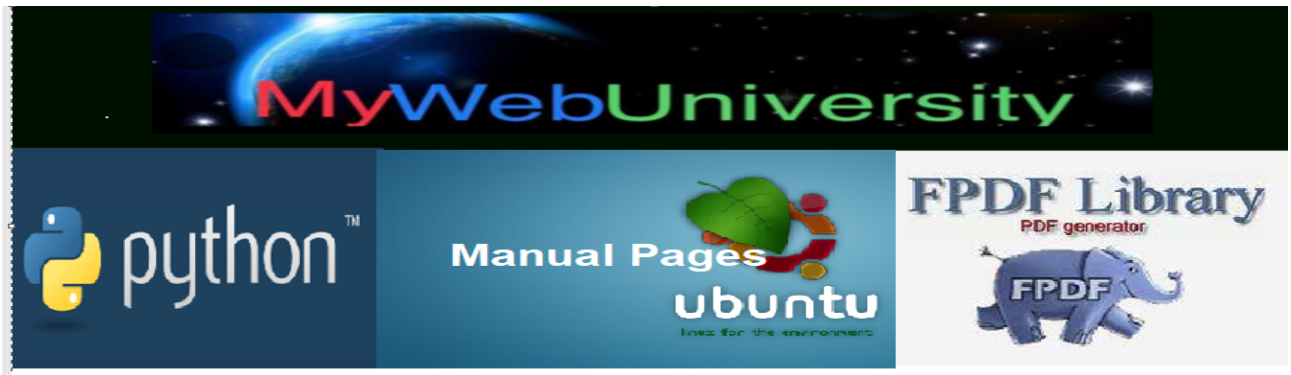

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'perf-amd-ibs.1'

$ man perf-amd-ibs.1

PERF-AMD-IBS(1) perf Manual PERF-AMD-IBS(1)

NAME

perf-amd-ibs - Support for AMD Instruction-Based Sampling (IBS) with perf tool

SYNOPSIS

perf record -e ibs_op//

perf record -e ibs_fetch//

DESCRIPTION

Instruction-Based Sampling (IBS) provides precise Instruction Pointer (IP) profiling support on AMD platforms. IBS has two independent components: IBS Op and IBS Fetch. IBS Op sampling provides information about instruction execution (micro-op execution to be precise) with details like d-cache hit/miss, d-TLB hit/miss, cache miss latency, load/store data source, branch behavior etc. IBS Fetch sampling provides information about instruction fetch with details like i-cache hit/miss, i-TLB hit/miss, fetch latency etc. IBS is per-smt-thread i.e. each SMT hardware thread contains standalone IBS units.

Both, IBS Op and IBS Fetch, are exposed as PMUs by Linux and can be exploited using the Linux perf utility. The following files will be created at boot time if IBS is supported by the hardware and kernel.

/sys/bus/event_source/devices/ibs_op/

/sys/bus/event_source/devices/ibs_fetch/

IBS Op PMU supports two events: cycles and micro ops. IBS Fetch PMU supports one event: fetch ops.

IBS PMUs do not have user/kernel filtering capability and thus it requires CAP_SYS_ADMIN or CAP_PERFMON privilege.

IBS VS. REGULAR CORE PMU

IBS gives samples with precise IP, i.e. the IP recorded with IBS sample has no skid. Whereas the IP recorded by regular core PMU will have some skid (sample was generated at IP X but perf would record it at IP X+n). Hence, regular core PMU might not help for profiling with instruction level precision. Further, IBS provides additional information about the sample in question. On the other hand, regular core PMU has it?s own advantages like plethora of events, counting mode (less interference), up to 6 parallel counters, event grouping support, filtering capabilities etc.

Three regular core PMU events are internally forwarded to IBS Op PMU when precise_ip attribute is set:

-e cpu-cycles:p becomes -e ibs_op//

-e r076:p becomes -e ibs_op//

-e r0C1:p becomes -e ibs_op/cnt_ctl=1/

EXAMPLES

IBS Op PMU

System-wide profile, cycles event, sampling period: 100000

```
# perf record -e ibs_op// -c 100000 -a
```

Per-cpu profile (cpu10), cycles event, sampling period: 100000

```
# perf record -e ibs_op// -c 100000 -C 10
```

Per-cpu profile (cpu10), cycles event, sampling freq: 1000

```
# perf record -e ibs_op// -F 1000 -C 10
```

System-wide profile, uOps event, sampling period: 100000

```
# perf record -e ibs_op/cnt_ctl=1/ -c 100000 -a
```

Same command, but also capture IBS register raw dump along with perf sample:

```
# perf record -e ibs_op/cnt_ctl=1/ -c 100000 -a --raw-samples
```

System-wide profile, uOps event, sampling period: 100000, L3MissOnly (Zen4 onward)

```
# perf record -e ibs_op/cnt_ctl=1,l3missonly=1/ -c 100000 -a
```

System-wide profile, cycles event, sampling period: 100000, LdLat filtering (Zen5 onward)

```
# perf record -e ibs_op/ldlat=128/ -c 100000 -a
```

Supported load latency threshold values are 128 to 2048 (both inclusive).

Latency value which is a multiple of 128 incurs a little less profiling overhead compared to other values.

Per process(upstream v6.2 onward), uOps event, sampling period: 100000

```
    # perf record -e ibs_op/cnt_ctl=1/ -c 100000 -p 1234
```

Per process(upstream v6.2 onward), uOps event, sampling period: 100000

```
    # perf record -e ibs_op/cnt_ctl=1/ -c 100000 -- ls
```

To analyse recorded profile in aggregate mode

```
    # perf report
    /* Select a line and press 'a' to drill down at instruction level. */
```

To go over each sample

```
    # perf script
```

Raw dump of IBS registers when profiled with --raw-samples

```
    # perf report -D
    /* Look for PERF_RECORD_SAMPLE */
    Example register raw dump:
    ibs_op_ctl:     000002c30006186a MaxCnt    100000 L3MissOnly 0 En 1
          Val 1 CntCtl 0=cycles CurCnt       707
    IbsOpRip:       ffffffff8204aea7
    ibs_op_data:    0000010002550001 CompToRetCtr     1 TagToRetCtr   597
          BrnRet 0  RipInvalid 0 BrnFuse 0 Microcode 1
    ibs_op_data2:   0000000000000013 RmtNode 1 DataSrc 3=DRAM
    ibs_op_data3:   0000000031960092 LdOp 0 StOp 1 DcL1TlbMiss 0
          DcL2TlbMiss 0 DcL1TlbHit2M 1 DcL1TlbHit1G 0 DcL2TlbHit2M 0
          DcMiss 1 DcMisAcc 0 DcWcMemAcc 0 DcUcMemAcc 0 DcLockedOp 0
          DcMissNoMabAlloc 0 DcLinAddrValid 1 DcPhyAddrValid 1
          DcL2TlbHit1G 0 L2Miss 1 SwPf 0 OpMemWidth 32 bytes
          OpDcMissOpenMemReqs 12 DcMissLat     0 TlbRefillLat     0
    IbsDCLinAd:     ff110008a5398920
    IbsDCPhysAd:    00000008a5398920
```

IBS applied in a real world usecase

```
    ~90% regression was observed in tbench with specific scheduler hint
    which was counter intuitive. IBS profile of good and bad run captured
    using perf helped in identifying exact cause of the problem:
    https://lore.kernel.org/r/20220921063638.2489-1-kprateek.nayak@amd.com
```

IBS Fetch PMU

Similar commands can be used with Fetch PMU as well.

System-wide profile, fetch ops event, sampling period: 100000

```
# perf record -e ibs_fetch// -c 100000 -a
```

System-wide profile, fetch ops event, sampling period: 100000, Random enable

```
# perf record -e ibs_fetch/rand_en=1/ -c 100000 -a
```

Random enable adds small degree of variability to sample period. This helps in cases like long running loops where PMU is tagging the same instruction over and over because of fixed sample period.

etc.

PERF MEM AND PERF C2C

perf mem is a memory access profiler tool and perf c2c is a shared data cacheline analyser tool. Both of them internally uses IBS Op PMU on AMD. Below is a simple example of the perf mem tool.

```
# perf mem record -c 100000 -- make
# perf mem report
```

A normal perf mem report output will provide detailed memory access profile. New output fields will show related access info together. For example:

```
# perf mem report -F overhead,cache,snoop,comm
...
# Samples: 92K of event 'ibs_op//'
# Total weight : 531104
#
#          ---------- Cache -----------  --- Snoop ----
# Overhead       L1     L2 L1-buf  Other     HitM  Other  Command
# ........  .............................  ..............  ..........
#
    76.07%     5.8%  35.7%   0.0%  34.6%    23.3%  52.8%  cc1
     5.79%     0.2%   0.0%   0.0%   5.6%     0.1%   5.7%  make
     5.78%     0.1%   4.4%   0.0%   1.2%     0.5%   5.3%  gcc
     5.33%     0.3%   3.9%   0.0%   1.1%     0.2%   5.2%  as
     5.00%     0.1%   3.8%   0.0%   1.0%     0.3%   4.7%  sh
     1.56%     0.1%   0.1%   0.0%   1.4%     0.6%   0.9%  ld
     0.28%     0.1%   0.0%   0.0%   0.2%     0.1%   0.2%  pkg-config
     0.09%     0.0%   0.0%   0.0%   0.1%     0.0%   0.1%  git
```

```
    0.03%     0.0%   0.0%   0.0%   0.0%     0.0%   0.0%  rm
    ...
```

Also, it can be aggregated based on various memory access info using the sort keys. For example:

```
    # perf mem report -s mem,snoop
    ...
    # Samples: 92K of event 'ibs_op//'
    # Total weight : 531104
    # Sort order   : mem,snoop
    #
    # Overhead       Samples  Memory access                        Snoop
    # ........  ............  ....................................  ............
    #
       47.99%          1509  L2 hit                               N/A
       25.08%           338  core, same node Any cache hit        HitM
       10.24%         54374  N/A                                  N/A
        6.77%         35938  L1 hit                               N/A
        6.39%           101  core, same node Any cache hit        N/A
        3.50%            69  RAM hit                              N/A
        0.03%           158  LFB/MAB hit                          N/A
        0.00%             2  Uncached hit                         N/A
```

Please refer to their man page for more detail.

SEE ALSO

perf-record(1), perf-script(1), perf-report(1), perf-mem(1), perf-c2c(1)