Linux Systems Performance
Tracing, Profiling, and Visualization

Performance is challenging

Some things that can go wrong:

- **Measurement**
  - Overhead from instrumentation and measurement
  - Variable CPU frequency scaling (turbo boost, thermal throttling)
  - 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
Measuring Performance
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 – 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

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

**Hardware events**
- branch-instructions OR branches
- branch-misses
- cache-misses
- cache-references
- cpu-cycles OR cycles
- instructions
- stalled-cycles-backend OR idle-cycles-backend
- stalled-cycles-frontend OR idle-cycles-frontend
- L1-dcache-load-misses
- L1-dcache-loads
- L1-dcache-prefetches
- L1-icache-load-misses
- L1-icache-loads
- branch-load-misses
- branch-loads
dTLB-load-misses
dTLB-loads
iTLB-load-misses
iTLB-loads

**Software events**
- alignment-faults
- bpf-output
- context-switches OR cs
- cpu-clock
- cpu-migrations OR migrations
dummy
- emulation-faults
- major-faults
- minor-faults
- page-faults OR faults
task-clock
duration_time

**Tool event**
- duration_time
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]
bash ~ $ perf list core

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

core:
  \textbf{ex\_div\_busy}
    [Div Cycles Busy count]
  \textbf{ex\_div\_count}
    [Div Op Count]
  \textbf{ex\_ret\_brn}
    [Retired Branch Instructions]
    \textbf{ex\_ret\_brn\_far}
      [Retired Far Control Transfers]
    \textbf{ex\_ret\_brn\_ind\_misp}
      [Retired Indirect Branch Instructions Mispredicted]
    \textbf{ex\_ret\_brn\_misp}
      [Retired Branch Instructions Mispredicted]
    \textbf{ex\_ret\_brn\_resync}
      [Retired Branch Resyncs]
    \textbf{ex\_ret\_brn\_tkn}
      [Retired Taken Branch Instructions]
    \textbf{ex\_ret\_brn\_tkn\_misp}
      [Retired Taken Branch Instructions Mispredicted]
  \textbf{ex\_ret\_cond}
    [Retired Conditional Branch Instructions]
    \textbf{ex\_ret\_cond\_misp}
      [Retired Conditional Branch Instructions Mispredicted]
...
perf – event sources

Dynamic Tracing

Tracepoints

- ext4:
- sock:
- sched:
- task:
- signal:
- timer:
- workqueue:
- syscalls:

Operating System

Applications

System Call Interface

VFS

File Systems

TCP/UDP

IP

Scheduler

Virtual Memory

Device Drivers

Software Events

uprobes

- jbd2:
- block:
- scsi:
- net:
- skb:

kprobes

- cpu-clock
- cs migrations
- page-faults
- minor-faults
- major-faults

Software Events

PMCs

- cycles
- instructions
- branch-
- L1- *
- LLC- *

CPU

Memory Bus

DRAM

mem-load

mem-store

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
- sched:sched_kthread_stop_ret
- sched:sched_migrate_task
- sched:sched_move_numa
- sched:sched_pi_setprio
- sched:sched_process_exec
- sched:sched_process_exit
- sched:sched_process_fork
- sched:sched_process_free
- sched:sched_process_wait
- sched:sched_stat_runtime
- sched:stick_numa
- sched:swap_numa
- sched:sched_switch
- sched:wait_task
- sched:wake_idle_without_ipi
- sched:syscall
- sched:wakeup
- sched:wakeup_new
- sched:waking
perf – 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 – 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
```

<table>
<thead>
<tr>
<th># Children</th>
<th>Self</th>
<th>Shared</th>
<th>Object</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>98.97%</td>
<td>0.00%</td>
<td></td>
<td>libG4event.so [.] G4EventManager::DoProcessing</td>
<td></td>
</tr>
<tr>
<td>98.57%</td>
<td>0.00%</td>
<td></td>
<td>libG4run.so [.] G4WorkerRunManager::ProcessOneEvent</td>
<td></td>
</tr>
<tr>
<td>98.55%</td>
<td>0.00%</td>
<td></td>
<td>libG4run.so [.] G4WorkerRunManager::DoEventLoop</td>
<td></td>
</tr>
<tr>
<td>98.52%</td>
<td>0.00%</td>
<td></td>
<td>libG4run.so [.] G4RunManager::BeamOn</td>
<td></td>
</tr>
<tr>
<td>98.49%</td>
<td>0.00%</td>
<td></td>
<td>libG4run.so [.] G4WorkerRunManager::DoWork</td>
<td></td>
</tr>
<tr>
<td>98.20%</td>
<td>0.00%</td>
<td></td>
<td>libG4run.so [.] G4MTRunManagerKernel::StartThread</td>
<td></td>
</tr>
<tr>
<td>98.19%</td>
<td>0.08%</td>
<td></td>
<td>libG4tracking.so [.] G4TrackingManager::ProcessOneTrack</td>
<td></td>
</tr>
<tr>
<td>98.16%</td>
<td>0.00%</td>
<td></td>
<td>libstdc++.so.6.0.28 [.] execute_native_thread Routine</td>
<td></td>
</tr>
<tr>
<td>98.14%</td>
<td>0.00%</td>
<td></td>
<td>libpthread-2.32.so [.] start_thread</td>
<td></td>
</tr>
<tr>
<td>98.06%</td>
<td>0.00%</td>
<td></td>
<td>libc-2.32.so [.] __GI___clone (inlined)</td>
<td></td>
</tr>
<tr>
<td>90.95%</td>
<td>0.92%</td>
<td></td>
<td>libG4tracking.so [.] G4SteppingManager::Stepping</td>
<td></td>
</tr>
<tr>
<td>52.67%</td>
<td>2.18%</td>
<td></td>
<td>libG4tracking.so [.] G4SteppingManager::DefinePhysicalStepLength</td>
<td></td>
</tr>
<tr>
<td>29.63%</td>
<td>0.00%</td>
<td></td>
<td>libG4tracking.so [.] G4VProcess::AlongStepGPIL (inlined)</td>
<td></td>
</tr>
<tr>
<td>25.74%</td>
<td>0.36%</td>
<td></td>
<td>libG4tracking.so [.] G4SteppingManager::InvokePostStepDoItProcs</td>
<td></td>
</tr>
<tr>
<td>25.36%</td>
<td>1.15%</td>
<td></td>
<td>libG4tracking.so [.] G4SteppingManager::InvokePSDIP</td>
<td></td>
</tr>
<tr>
<td>22.42%</td>
<td>0.95%</td>
<td></td>
<td>libG4processes.so [.] G4Transportation::AlongStepGetPhysicalInteractionLength</td>
<td></td>
</tr>
<tr>
<td>21.44%</td>
<td>0.00%</td>
<td></td>
<td>libG4tracking.so [.] G4VProcess::PostStepGPIL (inlined)</td>
<td></td>
</tr>
<tr>
<td>20.58%</td>
<td>1.12%</td>
<td></td>
<td>libG4geometry.so [.] G4Navigator::ComputeStep</td>
<td></td>
</tr>
</tbody>
</table>
```
perf – reporting by self time

```
bash ~ $ perf report --stdio -c g4run --no-children -g none -w 0,0,0,0,0 2>/dev/null | tail -n +10 | head -n 22

# Overhead Shared Object Symbol
# .............................................................. ................................................
  5.51% libm-2.32.so   [.] __ieee754_atan2_fma
  5.43% lib4physicslists.so  [.] G4PhysicsVector::LogVectorValue
  3.92% lib4processes.so   [.] G4CrossSectionDataStore::GetCrossSection
  3.36% lib4geometry.so    [.] G4PolyhedraSide::DistanceAway
  2.18% lib4tracking.so    [.] G4SteppingManager::DefinePhysicalStepLength
  2.15% lib4processes.so   [.] G4VEmProcess::PostStepGetPhysicalInteractionLength
  1.90% lib4geometry.so    [.] G4PolyhedraSide::DistanceToOneSide
  1.63% lib4processes.so   [.] G4VoxelNavigation::LevelLocate
  1.63% lib4processes.so   [.] G4CrossSectionDataStore::GetIsoCrossSection
  1.58% lib4geometry.so    [.] G4PolyPhiFace::InsideEdges
  1.26% lib4digits_hits.so [.] G4TouchableHistory::GetVolume
  1.21% lib4geometry.so    [.] G4PolyhedraSide::GetPhi
  1.19% lib4clhep.so       [.] CLHEP::MixMaxRng::iterate
  1.19% lib4geometry.so    [.] G4PolyhedraSide::Distance
  1.19% libm-2.32.so       [.] __ sincos
  1.15% lib4tracking.so    [.] G4SteppingManager::InvokePSDIP
  1.12% lib4geometry.so    [.] G4VCSGfaceted::Inside
  1.12% lib4geometry.so    [.] G4Navigator::ComputeStep
  1.09% lib4processes.so   [.] G4UrbanMscModel::SampleCosineTheta
```

bash ~ $
perf – hierarchical report

```bash
bash ~ $ perf report --stdio -c g4run --hierarchy -g none 2>/dev/null | tail -n +10 | cut -b -$COLUMNS | head -n $((SINES - 2))
# Overhead Command / Shared Object / Symbol
# ---------------------------------------------------------------

100.00% g4run
  36.61% libG4processes.so
  3.92% G4CrossSectionDataStore::GetCrossSection
  2.15% G4EmProcess::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
  2.09% libG4particles.so
```

bash ~ $
perf report --gtk

<table>
<thead>
<tr>
<th>Cycles</th>
<th>Self</th>
<th>Command</th>
<th>Shared Object</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>99.56%</td>
<td>0.11%</td>
<td>g4run</td>
<td>libG4event.so</td>
<td>[] G4EventManager::DoProcessing</td>
</tr>
<tr>
<td>98.70%</td>
<td>0.07%</td>
<td>g4run</td>
<td>libG4tracking.so</td>
<td>[] G4TrackingManager::ProcessOneTrack</td>
</tr>
<tr>
<td>98.63%</td>
<td></td>
<td></td>
<td></td>
<td>[] G4TrackingManager::ProcessOneTrack</td>
</tr>
<tr>
<td>91.08%</td>
<td></td>
<td></td>
<td></td>
<td>[] G4SteppingManager::Stepping</td>
</tr>
<tr>
<td>46.00%</td>
<td></td>
<td></td>
<td></td>
<td>[] G4SteppingManager::DefinePhysicalStepLength</td>
</tr>
<tr>
<td>18.80%</td>
<td></td>
<td></td>
<td></td>
<td>G4Transportation::AlongStepGetPhysicalInteractionLength</td>
</tr>
<tr>
<td>13.28%</td>
<td></td>
<td></td>
<td></td>
<td>G4DiscreteProcess::PostStepGetPhysicalInteractionLength</td>
</tr>
<tr>
<td>4.94%</td>
<td></td>
<td></td>
<td></td>
<td>G4MultipleScattering::AlongStepGetPhysicalInteractionLength</td>
</tr>
<tr>
<td>3.87%</td>
<td></td>
<td></td>
<td></td>
<td>G4EMProcess::PostStepGetPhysicalInteractionLength</td>
</tr>
<tr>
<td>1.62%</td>
<td></td>
<td></td>
<td></td>
<td>G4EnergyLossProcess::PostStepGetPhysicalInteractionLength</td>
</tr>
<tr>
<td>0.57%</td>
<td></td>
<td></td>
<td></td>
<td>G4EnergyLossProcess::AlongStepGetPhysicalInteractionLength</td>
</tr>
<tr>
<td>30.79%</td>
<td></td>
<td></td>
<td></td>
<td>G4SteppingManager::InvokePostStepDoltProcs</td>
</tr>
<tr>
<td>30.19%</td>
<td></td>
<td></td>
<td></td>
<td>G4SteppingManager::InvokePSDP</td>
</tr>
<tr>
<td>10.12%</td>
<td></td>
<td></td>
<td></td>
<td>G4SteppingManager::InvokeAlongStepDoltProcs</td>
</tr>
<tr>
<td>2.94%</td>
<td></td>
<td></td>
<td></td>
<td>G4SteppingManager::SetInitialStep</td>
</tr>
<tr>
<td>1.53%</td>
<td></td>
<td></td>
<td></td>
<td>G4ProcessManager::StartTracking</td>
</tr>
<tr>
<td>0.81%</td>
<td></td>
<td></td>
<td></td>
<td>G4TouchableHistory::GetVolume</td>
</tr>
<tr>
<td>91.10%</td>
<td>1.06%</td>
<td>g4run</td>
<td>libG4tracking.so</td>
<td>[] G4SteppingManager::Stepping</td>
</tr>
<tr>
<td>46.03%</td>
<td>1.90%</td>
<td>g4run</td>
<td>libG4tracking.so</td>
<td>[] G4SteppingManager::DefinePhysicalStepLength</td>
</tr>
<tr>
<td>30.84%</td>
<td>0.36%</td>
<td>g4run</td>
<td>libG4tracking.so</td>
<td>[] G4SteppingManager::InvokePostStepDoltProcs</td>
</tr>
<tr>
<td>30.22%</td>
<td>1.31%</td>
<td>g4run</td>
<td>libG4tracking.so</td>
<td>[] G4SteppingManager::InvokePSDP</td>
</tr>
<tr>
<td>18.86%</td>
<td>0.86%</td>
<td>g4run</td>
<td>libG4processes.so</td>
<td>[] G4Transportation::AlongStepGetPhysicalInteractionLength</td>
</tr>
</tbody>
</table>

Tip: To browse sample contexts use perf report --sample 10 and select in context menu
perf report --tui

Samples: 16K of event 'cycles', Event count (approx.): 1110655926

<table>
<thead>
<tr>
<th>Children</th>
<th>Self</th>
<th>Command</th>
<th>Shared Object</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ 32.84%</td>
<td>1.38%</td>
<td>ccache</td>
<td>[kernel.vmlinux]</td>
<td>[k] entry_SYSCALL_64</td>
</tr>
<tr>
<td>+ 31.81%</td>
<td>0.00%</td>
<td>root.exe</td>
<td>root.exe</td>
<td>[...] __start</td>
</tr>
<tr>
<td>+ 31.81%</td>
<td>0.00%</td>
<td>root.exe</td>
<td>libc-2.32.so</td>
<td>[...] __libc_start_main</td>
</tr>
<tr>
<td>- 31.81%</td>
<td>0.00%</td>
<td>root.exe</td>
<td>root.exe</td>
<td>[...] main</td>
</tr>
</tbody>
</table>

- main
  - 30.22% Taint::Taint
  - 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% dllopen@@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

```bash
Samples: 792K of event 'cycles', Event count (approx.): 31698254078309

<table>
<thead>
<tr>
<th>Children</th>
<th>Self</th>
<th>Command</th>
<th>Shared Object</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>- 98.70%</td>
<td>0.07%</td>
<td>g4run</td>
<td>libG4tracking.so</td>
<td>[.] G4TrackingManager::ProcessOneTrack</td>
</tr>
<tr>
<td>- 98.63%</td>
<td></td>
<td>G4TrackingManager::ProcessOneTrack</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- 91.08%</td>
<td></td>
<td>G4SteppingManager::Stepping</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- 46.00%</td>
<td></td>
<td>G4SteppingManager::DefinePhysicalStepLength</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ 18.80%</td>
<td></td>
<td>G4Transportation::AlongStepGetPhysicalInteractionLength</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ 13.28%</td>
<td></td>
<td>G4VDiscreteProcess::PostStepGetPhysicalInteractionLength</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ 4.94%</td>
<td></td>
<td>G4VMultipleScattering::AlongStepGetPhysicalInteractionLength</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.87%</td>
<td></td>
<td>G4VEmProcess::PostStepGetPhysicalInteractionLength</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ 1.62%</td>
<td></td>
<td>G4VEnergyLossProcess::PostStepGetPhysicalInteractionLength</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.57%</td>
<td></td>
<td>G4VEnergyLossProcess::AlongStepGetPhysicalInteractionLength</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- 30.79%</td>
<td></td>
<td>G4SteppingManager::InvokePostStepDoItProcs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- 30.19%</td>
<td></td>
<td>G4SteppingManager::InvokePSDIP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ 10.58%</td>
<td></td>
<td>G4HadronicProcess::PostStepDoIt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ 9.90%</td>
<td></td>
<td>G4Transportation::PostStepDoIt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ 3.61%</td>
<td></td>
<td>G4VEmProcess::PostStepDoIt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ 2.92%</td>
<td></td>
<td>G4HadronElasticProcess::PostStepDoIt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ 1.22%</td>
<td></td>
<td>G4VEnergyLossProcess::PostStepDoIt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ 10.12%</td>
<td></td>
<td>G4SteppingManager::InvokeAlongStepDoItProcs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ 2.94%</td>
<td></td>
<td>G4SteppingManager::SetInitialStep</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ 1.53%</td>
<td></td>
<td>G4ProcessManager::StartTracking</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.81%</td>
<td></td>
<td>G4TouchableHistory::GetVolume</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Press '?' for help on key bindings
perf – annotated source code

Samples: 840K of event 'cycles', 100 Hz, Event count (approx.): 33603550952244
G4Transportation::AlongStepGetPhysicalInteractionLength /usr/lib64/libG4processes.so [Percent: local period]

```c
Percent  mov  G4Transportation::fUseGravity@@Base-0x2edbe9,%rdx
         movzb1 (%rdx),%edx
2.50    and  %esi,%edx
         mov  %edx,%r8d
         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
         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
perf – functions where most cache misses happen

<table>
<thead>
<tr>
<th>Function</th>
<th>Library</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.05% cms</td>
<td>libG4geometry.so</td>
<td>G4Navigator::LocateGlobalPointAndSetup</td>
</tr>
<tr>
<td>6.41% cms</td>
<td>libG4geometry.so</td>
<td>G4VoxelNavigation::ComputeStep</td>
</tr>
<tr>
<td>4.45% cms</td>
<td>libG4geometry.so</td>
<td>G4VoxelNavigation::LocateNextVoxel</td>
</tr>
<tr>
<td>3.78% cms</td>
<td>libG4processes.so</td>
<td>G4VEnergyLossProcess::ComputeLambdaForScaledEnergy</td>
</tr>
<tr>
<td>3.02% cms</td>
<td>libG4processes.so</td>
<td>G4ElectroNuclearCrossSection::GetElementCrossSection</td>
</tr>
<tr>
<td>2.95% cms</td>
<td>libG4processes.so</td>
<td>G4VEmProcess::PostStepGetPhysicalInteractionLength</td>
</tr>
<tr>
<td>2.90% cms</td>
<td>libG4processes.so</td>
<td>G4ChipsNeutronElasticXS::CalculateCrossSection</td>
</tr>
<tr>
<td>2.52% cms</td>
<td>libG4geometry.so</td>
<td>G4PhysicalVolume::GetObjectTranslation</td>
</tr>
<tr>
<td>2.43% cms</td>
<td>libG4modeling.so</td>
<td>G4AffineTransform::G4AffineTransform</td>
</tr>
<tr>
<td>2.02% cms</td>
<td>libG4processes.so</td>
<td>G4VEnergyLossProcess::AlongStepGetPhysicalInteractionLength</td>
</tr>
<tr>
<td>1.76% cms</td>
<td>libG4processes.so</td>
<td>G4VEnergyLossProcess::AlongStepDoIt</td>
</tr>
<tr>
<td>1.56% cms</td>
<td>ld-2.25.so</td>
<td>__tls_get_addr</td>
</tr>
<tr>
<td>1.43% cms</td>
<td>libG4global.so</td>
<td>G4PhysicsVector::Value</td>
</tr>
<tr>
<td>1.33% cms</td>
<td>libG4geometry.so</td>
<td>G4PhysicalVolume::GetFrameRotation</td>
</tr>
<tr>
<td>1.10% cms</td>
<td>libG4geometry.so</td>
<td>G4NavigationLevel::operator=</td>
</tr>
<tr>
<td>1.05% cms</td>
<td>libG4particles.so</td>
<td>G4IonTable::FindIon</td>
</tr>
<tr>
<td>1.03% cms</td>
<td>libG4processes.so</td>
<td>G4VEmProcess::ComputeIntegralLambda</td>
</tr>
<tr>
<td>1.02% cms</td>
<td>libG4processes.so</td>
<td>G4CollisionOutput::reset</td>
</tr>
<tr>
<td>1.02% cms</td>
<td>libG4processes.so</td>
<td>G4WentzelVIModel::ComputeTruePathLengthLimit</td>
</tr>
</tbody>
</table>
perf – script subcommand for post processing data

bash ~ $ perf script -c root.exe | head -n 20
root.exe 271242 5996.319731: 1880 cycles:
    ffffffffac16d32  __mempy+0x12 (/lib/modules/5.9.8-gentoo/build/vmlinux)
    ffffffff9a87564 perf_output_copy+0x34 (/lib/modules/5.9.8-gentoo/build/vmlinux)
    ffffffff9a7e222 perf_event_comm_output+0xc2 (/lib/modules/5.9.8-gentoo/build/vmlinux)
    ffffffff9a7aa1a perf_iterate_ctx+0x5a (/lib/modules/5.9.8-gentoo/build/vmlinux)
    ffffffff9a7b1d9 perf_iterate_sb+0x169 (/lib/modules/5.9.8-gentoo/build/vmlinux)
    ffffffff9a85b23 perf_event_comm+0xf3 (/lib/modules/5.9.8-gentoo/build/vmlinux)
    ffffffff9b47c24 begin_new_exec+0x514 (/lib/modules/5.9.8-gentoo/build/vmlinux)
    ffffffff9b99e8d load_elf_binary+0x73d (/lib/modules/5.9.8-gentoo/build/vmlinux)
    ffffffff9b459a5 bprm_execute+0x2f5 (/lib/modules/5.9.8-gentoo/build/vmlinux)
    ffffffff9b46f92 do_execveat_common.isra.45+0x1a2 (/lib/modules/5.9.8-gentoo/build/vmlinux)
    ffffffff9b471b3 __x64_sys_execve+0x43 (/lib/modules/5.9.8-gentoo/build/vmlinux)
    ffffffffac13253 do_syscall_64+0x33 (/lib/modules/5.9.8-gentoo/build/vmlinux)
    ffffffff9aae007c entry_SYSCALL_64+0x7c (/lib/modules/5.9.8-gentoo/build/vmlinux)
      7fa07291c4f7 [unknown] ([unknown])

root.exe 271242 5996.319731: 4354 cycles:
    ffffffffac16d32  __mempy+0x12 (/lib/modules/5.9.8-gentoo/build/vmlinux)
    ffffffff9a87564 perf_output_copy+0x34 (/lib/modules/5.9.8-gentoo/build/vmlinux)
    ffffffff9a7e222 perf_event_comm_output+0xc2 (/lib/modules/5.9.8-gentoo/build/vmlinux)
bash ~ $ # use a visualization tool to post-process profiling data
bash ~ $ perf script -c root.exe | gprof2dot -f perf | dot -T png > graph.png
bash ~ $
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]
- t TOTALMETHOD, --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%
gprof2dot – convert profiling data to a graph

Filtered to G4SteppingManager::Stepping() and below
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 "#rrgbbb"
--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

kernel (orange)

C++ (yellow)

system (red)
FlameGraph Performance Diff

Hovering reveals more information

Legend: green = better (less cycles), red = worse (more cycles)
Saturation is proportional to relative change

width of each function corresponds to percentage of runtime
Treemap from perf report output using Perl + d3.js

- Created from perf report output after conversion to JSON
- Hierarchical instead of using stack traces
  - Binary
  - Library
  - Class
  - Method / Function
- Good to see if leaves from multiple stacks amount to large time
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
On-CPU Performance Analysis
perf stat -d – overview of Geant4 initialization

```bash
bash $ perf stat -r 3 -d -- taskset -c 0 /srv/geant4/install/gcc-10.2.0/10.6.6.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-10.2.0/10.6.6.r9-MT/bin/g4run -g /home/amadio/src/g4run/CMS.gdml -p pythia:ttbar -e 0' (3 runs):

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
<th>Utilization</th>
<th>Change</th>
<th>Change 100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>21454.06 msec task-clock</td>
<td>0.953 CPUs</td>
<td>( + - 0.07% )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1520 context-switches</td>
<td>0.071 K/sec</td>
<td>( + - 48.20% )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 cpu-migrations</td>
<td>0.000 K/sec</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110280 page-faults</td>
<td>0.005 M/sec</td>
<td>( + - 0.06% )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>93708948749 cycles</td>
<td>4.368 GHz</td>
<td>( + - 0.02% )</td>
<td>(74.96%)</td>
<td></td>
</tr>
<tr>
<td>428488171 stalled-cycles-frontend</td>
<td>0.46% frontend cycles idle</td>
<td>( + - 2.14% )</td>
<td>(74.95%)</td>
<td></td>
</tr>
<tr>
<td>62148664026 stalled-cycles-backend</td>
<td>66.31% backend cycles idle</td>
<td>( + - 0.04% )</td>
<td>(74.97%)</td>
<td></td>
</tr>
<tr>
<td>129389101781 instructions</td>
<td>1.38 insn per cycle</td>
<td>( + - 0.04% )</td>
<td>(75.02%)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.48 stalled cycles per insn</td>
<td>( + - 0.04% )</td>
<td>(75.02%)</td>
<td></td>
</tr>
<tr>
<td>16731397508 branches</td>
<td>779.871 M/sec</td>
<td>( + - 0.06% )</td>
<td>(75.06%)</td>
<td></td>
</tr>
<tr>
<td>156166747 branch-misses</td>
<td>0.93% of all branches</td>
<td>( + - 0.17% )</td>
<td>(75.09%)</td>
<td></td>
</tr>
<tr>
<td>58140887925 L1-dcache-loads</td>
<td>2710.018 M/sec</td>
<td>( + - 0.10% )</td>
<td>(75.02%)</td>
<td></td>
</tr>
<tr>
<td>685016614 L1-dcache-load-misses</td>
<td>1.18% of all L1-dcache accesses</td>
<td>( + - 1.28% )</td>
<td>(74.93%)</td>
<td></td>
</tr>
</tbody>
</table>
<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 --stats -g /srv/geant4/gdm1/CMS.gdm1 -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 \sim 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  [.]  G4ElasticHadronNucleusHE::HadnNucDifferCrSec
 3.76%  libG4processes.so  [.]  G4BremsstrahlungRelModel::ComputeLPMfunctions
 3.72%  libG4processes.so  [.]  G4BremsstrahlungModel::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  [.]  G4BremsstrahlungRelModel::ComputeXSectionPerAtom
 1.05%  libG4geometry.so   [.]  G4LogicalVolume::GetMaterial
```
bash ~ $  # Looks like \sim 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!

<table>
<thead>
<tr>
<th>Rank</th>
<th>File</th>
<th>Function</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>33.59%</td>
<td>libG4processes.so</td>
<td>G4hPairProductionModel::ComputeDMicroscopicCrossSection</td>
<td></td>
</tr>
<tr>
<td>14.26%</td>
<td>libG4processes.so</td>
<td>G4ElasticHadronNucleusHE::HadronNucleusDifferentialCrossSection</td>
<td></td>
</tr>
<tr>
<td>11.97%</td>
<td>libG4processes.so</td>
<td>G4MuPairProductionModel::ComputeDMicroscopicCrossSection</td>
<td></td>
</tr>
<tr>
<td>4.98%</td>
<td>libG4processes.so</td>
<td>G4eBremsstrahlungRelModel::ComputeLPMFunctions</td>
<td></td>
</tr>
<tr>
<td>4.90%</td>
<td>libG4processes.so</td>
<td>G4hBremsstrahlungModel::ComputeDMicroscopicCrossSection</td>
<td></td>
</tr>
<tr>
<td>3.80%</td>
<td>libG4global.so</td>
<td>G4PhysicsVector::Value</td>
<td></td>
</tr>
<tr>
<td>2.46%</td>
<td>libG4processes.so</td>
<td>G4MuBremsstrahlungModel::ComputeDMicroscopicCrossSection</td>
<td></td>
</tr>
<tr>
<td>2.09%</td>
<td>libG4geometry.so</td>
<td>G4Region::BelongsTo</td>
<td></td>
</tr>
<tr>
<td>1.65%</td>
<td>libG4processes.so</td>
<td>G4SeltzerBergerModel::ComputeDXSectionPerAtom</td>
<td></td>
</tr>
<tr>
<td>1.36%</td>
<td>libG4processes.so</td>
<td>G4eBremsstrahlungRelModel::ComputeDXSectionPerAtom</td>
<td></td>
</tr>
<tr>
<td>1.07%</td>
<td>libG4processes.so</td>
<td>G4LossTableBuilder::BuildRangeTable</td>
<td></td>
</tr>
<tr>
<td>1.01%</td>
<td>libpythia8.so</td>
<td>Pythia8::NNPDF::pointint</td>
<td></td>
</tr>
</tbody>
</table>

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...
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 [Perf core: 1]

0.46  vdivsd  %xmm0,%xmm7,%xmm7
      |    |    |
      G4double ale=G4Log(bbb/z13*sqrt(x1*y1)/(1.+screen*y1));

2.33  vmovsd  -0x88(%rbp),%xmm0

0.00  vdivsd  0x140(%r10),%xmm0,%xmm15
      |    |    |
      G4double ye1 = 1.+yeu/yed;
      vaddsd  %xmm2,%xmm7,%xmm7

2.48  vmulsd  %xmm0,%xmm15,%xmm0

0.15  vmulsd  %xmm7,%xmm11,%xmm0

0.78  vcomisd  %xmm0,%xmm3

0.32  → ja  649179 <G4hPairProductionModel::ComputeDMicroscopicCrossSection(double, double, double)+0x13e9>

0.31  vsqrtsd  %xmm0,%xmm0,%xmm0

0.45  vmovsd  -0x50(%rbp),%xmm15
      |    |    |
      vfmadd132sd  %xmm7,%xmm2,%xmm15
      vdivsd  %xmm15,%xmm0,%xmm15

G4LogConsts::dp2uint64(double):
  tmp.d = x;
  vmovq  %xmm15,%rax

G4LogConsts::getMantExponent(double, double&):
  uint64_t le = (n >> 52);
  vmovq  %xmm15,%rdx

Press 'h' for help on key bindings
Source code excerpt (1)

```c
// 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;
```

- 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.
Example of code change applied

```cpp
-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);
```

Source code excerpt (2)

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

// Gaussian integration in ln(1-ro)
for (G4int i=0; i<n; ++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 x1 = x10*a5;
    G4double xii = 1./x1;
    G4double xii = 1.+x1;
    G4double screen = screen0*xii/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(xii*ye1))/(1.+screen*ye1));
    G4double cre = 0.5*G4Log((1.+2.25*z23*xii*ye1/massratio2));
    G4double be;
```

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!

```
"/srv/geant4/src/geant4-10.6.r9/source/processes/electromagnetic/muons/src/G4MuPairProductionModel.cc" 712 lines --50%%
```
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 altogether
  - Remove code duplication from the two classes with essentially the same version of this function by inheriting the base version in the derived class

---

For faster computing, the expressions for \( L_{\phi} \) are further algebraically transformed. The functions \( L_{\phi} \) include the nuclear size correction \([KFP71]\) in comparison with parameterization \([KFP70]\):

\[
\begin{align*}
Y_\rho &= \frac{25 - \rho^2 + 4\beta(1 + \rho^2)}{(1 + \rho^2)^2} \\
Y_\phi &= \frac{4 + \rho^2 + 3\beta(1 + \rho^2)}{(1 + \rho^2)^2} \\
\rho_{\text{max}} &= \sqrt{1 - 4\rho^2/E_V(1 - 4\rho^2)}
\end{align*}
\]

Comment on the Calculation of the Integral \( \int dp \) in Eq.(154)

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

\[
t = \ln(1 - \rho), \quad \beta = \frac{\omega^2}{2(1 - \nu^2)},
\]

After that,

\[
\int_0^{\rho_{\text{max}}} G(Z, E, \rho) \, dp = \int_0^t G(Z, E, \nu) e^t \, dt,
\]

where

\[
\begin{align*}
L_{\phi} &= \ln \left( \frac{1 + \sqrt{1 + (1 + Y_\rho)Y_\phi}}{1 + \sqrt{1 + (1 + Y_\rho)Y_\phi}} \right) \\
L_{\phi} &= \ln \left( \frac{1 + \sqrt{1 + (1 + Y_\rho)Y_\phi}}{1 + \sqrt{1 + (1 + Y_\rho)Y_\phi}} \right)
\end{align*}
\]

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. \([KFP78]\), is used:

\[
\zeta(E, Z) = \begin{cases} 
0.073 \ln \frac{E_{\mu}}{1 + \gamma_{\mu} E_{\mu}} - 0.29 & \\
0.058 \ln \frac{E_{\mu}}{1 + \gamma_{\mu} E_{\mu}} - 0.14 
\end{cases} = 0 \quad \text{if the numerator is negative.}
\]

For \( E \leq 35 \mu, \zeta(E, Z) = 0 \). Also \( \gamma_1 = 1.95 \cdot 10^{-3} \) and \( \gamma_2 = 5.30 \cdot 10^{-5} \).

The above formulism 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:

- \( A^* = 202.4 \)
- \( \gamma_1 = 1.95 \cdot 10^{-5} \Rightarrow 4.4 \cdot 10^{-5} \)
- \( \gamma_2 = 5.30 \cdot 10^{-5} \Rightarrow 4.8 \cdot 10^{-5} \)
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
From \textasciitilde 40\% of initialization to \textasciitilde 27\%, not bad!

\texttt{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!

\begin{verbatim}
<table>
<thead>
<tr>
<th>Percentage</th>
<th>Library Name</th>
<th>Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>26.88%</td>
<td>libG4processes.so</td>
<td>G4MuPairProductionModel::ComputeDMicroscopicCrossSection</td>
</tr>
<tr>
<td>12.60%</td>
<td>libG4processes.so</td>
<td>G4ElasticHadronNucleusHE::HadronDifferCrSec</td>
</tr>
<tr>
<td>4.60%</td>
<td>libG4processes.so</td>
<td>G4eBremsstrahlungRelModel::ComputeLPMfunctions</td>
</tr>
<tr>
<td>4.53%</td>
<td>libG4processes.so</td>
<td>G4hBremsstrahlungModel::ComputeDMicroscopicCrossSection</td>
</tr>
<tr>
<td>3.96%</td>
<td>libG4global.so</td>
<td>G4PhysicsVector::Value</td>
</tr>
<tr>
<td>2.33%</td>
<td>libG4geometry.so</td>
<td>G4Region::BelongsTo</td>
</tr>
<tr>
<td>2.26%</td>
<td>libG4processes.so</td>
<td>G4MuBremsstrahlungModel::ComputeDMicroscopicCrossSection</td>
</tr>
<tr>
<td>1.78%</td>
<td>libG4processes.so</td>
<td>G4SeltzerBergerModel::ComputeDXSectionPerAtom</td>
</tr>
<tr>
<td>1.67%</td>
<td>libG4processes.so</td>
<td>G4ProductionCutsTable::ScanAndSetCouple</td>
</tr>
<tr>
<td>1.63%</td>
<td>libpythia8.so</td>
<td>Pythia8::NNPDF::polint</td>
</tr>
<tr>
<td>1.39%</td>
<td>libG4processes.so</td>
<td>G4eBremsstrahlungRelModel::ComputeXSectionPerAtom</td>
</tr>
<tr>
<td>1.30%</td>
<td>libG4geometry.so</td>
<td>G4LogicalVolume::GetMaterial</td>
</tr>
<tr>
<td>1.06%</td>
<td>libG4processes.so</td>
<td>G4LossTableBuilder::BuildRangeTable</td>
</tr>
<tr>
<td>1.01%</td>
<td>libc-2.32.so</td>
<td>malloc</td>
</tr>
</tbody>
</table>
\end{verbatim}
Revisiting overview of Geant4 initialization (before)

```bash
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.gdml -p pythia:ttbar -e 0' (3 runs):

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
<th>Utilization</th>
<th>+/- Relative Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>21454.06 msec task-clock</td>
<td>0.953 CPUs utilized</td>
<td>(+- 0.07%)</td>
<td></td>
</tr>
<tr>
<td>1520 context-switches</td>
<td>0.071 K/sec</td>
<td>(+- 48.20%)</td>
<td></td>
</tr>
<tr>
<td>1 cpu-migrations</td>
<td>0.000 K/sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110280 page-faults</td>
<td>0.005 M/sec</td>
<td>(+- 0.06%)</td>
<td></td>
</tr>
<tr>
<td>93708948749 cycles</td>
<td>4.368 GHz</td>
<td>(+- 0.02%) (74.96%)</td>
<td></td>
</tr>
<tr>
<td>428488171 stalled-cycles-frontend</td>
<td>0.46% frontend cycles idle</td>
<td>(+- 2.14%) (74.95%)</td>
<td></td>
</tr>
<tr>
<td>62148664026 stalled-cycles-backend</td>
<td>66.31% backend cycles idle</td>
<td>(+- 0.04%) (74.97%)</td>
<td></td>
</tr>
<tr>
<td>12938910178 instructions</td>
<td>1.38 insn per cycle</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.48 stalled cycles per insn</td>
<td>(+- 0.04%) (75.02%)</td>
<td></td>
</tr>
<tr>
<td>16731397508 branches</td>
<td>779.871 M/sec</td>
<td>(+- 0.06%) (75.06%)</td>
<td></td>
</tr>
<tr>
<td>156166747 branch-misses</td>
<td>0.93% of all branches</td>
<td>(+- 0.17%) (75.09%)</td>
<td></td>
</tr>
<tr>
<td>58140887925 L1-dcache-loads</td>
<td>2710.018 M/sec</td>
<td>(+- 0.10%) (75.02%)</td>
<td></td>
</tr>
<tr>
<td>685016614 L1-dcache-load-misses</td>
<td>1.18% of all L1-dcache accesses</td>
<td>(+- 1.28%) (74.93%)</td>
<td></td>
</tr>
<tr>
<td>&lt;not supported&gt; LLC-loads</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;not supported&gt; LLC-load-misses</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

22.513 +- 0.637 seconds time elapsed (+- 2.83%)
```
Revisiting overview of Geant4 initialization (after)

```bash
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 pythia: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):

<table>
<thead>
<tr>
<th>Event</th>
<th>Value</th>
<th>Unit</th>
<th>Relative Error</th>
<th>Percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td>17517.74 m/sec task-clock</td>
<td>0.982</td>
<td>CPUs utilized</td>
<td>(+- 0.20%)</td>
<td>74.98%</td>
</tr>
<tr>
<td>282 context-switches</td>
<td>0.016</td>
<td>K/sec</td>
<td>(+- 1.13%)</td>
<td>74.98%</td>
</tr>
<tr>
<td>1 cpu-migrations</td>
<td>0.000</td>
<td>K/sec</td>
<td>(+- 0.01%)</td>
<td>74.98%</td>
</tr>
<tr>
<td>110368 page-faults</td>
<td>0.006</td>
<td>M/sec</td>
<td>(+- 0.01%)</td>
<td>74.98%</td>
</tr>
<tr>
<td>76145083799 cycles</td>
<td>4.347</td>
<td>GHz</td>
<td>(+- 0.11%)</td>
<td>74.98%</td>
</tr>
<tr>
<td>428362729 stalled-cycles-frontend</td>
<td>0.56%</td>
<td>frontend cycles idle</td>
<td>(+- 0.83%)</td>
<td>74.98%</td>
</tr>
<tr>
<td>45942101836 stalled-cycles-backend</td>
<td>59.15%</td>
<td>backend cycles idle</td>
<td>(+- 0.14%)</td>
<td>74.98%</td>
</tr>
<tr>
<td>125423223087 instructions</td>
<td>1.65</td>
<td>insn per cycle</td>
<td>(+- 0.04%)</td>
<td>74.96%</td>
</tr>
<tr>
<td></td>
<td>0.36</td>
<td>stalled cycles per insn</td>
<td>(+- 0.04%)</td>
<td>74.96%</td>
</tr>
<tr>
<td>16930383319 branches</td>
<td>966.471</td>
<td>M/sec</td>
<td>(+- 0.23%)</td>
<td>74.94%</td>
</tr>
<tr>
<td>157044670 branch-misses</td>
<td>0.93%</td>
<td>of all branches</td>
<td>(+- 0.02%)</td>
<td>75.04%</td>
</tr>
<tr>
<td>57470258214 L1-dcache-misses</td>
<td>3280.690</td>
<td>M/sec</td>
<td>(+- 0.09%)</td>
<td>75.09%</td>
</tr>
<tr>
<td>675844171 L1-dcache-load-misses</td>
<td>1.18%</td>
<td>of all L1-dcache accesses</td>
<td>(+- 0.35%)</td>
<td>75.02%</td>
</tr>
</tbody>
</table>
<not supported> LLC-loads                  |         |                    |                 |              |
<not supported> LLC-load-misses            |         |                    |                 |              |

17.8476 +- 0.0237 seconds time elapsed    | (+- 0.13%)  |
DWARF can show time spent in inlined functions

Samples: 2M of event 'cycles', Event count (approx.): 113912086093

<table>
<thead>
<tr>
<th>Children</th>
<th>Self</th>
<th>Shared Object</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>- 38.11%</td>
<td>0.01%</td>
<td>libG4Processes.so</td>
<td>[...] G4MuPairProductionModel::ComputeCrossSectionPerAtom</td>
</tr>
<tr>
<td>- 38.10%</td>
<td>0.10%</td>
<td>G4MuPairProductionModel::ComputeCrossSectionPerAtom</td>
<td></td>
</tr>
<tr>
<td>- 38.10%</td>
<td>0.10%</td>
<td>G4MuPairProductionModel::ComputeMicroscopicCrossSection</td>
<td></td>
</tr>
<tr>
<td>- 28.15%</td>
<td>0.00%</td>
<td>G4hPairProductionModel::ComputeDMicroscopicCrossSection</td>
<td></td>
</tr>
<tr>
<td>+ 19.40%</td>
<td>0.00%</td>
<td>G4Log (inlined)</td>
<td></td>
</tr>
<tr>
<td>+ 0.60%</td>
<td>0.00%</td>
<td>G4Exp (inlined)</td>
<td></td>
</tr>
<tr>
<td>- 9.77%</td>
<td>0.00%</td>
<td>G4MuPairProductionModel::ComputeDMicroscopicCrossSection</td>
<td></td>
</tr>
<tr>
<td>+ 6.72%</td>
<td>0.00%</td>
<td>G4Log (inlined)</td>
<td></td>
</tr>
<tr>
<td>+ 38.10%</td>
<td>0.19%</td>
<td>libG4Processes.so</td>
<td>[...] G4MuPairProductionModel::ComputeMicroscopicCrossSection</td>
</tr>
<tr>
<td>+ 29.84%</td>
<td>29.82%</td>
<td>libG4Processes.so</td>
<td>[...] G4hPairProductionModel::ComputeDMicroscopicCrossSection</td>
</tr>
<tr>
<td>+ 20.55%</td>
<td>0.00%</td>
<td>libG4Processes.so</td>
<td>[...] G4Log (inlined)</td>
</tr>
<tr>
<td>+ 13.04%</td>
<td>0.00%</td>
<td>libG4Processes.so</td>
<td>[...] G4HadronicInteractionRegistry::InitialiseModels</td>
</tr>
<tr>
<td>+ 10.70%</td>
<td>0.00%</td>
<td>libG4Processes.so</td>
<td>[...] G4ElasticHadNucleusHE::InitialiseModel</td>
</tr>
<tr>
<td>+ 10.70%</td>
<td>0.00%</td>
<td>libG4Processes.so</td>
<td>[...] G4ElasticHadNucleusHE::FillData</td>
</tr>
<tr>
<td>+ 10.69%</td>
<td>0.00%</td>
<td>libG4Processes.so</td>
<td>[...] G4ElasticHadNucleusHE::FillFq2</td>
</tr>
<tr>
<td>+ 10.69%</td>
<td>10.30%</td>
<td>libG4Processes.so</td>
<td>[...] G4ElasticHadNucleusHE::HadruNucDiferCrSec</td>
</tr>
<tr>
<td>+ 10.50%</td>
<td>0.00%</td>
<td>libG4run.so</td>
<td>[...] G4RunManager::Initialize</td>
</tr>
<tr>
<td>+ 10.39%</td>
<td>10.38%</td>
<td>libG4Processes.so</td>
<td>[...] G4MuPairProductionModel::ComputeDMicroscopicCrossSection</td>
</tr>
<tr>
<td>+ 9.19%</td>
<td>0.00%</td>
<td>libG4run.so</td>
<td>[...] G4VUserPhysicsList::PreparePhysicsTable</td>
</tr>
<tr>
<td>+ 9.18%</td>
<td>0.00%</td>
<td>libG4Processes.so</td>
<td>[...] G4EmModelManager::Initialise</td>
</tr>
<tr>
<td>+ 8.93%</td>
<td>0.00%</td>
<td>g4run</td>
<td>[...] DetectorConstruction::Construct</td>
</tr>
</tbody>
</table>

Tip: See assembly instructions with percentage: perf annotate <symbol>
DWARF also allows to sort by source line

<table>
<thead>
<tr>
<th>Overhead</th>
<th>Source:Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.50%</td>
<td>G4Log.hh:250</td>
</tr>
<tr>
<td>4.85%</td>
<td>G4Exp.hh:213</td>
</tr>
<tr>
<td>3.64%</td>
<td>G4Log.hh:195</td>
</tr>
<tr>
<td>2.58%</td>
<td>G4Log.hh:258</td>
</tr>
<tr>
<td>2.55%</td>
<td>G4Log.hh:235</td>
</tr>
<tr>
<td>2.39%</td>
<td>G4Log.hh:254</td>
</tr>
<tr>
<td>2.14%</td>
<td>G4Log.hh:253</td>
</tr>
<tr>
<td>2.02%</td>
<td>G4Log.hh:244</td>
</tr>
<tr>
<td>1.93%</td>
<td>G4hPairProductionModel.cc:142</td>
</tr>
<tr>
<td>1.90%</td>
<td>G4hPairProductionModel.cc:157</td>
</tr>
<tr>
<td>1.88%</td>
<td>G4Log.hh:251</td>
</tr>
<tr>
<td>1.83%</td>
<td>G4Exp.hh:210</td>
</tr>
<tr>
<td>1.79%</td>
<td>G4Log.hh:256</td>
</tr>
<tr>
<td>1.45%</td>
<td>G4hPairProductionModel.cc:170</td>
</tr>
<tr>
<td>1.37%</td>
<td>G4Exp.hh:218</td>
</tr>
<tr>
<td>1.36%</td>
<td>G4Log.hh:148</td>
</tr>
<tr>
<td>1.30%</td>
<td>G4Log.hh:248</td>
</tr>
<tr>
<td>1.29%</td>
<td>G4Log.hh:117</td>
</tr>
<tr>
<td>1.20%</td>
<td>G4Log.hh:115</td>
</tr>
<tr>
<td>1.18%</td>
<td>G4hPairProductionModel.cc:140</td>
</tr>
<tr>
<td>1.18%</td>
<td>G4Log.hh:242</td>
</tr>
</tbody>
</table>

Tip: Use parent filter to see specific call path: perf report -p <regex>

Physics models call G4Log and G4Exp many times during initialization, so the results on the left are expected. However, this is also an indication that there may be room for optimization in G4Log and G4Exp as well, since we see backend stall cycles without many L1 cache misses. G4Log at least is also called many times in the event loop to fetch cross section data with energy interpolated as log(E).
Data dependencies seem to be the culprit again

```cpp
G4double px = G4LogConsts::get_log_px(x);

// for the final formula
const G4double x2 = x * x;
px *= x;
px *= x2;

const G4double qx = G4LogConsts::get_log_qx(x);

G4double res = px / qx;
res -= fe * 2.12194440546905827679e-4;
res -= 0.5 * x2;

res = x + res;
res += fe * 0.693359375;

if (original_x > G4LogConsts::LOG_UPPER_LIMIT)
    res = std::numeric_limits<G4double>::infinity();
if (original_x < G4LogConsts::LOG_LOWER_LIMIT)  // THIS IS NaN!
    res = -std::numeric_limits<G4double>::quiet_NaN();

return res;
```

Looking at the code for G4Log, we see that the line with most stalls (G4Log.hh: 250 in previous slide) is a line immediately using the result of a division after it is computed.
DWARF also allows to sort by source line (2)

Same as before, but sampled in the event loop.
G4Log inlined many times, maybe that’s a problem?

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

No big difference, so problem is not due to code bloat
What happens if we use std::log and std::exp?

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

16.2350 +- 0.0009 seconds time elapsed ( +- 0.50% )
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
  ○ Reorder 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
Memory Access and Latency Analysis
Using `perf stat` to measure average load latency

```
```

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

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

20.844816183 seconds time elapsed

148.564065000 seconds user
2.591617000 seconds sys

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

```
bash df102_NanoAODDimuonAnalysis $ perf list mem_load_uops_retired.hit_lfb | uniq
```

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

Metric Groups:
Latency Numbers Every Programmer Should Know

1ns (~5 CPU cycles)

L1 cache reference: 1ns

Branch mispredict: 3ns

L2 cache reference: 4ns

Mutex lock/unlock: 17ns

Main memory reference: 100ns

1,000ns ≈ 1μs

Compress 1KB with Zippy: 2,000ns ≈ 2μs

10,000ns ≈ 10μs =

Send 2,000 bytes over commodity network: 44ns

SSD random read: 16,000ns ≈ 16μs

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

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

1,000,000ns = 1ms =

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

Disk seek: 2,000,000ns ≈ 2ms

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

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

https://colin-scott.github.io/personal_website/research/interactive_latency.html
PERF-MEM(1)

NAME

perf-mem - Profile memory accesses

SYNOPSIS

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

DESCRIPTION

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

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

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

OPTIONS

<command>...

Any command you can specify in a shell.

-i, --input=<file>

Manual page perf-mem(1) line 1 (press h for help or q to quit)
perf – memory access analysis: loads and stores

 Couldn't synthesize cgroup events.
[ perf record: Woken up 130 times to write data ]
[ perf record: Captured and wrote 33.940 MB perf.data (550719 samples) ]
bash df102_NanoAODDimuonAnalysis $ perf mem report -s mem -q --stdio 2>/dev/null

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>38.64%</td>
<td>250297</td>
<td>L1 or L1 hit</td>
</tr>
<tr>
<td>20.24%</td>
<td>121901</td>
<td>L3 or L3 hit</td>
</tr>
<tr>
<td>18.59%</td>
<td>106253</td>
<td>LFB or LFB hit</td>
</tr>
<tr>
<td>15.63%</td>
<td>50825</td>
<td>Remote Remote Cache (1 hop) or L3 hit</td>
</tr>
<tr>
<td>4.22%</td>
<td>9863</td>
<td>L3 miss</td>
</tr>
<tr>
<td>1.45%</td>
<td>6289</td>
<td>Local RAM or RAM hit</td>
</tr>
<tr>
<td>0.99%</td>
<td>3978</td>
<td>Remote Remote RAM (1 hop) or RAM hit</td>
</tr>
<tr>
<td>0.24%</td>
<td>1310</td>
<td>L2 or L2 hit</td>
</tr>
<tr>
<td>0.00%</td>
<td>2</td>
<td>Uncached or N/A hit</td>
</tr>
<tr>
<td>0.00%</td>
<td>1</td>
<td>I/O or N/A hit</td>
</tr>
</tbody>
</table>

Red flag, too many remote cache accesses

Line Fill Buffer (sits between L1 and L2)

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

Samples: 543K of event 'cpu/mem-loads,ldlat=50/P', Event count (approx.): 172369311

<table>
<thead>
<tr>
<th>Overhead</th>
<th>Samples</th>
<th>Memory access</th>
</tr>
</thead>
<tbody>
<tr>
<td>- 41.51%</td>
<td>226882</td>
<td>L1 or L1 hit</td>
</tr>
<tr>
<td>+ 17.49%</td>
<td>0xffffffffffffffff</td>
<td></td>
</tr>
<tr>
<td>- 12.00%</td>
<td>__GI__clone (inlined)</td>
<td></td>
</tr>
<tr>
<td>- 12.00%</td>
<td>start_thread</td>
<td></td>
</tr>
<tr>
<td>- 12.00%</td>
<td>tbb::internal::rml::private_worker::threadRoutine</td>
<td></td>
</tr>
<tr>
<td>- 11.99%</td>
<td>tbb::internal::market::process</td>
<td></td>
</tr>
<tr>
<td>tbb::internal::arena::process</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tbb::internal::custom_scheduler<a href="">tbb::internal::IntelSchedulerTraits</a>::local_wait_for_all</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tbb::internal::custom_scheduler<a href="">tbb::internal::IntelSchedulerTraits</a>::process_bypass_loop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- tbb::interface9::internal::start_for&lt;tbb::blocked_range&lt;unsigned int&gt;, tbb::internal::parallel_for_body&lt;std</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.39%</td>
<td>0x7f4e4344317f</td>
<td></td>
</tr>
<tr>
<td>- 10.18%</td>
<td>0x7f4e424042</td>
<td></td>
</tr>
<tr>
<td>- ROOT::Detail::RDF::RLoopManager::RunAndCheckFilters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ 8.11%</td>
<td>ROOT::Internal::RDF::RAction&lt;ROOT::Internal::RDF::RAction&lt;ROOT::Internal::RDF::FillPa</td>
<td></td>
</tr>
<tr>
<td>+ 2.10%</td>
<td>0x55d581065ec7</td>
<td></td>
</tr>
<tr>
<td>+ 18.96%</td>
<td>113828</td>
<td>LFB or LFB hit</td>
</tr>
<tr>
<td>+ 17.36%</td>
<td>114120</td>
<td>L3 or L3 hit</td>
</tr>
<tr>
<td>+ 13.12%</td>
<td>59491</td>
<td>Remote Remote Cache (1 hop) or L3 hit</td>
</tr>
<tr>
<td>+ 5.86%</td>
<td>14681</td>
<td>N/A miss</td>
</tr>
</tbody>
</table>

Cannot load tips.txt file, please install perf!
perf mem report -s dso,symbol

<table>
<thead>
<tr>
<th>Overhead</th>
<th>Samples</th>
<th>CPU/mem-loads, ldlat=50/P</th>
<th>Event count (approx.): 172369311</th>
</tr>
</thead>
</table>
| 27.12%   | 131151  | df102_NanoAODDimuonAnalysis | [.] ROOT::Detail::RDF::RFilter<bo* (unsigned int), ROOT::Detail::RDF::RVector<
| 17.33%   | 66980   | libTree.so.6.22.02        | [.] TBranch::GetBasketAndFirst    |
| 13.86%   | 62434   | df102_NanoAODDimuonAnalysis | [.] ROOT::Detail::RDF::RFilter<bo* (ROOT::VecOps::RVec<int> const&),
| 5.51%    | 32071   | libTreePlayer.so.6.22.02   | [.] ROOT::Internal::TTreeReaderValueBase::ProxyReadTemplate<&ROOT::Detail::
| 5.33%    | 20471   | libROOTDataFrame.so.6.22.02 | [.] ROOT::Detail::RDF::RLoopManager::RunAndCheckFilters |
| 5.07%    | 21584   | libTree.so.6.22.02         | [.] TBranch::GetEntry             |
| 3.66%    | 24977   | df102_NanoAODDimuonAnalysis | [.] ROOT::Internal::RDF::RColumnValue<ROOT::VecOps::RVec<float> >::Get<R
| 2.76%    | 16451   | libTreePlayer.so.6.22.02   | [.] ROOT::Internal::TTreeReaderValueBase::GetAddress |
| 2.48%    | 18397   | df102_NanoAODDimuonAnalysis | [.] ROOT::Internal::RDF::RColumnValue<ROOT::VecOps::RVec<int> >::Get<R
| 2.28%    | 17109   | df102_NanoAODDimuonAnalysis | [.] std::swap<ROOT::VecOps::RVec<int> > |
| 2.15%    | 16151   | df102_NanoAODDimuonAnalysis | [.] std::swap<ROOT::VecOps::RVec<float> > |
| 2.02%    | 16377   | df102_NanoAODDimuonAnalysis | [.] ROOT::Detail::RDF::RCustomColumn<float (*)(ROOT::VecOps::RVec<float>
| 1.72%    | 12478   | df102_NanoAODDimuonAnalysis | [.] ROOT::Internal::RDF::RActionCRTP<ROOT::Internal::RDF::RAction<
| 1.23%    | 4550    | [kernel.kallsyms]          | [k] copy_user_enhanced_fast_string |
| 1.23%    | 5657    | df102_NanoAODDimuonAnalysis | [.] ROOT::Detail::RDF::RLoopManager::CheckFilters@plt |
| 0.88%    | 5830    | libRIO.so.6.22.02          | [.] TBranchFile::ReadFastArray   |
| 0.67%    | 10998   | df102_NanoAODDimuonAnalysis | [.] ROOT::VecOps::InvariantMass<float> |
| 0.58%    | 8216    | libHist.so.6.22.02         | [.] TH1::Fill                   |
| 0.49%    | 3864    | libTreePlayer.so.6.22.02   | [.] 0x8000000000df63f            |
| 0.36%    | 7884    | libz.so.1.2.11             | [.] inflate_table               |
| 0.32%    | 4976    | libRIO.so.6.22.02          | [.] TBranchFile::ReadArray      |

~46% of high latency loads happen in functions related to RDF filters...
Perf mem report – symbol annotation

<table>
<thead>
<tr>
<th>Percent</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.15</td>
<td>add $0x8,%rsp</td>
</tr>
<tr>
<td>0.68</td>
<td>pop %rbx</td>
</tr>
<tr>
<td>1.23</td>
<td>pop %rbp</td>
</tr>
<tr>
<td>0.04</td>
<td>pop %r12</td>
</tr>
<tr>
<td>0.27</td>
<td>pop %r13</td>
</tr>
<tr>
<td>0.01</td>
<td>pop %r15</td>
</tr>
<tr>
<td>0.00</td>
<td>retq</td>
</tr>
<tr>
<td></td>
<td>nop</td>
</tr>
<tr>
<td>15.13</td>
<td>mov %rdi,%rbx</td>
</tr>
</tbody>
</table>
|         | if (!fPrevData.CheckFilters(slot, entry)) {
|         |   mov $0xf0(%rdi),%rdi       |
|         |   mov %rdx,%rbp               |
|         |   callq ROOT::Detail::RDF::RLoopManager::CheckFilters@plt |
|         |   test %al,%al                |
|         |   jne 26d60 <ROOT::Detail::RDF::RFilter<bool (*)(unsigned int), ROOT::Detail::RDF::RLoopManager::CheckFilters(entry) fLastResult[slot] = false; |
|         |   mov $0x30(%rbx),%rdx        |
|         |   movl $0x0, (%rdx,%r13,4)    |
|         |   fLastCheckedEntry[slot] = entry; |
| 5.40    | mov $0x18(%rbx),%rdx         |

Per-thread values in contiguous memory
perf mem report -s phys_daddr

<table>
<thead>
<tr>
<th>Overhead</th>
<th>Samples</th>
<th>Data</th>
<th>Physical Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.00%</td>
<td>20056</td>
<td>[]</td>
<td>0x0000000e4fff6e08</td>
</tr>
<tr>
<td>3.53%</td>
<td></td>
<td>____</td>
<td>_GI___clone (inlined)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>start_thread</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>tbb::internal::rml::private_worker::threadRoutine</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>tbb::internal::rml::private_worker::run</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>tbb::internal::market::process</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>tbb::internal::arena::process</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>tbb::internal::custom_scheduler<a href="">tbb::internal::IntelSchedulerTraits</a>::local_wait_for_all</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>tbb::internal::custom_scheduler<a href="">tbb::internal::IntelSchedulerTraits</a>::process_bypass_loop</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>tbb::interface9::internal::start_for&lt;tbb::blocked_range&lt;unsigned int&gt;, tbb::internal::parallel_for_body&lt;std::function&lt;void(&gt;</td>
</tr>
<tr>
<td>3.85%</td>
<td>0x7f4e4344317f</td>
<td>0x7f4e42dffa42</td>
<td></td>
</tr>
<tr>
<td>2.20%</td>
<td>ROOT::Detail::RDF::RLoopManager::RunAndCheckFilters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.86%</td>
<td>ROOT::Detail::RDF::RFilter&lt;*(unsigned int), ROOT::Detail::RDF::RLoopManager::CheckFilters</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
+ 3.75%    | 9553    | []   | 0x0000000e4fff6df0 |
+ 2.68%    | 12526   | []   | 0x0000000e4fff6e00 |
+ 2.46%    | 12056   | []   | 0x0000000e4fff6e20 |
+ 2.18%    | 11397   | []   | 0x0000000e4fff6e8 |
+ 2.06%    | 10564   | []   | 0x0000000e4fff6eb0 |
+ 1.48%    | 9597    | []   | 0x000000058bb3c248 |

high load latency happens in nearby addresses
perf – memory access analysis: false sharing

NAME

perf-c2c - Shared Data C2C/HITM Analyzer.

SYNOPSIS

perf c2c record [options] command
perf c2c record [options] -- [record command options] command
perf c2c report [options]

DESCRIPTION

C2C stands for Cache To Cache.

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

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

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

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

See also: https://hpc-wiki.info/hpc/FalseSharing
## perf c2c report

### Shared Data Cache Line Table

<table>
<thead>
<tr>
<th>Index</th>
<th>Address</th>
<th>Node</th>
<th>PA cnt</th>
<th>Cacheline</th>
<th>Total records</th>
<th>Hitm</th>
<th>LLC Load Hitm</th>
<th>Reference</th>
<th>Store</th>
<th>Load</th>
<th>L1</th>
<th>L1Miss</th>
<th>L1Dram</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>0x561f510789c0</td>
<td>0-1</td>
<td>0</td>
<td>7463</td>
<td>13115</td>
<td>17.48%</td>
<td>5172</td>
<td>1368</td>
<td>3804</td>
<td>11</td>
<td>3</td>
<td>9</td>
<td>18</td>
</tr>
<tr>
<td>+</td>
<td>0x561f518b1740</td>
<td>0</td>
<td>0</td>
<td>5382</td>
<td>8536</td>
<td>12.44%</td>
<td>3698</td>
<td>1593</td>
<td>2105</td>
<td>1747</td>
<td>836</td>
<td>911</td>
<td>21</td>
</tr>
<tr>
<td>+</td>
<td>0x561f51078a00</td>
<td>0-1</td>
<td>0</td>
<td>6447</td>
<td>10846</td>
<td>10.82%</td>
<td>3999</td>
<td>892</td>
<td>2297</td>
<td>27</td>
<td>3</td>
<td>24</td>
<td>64</td>
</tr>
<tr>
<td>+</td>
<td>0x561f518b1e40</td>
<td>0</td>
<td>0</td>
<td>5749</td>
<td>8859</td>
<td>9.96%</td>
<td>2962</td>
<td>801</td>
<td>2161</td>
<td>181</td>
<td>18</td>
<td>163</td>
<td>7</td>
</tr>
<tr>
<td>+</td>
<td>0x561f518afdc0</td>
<td>0-1</td>
<td>0</td>
<td>3364</td>
<td>5386</td>
<td>7.34%</td>
<td>2182</td>
<td>738</td>
<td>1444</td>
<td>456</td>
<td>45</td>
<td>411</td>
<td>16</td>
</tr>
<tr>
<td>+</td>
<td>0x561f518b1a40</td>
<td>0</td>
<td>0</td>
<td>4888</td>
<td>7554</td>
<td>7.25%</td>
<td>2154</td>
<td>721</td>
<td>1433</td>
<td>2927</td>
<td>1244</td>
<td>1683</td>
<td>16</td>
</tr>
<tr>
<td>+</td>
<td>0x561f518b1b80</td>
<td>0</td>
<td>0</td>
<td>2923</td>
<td>4846</td>
<td>6.74%</td>
<td>2804</td>
<td>617</td>
<td>1387</td>
<td>36</td>
<td>1</td>
<td>35</td>
<td>19</td>
</tr>
<tr>
<td>+</td>
<td>0x561f518b1800</td>
<td>0</td>
<td>0</td>
<td>3320</td>
<td>5532</td>
<td>6.39%</td>
<td>1900</td>
<td>544</td>
<td>1356</td>
<td>1333</td>
<td>591</td>
<td>742</td>
<td>10</td>
</tr>
<tr>
<td>+</td>
<td>0x561f51078a40</td>
<td>0-1</td>
<td>0</td>
<td>3543</td>
<td>5982</td>
<td>4.45%</td>
<td>1322</td>
<td>352</td>
<td>970</td>
<td>21</td>
<td>5</td>
<td>16</td>
<td>9</td>
</tr>
<tr>
<td>+</td>
<td>0x561f518b1ac0</td>
<td>0</td>
<td>0</td>
<td>1446</td>
<td>2186</td>
<td>1.38%</td>
<td>387</td>
<td>73</td>
<td>314</td>
<td>868</td>
<td>327</td>
<td>541</td>
<td>16</td>
</tr>
<tr>
<td>+</td>
<td>0x561f518b2640</td>
<td>0-1</td>
<td>0</td>
<td>1364</td>
<td>2150</td>
<td>1.09%</td>
<td>324</td>
<td>10</td>
<td>314</td>
<td>290</td>
<td>54</td>
<td>236</td>
<td>3</td>
</tr>
<tr>
<td>+</td>
<td>0x561f518b2580</td>
<td>0-1</td>
<td>0</td>
<td>1711</td>
<td>2348</td>
<td>0.94%</td>
<td>279</td>
<td>181</td>
<td>98</td>
<td>402</td>
<td>165</td>
<td>297</td>
<td>4</td>
</tr>
<tr>
<td>+</td>
<td>0x561f518b1a80</td>
<td>0</td>
<td>0</td>
<td>510</td>
<td>1815</td>
<td>0.83%</td>
<td>247</td>
<td>9</td>
<td>238</td>
<td>734</td>
<td>148</td>
<td>586</td>
<td>3</td>
</tr>
<tr>
<td>+</td>
<td>0x561f518b24c0</td>
<td>0-1</td>
<td>0</td>
<td>1113</td>
<td>1535</td>
<td>0.82%</td>
<td>245</td>
<td>145</td>
<td>100</td>
<td>278</td>
<td>88</td>
<td>190</td>
<td>4</td>
</tr>
<tr>
<td>+</td>
<td>0x561f518b34c0</td>
<td>0-1</td>
<td>0</td>
<td>991</td>
<td>1990</td>
<td>0.66%</td>
<td>196</td>
<td>1</td>
<td>195</td>
<td>1108</td>
<td>1106</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>+</td>
<td>0x561f518b1840</td>
<td>0</td>
<td>0</td>
<td>347</td>
<td>1244</td>
<td>0.57%</td>
<td>170</td>
<td>7</td>
<td>163</td>
<td>346</td>
<td>66</td>
<td>289</td>
<td>2</td>
</tr>
<tr>
<td>+</td>
<td>0x561f518b2540</td>
<td>0-1</td>
<td>0</td>
<td>1358</td>
<td>2220</td>
<td>0.57%</td>
<td>170</td>
<td>0</td>
<td>170</td>
<td>906</td>
<td>984</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>+</td>
<td>0x561f518b2680</td>
<td>0-1</td>
<td>0</td>
<td>1723</td>
<td>2943</td>
<td>0.54%</td>
<td>160</td>
<td>1</td>
<td>159</td>
<td>1668</td>
<td>1470</td>
<td>198</td>
<td>6</td>
</tr>
<tr>
<td>+</td>
<td>0x561f518b3040</td>
<td>0-1</td>
<td>0</td>
<td>866</td>
<td>1830</td>
<td>0.50%</td>
<td>148</td>
<td>1</td>
<td>147</td>
<td>1950</td>
<td>1050</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>+</td>
<td>0x561f518b1ec0</td>
<td>0</td>
<td>0</td>
<td>17115</td>
<td>30175</td>
<td>0.49%</td>
<td>147</td>
<td>23</td>
<td>124</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>400</td>
</tr>
</tbody>
</table>

? - help
### perf c2c report

<table>
<thead>
<tr>
<th>Index</th>
<th>Address</th>
<th>Node</th>
<th>PA cnt</th>
<th>LLC records</th>
<th>Total Hitm</th>
<th>Tot</th>
<th>LLC Load Hitm</th>
<th>L1 Hit</th>
<th>L1 Miss</th>
<th>Load Dram</th>
<th>Lcl</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0</td>
<td>0x561f510789c0</td>
<td>0-1</td>
<td>7463</td>
<td>13115</td>
<td>17.40%</td>
<td>5172</td>
<td>1368</td>
<td>3804</td>
<td>11</td>
<td>3</td>
<td>9</td>
</tr>
</tbody>
</table>
+ 0.00% ROOT::Detail::RDF::RFilter<bool (*)(unsigned int), ROOT::Detail::RDF::RLoopManager>::CheckFilters
+ 0.00% ROOT::Detail::RDF::RLoopManager::CheckFilters@plt
0.00% apic_timer_interrupt
+ 1    | 0x561f518b1740  | 0    | 5382   | 8536        | 12.44%     | 3698 | 1593          | 2105    | 1747    | 836       | 911 |
+ 2    | 0x561f51078a00  | 0-1  | 6447   | 10046       | 10.42%     | 3099 | 892           | 2207    | 27      | 3         | 24  |
+ 3    | 0x561f518b1e40  | 0    | 5749   | 8859        | 9.96%      | 2962 | 801           | 2161    | 181     | 18        | 163 |
+ 4    | 0x561f518afdc0  | 0-1  | 3304   | 5386        | 7.34%      | 2182 | 738           | 1444    | 456     | 45        | 411 |
+ 5    | 0x561f518b1a40  | 0    | 4888   | 7554        | 7.25%      | 2154 | 721           | 1433    | 2927    | 1244      | 1683 |
+ 6    | 0x561f518b1b80  | 0    | 2923   | 4846        | 6.74%      | 2004 | 617           | 1387    | 36      | 1         | 35  |
+ 7    | 0x561f518b1800  | 0    | 3320   | 5532        | 6.39%      | 1900 | 544           | 1356    | 1333    | 591       | 742 |
+ 8    | 0x561f51078a40  | 0-1  | 3543   | 5982        | 4.45%      | 1322 | 352           | 970     | 21      | 5         | 16  |
+ 9    | 0x561f518b1ac0  | 0    | 1446   | 2186        | 1.38%      | 387  | 73            | 314     | 868     | 327       | 541 |
+ 10   | 0x561f518b2640  | 0-1  | 1364   | 2150        | 1.09%      | 324  | 10            | 314     | 290     | 54        | 236 |
+ 11   | 0x561f518b2580  | 0-1  | 1711   | 2348        | 0.94%      | 279  | 181           | 98      | 402     | 185       | 297 |
+ 12   | 0x561f518b1a80  | 0    | 510    | 1815        | 0.83%      | 247  | 9             | 238     | 734     | 148       | 586 |
+ 13   | 0x561f518b24c0  | 0-1  | 1113   | 1535        | 0.82%      | 245  | 145           | 190     | 278     | 88        | 190 |
+ 14   | 0x561f518b34c0  | 0-1  | 991    | 1990        | 0.66%      | 196  | 1             | 195     | 1108    | 1106      | 2   |
+ 15   | 0x561f518b1840  | 0    | 347    | 1244        | 0.57%      | 170  | 7             | 163     | 346     | 66        | 280 |
+ 16   | 0x561f518b2540  | 0-1  | 1358   | 2220        | 0.57%      | 170  | 0             | 170     | 906     | 904       | 2   |

? - help
perf c2c report – cacheline details

<table>
<thead>
<tr>
<th>Cacheline 0x561f510789c0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code address</td>
</tr>
<tr>
<td>0x561f4ebc2470 408 284 339 1002</td>
</tr>
<tr>
<td>0x561f4ebd0d23 515 327 320 540</td>
</tr>
<tr>
<td>0x561f4ebd0c8f 464 314 322 247</td>
</tr>
<tr>
<td>0x561f4ebd0d9e 360 241 309 173</td>
</tr>
<tr>
<td>0x561f4ebd0cfb 641 241 656 130</td>
</tr>
<tr>
<td>0x561f4ebd0d05 337 164 364 48</td>
</tr>
<tr>
<td>0x561f4ebd0d12 332 173 202 39</td>
</tr>
<tr>
<td>0x561f4ebd0d10 306 150 118 17</td>
</tr>
<tr>
<td>0x561f4ebd0d20 312 0 0 4</td>
</tr>
<tr>
<td>0x561f4ebd0d46 0 0 0 3</td>
</tr>
<tr>
<td>0x561f4ebc2470 389 283 343 934</td>
</tr>
<tr>
<td>0x561f4ebd0d23 467 372 247 529</td>
</tr>
<tr>
<td>0x561f4ebd0c8f 484 381 330 216</td>
</tr>
<tr>
<td>0x561f4ebd0d9e 394 181 384 178</td>
</tr>
<tr>
<td>0x561f4ebd0cfb 466 293 699 129</td>
</tr>
<tr>
<td>0x561f4ebd0d05 349 299 435 37</td>
</tr>
<tr>
<td>0x561f4ebd0d12 301 176 208 26</td>
</tr>
<tr>
<td>0x561f4ebd0d10 296 0 220 12</td>
</tr>
<tr>
<td>0x561f4ebd0d03 275 0 440 3</td>
</tr>
<tr>
<td>0x561f4ebd0d20 0 0 0 1</td>
</tr>
</tbody>
</table>

? - help
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Source:Line</th>
<th>Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>df102_NanoAODDimuonAnalysis</td>
<td>RFFilter.hxx:87</td>
<td>1</td>
</tr>
<tr>
<td>df102_NanoAODDimuonAnalysis</td>
<td>RFFilter.hxx:86</td>
<td>1</td>
</tr>
<tr>
<td>df102_NanoAODDimuonAnalysis</td>
<td>RFFilter.hxx:99</td>
<td>1</td>
</tr>
<tr>
<td>df102_NanoAODDimuonAnalysis</td>
<td>RFFilter.hxx:86</td>
<td>1</td>
</tr>
<tr>
<td>df102_NanoAODDimuonAnalysis</td>
<td>RFFilter.hxx:86</td>
<td>1</td>
</tr>
<tr>
<td>df102_NanoAODDimuonAnalysis</td>
<td>RFFilter.hxx:99</td>
<td>1</td>
</tr>
<tr>
<td>df102_NanoAODDimuonAnalysis</td>
<td>RFFilter.hxx:99</td>
<td>1</td>
</tr>
<tr>
<td>df102_NanoAODDimuonAnalysis</td>
<td>RFFilter.hxx:99</td>
<td>1</td>
</tr>
<tr>
<td>df102_NanoAODDimuonAnalysis</td>
<td>RFFilter.hxx:96</td>
<td>1</td>
</tr>
<tr>
<td>df102_NanoAODDimuonAnalysis</td>
<td>RFFilter.hxx:96</td>
<td>1</td>
</tr>
<tr>
<td>df102_NanoAODDimuonAnalysis</td>
<td>RFFilter.hxx:87</td>
<td>0</td>
</tr>
<tr>
<td>df102_NanoAODDimuonAnalysis</td>
<td>RFFilter.hxx:86</td>
<td>0</td>
</tr>
<tr>
<td>df102_NanoAODDimuonAnalysis</td>
<td>RFFilter.hxx:99</td>
<td>0</td>
</tr>
<tr>
<td>df102_NanoAODDimuonAnalysis</td>
<td>RFFilter.hxx:86</td>
<td>0</td>
</tr>
<tr>
<td>df102_NanoAODDimuonAnalysis</td>
<td>RFFilter.hxx:86</td>
<td>0</td>
</tr>
<tr>
<td>df102_NanoAODDimuonAnalysis</td>
<td>RFFilter.hxx:99</td>
<td>0</td>
</tr>
<tr>
<td>df102_NanoAODDimuonAnalysis</td>
<td>RFFilter.hxx:99</td>
<td>0</td>
</tr>
<tr>
<td>df102_NanoAODDimuonAnalysis</td>
<td>RFFilter.hxx:86</td>
<td>0</td>
</tr>
<tr>
<td>df102_NanoAODDimuonAnalysis</td>
<td>RFFilter.hxx:99</td>
<td>1</td>
</tr>
</tbody>
</table>
perf c2c report – cacheline details, expanded stack

```
Cacheline 0x561f510789c0

<table>
<thead>
<tr>
<th>Rmt</th>
<th>Lcl</th>
<th>L1 Hit</th>
<th>L1 Miss</th>
<th>Off</th>
<th>Node</th>
<th>PA cnt</th>
<th>Code</th>
<th>address</th>
<th>rmthit</th>
<th>lclhit</th>
<th>cycles</th>
<th>load</th>
<th>Total records</th>
<th>cpu cnt</th>
</tr>
</thead>
</table>
+    | 0.00% | 0.00% | 0.00% | 11.11% | 0x18 | 0 | 1 | 0x561f4ebd0d46 | 0 | 0 | 0 | 0 | 1 | 1 |
-    | 7.23% | 7.16% | 0.00% | 0.00% | 0x20 | 0-1 | 3 | 0x561f4ebc2470 | 413 | 257 | 333 | 891 | 3 |

ROOT::Detail::RDF::RLoopManager::CheckFilters@plt
ROOT::Detail::RDF::RFFilter<bool (*)(unsigned int), ROOT::Detail::RDF::RLoopManager>::::CheckFilters
ROOT::Detail::RDF::RFFilter<bool (*)(ROOT::VecOps::RVec<int> const&), ROOT::Detail::RDF::RFFilter<bool (*)(unsigned int), ROOT::Detail::RDF::RFFilter<bool (*)(unsigned int), ROOT::Detail::RDF::RFActionCRTP<ROOT::Internal::RDF::RFAction<ROOT::Internal::RDF::RFFillParHelper<TH1D>, ROOT::Detail::RDF::RLoopManager::RunAndCheckFilters
0x7feaa2613a42
0x7feaa2c5717f
tbb::interface9::internal::start_for<tbb::blocked_range<unsigned int>, tbb::internal::parallel_for_body<std::function<void(tbb::internal::custom_scheduler<tbb::internal::IntelSchedulerTraits>::process_bypass_loop
> >
tbb::internal::custom_scheduler<tbb::internal::IntelSchedulerTraits>::::local_wait_for_all

+ 3.63% | 3.36% | 0.00% | 0.00% | 0x20 | 0-1 | 3 | 0x561f4ebd0d23 | 469 | 375 | 334 | 501 | 3 |
+ 1.42% | 2.56% | 0.00% | 0.00% | 0x20 | 0-1 | 3 | 0x561f4ebd0d0e | 353 | 217 | 282 | 222 | 3 |
```

? - help
Top-down Microarchitecture Analysis
Microarchitecture Utilization Overview from VTune

Cycles split into categories based on type of bottleneck

Front-End Bound (instruction cache misses): 28.1%
Back-End Bound (cache misses, divisions): 49.0%
Retiring (useful work): 31.7%
Bad Speculation (branch mispredictions): 6.5%

Geant4 CMS Example

Retiring: 31.7% of Pipeline Slots
Front-End Bound: 28.1% of Pipeline Slots
Front-End Latency: 19.9% of Pipeline Slots
ICache Misses: 11.7% of Clockticks
ITLB Overhead: 3.3% of Clockticks
Branch Resteer: 13.4% of Clockticks
DSB Switches: 1.6% of Clockticks
Length Changing Prefixes: 0.0% of Clockticks
MS Switches: 0.3% of Clockticks
Front-End Bandwidth: 8.2% of Pipeline Slots
Bad Speculation: 6.5% of Pipeline Slots
Back-End Bound: 33.7% of Pipeline Slots
Memory Bound: 49.0% of Pipeline Slots
L1 Bound: 41.8% of Clockticks
DTLB Overhead: 20.3% of Clockticks
Loads Blocked by Store Forwarding: 6.3% of Clockticks
Lock Latency: 0.0% of Clockticks
Split Loads: 0.0% of Clockticks
4K Aliasing: 0.9% of Clockticks
FB Full: 0.3% of Clockticks
L2 Bound: 0.0% of Clockticks
L3 Bound: 16.8% of Clockticks
Contested Accesses: 0.0% of Clockticks
Data Sharing: 0.0% of Clockticks
L3 Latency: 30.0% of Clockticks
SQ Full: 0.1% of Clockticks
DRAM Bound: 1.1% of Clockticks
Store Bound: 4.0% of Clockticks
Core Bound: 0.0% of Pipeline Slots

Intel and AMD Microarchitectures

- Front End
  - Instruction Fetch and Decode
  - Branch Predator Unit
  - 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

source: https://en.wikichip.org
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
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 occurrence rate)]

IpCall
[Instruction per (near) call (lower number means higher occurrence rate)]

IpL
[Instructions per Load (lower number means higher occurrence 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':

<table>
<thead>
<tr>
<th>Counter</th>
<th>Value</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>cycles</td>
<td>535853293</td>
<td>0.32</td>
</tr>
<tr>
<td>idq_uops_not_delivered.core</td>
<td>676507752</td>
<td>0.10</td>
</tr>
<tr>
<td>uops_issued.any</td>
<td>803157447</td>
<td>0.28</td>
</tr>
<tr>
<td>cycles</td>
<td>540449552</td>
<td>0.31</td>
</tr>
<tr>
<td>idq_uops_not_delivered.core</td>
<td>676523326</td>
<td></td>
</tr>
<tr>
<td>int_misc.recovery_cycles</td>
<td>19393734</td>
<td></td>
</tr>
<tr>
<td>uops_retired.retire_slots</td>
<td>667220596</td>
<td></td>
</tr>
</tbody>
</table>

0.243072802 seconds time elapsed

0.158384000 seconds user
0.088028000 seconds sys

bash ~ $
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 relevant metrics

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 relevant metrics

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 relevant metrics

Example – using perf + awk to get percent retiring

```bash
```

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 (406808 samples) ]

```bash
df102_NanoAODDimuonAnalysis $ perf report -q --stdio --group -F period,symbol -w 0,90 | head
104728157092  9740014622  [. ROOT::Detail::RDF::RFilter<bool (*)(ROOT::VecOps::RVec<int> const&), ROOT::Detail::RDF::RLoopManager>::C
94152141228  10108015162  [. ROOT::Detail::RDF::RFilter<bool (*)(unsigned int), ROOT::Detail::RDF::RLoopManager>::C
79494119241  3454005181  [. TT::LoadTree
51302076953  92238138357  [. inflate_fast
3569805347  14764022146  [. TBranch::GetEntry
24610036915  2248003372  [. TLeaf::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::ReadNoParentNoBranchCountNoCollection&>__unknown
27.16% __exp1f
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% sysretry_check
5.58% ROOT::Detail::RDF::RColumnColumn<float (*)(ROOT::VecOps::RVec<float> const&, ROOT::VecOps::RVec<float> const&, ROOT::VecOps::RVec<int> const&)>::Get<ROOT::VecOps::RVec<float>, 0>
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)>::Get<ROOT::VecOps::RVec<int>, 0>
2.28% TLeafI::GetMaximum
1.09% TTree::LoadTree
bash df102_NanoAODDimuonAnalysis $ _
```
Thread State and Scheduling Analysis
Linux Thread States

Uninterruptible Sleep (D)
- disk I/O
- sleep, lock, wait
- wakeup

Runnable (R)
- schedule
- preemption or time slice expired
- new process or thread

Running (R)
- wakeup
- exit()

Zombie (Z)
- SIGCONT

Interruptible Sleep (S)
- wakeup

Stopped (T)
- SIGSTOP
**perf sched – thread state analysis**

**NAME**
perf-sched - Tool to trace/measure scheduler properties (latencies)

**SYNOPSIS**
```
perf sched {record|latency|map|replay|script|timehist}
```

**DESCRIPTION**
There are several variants of `perf sched`:

- 'perf sched record <command>}' to record the scheduling events of an arbitrary workload.

- 'perf sched latency' to report the per task scheduling latencies and other scheduling properties of the workload.

- 'perf sched script' to see a detailed trace of the workload that was recorded (aliased to 'perf script' for now).

- 'perf sched replay' to simulate the workload that was recorded via perf sched record. (This is done by starting up mockup threads that mimic the workload based on the events in the trace. These
perf sched – thread state analysis


Couldn't synthesize cgroup events.
[ perf record: Woken up 284 times to write data ]
Warning:
Processed 7321327 events and lost 7 chunks!

Check I/O/CPU overload!

[ perf record: Captured and wrote 719.341 MB perf.data (6677842 samples) ]
bash df102_NanoAODDimuonAnalysis $ perf sched latency -s runtime | head

Warning:
Processed 7321327 events and lost 7 chunks!

Check I/O/CPU overload!

<table>
<thead>
<tr>
<th>Task</th>
<th>Runtime ms</th>
<th>Switches</th>
<th>Average delay ms</th>
<th>Maximum delay ms</th>
<th>Maximum delay at</th>
</tr>
</thead>
<tbody>
<tr>
<td>df102_NanoAODDi:(9)</td>
<td>158948.839 ms</td>
<td>17477</td>
<td>avg: 0.002 ms</td>
<td>max: 0.448 ms</td>
<td>max at: 2268553.824795 s</td>
</tr>
<tr>
<td>perf:(2)</td>
<td>3803.596 ms</td>
<td>325</td>
<td>avg: 0.003 ms</td>
<td>max: 0.042 ms</td>
<td>max at: 2268567.710095 s</td>
</tr>
<tr>
<td>kworker/u65:19993</td>
<td>189.326 ms</td>
<td>15</td>
<td>avg: 0.012 ms</td>
<td>max: 0.023 ms</td>
<td>max at: 2268565.899074 s</td>
</tr>
<tr>
<td>:31412:31412</td>
<td>160.115 ms</td>
<td>2159</td>
<td>avg: 0.009 ms</td>
<td>max: 0.048 ms</td>
<td>max at: 2268551.465497 s</td>
</tr>
</tbody>
</table>
### perf sched – thread state analysis

```
bash df102_NanoAODDimuonAnalysis $ perf sched timehist -s -p 1354 2>&1| grep -v lost
```

**Runtime summary**

<table>
<thead>
<tr>
<th>comm</th>
<th>parent</th>
<th>sched-in (count)</th>
<th>run-time (msec)</th>
<th>min-run (msec)</th>
<th>avg-run (msec)</th>
<th>max-run (msec)</th>
<th>stddev</th>
<th>migrations</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;no still running tasks&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Terminated tasks:**

1. df102_NanoAODDi[1354]:
   - comm: 1353
   - parent: 8229
   - run-time: 18639.702
   - min-run: 0.007
   - avg-run: 2.265
   - max-run: 3958.990
   - stddev: 30.66
   - migrations: 0

2. df102_NanoAODDi[1379/1354]:
   - comm: 1354
   - parent: 486
   - run-time: 17533.934
   - min-run: 0.010
   - avg-run: 36.078
   - max-run: 1887.005
   - stddev: 25.12
   - migrations: 0

3. df102_NanoAODDi[1380/1354]:
   - comm: 1354
   - parent: 395
   - run-time: 17583.592
   - min-run: 0.005
   - avg-run: 44.515
   - max-run: 3999.970
   - stddev: 36.89
   - migrations: 0

4. df102_NanoAODDi[1381/1354]:
   - comm: 1379
   - parent: 588
   - run-time: 17389.895
   - min-run: 0.009
   - avg-run: 29.574
   - max-run: 2001.972
   - stddev: 21.78
   - migrations: 0

5. df102_NanoAODDi[1382/1354]:
   - comm: 1380
   - parent: 436
   - run-time: 16938.839
   - min-run: 0.008
   - avg-run: 38.850
   - max-run: 3999.982
   - stddev: 43.02
   - migrations: 0

6. df102_NanoAODDi[1383/1354]:
   - comm: 1381
   - parent: 484
   - run-time: 17399.590
   - min-run: 0.006
   - avg-run: 35.949
   - max-run: 3487.010
   - stddev: 35.70
   - migrations: 0

7. df102_NanoAODDi[1384/1354]:
   - comm: 1380
   - parent: 521
   - run-time: 17526.195
   - min-run: 0.008
   - avg-run: 33.639
   - max-run: 639.979
   - stddev: 15.84
   - migrations: 0

8. df102_NanoAODDi[1385/1354]:
   - comm: 1379
   - parent: 517
   - run-time: 17249.654
   - min-run: 0.009
   - avg-run: 33.364
   - max-run: 2898.975
   - stddev: 29.19
   - migrations: 0

```
bash df102_NanoAODDimuonAnalysis $ _
```
Final Thoughts
“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