基于采样的方法实现一个函数热点分析工具

邹永浩

2019211168

使用 `libunwind` 和 `ptrace` 编写程序

主要使用了`libunwind`和`ptrace`的相关函数编写程序，这样可以避免对源程序进行修改，而且也不用链接源程序时多增加链接库，是一种对用户透明的方案。

使用的关键函数为`unw_get_proc_name`和`unw_step`.

简略起见，并没有实现函数栈的维护，只是粗略统计了函数调用的统计。
主要代码如下(排序等其他函数省略):

```
int main(int argc, char **argv)
{
    // ...
    while (1)
    {
        ret = ptrace(PTRACE_ATTACH, pid);
        as = unw_create_addr_space(&_UPT_accessors, 0);
        ui = _UPT_create(pid);
        unw_init_remote(&c, as, ui);
        do
        {
            // 获取当前函数名
            unw_get_proc_name(&c, buf, sizeof(buf), NULL);
            int func_exist = 0;
            for (int i = 0; i < total_count; i++)
            {
                if (strcmp(functions[i].function_name, buf) == 0)
                {
                    func_exist = 1;
                    functions[i].function_count++;
                    break;
                }
            }
            if (!func_exist)
            {
                char *name = malloc(sizeof(buf));
                strcpy(name, buf);
                functions[total_count].function_name = name;
                total_count++;
            }
        } while ((ret = unw_step(&c)) > 0); // 继续获取整个函数栈
        total_profile_count++;
        _UPT_destroy(ui);
        ptrace(PTRACE_DETACH, pid, 0, 0);

        sleep(0.5);
```

```
        // ...
    }
    print_result();
    return 0;
}
```

使用graph500测试

测试脚本如下：

```
killall -9 graph500_reference_bfs
/home/zyh/graph500-newreference/src/graph500_reference_bfs 16 16 &
TEST_PID=$!
./profile $TEST_PID
```

以下是使用500ms间隔的结果：

```
                name  times   percentage
                      161185  0.59
   __libc_start_main    110218  0.41
               main    110217  0.41
              _start    110217  0.41
     validate_result    83512   0.31
            aml_send    80695   0.30
    edgepreddisthndl    42756   0.16
  makedepthmapforbfs    31882   0.12
        frompredhndl    23912   0.09
             run_bfs    14983   0.06
get_edge_count_for_teps 5833    0.02
generate_kronecker_range        4552    0.02
           visithndl    2374    0.01
   mrg_get_uint_orig    1412    0.01
            mrg_skip    1291    0.00
make_graph_data_structure       1216    0.00
convert_graph_to_oned_csr       1213    0.00
mrg_apply_transition    1209    0.00
        fulledgehndl    656     0.00
           aml_my_pe    390     0.00
        halfedgehndl    234     0.00
          mpirinitf_    230     0.00
        _IO_vfprintf    184     0.00
        PMPI_Testany    100     0.00
         aml_barrier    98      0.00
       __fprintf_chk    87      0.00
              __poll    84      0.00
            aml_poll    82      0.00
             __write    74      0.00
      _IO_file_write    74      0.00
     _IO_file_xsputn    74      0.00
    MPI_Win_set_attr    37      0.00
      PMPI_Allreduce    31      0.00
          clean_pred    25      0.00
            aml_init    22      0.00
aml_register_handler    21      0.00
           PMPI_Init    17      0.00
    PMPI_Initialized    15      0.00
```

```
 __nss_passwd_lookup    14       0.00
       __libc_malloc    10       0.00
         MPIR_Dup_fn    7        0.00
             vprintf    7        0.00
       PMPI_Ibarrier    6        0.00
        PMPI_Barrier    4        0.00
MPI_Type_get_extent_x   3        0.00
           MPI_Start    3        0.00
           PMPI_Test    3        0.00
   MPI_Comm_set_name    2        0.00
    PMPI_Cart_coords    2        0.00
         swapcontext    2        0.00
              getenv    2        0.00
           MPI_Wtime    2        0.00
     __clock_gettime    2        0.00
MPII_Grequest_set_lang_f77      1       0.00
     PMPI_T_finalize    1        0.00
     _IO_str_seekoff    1        0.00
      MPI_Comm_split    1        0.00
pthread_setaffinity_np  1        0.00
    PMPI_Cart_create    1        0.00
             cuserid    1        0.00
_pthread_cleanup_push_defer     1       0.00
  MPL_wtime_todouble    1        0.00
      MPI_Ialltoallw    1        0.00
           MPL_wtime    1        0.00
            __munmap    1        0.00
               cfree    1        0.00
total count is 271380
```

以下是使用`1s`间隔的结果：

```
                name    times   percentage
                main    6        1.00
   __libc_start_main    6        1.00
              _start    6        1.00
     validate_result    5        0.83
            aml_send    4        0.67
  makedepthmapforbfs    4        0.67
      MPL_wtime_init    1        0.17
           PMPI_Init    1        0.17
            aml_init    1        0.17
        frompredhndl    1        0.17
    edgepreddisthndl    1        0.17
total count is 6
```

可以看到间隔小的话会有很多详细结果，而且统计的函数不同。
在`500ms`时，除了`main`之外，可以看到占用时间最多的函数是`validate_result`，而`1s`测试结果则为`aml_send`和`run_bfs`，因此间隔大时可能会让人误判函数热点。
当然，间隔调小之后会使程序运行变慢，最好测试时权衡精度和效率。

使用`gprof`测试

在`graph500`编译时加入`-pg`选项

```
CFLAGS = -pg -Drestrict=__restrict__ -O3 -DGRAPH_GENERATOR_MPI -
DREUSE_CSR_FOR_VALIDATION -I../aml
```

这样运行后生成`gmon.out`，使用`gprof -b graph500_reference_bfs gmon.out`即可看到结果

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 27.45      1.91     1.91 136251050     0.00     0.00  edgepreddisthndl
 18.61      3.21     1.30 136247473     0.00     0.00  frompredhndl
 15.23      4.27     1.06                             run_bfs
 13.51      5.21     0.94 412941133     0.00     0.00  aml_send
  5.68      5.60     0.40                             validate_result
  5.10      5.96     0.36       65     5.46    28.56  makedepthmapforbfs
  3.02      6.17     0.21                             get_edge_count_for_teps
  2.73      6.36     0.19 136250270     0.00     0.00  visithndl
  2.01      6.50     0.14     1237     0.11     0.11  aml_poll
  1.44      6.60     0.10 270536453     0.00     0.00  aml_my_pe
  1.44      6.70     0.10                             generate_kronecker_range
  1.29      6.79     0.09  3072197     0.00     0.00  mrg_apply_transition
  0.86      6.85     0.06 16777239     0.00     0.00  mrg_get_uint_orig
  0.43      6.88     0.03  1048664     0.00     0.00  mrg_skip
  0.43      6.91     0.03                             aml_finalize
  0.36      6.93     0.03                             xMPI_Alloc_mem
  0.29      6.95     0.02                             frame_dummy
  0.14      6.96     0.01  2096170     0.00     0.00  fulledgehndl
  0.00      6.96     0.00  2096170     0.00     0.00  halfedgehndl
  0.00      6.96     0.00     1237     0.00     0.11  aml_barrier
  0.00      6.96     0.00      198     0.00     0.11  aml_register_handler
  0.00      6.96     0.00      174     0.00     0.00  mrg_get_double_orig
  0.00      6.96     0.00       88     0.00     0.00  mrg_seed
  0.00      6.96     0.00       87     0.00     0.00  make_mrg_seed
  0.00      6.96     0.00       10     0.00     0.00  xmalloc
  0.00      6.96     0.00        2     0.00     0.00  aml_n_pes
  0.00      6.96     0.00        1     0.00    45.42  convert_graph_to_oned_csr
  0.00      6.96     0.00        1     0.00     0.00  free_oned_csr_graph
  0.00      6.96     0.00        1     0.00     0.00  xcalloc
```

可以与我的结果有些差别，可能是我选的间隔还是太大，而且我忽略了函数栈的关系。不过可以看到前几名函数还是基本相同的。

参考文献

https://www.tutorialspoint.com/unix_commands/gprof.htm

https://www.hpl.hp.com/hosted/linux/mail-archives/libunwind/2003-December/000164.html