Benchmarking and Tuning
ACI-REF Workshop
Brett Zimmerman
University of Oklahoma – IT/OSCER
int power(int x, int y) {
    int result;
    if (y < 0) {
        result = 0;
    } else {
        for (result = 1; y; y--)
            result *= x;
    }
    return result;
}
Don't

KEEP IN MIND THAT I'M SELF-TAUGHT, SO MY CODE MAY BE A LITTLE MESSY. LEMME SEE—I'M SURE IT'S FINE.

...WOW. THIS IS LIKE BEING IN A HOUSE BUILT BY A CHILD USING NOTHING BUT A HATCHET AND A PICTURE OF A HOUSE.

IT'S LIKE A SALAD RECIPE WRITTEN BY A CORPORATE LAWYER USING A PHONE AUTOCorrect THAT ONLY KNEW EXCEL FORMULAS.

IT'S LIKE SOMEONE TOOK A TRANSCRIPT OF A COUPLE ARGUING AT IKEA AND MADE RANDOM EDITS UNTIL IT COMPILED WITHOUT ERRORS. OKAY, I'LL READ A STYLE GUIDE.

- Randall Munroe – http://xkcd.com/1513/
Complementary Processes

Systems

Processes

Measurement

Adaptation
Resource Utilization

- CPU
- Memory (cache, RAM; space, bandwidth)
- Disk (intentional as well as swap and paging)
- Network (bandwidth, latency)

A bottleneck always exists.
Benchmarking and Tuning

- Systems
  - Measure the performance characteristics of the system
  - Adjust the system to accommodate a given use
    - Hardware adaptation
    - System tunables

- Processes
  - Measure the resource usage of a given piece of code
  - Adjust the code to make efficient use of the system
Units

- SI vs. IEC units
  - KiB $2^{10}$ (1024)  
  - MiB $2^{20}$ (1048576)  
  - GiB $2^{30}$ (1073741824)  
  - TiB $2^{40}$ (1099511627776)
  - KB $10^3$ (1000)  
  - MB $10^6$ (1000000)  
  - GB $10^9$ (1000000000)  
  - TB $10^{12}$ (1000000000000)
Some Groundwork – Queuing

- $L = \lambda W$

  Units of work within the system is equal to the product of the arrival rate of the units of work and the time the unit spends in the system
Broad Optimization Approaches

- Reducing Visit Counts
  - Amortize fixed overhead by aggregating operations
  - Not doing stuff
- Reducing Wait time
  - Reducing Overhead
  - Using multiple queues
  - Space tradeoffs/Caching
  - Efficiency
- Overriding concerns
  - The first priority for tuning is stability
Optimization Overview

- Outside measurement of resources
- Code review
  - Algorithm
  - Implementation
- Compiler
- 80/20 Rule
- Time/space tradeoffs
- Effect of the Storage Hierarchy
The Storage Hierarchy

- ~2 ns - L1
- ~5 ns - L2
- ~14 ns - L3
- ~60 ns - RAM
- ~3 ms - Disk
Determining System Performance

- Published specifications
  - Processor
  - Bus
  - Memory

- Known Benchmarks
  - Network (ping, netperf, qperf, OSU Micro)
  - IO (lozone, Fio)
  - Global (HPL)
Coarse Timing Data

- /usr/bin/time rather than shell builtin
Measuring System Usage

- **Crontab**
  - */10 * * * * /usr/lib64/sa/sa1 1 1 -S XALL

- **Sysstat package**

- Reports paging, IO usage, per block device usage, interrupt counts, power management, network utilization, cpu utilization, run-queue length, memory utilization, swap space, inode and dentry cache, switching

- e.g. `sar -n EDEV` will display per-device network errors for each network device on the system
System Tunables

- /proc
  - net
  - vm
  - kernel
  - fs
- proc.sys.net.ipv4.neigh.default.gc_thresH2
- Documentation
  - Kernel-doc package
  - /usr/share/doc/kernel-version/Documentation/sysctl
Itrace -Sfc example

<table>
<thead>
<tr>
<th>% time</th>
<th>seconds</th>
<th>usecs/call</th>
<th>calls</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>50.03</td>
<td>2.159365</td>
<td>2159365</td>
<td>1</td>
<td>__libc_start_main</td>
</tr>
<tr>
<td>46.43</td>
<td>2.004238</td>
<td>35162</td>
<td>57</td>
<td>strcpy</td>
</tr>
<tr>
<td>0.63</td>
<td>0.027280</td>
<td>69</td>
<td>393</td>
<td>strlen</td>
</tr>
<tr>
<td>0.43</td>
<td>0.018509</td>
<td>18509</td>
<td>1</td>
<td>getaddrinfo</td>
</tr>
<tr>
<td>0.26</td>
<td>0.011290</td>
<td>68</td>
<td>166</td>
<td>malloc</td>
</tr>
<tr>
<td>0.25</td>
<td>0.010619</td>
<td>68</td>
<td>156</td>
<td>free</td>
</tr>
<tr>
<td>0.20</td>
<td>0.008802</td>
<td>68</td>
<td>128</td>
<td>realloc</td>
</tr>
<tr>
<td>0.20</td>
<td>0.008774</td>
<td>165</td>
<td>53</td>
<td>SYS_open</td>
</tr>
<tr>
<td>0.20</td>
<td>0.008771</td>
<td>67</td>
<td>129</td>
<td>ferror</td>
</tr>
<tr>
<td>0.20</td>
<td>0.008723</td>
<td>69</td>
<td>126</td>
<td>fgets</td>
</tr>
<tr>
<td>0.19</td>
<td>0.008099</td>
<td>8099</td>
<td>1</td>
<td>open</td>
</tr>
<tr>
<td>0.09</td>
<td>0.003855</td>
<td>68</td>
<td>56</td>
<td>memcpy</td>
</tr>
<tr>
<td>0.09</td>
<td>0.003750</td>
<td>1250</td>
<td>3</td>
<td>fclose</td>
</tr>
<tr>
<td>0.09</td>
<td>0.003712</td>
<td>100</td>
<td>37</td>
<td>SYS_close</td>
</tr>
<tr>
<td>0.06</td>
<td>0.002697</td>
<td>2697</td>
<td>1</td>
<td>connect</td>
</tr>
<tr>
<td>0.06</td>
<td>0.002650</td>
<td>662</td>
<td>4</td>
<td>SYS_connect</td>
</tr>
</tbody>
</table>
gprof

- Compile with profiling
- Run code
  - Normal output
  - Side effect: profiling data written to gmon.out
- Report profiling data
  - gprof --line --flat-profile area-serial gmon.out

<table>
<thead>
<tr>
<th>% cumulative</th>
<th>self</th>
<th>self</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>seconds</td>
<td>seconds</td>
<td>calls</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>200000</td>
</tr>
<tr>
<td>(area_under_curve.c:139 @ 400b30)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>(area_under_curve.c:55 @ 4008a8)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>(area_under_curve.c:108 @ 400a48)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Perf stat for Array walk

```plaintext
perf stat -Bd array_inner

Performance counter stats for 'array_inner':

<table>
<thead>
<tr>
<th>Counter</th>
<th>Value</th>
<th>Rate</th>
<th>% Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>task-clock</td>
<td>4175.666892</td>
<td>1.000 CPUs</td>
<td>100.00%</td>
</tr>
<tr>
<td>context-switches</td>
<td>7</td>
<td>0.002 K/sec</td>
<td>0.20%</td>
</tr>
<tr>
<td>cpu-migrations</td>
<td>7</td>
<td>0.002 K/sec</td>
<td>0.20%</td>
</tr>
<tr>
<td>page-faults</td>
<td>1,872</td>
<td>0.448 K/sec</td>
<td>0.20%</td>
</tr>
<tr>
<td>cycles</td>
<td>11,577,144,187</td>
<td>2.773 GHz</td>
<td>89.99%</td>
</tr>
<tr>
<td>stalled-cycles/frontend</td>
<td>3,943,947,255</td>
<td>34.07%</td>
<td>89.99%</td>
</tr>
<tr>
<td>stalled-cycles/backend</td>
<td>1,445,711,602</td>
<td>12.49%</td>
<td>79.98%</td>
</tr>
<tr>
<td>instructions</td>
<td>19,770,949,724</td>
<td>1.71 insns per cycle</td>
<td>89.99%</td>
</tr>
<tr>
<td>branches</td>
<td>4,361,497,083</td>
<td>1044.503 M/sec</td>
<td>89.99%</td>
</tr>
<tr>
<td>branch-misses</td>
<td>8,346,798</td>
<td>0.19% of all branches</td>
<td>90.01%</td>
</tr>
<tr>
<td>L1-dcache-loads</td>
<td>3,844,013,042</td>
<td>920.575 M/sec</td>
<td>90.01%</td>
</tr>
<tr>
<td>L1-dcache-load-misses</td>
<td>81,717,861</td>
<td>2.13% of all L1-dcache hits</td>
<td>90.01%</td>
</tr>
<tr>
<td>LLC-loads</td>
<td>3,871,347</td>
<td>0.927 M/sec</td>
<td>90.01%</td>
</tr>
<tr>
<td>LLC-load-misses</td>
<td>2,255,846</td>
<td>58.27% of all LL-cache hits</td>
<td>90.00%</td>
</tr>
</tbody>
</table>

4.176491482 seconds time elapsed
```
```markdown
**perf stat** -Bd array_outer

**Performance counter stats for 'array_outer':**

<table>
<thead>
<tr>
<th>Counter</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>13440.776165 task-clock</td>
<td>1.000 CPUs utilized</td>
<td></td>
</tr>
<tr>
<td>18 context-switches</td>
<td>0.001 K/sec</td>
<td></td>
</tr>
<tr>
<td>5 cpu-migrations</td>
<td>0.000 K/sec</td>
<td></td>
</tr>
<tr>
<td>1,872 page-faults</td>
<td>0.139 K/sec</td>
<td></td>
</tr>
<tr>
<td>37,238,405,421 cycles</td>
<td>2.771 GHz</td>
<td>[89.99%]</td>
</tr>
<tr>
<td>29,860,712,853 stalled-cycles-frontend</td>
<td>80.19% frontend cycles idle</td>
<td>[90.00%]</td>
</tr>
<tr>
<td>23,981,968,848 stalled-cycles-backend</td>
<td>64.40% backend cycles idle</td>
<td>[80.00%]</td>
</tr>
<tr>
<td>19,791,761,293 instructions</td>
<td>0.53 insns per cycle</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.51 stalled cycles per insn</td>
<td>[90.00%]</td>
</tr>
<tr>
<td>4,369,050,335 branches</td>
<td>325.059 M/sec</td>
<td>[90.00%]</td>
</tr>
<tr>
<td>8,363,924 branch-misses</td>
<td>0.19% of all branches</td>
<td>[90.00%]</td>
</tr>
<tr>
<td>3,851,175,888 L1-dcache-loads</td>
<td>286.529 M/sec</td>
<td>[90.00%]</td>
</tr>
<tr>
<td>563,864,367 L1-dcache-load-misses</td>
<td>14.64% of all L1-dcache hits</td>
<td>[90.00%]</td>
</tr>
<tr>
<td>18,290,787 LLC-loads</td>
<td>1.361 M/sec</td>
<td>[90.00%]</td>
</tr>
<tr>
<td>16,310,806 LLC-load-misses</td>
<td>89.17% of all LL-cache hits</td>
<td>[90.00%]</td>
</tr>
</tbody>
</table>

13.442679796 seconds time elapsed
```
Perf Counters

- perf package
- perf list – List available counters and tracepoints
- perf top – System level profiling
  - /proc/sys/kernel/perf_event_paranoid – Allow unprivileged users to collect performance counter data
    - 1 not paranoid
- perf stat – Run a command and gather stats
SystemTap

- Allows fine-detail monitoring of the kernel
- `systemtap`, `systemtap-runtime`, `kernel-debuginfo`, `kernel-debuginfo-common-arch`, `kernel-devel`
- `stap -v -e 'probe vfs.read {printf("read\n");exit()}'`
- Systemtap automates adding instrumentation modules to the running kernel
- `stapdev` – privileged stap users. Effective root
- `Stapusr` – can use `staprun` to run modules in `/lib/modules/version/system-tap/`
global wevent, revent

probe vfs.write.return {
    wevent[execname()] += $return
}

probe vfs.read.return {
    revent[execname()] += $return
}

probe timer.s(10) {
    printf("___READS___\n")
    foreach(exe in revent - limit 10)
        printf("%15s: %d bytes\n", exe, revent[exe])
    printf("\n")
    printf("___WRITES___\n")
    foreach(exe in wevent - limit 10)
        printf("%15s: %d bytes\n", exe, wevent[exe])
}

___READS___

dd: 335304334 bytes
irqbalance: 288890 bytes
crond: 57767 bytes
sshd: 52016 bytes
date: 13990 bytes
sshd: 11952 bytes
unix_chkpwd: 11934 bytes
sa1: 11495 bytes
screen: 5963 bytes
systemd-journal: 5112 bytes

___WRITES___

dd: 335300191 bytes
sshd: 12205 bytes
sshd: 11228 bytes
screen: 9077 bytes
stapio: 3558 bytes
systemd-logind: 2102 bytes
ping: 1831 bytes
auditd: 1783 bytes
systemd: 472 bytes
gdbus: 392 bytes