LG Electronics

Understanding the runtime behaviors of C++ programs using uftrace tool

https://github.com/namhyung/uftrace

Honggyu Kim(hong.gyu.kim@lge.com), Namhyung Kim(namhyung.kim@lge.com)

Introduction

uftrace is to trace and profile C/C++ user functions with compiler assist for function instrumentation. It allows user to understand the runtime behaviors and performance characteristics at the function level without source code modification. This poster introduces the basics of uftrace and its recently added enhanced features.

uftrace is able to trace

- C/C++ functions in user space programs
  - compiled with **-pg** or **-finstrument-functions** (or clang xray)
- Library functions (through PLT hooking)
- Linux kernel functions (with ftrace data integration)
- Some of events (SystemTap and kernel SDT, scheduler events, etc.)

uftrace Commands

- **record**
  - Run a command and record its trace data
- **replay**
  - Print recorded function trace
- **live**
  - Trace functions in a command lively
- **report**
  - Print statistics and summary for trace data
- **dump**
  - Print raw tracing data in the data files
- **info**
  - Print tracing information for trace data
- **recv**
  - Receive tracing data from socket and save it
- **graph**
  – Show function call graph
- **script**
  – Run a script for recorded function trace

Notable Options

- **-D DEPTH, --depth=DEPTH**
  Set global trace limit in nesting level.
- **-F FUNC, --filter=FUNC**
  Set filter to trace selected functions only.
- **-N FUNC, --notrace=FUNC**
  Set filter not trace selected functions only.
- **-T TRG, --trigger=TRG**
  Set trigger on selected functions.
- **-t TIME, --time-filter=TIME**
  Do not show small functions under the TIME threshold.
- **-A SPEC, --argument=SPEC**
  Record function arguments.
- **-R SPEC, --retval=SPEC**
  Record function return value.

Identifying constexpr function

```
// can be either compile time or runtime function
constexpr int fib(const int n) {
  if (n <= 2)
    return 1;
  return fib(n - 1) + fib(n - 2);
}

int main(int argc, char* argv[]) {
  if (argc > 1) {
    const int n = atoi(argv[1]);
    const int result = fib(n);  // run-time fib()
    printf("%d\n", result);
  } else {
    constexpr int n = 7;
    const int result = fib(n);  // compile-time fib()
    printf("%d\n", result);
  }
  return fib(5); // expect to be compile-time fib()
}

$ uftrace -A fib@arg1/i -R fib@retval \
          -A printf@arg1/s,arg2/i a.out
13
# DURATION     TID     FUNCTION
   1.540 us [16012] | __monstartup();
   0.900 us [16012] | __cxa_atexit();
            [16012] | main() {
   8.884 us [16012] |   printf("%d\n", 13);
            [16012] |   fib(5) {
            [16012] |     fib(4) {
            [16012] |       fib(3) {
   3.130 us [16012] |         fib(2) = 1;
   0.193 us [16012] |         fib(1) = 1;
   4.860 us [16012] |       } = 2; /* fib */
   0.210 us [16012] |       fib(2) = 1;
   5.873 us [16012] |     } = 3; /* fib */
            [16012] |     fib(3) {
   0.140 us [16012] |       fib(2) = 1;
   0.180 us [16012] |       fib(1) = 1;
   1.137 us [16012] |     } = 2; /* fib */
   7.880 us [16012] |   } = 5; /* fib */
  18.283 us [16012] | } /* main */
```

- Sometimes it's difficult to understand if constexpr function is evaluated at runtime. uftrace shows it clearly.

Avoid copying in std::string_view

```
$ cat string_view.cpp
#include <iostream>
#include <string>
#include <string_view>

void pr_string(const std::string& s)
{
  std::cout << s << '\n';
}

void pr_string_view(const std::string_view& sv)
{
  std::cout << sv << '\n';
}

int main()
{
  const char* msg = "long enough string";
  pr_string(msg);
  pr_string_view(msg);
}

$ g++ -std=c++1z -pg -O2 string_view.cpp
$ uftrace -F main a.out
long enough string
long enough string
# DURATION     TID     FUNCTION
            [18136] | main() {
   1.937 us [18136] |   operator new();
            [18136] |   print_string() {
  13.410 us [18136] |     std::__ostream_insert();
   0.554 us [18136] |     std::__ostream_insert();
  15.154 us [18136] |   } /* print_string */
   2.433 us [18136] |   operator delete();
            [18136] |   print_string_view() {
   0.300 us [18136] |     std::__ostream_insert();
   0.237 us [18136] |     std::__ostream_insert();
   1.390 us [18136] |   } /* print_string_view */
  23.957 us [18136] | } /* main */
```

- The above log shows that passing char* to std::string& creates a temp std::string while std::string_view doesn't.

STL containers performance comparison

```
std::vector<std::string> vec;
std::deque<std::string> deq;
std::list<std::string> lis;

int bench_vector_push_back(int i) {
  while (i--) vec.push_back(std::string("Hello"));
}
int bench_deque_push_back(int i) {
  while (i--) deq.push_back(std::string("Hello"));
}
int bench_list_push_back(int i) {
  while (i--) lis.push_back(std::string("Hello"));
}
int main()
{
  int iter = 3000000;
  bench_vector_push_back(iter);
  bench_deque_push_back(iter);
  bench_list_push_back(iter);
}

$ g++ bench.cpp -O3 -pg -fno-omit-frame-pointer
$ uftrace record --nest-libcall \
   -A malloc@arg1 -R malloc@retval -A free@arg1 \
   -A memcpy@arg3 -A memmove@arg3 ./a.out
$ uftrace graph
   4.261   s : (1) main
   1.272   s :  +-(1) bench_vector_push_back
   1.224   s :  | (23) std::vector::_M_insert_aux
 136.488 us :  |  +-(23) operator new
 116.618 us :  |  | (23) malloc
           :  |  |
 534.212 ms :  |  +-(4194326) memcpy
           :  |  |
   1.572 ms :  |  +-(22) operator delete
   1.545 ms :  |    (22) free
           :  |
 217.076 ms :  +-(1) bench_deque_push_back
 181.904 ms :  | (187500) std::deque::_M_push_back_aux
 111.554 ms :  |  +-(187515) operator new
  67.346 ms :  |  | (187515) malloc
           :  |  |
  16.473 ms :  |  +-(187500) memcpy
           :  |  |
   1.021 ms :  |  +-(15) memmove
           :  |  |
  10.447 us :  |  +-(15) operator delete
   4.962 us :  |    (15) free
           :  |
   2.771   s :  +-(1) bench_list_push_back
   1.186   s :     +-(3000000) operator new
 461.531 ms :     | (3000000) malloc
           :     |
 271.727 ms :     +-(3000000) memcpy
           :     |
 221.041 ms :     +-(3000000) std::__detail::_List_node_base::_M_hook

$ uftrace replay -f none -F main \
  -F bench_.* -F malloc -F free \
  -F memcpy -F memmove -D 1

# FUNCTION
 malloc(64) = 0x23a32d0;
 malloc(512) = 0x23a3320;
 main() {
   bench_vector_push_back() {
     malloc(32) = 0x23a3530;
     memcpy(5);
     malloc(64) = 0x23a3560;
     memcpy(5);
     memcpy(5);
     free(0x23a3530);
     malloc(128) = 0x23a35b0;
     memcpy(5);
     memcpy(5);
     memcpy(5);
     free(0x23a3560);
     malloc(256) = 0x23a3640;
     memcpy(5);
     memcpy(5);
     memcpy(5);
     memcpy(5);
     memcpy(5);
     free(0x23a35b0);
       ...
     memcpy(5);
     free(0x7fbafd1c5010);
   } /* bench_vector_push_back */
   bench_deque_push_back() {
     malloc(512) = 0x23a3530;
     memcpy(5);
     malloc(512) = 0x23a3740;
     memcpy(5);
     malloc(512) = 0x23a3950;
     memcpy(5);
     malloc(512) = 0x23a3b60;
     memcpy(5);
     malloc(144) = 0x23a3d70;
     memmove(40);
     free(0x23a32d0);
       ...
     malloc(512) = 0x84c37d0;
     memcpy(5);
   } /* bench_deque_push_back */
   bench_list_push_back() {
     malloc(48) = 0x23a32d0;
     memcpy(5);
     malloc(48) = 0x25af360;
     memcpy(5);
     malloc(48) = 0x23a7080;
     memcpy(5);
       ...
   } /* bench_list_push_back */
 } /* main */
```

(Python) Scripting Support (-S/--script)

- uftrace is able to run (python) script
  - for each C/C++ function

Script APIs

- **uftrace_entry**(**context**)
  - runs at every function entry
- **uftrace_exit**(**context**)
  - runs at every function exit
- **uftrace_begin**()
  - runs only once when program begins
- **uftrace_end**()
  - runs only once when program finishes

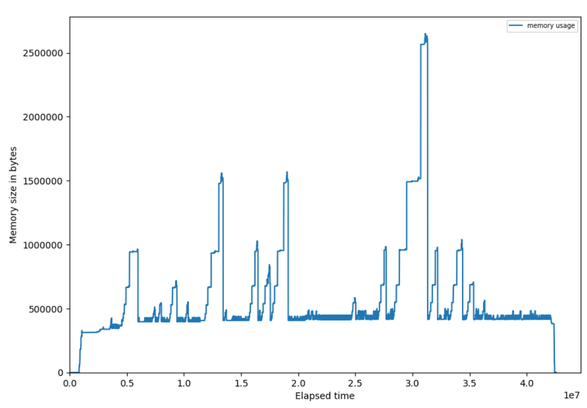
```
$ cat scripts/count.py
count = 0

def uftrace_begin():
    pass

def uftrace_entry(args):
    global count
    count += 1

def uftrace_exit(args):
    pass

def uftrace_end():
    print(count)
```

Context Information

- uftrace passes necessary context information to script.
- **uftrace_entry**() and **uftrace_exit**() have **context** parameter
- context is a python **dictionary** type and it includes

```
/* context info passed to script */
script_context = {
  int       tid;
  int       depth;
  long      timestamp;
  long      duration;  # exit only
  long      address;
  string    name;
  list      args;      # entry only
  value     retval;    # exit  only
};
```

- script allows uftrace to draw graphical chart (with **matplotlib**)
  - from "**malloc**" and "**free**" to draw memory allocation status.



Automatic arguments/return values display (WIP)

```
$ uftrace --force -t 200us --auto-args /usr/bin/gcc hello.c
# DURATION     TID     FUNCTION
            [88132] | } = 0; /* vfork */
            [88132] | execv("/usr/lib/gcc/x86_64-linux-gnu/5/cc1") {
            [88133] | } = 0; /* vfork */
            [88133] | execvp("as") {
 361.230 us [88133] | memset(0x7fce671b6020, 0, 524296) = 0x7fce671b6020;
 366.410 us [88133] | memset(0x7fce64e3a020, 0, 524296) = 0x7fce64e3a020;
 365.573 us [88133] | memset(0x7fce64db9020, 0, 524296) = 0x7fce64db9020;
 354.307 us [88133] | memset(0x7fce64d38020, 0, 524296) = 0x7fce64d38020;
 357.133 us [88133] | memset(0x7fce64cb7020, 0, 524296) = 0x7fce64cb7020;
 360.316 us [88133] | memset(0x7fce64c36020, 0, 524296) = 0x7fce64c36020;
 364.249 us [88133] | memset(0x7fce64bb5020, 0, 524296) = 0x7fce64bb5020;
            [88134] | } = 0; /* vfork */
            [88134] | } = 88134; /* vfork */
 205.504 us [88134] | gettext();
            [88134] | vfork() {
            [88135] | } = 0; /* vfork */
            [88135] | execvp("/usr/bin/ld") {
 359.919 us [88134] | } = 88135; /* vfork */
            [88134] | waitpid(88135, 0x1560650, 0) {
 314.197 us [88135] | bfd_link_hash_traverse();
   2.746 ms [88135] | bfd_elf_size_dynamic_sections();
   1.492 ms [88135] | bfd_elf_size_dynsym_hash_dynstr();
 373.356 us [88135] | _bfd_fix_excluded_sec_syms();
 340.170 us [88135] | bfd_close();
  49.408 ms [88134] | } = 88135; /* waitpid */
```

Tracing nested library calls (--nest-libcall)

```
$ uftrace --nest-libcall --auto-args /usr/bin/clang hello.c
# DURATION     TID     FUNCTION
   0.284 us [175968] | strlen("/usr/bin/ld") = 11;
            [175968] | llvm::sys::commandLineFitsWithinSystemLimits() {
           ...
  21.584 us [175968] | } /* llvm::sys::commandLineFitsWithinSystemLimits */
   0.197 us [175968] | llvm::opt::ArgList::getLastArg();
   0.420 us [175968] | memcpy(0x7ffc7ba7a020, 0x28a07d0, 384) = 0x7ffc7ba7a020;
   0.323 us [175968] | strlen("/usr/lib/llvm-3.8/bin/clang") = 27;
            [175968] | llvm::sys::ExecuteAndWait() {
   0.360 us [175968] |   memcpy(0x7ffc7ba79b18, 0x2883dc0, 27) = 0x7ffc7ba79b18;
   3.093 us [175968] |   access();
   0.153 us [175968] |   std::_V2::system_category();
            [175968] |   std::__cxx11::basic_string::_M_create() {
            [175968] |     operator new() {
   0.490 us [175968] |       malloc(28) = 0x28a1150;
   1.053 us [175968] |     } /* operator new */
   1.566 us [175968] |   } /* std::__cxx11::basic_string::_M_create */
   0.253 us [175968] |   memcpy(0x28a1150, 0x2883dc0, 27) = 0x28a1150;
 247.286 us [175968] |   posix_spawn();
            [175968] |   operator delete() {
   0.590 us [175968] |     free(0x28a1150);
   1.500 us [175968] |   } /* operator delete */
            [175968] |   waitpid(175980, 0x7ffc7ba79bfc, 0) {
           ...
```

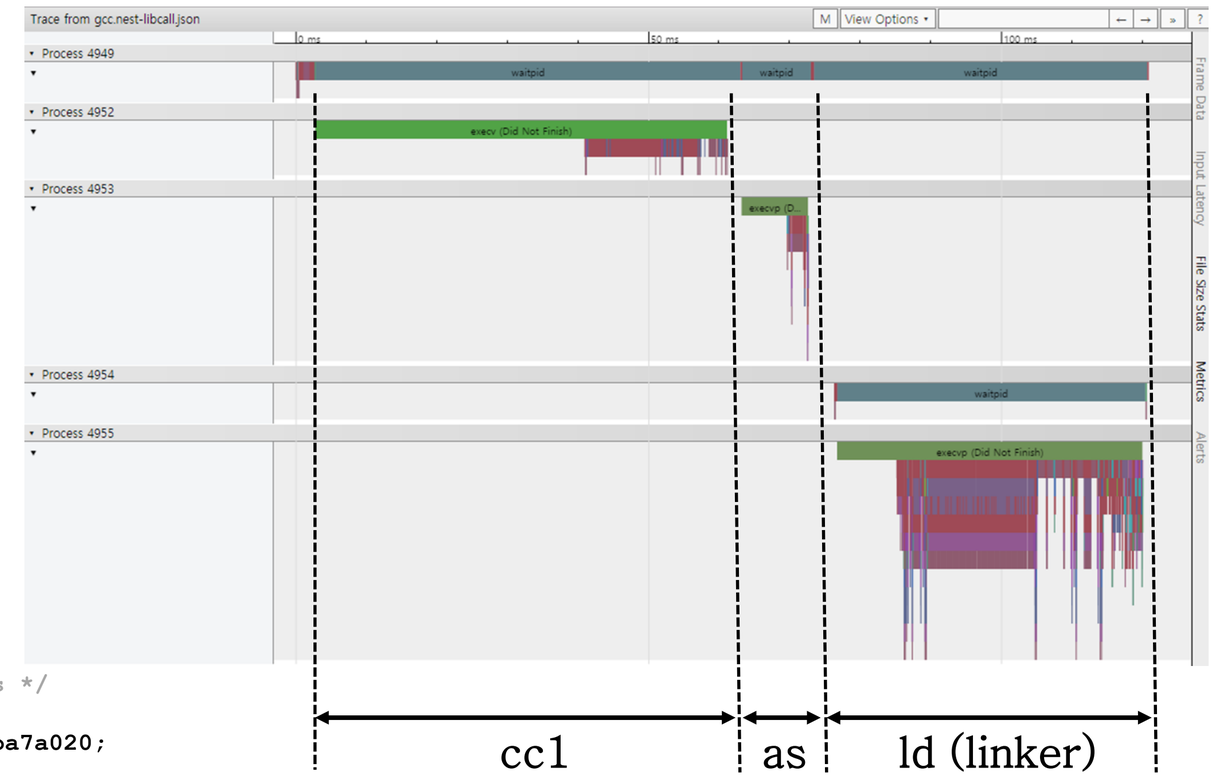
Using Google Chrome Tracing Facility

The recorded data by uftrace can be dumped as JSON style output that can be loaded by Google chrome browser. The output JSON file can also be converted into HTML file that can be easily shared as a web link.

Compilation Procedure Study of GCC

(without source code recompilation)

The below trace result is generated by pre-built gcc binary without recompilation.
It shows that the master process manages and waits for cc1, as, and ld in order.



Parallel Algorithm in C++17 - std::sort

The below trace result is generated by uftrace after running "sort" algorithm for std::vector in parallel.
It shows that the parallel algorithm is based on OpenMP. It uses libstdc++ in g++-7.1.0.