Thermal-aware Resource Management for Modern Computing Platforms in the Next Generation of Aircraft

Deliverable type	Demonstrator
Deliverable Name	Benchmark suite and evaluation techniques
Deliverable Number	D5.1
Work Package	WP5: Demonstrator and technology validation
Responsible Partner	ČVUT
Report Status	Final
Dissemination Level	Public
Version	1.0
Due Date of Deliverable	30/06/2020

	Signature	Date
WP Leader		
Project Coordinator		
Topic Leader		

This project has received funding from the Clean Sky 2 Joint Undertaking under the European Union's Horizon 2020 research and innovation programme under grant agreement No 832011.

Purpose & Scope

We describe a testbed built to perform thermal measurements of the i.MX8 system-on-chip (SoC). The testbed consists of the hardware components and software. We also describe our methodology used to process measured data and obtain reliable and relevant information about the executed workload. We validate all parts – hardware, software and methodology – on a set of benchmarks and present the obtained results.

Revision History

Version	Date	Author/Reviewer	Comments
0.1	2020-05-29	M. Sojka	Initial revision
0.2	2020-06-19	M. Sojka	Version for internal review
1.0 Final	2020-06-26	M. Sojka/O. Benedikt, C. Maia	Updates based on internal review

Authors

Name	Organization	Contact
Michal Sojka	ČVUT	michal.sojka@cvut.cz

Contents

1 Introduction

This deliverable describes the testbed that has been built in order to perform thermal measurements of the i.MX8 system-on-chip (SoC). i.MX8 was chosen as one of the project's demonstrators platforms. Still, the same or similar testbed can be used even for different hardware platforms. The testbed consists of the hardware part, described in Section 2 and software part (Section 3). In Section 4, we describe our methodology used to process measured data and get reliable and relevant information about the executed workload. We validate both hardware, software and methodology on a set of benchmarks and present results in Section 5.

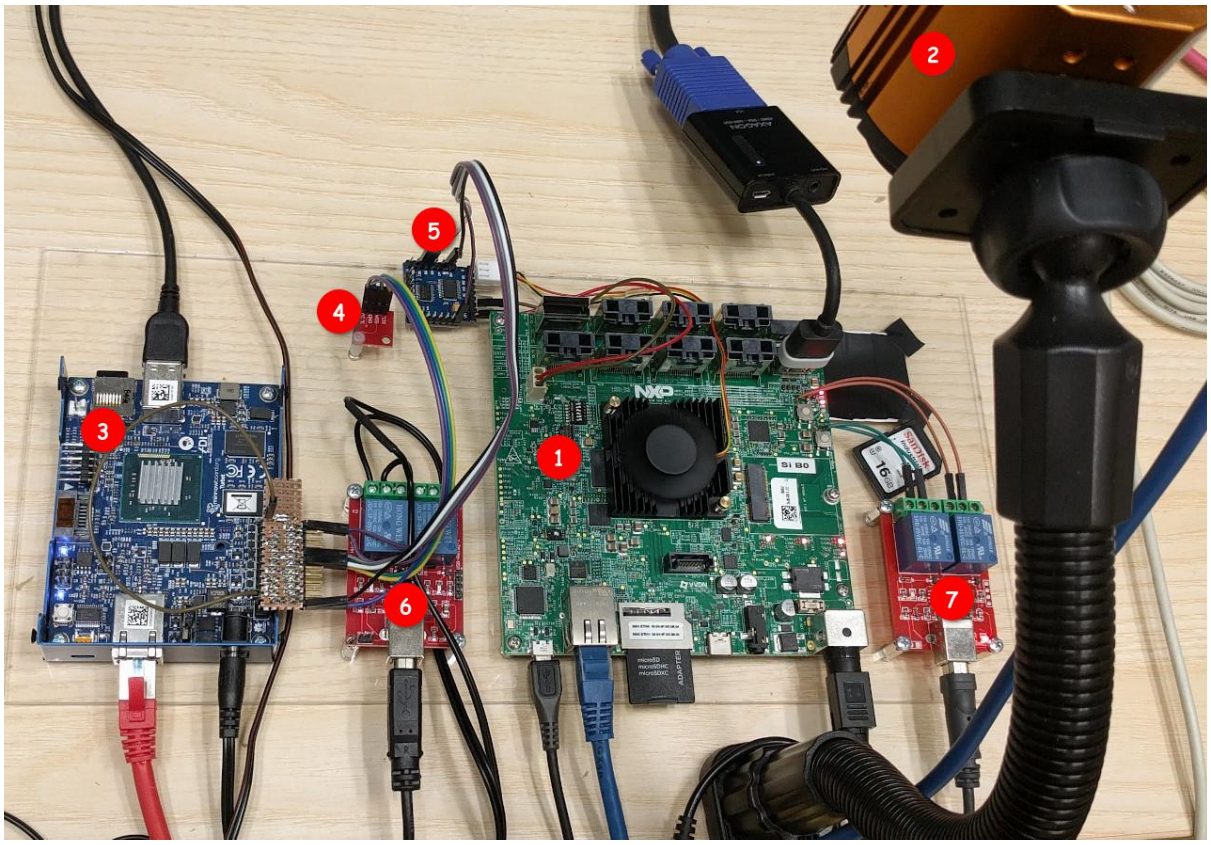

Figure 1: THERMAC testbed

2 Testbed setup

This section describes the setup of THERMAC testbed used at ČVUT (see Figure 1). The testbed consists of:

- i.MX 8QuadMax Multisensory Enablement Kit (MEK) ①
- Workswell thermal camera ②
- Minnowboard Turbot – similar to Raspberry Pi, but x86 architecture ③
  The following devices are connected to it:
  - Thermal camera (via USB3)
  - HTU21 ambient temperature sensor (via I2C) ④
  - WeMos D1 mini fan motor controller (via I2C) ⑤
- Two USB-controlled relays:
  - one can power-cycle the Minnowboard ⑥
  - one is connected to i.MX8 reset and power buttons ⑦

2.1 i.MX8 board

The i.MX8 board is the target of our thermal measurements and thermal-aware scheduling techniques. It is equipped with i.MX 8QM SoC consisting of two CPU clusters (4× ARM Cortex-A53 and 2× ARM Cortex-A72) and two GPUs.

To make testing of different software stacks and OS kernel easier, we do not boot the board from NXP-supplied SD card images, but from network. This also makes it easier to share the board between multiple users. All these aspects are described in more details in following subsections.

2.1.1 Network boot

We have configured the U-Boot bootloader of our i.MX8 board, to load the Linux kernel and mount the root file system from network. The kernel is fetched via TFTP protocol while the root filesystem is mounted via the NFS protocol.

Mounting the root filesystem from network has the following advantages compared to using an SD card:

- It is easy to switch to another filesystem/distribution (e.g. Yocto and Debian)
- Each board user can have its own private filesystem/distribution
- When the board crashes, data is stored safely on the server. Therefore, risk of data corruption is greatly reduced.

To load the kernel and root filesystem from network, we do the following:

1. Interrupt automatic boot by pressing a key after seeing U-Boot message: *Hit any key to stop autoboot:*
2. At "=>" prompt, enter the following U-Boot commands:

```
setenv autoload no
dhcp
run loadfdt # load devicetree from SD card
tftpboot /srv/tftp/imx8/Image--4.19.35-r0-imx8qmmek-20200402085702.bin
setenv bootargs console=ttyLP0,115200 earlycon=lpuart32,0x5a060000,115200 \
            rootwait root=/dev/nfs ip=dhcp rw \
            nfsroot=<server-ip>:<server-path>,v3,tcp
booti ${loadaddr} - ${fdt_addr}
```

The `<server-ip>` and `<server-path>` parameters should match the NFS server configuration. `<server-path>` is where the root file system is stored on the server. It can be either the file system generated by Yocto or another distribution such as Debian, which can be prepared as shown in the next section.

2.1.2 Debian root file system

This section shows, how to create a Debian file system for the i.MX8 board. We use the `debootsrap` tool for this purpose. Due to the fact, that we want to create the root file system for a different architecture (ARM64) than our server has (x86), we have to perform the installation in two stages:

1. The initial stage is run on the x86 server:

```
debootstrap --arch=arm64 --foreign buster /srv/nfs/imx8_debian
```

2. The second stage must be run on the board. To do this, boot the board from the initial stage file system as shown above, but append `init=/bin/sh` to the `setenv bootargs` line. This boots the Linux kernel and drops you directly into the shell. From the shell, run:

   ```
   /debootstrap/debootstrap --second-stage
   ```

For the installation to be useful, you should set at least root password and allow logging in from the serial line:

```
passwd
echo ttyLP0 >> /etc/securetty
```

Then you can reboot the system without `init=/bin/sh` and use it as an ordinary Debian system. For example, install whatever packages with `apt install` etc.

2.1.3 Board sharing and automated boot

To share the board between multiple users and to automate its booting, we use the novaboot[1] tool. The board is accessible by running the following command:

```
ssh imx8@rtime.ciirc.cvut.cz
```

This runs so called novaboot shell on the rtime server, which mediates connection to and control of the board. Without any argument (as above) or with the `console` argument, the command powers on and resets the board and then it connects to the board's serial console. If the board is currently used by another user, an information message is shown and the command waits until the board becomes free.

Depending on the configuration on the server (which can be different for each user), the board boots either a preconfigured system (e.g. Debian prepared as described above) or allows the user to interact with the bootloader and boot whatever the user wants.

To control the board, one can use `on`, `off` and `reset` commands as shown below:

```
ssh imx8@rtime.ciirc.cvut.cz reset
```

After the board boots up, one can log in via SSH as follows:

```
ssh root@imx8
```

2.1.4 Controlling the CPU fan

We use an external PWM motor controller to control the CPU fan speed as it seems that it cannot be controlled directly from the board. To control the fan, run (from the board):

```
ssh imx8fan@turbot <speed>
```

[1] *https://github.com/wentasah/novaboot*

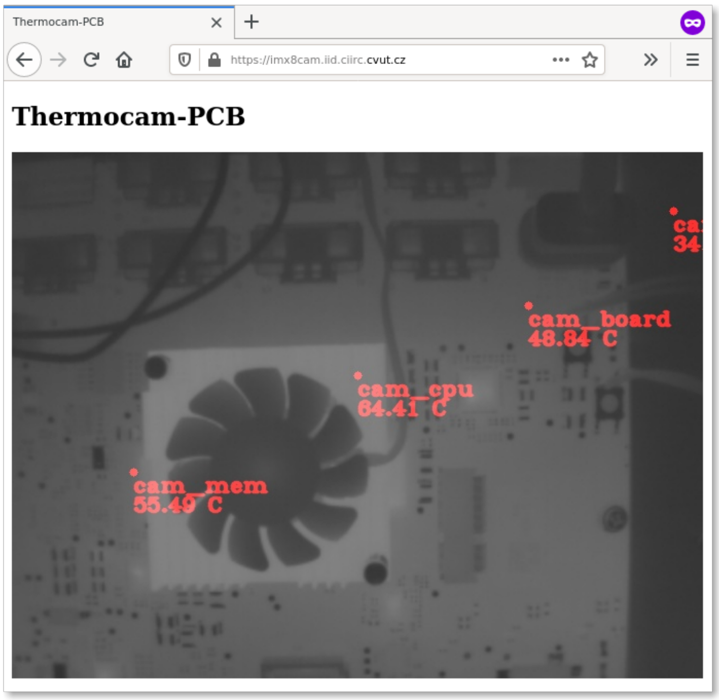

Figure 2: Web interface of thermocam-pcb

`<speed>` is a real number between 0 and 1. Zero means to switch the fan off, one means to rotate it at full speed. The numbers in between select PWM duty cycle of the fan motor controller. Note that speeds below approximately 0.3 are not sufficient to make the fan move. When the command is run from the board, it does not require any authentication.

2.1.5 Reading ambient temperature

Data from the ambient temperature sensor can be obtained from the imx8 board by running the following command:

```
ssh ambient@turbot
```

This will print the temperature to stdout every 10 seconds.

2.1.6 Thermal camera measurements

The thermal camera is connected to the Minnowboard Turbot. The Turbot board runs a custom application called thermocam-pcb[2], which processes the images from the camera and makes the results available over the HTTP protocol (see Fig. 2). From the external network, the camera is accessible via *https://imx8cam.iid.ciirc.cvut.cz/* and is protected by a password. From the internal network (i.e. from the imx8 board), it can be accessed without restrictions at *http://turbot:8080/*.

[2] *https://github.com/CTU-IIG/thermocam-pcb*

The thermocam-pcb tool measures the temperatures of selected points determined by thermo-cam-pcb configuration. Position of the points can be seen by pointing the web browser to one of the root URLs shown above. The temperatures can be read from imx8 with the following command:

```
curl -sS turbot:8080/temperatures.txt
```

Each line of the output has the format point_name=temperature, which is compatible with the thermobench tool (see Section 3). The thermobench sensor file (src/sensors.imx8) already contains this command so that the thermal camera temperatures are automatically captured during experiments when using this sensor file.

2.2 Minnowboard Turbot

Minnowboard Turbot[3] runs Ubuntu 16.04. This version is used because WIC_SDK, which is needed for reading the images from the thermal camera, does not support newer versions.

2.2.1 Ambient temperature sensor

We connected an ambient temperature sensor to the Minnowboard. We use the HTU21-based sensor bought in a local Czech store[4]. Similar modules are available from eBay, AliExpress or Adafruit. Temperature accuracy of the sensor is $\pm$0.3°C.

To make this sensor work, the following commands are executed during the boot:

```
modprobe htu21
echo htu21 0x40 > /sys/bus/i2c/devices/i2c-8/new_device
```

These commands are placed into /etc/rc.local.

Then, the temperature can be read with:

```
cat /sys/bus/i2c/devices/i2c-8/8-0040/iio:device0/in_temp_input
```

2.2.2 Fan motor controller

To control the i.MX8 CPU fan, we use WeMos D1 mini TB6612FNG dual motor driver shield[5] connected to the Minnowboard's I2C bus. We control the shield by a simple C program that sends required I2C messages.

2.2.3 NFS booting

To save SD card life and to have a faster system (our SD card is very slow), Minnowboard mounts its root filesystem from NFS too. The kernel and the bootloader remains on the SD card. This

[3]Information about the board was available from *http://minnowboard.org*, but this site is currently down. The raw content of the website is available at *https://github.com/MinnowBoard-org/website*.

[4]*https://www.gme.cz/i2c-senzor-teploty-a-vlhkosti-htu21d*

[5]*https://www.laskarduino.cz/wemos-d1-mini-tb6612fng-dual-motor-driver-shield--i2c/*

setup ensures that if the Ubuntu kernel package is updated (`apt upgrade`), the new kernel should boot automatically and the root filesystem stays on NFS. This was a bit tricky to set up, so we document what has been done here.

In `/etc/default/grub`, variable `GRUB_CMDLINE_LINUX` was set as follows:

```
GRUB_CMDLINE_LINUX=”console=ttyS0,115200 ip=dhcp \
  nfsroot=<server-ip>:<rootfs-path>,v3,nolock,tcp rw”
```

In `/usr/sbin/grub-mkconfig`, assignment to GRUB_DEVICE variable has been changed from:

```
GRUB_DEVICE=”`${grub_probe} --target=device /`”
```

to:

```
GRUB_DEVICE=/dev/nfs
```

Note that grub package update will revert these changes, but it seems there is no better way to accomplish the same result.

3 Thermobench tool

Thermobench is a tool that has been developed to perform temperature measurements needed for this project. It is an open source software hosted at GitHub[6]. It consists of three main parts:

- A C++ tool that collects various data, such as temperatures, and stores them to CSV files.
- Julia[7] package Thermobench.jl[8], for processing of the data measured with thermobench. It implements the methods described in Section 4 and allows one to generate various graphs from the data.
- A set of benchmarks that we run under thermobench to measure their thermal effects. These benchmarks are described in Section 5.

In a nutshell, the thermobench C++ tool runs a given benchmark and periodically collects various data (most importantly measured temperatures), which are then stored to a CSV file for later processing. Thermobench can store the following information to the CSV file:

- Timestamps,
- temperatures from Linux thermal-zone sensors,
- CPU frequencies,
- CPU load,
- standard output of the benchmark program – either everything or just selected values, for example values from lines matching `key=value`,
- output (or just selected values) from arbitrary commands; this feature is used for reading temperatures from the ambient temperature sensor and from the thermal camera with commands mentioned above in Sections 2.1.5 and 2.1.6.

Note that storing of the most of these pieces of information is optional and can be enabled or disabled via thermobench command line options.

Other notable thermobench features are:

- Waiting to cool down the platform to a given temperature before starting the benchmark.
- Controlling the fan.
- Possibility to specify the data to collect via a so called *sensor file*. This makes it easier to collect data from all relevant sensors available for the given board. The thermobench GitHub repository contains the `sensors.imx8` file, which defines the data sources used for our testbed.

Typically, thermobench is executed on our testbed with the following bash[9] command:

[6] *https://github.com/CTU-IIG/thermobench*
[7] *https://julialang.org/*
[8] *https://ctu-iig.github.io/thermobench/dev/*
[9] bash is needed for interpretation of *{0..5}* syntax.

```
thermobench --verbose --fan-cmd='ssh imx8fan@turbot' --fan-on=0.5 --time=900 \
         --wait=30 --wait-timeout=240 --sensors_file=sensors.imx8 \
         --cpu-usage --column=CPU{0..5}_work_done \
         --output=data.csv -- ./benchmark ...
```

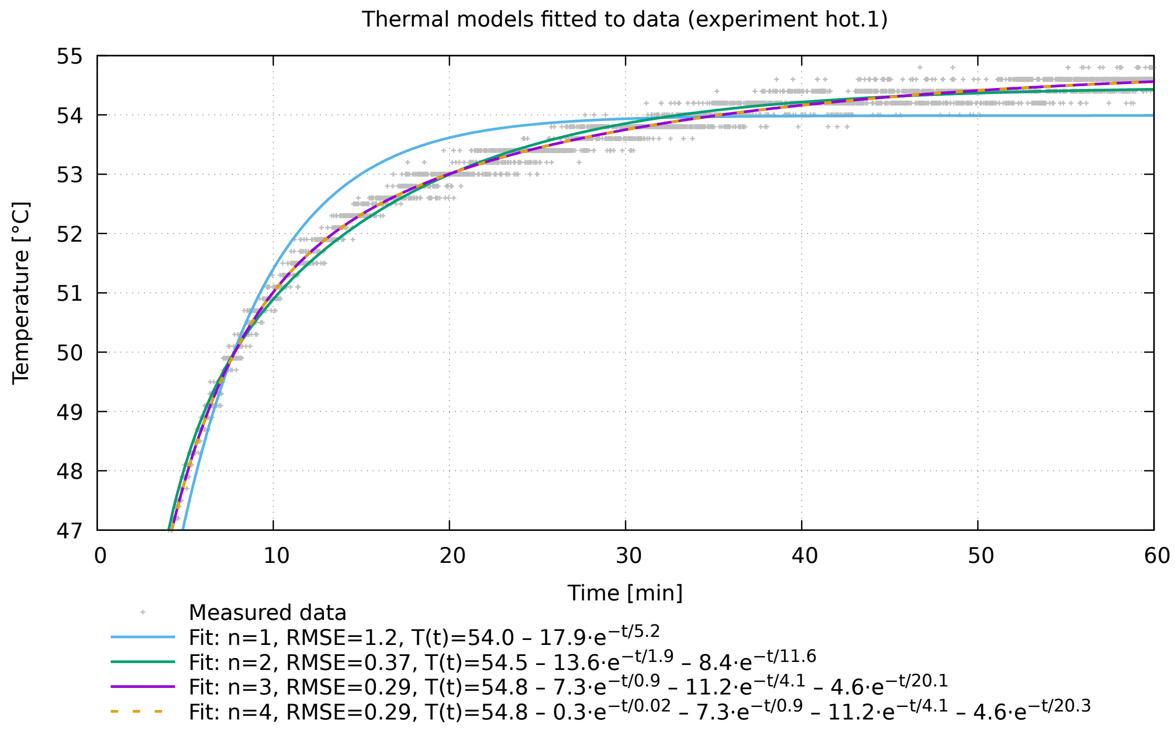

Figure 3: Measured data and fitted model (data < 47°C not visible)

4 Measurement processing

This section covers how to process the thermobench-generated CSV files to get useful and reproducible results. We also aim to evaluate the precision achievable with our testbed. The goal of themobench is to run certain workload and get a measure of the *heat flow* (with physical dimension $W = J \cdot s^{-1}$) it produces. In the following, we will denote the heat flow as $\dot{Q}$. We want to be able to precisely compare different workloads and say with high certainty which produces smaller heat flow, i.e., which one is more thermal efficient and how much.

4.1 Thermal model fitting

Figure 3 shows temperature measurements collected with Thermobench in a 60-minutes experiment, where the CPU fan was switched off. It is a well known fact [Bro06] that the heat flow produced by the chip is proportional to the difference of steady state chip temperature (denoted as T_∞ in the following) and ambient temperature T_{amb}:

$$\dot{Q} \propto T_\infty - T_{\text{amb}} = T'_\infty. \tag{1}$$

Therefore, to compare the heat flows of different workloads, it is sufficient to compare steady state temperatures T_∞ of those workloads. As the measured temperature is noisy, we estimate T_∞ by fitting a thermal model to the measured data and compute T_∞ from the model. Our

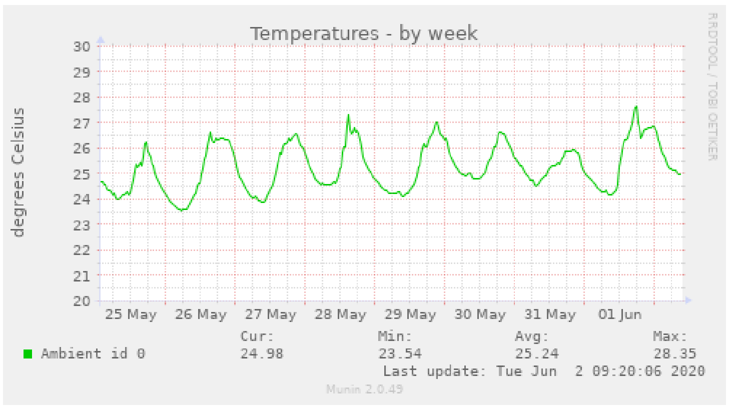

Figure 4: Evolution of ambient temperature over a week

thermal model describes evolution of the temperature as a function of time:

$$T(t) = T_\infty + \sum_{i=1}^{n} k_i e^{-\frac{t}{\tau_i}}, \tag{2}$$

where n is the order of the model and τ_i are so called *time constants* of the model, which determine "how fast" the temperature reacts to changes of the heat flow. We selected such a model because most thermal systems can be modeled with a set of linear differential equations and the solution of these equations has the same form as (2). Note that this model has the same form as the model derived in D3.1 [YR20].

By fitting the model (2) to the data measured with thermobench, we find the constants T_∞, k_i and τ_i. We select the order of the model manually.

In Figure 3, we see how models of different orders fit the data. First and second order models do not fit well – see their root-mean-square errors (RMSE). Third and fourth order models fit the best. The difference between them is negligible so we conclude that 3rd order model is sufficient. We see that for $n = 3$ the $T_\infty = 54.795 \pm 0.075$ °C (95% confidence intervals are the output of the fitting algorithm). Depending on model order, the estimated T_∞ differ by almost one degree. For the 3rd order model, time constants are $\tau_1 = 0.916 \pm 0.079$, $\tau_2 = 4.11 \pm 0.3$ and $\tau_3 = 20.1 \pm 2.1$ minutes. The highest time constant τ_3 is particularly important, because it determines how long we have to wait for temperature to reach the steady state – the exponential term reaches 95% of its contribution k_i in $3 \cdot \tau_i$. In case of τ_3, this gives ≈ 60 minutes. In Section 4.3, we examine how this time can be reduced.

4.2 Suppression of ambient temperature changes

Ambient temperature influences the temperature of the chip. Since experiments can run for long time, ambient temperature can easily change between different experiments or even during a single experiment, resulting in non-reproducible results. Figure 4 shows the evolution of ambient temperature of our testbed over a week. The testbed is located in an air-conditioned room with no possibility of opening windows.

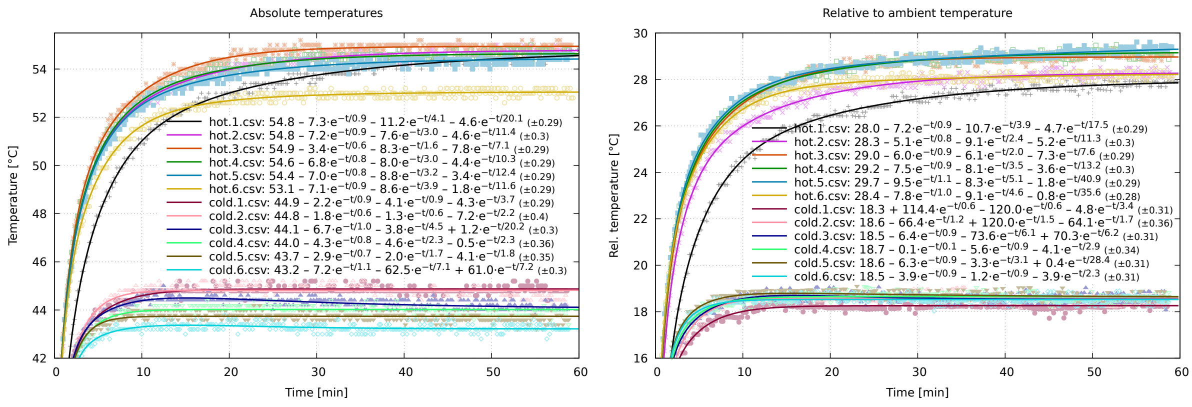

Figure 5: Comparison of model fitting with (right) and without (left) ambient temperature compensation. Fit error (RMSE) is given after ± sign in parentheses.

To suppress the effect of changes in ambient temperature, it would be best have a model of how ambient temperature influences the on-chip temperature. It should model the delays of the heat propagation from outside to the chip. Such a model is called a *transfer function* and can be estimated from measured by system identification methods based on models like OE, ARX or ARMAX[10]. We used these methods to estimate the transfer function, but the results were not satisfactory. It is a well know fact, that for these methods to give good results, it is necessary to excite the system a lot. Small changes in ambient temperature over long time (as in Fig. 4), together with relatively high measurement noise, rendered those methods ineffective.

Due to the lack of a better model, we compensate for the ambient temperature changes by simply subtracting ambient temperature from other measured temperatures. We call the resulting temperature as *relative temperature*. Figure 5 shows the results of fitting thermal models with and without ambient temperature compensation. The graphs show data from two kinds of experiments (hot and cold), each repeated 6 times. On the left, where uncompensated absolute temperatures were fitted, it can be seen that experiment hot.6 is an outlier, but after the compensation (right) it no longer stands out. Also, the difference between cold experiments is higher on the left than on the right.

This simple compensation for ambient temperature changes also helps with model fitting. Mean value of the fit errors (RMSE) from the mentioned experiments is 2% lower after compensation and maximum fit error is even 12% lower.

In the following text, we always fit thermal models to ambient temperature-compensated data. Hence, it can be assumed that

$$T_\infty = T'_\infty. \qquad (3)$$

[10] *https://www.mathworks.com/help/ident/ug/what-are-polynomial-models.html*

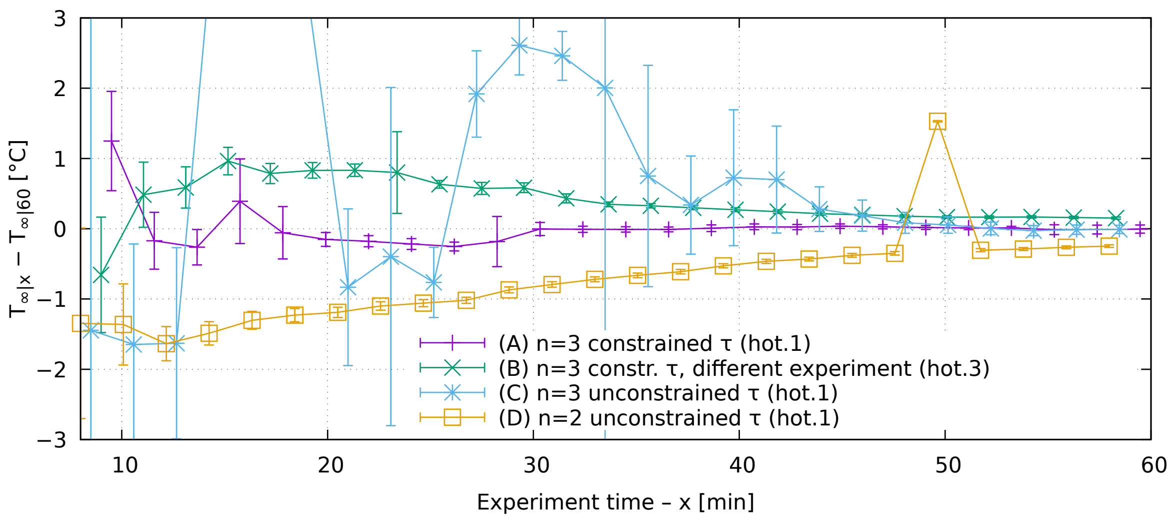

Figure 6: Relation between the length of the experiment and estimation of T_∞. Vertical axis shows the difference between T_∞ estimated from data of length x ($T_{\infty\,|x}$) and from full 60 minutes of data ($T_{\infty\,|60}$). Error bars represent 95% confidence intervals.

4.3 Reduction of experiment time

Running the experiments for one hour, as in Fig. 3, is very time consuming. We now investigate the possibility of fitting the thermal model from shorter experiments and estimating T_∞ from them. The difference of estimates from shorter and full 60minutes experiments can be seen in Figure 6.

We compare three ways of model fitting:

- 3rd order model with known time constant, i.e. τ_i are constrained to $\pm 1\%$ of the values estimated from 60 minutes of data.
- 3rd order model with unconstrained time constants, i.e., time constraints are fully estimated from the data,
- 2nd order model with unconstrained time constants.

For the 3rd order model and constrained τ, we see from line (A) in Fig. 6 that the model is able to predict the final temperature with good precision (< 0.5°C) from roughly 20 minutes of data. Unfortunately, time constants vary – see Fig. 5. We attribute this to the fact the our model represents a lumped-parameter system, where spacial distribution of heat production and transfer is ignored, whereas the real system is a distributed-parameter system where spacial dimension matters. Line (B) shows the results of the same method applied to ambient temperature-compensated data from the same workload but executed 2 hours later (hot.3 from Fig. 5). It can be seen that the difference follows the opposite trend than line A and good estimate is reached for 30 minutes experiment or longer.

Line (C) shows the results of fitting the “shortened” data without constraining the time constants close to the correct value. We see a lot of outliers and convergence to the constrained case roughly after 50 minutes.

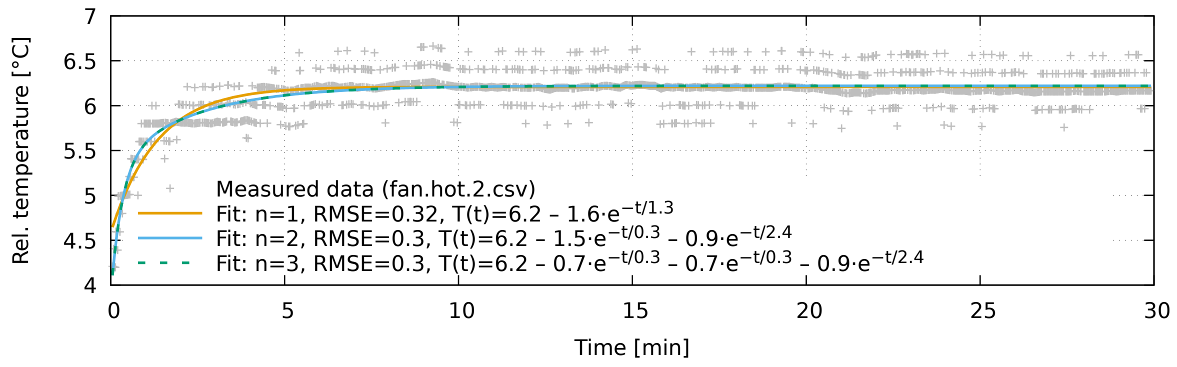

Figure 7: Measured data and fits with the fan switched on. Note that the ripples in the measured data are caused by compensation for ambient temperature.

Finally, line (D) shows results of fitting 2nd order model. We can observe a systematic estimation error.

To conclude, it is not possible to reliably estimate T_∞ when experiment time is shorter than $1.5 \max_i(\tau_i) \approx 30$ minutes. To have good estimations from experiments running for shorter time, it is necessary to decrease the time constants of the system.

4.4 Using fan to decrease time constants

In thermal systems, the time constant τ can be computed as $\tau = RC$, where R is *thermal resistance* between two objects with different temperature and C is *thermal capacity* of the object, whose temperature is being measured – in our case of the chip. If we want to reduce the time constant, we have two options:

- Reduce the capacity C, e.g., by removing the heat sink from the chip, or
- Reduce the thermal resistance R. This can achieved by a fan – the higher fan speed, the lower thermal resistance between the heat sink and the surrounding environment.

Removing the heat sink is a bit risky, especially for expensive boards like our i.MX8 so we decided to pursue the latter alternative with the fan. Figure 7 shows the temperatures from the same workload as in Fig. 3, but with the fan rotating at full speed. It can be seen that the steady state temperature is only 6.2 °C above ambient temperature and that the time constants are much smaller: 0.3 and 2.4 minutes. Also note that 3rd order model not necessary in this case as it is the same as 2nd order model.

When we try to estimate the T_∞ from shorter data, we can see (Fig. 8) that good results are obtained for experiments longer than 13 minutes with unconstrained-τ estimations. Constraining τ constants leads to systematic errors (line B). The 1st order model is slightly off even for $x \to 30$ (line D). The 3rd order model (line E) gives the same results as 2nd order model, but with few outliers.

It can happen, that for less intensive workloads, the T_∞ is even smaller than 6°C. In that case, the resolution of the temperature sensors might not be sufficient to give precise estimates. This problem can be mitigated by setting the fan to smaller than full speed. The results of 2nd

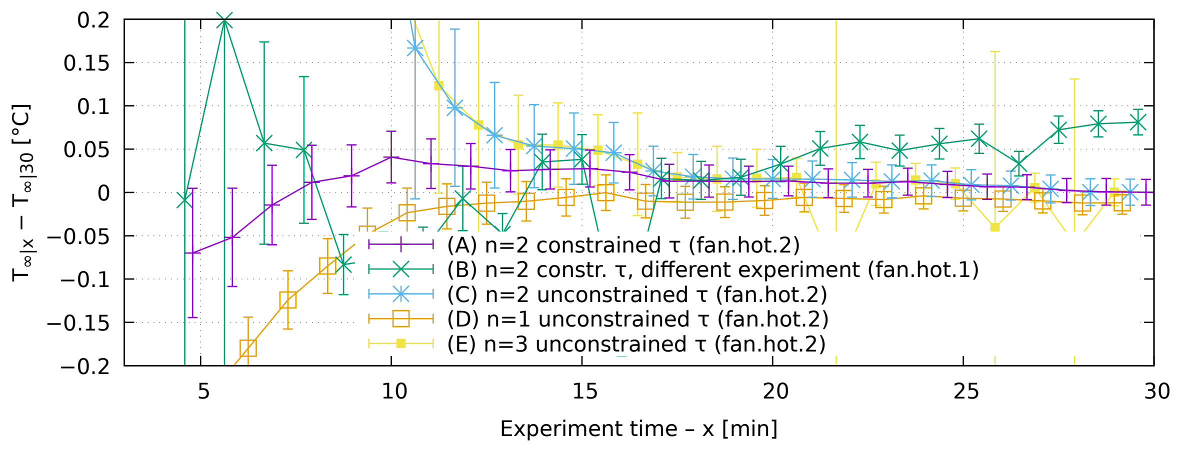

Figure 8: Relation between the length of the experiment and estimation of T_∞. Vertical axis shows the difference between T_∞ estimated from data of length x ($T_{\infty|x}$) and from full 30 minutes of data ($T_{\infty|30}$). Error bars represent 95% confidence intervals.

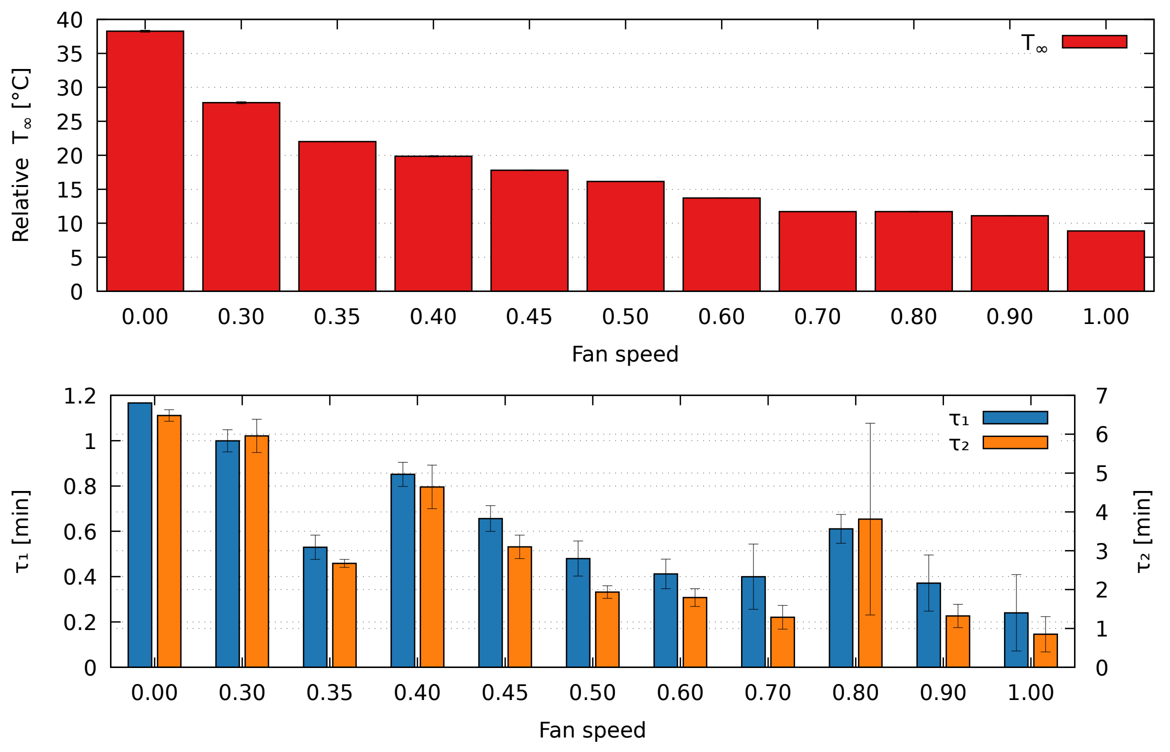

Figure 9: Relation between different fan speeds and estimated thermal model parameters. Note that 0.3 is the lowest speed that makes the fan move.

order thermal model fitting with different fan speeds are in Fig. 9. The T_∞ temperature clearly decreases with increasing fan speed and the same trend can be observed for time constants τ_1 and τ_2 except for few outliers (0.35 and 0.8). For some experiments, mostly with higher fan speeds (in the figure, it is visible for speed 0.8) the fitting algorithm is not able to precisely estimate τ_2, because the slow mode is almost not visible in the measured data. From the other experiments, we see that τ_2 drops from 6 to about 0.8 minute. The outliers in τ estimates are the reason why constraining the time constants to “correct” values during model fitting, as described in Section 4.3, does not always give good results.

4.5 Conclusion

Processing the data from thermobench measurements is not fully automatic and requires a few manual steps. These are mainly selection of model order, experiment duration and appropriate fan speed. Additionally, after fitting the thermal model, one has to check that the model fitting algorithm did not end up in a local minimum, which results in imprecise estimations of T_∞. After these manual steps, the methods described in this section give reasonably precise estimate of T_∞, which is proportional to the heat flow $\dot{Q}$ generated by the executed workload.

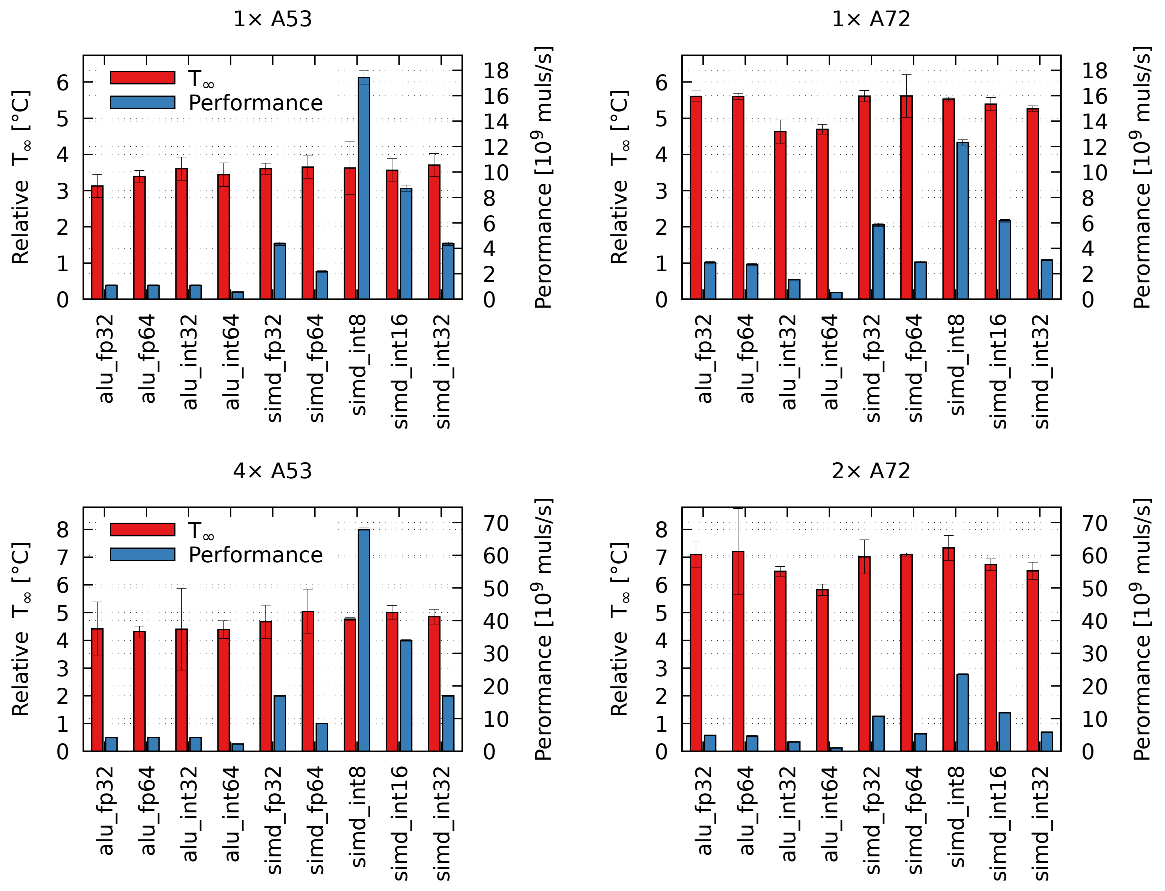

Figure 10: Comparison of relative steady state temperature T_∞ and performance of multiplication instructions on different CPUs.

5 Benchmarks

Thermobench repository also contains a few benchmarks that are used to assess thermal and performance characteristics of the i.MX8 platform and will be used to measure the performance of temperature reduction techniques. These benchmarks are described in the following subsections.

5.1 CPU instructions

The directory `benchmarks/CPU/instr` contains various micro-benchmarks that perform different (mostly) arithmetic instructions. With these benchmarks, we can compare performance and thermal efficiency of different CPUs and CPU clusters. For example, in Figure 10, we can compare multiplication operations.

The top row of Fig. 10 allows to compare single-core performance of A53 and A72 CPUs. A72 offers higher performance for non-SIMD and floating-point SIMD instructions. Surprisingly, integer SIMD instructions are faster on A53. CPU temperature does not depend significantly on particular type of instruction. In average, A72 produces 51.1 $\pm$ 5.7% more heat than A53,

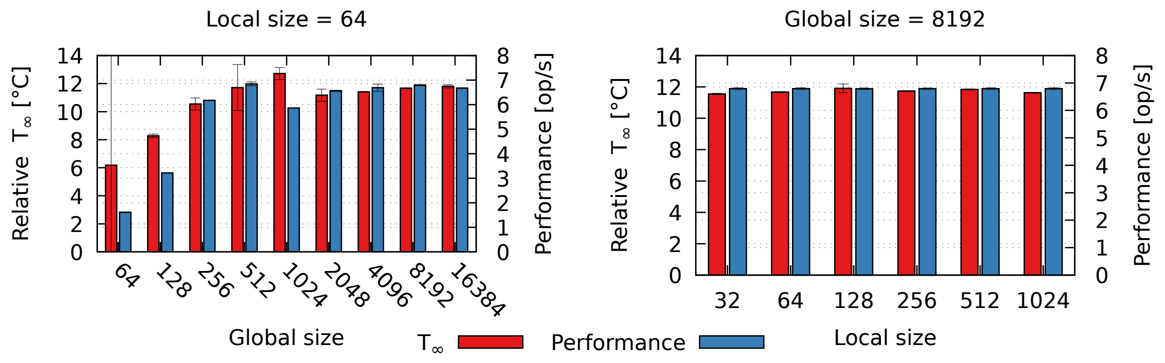

Figure 11: Performance of a compute-bound GPU benchmark

while delivering only 92.9 $\pm$ 1.6% of A53 performance.

The bottom row compares multi-core performance, where the benchmark was running on all CPUs in the cluster. For A53 we see that compared to single-core case, the performance increases 4 times but temperature rises only by 31.9$\pm$8.0%. For A72, performance rises 2× and temperature by 27.8 $\pm$ 4.4%. A53 cluster is always faster than A72, but A72 cluster produces 46.4 $\pm$ 8.6% more heat.

5.2 GPU workload

To evaluate performance of the GPU, we use several benchmarks:

- Graphics performance is assessed with benchmarks from gtec-demo-framework[11] available from imx-gpu-sdk Yocto package.
- OpenCL performance is evaluated by custom developed programs `mandelbrot` and `cl-mem`, executing compute-bound and memory-bound GPU workloads respectively.

For the future, we expect to use more OpenCL (e.g., *https://www.iwocl.org/resources/opencl-benchmarks/*) and Vulkan compute benchmarks.

Later in this section, we present some results of our OpenCL benchmarks. In OpenCL, compute work is divided into so called *work items*. The total number of work items is called *global size*. The work items are being worked on by *kernel* code running in so called *work groups*. Each work group processes a certain number (called *local size*) of work items in parallel. The work groups execute on the GPU either sequentially or in parallel, depending on their (local) size and size of the GPU.

The results from the OpenCL `mandelbrot` (compute-bound) benchmark can be seen in Figure 11. We run the benchmark kernel with different parameters and compare the results. The total work of the benchmark is divided into *global size* of work items and these are executed in work groups of different *local size*. The left graphs shows the results from experiments with different global size and same local size. It can be seen that maximum performance can be

[11] *https://github.com/nxpmicro/gtec-demo-framework.git*

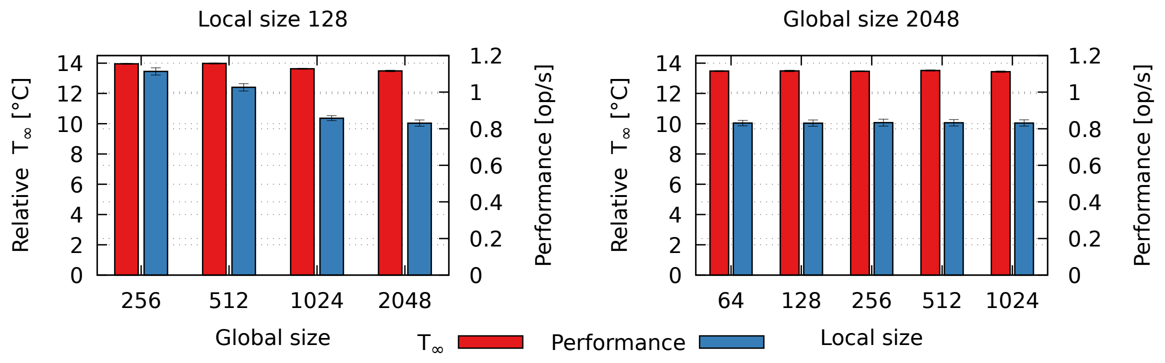

Figure 12: Performance of a memory-bound GPU benchmark

reached for global size of 512 or more (with small performance drop for size of 1024). Smaller global sizes cannot utilize the full GPU parallelism and hence the computation takes longer time. The steady state temperature T_∞ decreases with decreasing performance.

The right hand side graph in Fig. 11 shows that there is no significant difference in both performance and temperature when the same amount of work items is divided into differently sized work groups.

Figure 12 shows the results of the memory-bound benchmark, which just reads memory. In the left, we can see that higher parallelism (global size) leads to lower performance, because the memory bandwidth is the limiting factor. The temperature slightly decreases with the decreasing performance. The right graphs again shows that different work division makes no difference.

5.3 CPU memory subsystem

The thermobench repository also contains a program called `membench`. It can stress various parts of the CPU memory subsystem and measures achievable memory bandwidth.

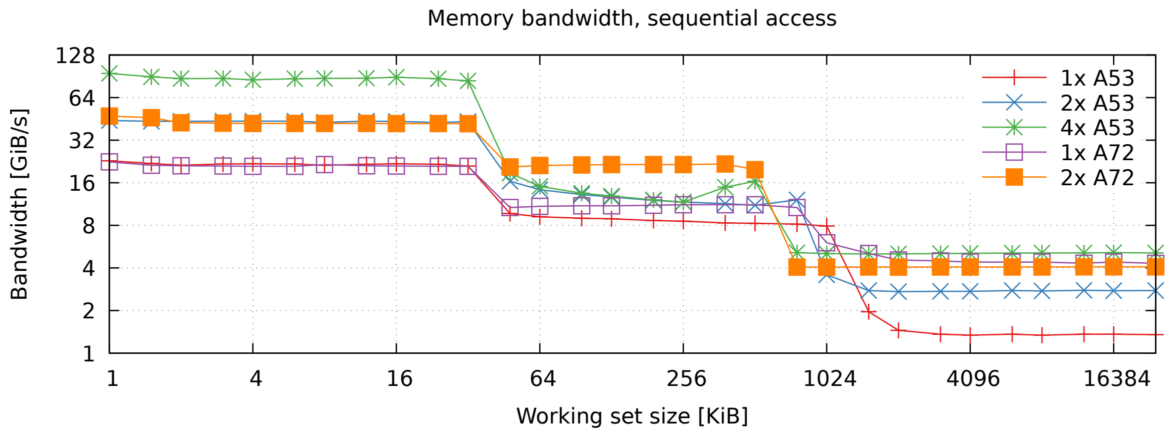

Figure 13: Memory bandwidth

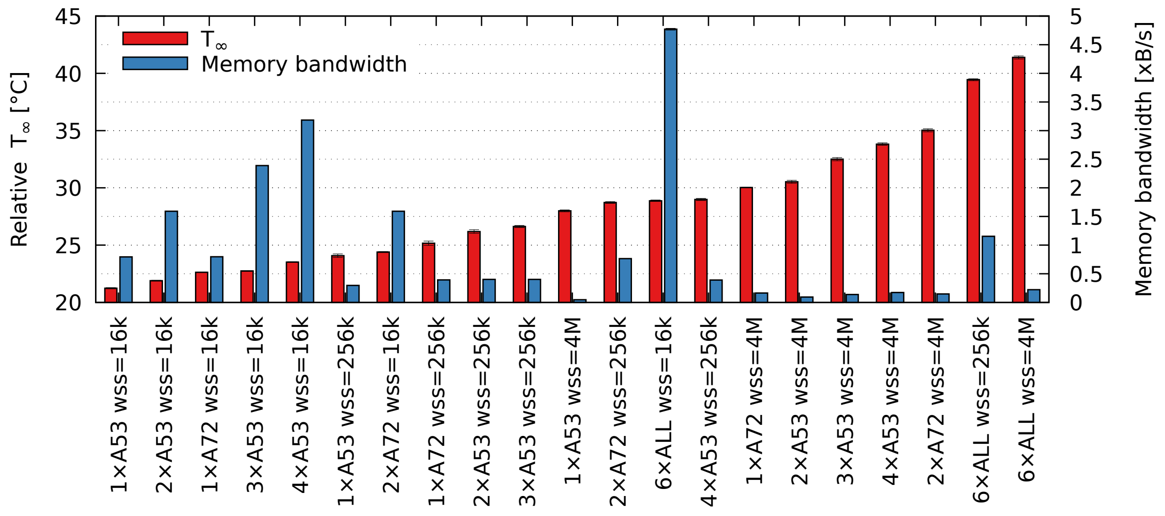

Figure 14: Memory performance and temperature (measured with the fan switched off)

The memory bandwidth achieved by various numbers of CPUs can be seen in Figure 13. The bandwidth of L1 cache memory, i.e., for working set size (WSS) ≤ 32 KiB, is the highest, followed by L2 cache bandwidth (32 < WSS < 1 MiB) and the lowest bandwidth is, unsurprisingly, available for DRAM accesses (WSS > 1 MiB). One can observe, that DRAM bandwidth available to 2 A72 cores is slightly lower than bandwidth available to 4 A53 cores.

The temperature effect of accessing different parts of the memory subsystem can be seen in Figure 14. Clearly, L1 cache accesses are the most thermal efficient, whereas DRAM accesses are the lest efficient.

5.4 Other benchmarks

It is planned that the performance of our scheduling mechanisms will be further assessed with the following benchmarks:

- CoreMark® (*https://www.eembc.org/coremark/*),
- AutoBench™ (*https://www.eembc.org/autobench/*), and
- 3D software renderers: *https://github.com/ssloy/tinyrenderer*, *https://github.com/GeekyMoose/3d-cpu-engine*.

6 Conclusions

We demonstrated the functionality of our testbed and of the toolchain for processing the data gathered with the testbed. In the next project phase, we will concentrate on development of various scheduling techniques and the developed testbed will be used for assessing performance of the proposed techniques.

References

[Bro06] Brown, F. T. *Engineering System Dynamics: A Unified Graph-Centered Approach, Second Edition*. 2 edition. Boca Raton, FL: CRC Press, Aug. 15, 2006. 1059 pp. ISBN: 978-0-8493-9648-9.

[YR20] Yomsi, P. M. and Rodríguez, J. P. *Preliminary Implementation of a Thermal-Aware Resource Management Policy*. THERMAC project deliverable D3.1. June 2020.