Analyzing caches: Replacement strategies and persistence

Christoph Berg

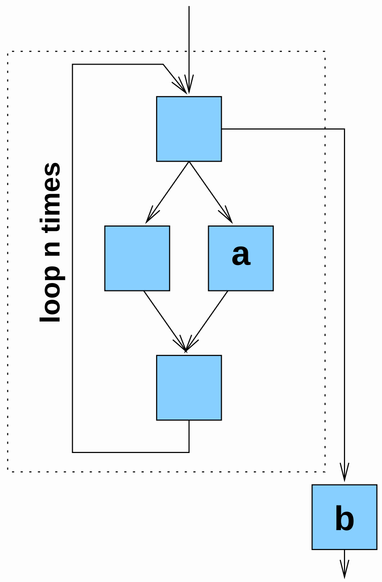
Universität des Saarlandes

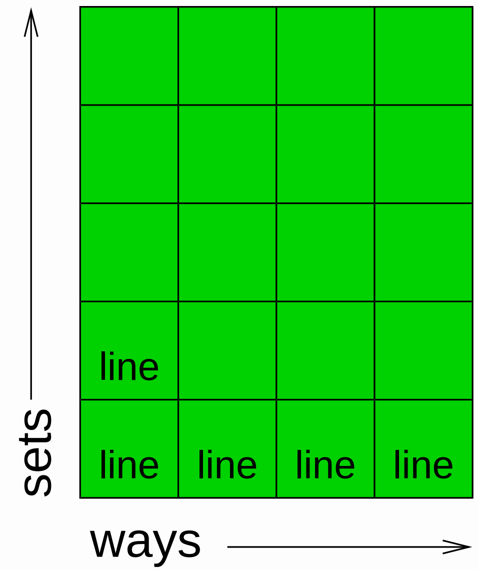
`cb@cs.uni-sb.de`

July 8, 2004

Introduction

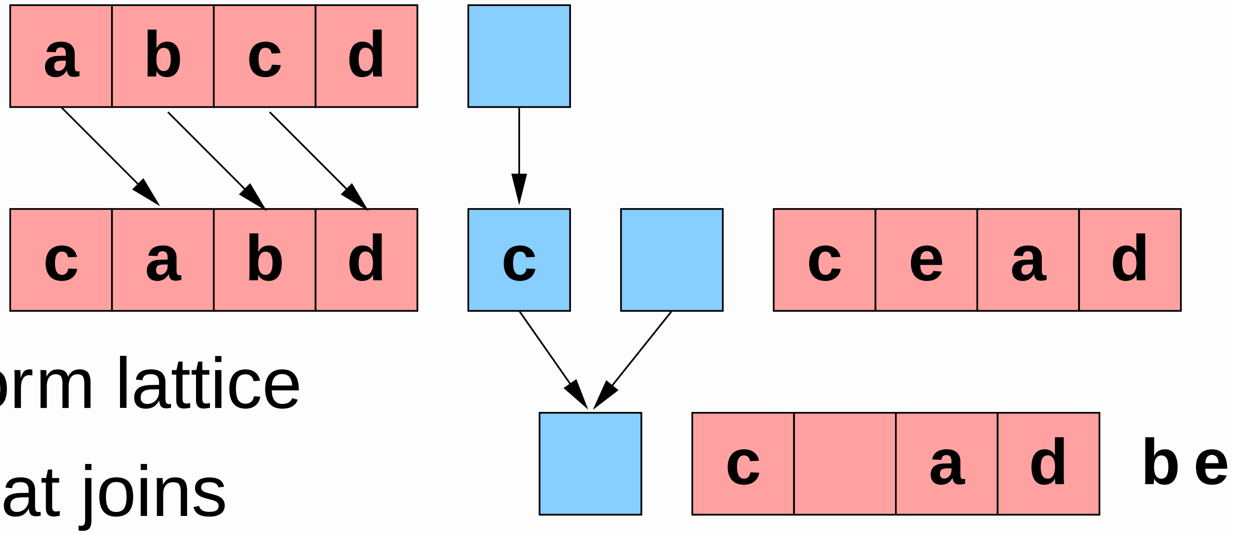
- static cache analysis
  - ▷ decide for each memory access in a given program always miss/always hit/other
  - ▷ compute upper/lower bounds for program runtime
- replacement strategy
- LRU and FIFO caches
- persistence





Abstract interpretation jump start

- state information flows along control flow graph
- transfer functions update state information
- abstract state is upper bound for all concrete state at node



- abstract states form lattice
- state union (lub) at joins
- unknown state (= all concrete states possible) is $\top$
- example: cache “must” analysis

Replacement strategy

- $c \in C$ cache states, $m \in M$ memory blocks
- *replacement strategy* := *update* + *content function*
  - $upd : C \times M \to C, (c, m) \mapsto c' = upd(c, m)$
  - $content : C \to \wp(M), c \mapsto content(c)$
- where the following hold:
  - $m \in content(upd(c, m))$
  - $m \in content(c) \Rightarrow content(c) = content(upd(c, m))$
  - $\forall m' \neq m : m' \notin content(c) \Rightarrow m' \notin content(upd(c, m))$
- access time
  - $time(c, m) := m \in content(c) \; ? \; 0 \; : \; 1$

Sequences of memory accesses

- sequence $m = \langle m_0, \ldots, m_i \rangle \in M^*$
  - $upd(c, \varepsilon) := c$
  - $upd(c, \langle m_0, \ldots, m_i \rangle) := upd(upd(c, m_0), \langle m_1, \ldots, m_i \rangle)$
- access time:
  - $time(c, \varepsilon) := 0,$
  - $time(c, \langle m \rangle) := time(c, m_0) + time(upd(c, m_0), \langle m_1, \ldots, m_i \rangle)$
- repeating the same sequence:
  - $upd^0(c, m) := c$
  - $upd^{n+1}(c, m) := upd(upd^n(c, m), m)$
  - abbreviation: $c^n := upd^n(c, m)$

LRU and FIFO definition

- $last_k(m)$ = set of last $\leq k$ unique elements in m
- k-LRU cache :=
  - $\forall c, m : \# last_k(m) \leq \# content(upd(c, m)) \leq k$
  - $last_k(m) \subseteq content(upd(c, m))$.
- k-FIFO cache := $(upd, content)$ is isomorphic to:
  - $c \in (M \cup \{\perp\})^k$
  - $content(c) = \{m_0, \ldots, m_{k-1}\}$
  - $m \in content(c) \Rightarrow upd(\langle m_0, \ldots, m_{k-1} \rangle, m) = c$
  - $m \notin content(c) \Rightarrow$
    $upd(\langle m_0, \ldots, m_{k-1} \rangle, m) = \langle m_1, \ldots, m_{k-1}, m \rangle$

Repeating access sequences

- loops in control flow graph
- does the cache behavior eventually stabilize?
- does the cache eventually forget its history?
- timing convergence :=

$$\forall m : \forall c : \exists n : \forall n' \geq n : time(c^{n'}, m) = time(c^{n}, m).$$

- no cache domino effect :=

$$\forall m : \forall c, c' : \exists n : \forall n' \geq n : time(c^{n'}, m) = time(c'^{n'}, m).$$

Example: 2-way FIFO, sequence a-b-c

```
   [. .]          [a c]
a: [. a] x     a: [a c]
b: [a b] x     b: [c b] x  1 miss
c: [b c] x     c: [c b]

a: [c a] x     a: [b a] x
b: [a b] x     b: [b a]
c: [b c] x     c: [a c] x  2 misses -> non-converging

a: [c a] x     a: [a c]
b: [a b] x     b: [c b] x
c: [b c] x     c: [c b]
    ---            --- same configuration as before
a: [c a] x     a: [b a] x   -> domino effect
b: [a b] x     b: [b a]
c: [b c] x     c: [a c] x
```

LRU theorem (1)

- **Lemma 1** *For all replacement strategies:*

$$time(c, m) = 0 \Rightarrow time(upd(c, m), m) = 0.$$

Proof. From the definition of $time$, we know $m \in content(c)$, therefore $content(c) = content(c^1)$, and hence $time(c^1, m) = time(c, m) = 0$.

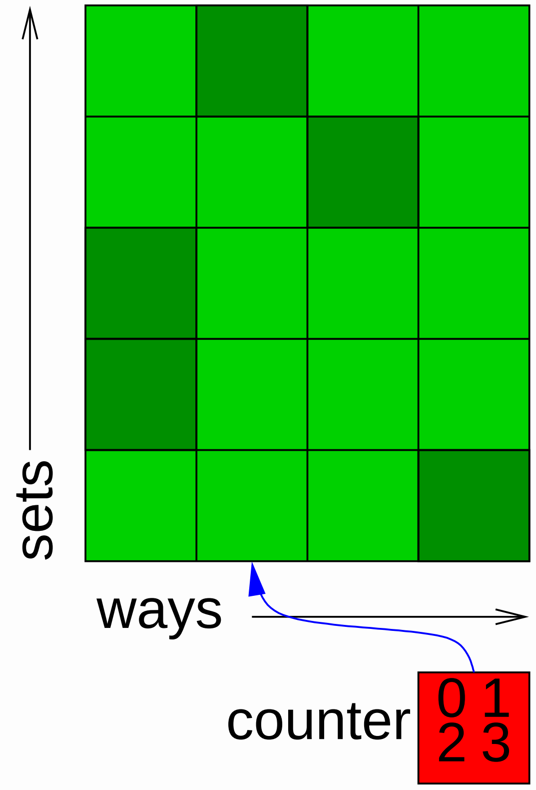
LRU theorem (2)

- **Theorem 1**
  - ▷ *LRU caches converge.*
  - ▷ *LRU caches do not show domino effects.*
- **Lemma 2** *For LRU caches one of the following holds:*
  - ▷ $time(c^n, m) = 0 \quad \forall n \geq 1$
  - ▷ $content(c^n) = content(c^1) \quad \forall n \geq 1$

  Proof. We start with c^0, let $i := time(c, m)$.
  - ▷ For $i \leq k$, $time(c^1) = 0$.
  - ▷ If $i \geq k$, we know $last_k(m) = content(c^1) = content(c^n)$.

How unpredictable is FIFO?

- ColdFire cache: 128 sets, 4 ways
- counter points to next way to be replaced
- problem in analysis:
  counter value?
- cache model used now:
  direct mapped cache for must analysis
  - throws 3/4 of cache capacity away
  - can we do better?
  - may analysis? (lower bound)



FIFO: abstract interpretation

- concrete cache state $c = (\bar{m}, z) \in C$:
  - content $\bar{m} \in (\mathbb{N} \cup \{\perp\})^{4\times 128}$
  - counter $z \in \{0, \ldots, 3\}$
- best model = no abstraction
- best lattice = powerset of (concrete) states $L = \wp(C)$
  - unknown state: $\top = C$ ("chaos cache")
- to make it simple:
  - single set
  - cache fully allocated ($\perp \notin \bar{m}$)
  - same as FIFO (but worse in general)

FIFO analysis start: unknown contents (1)

- unknown cache contents, unknown counter: state set C
- access to block m_1, new state C':

$$C' = \begin{cases} c & m_1 \in c \in C \quad \text{(hit)}, \\ c_{\bar{m}[z^{++}] \mapsto m_1} & m_1 \notin c \in C \quad \text{(miss)}. \end{cases}$$

- m_1 in cache, but unknown relation $z \sim m_1$:
  - $C' = \{c | m_1 \in c \in C\}$
  - m_1 might be at *any* position

FIFO analysis start: unknown contents (2)

- access to block $m_2 \neq m_1$ in same set:

$$C'' = \begin{cases} c' & m_2 \in c' \in C' \quad \text{(hit)}, \\ c'_{\bar{m}[z^{++}] \mapsto m_2} & m_2 \notin c' \in C' \quad \text{(miss)}. \end{cases}$$

$$= \begin{cases} c' & m_1 \in c, m_2 \in c' \quad \text{(hit, hit)}, \\ c'_{\bar{m}[z'^{++}] \mapsto m_2} & m_1 \in c, m_2 \notin c' \quad \text{(hit, miss)}, \\ c' & m_1 \notin c, m_2 \in c' \quad \text{(miss, hit)}, \\ c'_{\bar{m}[z'^{++}] \mapsto m_2} & m_1 \notin c, m_2 \notin c' \quad \text{(miss, miss)}. \end{cases}$$

- we don't know what's replaced, could be m_1:
  - $C'' = \{c | m_2 \in c \in C\}$
  - as before: only one block known to be in set
  - *all previous knowledge lost!*

FIFO analysis start: known state

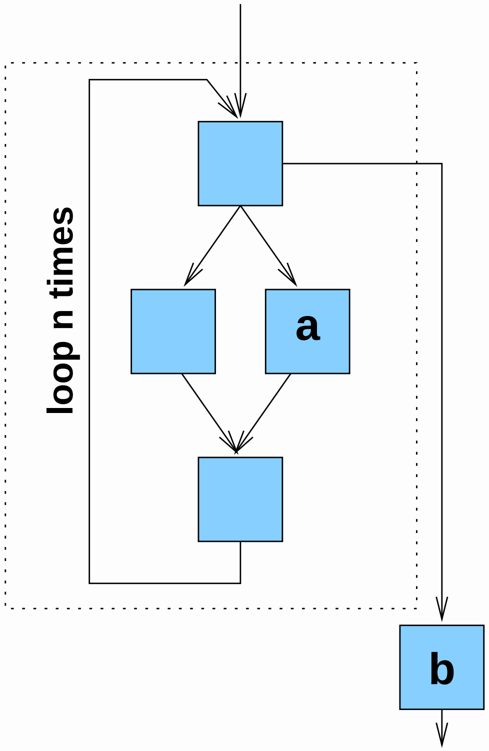
- $C = \{(\langle m_0, m_1, m_2, m_3 \rangle, z)\}$
- as long as z is known, we can track the whole set
- exact access address not known (“maybe access” to set):
  - ▷ $C' = C \cup C_{\bar{m}[z^{++}] \mapsto m_1}$
- control flow join: $C' = C_1 \cup C_2$
- as soon as m_i can be at every cache position:
  - ▷ m_i access (might) evict all other blocks
  - ▷ z still unknown
- nothing evicted for sure
- ...cf. chaos cache

FIFO analysis: summary

- (at most) one block per set in cache
  - more at the beginning, but not for long
  - no cache miss prediction
- must analysis works in subset of $\wp(C)$ isomorphic to direct mapped cache
- ColdFire is even worse: sets interact

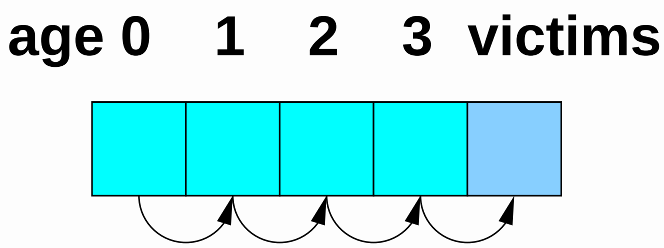
Cache persistence

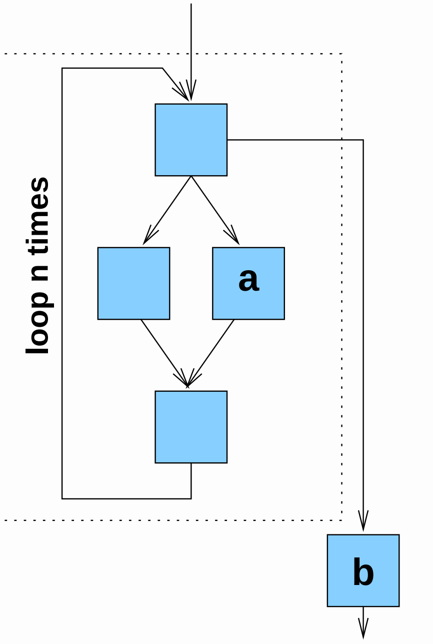
- up to now: hit/miss prediction per access
- unrolling loops helps
- find means to bound *total* number of misses
- persistence: “once loaded, block a will never be evicted”
- a cannot be classified as hit/miss, but is persistent in loop
- 1 miss in total for a (instead of n)



Persistence analysis

- extends must analysis
- collect blocks “dropping out” of must cache
- blocks not in victim buffer are loaded at most once
- global analysis can be refined
  - re-run analysis on procedure level
- improved cache prediction (hopefully. . .)





Future issues

- are there analyzable caches $\neq$ LRU?
- scratchpad memory?
- implement persistence
- apply similar methods to branch prediction:
  - gshare: PHT with hash function
  - which kinds of branch predictor are predictable?
- which pipelines “forget” their history?