

Reliable Benchmarking: Requirements and Solutions

Dirk Beyer, Stefan Löwe, and **Philipp Wendler**

2023-04-22 @ RRRR'23



Philipp Wendler



Benchmarking is Important

- ▶ Evaluation of new approaches
- ▶ Evaluation of tools
- ▶ Competitions
- ▶ Tool development (testing, optimizations)

Reliable, reproducible, and accurate results needed!

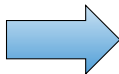
Benchmarking is Hard

- ▶ Influence of I/O
- ▶ Networking
- ▶ Distributed tools
- ▶ User input

Benchmarking is Hard

- ▶ Influence of I/O
- ▶ Networking
- ▶ Distributed tools
- ▶ User input

Not relevant
for many tools
(solver, verifiers, ...)



Easy?

Benchmarking is Hard

- ▶ Influence of I/O
- ▶ Networking
- ▶ Distributed tools
- ▶ User input

Not relevant
for many tools
(solver, verifiers, ...)

- ▶ Different hardware architectures
- ▶ Heterogeneity of tools
- ▶ Parallel benchmarks

Relevant!

Goals

- ▶ Reproducibility
 - ▶ Avoid non-deterministic effects and interferences
 - ▶ Provide defined set of resources
- ▶ Accurate results
- ▶ For solvers, verification tools, etc.
- ▶ On Linux

Checklist

1. Measure and Limit Resources Accurately
 - ▶ Time
 - ▶ Memory
2. Terminate Processes Reliably
3. Assign Cores Deliberately
4. Respect Non-Uniform Memory Access
5. Avoid Swapping
6. Isolate Individual Runs
 - ▶ Communication
 - ▶ File system

Measure and Limit Resources Accurately

- ▶ Wall time and CPU time
- ▶ Define memory consumption
 - ▶ Size of address space? Too large
 - ▶ Size of heap? Too low
 - ▶ Size of resident set (RSS)?
- ▶ Measure peak consumption (without sampling)
- ▶ Always define memory limit for reproducibility
- ▶ Include sub-processes

Measuring CPU time with “time”

~\$ time verifier

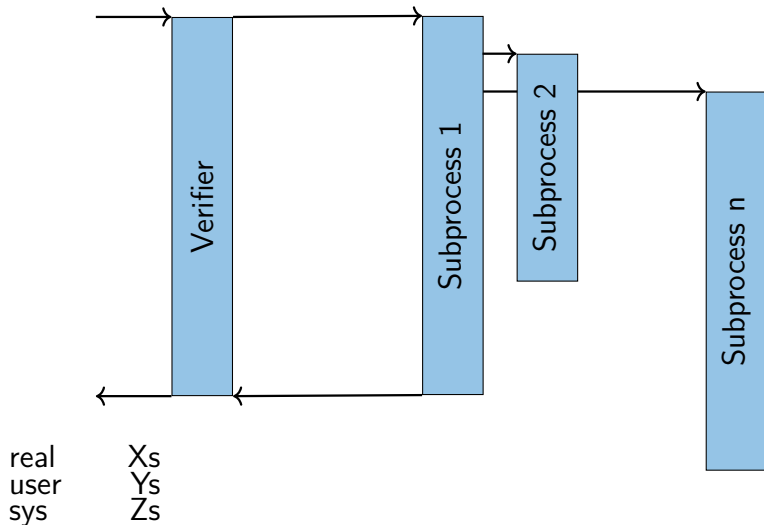


real
user
sys

X_s
 Y_s
 Z_s

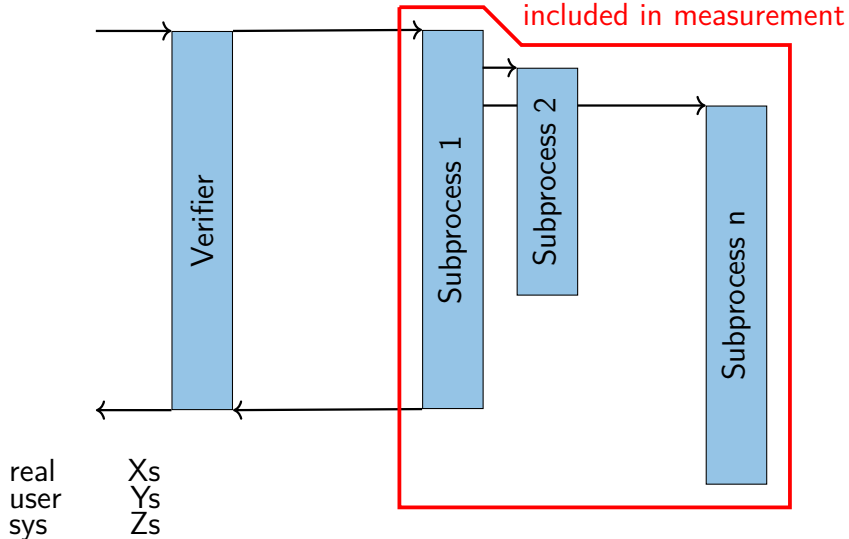
Measuring CPU time with “time”

~\$ time verifier



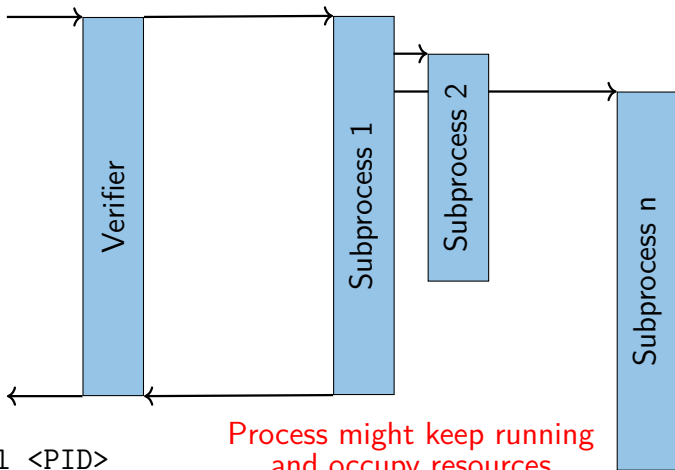
Measuring CPU time with "time"

~\$ time verifier



Terminate Processes Reliably

~\$ verifier



~\$ kill <PID>

Process might keep running
and occupy resources

Isolate Individual Runs

- ▶ Excerpt of start script taken from some verifier in SV-COMP:

```
# ... (tool started here)
```

```
killall z3 2> /dev/null
```

```
killall minisat 2> /dev/null
```

```
killall yices 2> /dev/null
```

- ▶ Thanks for thinking of cleanup



Isolate Individual Runs

- ▶ Excerpt of start script taken from some verifier in SV-COMP:

```
# ... (tool started here)
```

```
killall z3 2> /dev/null
```

```
killall minisat 2> /dev/null
```

```
killall yices 2> /dev/null
```

- ▶ Thanks for thinking of cleanup
- ▶ But what if there are parallel runs?



Isolate Individual Runs

- ▶ Temp files with constant names like `/tmp/mytool.tmp` collide
- ▶ State stored in places like `~/.mytool` hinders reproducibility
 - ▶ Sometimes even auto-generated
- ▶ Restrict changes to file system as far as possible

Cgroups

- ▶ Linux kernel “control groups”
- ▶ Reliable tracking of spawned processes
- ▶ Resource limits and measurements per cgroup
 - ▶ CPU time
 - ▶ Memory
 - ▶ I/O etc.

Only solution on Linux
for race-free handling of multiple processes!

Namespaces

- ▶ Light-weight virtualization
- ▶ Only one kernel running, no additional layers
- ▶ Change how processes see the system
- ▶ Identifiers like PIDs, paths, etc. can have different meanings in each namespace
 - ▶ PID 42 can be a different process in each namespace
 - ▶ Directory / can be a different directory in each namespace
 - ▶ ...
- ▶ Can be used to build application containers without possibility to escape
- ▶ Usable without root access

Benchmarking Containers

- ▶ Encapsulate groups of processes
- ▶ Limited resources (memory, cores)
- ▶ Total resource consumption measurable
- ▶ All other processes hidden and no communication with them
- ▶ Disabled network access
- ▶ Adjusted file-system layout
 - ▶ Private `/tmp`
 - ▶ Writes redirected to temporary RAM disk

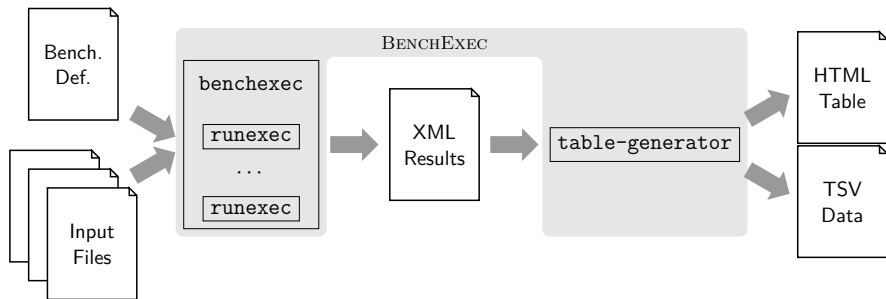
BenchExec

- ▶ A Framework for Reliable Benchmarking and Resource Measurement
- ▶ Provides benchmarking containers based on cgroups and namespaces
- ▶ Allocates hardware resources appropriately
- ▶ Low system requirements (modern Linux kernel and cgroups access)

BenchExec

- ▶ Open source: Apache 2.0 License
- ▶ Written in Python 3
- ▶ <https://github.com/sosy-lab/benchexec>
- ▶ Used in International Competition on Software Verification (SV-COMP) and by StarExec
- ▶ Originally developed for software verification, but applicable to arbitrary tools

BenchExec Architecture



`runexec`

Benchmarks a single run of a tool (in container)

`benchexec`

Benchmarks multiple runs

`table-generator`

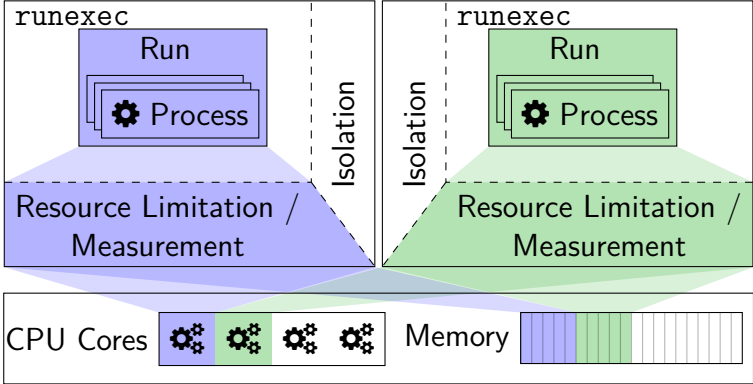
Generates TSV and interactive HTML tables

BenchExec: runexec

- ▶ Benchmarks a single run of a tool
- ▶ Measures and limits resources using cgroups
- ▶ Runnable as stand-alone tool and as Python module
- ▶ Easy integration into other benchmarking frameworks and infrastructure
- ▶ Example:

```
runexec --timelimit 100s --memlimit 16GB  
        --cores 0-7,16-23 --memoryNodes 0  
        -- <TOOL_CMD>
```

BenchExec: runexec



BenchExec: `benchexec`

- ▶ Benchmarks multiple runs
(e.g., a set of configurations against a set of files)
- ▶ Allocates hardware resources
- ▶ Can check whether tool result is as expected
for given input file and property

BenchExec: table-generator

- ▶ Aggregates results
- ▶ Extracts statistic values from tool output
- ▶ Generates TSV and interactive HTML tables (with plots)
- ▶ Computes result differences and regression counts

Please Read More

Dirk Beyer, Stefan Löwe, and Philipp Wendler.

Reliable Benchmarking: Requirements and Solutions. [1]

STTT 2019

- ▶ More details
- ▶ Study of hardware influence on benchmarking results
- ▶ Suggestions how to present results
(result aggregation, rounding, plots, etc.)

Conclusion

Be careful when benchmarking!

Don't use time, ulimit etc.
Always use cgroups and namespaces!

BenchExec
<https://github.com/sosy-lab/benchexec>

References I

- [1] Beyer, D., Löwe, S., Wendler, P.: Reliable benchmarking: Requirements and solutions. *Int. J. Softw. Tools Technol. Transfer* **21**(1), 1–29 (2019).
<https://doi.org/10.1007/s10009-017-0469-y>

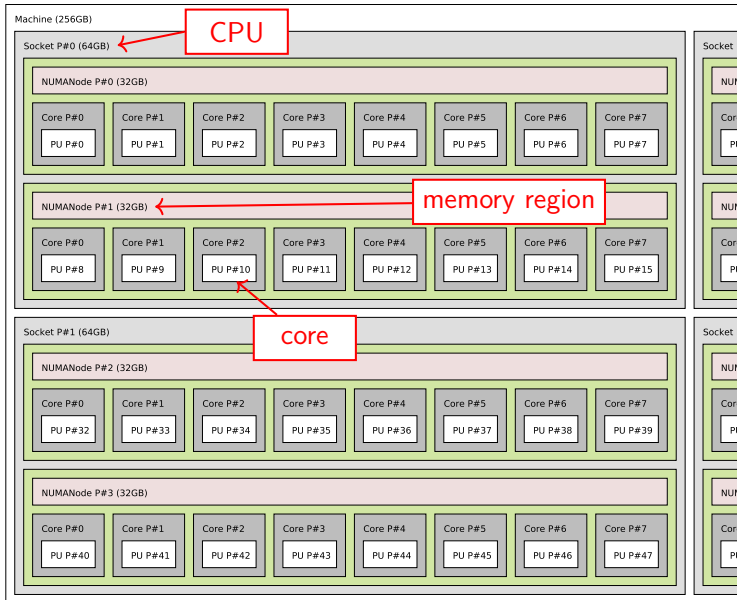
Assign Cores Deliberately

- ▶ Hyper Threading:
Multiple threads sharing execution units
- ▶ Shared caches

Respect Non-Uniform Memory Access (NUMA)

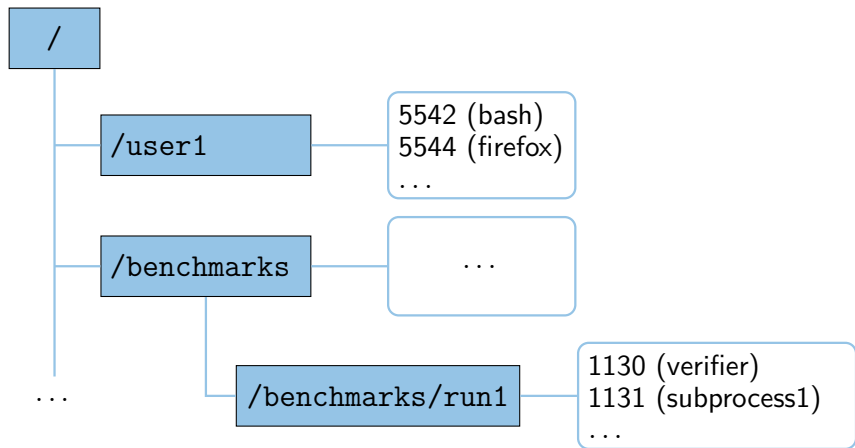
- ▶ Memory regions have different performance depending on current CPU core
- ▶ Hierarchical NUMA makes things worse

Type `lstopo` on your machine (Ubuntu: `package hwloc`)



Cgroups

- ▶ Hierarchical tree of sets of processes



BenchExec Configuration

- ▶ Tool command line
- ▶ Expected result
- ▶ Resource limits
 - ▶ CPU time, wall time
 - ▶ Memory
- ▶ Container setup
 - ▶ Network access
 - ▶ File-system layout
- ▶ Where to put result files

Directory Access Modes

	Read existing content	Write temp content	Write persistent content
hidden	X	✓	X
read only	✓	X	X
overlay	✓	✓	X
full access	✓	X	✓