Implementation of Automated Loop-bound Analysis for Static Timing Analysis Framework

Balaji V. Iyer and Won So
{bviyer, wso}@ncsu.edu
http://www4.ncsu.edu/~wso/csc714

Fall 2005 CSC714 Project Progress Report
31 October 2005

1. Work Completed

1) Set up an environment (Mainly by Won So)

After surveying existing tools, we have decided to work on existing compiler backend 'opt', which obtained from Dr. Mueller. Originally it works for Sparc ISA. It has the functionalities of constructing control flow graph (CFG) and detecting loops from CFG. Also the program is scalable to add or remove existing analysis and optimization paths. We modified 'opt' so that it only enables CFG construction (function: setupcontrolflow) and loop detection (function: findloops). Appropriate changes were applied to opt.c and Makefile. Additionally, we set up a CVS server to manage the source code of 'opt'. We also installed SimpleScalar to our Linux machine.

2) Port opt to PISA (Mainly by Won So)

Since existing timing analysis framework uses PISA assembly, we began to modify 'opt' program to accept PISA assembly and process PISA instructions. Reading an assembly file is done by the function (readfunc() in io.c) and it was modified so that it could process PISA assembly. The structure for defining instruction types (insttypes in io.c) was also redefined for PISA instructions. The function which constructs CFG (setupcontrolflow in flow.c) is modified because of the differences in branch instruction formats. Besides, other miscellaneous functions for output (E.g. dumpblk in io.c) were also modified to generate valid PISA assembly as an output. The function which sets bit vector for uses and set (setsuses in vars.c) is also dependent on the instruction format and it must be modified if it is ever used. However, we did not modify this function because we do not need bit vectors.

3) Setting up a method to find loop bounds for simple loops (Mainly by Balaji V. Iyer)

Please see section 4 for the pseudo code for detecting loop bounds.

4) Implementation of simple method (Mainly by Balaji V. Iyer)

In order to understand the code generation for loops, we analyzed the GCC source code available in the SimpleScalar locker. The Opt framework already finds all the loops and puts them in a singly linked list. We traverse through the linked list for each available block in the loop and the block whose successors are not in the loop is the root node (for simple loops). This node is analyzed to find the loop bound. This block also holds the loop induction variable (or the location in the stack where this information is stored).

Second, we have to find the loop increment value. If the loop induction variable is stored in a register, then we find a self destructive instruction whose destination register is the loop-induction register. Then we have our increment value.

If the loop induction variable is stored in the stack, then we find all the loads that load this value into a register, and find all the instruction that are dependent to that register, and find the constant computation done by these registers. The sum of all these computation will give an approximate value for the increment value.

We currently have an algorithm that catches the loop-exit-node of the loop. We are also able to find the loop bounds. We are currently working on implementing the function to calculate the loop increment value.

2. Open Issues

None so far.

3. Plans

1) Finish the method for simple loops mentioned in 3) and 4) of section 1. (By Balaji V. Iyer)

2) Test implemented method with various inputs derived from benchmarks. (By Won So and Balaji V. Iyer)

3) Extend to loops where the loop bound is derived from a global variable. (By Won So and Balaji V. Iyer)

4. Pseudo Code

```
procedure find_simple_loop_bounds

find_all_loops()

for (ii = first_loop; ii != last_loop = ii = next_loop(ii))
do
  for-each block(jj) in loop ii
  do
     if (the successors of jj is not a block in ii)
        /* means we have reached the root of the loop */
        find_loop_bound(jj);
        find_loop_increment();
     fi
  done
done
end-procedure


procedure find_loop_bound(block jj)
  for-each instructions(kk) in jj
  do
     if (kk == compare-instruction)
       find the two comparision values.
       if (one of the two values is not a constant)
         /* this isnot a simple loop by our definition */
             exit
       else
         return the constant /* this is the loop bound */
       mark the non-constant value as the induction variable.
       fi
     fi
  done
end-procedure


procedure find_loop_increment (loop X, loop_variable, loop_upperbound)

  increment_value = 0;
  if (loop_variable == register)
    for-each blocks (BB) in X
       for-each instructions (II) in block BB
         if ((II == self_destructive) &&
           (destination_register(II) == loop_variable.register))
            if (the increment/decrement value == constant)
                increment_value := Loop_Upper_Bound/immediate_value(II);
            fi
         fi
      done
    done
  fi


  variable look-for-stores := 0;
  if (loop_variable == stack_value)
    for-each blocks (BB) in X
      for-each instructions (II) in block BB
        if ((look-for-stores == 0) &&
          (II.type == load_instruction) &&
            (II.source_register == stack/frame pointer) &&
            (II.offset == loop_variable.offset))
           look-for-stores = 1;
        fi

        if ((look-for-stores == 1)  &&
            (II.type != store_instruction) &&
            (II == immediate instruction))
        if (II is dependent on the loop variable)
            loop_increment = loop_upperbound/immediate_value(II);
          fi
        fi
      if ((look-for-stores == 1) &&
            (II.type == store_instruction))
            look-for-stores = 1;
        fi

      done
    done
  fi
  return loop_increment;
end-procedure
```

5. Disclaimer

This paper and associated software changes are intended for **informational purposes only**. Therefore, any use of the information presented in this student work is at your own risk. Balaji V. Iyer, and Won So provide **no warranties of any kind** surrounding the use of this material.