THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
KOREA

서울

Advanced Performance Profiling
Using Perf Tools

Namhyung Kim

About me

- Namhyung Kim (김남형)
  - Co-maintainer of the perf tools
  - Creator of uftrace
  - Google prodkernel team

Agenda

- Data type profiling
  - Understand memory access patterns
- Latency profiling
  - Find out latency bottleneck of parallel programs
- Kernel lock contention profiling
  - Lightweight lock profiling

Data type profiling

Data type profiling

- Profile data access using DWARF
  - Sampling memory instructions precisely
  - Analyze base and offset to type and field
  - Report with the type sort keys
  - [OSDI '26] TypeCraft paper by Zecheng Li, et al.
- Status
  - Mostly tested on x86 kernels (in C)
  - Powerpc support added
  - ARM64 support is WIP
  - C++ support is WIP

perf mem

- Specialized wrapper for memory events
  - Basically for **perf record** and **perf report**
  - Use per-platform events for precise memory access
  - Show details about memory access info
    - Data source, latency, snoop, ...

perf report

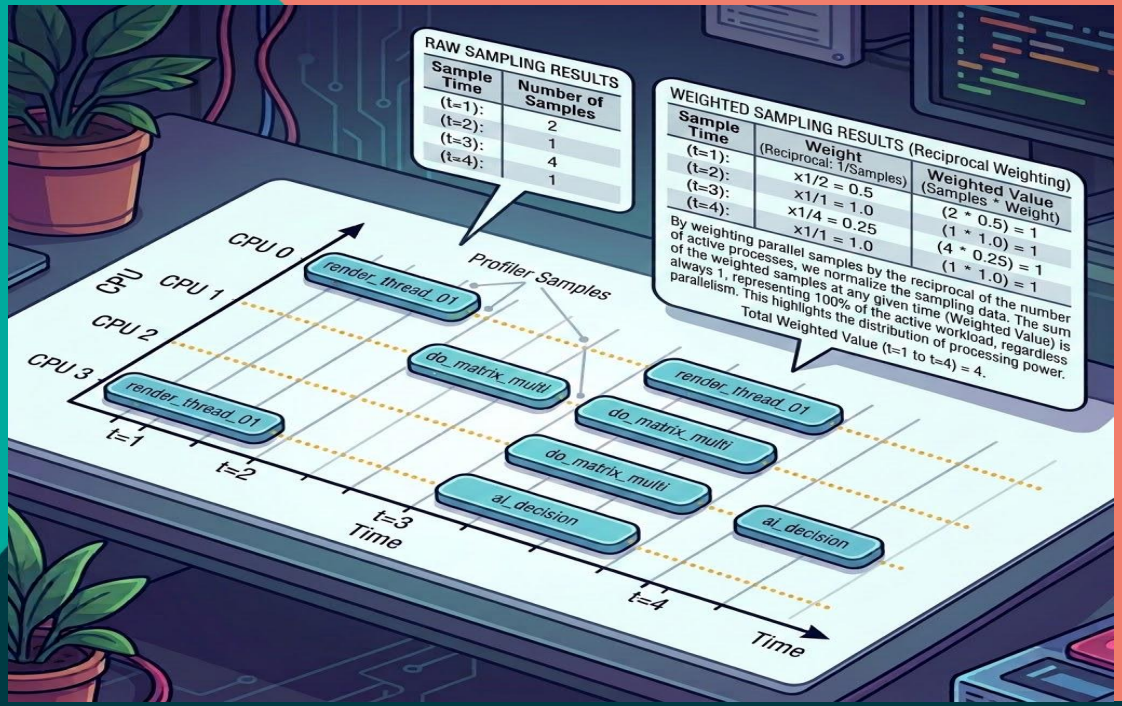
- Process samples
  - Use sort key to get desired output (-s)
    - Default is “comm, dso, symbol”
  - Hierarchy mode to group samples (-H)
  - New sort keys for memory analysis
    - type, typeoff, typecln

Example output

- Collected on my laptop
  - perf mem record -a sleep 3
  - perf mem report -s dso,type,typeoff -H

```
Samples: 4K of event 'cpu_atom/mem-loads,ldlat=30/P', Event count (approx.): 384815
  Overhead       Samples        Shared Object / Data Type / Data Type Offset
-   51.20%          2417        [kernel.kallsyms]
   +   16.81%           790           (unknown)
   +    2.21%           111           long unsigned int
   -    2.06%            79           struct folio
            1.62%            70              struct folio +0 (flags.f)
            0.35%             2              struct folio +0x38 (memcg_data)
            0.08%             3              struct folio +0x34 (_refcount.counter)
            0.01%             2              struct folio +0x18 (mapping)
            0.00%             2              struct folio +0x1 (flags.f)
   +    1.88%            78           struct task_struct
   +    0.95%            63           struct audit_entry
   +    0.88%           117           (stack operation)
   +    0.85%            40           struct cfs_rq
   +    0.78%            18           struct sched_avg
   +    0.77%            35           struct sched_entity
   +    0.72%            57           int
   +    0.67%            10           void*
Tip: To set sampling period of individual events use perf record -e cpu/cpu-cycles,period=100001
```

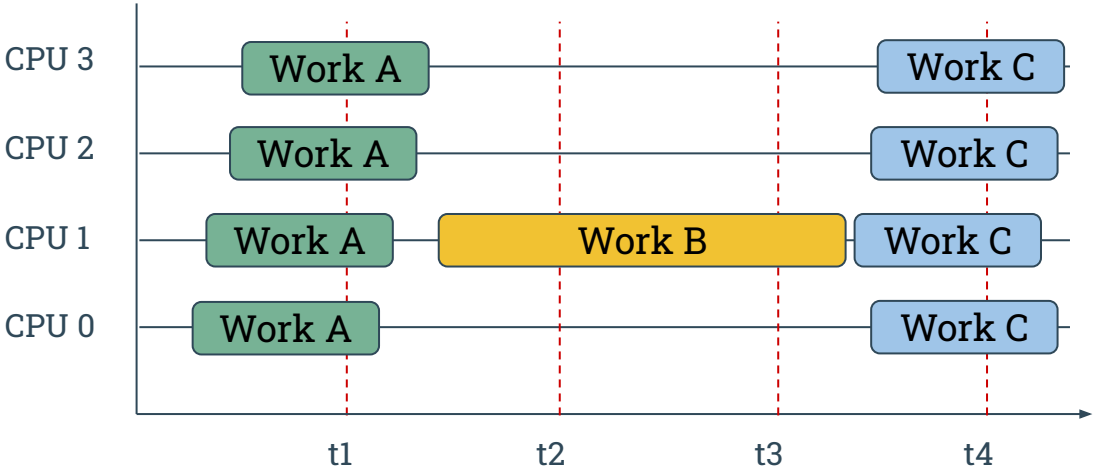
Latency profiling



Latency profiling

- Weight samples by parallelism
  - To find serial portion of execution
  - Track sched-switch to get parallelism
- Status
  - Works best for a new process
  - Proposed an idea for system-wide profiling

Latency profiling



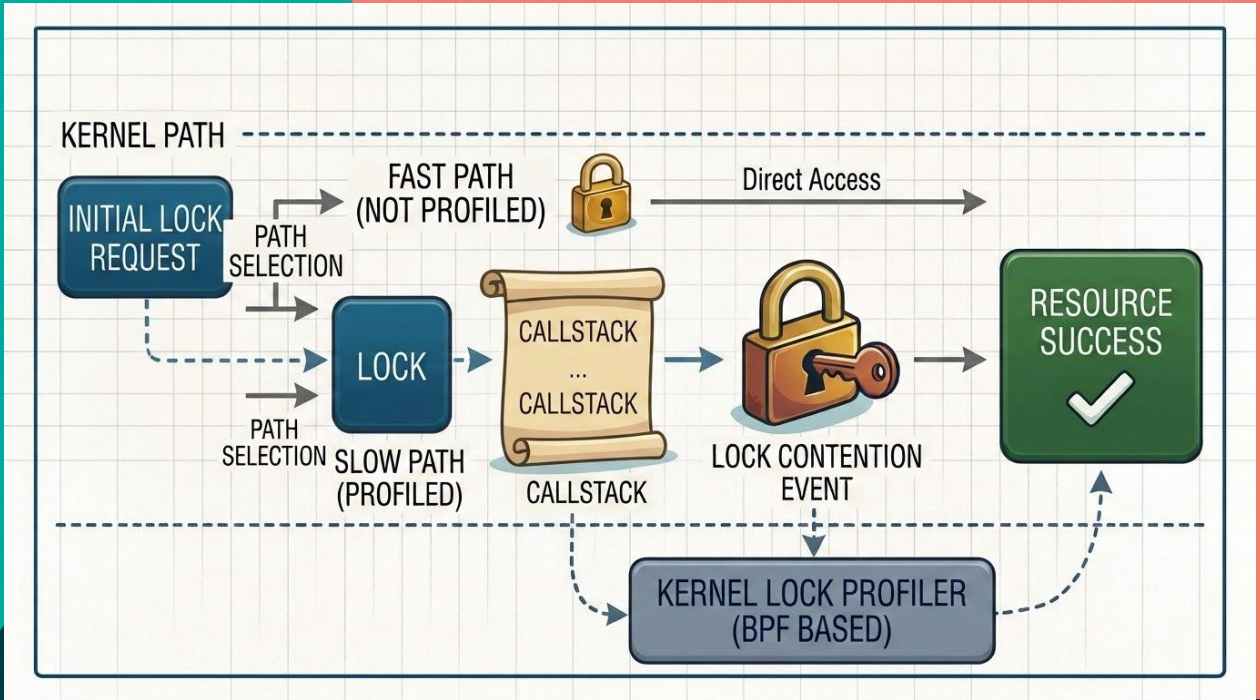
	CPU time	Parallelism	Weight	Latency
Work A	4 (40%)	4	0.25	1 (25%)
Work B	2 (20%)	1	1	2 (50%)
Work C	4 (40%)	4	0.25	1 (25%)

Example output

- Build performance of perf tools (v7.0)
  - perf record --latency -- make -C tools/perf
  - perf report --latency -s comm

```
Samples: 724K of event 'cpu/cycles/Pu', Event count (approx.): 419474386837
 Latency  Overhead  Command
  46.00%    71.54%  cc1
  33.59%    16.59%  python3
  11.19%     2.37%  ld
   2.42%     1.46%  x86_64-linux-gn
   2.18%     1.01%  cc1plus
   0.94%     0.79%  sh
   0.70%     0.15%  llvm-config
   0.61%     0.55%  clang
   0.54%     4.10%  shellcheck
   0.42%     0.12%  perl
   0.40%     0.26%  make
   0.20%     0.10%  ar
   0.16%     0.36%  gcc
   0.10%     0.02%  which
   0.09%     0.08%  awk
   0.08%     0.07%  grep
   0.08%     0.07%  sed
Tip: To record every process run by a user: perf record -u <user>
```

Kernel lock contention profiling



Kernel lock contention profiling

- Trace contentions with small overhead
  - Tracepoints only in the slow path
  - Do not touch uncontended locks
  - Do not increase size of locks
  - BPF for in-kernel aggregation

Lock symbolization

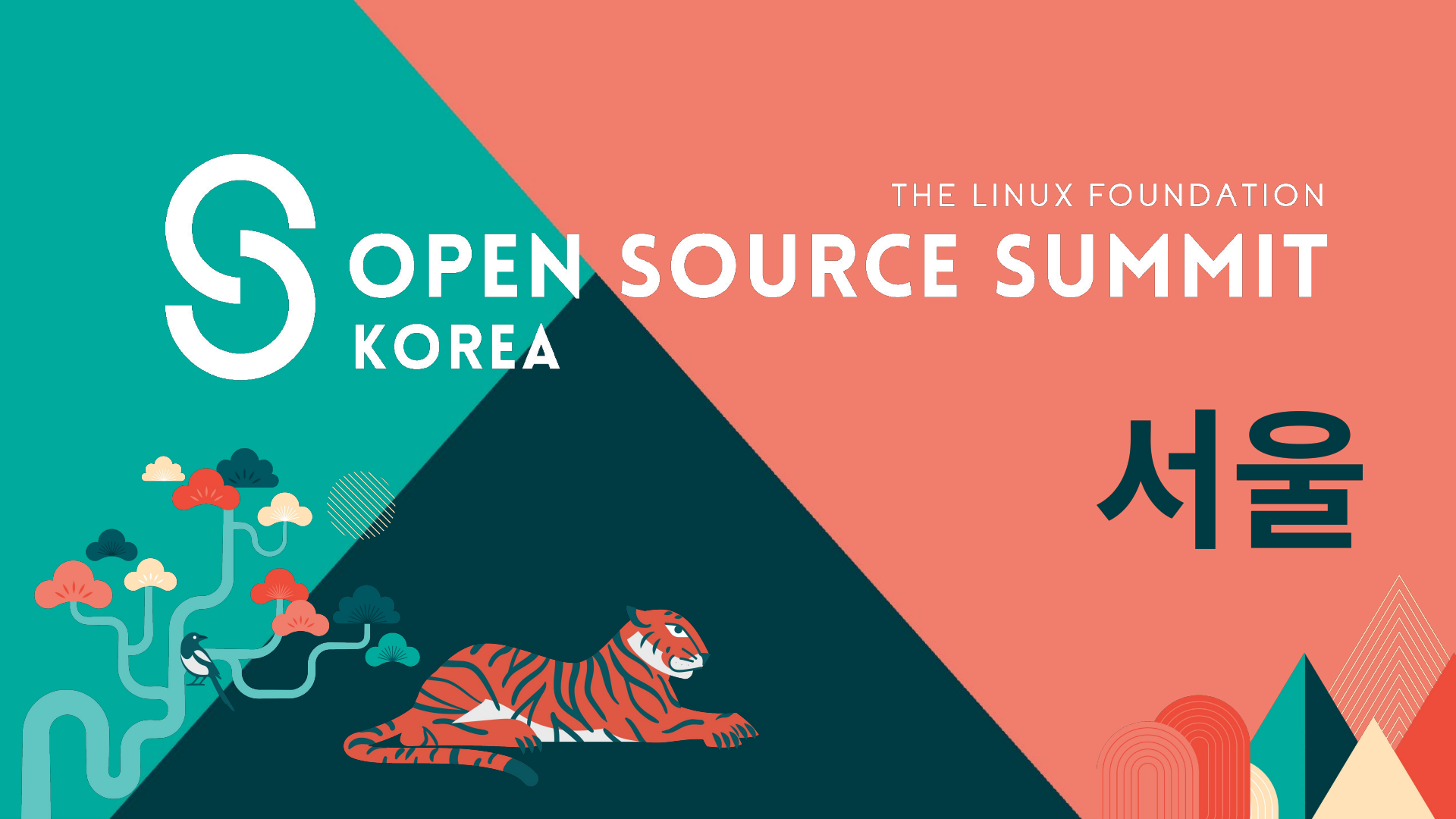
- How to display contentions
  - Global (static) locks have symbols
  - Dynamic locks?
    - Runtime resolution using BPF
      - Some well-known locks (like mmap-lock, rq-lock, zone-lock)
      - Slab allocated locks
    - Still some missing locks
  - Use call stacks instead
    - Pick a caller of lock functions
  - Other options
    - By task or cgroup

Example output

- Default mode
  - perf lock contention -ab sleep 1

```
 contended   total wait     max wait     avg wait         type   caller

         2    740.54 us    449.69 us    370.27 us        mutex   __i915_gem_object_unset_pages+0x1e8
         1    523.33 us    523.33 us    523.33 us        mutex   i915_gem_object_ggtt_pin_ww+0x18f
         4    346.27 us     92.64 us     86.57 us     spinlock   tick_do_update_jiffies64+0x25
         3    263.01 us     89.44 us     87.67 us     spinlock   tmigr_handle_remote+0x17b
         4    219.00 us     87.23 us     54.75 us     spinlock   rcu_core+0xaf
         2    114.42 us     58.86 us     57.21 us     spinlock   tick_do_update_jiffies64+0x25
         2     17.33 us     10.90 us      8.67 us        mutex   __i915_gem_object_unset_pages+0x1e8
```


THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
KOREA
서울