Retargetable Infeasible Path Detection for WCET Analysis

Zach Hall, Kory Kraft, Thomas Mincher, Joey Iannetta, Chris Healy
Department of Computer Science, Furman University, Greenville, SC

Background

An accurate estimate of the Worst Case Execution Time (WCET) of a task is essential in hard real-time system scheduling. Often, a function or loop has multiple execution paths, due to conditional branch instructions. Based on the nature of the comparisons, some paths may be infeasible. Eliminating infeasible paths from consideration can give a tighter bound on the WCET.

Approach

We created a tool, a Retargetable Assembly Level Hierarchical Organizer (RALPHO). Its goal is to glean control-flow facts to pass to our timing analyzer. These facts are used to create function instances and paths. RALPHO is easy to retarget and replaces the need for us to rely on a compiler to provide control-flow information.

RALPHO Preliminaries

First, read assembly definition file to understand the syntax of the target instruction set.
Next, read the assembly source.

- Find functions and labels (branch targets)
- Partition instructions into basic blocks
- Find predecessors, successors, dominators of blocks.
- Find loops, calculate number of iterations, and how the loops nest. Identify which blocks comprise each loop.

Branch Constraints

For each block, RALPHO computes the presence of branch prediction or branch correlation constraints.

- Block A makes block B unknown.
- Block A makes block B fall through.
- Block A makes block B branch.
- If A falls through, then B is unknown.
- If A branches, then B is unknown.
- If A falls through, then B will branch.
- If A branches, then B will branch.
- If A falls through, then B will fall through.
- If A branches, then B will fall through.

Branch Analysis

Correlation:

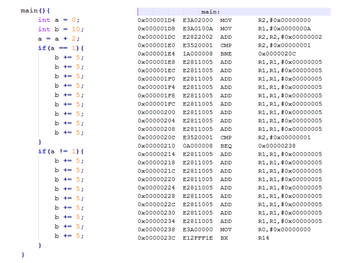
- We consider branches in pairs. If two branches in the same function compare the same register, we perform correlation analysis. For each pair, there are 4 possibilities: both taken, both fall through, first only taken, second only taken. We determine which possibilities could occur.
- There are 6*6*3 = 108 possible cases of branch correlation, due to 6 relational operators for each comparison, and the 3 ways that the constants compare to each other.
- The infeasible paths can be selected by scanning through the appropriate possibilities. For example, if the first branch is a >, the second branch is a <, and the two numbers being compared to the register are equal then (True, False) and (False, True) are the only feasible paths.

Prediction:

- If a block updates a register that is later used in a branch instruction, we determine if the effect makes the branch taken or not.

Sample Results

Path #	Blocks in Path	Feasibility
1	1,2,3,4,5	Not Feasible
2	1,2,3,5	Feasible
3	1,3,4,5	Feasible
4	1,3,5	Not Feasible



```
main() {
  int a = 0;
  int b = 10;
  a = a + 2;
  if(a == 1){
    b += 5;
    b += 5;
    b += 5;
    b += 5;
    b += 5;
    b += 5;
    b += 5;
    b += 5;
  }
  if(a != 1){
    b += 5;
    b += 5;
    b += 5;
    b += 5;
    b += 5;
    b += 5;
    b += 5;
    b += 5;
  }
}
```

```
main:
0x000001D4  E3A02000  MOV     R2,#0x00000000
0x000001D8  E3A0100A  MOV     R1,#0x0000000A
0x000001DC  E2822002  ADD     R2,R2,#0x00000002
0x000001E0  E3520001  CMP     R2,#0x00000001
0x000001E4  1A000008  BNE     0x00000200
0x000001E8  E2811005  ADD     R1,R1,#0x00000005
0x000001EC  E2811005  ADD     R1,R1,#0x00000005
0x000001F0  E2811005  ADD     R1,R1,#0x00000005
0x000001F4  E2811005  ADD     R1,R1,#0x00000005
0x000001F8  E2811005  ADD     R1,R1,#0x00000005
0x000001FC  E2811005  ADD     R1,R1,#0x00000005
0x00000200  E2811005  ADD     R1,R1,#0x00000005
0x00000204  E2811005  ADD     R1,R1,#0x00000005
0x00000208  E2811005  ADD     R1,R1,#0x00000005
0x0000020C  E3520001  CMP     R2,#0x00000001
0x00000210  0A000008  BEQ     0x00000238
0x00000214  E2811005  ADD     R1,R1,#0x00000005
0x00000218  E2811005  ADD     R1,R1,#0x00000005
0x0000021C  E2811005  ADD     R1,R1,#0x00000005
0x00000220  E2811005  ADD     R1,R1,#0x00000005
0x00000224  E2811005  ADD     R1,R1,#0x00000005
0x00000228  E2811005  ADD     R1,R1,#0x00000005
0x0000022C  E2811005  ADD     R1,R1,#0x00000005
0x00000230  E2811005  ADD     R1,R1,#0x00000005
0x00000234  E2811005  ADD     R1,R1,#0x00000005
0x00000238  E3A00000  MOV     R0,#0x00000000
0x0000023C  E12FFF1E  BX      R14
```

Above are the results of a sample test whose accuracy improved with infeasible path detection. Alongside each test is the C code and assembly code used to test the timing analyzer.

Ongoing Work

We plan to test our approach with established WCET benchmarks. Further refinements to our approach include the following: dealing with branches that compare more than one register, and determining on which loop iterations a branch should be taken. We also plan to incorporate RALPHO's infeasible path detection into our related work on stochastic analysis to improve its accuracy.

Special Thanks

The work was supported in part by grants from the Furman Advantage and Francis M. Hipp Research Fellowship and the South Carolina Independent Colleges and Universities Research Program.