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 3
15 November 2005

1. Work Completed

1) Set up an environment

Described in the previous report

2) Port opt to PISA

Described in the previous report

3) Setting up a method to find loop bounds for simple loops

Described in the previous report

4) Base Implementation: Loop bound determined by constant

We limit our scope of work to simple loops, single-entry and single-exit loops.
Basic loop bound detection algorithm is described by pseudo-code in the previous report.
Summarized steps follow:

1) Detect a loop. This is done using the *findloops* function already implemented in the opt compiler.

2) Detect a loop-exit block. This is done by looking at all the successors of each block, and the block whose successor is not part of the loop is determined to the loop-exit block.

3) Detect a register or stack offset which is used as an induction variable. This value is then stored in *loopnode* **à** *loop_var_addr*. (Please note that the starting point of " **à** " indicates a structure name and the end point of the arrow indicates a field in the structure).

4) Store initial and final values of the loops and the type of values it is storing. They are recorded in *loopnode* **à** *compare_type, loopnode* **à** *final_value, loopnode* **à** *final_val_type* and

loopnode **à** *final_val_reg*, respectively. Explanation of the init value types are given in Table 1. The exact same criterion is applied for final value field.

Value Type	Explanation	init_val	init_val_reg
LOOP_CONSTANT	Setting the init_val_type to this type indicates that the initial value provided in the code is a constant. The initial value is set in the init_val field of the loopnode structure.	Constant Value that starts (*or stop for final_val*) the loop	*Invalid Value*
LOOP_GLOBAL	Setting the init_val_type to this type indicates that the initial value provided in this code is a global value. The global variable name is stored in the init_val_reg field of the loopnode structure.	*Invalid Value*	Global value name (e.g. 'x')
LOOP_REGISTER	This means the initial value is derived from a local value in the function. The local register number (or stack offset) is stored in the init_val_reg field of the loopnode structure.	*Invalid Value*	Register name (E.g. $5) or stack location (E.g. 16($fp))

Table 1: Explanation of Loop Value Types

5) Detect a loop increment value and increment operator from a loop body. These results are stored in *loopnode* **à** *compare_type* and *loopnode* **à** *inc_type*. The inc_type field holds the name of the instruction that is doing the modification. The compare type holds the type of comparison done by the loop exit block. The possible values for compare type are: EQ (Equal to), NE (Not Equal), GT (Greater Than), GE (Greater than or Equal to), LE (Less than or Equal to), LT (Less Than). For this project, we limit our search to adds, subtracts, multiply, divide, shifts and rotates.

6) For loops whose initial or final values are not constant value, we denote them appropriately by setting *loopnode* **à** *<init/final>_val_type* to LOOP_GLOBAL or LOOP REGISTER. The only difference is that, in the former case, the field *loopnode* **à** *<init/final>_val_reg* holds a variable instead of a register number. This information is explained in detail in Table 1.

7) After we obtain a valid final and init value (this is true only when both the fields have the type LOOP_CONSTANT), we compute the number of times we execute the loop.

8) Some loops whose final value is not bound by a constant, there can be a situation where the final value is modified inside the loop. This kind of loop is unpredictable. We detect this kind of functions. The result whether the loop can be predictable or not is indicated in the

loopnode **à** *predictable* field. The initial values are not checked, since they are not a victim to this problem.

5) Implementation Extension: Loop bound determined by global or local variable

We extended our method to compute loop bounds determined by global variables. We added the function (readinglobals) in io.c for reading integer global variables from a assembly file and construct a symbol table storing a global variable name and a initial value as a linked list (struct globalinfo* globals in io.c) However, we have concluded that we can not use the initial value of a global variable for loop bound analysis because the global value can be modified anywhere in the other functions, even located in a different file. Therefore, rather than specifying a fixed loop bounds we specify the global value name for parametric analysis. For local variables, we can specify loop bounds of those as a register name or stack location. Furthermore, we are trying to specify it as a local variable name instead of register value.

For a non-rectangular loop, which inner loop bound is determined by an outer loop induction variable; we check it is non-rectangular loop. This information is indicated in loopnode **à** rectangular field.

6) Generation of .loop file

The input file for pcompiler is .loop file. For rectangular loops which loop bounds are determined by constants, we will provide the loop bounds. If the loop bounds are unpredictable then we indicate the appropriate field as "?" inside the .loop file so that the user can specify that value manually. For local or global variables, we are trying to use name of the variable. For non-rectangular loops, there is a format to be specified to .loop file. Every single piece of information needed is stored in loopnode structure so a valid .loop file will be generated.

2. Open Issues

None so far.

3. Plans

1) Finish the method (By Won So and Balaji V. Iyer)

2) Generate valid .loop file and print miscellaneous output for the user (By Won So and Balaji V. Iyer)

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

4. Pseudo Code

The code is identical as submitted in the previous report.

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.