ECE 571 – Advanced Microprocessor-Based Design Lecture 5

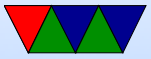
Vince Weaver

http://www.eece.maine.edu/~vweaver

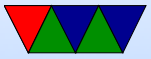
vincent.weaver@maine.edu

29 January 2013

Performance Analysis

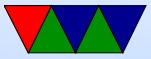


First some Last ARM Assembly



IT (If/Then) Instruction

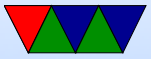
- Allows limited conditional execution in THUMB-2 mode.
- The directive is optional (and ignored in ARM32)
  the assembler can (in-theory) auto-generate the IT instruction
- Limit of 4 instructions



Example Code

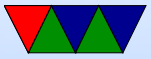
```
it cc
addcc r1,r2

itete cc
addcc r1,r2
addcs r1,r2
addcc r1,r2
addcs r1,r2
```

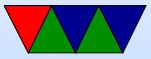


`ll` Example Code

```
        ittt cs @ If CS Then Next plus CS for next 3
discrete_char:
        ldrbcs  r4,[r3]       @ load a byte
        addcs   r3,#1         @ increment pointer
        movcs   r6,#1         @ we set r6 to one so byte
        bcs.n   store_byte    @ and store it
offset_length:
```

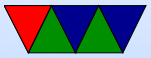


Introduction to Performance Analysis



What is Performance?

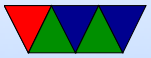
- Getting results as quickly as possible?
- Getting *correct* results as quickly as possible?
- What about Budget?
- What about Development Time?
- What about Hardware Usage?
- What about Power Consumption?



Motivation for HPC Optimization

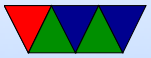
HPC environments are expensive:

- Procurement costs: ~$40 million
- Operational costs: ~$5 million/year
- Electricity costs: 1 MW / year ~$1 million
- Air Conditioning costs: ??

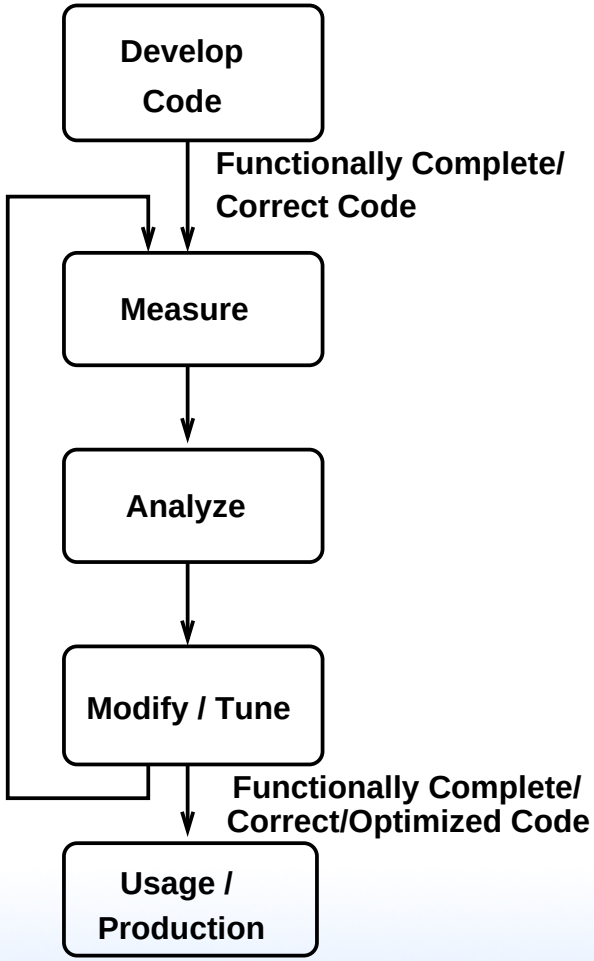


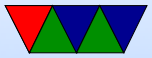
Know Your Limitation

- CPU Constrained
- Memory Constrained (Memory Wall)
- I/O Constrained
- Thermal Constrained
- Energy Constrained



Performance Optimization Cycle



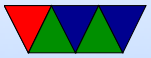


Wisdom from Knuth

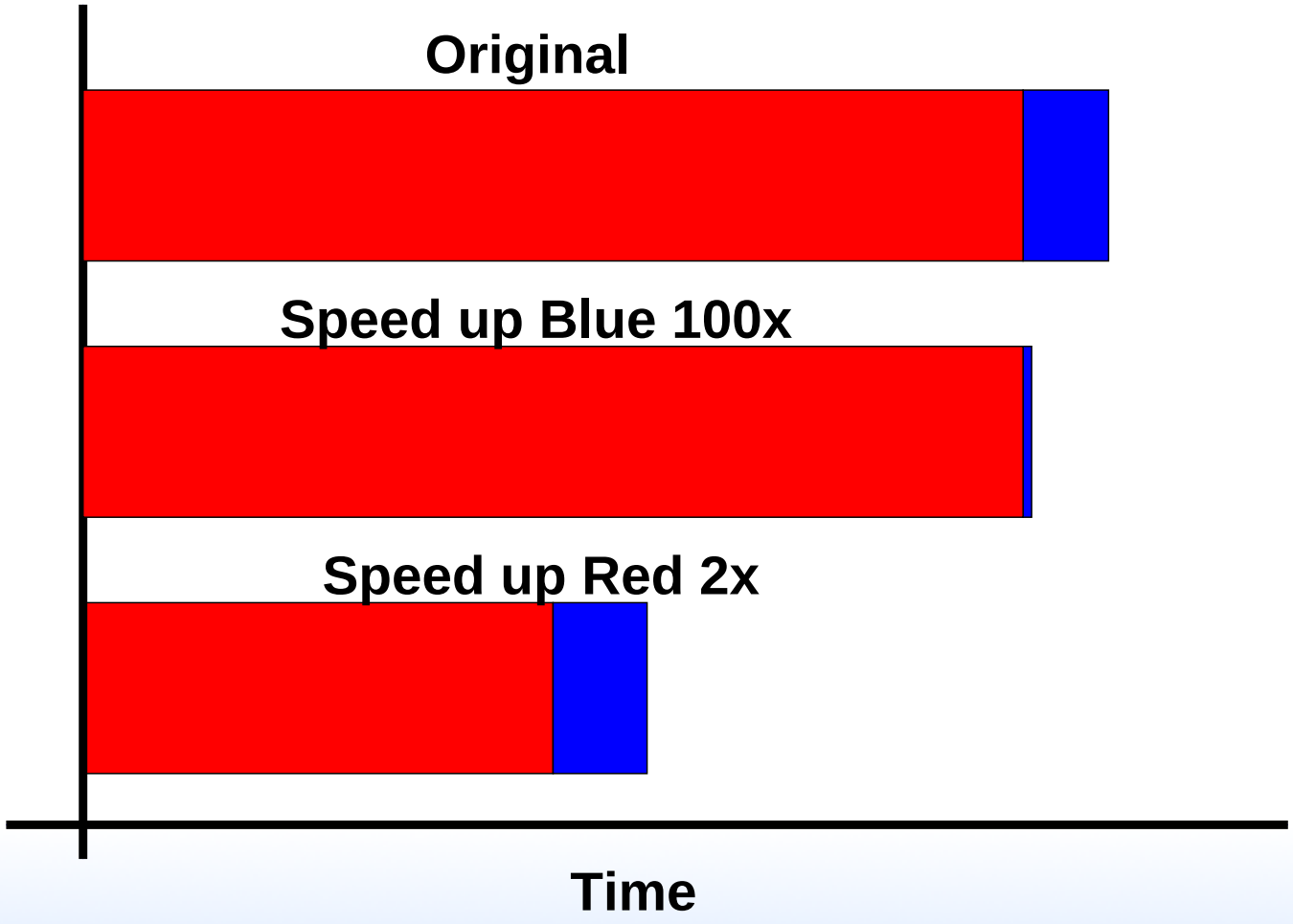
"We should forget about small efficiencies, say about 97% of the time:

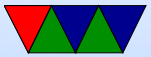
premature optimization is the root of all evil.

Yet we should not pass up our opportunities in that critical 3%. A good programmer will not be lulled into complacency by such reasoning, he will be wise to look carefully at the critical code; but only after that code has been identified" — Donald Knuth



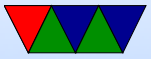
Amdahl's Law



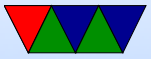


Gathering Performance Info

- User Level (instrumentation)
- Kernel Level (kernel metrics)
- Hardware Level (performance counters)



User Level: Profiling vs Tracing



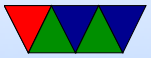
Profiling and Tracing

Profiling

- Records aggregate performance metrics
- Number of times routine invoked
- Structure of invocations

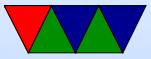
Tracing

- When and where events of interest took place
- Time-stamped events
- Shows when/where messages sent/received



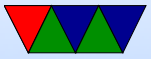
Profiling Details

- Records summary information during execution
- Usually Low Overhead
- Implemented via **Sampling** (execution periodically interrupted and measures what is happening) or **Measurement** (extra code inserted to take readings)



Tracing Details

- Records information on significant events
- Provides timestamps for events
- Trace files are typically *huge*
- When doing multi-processor or multi-machine tracing, hard to line up timestamps



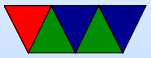
Performance Data Analysis

Manual Analysis

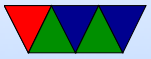
- Visualization, Interactive Exploration, Statistical Analysis
- Examples: TAU, Vampir

Automatic Analysis

- Try to cope with huge amounts of data by automatic analysis
- Examples: Paradyn, KOJAK, Scalasca, Perf-expert

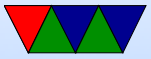


Software Tools for Performance Analysis



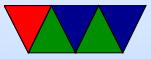
Simulators

- Architectural Simulators
- Can generate traces, profiles, or modeled metrics
- Slow, often 1000x or more slower
- Not real hardware, only a model
- Did I mention, slow?
- m5, gem5, simplescalar, etc



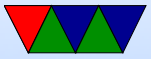
Dynamic Binary Instrumentation

- Pin, Valgrind (cachegrind), Qemu
- Still slow (10-100x slower)
- Can model things like cache behavior (can model parameters other than system running on)
- Complicated fine-tuned instrumentation can be created
- Architecture availability – Pin (no longer ARM), Valgrind, Qemu most architectures, hardest to use



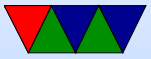
Compiler Profiling

- gprof
- gcc -pg
- Adds code to each function to track time spent in each function.
- Run program, gmon.out created. Run "gprof executable" on it.
- Adds overhead, not necessarily fine-tuned, only does

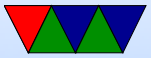


time based measurements.

- Pro: available wherever gcc is.

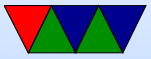


Hardware Tools for Measuring Performance



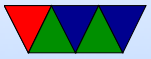
What are Hardware Performance Counters?

- Registers on CPU that measure low-level system performance
- Available on most modern CPUs; increasingly found on GPUs, network devices, etc.
- Low overhead to read



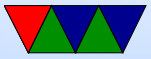
Hardware Implementation of Counters

- Not much documentation available
- Jim Callister/Intel: "Confessions of a Performance Monitor Hardware Designer" 2005, Workshop on Hardware Performance Monitor Design
  - Transistors free, wires not. Also design time, validation, documentation, time to market. PMU has tentacles "everywhere" bringing data back to center.
  - Architect too much, lower performance, events don't



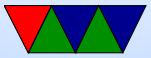
map well to hardware. Architect too little.. software design harder.

- Which events are important? Are cache misses important if don't hurt performance? (no stalls)
- Mapping events to signal difficult. On critical path. Not enough wires. Combining signals hard if distance between wires.
- Use logging. May miss events in "shadow" of another event being logged. Use random behavior?



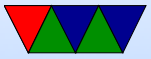
Other Concerns

- Power/Energy: on CMOS P=ACV**2f
  A = activity, C = capacitance, V=voltage, f = frequency
  fast-running counter increments every cycle, high activity



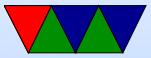
Learning About the Counters

- Number of counters varies from machine to machine
- Available events different for every vendor and every generation
- Available documentation not very complete (Intel Vol3b, AMD BKDG, ARM ARM/TRM)



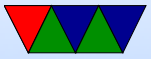
Low-level interface

- on x86: MSRs
- ARM: CP15 system control register



CP15 registers on Cortex A9

- 6 counters available
- 58 events, 17 architectural, 41 A9 Specific, split between Approximate, Precise
- No way to specify kernel vs user (Cortex A15 does?)
- Cortex A9 has bug where PMU interrupts may be lost

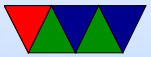


CP15 Interface

- use `mcr`, `mrc` to move values in/out

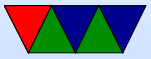
  ```
  MRC p15,0,Rt,c9,c12,0
  MCR p15,0,Rt,c9,c12,0
  ```

- Six EVNTCNT registers
- Cycle Counter register
- Six Event Config registers
- Count enable set/clear, count interrupt enable/clear,



overflow, software increment

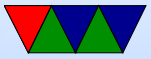
- PMU management registers
- in general only privileged access (why) but can be configured to let users access.



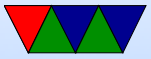
Registers

- PMCR – IMP/IDCODE (about implementer), N (number of counters, up to 32), Disable when prohibited (avoid counting in sensitive zones), X (export results to external debug hardware), D clock divider (optionally only count every 64th clock), Reset clock, reset all events, enable all events

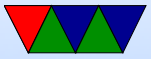
- ENSET – bitfield enabling events, also on read tells if all enabled



- ENCLR – bitfield clearing events, disables the counters
- PMOVSR – overflow flags for all events
- SWINC – increment software counter
- PMSELR – selects “current” counter
- PMCCNTR – set/read cycle counter value
- EVTYPER – sets which event is used for counter
- EVCNTR – set/read event counter value



- USERENR – allow user access to counters
- INTENSET – enable bits for overflow interrupts
- INTENCLR – clear bits for overflow interrupts



Overflow

- overflows after reaching 2**32
- If want to overflow earlier, init to a high value. So 0xc0000000 to overflow at 1 billion