CS184b: Computer Architecture [Single Threaded Architecture: abstractions, quantification, and optimizations]

Day15: February 27, 2000

Binary Translation

Today

- Problem
- Idea
- Complications
- Strategy
- ? Success?

Problem

- Lifetime of programs >> lifetime of piece of hardware (technology generation)
- Getting high performance out of old, binary code in hardware is expensive
  - superscalar overhead…
- Recompilation not viable
  - only ABI seems well enough defined; captures and encapsulates whole program
- There are newer/better architectures that can exploit hardware parallelism

Idea

- Treat ABI as a source language
  - the specification
- Cross compile (translate) old ISA to new architecture (ISA?)
- Do it below the model level
  - user doesn't need to be cognizant of translation
- Run on simpler/cheaper/faster/newer hardware

Complications

- User visibility
- Preserving semantics
  - e.g. condition code generation
- Interfacing
  - preserve visible machine state
  - interrupt state
- Finding the code
  - self-modifying/runtime generated code
  - library code

Base

- Each operation has a meaning
  - behavior
  - affect on state of machine
- stws r29, 8(r8)
  - tmp=r8+8
  - store r29 into [tmp]
- add r1,r2,r3
  - r1=(r2+r3) mod 2^{31}
  - carry flag = (r2+r3>= 2^{31})

Capture Meaning

- Build flowgraph of instruction semantics
  - not unlike the IR (intermediate representation) for a compiler
    - what use to translate from a high-level language to ISA/machine code
  - e.g. IR saw for Bulldog (trace scheduling)

Optimize

- Use IR/flowgraph
  - eliminate dead code
    - esp. dead conditionals
    - e.g. carry set which is not used
  - figure out scheduling flexibility
    - find ILP

Trace Schedule

- Reorganize code
- Pick traces as linearize
- Cover with target machine operations
- Allocate registers
  - (rename registers)
  - may have to preserve register assignments at some boundaries
- Write out code

Details

- Seldom instruction→instruction transliteration
  - extra semantics (condition codes)
  - multi-instruction sequences
    - loading large constants
    - procedure call return
  - different power
    - offset addressing?,
    - compare and branch vs. branch on register
- Often want to recognize code sequence

Complications

- How do we find the code?
  - Known starting point
  - ? Entry points
  - walk the code
  - …but, ultimately, executing the code is the original semantic definition
    - may not exist until branch to...

Finding the Code

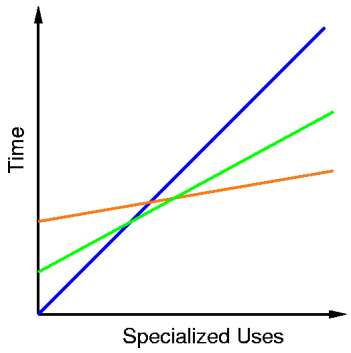
- **Problem:** can't always identify statically
- **Solution:** wait until "execution" finds it
  - delayed binding
  - when branch to a segment of code,
    - certainly know where it is
    - and need to run it
  - translate code when branch to it
    - first time
    - nth-time?

Common Prospect

- Translating code is large fixed cost
  - but has low incremental cost on each use
  - hopefully comparable to or less than running original on old machine
- Interpreting/Emulating code may be faster than "compiling" it
  - if the code is run once
- Which should we do?

Optimization Prospects

- Translation vs. Emulation
  - $T_{trun} = T_{trans} + nT_{op}$
  - $T_{trns} > T_{em\_op} > T_{op}$
- If compute long enough
  - $nT_{op} >> T_{trans}$
  - $\rightarrow$ amortize out load



Competitive Approach

- Run program emulated
- When a block is run "enough", translate
- Consider
  - $N_{thresh}\ T_{emop} = T_{translate}$
- Always w/in factor of two of optimal
  - if $N<N_{thresh}$ optimal
  - if $N=N_{thresh}$ paid extra $T_{translate}$ =2×optimal
  - as $N>>N_{thresh}$ extra time amortized out with translation overhead
    - think $T_{translate} \sim=2T_{translate}$

On-the-fly Translation Flow

- Emulate operations
- Watch frequency of use on basic blocks
- When run enough,
  - translate code
  - save translation
- In future, run translated code for basic block

Translation "Cache"

- When branch
  - translate branch target to new address
  - if hit, there is a translation,
    - run translation
  - if miss, no translation
    - run in emulation (update run statistics)

Alternately/Additionally

- Rewrite branch targets so address translated code sequence
  - when emulator finds branch from translated sequence to translated sequence
  - update the target address of the branching instruction to point to the translated code

Self-Modifying Code

- Mark pages holding a translated branch as read only
- Take write fault when code tries to write to translated code
- In fault-handler, flush old page translation

Precise Exceptions

- Again, want exception visibility relative to simple, sequential model
  - …and now old instruction set model
- Imposing ordering/state preservation is expensive

Precise Exceptions

- Modern BT technique [hardware support]
  - "backup register" file
  - commit/rollback of register file
  - commit on memories
  - on rollback, recompute preserving precise state
    - drop back to emulation?
- …active work on software-only solutions
  - e.g. IBM/WBT'00

Remarkable Convergence?

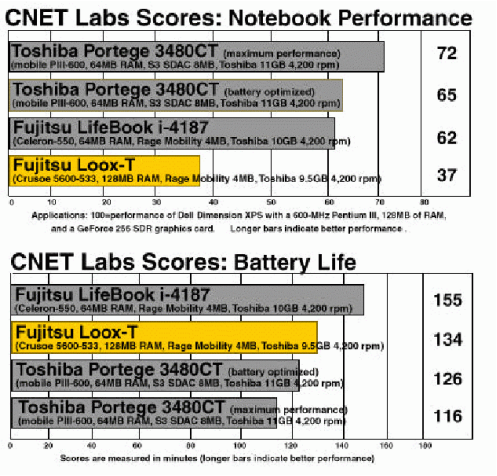
- Aries: HP PA-RISC→IA-64
  - new architecture
- IBM: PowerPC→BOA
  - ultra-high clock rate architecture? (2GHz)
    - IBM claims 50% improvement over scaling?
    - 700ps = 1.4GHz in 0.18μm
- Transmeta: x86 →Crusoe
  - efficient architecture, avoid x86 baggage

Remarkable Convergence?

- All doing dynamic translation
  - frequency based
- To EPIC/VLIW architectures

Performance

[CAVEAT: trade magazine, numbers for system]



Academic Static Binary Translation

Program	Translated Code		Native Code	
	gcc opt	cc opt	-O0	-O4
Fibo(40) sec	27.7	28.5	28.6	25.9
bytes	16,512	7,292	16,144	16,152
Sieve(3000) sec	17.8	17.4	18.9	18.6
bytes	16,244	6,548	15,964	15,944
Mbanner(500K) sec	42.5	n/a	80.5	44.8
bytes	22,240		21,524	25,436

Static SPARC to Pentium Translation

[Cifuentes et. al., Binary Translation Workshop 1999]

Academic/Static BT

Program	Translated Code		Native Code	
	gcc opt	cc opt	-O0	-O4
Fibo(40) sec	23.0	24.3	41.0	23.0
bytes	24,916	6,680	24,628	24,564
Sieve(3000) sec	26.9	23.9	29.3	24.5
bytes	24,776	6,312	24,552	24,452
Mbanner(500K) sec	53.3	36.9	63.7	26.6
bytes	34,188	21,448	30,652	30,268

Static Pentium to SPARC Translation

[Cifuentes et. al., Binary Translation Workshop 1999]

Academic/Dynamic BT

			Without Hot Paths		With Hot Paths			
Test programs	Startup time	Translation time	Execution time without reg caching	Execution time with reg caching	Optimisation time	Execution time without reg caching	Execution time with reg caching	Native gcc compiled
Sieve3000	0.54	0.14	98.25	73.14	0.16	80.98	66.29	29.22
Fibonacci	0.54	0.10	186.17	154.97	0.11	147.56	133.69	41.18
mbanner	0.52	0.34	219.01	146.28	0.37	146.22	126.28	22.85

Table 1: Pentium to SPARC translation (second)

[Ung+Cifuentes, Binary Translation Workshop 2000]

Upcoming

- Next Class (Last): Thursday, March 8th
  - no class this Thursday
  - no class Tuesday (week from today)
- Want to see:
  - assignments 6, 7, 8
  - finished by next class (3/8)

Big Ideas

- Well-defined model
  - High value for longevity
  - Preserve semantics of model
  - How implemented irrelevant
- Hoist work to earliest possible binding time
  - dependencies, parallelism, renaming
  - hoist ahead of execution
    - ahead of heavy use
  - reuse work across many uses
- Use feedback to discover common case