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 $O(n^2)$.

• 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()

1. `save %sp,-96,%sp` Block 1
   m h=always hit
2. `sethi %hi(_min),%o0` Block 1
   h m=always miss
3. `ldsb [%o0+%lo(_min)],%l2` Block 1
   h fh=first hit
4. `mov %g0,%l1` Block 1
5. `mov %l2,%g0` Block 1
6. `mov %l2,%l0` Block 1
7. `bor%e,a L16` Block 2
   h
8. `call _value,1` Block 2
   fh / fh
9. `mov %l2,%l0` Block 3
   h
10. `cmp %o0,10` Block 3
11. `add %l1,1,%l1` Block 3
12. `mov %o0,%l2` Block 3
13. `mov %l1,%o0` Block 3
14. `call _value,1` Block 3
15. `add %l1,1,%l1` Block 4
16. `cmp %l0,%o0` Block 4
17. `mov %l1,%o0` Block 4
18. `mov %l2,%l0` Block 4
19. `mov %g0,%l1` Block 4
20. `bge,a L18` Block 4
   h
21. `call _value,1` Block 4
   fm / fm / m
22. `ret` Block 7
   h
23. `restore %l2,%g0,%o0` Block 7
   h

*value()*

(a)

1. `sethi %hi(_a),%o1` Block 8
   fm / fm / m h
2. `add %o0,%lo(_a),%o1` Block 8
   m m
3. `ldsb [%o0+%o1],%o1` Block 8
   h
4. `retl` Block 8
5. `mov %o1,%o0` Block 8
   h

(b)
Timing Analyzer

- Construct the timing analysis tree.
- Calculate the worst-case time for each node based on the instruction categorization, complexity $O(n^2)$.
- Respond to user timing requests.
Algorithm for Estimating a Loop’s Worst-Case Performance

\[ \text{min\_time} = \text{max time for any loop path assuming each path has been previously executed;} \]
\[ k = 0; \]
\[ \text{WHILE } k < n - 1 \text{ DO} \]
\[ \quad \text{max\_time} = \text{current max time for any loop path;} \]
\[ \quad \text{IF } \text{max\_time} == \text{min\_time} \text{ THEN} \]
\[ \quad \text{BREAK;} \]
\[ \quad \text{total\_time} += \text{max\_time;} \]
\[ \quad k += 1; \]
\[ \text{total\_time} += (n - 1 - k) \ast \text{min\_time;} \]
\[ \text{total\_time} += \text{max time for exit paths;} \]
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