perf工具及PAPI的使用

邹永浩

2019211168

`perf`分析热点

因为我是自己编译的内核，所以需要进入内核目录下`tools/perf`进行编译安装，直接`make`再`make install`就可以了。

首先对程序运行概况进行分析，使用`perf stat`查看：

```
bugscan@bugscan-laptop:~/Desktop$ perf stat ./a.out
0 25 20 15   0.24    7.03 28946421

 Performance counter stats for './a.out':

          7,270.75 msec task-clock:u              #    1.000 CPUs utilized
                 0      context-switches:u        #    0.000 K/sec
                 0      cpu-migrations:u          #    0.000 K/sec
             4,913      page-faults:u             #  675.791 M/sec
    21,031,111,961      cycles:u                  # 2892862.718 GHz
    34,642,519,923      instructions:u            #    1.65  insn per cycle
     1,567,000,841      branches:u                # 215543444.429 M/sec
        11,937,794      branch-misses:u           #    0.76% of all branches

       7.271439869 seconds time elapsed

       7.259089000 seconds user
       0.011998000 seconds sys
```

可以看到程序是纯计算类的，因为`task-clock`为1.

进一步分析程序热点，使用`perf record`记录，再使用`perf report`查看结果：

```
Samples: 36K of event 'cycles:uppp', Event count (approx.): 26444281452
Overhead  Command  Shared Object     Symbol
  95.70%  a.out    a.out             [.] do_main_sieve
   1.68%  a.out    a.out             [.] count_zero_bits
   1.39%  a.out    a.out             [.] init_main_sieve
   1.19%  a.out    a.out             [.] update_small_sieve
   0.03%  a.out    [unknown]         [k] 0xffffffff99600a27
   0.00%  a.out    libc-2.27.so      [.] _IO_file_xsputn@@GLIBC_2.2.5
   0.00%  a.out    ld-2.27.so        [.] strcmp
   0.00%  a.out    ld-2.27.so        [.] _start
```

可以看到，程序热点函数为`do_main_sieve`，如果需要进行优化，应重点关注该函数。

`PAPI`采集指标

安装`PAPI`比较简单，只要把源码下载编译安装即可，编译完成后，我们可以通过`utils/papi_avail`查看支持的事件，如下图所示：

```
================================================================================
  PAPI Preset Events
================================================================================
    Name        Code    Avail Deriv Description (Note)
PAPI_L1_DCM  0x80000000  Yes   No   Level 1 data cache misses
PAPI_L1_ICM  0x80000001  Yes   No   Level 1 instruction cache misses
PAPI_L2_DCM  0x80000002  Yes   Yes  Level 2 data cache misses
PAPI_L2_ICM  0x80000003  Yes   No   Level 2 instruction cache misses
PAPI_L3_DCM  0x80000004  No    No   Level 3 data cache misses
PAPI_L3_ICM  0x80000005  No    No   Level 3 instruction cache misses
PAPI_L1_TCM  0x80000006  Yes   Yes  Level 1 cache misses
PAPI_L2_TCM  0x80000007  Yes   No   Level 2 cache misses
PAPI_L3_TCM  0x80000008  Yes   No   Level 3 cache misses
```

可以看到`PAPI`支持查看各类缓存的数据或者指令miss，在我的机器上, 有很多指标无法统计, 其中影响实验的是`PAPI_L1_TCA` 以及 `PAPI_TLB_TL` , 因此我在测试`L1 cache`时仅统计了miss的次数, 对于`TLB`的 miss rate 则未统计.

修改被测程序，添加如下代码：

```
// ...
#include <papi.h>
// ...
int main(int argc, char **argv)
{
  /* Initialize the PAPI library */
  if (PAPI_library_init(PAPI_VER_CURRENT) != PAPI_VER_CURRENT)
    exit(1);
  /* Create an EventSet */
  int EventSet = PAPI_NULL;
  int retval = PAPI_create_eventset(&EventSet);
  assert(retval == PAPI_OK);

  /* Total cycles */
  retval = PAPI_add_event(EventSet, PAPI_TOT_CYC);
  assert(retval == PAPI_OK);
  /* Instructions completed */
  retval = PAPI_add_event(EventSet, PAPI_TOT_INS);
  assert(retval == PAPI_OK);

  /* L1 cache miss */
  retval = PAPI_add_event(EventSet, PAPI_L1_TCM);
  assert(retval == PAPI_OK);
  /* L1 cache access */
  // retval = PAPI_add_event(EventSet, PAPI_L1_TCA);
  // assert(retval == PAPI_OK);
  /* L2 cache miss */
  retval = PAPI_add_event(EventSet, PAPI_L2_TCM);
  assert(retval == PAPI_OK);
  /* L2 cache access */
  retval = PAPI_add_event(EventSet, PAPI_L2_TCA);
  assert(retval == PAPI_OK);
  /* L3 cache miss */
  retval = PAPI_add_event(EventSet, PAPI_L3_TCM);
  assert(retval == PAPI_OK);
  /* L3 cache access */
  retval = PAPI_add_event(EventSet, PAPI_L3_TCA);
  assert(retval == PAPI_OK);

  /* TLB miss */
  // retval = PAPI_add_event(EventSet, PAPI_TLB_TL);
  // assert(retval == PAPI_OK);

  /* branch mispredict */
  retval = PAPI_add_event(EventSet, PAPI_BR_MSP);
  assert(retval == PAPI_OK);
  /* branch */
  retval = PAPI_add_event(EventSet, PAPI_BR_CN);
  assert(retval == PAPI_OK);

  /* Start counting events */
```

```
  if (PAPI_start(EventSet) != PAPI_OK)
    retval = PAPI_start(EventSet);
  assert(retval == PAPI_OK);

  long long values1[9];
  long long values2[9];
  PAPI_read(EventSet, values1);
  assert(retval == PAPI_OK);

  // logic ignored

  /* Stop counting events */
  retval = PAPI_stop(EventSet, values2);
  assert(retval == PAPI_OK);

  int index = 0;
  printf("CPI: %f\n", (values2[index] - values1[index]) / 1.0 / (values2[index +
1] - values1[index + 1]));
  index += 2;
  printf("L1 cache miss: %lld\n", (values2[index] - values1[index]));
  index++;
  printf("L2 cache miss rate: %f\n", (values2[index] - values1[index]) / 1.0 /
(values2[index + 1] - values1[index + 1]));
  index += 2;
  printf("L3 cache miss rate: %f\n", (values2[index] - values1[index]) / 1.0 /
(values2[index + 1] - values1[index + 1]));

  // printf("TLB cache miss: %lld\n", (values2[8] - values1[8]));

  index += 2;
  printf("Branch miss prediction rate: %f\n", (values2[index] - values1[index])
/ 1.0 / (values2[index + 1] - values1[index + 1]));

  /* Clean up EventSet */
  if (PAPI_cleanup_eventset(EventSet) != PAPI_OK)
    exit(-1);

  /* Destroy the EventSet */
  if (PAPI_destroy_eventset(&EventSet) != PAPI_OK)
    exit(-1);

  /* Shutdown PAPI */
  PAPI_shutdown();

  return 0;
}
```

编译命令如下:

```
gcc prime_sieve-64.c /usr/local/lib/libpapi.a
```

运行结果为:

```
zyh@pc:~/Desktop/MeasuringComputerPerformance/hw11$ ./a.out
0 25 20 15   0.19    4.98 28946421
CPI: 0.565563
L1 cache miss: 614125883
L2 cache miss rate: 1.086262
L3 cache miss rate: 0.091261
Branch miss prediction rate: 0.007873
```

参考文献

https://www.ibm.com/developerworks/cn/linux/l-cn-perf1/index.html

https://www.cnblogs.com/dmyu/p/4648413.html