The Performance of Runtime Instrumentation and its Interaction with Processor Features

Bryan Buck

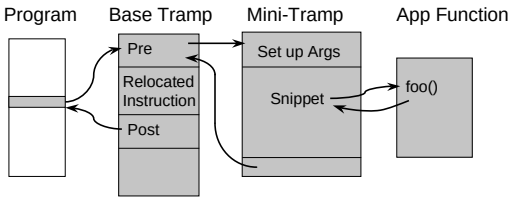
Jeff Hollingsworth

© Copyright 1998, Jeffrey K. Hollingsworth, All Rights Reserved.

Dyninst API

- Library built on top of code from Paradyn
- Allows instrumentation of running program
  - Remove as well as insert instrumentation
  - Runs on several platforms (same as Paradyn)
- For study, we used small instrumentation
  - A stress test for instrumentation overhead
  - How constraint clauses in Paradyn are implemented

Inserting Code into an Application



Base tramp provides hooks for instrumentation before and after instruction.
Mini-tramps can be chained together to call several snippets per point.

Challenges on the x86

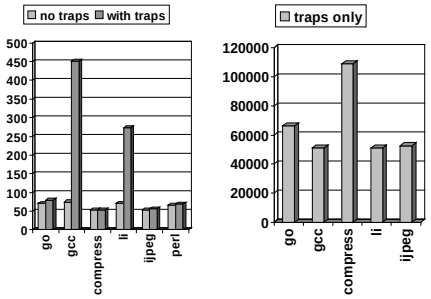
- Instructions are of variable size
  - Jump to base tramp is one byte plus 32-bit offset
- Original instruction may be too small
  - We try to relocate neighboring instructions as well
  - Sometimes we can't relocate enough
- As a last resort, use a trap
  - Instruction to generate a trap is one byte
  - On Solaris, trap is handled in the mutatee
  - On Windows NT, mutator must become involved

Applications

	time (billion cycles)	number of traps	millions of non-traps	% traps
go	7.90	26,536	210.2	0.01262
gcc	1.79	72,469	9.7	0.73616
compress	5.59	325	295.8	0.00010
li	1.07	361,220	90.5	0.39750
ijpeg	27.44	7,488	116.7	0.00641
perl	7.16	1	84.5	0.00000

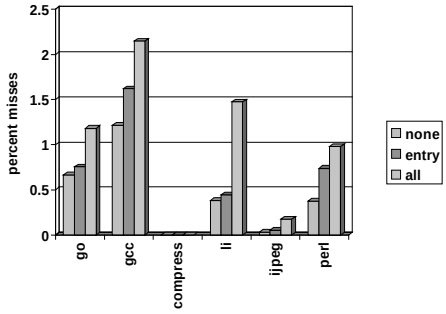
- Proportion of traps varies widely
  - compress and Perl have very few
  - gcc and li have many

Time (Cycles) per Instrumentation Point



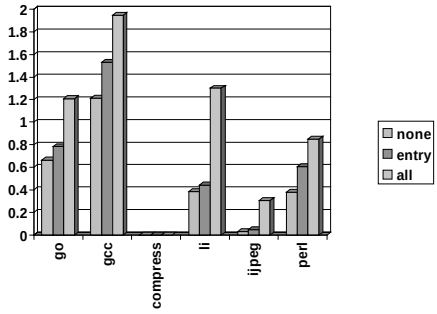
- Perl "traps only" case (not shown) was 227 million cycles

Instruction Cache Misses with Traps



University of Maryland 7

Instruction Cache Misses without Traps



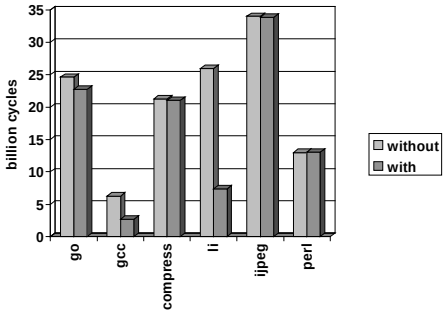
University of Maryland 8

What Can We Do about Traps?

- Things we can do at runtime
  - Use short jump to where we can put a longer one
  - For exit points, modify return address on stack
- Things we'd have to do before runtime
  - Place NOPs around points that would use traps
  - Supply "instrumentable" versions of problem functions, and switch at runtime

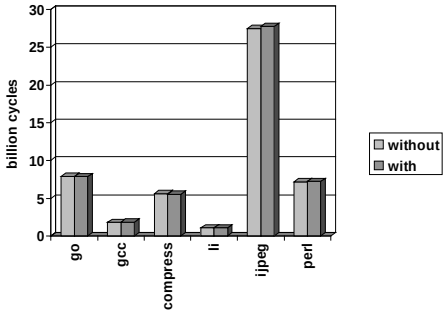
University of Maryland 9

Instrumented Running Time with and without NOPs



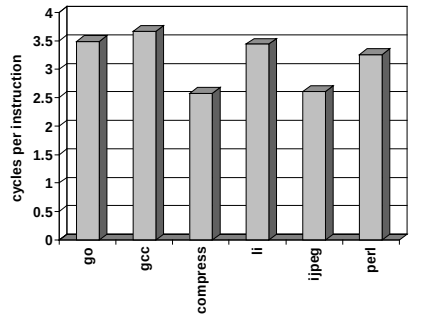
University of Maryland 10

Uninstrumented Running Time with and without NOPs



University of Maryland 11

Cycles per Instruction of Instrumentation without Traps



University of Maryland 12

Conclusions

- Substantial performance penalty for traps
  - We execute 100 times more instructions for a trap
  - We can eliminate traps by adding NOPs
  - NOPs do not significantly affect performance
- We are affecting the instruction cache
  - Data cache doesn't seem to be similarly affected
- Future work
  - Look at larger instrumentation
  - Look at other factors (data cache, etc.)
  - Look at other processor architectures

University of Maryland 13