

报告四：调试与性能分析、元编程和 PyTorch 编程

23020007011 崔涛

2024 年 9 月 15 日

1 实验目的

本次课程主要讲授了调试及性能分析，元编程和 PyTorch 编程，通过学习这些工具可以帮助我们优化模型性能，提高代码质量和加速以后的开发过程。

2 介绍

2.1 调试及性能分析，元编程和 PyTorch 编程

- 调试及性能分析的工具提供了可视化界面，使得开发者能够直观地看到程序运行过程中的性能数据，从而更容易发现潜在问题，还能提供关于如何优化瓶颈的具体建议，帮助开发者改进模型性能。
- 元编程允许开发者在运行时动态地创建和修改代码，这为 PyTorch 等深度学习框架提供了极大的灵活性。通过元编程，可以自动化许多重复性的编程任务，减少人为错误并提高开发效率。
- PyTorch 是一个广泛使用的深度学习框架，它提供了丰富的 API 和工具来构建、训练和部署深度学习模型。PyTorch 以其易用性和灵活性而闻名，支持多种编程范式（如命令式和函数式编程），并且与 Python 生态系统紧密集成。

3 练习内容

3.1 学习样例 20 个

1. 使用 Linux 上的 `journalctl` 命令来获取最近一天中超级用户的登录信息及其所执行的指令。
输入 `journalctl` 来得到超级用户的执行和登录信息，输入 `journalctl|grep sudo` 来得到超级用户执行的指令。

```
cuttao@ubuntu:~$ journalctl
-- Logs begin at Sun 2024-09-15 19:40:35 CST, end at Sun 2024-09-15 19:41:30 CS
Sep 15 19:40:35 ubuntu systemd-journald[338]: Runtime journal (/run/log/journal
Sep 15 19:40:35 ubuntu kernel: Linux version 4.15.0-142-generic (buildd@lgw01-a
Sep 15 19:40:35 ubuntu kernel: Command line: BOOT_IMAGE=/boot/vmlinuz-4.15.0-14
Sep 15 19:40:35 ubuntu kernel: KERNEL supported cpus:
Sep 15 19:40:35 ubuntu kernel: Intel GenuineIntel
Sep 15 19:40:35 ubuntu kernel: AMD AuthenticAMD
Sep 15 19:40:35 ubuntu kernel: Centaur CentaurHauls
Sep 15 19:40:35 ubuntu kernel: x86/fpu: Supporting XSAVE feature 0x001: 'x87 fl
Sep 15 19:40:35 ubuntu kernel: x86/fpu: Supporting XSAVE feature 0x002: 'SSE re
Sep 15 19:40:35 ubuntu kernel: x86/fpu: Supporting XSAVE feature 0x004: 'AVX re
Sep 15 19:40:35 ubuntu kernel: x86/fpu: xstate_offset[2]: 576, xstate_sizes[2]
Sep 15 19:40:35 ubuntu kernel: x86/fpu: Enabled xstate features 0x7, context si
Sep 15 19:40:35 ubuntu kernel: e820: BIOS-provided physical RAM map:
Sep 15 19:40:35 ubuntu kernel: BIOS-e820: [mem 0x0000000000000000-0x00000000000
Sep 15 19:40:35 ubuntu kernel: BIOS-e820: [mem 0x00000000000009e800-0x0000000000000
Sep 15 19:40:35 ubuntu kernel: BIOS-e820: [mem 0x000000000000dc000-0x0000000000000
Sep 15 19:40:35 ubuntu kernel: BIOS-e820: [mem 0x00000000000100000-0x00000000000bfe
Sep 15 19:40:35 ubuntu kernel: BIOS-e820: [mem 0x00000000000bfed0000-0x00000000000bfe
Sep 15 19:40:35 ubuntu kernel: BIOS-e820: [mem 0x00000000000bfff000-0x00000000000bffe
Sep 15 19:40:35 ubuntu kernel: BIOS-e820: [mem 0x00000000000bfff0000-0x00000000000bfff
Sep 15 19:40:35 ubuntu kernel: BIOS-e820: [mem 0x000000000f0000000-0x000000000f7f
Sep 15 19:40:35 ubuntu kernel: BIOS-e820: [mem 0x00000000fec000000-0x00000000fec
```

图 1: 获取超级用户执行信息

2. 安装 **shellcheck** 并尝试对脚本进行检查。这段代码有什么问题吗？
安装 **shellcheck** 后输入 **shellcheck** 加文件名字，即可检查

```
cuttao@ubuntu:~/Desktop$ shellcheck sorts.py

In sorts.py line 1:
import random
^-- SC2140: Tips depend on target shell and yours is unknown. Add a shebang.

In sorts.py line 4:
def test_sorted(fn, iters=1000):
^-- SC1070: Parsing stopped here because of parsing errors.
^-- SC1030: '(' is invalid here. Did you forget to escape it?
```

图 2: Shellcheck 检查代码

3. 使用 **cProfile** 来比较插入排序和快速排序的性能
输入如下代码使用 **cprofile** 比较测试，`python -m cProfile -s time sorts.py` 比较得出：在测试情况下，插入排序的性能比较好。

```
ubuntu:~$ python -m cProfile -s time sorts.py
399743 function calls (333161 primitive calls) in 0.178 seconds

Ordered by: internal time

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
78702  0.053    0.000    0.057    0.000 random.py:177(randrange)
1690/1000  0.028    0.000    0.030    0.000 sorts.py:23(quicksort)
1892/1000  0.025    0.000    0.029    0.000 sorts.py:32(quicksort_inplace)
1000    0.019    0.000    0.019    0.000 sorts.py:11(insertionsort)
3       0.018    0.006    0.176    0.059 sorts.py:4(test sorted)
```

图 3: CProfile 比较性能

4. 限制进程资源也是一个非常有用的技术。执行 **stress -c 3** 并使用 **htop** 对 CPU 消耗进行可视化。先输入 **sudo apt install htop** 下载 **htop**，再输入 **stress -c 3** 来创建负载，再输入 **taskset -cpu-list 0,2 stress -c 3** 限制负载，使用 **htop** 命令比较前后差异。



图 6: 限制指令

使用 `line_profiler` 工具比较性能需要在代码中相应位置插入装饰器 `@profile`, 插入后再输入指令 `kernprof -l -v sorts.py` 即可得出代码块性能。

```
wrote profile results to sorts.py.tprof
Timer unit: 1e-06 s

Total time: 0.201273 s
File: sorts.py
Function: insertionsort at line 10
```

Line #	Hits	Time	Per Hit	% Time	Line Contents
10					@profile
11					def insertionsort(array):
12					
13	25978	6586.0	0.3	3.3	for i in range(len(array)):
14	24978	6297.0	0.3	3.1	j = i-1
15	24978	6478.0	0.3	3.2	v = array[i]
16	227751	66245.0	0.3	32.9	while j >= 0 and v < ar
17	202773	57803.0	0.3	28.7	array[j+1] = array[
18	202773	50480.0	0.3	25.1	array[j] = array[

图 7: Line_profiler 分析算法性能

6.lsof 命令的使用命令格式为 **lsof [选项] [绝对路径的文件名]** 列出被各种进程打开的文件信息

```
cuictao@ubuntu: ~
8153 /lib/x86_64-linux-gnu/libc-2.23.so
gdbus 2451 2461 cuictao mem REG 8,1 138696 39
8131 /lib/x86_64-linux-gnu/libpthread-2.23.so
gdbus 2451 2461 cuictao mem REG 8,1 1115136 39
2752 /lib/x86_64-linux-gnu/libglib-2.0.so.0.4800.2
gdbus 2451 2461 cuictao mem REG 8,1 339000 13
0959 /usr/lib/x86_64-linux-gnu/libgobject-2.0.so.0.4800.2
gdbus 2451 2461 cuictao mem REG 8,1 39800 14
3298 /usr/lib/x86_64-linux-gnu/libgudev-1.0.so.0.2.0
gdbus 2451 2461 cuictao mem REG 8,1 1599336 13
1907 /usr/lib/x86_64-linux-gnu/libgio-2.0.so.0.4800.2
gdbus 2451 2461 cuictao mem REG 8,1 43160 14
3221 /usr/lib/x86_64-linux-gnu/libgphoto2_port.so.12.0.0
gdbus 2451 2461 cuictao mem REG 8,1 145744 14
3219 /usr/lib/x86_64-linux-gnu/libgphoto2.so.6.0.0
gdbus 2451 2461 cuictao mem REG 8,1 162632 39
8143 /lib/x86_64-linux-gnu/ld-2.23.so
gdbus 2451 2461 cuictao mem REG 8,1 91381 108
4730 /usr/share/locale-langpack/zh_CN/LC_MESSAGES/glib20.mo
gdbus 2451 2461 cuictao mem REG 8,1 126840 39
7924 /lib/x86_64-linux-gnu/libudev.so.1.6.4
gdbus 2451 2461 cuictao mem REG 8,1 48338 108
4635 /usr/share/locale-langpack/zh_CN/LC_MESSAGES/gvfs.mo
gdbus 2451 2461 cuictao mem REG 8,1 26258 26
```

图 8: lsof 命令

7.strace 调试命令 **strace <命令>**: 跟踪系统调用和信号。它可以显示程序执行过程中所调用的所有系统调用以及这些调用的参数、返回值等信息。例如: **strace ls -l**会显示执行**ls -l**命令时的系统调用信息。

```
cuictao@ubuntu: ~
available)
stat("out.log", {st_mode=S_IFREG|0664, st_size=32, ...}) = 0
getxattr("out.log", "security.selinux", 0x1f35dd0, 255) = -1 ENODATA (No data available)
getxattr("out.log", "system.posix_acl_access", NULL, 0) = -1 ENODATA (No data available)
stat("vin", {st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0
getxattr("vin", "security.selinux", 0x1f35df0, 255) = -1 ENODATA (No data available)
getxattr("vin", "system.posix_acl_access", NULL, 0) = -1 ENODATA (No data available)
getxattr("vin", "system.posix_acl_default", NULL, 0) = -1 ENODATA (No data available)
open("/etc/passwd", O_RDONLY|O_CLOEXEC) = 4
lseek(4, 0, SEEK_CUR) = 0
stat(4, {st_mode=S_IFREG|0644, st_size=2410, ...}) = 0
mmap(NULL, 2410, PROT_READ, MAP_SHARED, 4, 0) = 0x7eff40626000
lseek(4, 2410, SEEK_SET) = 2410
munmap(0x7eff40626000, 2410) = 0
close(4) = 0
open("/etc/group", O_RDONLY|O_CLOEXEC) = 4
lseek(4, 0, SEEK_CUR) = 0
stat(4, {st_mode=S_IFREG|0644, st_size=1051, ...}) = 0
mmap(NULL, 1051, PROT_READ, MAP_SHARED, 4, 0) = 0x7eff40626000
```

图 9: Strace 调试

8.ltrace 指令 **ltrace <命令>**: 跟踪程序运行过程中库函数的调用情况。比如查看一个程序在运行时调用了哪些动态链接库中的函数以及这些函数的参数和返回值。例如: **ltrace my_program**可以跟踪**my_program**运行时的库函数调用

```

cui tao@ubuntu: ~
errno_location() = 0x7fe0fa3d6b8
ctype_get_mb_cur_max() = 6
ctype_get_mb_cur_max() = 6
errno_location() = 0x7fe0fa3d6b8
ctype_get_mb_cur_max() = 6
ctype_get_mb_cur_max() = 6
write_unlocked("myssh", 1, 5, 0x7fe0f60e620) = 5
overflow(0x7fe0f60e620, 10, 0, 0x68737379myssh
) = 10
errno_location() = 0x7fe0fa3d6b8
ctype_get_mb_cur_max() = 6
ctype_get_mb_cur_max() = 6
errno_location() = 0x7fe0fa3d6b8
ctype_get_mb_cur_max() = 6
ctype_get_mb_cur_max() = 6
write_unlocked("myssh.pub", 1, 9, 0x7fe0f60e620) = 9
overflow(0x7fe0f60e620, 10, 0, 0x6275702e68737379myssh.pub
) = 10
errno_location() = 0x7fe0fa3d6b8
ctype_get_mb_cur_max() = 6
ctype_get_mb_cur_max() = 6
errno_location() = 0x7fe0fa3d6b8
ctype_get_mb_cur_max() = 6

```

图 10: ltrace 指令

9.valgrind 相关

- `valgrind --tool=memcheck <可执行程序>`: 用于检测程序中的内存错误，比如内存泄漏、越界访问、使用未初始化的内存等。它可以提供详细的内存错误信息以及错误发生的位置。
- `valgrind --tool=callgrind <可执行程序>`: 分析程序中函数的调用关系和执行时间，用于性能调优，可帮助找出程序中哪些函数占用了过多的时间。

```

root@init1 valgrind]# valgrind ./test
==16250== Memcheck, a memory error detector
==16250== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==16250== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==16250== Command: ./test
==16250==
==16250== Invalid write of size 4
==16250==    at 0x40082E: func() (test.cpp:6)
==16250==    by 0x40083E: main (test.cpp:9)
==16250== Address 0x5a23068 is 0 bytes after a block of size 40 alloc'd
==16250==    at 0x4C29E33: malloc (vg_replace_malloc.c:299)
==16250==    by 0x400821: func() (test.cpp:5)
==16250==    by 0x40083E: main (test.cpp:9)
==16250==
done

```

图 11: valgrind 指令

10.core dump 相关

`ulimit -c unlimited`: 设置 core 文件大小无限制。当程序崩溃时，系统会生成一个 core 文件，里面包含了程序崩溃时的内存、寄存器等信息。

11.gdb 调试分析

`gdb <可执行程序> <core 文件>`: 利用生成的 core 文件进行调试，可以找出程序崩溃的位置以及当时的变量状态等信息。例如，程序崩溃后生成了 core 文件，可以通过 `gdb my_program core` 来进行调试分析。

12.nm

- 格式: `nm <可执行文件或目标文件>`。
- 作用: 列出目标文件或可执行文件中的符号，比如全局函数、全局变量等，可用于检查符号是否正确链接等问题。

```

cuitao@ubuntu:~/Desktop$ gcc nm_test.c
cuitao@ubuntu:~/Desktop$ nm a.out
00000000004004d6 T add
000000000060103a B __bss_start
000000000060103c b completed.7594
0000000000601020 D __data_start
0000000000601020 W data_start
0000000000400410 t deregister_tm_clones
0000000000400490 t __do_global_dtors_aux
0000000000600e18 t __do_global_dtors_aux_fini_array_entry
0000000000601028 D __dso_handle
0000000000600e28 d _DYNAMIC
000000000060103a D __edata
0000000000601048 B __end
0000000000400574 T __fini
00000000004004b0 t __frame_dummy
0000000000600e10 t __frame_dummy_init_array_entry
00000000004006d0 r __FRAME_END__
0000000000601040 B global_a
0000000000601030 D global_b
0000000000601000 d _GLOBAL_OFFSET_TABLE_
                                w __gmon_start__
0000000000400584 r __GNU_EH_FRAME_HDR
0000000000400390 T __init
0000000000600e18 t __init_array_end
0000000000600e10 t __init_array_start
0000000000400580 R __IO_stdin_used
                                w __ITM_deregisterTMCloneTable

```

图 12: nm 指令

13.readelf

- 格式: `readelf -a <可执行文件或目标文件>`。
- 作用: 显示 elf 文件（可执行文件和目标文件在 Linux 下多为 elf 格式）的详细信息，包括头信息、段信息、符号表等，对分析文件结构和链接相关问题有帮助。

```

cuitao@ubuntu:~/Desktop$ readelf -h a.out
ELF 头:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  类别:                ELF64
  数据:                2 补码, 小端序 (little endian)
  版本:                1 (current)
  OS/ABI:              UNIX - System V
  ABI 版本:            0
  类型:                EXEC (可执行文件)
  系统架构:            Advanced Micro Devices X86-64
  版本:                0x1
  入口点地址:          0x4003e0
  程序头起点:          64 (bytes into file)
  Start of section headers: 6704 (bytes into file)
  标志:                0x0
  本头的大小:          64 (字节)
  程序头大小:          56 (字节)
  Number of program headers: 9
  节头大小:            64 (字节)
  节头数量:            31
  字符串表索引节头:    28

```

图 13: readelf 指令

14. 使用 `uname` 查看计算机信息 `uname` 可显示电脑以及操作系统的相关信息，例如内核版本、主机名、处理器类型等。

```

cuitao@ubuntu:~/Desktop$ uname
linux

```

图 14: uname 指令

15.objdump

- 格式: `objdump -d <可执行文件或目标文件>`。

- 作用：用于反汇编可执行文件或者目标文件，可以查看机器码对应的汇编指令，帮助调试程序在底层的执行逻辑。

```
cultao@ubuntu:~/Desktop$ objdump -d a.out
a.out:      文件格式 elf64-x86-64

Disassembly of section .init:

0000000000400390: <_init>:
400390:  48 83 ec 08      sub    $0x8,%rsp
400394:  48 8b 05 5d 0c 20 00 mov    0x200c5d(%rip),%rax # 6
f8 <_DYNAMIC+0x1d0>
40039b:  48 85 c0         test   %rax,%rax
40039e:  74 05          je     4003a5 <_init+0x15>
4003a0:  e8 2b 00 00 00   callq 4003d0 <__libc_start_main@plt+
0>
4003a5:  48 83 c4 08      add    $0x8,%rsp
4003a9:  c3             retq

Disassembly of section .plt:

00000000004003b0: <__libc_start_main@plt+0x10>:
```

图 15: objdump 指令

16.dmesg

- 格式：直接运行dmesg。
- 作用：查看内核环形缓冲区的信息，其中可能包含硬件相关的错误信息、驱动加载相关信息等，对调试与内核相关或者硬件相关的问题提供参考。

```
3.064968] ata5: SATA link down (SStatus 0 SControl 300)
3.065128] ata4.00: configured for UDMA/33
3.066230] scsi 4:0:0:0: CD-ROM           NECVMWar VMware SATA CD01 1.0
0 ANSI: 5
3.068971] ata10: SATA link down (SStatus 0 SControl 300)
3.068999] ata13: SATA link down (SStatus 0 SControl 300)
3.069048] ata11: SATA link down (SStatus 0 SControl 300)
3.069062] ata9: SATA link down (SStatus 0 SControl 300)
3.069075] ata14: SATA link down (SStatus 0 SControl 300)
3.069091] ata12: SATA link down (SStatus 0 SControl 300)
3.069955] ata7: SATA link down (SStatus 0 SControl 300)
3.069999] ata8: SATA link down (SStatus 0 SControl 300)
3.075011] ata16: SATA link down (SStatus 0 SControl 300)
3.075035] ata21: SATA link down (SStatus 0 SControl 300)
3.075105] ata22: SATA link down (SStatus 0 SControl 300)
3.075168] ata23: SATA link down (SStatus 0 SControl 300)
3.075213] ata15: SATA link down (SStatus 0 SControl 300)
3.075257] ata17: SATA link down (SStatus 0 SControl 300)
3.075280] ata24: SATA link down (SStatus 0 SControl 300)
3.075324] ata19: SATA link down (SStatus 0 SControl 300)
3.075363] ata25: SATA link down (SStatus 0 SControl 300)
3.075414] ata20: SATA link down (SStatus 0 SControl 300)
3.075460] ata18: SATA link down (SStatus 0 SControl 300)
3.080516] ata26: SATA link down (SStatus 0 SControl 300)
```

图 16: dmesg 指令

17.perf

- 格式：例如perf record <命令>进行数据采集，perf report查看报告。
- 作用：用于 Linux 性能分析，它可以监测诸如 CPU 时钟周期、指令执行次数、缓存命中率等多种性能相关的数据，能帮助定位性能瓶颈所在。

18.stap

- 格式：编写 SystemTap 脚本，然后使用stap <脚本>运行。
- 作用：可以动态地收集和分析内核及用户空间程序的运行时信息，可用于调试复杂的系统级问题，比如内核模块间的交互、系统调用的行为等。

19.gprof

- 格式：先使用gcc -pg <源文件>编译程序，运行程序后再执行gprof <可执行程序>。
- 作用：用于分析程序的性能瓶颈，它可以给出每个函数被调用的次数以及在每个函数上花费的时间等信息，帮助开发者进行性能优化相关的调试工作。

```
cuitao@ubuntu:~/Desktop$ gcc -pg nm_test.c
cuitao@ubuntu:~/Desktop$ gprof a.out
gmon.out: No such file or directory
cuitao@ubuntu:~/Desktop$ ./a.out
cuitao@ubuntu:~/Desktop$ gprof a.out
Flat profile:

Each sample counts as 0.01 seconds.
no time accumulated

%   cumulative   self           self       total
time  seconds    seconds   calls   Ts/call   Ts/call   name
%
time      the percentage of the total running time of the
         program used by this function.
cumulative a running sum of the number of seconds accounted
seconds    for by this function and those listed above it.
self
seconds    the number of seconds accounted for by this
         function alone. This is the major sort for this
         listing.
calls      the number of times this function was invoked, if
         this function is profiled, else blank.
self
ns/call    the average number of milliseconds spent in this
         function per call, if this function is profiled,
         else blank.
```

图 17: gprof 命令

20.systemctl status

- 格式：systemctl status <服务名称>。
- 作用：当调试与系统服务相关的问题时，这个指令可以查看服务的运行状态、最近的日志信息等，帮助判断服务是否正常工作以及出现问题的可能原因。

```
cuitao@ubuntu:~/Desktop$ systemctl status
● ubuntu
   State: running
   Jobs: 0 queued
   Failed: 0 units
   Since: Sun 2024-09-15 20:19:21 CST; 1h 22min ago
   CGroup: /
           └─user.slice
               └─user-1000.slice
                   └─user@1000.service
                       └─init.scope
                           └─1923 /lib/systemd/systemd --user
                               └─1924 (sd-pam)
                                   └─session-c2.scope
                                       └─1166 lightdm --session-child 12 19
                                           └─1929 /usr/bin/gnome-keyring-daemon --daemonize --login
                                               └─1931 /sbin/upstart --user
                                                   └─1998 upstart-udev-bridge --daemon --user
                                                       └─2009 dbus-daemon --fork --session --address=unix:abstract=
                                                           └─2021 /usr/lib/x86_64-linux-gnu/hud/window-stack-bridge
                                                               └─2047 /usr/bin/fcitx
                                                                   └─2055 /usr/bin/dbus-daemon --fork --print-pid 5 --print-addr
                                                                       └─2070 upstart-file-bridge --daemon --user
                                                                           └─2073 upstart-dbus-bridge --daemon --session --user --bus-na
                                                                               └─2075 /usr/bin/fcitx-dbus-watcher unix:abstract=/tmp/dbus-s
                                                                                   └─2078 /usr/lib/x86_64-linux-gnu/bamf/bamfdaemon
                                                                                       └─2080 upstart-dbus-bridge --daemon --system --user --bus-na
                                                                                           └─2082 /usr/lib/x86_64-linux-gnu/notify-osd
                                                                                               └─2092 /usr/lib/at-spi2-core/at-spi-bus-launcher
```

图 18: systemctl status 命令

4 收获感悟

在学习 Linux 下调试与分析的过程中，我收获颇丰。我了解了各种调试工具如 GDB 。通过 GDB，我能深入到程序运行的内部，逐行跟踪代码执行，精准定位到程序崩溃或者出现异常的位置。我学会了分析内存相关问题，理解了内存布局并且学会使用一些内存泄漏检测工具，这有助于我优化程序的内存使用，避免因内存问题导致的程序不稳定。同时，学习分析系统调用和内核日志也让我对 Linux 系统的运行机制有了更深入的认识。从系统调用的角度去理解程序与系统的交互，能发现隐藏在程序运行背后的问题根源。总的来说，本次学习极大丰富了我的技能。它不仅能提高我的程序的质量和稳定性，还能加深我对操作系统原理的理解，为进一步深入学习和开发工作奠定了坚实的基础。

提交记录如下：

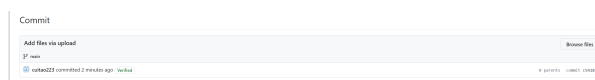


图 19: 提交记录

github 路径您可以在这里查看项目的源代码:

<https://github.com/cuitao223/homework4>