A. Artifact Appendix

A.1 Abstract

We provide artifacts to measure key results to perform the trace-based simulation resulting in the numbers in Table 6. While we provide artifacts to reproduce all results as well as all our measurement and simulation results, only a subset of the experiments is described in this document and part of the artifact reproducibility evaluation.

A.2 Artifact check-list (meta-information)

- **Program: SPEC-CPU2017 (proprietary, not included)**
- **Compilation: gcc / g++**
- **Run-time environment: debian-based Linux, root access**
- **Hardware: Intel CPU x generation**
- **Execution: Sole user, process pinning**
- **Metrics: Power savings, performance impact**
- **Experiments: CPU Microbenchmarks and Simulation**
- **How much disk space required (approximately)?:** ≈ 200 GB
- **How much time is needed to prepare workflow (approximately)?:** 4 h
- **How much time is needed to complete experiments (approximately)?:** 24 h
- **Publicly available?: Yes**
- **Code licenses (if publicly available)?: GPLv3 License**
- **Data licenses (if publicly available)?: MIT License**
- **Archived?: 10.5281/zenodo.10479443**

A.3 Description

A.3.1 How to access

All code and data is available at 10.5281/zenodo.10479443. Due to license restrictions, the user has to provide an iso image of the SPEC CPU2017 benchmarks

A.3.2 Hardware dependencies

An Intel CPU where undervolting with the 0x150 MSR still works. Every CPU of the 9th generation or older should work if SGX is disabled in the BIOS. The following tool can be used to test if the CPU supports undervolting:
https://github.com/georgewhewell/undervolt.

A.3.3 Software dependencies

Debian-based Linux distribution with root permissions and the possibility to install kernel modules, we used Ubuntu 20.04.
SPEC CPU2017 iso image.

A.3.4 Data sets

We provide some data that takes too long to record for the artifact evaluation. However, all code and instructions to record this data are available.

A.4 Installation

Every experiment directory contains a Makefile to build the program required for the experiment. There are some build dependencies:

```
# apt install coreutils build-essential \
              libarchive-dev make \
              linux-tools-generic
```

A.5 Experiment workflow

The experiments consist of C programs, kernel modules and python scripts. Outputs of earlier experiments are inputs for later experiments.

We provide artifacts to verify the following claims. Chained together they result in similar numbers to the numbers of Table 6. Due to process variations every CPU is affected differently by undervolting, therefore, the results will not exactly, i.e., they depend on the specific variations.

C1 Voltage Change Delay

C2 Frequency Change Delay

C3 Efficiency and Performance Impact of Undervolting

C4 Final Simulation

Some experiments take hundreds of hours to run, while we publish all artifacts to run these experiments we also publish the results. In particular these are the gem5 simulations of the increased IMUL latency and the recording of the instruction traces of the SPEC CPU2017 benchmarks.

All experiments come with a `README.md` file containing additional information that does not fit this document.

A.6 Disclaimer

CPU undervolting can cause instabilities! This can break file systems and cause data loss or corruption. Never run this software on a system containing important data without backups. We are not responsible for any damage caused by this software. A -70mV voltage offset was stable on all CPUs we tested.

A.7 Evaluation and expected results

For the first three claims **C1** to **C3** we do not expect specific results. The simulation results of **C4** show that SUIT has an overall positive impact on CPU energy efficiency.

A.7.1 Voltage Change Delay

Measure the voltage change delay from Section 5.2.

Path: `5_microbenchmarks/1_voltage_change_delay`

Prepare: –

Run: `./run.sh`
The run script first builds the kernel module and user space application. It then loads the kernel module. If successful, the measurement is run, measuring 20 voltage offset

changes of $-70\,\mathrm{mV}$. Finally, it calls the plot script with the `result.csv` file to plot the data.

Plot: `python3 user/plot.py result.csv`
The plot script also prints the average time it takes to change the voltage to stdout. This time is required later to define the CPU for the simulation. To plot the data without invoking the measurement again start the plot script directly like shown.

Result: Time it takes to change the core voltage.

A.7.2 Frequency Change Delay

Measure the voltage change delay from Section 5.2.

Path: `5_microbenchmarks/3_freq_change_delay`

Prepare: Turn off hardware controlled p-states (HWP). To easily be able to change the core clock frequency from the operating system, the CPU must not use HWP. To disable it, boot the kernel with the `intel_pstate=passive` command line option.

Run: `./run.sh`
The run script first builds the kernel module and user space application. It then loads the kernel module. If successful, the measurement is run, measuring 20 frequency changes of $-500\,\mathrm{MHz}$ from $3\,\mathrm{GHz}$. Finally it calls the plot script with the `result.csv` file to plot the data.

Plot: `python3 user/plot.py result.csv`
The plot script also prints the average time it takes to change the frequency to stdout. This time is required later to define the CPU for the simulation. To plot the data without invoking the measurement again start the plot script directly like shown.

Result: Time it takes to change the frequency.

A.7.3 Efficiency and Performance Impact of Undervolting

Measure the efficiency and performance impact of undervolting from Section 5.4.

Path: `5.4_power_efficiency_and_performance`
The `README.md` in this directory contains additional details.

Prepare: There are multiple preparation steps:

Install SPEC CPU2017
Do not forget to `runcpu --update` to update the installation to the newest version. Not up-to-date versions may have complications when compiling.

Update the `SPEC_PATH` in the first line of `5.4_power_efficiency_and_performance/Makefile` to point to the SPEC CPU2017 installation directory.

Build with `make` and copy the built and axillary files into the SPEC CPU2017 installation directory with `make copy`. The later command requires root privileges to set the root sticky bit for the `measure` binary, because it must run as root but we do not want to start the SPEC CPU2017 benchmarks as root.

Test the SPEC CPU2017 installation with:
```
runcpu --config=power --size=test --define \
undervolting=0 specrate
```

Run: With the preparation done SPEC CPU2017 is run twice, once with the CPU at the default operating voltage and once with the CPU undervolted. The undervolt offset is defined in the `--define undervolting=X` command line argument when starting the benchmark.

Run with the default voltage:
```
runcpu --config=power --size=ref -n 2 \
--output_format config,csv --copies=$(nproc) \
--define undervolting=0 specrate
```

Run with a $-70\,\mathrm{mV}$ *offset:*
```
runcpu --config=power --size=ref -n 2 \
--output_format config,csv --copies=$(nproc) \
--define undervolting=-70 specrate
```

This runs all integer and floating point SPEC CPU2017 rate benchmarks. It will take several hours to finish.

Gather Results: The power measurement results are in `$SPEC_PATH/power_measurements` the benchmark scores in `$SPEC_PATH/results`. Combine them by copying them into one `results` directory:

```
mkdir -p results/spec
cp $SPEC_PATH/power_measurements/* results/
cp $SPEC_PATH/results/* results/spec
```

The files for the three CPUs from Table 2 are available at 10.5281/zenodo.10479443.

Plot: `./plot.py result`
This plots the changes in CPU frequency, voltage, power and benchmark scores. It additionally prints the exact changes in % to stdout.

Result: In the last printed table, the relevant columns are: `pct` showing the benchmark score increase and `eff` showing the efficiency increase at $-70\,\mathrm{mV}$. We expect the benchmark scores and efficiency to increase.

A.7.4 Final Simulation

Perform the final simulation from Section 6.

Path: `6.2_instruction_trace_based_evaluation`

Prepare:

CPU Configuration File
Copy `cpus/i9-9900K-70mV.csv` to `cpus/cpu_name-70mV.csv` as a starting point. The file is then configured with the results from the previous experiments. An exact description of all fields is in the `README.md` file.

Instruction Traces
Recording the instruction traces of all benchmarks with qemu takes a very long time. We provide the instruction traces for all benchmarks at 10.5281/zenodo.10479443 `instruction_traces.tar.gz` except for 520.omnetpp and 521.wrf because they are too large.

Run:
```
./simulate_all.py cpu_name-70mV \
30,15,3,14 voltfreq "" path/to/spec/traces/5*
```

The simulation is multithreaded and performs best on a CPU with 21 or more cores. It takes several hours to run.

Postprocess: `./postprocess.py results_XXX.json`
The simulation creates an output `.json` file containing the results of each benchmark. Running the post-process script prints the results formatted as a table to stdout and creates a .csv file with the same content.

Result: The last three columns of the results table are the most relevant, the row `geomean` contains the numbers of Table 6:

- `power perc`: change in power consumption in %: “Pwr”
- `perf perc`: change in performance in %: “Perf.”
- `eff perc`: change in efficiency in %: “Eff.”

We expect the efficiency change to be positive to show that SUIT is viable. Additionally, all three numbers should be similar to numbers of $\mathcal{A}_1$ with the fV operating strategy and column $\text{SPEC}_{\text{gmean}}$. The performance impact is allowed to be slightly negative.

Because we cannot provide the traces for 520.omnetpp and 521.wrf the resulting mean is slightly better than with the two benchmarks included. On our tested CPU $\mathcal{A}$ at $-70\,\text{mV}$, excluding the two benchmarks increases the efficiency gain from $5.7\,\%$ (see Table 6) to $6.1\,\%$.

A.7.5 Regenerate Table 6

We are now able to generate Table 6 from the paper with the additional results from this evaluation.

Path: `6.2_instruction_trace_based_evaluation/scripts`

Prepare: Update the file path at `generate_results_table.py:246` to point to the .json result file from the simulation.

Run: `./generate_results_table.py --full`

Result: `results_table.tex`
When compiled it shows Table 6 with an additional CPU $\mathcal{D}$ showing the results of this evaluation.