Program Instrumentation

Ed Hong
UW CSE548

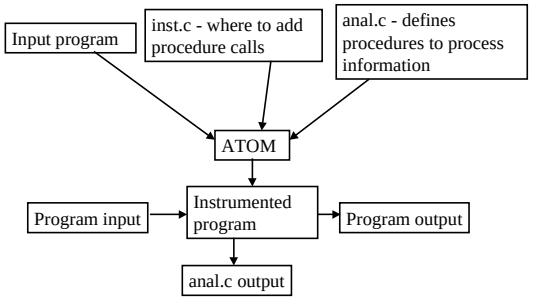
fIntroduction

- What- add code to existing program at specified points
- Why
  - analyze program behavior
  - compute performance bottlenecks
  - examine how different architectures result in different performance

ATOM

- general purpose framework for instrumenting alpha code
- makes creating program analysis tools easy
  - instruction counts
  - Cache modeling
  - branch prediction

How to use it



How to specify instrumentation sites

- Program = list of procedures
- Procedure = list of basic blocks
- Basic Block = list of instructions
- Navigate with GetFirstProc, GetNextProc, etc.
- use AddCallProto to add prototypes for your procedures
- AddCallProgram, AddCallBlock, etc. to add procedure calls

High-level examples

- cache analysis:
  - instrument- at call after load/stores
  - analysis- model cache in memory to check for hits/misses
- Branch prediction
- Dynamic memory allocation:
  - instrument: replace malloc calls with call to custom memory routine
  - analysis: could look at fragmentation, memory usage over time

Performance Issues

- minimize time spent in analysis, number of calls to analysis routines
- Counting dynamic number of instructions
  - slow: add call after every instruction to increment counter
  - faster: add call after every basic block to increment counter by number of instructions in that block.

Implementation of ATOM

- Goals
  - keep addresses the same as in uninstrumented program
  - addition of instrumented code should not affect statistics

Implementation of ATOM cont.

- Use 2 standard run-time libraries, one instrumented, one not
- addresses for data stay in place
- program addresses may change, but mapping to original location known
- Dynamically allocated memory for analysis & program either together or separate.