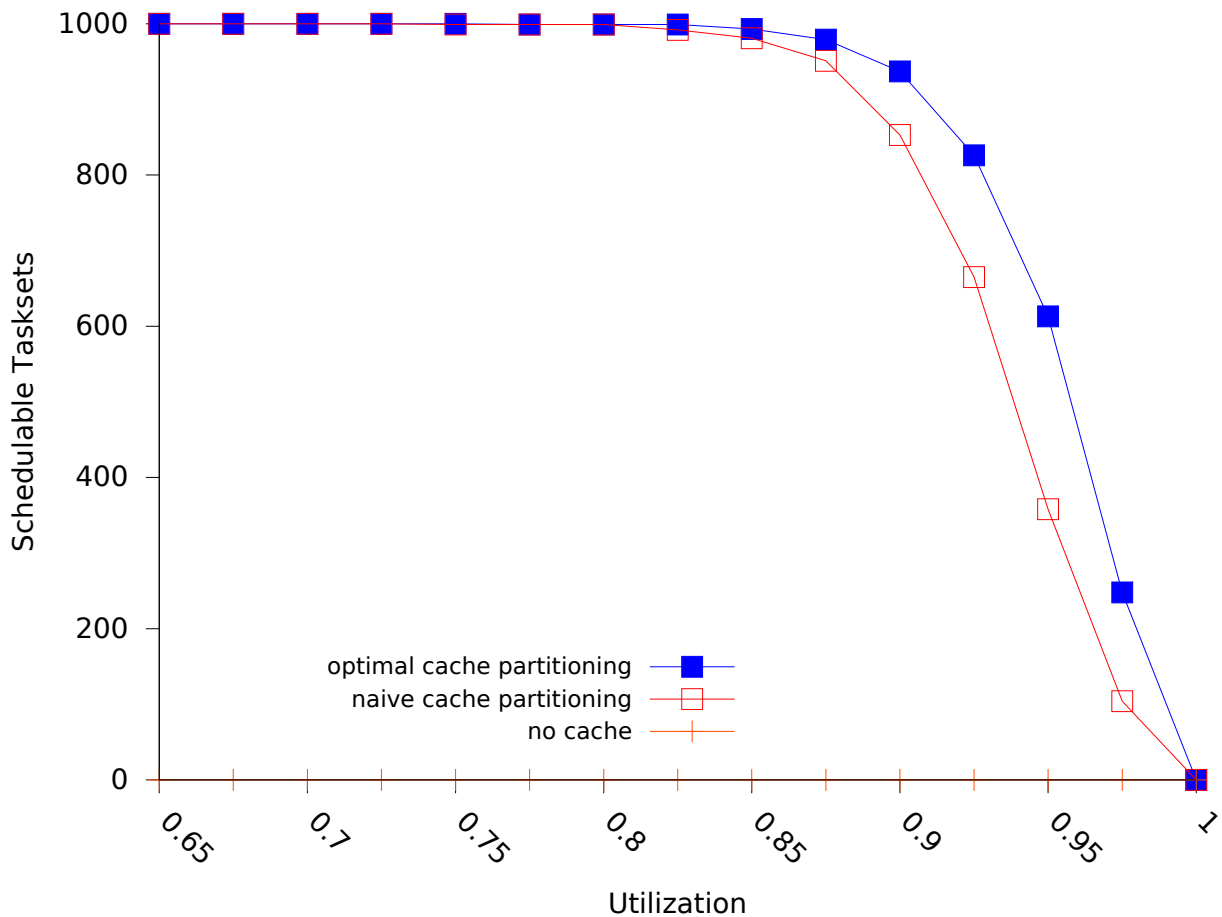
Instruction cache (perfect data cache) - PapaBench



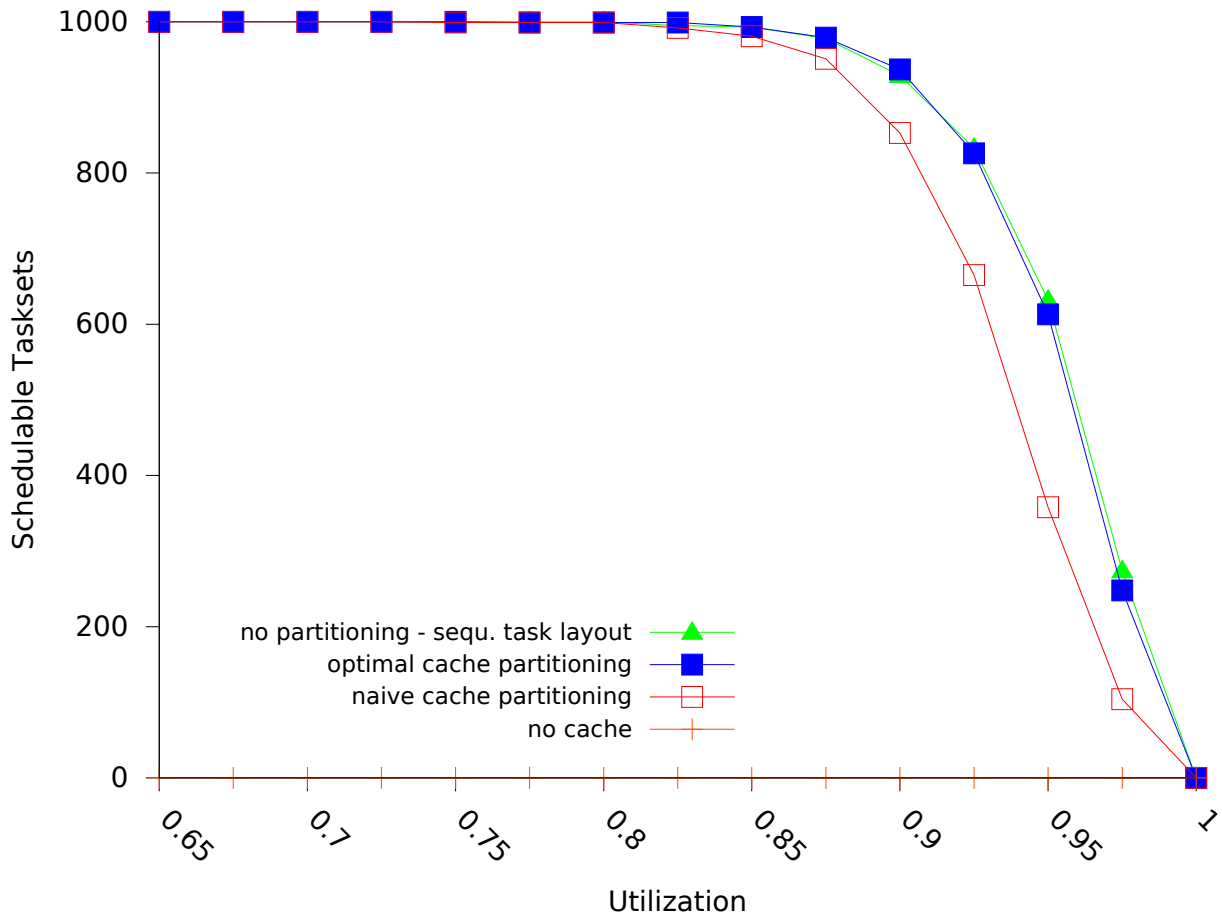
Instruction cache (perfect data cache) - PapaBench



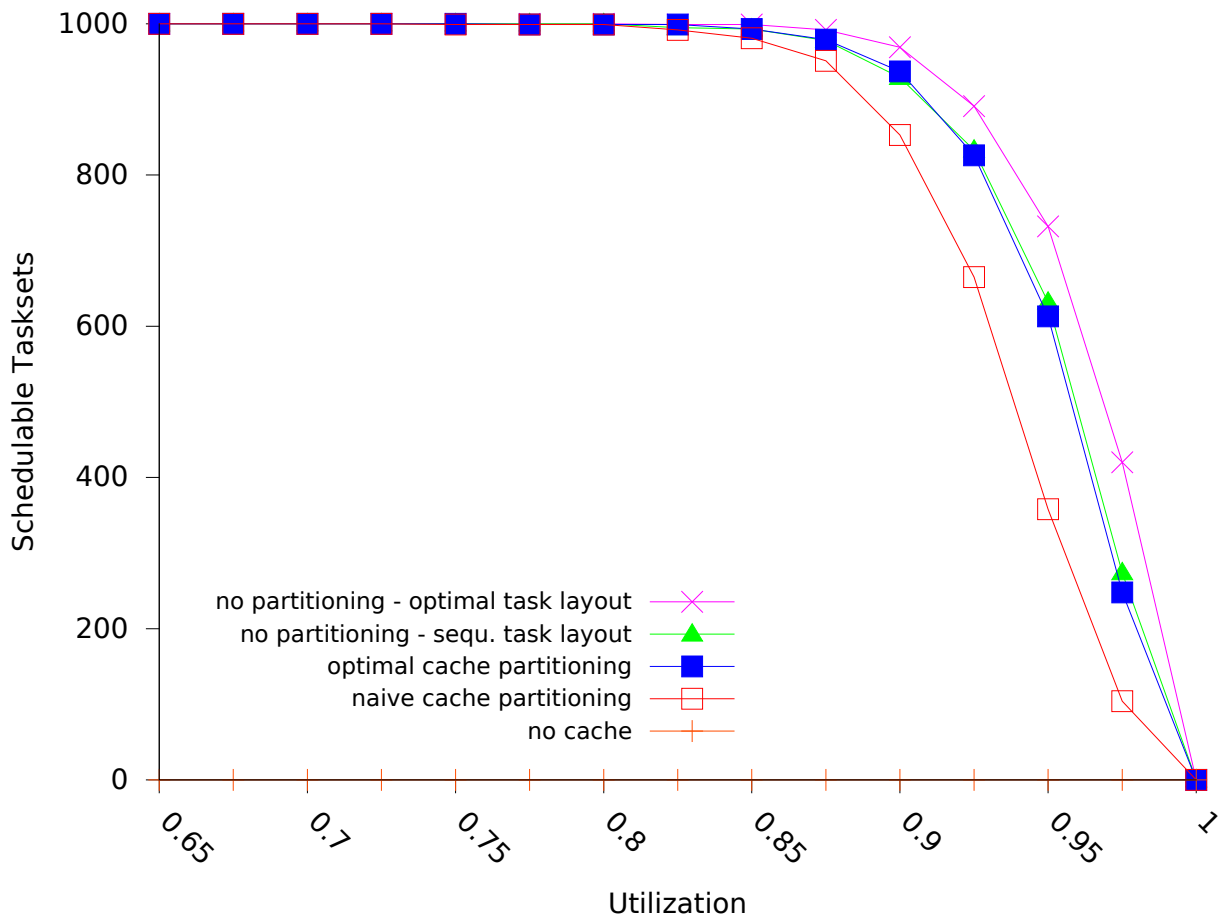
Instruction cache (perfect data cache) - PapaBench



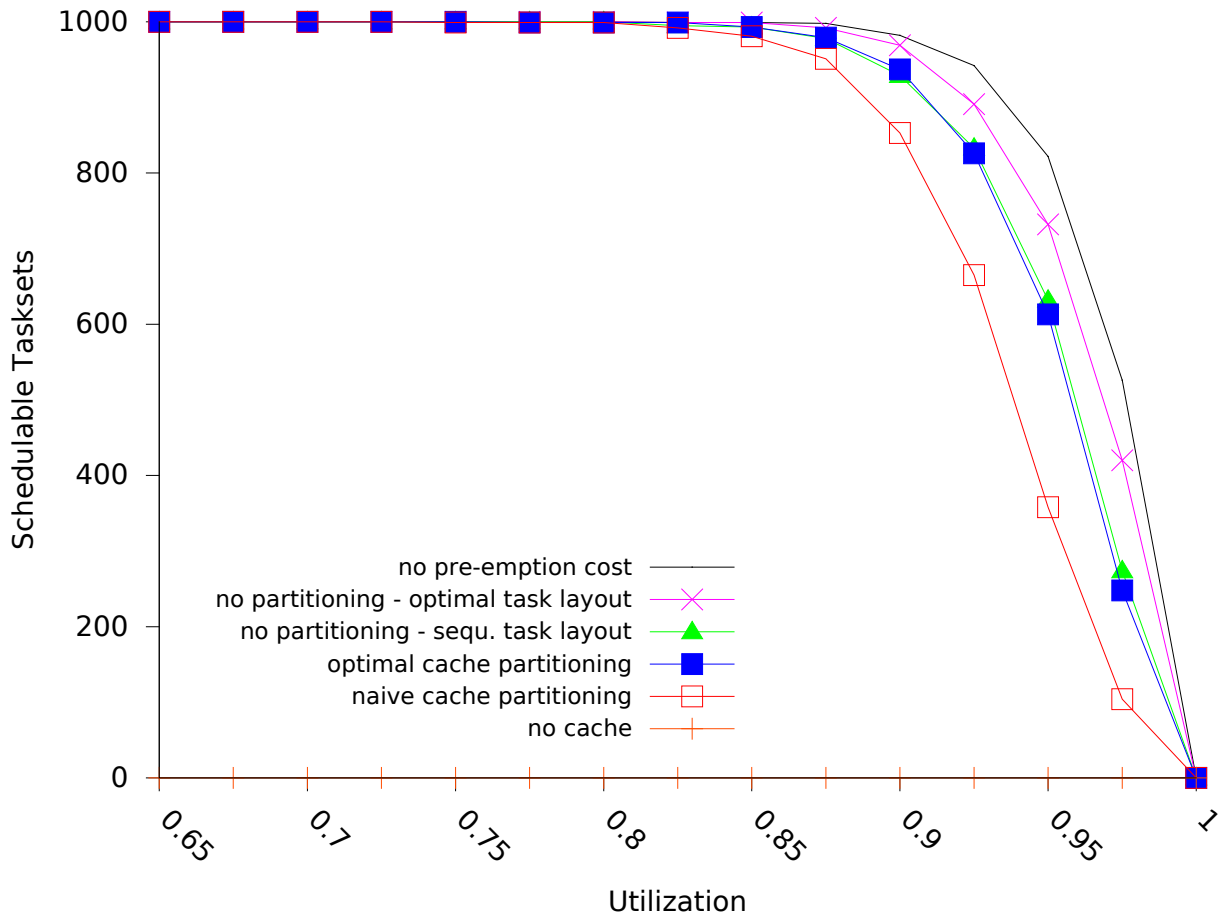
Instruction cache (perfect data cache) - PapaBench



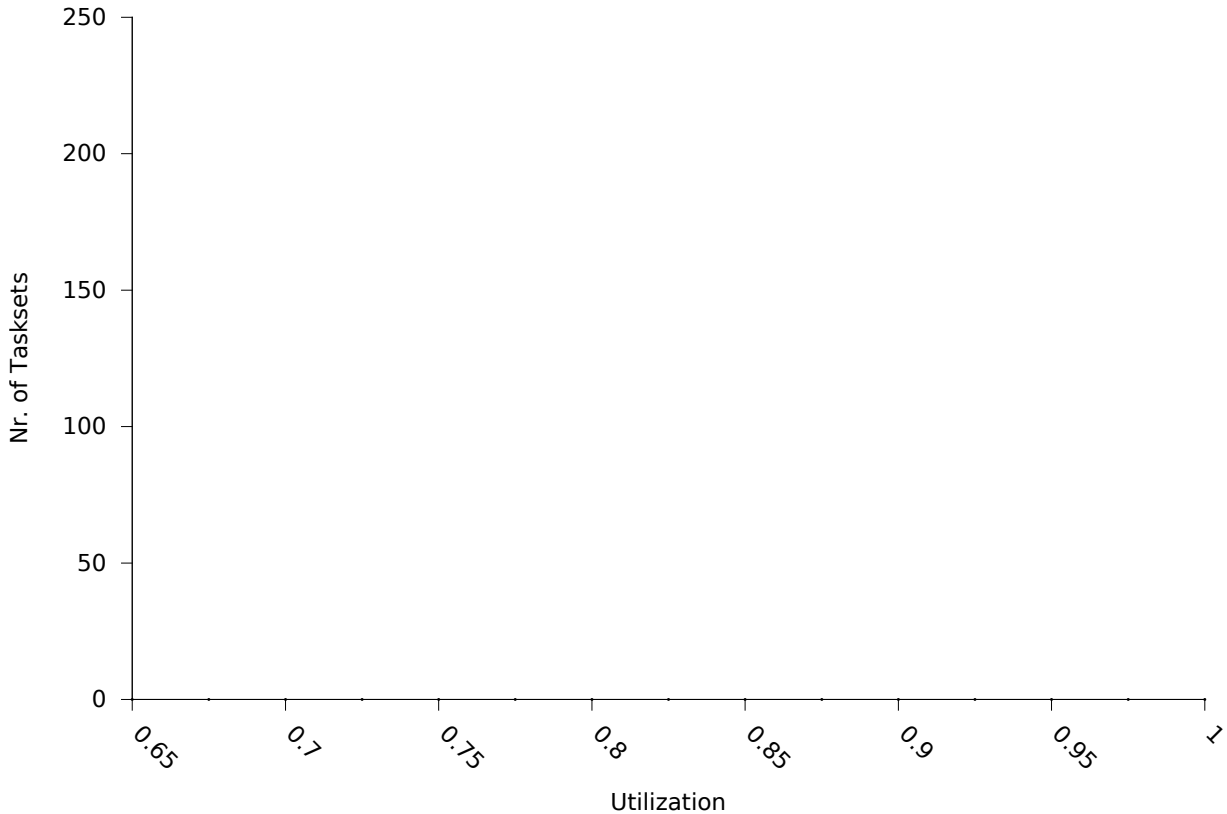
Instruction cache (perfect data cache) - PapaBench



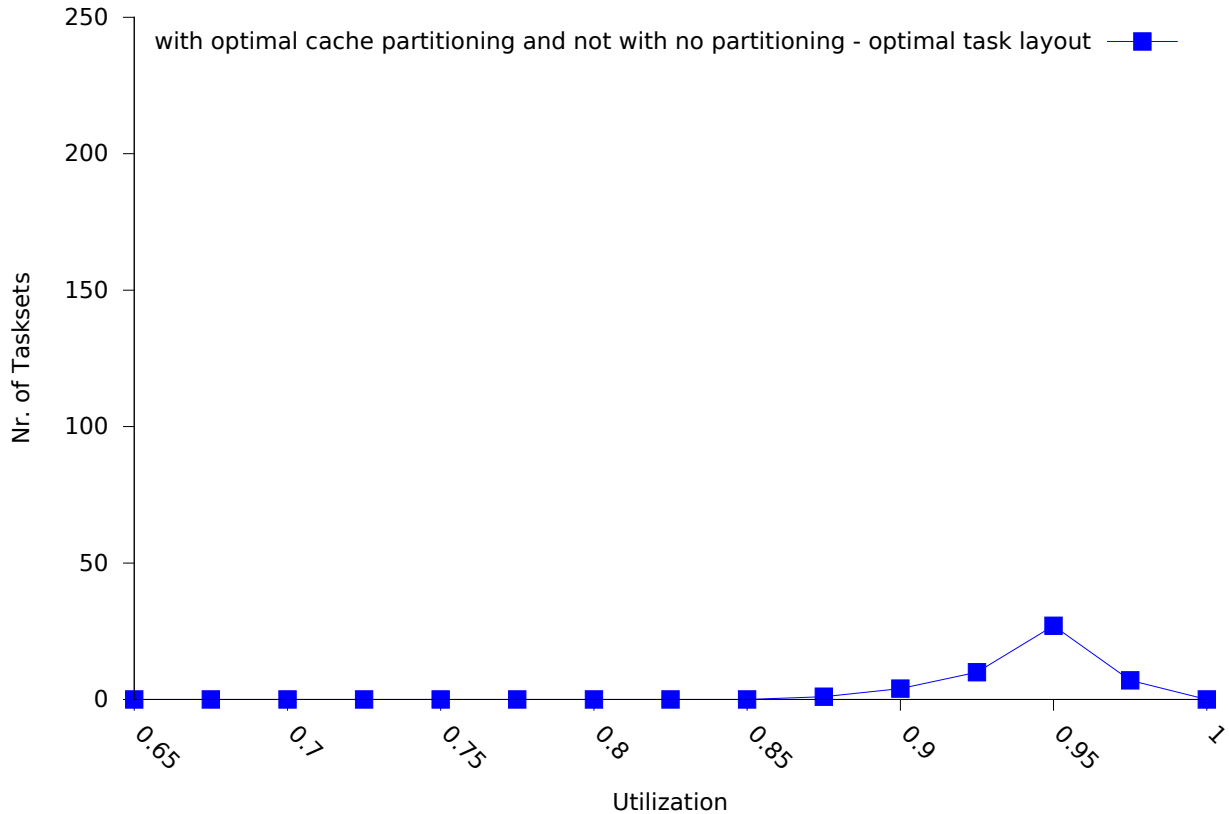
Instruction cache (perfect data cache) - PapaBench



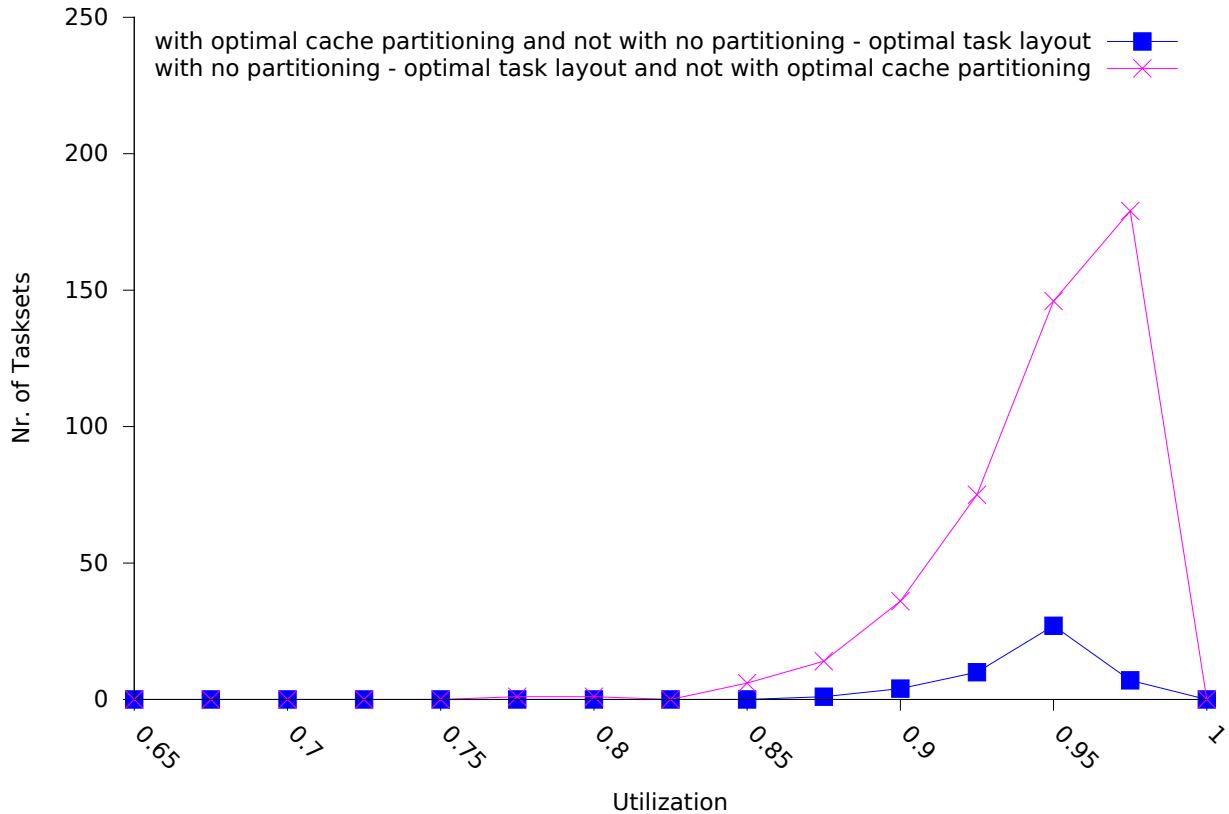
PapaBench - Detailed Comparison



PapaBench - Detailed Comparison



PapaBench - Detailed Comparison

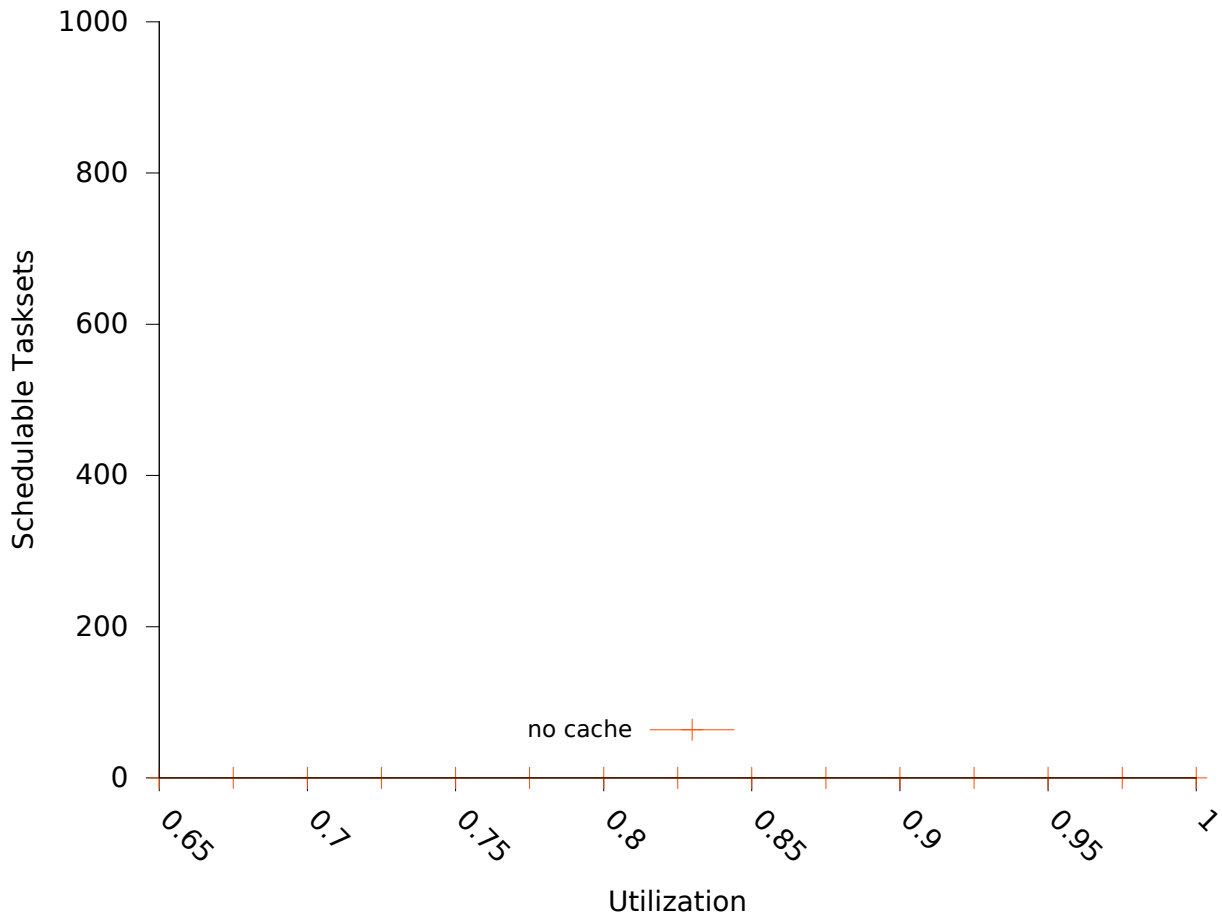


Mälardalen Benchmark Suite (M) and SCADE Benchmarks (S)

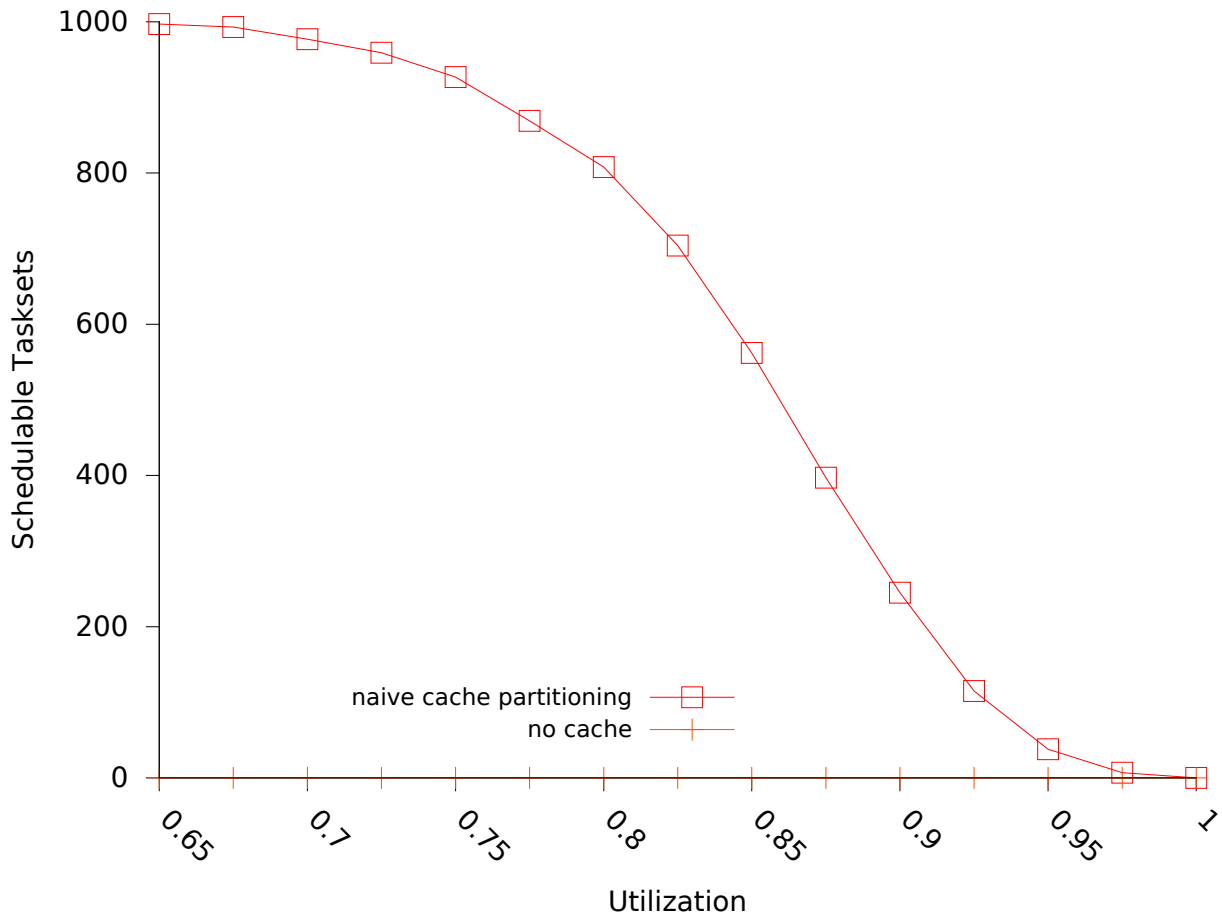
	Description	UCBs	ECBs	WCET
M	adpcm	24	226	$5,541s$
M	compress	25	114	$3,664s$
M	edn	56	98	$244,8ms$
M	fir	28	50	$21,52ms$
M	jfdctinit	40	162	$13,89ms$
M	ns	17	26	$73,38ms$
M	nsichneu	53	256	$77,96ms$
M	statemate	3	256	$9,757s$
S	cruise control system	25	107	$1,959s$
S	flight control system	70	256	$2,138s$
S	navigation system	45	82	$1,409s$
S	stopwatch	58	130	$3,786s$
S	elevator simulation	40	114	$1,586s$
S	robotics systems	68	256	$4,311s$

Instruction cache with perfect data cache

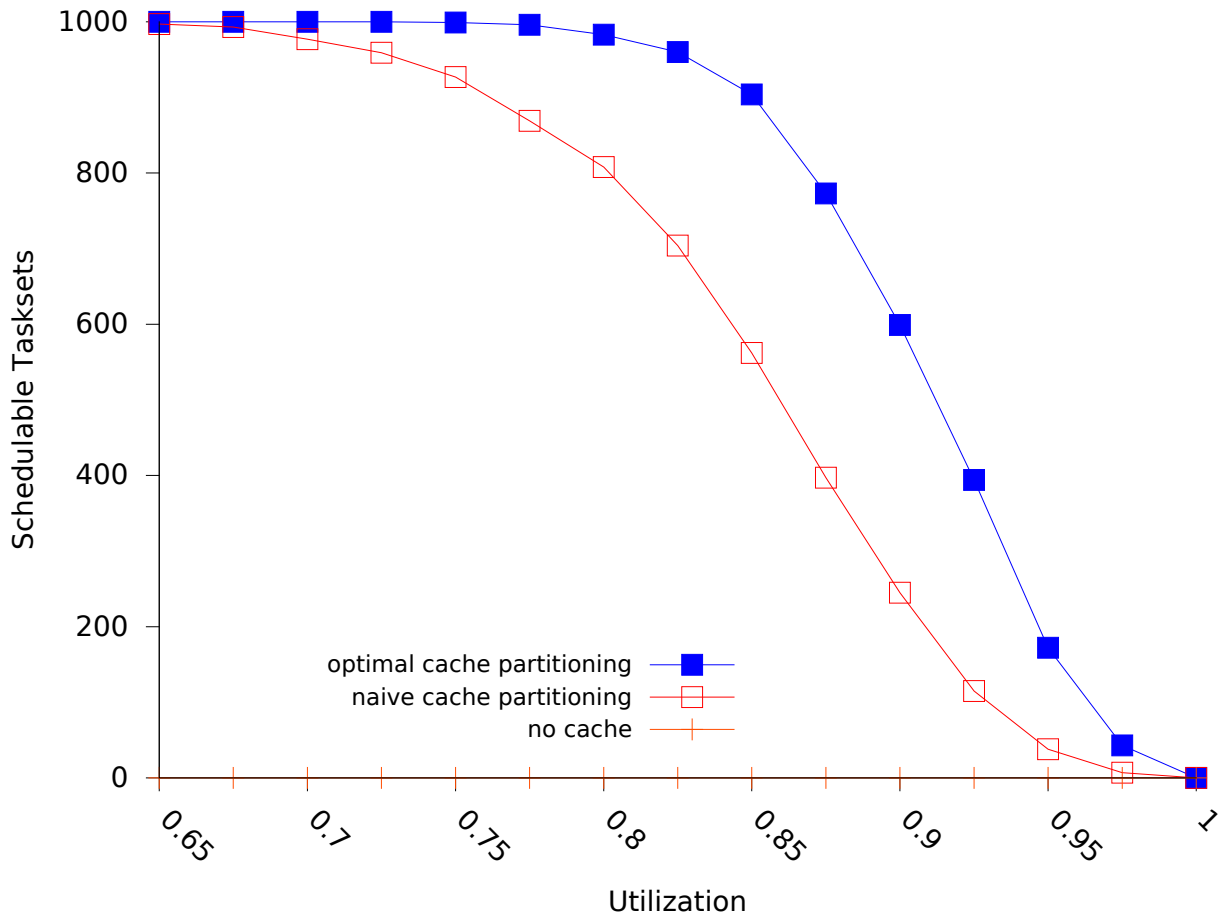
Instruction cache (perfect data cache) - Mälardalen



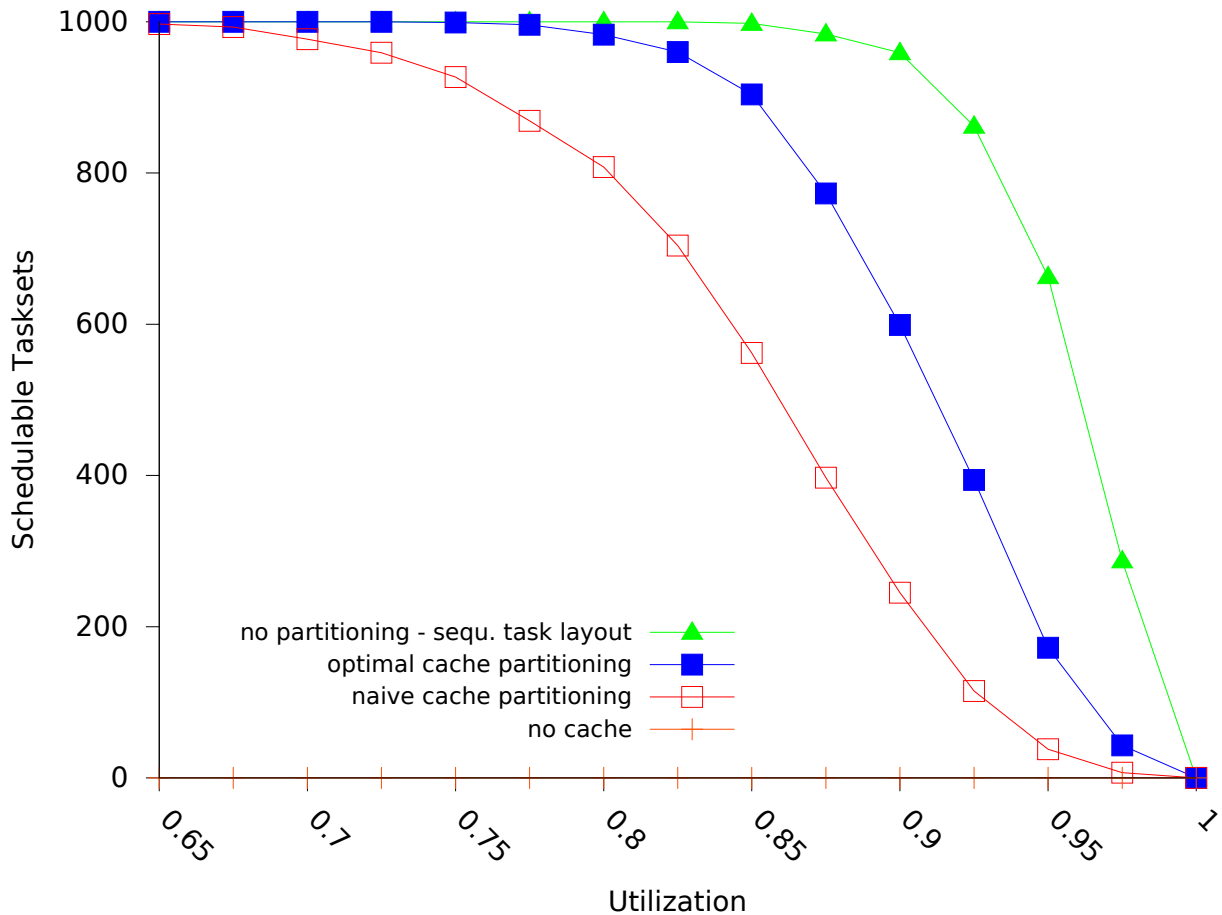
Instruction cache (perfect data cache) - Mälardalen



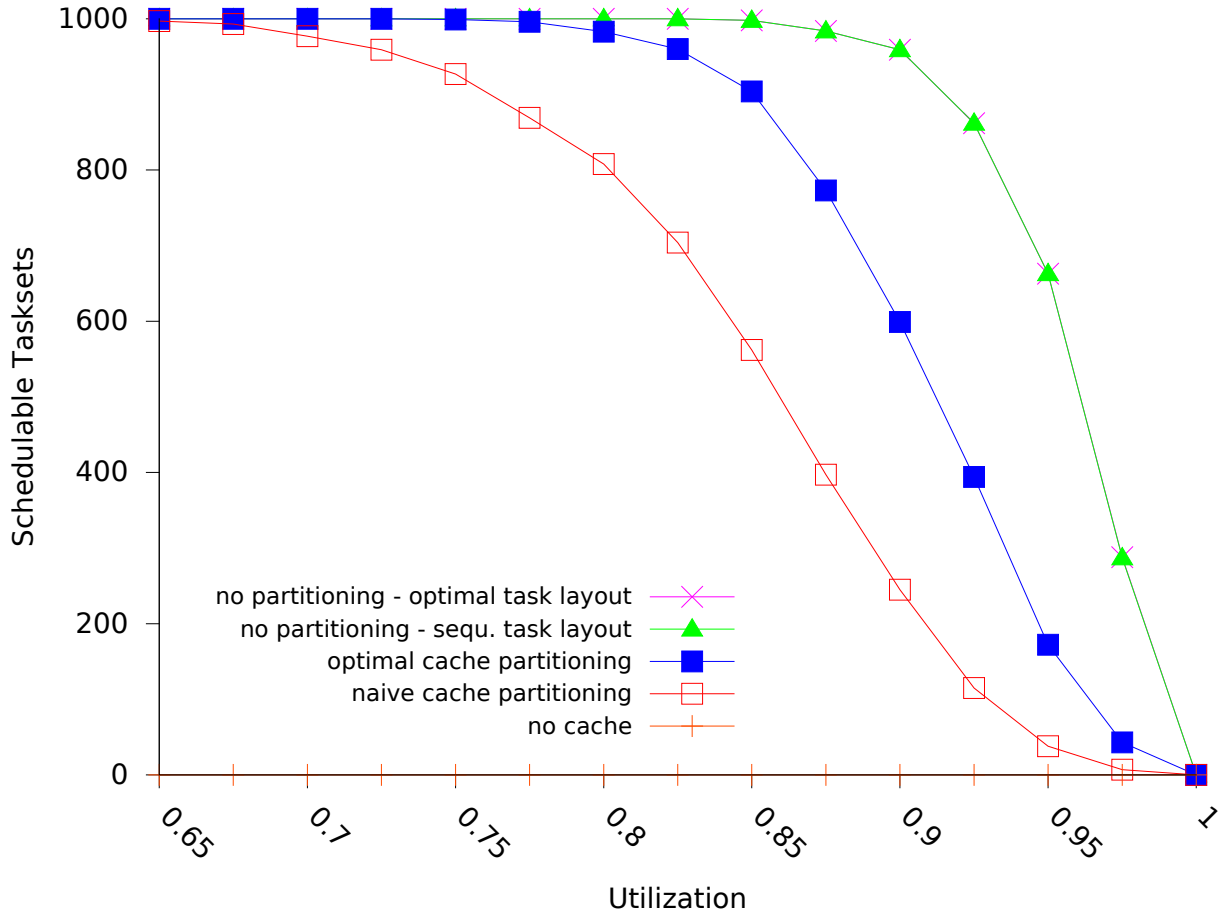
Instruction cache (perfect data cache) - Mälardalen



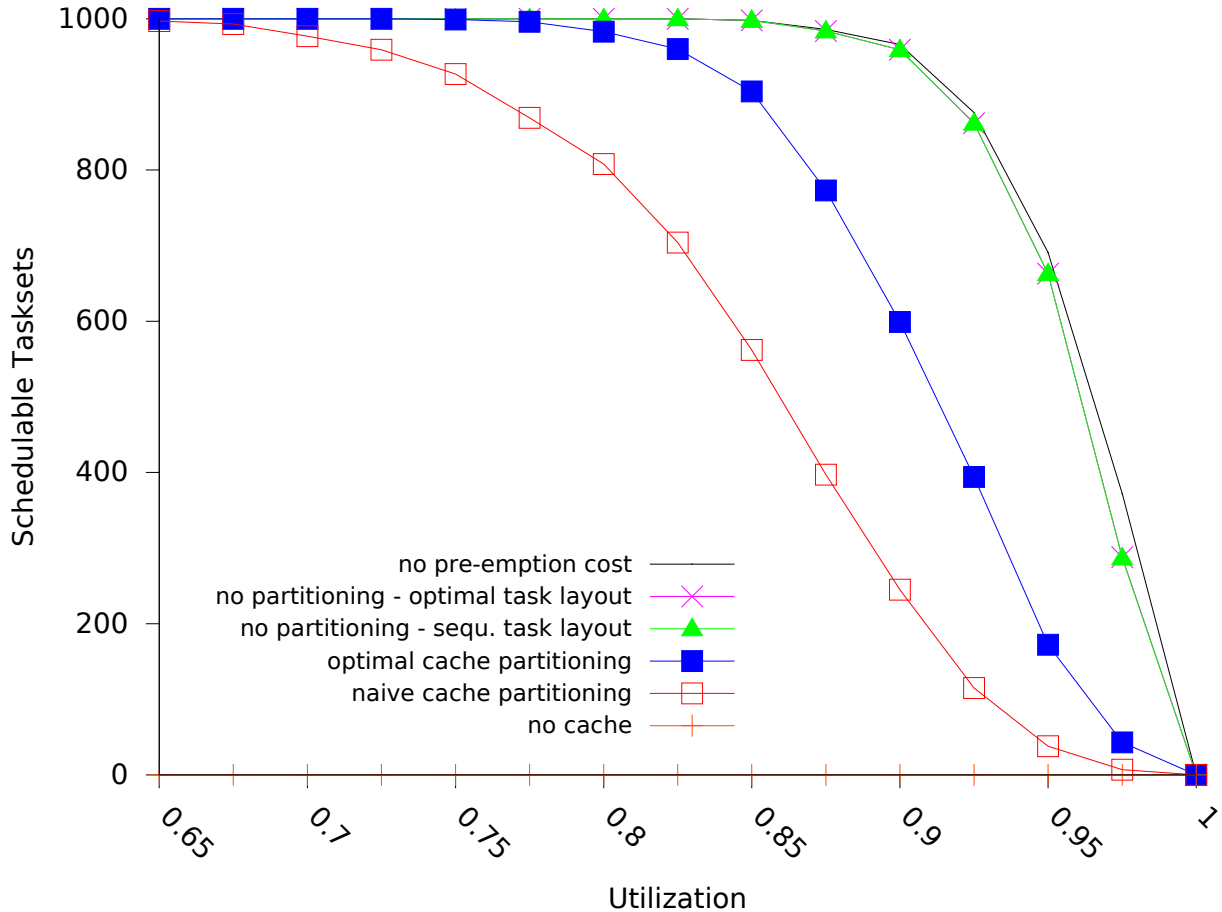
Instruction cache (perfect data cache) - Mälardalen



Instruction cache (perfect data cache) - Mälardalen



Instruction cache (perfect data cache) - Mälardalen



Lesson learned from the case study:

- cache partitioning and pre-emption cost analysis are incomparable
- cache partitioning often detrimental to overall schedulability, especially with comparably low pre-emption costs
- optimal task layout (often) as effective as optimal cache partitioning

(similar behaviour for data caches, see paper)

⇒ **Assumption does not hold in general:**
cache partitioning is often not useful

Evaluation for synthetic tasksets

Key performance parameters:

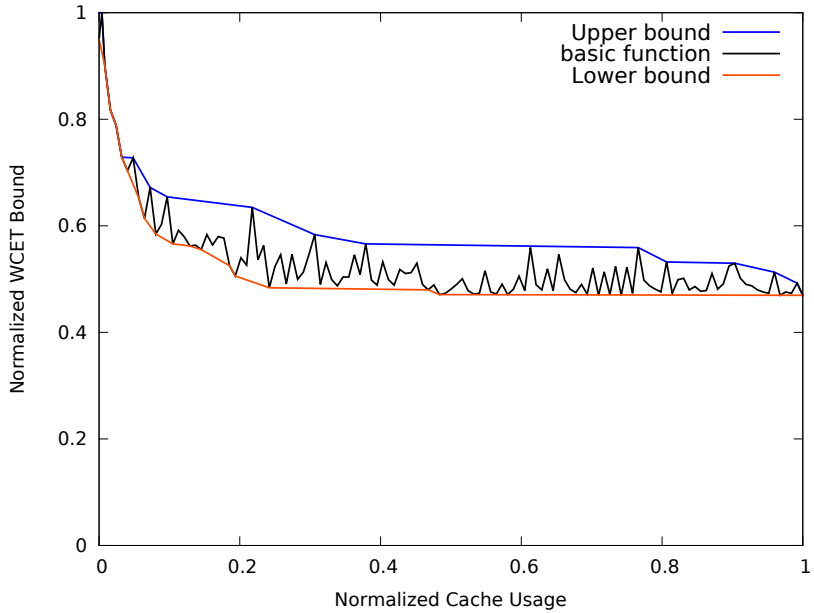
Ratio of pre-emption cost to WCRT partitioning does not pay off in case of low pre-emption costs

Cache utilization non-partitioned approach suffers more from pessimism at high utilizations

Number of tasks non-partitioned approach suffers from pessimism in the computation of the pre-emption costs

Number of cache sets has very limited impact on the relative performance of both approaches

Assumption of monotonic WCETs: How large is the uncertainty?



$$Error \leq \frac{\sum_{ts}\left(\text{schedulable(ts)}^{\text{lower-bound}} \wedge \neg\text{schedulable(ts)}^{\text{upper-bound}}\right)}{\text{total number of task sets}} \approx 1.2\%$$

Optimal Cache Partitioning

Alternative to Cache Partitioning

Evaluation

Conclusions

Conclusions

- Sensitivity analysis of WCET with respect to partition size
- Optimal cache partitioning algorithm
- Thorough evaluation
  - increased predictability does not compensate for the WCET degradation
  - precise pre-emption cost analysis justifies unconstrained cache usage

Future Work:

- Evaluation for dynamic, instead of fixed priorities
- Hybrid Cache Partitioning (as used in hierarchical scheduling/spatial isolation)
- Evaluation for more complex cache architectures.

Bibliography

[1] S. Baruah and A. Burns.
Sustainable scheduling analysis.
In *RTSS*, pages 159–168, December 2006.

[2] E. Bini and G. Buttazzo.
Measuring the performance of schedulability tests.
Real-Time Systems, 30:129–154, 2005.

[3] B. D. Bui, M. Caccamo, L. Sha, and J. Martinez.
Impact of cache partitioning on multi-tasking real time embedded systems.
In *RTCSA*, pages 101–110, August 2008.

[4] J. V. Busquets-Mataix and A. Wellings.
Hybrid instruction cache partitioning for preemptive real-time systems.
In *RTS*, June 1997.

[5] C. Ferdinand and R. Heckmann.
aiT: worst case execution time prediction by static program analysis.
In *IFIP*, pages 377–384, August 2004.

[6] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper.
The Mälardalen WCET benchmarks – past, present and future.
In *WCET*, pages 137–147, July 2010.

[7] W. Lunniss, S. Altmeyer, and R. I. Davis.
Optimising task layout to increase schedulability via reduced cache related pre-emption delays.
In *RTNS*, pages 161–170, November 2012.

[8] F. Mueller.
Compiler support for software-based cache partitioning.
SIGPLAN Not., 30(11):125–133, 1995.

[9] F. Nemer, H. Cassé, P. Sainrat, J.-P. Bahsoun, and M. D. Michiel.
Papabench: a free real-time benchmark.
In *WCET*, July 2006.

[10] S. Plazar, P. Lokuciejewski, and P. Marwedel.
Wcet-aware software based cache partitioning for multi-task real-time systems.
In *WCET*, July 2009.

[11] I. Puaut and D. Decotigny.
Low-complexity algorithms for static cache locking in multitasking hard real-time systems.
In *RTSS*, pages 114–124, December 2002.

[12] X. Vera, B. Lisper, and J. Xue.
Data cache locking for tight timing calculations.
ACM Transactions on Embedded Computing Systems, 7(1):4:1–4:38, December 2007.

```
1: function checkPartition(int i, partition P, int s)
2:     if i = n then
3:         return isSchedulable(P)
4:     end if
5:     let P = (p_1, .., p_n)
6:     if not isSchedulable((p_1, .., p_{i-1}, s, .., s)) then
7:         return false
8:     end if
9:     if isSchedulable((p_1, .., p_{i-1}, ⌊s/(n-i+1)⌋, .., ⌊s/(n-i+1)⌋)) then
10:        return true
11:    end if
12:    p_i = 0
13:    while p_i ≤ s do
14:        if checkPartition(i + 1, (p_1, .., p_i, .., p_n), s − p_i) then
15:            return true
16:        else
17:            p_i = nextStep(i,p_i)
18:        end if
19:    end while
20:    return false
21: end function
```

PapaBench Benchmarks

	Description	UCBs	ECBs	$WCET^1$	$WCET^2$
I4	interrupt-modem	2	10	$303\mu s$	$520\mu s$
I5	interrupt-spi-1	1	10	$251\mu s$	$447\mu s$
I6	interrupt-spi-2	1	4	$151\mu s$	$228\mu s$
I7	interrupt-gps	3	26	$283\mu s$	$493\mu s$
T5	altitude-control	20	66	$1478\mu s$	$1660\mu s$
T6	climb-control	1	210	$5429\mu s$	$6241\mu s$
T7	link-fbw-send	1	10	$233\mu s$	$471\mu s$
T8	navigation	1	256	$44,42ms$	$54,35ms$
T9	radio-control	0	256	$15,6ms$	$21,1ms$
T10	receive-gps-data	22	194	$5987\mu s$	$6659\mu s$
T11	reporting	2	256	$12,22ms$	$5ms$
T12	stabilization	11	194	$5681\mu s$	$6654\mu s$

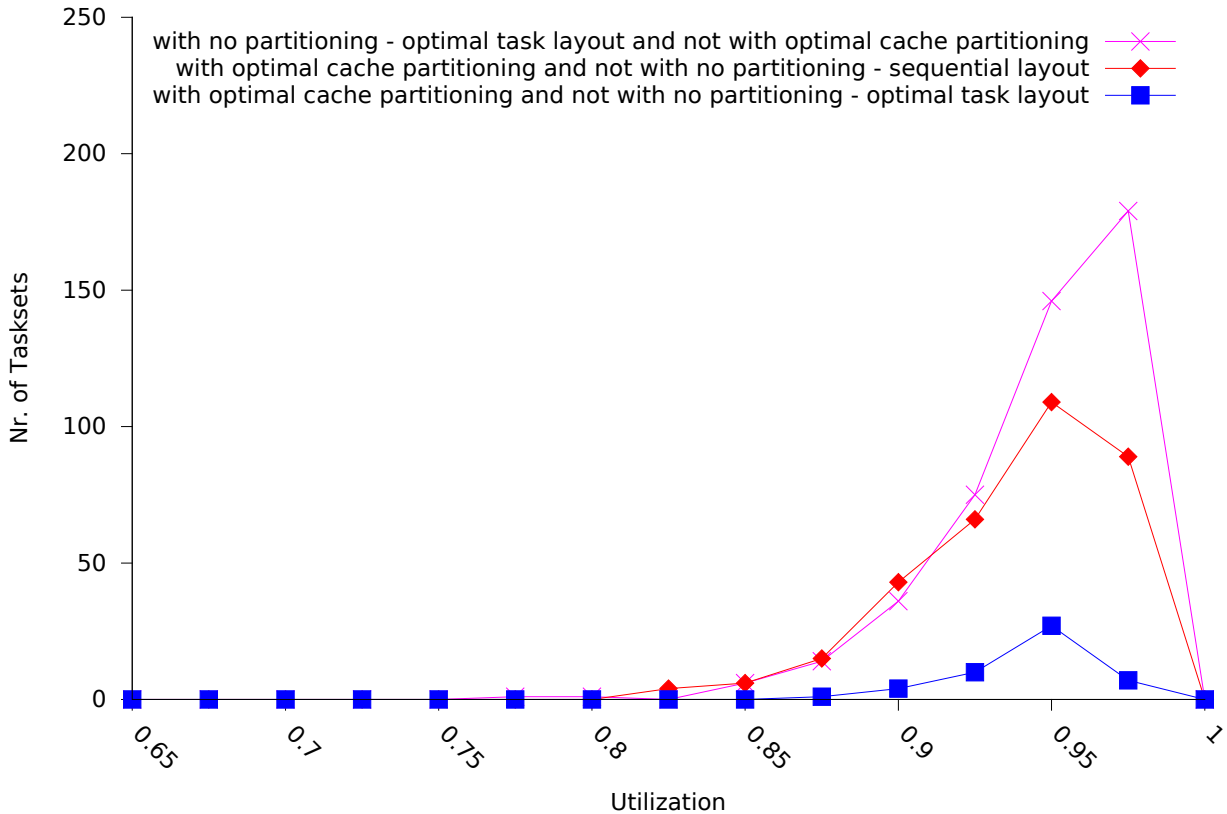
Instruction cache with perfect data cache ($WCET^1$),
and without data cache ($WCET^2$)

Mälardalen Benchmark Suite (M) and SCADE Benchmarks (S)

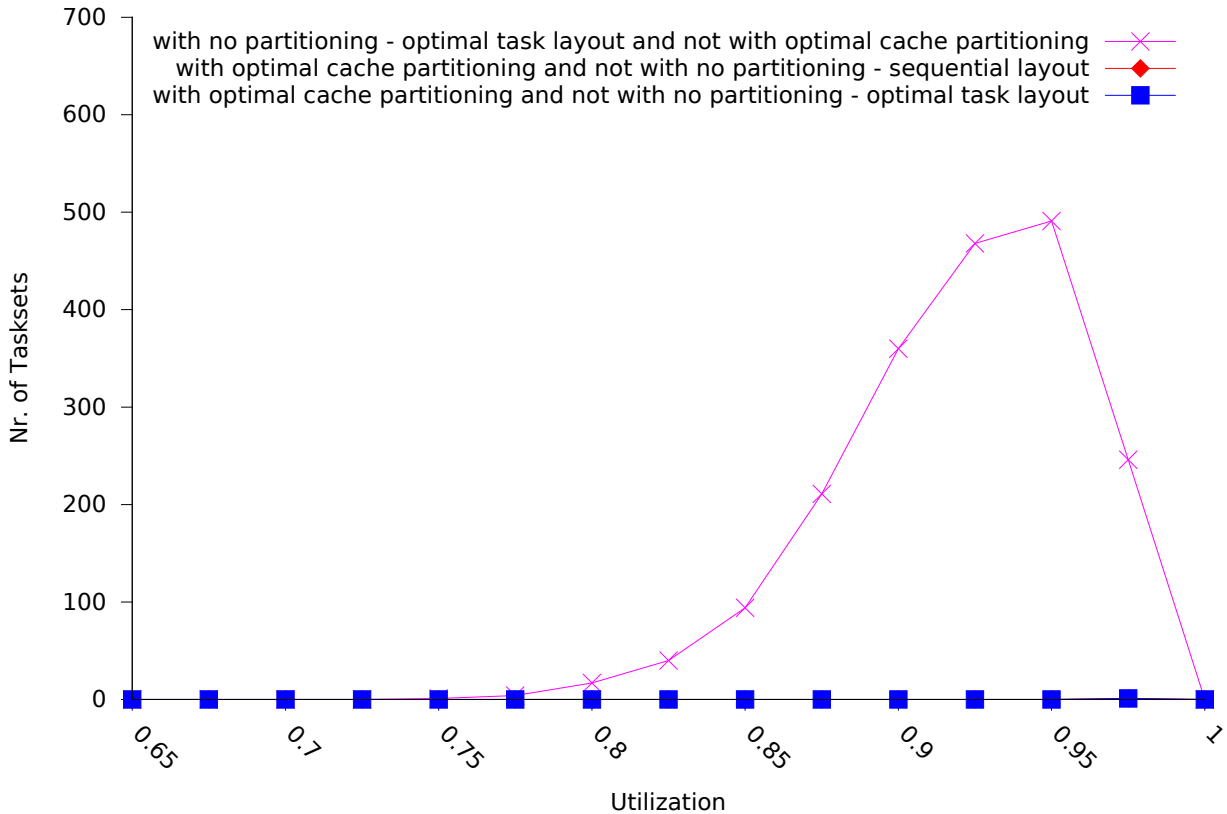
	Description	UCBs	ECBs	$WCET^1$	$WCET^2$
M	adpcm	24	226	$5,541s$	$6,521s$
M	compress	25	114	$3,664s$	$8,426s$
M	edn	56	98	$244,8ms$	$458,2ms$
M	fir	28	50	$21,52ms$	$497ms$
M	jfdctinit	40	162	$13,89ms$	$32,98ms$
M	ns	17	26	$73,38ms$	$168ms$
M	nsichneu	53	256	$77,96ms$	$163ms$
M	statemate	3	256	$9,757s$	$20,07s$
S	cruise control system	25	107	$1,959s$	$3,548s$
S	flight control system	70	256	$2,138s$	$4,083s$
S	navigation system	45	82	$1,409s$	$3,712s$
S	stopwatch	58	130	$3,786s$	$5,533s$
S	elevator simulation	40	114	$1,586s$	$2,917s$
S	robotics systems	68	256	$4,311s$	$6,377s$

Instruction cache with perfect data cache ($WCET^1$), and without data cache ($WCET^2$)

Instruction cache (perfect data cache) - PapaBench



Instruction cache (perfect data cache) - Mälardalen



PapaBench Benchmarks

	Description	UCBs	ECBs	$WCET^1$	$WCET^2$	Period
I4	interrupt-modem	3	10	$335\mu s$	$790\mu s$	-
I5	interrupt-spi-1	2	10	$287\mu s$	$644\mu s$	-
I6	interrupt-spi-2	1	4	$135\mu s$	$338\mu s$	-
I7	interrupt-gps	3	26	$278\mu s$	$712\mu s$	-
T5	altitude-control	2	66	$654\mu s$	$3860\mu s$	$250ms$
T6	climb-control	5	210	$2375\mu s$	$14,21mss$	$250ms$
T7	link-fbw-send	2	10	$298\mu s$	$634\mu s$	$250ms$
T8	navigation	10	256	$23,38ms$	$138ms$	$50ms$
T9	radio-control	14	256	$10,2ms$	$51ms$	$50ms$
T10	receive-gps-data	4	194	$3058\mu s$	$20,5mss$	$25ms$
T11	reporting	6	242	$12,8ms$	$32ms$	$100ms$
T12	stabilization	6	194	$2711\mu s$	$16,1mss$	$50ms$

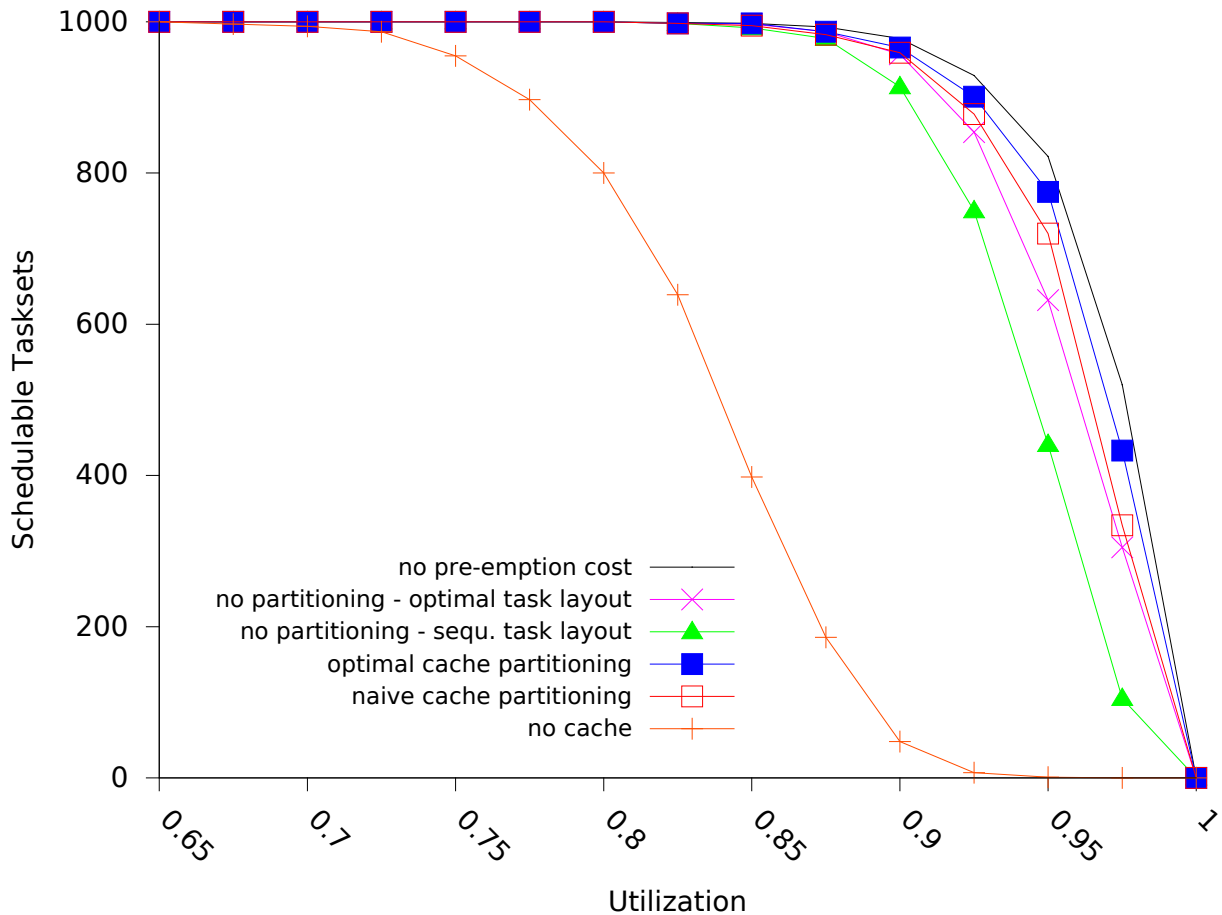
Data cache with perfect instruction cache ($WCET^1$),
and without instruction cache ($WCET^2$)

Mälardalen Benchmark Suite (M) and SCADE Benchmarks (S)

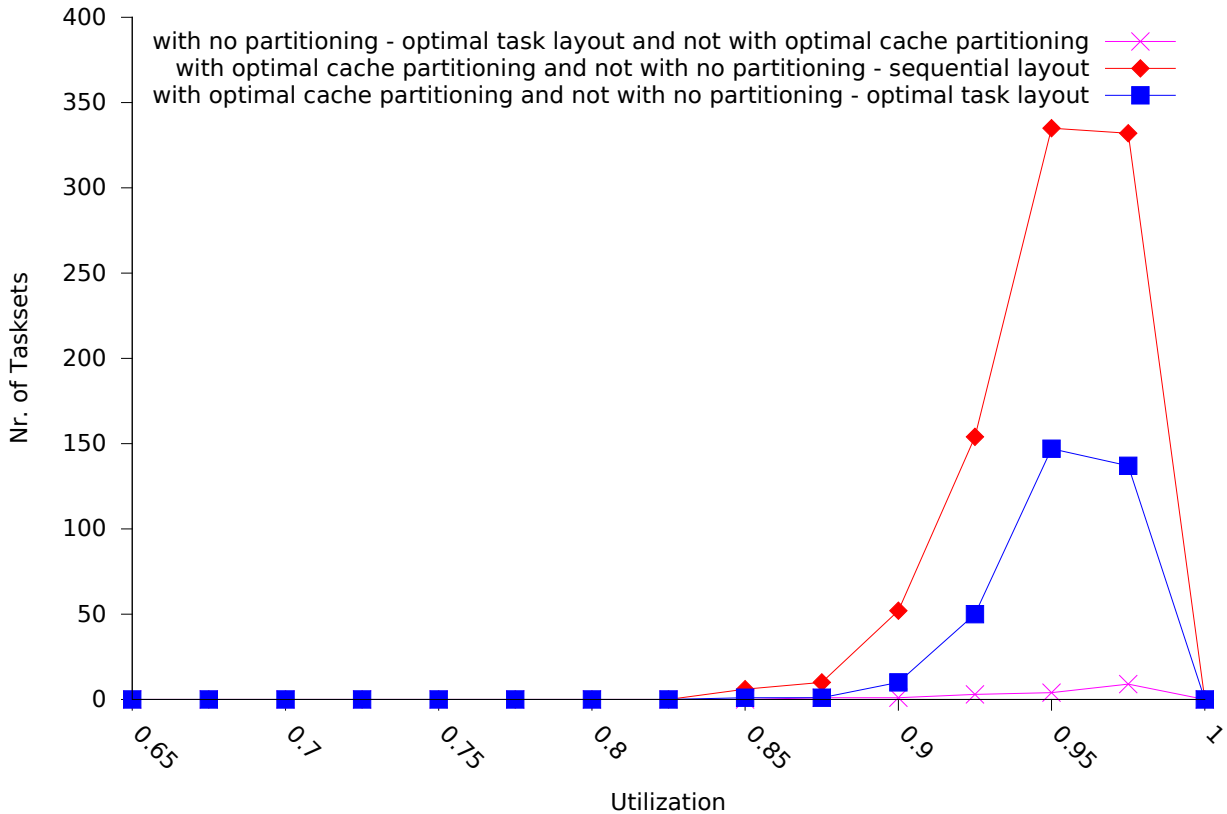
	Description	UCBs	ECBs	$WCET^1$	$WCET^2$
M	adpcm	7	242	$5,856s$	$43,17s$
M	compress	6	242	$9,740s$	$25,26s$
M	edn	5	98	$518,9ms$	$1,422s$
M	fir	5	50	$42,65ms$	$121ms$
M	jfdctinit	8	242	$23,2ms$	$73,63ms$
M	ns	3	26	$133,7ms$	$466,9ms$
M	nsichneu	8	242	$66,74ms$	$178,3ms$
M	statemate	30	242	$8,143s$	$22,45s$
S	cruise control system	15	98	$1,77s$	$6,207s$
S	flight control system	12	242	$3,24s$	$11,02s$
S	navigation system	3	82	$2,96s$	$7,566s$
S	stopwatch	9	130	$4,417s$	$25,03s$
S	elevator simulation	4	114	$1,863s$	$5,432s$
S	robotics systems	5	242	$3,427s$	$22,45s$

Data cache with perfect instruction cache ($WCET^1$), and without instruction cache ($WCET^2$)

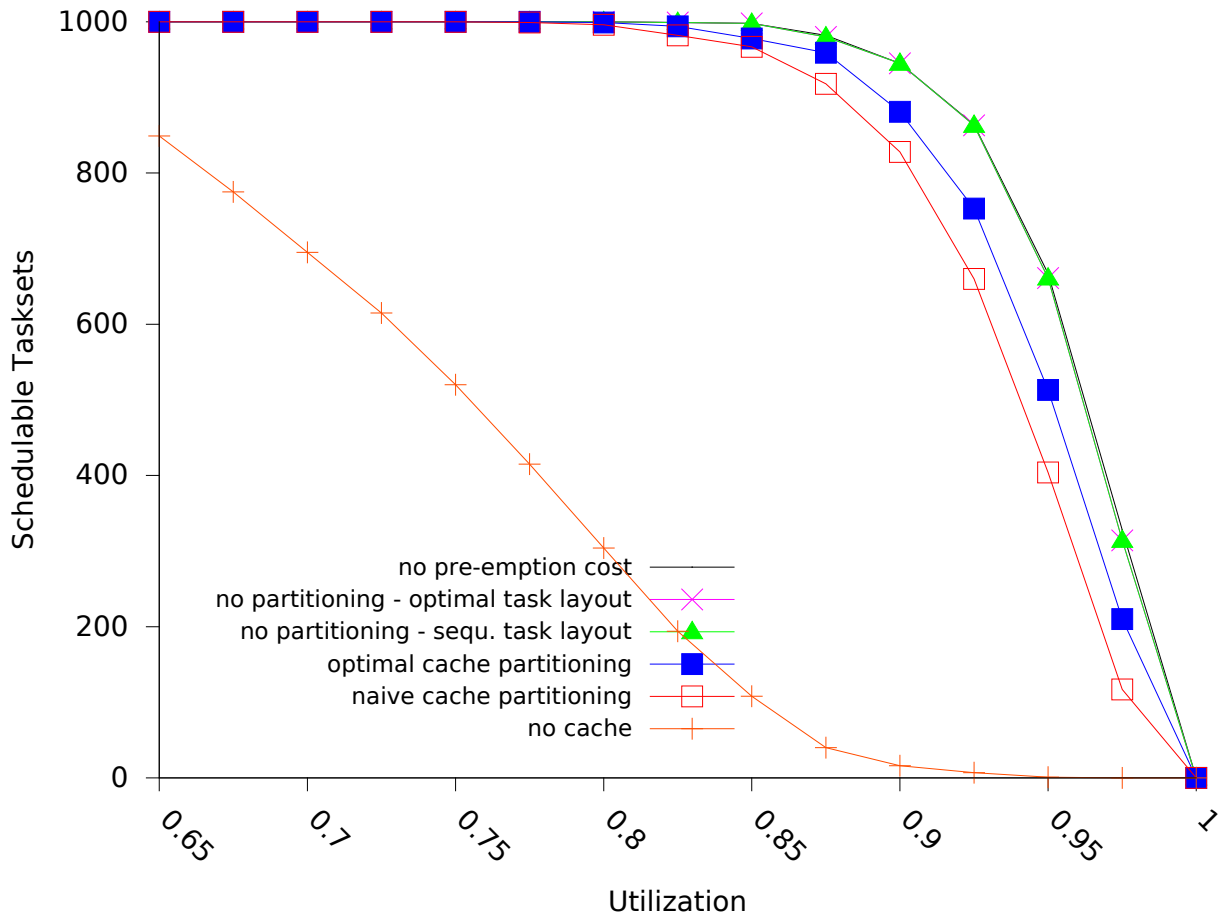
Data cache (perfect instruction cache) - PapaBench



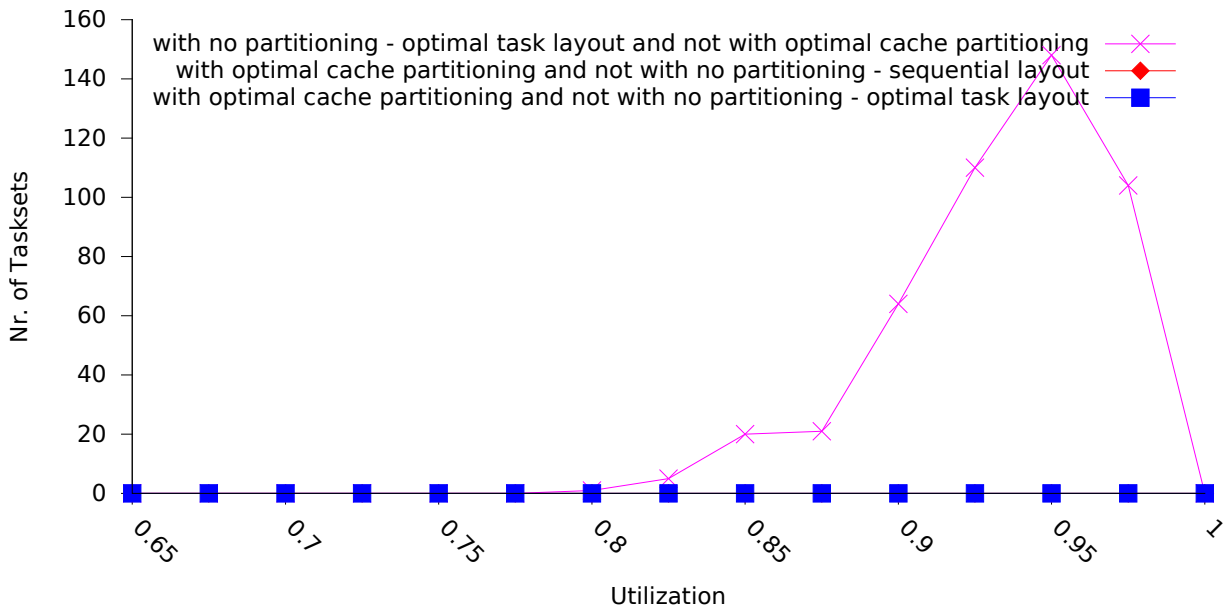
Data cache (perfect instruction cache) - PapaBench



Data cache (perfect instruction cache) - Mälardalen



Data cache (perfect instruction cache) - Mälardalen



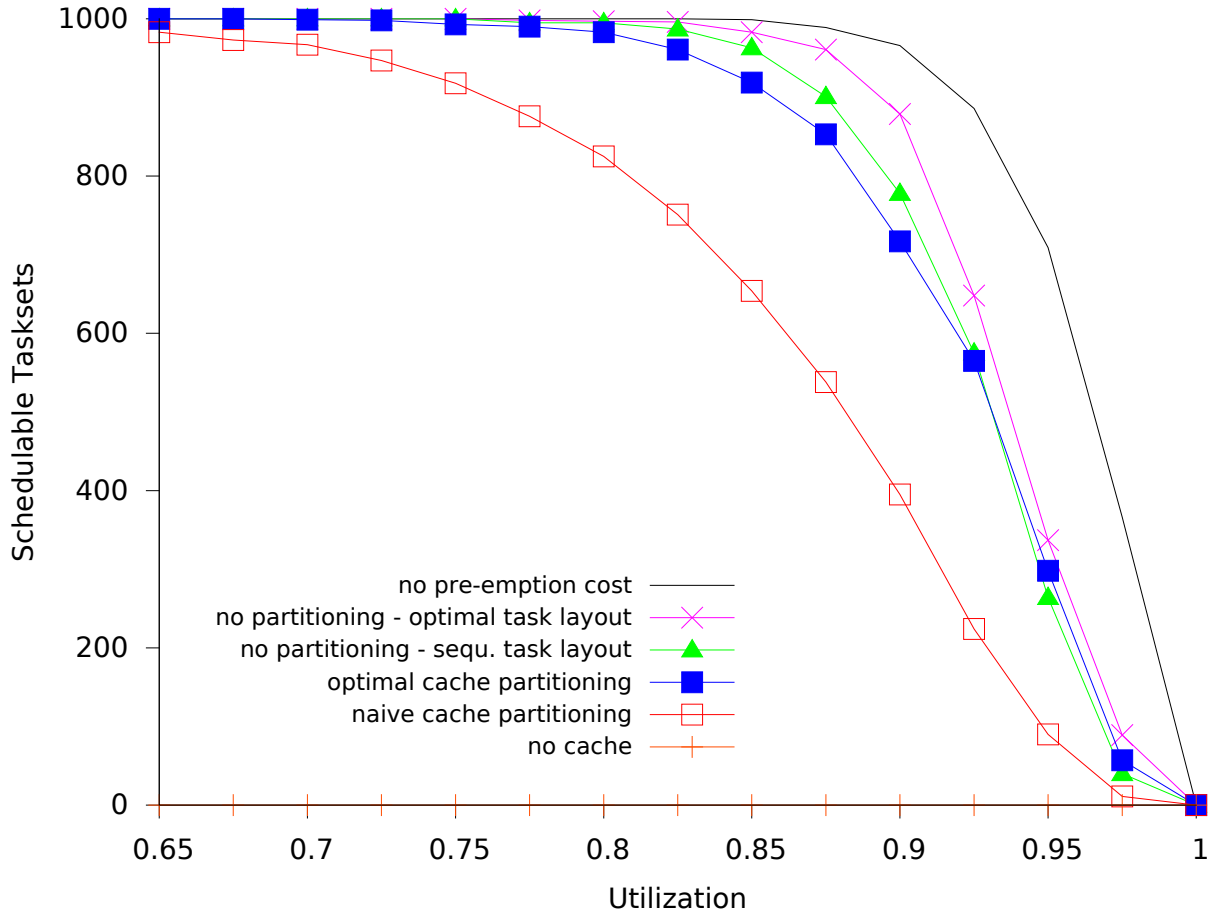
Synthetic Tasksets

Same configuration parameters, but in addition:

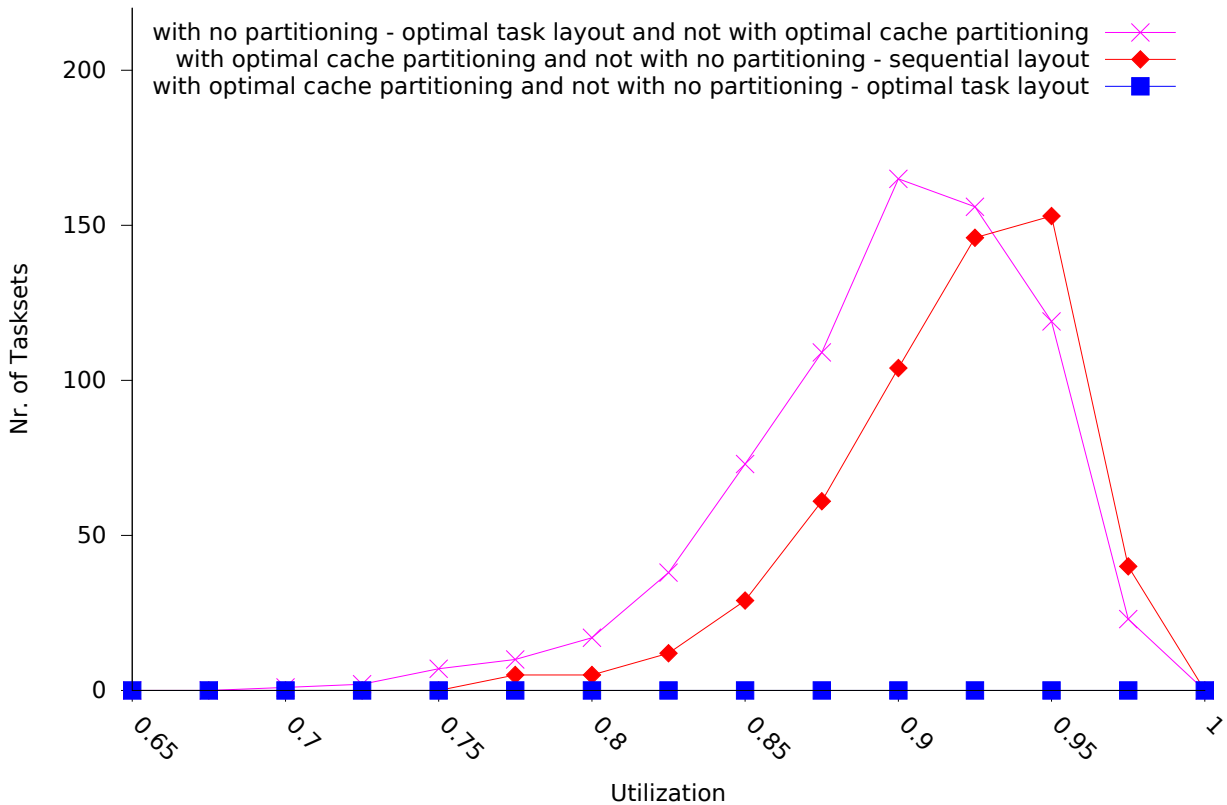
- task periods randomly selected from 5ms to 5s
- execution time variation according to our benchmarks
- cache usage using UUnifast [2] ($CU = \sum_i |\text{ECB}|/CS = 4$)
- UCBs randomly taken from range $[0, RF \cdot |ECB|]$ with $RF = 0.3$

Allows us to observe the dependency on different task set parameters

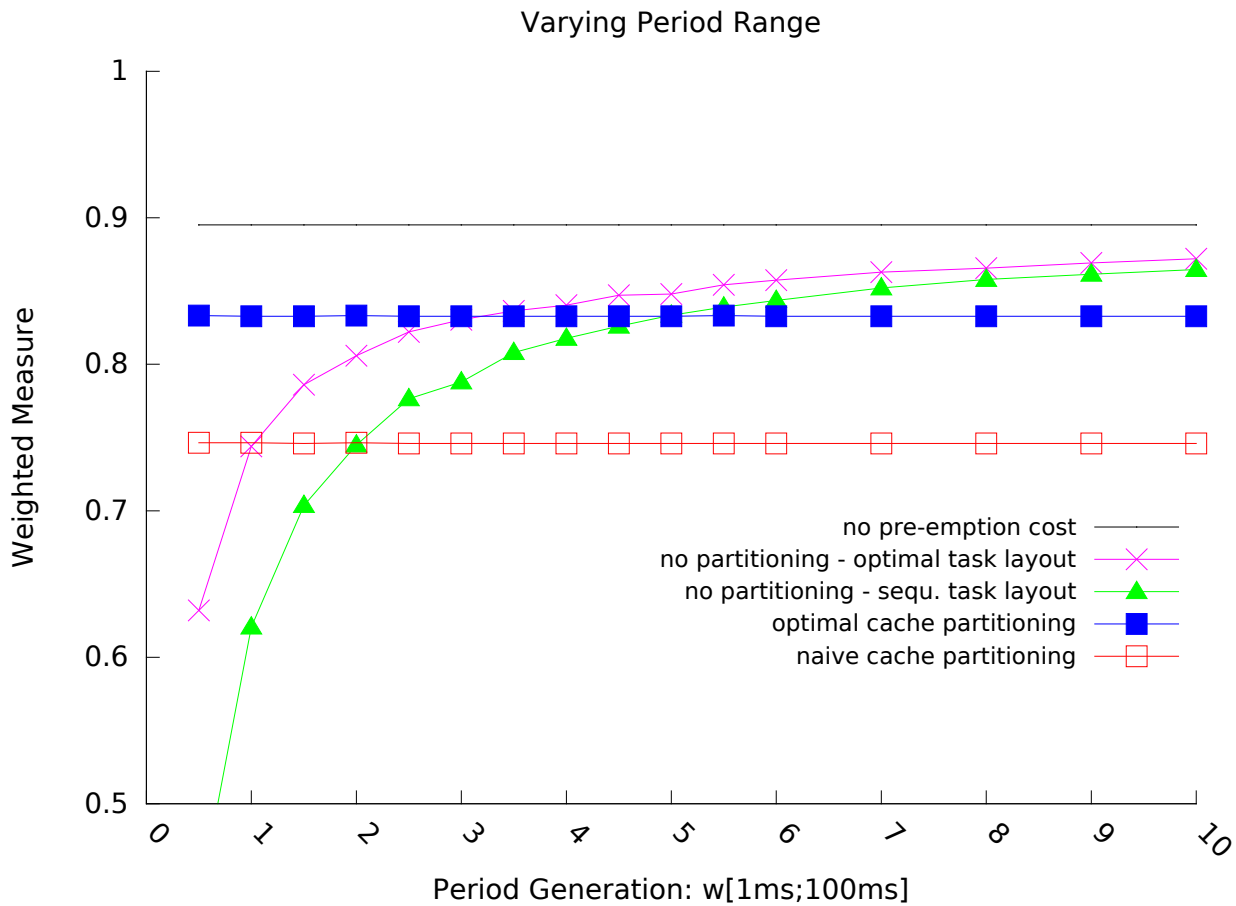
Evaluation for the base configuration



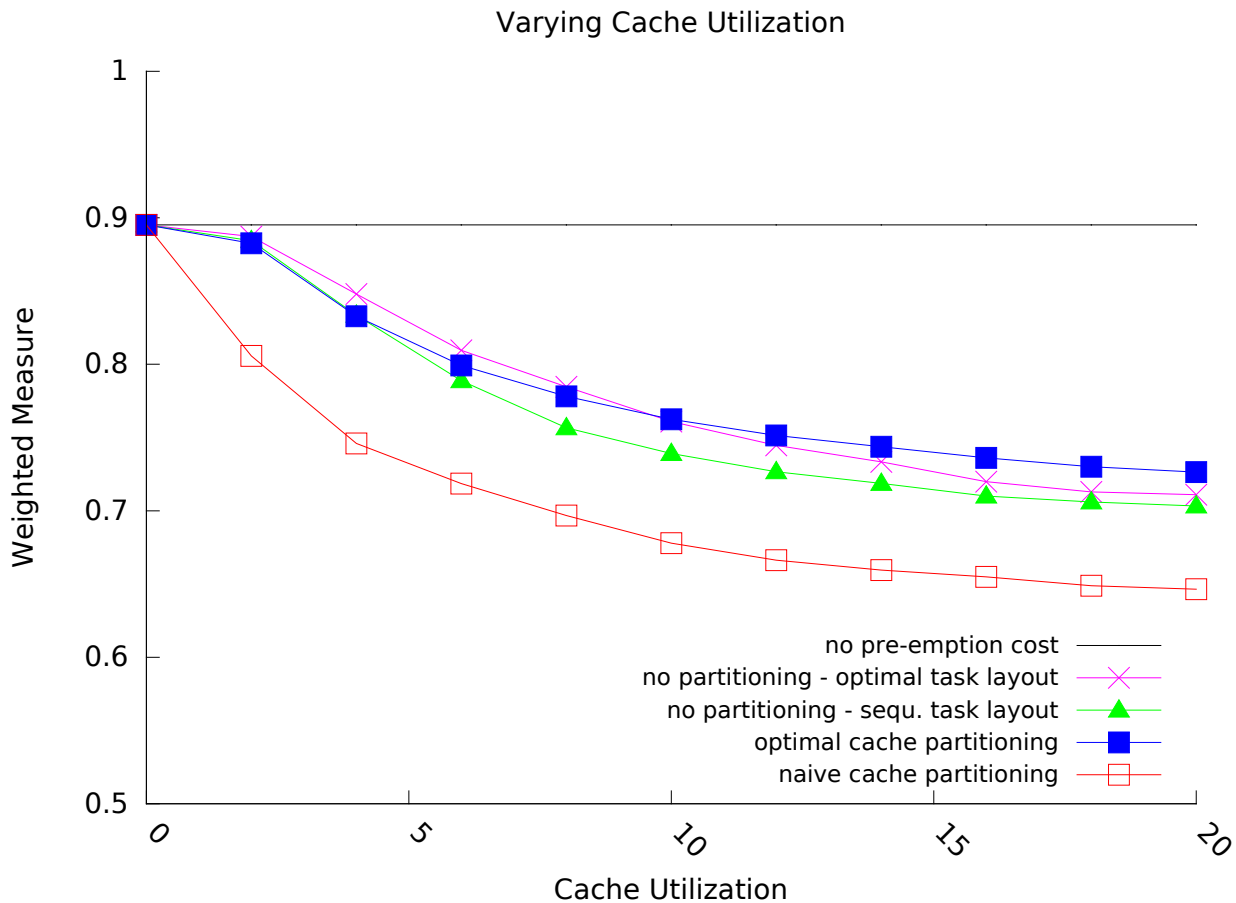
Evaluation for the base configuration



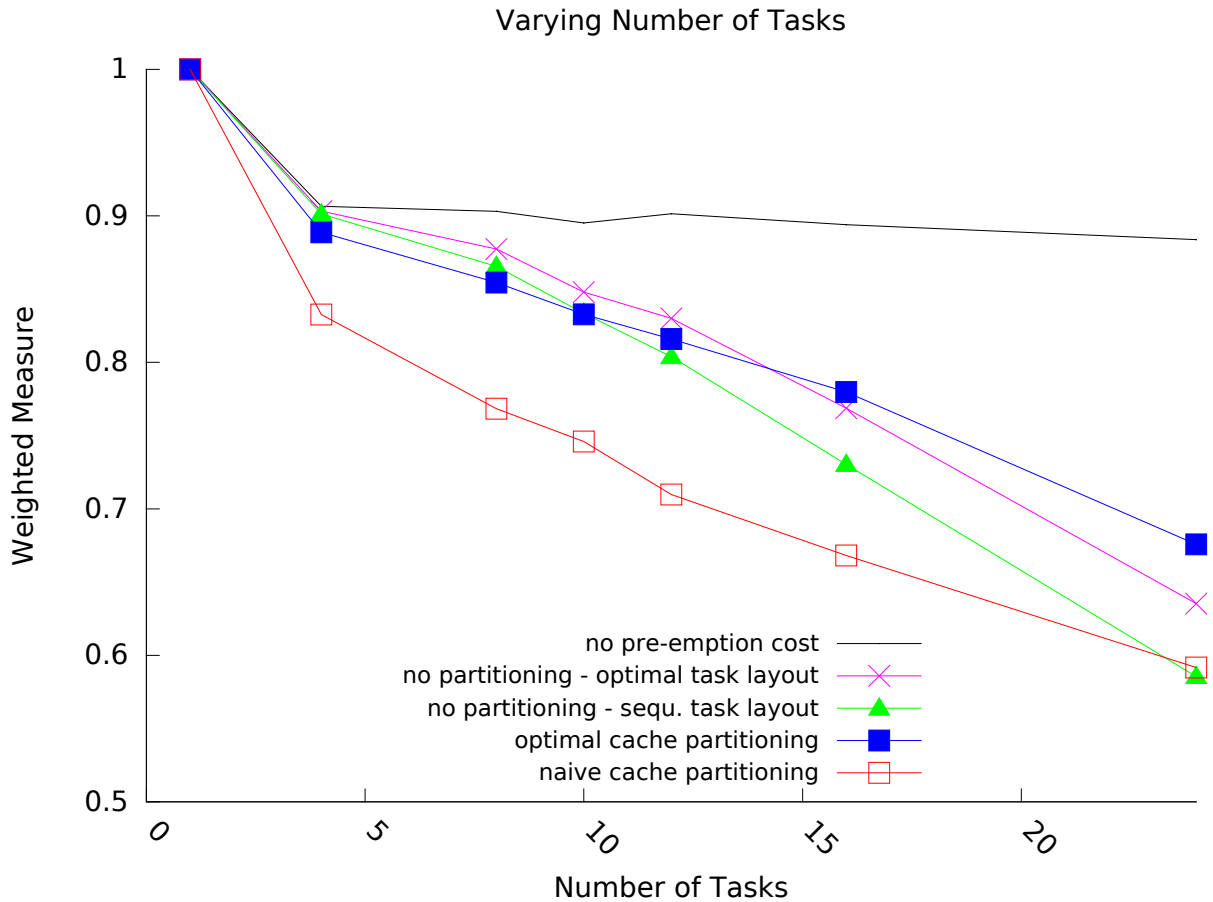
Varying the scale of task periods $w[1, 100]$



Varying cache utilization from 0 to 20

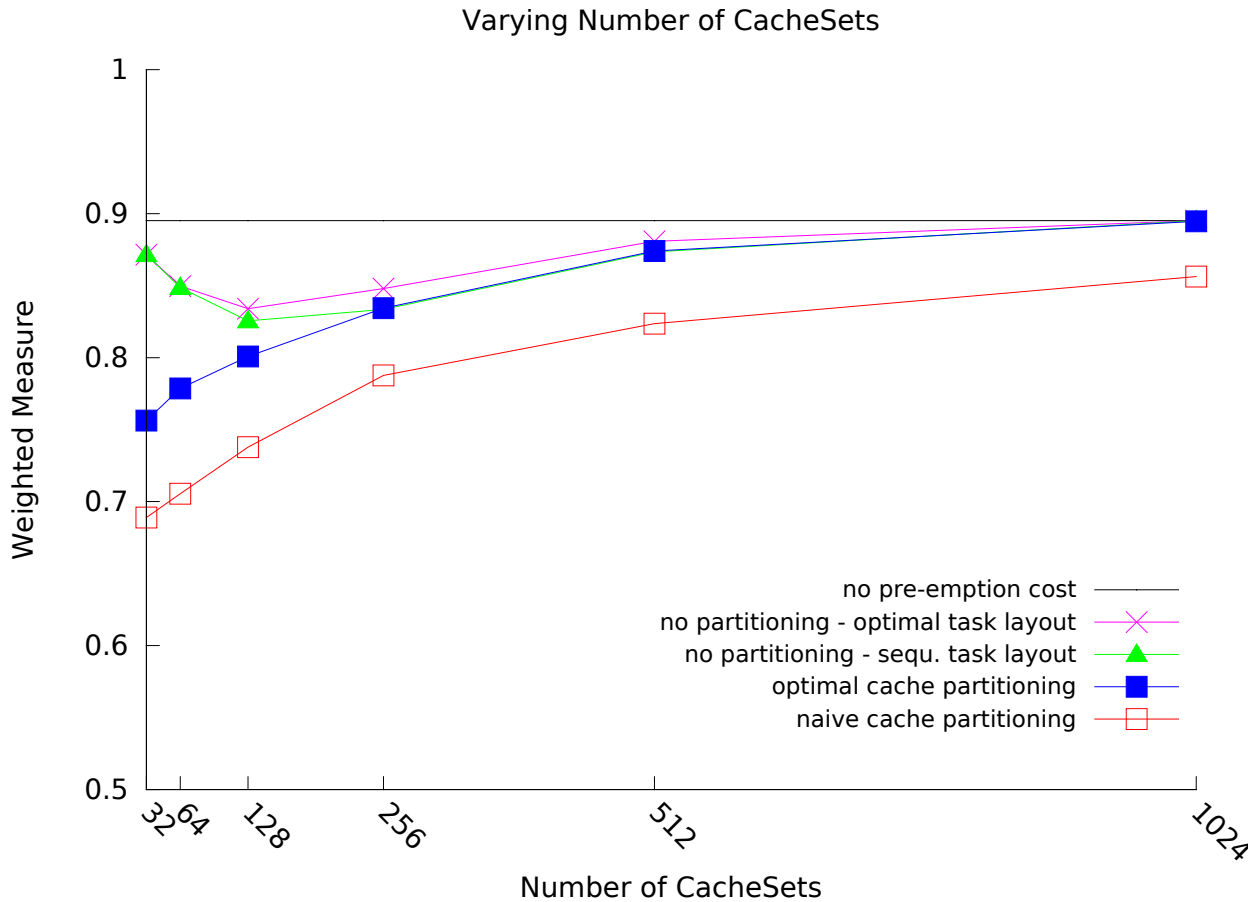


Varying the number of tasks from 2 to 24



(with constant ratio of number of tasks to cache usage)

Varying the number of cache sets 64 to 1024



(with constant ratio (CU/CS))

Varying block reload time from 1μs to 20μs