Bounding Worst-Case Instruction Cache Performance

by

Robert Arnold
Frank Mueller
David Whalley
Florida State University

Marion Harmon
Florida A&M University

Motivation

- WCET Required for Scheduling Analysis
- Performance Penalty for Disabling Cache
  - Are Caches Unpredictable?
- Limitations
  - non-preemptive systems
  - direct-mapped caches
  - RISC architectures

Static Cache Simulator

- Construct a control-flow graph for the program.
- Determine which program lines may be in cache for each basic block, complexity O(nˆ2).
- Classify the caching behavior for each instruction.

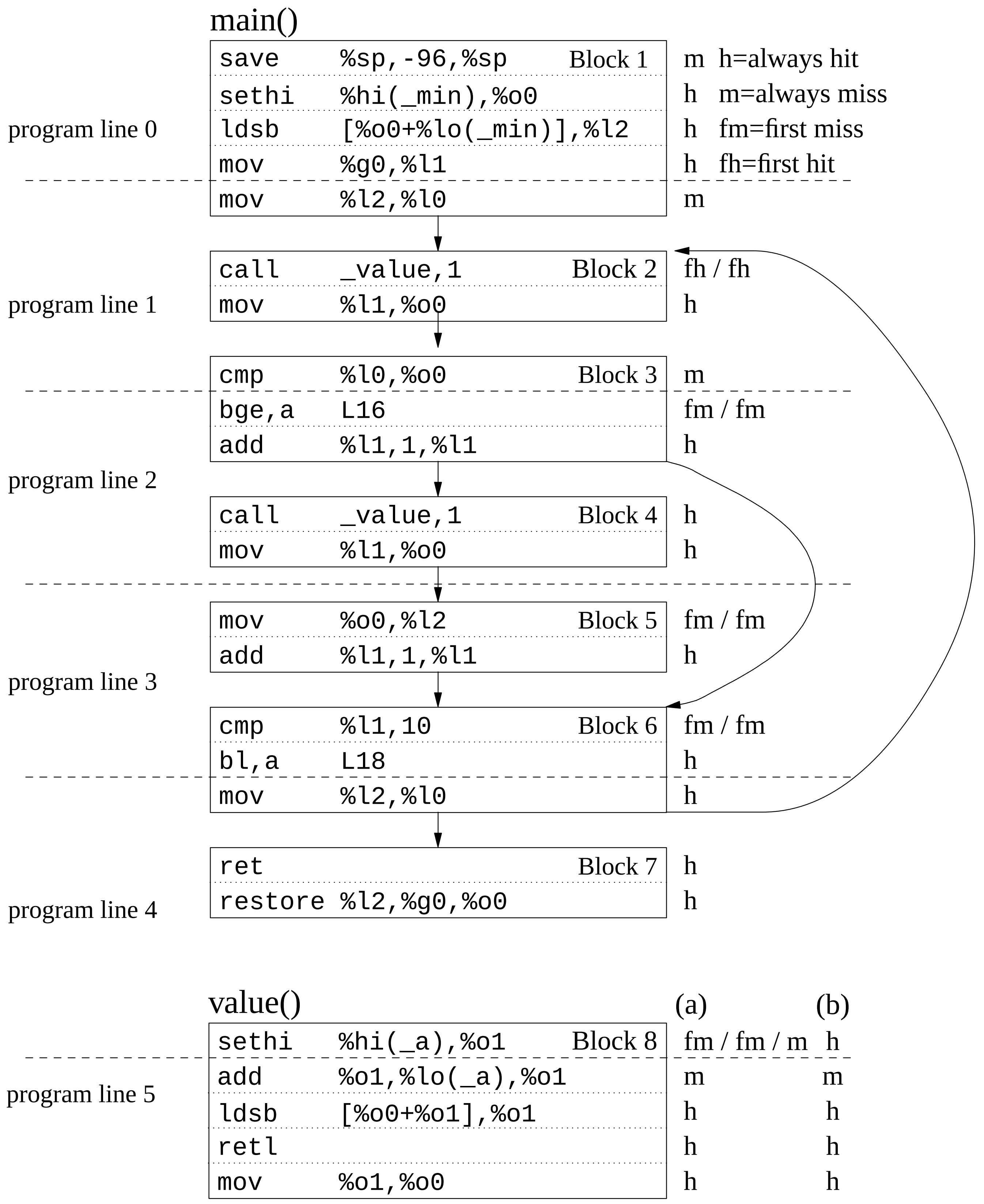
Definition Of Instruction Cache Categories

- ALWAYS HIT
  - all references: hits
- ALWAYS MISS
  - all references: misses
- FIRST HIT
  - first reference: hit
  - subsequent references: misses
- FIRST MISS
  - first reference: miss
  - subsequent references: hits

Algorithm to Calculate Cache States

- similar data-flow analysis in optimizing compilers
- input state for a basic block :=
  set of program lines that can potentially be cached at the point the basic block is entered

```
input_state(top) := all invalid lines
WHILE any change DO
  FOR each basic block instance B DO
    input_state(B) := NULL
    FOR each immed pred P of B DO
      input_state(B) += output_state(P)
    output_state(B) :=
      (input_state(B) + prog_lines(B))
      - conf_lines(B)
```



main()

```
                 save     %sp,-96,%sp         Block 1   m  h=always hit
                 sethi    %hi(_min),%o0                 h  m=always miss
program line 0   ldsb     [%o0+%lo(_min)],%l2           h  fm=first miss
                 mov      %g0,%l1                       h  fh=first hit
-----------------------------------------------------------------
                 mov      %l2,%l0                       m

                 call     _value,1            Block 2   fh / fh
program line 1   mov      %l1,%o0                       h

                 cmp      %l0,%o0             Block 3   m
-----------------------------------------------------------------
                 bge,a    L16                           fm / fm
                 add      %l1,1,%l1                     h
program line 2
                 call     _value,1            Block 4   h
                 mov      %l1,%o0                       h
-----------------------------------------------------------------
                 mov      %o0,%l2             Block 5   fm / fm
                 add      %l1,1,%l1                     h
program line 3
                 cmp      %l1,10              Block 6   fm / fm
                 bl,a     L18                           h
-----------------------------------------------------------------
                 mov      %l2,%l0                       h

                 ret                          Block 7   h
program line 4   restore  %l2,%g0,%o0                   h
```

value()

```
                                                        (a)            (b)
                 sethi    %hi(_a),%o1         Block 8   fm / fm / m    h
-----------------------------------------------------------------
                 add      %o1,%lo(_a),%o1               m              m
program line 5   ldsb     [%o0+%o1],%o1                 h              h
                 retl                                   h              h
                 mov      %o1,%o0                       h              h
```

Timing Analyzer

- Construct the timing analysis tree.
- Calculate the worst-case time for each node based on the instruction categorization, complexity O(nˆ2).
- Respond to user timing requests.

Algorithm for Estimating a Loop’s Worst-Case Performance

```
min_time = max time for any loop path assuming
           each path has been previously executed;
k = 0;
WHILE k < n - 1 DO
   max_time = current max time for any loop path;
   IF max_time == min_time THEN
      BREAK;
   total_time += max_time;
   k += 1;
total_time += (n - 1 - k) * min_time;
total_time += max time for exit paths;
```

Future Work

- Best-Case Predictions
- Data Cache Predictions
- Pipelining
- Verifying Timing Predictions for an Actual Machine
- User Interface

Conclusions

- Technique for Predicting Instruction Cache Performance
  - Static Cache Simulation to Categorize Each Instruction
  - Timing Analysis for Each Loop in the Program
- Instruction Cache Behavior is Sufficiently Predictable for Real-Time Applications