Code Detective

How to investigate Linux Performance Issues

Gabriel Krisman Bertazi
krisman@collabora.co.uk
Transistor count and complexity

The curve shows transistor count doubling every two years.

Date of introduction

Transistor count

Image: © Wgsimon. Licensed under CC-BY-SA 3.0 unported.
Major processor selling point

Product: Intel Core i7-4790K Processor (8M Cache, up to 4.40 GHz) BX80646I74790K
Price: $350.93 & FREE Shipping
You Save: $29.06 (8%)

Only 1 left in stock - order soon.
Sold by SavingCost and Fulfilled by Amazon. Gift-wrap available.

Style: Processor Only

Processor + Cooler | Processor + Liquid Cooler | Processor Only | Processor with SSD Bundle
Major processor selling point

- Intel Core i7-4790K Processor (8M Cache, up to 4.40 GHz) BX80646I74790K
  - List Price: $579.99
  - Price: $350.93 & FREE Shipping
  - You Save: $29.06 (8%)

- Intel Core™ i7-6950X Processor Extreme Edition
  - 25M Cache, up to 3.50 GHz
Major processor selling point

Intel Core™ i7-6950X Processor Extreme Edition

POWER8 Highlights

Announced 2013 at Hot Chips Conference
12 Cores per Socket/Chip
Significantly Strengthened Cores
8 threads per core (SMT8)
Wider fetch / dispatch/issue of instructions (8 fetch / dispatch, 10 issue)
Doubled highly utilized execution units

Larger Caches:
64K D Cache, 32K L1 Cache, 512K private L2, 8M L3 / Core (96M)
Compiler Optimizations

- Modify the code at compile-time to improve performance
- Configurable levels of aggressiveness in optimization
- LLVM implements way over 50 optimization passes
- Based on Heuristics – Best effort scenario
- Optimizations create more room for more optimizations
- Theoretically Infinite optimization search space
When performance (really) matters

• When is maximum performance needed?
When performance (really) matters

- When is maximum performance needed?
- When it doesn’t matter?

https://commons.wikimedia.org/wiki/File:Optimizing-different-parts.svg
When performance (really) matters

- When is maximum performance needed?
- When it doesn’t matter?
- When it won’t make a difference?

https://commons.wikimedia.org/wiki/File:Optimizing-different-parts.svg
When performance (really) matters

- When is maximum performance needed?
- When it doesn’t matter?
- When it won’t make a difference?
- Hot Code

https://commons.wikimedia.org/wiki/File:Optimizing-different-parts.svg
When performance (really) matters

- When is maximum performance needed?
- When it doesn’t matter?
- When it won’t make a difference?
- Hot Code
- Amdahl's Law

Two independent parts

Original process

Make B 5x faster

Make A 2x faster

https://commons.wikimedia.org/wiki/File:Optimizing-different-parts.svg
Premature Optimization

• When to start optimizing?
  – Don’t put the cart before the horse
• Dangers of premature optimization
• Maybe it won’t make a difference?
• Write good algorithms first
• Code Maintenance x Readability x Oversmart code
Investigating performance issues (When things go wrong)

- Innocent looking code can hide major issues
- Collect information: Profiling
- Symptoms x root cause
- Measurement can affect results
- HW and SW Performance counters
- Perf_events
### perf_events

**Samples:** 1K of event 'cycles:ppp', Event count (approx.): 57059081274592

<table>
<thead>
<tr>
<th>Overhead</th>
<th>Shared Object</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.19%</td>
<td>[kernel]</td>
<td>[k] perf_event_task_tick</td>
</tr>
<tr>
<td>9.22%</td>
<td>[kernel]</td>
<td>[k] i915_gem_object_get_pages_gtt</td>
</tr>
<tr>
<td>8.99%</td>
<td>libpthread-2.24.so</td>
<td>[.] pthread_cond_signal@@GLIBC_2.3.2</td>
</tr>
<tr>
<td>8.99%</td>
<td>firefox-esr</td>
<td>[.] 0x0000000000010859</td>
</tr>
<tr>
<td>7.10%</td>
<td>[kernel]</td>
<td>[k] scheduler_tick</td>
</tr>
<tr>
<td>6.88%</td>
<td>[kernel]</td>
<td>[k] load_balance</td>
</tr>
<tr>
<td>6.82%</td>
<td>[kernel]</td>
<td>[k] pci_azx_read</td>
</tr>
<tr>
<td>6.02%</td>
<td>libxul.so</td>
<td>[.] pthread_cond_broadcast@plt</td>
</tr>
<tr>
<td>5.27%</td>
<td>[kernel]</td>
<td>[k] update_rq_clock.part.82</td>
</tr>
<tr>
<td>4.04%</td>
<td>[kernel]</td>
<td>[k] resched_curr</td>
</tr>
<tr>
<td>4.04%</td>
<td>libglib-2.0.so.0.5000.3</td>
<td>[.] 0x0000000000047a16</td>
</tr>
<tr>
<td>4.04%</td>
<td>libpthread-2.24.so</td>
<td>[.] __GI___libc_read</td>
</tr>
<tr>
<td>3.09%</td>
<td>firefox-esr</td>
<td>[.] 0x0000000000014d6d</td>
</tr>
<tr>
<td>2.70%</td>
<td>[kernel]</td>
<td>[k] get_futex_key_refs.isra.11</td>
</tr>
<tr>
<td>2.37%</td>
<td>[kernel]</td>
<td>[k] generic_permission</td>
</tr>
<tr>
<td>2.07%</td>
<td>Xorg</td>
<td>[.] DamageReportDamage</td>
</tr>
<tr>
<td>1.33%</td>
<td>[kernel]</td>
<td>[k] strlcpy</td>
</tr>
</tbody>
</table>
Example of Performance Problem:
Dealing with Cache Misses
Dealing with Cache Misses - Problem

```
int plane_6(img[m][n])
{
    for (i = 0; i < w; i ++) {
        for (j = 0; j < h; j ++) {
            img[j][i] &= 0x20;
        }
    }
}
```
Dealing with Cache Misses - Problem

```c
int plane_6(img[m][n])
{
    for (i = 0; i < w; i++) {
        for (j = 0; j < h; j++) {
            img[j][i] &= 0x20;
        }
    }
}
```
Dealing with Cache Misses - Analysis

[krisman@dilma examples]$ sudo perf stat -B \
> -e cache-references,cache-misses,cycles,instructions,branches ./plane-6

Performance counter stats for './plane-6':

    2,854,082,869   cache-references
    2,403,635,896   cache-misses       # 84.217 % of all cache refs
    61,301,995,110  cycles             #
    26,336,518,447  instructions      # 0.43 insn per cycle
    1,218,576,320   branches

    23.439412393 seconds time elapsed
Dealing with Cache Misses - Solution

```c
int plane_6(img[m][n])
{
    for (i = 0; i < w; i ++) {
        for (j = 0; j < h; j ++) {
            img[i][j] &= 0x20;
        }
    }
}
```
Dealing with Cache Misses - Analysis

[krisman@dilma examples]$ sudo perf stat -B -e cache-references,cache-misses,cycles,instructions,branches ./plane-6-opt

Performance counter stats for './plane-6-opt':

2,217,115 cache-references
747,290 cache-misses  # 33.706 % of all cache refs
11,418,750,995 cycles
26,460,541,744 instructions  # 2.32 insn per cycle
1,262,542,792 branches

3.878753109 seconds time elapsed
Example of Performance Problem:
Forcing branch misses
Forcing Branch Mispredictions

```c
long rand_partsum(int *vec, int n) {
    int i, k;
    long sum = 0;

    // Part 1 – Initialize with random values
    for (i = 0; i < n; i++)
        vec[i] = rand() % n;

    // Part 2 – Sum high elements
    for (k = 0; k < 1000000; k++)
        for (i = 0; i < n; i++)
            if (vec[i] > n/2)
                sum += vec[i];

    return sum;
}
```


Gabriel Krisman Bertazi  22 - Code Detective
Forcing Branch Mispredictions - Analysis

[krisman@dilma branch-miss]$ sudo perf stat ./part-sum
104074572925232

Performance counter stats for './part-sum':

<table>
<thead>
<tr>
<th>Counter</th>
<th>Value</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>task-clock (msec)</td>
<td>32506.113304</td>
<td>0.998 CPUs utilized</td>
</tr>
<tr>
<td>context-switches</td>
<td>126</td>
<td>0.004 K/sec</td>
</tr>
<tr>
<td>cpu-migrations</td>
<td>3</td>
<td>0.000 K/sec</td>
</tr>
<tr>
<td>page-faults</td>
<td>54</td>
<td>0.002 K/sec</td>
</tr>
<tr>
<td>cycles</td>
<td>95,291,641,720</td>
<td>2.931 GHz</td>
</tr>
<tr>
<td>instructions</td>
<td>105,055,182,871</td>
<td>1.11 insn per cycle</td>
</tr>
<tr>
<td>branches</td>
<td>10,318,677,682</td>
<td>317.438 M/sec</td>
</tr>
<tr>
<td>branch-misses</td>
<td>1,675,446,879</td>
<td>16.24% of all branches</td>
</tr>
</tbody>
</table>

32.570957258 seconds time elapsed
Branch Predictor

Two possible destinations
Which instruction to load next?

Fetch instruction unit
Decode Instruction
Execute
MEM
Writeback

99d: add $0x5,%rax
9a1: cmpb $0x0,($r14,%r13,1)
9a6: jne 940 <fw@plt+0x90>
9a8: jmp 972 <fw@plt+0xc2>
9aa: movsbl (%r14,%r13,1),%edi
9af: mov 0x2016ca(%rip),%rsi
9b6: callq 830 <_IO_putc@plt>
Branch Predictor

---

**Flowchart:**

1. **Fetch instruction unit**
2. **Decode Instruction**
3. **Execute**
4. **MEM**
5. **Writeback**

---

**Code Snippet:**

```
99d: add $0x5,%rax
9a1: cmpb $0x0,(%r14,%r13,1)
9a6: jne 940 <fw@plt+0x90>
9a8: jmp 972 <fw@plt+0xc2>
9aa: movsbl (%r14,%r13,1),%edi
9af: mov 0x2016ca(%rip),%rsi
9b6: callq 830 <_IO_putc@plt>
```

---

**Question:**

- More elements of vec?
- X > median?
- Sum = Sum + X

---

**Notes:**

- Two possible destinations
- Which instruction to load next?
Branch Predictor

- Stop and Wait?
- Try to guess

Fetch instruction unit → Decode Instruction → Execute → MEM → Writeback

99d: add $0x5,%rax
9a1: cmpb $0x0,(%r14,%r13,1)
9a6: jne 940 <fw@plt+0x90>
9a8: jmp 972 <fw@plt+0xc2>
9aa: movsbl (%r14,%r13,1),%edi
9af: mov 0x2016ca(%rip),%rsi
9b6: callq 830 <_IO_putchar@plt>

Two possible destinations
Which instruction to load next?

X > median?
Yes
No

Sum = Sum + X

More elements of vec?

...
Forcing Branch Mispredictions - Analysis 2

- Perf record
- Perf annotate
- Debug Symbols

```
0.02 | 76: movl $0x0,-0x14(%rbp)
     ↓ jmp c5
    if (vec[i] > n/2)
1.49 | 7f: mov -0x14(%rbp),%eax
     cltq
2.69 | lea 0x0(,%rax,4),%rdx
4.24 | mov -0x30(%rbp),%rax
4.64 | add %rdx,%rax
23.23 | mov (%rax),%edx
0.08 | mov -0x24(%rbp),%eax
1.10 | mov %eax,%ecx
0.22 | shr $0x1f,%ecx
1.62 | add %ecx,%eax
0.01 | sar %eax
0.47 | cmp %eax,%edx
0.43 | jle c1
   sum += vec[i];

7.82 | mov -0x14(%rbp),%eax
5.70 | cltq
0.81 | lea 0x0(,%rax,4),%rdx
0.27 | mov -0x30(%rbp),%rax
0.93 | add %rdx,%rax
10.09 | mov (%rax),%eax
1.11 | cltq
6.53 | add %rax,-0x20(%rbp)
   for (i = 0; i < n; i++)
15.41 c1: addl $0x1,-0x14(%rbp)
3.25 c5: mov -0x14(%rbp),%eax
0.01 cmp -0x24(%rbp),%eax
```
Forcing Branch Mispredictions - Results

[krisman@dilma branch-miss]$ sudo perf stat ./part-sum-sorted
103505069265552

Performance counter stats for './part-sum-sorted':

16247.683571 task-clock (msec) # 0.998 CPUs utilized
246 context-switches # 0.015 K/sec
8 cpu-migrations # 0.000 K/sec
59 page-faults # 0.004 K/sec
47,615,524,420 cycles # 2.931 GHz
105,282,365,954 instructions # 2.21 insn per cycle
10,225,033,988 branches # 629.323 M/sec
2,203,394 branch-misses # 0.02% of all branches

16.275325653 seconds time elapsed
Forcing Branch Mispredictions - Opt

- Adding a new step reduces the execution time!
- Sorting is not trivial
- 50% reduction in execution time

---

Initialize vec with random elements

Sort the vector

for each element X of vec

X > median?

Yes

Sum = Sum + X

No

More elements of vec?

Yes

No

End
How easily can we disrupt performance?

- Producing wrong data without doing anything obviously wrong – University of Colorado
- Changing the linking order of libraries
- Changing the size of the shell environment
- Changing the order and sequence of compiler optimizations
- Forcing a function to be inlined
- Adding a single NOP instruction to the code
The Memory Layout

- Snapshot of the program loaded to memory
- Any addition or removal of data modifies whatever comes next in memory
- Affects cache hit/miss, branches hit/miss, page faults, code alignment, etc.
- Adding a single NOP can change the layout completely
- Brainfuck interpreter benchmark on SandyBridge:

<table>
<thead>
<tr>
<th></th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Without</strong> single NOP before code section</td>
<td>0m5.293s</td>
</tr>
<tr>
<td><strong>With</strong> single NOP before code section</td>
<td>0m2.275s</td>
</tr>
</tbody>
</table>
Take away

- Optimize by hand only when required. Not before.
- Write good algorithms first
- Increased system complexity means increased challenges
- Compilers and binary optimizers are always improving
- Take time to learn how to use performance analysis tools...
- ... But they will only take you so far...
- A solid understanding of the system remains fundamental
Thank you!

Gabriel Krisman Bertazi
krisman@collabora.co.uk