Analysis of Write-back Caches under Fixed-priority Preemptive and Non-preemptive Scheduling

Robert I. Davis[1,2], Sebastian Altmeyer[3] , Jan Reineke[4]

[1]Real-Time Systems Research Group, University of York, UK

[2]INRIA, Paris, France

[3]University of Amsterdam, Netherlands

[4]Saarland University, Saarland Informatics Campus, Saarbrücken, Germany

Overview

- **What is the presentation about?**
  - The integration of information from analysis of data caches using a write-back policy, such as:

    Dirty Cache Blocks (DCBs)

    Final Dirty Cache Blocks (FDCBs)

    Evicting Cache Blocks (ECBs)

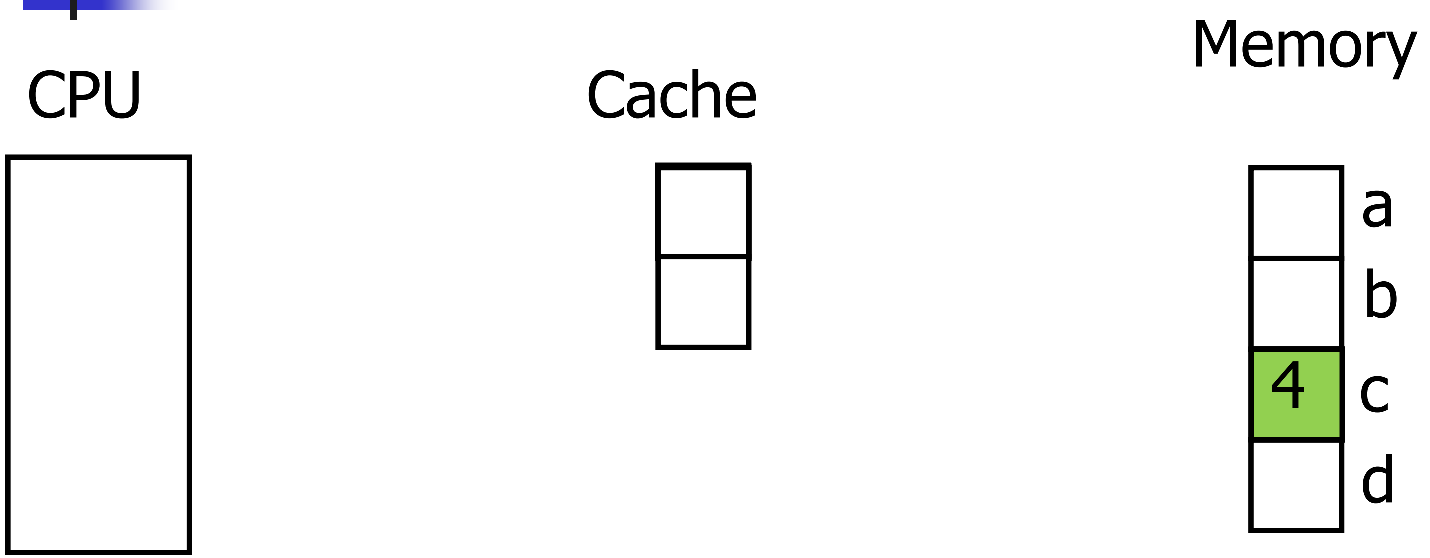
  into schedulability analysis for fixed priority preemptive (FPPS) and fixed priority non-preemptive (FPNS) scheduling
  - Aiming to account for the overheads of write backs in the schedulability analysis

- **What is it not about?**
  - The actual analysis of data caches that use a write-back policy to provide the information needed by schedulability analysis

Caches and memory

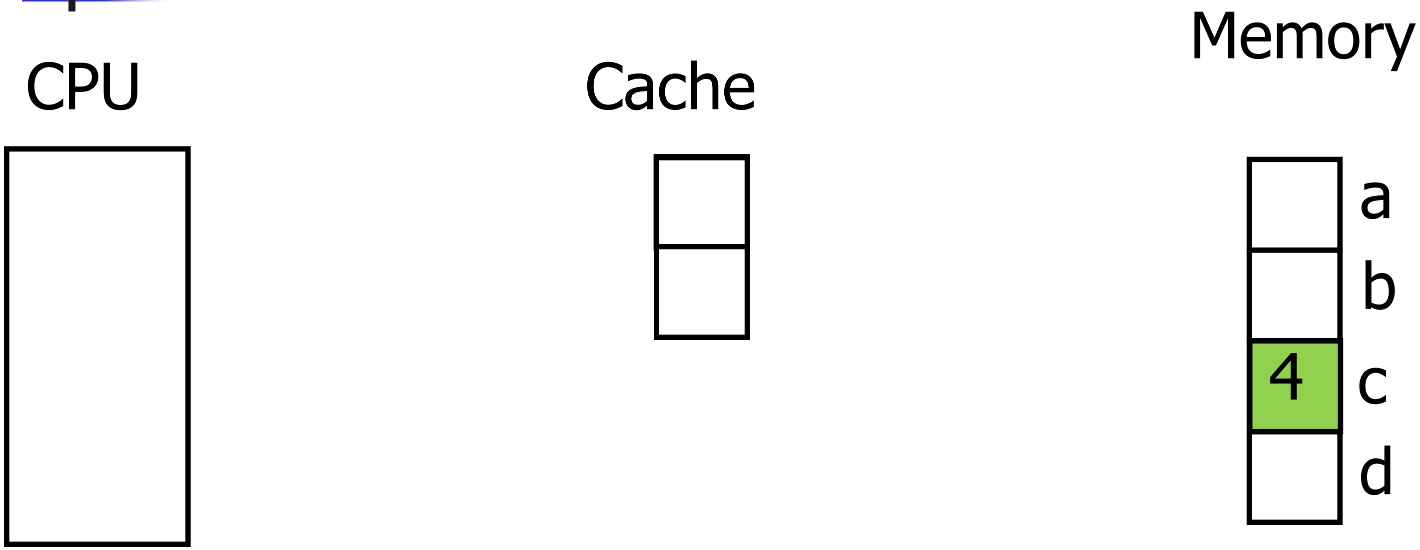
- **Main memory**
  - Slow to access (e.g. 10 – 100 clock cycles)
  - Logically divided into memory blocks (typically 32-128 bytes each)
- **Caches**
  - Small fast memories (e.g. 1 cycle) that bridge the gap in terms of speed between CPU and main memory
  - This paper considers direct mapped caches: different memory blocks can map to the same cache line, only action on a miss is to replace the memory block in the cache line
  - Interested in data caches and unified caches
- **Write Policies**
  - Write through and Write back

Write policies: write through



- **Key points: write through**
  - Write to memory requested at the same time as the write to cache
  - Results in many (unnecessary) accesses to memory when a memory block is written to multiple times without being evicted from cache
  - Can re-use a cache line (evicting contents) with no additional delay

Write policies: write back



- **Key points: write back**
  - Memory block is only written to memory when it is evicted
  - Multiple writes can take place efficiently to the cache (only)
  - Need to keep track of dirty cache lines which need to be written back
  - Write back can delay other read and write accesses

Classification of write backs

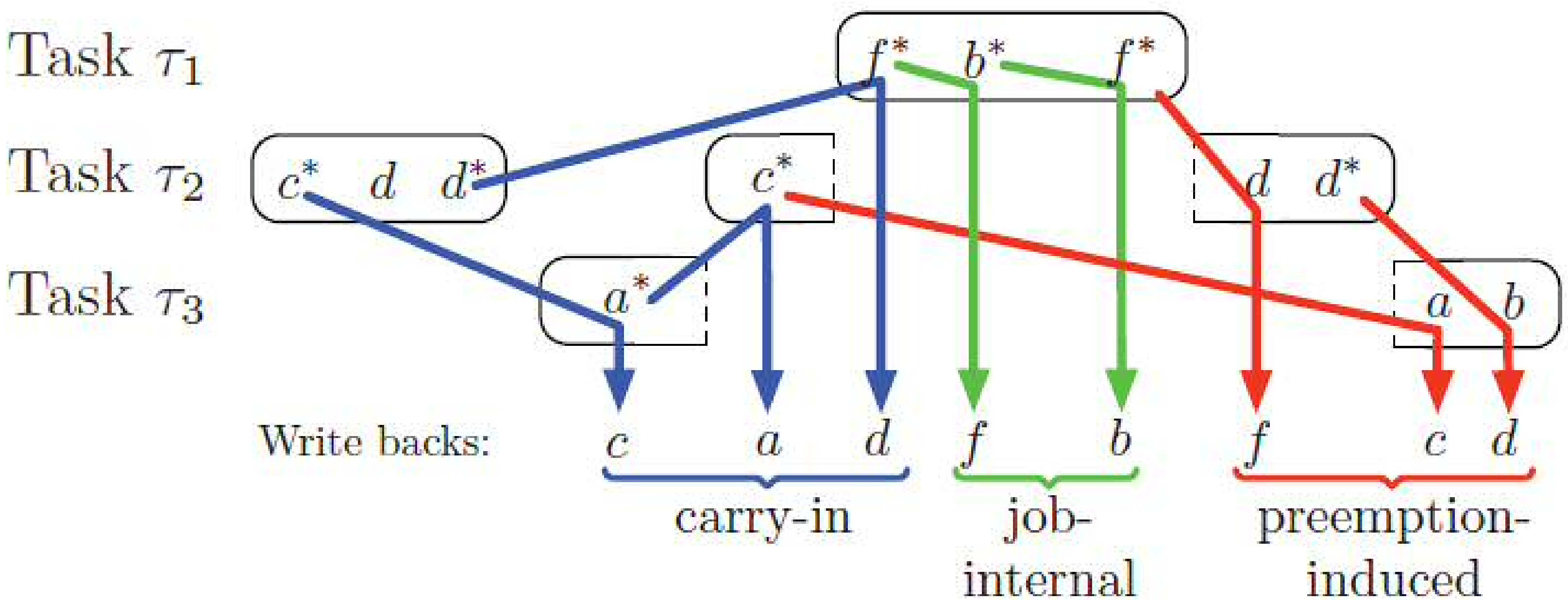
- **Job-internal write backs**
  - Write backs of dirty cache lines written by the same job
  - Assumed to be accounted for in WCET analysis
- **Carry-in write backs**
  - Write backs of dirty cache lines that were in the cache before the job started
  - **lp-carry-in** write backs from lower priority jobs that are still active
  - **finished-carry-in** write backs from lower or higher priority jobs that have finished
- **Preemption-induced write backs**
  - Write backs of dirty cache lines that were introduced by a preempting job (that has finished).

Classification of write backs

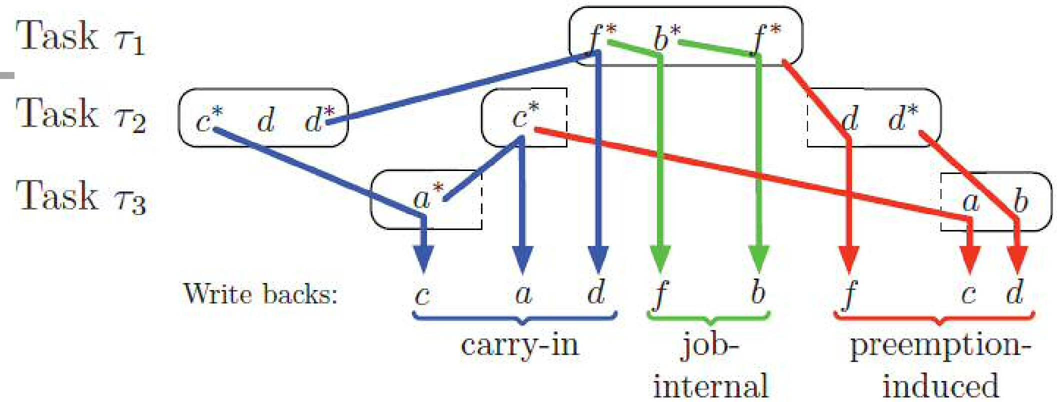
Example

Memory blocks a,c share a cache line as do blocks b,d,f

c* means a write access to block c



What information is needed to analyse write backs?



- **Evicting Cache Blocks (ECBs)**
  - Set of cache lines that the task touches (reads or writes) during execution
- **Dirty Cache Blocks (DCBs)**
  - Set of cache lines that the task writes to at some point in its execution and could as a result be dirty when the task is preempted
- **Final Dirty Cache Blocks (FDCBs)**
  - Set of cache lines that the task writes at some point in its execution that could as a result be dirty when the task finishes execution

Task model

- **Sporadic task model**
  - Static set of *n* tasks τ_i with priorities 1..*n*
  - Worst-Case Execution Time C_i assuming non-preemptive scheduling starting from an empty (clean) cache (includes job-internal write backs)
  - Sporadic/periodic arrivals: minimum inter-arrival time T_i
  - Relative deadline D_i (constrained $D_i \leq T_i$)
  - Response time R_i
- **Scheduling policies**
  - Fixed Priority Preemptive Scheduling (FPPS)
  - Fixed Priority Non-preemptive Scheduling (FPNS)

Write backs under FPPS

- **FPPS (exact test)**

$$R_i^P = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^P}{T_j} \right\rceil C_j$$

- **Extended schedulability analysis**

$$R_i^P = \underline{\delta_i} + C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil (C_j + \underline{\gamma_{i,j}^{\text{miss}}} + \underline{\gamma_{i,j}^{\text{wb}}})$$

  - $\underline{\delta_i}$ write backs due to initially dirty cache lines (at start of busy period)
  - $\underline{\gamma_{i,j}^{\text{miss}}}$ accounts for CRPD
  - $\underline{\gamma_{i,j}^{\text{wb}}} = \gamma_{i,j}^{\text{wb-lp}} + \gamma_{i,j}^{\text{wb-fin}}$ lp-carry-in and finished-carry-in and preemption induced write backs

Write backs under FPPS

$$R_i^P = \delta_i + C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil (C_j + \gamma_{i,j}^{\text{miss}} + \gamma_{i,j}^{\text{wb}})$$

$$\gamma_{i,j}^{\text{wb}} = \gamma_{i,j}^{\text{wb-lp}} + \gamma_{i,j}^{\text{wb-fin}}$$

- **Initially dirty cache lines**
  - Due to pre-empted lower priority jobs and due to finished higher priority tasks (and previous job of task τ_i)

$$\delta_i = WBT \cdot \left| \left(\bigcup_{j \in lp(i)} DCB_j \cup \bigcup_{k \in hep(i)} FDCB_k \right) \cap \left(\bigcup_{k \in hep(i)} ECB_k \right) \right|$$

- **Finished-carry-in and preemption induced write backs**
  - Left by jobs that complete during the busy period

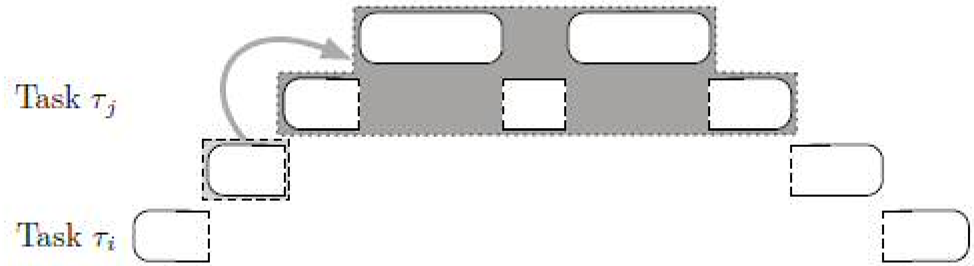
$$\gamma_{i,j}^{\text{wb-fin}} = WBT \cdot |FDCB_j|$$

Write backs under FPPS

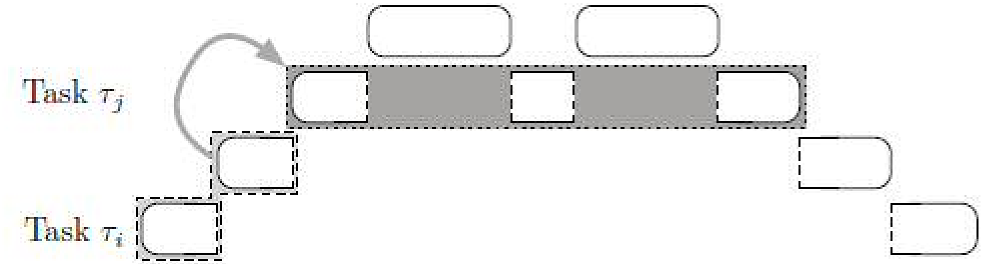
$$R_i^P = \delta_i + C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil (C_j + \gamma_{i,j}^{\text{miss}} + \gamma_{i,j}^{\text{wb}})$$

$$\gamma_{i,j}^{\text{wb}} = \gamma_{i,j}^{\text{wb-lp}} + \gamma_{i,j}^{\text{wb-fin}}$$

- **Lower priority carry-in write backs due to preempted tasks Two ways of accounting for these:**
  - (a) Write backs due to dirty cache lines introduced by the job immediately preempted by task τ_j that occur at some point within the response time of τ_j

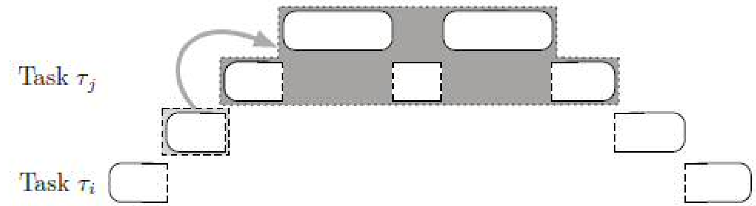


  - (b) Write backs due to dirty cache lines introduced by any (nested) preempted lower priority task(s) that occur within the execution of task τ_j



THE UNIVERSITY of York | UNIVERSITY OF AMSTERDAM | UNIVERSITÄT DES SAARLANDES | Inria INVENTORS FOR THE DIGITAL WORLD | RTS York

FPPS: lp-carry-in method (a)



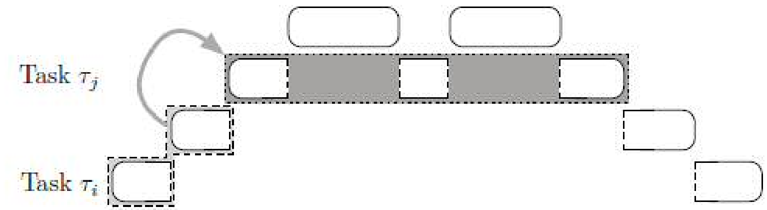
- **DCB-Only**
  - Any task that is active in the busy period and of lower priority than task τ_j i.e in $aff(i,j) = hep(i) \cap lp(j)$ could be the immediately preempted task

$$\gamma_{i,j}^{\text{wb-lp}} = WBT \cdot \max_{h \in aff(i,j)} |DCB_h|$$

- **ECB-Union**
  - Refines DCB-Only approach by only including write backs that could happen due to evictions by tasks that can execute during the response time of τ_j

$$\gamma_{i,j}^{\text{wb-lp}} = WBT \cdot \max_{h \in aff(i,j)} \left| DCB_h \cap \bigcup_{l \in hep(j)} ECB_l \right|$$

FPPS: lp-carry-in method (b)



- **ECB-Only**
  - Lp-carry-in write backs introduced by any (nested) preempted lower priority task(s) written back by task τ_j are upper bounded by the ECBs of task τ_j

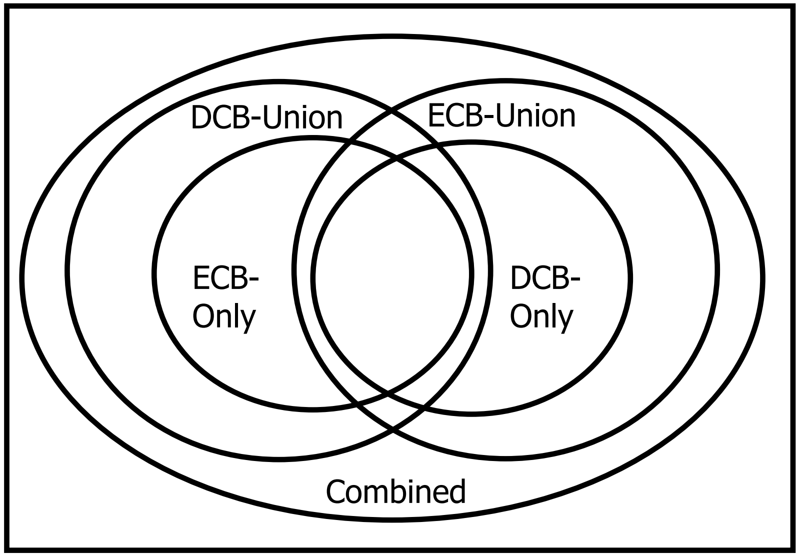
$$\gamma_{i,j}^{\text{wb-lp}} = WBT \cdot |ECB_j|$$

- **DCB-Union**
  - Refines ECB-only by noting that we are only interested in write backs of dirty cache lines introduced by *preempted lower priority* tasks

$$\gamma_{i,j}^{\text{wb-lp}} = WBT \cdot \left| \left(\bigcup_{h \in aff(i,j)} DCB_h \right) \cap ECB_j \right|$$

FPPS approaches

- **Dominance relations**
  - ECB-Union dominates DCB-Only
  - DCB-Union dominates ECB-Only
  - DCB-Union and ECB-Union incomparable
  - Combined approach more effective than DCB-Union and ECB-Union since it is applied on a per task basis

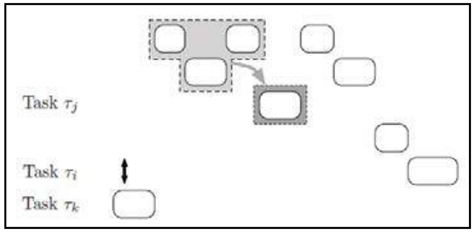


Worked examples showing these relations in the technical report

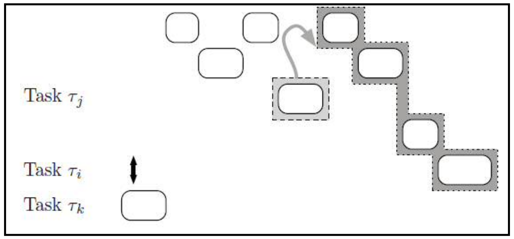
Write backs under FPNS: four approaches

- **ECB-only**
  - Number of write backs upper bounded by ECBs of the job
- **FDCB-Union**
  - Improves upon ECB-only by accounting for which cache lines may be dirty when a task executes
- **FDCB-Only**
  - Covers write backs in subsequent jobs due to dirty cache lines left by task that run during the busy period or before it starts
- **ECB-Union approach**
  - Improves upon FDCB-only by accounting for the dirty cache lines which may actually be evicted

Method (a)



Method (b)



Details of all 4 approaches in the paper

Similar dominance and incomparability relationships to FPPS

Evaluation: write back v. write through

- **Benchmarks**
  - Code from Mälardalen and EEMBC benchmark suites
  - Compiled using ARM cross compiler
  - Traces generated using gem5 instruction set simulator
  - Bounds for ECBs, DCBs, FDCBs obtained from traces via cache simulation
  - Assume 1 cycle for cache hit, 10 cycles for cache miss / write back
  - Separate Instruction and Data Caches (each of 512 lines, 32 bytes per line)
- **Task set generation**
  - Random choice of benchmark to represent each task's code
  - Utilisations chosen using UUnifast
  - Task periods set based on U_i and WCET for write back cache
  - Enables generation of a large number of task sets with different utilisations based on limited benchmarks

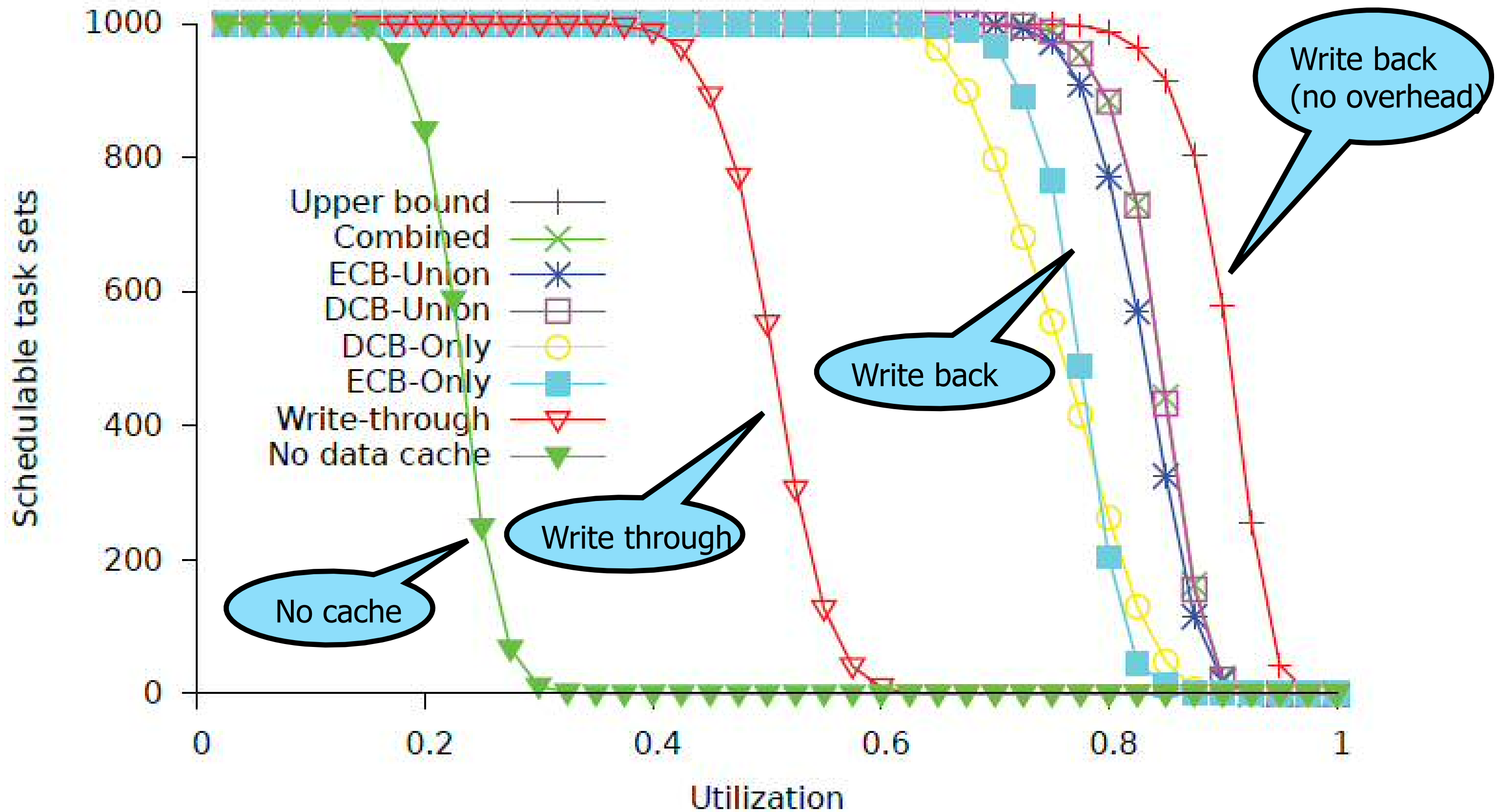
Evaluation data

- **Benchmarks**
  - Different WCETs for write back and write through
  - Write back has WCETs a factor of 1.28 to 3.02 better than write through
  - UCBs, ECBs, DCBs, FDCBs (instruction and data caches)

Name	C^{wb}	C^{wt}	C^{wt}/C^{wb}	C^{nc}	C^{nc}/C^{wb}	$\lvert UCB^I \rvert$	$\lvert ECB^I \rvert$	$\lvert UCB^D \rvert$	$\lvert ECB^D \rvert$	$\lvert DCB \rvert$	$\lvert FDCB \rvert$
cnt	9325	13485	1.44	24565	2.63	12	82	21	68	28	28
compress	10673	18713	1.75	43443	4.07	21	71	53	103	60	60
countneg	36180	57250	1.58	114340	3.16	15	77	59	103	66	66
crc	68889	133909	1.94	272859	3.96	19	89	25	73	40	39
expint	9268	15208	1.64	31098	3.35	16	76	11	42	13	13
fdct	7883	16793	2.13	38423	4.87	52	144	15	48	19	19
fir	8328	18998	2.28	43668	5.24	22	83	17	57	17	16
jfdctint	9711	18621	1.91	39181	4.03	46	145	17	53	23	23
loop3	14189	28729	2.02	57929	4.08	7	309	9	42	12	12
ludcmp	10058	15948	1.58	39668	3.94	38	128	21	61	28	28
minver	18976	30616	1.61	54746	2.88	103	213	18	71	33	33
ns	27464	37674	1.37	98634	3.59	14	70	9	116	13	11
nsichneu	18988	24458	$\boxed{1.28}$	66808	3.51	345	494	52	95	54	53
qurt	10473	16003	1.52	23573	2.25	61	132	14	49	17	17
select	8981	17031	1.89	30331	3.37	47	124	10	49	16	16
sqrt	27667	40537	1.46	59117	2.13	51	102	11	48	16	16
statemate	64638	195778	$\boxed{3.02}$	581908	9.00	92	167	25	68	21	20
a2time	12655	22975	1.81	53815	4.25	16	122	8	100	69	67
aifirf	44898	86768	1.93	181698	4.04	25	141	33	188	161	54
basefp	50491	92221	1.82	213771	4.23	11	88	15	512	507	467
canrdr	32641	65211	1.99	156611	4.79	8	40	9	371	195	186
iirflt	29995	56995	1.90	127605	4.25	35	288	28	259	147	138
pntrch	23887	43137	1.80	109257	4.57	24	38	20	237	176	70
puwmod	48782	97072	1.98	239752	4.91	3	50	5	512	307	275
rspeed	10913	21393	1.96	51713	4.73	8	53	7	122	71	70
tblook	12533	25493	2.03	58813	4.69	12	115	14	125	71	71

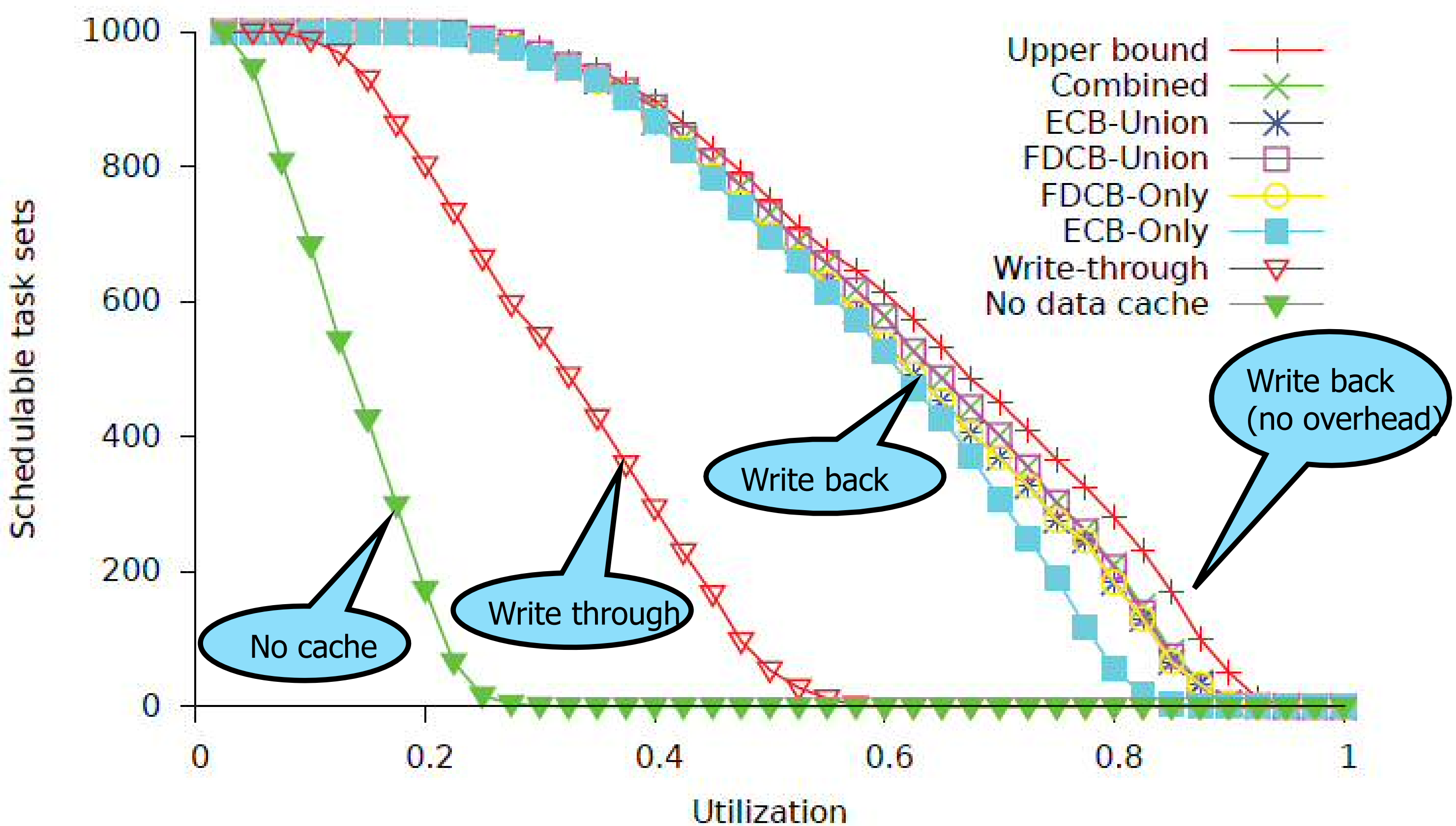
Evaluation: results for FPPS

Approach	FPPS	FPNS
Write-back (upper bound)	0.793458	0.445750
Combined	0.693003	0.412270
(F)DCB-Union	0.692087	0.411087
ECB-Union	0.672489	0.396159
(F)DCB-Only	0.561542	0.396159
ECB-Only	0.581876	0.365523
Write-through	0.249231	0.112666
No data cache	0.052548	0.021463



Evaluation: results for FPNS

Approach	FPPS	FPNS
Write-back (upper bound)	0.793458	0.445750
Combined	0.693003	0.412270
(F)DCB-Union	0.692087	0.411087
ECB-Union	0.672489	0.396159
(F)DCB-Only	0.561542	0.396159
ECB-Only	0.581876	0.365523
Write-through	0.249231	0.112666
No data cache	0.052548	0.021463

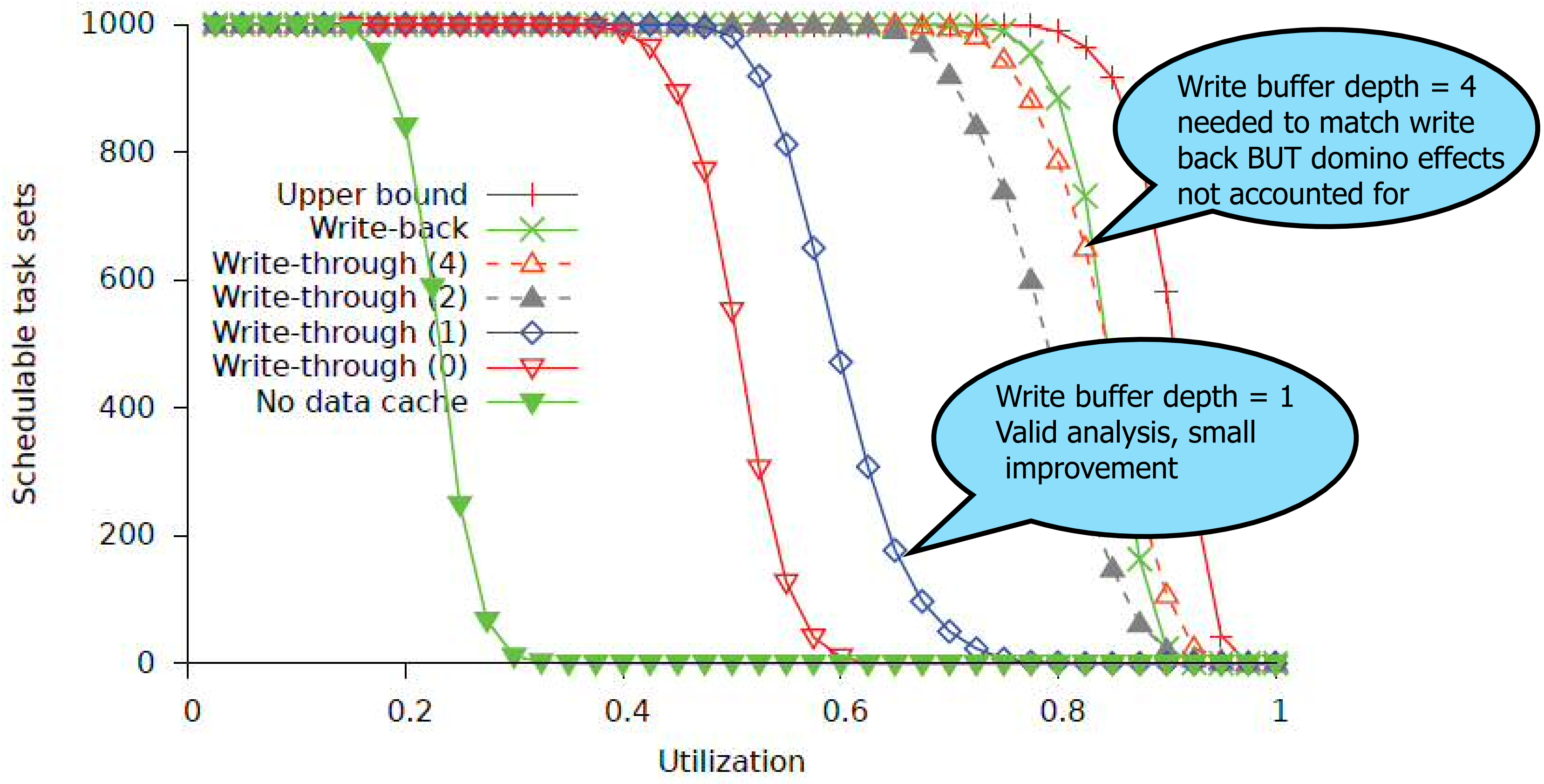


Write buffers (technical report)

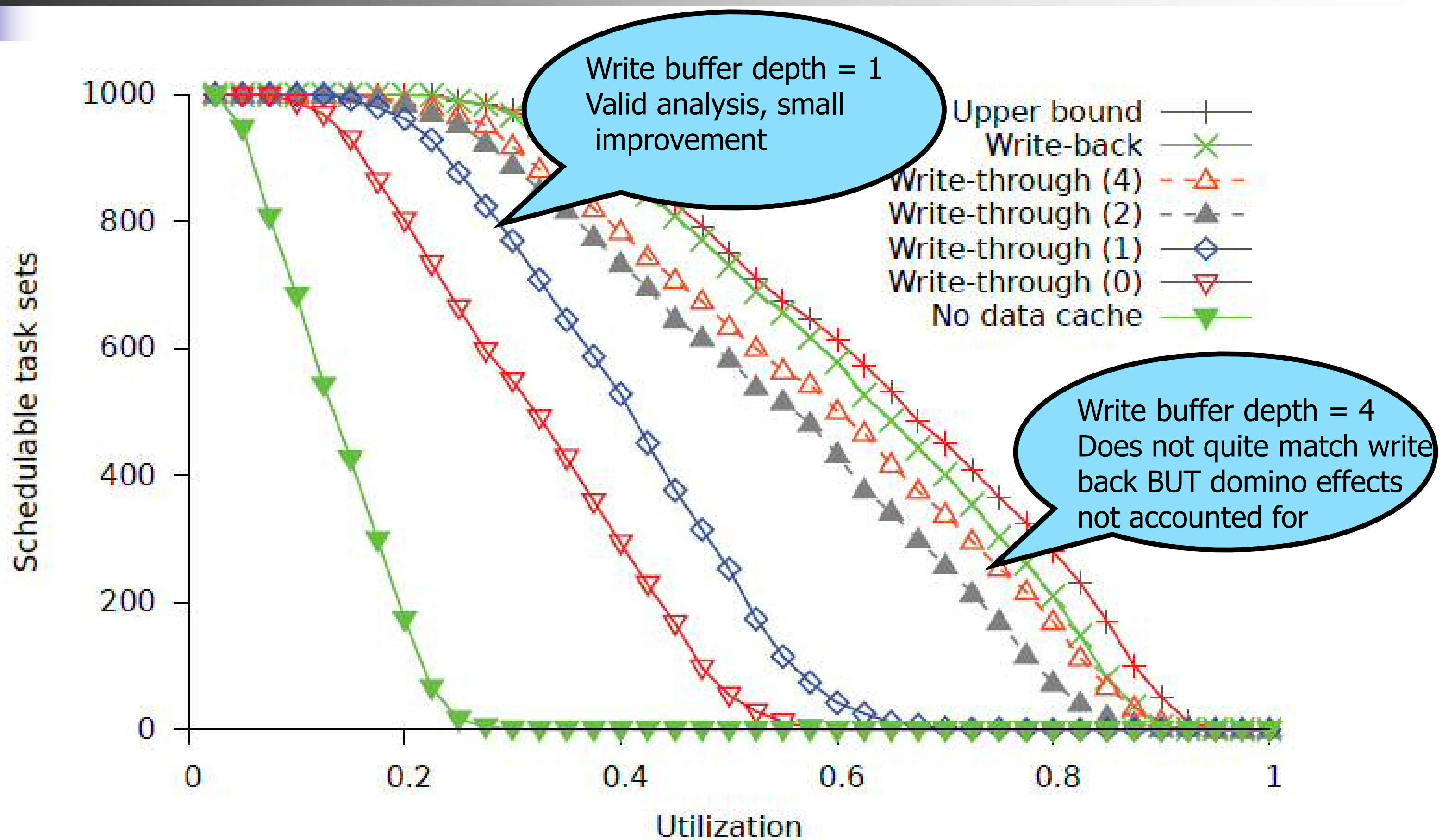
- **Latency hiding**
  - Write buffer can hide the write latency with write-through caches (and write-back caches)
- **Behaviours**
  - Lazy / eager retirement
  - Read from write buffer /flush
  - Write merge / no merge
- **Domino effects**
  - Small change in memory access sequence can cause an unbounded increase in total latency for an arbitrarily long sequence of accesses
  - Examples showing how domino effects can occur with write buffers (similar to FIFO caches)

Details in the technical report: https://www.cs.york.ac.uk/ftpdir/reports/2016/YCS/502/YCS-2016-502.pdf

Write buffers: evaluation: FPPS



Write buffers: evaluation: FPNS



Summary

- **What we have done**
  - Classified different types of write back and the information needed from cache analysis (ECBs, DCBs, FDCBs)
  - Integrated information from analysis of write back caches into schedulability analysis for FPPS and FPNS:
    4 methods and combined approaches for each
  - Demonstrated the effectiveness of the analysis via evaluation using multiple benchmarks
    - WCET with write back 1.2 to 3.0 times lower than with write through (0.98 to 1.98 compared to write through with a write buffer of depth 1)
      Showed that write buffers can result in domino effects
    - Analysable overheads of write backs were small – little degradation compared to upper bound assuming no write back cost.

Improvement in WCET more than compensates for overheads

Analysable performance of write back cache was significantly better than write through

Open issues

- **Difficulty in precisely analysing write back caches**
  - Our **proof of concept** evaluation used simple benchmarks with fixed inputs, this enabled analysis of ECBs, DCBs, FDCBs via traces and cache simulation
  - More complex software requires the use of static analysis
  - Assuming critical real-time software can expect minimal use of pointers, no recursion, statically allocated data structures, fixed stack location for each calling context, hence many memory accesses can be resolved
  - Difficulties remain in resolving memory accesses inside loops – could potentially be addressed via virtual loop unrolling
  - input data dependent locations cannot be resolved, leads to imprecision in ECBs, DCBs, FDCBs

Note review of prior work in the technical report

Future work

- **Handling Imprecision in ECBs, DCBs, FDCBs**
  - Inevitably there will be degrees of imprecision dependent on the actual code
  - One challenge is to handle this uncertainty without incurring significant or unbounded pessimism
  - Analysis needs to be adapated to this challenge
- **Set-associative caches**
  - Analysis in the paper is for direct mapped caches – extension needed to set-associative LRU caches

Questions?