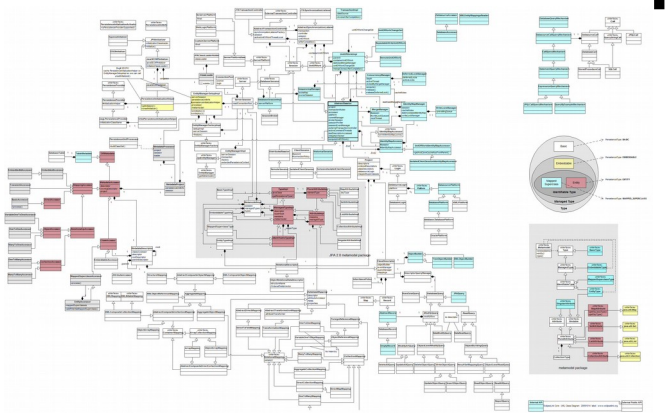
Mental models for modern program tuning

Andi Kleen

Intel Corporation

Jun 2016

How can we see program performance?



vs

High level

Army of ants

Important to get the common ants fast

“Preliminary optimization is the root of all evil”

- Donald Knuth, 1974, “Structured programing with gotos”

“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%”**

Flat profiles

- Is it only for 3%?
  - Often not true for complex workloads with flat performance profile

```
Samples: 152K of event 'cycles', Event count (approx.): 96197390806
Overhead  Command          Shared Object          Symbol
   3.01%  cc1              libc-2.19.so           [.] _int_malloc
   1.25%  cc1              libc-2.19.so           [.] _int_free
   0.93%  cc1              libc-2.19.so           [.] malloc_consolidate
   0.69%  cc1              cc1                    [.] 0x00000000007e8de9
   0.61%  cc1              libc-2.19.so           [.] malloc
   0.54%  sh               libc-2.19.so           [.] __mbrtowc
   0.54%  cc1              [kernel.vmlinux]       [k] page_fault
   0.54%  cc1              [kernel.vmlinux]       [k] clear_page_c_e
   0.53%  sh               libc-2.19.so           [.] __gconv_transform_utf8_internal
   0.43%  cc1              libc-2.19.so           [.] free
   0.39%  as               ld-2.19.so             [.] _dl_relocate_object
   0.39%  cc1              libc-2.19.so           [.] __memcmp_sse4_1
   0.38%  as               [kernel.vmlinux]       [k] copy_page
   0.36%  as               [kernel.vmlinux]       [k] page_fault
   0.36%  make             make                   [.] parse_variable_definition
   0.35%  cc1              cc1                    [.] 0x00000000007e8e60
   0.32%  cc1              libc-2.19.so           [.] memset
   0.31%  sh               libc-2.19.so           [.] _int_malloc
   0.31%  make             libc-2.19.so           [.] _int_malloc
   0.31%  cc1              libc-2.19.so           [.] getenv
   0.28%  sh               ld-2.19.so             [.] do_lookup_x
   0.28%  sh               [kernel.vmlinux]       [k] page_fault
   0.27%  cc1              cc1                    [.] 0x00000000007e8e0a
   0.26%  cc1              cc1                    [.] 0x0000000000417abe
   0.26%  as               [kernel.vmlinux]       [k] clear_page_c_e
   0.25%  fixdep           fixdep                 [.] parse_dep_file
```

Strategy

- Have to be aware of performance when writing code
  - At least for critical paths
  - Otherwise it needs too many changes later to get fast
  - Also need a model to understand results

Mental model

vs

vs

Latency numbers every programer should know

L1 cache reference	0.5ns	
Branch mispredict	5 ns	
L2 cache reference	7 ns	14x L1 cache
Mutex lock/unlock	25 ns	
Main memory reference	100 ns	20x L2 cache, 200x L1
Compress 1K bytes with snappy	3000 ns	
Send 1K bytes over 1Gbps network	10,000 ns	0.01 ms
Read 4K randomly from SSD	150,000 ns	0.15 ms
Read 1MB sequentially from memory	250,000 ns	0.25 ms
Round trip within same data center	500,000 ns	0.5 ms
Read 1MB sequentially from SSD	1,000,000 ns	1ms 4x memory
Disk seek	10,000,000 ns	10 ms 20x data center r-t
Read 1MB sequentially from disk	20,000,000 ns	20 ms 20x SSD
Send packet CA->Netherlands->CA	150,000,000 ns	150 ms

Originally from Peter Norvig. Generic numbers. Does not represent a particular part.

Critical bottleneck

CPU bound

Could be bound on other things:

IO

Network

GPU

For finding bottlenecks in other resources see Brendan Gregg's talks

What is CPU bound?

- Computation

- Accessing memory
- Communicating with other cores

What versus where

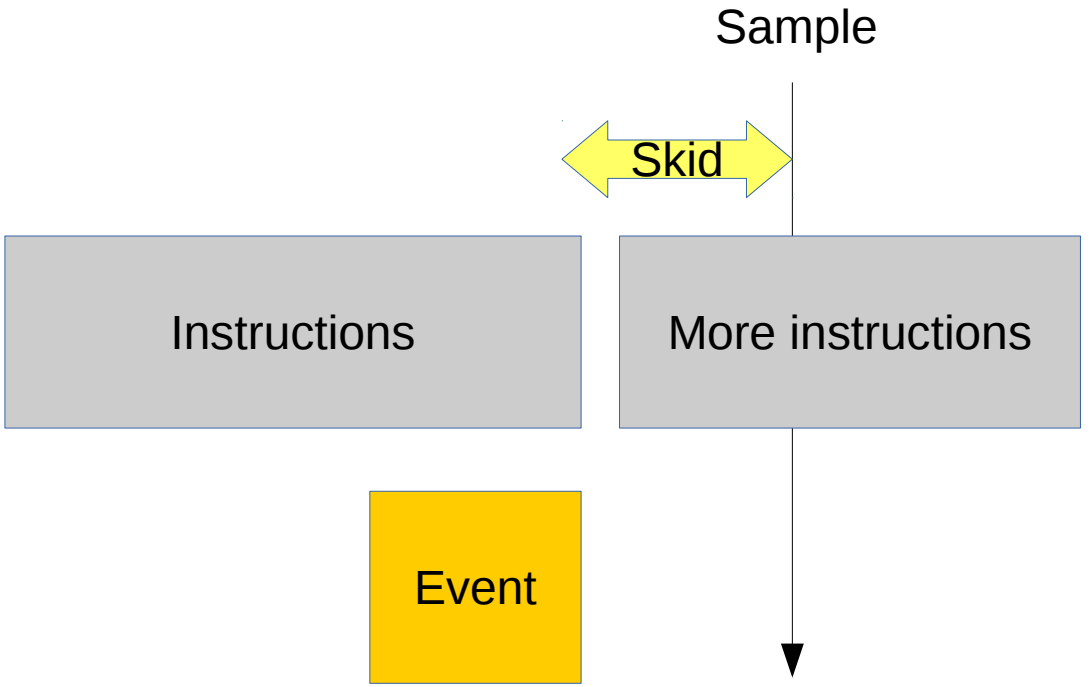
- Counting
  - What accurately but not where
  - Look at ratios
- Sampling
  - Losely where, but not always what
  - Need more than just cycles
- Tracing
  - Where very accurately, but lots of data
  - Not necessarily what

Sampling skid

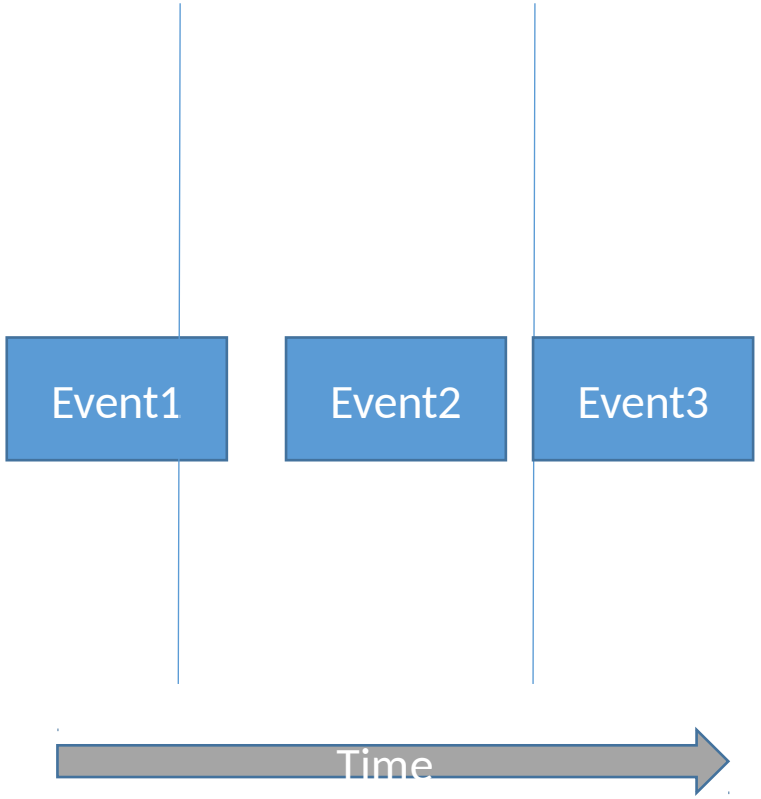
- Sampling may not find the exact place
- Use Precise Event Based Sampling (:pp) to minimize skid
  - Does not work in VMs

perf record -e cycles:pp ..

More PEBS events available, see ocperf list output



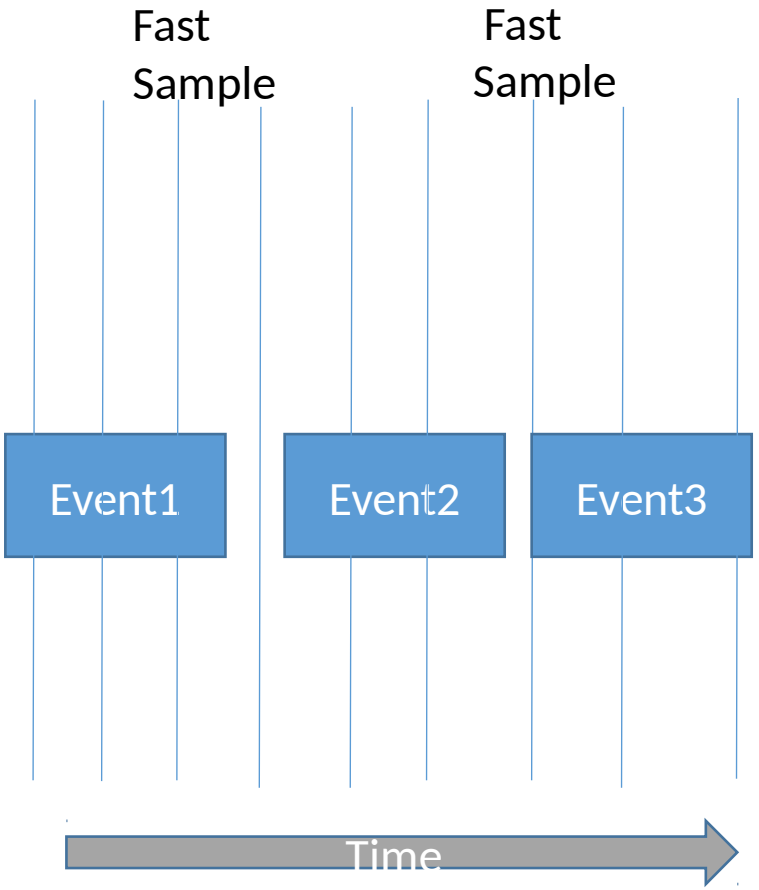
Fast sampling versus slow sampling



Slow sampling does not accurately measure fast events

PEBS Interrupt less sampling allows fast sampling at reasonable overhead

Feature in recent Linux perf 4.1+
perf record -c 25000 –no-time …



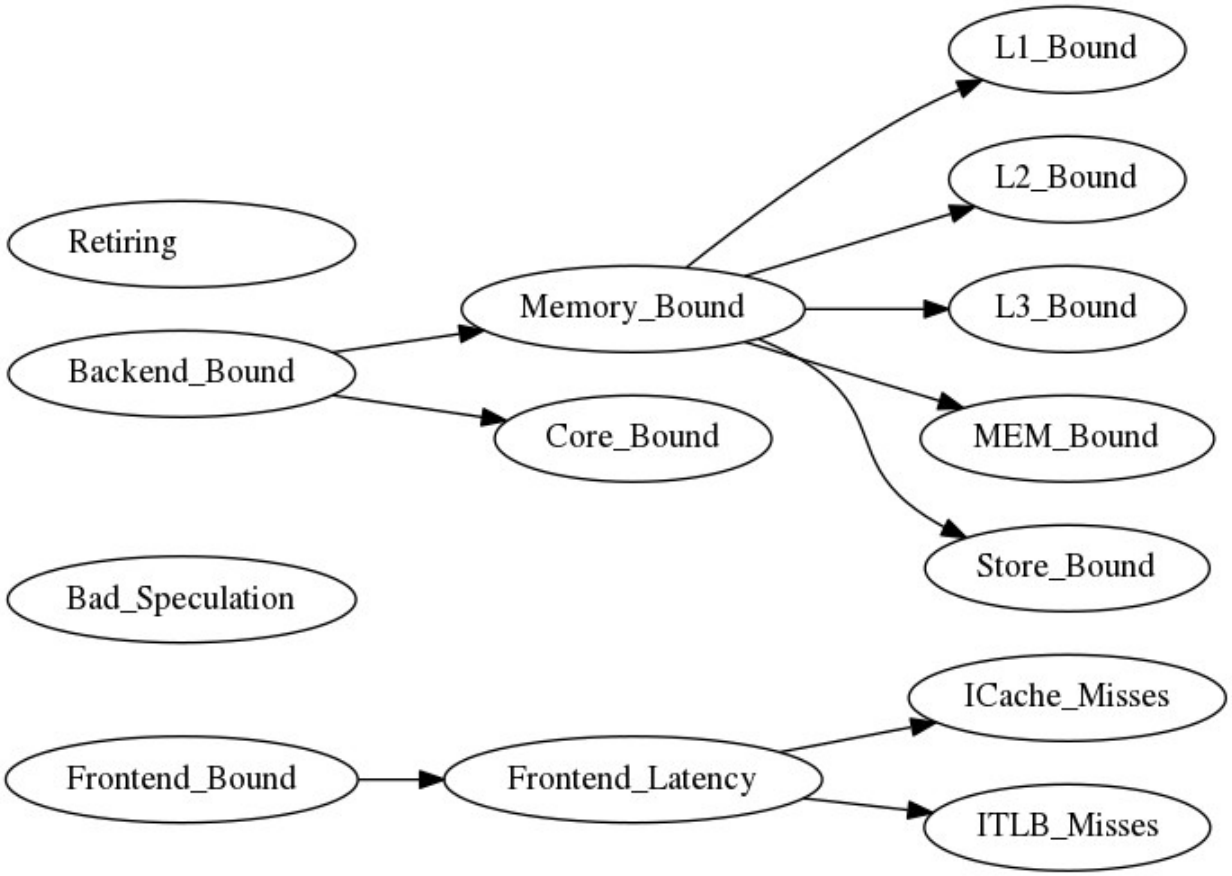
Ratios can mislead

```
% perf stat -e cache-references,cache-misses ./fib
fib 100000 = 13120482006059434805

 Performance counter stats for './fib':

            23,283      cache-references
             8,483      cache-misses              #   36.434 % of all cache refs

       0.000535555 seconds time elapsed
```

But Fibonacci is not using memory much

Use TopDown instead to find real bottleneck

TopDown methology

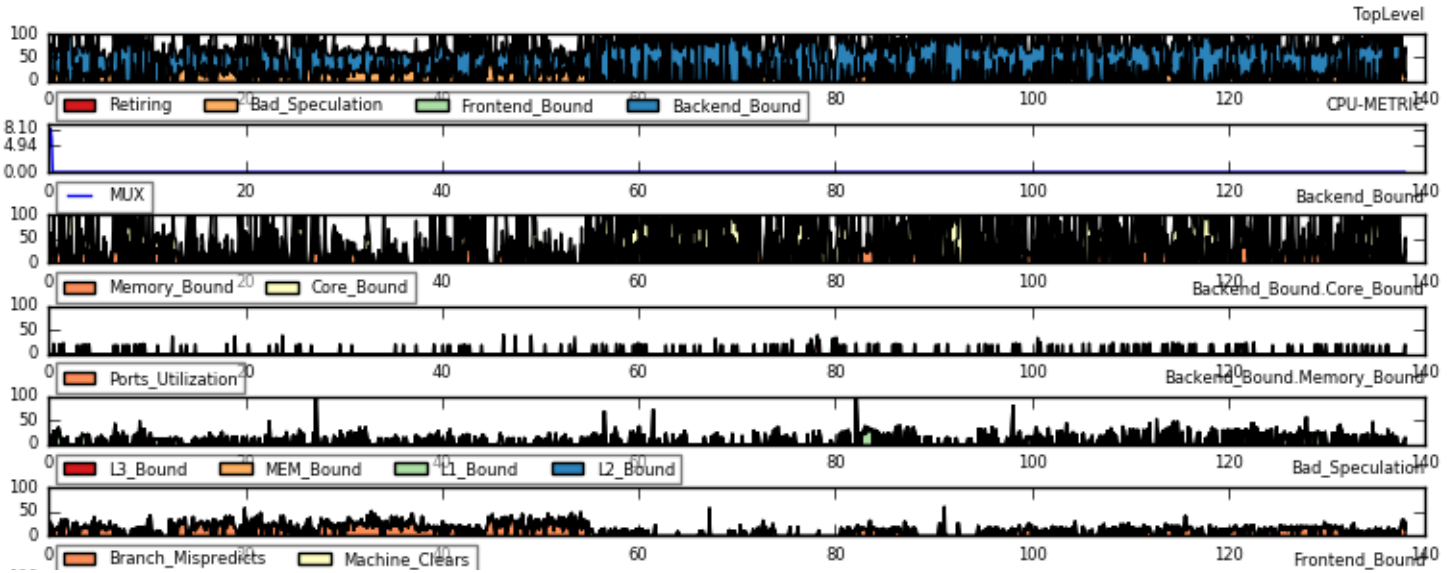


Only showing cache nodes

Use performance counter measurement to find bottleneck in CPU pipeline

TopDown output

```
C0    BE      Backend_Bound:                                  98.97 %           [100.00%]
C0    BE/Mem  Backend_Bound.Memory_Bound:                     96.09 %           [100.00%]
C0-T0 BE/Mem  Backend_Bound.Memory_Bound.MEM_Bound:           96.07 %           [100.00%]
        This metric estimates how often the CPU was stalled on
        accesses to external memory (DRAM) by loads...
        Sampling events:  mem_load_uops_retired.l3_miss:pp
C0-T0 BOTTLENECK Backend_Bound.Memory_Bound.MEM_Bound 96.07%
C0-T1 BE/Mem  Backend_Bound.Memory_Bound.L1_Bound:            18.54 %           [100.00%]
        This metric estimates how often the CPU was stalled without
        loads missing the L1 data cache...
        Sampling events:  mem_load_uops_retired.l1_hit:pp mem_load_uops_retired.hit_lfb:pp
C0-T1 BOTTLENECK Backend_Bound.Memory_Bound.L1_Bound 18.54%
```



toplev tool in http://github.com/andikleen/pmu-tools

Basic performance unit

- Less than that is not cheaper

CPU cache lines	64 bytes
Pages	4K / 2MB (1GB)
Network IO packets	Hundreds of bytes
Network IO connection	Thousands of bytes
Block IO	Several MB

Simplified cache model

- 64byte cache lines
- Local
  - Temporal
  - Predictible
  - Cold
- Communication
  - Shared read-only
  - Bouncing

Temporal

- What we recently used
- What fits in caches of our working set

- Best performance if working set fits

Cold

- Not temporal, large or thrashing
- Can be quite slow: several orders of magnitude
  - Some optimizations possible: NUMA locality or interleaving using libnuma
- When data is not reused can use non temporal loads/stores to avoid polluting caches
  - But only for very large data sets

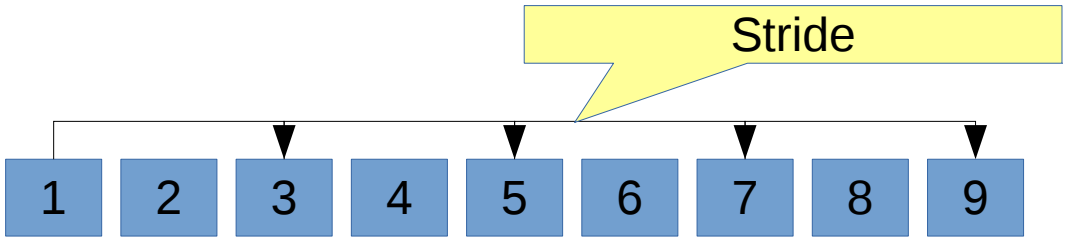
Predictible

- Large, not hot
- But predictible access pattern
- Cache can prefetch in background and hide latency

- Arrays are usually good, pointers can be bad

Prefetchers 101

- Only work for larger amounts of data
  - Need training
- Support strides and forward/backward stream
- Multiple streams supported, but only small number



Kernel software prefetching

- Event loop

  offset += pread(shared_fd, buffer, size, offset)

  vs

- read(private_fd, buffer, size);

Hot loop model

- We understand everything about the hot loop
- Can use cache blocking and other tricks to optimize temporal locality for everything
  - Compiler may even be able to do automatically

- Unfortunately a lot of software is not like that

Library model

- Called by (random) other code which has own cache foot print
- Tradeoff: more computation or more caching/table lookup
  - Tragedy of the commons: if all together use too much everyone suffers
- Non composable problem
  - Everything depends on everything

Longer essay on the topic: http://halobates.de/blog/p/227

Cache friendly libraries

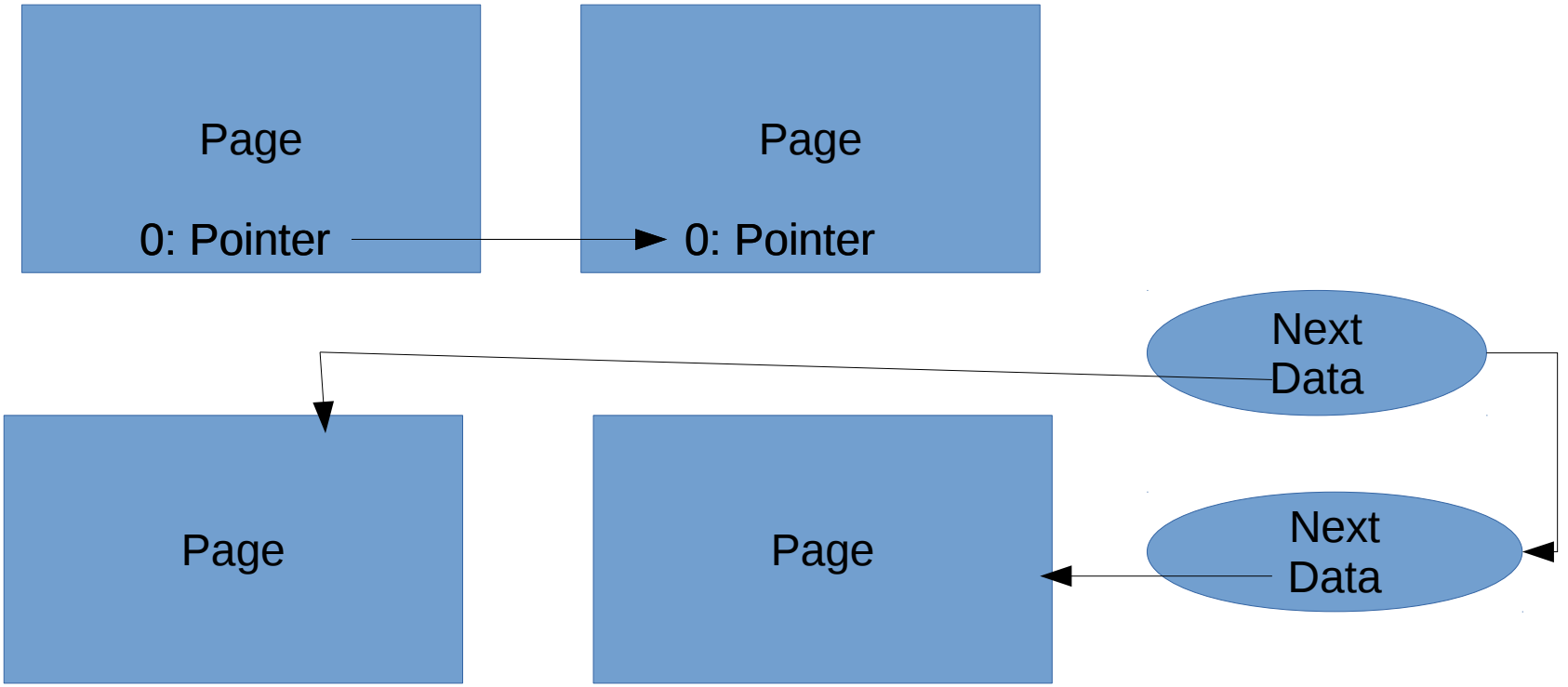
- May need to optimize for cache cold
  - Minimize foot print
  - But only if memory is not cheaper
- Cluster hot fields in data structures together.
- Support big and small versions
  - Add knobs for foot print
  - Could use automatic tuning

Cache coloring

- Caches can store addresses at specific multiplies of 64 only in limited number of positions

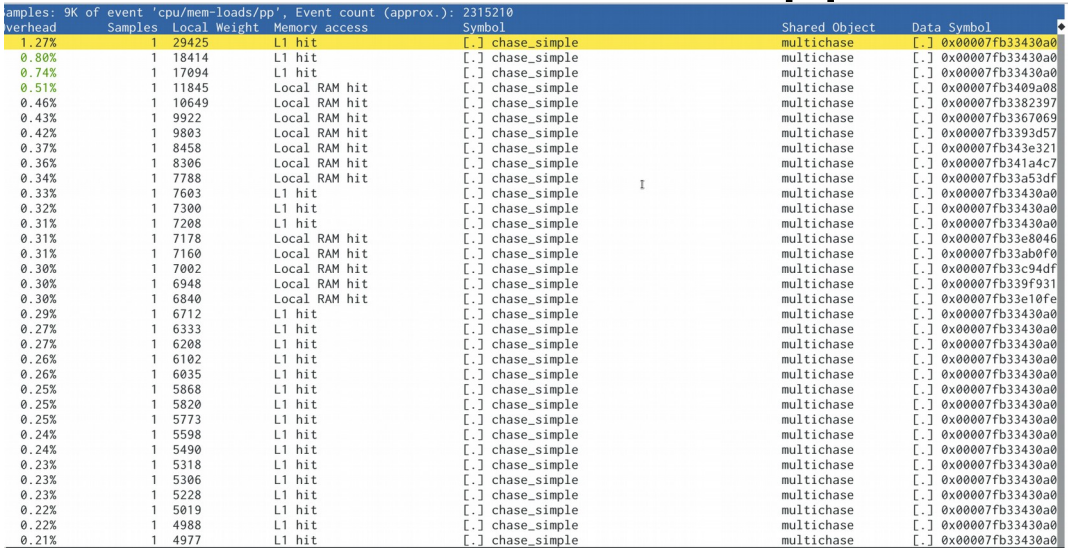
Inefficient use of cache

- Metadata always at same offset in data page
  - Only uses fraction of the cache for pointers
  - Can use separate packed metadata instead



How to find cache issues

- First check TopDown Memory Bound
- Count cache misses
  - perf stat -e cache-references,cache-misses ...
- Sample L3 cache misses:
  - ocperf.py record -e \
    mem_load_uops_l3_miss_retired.local_dram:pp ...
- Sample addresses
  - perf mem record …



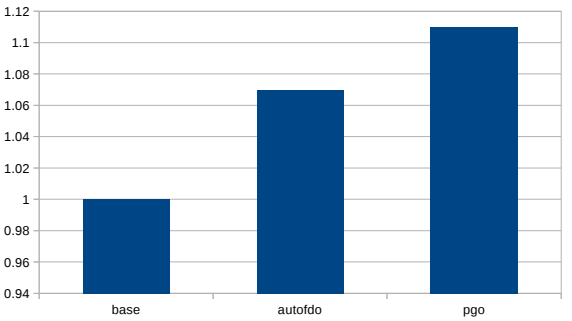
```
amples: 9K of event 'cpu/mem-loads/pp', Event count (approx.): 2315210
verhead       Samples  Local Weight  Memory access               Symbol                  Shared Object     Data Symbol
  1.27%             1         29425  L1 hit                      [.] chase_simple        multichase        [.] 0x00007fb33430a0
  0.80%             1         18414  L1 hit                      [.] chase_simple        multichase        [.] 0x00007fb33430a0
  0.74%             1         17094  L1 hit                      [.] chase_simple        multichase        [.] 0x00007fb33430a0
  0.51%             1         11845  Local RAM hit               [.] chase_simple        multichase        [.] 0x00007fb3409a08
  0.46%             1         10649  Local RAM hit               [.] chase_simple        multichase        [.] 0x00007fb3382397
  0.43%             1          9922  Local RAM hit               [.] chase_simple        multichase        [.] 0x00007fb3367069
  0.42%             1          9803  Local RAM hit               [.] chase_simple        multichase        [.] 0x00007fb3393d57
  0.37%             1          8458  Local RAM hit               [.] chase_simple        multichase        [.] 0x00007fb343e321
  0.36%             1          8306  Local RAM hit               [.] chase_simple        multichase        [.] 0x00007fb341a4c7
  0.34%             1          7788  Local RAM hit               [.] chase_simple        multichase        [.] 0x00007fb33a53df
  0.33%             1          7603  L1 hit                      [.] chase_simple        multichase        [.] 0x00007fb33430a0
  0.32%             1          7300  L1 hit                      [.] chase_simple        multichase        [.] 0x00007fb33430a0
  0.31%             1          7208  L1 hit                      [.] chase_simple        multichase        [.] 0x00007fb33430a0
  0.31%             1          7178  Local RAM hit               [.] chase_simple        multichase        [.] 0x00007fb33e8046
  0.31%             1          7160  Local RAM hit               [.] chase_simple        multichase        [.] 0x00007fb33ab0f0
  0.30%             1          7002  Local RAM hit               [.] chase_simple        multichase        [.] 0x00007fb33c94df
  0.30%             1          6948  Local RAM hit               [.] chase_simple        multichase        [.] 0x00007fb339f931
  0.30%             1          6840  Local RAM hit               [.] chase_simple        multichase        [.] 0x00007fb33e10fe
  0.29%             1          6712  L1 hit                      [.] chase_simple        multichase        [.] 0x00007fb33430a0
  0.27%             1          6333  L1 hit                      [.] chase_simple        multichase        [.] 0x00007fb33430a0
  0.27%             1          6208  L1 hit                      [.] chase_simple        multichase        [.] 0x00007fb33430a0
  0.26%             1          6102  L1 hit                      [.] chase_simple        multichase        [.] 0x00007fb33430a0
  0.26%             1          6035  L1 hit                      [.] chase_simple        multichase        [.] 0x00007fb33430a0
  0.25%             1          5868  L1 hit                      [.] chase_simple        multichase        [.] 0x00007fb33430a0
  0.25%             1          5820  L1 hit                      [.] chase_simple        multichase        [.] 0x00007fb33430a0
  0.25%             1          5773  L1 hit                      [.] chase_simple        multichase        [.] 0x00007fb33430a0
  0.24%             1          5598  L1 hit                      [.] chase_simple        multichase        [.] 0x00007fb33430a0
  0.24%             1          5490  L1 hit                      [.] chase_simple        multichase        [.] 0x00007fb33430a0
  0.23%             1          5318  L1 hit                      [.] chase_simple        multichase        [.] 0x00007fb33430a0
  0.23%             1          5306  L1 hit                      [.] chase_simple        multichase        [.] 0x00007fb33430a0
  0.23%             1          5228  L1 hit                      [.] chase_simple        multichase        [.] 0x00007fb33430a0
  0.22%             1          5019  L1 hit                      [.] chase_simple        multichase        [.] 0x00007fb33430a0
  0.22%             1          4988  L1 hit                      [.] chase_simple        multichase        [.] 0x00007fb33430a0
  0.21%             1          4977  L1 hit                      [.] chase_simple        multichase        [.] 0x00007fb33430a0
```

http://github.com/andikleen/pmu-tools
for ocperf

Automated icache tuning

- Some cache problems (like instruction cache) can be automatically optimized
  - Indicated by FrontendBound in TopDown
  - Split hot and cold code using compiler profile feedback
  - Can be done automatically from profiles without instrumentation

http://github.com/google/autofdo with gcc 5.x+



gcc 5 compile performance with profile feedback

Noisy neighbour problem

- Caches can be shared resources
- Other processes/threads/VMs cause slowdowns

- Co-locate processes with different characteristics
  - For example low IPC and high IPC
- New hardware capabilities:
  - Cache QoS monitoring
  - Cache QoS enforcement

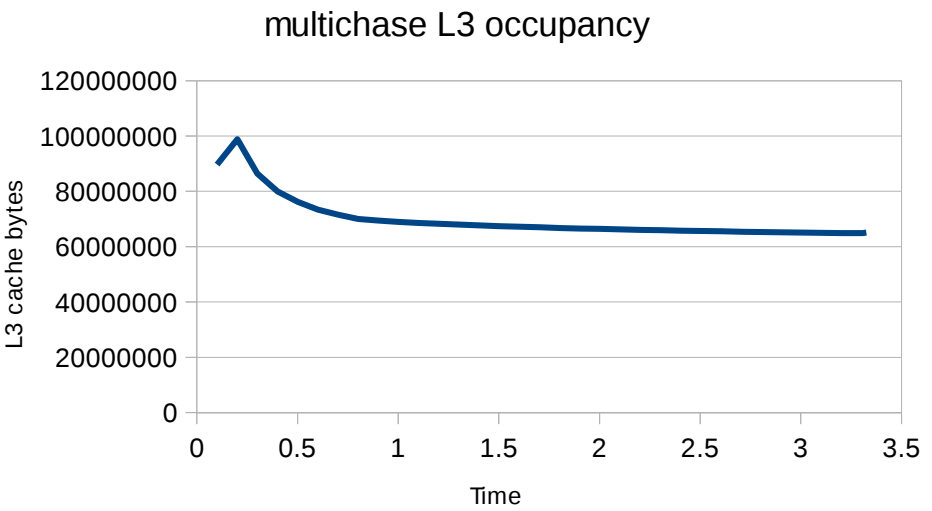
L3 cache occupancy measurements

```
% perf stat -I 100 -e intel_cqm/llc_occupancy/ ./multichase
#           time             counts  unit events
     0.100182765        75202560.00 Bytes intel_cqm/llc_occupancy/
     0.200402773        95453184.00 Bytes intel_cqm/llc_occupancy/
     0.300596047       102170624.00 Bytes intel_cqm/llc_occupancy/
     0.400770355        92749824.00 Bytes intel_cqm/llc_occupancy/
     0.500946912        87038361.60 Bytes intel_cqm/llc_occupancy/
     0.601122900        83247104.00 Bytes intel_cqm/llc_occupancy/
     0.701300209        80567149.71 Bytes intel_cqm/llc_occupancy/
     0.801500316        78557184.00 Bytes intel_cqm/llc_occupancy/
     0.901680815        76982954.67 Bytes intel_cqm/llc_occupancy/
     1.001860767        75733401.60 Bytes intel_cqm/llc_occupancy/
     1.102038456        74657419.64 Bytes intel_cqm/llc_occupancy/
     1.202216477        73760768.00 Bytes intel_cqm/llc_occupancy/
     1.302390360        73002062.77 Bytes intel_cqm/llc_occupancy/
     1.402594390        72351744.00 Bytes intel_cqm/llc_occupancy/
     1.502793635        71768473.60 Bytes intel_cqm/llc_occupancy/
     1.602970612        71258112.00 Bytes intel_cqm/llc_occupancy/
     1.703142929        70825140.71 Bytes intel_cqm/llc_occupancy/
     1.803316869        70440277.33 Bytes intel_cqm/llc_occupancy/
     1.903494395        70085578.11 Bytes intel_cqm/llc_occupancy/
     2.003675296        69766348.80 Bytes intel_cqm/llc_occupancy/
     2.103856300        69477522.29 Bytes intel_cqm/llc_occupancy/
     2.204032964        69210484.36 Bytes intel_cqm/llc_occupancy/
     2.304206774        68962393.04 Bytes intel_cqm/llc_occupancy/
     2.404380434        68751360.00 Bytes intel_cqm/llc_occupancy/
     2.504553208        68545413.12 Bytes intel_cqm/llc_occupancy/
#           time             counts  unit events
     2.604745396        68355308.31 Bytes intel_cqm/llc_occupancy/
     2.704926256        68190208.00 Bytes intel_cqm/llc_occupancy/
     2.805102509        68029878.86 Bytes intel_cqm/llc_occupancy/
     2.905275510        67883996.69 Bytes intel_cqm/llc_occupancy/
     3.005453370        67747840.00 Bytes intel_cqm/llc_occupancy/
     3.105627000        67620467.61 Bytes intel_cqm/llc_occupancy/
     3.205800309        67497984.00 Bytes intel_cqm/llc_occupancy/
     3.305979966        67382923.64 Bytes intel_cqm/llc_occupancy/
 106.2
```

Measure L3 cache occupancy for a process/container/VM

Gives us the upper consumption and avoids co-placement problems

Requires Intel Xeon E5 v3 and Linux 4.1+



Communication

- Communication happens when a core accesses a cache line written by another core
- Like message passing in a fast network

Finding Communication

- Check for TopDown Contested_Accesses

- Find with:

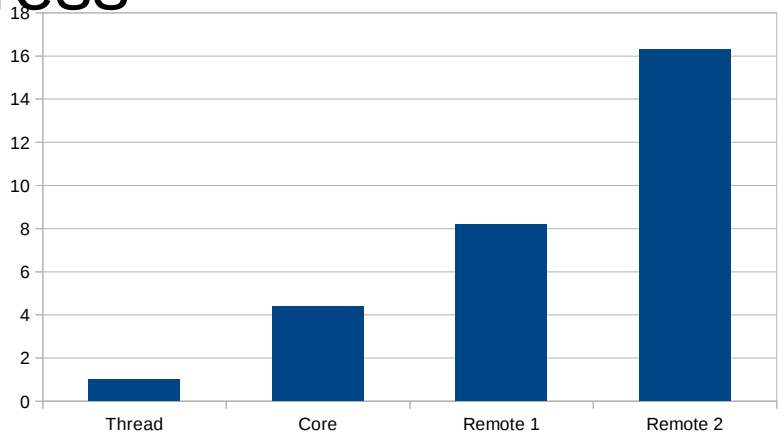
  ocperf.py stat -e \
  mem_load_uops_l3_hit_retired.xsnp_hitm ...

  ocperf.py record -g -e \
  mem_load_uops_l3_hit_retired.xsnp_hitm:pp ...

  - Or similar event name

Latency levels

- SMT thread<->other thread
  - Really fast
- Core<->other Core in socket
- Other socket
  - Latency depends on home address
  - Possibly multiple hops



Cache2Cache Ratios

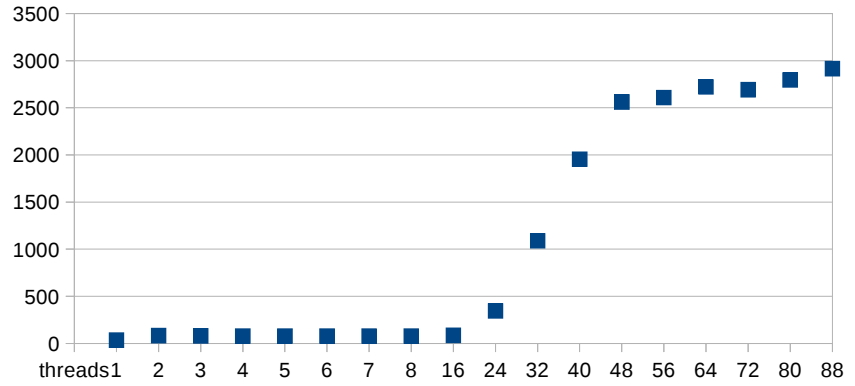
Tool to find latencies:
Intel Latency Checker

http://www.intel.com/software/mlc

Cache queueing effects

- Cache communication is message passing
  - Has ordering requirements
- Can cause conflicts and queue delays
  - Gets worse with more and more cores

Time to do 250k increments on shared counter



Cache messaging optimizations

- Avoid unnecessary round trips
- Avoid false sharing
- Prefer nearby thread/core/socket
- Design for load, use backoffs
- Avoid thundering herds

Example

- global_flag = true;
- vs
- if (!global_flag)
  - global_flag = true;

Classic Lock optimization

- Start with big lock per subsystem
- Push down to fine grained data objects to lower contention
- End goal: lock per cache line?

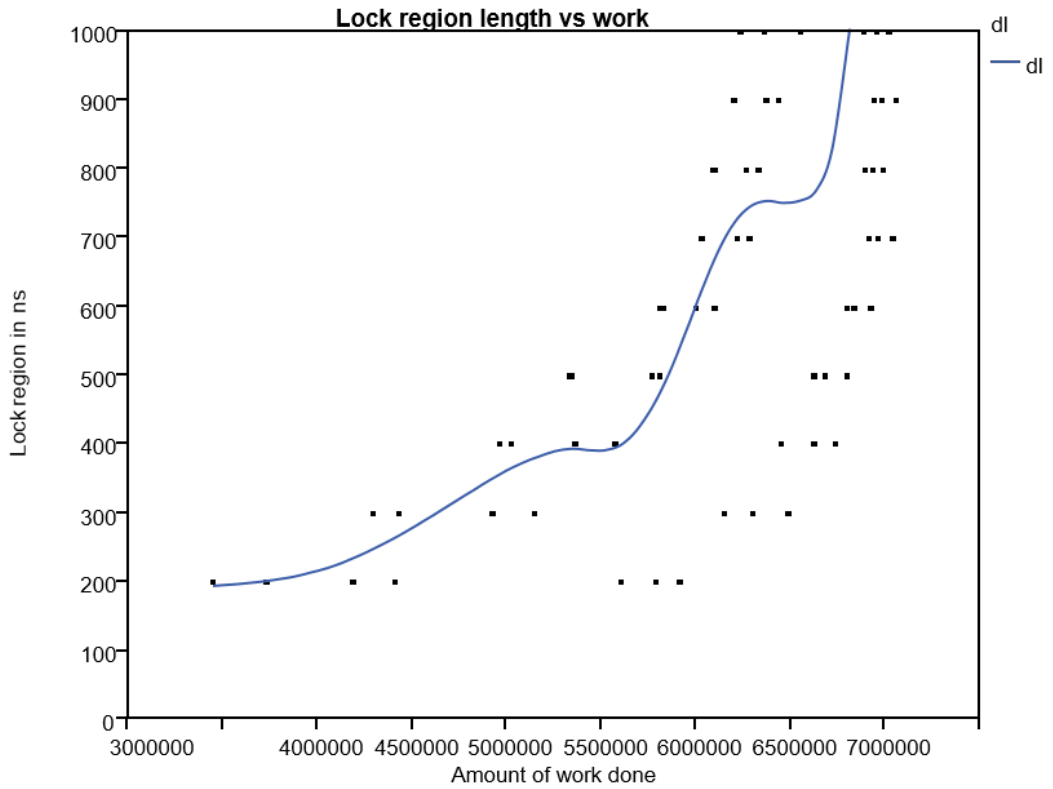
Lock overhead

- Locks have overhead even when cache line is not contended
  - Can depend on nearby operations (sometimes slow)
  - Can be 3 orders of magnitude delta even when not contended
- Too many locks can be costly
  - “Locks as red tape”

Another problem with small locks:

```
/*
 * Lock ordering in mm:
 *
 * inode->i_mutex       (while writing or truncating, not reading or faulting)
 *   mm->mmap_sem
 *     page->flags PG_locked (lock_page)
 *       hugetlbfs_i_mmap_rwsem_key (in huge_pmd_share)
 *         mapping->i_mmap_rwsem
 *           anon_vma->rwsem
 *             mm->page_table_lock or pte_lock
 *               zone->lru_lock (in mark_page_accessed, isolate_lru_page)
 *               swap_lock (in swap_duplicate, swap_info_get)
 *                 mmlist_lock (in mmput, drain_mmlist and others)
 *                 mapping->private_lock (in __set_page_dirty_buffers)
 *                   mem_cgroup_{begin,end}_page_stat (memcg->move_lock)
 *                     mapping->tree_lock (widely used)
 *                 inode->i_lock (in set_page_dirty's __mark_inode_dirty)
 *                 bdi.wb->list_lock (in set_page_dirty's __mark_inode_dirty)
 *                   sb_lock (within inode_lock in fs/fs-writeback.c)
 *                   mapping->tree_lock (widely used, in set_page_dirty,
 *                             in arch-dependent flush_dcache_mmap_lock,
 *                             within bdi.wb->list_lock in __sync_single_inode)
 *
 * anon_vma->rwsem,mapping->i_mutex      (memory_failure, collect_procs_anon)
 *   ->tasklist_lock
 *     pte map lock
 */
```

Lock region size versus work



Lock acquisition needs to amortize communication

Requires doing enough work inside lock

Lock stability

- Short lock regions can be instable
  - Small timing variations can cause big performance changes
  - Due to timing and queueing effects on the locks and data
- Rule of thumb: critical section at least 200us for stable behavior
  - That is a lot of instructions!

In depth paper:
http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/xeon-lock-scaling-analysis-paper.pdf

Library design for lock batching

- Design interfaces to process multiple operations in the same call
  - Allows doing more work in lock regions
  - Improves temporal locality

Lock elision

- Can use hardware transactional memory support with TSX to do lock elision
- Use coarse lock and let the hardware figure it out
- Lock cache line stays shared
  - Only uses real lock on real conflict
  - Still need to minimize common data conflicts

Available on Intel Xeon v4 or Xeon E7 v3

http://www.intel.com/software/tsx

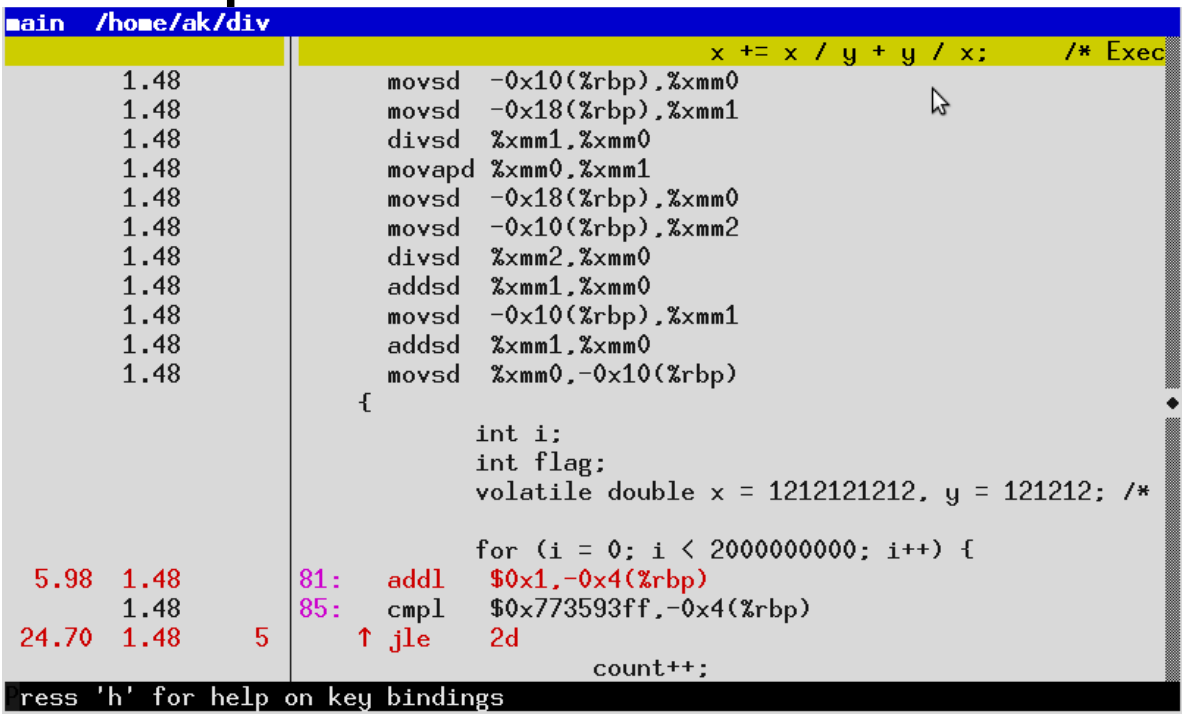
Measure: perf stat -T ...
Find aborts: perf record -e tx-aborts:pp ….

Microbenchmarks are difficult

- CPU Frequency can change
- Accurate timing is tricky
- Compilers are good at optimizing them away
- Caching effects are difficult to reproduce
  - For example calling library with always the same input always hits cache
  - Calling always with different input thrashes caches
  - Both is unrealistic

Automatic micro benchmarking

- Using timed last branch records in Skylake CPUs
- Sample real workload and get cycle count of 32 program blocks per sample



$ perf record -b …
$ perf report

Needs Linux perf 4.3+

Indepth articles:
http://lwn.net/Articles/680985/

http://lwn.net/Articles/680996

Tracing

- Using Processor Trace (PT)
  - Hardware feature inBroadwell/Skylake
    - Supported in Linux perf since Linux 4.1
  - Fine grained execution trace with time stamps

```
$ perf record -e intel_pt// …
$ perf script --ns --itrace=cr \
-F time,event,callindent,addr,sym
```

```
120005.948724863:   branches:                  =>          401440 [unknown]
120005.948724863:   branches:                      =>     7f3fd401dc00 _dl_fixup
120005.948724863:   branches:                          =>     7f3fd4019550 _dl_lookup_symbol_x
120005.948724863:   branches:                              =>     7f3fd4018be0 do_lookup_x
120005.948724863:   branches:                                  =>     7f3fd401f910 _dl_name_match_p
120005.948724863:   branches:                                      =>     7f3fd4027c70 strcmp
120005.948724863:     return:                                  =>     7f3fd401f925 _dl_name_match_p
120005.948724863:   branches:                                      =>     7f3fd4027c70 strcmp
120005.948724863:     return:                                  =>     7f3fd401f954 _dl_name_match_p
120005.948724863:     return:                              =>     7f3fd4018d57 do_lookup_x
120005.948724863:   branches:                                  =>     7f3fd4018a60 check_match.9478
120005.948724863:   branches:                                      =>     7f3fd4027c70 strcmp
120005.948724863:     return:                                  =>     7f3fd4018ab9 check_match.9478
120005.948724863:   branches:                                      =>     7f3fd4027c70 strcmp
120005.948724863:     return:                                  =>     7f3fd4018b8c check_match.9478
120005.948724863:     return:                              =>     7f3fd40194fe do_lookup_x
120005.948724863:     return:                          =>     7f3fd40196ad _dl_lookup_symbol_x
120005.948724863:     return:                      =>     7f3fd401dcf7 _dl_fixup
120005.948724863:     return:                  =>     7f3fd40241e5 _dl_runtime_resolve
120005.948724863:   branches:                      =>     7f3fd3c9fa40 __run_exit_handlers
120005.948724863:   branches:                          =>     7f3fd3ca0090 __call_tls_dtors
120005.948724863:   branches:                              => ffffffff815d9500 page_fault
:
```

For older kernels
http://github.com/andikleen/simple-pt

Assembler Tracing

- Using Processor Trace (PT)
- Timing cycle accurate to ~4 basic blocks on Skylake

```
$ perf record -e intel_pt// …
$ perf script --ns --itrace=i0ns \
-F time,pid,comm,ip,asm
```

Requires patched perf for disassembler

Indepth article on PT:
http://lwn.net/Articles/648154/

```
true  8718 326329.269963689:        3a0a400c93      call _dl_start
true  8718 326329.269963689:        3a0a4048b0      push %rbp
true  8718 326329.269963689:        3a0a4048b1      mov %rsp, %rbp
true  8718 326329.269963689:        3a0a4048b4      push %r15
true  8718 326329.269963689:        3a0a4048b6      push %r14
true  8718 326329.269963689:        3a0a4048b8      push %r13
true  8718 326329.269963689:        3a0a4048ba      push %r12
true  8718 326329.269963689:        3a0a4048bc      mov %rdi, %r12
true  8718 326329.269963689:        3a0a4048bf      push %rbx
true  8718 326329.269963689:        3a0a4048c0      sub $0x38, %rsp
true  8718 326329.269963689:        3a0a4048c4      rdtsc
true  8718 326329.269963689:        3a0a4048c6      shl $0x20, %rdx
true  8718 326329.269963689:        3a0a4048ca      mov %eax, %eax
true  8718 326329.269963689:        3a0a4048cc      or %rdx, %rax
true  8718 326329.269963689:        3a0a4048cf      lea 0x21c53a(%rip), %rdx
true  8718 326329.269965355:        3a0a4048d6      mov %rax, 0x21c363(%rip)
true  8718 326329.269965355:        3a0a4048dd      mov 0x21c52c(%rip), %rax
true  8718 326329.269965355:        3a0a4048e4      mov %rdx, %r14
true  8718 326329.269965355:        3a0a4048e7      sub 0x21c6b2(%rip), %r14
true  8718 326329.269965355:        3a0a4048ee      mov %rdx, 0x21d0b3(%rip)
true  8718 326329.269965355:        3a0a4048f5      test %rax, %rax
true  8718 326329.269965355:        3a0a4048f8      mov %r14, 0x21d099(%rip)
true  8718 326329.269965355:        3a0a4048ff      jz _dl_start+226
true  8718 326329.269965355:        3a0a404905      lea 0x21c6f4(%rip), %rcx
true  8718 326329.269965355:        3a0a40490c      mov $0x3800003d8, %r9
true  8718 326329.269965355:        3a0a404916      mov $0x37ffffb78, %r8
true  8718 326329.269965355:        3a0a404920      mov $0x6fffffff, %esi
true  8718 326329.269965355:        3a0a404925      mov $0x6ffffdff, %r11d
true  8718 326329.269965355:        3a0a40492b      mov $0x6ffffeff, %ebx
true  8718 326329.269965355:        3a0a404930      add %rcx, %r9
true  8718 326329.269965355:        3a0a404933      add %rcx, %r8
true  8718 326329.269965355:        3a0a404936      mov $0x31, %r10d
true  8718 326329.269965355:        3a0a40493c      mov $0x70000021, %edi
true  8718 326329.269965355:        3a0a404941      jmp _dl_start+175
true  8718 326329.269965355:        3a0a40495f      cmp $0x21, %rax
```

Summary

- Focus on critical bottlenecks
- Remember the order of magnitudes
- Cache communication is message passing
- Lock coarsely
- Measure properly

- http://github.com/andikleen/pmu-tools
- http://halobates.de

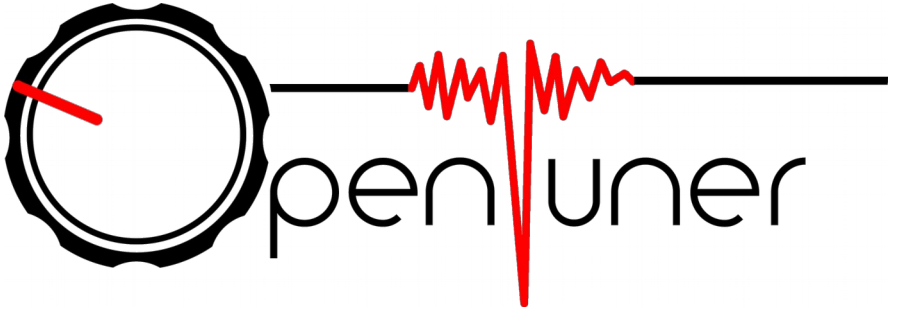
Backup

Autotuning

- Add knobs to size tables and algorithms and use an auto tuner to find best trade off for whole program
  - Can adapt to changing circumstances
  - Use generic optimization frameworks

ATLAS: Automatically tuned linear algebra kernels

http://github.com/jansel/opentuner



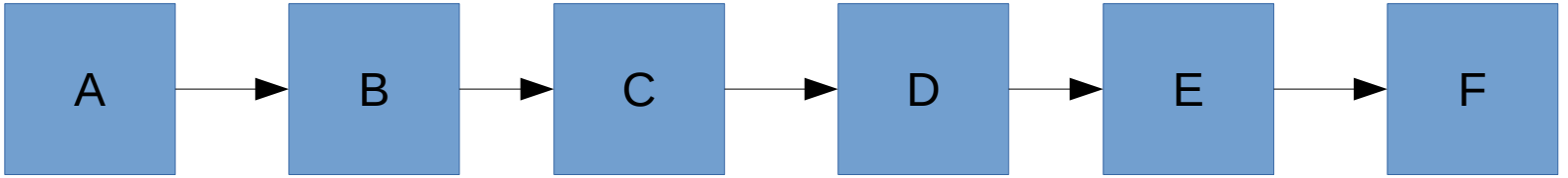
Cache events

- Communication:
  - mem_load_uops_l3_hit_retired.xsnp_hit
  - mem_load_uops_l3_hit_retired.xsnp_hitm
  - mem_load_uops_l3_hit_retired.xsnp_miss

- Locality:
  - mem_load_uops_retired.l1_miss / hit
  - mem_load_uops_retired.l2_miss / hit
  - mem_load_uops_retired.l3_miss / hit

- Can be counted or sampled with ocperf in pmu-tools

Linked lists versus ropes



VS