Data-Oriented Design

A pinch of UNIX wisdom – on handling complexity

Rule 1 *You can't tell where a program is going to spend its time.* Bottlenecks occur in surprising places, so don't try to second guess and put in a speed hack until you've proven that's where the bottleneck is.

Rule 2 *Measure.* Don't tune for speed until you've measured, and even then don't unless one part of the code overwhelms the rest.

Rule 3 *Fancy algorithms are slow when **n** is small, and **n** is usually small.* Fancy algorithms have big constants. Until you know that **n** is frequently going to be big, don't get fancy. (Even if **n** does get big, use Rule 2 first.)

Rule 4 *Fancy algorithms are buggier than simple ones, and they're much harder to implement.* Use simple algorithms as well as simple data structures.

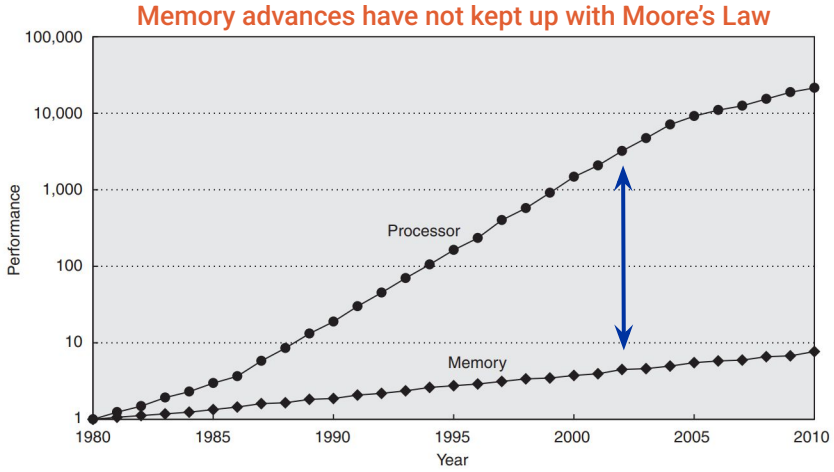
Rule 5 ***Data dominates.*** **If you've chosen the right data structures and organized things well, the algorithms will almost always be self-evident.** ***Data structures, not algorithms, are central to programming.***

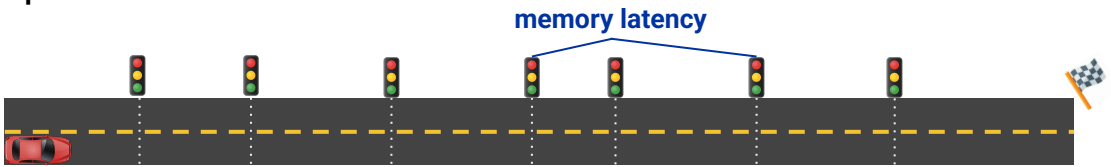
Rule 6 There is no Rule 6.

from "Notes on C Programming", by Rob Pike

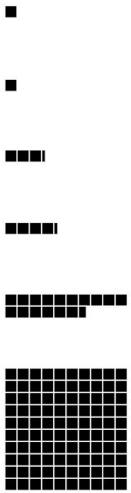
CPU vs Memory Performance Gap

- CPU performance has grown much faster than memory performance
- Memory access latency is often the main performance issue
- L3 cache has grown significantly, but L1 and L2 remain relatively small
- L2 cache misses are still expensive
- Ultimately, speed of light limits peak performance (c ≈ 30cm / ns)





Latency Numbers Every Programmer Should Know



1ns (~5 CPU cycles)

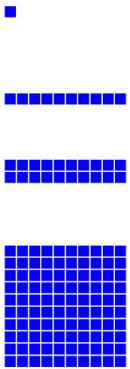
L1 cache reference: 1ns

Branch mispredict: 3ns

L2 cache reference: 4ns

Mutex lock/unlock: 17ns

100ns = ■



Main memory reference: 100ns

1,000ns ≈ 1µs

Compress 1KB wth Zippy: 2,000ns ≈ 2µs

10,000ns ≈ 10µs = ■

Send 2,000 bytes over commodity network: 44ns

SSD random read: 16,000ns ≈ 16µs

Read 1,000,000 bytes sequentially from memory: 3,000ns ≈ 3µs

Round trip in same datacenter: 500,000ns ≈ 500µs

1,000,000ns = 1ms = ■

Read 1,000,000 bytes sequentially from SSD: 49,000ns ≈ 49µs

Disk seek: 2,000,000ns ≈ 2ms

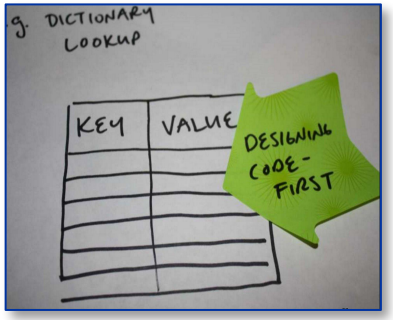
Read 1,000,000 bytes sequentially from disk: 825,000ns ≈ 825µs

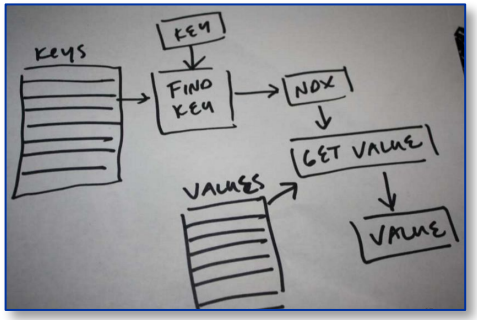
Packet roundtrip CA to Netherlands: 150,000,000ns ≈ 150ms

Source: https://colin-scott.github.io/personal_website/research/interactive_latency.html

What is Data-Oriented Design?

- CppCon 2014 talk by Mike Acton
- Reminder of first principles on how data structures influence final performance
- A program is something that transforms input data from one form to another
- If you don't understand the data, you don't understand the problem
- Different data ⇒ different problem
- Where there's one, there are many, optimize for the many
- You need to understand the hardware your software is running on





Object-Oriented vs Data-Oriented Design

Object-Oriented

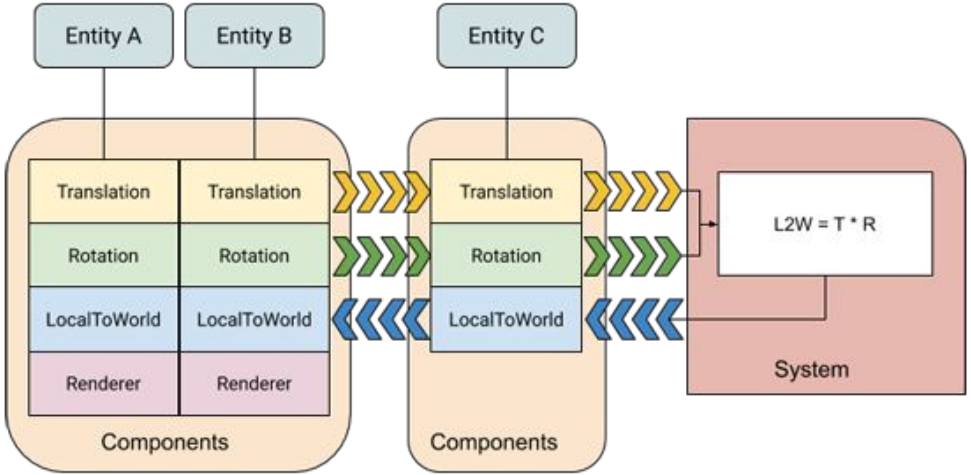
- Polymorphism, abstract interfaces
  - Defer implementation to concrete types
- Classes, Inheritance, Encapsulation
  - Data available only via exposed interface
- Data and operations/behavior together
  - Extend data and behavior
  - Reuse code from parent classes
  - Less chance to optimize data layout
  - Higher demand on instruction cache
    - Many methods per object type

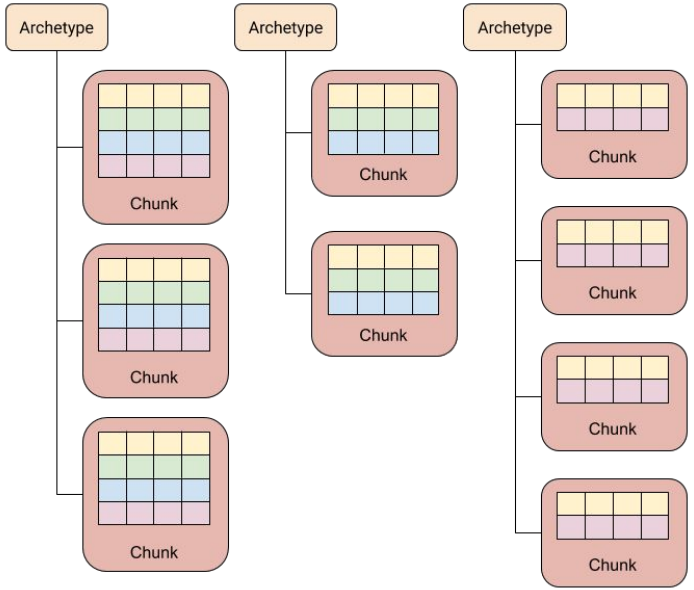
Data-Oriented

- Optimize memory access pattern
  - Make use of full cachelines
  - Use all available memory bandwidth
  - Spatial and temporal cache locality
- Data structures separate from code
  - Improves also front-end metrics
  - Move all movable objects together
  - Single function rather than methods
  - Commonly implemented as an Entity-Component-System in games

Entity Component System Concepts

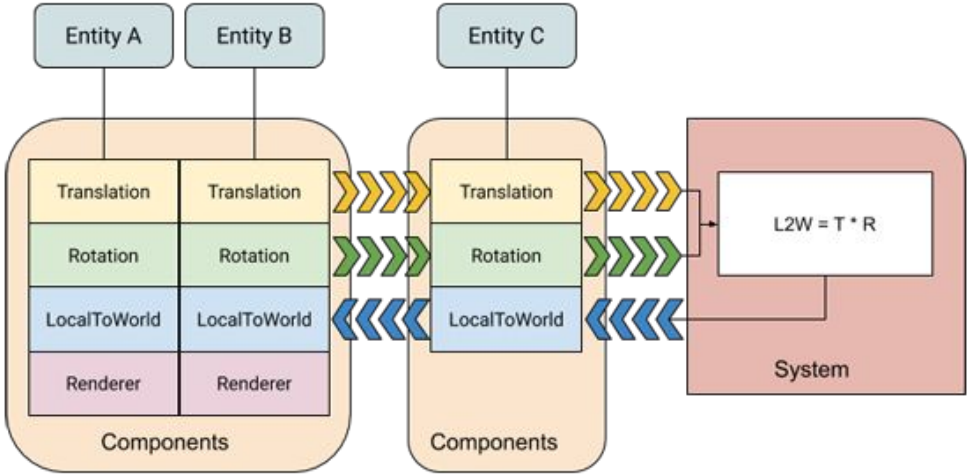
What they do it in computer games



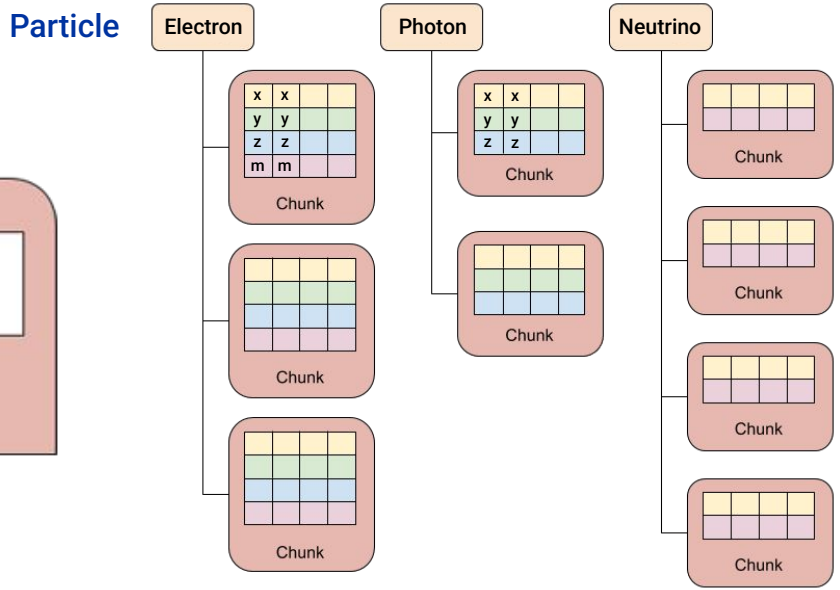


Entity Component System Concepts

What they do it in computer games



…could also be applied to detector simulation



Array of Structure vs Structure of Array Data Layouts

AoS Structure

```
struct track {
       /* event structure */
       int32_t id;
       int32_t parent;

       /* geometry data */
       float x;
       float y;
       float z;

       int32_t geometry_id;

       /* physics data */
       float vx;
       float vy;
       float vz;
       float E;
        ...
};
```

SoA Structure

```
template<unsigned int N>
struct TrackBlock {
  static constexpr unsigned int nElem = N;

  /* event structure */
  std::array<int32_t, nElem> id;
  std::array<int32_t, nElem> parent;

  /* geometry data */
  std::array<float, nElem> x;
  std::array<float, nElem> y;
  std::array<float, nElem> z;

  static constexpr mass = 0.511f;
  ...
};
```

How does the kernel look like?

Source Listing 1

```
template <int tracks_per_page>
inline void kinetic_energy_soa(track* tracks)
{
  auto vx = get_component(tracks, vx, tracks_per_page);
  auto vy = get_component(tracks, vy, tracks_per_page);
  auto vz = get_component(tracks, vz, tracks_per_page);
  auto  E = get_component(tracks,  E, tracks_per_page);

  for (int i = 0; i < tracks_per_page; ++i)
   E[i] = kinetic_energy(track::mass,vx[i],vy[i],vz[i]);
}



float kinetic_energy(float mass, float vx,
                     float vy,   float vz)
{
      return 0.5 * mass * (vx*vx + vy*vy + vz*vz);
}
```

Source Listing 2

```
void update_kinetic_energy_cpu_soa(struct slab* s)
{
  #pragma omp parallel for
  for (int i = 0; i < slab::blocks_per_slab; ++i)
    for (int j = 0; j < slab::pages_per_block; ++j)
    kinetic_energy_soa<slab::tracks_per_page>(
      s->tracks[i][j]);
}



struct slab {
static constexpr int tracks_per_page =  32;
static constexpr int pages_per_block = 512;
static constexpr int blocks_per_slab = 512;

struct track
tracks[blocks_per_slab][pages_per_block][tracks_per_page];
};
```

Performance numbers for simple kinetic energy kernel

AoS Data Layout

```
Performance counter stats for 'AoS' (average of 20 runs):

          483.46 msec task-clock                #    0.999 CPUs utilized
               1      context-switches          #    0.001 K/sec
               0      cpu-migrations            #    0.000 K/sec
            1138      page-faults               #    0.002 M/sec
      1687395641      cycles                    #    3.490 GHz
       106037669      stalled-cycles-frontend   #    6.28% frontend cycles idle
      1283753418      stalled-cycles-backend    #   76.08% backend cycles idle
      1996927212      instructions              #    1.18  insn per cycle
                                                #    0.64  stalled cycles per insn
        19497066      branches                  #   40.328 M/sec
           40196      branch-misses             #    0.21% of all branches
       622021303      L1-dcache-loads           # 1286.609 M/sec
       123124358      L1-dcache-load-misses     #   19.79% of all L1-dcache accesses
        42493039      L1-icache-loads           #   87.894 M/sec
          149328      L1-icache-load-misses     #    0.35% of all L1-icache accesses
           31003      dTLB-loads                #    0.064 M/sec
            8598      dTLB-load-misses          #   27.73% of all dTLB cache accesses
              26      iTLB-loads                #    0.055 K/sec
              19      iTLB-load-misses          #   72.30% of all iTLB cache accesses

       0.484160 +- 0.000367 seconds time elapsed
```

SoA Data Layout

```
Performance counter stats for 'SoA' (average of 20 runs):

          118.78 msec task-clock                #    0.994 CPUs utilized
               1      context-switches          #    0.010 K/sec
               0      cpu-migrations            #    0.000 K/sec
            1138      page-faults               #    0.010 M/sec
       414521761      cycles                    #    3.490 GHz
       131647110      stalled-cycles-frontend   #   31.76% frontend cycles idle
       182606710      stalled-cycles-backend    #   44.05% backend cycles idle
       122945544      instructions              #    0.30  insn per cycle
                                                #    1.49  stalled cycles per insn
        17639259      branches                  #  148.508 M/sec
           38941      branch-misses             #    0.22% of all branches
        81264204      L1-dcache-loads           #  684.179 M/sec
        42555034      L1-dcache-load-misses     #   52.37% of all L1-dcache accesses
         8799361      L1-icache-loads           #   74.083 M/sec
           38984      L1-icache-load-misses     #    0.44% of all L1-icache accesses
           13098      dTLB-loads                #    0.110 M/sec
            6723      dTLB-load-misses          #   51.33% of all dTLB cache accesses
               5      iTLB-loads                #    0.040 K/sec
             112      iTLB-load-misses          # 2363.16% of all iTLB cache accesses

       0.119446 +- 0.000115 seconds time elapsed
```

About 4x faster for SoA. Much lower number of cycles, L1 loads, and TLB loads, easily auto-vectorized.

perf – memory access analysis: loads and stores

```
PERF-MEM(1)                                perf Manual                                PERF-MEM(1)

NAME
       perf-mem - Profile memory accesses

SYNOPSIS
       perf mem [<options>] (record [<command>] | report)

DESCRIPTION
       "perf mem record" runs a command and gathers memory operation data from it, into perf.data. Perf record options are
       accepted and are passed through.

       "perf mem report" displays the result. It invokes perf report with the right set of options to display a memory
       access profile. By default, loads and stores are sampled. Use the -t option to limit to loads or stores.

       Note that on Intel systems the memory latency reported is the use-latency, not the pure load (or store latency). Use
       latency includes any pipeline queueing delays in addition to the memory subsystem latency.

       On Arm64 this uses SPE to sample load and store operations, therefore hardware and kernel support is required. See
       perf-arm-spe(1) for a setup guide. Due to the statistical nature of SPE sampling, not every memory operation will be
       sampled.

OPTIONS
 Manual page perf-mem(1) line 1 (press h for help or q to quit)
```

perf – memory access analysis: ROOT RDataFrame

```
bash df102_NanoAODDimuonAnalysis $ perf mem record -t load -F 5000 df102_NanoAODDimuonAnalysis 8 Run2012B_DoubleMuParked.root Ru
n2012C_DoubleMuParked.root
Couldn't synthesize cgroup events.
[ perf record: Woken up 130 times to write data ]
[ perf record: Captured and wrote 33.940 MB perf.data (550719 samples) ]
bash df102_NanoAODDimuonAnalysis $ perf mem report -s mem -q --stdio 2>/dev/null
    38.64%        250297  L1 or L1 hit
    20.24%        121901  L3 or L3 hit
    18.59%        106253  LFB or LFB hit
    15.63%         50825  Remote Remote Cache (1 hop) or L3 hit
     4.22%          9863  L3 miss
     1.45%          6289  Local RAM or RAM hit
     0.99%          3978  Remote Remote RAM (1 hop) or RAM hit
     0.24%          1310  L2 or L2 hit
     0.00%             2  Uncached or N/A hit
     0.00%             1  I/O or N/A hit


bash df102_NanoAODDimuonAnalysis $ _
```

Line Fill Buffer (sits between L1 and L2)

Red flag, too many remote cache accesses

See also: https://community.intel.com/t5/Intel-Moderncode-for-Parallel/What-is-the-aim-of-the-line-fill-buffer/td-p/1180777

Using perf to measure average load latency

```
bash df102_NanoAODDimuonAnalysis $ perf stat -M Load_Miss_Real_Latency -- df102_NanoAODDimuonAnalysis 8 Run2012B_DoubleMuParked.
root Run2012C_DoubleMuParked.root

 Performance counter stats for 'df102_NanoAODDimuonAnalysis 8 Run2012B_DoubleMuParked.root Run2012C_DoubleMuParked.root':

       970,326,211      mem_load_uops_retired.hit_lfb #   188.23 Load_Miss_Real_Latency
   438,536,118,970      l1d_pend_miss.pending
     1,359,436,464      mem_load_uops_retired.l1_miss

      20.844816183 seconds time elapsed

     148.564065000 seconds user
       2.591617000 seconds sys


bash df102_NanoAODDimuonAnalysis $ perf list mem_load_uops_retired.hit_lfb | uniq

cache:
  mem_load_uops_retired.hit_lfb
       [Retired load uops which data sources were load uops missed L1 but hit
        FB due to preceding miss to the same cache line with data not ready
        Supports address when precise. Spec update: HSM30 (Precise event)]

Metric Groups:
```

in CPU cycles

Avg. load latency = L1 miss pending cycles / (L1 misses + LFB hits)

perf mem report -s mem

```
Samples: 543K of event 'cpu/mem-loads,ldlat=50/P', Event count (approx.): 172369311
  Overhead       Samples  Memory access
-   41.51%        226882  L1 or L1 hit
   + 17.49% 0xffffffffffffffff
   - 12.00% __GI___clone (inlined)
      - 12.00% start_thread
         - 12.00% tbb::internal::rml::private_worker::thread_routine
            - tbb::internal::rml::private_worker::run
               - 11.99% tbb::internal::market::process
                    tbb::internal::arena::process
                    tbb::internal::custom_scheduler<tbb::internal::IntelSchedulerTraits>::local_wait_for_all
                    tbb::internal::custom_scheduler<tbb::internal::IntelSchedulerTraits>::process_bypass_loop
                  - tbb::interface9::internal::start_for<tbb::blocked_range<unsigned int>, tbb::internal::parallel_for_body<std
                     - 10.39% 0x7f4e4344317f
                        - 10.18% 0x7f4e42dffa42
                           - ROOT::Detail::RDF::RLoopManager::RunAndCheckFilters
                              + 8.11% ROOT::Internal::RDF::RActionCRTP<ROOT::Internal::RDF::RAction<ROOT::Internal::RDF::FillPa
   + 2.10% 0x55d581065ec7
+   18.96%        113828  LFB or LFB hit
+   17.36%        114120  L3 or L3 hit
+   13.12%         59491  Remote Remote Cache (1 hop) or L3 hit
+    5.86%         14681  N/A miss

Cannot load tips.txt file, please install perf!
```

perf mem report -s dso,symbol

```
Samples: 543K of event 'cpu/mem-loads,ldlat=50/P', Event count (approx.): 172369311
  Overhead       Samples  Shared Object                Symbol
+   27.12%        131151  df102_NanoAODDimuonAnalysis  [.] ROOT::Detail::RDF::RFilter<bool (*)(unsigned int), ROOT::Detail::RDF:
+   17.33%         66980  libTree.so.6.22.02           [.] TBranch::GetBasketAndFirst
+   13.86%         62434  df102_NanoAODDimuonAnalysis  [.] ROOT::Detail::RDF::RFilter<bool (*)(ROOT::VecOps::RVec<int> const&),
+    5.51%         32071  libTreePlayer.so.6.22.02     [.] ROOT::Internal::TTreeReaderValueBase::ProxyReadTemplate<&ROOT::Detail
+    5.33%         20471  libROOTDataFrame.so.6.22.02  [.] ROOT::Detail::RDF::RLoopManager::RunAndCheckFilters
+    5.07%         21584  libTree.so.6.22.02           [.] TBranch::GetEntry
+    3.66%         24977  df102_NanoAODDimuonAnalysis  [.] ROOT::Internal::RDF::RColumnValue<ROOT::VecOps::RVec<float> >::Get<RO
     2.76%         16451  libTreePlayer.so.6.22.02     [.] ROOT::Internal::TTreeReaderValueBase::GetAddress
+    2.48%         18397  df102_NanoAODDimuonAnalysis  [.] ROOT::Internal::RDF::RColumnValue<ROOT::VecOps::RVec<int> >::Get<ROOT
+    2.28%         17109  df102_NanoAODDimuonAnalysis  [.] std::swap<ROOT::VecOps::RVec<int> >
+    2.15%         16151  df102_NanoAODDimuonAnalysis  [.] std::swap<ROOT::VecOps::RVec<float> >
+    2.02%         16377  df102_NanoAODDimuonAnalysis  [.] ROOT::Detail::RDF::RCustomColumn<float (*)(ROOT::VecOps::RVec<float>
+    1.72%         12478  df102_NanoAODDimuonAnalysis  [.] ROOT::Internal::RDF::RActionCRTP<ROOT::Internal::RDF::RAction<ROOT::I
     1.23%          4550  [kernel.kallsyms]            [k] copy_user_enhanced_fast_string
+    1.23%          5657  df102_NanoAODDimuonAnalysis  [.] ROOT::Detail::RDF::RLoopManager::CheckFilters@plt
+    0.88%          5830  libRIO.so.6.22.02            [.] TBufferFile::ReadFastArray
+    0.67%         10988  df102_NanoAODDimuonAnalysis  [.] ROOT::VecOps::InvariantMass<float>
     0.58%          8216  libHist.so.6.22.02           [.] TH1::Fill
     0.49%          3864  libTreePlayer.so.6.22.02     [.] 0x00000000000df63f
     0.36%          7884  libz.so.1.2.11               [.] inflate_table
     0.32%          4976  libRIO.so.6.22.02            [.] TBufferFile::ReadArray
```

~46% of high latency loads happen in functions related to RDF filters...

perf mem report – symbol annotation

```
Samples: 543K of event 'cpu/mem-loads,ldlat=50/P', 5000 Hz, Event count (approx.): 172369311
ROOT::Detail::RDF::RFilter<bool (*)(unsigned int), ROOT::Detail::RDF::RLoopManager>::CheckFilters  /nvme/amadio/df102_NanoAODDim
Percent |       add    $0x8,%rsp
  0.15  |       pop    %rbx
  0.68  |       pop    %rbp
  1.23  |       pop    %r12
  0.04  |       pop    %r13
  0.27  |       pop    %r14
  0.01  |       pop    %r15
        |     ← retq
        |       nop
  0.00  |       mov    %rdi,%rbx
        |     if (!fPrevData.CheckFilters(slot, entry)) {
 15.13  |       mov    0xf0(%rdi),%rdi
        |       mov    %rdx,%rbp
        |     → callq  ROOT::Detail::RDF::RLoopManager::CheckFilters@plt
        |       test   %al,%al
        |     → jne    26d60 <ROOT::Detail::RDF::RFilter<bool (*)(unsigned int), ROOT::Detail::RDF::RLoopManager>::CheckFilters(u
        |     fLastResult[slot] = false;
        |       mov    0x30(%rbx),%rdx
        |       movl   $0x0,(%rdx,%r13,4)
        |     fLastCheckedEntry[slot] = entry;
  5.40  |       mov    0x18(%rbx),%rdx
Press 'h' for help on key bindings
```

per-thread values in contiguous memory

perf mem report -s phys_daddr

```
Samples: 543K of event 'cpu/mem-loads,ldlat=50/P', Event count (approx.): 172369311
  Overhead       Samples  Data Physical Address
-     4.00%         20056  [.] 0x0000000e4fff6e08
     3.53% __GI___clone (inlined)
        start_thread
        tbb::internal::rml::private_worker::thread_routine
        tbb::internal::rml::private_worker::run
        tbb::internal::market::process
        tbb::internal::arena::process
        tbb::internal::custom_scheduler<tbb::internal::IntelSchedulerTraits>::local_wait_for_all
        tbb::internal::custom_scheduler<tbb::internal::IntelSchedulerTraits>::process_bypass_loop
      - tbb::interface9::internal::start_for<tbb::blocked_range<unsigned int>, tbb::internal::parallel_for_body<std::function<v
         - 3.05% 0x7f4e4344317f
              0x7f4e42dffa42
           - ROOT::Detail::RDF::RLoopManager::RunAndCheckFilters
              + 2.20% ROOT::Internal::RDF::RActionCRTP<ROOT::Internal::RDF::RAction<ROOT::Internal::RDF::FillParHelper<TH1D>,
              0.86% ROOT::Detail::RDF::RFilter<bool (*)(unsigned int), ROOT::Detail::RDF::RLoopManager>::CheckFilters
+     3.75%          9553  [.] 0x0000000e4fff6df0
+     2.68%         12526  [.] 0x0000000e4fff6ee0
+     2.46%         12056  [.] 0x0000000e4fff6e20
+     2.10%         11397  [.] 0x0000000e4fff6ee8
+     2.06%         10564  [.] 0x0000000e4fff6eb0
+     1.48%          9597  [.] 0x000000058bb3c248
Press '?' for help on key bindings
```

high load latency happens in nearby addresses

perf c2c – cache to cache analysis

```
PERF-C2C(1)                                perf Manual                                PERF-C2C(1)

NAME
       perf-c2c - Shared Data C2C/HITM Analyzer.

SYNOPSIS
       perf c2c record [<options>] <command>
       perf c2c record [<options>] -- [<record command options>] <command>
       perf c2c report [<options>]

DESCRIPTION
       C2C stands for Cache To Cache.

       The perf c2c tool provides means for Shared Data C2C/HITM analysis. It allows you to track down the cacheline
       contentions.

       On x86, the tool is based on load latency and precise store facility events provided by Intel CPUs. On PowerPC, the
       tool uses random instruction sampling with thresholding feature.

       These events provide: - memory address of the access - type of the access (load and store details) - latency (in
       cycles) of the load access

       The c2c tool provide means to record this data and report back access details for cachelines with highest contention
       - highest number of HITM accesses.
```

See also: https://hpc-wiki.info/hpc/FalseSharing

perf c2c report

Shared Data Cache Line Table (44 entries, sorted on Total HITMs)

	Index	Cacheline Address	Node	PA cnt	Total records	Tot Hitm	LLC Load Hitm Total	LLC Load Hitm Lcl	LLC Load Hitm Rmt	Store Reference Total	Store Reference L1Hit	Store Reference L1Miss	Load Dram Lcl
+	0	0x561f510789c0	0-1	7463	13115	17.40%	5172	1368	3804	11	3	9	18
+	1	0x561f518b1740	0	5382	8536	12.44%	3698	1593	2105	1747	836	911	21
+	2	0x561f51078a00	0-1	6447	10046	10.42%	3099	892	2207	27	3	24	64
+	3	0x561f518b1e40	0	5749	8859	9.96%	2962	801	2161	181	18	163	7
+	4	0x561f518afdc0	0-1	3304	5386	7.34%	2182	738	1444	456	45	411	16
+	5	0x561f518b1a40	0	4888	7554	7.25%	2154	721	1433	2927	1244	1683	16
+	6	0x561f518b1b80	0	2923	4846	6.74%	2004	617	1387	36	1	35	19
+	7	0x561f518b1800	0	3320	5532	6.39%	1900	544	1356	1333	591	742	10
+	8	0x561f51078a40	0-1	3543	5902	4.45%	1322	352	970	21	5	16	9
+	9	0x561f518b1ac0	0	1446	2186	1.30%	387	73	314	868	327	541	16
+	10	0x561f518b2640	0-1	1364	2150	1.09%	324	10	314	290	54	236	3
+	11	0x561f518b2580	0-1	1711	2348	0.94%	279	181	98	402	105	297	4
+	12	0x561f518b1a80	0	510	1815	0.83%	247	9	238	734	148	586	3
+	13	0x561f518b24c0	0-1	1113	1535	0.82%	245	145	100	278	88	190	4
+	14	0x561f518b34c0	0-1	991	1990	0.66%	196	1	195	1108	1106	2	5
+	15	0x561f518b1840	0	347	1244	0.57%	170	7	163	346	66	280	2
+	16	0x561f518b2540	0-1	1358	2220	0.57%	170	0	170	906	904	2	3
+	17	0x561f518b2680	0-1	1723	2943	0.54%	160	1	159	1668	1470	198	6
+	18	0x561f518b3040	0-1	866	1830	0.50%	148	1	147	1050	1050	0	1
+	19	0x561f518b1ec0	0	17115	30175	0.49%	147	23	124	0	0	0	400

? - help

perf c2c report

```
Shared Data Cache Line Table     (44 entries, sorted on Total HITMs)
          ---------- Cacheline ----------    Total      Tot  ----- LLC Load Hitm -----  ---- Store Reference ----  --- Load Dram
  Index             Address  Node  PA cnt  records     Hitm     Total      Lcl      Rmt    Total    L1Hit    L1Miss       Lcl
-     0      0x561f510789c0   0-1    7463    13115   17.40%      5172     1368     3804       11        3         9        18
    + 0.00% ROOT::Detail::RDF::RFilter<bool (*)(unsigned int), ROOT::Detail::RDF::RLoopManager>::CheckFilters
    + 0.00% ROOT::Detail::RDF::RLoopManager::CheckFilters@plt
      0.00% apic_timer_interrupt
+     1      0x561f518b1740      0    5382     8536   12.44%      3698     1593     2105     1747      836       911        21
+     2      0x561f51078a00   0-1     6447    10046   10.42%      3099      892     2207       27        3        24        64
+     3      0x561f518b1e40      0    5749     8859    9.96%      2962      801     2161      181       18       163         7
+     4      0x561f518afdc0   0-1     3304     5386    7.34%      2182      738     1444      456       45       411        16
+     5      0x561f518b1a40      0    4888     7554    7.25%      2154      721     1433     2927     1244      1683        16
+     6      0x561f518b1b80      0    2923     4846    6.74%      2004      617     1387       36        1        35        19
+     7      0x561f518b1800      0    3320     5532    6.39%      1900      544     1356     1333      591       742        10
+     8      0x561f51078a40   0-1     3543     5902    4.45%      1322      352      970       21        5        16         9
+     9      0x561f518b1ac0      0    1446     2186    1.30%       387       73      314      868      327       541        16
+    10      0x561f518b2640   0-1     1364     2150    1.09%       324       10      314      290       54       236         3
+    11      0x561f518b2580   0-1     1711     2348    0.94%       279      181       98      402      105       297         4
+    12      0x561f518b1a80      0     510     1815    0.83%       247        9      238      734      148       586         3
+    13      0x561f518b24c0   0-1     1113     1535    0.82%       245      145      100      278       88       190         4
+    14      0x561f518b34c0   0-1      991     1990    0.66%       196        1      195     1108     1106         2         5
+    15      0x561f518b1840      0     347     1244    0.57%       170        7      163      346       66       280         2
+    16      0x561f518b2540   0-1     1358     2220    0.57%       170        0      170      906      904         2         3
? - help
```

perf c2c report – cacheline details (press ‘d’)

Cacheline 0x561f510789c0

	----- HITM -----		-- Store Refs --		------- CL --------				---------- cycles ----------			Total	cpu
	Rmt	Lcl	L1 Hit	L1 Miss	Off	Node	PA cnt	Code address	rmt hitm	lcl hitm	load	records	cnt
+	8.39%	9.14%	0.00%	0.00%	0x10	0-1	3	0x561f4ebc2470	408	284	339	1002	2
+	4.02%	2.63%	0.00%	0.00%	0x10	0-1	3	0x561f4ebd0d23	515	327	320	540	2
+	1.74%	1.90%	0.00%	0.00%	0x10	0-1	3	0x561f4ebd0cff	464	314	322	247	2
+	1.24%	2.27%	0.00%	0.00%	0x10	0-1	3	0x561f4ebd0d0e	360	241	309	173	2
+	1.05%	0.73%	0.00%	0.00%	0x10	0-1	3	0x561f4ebd0cfb	641	241	656	130	2
+	0.34%	0.22%	0.00%	0.00%	0x10	0-1	3	0x561f4ebd0d05	337	164	364	48	2
+	0.26%	0.29%	0.00%	0.00%	0x10	0-1	3	0x561f4ebd0d12	332	173	202	39	2
+	0.11%	0.07%	0.00%	0.00%	0x10	0-1	3	0x561f4ebd0d10	306	150	118	17	2
+	0.03%	0.00%	33.33%	11.11%	0x10	1	1	0x561f4ebd0d20	312	0	0	4	1
+	0.00%	0.00%	0.00%	33.33%	0x10	0-1	2	0x561f4ebd0d46	0	0	0	3	1
+	7.33%	8.48%	0.00%	0.00%	0x18	0-1	3	0x561f4ebc2470	389	283	343	934	5
+	3.52%	3.36%	0.00%	0.00%	0x18	0-1	3	0x561f4ebd0d23	467	372	247	529	4
+	1.42%	1.54%	0.00%	0.00%	0x18	0-1	3	0x561f4ebd0cff	404	381	330	216	4
+	1.52%	0.80%	0.00%	0.00%	0x18	0-1	3	0x561f4ebd0d0e	394	181	304	178	4
+	1.13%	0.58%	0.00%	0.00%	0x18	0-1	3	0x561f4ebd0cfb	466	293	699	129	4
+	0.21%	0.37%	0.00%	0.00%	0x18	0-1	3	0x561f4ebd0d05	349	299	435	37	4
+	0.18%	0.22%	0.00%	0.00%	0x18	0-1	2	0x561f4ebd0d12	301	176	208	26	3
+	0.08%	0.00%	0.00%	0.00%	0x18	0-1	2	0x561f4ebd0d10	296	0	220	12	2
+	0.03%	0.00%	0.00%	0.00%	0x18	0-1	2	0x561f4ebd0d03	275	0	440	3	2
+	0.00%	0.00%	33.33%	0.00%	0x18	0	1	0x561f4ebd0d20	0	0	0	1	1

? - help

perf c2c report – cacheline details, more columns

Cacheline 0x561f510789c0

	load	Total records	cpu cnt		Symbol	Shared Object	Source:Line	Node
+	339	1002	2	[.]	ROOT::Detail::RDF::RLoopMa	df102_NanoAODDimuonAnalysis	ROOT::Detail::RDF::RLoopManage	1
+	320	540	2	[.]	ROOT::Detail::RDF::RFilter	df102_NanoAODDimuonAnalysis	RFilter.hxx:87	1
+	322	247	2	[.]	ROOT::Detail::RDF::RFilter	df102_NanoAODDimuonAnalysis	RFilter.hxx:86	1
+	309	173	2	[.]	ROOT::Detail::RDF::RFilter	df102_NanoAODDimuonAnalysis	RFilter.hxx:99	1
+	656	130	2	[.]	ROOT::Detail::RDF::RFilter	df102_NanoAODDimuonAnalysis	RFilter.hxx:86	1
+	364	48	2	[.]	ROOT::Detail::RDF::RFilter	df102_NanoAODDimuonAnalysis	RFilter.hxx:86	1
+	202	39	2	[.]	ROOT::Detail::RDF::RFilter	df102_NanoAODDimuonAnalysis	RFilter.hxx:99	1
+	118	17	2	[.]	ROOT::Detail::RDF::RFilter	df102_NanoAODDimuonAnalysis	RFilter.hxx:99	1
+	0	4	1	[.]	ROOT::Detail::RDF::RFilter	df102_NanoAODDimuonAnalysis	RFilter.hxx:99	1
+	0	3	1	[.]	ROOT::Detail::RDF::RFilter	df102_NanoAODDimuonAnalysis	RFilter.hxx:96	1
+	343	934	5	[.]	ROOT::Detail::RDF::RLoopMa	df102_NanoAODDimuonAnalysis	ROOT::Detail::RDF::RLoopManage	0
+	247	529	4	[.]	ROOT::Detail::RDF::RFilter	df102_NanoAODDimuonAnalysis	RFilter.hxx:87	0
+	330	216	4	[.]	ROOT::Detail::RDF::RFilter	df102_NanoAODDimuonAnalysis	RFilter.hxx:86	0
+	304	178	4	[.]	ROOT::Detail::RDF::RFilter	df102_NanoAODDimuonAnalysis	RFilter.hxx:99	0
+	699	129	4	[.]	ROOT::Detail::RDF::RFilter	df102_NanoAODDimuonAnalysis	RFilter.hxx:86	0
+	435	37	4	[.]	ROOT::Detail::RDF::RFilter	df102_NanoAODDimuonAnalysis	RFilter.hxx:86	0
+	208	26	3	[.]	ROOT::Detail::RDF::RFilter	df102_NanoAODDimuonAnalysis	RFilter.hxx:99	0
+	220	12	2	[.]	ROOT::Detail::RDF::RFilter	df102_NanoAODDimuonAnalysis	RFilter.hxx:99	0
+	440	3	2	[.]	ROOT::Detail::RDF::RFilter	df102_NanoAODDimuonAnalysis	RFilter.hxx:86	0
+	0	1	1	[.]	ROOT::Detail::RDF::RFilter	df102_NanoAODDimuonAnalysis	RFilter.hxx:99	1

? - help

perf c2c report – cacheline details, expanded stack

```
Cacheline 0x561f510789c0
  ----- HITM -----  -- Store Refs --  ------- CL --------                    ---------- cycles ----------    Total        cpu
      Rmt      Lcl   L1 Hit  L1 Miss     Off  Node  PA cnt        Code address rmt hitm  lcl hitm      load  records        cnt
+    0.00%    0.00%    0.00%   11.11%   0x18     0       1  0x561f4ebd0d46         0         0         0        1          1
-    7.23%    7.16%    0.00%    0.00%   0x20   0-1       3  0x561f4ebc2470       413       257       333      891          3
     ROOT::Detail::RDF::RLoopManager::CheckFilters@plt
     ROOT::Detail::RDF::RFilter<bool (*)(unsigned int), ROOT::Detail::RDF::RLoopManager>::CheckFilters
     ROOT::Detail::RDF::RFilter<bool (*)(ROOT::VecOps::RVec<int> const&), ROOT::Detail::RDF::RFilter<bool (*)(unsigned int), RO
     ROOT::Internal::RDF::RActionCRTP<ROOT::Internal::RDF::RAction<ROOT::Internal::RDF::FillParHelper<TH1D>, ROOT::Detail::RDF:◆
     ROOT::Detail::RDF::RLoopManager::RunAndCheckFilters
     0x7feaa2613a42
     0x7feaa2c5717f
     tbb::interface9::internal::start_for<tbb::blocked_range<unsigned int>, tbb::internal::parallel_for_body<std::function<void
     tbb::internal::custom_scheduler<tbb::internal::IntelSchedulerTraits>::process_bypass_loop
     tbb::internal::custom_scheduler<tbb::internal::IntelSchedulerTraits>::local_wait_for_all
     tbb::internal::arena::process
     tbb::internal::market::process
     tbb::internal::rml::private_worker::run
     tbb::internal::rml::private_worker::thread_routine
     start_thread
     __GI___clone (inlined)
+    3.63%    3.36%    0.00%    0.00%   0x20   0-1       3  0x561f4ebd0d23       469       375       334      501          3
+    1.42%    2.56%    0.00%    0.00%   0x20   0-1       3  0x561f4ebd0d0e       353       217       282      222          3
? - help
```

Data-Type Profiling

- Introduced in latest versions of perf
  - First available in perf 6.8
  - Better with perf 6.10 and later
- Data recorded with **`perf mem record`**
- Attributes sample addresses back to the data-type associated with that address
- Two methods to make use of it
  - **`perf mem report -s type,typeoff`**
  - **`perf annotate --data-type=<type>`**
- Still needs some help for pointer types
  - Part of LLVM's `-fdebug-info-for-profiling`

Data profiling of git fsck on Linux kernel repository

```
linux $ perf mem record --ldlat=50 -- git fsck # collect loads with latency > 50 cycles
Checking objects: 100% (10271178/10271178), done.
Checking connectivity: 10223233, done.
[ perf record: Woken up 395 times to write data ]
[ perf record: Captured and wrote 99.888 MB perf.data (1629334 samples) ]
linux $ perf mem report --stdio -s type,typeoff --percent-limit 1
# Overhead       Samples  Data Type  Data Type Offset
# ........  ............  .........  .................
    83.88%       1450665  (unknown)  (unknown) +0 (no field)
     4.30%        100109  (stack operation)  (stack operation) +0 (no field)
     3.79%          3712  struct malloc_chunk  struct malloc_chunk +8 (mchunk_size)
     1.40%          2336  struct object  struct object +0 (parsed)
     1.08%           842  struct malloc_chunk  struct malloc_chunk +32 (fd_nextsize)
linux $ perf annotate --data-type=object
Annotate type: 'struct object' in /usr/libexec/git-core/git (3282 samples):
================================================================================
    samples     offset       size  field
        104          0         40  struct object         {
          0          0          4      unsigned int     parsed;
          0          0          4      unsigned int     type;
          0          0          4      unsigned int     flags;
        104          4         36      struct object_id oid {
        104          4         32          unsigned char*       hash;
          0         36          4          int  algo;
                                         };
                                     };
```

See also: https://lwn.net/Articles/955709/

Packing Simulation Revisited

```
$ perf record -e cycles -F 1000 -- pack -f 0.6 ellipsoids
100.00% 0.6000 0.0000/min 2.6e-02 ev/s 38.2 s
[ perf record: Woken up 5 times to write data ]
[ perf record: Captured 1.457 MB perf.data (38005 samples) ]
$ perf report --stdio --percent-limit 0.25
# Overhead  Command  Shared Object Symbol
# ........  .......  ............ ......................
#
    39.53%  pack     pack         intersect
    32.60%  pack     pack         HGrid::find_neighbors
     7.62%  pack     pack         Ellipsoid::support
     3.76%  pack     pack         closest_point_tetrahedron
     3.37%  pack     pack         closest_point_triangle
     2.77%  pack     pack         Simplex::closest
     2.46%  pack     pack         Ellipsoid::bounding_radius
     2.07%  pack     pack         check_overlap
     1.94%  pack     pack         Simplex::contains
     1.15%  pack     libm.so.6    __sincos
     0.37%  pack     pack         HGrid::make_hash
     0.36%  pack     pack         HGrid::insert
```

```
void HGrid::find_neighbors(const Particle* p, std::vector<Particle*>& neighbors) {
  hash_t hash; unsigned int mask = occupied_level_mask;
  Vector x = p->position();

  for (unsigned int level = 0; level <= MAX_LEVEL; mask >>=1, level++) {
    if (mask == 0) return; /* no more occupied levels to check */
    if ((mask & 1) == 0) continue; /* level is not occupied */

    float cell_size = MAX_CELL_SIZE / (1 << level);
    float inv_cell_size = 1.0f / cell_size;
    float delta = p->bounding_radius(t_curr) + cell_size/2.0 + EPSILON;

    hash.data.level = level;
    short int imin = (short int) floor((x[0] - delta) * inv_cell_size);
    short int jmin = (short int) floor((x[1] - delta) * inv_cell_size);
    short int kmin = (short int) floor((x[2] - delta) * inv_cell_size);
    short int imax = (short int)  ceil((x[0] + delta) * inv_cell_size);
    short int jmax = (short int)  ceil((x[1] + delta) * inv_cell_size);
    short int kmax = (short int)  ceil((x[2] + delta) * inv_cell_size);

    for (short int i = imin; i < imax; i++) {
      hash.data.x = i;
      for (short int j = jmin; j < jmax; j++) {
        hash.data.y = j;
        for (short int k = kmin; k < kmax; k++) {
          hash.data.z = k;
          if (grid.find(hash.value) != grid.end()) {
            Particle *neighbor = grid[hash.value];
            while(neighbor != NULL) {
              if (neighbor != p)
                neighbors.push_back(neighbor);
              neighbor = neighbor->next();
            }
          }
        }
      }
    }
  }
}
```

grid is an instance of std::map<hash, Particle*>, but this class stores data as an array of red-black tree nodes, with the value contiguously in memory in each node. This means that it loads all values along with the keys when traversing it.

Packing Simulation Revisited

```
$ perf record -e cycles -F 1000 -- pack -f 0.6 ellipsoids
100.00% 0.6000 0.0000/min 2.6e-02 ev/s 38.2 s
[ perf record: Woken up 5 times to write data ]
[ perf record: Captured 1.457 MB perf.data (38005 samples) ]
$ perf report --stdio --percent-limit 0.25
# Overhead  Command  Shared Object Symbol
# ........  .......  ............ ......................
#
    39.53%  pack     pack         intersect
    32.60%  pack     pack         HGrid::find_neighbors
     7.62%  pack     pack         Ellipsoid::support
     3.76%  pack     pack         closest_point_tetrahedron
     3.37%  pack     pack         closest_point_triangle
     2.77%  pack     pack         Simplex::closest
     2.46%  pack     pack         Ellipsoid::bounding_radius
     2.07%  pack     pack         check_overlap
     1.94%  pack     pack         Simplex::contains
     1.15%  pack     libm.so.6    __sincos
     0.37%  pack     pack         HGrid::make_hash
     0.36%  pack     pack         HGrid::insert
```

```
enum _Rb_tree_color { _S_red = false, _S_black = true };

_Rb_tree_node_base
{
  typedef _Rb_tree_node_base* _Base_ptr;
  typedef const _Rb_tree_node_base* _Const_Base_ptr;

  _Rb_tree_color      _M_color;
  _Base_ptr           _M_parent;
  _Base_ptr           _M_left;
  _Base_ptr           _M_right;
};

template<typename _Val>
  struct _Rb_tree_node : public _Rb_tree_node_base
  {
    typedef _Rb_tree_node<_Val>* _Link_type;

    _Val _M_value_field;

    _Val*
    _M_valptr()
    { return _M_storage._M_ptr(); }

    const _Val*
    _M_valptr() const
    { return _M_storage._M_ptr(); }
};
```

Value contiguous in memory after enum and 3 pointers.

enum = int = 4 bytes
4 pointers = 4 x 8 = 32 bytes
node = 36 bytes, misaligned with cachelines

Traversing the map to find key is expensive

```
Samples: 41K of event 'ibs_op//', 1000 Hz, Event count (approx.): 174896447829
HGrid::find_neighbors  /home/amadio/src/rocpack/pack [Percent: local period]
Samples |      struct less : public binary_function<_Tp, _Tp, bool>
        |      {
        |      _GLIBCXX14_CONSTEXPR
        |      bool
        |      operator()(const _Tp& __x, const _Tp& __y) const
        |      { return __x < __y; }
   159  |        mov       -0x48(%rbp),%rdx
    88  |        mov       %r13,%r9
   100  |        mov       %rsi,%rax
   121  |        xchg      %ax,%ax
        |      if (!_M_impl._M_key_compare(_S_key(__x), __k))
  2266  | 240:   cmp       %rdx,0x20(%rax)
        |      ↓ jae       450
        |      { return static_cast<_Link_type>(__x->_M_right); }
   766  |        mov       0x18(%rax),%rax
        |      while (__x != 0)
   898  |        test      %rax,%rax
     2  |      ↑ jne       240
   258  |        mov       %r9,%rax
        |      find(const _Key& __k)
        |      {
Press 'h' for help on key bindings
```

Cache hit/miss rates with perf mem

```
bash $ perf mem record -C 0 -t load --ldlat=20 -- taskset -c 0 ./pack --seed 17 -f 0.45 bench2.in
100.00% 0.4500 0.0000/min 1.8e-02 ev/s 54.1 s
[ perf record: Woken up 64 times to write data ]
[ perf record: Captured and wrote 16.013 MB perf.data (208040 samples) ]
bash $ perf mem report --stdio -s mem
# To display the perf.data header info, please use --header/--header-only options.
#
#
# Total Lost Samples: 0
#
# Samples: 208K of event 'cpu/mem-loads,ldlat=20/P'
# Total weight : 9857968
# Sort order   : mem
#
# Overhead       Samples  Memory access
# ........  ............  ....................................
#
    90.65%        178426  L1 hit
     4.36%         12798  LFB/MAB hit
     3.95%         15602  L2 hit
     0.98%          1203  L3 hit
     0.04%             7  Remote Any cache hit
     0.01%             3  RAM hit
     0.00%             1  L3 miss
```

perf mem report -s mem,sym --hierarchy

```
Samples: 208K of event 'cpu/mem-loads,ldlat=20/P', Event count (approx.): 9857968
  Overhead       Samples     Memory access / Symbol
-   90.65%        178426     L1 hit
       47.89%         84778     [.] HGrid::find_neighbors
       22.39%         47769     [.] intersect
        8.78%         19603     [.] Simplex::closest
        3.42%          9093     [.] Simplex::add_vertex
        3.13%          6182     [.] closest_point_tetrahedron
        2.01%          3721     [.] Simplex::contains
        1.56%          3719     [.] Polyhedron::support
        0.26%           553     [.] PeriodicBoundary::reposition
-    4.36%         12798     LFB/MAB hit
        2.61%          8059     [.] intersect
        1.00%          3449     [.] HGrid::find_neighbors
-    3.95%         15602     L2 hit
        2.18%          8606     [.] HGrid::find_neighbors
        1.62%          6434     [.] intersect
-    0.98%          1203     L3 hit
        0.41%           594     [.] intersect
        0.33%           469     [.] HGrid::find_neighbors

Tip: To compute metrics for samples use perf record -e '{cycles,instructions}' ... ; perf script -F +metric
```

New data profiling can show which structures are hit

```
bash $ perf annotate --data-type=Particle

Annotate type: 'Particle' in pack (7940 samples):
================================================================================
    samples     offset       size  field
       7940          0        160  Particle      {
          5         80          8      Shape*   m_shape;
          0         88          4      float    m_growth_rate;
          0         92          4      float    m_mass;
          0         96          4      float    m_inv_mass;
          0        100          4      unsigned int     m_tag;
          0        104          4      unsigned int     m_event_id;
          2        112          8      long unsigned int        m_hash;
          1        120          8      long unsigned int        m_collisions;
          1        128          8      Particle*        m_prev;
       7931        136          8      Particle*        m_next;
          0        144         16      union     {
          0        144         16          __m128       m_color;
          0        144         16          struct        {
          0        144          4              float    r;
          0        148          4              float    g;
          0        152          4              float    b;
          0        156          4              float    a;
                                      };
                                  };
```

When looking for adjacent particles to check for intersections, we build the list of neighbors and iterate through a list, so m_next is a hot field for slow loads (>20 cycles).

Color not used during simulation, can make it compile-time optional for better performance, and reorder hot field into the first cache line.

After optimization, less samples on hot member

```
bash $ perf annotate --data-type=Particle

Annotate type: 'Particle' in pack (7541 samples):
================================================================================
    samples     offset       size  field
       7541          0        128  Particle      {
       7530          0          8      Particle*        m_next;
          0          8          8      Particle*        m_prev;
          2         16          8      long unsigned int        m_hash;
          2         24          8      Shape*   m_shape;
          0         32         16      Point    m_position;
          0         48         16      Quaternion       m_orientation;
          0         64         16      Vector   m_velocity;
          0         80         16      Vector   m_ang_velocity;
          0         96          4      float    m_time;
          0        100          4      float    m_growth_rate;
          0        104          4      float    m_mass;
          0        108          4      float    m_inv_mass;
          6        112          8      long unsigned int        m_collisions;
          0        120          4      unsigned int     m_tag;
          1        124          4      unsigned int     m_event_id;
                                   };
```

L1 cache hit rate also improved after optimization

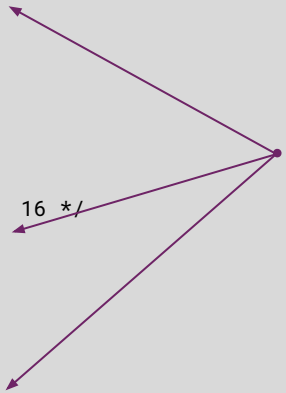
```
bash $ perf mem record -C 0 -t load --ldlat=20 -- taskset -c 0 ./pack --seed 17 -f 0.45 bench2.in
100.00% 0.4500 0.0000/min 1.9e-02 ev/s 53.4 s
[ perf record: Woken up 63 times to write data ]
[ perf record: Captured and wrote 15.887 MB perf.data (205679 samples) ]
bash $ perf mem report --stdio -s mem
# To display the perf.data header info, please use --header/--header-only options.
#
#
# Total Lost Samples: 0
#
# Samples: 205K of event 'cpu/mem-loads,ldlat=20/P'
# Total weight : 9887519
# Sort order   : mem
#
# Overhead       Samples  Memory access
# ........  ............  ....................................
#
    93.01%        183502  L1 hit
     3.12%         12339  L2 hit
     3.11%          9010  LFB/MAB hit
     0.68%           813  L3 hit
     0.07%            12  Remote RAM hit
     0.01%             2  L3 miss
     0.00%             1  RAM hit
```

pahole – inspect layout of data structures

```
bash $ pahole -M -C G4LogicalVolume /usr/lib64/libG4geometry.so
class G4LogicalVolume {
public:

        int ()(void) * *           _vptr.G4LogicalVolume; /*     0     8 */
        static G4LVManager                 subInstanceManager; /*     0     0 */
        G4PhysicalVolumeList       fDaughters;           /*     8    24 */
        class G4String             fName;                /*    32    32 */
        /* --- cacheline 1 boundary (64 bytes) --- */
        class G4UserLimits *       fUserLimits;          /*    64     8 */
        class G4SmartVoxelHeader * fVoxel;               /*    72     8 */
        G4double                   fSmartless;           /*    80     8 */
        class G4Region *           fRegion;              /*    88     8 */
        G4double                   fBiasWeight;          /*    96     8 */
        class shared_ptr<const G4VisAttributes> fVisAttributes; /*   104    16 */
        class G4VSolid *           fSolid;               /*   120     8 */
        /* --- cacheline 2 boundary (128 bytes) --- */
        class G4VSensitiveDetector * fSensitiveDetector; /*   128     8 */
        class G4FieldManager *     fFieldManager;        /*   136     8 */
        class G4LVData *           lvdata;               /*   144     8 */
        G4int                      instanceID;           /*   152     4 */
        enum EVolume               fDaughtersVolumeType; /*   156     4 */
        G4bool                     fOptimise;            /*   160     1 */
        G4bool                     fRootRegion;          /*   161     1 */
        G4bool                     fLock;                /*   162     1 */

        /* size: 168, cachelines: 3, members: 18, static members: 1 */
        /* padding: 5 */
        /* last cacheline: 40 bytes */
};
```



Members which are accessed most often are in 3 different cachelines.

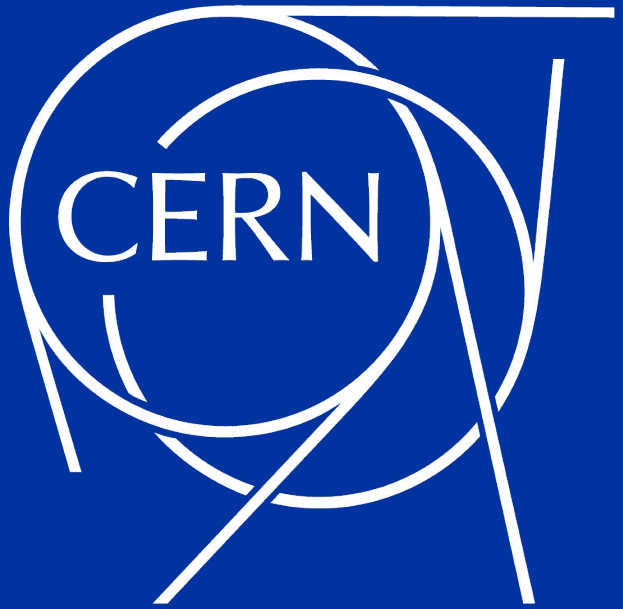
Must load fName whenever checking fDaughters. Inefficient use of memory hierarchy due to mix of hot and cold data on each cacheline.

Summary and Conclusions

- Compiler can help with optimizing computations, not so much with memory access
- Many tools are available to inspect and optimize memory access patterns
  - pahole, perf mem, perf c2c, VTune memory access analysis
- Data-Oriented design collects key concepts to design memory efficient software
  - Separate data structures and operations on data
  - Focus on avoiding high latency and wasting memory bandwidth
- Many other indicators of bad memory access patterns
  - Backend Bound - stalled cycles at backend are a good indicator of inefficiencies
    - Core Bound - data dependencies in arithmetics, chains of high latency instructions
    - Memory Bound - not only read, but also write, some codes can be store bound
  - Cache Misses - main indicator of memory access issues
    - Need to watch for problem size: hit rate high for small workloads, inevitably higher on larger workloads
    - Cache associativity and locality can lead to complex issues, avoid loops with power of 2 trip length

“Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. 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%”

— Donald Knuth


CERN