Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Descartes
research

Measuring the Performance Impact of Branching Instructions

SSP 2021, Leipzig

Lukas Beierlieb, **Lukas Iffländer**, Aleksandar Milenkoski,

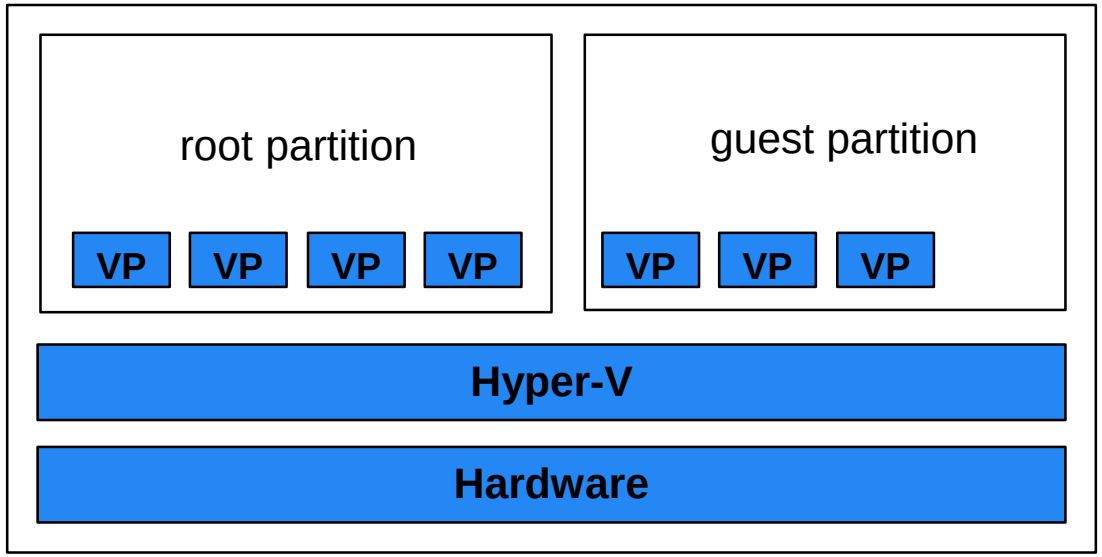
Thomas Prantl, Samuel Kounev

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

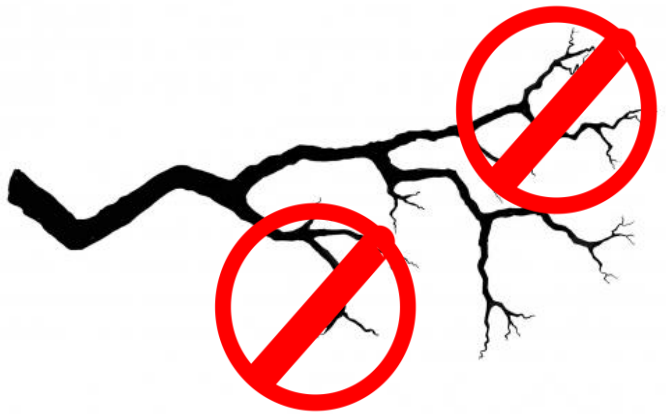


- Unproblematic at lower execution speed
- Crash likelihood increases with execution speed

Motivation: Branches are critical

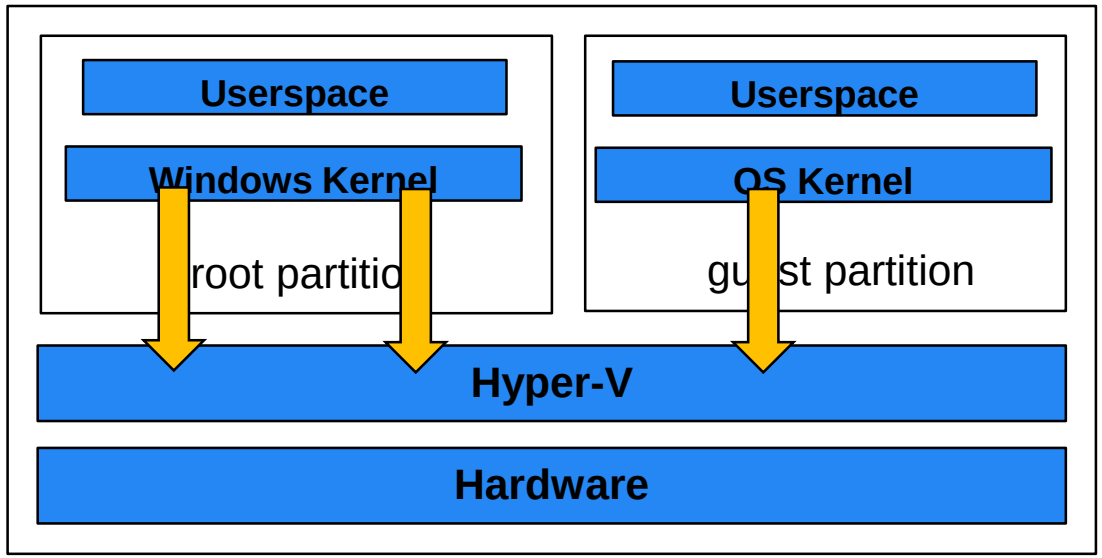
- Related work: How can the cost of branch executions be reduced [1, 2, 3]

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



Background: Hyper-V

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

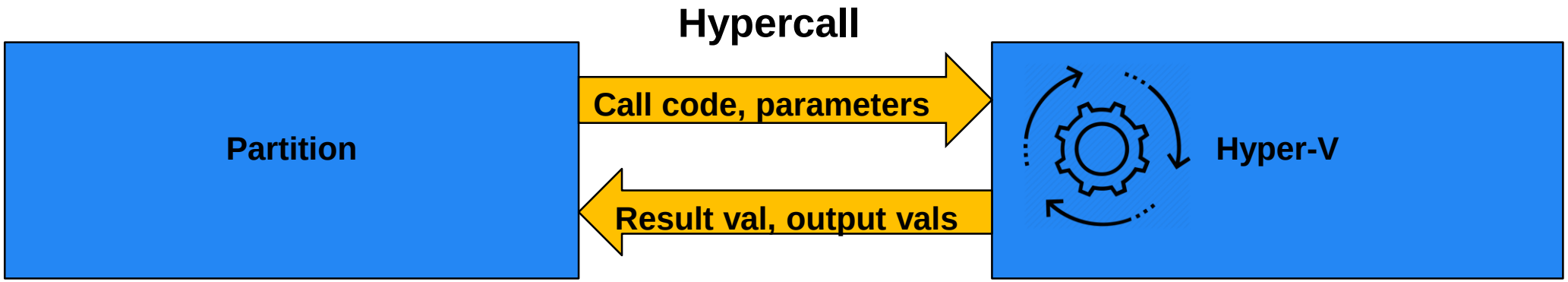


- Partitions can request hypervisor services using hypercalls

Background: Hypercalls

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

Input Parameter Header			
0	PartitionId (8 bytes)		
8	VpIndex (4 bytes)	TargetVtl (1 byte)	RsvdZ (3 bytes)

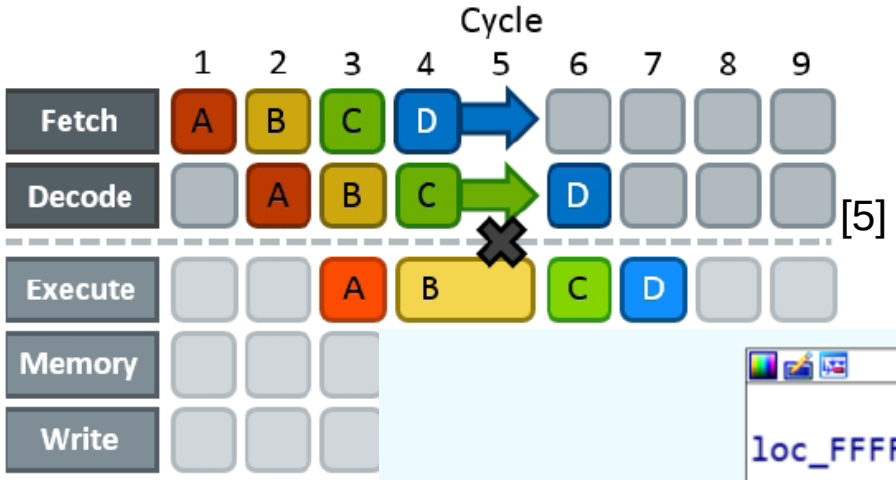


Background: Hypercall Injector

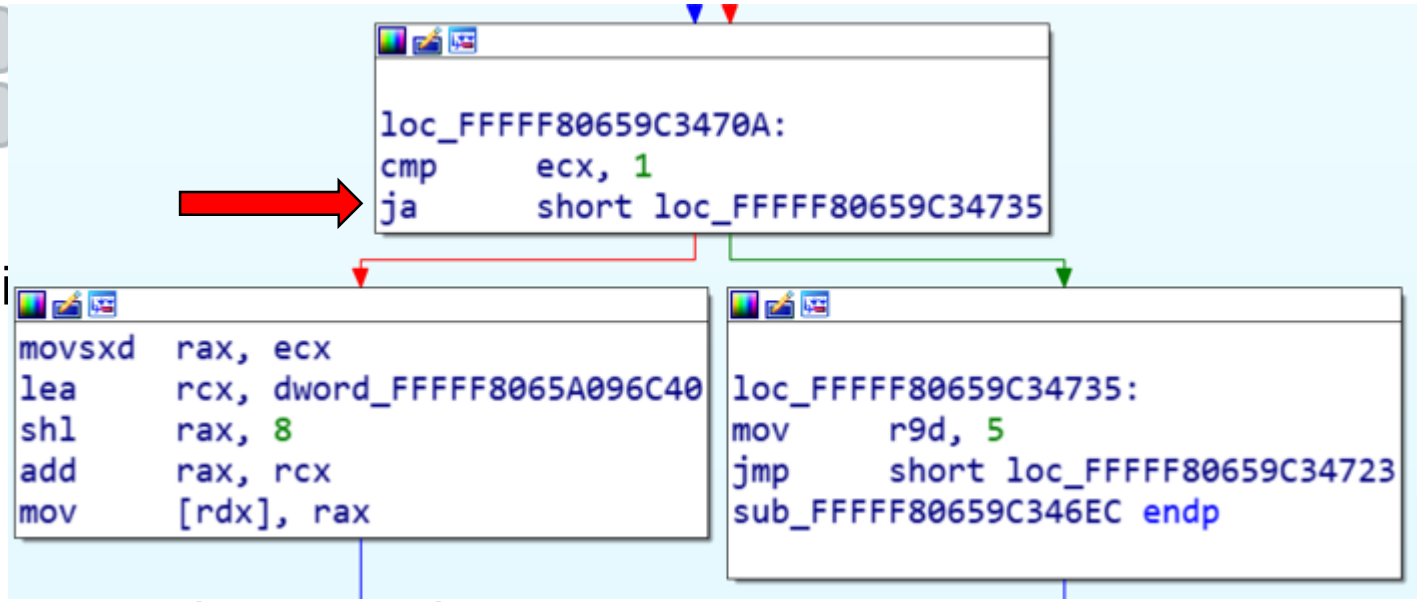
- Windows kernel module (proposed at SSP 2019 [4])
- Can inject arbitrary hypercalls into Hyper-V
- Logging for these values is possible:
  - Result value
  - Output values
  - Timesteps
  - Execution time
- Logging is optional
  - Memory constraints
  - Execution overhead

Background: Branching

- Processors try to parallelize instruction execution as much as possible
- Pipelining:



[5]



- Parallel execution (superscalar archi
- 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

```
1  // prepare
2  while (/* more to execute */) {
3      switch (/* type */) {
4      case TYPE_WAIT:
5          // sleep and maybe log times, details left out
6          break;
7      case TYPE_CALL:
8          // prepare memory for call
9          if (/* timesteps or execution time requested */)
10             // take start time
11         // issue hypercall
12         if (/* timesteps or execution time requested */)
13             // take end time
14         if (/* timesteps requested */)
15             // store time stamps
16         if (/* execution time requested */)
17             // calculate execution time and store
18         if (/* result value requested */)
19             // store result value
20         if (/* output values requested /*)
21             // store output memory page
22     }
23 }
```

Execution Loop without branches

```
1  // prepare
2  while (/* more to execute */) {
3      switch (/* type */) {
4      case TYPE_WAIT:
5          // sleep and maybe log times, details left out
6          break;
7      case TYPE_CALL:
8          // prepare memory for call
9
10
11         // issue hypercall
12
13
14
15
16
17
18
19
20
21
22     }
23 }
```

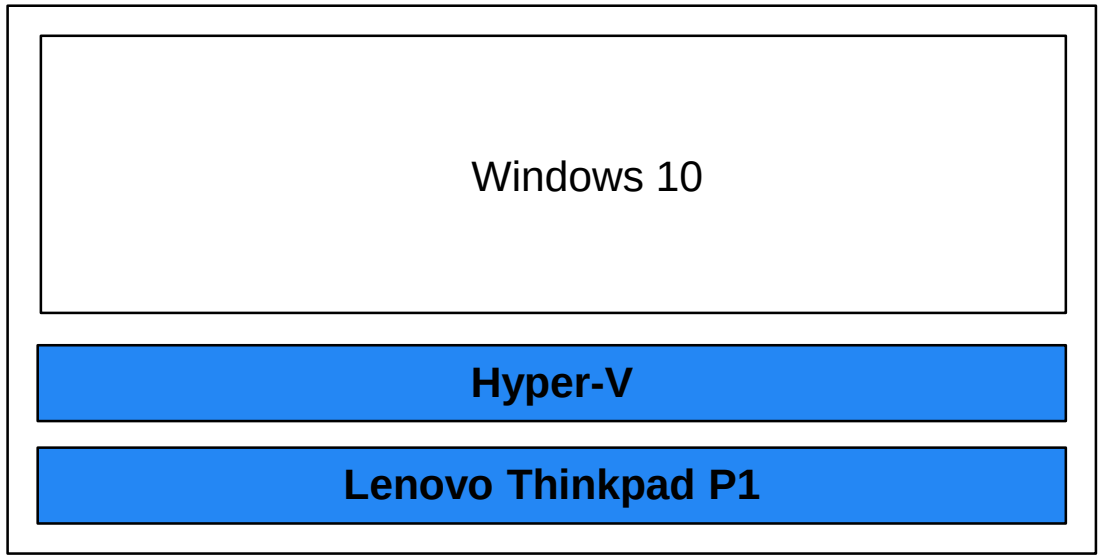
The Disadvantage…

- Choosing correct loop beforehand based on request log values

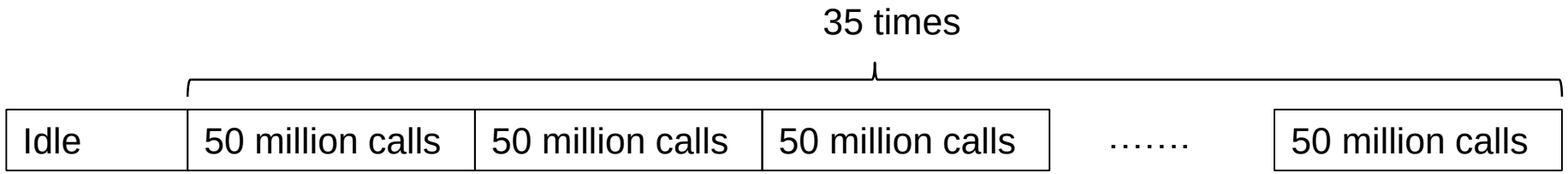
```
if (flags_received->memory && !flags_received->exectime && !flags_received->timestamps && !flags_received->result && !flags_received->output)
        status = perform_memory(input_file, output_file);
else if (flags_received->memory && !flags_received->exectime && !flags_received->timestamps && flags_received->result && !flags_received->output)
        status = perform_memory_log_result(input_file, output_file);
else if (flags_received->memory && !flags_received->exectime && !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->exectime && !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->exectime && !flags_received->timestamps && !flags_received->result && !flags_received->output)
        status = perform_memory_log_exectime(input_file, output_file);
else if (flags_received->memory && flags_received->exectime && !flags_received->timestamps && flags_received->result && !flags_received->output)
        status = perform_memory_log_exectime_result(input_file, output_file);
```

- Around 40 implementations of the essentially the same code…

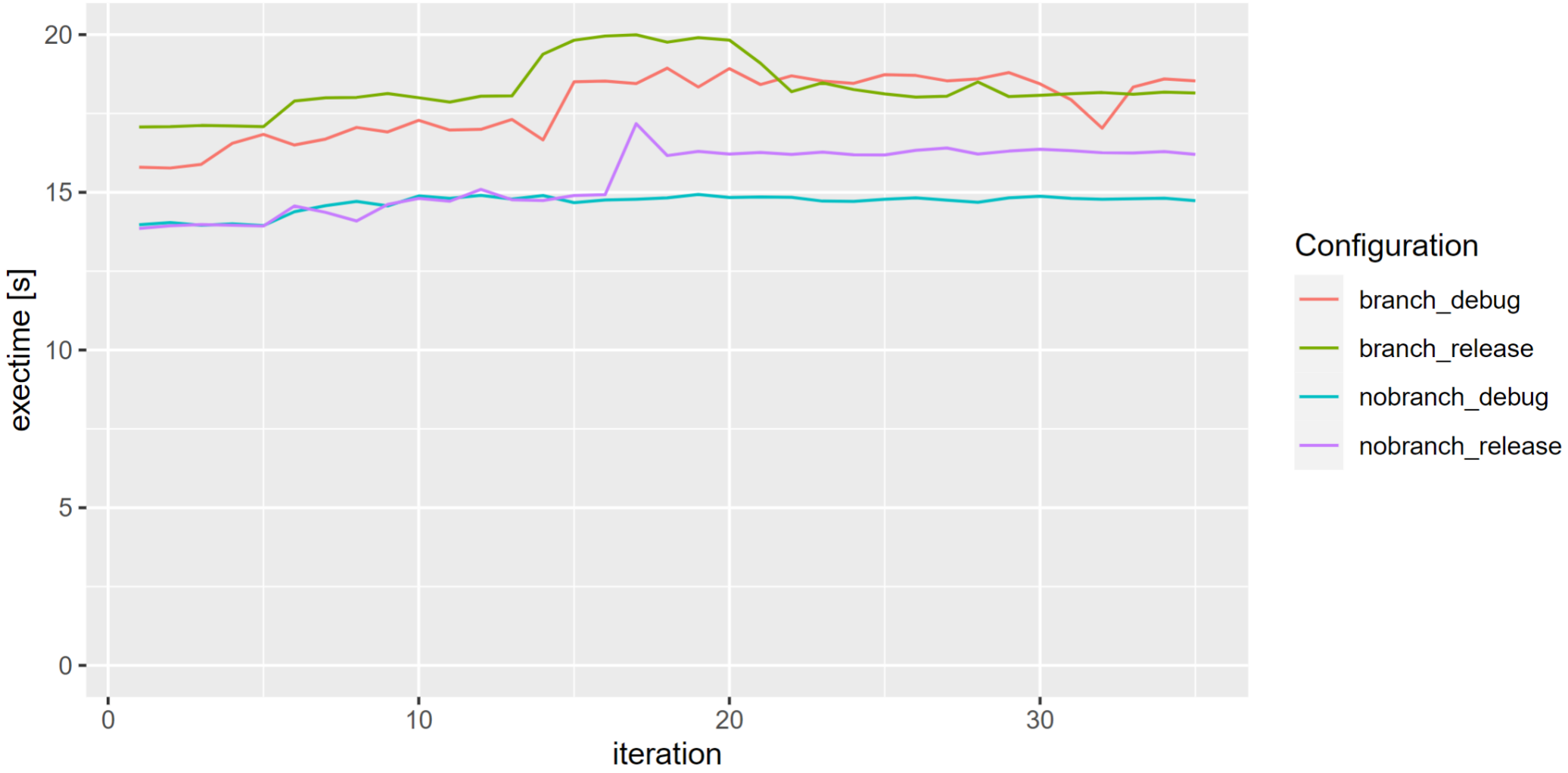
Measurement Methodology



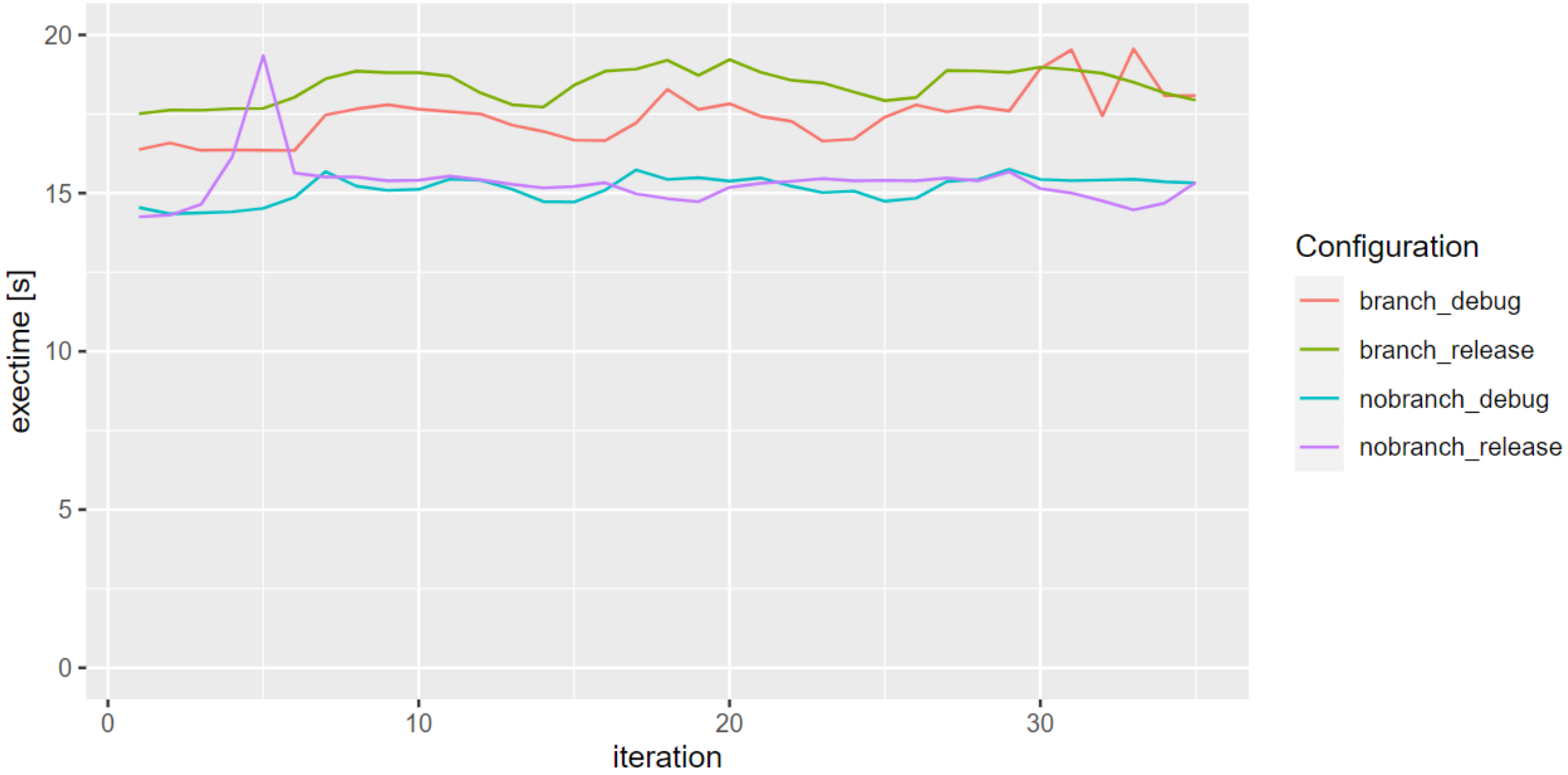
- Test campaign: 50 million invalid hypercall
- Execution time is measured; can be used to calculate throughput
- Execution of 35 consecutive runs



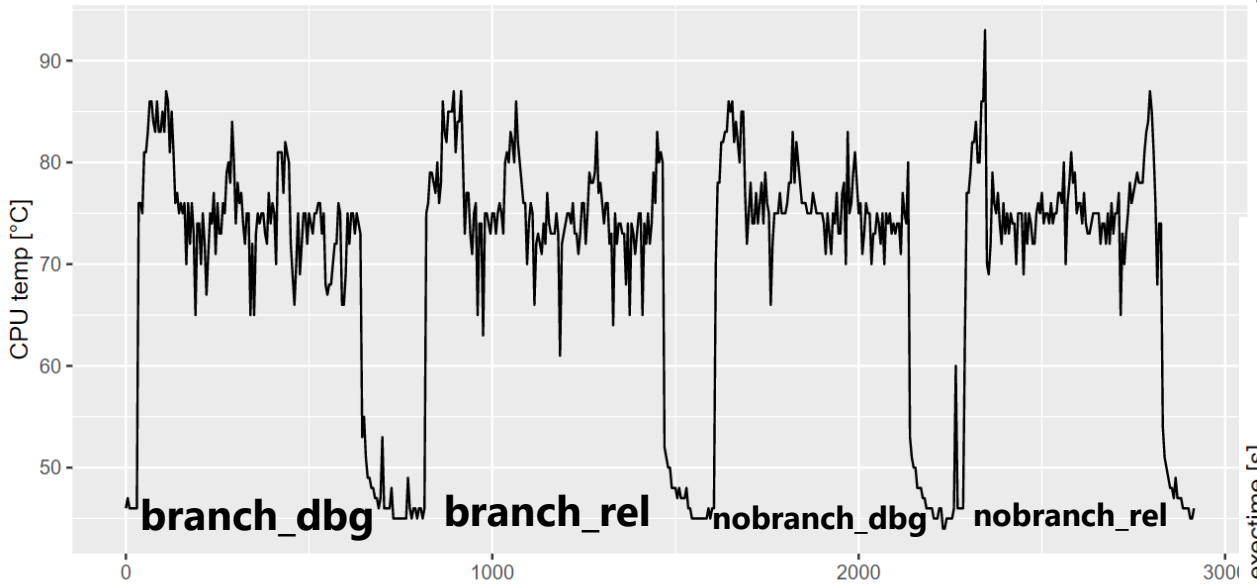
Results

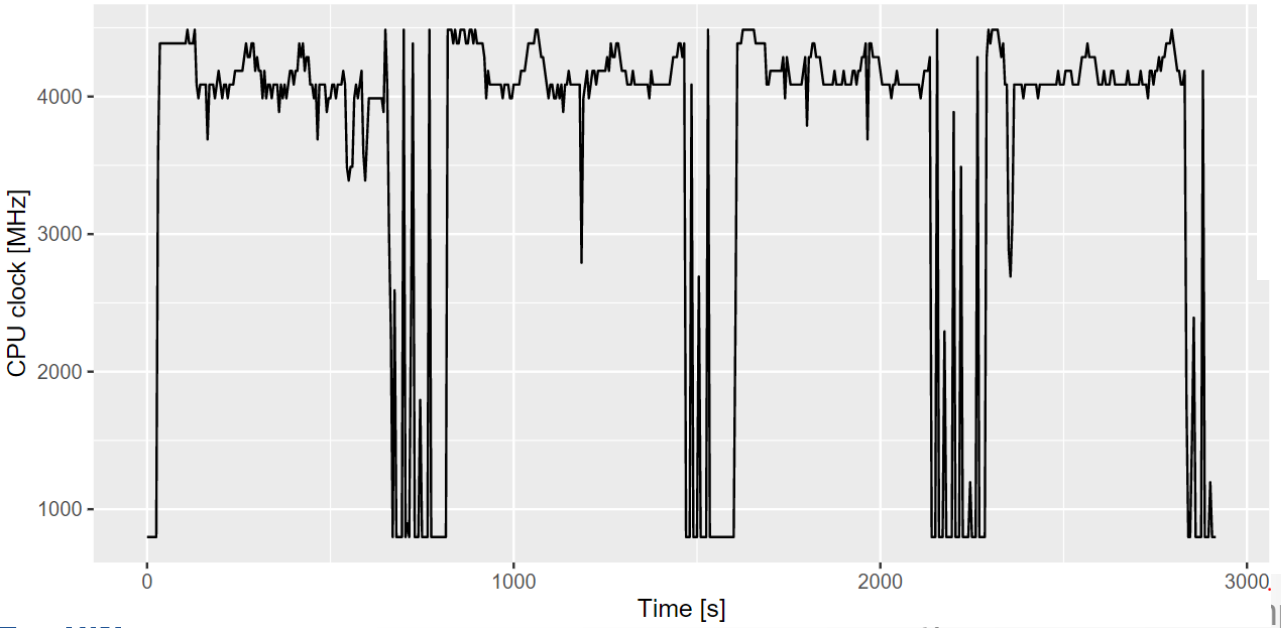

20
15
10
5
0
exectime [s]
0
10
20
30
iteration
Configuration
branch_debug
branch_release
nobranch_debug
nobranch_release

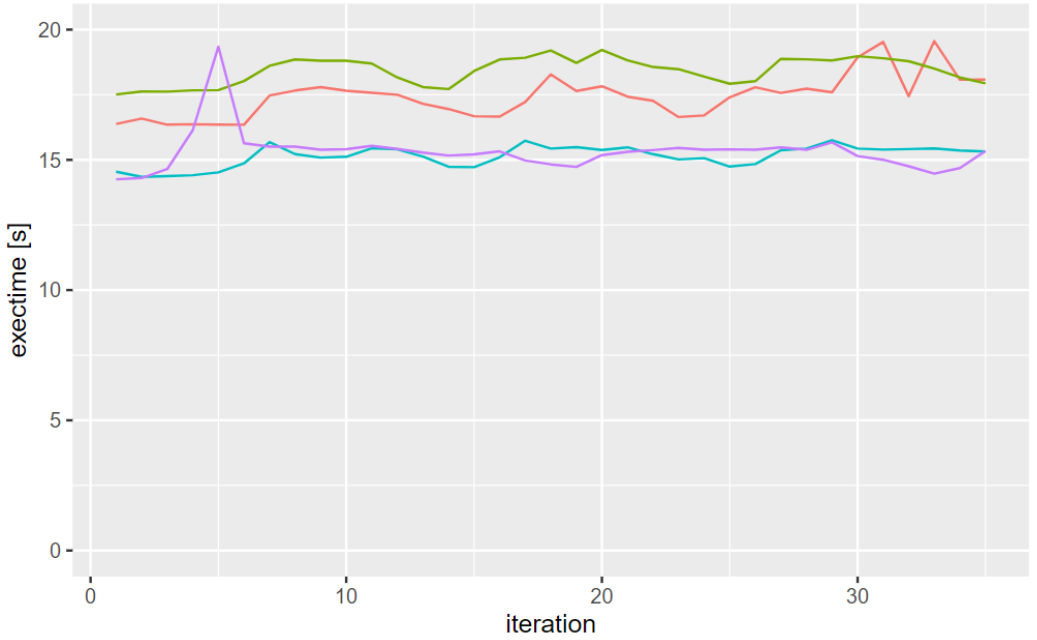
Further Measurements



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

Microsoft Hyper-v

References

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

[2] McFarling, S. et al.: Reducing the cost of branches

[3] Kim, H. et al.: Vpc prediction: reducing thecost of indirect branches via hardware-based dynamic devirtualization

[4] Beierlieb, L. et al.: Towards Testing the Performance Influence of Hypervisor Hypercall Interface Behavior

[5] https://www.intel.com/content/www/us/en/developer/tools/oneapi/tech-articles-how-to/overview.html#gs.fw8np8

The End

Thank you for your attention!