The Influence of the Replacement Strategy on the Static Prediction of Cache Contents

Reinhold Heckmann

AbsInt
Angewandte Informatik GmbH

Overview:

Worst Case Execution Time (WCET)

- Controllers in planes, cars, plants, ... are expected to finish their tasks within reliable time bounds.
- Hence it is essential that the Worst-Case Execution Time (WCET) of all program tasks is known.

Static program analysis:

- Determine safe upper bounds for the execution time statically for all inputs.
- Need to consider cache and pipeline to get safe and tight upper bounds.

Cache Memory

A cache is a small high-speed memory coupled closely to the processor. It is used to duplicate a subset of memory locations.

For the purpose of caching, the main memory is partitioned into memory blocks of B bytes (ColdFire: $B = 2^4 = 16$, PowerPC: $B = 2^5 = 32$).

When the CPU wants to access block m, it checks whether the cache contains (a copy of) m.

- If so (cache hit), m is accessed in the cache, without involving main memory.
- If not (cache miss), m is copied from main memory to the cache and then accessed there.

Hits are much quicker than misses.

A-way Set Associative Caches

An A-way set associative cache consists of S cache sets (CF and PPC: $S = 2^7 = 128$) where each set consists of A cache lines (ColdFire: $A = 2^2 = 4$, PowerPC: $A = 2^3 = 8$).

Each line consists of

- a valid bit telling whether the line is in use,
- a tag identifying the memory block in the line,
- space for a memory block.

Each block can only be stored in a fixed set.
The cache controller splits 32 bit addresses into

	CF	PPC
• byte number in block	4 bits	5 bits
• set number s	7 bits	7 bits
• tag t	21 bits	20 bits

and then looks in set s for a valid line with tag t.

Update of the Cache Contents

When a memory block m with set number s is accessed, the cache contents change as follows:

1. If m is already in the cache (hit), the cache contents remain unchanged.
2. If not (miss), m is transferred to the cache. It is stored in one of the lines of set s.
   - 2.1. If there is an empty (invalid) line in s, m is placed in the first such line (allocate).
   - 2.2. If all lines of s are full, a replacement logic chooses the line where m is put. The block that is already in this line is removed from the cache (replace).

Common replacement strategies:

- LRU (least recently used)
- FIFO (first-in first-out)
- ColdFire MCF 5307: Pseudo Round Robin
- PowerPC MPC 755: Pseudo LRU

LRU Strategy

Each cache set has its own replacement logic
$\Rightarrow$ cache sets are independent.

In the sequel, we describe only one cache set.
When speaking of the cache, we actually mean this set.

LRU Replacement Strategy:

Replace the block that has been
Least Recently Used.
This can be modeled by ages.

Example (4-way set associative):

age	0	1	2	3
	m_0	m_1	m_2	m_3
access m_4 (miss)	m_4	m_0	m_1	m_2
access m_1 (hit)	m_1	m_4	m_0	m_2
access m_5 (miss)	m_5	m_1	m_4	m_0

Observation:

A block that is no longer accessed
will be removed after at most 4 misses.

Concrete Update Function (LRU)

Let $\mathcal{A} = \{0, \ldots, 3\}$ be the set of ages,
and $\mathcal{A}' = \mathcal{A} \cup \{\top\}$ ($\top$ means not in cache).

Concrete cache state: $c : \mathcal{M} \to \mathcal{A}'$
where $\mathcal{M}$ is the set of memory blocks.

Update when block m_0 is accessed in state c:
Result $up(m_0)(c) = c'$ defined by

$$c'(m) = \begin{cases} 0 & \text{if } m = m_0, \\ up_{\mathcal{A}}\,(c(m_0))\,(c(m)) & \text{otherwise} \end{cases}$$

using the age update function $up_{\mathcal{A}} : \mathcal{A}' \to (\mathcal{A}' \to \mathcal{A}')$:

$up_{\mathcal{A}}$	0	1	2	3	$\top$
0	–	1	2	3	$\top$
1	1	–	2	3	$\top$
2	1	2	–	3	$\top$
3	1	2	3	–	$\top$
$\top$	1	2	3	$\top$	$\top$

Cache Analysis

- Full cache analysis:
  For each program point and calling context, compute the set of all possible concrete cache states.
  Note: At the beginning of a task, all cache states are possible
  $\Rightarrow$ too expensive!
- Must analysis:
  Sets of concrete cache states are abstracted by an upper bound.
  If $C^u(m) < \top$ in the abstract state,
  then $c(m) < \top$ in all possible concrete states.
  (always hit)
- May analysis:
  Sets of concrete cache states are abstracted by a lower bound.
  If $C^\ell(m) = \top$ in the abstract state,
  then $c(m) = \top$ in all possible concrete states.
  (always miss)

Must and May Analysis (LRU)

Abstract state: $C : \mathcal{M} \to \mathcal{A}'$

Order: $C_1 \sqsubseteq C_2 \iff \forall m \in \mathcal{M} : C_1(m) \le C_2(m)$

where $\mathcal{A}'$ is ordered by $0 \le 1 \le 2 \le 3 \le \top$.

Must analysis:

Abstract C^u approximates concrete $c \iff c \sqsubseteq C^u$

$\Rightarrow$ abstract age a stands for concrete ages $0, \ldots, a$

$\Rightarrow$ $up^u_{\mathcal{A}}(a_0)(a) \;=\; \max\{\, up_{\mathcal{A}}(a'_0)(a') \mid a'_0 \le a_0,\ a' \le a \,\}$.

$up^\ell_{\mathcal{A}}$	0	1	2	3	$\top$
0	1	1	2	3	$\top$
1	1	2	2	3	$\top$
2	1	2	3	3	$\top$
3	1	2	3	$\top$	$\top$
$\top$	1	2	3	$\top$	$\top$

$up^u_{\mathcal{A}}$	0	1	2	3	$\top$
0	0	1	2	3	$\top$
1	1	1	2	3	$\top$
2	1	2	2	3	$\top$
3	1	2	3	3	$\top$
$\top$	1	2	3	$\top$	$\top$

For both: $up^{\ell/u}(m_0)(c) \;=\; c'$ defined by

$$\text{where } c'(m) \;=\; \begin{cases} 0 & \text{if } m = m_0, \\ up^{\ell/u}_{\mathcal{A}}\,(c(m_0))\,(c(m)) & \text{otherwise.} \end{cases}$$

Intervals and Examples (LRU)

Must and may analysis for LRU are independent, but can be combined to an interval analysis:

$C : \mathcal{M} \to \mathcal{I}(\mathcal{A}') \;=\; \{[l,u] \in \mathcal{A}' \times \mathcal{A}' \mid l \le u\}$.

Abstract state at task entry: $C(m) = [0,\top]\ \forall m \in \mathcal{M}$.

	m_1	m_2	m_3	m_4	m_5
	$[0,\top]$	$[0,\top]$	$[0,\top]$	$[0,\top]$	$[0,\top]$
Access m_1 (?)	$[0]$	$[1,\top]$	$[1,\top]$	$[1,\top]$	$[1,\top]$
Access m_2 (?)	$[1]$	$[0]$	$[2,\top]$	$[2,\top]$	$[2,\top]$
Access m_3 (?)	$[2]$	$[1]$	$[0]$	$[3,\top]$	$[3,\top]$
Access m_2 (hit)	$[2]$	$[0]$	$[1]$	$[3,\top]$	$[3,\top]$
Access m_4 (?)	$[3]$	$[1]$	$[2]$	$[0]$	$[\top]$
Access m_5 (miss)	$[\top]$	$[2]$	$[3]$	$[1]$	$[0]$

Control-flow join:

arg's	m_1	m_2
C_1	$[0]$	$[2]$
C_2	$[\top]$	$[1]$
C	$[0,\top]$	$[1,2]$

access m_1 $\Longrightarrow$

arg's	m_1	m_2
C_1	$[0]$	$[2]$
C_2	$[0]$	$[2]$
C	$[0]$	$[1,3]$

FIFO Strategy

Each cache set has its own replacement logic so that it suffices to consider a single set.

FIFO Replacement Strategy:

Replace the block that was first put into the cache. Cache hits do not matter.

The FIFO strategy can be realized by a counter, and modeled by assigning ages.

Example (4-way set associative):

counter	lines 0	1	2	3	age 0	1	2	3
0	–	–	–	–	–	–	–	–
0	m_0	–	–	–	m_0	–	–	–
0	m_0	m_1	–	–	m_1	m_0	–	–
0	m_0	m_1	m_2	–	m_2	m_1	m_0	–
0	m_0	m_1	m_2	m_3	m_3	m_2	m_1	m_0
1	m_4	m_1	m_2	m_3	m_4	m_3	m_2	m_1
2	m_4	m_5	m_2	m_3	m_5	m_4	m_3	m_2
3	m_4	m_5	m_6	m_3	m_6	m_5	m_4	m_3
0	m_4	m_5	m_6	m_7	m_7	m_6	m_5	m_4
1	m_8	m_5	m_6	m_7	m_8	m_7	m_6	m_5

Concrete Update Function (FIFO)

Let $\mathcal{A} \;=\; \{0, \ldots, 3\}$ be the set of ages,
and $\mathcal{A}' \;=\; \mathcal{A} \cup \{\top\}$ ($\top$ means not in cache).

Concrete cache state: $c : \mathcal{M} \to \mathcal{A}'$

where $\mathcal{M}$ is the set of memory blocks.

Update when block m_0 is accessed in state c:

Result $up(m_0)(c) \;=\; c'$ defined by

$$c'(m) \;=\; \begin{cases} up^{=}_{\mathcal{A}}\,(c(m)) & \text{if } m = m_0, \\ up^{\neq}_{\mathcal{A}}\,(c(m_0))\,(c(m)) & \text{if } m \neq m_0 \end{cases}$$

using $up^{\neq}_{\mathcal{A}} : \mathcal{A}' \to (\mathcal{A}' \to \mathcal{A}')$ and $up^{=}_{\mathcal{A}} : \mathcal{A}' \to \mathcal{A}'$:

$up^{\neq}_{\mathcal{A}}$	0	1	2	3	$\top$
0	–	1	2	3	$\top$
1	0	–	2	3	$\top$
2	0	1	–	3	$\top$
3	0	1	2	–	$\top$
$\top$	1	2	3	$\top$	$\top$

$up^{=}_{\mathcal{A}}$	
0	0
1	1
2	2
3	3
$\top$	0

Must and May Analysis (FIFO)

$up_{\mathcal{A}}^{\neq,\ell/u}$	0	1	2	3	$\top$	$(up_{\mathcal{A}})$	$up_{\mathcal{A}}^{=,\ell}$	$up_{\mathcal{A}}^{=,u}$
0	0	1	2	3	$\top$	(0)	0	0
1	0	1	2	3	$\top$	(1)	0	1
2	0	1	2	3	$\top$	(2)	0	2
3	0	1	2	3	$\top$	(3)	0	3
$\top$	1	2	3	$\top$	$\top$	(0)	0	3

	m_0	m_1	m_2	m_3	m_4	m_5
	$[0]$	$[1]$	$[2]$	$[3]$	$[\top]$	$[\top]$
Access m_4 (miss)	$[1]$	$[2]$	$[3]$	$[\top]$	$[0,3]$	$[\top]$
Access m_3 (miss)	$[2]$	$[3]$	$[\top]$	$[0,3]$	$[1,\top]$	$[\top]$
Access m_4 (?)	$[2,3]$	$[3,\top]$	$[\top]$	$[0,\top]$	$[0,3]$	$[\top]$

Interval and Set Analysis (FIFO)

Combination of must and may analysis
to interval analysis $C: \mathcal{M} \to \mathcal{I}(\mathcal{A}') \subseteq \mathcal{A}' \times \mathcal{A}'$
loses too much precision.

More precise interval analysis with interaction
between must and may analysis preserves precision.

Further refinement: Set analysis $C: \mathcal{M} \to \mathcal{P}(\mathcal{A}')$.
Definition of $up^s\,(m_0)\,(C)\,(m)$:

	$\top \notin C(m_0)$	$\{\top\} \subset C(m_0)$	$\{\top\} = C(m_0)$
$m = m_0$	$C(m)$	$(C(m) \setminus \{\top\}) \cup \{0\}$	$\{0\}$
$m \neq m_0$	$C(m)$	$C(m) \cup (C(m)+1)$	$C(m)+1$

where $C(m)+1 \;=\; \{x+1 \mid x \in C(m)\}$.

Set analysis preserves precision, but . . .

	m_1	m_2	m_3
	$\{0-3,\top\}$	$\{0-3,\top\}$	$\{0-3,\top\}$
Access m_1 (?)	$\{0-3\}$	$\{0-3,\top\}$	$\{0-3,\top\}$
Access m_1 (hit)	$\{0-3\}$	$\{0-3,\top\}$	$\{0-3,\top\}$
Access m_2 (?)	$\{0-3,\top\}$	$\{0-3\}$	$\{0-3,\top\}$

Equivalent to must analysis with ≤ 1 block per set.

ColdFire MCF 5307

The cache of the ColdFire MCF 5307
consists of 128 sets of 4 lines each.

Each line may store a memory block of 16 bytes.

The block m at address a can only be stored
in the set with number $\text{set}\,(m) \;=\; (a/16) \bmod 128.$

The ColdFire MCF 5307 has a
Pseudo Round Robin replacement strategy.

The state of the replacement logic
is given by a 2 bit counter.

- Hit or allocate:
  The counter is neither used nor modified.
- Replace:
  The counter indicates
  which of the 4 lines is replaced.
  After the replacement,
  the counter is increased by 1 (modulo 4).

There is only one counter for the whole cache!

Example

Assume a program accesses blocks 0, 1, 2, 3, . . . ,
and block i is put in cache set $i \bmod 128.$

Assume further the program starts with an empty cache.

After accessing block 511: counter = 0

Line 0	0	1	2	3	4	5	$\cdots$	127
Line 1	128	129	130	131	132	133	$\cdots$	255
Line 2	256	257	258	259	260	261	$\cdots$	383
Line 3	384	385	386	387	388	389	$\cdots$	511

After accessing block 639: counter = 0

Line 0	512	1	2	3	516	5	$\cdots$	127
Line 1	128	513	130	131	132	517	$\cdots$	255
Line 2	256	257	514	259	260	261	$\cdots$	383
Line 3	384	385	386	515	388	389	$\cdots$	639

Problems

- Abstract cache states = sets of concrete cache states
  $\Rightarrow$ too much space
- Modeling the counter
  The counter stays the same or increases by 1;
  sometimes one does not know what is done.
  After 3 such uncertainties all counter information is lost
  and can never be recovered again.
- Modeling ages
  Absolute counter values can be avoided
  by assigning ages to the lines:
  The line the counter points to has age 3,
  the next line age 2 etc.
  Ages stay the same or increase by 1;
  sometimes one does not know what is done
  $\Rightarrow$ same problem as above.
- May analysis: Which blocks may be in the cache?
  Without counter or age information, one can never
  be sure that a block is removed from the cache
  $\Rightarrow$ may-set gets larger and larger.

Must Analysis

Compute which memory blocks are definitely
in the cache (for each program point).

Abstract state = set M of memory blocks

Initial state: $M = \emptyset$ (empty cache or chaos)

Update when m_0 is accessed:

1. $m_0 \in M \Rightarrow M$ is unchanged.
2. $m_0 \notin M \Rightarrow$ remove all m with $\mathrm{set}(m) = \mathrm{set}(m_0)$ from M
   and then add m_0 to M.

Control flow join: $M = M_1 \cap M_2$.

Observation:

M contains at most one memory block
for each cache set
$\Rightarrow$ M has at most 128 elements.

Simple and efficient analysis,
but only 1/4 of the cache is modeled.

PowerPC MPC 750 and 755

Cache Architecture

The PowerPC 75x has two separate caches
for instructions and data.

Each cache consists of 128 sets of 8 lines each.

Each cache set has its own replacement logic
$\Rightarrow$ cache sets are independent.

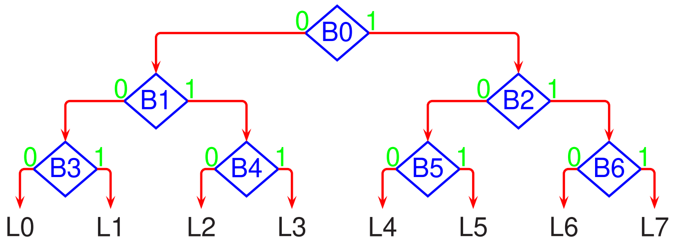
In the sequel, we describe only one cache set.
When speaking of the cache, we actually mean this set.

The state of the replacement logic
is given by 7 bits B[0..6].

- Replace:
  The 7 bits point to a line,
  which is selected for replacement.
- All accesses:
  After each access, the 7 bits are updated
  so that they point away from the accessed line.

Pseudo LRU Replacement Strategy

Each setting of B[0..6] points to a specific line:



Update of B[0..6] so that it points away from line l_0:

l_0	Bit update		
0	B0 := 1;	B1 := 1;	B3 := 1;
1	B0 := 1;	B1 := 1;	B3 := 0;
2	B0 := 1;	B1 := 0;	B4 := 1;
3	B0 := 1;	B1 := 0;	B4 := 0;
4	B0 := 0;	B2 := 1;	B5 := 1;
5	B0 := 0;	B2 := 1;	B5 := 0;
6	B0 := 0;	B2 := 0;	B6 := 1;
7	B0 := 0;	B2 := 0;	B6 := 0;

Example

Alternation between hit on L0 and miss:

0 00 0000 → 0: hit 0
1 10 1000 → 4: miss (replace 4)
0 11 1010 → 2: hit 0
1 11 1010 → 6: miss (replace 6)
0 10 1011 → 2: hit 0
1 10 1011 → 5: miss (replace 5)
0 11 1001 → 2: hit 0
1 11 1001 → 7: miss (replace 7)
0 10 1000 → 2: hit 0
1 10 1000 → 4: . . .

Observation:

- Continued hits to L0 also help its “friends” L1, L2, L3 to survive.

Examples (only misses)

0 00 0000 → 0
1 10 1000 → 4
0 11 1010 → 2
1 01 1110 → 6
0 00 1111 → 1
1 10 0111 → 5
0 11 0101 → 3
1 01 0001 → 7
0 00 0000 → 0

1 10 1011 → 5
0 11 1001 → 2
1 01 1101 → 7
0 00 1100 → 1
1 10 0100 → 4
0 11 0110 → 3
1 01 0010 → 6
0 00 0011 → 0
1 10 1011 → 5

Observations:

- After 8 misses, we are back in the original bit state.
- In these 8 steps, all lines are replaced exactly once (in some strange order).

Theorem 1: This is true for all initial states.

Introducing Ages

- Ages: 0, 1, . . . , 7, plus ⊤ for not in cache.
- If the cache is full, blocks with age 7 are replaced when misses occur.
- After an access, the accessed block has age 0.
- With each access, the ages are updated.

$$\begin{tikzcd}
\boxed{\text{bit setting } \beta \quad c : \text{blocks} \leftrightarrow \text{lines}} \arrow[r, "\text{access}", "\text{(bit update)}"'] \arrow[d, "T_\beta"] & \boxed{\text{bit setting } \beta' \quad c' : \text{blocks} \leftrightarrow \text{lines}} \arrow[d, "T_{\beta'}"] \\
\boxed{\bar{c} : \text{blocks} \leftrightarrow \text{ages}} \arrow[r, "\text{access}", "\text{(age update)}"'] & \boxed{\bar{c}' : \text{blocks} \leftrightarrow \text{ages}}
\end{tikzcd}$$

Theorem 2:

There is an appropriate age update function, which does not depend on the initial bit setting β.

Age Update Function

Let $\mathcal{A} = \{0, \ldots, 7\}$ be the set of ages,
and $\mathcal{A}' = \mathcal{A} \cup \{\top\}$.

Update function $up_{\mathcal{A}} : \mathcal{A}' \to (\mathcal{A}' \to \mathcal{A}')$:

When a block with age a_0 is accessed,
all other blocks change their age from a to $up_{\mathcal{A}}(a_0)(a)$.

$up_{\mathcal{A}}$	0	1	2	3	4	5	6	7	⊤
0	–	1	2	3	4	5	6	7	⊤
1	1	–	3	2	5	4	7	6	⊤
2	2	1	–	3	6	5	4	7	⊤
3	1	2	3	–	5	6	7	4	⊤
4	4	1	2	3	–	5	6	7	⊤
5	1	4	3	2	5	–	7	6	⊤
6	2	1	4	3	6	5	–	7	⊤
7	1	2	3	4	5	6	7	–	⊤
⊤	1	2	3	4	5	6	7	⊤	⊤

Must Analysis: Setup

Sets of concrete states are abstracted by an upper bound.

Abstract state: $C : \mathcal{M} \to \mathcal{A}'$

Order: $C_1 \sqsubseteq C_2 \iff \forall m \in \mathcal{M} : C_1(m) \leq C_2(m)$
where $\mathcal{A}'$ is ordered by $0 \leq 1 \leq \cdots \leq 7 \leq \top$.

Abstract C approximates concrete $c \iff c \sqsubseteq C$

$\Rightarrow$ abstract age a stands for concrete ages $0, \ldots, a$

$\Rightarrow$ $up^u_{\mathcal{A}}(a_0)(a) = \max\{up_{\mathcal{A}}(a'_0)(a') \mid a'_0 \leq a_0,\ a' \leq a\}$.

$up^u_{\mathcal{A}}$	0	1	2	3	4	5	6	7	$\top$
0	0	1	2	3	4	5	6	7	$\top$
1	1	1	3	3	5	5	7	7	$\top$
2	2	2	3	3	6	6	7	7	$\top$
3	2	2	3	3	6	6	7	7	$\top$
4	4	4	4	4	6	6	7	7	$\top$
5	4	4	4	4	6	6	7	7	$\top$
6	4	4	4	4	6	6	7	7	$\top$
7	4	4	4	4	6	6	7	7	$\top$
$\top$	4	4	4	4	6	6	7	$\top$	$\top$

Must Analysis

When block m_0 is accessed, the abstract state $C : \mathcal{M} \to \mathcal{A}'$ becomes $C' = up^u(m_0)(C)$ defined by

$$C'(m) = \begin{cases} 0 & \text{if } m = m_0, \\ up^u_{\mathcal{A}}(C(m_0))(C(m)) & \text{otherwise.} \end{cases}$$

Remarks:

- $up^u(m_0)$ is monotonic for all $m_0 \in \mathcal{M}$.
- C approximates $c \Rightarrow$ $up^u(m_0)(C)$ approximates $up(m_0)(c)$.
- Only the abstract ages 0, 4, 6, 7, $\top$ occur in practice.
- $C(m) \neq \top$ means hit,
  $C(m) = \top$ means not classified.
- $C(m) \neq \top$ holds for at most 4 memory blocks (in each set).
- This analysis is more complicated than the ColdFire analysis, but models 1/2 of the cache.

Further Analyses

May Analysis: Sets of concrete set states are abstracted by a lower bound.

Problem: Starting from 0 (chaos), only the values 0 and 1 are reachable $\Rightarrow$ no useful information

Combined Must and May Analysis = Interval Analysis:

Abstract ages are intervals $[l, u]$.

Age a is approximated by $[l, u] \iff a \in [l, u]$

Problem: Starting from $[0, \top]$ (chaos), only the values $[0, 0]$ and $[l, u]$ with $l \in \{0, 1\}$ and $u \in \{4, 6, 7, \top\}$ are reachable $\Rightarrow$ equivalent to must analysis

Set Analysis:

Abstract ages are sets $S \subseteq \mathcal{A}'$.

Age a is approximated by $S \iff a \in S$

Problem: Starting from $\{0, 1, \ldots, 7, \top\}$ (chaos), only a few sets are reachable.

This analysis is equivalent to must analysis, too!

Conclusion

The precision of cache analysis heavily depends on the replacement strategy of the cache.

- LRU strategy:
  Best results; must and may analysis together predict all cache lines.
- FIFO strategy:
  Problematic; when starting from an unknown state, only must information for a single line is obtained.
- ColdFire MCF 5307:
  Must information for 1 block per set of 4 lines (1/4 of the cache is modeled).
- PowerPC MPC 75x:
  Must information for 4 blocks per set of 8 lines (1/2 of the cache is modeled).