Lecture 21 - More Profiling Tools

ECE 459: Programming for Performance

Jon Eyolfson

University of Waterloo

February 29, 2012

Introduction

- Google Performance Tools include:
  - CPU profiler
  - Heap profiler
  - Heap checker
  - Faster `malloc`

- We'll mostly use the CPU profiler
  - Purely statistical sampling
  - No recompilation, or just linking
  - Built-in visual output

Usage

- You can use the profiler without any recompilation
  - Not recommended

```
LD_PRELOAD="/usr/lib/libprofiler.so" CPUPROFILE=test.prof
./test
```

- The other option is to link to the profiler
  - `-lprofiler`
- Both options read the `CPUPROFILE` environment variable
  - Location to write the profile data

Other Usage

- You can use the profiling library directly as well:
  - `#include <gperftools/profiler.h>`
- Bracket code you want profiled with:
  - `ProfilerStart()`
  - `ProfilerEnd()`

- You can change the sampling frequency with the `CPUPROFILE_FREQUENCY` environment variable
  - **Default value:** 100

pprof Usage

- Similar to gprof, it will analyze the results

```
% pprof test test.prof
    Enters "interactive" mode
% pprof --text test test.prof
    Outputs one line per procedure
% pprof --gv test test.prof
     Displays annotated call-graph via 'gv'
% pprof --gv --focus=Mutex test test.prof
    Restricts to code paths including a .*Mutex.* entry
% pprof --gv --focus=Mutex --ignore=string test test.prof
    Code paths including Mutex but not string
% pprof --list=getdir test test.prof
    (Per-line) annotated source listing for getdir()
% pprof --disasm=getdir test test.prof
    (Per-PC) annotated disassembly for getdir()
```

- Also output dot, ps, pdf or gif instead of gv

Text Output

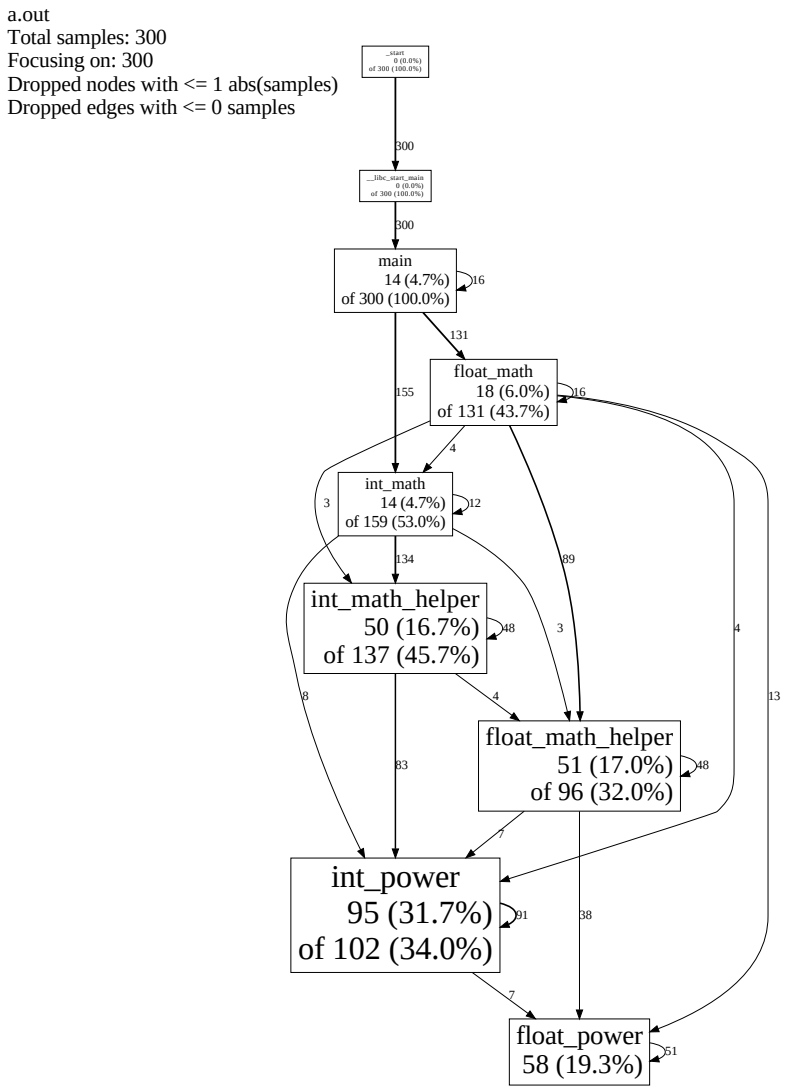
- Similar to the flat profile in `gprof`

```
jon@riker examples master % pprof --text test test.prof
Using local file test.
Using local file test.prof.
Removing killpg from all stack traces.
Total: 300 samples
      95  31.7%  31.7%      102  34.0% int_power
      58  19.3%  51.0%       58  19.3% float_power
      51  17.0%  68.0%       96  32.0% float_math_helper
      50  16.7%  84.7%      137  45.7% int_math_helper
      18   6.0%  90.7%      131  43.7% float_math
      14   4.7%  95.3%      159  53.0% int_math
      14   4.7% 100.0%      300 100.0% main
       0   0.0% 100.0%      300 100.0% __libc_start_main
       0   0.0% 100.0%      300 100.0% _start
```

Text Output Explained

- Number of checks in this function
- Percentage of checks in this function
  - Same as **time** in `gprof`
- Percentage of checks in the functions printed so far
  - Equivalent to **cumaltive** (but in %)
- Number of checks in this function and its callees
- Percentage of checks in this function and its callees
- Function name

Graphical Output



Graphical Output Explained

Likely the output this will be too small to read on the slide

- Shows the same numbers as the text output
- An edge tells you the function calls the one pointed to
- Number of samples in callees = Number in this function and callees - Number in this function

- **Example (`float_math_helper`):** 91 - 51 = 45
  - To `int_power` = 7
  - To `float_power` = 38
  - Total = 45

Things You May Notice

- The call graph is not exact
  - In fact, it shows many relations we know don't exist
  - For instance, the `int` and `float` functions are separate
- Similar to `gprof` more optimizations enabled will change the graph
- You'll probably want to look at the text profile first, then use the `-focus` flag to look at individual functions

Introduction

- Uses CPU clock cycles, checks the current function
- Another sampling-based tool
- Runs as a Linux kernel module
- Records profiling data for every application run

Setup

- Since this is a system profiler, you have to start it as root

```
% sudo opcontrol
      --vmlinux=/usr/src/linux-3.2.7-1-ARCH/vmlinux
% echo 0 | sudo tee /proc/sys/kernel/nmi_watchdog
% sudo opcontrol --start
Using default event: CPU_CLK_UNHALTED:100000:0:1:1
Using 2.6+ OProfile kernel interface.
Reading module info.
Using log file /var/lib/oprofile/samples/oprofiled.log
Daemon started.
Profiler running.
```

Usage (1)

- Use `opreport` and your executable

```
% sudo opreport -l ./test
CPU: Intel Core/i7, speed 1595.78 MHz (estimated)
Counted CPU_CLK_UNHALTED events (Clock cycles when not
halted) with a unit mask of 0x00 (No unit mask) count 100000
samples  %        symbol name
7550      26.0749  int_math_helper
5982      20.6596  int_power
5859      20.2348  float_power
3605      12.4504  float_math
3198      11.0447  int_math
2601       8.9829  float_math_helper
160        0.5526  main
```

- If you have debug symbols (-g) you could use:

```
% sudo opannotate --source
--output-dir=/path/to/annotated-source /path/to/mybinary
```

Usage (2)

- Use `opreport` by itself for a whole system view
- You can also reset and stop the profiling

```
% sudo opcontrol --reset
Signalling daemon... done
% sudo opcontrol --stop
Stopping profiling.
```

- You need to use reset if you reinstalled a newer version of oprofile

Introduction

- Intrumentation-based
- System-wide
- Meant to be used on production systems
  - Typical instrumentation can have a slowdown of 100x (Valgrind)
- No overhead when not in use and doesn't crash (strict usage)

Operation

- DTrace dynamically rewrites code by placing a branch to your instrumentation code
- Uninstrumented run as if nothing changed
- The main use is at function entry or exit points
- You can also instrument kernel functions, locking, instrument based on other events
- There's actions for sampling as well

Example

```
syscall::read:entry {
    self->t = timestamp;
}

syscall::read:return
/self->t/ {
    printf("%d/%d spent %d nsecs in read\n"
           pid, tid, timestamp - self->t);
}
```

- `t` is a thread-local variable
- This code prints how long each call to `read` takes, along with context
- Very limited to what you can put in, i.e no loops
  - This is how DTrace guarantees no infinite loops

Other Tools

- AMD CodeAnalyst did not compile, hopefully be availabe on the server
- WAIT
  - IBM's tool tells you what operations your JVM is waiting on while idle
  - Non-free and not available
- Not limited to code, or C/C++
  - Google's Page Speed Tool
  - Python `cProfile`
  - ...

Assignment 3

- New goal: Friday
- There's the midterm anyways
- If we have to, we can make Assignment 4 shorter

Future Lectures

- OpenMPI
- OpenCL
- Hadoop MapReduce
- Software Transactional Memory

- Got a suggestion to have a live scoreboard for Assignment 3
- I'll try to piece this together
- If there's anything else you want to see in this course, let me know

Friday's Lecture

- Midterm day
- The lecture will be extended office hours as well
- You may come here to study / ask questions
- Also at 1:30PM