verifying simulated program correspondence to actual execution through function coverage

Alireza Khalafi
David Morano
David Kaeli
Northeastern University

Northeastern University
Computer Architecture Research

Augustus Uht
University of Rhode Island

Microarchitecture Research Institute
University of Rhode Island

March 28, 2003

outline

- **problem**
- **simulation approaches**
- **verification approaches**
- **function coverage**
- **results**
  - using refernece inputs
  - using reduced inputs
- **summary**

problem

- we want to perform microarchitectural simulation on a program
- microarchitectural simulation takes a long time !
- we generally cannot simulate the program to completion
- how do we verify that our simulated program execution is characteristic of an actual program execution done in its entirety ?

simulation approaches

- **simulation approaches**
  - run the program from the start but only for a limited number of instructions
  - skip some number of instructions at the start and then run the full microarchitectural ("performance") simulation for a limited number of instructions
  - use a different program input for the simulation as compared with what would be used normally
  - alternately run the skipping simulator with the microarchitectural simulator
- **in any of these cases, how do we verify correspondence to actual program execution ?**

verification approaches

- **compare statistics from the actual execution with those from the simulation**
  - types of instructions execution
  - percent memory operations
  - percent control-flow instructions
  - other
- **what about when all (or most all) of the above are the same ?**
- **try function coverage !**

function coverage

- **what is it ?**
  - the percentage of instructions that the program executes in each of its subroutines
- **how do we determine it ?**
  - gather statistical (in time)
    - ⇒ monitor(3c)
    - ⇒ profil(2)
  - gather actual (instruction counts)
    - ⇒ Pixie
    - ⇒ ATOM
  - analyze
    - ⇒ prof(1)
    - ⇒ gprof(1)
- **in simulation, we can also determine it using instructions counts**
  - but we need a FAST algorithm ! (data structures ?)

implementation overview

- **at load time**
  - read symbol table from the program object file
  - record in DB the subroutine names and entry addresses
  - sort all subroutines by entry address for quick access later
  - on SGI OS determine subroutine length from successive entry points
- **at run time**
  - for each instruction: search DB for which function the instuction address is in
    - ⇒use cache of most recent subroutine encountered for extra speed
    - ⇒use binary search for speed -- complexity O(log N)
  - increment the associated subroutine counter
- **after execution**
  - sort subroutines by hit counts

example (BZIP2)

Pixie (whole program)	
function	% instructions
generateMTFValues	21.5%
qSort3	15.1%
sendMTFValues	13.7%
sortIt	11.4%
fullGtU	9.6%
getRLEpair	8.1%
simpleSort	6.7%
loadAndRLEsource	3.7%
fgetc	3.7%

simulation (500 M instructions)	
function	% instructions
generateMTFValues	18.8%
sendMTFValues	13.3%
qSort3	10.0%
fullGtU	8.6%
sortIt	7.2%
getRLEpair	6.9%
simpleSort	6.4%
bsW	3.3%
fgetc	2.9%

example (COMPRESS)

Pixie
(whole program)

function	% instructions
compress	37.1%
decompress	14.4%
output	12.3%
getcode	10.7%
putbyte	9.0%
getbyte	6.6%
readbytes	4.8%
getranchar	4.3%
fgetc	0.4%

simulation
(500 M instructions,
reference input)

function	% instructions
getranchar	67.7%
ran2	9.0%
fill_text_buffer	6.7%
compress	0.0%
output	0.0%
decompress	0.0%
getcode	0.0%
rindex	0.0%
onintr	0.0%

example (COMPRESS)

Pixie
(whole program)

function	% instructions
compress	37.1%
decompress	14.4%
output	12.3%
getcode	10.7%
putbyte	9.0%
getbyte	6.6%
readbytes	4.8%
getranchar	4.3%
fgetc	0.4%

simulation
(500 M instructions,
reduced input)

function	% instructions
compress	24.5%
getranchar	14.7%
output	10.1%
decompress	9.6%
getcode	7.1%
putbyte	5.9%
getbyte	4.9%
readbytes	2.9%
ran2	2.0%

summary

- shown how function coverage can be used to verify corresponding simulation characteristics with actual program execution
- shown how function coverage is calculated (fast) in simulation environment
- used function coverage analysis to change the input on the COMPRESS program to get closer to the actual whole-program execution characteristics -- saved us on our Micro'02 submission !