Measuring the Performance Impact of Branching Instructions

SSP 2021, Leipzig

Lukas Beierlieb, Lukas Iffländer, Aleksandar Milenkoski, Thomas Prantl, Samuel Kounev

09.11.2021

https://se.informatik.uni-wuerzburg.de
Motivation: Performance is important!

- Case example: Stress testing Microsoft's Hyper-V hypervisor
- Repeated addition and removal of a virtual processor

- Unproblematic at lower execution speed
- Crash likelihood increases with execution speed
Motivation: Branches are critical

- Related work: How can the cost of branch executions be reduced [1, 2, 3]

- Here: We can avoid branches – what is the performance gain?
Background: Hyper-V

- Type I-hypervisor
- Virtual machines are called partitions
- Microkernel-architecture: outsourcing of functionality into a privileged virtual machine (the „root partition“)

- Partitions can request hypervisor services using hypercalls
Background: Hypercalls

- Examples:
  - HvCreatePartition
  - HvCreateVp (virtual processor)
  - HvNotifyLongSpinWait

![Hypercall diagram]

- Input Parameter Header:
  - 0: PartitionId (8 bytes)
  - 8: VpIndex (4 bytes)
    - TargetVtl (1 byte)
    - RsvdZ (3 bytes)
Background: Hypercall Injector

- Windows kernel module (proposed at SSP 2019 [4])
- Can inject arbitrary hypercalls into Hyper-V

- Logging for these values is possible:
  - Result value
  - Output values
  - Timesteps
  - Execution time

- Logging is optional
  - Memory constraints
  - Execution overhead
Background: Branching

- Processors try to parallelize instruction execution as much as possible
- Pipelining:
  - Parallel execution (superscalar architecture)
  - Out-of-order execution
- Branches: Next instruction unclear, speculative execution
Research Question

- Goal is highest throughput for stress testing
- No values will be logged

**What is the cost of checking if something should be logged, if nothing is logged?**

- Certainly branchless will be faster
- Branches always take the same branch, so by how much?
Execution Loop with Branches

```java
// prepare
while (/* more to execute */) {
    switch (/* type */) {
        case TYPE_WAIT:
            // sleep and maybe log times, details left out
            break;
        case TYPE_CALL:
            // prepare memory for call
            if (/* timesteps or execution time requested */) // take start time
                // issue hypercall
            if (/* timesteps or execution time requested */) // take end time
            if (/* timesteps requested */) // store time stamps
                if (/* execution time requested */) // calculate execution time and store
                if (/* result value requested */) // store result value
                    if (/* output values requested */) // store output memory page
                }
    }
```
 Execution Loop without branches

```c
// prepare
while(/* more to execute */) {
  switch(/* type */) {
    case TYPE_WAIT:
      // sleep and maybe log times, details left out
      break;
    case TYPE_CALL:
      // prepare memory for call

      // issue hypercall
  }
}
```
Choosing correct loop beforehand based on request log values

```c
if (flags_received->memory && !flags_received->execetime && !flags_received->timestamps && !flags_received->result && !flags_received->output)
    status = perform_memory(input_file, output_file);
else if (flags_received->memory && !flags_received->execetime && !flags_received->timestamps && flags_received->result && !flags_received->output)
    status = perform_memory_log_result(input_file, output_file);
else if (flags_received->memory && !flags_received->execetime && !flags_received->timestamps && !flags_received->result && flags_received->output)
    status = perform_memory_log_output(input_file, output_file);
else if (flags_received->memory && !flags_received->execetime && !flags_received->timestamps && flags_received->result && flags_received->output)
    status = perform_memory_log_result_output(input_file, output_file);
else if (flags_received->memory && flags_received->execetime && !flags_received->timestamps && !flags_received->result && !flags_received->output)
    status = perform_memory_log_execetime(input_file, output_file);
else if (flags_received->memory && flags_received->execetime && !flags_received->timestamps && flags_received->result && !flags_received->output)
    status = perform_memory_log_execetime_result(input_file, output_file);
```

Around 40 implementations of the essentially the same code...
Measurement Methodology

- Test campaign: 50 million invalid hypercall
- Execution time is measured; can be used to calculate throughput
- Execution of 35 consecutive runs

<table>
<thead>
<tr>
<th>Lenovo Thinkpad P1</th>
<th>Hyper-V</th>
<th>Windows 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>50 million calls</td>
<td>50 million calls</td>
</tr>
</tbody>
</table>

35 times
Results

![Graph showing execution times for different configurations over iterations.](image)

**Configuration**
- branch_debug
- branch_release
- nobranch_debug
- nobranch_release

**Axes:**
- x-axis: iteration
- y-axis: execution time [s]
Further Measurements

![Graph showing the performance impact of branching instructions with different configurations over iterations. The x-axis represents iteration, and the y-axis represents execution time in seconds. Five lines represent different configurations: branch_debug, branch_release, nobranch_debug, and nobranch_release.](image)

*Configuration*
- branch_debug
- branch_release
- nobranch_debug
- nobranch_release

*Measuring the Performance Impact of Branching Instructions*

*Lukas Beierlieb*
Additional Data
Summary

- Testing tools require high performance for stress testing
- Idea: try to minimize overhead by reducing branching instructions (used for logging)
- Effects are significant

Compromise:
- Use dedicated branchless implementation for high throughput configurations (no logging, only execution times)
- Use branch-based implementation to cover all other cases
- Performance gains where required
- Small penalty for maintainability
References

[1] Hwu, W. et al.: Comparing software and hardware schemes for reducing the cost of branches


The End

Thank you for your attention!