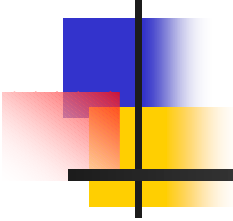


# Automatic Synthesis of High-Speed Processor Simulators



---

Martin Burtscher and Ilya Ganusov  
Computer Systems Laboratory  
Cornell University



# Motivation

---

- Processor simulators are invaluable tools
  - They allow us to cheaply and quickly test ideas
- Problem
  - Portable simulators tend to be slow
  - Fast simulators are complex and either require access to source code or symbol table info, are ISA specific (non-portable), need dynamic compilation support, or perturb the simulation





# Functional Simulation

---

- Simulates correct behavior but not timing
  - Used for prototyping, trace generation, etc.
- Needed for fast forwarding (sampling)
  - Integral part of cycle-accurate simulators
  - Average fast-forwarding and simulation time for SPECcpu2000 with early SimPoints
    - sim-fast + sim-mase:  $1.9h + 1.25h = 3.15h$
    - SyntSim + sim-mase:  $0.25h + 1.25h = 1.5h$





# Contributions

---

- Goal
  - Develop a functional simulator that is simple, portable, and fast (+ supports instrumentation)
- Our approach: SyntSim
  - Before every run, statically synthesize a simulator that is optimized for the given binary
  - Combine interpreted- and compiled-mode simulation for speed and simplicity
  - Perform other important optimizations





# SyntSim's Features

---

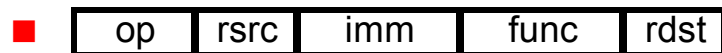
- **Simplicity**
  - Only a little more complex than an interpreter
  - Even works with stripped executables
  - Easy to add code to simulate caches, etc.
- **Portability**
  - Emits C source code
  - Does not perturb simulation
- **Performance**
  - Only 6.6x slower than native execution on SPECcpu2000 reference runs (geo. mean)



# Interpreted-Mode Simulation

- Instruction example

- `addq r7, 200, r22`



- Interpreted code

- Slow simulation speed
- Handles all adds in all programs
- Compiled once

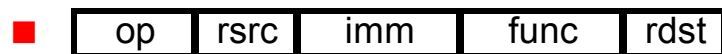
```
inst = mem[pc];
op = inst >> 26;
switch (op) {
  case ALUop:
    rsrc = (inst >> 21) & 31;
    imm = (inst >> 13) & 255;
    func = (inst >> 5) & 255;
    rdst = inst & 31;
    switch (func) {
      case AddI:
        reg[rdst] = reg[rsrc] + imm;
        pc++;
    }
}
```



# Compiled-Mode Simulation

- Instruction example

- `addq r7, 200, r22`



- Translated code

- Fast simulation speed
- Only handles this add in this program
- Incurs synthesis and compilation overhead

`reg[22] = reg[7] + 200;`

- Optimizations

- No decoding
- Hardcoded indices and immediates
- Other optims.





# Mixed-Mode Simulation

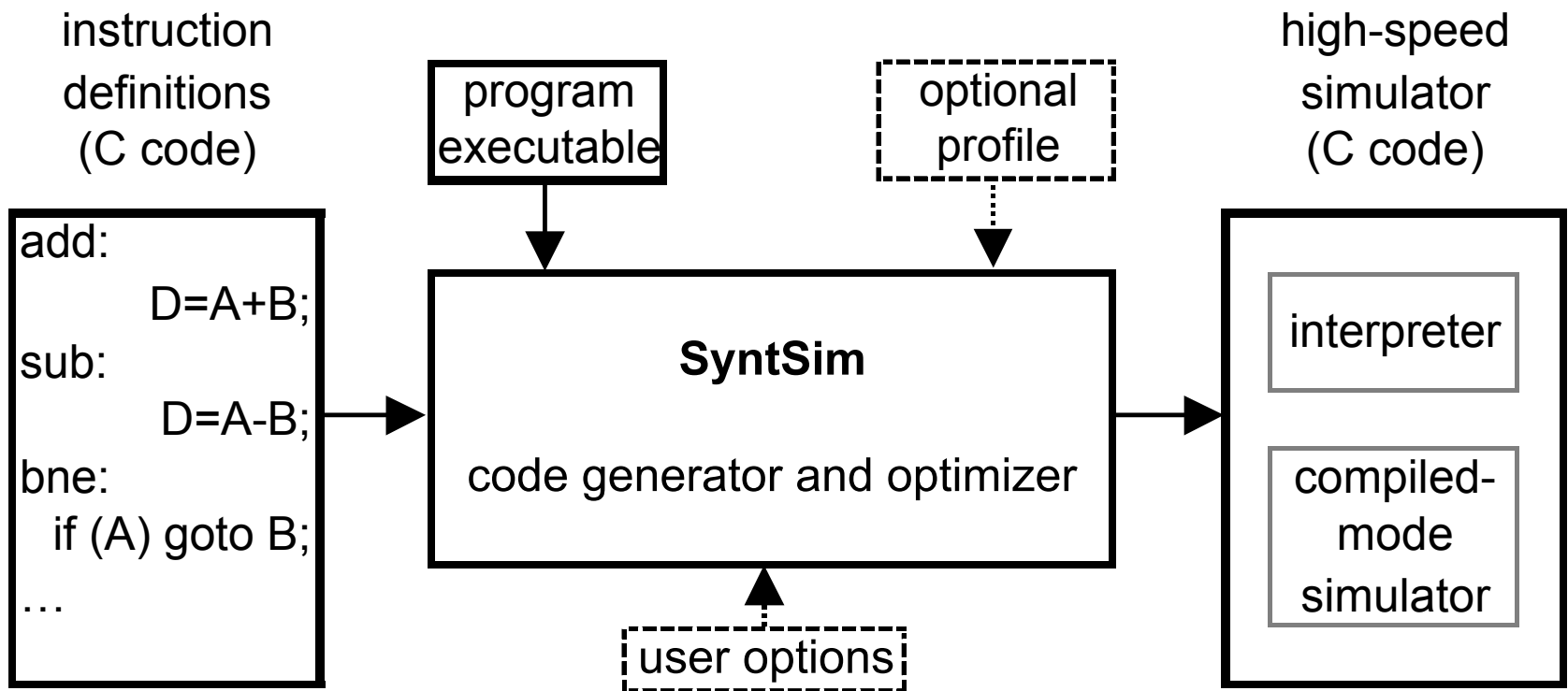
---

- Combine interpreted and compiled mode
  - Translating the 15% most-frequently executed static instructions suffices to run 99.9% of the dynamic instruction in compiled mode
  - Remaining instructions are interpreted
- Translating only frequently executed instrs
  - Much shorter compilation time
  - Smaller executable (better i-cache performance)





# SyntSim's Operation





# Compiled-Mode Simulator

```
forever {
  switch (pc/4) {
    case 0x4800372c:
      r[2] = RdMem8(r[1]-30768);          // 12000dcb8: ldq r2, -30768(r1)
      s1 = r[0]; s2 = r[4];              // 12000dcd0: cmplt r0, r4, r0
      r[0] = 0; if (s1<s2) r[0] = 1;
      ic += 3;
      if (0!=r[0]) goto L12000dcf0;      // 12000dcdc: bne r0, 12000dcf0
      ic += 1;
      goto L12000c970;                   // 12000dcec: br r31, 12000c970
    L12000dcf0:
      ic += 1;
      pc = r[26]&(~3ULL);                // 12000dcf0: ret r31, (r26), 1
      icnt[fnc(lasttarget)] += ic; ic = 0; lasttarget = pc;
      break;
    default:
      RunInterpreted();
  } // switch
} // forever
```





# Related Work

---

- MINT 1994
  - Dyn. decompile short code sequences into fncs
- QPT/EEL 1994/1995
  - Rewrite executable, use quite precise algorithm for indirect branches, need dyn. translation
- SuperSim 1996
  - Static decompilation into C, fully labeled
- UQBT 2000
  - Decompilation into special high-level language, static hooks to interpret untranslated code





# Evaluation Methodology

---

- System

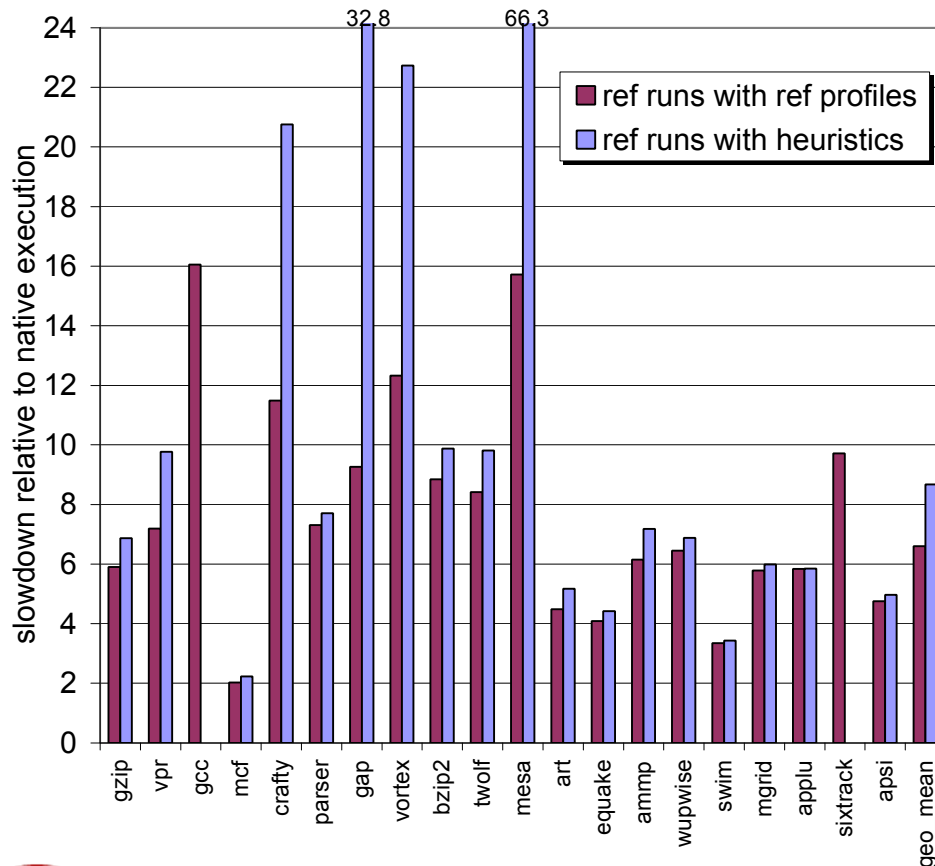
- 750MHz 64-bit Alpha 21264A
- 64kB L1, 8MB L2, 2GB RAM
- Tru64 UNIX V5.1

- Benchmarks

- 20 SPECcpu2000 programs, highly optimized
- All F77 and C programs except perlbnk
- Full test, train, and reference runs



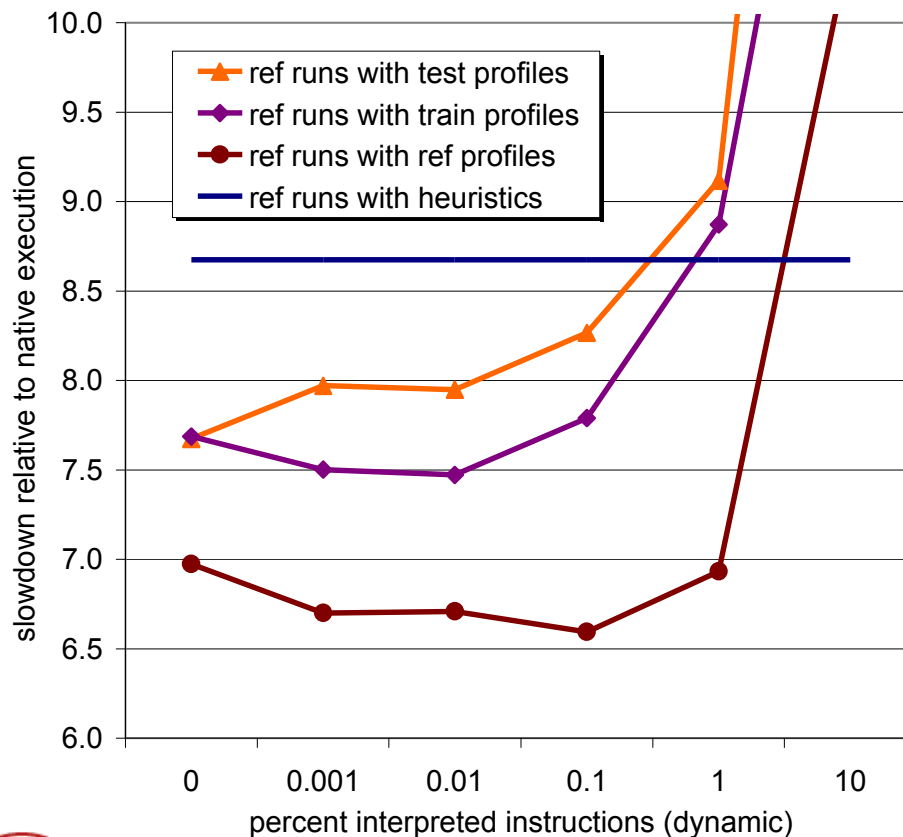
# Profile vs. Heuristic Performance



- Runtimes include
  - Synthesis time (0.08s)
  - Compilation time (33s)
  - Simulation time (3160s)
- Profile based
  - 6.6x gmean slowdown (2x to 16x)
- Heuristic based
  - 8.7x gmean slowdown (2.2x to 66x)



# Mixed-Mode Performance

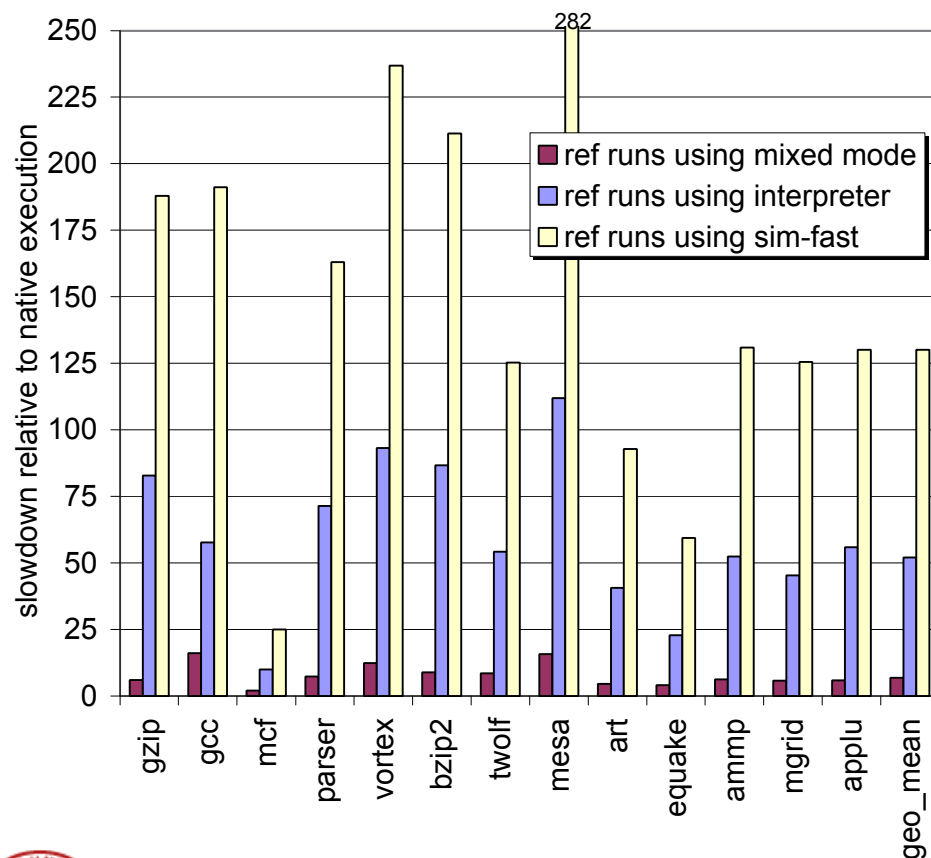


## ■ Observations

- Better profiles help
- Pure compiled mode is slower than mixed mode with good profile (24% on train runs)
- Best c/i ratio decreases with quality of profile
- 99.9% compiled mode is best with self profile (15% of static instrs)



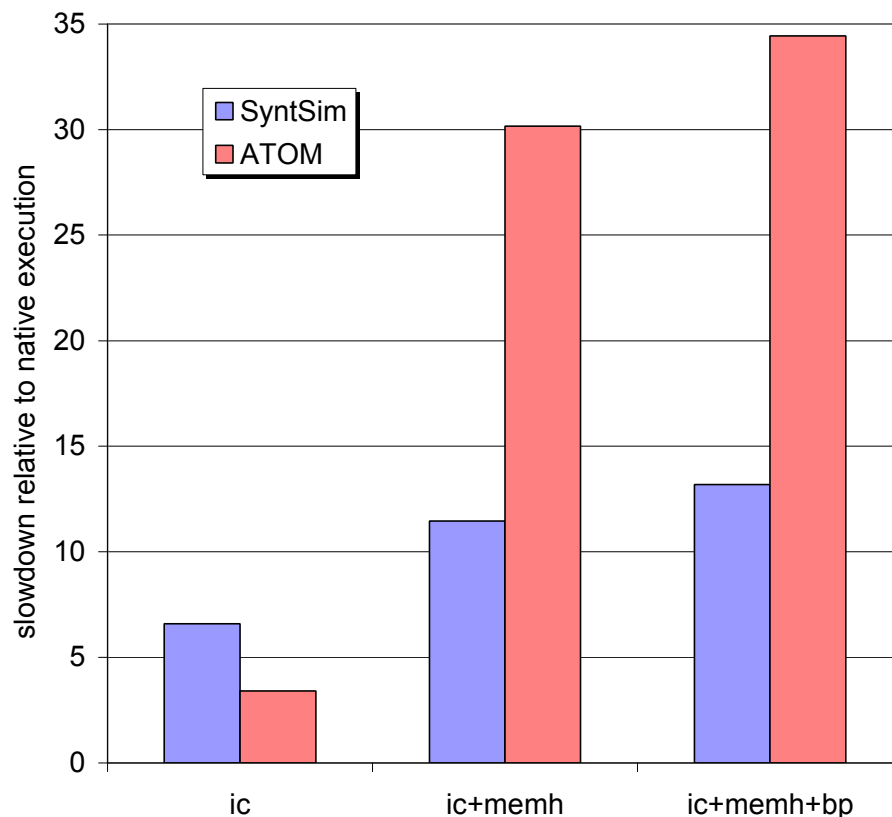
# Comparison with Interpreters



- SyntSim's interpreter
  - 2.5x faster than sim-fast
- Mixed mode
  - 19x faster than sim-fast
  - 8x faster on ref runs than SyntSim interpreter (3.6x to 14x)
  - 7x faster on train runs
  - 3.7x faster on test runs



# Comparison with ATOM



## ■ Adding instrumentation

- Identical C code
- Instruction count (ic)
- Mem hierarchy (memh)
- Branch predictor (bp)

## ■ Results

- ic: ATOM is 2x faster
- rest: SyntSim is 2.6x faster than ATOM







# Conclusions

---

- Presented a fully automated technique to statically create fast yet portable simulators
- Interleaves compiled- and interpreted-mode simulation for speed and simplicity
- Only 6.6x slower than native execution
- Only 13x slowdown when counting instructions and simulating a memory hierarchy and a branch predictor (warmup)

