Linux Perf advancements for compute intensive and server systems

BoF
Linux Plumbers 2019
Summary

- Asynchronous record trace streaming
- NUMA awareness in record profiling
- Runtime record trace compression
- Extended event naming in stat and record modes
- Typed context switches
- Perf privileged user groups
- Parallel record trace streaming
Asynchronous record trace streaming

The kernel can lose profiling data when it’s rate and volume is high:

```
--aio[=<n>]
  Use <n> control blocks in asynchronous trace writing mode
  (default: 1, max: 4)
```

Example: matrix multiplication executing 128 threads on Intel Xeon Phi

```
$ perf record -a -N -B -T -R -g -F K --call-graph dwarf,1024 --user-regs=IP,SP,BP \
  --switch-events -e cycles,instructions,ref-cycles,\ 
  software/period=1,name=cs,config=0x3/Duk \ 
  [--aio=N] -- matrix.gcc.O3
```

Metrics: % data_loss = paused_time / elapsed_time
Asynchronous record trace streaming

paused time

elapsed time

~2x data loss decrease!
NUMA awareness in record profiling

Up to 30% runtime overhead when a compute intensive workload fully utilizes a large server system with NUMA.

--affinity <node|cpu>
set affinity mask of trace reading thread to NUMA node CPU mask or the CPU of processed event buffer

Example:
Perl workload thread
Intel Broadwell [0-43] [0]
2 sockets
44 cores

--affinity node : cpu mask = [0-21] | [22-43]
--affinity cpu : cpu mask = [0] | [1] | [2] ... | [43]
NUMA awareness in record profiling

NASA Parallel BT benchmark:

```bash
$ perf record -a -N -B -T -R -g -F K --call-graph dwarf,1024 --user-regs=IP,SP,BP \ 
--switch-events -e cycles,instructions,ref-cycles,\ 
software/period=1,name=cs,config=0x3/Duk \ 
[--aio] [--affinity=node|cpu] -- bt-mz.B
```

<table>
<thead>
<tr>
<th>v4.20.0-rc5+</th>
<th>BENCH REPORT BASED</th>
<th>ELAPSED TIME BASED</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERIAL-SYS / BASE</td>
<td>1,27x (14,37/11,31)</td>
<td>1,29x (15,19/11,69)</td>
</tr>
<tr>
<td>SERIAL-NODE / BASE</td>
<td>1,15x (13,04/11,31)</td>
<td>1,17x (13,79/11,69)</td>
</tr>
<tr>
<td>SERIAL-CPU / BASE</td>
<td>1,00x (11,32/11,31)</td>
<td>1,01x (11,89/11,69)</td>
</tr>
<tr>
<td>AIO1-SYS / BASE</td>
<td>1,29x (14,58/11,31)</td>
<td>1,29x (15,26/11,69)</td>
</tr>
<tr>
<td>AIO1-NODE / BASE</td>
<td>1,08x (12,23/11,31)</td>
<td>1,11x (13,01/11,69)</td>
</tr>
<tr>
<td>AIO1-CPU / BASE</td>
<td>1,07x (12,14/11,31)</td>
<td>1,08x (12,83/11,69)</td>
</tr>
</tbody>
</table>
Runtime record trace compression

~3-5x average trace size reduction, especially when profiling with dwarf call stacks and context switches:

-z, --compression-level[=<n>]
Compress records using specified level
(default: 1 - fastest compression, 22 - greatest compression)

--mmap-flush <number>
Minimal number of bytes that is extracted from mmap data pages
(default: 1)

<table>
<thead>
<tr>
<th></th>
<th>SERIAL</th>
<th></th>
<th>AIO1</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ovh (x)</td>
<td>Ratio (x)</td>
<td>Ovh (x)</td>
<td>Ratio(x)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1,00</td>
<td>1,000</td>
<td>1,00</td>
<td>1,000</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1,04</td>
<td>8,427</td>
<td>1,01</td>
<td>8,474</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1,07</td>
<td>8,055</td>
<td>1,03</td>
<td>7,912</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1,04</td>
<td>8,283</td>
<td>1,03</td>
<td>8,220</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1,09</td>
<td>8,101</td>
<td>1,05</td>
<td>7,780</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1,05</td>
<td>9,217</td>
<td>1,12</td>
<td>6,111</td>
<td></td>
</tr>
</tbody>
</table>
Extended event naming

$ perf stat -e cpu/name="CPU_CLK_UNHALTED.THREAD:cmask=0x1\",
    period=0x3567e0,event=0x3c,cmask=0x1/Duk -- futex

$ perf record -e
    cpu/name="OFFCORE_RESPONSE:request=DEMAND_RFO:response=L3_HIT.SNOOP_HITM\",
    period=0x3567e0,event=0x3c,cmask=0x1/Duk -- futex

# Samples: 24K of event \\n"OFFCORE_RESPONSE:request=DEMAND_RFO:response=L3_HIT.SNOOP_HITM" # Event count \\
(approx.): 86492000000 #
# Overhead Command Shared Object Symbol
# ........ ........ ........................ ................................................
#
14.75% futex [kernel.vmlinux] [k] __entry_trampoline_start
Typed context switches

<table>
<thead>
<tr>
<th>Grouping</th>
<th>Function</th>
<th>Call Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inactive Sync Wait Count</td>
<td>System wide: Up to ~1,5x on desktop</td>
<td>Preemption Wait Count</td>
</tr>
<tr>
<td>CPU Time</td>
<td>Inactive Wait Count</td>
<td>Inactive Sync Wait Count</td>
</tr>
<tr>
<td>0.002s</td>
<td>33.885s</td>
<td>4</td>
</tr>
<tr>
<td>0s</td>
<td>33.885s</td>
<td>4</td>
</tr>
</tbody>
</table>

System wide:
- Up to ~1.5x on desktop
- Up to ~2.0x on server

Per process:
- Up to ~3.0x

$ perf record [-a] -N -B -T -R -F K -g --call-graph dwarf,1024 --switch-events \ -e cycles,instructions,ref-cycles,\ software/period=1,name=cs,config=0x3/Duk -- <futex|sleep0>

<table>
<thead>
<tr>
<th>Hardware</th>
<th>OS</th>
<th>v4.17.0+</th>
<th>Perf pp</th>
<th>Perf sw</th>
<th>Perf pp</th>
<th>Perf sw</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Skylake 8C</td>
<td>Fedora 25</td>
<td>2,85x</td>
<td>1,96x</td>
<td>0,92x</td>
<td>0,87x</td>
<td></td>
</tr>
<tr>
<td>Intel Skylake EP 48C 2S</td>
<td>Ubuntu 17.04</td>
<td>2,80x</td>
<td>1,43x</td>
<td>1,30x</td>
<td>1,07x</td>
<td></td>
</tr>
<tr>
<td>Intel Broadwell 8C</td>
<td>Ubuntu 16.04 LTS</td>
<td>2,50x</td>
<td>1,81x</td>
<td>0,99x</td>
<td>0,91x</td>
<td></td>
</tr>
<tr>
<td>Intel Broadwell EP 44C 2S</td>
<td>Ubuntu server 17.04</td>
<td>5,50x</td>
<td>1,40x</td>
<td>1,22x</td>
<td>1,06x</td>
<td></td>
</tr>
</tbody>
</table>
Perf privileged user groups

- **Privileged processes**
  - euid == 0, CAP_SYS_ADMIN etc.

- **Perf privileged processes**
  - access to performance monitoring
  - bypass `perf_event_paranoid`
  - and `system limits`
  - configured and controlled by root:
    
    ```
    # ls -alhF
    -rwxr-x---  2 root perf_users 11M Oct 19 15:12 perf
    # getcap perf
    perf = cap_sys_ptrace,cap_sys_admin,cap_syslog+ep
    ```

- **Unprivileged processes**
  - access to performance monitoring bound by `perf_event_paranoid`
  - and `system limits`

*Replacing CAP_SYS_ADMIN with CAP_SYS_PTRACE or CAP_SYS_PERFMON?*
Parallel record trace streaming

$ perf record --threads --dir --output-dir

• record part prototype currently exists, no compression yet
• outperforms SERIAL and AIO in terms of runtime overhead, up to ~20%:
  • NASA parallel BT benchmark
  • 14 events with sampling interval 1ms on client and 5ms on server
  • dwarf stack size up to 32KiB

report part is still at PoC stage

some approach is required to load Perf data directories of ~200GiB
Q/A

Special thanks to:

Andi Kleen, Peter Zijlstra, Arnaldo Carvalho D’melo, Jiri Olsa, Thomas Gleixner, Case Kook, Jonatan Corbet, Namhun Kim, Alex Shishkin, Mark Rutland and others ...

Linux Perf users: linux-perf-users@vger.kernel.org
Linux Perf developers: linux-kernel@vger.kernel.org
Contact: Alexey Budankov <alexey.budankov@linux.intel.com>
Backup
### Typed context switches

<table>
<thead>
<tr>
<th>Function / Call Stack</th>
<th>CPU Time</th>
<th>Inactive Sync Wait Time</th>
<th>Preemption Wait Count</th>
<th>Preemption Wait Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>schedule</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>exit_to_usermode_loop</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_prepare_exit_to_usermode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_do_syscall64_entry_SYSCALL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_write</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_readl64</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_lock_page_failsafe</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_futex_wait_queue_me</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Inactive Sync Wait Count**
- Inactive Sync Wait Count is the number of context switches a thread experiences when it is excluded from execution by the OS scheduler due to synchronization.
- Excessive number of thread context switches may negatively impact application performance. Apply optimization techniques to reduce synchronization contention and eliminate the problem.

**Preemption Wait Count**
- Preemption Wait Count is the number of context switches a thread experiences when it is excluded from execution by the OS scheduler due to thread preemption.
- Excessive number of thread context switches may negatively impact application performance. Apply optimization techniques to reduce synchronization contention and eliminate the problem. Explore Total Thread count and eliminate thread oversubscription changing the number of threads in the application.

**Thread (TID: 31475)**
- Context Switches:
  - Start: 8.3385s
  - Duration: 18.05ms
  - Reason: Preemption
  - CPU Time: 5.2%
  - Spin and Overhead Time: 0.0%

**Thread (TID: 31474)**
- Context Switches:
  - Start: 8.3385s
  - Duration: 18.05ms
  - Reason: Spin and Overhead
  - CPU Time: 5.2%
  - Spin and Overhead Time: 0.0%

**Thread (TID: 31476)**
- Context Switches:
  - Start: 8.3385s
  - Duration: 18.05ms
  - Reason: Spin and Overhead
  - CPU Time: 5.2%
  - Spin and Overhead Time: 0.0%
Parallel perf-record trace streaming

$ perf record --threads --dir --output-dir ___bt.$stack_size.data \
-N -B -T -R --call-graph dwarf,$stack_size --user-regs=ip,bp,sp \n-e 'cpu/period=P,event=0x3c,name="CPU_CLK_UNHALTED.THREAD"/Duk,\n  cpu/period=P,umask=0x3,name="CPU_CLK_UNHALTED.REF_TSC"/Duk,\n  cpu/period=P,event=0xc0,name="INST_RETIRED.ANY"/Duk,\n  cpu/period=P,event=0x3c,umask=0x1,name="CPU_CLK_UNHALTED.REF_XCLK"/uk,\n  cpu/period=P,event=0x3c,umask=0x2,name="CPU_CLK_UNHALTED.ONE_THREAD_ACTIVE"/uk,\n  cpu/period=P,event=0xc2,umask=0x2,name="UOPS_RETIRED.RETIRE_SLOTS"/uk,\n  cpu/period=P,event=0xc7,umask=0x2,name="FP_ARITH_INST_RETIRED.SCALAR_SINGLE"/uk,\n  cpu/period=P,event=0xc7,umask=0x8,name="FP_ARITH_INST_RETIRED.128B_PACKED_SINGEL"/uk,\n  cpu/period=P,event=0xc7,umask=0x20,name="FP_ARITH_INST_RETIRED.256B_PACKED_SINGLE"/uk,\n  cpu/period=P,event=0xc7,umask=0x1,name="FP_ARITH_INST_RETIRED.SCALAR_DOUBLE"/uk,\n  cpu/period=P,event=0xc7,umask=0x4,name="FP_ARITH_INST_RETIRED.128B_PACKED_DOUBLE"/uk,\n  cpu/period=P,event=0xb1,umask=0x10,name="UOPS_EXECUTED.THREAD"/uk,\n  cpu/period=P,event=0xb1,umask=0x1,name="UOPS_EXECUTED.X87"/uk,\n  cpu/period=P,event=0xb1,umask=0x1,name="UOPS_EXECUTED.THREAD"/uk\n--clockid=monotonic_raw -- bt-mz.<B|C>
Augmented call stacks

$ perf record --g --call-graph dwarf,SIZE -j stack,u

Limitations:
1. available for HW events only
2. branch stack size is limited in HW
3. kernel samples can lack branch stack
4. C/C++ exceptions are not handled
5. branch stack is shorter in system wide mode
6. runtime overhead
Patches

AIO: https://marc.info/?l=linux-kernel&m=154149439404555&w=2
NUMA: https://marc.info/?l=linux-kernel&m=154817912621465&w=2
Compression: https://marc.info/?l=linux-kernel&m=155293062518459&w=2
Event names: https://marc.info/?l=linux-kernel&m=152809506802631&w=2
Typed context switches: https://marc.info/?l=linux-kernel&m=152325846408261&w=2
1ms uncore printing: https://marc.info/?l=linux-kernel&m=152277952316450&w=2
frame pointer on sample: https://marc.info/?l=linux-kernel&m=152717113521558&w=2
clockid frequency: https://marc.info/?l=linux-kernel&m=153909580104443&w=2
Optimized register set capture: https://marc.info/?l=linux-kernel&m=155594387916976&w=2
Augmented call stacks: https://marc.info/?l=linux-kernel&m=156536377224480&w=2
Decompression fix: https://marc.info/?l=linux-kernel&m=156268372122638&w=2
Overhead mitigation: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=486adcea4a63bec206cba6f0d7f301fb945ae9d3