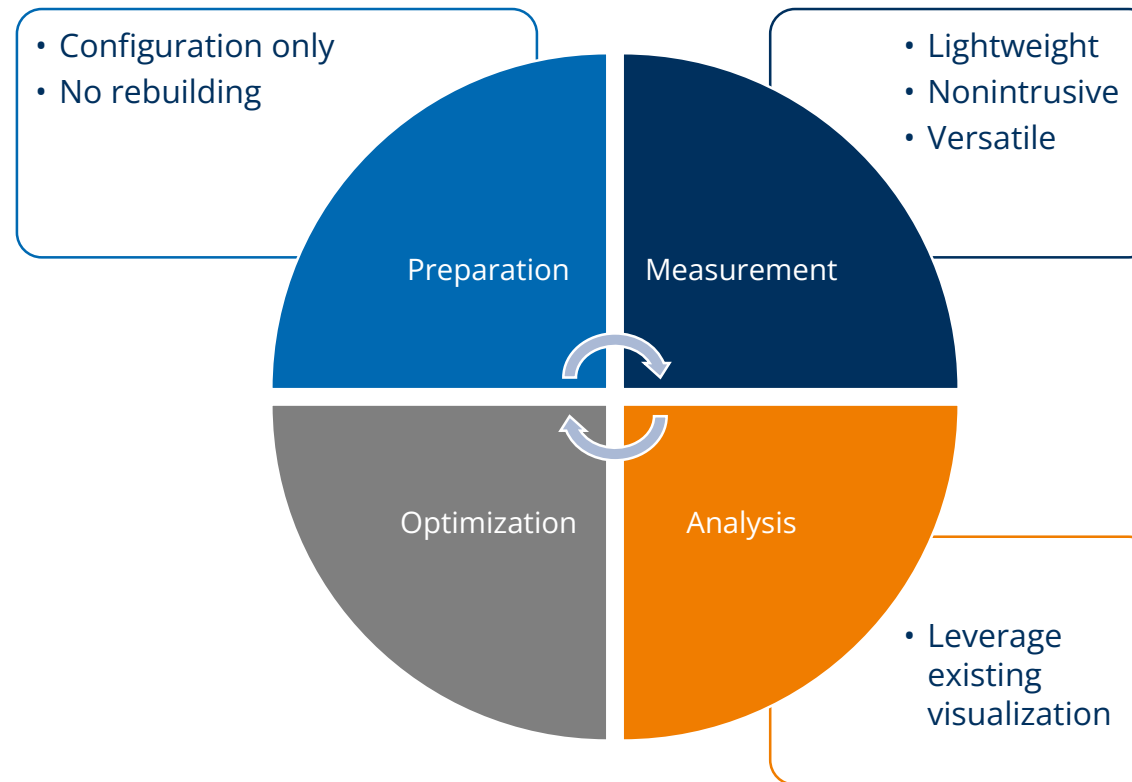


 ZIH
Center for Information Services
and High Performance Computing

Node-Level Analysis using Io2s

VI-HPS, March 2024

Lo2s Advantages in the Performance Engineering Workflow



Technology

Based centrally around Linux `perf_event_open` system call

- Requires kernel 4.3 or newer
- Required features are backported often times

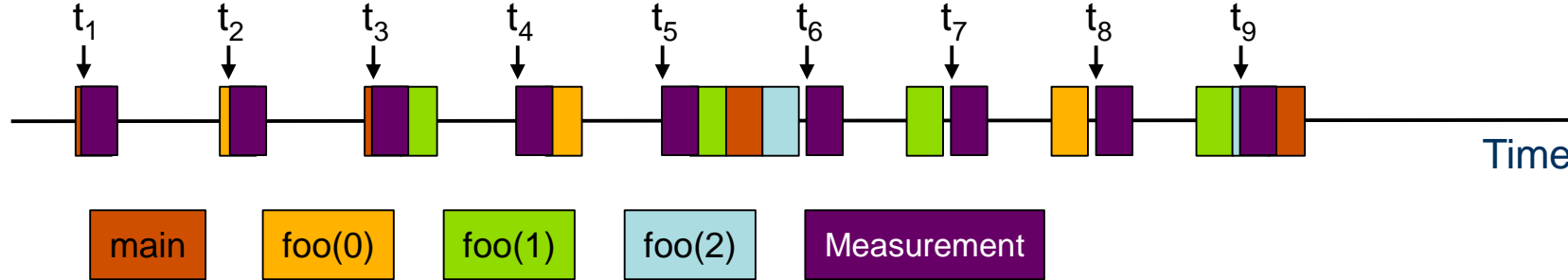
Scalable parallel recording

- Separate independent monitoring threads
- No explicit runtime synchronization

Writes OTF2 (Open Trace Format 2) Traces

- Parallel output and file format
- Leverage Vampir for visualization

Reminder: Sampling



```
int main()
{
    int i;

    for (i=0; i < 3; i++)
        foo(i);

    return 0;
}

void foo(int i)
{
    if (i > 0)
        foo(i - 1);
}
```

Running program is periodically interrupted to take measurement

- Timer interrupt, OS signal, or HWC overflow
- Service routine examines return-address stack
- Addresses are mapped to routines using symbol table information

Statistical inference of program behaviour

- Not very detailed information on highly volatile metrics
- Requires long-running applications

Works with unmodified executables

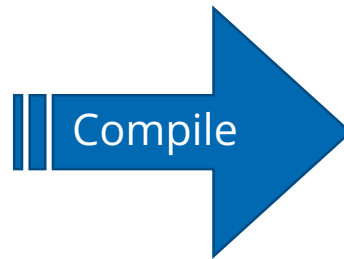
Instruction Sampling

```
int main()
{
    int i;

    for (i=0; i < 3; i++)
        foo(i);

    return 0;
}

void foo(int i)
{
    if (i > 0)
        foo(i - 1);
}
```



```
1  main:
2      push    rbp
3      mov     rbp, rsp
4      sub     rsp, 16
5      mov     DWORD PTR [rbp-4], 0
6      jmp     .L2
7  .L3:
8      mov     eax, DWORD PTR [rbp-4]
9      mov     edi, eax
10     mov     eax, 0
11     call    foo
12     add     DWORD PTR [rbp-4], 1
13  .L2:
14     cmp     DWORD PTR [rbp-4], 2
15     jle     .L3
16     mov     eax, 0
17     leave
18     ret
19  foo:
20     push    rbp
21     mov     rbp, rsp
22     sub     rsp, 16
23     mov     DWORD PTR [rbp-4], edi
24     cmp     DWORD PTR [rbp-4], 0
25     jle     .L7
26     mov     eax, DWORD PTR [rbp-4]
27     sub     eax, 1
28     mov     edi, eax
29     call    foo
30  .L7:
31     nop
32     leave
33     ret
```

Compiler Explorer: <https://godbolt.org/>

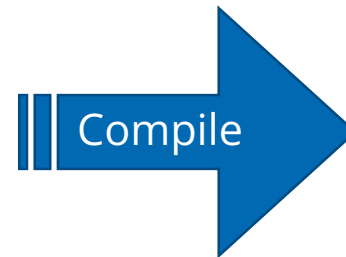
Instruction Sampling

```
int main()
{
    int i;

    for (i=0; i < 3; i++)
        foo(i);

    return 0;
}

void foo(int i)
{
    if (i > 0)
        foo(i - 1);
}
```



```
1  main:
2      push    rbp
3      mov     rbp, rsp
4      sub     rsp, 16
5      mov     DWORD PTR [rbp-4], 0
6      jmp     .L2
7  .L3:
8      mov     eax, DWORD PTR [rbp-4]
9      mov     edi, eax
10     mov     eax, 0
11     call    foo
12     add     DWORD PTR [rbp-4], 1
13  .L2:
14     cmp     DWORD PTR [rbp-4], 2
15     jle     .L3
16     mov     eax, 0
17     leave
18     ret
19  foo:
20     push    rbp
21     mov     rbp, rsp
22     sub     rsp, 16
23     mov     DWORD PTR [rbp-4], edi
24     cmp     DWORD PTR [rbp-4], 0
25     jle     .L7
26     mov     eax, DWORD PTR [rbp-4]
27     sub     eax, 1
28     mov     edi, eax
29     call    foo
30  .L7:
31     nop
32     leave
33     ret
```

Compiler Explorer: <https://godbolt.org/>

Instruction Sampling

Sampling based on perf event

- Instructions, cycles, ...

Record instruction pointer in kernel

- Call stack if available

Buffers are read on demand

- Readout triggered by watermark
- Readout interval can be set for fine-tuning

```
1 main:
2   push  rbp
3   mov   rbp, rsp
4   sub   rsp, 16
5   mov   DWORD PTR [rbp-4], 0
6   jmp  .L2
7 .L3:
8   mov   eax, DWORD PTR [rbp-4]
9   mov   edi, eax
10  mov   eax, 0
11  call  foo
12  add   DWORD PTR [rbp-4], 1
13 .L2:
14  cmp   DWORD PTR [rbp-4], 2
15  jle   .L3
16  mov   eax, 0
17  leave
18  ret
19 foo:
20  push  rbp
21  mov   rbp, rsp
22  sub   rsp, 16
23  mov   DWORD PTR [rbp-4], edi
24  cmp   DWORD PTR [rbp-4], 0
25  jle   .L7
26  mov   eax, DWORD PTR [rbp-4]
27  sub   eax, 1
28  mov   edi, eax
29  call  foo
30 .L7:
31  nop
32  leave
33  ret
```

Basic Usage

- Can be used as prefix command to any command

```
# lo2s ... -- /path/to/executable arg0 arg1
```

All argument after -- belong to the application

- Gives a summary at the end

```
# lo2s -- sleep 10  
[ lo2s: Child exited. Stopping measurements and closing trace. ]  
[ lo2s: sleep 10 (0), 1 threads, 0.014507s CPU, 10.0118s total ]  
[ lo2s: 5 wakeups, wrote 2.94 KiB lo2s_trace_2022-03-28T17-27-51 ]
```

- Recording can be interrupted at any point using ctrl-C

```
# lo2s -- sleep 10  
^C[ lo2s: Child exited. Stopping measurements and closing trace. ]  
[ lo2s: sleep 10 (0), 1 threads, 0.018307s CPU, 2.7757s total ]  
[ lo2s: 6 wakeups, wrote 2.98 KiB lo2s_trace_2022-03-28T17-28-05 ]
```


Required permissions

— Check sysctl setting:

```
# sudo sysctl kernel.perf_event_paranoid  
kernel.perf_event_paranoid = 3
```

```
# cat /proc/sys/kernel/perf_event_paranoid  
3
```

— Update sysctl setting:

```
# sudo sysctl kernel.perf_event_paranoid=XXX  
kernel.perf_event_paranoid = XXX
```

— Where XXX allows you:

- 2: Sample your own application w/o kernel
- 1: Sample what own application with kernel
- 0: Allow system-wide measurements (no tracepoints!)
- -1: Allow system-wide measurements (with tracepoints!)

Additional requirements

For tracepoints:

- Mount debugfs:

```
# sudo mount -t debugfs none /sys/kernel/debug  
# sudo chmod -R og+rX /sys/kernel/debug
```

For BPF based measurements (POSIX I/O)

- root is required!
- No perf_event Paranoid equivalent for BPF yet

Process-mode and System-mode

- lo2s can operate in two basic modes: per-process recording and system-wide mode

Per-process mode

```
# Will record per-process information for /path/to/executable  
# lo2s ... -- /path/to/executable arg0 arg1
```

System-wide monitoring

```
# Will run until Ctrl-C is pressed  
# lo2s -a ...  
# Will run until /path/to/executable is finished  
# lo2s -a ... -- /path/to/executable
```

System-wide monitoring with function samples

```
# lo2s -A ...
```

- Some of the things shown later will only be available in one of the modes

Perf Metrics

- Large list of hardware metrics, such as L1-Cache misses etc.

```
# lo2s --list-events
```

```
List of predefined events:
```

```
...  
cache-misses  
...
```

- Use `-E` to record the specified metric

```
# lo2s -E cache-misses ...
```

- Use `--standard-metrics` to record a set of default metrics

```
# lo2s --standard-metrics ...
```

Userspace perf Metric events

- Uses a more compatible but slower/higher overhead read-out mode

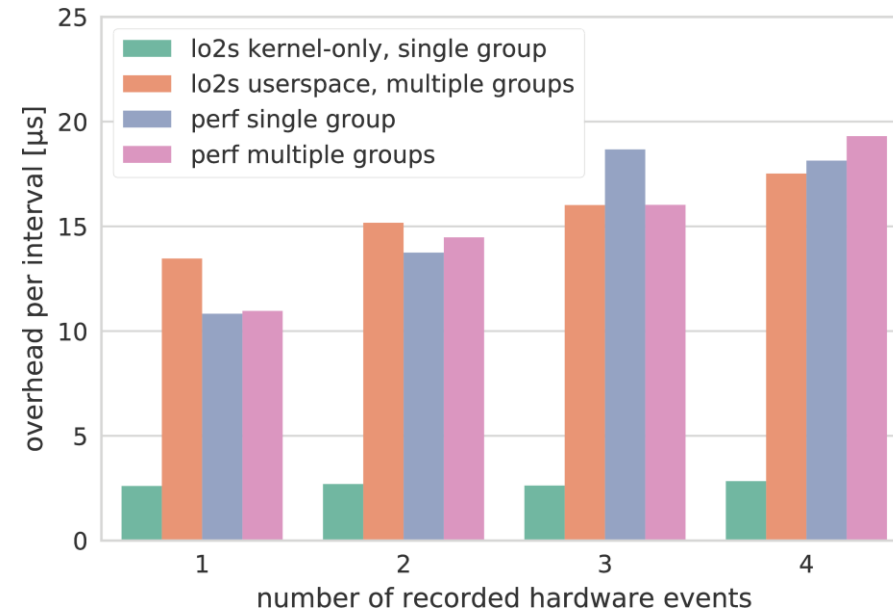
```
# lo2s -userspace-metric-event power/energy-pkg
```

- Try to use this option if opening a metric the standard way fails

Reducing Measurement Overhead of **perf** Metrics

- Grouping performance events
- Group leader determines readout rate
- Kernel collects counter in ring buffer
- Flush buffer when it reaches watermark

- Reduces overall overhead
- No overhead for additional hardware events



Tracepoints

Linux tracepoints

- Leverage large amount of kernel instrumentation
- Records all fields with a numerical value
- Requires read access to debugfs at `/sys/kernel/debug/tracing`

```
# lo2s --list-tracepoints  
  
List of Kernel tracepoint events:  
  
...  
sched:sched_switch  
...
```

- Use `-t` to record the specified tracepoint

```
# lo2s -t sched:sched_switch ...
```

Additional Metrics

Available in both monitoring modes

- x86_energy
 - Access to RAPL power metrics
 - https://github.com/tud-zih-energy/x86_energy
- x86_adapt
 - Generic access to hardware specific information (MSRs)
 - https://github.com/tud-zih-energy/x86_adapt

Additional Metrics

Available in both monitoring modes

- Plugins
 - Compatible with Score-P plugins (asynchronous, per-host)
 - Leverage existing plugins (e.g. PAPI, CPU energy counter) <https://github.com/score-p>

Using with 1o2s specific environment variables

- LO2S_METRIC_PLUGINS and LO2S_{\$METRIC}_PLUGIN

... or compatible to Score-P environment variables

- SCOREP_METRIC_PLUGINS and SCOREP_{\$METRIC}_PLUGIN

Additional Metrics

Available in both monitoring modes

- Sensors read-out with `lm_sensors`
 - Data from common hardware sensors such as fan-speed, cpu and mainboard temperature sensors etc.

```
# Read Sensors  
# ls2s -S ...
```

Additional Metrics for I/O

POSIX I/O:

- Track POSIX I/O calls (read, open, write etc.) without instrumentation
- Based on BPF, so requires root

```
# lo2s -posix-io ...
```

Block I/O:

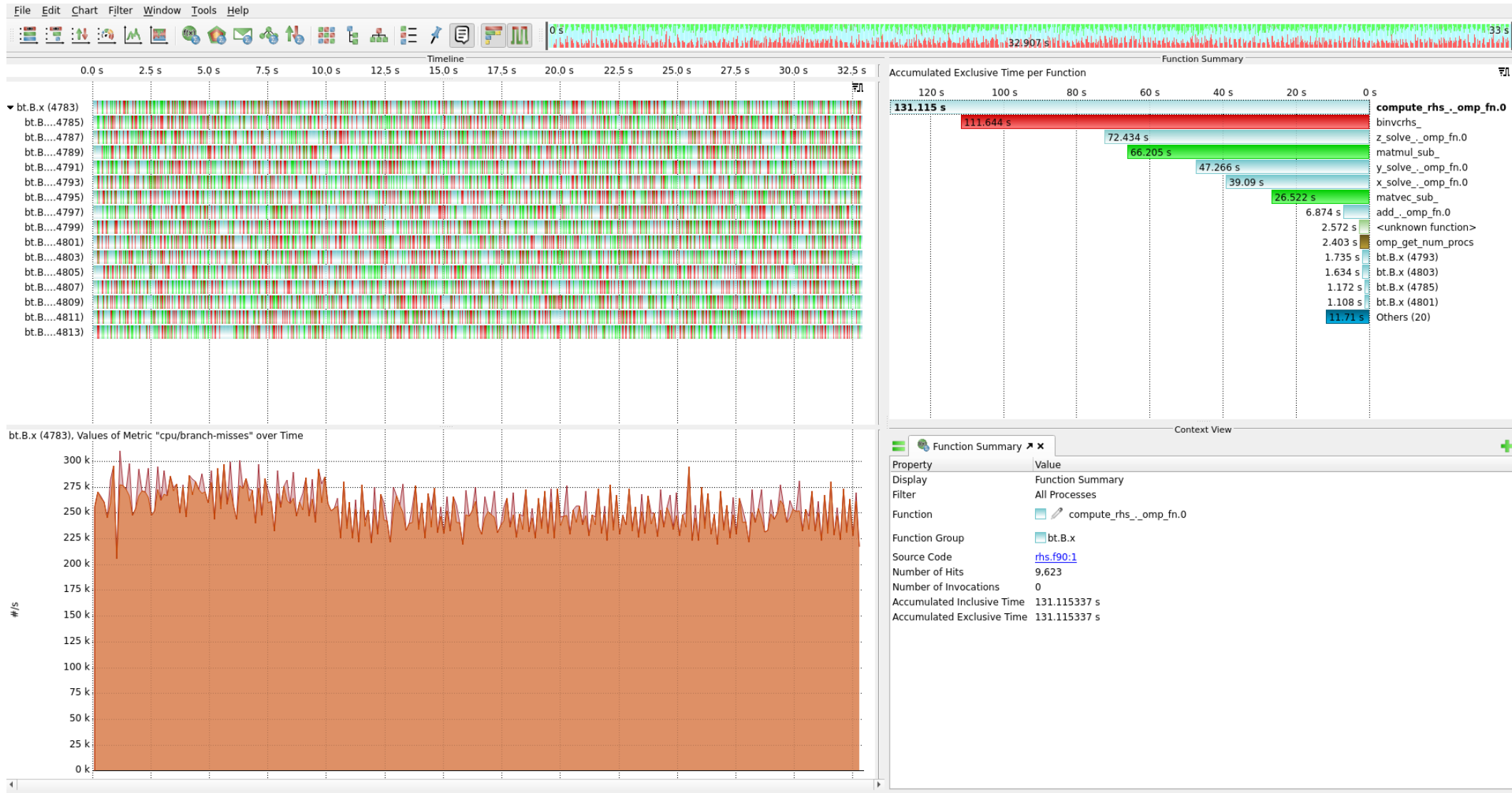
- Records I/O per-block-device on the bio_queue layer

```
# lo2s -block-io ...
```

- Allows for interesting differential diagnosis of I/O problems

Example: NPB BT-OMP Class C

```
# lo2s -E cache-misses -- ./BT.C.x
```



Example: NPB BT-OMP Class C

```
# lo2s -A -userspace-metric-event -- ./BT.C.x
```

The screenshot displays a performance analysis tool interface. The top section shows a timeline from 0.0 s to 30.0 s with 16 horizontal tracks for different CPU samples (cpu 0 to cpu 15). The tracks show various colored bars representing different processes or functions. The bottom section is a graph titled "samples for cpu 1. Values of Metric 'power/energy-pkg' over Time". The y-axis is labeled "joules/s" and ranges from 0 to 20. The x-axis is time from 0 to 30 seconds. The graph shows a high power consumption (around 18-20 joules/s) for the first 5 seconds, which then drops and fluctuates between 12 and 16 joules/s for the remainder of the run.

Accumulated Exclusive Time per Function

Time (s)	Function
46.198 s	binvrhs_
32.786 s	compute_rhs_omp_fn.0
28.059 s	matmul_sub
25.322 s	z_solve_omp_fn.0
22.063 s	bt.B.x (6076)
21.301 s	y_solve_omp_fn.0
20.879 s	bt.B.x (6083)
20.326 s	bt.B.x (6085)
19.449 s	x_solve_omp_fn.0
19.36 s	bt.B.x (6075)
19.357 s	bt.B.x (6082)
18.927 s	bt.B.x (6078)
18.729 s	bt.B.x (6077)
18.672 s	bt.B.x (6081)
18.576 s	bt.B.x (6090)
10.540 s	bt.B.x (6070)

Trace Info

Property	Value
File	/home/cvoneim/lo2s/build/lo2s_trace_2024-03-01T09-57-45/traces.otf2
Creator	lo2s - v1.7.0-49-gccbae16 (Mar 1 2024)
Version	3.0.3
Timer Resolution	1 ns

Context Counts

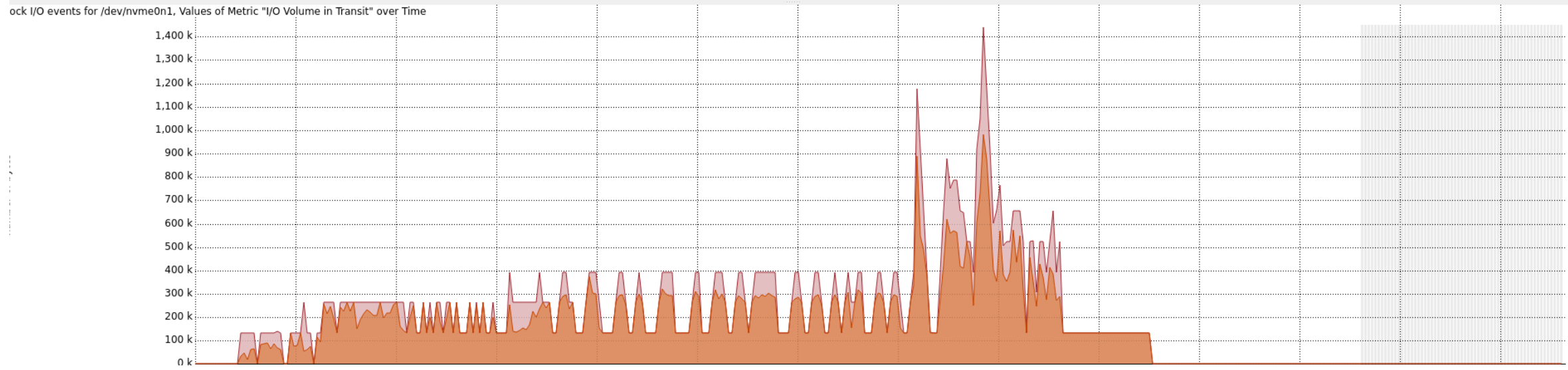
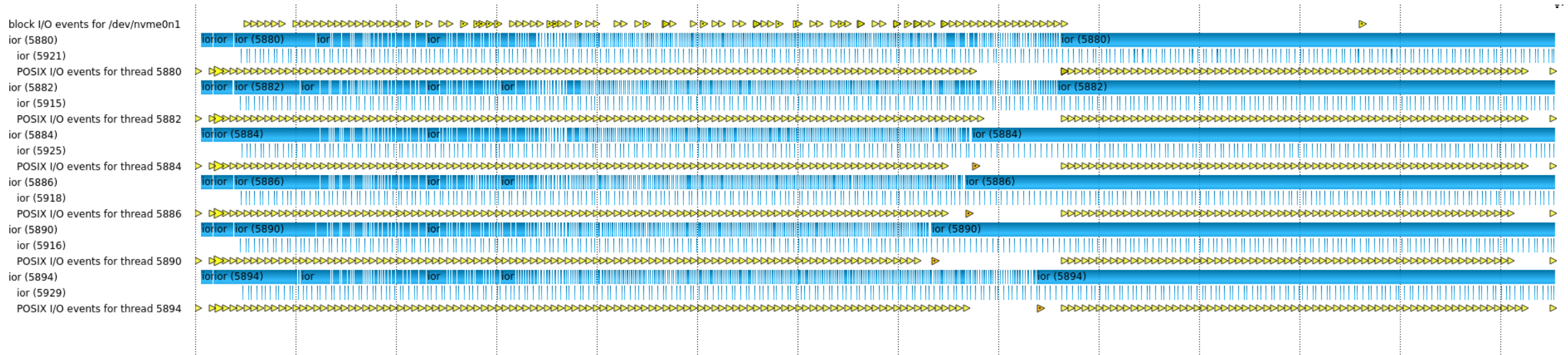
Category	Count
Processes	16
Process Groups	43
Process Group Paradigms	2
Source Code Files	23
Source Code Locations	180
Functions	1367
Function Groups	420
Communicators	2
Message Tags	0
Files	0

Function Legend

- (sd-pam)
- (vdso)
- <idle>
- <unknown binary>
- abrt-applet
- abrt-dbus
- abrt-dump-journ
- abrt-d
- accounts-daemon
- agent
- alsactl
- Application

```
# lo2s -A --block-o
```

Example: Block I/O



Summary

- Lightweight node-level monitoring for Linux
- Versatile through leveraging perf
- Visualization with Vampir
- Deep insight into node via perf metrics, tracepoints, and I/O recording
- Available at <https://github.com/tud-zih-energy/1o2s>