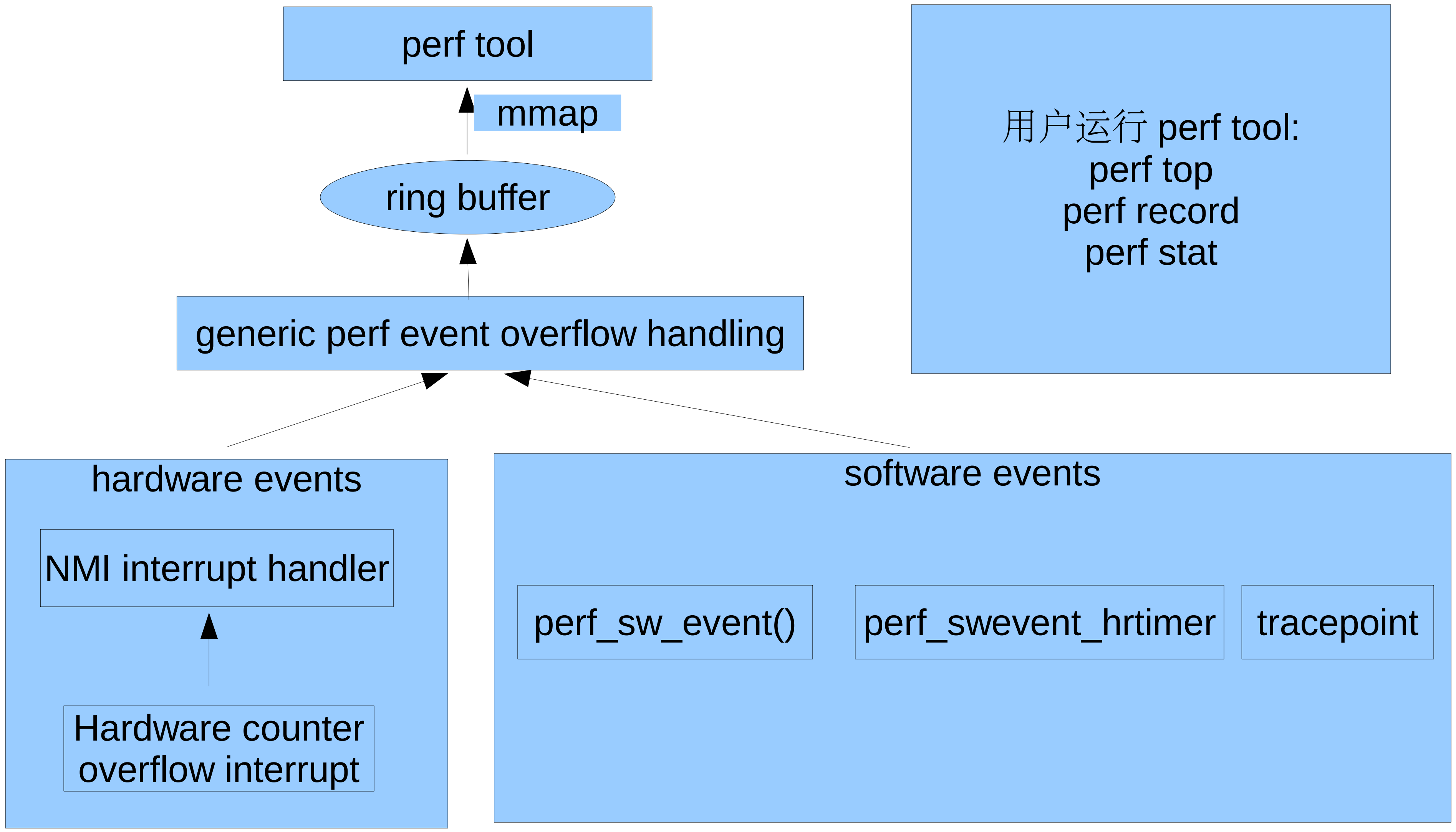
PerfEvent 与 Intel PMU 介绍

林铭 Intel 开源技术中心
ming.m.lin@intel.com

PerfEvent 简介

- 功能强大的性能分析工具
  - kernel code + userspace tool
  - 支持多种CPU: x86, PowerPC, ARM, Sparc, MIPS, Alpha, Blackfin, SH ...
- 范围
  - 整个系统（所有CPU ＋ 所有进程）
  - 某个或某几个CPU
  - 某个进程
- 对象
  - hardware event: cpu cycles, instructions, cache hit/miss ......
  - software event: page fault, context switch, cpu migrations ......
  - tracepoint
- 形式
  - 采样（实时观测，保存分析）
  - 统计
- 收集call graph
- Kernel/userspace分别收集

PerfEvent 架构



用户运行 perf tool:
perf top
perf record
perf stat

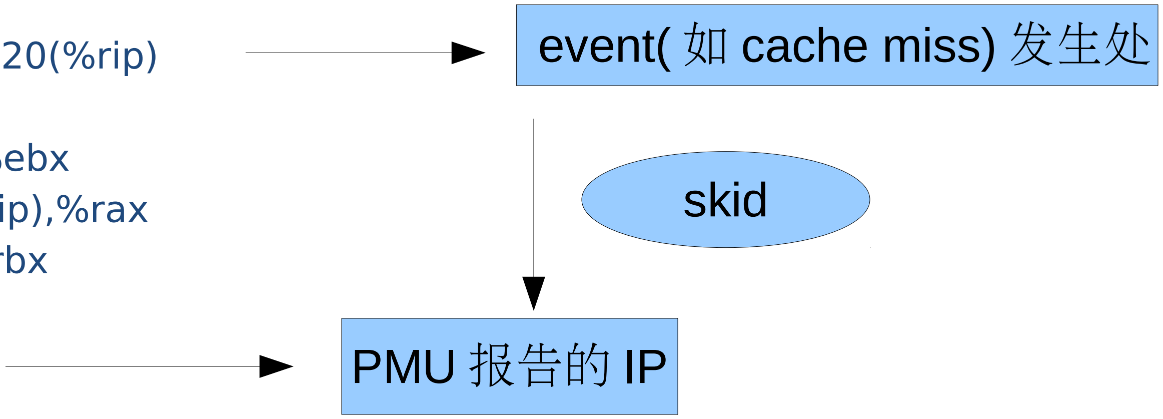
Intel PMU

- PMU: Performance Monitoring Unit
- performance counter 和 performance event
  - counter 记录 event 发生的次数
  - counter: Core2 2 个, Nehalem/Westmere/SandyBridge: 4 个
  - event: 架构化 7 个： cpu cycles, bus cycles, cpu instruction, LLC hit, LLC miss, branch instruction, branch misses， 非架构化上百个
  - counter overflow interrupt
- PerfEvent counter 调度

skid issue

Skid: PMU报告的IP不是event发生的准确IP

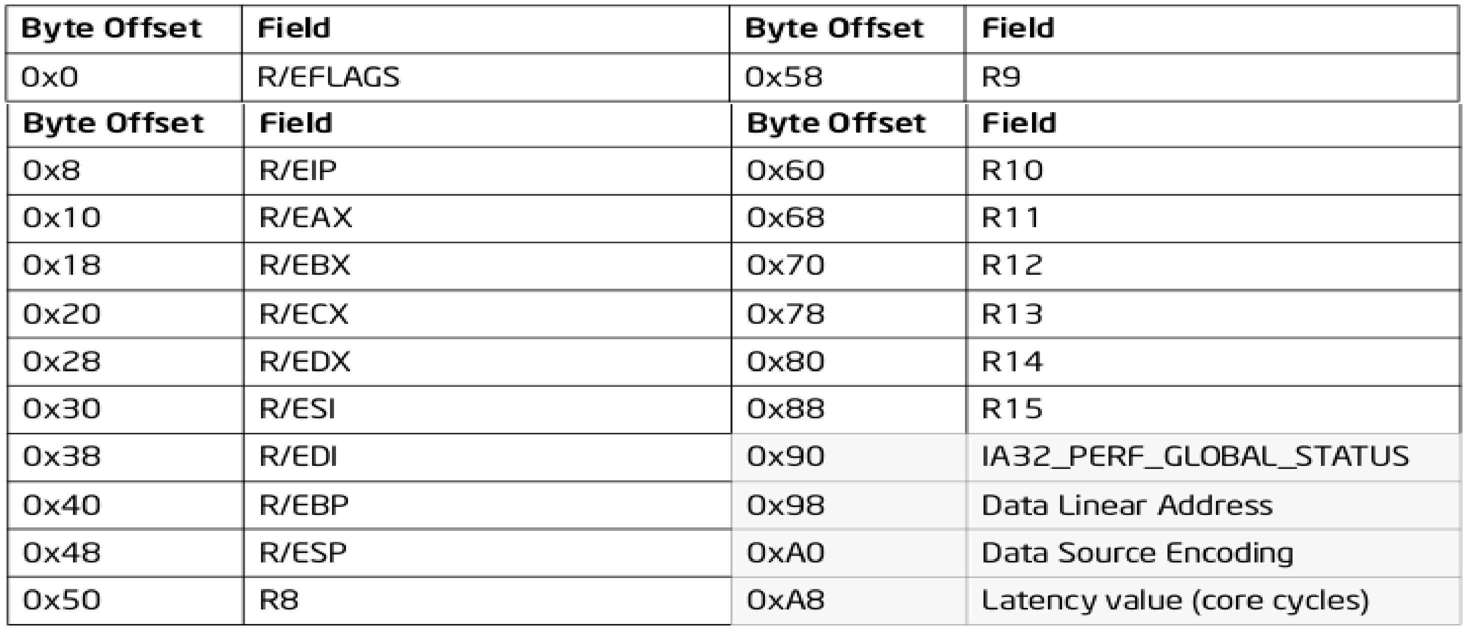
```
400400:   push   %rbp
400401:   mov    %rsp,%rbp
400404:   push   %rbx
400405:   sub    $0x8,%rsp
400409:   cmpb   $0x0,0x200420(%rip)
400410:   jne    40045d
400412:   mov    $0x600648,%ebx
400417:   mov    0x20041a(%rip),%rax
40041e:   sub    $0x600640,%rbx
400425:   sar    $0x3,%rbx
400429:   sub    $0x1,%rbx
40042d:   cmp    %rbx,%rax
400430:   jae    400456
400432:   nopw   0x0(%rax,%rax,1)
```



PEBS(Precise Event Based Sampling)

- PEBS record format

Table 30-12. PEBS Record Format for Intel Core i7 Processor Family



Byte Offset	Field	Byte Offset	Field
0x0	R/EFLAGS	0x58	R9
Byte Offset	**Field**	**Byte Offset**	**Field**
0x8	R/EIP	0x60	R10
0x10	R/EAX	0x68	R11
0x18	R/EBX	0x70	R12
0x20	R/ECX	0x78	R13
0x28	R/EDX	0x80	R14
0x30	R/ESI	0x88	R15
0x38	R/EDI	0x90	IA32_PERF_GLOBAL_STATUS
0x40	R/EBP	0x98	Data Linear Address
0x48	R/ESP	0xA0	Data Source Encoding
0x50	R8	0xA8	Latency value (core cycles)

- PEBS events
  - cycles, instructions, branch instructions, branch misses
- PEBS 用法示例： perf top -e cycles:p

PEBS IP+1 issue

```
400400:   push   %rbp
400401:   mov    %rsp,%rbp
400404:   push   %rbx
400405:   sub    $0x8,%rsp
400409:   cmpb   $0x0,0x200420(%rip)
400410:   jne    40045d
400412:   mov    $0x600648,%ebx
400417:   mov    0x20041a(%rip),%rax
40041e:   sub    $0x600640,%rbx
400425:   sar    $0x3,%rbx
400429:   sub    $0x1,%rbx
40042d:   cmp    %rbx,%rax
400430:   jae    400456
400432:   nopw   0x0(%rax,%rax,1)
```

event 发生处

PEBS 报告的 IP

LBR(Last Branch Record)

- LBR MSR

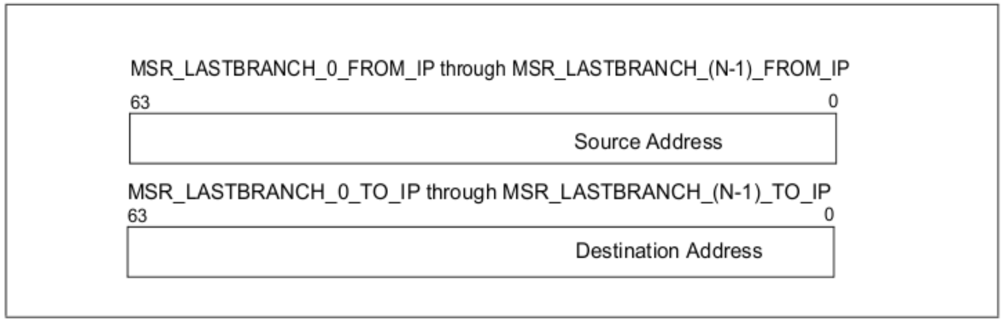

MSR_LASTBRANCH_0_FROM_IP through MSR_LASTBRANCH_(N-1)_FROM_IP
63
0
Source Address
MSR_LASTBRANCH_0_TO_IP through MSR_LASTBRANCH_(N-1)_TO_IP
63
0
Destination Address

Figure 16-4. 64-bit Address Layout of LBR MSR

```
4004ec:     movl   $0x0,-0x4(%rbp)
4004f3:     cmpl   $0x0,-0x4(%rbp)
4004f7:     je     400505
4004f9:     mov    $0x40060c,%edi
4004fe:     callq  4003e0
400503:     jmp    40050f
400505:     mov    $0x40060c,%edi
40050a:     callq  4003e0
40050f:     mov    $0x0,%eax
```

Source Address: 4004f7
Destination Address: 400505

LBR: PEBS IP+1 issue fixup

- 第一种情况： event 发生在 branch instruction 处

```
4004f7:     je     400505                    → event 发生处
4004f9:     mov    $0x40060c,%edi
4004fe:     callq  4003e0
400503:     jmp    40050f
400505:     mov    $0x40060c,%edi            → PEBS 记录的 IP
40050a:     callq  4003e0
```

Source Address: 4004f7
Destination Address: 400505

将 PEBS IP 纠正为 Source Address
用法示例： perf top -e cycles:pp

LBR: PEBS IP+1 issue fixup cont.

- 第二种情况： event 不是发生在 branch instruction 处

```
400503:    jmp    40050f
400505:    mov    $0x40062c,%edi
40050a:    callq  4003e0
40050f:    addl   $0x1,-0x4(%rbp)
400513:    movl   $0x64,-0x4(%rbp)
40051a:    mov    -0x4(%rbp),%ecx        ──► event 发生处
40051d:    mov    $0x66666667,%edx       ──► PEBS 记录的 IP
```

Source Address: 400503
Destination Address: 40050f

将 PEBS IP 纠正为 40051a

Kernel instruction decoder:

```
40050f:   83 45 fc 01 ( 长度 4)
400513:   c7 45 fc 64 00 00 00 ( 长度 8)
40051a:   8b 4d fc  ( 长度 3)
40051d:   ba 67 66 66 66 ( 长度 5)
```

LBR: sampling taken branches

```
void f2(void) {}
void f3(void) {}
void f1(unsigned long n)
{
   if (n & 1UL)
      f2();
   else
      f3();
}
int main(void)
{
    unsigned long i;
    for (i=0; i < N; i++)
       f1(i);
    return 0;
}
```

```
$ perf record -b any_call -e cycles:u branchy
$ perf report
# Events: 19K cycles
#
# Overhead  Source Symbol   Target Symbol
# ..............  .........................  ........................
#
    52.50%     [.] main              [.] f1
    23.99%     [.] f1                [.] f3
    23.48%     [.] f1                [.] f2
```

Patches by Stephane Eranian @google
[PATCH 00/12] perf_events: add support for sampling taken branches
http://marc.info/?l=linux-kernel&m=131805702307992&w=2

Load latency & Precise store

- Load latency
  - Load data linear address
  - Latency value
  - Data source
- Precise store
  - Store data linear address
  - Store status
- Patches

  [PATCH 0/4] perf: memory load/store events generalization

  http://marc.info/?l=linux-kernel&m=130976621805927&w=2

perf mem 例子 1

```
$ perf mem

 usage: perf mem [<options>] {record <command> |report}

    -t, --type <type>     memory operations(load/store)
    -L, --latency <n>     latency to sample(only for load op)

$ perf mem -t load record make -j8

<building kernel ..., monitoring memory load opeartion>

$ perf mem -t load report

Memory load operation statistics
================================
                      L1-local: total latency=   28027, count=    3355(avg=8)
                      L2-snoop: total latency=    1430, count=      29(avg=49)
                      L2-local: total latency=     124, count=       8(avg=15)
             L3-snoop, found M: total latency=     452, count=       4(avg=113)
          L3-snoop, found no M: total latency=       0, count=       0(avg=0)
L3-snoop, no coherency actions: total latency=     875, count=      18(avg=48)
        L3-miss, snoop, shared: total latency=       0, count=       0(avg=0)
     L3-miss, local, exclusive: total latency=       0, count=       0(avg=0)
        L3-miss, local, shared: total latency=       0, count=       0(avg=0)
    L3-miss, remote, exclusive: total latency=       0, count=       0(avg=0)
       L3-miss, remote, shared: total latency=       0, count=       0(avg=0)
                    Unknown L3: total latency=       0, count=       0(avg=0)
                            IO: total latency=       0, count=       0(avg=0)
                      Uncached: total latency=     464, count=      30(avg=15)

$ perf mem -t store record make -j8

<building kernel ..., monitoring memory store opeartion>

$ perf mem -t store report

Memory store operation statistics
=================================
                data-cache hit:     8138
               data-cache miss:        0
                      STLB hit:     8138
                     STLB miss:        0
                Locked access:        0
               Unlocked access:     8138
```

perf mem 例子 2

Data structure 分析

```
       struct bar {
 0.00 :         int poekoe[5];
60.00 :         int fubar;
        };

        struct foo {
 0.00 :         long poekoe[3];
10.00 :         struct bar *bar;
        };

        struct tmp {
 0.00 :         long poekoe[4];
30.00 :         int blah;
        };
```

将地址与 structure field 关联

Questions?

Legal Information