ECE 571 – Advanced Microprocessor-Based Design Lecture 14

Vince Weaver

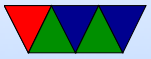
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

20 March 2018

Announcements

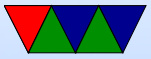
- Midterm Thurs
- Posted project description to website.



Midterm Review

Closed book/laptop/phone but can have front of one 8.5x11 piece of paper worth of notes if you want.

1. Performance/Benchmarking
   - Be familiar with the general idea of performance counters and interpreting perf results.
   - Benchmark choice: it should match what you plan to do with the computer.
   - Know a little about the difference between integer benchmarks and floating point (integer have

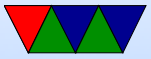


more random/ unpredictable behavior with lots of conditionals; floating point are often regular looped strides over large arrays or data sets)

- Be familiar with concept of skid.

2. Power
   - Know the CMOS Power equation
   - Energy, Energy Delay, Energy Delay Squared
   - Idle Power Question

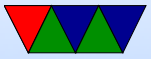
3. Branch Prediction
   - Static vs Dynamic



- 2-bit up/down counter
- Looking at some simple C constructs say expected branch predict rate

4. Cache

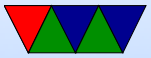
- Given some parameters (size, way, blocksize, addr space) be able to calculate number of bits in tag, index, and offset.
- Know why caches are used, that they exploit temporal and spatial locality, and know the tradeoffs (speed vs nondeterminism)



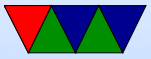
- Be at least familiar with the types of cache misses (cold, conflict, capacity)
- Know difference between writeback and write-through
- Be able to work a few simple steps in a cache example (like in HW#5)

5. Prefetch
   - Why have prefetchers?
   - Common prefetch patterns?

6. Virtual Memory
   - General concept of VM



- Benefits of VM?
  Memory Protection, each program has own address space, allows having more memory than physical memory, demand paging, copy-on-write for fork, less memory fragmentation, etc.
- Why is TLB behavior important?
  Depending on cache config:
  worst case: (VIVT) every memory access looked up in TLB best case: (PIPT) every cache miss looked up in TLB

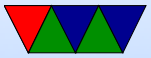


HW#6 Review

- Intel: prefetcht0/prefetcht1/prefetcht2/prefetchnta
  Level of cache, non temporal.

  objdump --disassemble-all ./bzip2 | grep prefetch | wc -l
  5 of them
  objdump --disassemble-all ./bzip2.swprefetch | grep prefetch
  ./bzip2.swprefetch: file format elf64-x86-64
    401397:   0f 18 0b   prefetcht0 (%rbx)
    401e8c: 0f 18 88 a0 00 00 00   prefetcht0 0xa0(%rax)
  ... * 25 lines

- objdump --disassemble-all ./equake_l | grep prefetc
  (nothing)
  objdump --disassemble-all ./equake_l.swprefetch | grep prefetch
  ./equake_l.swprefetch: file format elf64-x86-64

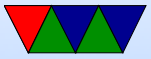


```
40192a:       0f 18 0a                prefetcht0 (%rdx)
401b8d:       0f 18 4d 48             prefetcht0 0x48(%rbp)
401be8:       0f 18 4d 48             prefetcht0 0x48(%rbp)
402036:       0f 18 09                prefetcht0 (%rcx)
40479a:       0f 18 4d 00             prefetcht0 0x0(%rbp)
4047a9:       0f 18 0b                prefetcht0 (%rbx)
```

- BZIP

		l2-cache-misses	prefetches	time
1a:	bzip2:	33.1%	167M	3.14s
2a:	SW prefetch:	33.3%	167M	3.16s
5a:	HWdisable	43.3%	174k	3.28s
5a:	HWdisable+Sw	43.9%	197k	3.25s

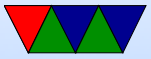
- Equake



		l2-cache-misses	prefetches	time
3a:	equake_l:	15.2%	38B	79.2s
4a:	equale_l swpref	16.0%	38B	79.3s
5a:	hwdisable	69.7%	9M	137.5s
5a:	hwdisable swpref	68.3%	8M	127.4s

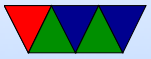
- Summary: disabling prefetch hurt, dramatically so on equake.
  Unclear what exactly the prefetch perf counter is measuring
  Enabling SW prefetch does not seem to do much, even with HW prefetch disabled.

- Why? Lots of possible reasons. compiler bug. hardware



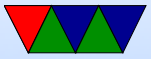
bug. hardware engineers not enable SW prefetch (is it incorrect to ignore?) other.

- Errata / Specification Update. Surprised a major chip has so many? Did this affect our results? Errata says not our event, but the perf list description (also written by Intel) says it might.



Spectre Security

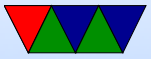
- Unlike Meltdown, pretty much any processor with speculative execution affected
- Doesn't leak info from kernel, but from one part of program to another
- Why a problem? Well if javascript can read anything in rest of browser (passwords, history, etc)
- SPECulative execution, will haunt us for ages



Spectre Variant 1 – Bounds Check

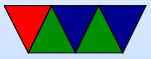
```
if (x<array1_size)
        y = array2[array1[x]*256];
```

- Ideally finds this code already existing in user code
- If mispredicts the check, will speculatively access the out-of-bounds value
- Attacker controls X
- Attacker trains the branch predictor that value is true with lots of runs
- Then passes in a value that is wrong but branch is



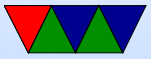
predicted the previous way.

- array1_size is not cached, so it stalls and execution goes beyond
- Probe the cache much like meltdown



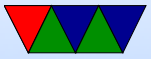
Indirect Branches

- Instead of relying on user code, train up the BTB
- Doesn't have to be the same address space, just has to alias in the BTB
- On many machines only 30 or fewer bits of BTB used to index



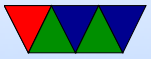
Spectre Variant 2 – Branch Target Injection

- X maliciously chosen
- Branch prediction manipulated to predict wrong
- arrays all kicked out of memory
- array1size was kicked out of RAM, so cache miss and slowly get value for RAM
- meanwhile bredicts branch is good and so fetches array2[k*256]
- Eventually figures out and squashes wrong branch, but the fetch already underway into cache



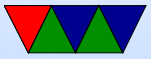
Finding a gadget

- Need to find code that runs with adversarial values are in register
- Not hard, often unused values leak across function calls (if a function doesn't use them)
- Need to find way to trigger a branch in a way that acts on these as pointers.
- Then find existing indirect jump
- Train the BTB to want to jump to our gadget
- clear out cache, perform attack



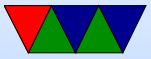
Notes

- some i7 up to 188 instructions can execute speculatively between
- can be triggered from Javascript. No clflush, but can evict all of cache by reading through an array.
- branch predictors on cpus are independent?



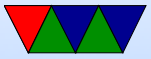
Workarounds

- Software
  - Disabe hires timers in javascript
  - Memory barriers – can halt speculation with special instructions, but have to insert them all through code where it might be an issue.
  - Kaiser/KPTI not help
  - Retpoline and IBRS, see next slides



New barriers

- Added by Intel with firmware update, new MSR
- IBRS – indirect branch restricted speculation
  flush branch predicto on entry to kernel, disable brpred on hyperthread
- STIBP – single-thread indirect branch prediction – disable brpred on sibling thread (currently they share brpred)
- IBPB – indirect br pred barrier – flush branch predictor state

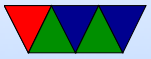


retpoline

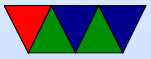
```
        jmp     #%r11

        call    set_up_target
capture_spec:
        pause;
        jmp     capture_spec

set_up_target:
        mov     %r11, (%rsp)
        ret
```

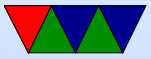


- Return trampoline
- Convert indirect branch into a `ret` in a common location, makes it hard to train branch predictor.
- Also adds a code-trap so that if code speculates past the branch it gets trapped in an infinite loop
- Downside: all indirect branches now slower retpoline.



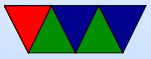
CPU Power and Energy

Lookup http://ieeexplore.ieee.org/document/6757323/

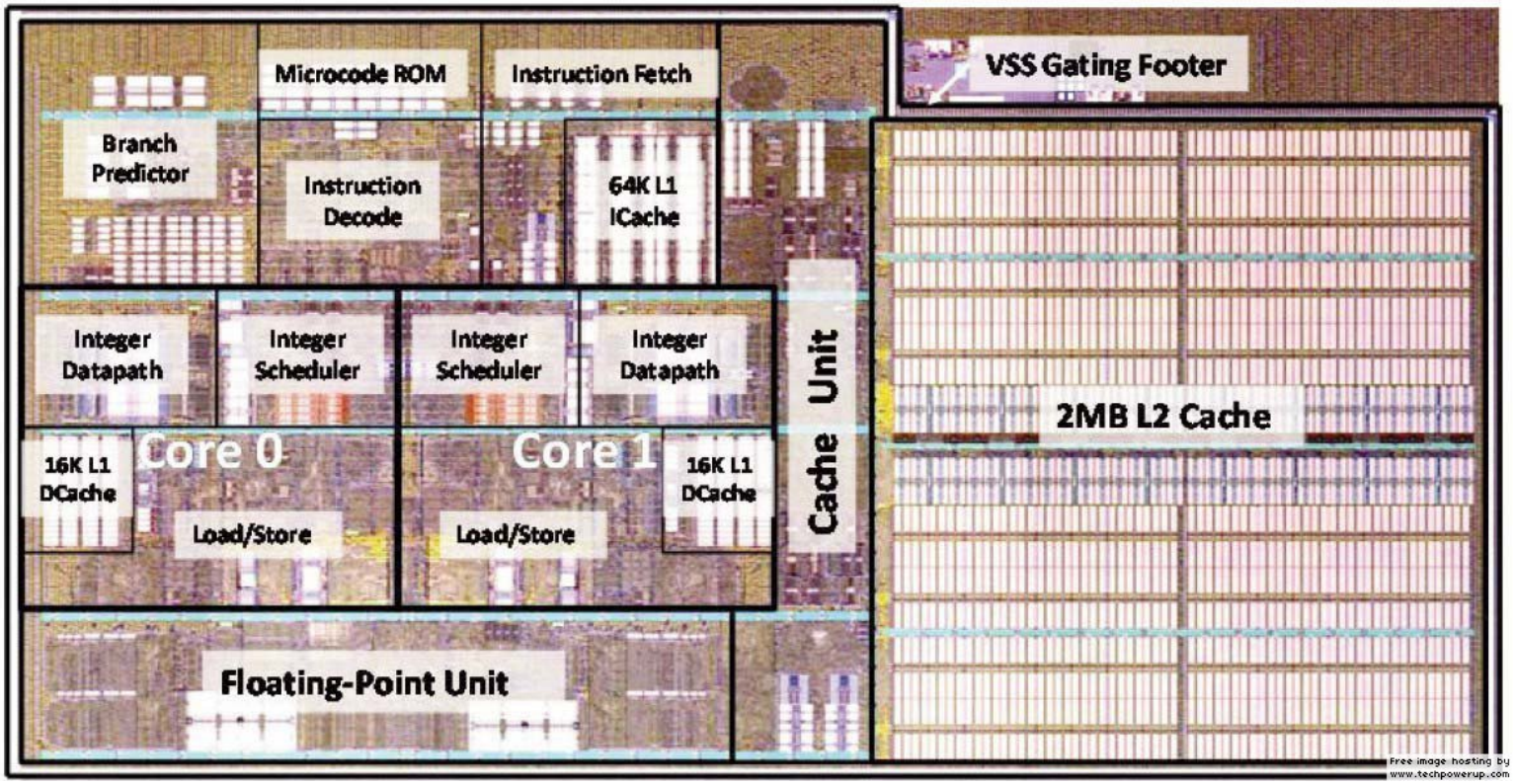


CPU Power and Energy

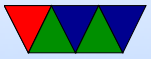
- Became a trendy thing to research in 1999-2002 timeframe.
- Before that usually concern was with performance.
- These days energy results are often reported as a core part of any architectural proposal, not as a separate issue.
- The results discussed here are academic and may or may not be implemented in actual chips.



AMD Bulldozer Die Shot



Note which structures are big, using static power.

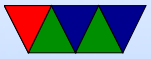


CPU Power Breakdown

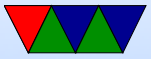
From Fan, Tang, Huan, Gao (ISLPED'05), Chinese Godson MIPS CPU

They gave numbers, but unclear of workload, if static or dynamic, etc.

- Cache 36%
- TLB 13%
- FALU 10%
- ROQueue 7%
- FMUL 6%



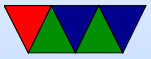
- Float reg 5%
- Gen reg 5%
- MUL 2%
- MCUControl 2%
- ALU 1%
- Other 13%



Thermal Concerns Too

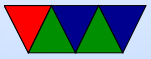
Power density exceed hot plate, approaching rocket nozzle

TODO: Find the Intel cite for this statement.



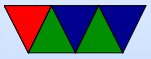
Methodologies Used in These Papers

It varies, but many of these are from simulations (sometimes validated). Anything from SPICE to "cycle-accurate" simulators.



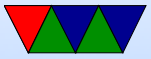
Clock Generation

- Driving high-frequency load against capacitance, trying to keep whole chip in sync.
- Extreme Case: Alpha 21264 H-tree, 32% of power?
- Half-frequency clocks (on both edge, so clock run half as fast) (Mudge 2001)
- Asynchronous
- Locally Asynchronous (Divide to multiple clock domains)



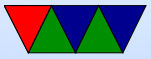
DVFS and other CPU Power/Energy Saving Methods

- A lot of related work
- Will focus on actual implementations rather than academic papers this time

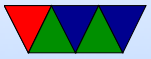


DVFS

- Voltage planes – on CMP might share voltage planes so have to scale multiple processors at a time
- DC to DC converter, programmable.
- Phase-Locked Loops. Orders of ms to change. Multiplier of some crystal frequency.
- Senger et al ISCAS 2006 lists some alternatives. Two phase locked loops? High frequency loop and have programmable divider?

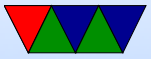


- Often takes time, on order of milliseconds, to switch frequency. Switching voltage can be done with less hassle.



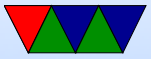
Adaptive Body Biasing

- Related to but not always considered part of DVFS
- Control voltage applied to body
- Change the threshold voltage
- Reduces leakage but slows performance



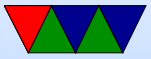
DVFS and other CPU Power/Energy Saving Methods

- A lot of related work
- Will focus on actual implementations rather than academic papers this time

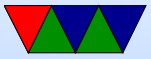


DVFS

- Voltage planes – on CMP might share voltage planes so have to scale multiple processors at a time
- DC to DC converter, programmable.
- Phase-Locked Loops. Orders of ms to change. Multiplier of some crystal frequency.
- Senger et al ISCAS 2006 lists some alternatives. Two phase locked loops? High frequency loop and have programmable divider?

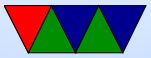


- Often takes time, on order of milliseconds, to switch frequency. Switching voltage can be done with less hassle.



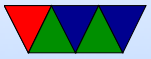
Adaptive Body Biasing

- Related to but not always considered part of DVFS
- Control voltage applied to body
- Change the threshold voltage
- Reduces leakage but slows performance



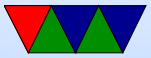
Cache Power and Energy

Large area, low-hanging fruit



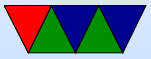
Decay Caches

- Kaxiras, Ho, Martinosi (ISCA 2001)
- Turn off cache lines not being used to reduce leakage
- DRAM cache with no refresh
- Decayed values can be re-fetched from memory. Tradeoff.



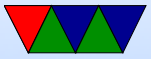
Drowsy Caches

- Flautner, Kim, Martin, Blaauw, Mudge. ISCA 2002.
- Move cold cache lines into "drowsy" mode.
  Lower power enough to hold state, not enough to lose contents. Reduce leakage. Better than decay as not lose data.
- Note: in Intel Volume 3b 17.17.5.2 it mentions certain C states might power down or otherwise turn off parts of cache.



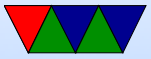
Adaptive Caches

- Albonesi (Micro 1999). Manually turn off ways in cache with an instruction.
- Size the caches



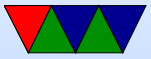
Cache Compression

- Dynamic zero compression for cache energy reduction (L Villa, M Zhang, K Asanović. Micro 2001).
- Cache Compression ("sign compression" – top bits) Energy savings 20% (simulated) (Kim, Austin, Mudge WMPI 2002)



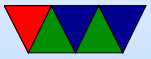
Banking and Filtering

- Filter cache, banking (only have half of cache active) (Mudge 2001)
- Slowing Down Cache Hits, Banked Data Cache. (Huang, Renau, Yoo, and Torrellas. Micro 2000.)
- Vertical Banking, Horizontal Banking (Su and Despain, ISLPED 1995).



Code Scheduling

- Can Schedule code for lower power.
- Better cache rates lower power. performance/power can go hand in hand. (Kandemir, Vijaykrishnan, Irwin)



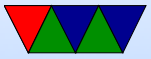
Branch Predictors

- Parikh, Skadron, Zhang, Barcella, Stan

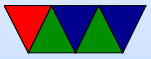
- 4 concerns:

  1. Accuracy. Not affect power, but performance
  2. Configuration (may affect power)
  3. Number of lookups
  4. Number of updates

- Tradeoff power vs time.

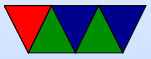


- brpred can be size of small cache, 10% of power
- Can use banking to mitigate



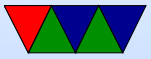
Branch Predictors

- can watch icache, not activate predictor if nobranches
- Pipeline gating, keep track of each predicted branch confidence. If confidence hits certain threshold, stop speculating. Show this may or may not be good.
- Integer code, large predictors good
- FP, tight loops, predictors not as important.



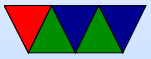
Branch Predictor Evaluation

- (Strasser, 1999). Simulation, small branch predictor can help energy.
- (Co, Weikle, Skadron) Formula for break even point. Leakage matters, what brpred hides is stall cycles.
- SEPAS: A Highly Accurate Energy-Efficient Branch Predictor (Baniasadi, Moshovos. ISLPED 2004). Once a branch prediction reaches steady state (unlikely to change) stop accessing/updating predictor, saving



energy.

- Low Power/Area Branch Prediction Using Complementary Branch Predictors (Sendag, Yi, Chuang, Lija. IPDPS 2008)
  Complementary Branch Predictor to handle the tough cases.

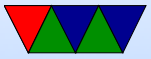


Prefetching

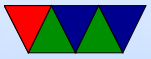
- Prefetching does not get looked at as closely.
  Various studies show it can be a win energy wise, but it is a close thing.

- (Guo, Chheda, Koren, Krishna, Moritz. PACS'04)
  HW Prefetch increase power 30%; have compiler help augment with hints, filters.

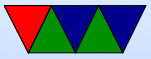
- (Tang, Liu, Gu, Liu, Gaudiot. Computer Architecture Letters, 2011).



Mixed results.



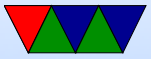
TLB Energy



TLB Optimization – Assume in Same Page

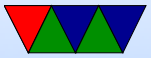
- Optimizing instruction TLB energy using software and hardware techniques (Kadayif, Sivasubramaniam, Kandemir, Kandiraju, Chen. TODAES 2005).
  Don't access TLB if not necessary. Compare to last access (assume stay in same page) Circuit improvements

- (Kadayif,Sivasubramaniam, Kandemir, Kandiraju, Chen. Micro 2002)
  Generating Physical Addresses Directly for Saving Instruction TLB Energy Cache page value.



TLB Optimization – Use Virtual Caches

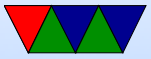
- (Ekman and Stenström, ISLPED 2002) Use virt address cache. Less TLB energy, more snoop energy. TLB keeps track of shared pages.



TLB Optimization – Reconfiguring

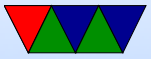
- (Basu, Hill, Swift. ISCA 2012) Reducing Memory Reference Energy with Opportunistic Virtual Caching
  Have the OS select if memory region physical or virtual cached.

- (Delaluz, Kandemir, Sivasubramaniam, Irwin, Vijaykrishnan. ICCD 2013) Reducing dTLB Energy Through Dynamic Resizing.
  Size TLB as needed, shutting off banks. Easier if fully-associative.



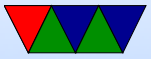
TLB Optimization – Memory Placement

- (Jeyapaul, Marathe, Shrivastava, VLSI'09) Try to keep as much in one page as possible via compiler.
- Energy Efficient D-TLB and Data Cache using Semantic-Aware Multilateral Partitioning (Lee, Ballapuram. ISLPED'03) Split memory regions by region (text/data/heap). Better TLB performance, better energy.



Bus Protocols

- Bus Protocols
- Cache-Coherence Protocols



Busses

- Grey Code, only one bit change when incrementing. Lower energy on busses? (Su and Despain, ISLPED 1995).