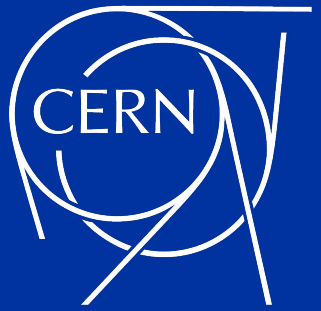


Performance Analysis and Optimization on Linux

G. Amadio (EP-SFT)

17 Aug 2022

Performance is challenging

- Measurement
  - Sophisticated hardware architecture (out of order, superscalar)
  - Variable CPU frequency scaling (turbo boost, thermal throttling)
  - Overhead from instrumentation and measurement
  - Missing symbols (JIT, interpreted languages, stripped binaries)
  - Broken stack unwinding (deep call stacks, inlining, missing frame pointer)
- Optimization and Tuning
  - Concurrency issues (shared resources with hyperthreading, contention)
  - Compiler optimizations (exceptions vs vectorization, denormals, dead code)
  - Memory alignment, access patterns, fragmentation
  - Addressing the wrong issue (optimizing compute for memory bound problem and vice-versa)

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.

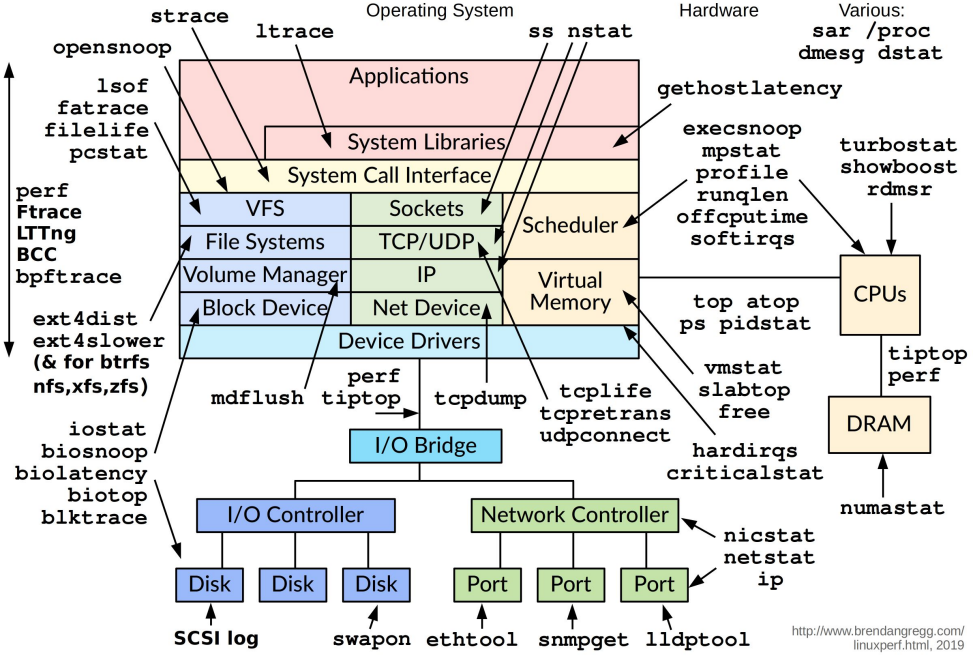
Rule 5 *Data dominates.* If you've chosen the right data structures and organized things well, the algorithms will almost always be self-evident. *Data structures, not algorithms, are central to programming.*

Rule 6 There is no Rule 6.

from “Notes on C Programming”, by Rob Pike

Performance Analysis Process

- System Level
  - Kernel
  - Disk
  - Network
- Application Level
  - CPU Utilization
  - Algorithmic complexity
- Hardware Level
  - Cache misses
  - Branch mispredictions
  - Data dependencies
  - Divisions, square roots

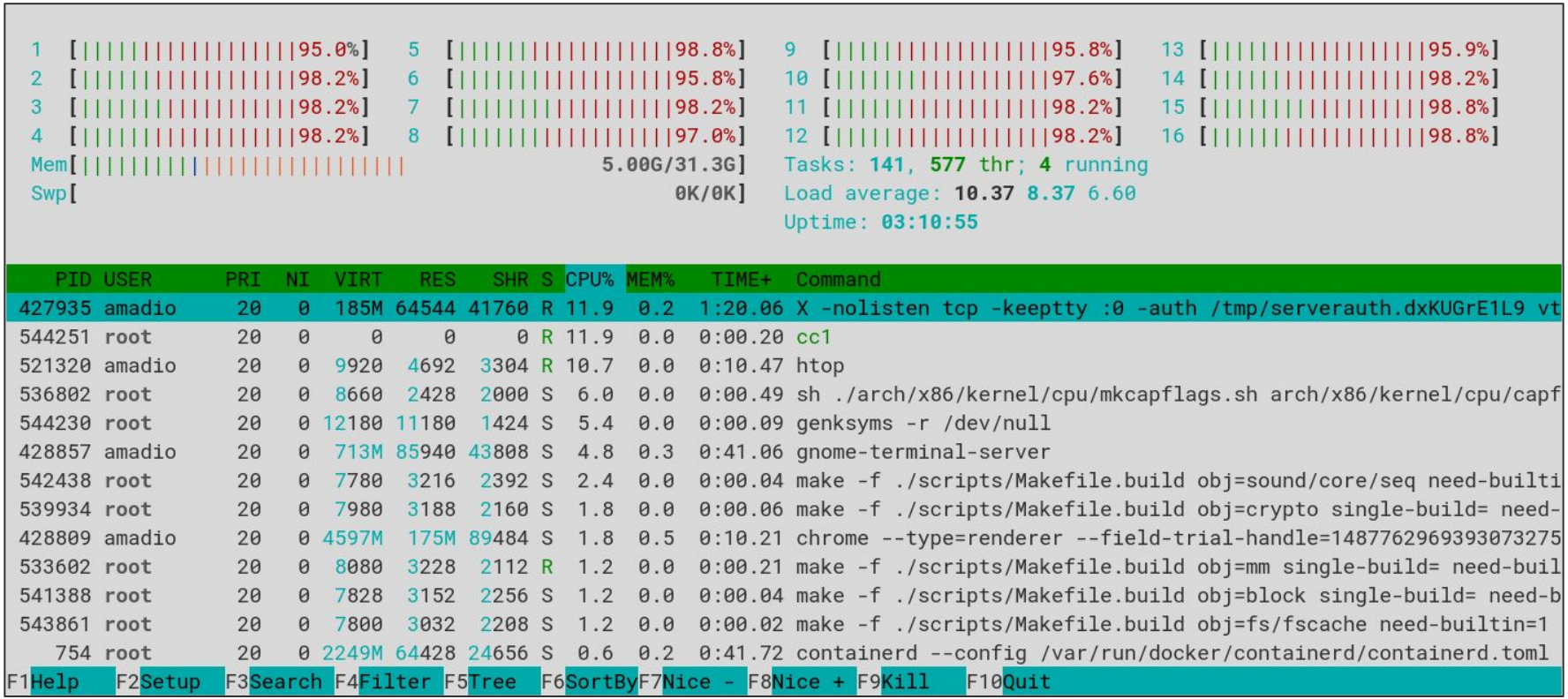


Use system observability tools to identify and fix performance bottlenecks.

top – display Linux processes

```
top - 13:35:10 up  2:46,  1 user,  load average: 4.55, 4.50, 4.65
Tasks: 377 total,   1 running, 376 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.5 us,  2.2 sy,  0.0 ni, 97.0 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
MiB Mem :  32068.8 total,  15999.6 free,   2788.3 used,  13280.9 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.  28308.2 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 428110 amadio    20   0  988864 331980 116156 S   5.6   1.0   1:03.66 chrome
 428758 amadio    20   0 5197800 569200 309072 S   5.2   1.7   0:47.03 chrome
 428033 amadio    20   0 1645520 350320 163768 S   3.3   1.1   0:56.24 chrome
 427935 amadio    20   0  224688  97592  42768 S   2.6   0.3   0:55.38 X
 429718 amadio    20   0 4926216 343652 225104 S   2.3   1.0   0:31.50 chrome
 474382 amadio    20   0   11940   4032   3340 R   1.6   0.0   0:00.15 top
 428857 amadio    20   0  718868  73724  43804 S   1.0   0.2   0:34.22 gnome-terminal-
 474494 amadio    20   0   53720   9960   8864 S   0.7   0.0   0:00.02 import
     46 root      20   0       0      0      0 S   0.3   0.0   0:06.85 ksoftirqd/7
    661 root      20   0    9836   5812   5272 S   0.3   0.0   0:07.10 systemd-logind
   1263 root     -51   0       0      0      0 S   0.3   0.0   0:18.03 irq/105-nvidia
   1265 root      20   0       0      0      0 S   0.3   0.0   0:02.16 nv_queue
   1345 amadio    20   0  646400  11836   7856 S   0.3   0.0   0:19.91 pulseaudio
 427988 amadio    20   0   20684  15520  13604 S   0.3   0.0   0:00.56 i3bar
 427989 amadio    20   0    8656   3456   3092 S   0.3   0.0   0:01.38 status
 428112 amadio    20   0  382248 108280  70220 S   0.3   0.3   0:08.61 chrome
 429302 amadio    20   0 4698660 180700  88036 S   0.3   0.6   0:05.41 chrome
```

htop – interactive process viewer



mpstat – report processor statistics

```
MPSTAT(1)                                Linux User's Manual                                MPSTAT(1)

NAME
       mpstat - Report processors related statistics.

SYNOPSIS
       mpstat  [  -A ] [ --dec={ 0 | 1 | 2 } ] [ -n ] [ -u ] [ -T ] [ -V ] [ -I { keyword[,...] | ALL } ] [ -N { node_list |
       ALL } ] [ -o JSON ] [ -P { cpu_list | ALL } ] [ interval [ count ] ]

DESCRIPTION
       The mpstat command writes to standard output activities for each available processor, processor  0  being  the  first
       one.   Global  average activities among all processors are also reported.  The mpstat command can be used both on SMP
       and UP machines, but in the latter, only global average activities will be printed. If no activity has been selected,
       then the default report is the CPU utilization report.

       The  interval  parameter specifies the amount of time in seconds between each report.  A value of 0 (or no parameters
       at all) indicates that processors statistics are to be reported for the time since system startup (boot).  The  count
       parameter  can  be  specified in conjunction with the interval parameter if this one is not set to zero. The value of
       count determines the number of reports generated at interval seconds apart. If the interval  parameter  is  specified
       without the count parameter, the mpstat command generates reports continuously.

OPTIONS
       -A     This  option  is equivalent to specifying -n -u -I ALL.  This option also implies specifying -N ALL -P ALL un-
 Manual page mpstat(1) line 1 (press h for help or q to quit)
```

mpstat – report processor statistics

```
bash ~ $ mpstat 2 15
Linux 5.9.8-gentoo (gentoo)     11/13/20        _x86_64_        (16 CPU)

14:35:07     CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
14:35:09     all   79.19    0.00    9.70    0.72    0.00    0.00    0.00    0.00    0.00   10.39
14:35:11     all   78.99    0.00    9.12    1.97    0.00    0.03    0.00    0.00    0.00    9.90
14:35:13     all   77.92    0.00   11.29    0.72    0.00    0.00    0.00    0.00    0.00   10.07
14:35:15     all   78.72    0.00   10.15    0.88    0.00    0.03    0.00    0.00    0.00   10.22
14:35:17     all   77.10    0.00    9.57    1.73    0.00    0.00    0.00    0.00    0.00   11.61
14:35:19     all   82.42    0.00    6.52    0.34    0.00    0.00    0.00    0.00    0.00   10.71
14:35:21     all   81.50    0.00    8.17    0.63    0.00    0.03    0.00    0.00    0.00    9.67
14:35:23     all   81.25    0.00    7.76    0.60    0.00    0.00    0.00    0.00    0.00   10.40
14:35:25     all   81.02    0.00    7.89    0.25    0.00    0.00    0.00    0.00    0.00   10.84
14:35:27     all   79.71    0.00    8.59    1.32    0.00    0.00    0.00    0.00    0.00   10.38
14:35:29     all   80.21    0.00    9.21    2.72    0.00    0.00    0.00    0.00    0.00    7.87
14:35:31     all   69.21    0.00   10.38   10.35    0.00    0.09    0.00    0.00    0.00    9.97
14:35:33     all   73.29    0.00   10.22    5.02    0.00    0.03    0.00    0.00    0.00   11.44
14:35:35     all   79.51    0.00    9.36    0.47    0.00    0.03    0.00    0.00    0.00   10.64
14:35:37     all   77.52    0.00   10.99    1.35    0.00    0.03    0.00    0.00    0.00   10.11
Average:     all   78.50    0.00    9.26    1.94    0.00    0.02    0.00    0.00    0.00   10.28
bash ~ $ _
```

iostat – report CPU and I/O statistics

```
IOSTAT(1)                                Linux User's Manual                                IOSTAT(1)

NAME
       iostat - Report Central Processing Unit (CPU) statistics and input/output statistics for devices and partitions.

SYNOPSIS
       iostat [ -c ] [ -d ] [ -h ] [ -k | -m ] [ -N ] [ -s ] [ -t ] [ -V ] [ -x ] [ -y ] [ -z ] [ --dec={ 0 | 1 | 2 } ] [ -j
       { ID | LABEL | PATH | UUID | ... } ] [ -o JSON ] [ [ -H ] -g group_name ] [ --human  ]  [  --pretty  ]  [  -p  [  de-
       vice[,...] | ALL ] ] [ device [...] | ALL ] [ interval [ count ] ]

DESCRIPTION
       The  iostat  command  is used for monitoring system input/output device loading by observing the time the devices are
       active in relation to their average transfer rates. The iostat command generates reports that can be used  to  change
       system configuration to better balance the input/output load between physical disks.

       The first report generated by the iostat command provides statistics concerning the time since the system was booted,
       unless the -y option is used (in this case, this first report is omitted).  Each subsequent report  covers  the  time
       since  the previous report. All statistics are reported each time the iostat command is run. The report consists of a
       CPU header row followed by a row of CPU statistics. On multiprocessor systems, CPU statistics are calculated  system-
       wide as averages among all processors. A device header row is displayed followed by a line of statistics for each de-
       vice that is configured.

       The interval parameter specifies the amount of time in seconds between each report. The count parameter can be speci-
 Manual page iostat(1) line 1 (press h for help or q to quit)
```

iostat – report CPU and I/O statistics

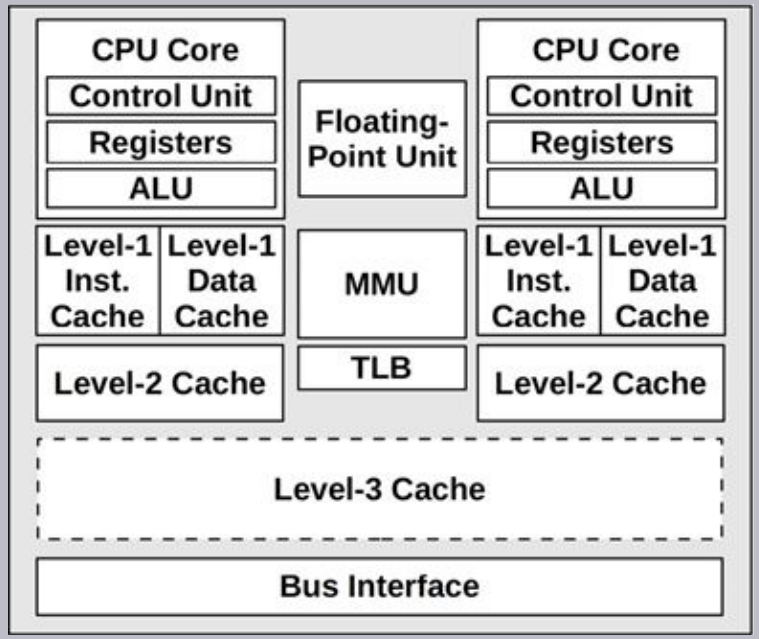
```
bash ~ $ iostat --human --pretty 20 3 # 3 periods of 20 seconds of I/O activity (first summary is since boot)
Linux 5.9.8-gentoo (epsftws)    11/18/20        _x86_64_        (16 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           2.6%    0.0%    0.5%    0.0%    0.0%   96.8%

      tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd Device
    10.31        64.8k       520.4k         0.0k       5.3G      42.2G       0.0k nvme0n1
     0.02         3.8k         0.3k         0.0k     316.2M      27.7M       0.0k sda


avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          73.0%    0.0%    4.7%    0.0%    0.0%   22.3%

      tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd Device
    96.35       229.8k        29.3M         0.0k       4.5M     586.1M       0.0k nvme0n1
     0.00         0.0k         0.0k         0.0k       0.0k       0.0k       0.0k sda


avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          80.3%    0.0%    3.9%    0.0%    0.0%   15.7%

      tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd Device
    48.45        64.4k        10.2M         0.0k       1.3M     203.5M       0.0k nvme0n1
     0.00         0.0k         0.0k         0.0k       0.0k       0.0k       0.0k sda
```

CPU Hardware Architecture

CPU Architecture

- Logical Components
  - Control Unit
  - Arithmetic Logic Units (ALUs)
  - Floating Point Unit (FPU)
  - Branch Predictor Unit (BPU)
  - Memory Management Unit (MMU)
  - Translation Lookaside Buffer (TLB)
- Memory Subsystem
  - L1(~32–512KB)
    - L1 Instruction Cache
    - L1 Data Cache
  - L2 (~1–8MB)
    - Instruction/Data Shared Cache
  - L3 (up to ~64MB)
    - Last level cache (LLC)

Generic Dual Core CPU

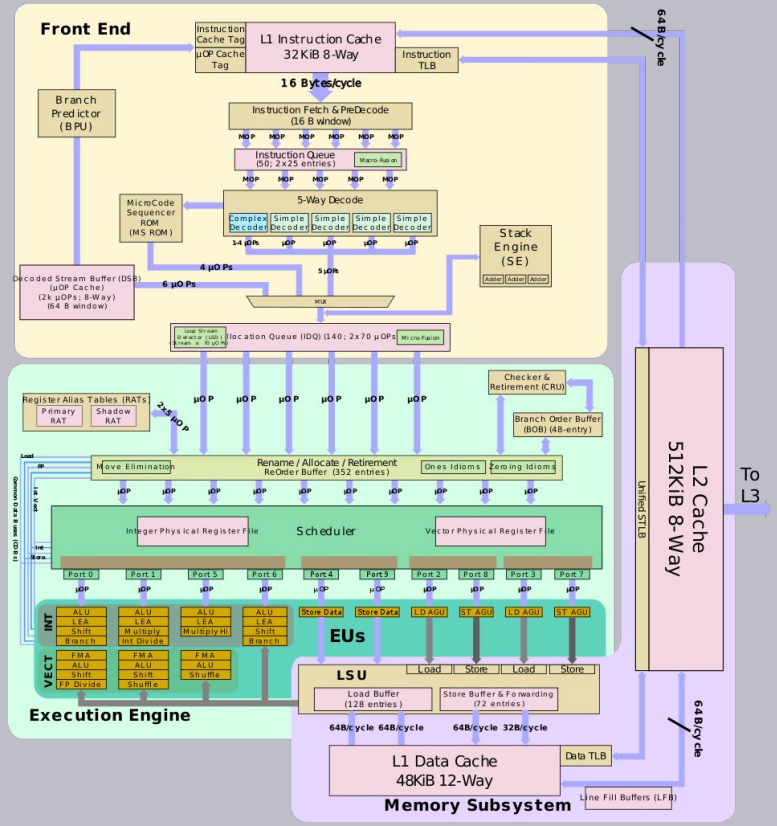


source: Systems Performance 2nd Edition, Brendan Gregg

Microarchitecture of Intel's Sunny Cove Core

- Front End
  - Instruction Fetch and Decode
  - Branch Predictor Unit (BPU)
  - L1 Instruction Cache
  - Instruction TLB
- Back End
  - Execution Engine
    - Register Renaming
    - Move Elimination
  - Memory Subsystem
    - Load/Store Units
    - L1 Data Cache
    - L2 Shared Cache
    - Data TLB

Sunny Cove Core Microarchitecture

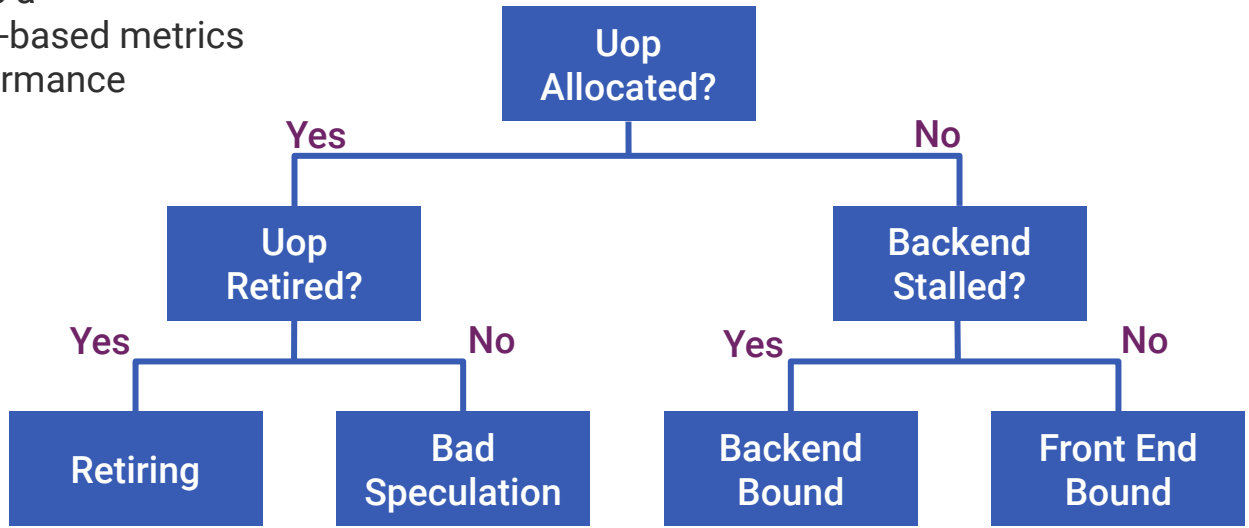


source: https://en.wikichip.org

Top-Down Microarchitecture Analysis

The Top-Down Characterization is a hierarchical organization of event-based metrics that identifies the dominant performance bottlenecks in an application.

Its aim is to show, on average, how well the CPU's pipelines are being utilized while running an application.

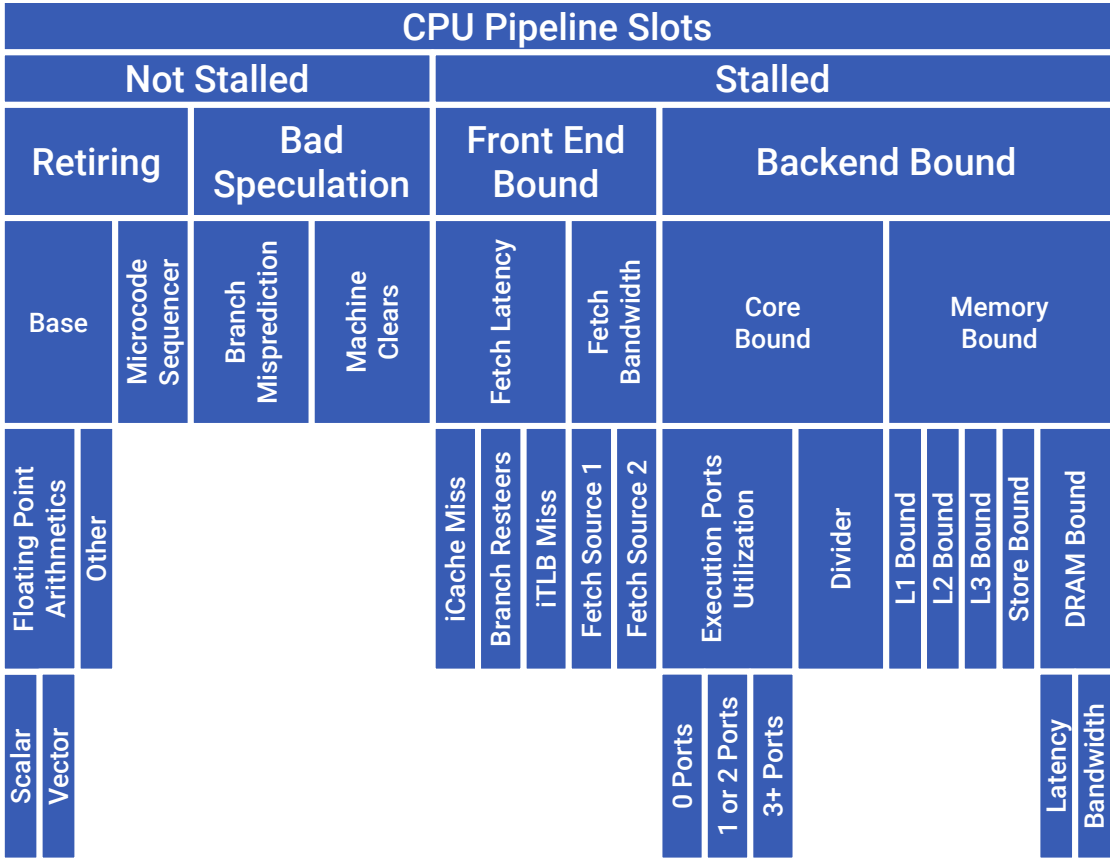


Ahmad Yasin, "A Top-Down method for performance analysis and counters architecture," *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Monterey, CA, 2014, pp. 35-44, doi: 10.1109/ISPASS.2014.6844459.

https://www.intel.com/content/www/us/en/develop/documentation/vtune-cookbook/top/methodologies/top-down-microarchitecture-analysis-method.html

Top-Down Microarchitecture Analysis

- Front End Bound
  - Code Duplication
  - Code Layout (Locality)
  - Frequent Branching
  - Unnecessary Work
- Back End Bound
  - Core Bound
    - Data Dependencies
    - Divisions and Special Functions
  - Memory Bound
    - False Sharing
    - Remote Memory Accesses
    - Scattered Memory Accesses
    - Excessive Memory Accesses



Ahmad Yasin, "A Top-Down method for performance analysis and counters architecture," *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Monterey, CA, 2014, pp. 35-44, doi: 10.1109/ISPASS.2014.6844459.

Intel CPU Metrics Reference

Performance Analysis with perf

perf – Performance analysis tools for Linux

- Official Linux profiler (source code is part of the kernel itself)
- Both hardware and software based performance monitoring
- Much lower overhead compared with instrumentation-based profiling
- Kernel and user space
- Counting and Sampling
  - Counting — count occurrences of a given event (e.g. cache misses)
  - Event-based Sampling — a sample is recorded when a threshold of events has occurred
  - Time-based Sampling — samples are recorded at a given fixed frequency
  - Instruction-based Sampling — processor follows instructions and samples events they create
- Static and Dynamic Tracing
  - Static — pre-defined tracepoints in software
  - Dynamic — tracepoints created using uprobes (user) or kprobes (kernel)

Perf Wiki *https://www.brendangregg.com/perf.html*

perf – subcommands

```
bash ~ $ perf

 usage: perf [--version] [--help] [OPTIONS] COMMAND [ARGS]

 The most commonly used perf commands are:
   annotate        Read perf.data (created by perf record) and display annotated code
   archive         Create archive with object files with build-ids found in perf.data file
   c2c             Shared Data C2C/HITM Analyzer.
   config          Get and set variables in a configuration file.
   data            Data file related processing
   diff            Read perf.data files and display the differential profile
   evlist          List the event names in a perf.data file
   list            List all symbolic event types
   mem             Profile memory accesses
   record          Run a command and record its profile into perf.data
   report          Read perf.data (created by perf record) and display the profile
   sched           Tool to trace/measure scheduler properties (latencies)
   script          Read perf.data (created by perf record) and display trace output
   stat            Run a command and gather performance counter statistics
   timechart       Tool to visualize total system behavior during a workload
   top             System profiling tool.
   version         display the version of perf binary
   probe           Define new dynamic tracepoints
   trace           strace inspired tool

 See 'perf help COMMAND' for more information on a specific command.
```

perf – hardware and software events

```
bash ~ $ perf list hw cache

List of pre-defined events (to be used in -e):

  branch-instructions OR branches                    [Hardware event]
  branch-misses                                      [Hardware event]
  cache-misses                                       [Hardware event]
  cache-references                                   [Hardware event]
  cpu-cycles OR cycles                               [Hardware event]
  instructions                                       [Hardware event]
  stalled-cycles-backend OR idle-cycles-backend      [Hardware event]
  stalled-cycles-frontend OR idle-cycles-frontend    [Hardware event]

  L1-dcache-load-misses                              [Hardware cache event]
  L1-dcache-loads                                    [Hardware cache event]
  L1-dcache-prefetches                               [Hardware cache event]
  L1-icache-load-misses                              [Hardware cache event]
  L1-icache-loads                                    [Hardware cache event]
  branch-load-misses                                 [Hardware cache event]
  branch-loads                                       [Hardware cache event]
  dTLB-load-misses                                   [Hardware cache event]
  dTLB-loads                                         [Hardware cache event]
  iTLB-load-misses                                   [Hardware cache event]
  iTLB-loads                                         [Hardware cache event]
```

```
bash ~ $ perf list sw

List of pre-defined events (to be used in -e):

  alignment-faults                                   [Software event]
  bpf-output                                         [Software event]
  context-switches OR cs                             [Software event]
  cpu-clock                                          [Software event]
  cpu-migrations OR migrations                       [Software event]
  dummy                                              [Software event]
  emulation-faults                                   [Software event]
  major-faults                                       [Software event]
  minor-faults                                       [Software event]
  page-faults OR faults                              [Software event]
  task-clock                                         [Software event]

  duration_time                                      [Tool event]
```

perf – Intel pipeline events

```
bash ~ $ perf list pipeline

List of pre-defined events (to be used in -e):

pipeline:
  arith.divider_active
       [Cycles when divide unit is busy executing divide or square root operations. Accounts for integer and floating-point operations]
  baclears.any
       [Counts the total number when the front end is resteered, mainly when the BPU cannot provide a correct prediction]
  br_inst_retired.all_branches
       [All (macro) branch instructions retired Spec update: SKL091]
  br_inst_retired.all_branches_pebs
       [All (macro) branch instructions retired Spec update: SKL091 (Must be precise)]
  br_inst_retired.conditional
       [Conditional branch instructions retired Spec update: SKL091 (Precise event)]
  br_inst_retired.far_branch
       [Counts the number of far branch instructions retired Spec update: SKL091 (Precise event)]
  br_inst_retired.near_call
       [Direct and indirect near call instructions retired Spec update: SKL091 (Precise event)]
  br_inst_retired.near_return
       [Return instructions retired Spec update: SKL091 (Precise event)]
  br_inst_retired.near_taken
       [Taken branch instructions retired Spec update: SKL091 (Precise event)]
  br_inst_retired.not_taken
       [Counts all not taken macro branch instructions retired Spec update: SKL091 (Precise event)]
  br_misp_retired.all_branches
       [All mispredicted macro branch instructions retired]
  ...
```

https://perfmon-events.intel.com

perf – AMD core events

```
bash ~ $ perf list core

List of pre-defined events (to be used in -e):

core:
  ex_div_busy
       [Div Cycles Busy count]
  ex_div_count
       [Div Op Count]
  ex_ret_brn
       [Retired Branch Instructions]
  ex_ret_brn_far
       [Retired Far Control Transfers]
  ex_ret_brn_ind_misp
       [Retired Indirect Branch Instructions Mispredicted]
  ex_ret_brn_misp
       [Retired Branch Instructions Mispredicted]
  ex_ret_brn_resync
       [Retired Branch Resyncs]
  ex_ret_brn_tkn
       [Retired Taken Branch Instructions]
  ex_ret_brn_tkn_misp
       [Retired Taken Branch Instructions Mispredicted]
  ex_ret_cond
       [Retired Conditional Branch Instructions]
  ex_ret_cond_misp
       [Retired Conditional Branch Instructions Mispredicted]
  ...
```

https://developer.amd.com/wordpress/media/2013/12/56255_OSRR-1.pdf

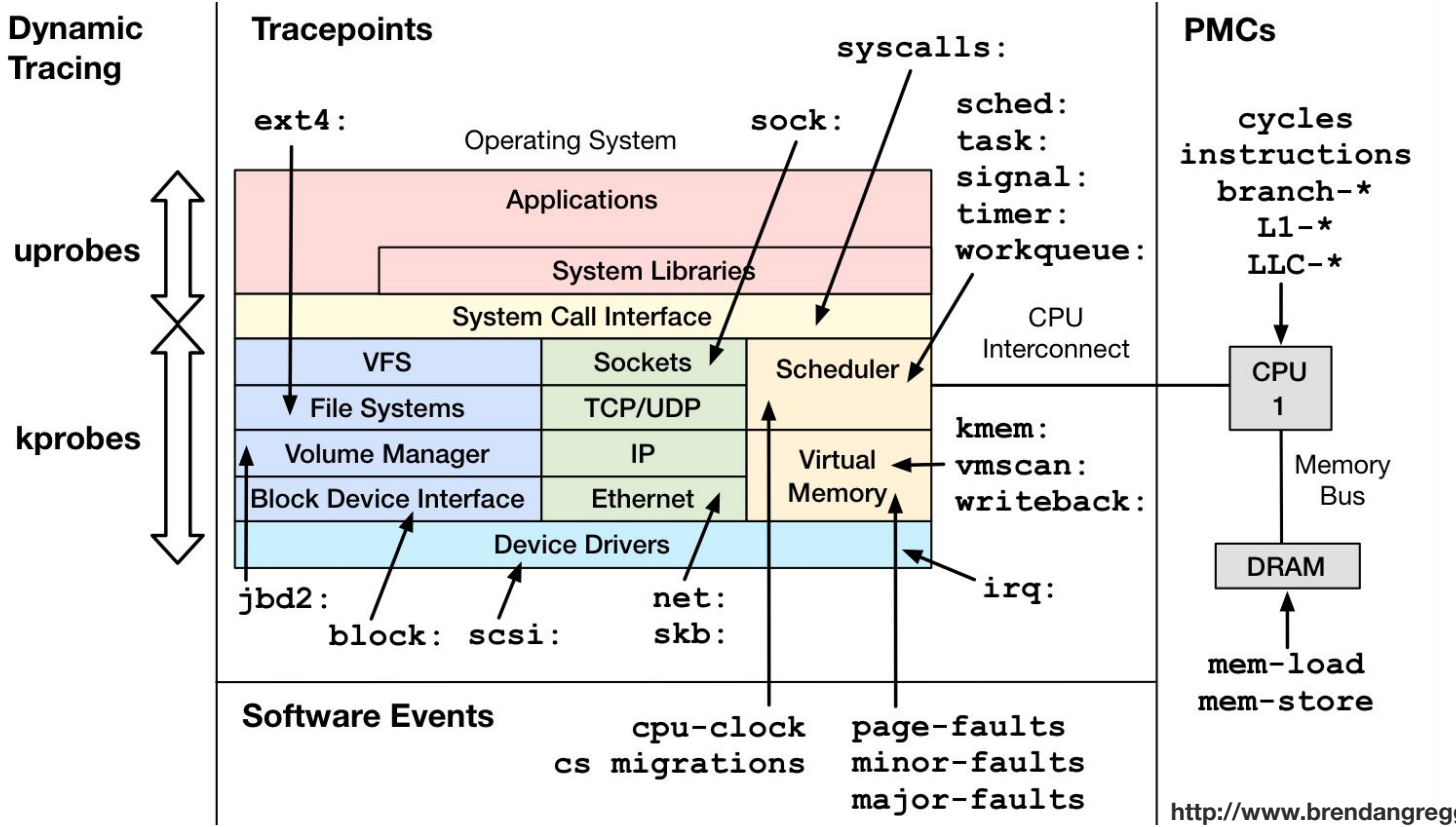
perf – pre-packaged metrics (Intel only)

```
bash ~ $ perf list metrics

Metrics:

  Backend_Bound
       [This category represents fraction of slots where no uops are delivered due to a lack of required resources for accepting new uops in the Backend]
  Bad_Speculation
       [This category represents fraction of slots wasted due to incorrect speculations]
  BpTB
       [Branch instructions per taken branch]
  CLKS
       [Per-Logical Processor actual clocks when the Logical Processor is active]
  CPI
       [Cycles Per Instruction (per Logical Processor)]
  CPU_Utilization
       [Average CPU Utilization]
  CoreIPC
       [Instructions Per Cycle (per physical core)]
  Frontend_Bound
       [This category represents fraction of slots where the processor's Frontend undersupplies its Backend]
  ILP
       [Instruction-Level-Parallelism (average number of uops executed when there is at least 1 uop executed)]
  IPC
       [Instructions Per Cycle (per Logical Processor)]
  Instructions
       [Total number of retired Instructions]
  IpB
       [Instructions per Branch (lower number means higher occurance rate)]
  IpCall
       [Instruction per (near) call (lower number means higher occurance rate)]
  IpL
       [Instructions per Load (lower number means higher occurance rate)]
  ...
```

perf – pre-packaged metrics (Intel only)

```
bash ~ $ perf stat -M Frontend_Bound,Backend_Bound,Bad_Speculation,Retiring -- root -l -q


 Performance counter stats for 'root -l -q':

         535853293      cycles
                                                  #      0.32 Frontend_Bound          (50.07%)
         676507752      idq_uops_not_delivered.core                                  (50.07%)
         803157447      uops_issued.any           #      0.10 Bad_Speculation
                                                  #      0.28 Backend_Bound           (49.93%)
         540449552      cycles
                                                  #      0.31 Retiring                (49.93%)
         676523326      idq_uops_not_delivered.core                                  (49.93%)
          19393734      int_misc.recovery_cycles                                     (49.93%)
         667220596      uops_retired.retire_slots                                    (49.93%)

       0.243072802 seconds time elapsed

       0.158384000 seconds user
       0.088028000 seconds sys

bash ~ $
```

perf – event sources



http://www.brendangregg.com/perf.html

perf – static tracepoint events

```
bash ~ $ sudo perf list 'sched:*'

List of pre-defined events (to be used in -e):

  sched:sched_kthread_stop                           [Tracepoint event]
  sched:sched_kthread_stop_ret                       [Tracepoint event]
  sched:sched_migrate_task                           [Tracepoint event]
  sched:sched_move_numa                              [Tracepoint event]
  sched:sched_pi_setprio                             [Tracepoint event]
  sched:sched_process_exec                           [Tracepoint event]
  sched:sched_process_exit                           [Tracepoint event]
  sched:sched_process_fork                           [Tracepoint event]
  sched:sched_process_free                           [Tracepoint event]
  sched:sched_process_wait                           [Tracepoint event]
  sched:sched_stat_runtime                           [Tracepoint event]
  sched:sched_stick_numa                             [Tracepoint event]
  sched:sched_swap_numa                              [Tracepoint event]
  sched:sched_switch                                 [Tracepoint event]
  sched:sched_wait_task                              [Tracepoint event]
  sched:sched_wake_idle_without_ipi                  [Tracepoint event]
  sched:sched_wakeup                                 [Tracepoint event]
  sched:sched_wakeup_new                             [Tracepoint event]
  sched:sched_waking                                 [Tracepoint event]
```

perf stat – report performance counter statistics

```
bash ~ $ perf stat -d -r 10 root -l -q >/dev/null

 Performance counter stats for 'root -l -q' (10 runs):

          227.48 msec task-clock                #    1.002 CPUs utilized            ( +-  0.36% )
              91      context-switches          #    0.401 K/sec                    ( +-  1.63% )
              11      cpu-migrations            #    0.047 K/sec                    ( +-  2.80% )
           18300      page-faults               #    0.080 M/sec                    ( +-  0.43% )
       978777612      cycles                    #    4.303 GHz                      ( +-  0.31% )  (75.63%)
        38110759      stalled-cycles-frontend   #    3.89% frontend cycles idle     ( +-  0.55% )  (73.89%)
       257316592      stalled-cycles-backend    #   26.29% backend cycles idle      ( +-  0.81% )  (73.18%)
      1560795166      instructions              #    1.59  insn per cycle
                                                #    0.16  stalled cycles per insn  ( +-  0.32% )  (73.20%)
       320431808      branches                  # 1408.637 M/sec                    ( +-  0.31% )  (73.87%)
         3174211      branch-misses             #    0.99% of all branches          ( +-  1.34% )  (75.84%)
       686386527      L1-dcache-loads           # 3017.395 M/sec                    ( +-  0.24% )  (77.32%)
        16998166      L1-dcache-load-misses     #    2.48% of all L1-dcache hits    ( +-  0.91% )  (77.07%)
   <not supported>    LLC-loads
   <not supported>    LLC-load-misses

         0.226973 +- 0.000781 seconds time elapsed  ( +-  0.34% )

bash ~ $ _
```

perf stat – report performance counter statistics

```
bash ~ $ perf stat -e cache-references,cache-misses -a -I 1000 sleep 6
#           time             counts unit events
     1.000754704          13141899      cache-references
     1.000754704           4368606      cache-misses              #   33.242 % of all cache refs
     2.001475489          21386311      cache-references
     2.001475489           5695019      cache-misses              #   26.629 % of all cache refs
     3.002077001          23851940      cache-references
     3.002077001           7274179      cache-misses              #   30.497 % of all cache refs
     4.002639322          19548423      cache-references
     4.002639322           6168710      cache-misses              #   31.556 % of all cache refs
     5.003088102          15088969      cache-references
     5.003088102           4608206      cache-misses              #   30.540 % of all cache refs
     6.000807605          19465673      cache-references
     6.000807605           5109426      cache-misses              #   26.248 % of all cache refs

 Performance counter stats for 'system wide':

         112483215      cache-references
          33224146      cache-misses              #   29.537 % of all cache refs

       6.001844739 seconds time elapsed

bash ~ $ _
```

perf record – recording and reporting data

```
bash ~ $ perf record -g -F max -- root.exe -l -q
info: Using a maximum frequency rate of 63200 Hz

[ perf record: Woken up 9 times to write data ]
[ perf record: Captured and wrote 3.232 MB perf.data (16954 samples) ]
bash ~ $ perf report -q --stdio -c root.exe | head -n 17
# comm: root.exe
    31.81%     0.00%  root.exe               [.] _start
           |
           ---_start
              __libc_start_main
              main
              |
              |--30.22%--TRint::TRint
              |          |
              |          |--28.25%--TApplication::TApplication
              |          |          |
              |          |           --28.23%--ROOT::Internal::GetROOT2
              |          |                     |
              |          |                      --28.23%--TROOT::InitInterpreter
              |          |                                |
              |          |                                |--22.34%--CreateInterpreter
              |          |                                |          |
bash ~ $ _
```

perf record – recording from a running process

```
bash ~ $ perf record --call-graph=dwarf -F10000 -p $(pgrep g4run) -- sleep 10 2>/dev/null # record 10s from running process
bash ~ $ perf report --stdio -c g4run -g none -w 0,0,0,19,80 2>/dev/null | tail -n +10 | head -n 21
# Children      Self  Shared Object        Symbol
# ........  ........  ...................  ..........................................................................
#
    98.97%     0.09%  libG4event.so        [.] G4EventManager::DoProcessing
    98.57%     0.00%  libG4run.so          [.] G4WorkerRunManager::ProcessOneEvent
    98.55%     0.00%  libG4run.so          [.] G4WorkerRunManager::DoEventLoop
    98.52%     0.00%  libG4run.so          [.] G4RunManager::BeamOn
    98.49%     0.00%  libG4run.so          [.] G4WorkerRunManager::DoWork
    98.20%     0.00%  libG4run.so          [.] G4MTRunManagerKernel::StartThread
    98.19%     0.08%  libG4tracking.so     [.] G4TrackingManager::ProcessOneTrack
    98.16%     0.00%  libstdc++.so.6.0.28  [.] execute_native_thread_routine
    98.14%     0.00%  libpthread-2.32.so   [.] start_thread
    98.06%     0.00%  libc-2.32.so         [.] __GI___clone (inlined)
    90.95%     0.92%  libG4tracking.so     [.] G4SteppingManager::Stepping
    52.67%     2.18%  libG4tracking.so     [.] G4SteppingManager::DefinePhysicalStepLength
    29.63%     0.00%  libG4tracking.so     [.] G4VProcess::AlongStepGPIL (inlined)
    25.74%     0.36%  libG4tracking.so     [.] G4SteppingManager::InvokePostStepDoItProcs
    25.36%     1.15%  libG4tracking.so     [.] G4SteppingManager::InvokePSDIP
    22.42%     0.95%  libG4processes.so    [.] G4Transportation::AlongStepGetPhysicalInteractionLength
    21.44%     0.00%  libG4tracking.so     [.] G4VProcess::PostStepGPIL (inlined)
    20.58%     1.12%  libG4geometry.so     [.] G4Navigator::ComputeStep
bash ~ $ _
```

perf report – reporting by self time

```
bash ~ $ perf report --stdio -c g4run --no-children -g none -w 0,0,0,80 2>/dev/null | tail -n +10 | head -n 22
# Overhead  Shared Object             Symbol
# ........  ........................  ..........................................................................
#
     5.51%  libm-2.32.so              [.] __ieee754_atan2_fma
     5.43%  libG4physicslists.so      [.] G4PhysicsVector::LogVectorValue
     3.92%  libG4processes.so         [.] G4CrossSectionDataStore::GetCrossSection
     3.36%  libG4geometry.so          [.] G4PolyhedraSide::DistanceAway
     2.18%  libG4tracking.so          [.] G4SteppingManager::DefinePhysicalStepLength
     2.15%  libG4processes.so         [.] G4VEmProcess::PostStepGetPhysicalInteractionLength
     1.90%  libG4geometry.so          [.] G4PolyhedraSide::DistanceToOneSide
     1.63%  libG4processes.so         [.] G4VoxelNavigation::LevelLocate
     1.63%  libG4processes.so         [.] G4CrossSectionDataStore::GetIsoCrossSection
     1.58%  libG4geometry.so          [.] G4PolyPhiFace::InsideEdges
     1.26%  libG4digits_hits.so       [.] G4TouchableHistory::GetVolume
     1.21%  libG4geometry.so          [.] G4PolyhedraSide::GetPhi
     1.19%  libG4clhep.so             [.] CLHEP::MixMaxRng::iterate
     1.19%  libG4geometry.so          [.] G4PolyhedraSide::Distance
     1.19%  libm-2.32.so              [.] __sincos
     1.15%  libG4tracking.so          [.] G4SteppingManager::InvokePSDIP
     1.12%  libG4geometry.so          [.] G4VCSGfaceted::Inside
     1.12%  libG4geometry.so          [.] G4Navigator::ComputeStep
     1.09%  libG4processes.so         [.] G4UrbanMscModel::SampleCosineTheta
bash ~ $ _
```

perf report – hierarchical report

```
bash ~ $ perf report --stdio -c g4run --hierarchy -g none 2>/dev/null | tail -n +10 | cut -b -$COLUMNS | head -n $(($LINES - 2))
#       Overhead  Command / Shared Object / Symbol
# ...............  ..........................................................................................
#
   100.00%        g4run
       36.61%        libG4processes.so
           3.92%        [.] G4CrossSectionDataStore::GetCrossSection
           2.15%        [.] G4VEmProcess::PostStepGetPhysicalInteractionLength
           1.63%        [.] G4VoxelNavigation::LevelLocate
           1.63%        [.] G4CrossSectionDataStore::GetIsoCrossSection
       33.07%        libG4geometry.so
           3.36%        [.] G4PolyhedraSide::DistanceAway
           1.90%        [.] G4PolyhedraSide::DistanceToOneSide
           1.58%        [.] G4PolyPhiFace::InsideEdges
        7.30%        libm-2.32.so
           5.51%        [.] __ieee754_atan2_fma
        6.67%        libG4tracking.so
           2.18%        [.] G4SteppingManager::DefinePhysicalStepLength
        5.43%        libG4physicslists.so
           5.43%        [.] G4PhysicsVector::LogVectorValue
        3.59%        libG4clhep.so
                        no entry >= 1.50%
        2.09%        libG4particles.so
bash ~ $
```

perf report --tui

```
Samples: 16K of event 'cycles', Event count (approx.): 1110655926
  Children      Self  Command          Shared Object           Symbol
+   32.84%     1.38%  ccache           [kernel.vmlinux]        [k] entry_SYSCALL_64
+   31.81%     0.00%  root.exe         root.exe                [.] _start
+   31.81%     0.00%  root.exe         libc-2.32.so            [.] __libc_start_main
-   31.81%     0.00%  root.exe         root.exe                [.] main
   - main
      - 30.22% TRint::TRint
         - 28.25% TApplication::TApplication
            - 28.23% ROOT::Internal::GetROOT2
               - 28.23% TROOT::InitInterpreter
                  - 22.34% CreateInterpreter
                     - 22.33% TCling::TCling
                        - 22.01% cling::Interpreter::Interpreter
                           + 16.57% cling::IncrementalParser::Initialize
                           + 3.51% cling::IncrementalParser::commitTransaction
                           + 0.75% cling::IncrementalExecutor::getPointerToGlobalFromJIT
                           + 0.59% cling::IncrementalParser::IncrementalParser
                  + 4.29% TCling::Initialize
                  + 1.38% dlopen@@GLIBC_2.2.5
         + 1.65% TSystem::Load
      + 1.58% TRint::Run
+   30.22%     0.00%  root.exe         libRint.so.6.22.02      [.] TRint::TRint
Press '?' for help on key bindings
```

perf report --tui

```
Samples: 792K of event 'cycles', Event count (approx.): 31698254078309
  Children      Self  Command  Shared Object              Symbol
-   98.70%     0.07%  g4run    libG4tracking.so           [.] G4TrackingManager::ProcessOneTrack
   - 98.63% G4TrackingManager::ProcessOneTrack
      - 91.08% G4SteppingManager::Stepping
         - 46.00% G4SteppingManager::DefinePhysicalStepLength
            + 18.80% G4Transportation::AlongStepGetPhysicalInteractionLength
            + 13.28% G4VDiscreteProcess::PostStepGetPhysicalInteractionLength
            + 4.94% G4VMultipleScattering::AlongStepGetPhysicalInteractionLength
              3.87% G4VEmProcess::PostStepGetPhysicalInteractionLength
            + 1.62% G4VEnergyLossProcess::PostStepGetPhysicalInteractionLength
              0.57% G4VEnergyLossProcess::AlongStepGetPhysicalInteractionLength
         - 30.79% G4SteppingManager::InvokePostStepDoItProcs
            - 30.19% G4SteppingManager::InvokePSDIP
               + 10.58% G4HadronicProcess::PostStepDoIt
               + 9.90% G4Transportation::PostStepDoIt
               + 3.01% G4VEmProcess::PostStepDoIt
               + 2.92% G4HadronElasticProcess::PostStepDoIt
               + 1.22% G4VEnergyLossProcess::PostStepDoIt
         + 10.12% G4SteppingManager::InvokeAlongStepDoItProcs
      + 2.94% G4SteppingManager::SetInitialStep
      + 1.53% G4ProcessManager::StartTracking
        0.81% G4TouchableHistory::GetVolume
Press '?' for help on key bindings
```

perf annotate – annotated source code

```
Samples: 840K of event 'cycles', 100 Hz, Event count (approx.): 33603550952244
G4Transportation::AlongStepGetPhysicalInteractionLength  /usr/lib64/libG4processes.so [Percent: local period]
Percent |       mov         G4Transportation::fUseGravity@@Base-0x2edbe69,%rdx
  0.15  |       movzbl      (%rdx),%edx
  2.50  |       and         %esi,%edx
        |       mov         %edx,%r8d
  0.01  |       and         $0x1,%edx
        |     → jne         f672d0 <G4Transportation::AlongStepGetPhysicalInteractionLength(G4Track const&, double, double, dou
        |     if( (fieldMgr!=nullptr) && (eligibleEM||eligibleGrav) )
  0.35  |       test        %rax,%rax
  0.19  |     → je          f66fc1 <G4Transportation::AlongStepGetPhysicalInteractionLength(G4Track const&, double, double, dou
        |     fieldMgr->ConfigureForTrack( &track );
        |       mov         (%rax),%rdx
  8.91  |       mov         %rax,%rdi
        |       mov         %rax,-0x318(%rbp)
        |       mov         %r12,%rsi
        |     → callq       *0x10(%rdx)
        |     if( ptrField )
  0.75  |       mov         -0x318(%rbp),%rax
        |       cmpq        $0x0,0x8(%rax)
  0.01  |     → je          f66fc1 <G4Transportation::AlongStepGetPhysicalInteractionLength(G4Track const&, double, double, dou
        |     → jmpq        f6731e <G4Transportation::AlongStepGetPhysicalInteractionLength(G4Track const&, double, double, dou
        |       nop
Press 'h' for help on key bindings
```

Optimization Example

Matrix Multiplication

Matrix Multiplication

```c
#include <stdio.h>
#include <stdlib.h>

// This version has minor modifications applied, the
// original version is linked at the bottom of the slide

#define SIZE 1024
#define LENGTH 32

int **mkmatrix(int rows, int cols);
void zeromatrix(int rows, int cols, int **m);
void freematrix(int rows, int **m);

int **mmult(int rows, int cols,
            int **m1, int **m2, int **m3) {
    int i, j, k;

    for (i=0; i<rows; i++) {
        for (j=0; j<cols; j++) {
            m3[i][j] = 0;
            for (k=0; k<cols; k++)
                m3[i][j] += m1[i][k] * m2[k][j];
        }
    }
    return(m3);
}
```

```c
int main(int argc, char *argv[]) {

    int i, n = ((argc == 2) ? atoi(argv[1]) : LENGTH);

    int **m1 = mkmatrix(SIZE, SIZE);
    int **m2 = mkmatrix(SIZE, SIZE);
    int **mm = mkmatrix(SIZE, SIZE);

    zeromatrix(SIZE, SIZE, mm);

    for (i=0; i<n; i++)
       mm = mmult(SIZE, SIZE, m1, m2, mm);

    printf("%d %d %d %d\n",
           mm[0][0], mm[2][3], mm[3][2], mm[4][4]);

    freematrix(SIZE, m1);
    freematrix(SIZE, m2);
    freematrix(SIZE, mm);
    return(0);
}
```

https://github.com/llvm-mirror/test-suite/blob/master/SingleSource/Benchmarks/Shootout/matrix.c

Simple Top-Down Analysis with perf

```
bash ~ $ perf stat -M Retiring,Bad_Speculation,Frontend_Bound,Backend_Bound a.out
1431831040 368052224 -168294912 -692581888

 Performance counter stats for 'a.out':

       2686289661      IDQ_UOPS_NOT_DELIVERED.CORE #     0.00 Frontend_Bound
                                                  #     0.55 Backend_Bound          (50.01%)
        200034632      INT_MISC.RECOVERY_CYCLES                                    (50.01%)
     135846388590      CPU_CLK_UNHALTED.THREAD                                     (50.01%)
     241410802284      UOPS_ISSUED.ANY                                             (50.01%)
      30384549081 ns   duration_time
        199807164      INT_MISC.RECOVERY_CYCLES  #     0.00 Bad_Speculation        (49.99%)
     135871753474      CPU_CLK_UNHALTED.THREAD   #     0.44 Retiring               (49.99%)
     240760535477      UOPS_RETIRED.RETIRE_SLOTS                                   (49.99%)
     241407738202      UOPS_ISSUED.ANY                                             (49.99%)
      30384549081 ns   duration_time

     30.384549081 seconds time elapsed

     30.356367000 seconds user
      0.009971000 seconds sys


bash ~ $
```

Annotated Source

```
Samples: 30K of event 'cycles', 1000 Hz, Event count (approx.): 135123213483
main  /home/amadio/a.out [Percent: local period]
  0.00 |  58:   mov  (%r12,%r9,1),%rdi
       |      m3[i][j] += m1[i][k] * m2[k][j];
  0.00 |        mov  0x0(%r13,%r9,1),%r8
       |        xor  %edx,%edx
       |        nop
       |      m3[i][j] = 0;
  0.02 |  68:   movl $0x0,(%rdi,%rdx,1)
  0.00 |        xor  %eax,%eax
  0.01 |        xor  %esi,%esi
       |      m3[i][j] += m1[i][k] * m2[k][j];
  5.37 |  73:   mov  0x0(%rbp,%rax,8),%rcx
 35.56 |        mov  (%rcx,%rdx,1),%ecx
 20.16 |        imul (%r8,%rax,4),%ecx
       |      for (k = 0; k < cols; k++)
  5.62 |        add  $0x1,%rax
       |      m3[i][j] += m1[i][k] * m2[k][j];
  9.61 |        add  %ecx,%esi
 17.01 |        mov  %esi,(%rdi,%rdx,1)
       |      for (k = 0; k < cols; k++)
  0.01 |        cmp  $0x400,%rax
  6.63 |      ↑ jne  73
Press 'h' for help on key bindings
```

Load `m1[i][k]` and `m2[k][j]` into memory and multiply

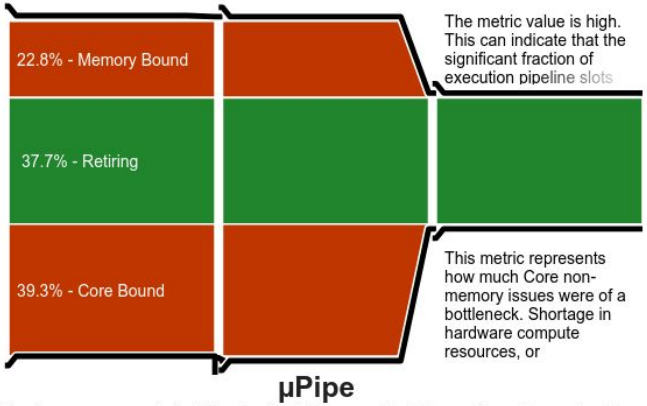
Add result into `m3[i][j]`

Loading m2 matrix elements in column major order is causing backend stalls.

Top-Down Analysis with Intel VTune Profiler

Elapsed Time: 30.547s

Clockticks:	134,784,000,000
Instructions Retired:	275,184,000,000
CPI Rate:	0.490
Retiring:	37.7%
Front-End Bound:	0.3%
Bad Speculation:	0.0%
Back-End Bound:	62.1%
Memory Bound:	22.8%
L1 Bound:	0.0%
L2 Bound:	2.6%
L3 Bound:	12.2%
Contested Accesses:	0.0%
Data Sharing:	0.0%
L3 Latency:	100.0%
SQ Full:	0.0%
DRAM Bound:	0.0%
Store Bound:	0.0%
Core Bound:	39.3%
Divider:	0.0%
Port Utilization:	24.8%
Cycles of 0 Ports Utilized:	6.2%
Cycles of 1 Port Utilized:	6.9%
Cycles of 2 Ports Utilized:	10.0%
Cycles of 3+ Ports Utilized:	20.0%
Vector Capacity Usage (FPU):	0.0%
Average CPU Frequency:	4.4 GHz
Total Thread Count:	2
Paused Time:	0s



As shown by the red arrows, the loop is being performed in column major order, which in C/C++ is not optimal, because the memory layout is row major. Therefore, we need to perform a loop inversion for the indices `j` and `k` to improve performance.

Source	Clockticks	Instructions Retired	CPI Rate	Locators: Retiring	Front-End Bound	Bad Speculation	Back-End Bound: Memory Bound: L1 Bound	L2 Bound	L3 Bound / L3 Latency	Core Bound
`int **mmult(int rows, int cols, int **m1, int **m2, int **m3) {`										
`  int i, j, k;`										
`  for (i = 0; i < rows; i++) {`										
`  for (j = 0; j < cols; j++) {`	0.0%	0.0%	0.000	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
`  m3[i][j] = 0;`	0.0%	0.0%	0.000	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
`  for (k = 0; k < cols; k++)`	35.8%	42.9%	0.414	16.1%	0.0%	0.0%	0.0%	0.0%	0.0%	13.5%
`  m3[i][j] += m1[i][k] * m2[k][j];`	64.1%	57.0%	0.558	22.1%	0.0%	0.6%	0.0%	2.5%	100.0%	24.6%
`  }`										
`  }`										
`  return(m3);`										
`}`										

Loop inversion solves the problem

```
bash ~ $ perf stat -M Retiring,Bad_Speculation,Frontend_Bound,Backend_Bound a.out
1431831040 368052224 -168294912 -692581888

 Performance counter stats for 'a.out':

        297649292      IDQ_UOPS_NOT_DELIVERED.CORE #     0.00 Frontend_Bound
                                                   #     0.03 Backend_Bound          (50.00%)
        212555840      INT_MISC.RECOVERY_CYCLES                                     (50.00%)
      71499685017      CPU_CLK_UNHALTED.THREAD                                      (50.00%)
     276063180566      UOPS_ISSUED.ANY                                              (50.00%)
      16081241308 ns   duration_time
        212615678      INT_MISC.RECOVERY_CYCLES  #     0.01 Bad_Speculation        (50.00%)
      71533499469      CPU_CLK_UNHALTED.THREAD   #     0.96 Retiring               (50.00%)
     275204536376      UOPS_RETIRED.RETIRE_SLOTS                                    (50.00%)
     276178541892      UOPS_ISSUED.ANY                                              (50.00%)
      16081241308 ns   duration_time

     16.081241308 seconds time elapsed

     16.061082000 seconds user
      0.009992000 seconds sys


bash ~ $ _
```

Now we are no longer bound by the backend. The speedup obtained was ≈2x with this change. Can we improve this result? We can parallelize the code with OpenMP, for example.

Parallel code with OpenMP gains more performance

```
bash ~ $ perf stat -M Retiring,Bad_Speculation,Frontend_Bound,Backend_Bound env OMP_NUM_THREADS=8 a.out
1431831040 368052224 -168294912 -692581888

 Performance counter stats for 'env OMP_NUM_THREADS=8 a.out':

       1164303702      IDQ_UOPS_NOT_DELIVERED.CORE #     0.00 Frontend_Bound
                                                  #     0.03 Backend_Bound           (49.99%)
        204914876      INT_MISC.RECOVERY_CYCLES                                      (49.99%)
      71650959743      CPU_CLK_UNHALTED.THREAD                                       (49.99%)
     275645117900      UOPS_ISSUED.ANY                                               (49.99%)
       2277867268 ns   duration_time
        205160954      INT_MISC.RECOVERY_CYCLES  #     0.01 Bad_Speculation         (50.07%)
      71630170542      CPU_CLK_UNHALTED.THREAD   #     0.96 Retiring                (50.07%)
     275250371320      UOPS_RETIRED.RETIRE_SLOTS                                     (50.07%)
     276156990337      UOPS_ISSUED.ANY                                               (50.07%)
       2277867268 ns   duration_time

      2.277867268 seconds time elapsed

     18.073005000 seconds user
      0.009998000 seconds sys


bash ~ $ _
```

The percentage of time spent retiring is too high. This is also indicative of a problem. Let's look again at the annotated source.

Annotated Source with perf annotate

```
Samples: 35K of event 'cycles', 1000 Hz, Event count (approx.): 142392966330
mmult._omp_fn.0  /home/amadio/a.out [Percent: local period]
Percent |       mov    %rax,%rcx
        |     for (k = 0; k < cols; k++)
        |       lea    (%rsi,%rbp,1),%r11
        |       nop
        |     for (j = 0; j < cols; j++)
        |     m3[i][j] += m1[i][k] * m2[k][j];
  0.18  | b0:   mov    (%r10),%rdi
        |       xor    %eax,%eax
        |       nop
 24.57  | b8:   mov    (%rsi),%edx
 11.31  |       imul   (%rdi,%rax,4),%edx
 63.80  |       add    %edx,(%rcx,%rax,4)
        |     for (j = 0; j < cols; j++)
        |       mov    %rax,%rdx
        |       add    $0x1,%rax
        |       cmp    %rdx,%r14
        |     ↑ jne    b8
        |     for (k = 0; k < cols; k++)
  0.09  |       add    $0x4,%rsi
        |       add    $0x8,%r10
        |       cmp    %rsi,%r11
Press 'h' for help on key bindings
```

The loop is still using scalar instructions.
We can further improve performance with vectorization.

Vectorization significantly improves performance

```
bash ~ $ # Baseline
bash ~ $ gcc -w -O2 -g matrix.c && time a.out # Using -w to avoid warning about unused pragma
1431831040 368052224 -168294912 -692581888
16.02
bash ~ $ # Parallel code with OpenMP
bash ~ $ gcc -Wall -fopenmp -O2 -g matrix.c && time a.out
1431831040 368052224 -168294912 -692581888
2.26
bash ~ $ # Parallel code with OpenMP and vectorization using AVX2
bash ~ $ gcc -Wall -fopenmp -O2 -ftree-vectorize -mavx2 -g matrix.c && time a.out
1431831040 368052224 -168294912 -692581888
0.54
bash ~ $
```

We've improved performance from ~30s down to 0.54s, not bad!
That's a speedup of about 56.3x.

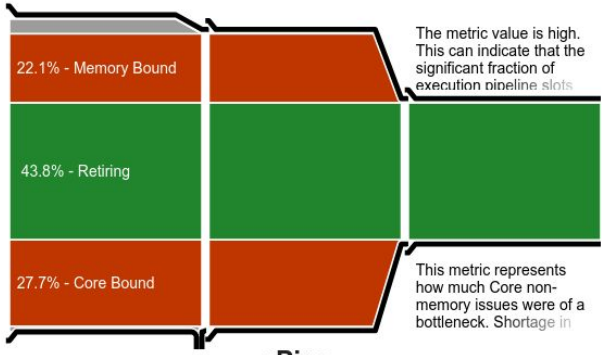
Comparison between initial and final versions

```
bash ~ $ diff -u matrix.orig.c matrix.c
--- matrix.orig.c       2022-08-15 15:12:15.457813585 +0200
+++ matrix.c    2022-08-15 15:50:32.247841744 +0200
@@ -28,14 +28,15 @@
     free(m);
 }

-int **mmult(int rows, int cols, int **m1, int **m2, int **m3) {
+int **mmult(int rows, int cols, int ** restrict m1, int ** restrict m2, int ** restrict m3) {
     int i, j, k;
+#pragma omp parallel for
     for (i = 0; i < rows; i++) {
-        for (j = 0; j < cols; j++) {
+        for (j = 0; j < cols; j++)
             m3[i][j] = 0;
-            for (k = 0; k < cols; k++)
+        for (k = 0; k < cols; k++)
+            for (j = 0; j < cols; j++)
                 m3[i][j] += m1[i][k] * m2[k][j];
-        }
     }
     return(m3);
 }
bash ~ $ _
```

Final performance summary in VTune (10x runtime)

Elapsed Time: 5.700s

Clockticks:	330,449,056,717	
Instructions Retired:	267,867,099,936	
CPI Rate:	1.234	
MUX Reliability:	0.780	
Retiring:	43.8%	of Pipeline Slots
Light Operations:	33.7%	of Pipeline Slots
Heavy Operations:	10.0%	of Pipeline Slots
Front-End Bound:	4.7%	of Pipeline Slots
Bad Speculation:	1.8%	of Pipeline Slots
Back-End Bound:	49.8%	of Pipeline Slots
Memory Bound:	22.1%	of Pipeline Slots
L1 Bound:	13.0%	of Clockticks
DTLB Overhead:	100.0%	of Clockticks
Loads Blocked by Store Forwarding:	0.0%	of Clockticks
Lock Latency:	0.0%	of Clockticks
Split Loads:	5.3%	of Clockticks
4K Aliasing:	0.9%	of Clockticks
FB Full:	0.1%	of Clockticks
L2 Bound:	5.8%	of Clockticks
L3 Bound:	6.6%	of Clockticks
DRAM Bound:	0.1%	of Clockticks
Store Bound:	0.1%	of Clockticks
Core Bound:	27.7%	of Pipeline Slots
Divider:	0.0%	of Clockticks
Port Utilization:	32.0%	of Clockticks
Cycles of 0 Ports Utilized:	25.8%	of Clockticks
Serializing Operations:	5.2%	of Clockticks
Mixing Vectors:	100.0%	of Clockticks
Cycles of 1 Port Utilized:	12.6%	of Clockticks
Cycles of 2 Ports Utilized:	14.3%	of Clockticks
Cycles of 3+ Ports Utilized:	47.7%	of Clockticks
Vector Capacity Usage (FPU):	6.2%	
Average CPU Frequency:	3.7 GHz	
Total Thread Count:	N/A*	
Paused Time:	0s	



µPipe

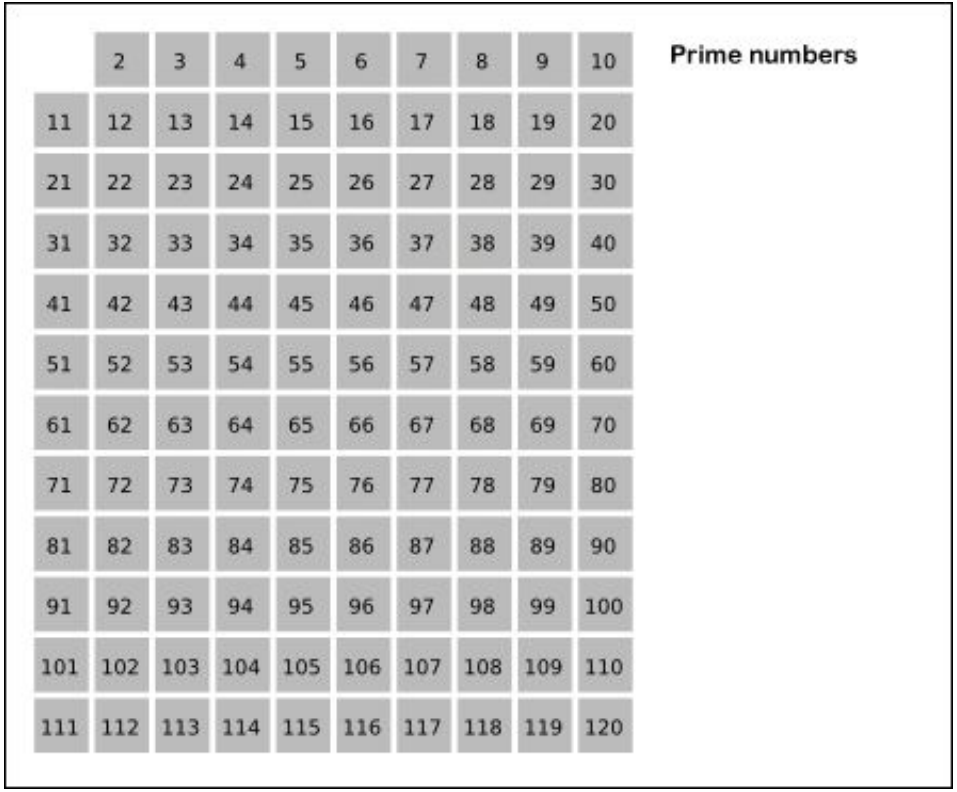
This diagram represents inefficiencies in CPU usage. Treat it as a pipe with an output flow equal to the "pipe efficiency" ratio: (Actual Instructions Retired)/(Maximum Possible Instruction Retired). If there are pipeline stalls decreasing the pipe efficiency, the pipe shape gets more narrow.

Optimization Example

Sieve of Erastosthenes

Sieve of Eratosthenes

```c
int main(int argc, char *argv[])
{
  int NUM = ((argc == 2) ? atoi(argv[1]) : 1000000);
  static char flags[8192 + 1];
  long i, k;
  int count = 0;

  while (NUM--) {
    count = 0;
    for (i = 2; i <= 8192; i++)
      flags[i] = 1;

    for (i = 2; i <= 8192; i++) {
      if (flags[i]) {
        /* remove all multiples of prime: i */
        for (k = i + i; k <= 8192; k += i)
          flags[k] = 0;
        count++;
      }
    }
  }
  printf("Count: %d\n", count);
  return 0;
}
```



https://github.com/llvm-mirror/test-suite/blob/master/SingleSource/Benchmarks/Shootout/sieve.c

Profiling line by line with perf

```c
int main(int argc, char *argv[])
{
  int NUM = ((argc == 2) ? atoi(argv[1]) : 1000000);
  static char flags[8192 + 1];
  long i, k;
  int count = 0;

  while (NUM--) {
    count = 0;
    for (i = 2; i <= 8192; i++)
      flags[i] = 1;

    for (i = 2; i <= 8192; i++) {
      if (flags[i]) {
        /* remove all multiples of prime: i */
        for (k = i + i; k <= 8192; k += i)
          flags[k] = 0;
        count++;
      }
    }
  }
  printf("Count: %d\n", count);
  return 0;
}
```

```
$ gcc -Wall -O3 -mavx2 -g -o sieve sieve.c
$ perf record --call-graph=dwarf -F50 -e cycles -- sieve
Count: 1028
[ perf record: Woken up 21 times to write data ]
[ perf record: Captured and wrote 5.28 MB perf.data (655 samples) ]
$ perf report --stdio --no-children -g none -s srcline
#
# Total Lost Samples: 0
#
# Samples: 655  of event 'cycles'
# Event count (approx.): 55672367816
#
# Overhead  Source:Line
# ........  ..............
#
    40.72%  sieve.c:16
    21.38%  sieve.c:13
    19.54%  sieve.c:14
    10.10%  sieve.c:17
     7.64%  sieve.c:18
     0.62%  sieve.c:11
     0.00%  jump_label.h:27
```

Performance statistics

```
$ perf stat -r3 -M Retiring,Bad_Speculation,Frontend_Bound,Backend_Bound -- sieve.baseline
Count: 1028
Count: 1028
Count: 1028

 Performance counter stats for 'sieve.baseline' (3 runs):

      28277988476      IDQ_UOPS_NOT_DELIVERED.CORE #     0.13 Frontend_Bound
                                                   #     0.07 Backend_Bound          ( +-  0.07% )  (49.99%)
       5503093580      INT_MISC.RECOVERY_CYCLES                                      ( +-  0.15% )  (49.99%)
      55629916812      CPU_CLK_UNHALTED.THREAD                                       ( +-  0.04% )  (49.99%)
     157421509301      UOPS_ISSUED.ANY                                               ( +-  0.02% )  (49.99%)
      12438582691 ns   duration_time                                                 ( +-  1.51% )
       5495475107      INT_MISC.RECOVERY_CYCLES  #       0.31 Bad_Speculation        ( +-  0.11% )  (50.01%)
      55643570935      CPU_CLK_UNHALTED.THREAD   #       0.50 Retiring               ( +-  0.06% )  (50.01%)
     110533393570      UOPS_RETIRED.RETIRE_SLOTS                                     ( +-  0.01% )  (50.01%)
     157469669912      UOPS_ISSUED.ANY                                               ( +-  0.04% )  (50.01%)
      12438582691 ns   duration_time                                                 ( +-  1.51% )

           12.620 +- 0.187 seconds time elapsed  ( +-  1.48% )
```

Potential Optimizations

- Initialize evens to directly with 0
- Start from i = 3 in line 13
- Check only up to the square root of 8192
- Do not iterate over even numbers again
- Skip even multiples of the primes
- Start marking composites from the squared of the current prime number
- Unroll hot loops (do more per iteration)
- Use SIMD instructions (-mavx2)
- Use simpler loop logic

```
 1 int main(int argc, char *argv[])
 2 {
 3   int NUM = ((argc == 2) ? atoi(argv[1]) : 1000000);
 4   static char flags[8192 + 1];
 5   long i, k;
 6   int count = 0;
 7
 8   while (NUM--) {
 9     count = 0;
10     for (i = 2; i <= 8192; i++)
11       flags[i] = 1;
12
13     for (i = 2; i <= 8192; i++) {
14       if (flags[i]) {
15         /* remove all multiples of prime: i */
16         for (k = i + i; k <= 8192; k += i)
17           flags[k] = 0;
18         count++;
19       }
20     }
21   }
22   printf("Count: %d\n", count);
23   return 0;
24 }
```

Sieve of Eratosthenes (Solution)

```c
int main(int argc, char *argv[])
{
  static char flags[8192 + 1];
  int NUM = ((argc == 2) ? atoi(argv[1]) : 1000000);
  long sqrt_max = sqrt(8192);
  int count;

  while (NUM--) {
    for (long i = 2; i < 8192; i += 2) {
      flags[i] = 0;     // evens are not prime
      flags[i + 1] = 1; // odd numbers might be
    } // flags[8192] doesn't need to be set, because it's even

    // since even numbers already crossed out we can:
    //  - start from i=3
    //  - iterate over odd numbers (i+=2)
    for (long i = 3; i <= sqrt_max; i += 2) {
      if (flags[i]) {
        /* remove all multiples of prime: i */
        // 1. less than i*i already marked
        // 2. only mark odd multiples (i*i+i will
        //    produce even number, which is already marked)
        for (long k = i * i; k <= 8192; k += 2 * i)
          flags[k] = 0;
      }
    }
    count = 1; // accounting for 2 is prime
    for (long i = 2; i <= 8192; i++)
      if (flags[i])
        count++;
  }
  printf("Count: %d\n", count);
  return 0;
}
```

```
$ perf stat -r3 \
    -M Retiring,Bad_Speculation,Frontend_Bound,Backend_Bound -- sieve.opt
Count: 1028
Count: 1028
Count: 1028

 Performance counter stats for 'sieve.opt' (3 runs):

       4137049582      IDQ_UOPS_NOT_DELIVERED.CORE #   0.12 Frontend_Bound
                                                   #   0.23 Backend_Bound
        181769269      INT_MISC.RECOVERY_CYCLES
       8441552456      CPU_CLK_UNHALTED.THREAD
      21158855301      UOPS_ISSUED.ANY
       1920622523 ns   duration_time
        180655671      INT_MISC.RECOVERY_CYCLES  #     0.06 Bad_Speculation
       8433127416      CPU_CLK_UNHALTED.THREAD   #     0.59 Retiring
      19962903052      UOPS_RETIRED.RETIRE_SLOTS
      21139282317      UOPS_ISSUED.ANY
       1920622523 ns   duration_time

           1.9283 +- 0.0261 seconds time elapsed  ( +-  1.36% )
```

https://easyperf.net/blog/2019/02/16/Performance-optimization-contest-1

Sieve of Eratosthenes (Further Optimization)

```
int main(int argc, char *argv[])
{
  static char flags[8192 + 1];
  int NUM = ((argc == 2) ? atoi(argv[1]) : 1000000);
  int sqrt_max = sqrt(8192);
  unsigned long long count;

  while (NUM--) {
    for (int i = 2; i < 8192; i += 2) {
      flags[i] = 0;     // evens are not prime
      flags[i + 1] = 1; // odd numbers might be
    } // flags[8192] doesn't need to be set, because it's even

    // since even numbers already crossed out we can:
    //  - start from i=3
    //  - iterate over odd numbers (i+=2)
    for (int i = 3; i <= sqrt_max; i += 2) {
      if (flags[i]) {
        /* remove all multiples of prime: i */
        // 1. less than i*i already marked
        // 2. only mark odd multiples (i*i+i will
        //    produce even number, which is already marked)
        for (int k = i * i; k <= 8192; k += 2 * i)
          flags[k] = 0;
      }
    }

    count = 1; // accounting for 2 is prime
    unsigned long long int *ptr = (unsigned long long int*) &flags;
    unsigned long long int *end = (unsigned long long int*) &flags[8192];

    while (ptr < end)
      count += _mm_popcnt_u64(*ptr++); // popcnt counts bits in a 64bit integer
  }
  printf("Count: %llu\n", count);
  return 0;
}
```

```
$ perf stat -r3 \
    -M Retiring,Bad_Speculation,Frontend_Bound,Backend_Bound -- sieve
Count: 1028
Count: 1028
Count: 1028

Performance counter stats for 'sieve' (3 runs):

        4418503187      IDQ_UOPS_NOT_DELIVERED.CORE #   0.14 Frontend_Bound
                                                    #   0.12 Backend_Bound
         180837497      INT_MISC.RECOVERY_CYCLES
        7664424596      CPU_CLK_UNHALTED.THREAD
       21842700971      UOPS_ISSUED.ANY
        1728499772 ns   duration_time
         180506496      INT_MISC.RECOVERY_CYCLES  #     0.06 Bad_Speculation
        7650876253      CPU_CLK_UNHALTED.THREAD   #     0.68 Retiring
       20710368713      UOPS_RETIRED.RETIRE_SLOTS
       21805541168      UOPS_ISSUED.ANY
        1728499772 ns   duration_time

          1.73394 +- 0.00885 seconds time elapsed  ( +-  0.51% )
```

https://easyperf.net/blog/2019/02/16/Performance-optimization-contest-1

Optimization Examples from Geant4 Simulation Toolkit

perf stat -d – overview of Geant4 initialization

```
bash ~ $ perf stat -r 3 -d -- taskset -c 0 /srv/geant4/install/gcc-10.2.0/10.6.r9-MT/bin/g4run -g ~/src/g4run/CMS.gdml -p pythia
:ttbar -e 0

 Performance counter stats for 'taskset -c 0 /srv/geant4/install/gcc-10.2.0/10.6.r9-MT/bin/g4run -g /home/amadio/src/g4run/CMS.g
dml -p pythia:ttbar -e 0' (3 runs):

         21454.06 msec task-clock                #    0.953 CPUs utilized           ( +-  0.07% )
             1520      context-switches          #    0.071 K/sec                   ( +- 48.20% )
                1      cpu-migrations            #    0.000 K/sec
           110280      page-faults               #    0.005 M/sec                   ( +-  0.06% )
      93708948749      cycles                    #    4.368 GHz                     ( +-  0.02% )  (74.96%)
        428488171      stalled-cycles-frontend   #    0.46% frontend cycles idle    ( +-  2.14% )  (74.95%)
      62140664026      stalled-cycles-backend    #   66.31% backend cycles idle     ( +-  0.04% )  (74.97%)
     129389101781      instructions              #    1.38  insn per cycle
                                                 #    0.48  stalled cycles per insn  ( +-  0.04% )  (75.02%)
      16731397508      branches                  #  779.871 M/sec                   ( +-  0.06% )  (75.06%)
        156166747      branch-misses             #    0.93% of all branches         ( +-  0.17% )  (75.09%)
      58140887925      L1-dcache-loads           # 2710.018 M/sec                   ( +-  0.10% )  (75.02%)
        685016614      L1-dcache-load-misses     #    1.18% of all L1-dcache accesses  ( +-  1.28% )  (74.93%)
   <not supported>      LLC-loads
   <not supported>      LLC-load-misses

           22.513 +- 0.637 seconds time elapsed  ( +-  2.83% )
```

Whoa, that's a lot of backend cycles idle!

Record Geant4 initialization for further analysis

```
bash ~ $ perf record -e cycles --call-graph=dwarf -F max -- taskset -c 0 /srv/geant4/install/gcc-10.2.0/10.6.r9-MT/bin/g4run --s
tats -g /srv/geant4/gdml/CMS.gdml -p pythia:ttbar -e 0 # record zero event run (initialization only) 2>/dev/null
info: Using a maximum frequency rate of 100000 Hz
       Throughput [events/min]  0
       Initialization Cost [%]  100
       Initialization Time [s]  31.0642
       Event Loop Run Time [s]  2.404e-06
       Init + Ev.Loop Time [s]  31.0642
       Max RSS Before Init [M]  38.7227
       Max RSS  After Init [M]  272.641
       Max RSS  After Loop [M]  272.641
[ perf record: Woken up 76218 times to write data ]
Warning:
Processed 3176007 events and lost 4 chunks!

Check IO/CPU overload!

[ perf record: Captured and wrote 21387.152 MB perf.data (2656290 samples) ]
bash ~ $
bash ~ $ # That's 20GB for ~30s run, dwarf generates a *lot* of data!
bash ~ $ _
```

G4{h,Mu}PairProd. account for ~40% of initialization

```
bash ~ $ perf report -q --stdio --no-children -g none --percent-limit 1 -F overhead,dso,symbol
Warning:
Processed 3176007 events and lost 4 chunks!

Check IO/CPU overload!

    29.82%  libG4processes.so          [.] G4hPairProductionModel::ComputeDMicroscopicCrossSection
    10.38%  libG4processes.so          [.] G4MuPairProductionModel::ComputeDMicroscopicCrossSection
    10.30%  libG4processes.so          [.] G4ElasticHadrNucleusHE::HadrNucDifferCrSec
     3.76%  libG4processes.so          [.] G4eBremsstrahlungRelModel::ComputeLPMfunctions
     3.72%  libG4processes.so          [.] G4hBremsstrahlungModel::ComputeDMicroscopicCrossSection
     3.27%  libG4global.so             [.] G4PhysicsVector::Value
     1.92%  libG4geometry.so           [.] G4Region::BelongsTo
     1.87%  libG4processes.so          [.] G4MuBremsstrahlungModel::ComputeDMicroscopicCrossSection
     1.45%  libG4processes.so          [.] G4SeltzerBergerModel::ComputeDXSectionPerAtom
     1.45%  libG4processes.so          [.] G4ProductionCutsTable::ScanAndSetCouple
     1.33%  libpythia8.so              [.] Pythia8::NNPDF::polint
     1.10%  libG4processes.so          [.] G4eBremsstrahlungRelModel::ComputeXSectionPerAtom
     1.05%  libG4geometry.so           [.] G4LogicalVolume::GetMaterial


bash ~ $  # Looks like ~40% of the time is spent in G4{Mu,h}PairProductionModel::ComputeDMicroscopicCrossSection
bash ~ $  # G4hPairProductionModel::ComputeDMicroscopicCrossSection actually is the same as in G4MuPairProductionModel
bash ~ $  # Note that no inlined functions are shown above, as we asked for no callchain information_
```

Top 3 models account for ~60% of backend stalls

```
bash ~ $ # Same as previous report, but for stalled-cycles-backend
bash ~ $ perf report -q --stdio --no-children -g none --percent-limit 1 -F overhead,dso,symbol
Warning:
Processed 3080687 events and lost 104 chunks!

Check IO/CPU overload!

    33.59%  libG4processes.so          [.] G4hPairProductionModel::ComputeDMicroscopicCrossSection
    14.26%  libG4processes.so          [.] G4ElasticHadrNucleusHE::HadrNucDifferCrSec
    11.97%  libG4processes.so          [.] G4MuPairProductionModel::ComputeDMicroscopicCrossSection
     4.98%  libG4processes.so          [.] G4eBremsstrahlungRelModel::ComputeLPMfunctions
     4.90%  libG4processes.so          [.] G4hBremsstrahlungModel::ComputeDMicroscopicCrossSection
     3.80%  libG4global.so             [.] G4PhysicsVector::Value
     2.46%  libG4processes.so          [.] G4MuBremsstrahlungModel::ComputeDMicroscopicCrossSection
     2.09%  libG4geometry.so           [.] G4Region::BelongsTo
     1.65%  libG4processes.so          [.] G4SeltzerBergerModel::ComputeDXSectionPerAtom
     1.36%  libG4processes.so          [.] G4eBremsstrahlungRelModel::ComputeXSectionPerAtom
     1.07%  libG4processes.so          [.] G4LossTableBuilder::BuildRangeTable
     1.01%  libpythia8.so              [.] Pythia8::NNPDF::polint


bash ~ $ # The top 3 models are responsible for ~60% of all stalled cycles
bash ~ $ # However, L1 cache misses are about ~1.2%, so this is likely *not* a memory access problem_
```

How to improve performance?

- Look for pair production model in Geant4 Physics Manual

Formulae

The differential cross section for electron pair production by muons $\sigma(Z,A,E,\epsilon)$ can be written as :

$$\sigma(Z,A,E,\epsilon) = \frac{4}{3\pi}\,\frac{Z(Z+\zeta)}{A}\,N_A\,(\alpha r_0)^2\,\frac{1-v}{\epsilon}\int_0^{\rho_{\max}} G(Z,E,v,\rho)\,d\rho, \qquad (154)$$

where

$$G(Z,E,v,\rho) = \Phi_e + (m/\mu)^2\Phi_\mu,$$
$$\Phi_{e,\mu} = B_{e,\mu}L'_{e,\mu}$$

and

$$\Phi_{e,\mu} = 0 \quad \text{whenever} \quad \Phi_{e,\mu} < 0.$$

B_e and B_μ do not depend on Z, A, and are given by

$$B_e = [(2+\rho^2)(1+\beta)+\xi(3+\rho^2)]\ln\left(1+\frac{1}{\xi}\right)+\frac{1-\rho^2-\beta}{1+\xi}-(3+\rho^2);$$
$$\approx \frac{1}{2\xi}\,[(3-\rho^2)+2\beta(1+\rho^2)] \quad \text{for} \quad \xi \geq 10^3;$$

$$B_\mu = \left[(1+\rho^2)\left(1+\frac{3\beta}{2}\right)-\frac{1}{\xi}(1+2\beta)(1-\rho^2)\right]\ln(1+\xi)+\frac{\xi(1-\rho^2-\beta)}{1+\xi}+(1+2\beta)(1-\rho^2);$$
$$B_\mu \approx \frac{\xi}{2}\,[(5-\rho^2)+\beta(3+\rho^2)] \quad \text{for} \quad \xi \leq 10^{-3};$$

Also,

$$\xi = \frac{\mu^2 v^2}{4m^2}\,\frac{(1-\rho^2)}{(1-v)}; \quad \beta = \frac{v^2}{2(1-v)};$$

$$L'_e = \ln\frac{A^* Z^{-1/3}\sqrt{(1+\xi)(1+Y_e)}}{1+\frac{2m\sqrt{e}A^* Z^{-1/3}(1+\xi)(1+Y_e)}{Ev(1-\rho^2)}} - \frac{1}{2}\ln\left[1+\left(\frac{3mZ^{1/3}}{2\mu}\right)^2(1+\xi)(1+Y_e)\right];$$

$$L'_\mu = \ln\frac{(\mu/m)A^* Z^{-1/3}\sqrt{(1+1/\xi)(1+Y_\mu)}}{1+\frac{2m\sqrt{e}A^* Z^{-1/3}(1+\xi)(1+Y_\mu)}{Ev(1-\rho^2)}} - \ln\left[\frac{3}{2}Z^{1/3}\sqrt{(1+1/\xi)(1+Y_\mu)}\right].$$

For faster computing, the expressions for $L'_{e,\mu}$ are further algebraically transformed. The functions $L'_{e,\mu}$ include the nuclear size correction [KP71] in comparison with parameterization [KP70] :

$$Y_e = \frac{5-\rho^2+4\,\beta\,(1+\rho^2)}{2(1+3\beta)\ln(3+1/\xi)-\rho^2-2\beta(2-\rho^2)};$$
$$Y_\mu = \frac{4+\rho^2+3\,\beta\,(1+\rho^2)}{(1+\rho^2)(\frac{3}{2}+2\beta)\ln(3+\xi)+1-\frac{3}{2}\,\rho^2};$$
$$\rho_{\max} = [1-6\mu^2/E^2(1-v)]\sqrt{1-4m/Ev}.$$

Comment on the Calculation of the Integral $\int d\rho$ in Eq.(154)

The integral $\int_0^{\rho_{\max}} G(Z,E,v,\rho)\,d\rho$ is computed with the substitutions:

$$t = \ln(1-\rho),$$
$$1-\rho = \exp(t),$$
$$1+\rho = 2-\exp(t),$$
$$1-\rho^2 = e^t\,(2-e^t).$$

After that,

$$\int_0^{\rho_{\max}} G(Z,E,v,\rho)\,d\rho = \int_{t_{\min}}^{0} G(Z,E,v,\rho)\,e^t\,dt, \qquad (155)$$

where

$$t_{\min} = \ln\frac{\frac{4m}{\epsilon}+\frac{12\mu^2}{EE'}\left(1-\frac{4m}{\epsilon}\right)}{1+\left(1-\frac{6\mu^2}{EE'}\right)\sqrt{1-\frac{4m}{\epsilon}}}.$$

To compute the integral of Eq.(155) with an accuracy better than 0.5%, Gaussian quadrature with $N = 8$ points is sufficient.

The function $\zeta(E,Z)$ in Eq.(154) serves to take into account the process on atomic electrons (inelastic atomic form factor contribution). To treat the energy loss balance correctly, the following approximation, which is an algebraic transformation of the expression in Ref. [Kel98], is used:

$$\zeta(E,Z) = \frac{0.073\ln\frac{E/\mu}{1+\gamma_1 Z^{2/3}E/\mu}-0.26}{0.058\ln\frac{E/\mu}{1+\gamma_2 Z^{1/3}E/\mu}-0.14};$$
$$= 0 \quad \text{if the numerator is negative.}$$

For E $\leq 35\,\mu$, $\zeta(E,Z) = 0$. Also $\gamma_1 = 1.95\cdot 10^{-5}$ and $\gamma_2 = 5.30\cdot 10^{-5}$.

The above formulae make use of the Thomas-Fermi model which is not good enough for light elements. For hydrogen ($Z = 1$) the following parameters must be changed:

- $A^* = 183 \;\Rightarrow\; 202.4$;
- $\gamma_1 = 1.95\cdot 10^{-5} \;\Rightarrow\; 4.4\cdot 10^{-5}$;
- $\gamma_2 = 5.30\cdot 10^{-5} \;\Rightarrow\; 4.8\cdot 10^{-5}$.

How to improve performance?

- Look for pair production model in Geant4 Physics Manual
  - Rework expressions for cross sections with pencil/paper to reduce arithmetic operations
- Avoid unnecessary calls to G4Log function when calculating zeta
- Remove data dependencies
  - Break up large for loop into several smaller for loops
  - Compute together things that don’t depend on each other
  - Hide latency from divisions
  - When calling G4Log, input is already available
  - Move common expressions out of for loop
- Remove code duplication from the two classes with essentially the same version of this function by inheriting the base version in the derived class

Result of expensive call to G4Log not always used

```
// zeta calculation
G4double bbb,g1,g2;
if( Z < 1.5 ) { bbb = bbbh ; g1 = g1h ; g2 = g2h ; }
else          { bbb = bbbtf; g1 = g1tf; g2 = g2tf; }

G4double zeta = 0;
G4double zeta1 =
  0.073*G4Log(totalEnergy/(particleMass+g1*z23*totalEnergy))-0.26;
if ( zeta1 > 0.)
{
  G4double zeta2 =
    0.058*G4Log(totalEnergy/(particleMass+g2*z13*totalEnergy))-0.14;
  zeta  = zeta1/zeta2 ;
}

G4double z2 = Z*(Z+zeta);
G4double screen0 = 2.*electron_mass_c2*sqrte*bbb/(z13*pairEnergy);
G4double a0 = totalEnergy*residEnergy;
G4double a1 = pairEnergy*pairEnergy/a0;
G4double bet = 0.5*a1;
G4double xi0 = 0.25*massratio2*a1;
G4double del = c8/a0;
_                                                    357,1         48%
```

this if statement for treating hydrogen differently can be replaced by branchless code (index based on boolean result)

this call to G4Log can be avoided when `zeta1 <= 0.0`

result of division used right away

Avoid calling G4Log by replacing condition

```
@@ -350,14 +350,15 @@ G4double G4MuPairProductionModel::ComputeDMicroscopicCrossSection(
   if( Z < 1.5 ) { bbb = bbbh ; g1 = g1h ; g2 = g2h ; }
   else          { bbb = bbbtf; g1 = g1tf; g2 = g2tf; }

-  G4double zeta = 0;
-  G4double zeta1 =
-    0.073*G4Log(totalEnergy/(particleMass+g1*z23*totalEnergy))-0.26;
-  if ( zeta1 > 0.)
+  G4double zeta = 0.0;
+  G4double z1exp = totalEnergy / (particleMass + g1*z23*totalEnergy);
+
+  // 35.221047195922 is the root of zeta1(x) = 0.073 * log(x) - 0.26, so the
+  // condition below is the same as zeta1 > 0.0, but without calling log(x)
+  if (z1exp > 35.221047195922)
   {
-    G4double zeta2 =
-      0.058*G4Log(totalEnergy/(particleMass+g2*z13*totalEnergy))-0.14;
-    zeta  = zeta1/zeta2 ;
+    G4double z2exp = totalEnergy / (particleMass + g2*z13*totalEnergy);
+    zeta = (0.073 * G4Log(z1exp) - 0.26) / (0.058 * G4Log(z2exp) - 0.14);
   }

   G4double z2 = Z*(Z+zeta);
lines 2339-2361/3238 74%
```

We can avoid calling G4Log by replacing the condition with an equivalent one for the input argument of G4Log.

Return early to avoid unnecessary divisions

```
commit 1c0b893bef0fe6f32fd8593989882239628262a9
Author: Guilherme Amadio <amadio@cern.ch>

    G4MuPairProductionModel: make early return condition more explicit

diff --git a/source/processes/electromagnetic/muons/src/G4MuPairProductionModel.cc b/source/processes/electromagnetic/muons/src/G4MuPairProductionMode
l.cc
index f7ba48dc7d..e0b7c550f0 100644
--- a/source/processes/electromagnetic/muons/src/G4MuPairProductionModel.cc
+++ b/source/processes/electromagnetic/muons/src/G4MuPairProductionModel.cc
@@ -321,6 +321,9 @@ G4double G4MuPairProductionModel::ComputeDMicroscopicCrossSection(
   static const G4double g1h  = 4.4e-5 ;
   static const G4double g2h  = 4.8e-5 ;

+  if (pairEnergy <= 4.0 * electron_mass_c2)
+    return 0.0;
+
   G4double totalEnergy  = tkin + particleMass;
   G4double residEnergy  = totalEnergy - pairEnergy;
   G4double massratio    = particleMass/electron_mass_c2;
@@ -334,7 +337,6 @@ G4double G4MuPairProductionModel::ComputeDMicroscopicCrossSection(
   G4double c8 = 6.*particleMass*particleMass;
   G4double alf = c7/pairEnergy;
   G4double a3 = 1. - alf;
-  if (a3 <= 0.) { return cross; }

   // zeta calculation
   G4double bbb,g1,g2;

lines 685-712/3673 18%
```

Moving the early return up reduces unnecessary divisions.
Also a3 <= 0 is harder to understand than the new form.

Big loop with many data dependencies

```
for (G4int i=0; i<8; ++i)
{
  G4double a4 = G4Exp(tmn*xgi[i]);      // a4 = (1.-asymmetry)
  G4double a5 = a4*(2.-a4) ;
  G4double a6 = 1.-a5 ;
  G4double a7 = 1.+a6 ;
  G4double a9 = 3.+a6 ;
  G4double xi = xi0*a5 ;
  G4double xii = 1./xi ;
  G4double xi1 = 1.+xi ;
  G4double screen = screen0*xi1/a5 ;
  G4double yeu = 5.-a6+4.*bet*a7 ;
  G4double yed = 2.*(1.+3.*bet)*G4Log(3.+xii)-a6-a1*(2.-a6) ;
  G4double ye1 = 1.+yeu/yed ;
  G4double ale = G4Log(bbb/z13*sqrt(xi1*ye1)/(1.+screen*ye1)) ;
  G4double cre = 0.5*G4Log(1.+2.25*z23*xi1*ye1/massratio2) ;
  G4double be;

  if (xi <= 1.e3) {
    be = ((2.+a6)*(1.+bet)+xi*a9)*G4Log(1.+xii)+(a5-bet)/xi1-a9;
  } else {
    be = (3.-a6+a1*a7)/(2.*xi);
  }
  G4double fe = (ale-cre)*be;
  if ( fe < 0.) fe = 0. ;

  G4double ymu = 4.+a6 +3.*bet*a7 ;
  G4double ymd = a7*(1.5+a1)*G4Log(3.+xi)+1.-1.5*a6 ;
  G4double ym1 = 1.+ymu/ymd ;
```

363,26 57%

Data dependencies between arithmetic operations can create execution latency even without cache misses.

Breaking up long loops into smaller parts makes it possible to hide some of the latency from divisions and math function calls with instruction level parallelism.

Use perf annotate to find hottest parts of the code

```
Samples: 2M of event 'cycles', 100000 Hz, Event count (approx.): 113912086093
G4hPairProductionModel::ComputeDMicroscopicCrossSection  /srv/geant4/install/gcc-10.2.0/10.6.r9-MT/lib64/libG4processes.so [Perc
  0.46          vdivsd       %xmm0,%xmm7,%xmm7
                G4double ale=G4Log(bbb/z13*sqrt(xi1*ye1)/(1.+screen*ye1)) ;
  2.33          vmovsd       -0x88(%rbp),%xmm0
  0.00          vdivsd       0x140(%r10),%xmm0,%xmm15
                G4double ye1 = 1.+yeu/yed ;
                vaddsd       %xmm2,%xmm7,%xmm7
                G4double ale=G4Log(bbb/z13*sqrt(xi1*ye1)/(1.+screen*ye1)) ;
  0.15          vmulsd       %xmm7,%xmm11,%xmm0
  0.78          vucomisd     %xmm0,%xmm3
  0.32        → ja           649179 <G4hPairProductionModel::ComputeDMicroscopicCrossSection(double, double, double)+0x13e9>
  0.31          vsqrtsd      %xmm0,%xmm0,%xmm0
  2.48          vmulsd       %xmm0,%xmm15,%xmm0
  0.45          vmovsd       -0x50(%rbp),%xmm15
                vfmadd132sd  %xmm7,%xmm2,%xmm15
                vdivsd       %xmm15,%xmm0,%xmm15
                G4LogConsts::dp2uint64(double):
                tmp.d = x;
  1.99          vmovq        %xmm15,%rax
                G4LogConsts::getMantExponent(double, double&):
                uint64_t le = (n >> 52);
  0.15          vmovq        %xmm15,%rdx
Press 'h' for help on key bindings
```

hottest instructions

Big loop with many data dependencies (cont.)

```
  G4double tmn = G4Log(tmnexp);
  G4double sum = 0.;

  // Gaussian integration in ln(1-ro)
  for (G4int i=0; i<8; ++i)
  {
    G4double a4 = G4Exp(tmn*xgi[i]);
    G4double a5 = a4*(2.-a4) ;
    G4double a6 = 1.-a5 ;
    G4double a7 = 1.+a6 ;
    G4double a9 = 3.+a6 ;
    G4double xi = xi0*a5 ;
    G4double xii = 1./xi ;
    G4double xi1 = 1.+xi ;
    G4double screen = screen0*xi1/a5 ;
    G4double yeu = 5.-a6+4.*bet*a7 ;
    G4double yed = 2.*(1.+3.*bet)*G4Log(3.+xii)-a6-a1*(2.-a6) ;
    G4double ye1 = 1.+yeu/yed ;
    G4double ale = G4Log(bbb/z13*sqrt(xi1*ye1)/(1.+screen*ye1)) ;
    G4double cre = 0.5*G4Log(1.+2.25*z23*xi1*ye1/massratio2) ;
    G4double be;
"/srv/geant4/src/geant4-10.6.r9/source/processes/electromagnetic/muons/src/G4MuPairProductionModel.cc" 712 lines --50%--
Press ENTER or type command to continue
```

Observations:

- Big for loop with fixed iteration count, but no vectorization
  - Loop has common expressions that can be moved out
- Variable names make code hard to understand
- Many data dependencies reduce parallelism
  - Results of divisions and sqrt used immediately
    - Result of `tmn = G4Log(tmnexp)` used immediately
    - Results of divisions and sqrt used inside call to G4Log
  - G4Log is called (and inlined!) 4 times just here
    - G4Log **inlined 10 times** just in this function!

hottest source lines shown by perf annotate

Break large loop to hide latency from divisions

```
-  for (G4int i=0; i<8; ++i)
+  G4double rho[8];
+  G4double rho2[8];
+  G4double xi[8];
+  G4double xi1[8];
+  G4double xii[8];
+
+  for (G4int i = 0; i < 8; ++i)
   {
-    G4double rho = G4Exp(tmn*xgi[i]) - 1.0; // rho = -asymmetry
-    G4double rho2 = rho * rho;
-    G4double xi = xi0*(1.0-rho2) ;
-    G4double xii = 1./xi ;
-    G4double xi1 = 1.+xi ;
-    G4double screen = screen0*xi1/(1.0-rho2) ;
-    G4double yeu = 5.-rho2+4.*bet*(1.0+rho2) ;
-    G4double yed = 2.*(1.+3.*bet)*G4Log(3.+xii)-rho2-a1*(2.-rho2) ;
+    rho[i] = G4Exp(tmn*xgi[i]) - 1.0; // rho = -asymmetry
+    rho2[i] = rho[i] * rho[i];
+    xi[i] = xi0*(1.0-rho2[i]);
+    xi1[i] = 1.0 + xi[i];
+    xii[i] = 1.0 / xi[i];
+  }
+
+  for (G4int i = 0; i < 8; ++i)
+  {
+    G4double screen = screen0*xi1[i]/(1.0-rho2[i]) ;
+    G4double yeu = 5.-rho2[i]+4.*bet*(1.0+rho2[i]) ;
+    G4double yed = 2.*(1.+3.*bet)*G4Log(3.+xii[i])-rho2[i]-a1*(2.-rho2[i]) ;
lines 1088-1116
```

Data dependencies between arithmetic operations can create execution latency even without cache misses.

Breaking up long loops into smaller parts makes it possible to hide some of the latency from divisions and math function calls with instruction level parallelism.

Reduce Code Duplication

```
-#include "G4PhysicalConstants.hh"
-#include "G4Log.hh"
-#include "G4Exp.hh"
-
-using namespace std;

 G4hPairProductionModel::G4hPairProductionModel(const G4ParticleDefinition* p,
                                                const G4String& nam)
@@ -65,114 +60,3 @@ G4hPairProductionModel::~G4hPairProductionModel()
 {}

 //....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......
-
-G4double G4hPairProductionModel::ComputeDMicroscopicCrossSection(
-                                        G4double tkin,
-                                        G4double Z,
-                                        G4double pairEnergy)
-//  differential cross section
-{
-  static const G4double bbbtf= 183. ;
-  static const G4double bbbh = 202.4 ;
-  static const G4double g1tf = 1.95e-5 ;
-  static const G4double g2tf = 5.3e-5 ;
-  static const G4double g1h  = 4.4e-5 ;
-  static const G4double g2h  = 4.8e-5 ;
-
-  G4double totalEnergy  = tkin + particleMass;
-  G4double residEnergy  = totalEnergy - pairEnergy;
-  G4double massratio    = particleMass/electron_mass_c2 ;
lines 511-539
```

This was a copy of G4MuPairProductionModel::ComputeDMicroscopicCrossSection. We can keep only the copy from the base class.

From ~40% of initialization to ~27%, not bad!

```
bash ~ $ perf report -q --stdio --no-children -g none --percent-limit 1 -F overhead,dso,symbol
Warning:
Processed 2720752 events and lost 7 chunks!

Check IO/CPU overload!

    26.88%  libG4processes.so          [.] G4MuPairProductionModel::ComputeDMicroscopicCrossSection
    12.60%  libG4processes.so          [.] G4ElasticHadrNucleusHE::HadrNucDifferCrSec
     4.60%  libG4processes.so          [.] G4eBremsstrahlungRelModel::ComputeLPMfunctions
     4.53%  libG4processes.so          [.] G4hBremsstrahlungModel::ComputeDMicroscopicCrossSection
     3.96%  libG4global.so             [.] G4PhysicsVector::Value
     2.33%  libG4geometry.so           [.] G4Region::BelongsTo
     2.26%  libG4processes.so          [.] G4MuBremsstrahlungModel::ComputeDMicroscopicCrossSection
     1.78%  libG4processes.so          [.] G4SeltzerBergerModel::ComputeDXSectionPerAtom
     1.67%  libG4processes.so          [.] G4ProductionCutsTable::ScanAndSetCouple
     1.63%  libpythia8.so              [.] Pythia8::NNPDF::polint
     1.39%  libG4processes.so          [.] G4eBremsstrahlungRelModel::ComputeXSectionPerAtom
     1.30%  libG4geometry.so           [.] G4LogicalVolume::GetMaterial
     1.06%  libG4processes.so          [.] G4LossTableBuilder::BuildRangeTable
     1.01%  libc-2.32.so               [.] malloc


bash ~ $
```

Revisiting overview of Geant4 initialization (before)

```
bash ~ $ perf stat -r 3 -d -- taskset -c 0 /srv/geant4/install/gcc-10.2.0/10.6.r9-MT/bin/g4run -g ~/src/g4run/CMS.gdml -p pythia
:ttbar -e 0

 Performance counter stats for 'taskset -c 0 /srv/geant4/install/gcc-10.2.0/10.6.r9-MT/bin/g4run -g /home/amadio/src/g4run/CMS.g
dml -p pythia:ttbar -e 0' (3 runs):

         21454.06 msec task-clock                #    0.953 CPUs utilized           ( +-  0.07% )
             1520      context-switches          #    0.071 K/sec                   ( +- 48.20% )
                1      cpu-migrations            #    0.000 K/sec
           110280      page-faults               #    0.005 M/sec                   ( +-  0.06% )
      93708948749      cycles                    #    4.368 GHz                     ( +-  0.02% )  (74.96%)
        428488171      stalled-cycles-frontend   #    0.46% frontend cycles idle    ( +-  2.14% )  (74.95%)
      62140664026      stalled-cycles-backend    #   66.31% backend cycles idle     ( +-  0.04% )  (74.97%)
     129389101781      instructions              #    1.38  insn per cycle
                                                 #    0.48  stalled cycles per insn  ( +-  0.04% )  (75.02%)
      16731397508      branches                  #  779.871 M/sec                   ( +-  0.06% )  (75.06%)
        156166747      branch-misses             #    0.93% of all branches         ( +-  0.17% )  (75.09%)
      58140887925      L1-dcache-loads           # 2710.018 M/sec                   ( +-  0.10% )  (75.02%)
        685016614      L1-dcache-load-misses     #    1.18% of all L1-dcache accesses  ( +-  1.28% )  (74.93%)
   <not supported>      LLC-loads
   <not supported>      LLC-load-misses

           22.513 +- 0.637 seconds time elapsed  ( +-  2.83% )
```

Revisiting overview of Geant4 initialization (after)

```
bash ~ $ perf stat -r 3 -d -- taskset -c 0 /srv/geant4/install/gcc-10.2.0/10.6.r10-MT/bin/g4run -g ~/src/g4run/CMS.gdml -p pythi
a:ttbar -e 0

 Performance counter stats for 'taskset -c 0 /srv/geant4/install/gcc-10.2.0/10.6.r10-MT/bin/g4run -g /home/amadio/src/g4run/CMS.
gdml -p pythia:ttbar -e 0' (3 runs):

         17517.74 msec task-clock                #    0.982 CPUs utilized            ( +-  0.20% )
              282      context-switches          #    0.016 K/sec                    ( +-  1.13% )
                1      cpu-migrations            #    0.000 K/sec
           110368      page-faults               #    0.006 M/sec                    ( +-  0.01% )
      76145083799      cycles                    #    4.347 GHz                      ( +-  0.11% )  (74.98%)
        428362729      stalled-cycles-frontend   #    0.56% frontend cycles idle     ( +-  0.83% )  (74.98%)
      45042101836      stalled-cycles-backend    #   59.15% backend cycles idle      ( +-  0.14% )  (74.99%)
     125423223087      instructions              #    1.65  insn per cycle
                                                 #    0.36  stalled cycles per insn  ( +-  0.04% )  (74.96%)
      16930383319      branches                  #  966.471 M/sec                    ( +-  0.23% )  (74.94%)
        157044670      branch-misses             #    0.93% of all branches          ( +-  0.02% )  (75.04%)
      57470258214      L1-dcache-loads           # 3280.690 M/sec                    ( +-  0.09% )  (75.09%)
        675844171      L1-dcache-load-misses     #    1.18% of all L1-dcache accesses  ( +-  0.35% )  (75.02%)
   <not supported>      LLC-loads
   <not supported>      LLC-load-misses

          17.8476 +- 0.0237 seconds time elapsed  ( +-  0.13% )
```

G4Log inlined many times, maybe that's a problem?

```
bash geant4-10.6.r9 $ git grep 'G4Log(' source/ | head
source/global/HEPNumerics/src/G4JTPolynomialSolver.cc:      l      = (G4int)(G4Log(sc) / G4Log(base) + 0.5);
source/global/HEPNumerics/src/G4JTPolynomialSolver.cc:    x = G4Exp((G4Log(-pt[n]) - G4Log(pt[0])) / (G4double) n);
source/global/management/include/G4Log.hh:inline G4double G4Log(G4double x)
source/global/management/include/G4PhysicsVector.icc:    bin = size_t(std::max(G4Log(theEnergy) * invdBin - baseBin, 0.0));
source/global/management/include/G4Pow.hh:    res = G4Log(a);
source/global/management/include/G4Pow.hh:    res = G4Log(a);
source/global/management/src/G4PhysicsLogVector.cc:  invdBin = 1. / (G4Log(theEmax / theEmin) / (G4double) (numberOfNodes - 1));
source/global/management/src/G4PhysicsLogVector.cc:  baseBin = G4Log(theEmin) * invdBin;
source/global/management/src/G4PhysicsLogVector.cc:    invdBin = 1. / G4Log(binVector[1] / edgeMin);
source/global/management/src/G4PhysicsLogVector.cc:    baseBin = G4Log(edgeMin) * invdBin;
bash geant4-10.6.r9 $ git grep -c 'G4Log(' source/ | head
source/global/HEPNumerics/src/G4JTPolynomialSolver.cc:2
source/global/management/include/G4Log.hh:1
source/global/management/include/G4PhysicsVector.icc:1
source/global/management/include/G4Pow.hh:2
source/global/management/src/G4PhysicsLogVector.cc:6
source/global/management/src/G4Pow.cc:4
source/materials/include/G4IonisParamMat.hh:1
source/materials/src/G4DensityEffectCalculator.cc:4
source/materials/src/G4Element.cc:3
source/materials/src/G4IonisParamMat.cc:11
bash geant4-10.6.r9 $ git grep -c 'G4Log(' source/processes/ | awk -F : '{sum += $2} END { print sum }'
932
```

G4Log inlined at least 932 times in physics processes.
Makes libG4processes.so 1–2% larger because of this.
(release ~300K larger / debug 10MB larger)

G4Log inlined many times, but it's not the problem

G4Log function inlined

```
bash ~ $ perf stat -r 5 -- taskset -c 0 /srv/geant4/install/gcc-10.2.0/master-MT/bi
n/g4run -g /srv/geant4/gdml/CMS.gdml -p pythia:ttbar -e 0 2>&1 | cut -b -83

 Performance counter stats for 'taskset -c 0 /srv/geant4/install/gcc-10.2.0/master-

         17290.23 msec task-clock                #    0.961 CPUs utilized
              291      context-switches          #    0.017 K/sec
                1      cpu-migrations            #    0.000 K/sec
           111152      page-faults               #    0.006 M/sec
      76427613719      cycles                    #    4.420 GHz
        448704342      stalled-cycles-frontend   #    0.59% frontend cycles idle
      45431920601      stalled-cycles-backend    #   59.44% backend cycles idle
     125558491994      instructions              #    1.64  insn per cycle
                                                 #    0.36  stalled cycles per ins
      17019334780      branches                  #  984.333 M/sec
        157955893      branch-misses             #    0.93% of all branches

           17.999 +- 0.392 seconds time elapsed  ( +-  2.18% )

bash ~ $ _
```

G4Log function *not* inlined

```
bash ~ $ perf stat -r 5 -- taskset -c 0 /srv/geant4/install/gcc-10.2.0/amadio-MT/bi
n/g4run -g /srv/geant4/gdml/CMS.gdml -p pythia:ttbar -e 0 2>&1 | cut -b -83

 Performance counter stats for 'taskset -c 0 /srv/geant4/install/gcc-10.2.0/amadio-

         17867.15 msec task-clock                #    0.982 CPUs utilized
              301      context-switches          #    0.017 K/sec
                1      cpu-migrations            #    0.000 K/sec
           111092      page-faults               #    0.006 M/sec
      79215774857      cycles                    #    4.434 GHz
        440356271      stalled-cycles-frontend   #    0.56% frontend cycles idle
      44536128217      stalled-cycles-backend    #   56.22% backend cycles idle
     134984466012      instructions              #    1.70  insn per cycle
                                                 #    0.33  stalled cycles per ins
      19095299031      branches                  # 1068.738 M/sec
        166932047      branch-misses             #    0.87% of all branches

           18.1884 +- 0.0134 seconds time elapsed  ( +-  0.07% )

bash ~ $
```

No big difference, so problem is not due to code bloat

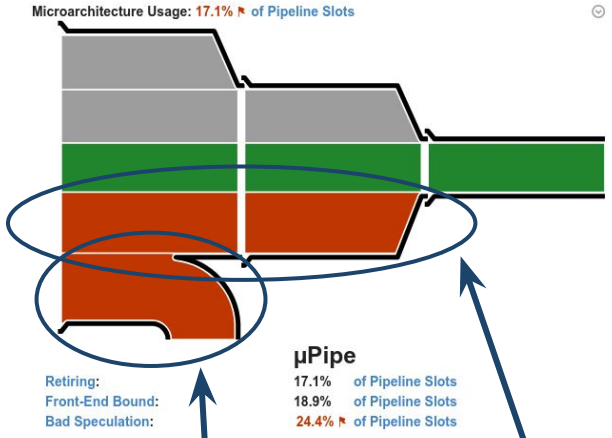
What happens if we use std::log and std::exp?

```
bash ~ $ perf stat -r 3 -d -- taskset -c 0 /srv/geant4/install/gcc-10.2.0/amadio-MT/bin/g4run -g ~/src/g4run/CMS.gdml -p pythia:
ttbar -e 0

 Performance counter stats for 'taskset -c 0 /srv/geant4/install/gcc-10.2.0/amadio-MT/bin/g4run -g /home/amadio/src/g4run/CMS.gd
ml -p pythia:ttbar -e 0' (3 runs):

         15773.13 msec task-clock                #    0.972 CPUs utilized           ( +-  0.08% )
             1284      context-switches          #    0.081 K/sec                   ( +- 40.16% )
                1      cpu-migrations            #    0.000 K/sec
           111287      page-faults               #    0.007 M/sec                   ( +-  0.05% )
      68324629875      cycles                    #    4.332 GHz                     ( +-  0.06% )  (75.04%)
        373986440      stalled-cycles-frontend   #    0.55% frontend cycles idle    ( +-  1.07% )  (74.95%)
      36522879398      stalled-cycles-backend    #   53.45% backend cycles idle     ( +-  0.26% )  (75.00%)
     137884394122      instructions              #    2.02  insn per cycle
                                                #    0.26  stalled cycles per insn  ( +-  0.05% )  (74.99%)
      19884639889      branches                  # 1260.666 M/sec                    ( +-  0.11% )  (74.98%)
         86643525      branch-misses             #    0.44% of all branches         ( +-  0.68% )  (74.98%)
      58388354221      L1-dcache-loads           # 3701.762 M/sec                    ( +-  0.08% )  (74.98%)
        694825388      L1-dcache-load-misses     #    1.19% of all L1-dcache accesses  ( +-  0.42% )  (75.09%)
   <not supported>      LLC-loads
   <not supported>      LLC-load-misses

          16.2350 +- 0.0809 seconds time elapsed  ( +-  0.50% )
```

Extra ~10% speedup! Could make sense to use std::log at initialization only.

Example of Bad Speculation

Be careful with what you assume the compiler can optimize for you.



Source Line ▲	Source	Clockticks	Instructions Retired	CPI Rate	Locators					
					Retiring	Front-End Bound	Bad Speculation	Back-End Bound		
								Memory Bound	Core Bound	
									Divider	Port Utilization
40										
41	inline double Hep3Vector::operator () (int i) const {									
42	switch(i) {	20,232,000,000	18,900,000,000	1.070	16.7%	18.9%	23.3%	18.5%	0.0%	20.0%
43	case X:									
44	return x();									
45	case Y:									
46	return y();	72,000,000	36,000,000	2.000	0.4%	0.0%	0.0%	0.0%	0.0%	0.0%
47	case Z:									
48	return z();	144,000,000	0		0.0%	0.0%	1.5%	0.0%	0.0%	1.3%
49	}									
50	return 0.;									
51	}									

Example of Bad Speculation

Be careful with what you assume the compiler can optimize for you.

```
1   class vector {
2       public:
3       double operator[](int i) {
4           switch(i) {
5               case 0: return x;
6               case 1: return y;
7               case 2: return z;
8           }
9       }
10
11      double operator()(int i) { return (&x)[i]; }
12
13      private:
14      double x, y, z;
15  };
```

```
1   vector::operator[](int):
2     cmp esi, 1
3     je .L2
4     cmp esi, 2
5     jne .L6
6     vmovsd xmm0, QWORD PTR [rdi+16]
7     ret
8   .L2:
9     vmovsd xmm0, QWORD PTR [rdi+8]
10    ret
11  .L6:
12    vmovsd xmm0, QWORD PTR [rdi]
13    ret
14  vector::operator()(int):
15    movsx rsi, esi
16    vmovsd xmm0, QWORD PTR [rdi+rsi*8]
17    ret
```

Branching Efficiently

```
    Reorder condition in G4CrossSectionDataStore::ComputeCrossSection()

diff --git a/source/processes/hadronic/cross_sections/src/G4CrossSectionDataStore.cc b/source/processes/hadronic/cross_sections/src/G4CrossSectionData
Store.cc
index d81d3239ba..06e446f192 100644
--- a/source/processes/hadronic/cross_sections/src/G4CrossSectionDataStore.cc
+++ b/source/processes/hadronic/cross_sections/src/G4CrossSectionDataStore.cc
@@ -271,10 +271,10 @@ G4double
 G4CrossSectionDataStore::ComputeCrossSection(const G4DynamicParticle* part,
                                              const G4Material* mat)
 {
-  if(mat == currentMaterial && part->GetDefinition() == matParticle
-     && part->GetKineticEnergy() == matKinEnergy) {
+  if(part->GetKineticEnergy() == matKinEnergy && mat == currentMaterial &&
+     part->GetDefinition() == matParticle)
     return matCrossSection;
-  }
+
   currentMaterial = mat;
   matParticle = part->GetDefinition();
   matKinEnergy = part->GetKineticEnergy();

commit 3a2c4714fddc62e6ea31b6c5f074f60461ac05f4
Author: Guilherme Amadio <amadio@cern.ch>

    Update History file for G4CrossSectionDataStore

diff --git a/source/processes/hadronic/cross_sections/History b/source/processes/hadronic/cross_sections/History
lines 3391-3418/3673 93%
```

When a branching condition has several terms, order it from the most discriminant term to the least. Here, the energy is different much more frequently than material or particle type, so this order leads to more early decisions and better performance.

Probabilities:

mat == currentMaterial ⇒ 98.9%
matParticle == part->GetDefinition ⇒ 71.6%
matKinEnergy == part->GetKineticEnergy() ⇒ 32.1%

Note that these are independent. They are equal together only about 6.9% of the time.

Unnecessary Work: Caching Data

```
@@ -297,38 +300,30 @@ G4CrossSectionDataStore::GetCrossSection(const G4DynamicParticle* part,
                                         const G4Element* elm,
                                         const G4Material* mat)
 {
-  if(mat == elmMaterial && elm == currentElement &&
-     part->GetDefinition() == elmParticle &&
-     part->GetKineticEnergy() == elmKinEnergy)
-    { return elmCrossSection; }
-
-  elmMaterial = mat;
-  currentElement = elm;
-  elmParticle = part->GetDefinition();
-  elmKinEnergy = part->GetKineticEnergy();
-  elmCrossSection = 0.0;
-
-  G4int i = nDataSetList-1;
+  G4int i = nDataSetList-1;
   G4int Z = elm->GetZasInt();
+
   if (elm->GetNaturalAbundanceFlag() &&
       dataSetList[i]->IsElementApplicable(part, Z, mat)) {

     // element wise cross section
-    elmCrossSection = dataSetList[i]->GetElementCrossSection(part, Z, mat);
-  } else {
-    // isotope wise cross section
-    size_t nIso = elm->GetNumberOfIsotopes();
-
-    // user-defined isotope abundances
lines 3282-3310/3673 91%
```

This method is always called with each element of each material in a loop, so the element is never the same, and the cache was missed 100% of the time.

Unnecessary Work: Avoid Distant Data Accesses

```
   // Check if the particle has a force, EM or gravitational, exerted on it
   //
-  G4FieldManager* fieldMgr = 0;
-  G4bool fieldExertsForce  = false;

-  fieldMgr = fFieldPropagator->FindAndSetFieldManager(track.GetVolume());
   G4bool eligibleEM =
-    (particleCharge != 0.0) || (fUseMagneticMoment && (magneticMoment != 0.0));
-  G4bool eligibleGrav = fUseGravity && (restMass != 0.0);
+    (particleCharge != 0.0) || ((magneticMoment != 0.0) && fUseMagneticMoment);
+  G4bool eligibleGrav = (restMass != 0.0) && fUseGravity;

-  if((fieldMgr != nullptr) && (eligibleEM || eligibleGrav))
+  fFieldExertedForce = false;
+
+  if(eligibleEM || eligibleGrav)
   {
-    // User can configure the field Manager for this track
-    fieldMgr->ConfigureForTrack(&track);
-    // Called here to allow a transition from no-field pointer
-    // to finite field (non-zero pointer).
-
-    // If the field manager has no field ptr, the field is zero
-    //   by definition ( = there is no field ! )
-    const G4Field* ptrField = fieldMgr->GetDetectorField();
-    if(ptrField)
+    if(G4FieldManager* fieldMgr =
+         fFieldPropagator->FindAndSetFieldManager(track.GetVolume()))
     {
lines 2502-2530/3673 69%
```

Finding the field manager is expensive. It requires accessing distant pieces of data like the fFieldPropagator class member to call its method, and the track's current volume. However, we can avoid checking the field for neutral and/or massless particles, as the field has no effect on them.

Data Access Patterns: Group Nearby Reads & Writes

```
    G4Transportation: Move changes to fParticleChange closer together

diff --git a/source/processes/transportation/src/G4Transportation.cc b/source/processes/transportation/src/G4Transportation.cc
index 1fc97e1595..ba7a621299 100644
--- a/source/processes/transportation/src/G4Transportation.cc
+++ b/source/processes/transportation/src/G4Transportation.cc
@@ -195,12 +195,6 @@ G4double G4Transportation::AlongStepGetPhysicalInteractionLength(
    //
    *selection = CandidateForSelection;

-   fFirstStepInVolume = fNewTrack || fLastStepInVolume;
-   fLastStepInVolume  = false;
-   fNewTrack          = false;
-
-   fParticleChange.ProposeFirstStepInVolume(fFirstStepInVolume);
-
    // Get initial Energy/Momentum of the track
    //
    const G4DynamicParticle* pParticle       = track.GetDynamicParticle();
@@ -510,6 +504,11 @@ G4double G4Transportation::AlongStepGetPhysicalInteractionLength(
      }
    }

+   fFirstStepInVolume = fNewTrack || fLastStepInVolume;
+   fLastStepInVolume  = false;
+   fNewTrack          = false;
+
+   fParticleChange.ProposeFirstStepInVolume(fFirstStepInVolume);
    fParticleChange.ProposeTrueStepLength(geometryStepLength);
lines 2361-2389/3673 65%
```

If a class member needs to be accessed multiple times inside a function or method, prefer keeping these accesses close together to avoid unnecessary cache misses.

Data Access Patterns: Avoid Indirections

```
Author: Guilherme Amadio <amadio@cern.ch>

    G4SteppingManager: reuse value of fCurrentVolume to find current region

diff --git a/source/tracking/src/G4SteppingManager.cc b/source/tracking/src/G4SteppingManager.cc
index 0e00aca0d3..1108ef957d 100644
--- a/source/tracking/src/G4SteppingManager.cc
+++ b/source/tracking/src/G4SteppingManager.cc
@@ -257,9 +257,10 @@ G4StepStatus G4SteppingManager::Stepping()
   {
     fUserSteppingAction->UserSteppingAction(fStep);
   }
-  G4UserSteppingAction* regionalAction = fStep->GetPreStepPoint()
-        ->GetPhysicalVolume()->GetLogicalVolume()
-        ->GetRegion()->GetRegionalSteppingAction();
+
+  G4UserSteppingAction* regionalAction =
+    fCurrentVolume->GetLogicalVolume()->GetRegion()->GetRegionalSteppingAction();
+
   if(regionalAction)
     regionalAction->UserSteppingAction(fStep);


commit 3c70a7e614d885364905bb5208f6cefbff94c2d9
Author: Guilherme Amadio <amadio@cern.ch>
```

Chained accessors via pointers require several memory accesses to retrieve a single piece of data, with similar cost to traversing a linked list. Here we can avoid 3 access indirections by reusing the value of fCurrentVolume.

Data Access Patterns: Avoid Indirections

```
  // Initialize G4StepPoint attributes.
  // To avoid the circular dependency between G4Track, G4Step
  // and G4StepPoint, G4Step has to manage the copy actions.
  fpPreStepPoint->SetPosition(fpTrack->GetPosition());
  fpPreStepPoint->SetGlobalTime(fpTrack->GetGlobalTime());
  fpPreStepPoint->SetLocalTime(fpTrack->GetLocalTime());
  fpPreStepPoint->SetProperTime(fpTrack->GetProperTime());
  fpPreStepPoint->SetMomentumDirection(fpTrack->GetMomentumDirection());
  fpPreStepPoint->SetKineticEnergy(fpTrack->GetKineticEnergy());
  fpPreStepPoint->SetTouchableHandle(fpTrack->GetTouchableHandle());
  fpPreStepPoint->SetMaterial(
    fpTrack->GetTouchable()->GetVolume()->GetLogicalVolume()->GetMaterial());
  fpPreStepPoint->SetMaterialCutsCouple(fpTrack->GetTouchable()
                                        ->GetVolume()
                                        ->GetLogicalVolume()
                                        ->GetMaterialCutsCouple());
  fpPreStepPoint->SetSensitiveDetector(fpTrack->GetTouchable()
                                       ->GetVolume()
                                       ->GetLogicalVolume()
                                       ->GetSensitiveDetector());
  fpPreStepPoint->SetPolarization(fpTrack->GetPolarization());
  fpPreStepPoint->SetSafety(0.);
  fpPreStepPoint->SetStepStatus(fUndefined);
  fpPreStepPoint->SetProcessDefinedStep(0);
  fpPreStepPoint->SetMass(fpTrack->GetDynamicParticle()->GetMass());
  fpPreStepPoint->SetCharge(fpTrack->GetDynamicParticle()->GetCharge());
  fpPreStepPoint->SetWeight(fpTrack->GetWeight());

  // Set Velocity
"track/include/G4Step.icc" 306 lines --54%--                    166,1         59%
```

These actually return fDynamicParticle->Get...()!

Not good! Traverses pointer chains multiple times, and G4TouchableHandle is actually reference counted, so this has branches and is incrementing and decrementing counters multiple times too!

Data Access Patterns: Avoid Indirections

Source Line ▲	Source
166	// Initialize G4StepPoint attributes.
167	// To avoid the circular dependency between G4Track, G4Step
168	// and G4StepPoint, G4Step has to manage the copy actions.
169	fpPreStepPoint->SetPosition(fpTrack->GetPosition());
170	fpPreStepPoint->SetGlobalTime(fpTrack->GetGlobalTime());
171	fpPreStepPoint->SetLocalTime(fpTrack->GetLocalTime());
172	fpPreStepPoint->SetProperTime(fpTrack->GetProperTime());
173	fpPreStepPoint->SetMomentumDirection(fpTrack->GetMomentumDirection());
174	fpPreStepPoint->SetKineticEnergy(fpTrack->GetKineticEnergy());
175	fpPreStepPoint->SetTouchableHandle(fpTrack->GetTouchableHandle());
176	fpPreStepPoint->SetMaterial(
177	fpTrack->GetTouchable()->GetVolume()->GetLogicalVolume()->GetMaterial());
178	fpPreStepPoint->SetMaterialCutsCouple(fpTrack->GetTouchable()
179	->GetVolume()
180	->GetLogicalVolume()
181	->GetMaterialCutsCouple());
182	fpPreStepPoint->SetSensitiveDetector(fpTrack->GetTouchable()
183	->GetVolume()
184	->GetLogicalVolume()
185	->GetSensitiveDetector());
186	fpPreStepPoint->SetPolarization(fpTrack->GetPolarization());
187	fpPreStepPoint->SetSafety(0.);
188	fpPreStepPoint->SetStepStatus(fUndefined);
189	fpPreStepPoint->SetProcessDefinedStep(0);
190	fpPreStepPoint->SetMass(fpTrack->GetDynamicParticle()->GetMass());
191	fpPreStepPoint->SetCharge(fpTrack->GetDynamicParticle()->GetCharge());
192	fpPreStepPoint->SetWeight(fpTrack->GetWeight());
193	
194	// Set Velocity
195	// should be placed after SetMaterial for preStep point
196	fpPreStepPoint->SetVelocity(fpTrack->CalculateVelocity());
197	
198	(*fpPostStepPoint) = (*fpPreStepPoint);
199	}
200	

Address ▲	Source Line	Assembly	Clockticks	Instructions Retired	CPI Rate	Retiring
0x2bd92	176	callq 0xf580	1,674,000,000	1,872,000,000	0.894	12.6%
0x2bd97		**Block 8:**				
0x2bd97	176	movq %rax, 0x60(%r15)				
0x2bd9b	178	movq 0x28(%rbx), %rax				
0x2bd9f	178	movq 0x10(%rbx), %r12				
0x2bda3	178	movq 0x38(%rax), %rax				
0x2bda7	178	test %rax, %rax				
0x2bdaa	178	jz 0x2bfb3 <Block 40>				
0x2bdb0		**Block 9:**				
0x2bdb0	178	movq 0x8(%rax), %rdi				
0x2bdb4	178	xor %esi, %esi	0	18,000,000	0.000	0.0%
0x2bdb6	179	movq (%rdi), %rax				
0x2bdb9	178	callq 0x20(%rax)	0	54,000,000	0.000	0.0%
0x2bdbc		**Block 10:**				
0x2bdbc	178	movq 0x10(%rax), %rdi	36,000,000	0		0.0%
0x2bdc0	178	callq 0xf690	684,000,000	504,000,000	1.357	3.1%
0x2bdc5		**Block 11:**				
0x2bdc5	178	movq %rax, 0x68(%r12)				
0x2bdca	182	movq 0x28(%rbx), %rax				
0x2bdce	182	movq 0x10(%rbx), %r12				
0x2bdd2	182	movq 0x38(%rax), %rax				
0x2bdd6	182	test %rax, %rax				
0x2bdd9	182	jz 0x2bfb3 <Block 40>				
0x2bddf		**Block 12:**				
0x2bddf	182	movq 0x8(%rax), %rdi				
0x2bde3	182	xor %esi, %esi				
0x2bde5	183	movq (%rdi), %rax				
0x2bde8	182	callq 0x20(%rax)	54,000,000	36,000,000	1.500	0.0%
0x2bdeb		**Block 13:**				
0x2bdeb	182	movq 0x10(%rax), %rdi	0	0	0.000	0.0%
0x2bdef	182	callq 0xf9e0	720,000,000	540,000,000	1.333	0.0%
0x2bdf4		**Block 14:**				
0x2bdf4	182	movq 0x28(%rbx), %rdi	0	0	0.000	0.0%
0x2bdf8	182	movq %rax, 0x70(%r12)				
0x2bdfd	186	movq 0x50(%rdi), %r12				

G4PhysicsVector::Interpolation()

Source Line ▲	Source	Clockticks	Instructions Retired	CPI Rate	Locators				
					Retiring	Front-End Bound	Bad Speculation	Back-End Bound	
								Memory Bound	Core Bound
184	`// ------------------------------------------------------------------`								
185									
186	`inline G4double G4PhysicsVector::Interpolation(const std::size_t idx,`								
187	`                                               const G4double e) const`								
188	`{`								
189	`  // perform the interpolation`								
190	`  const G4double x1 = binVector[idx];`	0.2%	0.1%	1.011	3.8%	6.0%	3.4%	3.9%	3.8%
191	`  const G4double d1 = binVector[idx + 1] - x1;`								
192	`  // note: all corner cases of the previous methods are covered and eventually`								
193	`  //       gives b=0/1 that results in y=y0\y_{N-1} if e<=x[0]/e>=x[N-1] or`								
194	`  //       y=y_i/y_{i+1} if e<x[i]/e>=x[i+1] due to small numerical errors`								
195	`  const G4double b = std::max(0., std::min(1., (e - x1) / d1));`	0.0%	0.0%	0.199	0.3%	0.0%	0.0%	0.0%	0.0%
196	`  G4double res;`								
197	`  if(useSpline)  // spline interpolation`	0.0%	0.0%	1.000	0.4%	0.0%		0.4%	
198	`  {`								
199	`    const G4double os = 0.166666666667;  // 1./6.`								
200	`    const G4double a  = 1.0 - b;`	0.1%	0.0%	3.851	1.4%	0.0%	0.7%	2.3%	2.3%
201	`    const G4double c0 = (a * a * a - a) * secDerivative[idx];`	0.1%	0.0%	4.105	0.5%	0.0%	0.9%	1.9%	1.1%
202	`    const G4double c1 = (b * b * b - b) * secDerivative[idx + 1];`	0.1%	0.0%	5.371	1.3%	0.0%	2.0%	3.6%	3.8%
203	`    res =`								
204	`      a * dataVector[idx] + b * dataVector[idx + 1] + (c0 + c1) * d1 * d1 * os;`	0.6%	0.3%	1.240	9.7%	0.0%	3.7%	17.4%	23.9%
205	`  }`								
206	`  else  // linear interpolation`								
207	`  {`								
208	`    const G4double y1 = dataVector[idx];`								
209	`    const G4double y2 = dataVector[idx + 1];`								
210	`    res               = y1 + b * (y2 - y1);`	0.0%	0.0%	7.667	0.0%	1.5%	0.0%	0.1%	0.0%
211	`  }`								
212	`  return res;`								
213	`}`								

Arithmetics: Instruction Level Parallelism

```
+++ b/source/global/management/include/G4PhysicsVector.icc
@@ -129,10 +129,10 @@ inline G4double G4PhysicsVector::Interpolation(const std::size_t idx,
 {
   // perform the interpolation
   const G4double x1 = binVector[idx];
-  const G4double dl = binVector[idx + 1] - binVector[idx];
+  const G4double dl = binVector[idx + 1] - x1;

   const G4double y1 = dataVector[idx];
-  const G4double dy = dataVector[idx + 1] - dataVector[idx];
+  const G4double dy = dataVector[idx + 1] - y1;

   // note: all corner cases of the previous methods are covered and eventually
   //       gives b=0/1 that results in y=y0\y_{N-1} if e<=x[0]/e>=x[N-1] or
@@ -143,10 +143,9 @@ inline G4double G4PhysicsVector::Interpolation(const std::size_t idx,

   if(useSpline)  // spline interpolation
   {
-    const G4double a  = 1.0 - b;
-    const G4double c0 = (a * a * a - a) * secDerivative[idx];
-    const G4double c1 = (b * b * b - b) * secDerivative[idx + 1];
-    res += (c0 + c1) * dl * dl * (1.0/6.0);
+    const G4double c0 = (2.0 - b) * secDerivative[idx];
+    const G4double c1 = (1.0 + b) * secDerivative[idx + 1];
+    res += (b * (b - 1.0)) * (c0 + c1) * (dl * dl * (1.0/6.0));
   }

   return res;
lines 3626-3654/3673 100%
```

Refactoring terms saves some multiplications, but note also the parenthesis. Floating point arithmetics is not associative. Parenthesizing ensures that each of the independent multiplications can be performed in parallel. This alone reduces estimated execution from 60 to 51 cycles (llvm-mca).

Without parenthesis

```
vmovsd xmm5, QWORD PTR .LC0[rip]
vmovsd xmm4, QWORD PTR .LC2[rip]
vsubsd xmm2, xmm1, xmm5
vsubsd xmm4, xmm4, xmm1
vmulsd xmm2, xmm2, xmm1
vaddsd xmm1, xmm1, xmm5
vmulsd xmm1, xmm1, QWORD PTR [rcx+8+rdi]
vfmadd231sd xmm1, xmm4, QWORD PTR [rcx+rax*8]
vmulsd xmm1, xmm2, xmm1
vmulsd xmm1, xmm1, xmm3
vmulsd xmm1, xmm1, xmm3
vfmadd231sd xmm0, xmm1, QWORD PTR .LC1[rip]
```

same register ⇒ sequential

With parenthesis

```
vmovsd xmm5, QWORD PTR .LC0[rip]
vmovsd xmm4, QWORD PTR .LC2[rip]
vsubsd xmm3, xmm1, xmm5
vsubsd xmm4, xmm4, xmm1
vmulsd xmm2, xmm2, xmm2
vmulsd xmm3, xmm3, xmm1
vaddsd xmm1, xmm1, xmm5
vmulsd xmm1, xmm1, QWORD PTR [rcx+8+rdi]
vmulsd xmm2, xmm2, QWORD PTR .LC1[rip]
vfmadd132sd xmm4, xmm1, QWORD PTR [rcx+rax*8]
vmulsd xmm3, xmm3, xmm4
vfmadd231sd xmm0, xmm3, xmm2
```

different registers ⇒ parallel

Arithmetics: Instruction Level Parallelism

```
                    0123456789          0123456789          0123456789
Index     0123456789          0123456789          0123456789

[0,0]     DeER .    .    .    .    .    .    .    .    .    .    .   .   movslq %edi, %rax
[0,1]     D=eER.    .    .    .    .    .    .    .    .    .    .   .   leaq   (,%rax,8), %rdi
[0,2]     D=eeeeeER .    .    .    .    .    .    .    .    .    .   .   vmovsd (%rsi,%rax,8), %xmm1
[0,3]     D==eeeeeER.    .    .    .    .    .    .    .    .    .   .   vmovsd 8(%rsi,%rdi), %xmm3
[0,4]     D=eeeeeE-R.    .    .    .    .    .    .    .    .    .   .   vmovsd (%rdx,%rax,8), %xmm2
[0,5]     D=======eeeeER .    .    .    .    .    .    .    .    .   .   vsubsd %xmm1, %xmm3, %xmm3
[0,6]     .D=====eeeeE-R .    .    .    .    .    .    .    .    .   .   vsubsd %xmm1, %xmm0, %xmm1
[0,7]     .D=eeeeeE----R .    .    .    .    .    .    .    .    .   .   vmovsd 8(%rdx,%rdi), %xmm0
[0,8]     .D==========eeeeeeeeeeeeER  .    .    .    .    .    .   .   vdivsd %xmm3, %xmm1, %xmm1
[0,9]     .D======eeeeE-------------R  .    .    .    .    .    .   .   vsubsd %xmm2, %xmm0, %xmm0
[0,10]    .D========================eeeeER   .    .    .    .    .   .   vfmadd132sd    %xmm1, %xmm2, %xmm0
[0,11]    .DeE--------------------------R   .    .    .    .    .   .   testb  %r8b, %r8b
[0,12]    . DeE-------------------------R   .    .    .    .    .   .   je     .L1
[0,13]    . D=eeeeeE--------------------R   .    .    .    .    .   .   vmovsd .LC0(%rip), %xmm5
[0,14]    . D=eeeeeE--------------------R   .    .    .    .    .   .   vmovsd .LC1(%rip), %xmm4
[0,15]    . D======================eeeeER   .    .    .    .    .   .   vsubsd %xmm5, %xmm1, %xmm2
[0,16]    . D=========================eeeeER  .    .    .    .    .   .   vsubsd %xmm1, %xmm4, %xmm4
[0,17]    . D===========================eeeeER    .    .    .    .   .   vmulsd %xmm1, %xmm2, %xmm2
[0,18]    .  D=======================eeeeE---R    .    .    .    .   .   vaddsd %xmm5, %xmm1, %xmm1
[0,19]    .  D=========================eeeeeeeeeER .    .    .    .   .   vmulsd 8(%rcx,%rdi), %xmm1, %xmm1
[0,20]    .  D============================eeeeeeeeeER  .    .    .   .   vfmadd231sd    (%rcx,%rax,8), %xmm4, %xmm1
[0,21]    .  D======================================eeeeER   .    .   .   vmulsd %xmm1, %xmm2, %xmm1
[0,22]    .   D=======================================eeeeER    .    .   vmulsd %xmm3, %xmm1, %xmm1
[0,23]    .   D================================================eeeeER.   .   vmulsd %xmm3, %xmm1, %xmm1
[0,24]    .   D=============================================eeeeeeeeeER   vfmadd231sd    .LC2(%rip), %xmm1, %xmm0
[0,25]    .    DeeeeeeeE-----------------------------------------R   retq
titanx ~ $ g++-11.2.0 -march=native -O3 -S test.cc -DV2=1 -o - | llvm-mca -iterations=1 -timeline 2>/dev/null | sed -n 97,125p
```

Legend

```
D : Instruction dispatched.

e : Instruction executing.

E : Instruction executed.

R : Instruction retired.

= : Instruction already dispatched, waiting to be executed.

- : Instruction executed, waiting to be retired.
```

Arithmetics: Instruction Level Parallelism

```
                    0123456789          0123456789          0
Index     0123456789          0123456789          0123456789

[0,0]     DeER .    .    .    .    .    .    .    .    .    .   movslq  %edi, %rax
[0,1]     D=eER.    .    .    .    .    .    .    .    .    .   leaq    (,%rax,8), %rdi
[0,2]     D=eeeeeER .    .    .    .    .    .    .    .    .   vmovsd  (%rsi,%rax,8), %xmm1
[0,3]     D==eeeeeER.    .    .    .    .    .    .    .    .   vmovsd  8(%rsi,%rdi), %xmm2
[0,4]     D=eeeeeE-R.    .    .    .    .    .    .    .    .   vmovsd  (%rdx,%rax,8), %xmm3
[0,5]     D=======eeeeER .    .    .    .    .    .    .    .   vsubsd  %xmm1, %xmm2, %xmm2
[0,6]     .D=====eeeeE-R .    .    .    .    .    .    .    .   vsubsd  %xmm1, %xmm0, %xmm1
[0,7]     .D=eeeeeE----R .    .    .    .    .    .    .    .   vmovsd  8(%rdx,%rdi), %xmm0
[0,8]     .D==========eeeeeeeeeeeeeER  .    .    .    .    .   vdivsd  %xmm2, %xmm1, %xmm1
[0,9]     .D======eeeeE-------------R  .    .    .    .    .   vsubsd  %xmm3, %xmm0, %xmm0
[0,10]    .D========================eeeeER   .    .    .    .   vfmadd132sd     %xmm1, %xmm3, %xmm0
[0,11]    .DeE--------------------------R   .    .    .    .   testb   %r8b, %r8b
[0,12]    . DeE-------------------------R   .    .    .    .   je      .L1
[0,13]    . D=eeeeeE--------------------R   .    .    .    .   vmovsd  .LC0(%rip), %xmm5
[0,14]    . D=eeeeeE--------------------R   .    .    .    .   vmovsd  .LC1(%rip), %xmm4
[0,15]    . D======================eeeeER   .    .    .    .   vsubsd  %xmm5, %xmm1, %xmm3
[0,16]    . D=========================eeeeER  .    .    .    .   vsubsd  %xmm1, %xmm4, %xmm4
[0,17]    . D=========eeeeE--------------R  .    .    .    .   vmulsd  %xmm2, %xmm2, %xmm2
[0,18]    .  D=========================eeeeER    .    .    .   vmulsd  %xmm1, %xmm3, %xmm3
[0,19]    .  D=======================eeeeE---R    .    .    .   vaddsd  %xmm5, %xmm1, %xmm1
[0,20]    .  D=========================eeeeeeeeeER .    .    .   vmulsd  8(%rcx,%rdi), %xmm1, %xmm1
[0,21]    .  D=========eeeeeeeeeE--------------R .    .    .   vmulsd  .LC2(%rip), %xmm2, %xmm2
[0,22]    .   D=========================eeeeeeeeeER  .    .   vfmadd132sd     (%rcx,%rax,8), %xmm1, %xmm4
[0,23]    .   D=====================================eeeeER   .   vmulsd  %xmm4, %xmm3, %xmm3
[0,24]    .   D=====================================eeeeER   vfmadd231sd     %xmm2, %xmm3, %xmm0
[0,25]    .    DeeeeeeeE-----------------------------------R   retq
titanx ~ $ g++-11.2.0 -march=native -O3 -S test.cc -DV3=1 -o - | llvm-mca -iterations=1 -timeline 2>/dev/null | sed -n 97,125p_
```

Legend

```
D : Instruction dispatched.

e : Instruction executing.

E : Instruction executed.

R : Instruction retired.

= : Instruction already dispatched, waiting to be executed.

- : Instruction executed, waiting to be retired.
```

Top 20 classes in Geant4 10.5.1

Class / Source Function / Call Stack	CPU Time ▼	Instructions Retired	Microarchitecture Usage	
			Microarchitecture Usage	CPI Rate
CLHEP::Hep3Vector	9.7%	10.7%	37.3%	0.589
LArWheelCalculator_Impl::DistanceCalculatorSaggingOff	8.2%	11.3%	55.8%	0.473
G4PhysicsVector	5.4%	2.9%	21.3%	1.219
[Not part of any known object class]	5.2%	4.4%	33.5%	0.777
G4PolyconeSide	4.6%	7.1%	57.8%	0.431
G4VoxelNavigation	4.0%	1.6%	14.2%	1.661
MagField::AtlasFieldSvc	3.6%	3.7%	40.6%	0.633
G4SteppingManager	2.8%	3.0%	39.7%	0.611
LArWheelSolid	2.2%	1.7%	31.2%	0.860
LArWheelCalculator	2.2%	3.6%	51.4%	0.394
G4ProductionCutsTable	1.6%	0.7%	14.8%	1.555
CLHEP::MixMaxRng	1.6%	2.1%	50.0%	0.493
BFieldCache	1.5%	2.5%	60.3%	0.385
G4CrossSectionDataStore	1.4%	1.6%	39.7%	0.548
G4LogicalVolume	1.4%	1.1%	28.3%	0.893
G4Navigator	1.4%	1.4%	32.7%	0.649
G4Tubs	1.4%	1.2%	34.3%	0.751
G4UrbanMscModel	1.3%	0.8%	24.1%	1.023
G4VProcess	1.3%	0.7%	26.2%	1.159
G4VCSGfaceted	1.3%	1.4%	49.4%	0.607

Top 20 classes in Geant4 10.7 + Optimizations

Class / Source Function / Call Stack	CPU Time ▼	Instructions Retired	Microarchitecture Usage	
			Microarchitecture Usage	CPI Rate
CLHEP::Hep3Vector	9.6%	10.9%	40.4%	0.565
LArWheelCalculator_Impl::DistanceCalculatorSaggingOff	8.9%	11.8%	55.7%	0.485
G4PolyconeSide	5.2%	7.5%	55.0%	0.439
G4VoxelNavigation	4.6%	1.7%	14.7%	1.706
[Not part of any known object class]	4.3%	3.5%	31.0%	0.796
MagField::AtlasFieldSvc	3.9%	3.9%	40.7%	0.649
G4PhysicsVector	2.9%	2.0%	27.3%	0.944
G4SteppingManager	2.7%	3.1%	41.4%	0.554
LArWheelSolid	2.6%	1.8%	28.3%	0.892
LArWheelCalculator	2.4%	3.9%	51.1%	0.399
G4Navigator	1.8%	1.8%	35.3%	0.647
G4ProductionCutsTable	1.7%	0.7%	14.6%	1.596
BFieldCache	1.7%	2.7%	61.4%	0.398
G4UrbanMscModel	1.6%	0.9%	20.5%	1.115
G4VEmProcess	1.6%	1.1%	27.9%	0.883
G4LogicalVolume	1.5%	1.1%	29.5%	0.902
G4Tubs	1.5%	1.2%	33.9%	0.785
G4VCSGfaceted	1.5%	1.4%	46.7%	0.650
CLHEP::MixMaxRng	1.3%	1.6%	47.6%	0.527
G4AffineTransform	1.3%	1.3%	37.2%	0.645
G4CrossSectionDataStore	1.3%	1.4%	41.7%	0.580

Conclusions and performance tips

- Problems don’t always happen where we expect
  - Always measure to make sure your hypothesis for the cause is correct
- The fastest thing you can do is to not do anything
  - Avoid unnecessary work in your code (e.g. checking field manager for neutral particles)
- Beware of data dependencies
  - Reoder computations to take advantage of instruction level parallelism
  - Strong dependencies can make your code slow even if L1 misses are low
- Beware of indirect accesses via pointers and calls to other shared objects
  - Patterns like `obj->GetFoo()->GetBar()->GetBaz()` are too common in C++
  - Accessing Baz becomes as expensive as traversing a list every time, bad for locality
  - Frequent calls across shared objects are expensive, it’s better to merge into a single library

General Performance Guidelines

Frontend Optimizations

- Check for excessive inlining
- Reduce code size, code duplication
- Avoid unnecessary functions calls
- Avoid frequent calls to distant code
  - Merge libraries that call each other frequently
  - Place functions that call each other nearby
- Optimize conditionals
  - Order by true/false probability
  - Replace conditions by arithmetics
- Use SIMD vectorization (less instructions)

Intel Software Optimization Reference Manuals

Backend Optimizations

- Group nearby data accesses
- Prefer regular data members to pointers
- Avoid indirections from accessor chains
- Break up long for loops into smaller ones
- Avoid data dependencies in arithmetics
- Check cache performance, hit/miss rates
- Be conservative with your assumptions about what the compiler can optimize

AMD Software Optimization Guide for EPYC™ Processors

“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

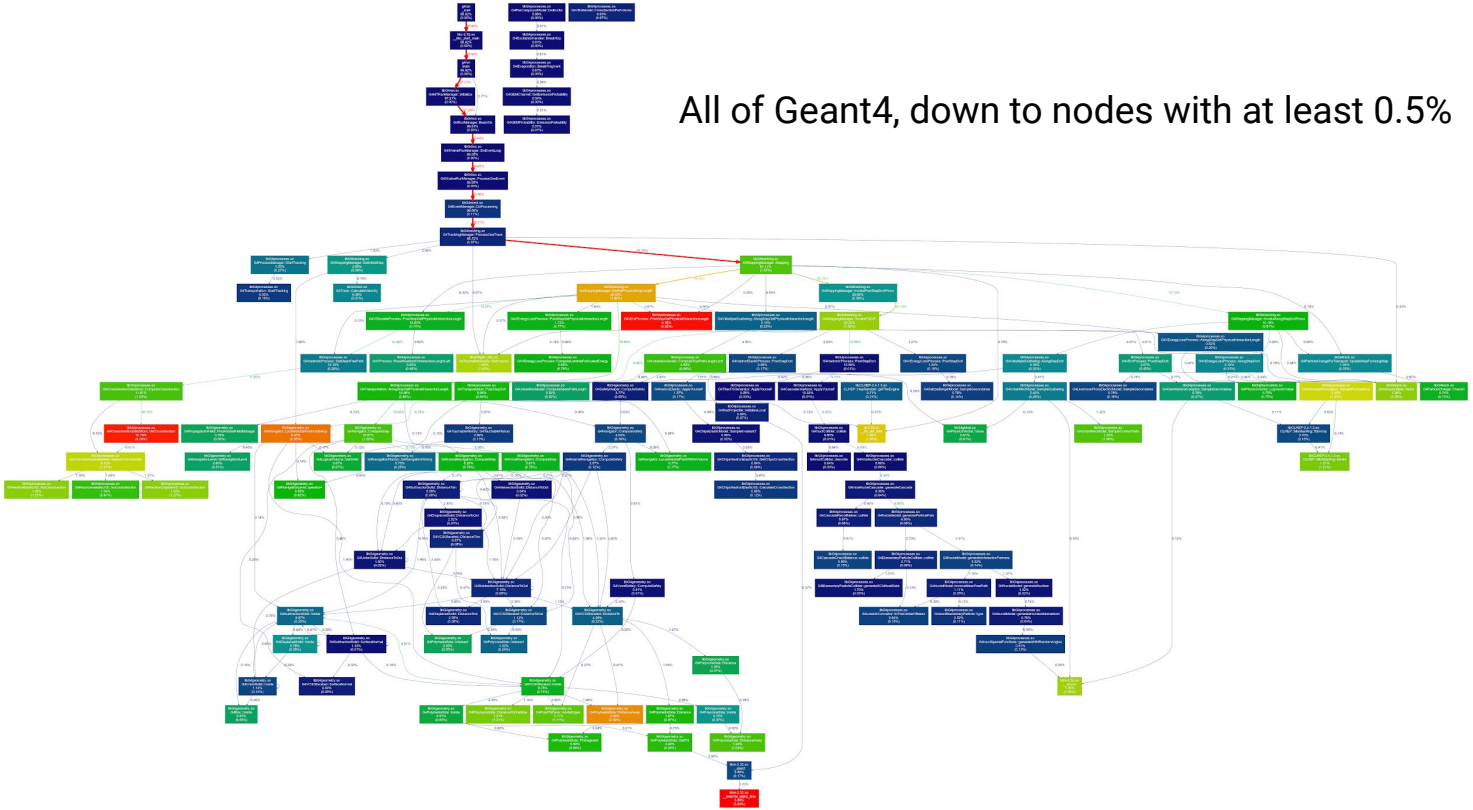
Backup Slides

Profiling Data Post-processing & Visualization

gprof2dot – convert profiling data to a graph

```
bash ~ $ gprof2dot --help
Usage:
        gprof2dot [options] [file] ...

Options:
  -h, --help            show this help message and exit
  -o FILE, --output=FILE
                        output filename [stdout]
  -n PERCENTAGE, --node-thres=PERCENTAGE
                        eliminate nodes below this threshold [default: 0.5]
  -e PERCENTAGE, --edge-thres=PERCENTAGE
                        eliminate edges below this threshold [default: 0.1]
  -f FORMAT, --format=FORMAT
                        profile format: axe, callgrind, dtrace, hprof, json,
                        oprofile, perf, prof, pstats, sleepy, sysprof or xperf
                        [default: prof]
  --total=TOTALMETHOD   preferred method of calculating total time: callratios
                        or callstacks (currently affects only perf format)
                        [default: callratios]
  -c THEME, --colormap=THEME
                        color map: bw, color, gray, pink or print [default:
                        color]
  -s, --strip           strip function parameters, template parameters, and
                        const modifiers from demangled C++ function names
```

gprof2dot – convert profiling data to a graph



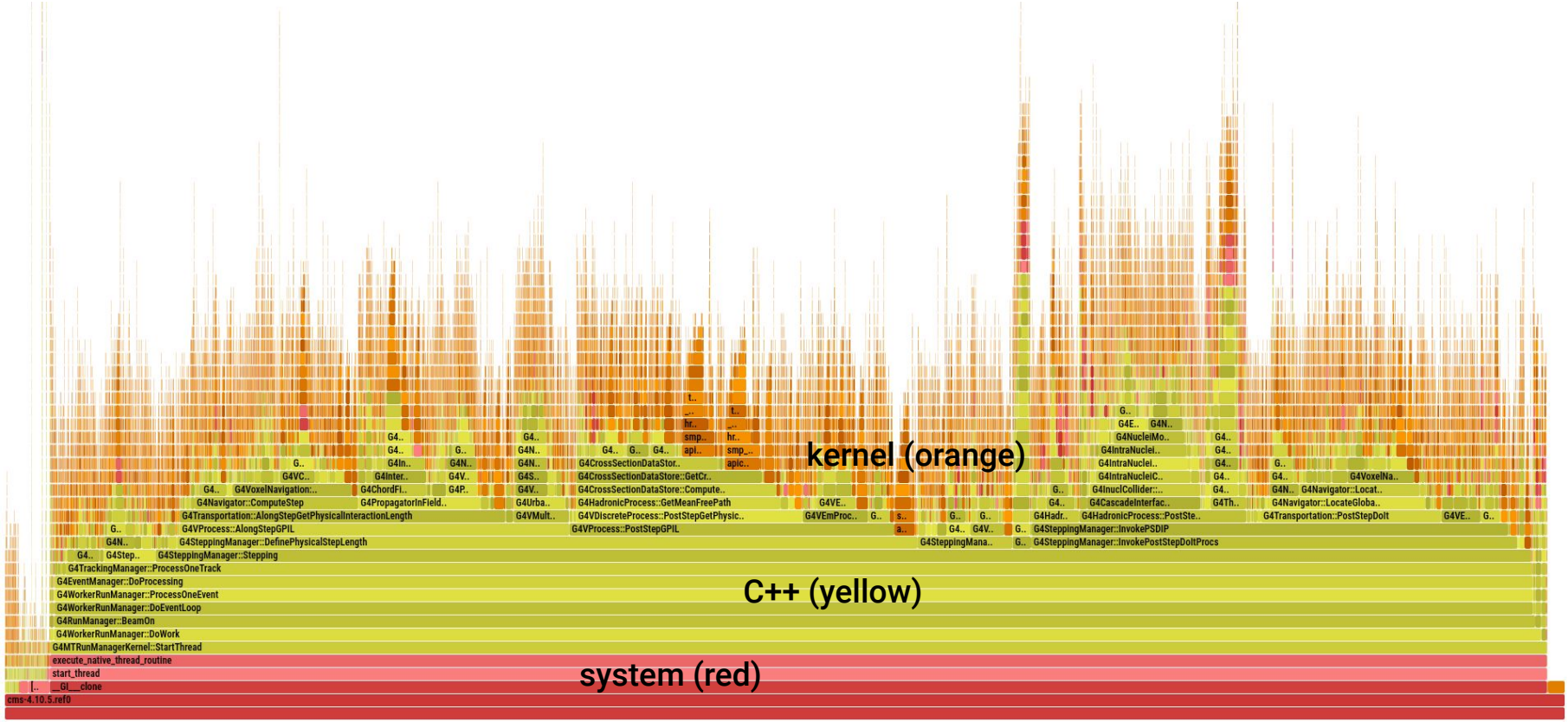
All of Geant4, down to nodes with at least 0.5%

FlameGraph – convert profiling data to a flamegraph

```
bash ~ $ flamegraph.pl --help
USAGE: flamegraph.pl [options] infile > outfile.svg

        --title TEXT     # change title text
        --subtitle TEXT  # second level title (optional)
        --width NUM      # width of image (default 1200)
        --height NUM     # height of each frame (default 16)
        --minwidth NUM   # omit smaller functions (default 0.1 pixels)
        --fonttype FONT  # font type (default "Verdana")
        --fontsize NUM   # font size (default 12)
        --countname TEXT # count type label (default "samples")
        --nametype TEXT  # name type label (default "Function:")
        --colors PALETTE # set color palette. choices are: hot (default), mem,
                         # io, wakeup, chain, java, js, perl, red, green, blue,
                         # aqua, yellow, purple, orange
        --bgcolors COLOR # set background colors. gradient choices are yellow
                         # (default), blue, green, grey; flat colors use "#rrggbb"
        --hash           # colors are keyed by function name hash
        --cp             # use consistent palette (palette.map)
        --reverse        # generate stack-reversed flame graph
        --inverted       # icicle graph
        --flamechart     # produce a flame chart (sort by time, do not merge stacks)
        --negate         # switch differential hues (blue<->red)
        --notes TEXT     # add notes comment in SVG (for debugging)
        --help           # this message

        eg,
        flamegraph.pl --title="Flame Graph: malloc()" trace.txt > graph.svg
```

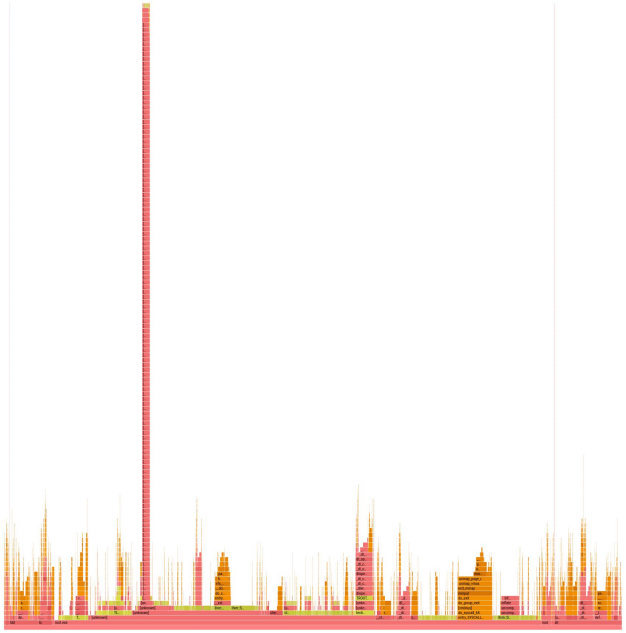
https://github.com/brendangregg/FlameGraph

FlameGraph – convert profiling data to a flamegraph

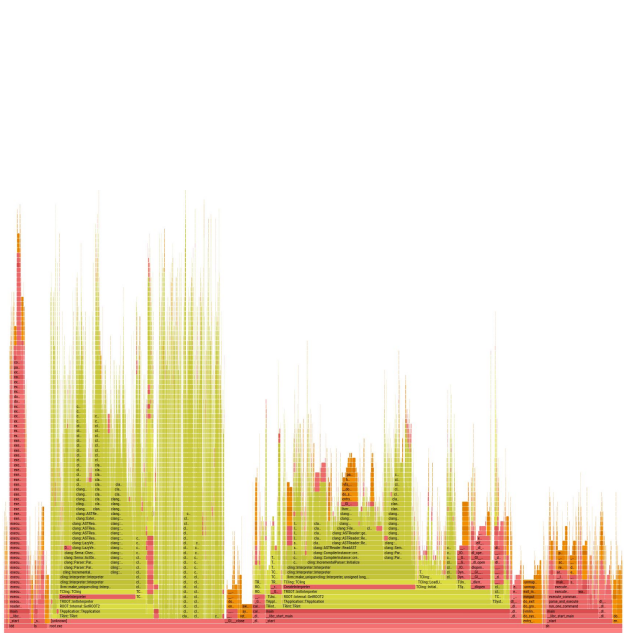


Caveats and Gotchas

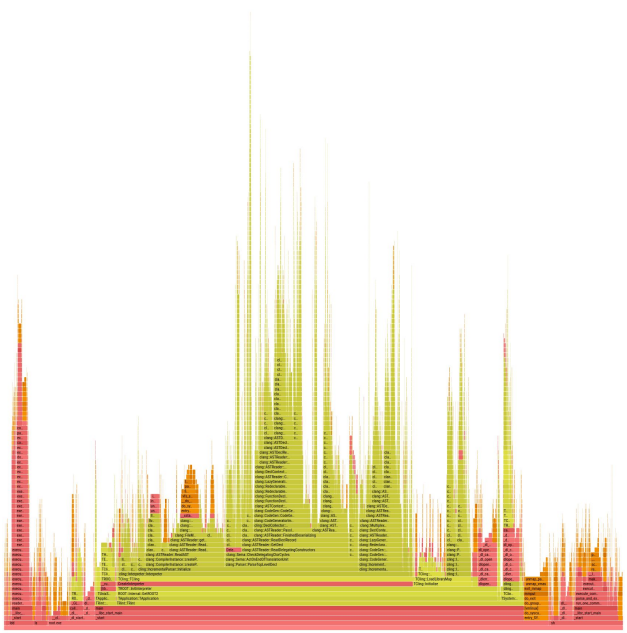
ROOT startup flamegraph for various configurations



perf record --call-graph=fp
(debugging info not available)

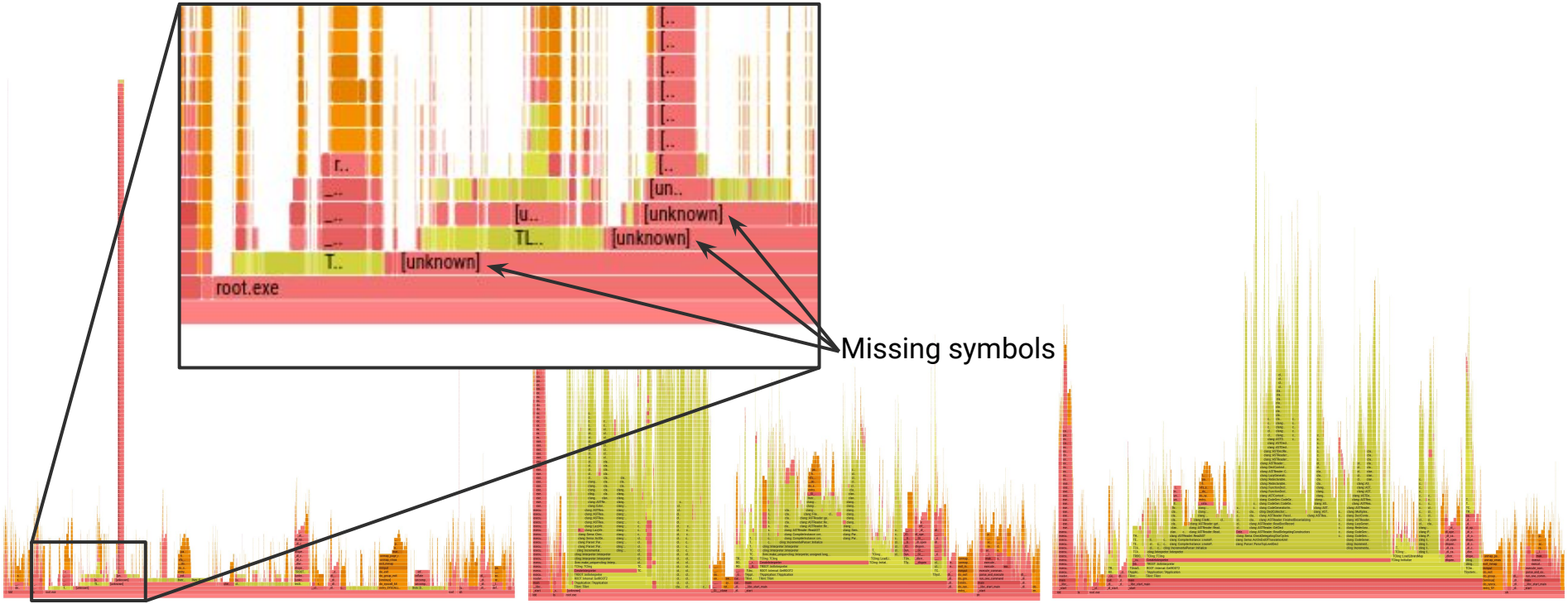


perf record --call-graph=dwarf
(frame pointer not available)



perf record --call-graph=fp
(frame pointer and debugging info)

ROOT startup flamegraph for various configurations



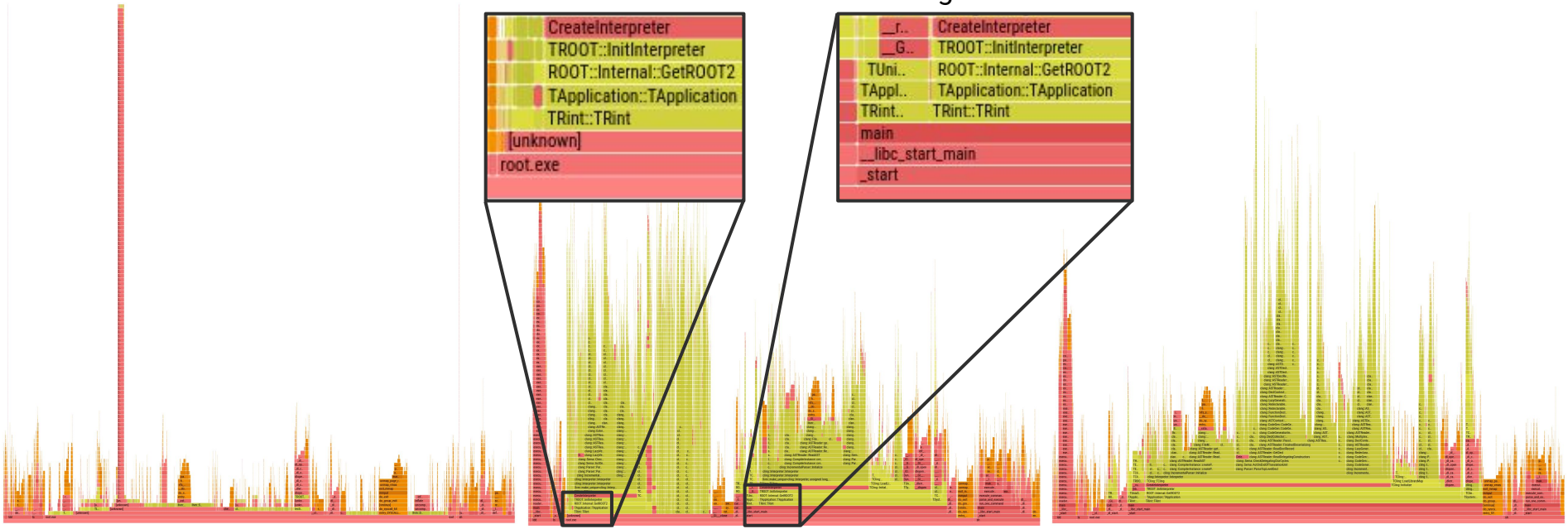
perf record --call-graph=fp
(debugging info not available)

perf record --call-graph=dwarf
(frame pointer not available)

perf record --call-graph=fp
(frame pointer and debugging info)

ROOT startup flamegraph for various configurations

Broken stack unwinding



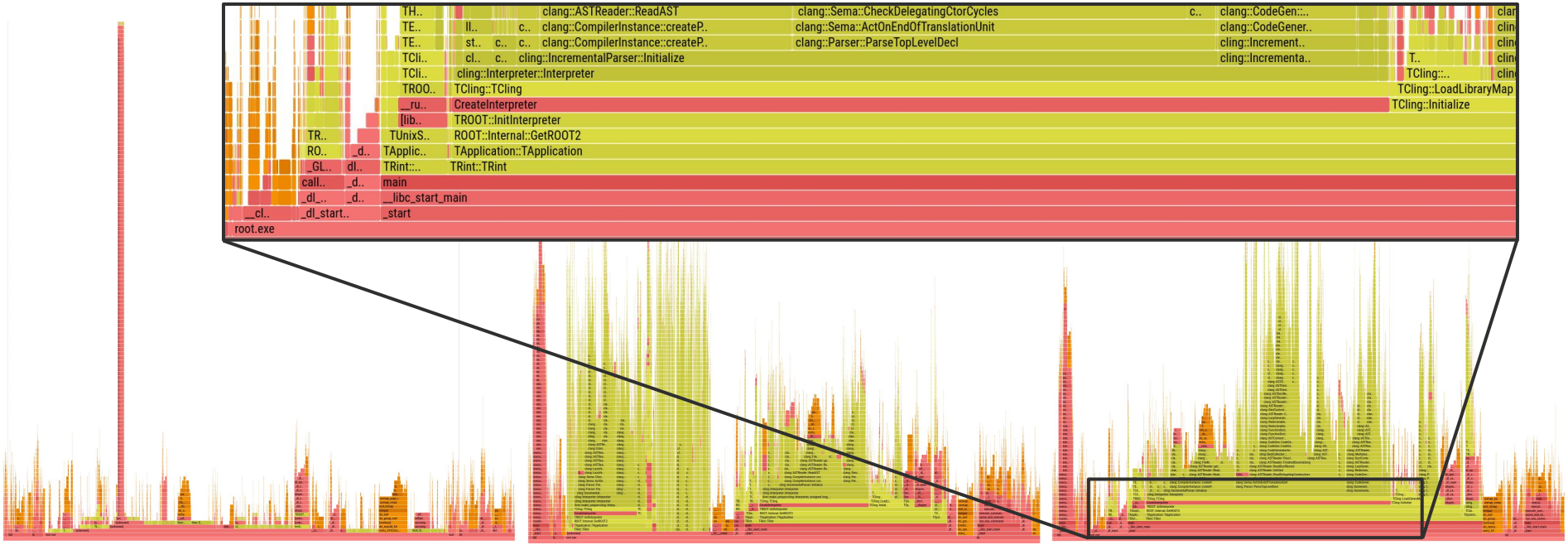
perf record --call-graph=fp
(debugging info not available)

perf record --call-graph=dwarf
(frame pointer not available)

perf record --call-graph=fp
(frame pointer and debugging info)

ROOT startup flamegraph for various configurations

Correctly merged stacks



perf record --call-graph=fp
(debugging info not available)

perf record --call-graph=dwarf
(frame pointer not available)

perf record --call-graph=fp
(frame pointer and debugging info)

Avoid broken stack traces and missing symbols

- Compile code with debugging information (`-g`)
- Add `-fno-omit-frame-pointer` to compile options to keep frame pointer
- Install system packages with debugging info for the kernel and system libs

When recording data:

- Use `--call-graph=fp/dwarf` + DWARF debugging information
- Use precise events to avoid skidding (`cycles:pp` instead of just `cycles`)
- Adjust sampling rate to avoid large amounts of data and high overhead
- Sample events in a group if computing derived metrics (e.g. instr. per cycle)
- See `man perf-list` for more information on events and their modifiers

Microarchitecture Analysis with perf

Metrics are only available with **perf stat**. To be able to get metrics per-symbol with **perf record**:

- Use classification from Intel VTune
- Use formulas for each category based on events known to perf and properties of the hardware
- Record all perf events *in the same sampling group*
- Report counts per symbol using perf
- Post-process with AWK to calculate metrics per symbol
- Can also use similar events and own formulas to create new metrics

Table 7: Intel's implementation of Top-Down Metrics

Metric Name	Intel Core™ events
Clocks	*CPU_CLK_UNHALTED.THREAD*
Slots	*4 * Clocks*
Frontend Bound	IDQ_UOPS_NOT_DELIVERED.CORE / Slots
Bad Speculation	(UOPS_ISSUED.ANY - UOPS_RETIRED.RETIRE_SLOTS + 4* INT_MISC.RECOVERY_CYCLES) / Slots
Retiring	UOPS_RETIRED.RETIRE_SLOTS / Slots
Frontend Latency Bound	IDQ_UOPS_NOT_DELIVERED.CORE: [≥ 4] / Clocks
#BrMispredFraction	*BR_MISP_RETIRED.ALL_BRANCHES / (BR_MISP_RETIRED.ALL_BRANCHES + MACHINE_CLEARS.COUNT)*
MicroSequencer	#RetireUopFraction * IDQ.MS_UOPS / Slots
#ExecutionStalls	*(CYCLE_ACTIVITY.CYCLES_NO_EXECUTE - RS_EVENTS.EMPTY_CYCLES + UOPS_EXECUTED.THREAD: [≥ 1] - UOPS_EXECUTED.THREAD: [≥ 2]) / Clocks*
Memory Bound	*(CYCLE_ACTIVITY.STALLS_MEM_ANY + RESOURCE_STALLS.SB)* / Clocks
L1 Bound	(CYCLE_ACTIVITY.STALLS_MEM_ANY - CYCLE_ACTIVITY.STALLS_L1D_MISS) / Clocks
L2 Bound	(CYCLE_ACTIVITY.STALLS_L1D_MISS - CYCLE_ACTIVITY.STALLS_L2_MISS)/Clocks
#L3HitFraction	*MEM_LOAD_UOPS_RETIRED.LLC_HIT / (MEM_LOAD_UOPS_RETIRED.LLC_HIT +7*MEM_LOAD_UOPS_RETIRED.LLC_MISS)*
L3 Bound	(1 - #L3HitFraction) * CYCLE_ACTIVITY.STALLS_L2_MISS / Clocks
Ext. Memory Bound	CYCLE_ACTIVITY.STALLS_MEM_ANY
MEM Bandwidth	UNC_ARB_TRK_OCCUPANCY.ALL: [≥ 28]/ UNC_CLOCK.SOCKET
MEM Latency	(UNC_ARB_TRK_OCCUPANCY.ALL: [≥ 1] - UNC_ARB_TRK_OCCUPANCY.ALL: [≥ 28]) / UNC_CLOCK.SOCKET

A. Yasin, "A Top-Down method for performance analysis and counters architecture," *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Monterey, CA, 2014, pp. 35-44, doi: 10.1109/ISPASS.2014.6844459.

Example – using perf + awk to get percent retiring

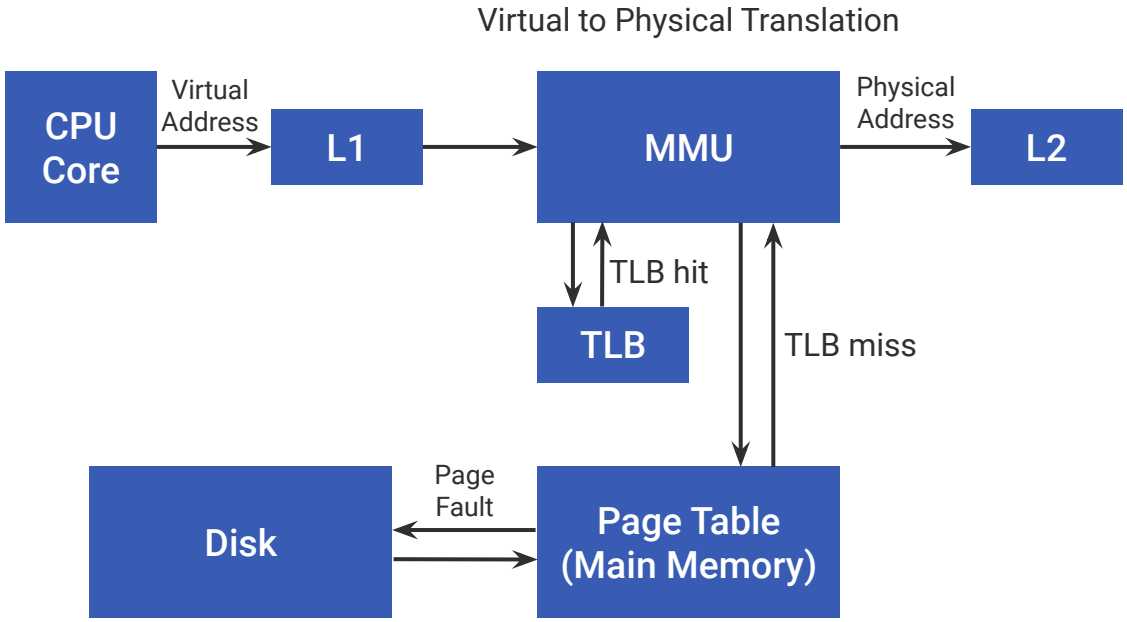
```
bash df102_NanoAODDimuonAnalysis $ perf record -F max -e '{cpu_clk_unhalted.thread,uops_retired.retire_slots}' -- df102_NanoAODD
imuonAnalysis 8 Run2012B_DoubleMuParked.root Run2012C_DoubleMuParked.root
info: Using a maximum frequency rate of 8,000 Hz
Couldn't synthesize cgroup events.
[ perf record: Woken up 57 times to write data ]
[ perf record: Captured and wrote 15.548 MB perf.data (406080 samples) ]
bash df102_NanoAODDimuonAnalysis $ perf report -q --stdio --group -F period,symbol -w 0,90 | head
   104728157092  9748014622  [.] ROOT::Detail::RDF::RFilter<bool (*)(ROOT::VecOps::RVec<int> const&), ROOT::Detail::RDF
   94152141228 10108015162  [.] ROOT::Detail::RDF::RFilter<bool (*)(unsigned int), ROOT::Detail::RDF::RLoopManager>::C
   79494119241  3454005181  [.] TTree::LoadTree
   51302076953 92238138357  [.] inflate_fast
   35698053547 14764022146  [.] TBranch::GetEntry
   24610036915  2248003372  [.] TLeafI::GetMaximum
   14942022413 13312019968  [.] tbb::internal::custom_scheduler<tbb::internal::IntelSchedulerTraits>::receive_or_steal
    8372012558  2912004368  [k] sysret_check
    7632011448  1702002553  [.] ROOT::Detail::RDF::RCustomColumn<float (*)(ROOT::VecOps::RVec<float> const&, ROOT::Vec
    7476011214  6442009663  [.] ROOT::Internal::RDF::RColumnValue<ROOT::VecOps::RVec<float> >::Get<ROOT::VecOps::RVec<
bash df102_NanoAODDimuonAnalysis $
```

Example – using perf + awk to get percent retiring

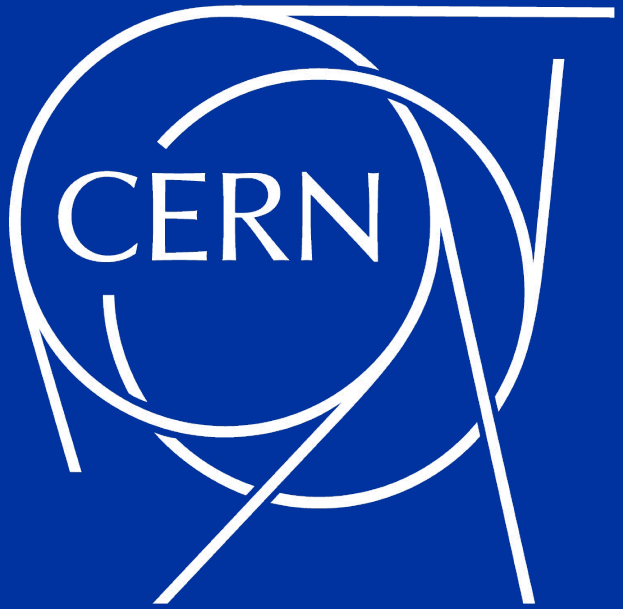
```
bash df102_NanoAODDimuonAnalysis $ echo "Retiring  Symbol"; perf report -q -F period,symbol --percent-limit 1 | awk '/^$/{next}
{ symbol = gensub(".*\\[.\\] ","","g"); slots = 4*$1; retiring = 100*$2/slots; printf("%7.2f%%  %s\n", retiring, symbol) | "sort
 -nr"; }' | cut -b -128
Retiring  Symbol
  70.18%  adler32_z
  44.95%  inflate_fast
  37.48%  ROOT::Internal::TTreeReaderValueBase::ProxyReadTemplate<&ROOT::Detail::TBranchProxy::ReadNoParentNoBranchCountNoCollec
  27.16%  __expm1f
  22.27%  tbb::internal::custom_scheduler<tbb::internal::IntelSchedulerTraits>::receive_or_steal_task
  21.54%  ROOT::Internal::RDF::RColumnValue<ROOT::VecOps::RVec<float> >::Get<ROOT::VecOps::RVec<float>, 0>
  10.36%  ROOT::Detail::RDF::RLoopManager::RunAndCheckFilters
  10.34%  TBranch::GetEntry
   8.70%  sysret_check
   5.58%  ROOT::Detail::RDF::RCustomColumn<float (*)(ROOT::VecOps::RVec<float> const&, ROOT::VecOps::RVec<float> const&, ROOT::V
   2.68%  ROOT::Detail::RDF::RFilter<bool (*)(unsigned int), ROOT::Detail::RDF::RLoopManager>::CheckFilters
   2.33%  ROOT::Detail::RDF::RFilter<bool (*)(ROOT::VecOps::RVec<int> const&), ROOT::Detail::RDF::RFilter<bool (*)(unsigned int)
   2.28%  TLeafI::GetMaximum
   1.09%  TTree::LoadTree
bash df102_NanoAODDimuonAnalysis $ _
```

The Translation Lookaside Buffer

“A translation lookaside buffer (TLB) is a memory cache that is used to reduce the time taken to access a user memory location. It is a part of the chip's memory management unit (MMU). The TLB stores the recent translations of virtual memory to physical memory and can be called an address-translation cache.”



https://en.wikipedia.org/wiki/Translation_lookaside_buffer


CERN