ECE 571 – Advanced Microprocessor-Based Design Lecture 12

Vince Weaver

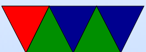
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

2 March 2017

Announcements

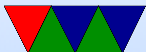
- HW#6 Due
- Tentative project assignment posted to website.
- No homework over break



HW#6 Review

1. Haswell machine has a 44-bit physical address space, 32-kB L1 data cache, 8-way set associative, 64-bytes per line.

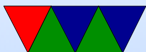
   (a) Offset $=$ 6 bits
   (b) Index $= 2^{15}/2^{3}/2^{6} = 2^{6} =$ 6 bits
       We'll find out later that having 12 bits of offset+index makes VIPT caches easier (4096 bytes)
   (c) 32-bit tag



2. Cache Example

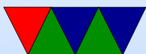
(a) ld 0000 080f = line 0, tag 8 = hit
(b) ld ffff ffff = line f, tag ffffff = miss (cold)
(c) strb 0000 0810 = line 1 tag 8 = miss (unknown type), and LRU says we throw out tag a, which is dirty, so writeback
(d) strb r0, 0xffffffff – hit. Set dirty bit

3. Bzip2 on Haswell
Haswell memory parameters: L1-icache 32k/8-way/64B
L1-cache 32k/8-WAY/64B,4/5 cycles



L2 cache 256k/8-way/64B, 12 cycles
L3 cache 8MB,64B
(What doesn't this say? replacement policy? inclusive/exclusive? write-back?)
Bzip: 11MB footprint

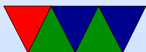
(a) L1-icache = 13k/19B = 0% miss rate
(b) L1-dcache-load = 311M/6.2B = 5% miss rate
(c) L2 = 208M/411M = 50% miss rate
(d) LLC 601k/139M = 0.4% miss rate
(e) Note, l1-dcache is loads. Issue with l1d-stores, in Linux



4.1 (17 Feb 2015) Kleen posted patch to separate SNB evens from HSW in Linux kernel. So your results will change based on kernel version. Annoying. Before there was a l1d-store events

(f) Why do the results not match up? Shouldn't L1-misses be same as L2-accceses? Why would they not match up? Bug in counters, not counting stores, bug in counter selection, other things going on in system, shared resources, chip errata, prefetching, etc. LLC actually uses offcore-response events

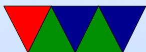
(g) What can we tell about bzip2 behavior? Fits well in



icache. Why is L2 so bad?

4. equake_l on Haswell
   e-quake mem footprint 700MB
   (a) L1-icache = 14M/1.4T = 0%
   (b) L1-dcache = 31B/526B = 5.8%
   (c) L2 = 22B/52B = 42%
   (d) LLC = 8B/13B=58%

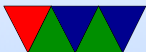
5. bzip2 on Jetson
   Jetson TX-1 4 GB LPDDR4
   L1 i-cache=48 kB, 3-way,



l1 d-cache=32 kB, 2-way

l2 = 2 MB, 16-way (big?) 512 kB (little?)

(a) L1-icache = 184k/10B = 0%
(b) L1-dcache-load = 254M/6B = 4%
(c) L1-dcache-store = 56M/2.2B = 2.5%
(d) L2-dcache-load = 28M/330M = 8.5%
(e) L2-dcache-store = 21M/308M = 6.8%
(f) Why did we have to use raw events? Proper Cortex-A57 event support not added until Linux 4.4. Need to update the kernel, tricky on Jetson.

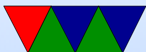


Prefetching

As we saw, Cold misses are very common.

Try to avoid cache misses by bringing values into the cache before they are needed.

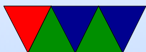
Caches with large blocksize already bring in extra data in advance, but can we do more?



Prefetching Concerns

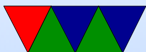
- When?
  We want to bring in data before we need it, but not too early or it wastes space in the cache.

- Where? What part of cache? Dedicated buffer?



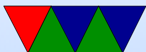
Limits of Prefetching

- May kick data out of cache that is useful
- Costs energy, especially if we do not use the data



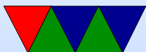
Implementation Issues

- Which cache level to bring into? (register, L1, L2)
- Faulting, what happens if invalid address
- Non-catchable areas (MTRR, PAT).
  Bad to prefetch mem-mapped registers!



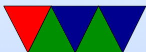
Software Prefetching

- ARM has PLD instruction
- PREFETCHW for write (3dnow, Alpha) cache protocol
- Prefetch, evict next (make it LRU) Alpha
- Prefetch a stream (Altivec)
- Prefetch0, 1, 2 to all cache levels (x86 SSE)
  Prefecthnta, non-temporal



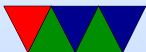
Hardware Prefetching – icache

- Bring in two cache lines
- Branch predictor can provide hints, targets
- Bring in both targets of a branch



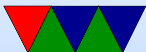
Hardware Prefetching – dcache

- Bring in next line – on miss bring in N and N+1 (or more?)
- Demand – bring in on miss (every other access a miss with linear access)
  Tagged – bring in N+1 on first access to cache line (no misses with linear access)

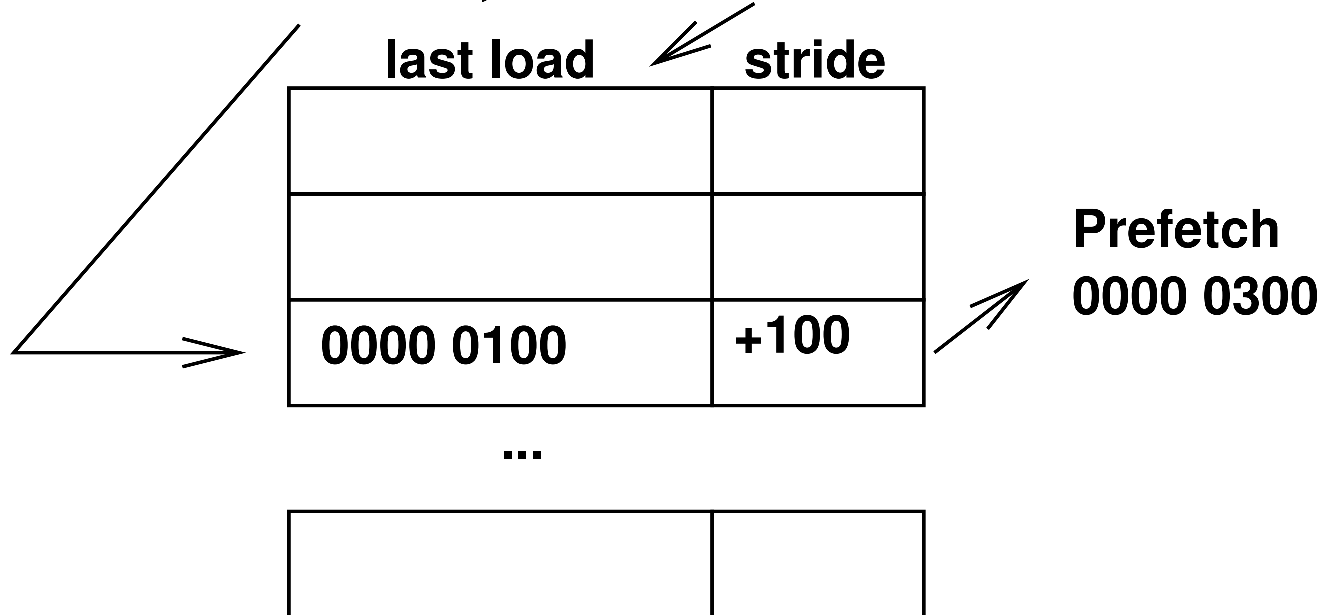


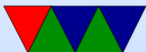
Hardware Prefetching – Stride Prefetching

- Stride predictors – like branch predictor, but with load addresses, keep track of stride
- Separate stream buffer?



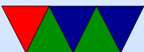
Stride Predictor





Hardware Prefetching – Correlation/Content-Directed Prefetching

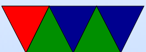
- How to handle things like pointer chasing / linked lists?
- Correlation – records sequence of misses, then when traversing again prefetches in that order
- Content directed – recognize pointers and pre-fetch what they point to



Using 2-bit Counters

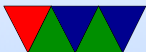
- Use 2-bit counter to see if load causing lots of misses, if so automatically treat as streaming load (Rivers)

- Partitioned cache: cache stack, heap, etc, (or little big huge) separately (Lee and Tyson)



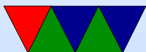
Cortex A9 Prefetch

- PLD – prefetch instruction
  has dedicated instruction unit
- Optional hardware prefetcher. (Disabled on pandaboard)
- Can prefetch 8 data streams, detects ascending and descending with stride of up to 8 cache lines
- Keeps prefetching as long as causing hits
- Stops if: crosses a 4kB page boundary, changes context,



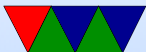
a DSB (barrier) or a PLD instruction executes, or the program does not hit in the prefetched lines.

- PLD requests always take precedence

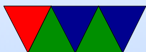


Quick Look at Haswell Prefetch

- https://software.intel.com/en-us/articles/disclosure-of-hw-prefetcher-control-on-some-intel-proce
- 4 prefetches, can independently disable
- L2 hardware prefetcher – fetch data or code into L2
- L2 adjacent cache line prefetcher – bring in 2 cache lines (128B)
- DCU prefetcher – fetch into L1-D cache
- DCU IP prefetcher – use load history to predict what to bring in

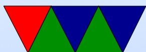


Investigating Prefetching Using Hardware Performance Counters



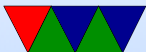
Quick Look at Core2 Prefetch

- Instruction prefetcher
- L1 Data Cache Unit Prefetcher (streaming).
  Ascending data accesses prefetch next line
- L1 Instruction Pointer Strided Prefetcher.
  Looks for strided access from particular load instructions.
  Forward or Backward up to 2k apart
- L2 Data Prefetch Logic.
  Fetches to L2 based on the L1 DCU



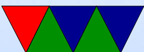
x86 SW Prefetch Instructions (AMD)

- `PREFETCHNTA` – SSE1, non temporal (use once)
- `PREFETCHT0` – SSE1, prefetch to all levels
- `PREFETCHT1` – SSE1, prefetch to L2 + higher
- `PREFETCHT2` – SSE1, prefetch to L3 + higher
- `PREFETCH` – AMD 3DNOW! prefetch to L1
- `PREFETCHW` – AMD 3DNOW! prefetch for write



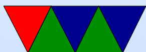
Core2

- `SSE_PRE_EXEC:NTA` – counts NTA
- `SSE_PRE_EXEC:L1` – counts T0 (`fxsave`+2, `fxrstor`+5)
- `SSE_PRE_EXEC:L2` – counts T1/T2
- Problem: Only 2 counters available on Core2



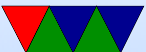
AMD (Istanbul and Later)

- PREFETCH_INSTRUCTIONS_DISPATCHED:NTA
- PREFETCH_INSTRUCTIONS_DISPATCHED:LOAD
- PREFETCH_INSTRUCTIONS_DISPATCHED:STORE
- These events appear to be speculative, and won't count SW prefetches that conflict with HW prefetches



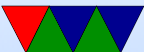
Atom

- PREFETCH:PREFETCHNTA
- PREFETCH:PREFETCHT0
- PREFETCH:SW_L2
- These events will count SW prefetches, but numbers counted vary in complex ways



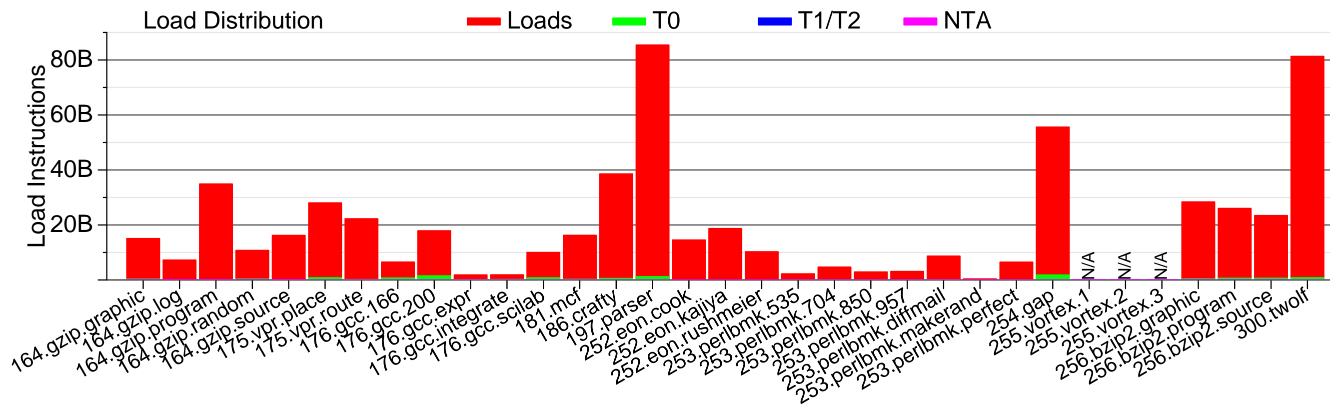
Does anyone use SW Prefetch?

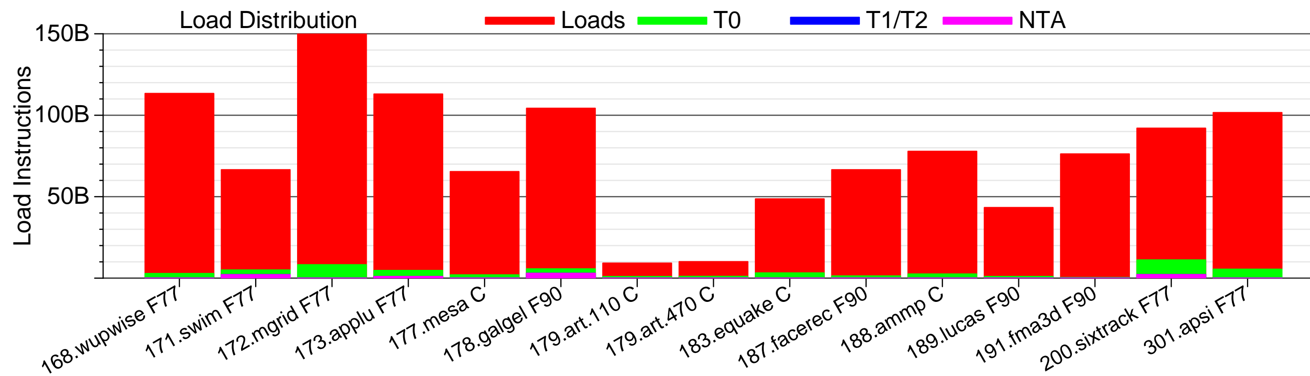
- gcc by default disables SW prefetch unless you specify `-fprefetch-loop-arrays`
- icc disables unless you specify `-xsse4.2 -op-prefetch=4`
- glibc has hand-coded SW prefetch in `memcpy()`
- Prefetch can hurt behavior:
  - Can throw out good cache lines,
  - Can bring lines in too soon,
  - Can interfere with the HW prefetcher

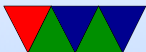


SW Prefetch Distribution

SPEC CPU 2000, Core2, gcc `-fprefetch-loop-arrays`

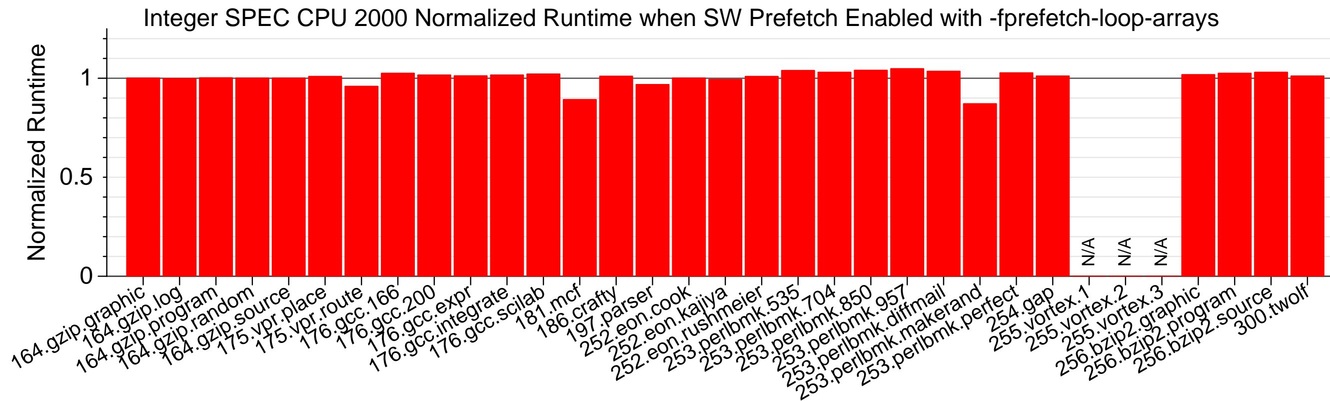


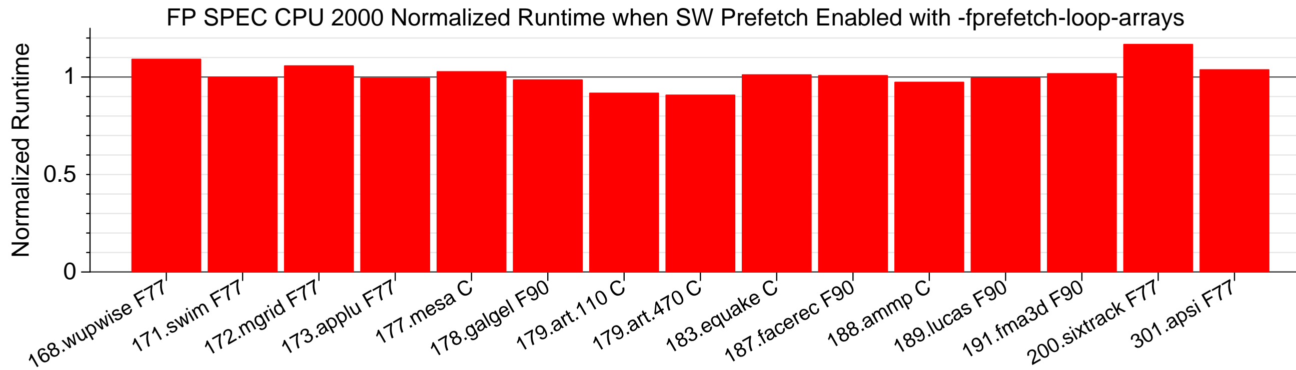


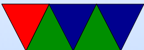


Normalized SW Prefetch Runtime

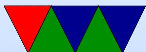
on Core2 (Smaller is Better)





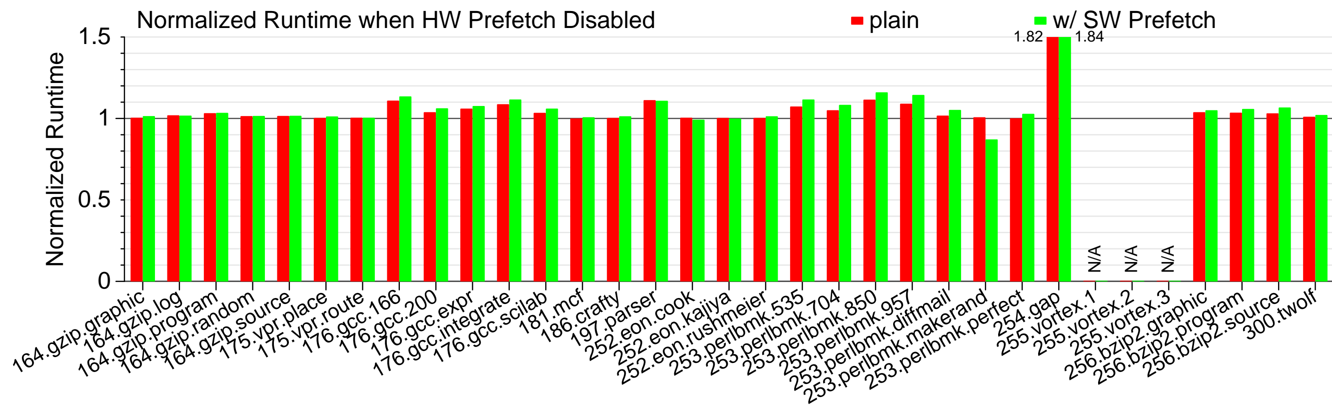


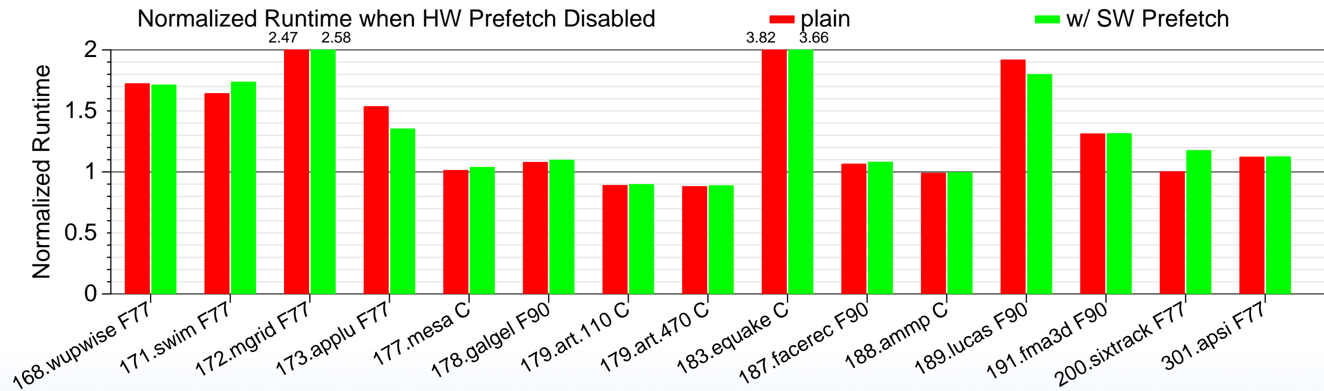
The HW Prefetcher on Core2 can be Disabled

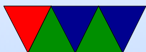


Runtime with HW Prefetcher Disabled

Normalized against Runtime with HW Prefetcher Enabled
on Core2 (Smaller is Better)

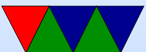






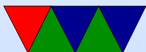
PAPI_PRF_SW Revisited

- Can multiple machines count SW Prefetches?
  Yes.
- Does the behavior of the events match expectations?
  Not always.
- Would people use the preset?
  Maybe.

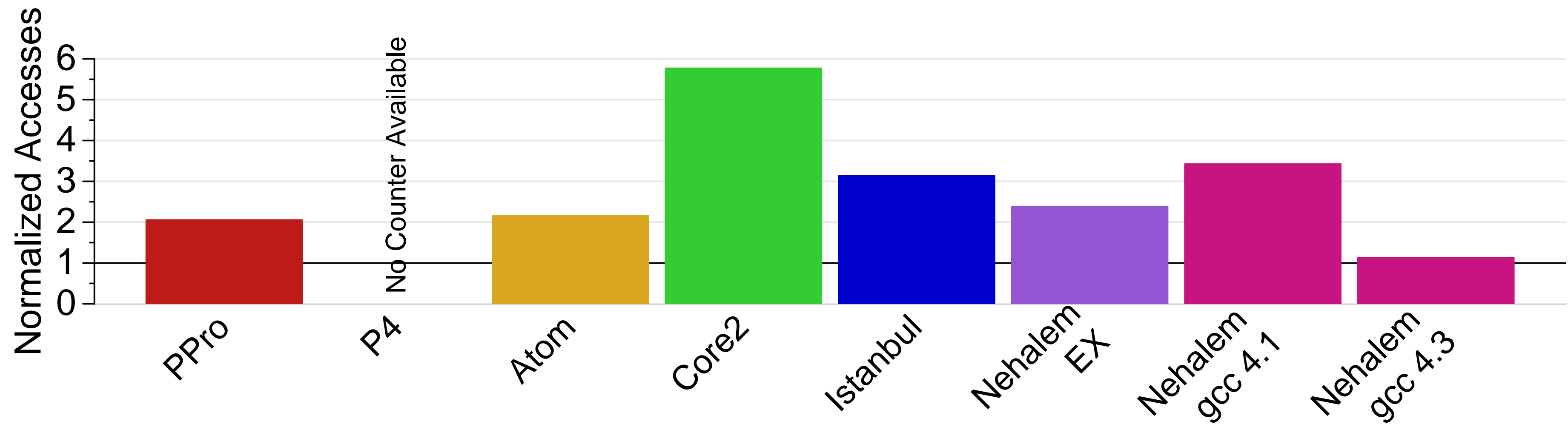


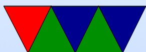
L1 Data Cache Accesses

```
float array[1000],sum = 0.0;

PAPI_start_counters(events,1);

for(int i=0; i<1000; i++) {
    sum += array[i];
}

PAPI_stop_counters(counts,1);
```



PAPI_L1_DCA





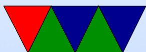
PAPI_L1_DCA

Expected Code

```
* 4020d8:       f3 0f 58 00             addss  (%rax),%xmm0
  4020dc:       48 83 c0 04             add    $0x4,%rax
  4020e0:       48 39 d0                cmp    %rdx,%rax
  4020e3:       75 f3                   jne    4020d8 <main+0x328>
```

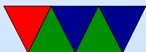
Unexpected Code

```
* 401e18:       f3 0f 10 44 24 0c       movss  0xc(%rsp),%xmm0
* 401e1e:       f3 0f 58 04 82          addss  (%rdx,%rax,4),%xmm0
  401e23:       48 83 c0 01             add    $0x1,%rax
  401e27:       48 3d e8 03 00 00       cmp    $0x3e8,%rax
* 401e2d:       f3 0f 11 44 24 0c       movss  %xmm0,0xc(%rsp)
  401e33:       75 e3                   jne    401e18 <main+0x398>
```

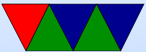


L1 Data Cache Misses

- Allocate array as big as L1 DCache
- Walk through the array byte-by-byte
- Count misses with `PAPI_L1_DCM` event
- If 32B line size, if linear walk through memory, first time will have 1/32 miss rate or 3.125%. Second time through (if fit in cache) should be 0%.



PAPI_L1_DCM – Forward/Reverse/Random

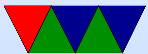


Normalized Misses

1.25
1.00
0.75
0.50
0.25
0.00

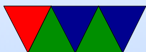
No Counter Available

PPro
P4
Atom
Core2
Core2 NoPrefetch
Istanbul
Nehalem
Nehalem EX



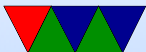
L1D Sources of Divergences

- Hardware Prefetching
- PAPI Measurement Noise
- Operating System Activity
- Non-LRU Cache Replacement



L2 Total Cache Misses

- Allocate array as big as L2 Cache
- Walk through the array byte-by-byte
- Count misses with `PAPI_L2_TCM` event



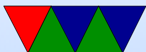
PAPI_L2_TCM – Forward/Reverse/Random

Normalized L2 Misses

1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00

No Counter Available

PPro
P4
Atom
Core2
Core2 NoPrefetch
Istanbul
Nehalem
Nehalem EX



L2 Sources of Divergences

- Hardware Prefetching
- PAPI Measurement Noise
- Operating System Activity
- Non-LRU Cache Replacement
- Cache Coherency Traffic