IMC (In-Memory Collection Counters)

Anju T Sudhakar, 10 May 2019

Contents

Basic overview

IMC (In-Memory collection counters) is a hardware monitoring facility that collects large numbers of hardware performance events at Nest level (these are on-chip but off-core), Core level and Thread level.

The Nest PMU counters are handled by a Nest IMC microcode which runs in the OCC (On-Chip Controller) complex. The microcode collects the counter data and moves the nest IMC counter data to memory.

The Core and Thread IMC PMU counters are handled in the core. Core level PMU counters give us the IMC counters' data per core and thread level PMU counters give us the IMC counters' data per CPU thread.

OPAL obtains the IMC PMU and supported events information from the IMC Catalog and passes on to the kernel via the device tree. The event's information contains:

- Event name
- Event Offset
- Event description

and possibly also:

- Event scale
- Event unit

Some PMUs may have a common scale and unit values for all their supported events. For those cases, the scale and unit properties for those events must be inherited from the PMU.

The event offset in the memory is where the counter data gets accumulated.

IMC catalog is available at:
https://github.com/open-power/ima-catalog

The kernel discovers the IMC counters information in the device tree at the *imc-counters* device node which has a compatible field *ibm,opal-in-memory-counters*. From the device tree, the kernel parses the PMUs and their event's information and register the PMU and its attributes in the kernel.

IMC example usage

```
# perf list
[...]
nest_mcs01/PM_MCS01_64B_RD_DISP_PORT01/            [Kernel PMU event]
nest_mcs01/PM_MCS01_64B_RD_DISP_PORT23/            [Kernel PMU event]
[...]
core_imc/CPM_0THRD_NON_IDLE_PCYC/                  [Kernel PMU event]
core_imc/CPM_1THRD_NON_IDLE_INST/                  [Kernel PMU event]
[...]
thread_imc/CPM_0THRD_NON_IDLE_PCYC/                [Kernel PMU event]
thread_imc/CPM_1THRD_NON_IDLE_INST/                [Kernel PMU event]
```

To see per chip data for nest_mcs0/PM_MCS_DOWN_128B_DATA_XFER_MC0/:

```
# ./perf stat -e "nest_mcs01/PM_MCS01_64B_WR_DISP_PORT01/" -a --per-socket
```

To see non-idle instructions for core 0:

```
# ./perf stat -e "core_imc/CPM_NON_IDLE_INST/" -C 0 -I 1000
```

To see non-idle instructions for a "make":

```
# ./perf stat -e "thread_imc/CPM_NON_IDLE_PCYC/" make
```

IMC Trace-mode

POWER9 supports two modes for IMC which are the Accumulation mode and Trace mode. In Accumulation mode, event counts are accumulated in system Memory. Hypervisor then reads the posted counts periodically or when requested. In IMC Trace mode, the 64 bit trace SCOM value is initialized with the event information. The CPMCxSEL and CPMC_LOAD in the trace SCOM, specifies the event to be monitored and the sampling duration. On each overflow in the CPMCxSEL, hardware snapshots the program counter along with event counts and writes into memory pointed by LDBAR.

LDBAR is a 64 bit special purpose per thread register, it has bits to indicate whether hardware is configured for accumulation or trace mode.

LDBAR Register Layout

0	Enable/Disable
1	0: Accumulation Mode
	1: Trace Mode
2:3	Reserved
4-6	PB scope
7	Reserved
8:50	Counter Address
51:63	Reserved

TRACE_IMC_SCOM bit representation

0:1	SAMPSEL
2:33	CPMC_LOAD
34:40	CPMC1SEL
41:47	CPMC2SEL
48:50	BUFFERSIZE
51:63	RESERVED

CPMC_LOAD contains the sampling duration. SAMPSEL and CPMCxSEL determines the event to count. BUFFERSIZE indicates the memory range. On each overflow, hardware snapshots the program counter along with event counts and updates the memory and reloads the CMPC_LOAD value for the next sampling duration. IMC hardware does not support exceptions, so it quietly wraps around if memory buffer reaches the end.

Currently the event monitored for trace-mode is fixed as cycle.

Trace IMC example usage

```
# perf list
[....]
trace_imc/trace_cycles/                            [Kernel PMU event]
```

To record an application/process with trace-imc event:

```
# perf record -e trace_imc/trace_cycles/ yes > /dev/null
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.012 MB perf.data (21 samples) ]
```

The *perf.data* generated, can be read using perf report.

Benefits of using IMC trace-mode

PMI (Performance Monitoring Interrupts) interrupt handling is avoided, since IMC trace mode snapshots the program counter and updates to the memory. And this also provide a way for the operating system to do instruction sampling in real time without PMI processing overhead.

Performance data using *perf top* with and without trace-imc event.

PMI interrupts count when *perf top* command is executed without trace-imc event.

```
# grep PMI /proc/interrupts
PMI:          0          0          0          0   Performance monitoring interrupts
# ./perf top
...
# grep PMI /proc/interrupts
PMI:      39735       8710      17338      17801   Performance monitoring interrupts
# ./perf top -e trace_imc/trace_cycles/
```

```
...
# grep PMI /proc/interrupts
PMI:      39735       8710      17338      17801   Performance monitoring interrupts
```

That is, the PMI interrupt counts do not increment when using the *trace_imc* event.