F@RasPi Project

Building a Raspberry Pi Rack for Folding@Home

Silvio Martin, November 2020 – January 2021

Introduction

I got this idea in early November 2020 with a COVID-19 lockdown ahead here in Germany, when I was looking for something more useful to do than staring at news tickers. I already had Folding@Home running on my iMac for some months. Then I read, that there is a client for the Raspberry Pi available. At this very moment this project was born. So what is this about?

Folding@Home is a crowd computing project, which allows anyone to donate computing time to scientists, who investigate the shape of proteins. The analysis process is called „folding“. This research may help to understand and hopefully to cure several deseases. The Raspberry Pi is a tiny, versatile and cheap computer.

Combining both sounds useful and like fun at the same time. But just putting a Raspberry Pi together an installing the Folding@Home client doesn't even keep you busy for a single evening. So in order to be a true lockdown project, some more needed to be done. So here is my plan from early November 2020:

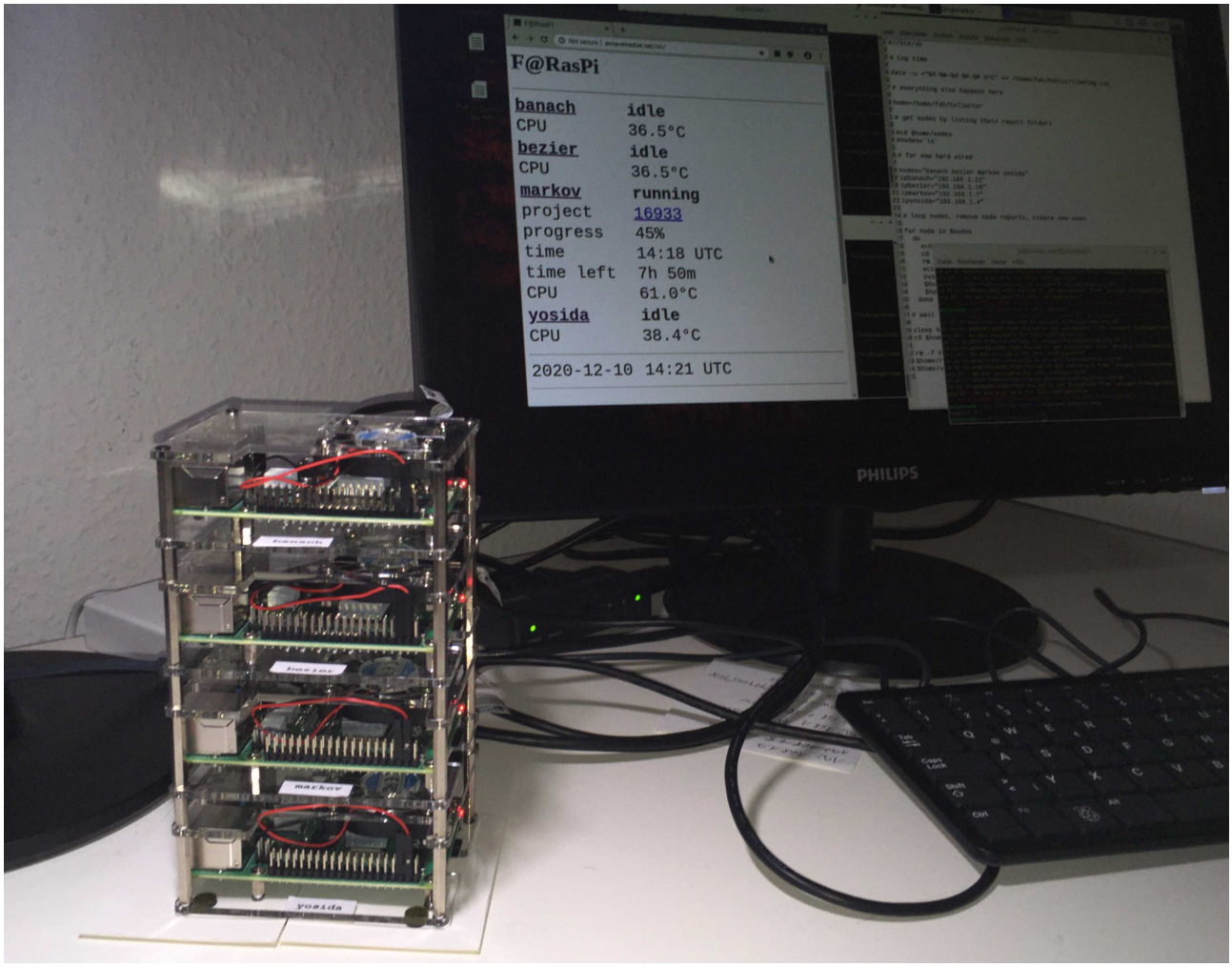
- Build a rack of some Raspberry Pi computers, maybe 4 or 6
- Get the Folding@Home client working on them
- Compile a report and upload it to my web space on a regular schedule:
  - Which Raspberry Pi is working on which project?
  - What is the progress?
  - How is the Raspberry doing, i.e. what is the temperature of the CPU?
- When done, brag with it (which is, what I am doing right now)

This report tells how I set this up. It assumes that the reader has some basic knowledge of Linux, command line interfaces and some script programming. If you are bored by it, just flip through and look at the pictures of my rack.

Silvio Martin, Oberhausen (Germany), January 2021

Content

The Target



Let us first have a look at the result. It contains four Raspberry Pi. The display in the background shows the HTML report. The node named „markov“ is currently running and some details on the status of the run are printed. The other nodes, named „banach“, „bezier“ and „yosida“ are idle, which happens from time to time, if there is no workpackage available for your configuration. Actually I even had a few days in a row where all nodes were idle.

The four nodes rotate in creating the report. Every 15 minutes another node creates it. So there is no primary or secondary node. They are all equal. If one drops out, the rest will still work.

In order to create a report, the reporting node needs to find the other nodes in the network, then login and read the CPU temperature, get the logfile of the Folding@Home client, analyze the logfile to find out the status, create a HTML file and upload it to the FTP server. The result from my rack is here:

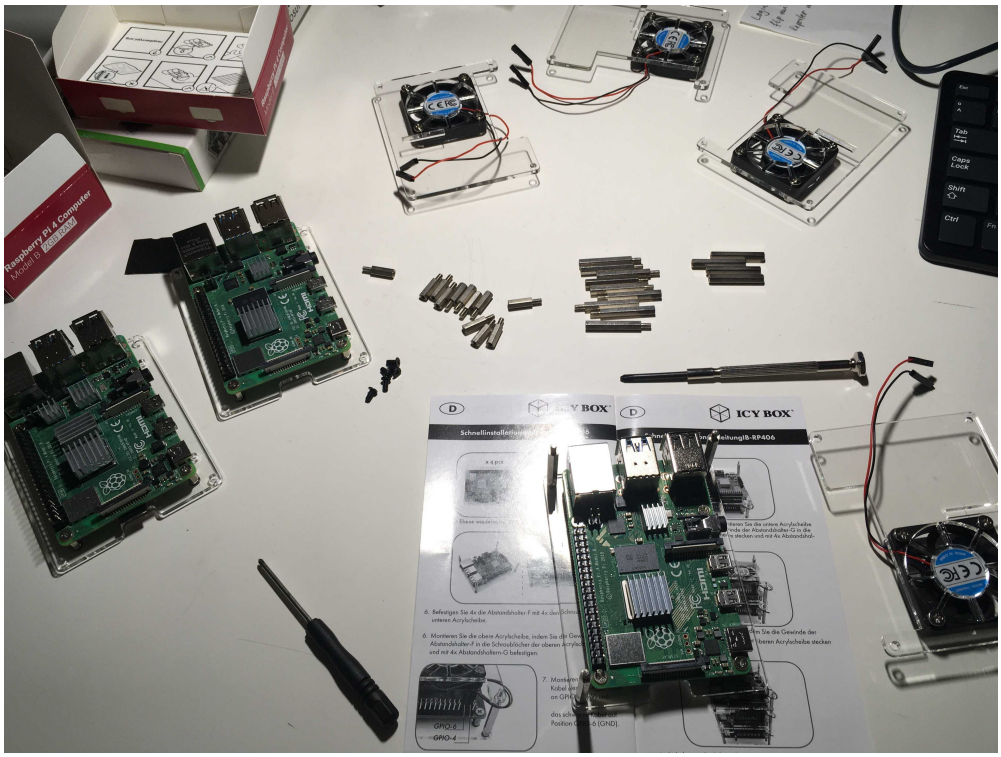
http://www.anne-emscher.net/fah/

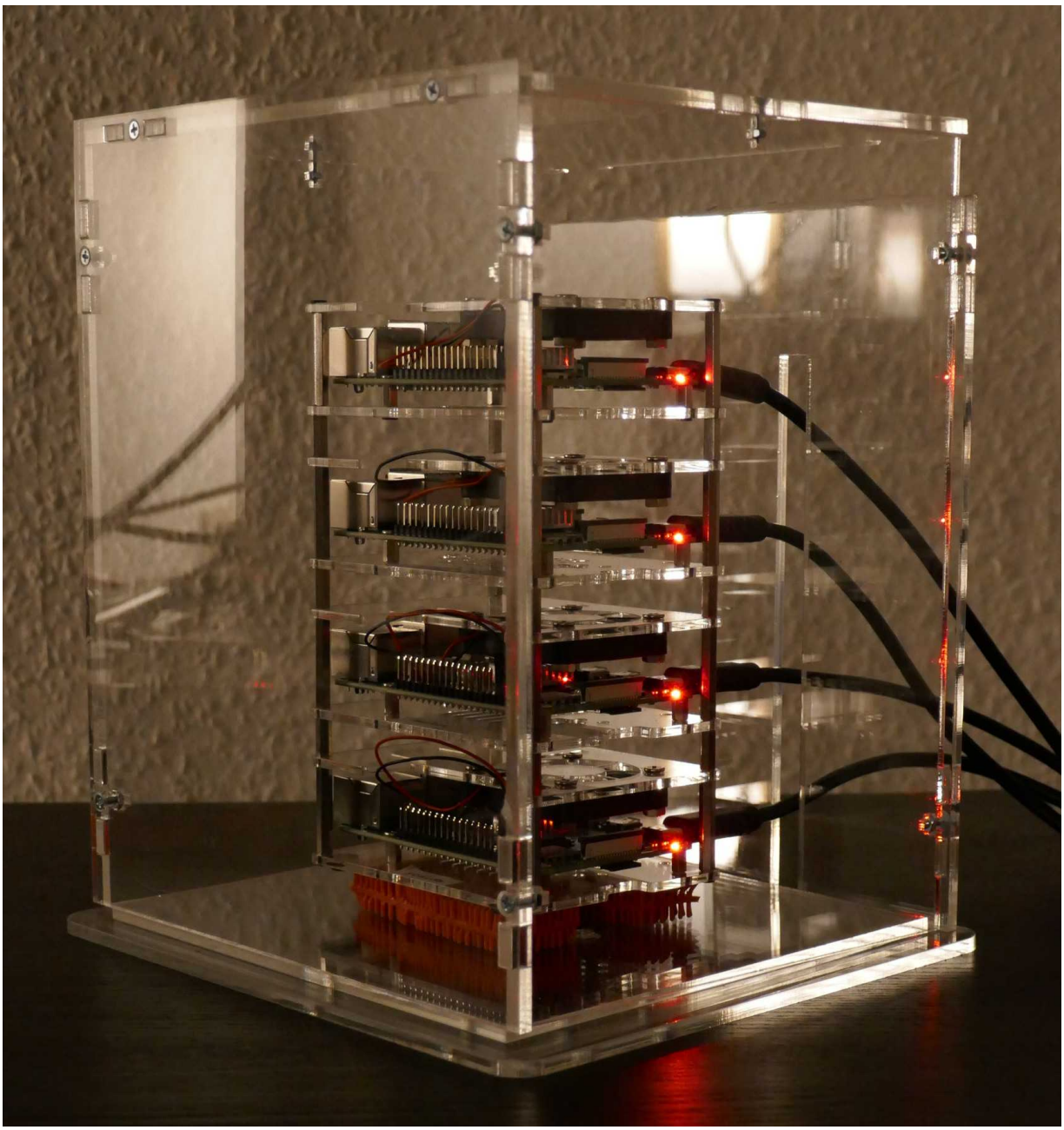
Is this project now finished? For good luck the Raspberry Pi is very versatile. I already have a 2 row 16 character LCD display here, which I want to attach to the rack.

The Hardware

One can build essentally the same rack with different hardware. Here is what I used. Links to some products are at the end of this report.

- 4 Raspberry Pi 4B with 2GB memory. The 4B has a 64 bit ARM CPU with 4 cores and runs with 1,5 GHz. I run Folding@Home on my iMac as well. This is also a 64 bit UNIX type system. The client needs around 200 MB RAM there. So 2GB RAM should be enough.
- Casing ICY BOX IB-RP406 from Radsonic Technology. The sides are open. The casing comes with heat sinks and 40 mm fans for all Raspberries.The fans of the ICY BOX casing are really quiet compared to the 20 mm fan, which I used before. The rubber stickers for the bottom don't decouple it from my desk, thus causing a humming sound without further measures. All in all a very good buy, especially due to the quiet fans. The casing comes with good directions – no puzzle at all.
- Multi-USB charger fantec QC3-A51 with 4 USB-A sockets and one USB-C. It delivers enough power so that the Raspberries do not crash. I often to disconnect and connect my passive HDMI switch, which does not happen with the original Raspberry power supply.
- Four cables USB-A / USB-C for power supply
- USB mouse and keyboard. I got the ones from Raspberry.
- Monitor and cable micro-HDMI to whatever your monitor wants. Mine has a HDMI socket.
- 4 micro SD cards16GB SanDisk Ultra. I think these are the smallest, which are now available. In my rack less than 4GB is used. So maybe any currently available card works.





The Raspberry Pi case from ICY BOX is open at the sides. In order to protect the computers from dust, another case around is needed. At sora.de one can configure acrylic cases e.g. for record players. A case for a record player needs a slot for cables. On can configure it even to be 15 cm high, as in my case. Here is how I ordered it:

Inner width 170 cm, inner depth 15 cm, inner height 21 cm, with base, cable slot 150 mm high and 50 mm from right inner wall.

When running at full power in the naked ICY BOX, the CPU temperature is around 60 °C, but in the acrylic case it went up to almost 80 °C, which is too much for me. At 85°C the Raspberry Pi clocks down the CPU. So I cut a part off the back wall, such that there is a horizontal gap of 15 mm height at the top of the back. Now I see temperatures somtimes touching 70 °C, which is okay.

Setting Up the Raspberries for Folding

In this chapter the Raspberry Pi is set up so that the Folding@Home client is running.

The Folding@Home client for 64 bit ARM processors requires a 64 bit operating system. However the currently released Raspberry Pi OS is only a 32 bit operating system. There is a beta version of the 64 bit Raspberry Pi OS, which we need to use.

Running Folding@Home with the beta version works, however other things may not work. E.g. VNC was disabled in the first beta version, which I used. Other things might fail as well. Consider this, if you plan to use one of your Raspberries for such a project. You may not be able to use it for what you are currently doing with it.

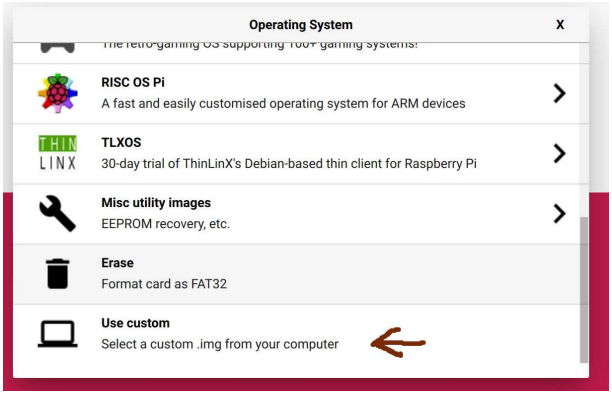
To get started, you need to download and install the Raspberry Pi Imager on your „boss“ computer. You can download it from here for macOS, Windows and Ubuntu x86 Linux:

https://www.raspberrypi.org/software/

Then download the newest 64 bit Raspberry Pi OS here:

https://downloads.raspberrypi.org/raspios_arm64/images/

Put the micro-SD card in your card reader and install the OS in the Raspberry Pi Imager. You need to select „Use custom“ and then select the file.



My iMac asks me for the password of an administrator before it allows writing to the card. So make sure that you have the required permissions.

Now place the SD card in the Raspberry Pi, connect mouse, keyboard, monitor and finally the power chord. This turns the Raspberry Pi on and it boots the operating system.

When a Raspberry Pi with a freshly installed operating system is started for the first time, it asks for some setup:

- Select country specific settings according to your situation and location.
- Raspberry Pi OS has a default user „pi“ with the default password „raspberry“. Change the

default password. In our setting with four identical Raspberries, use the same password for all of them.

- Connect the Raspberry Pi to your WLAN
- The last step in the setup is to load updates and to restart the Raspberry Pi.

Now the Raspberry Pi is set up to get started. To download the software for Folding@Home, open the web browser by clicking on the globe on top left:

In the web browser go to https://foldingathome.org/alternative-downloads/ and download the three files, which are listed in the box ARM Linux / Raspberry Pi. Confirm to keep files when asked.

Install the three files viewer with

```
sudo apt install ./fahclient_7.6.21_arm64.deb
```

Install the other two files accordingly. When installing the client, it asks for a user name, team number and passkey. The preset name is Anonymous. If you are registered at Folding@Home, you may enter your credentials here, or you just hit return. You may change this later in the FAHControl tool. Also it asks, how much folding power you want to use initially. Select full if you don't do anything else on this Raspberry Pi. Finally select that the FAH client is started automatically. This is useful if you had e.g. a power outage, because otherwise you would need to connect each node with mouse, keyboard and monitor and start Folding@Home again.

Folding@Home is now running. You can launch the FAHControl tool now from the Raspberry menu, in the second category. It may be „Education“ in English. In German it is „Bildung“.

Prepare the Raspberry for Collaboration

In order to enable the Raspberry Pi to communicate with its siblings, some more setup is required. Go to the Raspberry menu for settings and launch the Raspberry Pi configuration tool. Then:

- Enter the name of the node (Hostname). The default is raspberry.
- Disable the automatic login as user „pi“ (for security reasons)

- Enable SSH in the interfaces tab. This allows `ssh` and `sftp` connections to nodes.

Click OK and confirm to reboot.

Install Expect, which allows to remote control interactive shells. Open a terminal and execute

```
sudo apt-get install expect
```

Finally connect to the FTP server of your web space with sftp:

```
sftp (user)@(host)
```

You should know user and host, if you have some web space somewhere. You will need to confirm to trust the host at first login.

You may reward yourself with a can of beer now, just to give you an idea of the size of the rack

The Reporting System

F@RasPi

banach	**running**
project	16922
progress	98%
time	04:55 UTC
time left	0h 17m
CPU	57.4°C
bezier	**running**
project	16933
progress	1%
time	04:55 UTC
time left	14h 21m
CPU	57.9°C
markov	**offline**
yosida*	**running**
project	16933
progress	4%
time	04:51 UTC
time left	13h 47m
CPU	57.4°C

2020-12-17 04:57 UTC

What is this? (PDF report to come soon)

Folding@Home

Let us have a look at the final report, which is sent here: http://www.anne-emscher.net/fah/

For each node it reports the status (`running`, `idle` or `offline`), and depending on availability some more data:

- project: the project ID at F@H as a link to the project description
- progress: the percentage of completion of the currect workpackage
- time: the time when these infos were generated, as UTC (world time)
- time left: the estimated time left at the given time
- CPU: the temperature of the CPU

At the bottom it reports the date and time when this report was generated, also as UTC. The reporting node gets a `*` appended to its name (here: `yosida`).

Since this report is generated on a regular schedule, a cron job to run a master script is required. The temperature can be read with a simple command on each node. For this, one needs to login to all nodes and run this command remotely. A proven way to do this is with an Expect script. The other information needs to be extracted from the F@H log file. So first the log files must the pulled over from the other nodes with sftp (which also is done with an Expect script) and then the log file must be analyzed in order to create the report. If at work I would have done this with Tcl, because I have been developing in Tcl for more than 20 years now. But the „Pi“ in Raspberry stands for „Python“. So I worked through P.M. Heathcote's Python book first and used Python. All „bracket“ scripts are written in Bourne shell, just to make it harder than it needs to be. Keep in mind please, that this is not an urgent task, but a lockdown project.

Project Folder

The home folder for the project on each node is `~/FAH`. Let us have a look at the folder structure:

```
~/FAH/
      /html
      /ip
      /nodes/banach
            /bezier
            /binomi
            /markov
            /yosida
```

The toplevel folder `~/FAH` contains the scripts, which search for the nodes in the WLAN, collect data, generate the report and upload it to the web space.

The folder `~/FAH/html` contains three files: `head.html`, `tail.html`, `index.html`. The first two files contain the static head and tail of the report file. The file `index.html` is created after all required data are collected.

The folder `~/FAH/ip` is used while finding the nodes in the WLAN. While this in in progress, it contains files, whose names are the IP addresses in the WLAN. At the end each of these files is either empty, or it contains a string „`host=` *`(hostname)`*“, e.g. „`host=markov`“. The latter ones are copied to the accoding node folder. After searching the IP addresses of the nodes, the files in `~/FAH/ip` are deleted. So most of the time the folder is empty.

The node folders, e.g. `~/FAH/nodes/banach` ideally contain three or four files: *`192.168.1.6`* (or some other valid IP address in your WLAN), `log.txt`, `node.url` (optional), `temperature.txt`. The name of the first file is the IP address of the node. The file `log.txt` is the latest log file of the FAH Client. The file `temperature.txt` contains the latest CPU temperature of the node.

You may add more files everywhere with one exception: Scripts get a list of nodes by executing `ls` in the folder `~/FAH/nodes`. So any additional file or folder will appear as an offline node.

Scripts and Configuration Files

In principle one may execute each script manually, as long as they are executed with the right argument (IP address) in the right directory and if the required input is available. But typically the user will run only three of them. The other scripts are called from within these three scripts.

`searchnodes.sh`	Probe into all IP addresses in the WLAN and check, if this is a node. A node is recognized by logging into it as the common user of all FAH nodes. If an IP address belongs to a recognized node, it puts a file named with the IP address into the node's folder. It uses the IPv4 address.
`deploy.sh`	Deploy the complete folder `~/FAH` to all known nodes. This feature is used when adding a new node or after changing or extending the scripts. This script assumes that the node folders contain the IP addresses. So it is a good idea to run the script `searchnodes.sh` first.
`fah.sh`	This script generates the report. It is added to the cron tab, so that it runs on a regular schedule, but it can also be run manually. The script calls `searchnodes.sh` first to make sure, that the IP addresses in the node folders are up to date.

In the next sections each of the scripts is explained in more detail. Also some traps and tricks are explained. Three script languages are used: Bourne shell scripts (`*.sh`), Expect (`*.expect`) and Python 3 (`*.py`).

Login Data

Configuration files: `local.login`, `webftp.login`

The Expect scripts need to login with `ssh` or `sftp` to the nodes and the FTP server of the web space. Expect is actually Tcl plus an extension. With the right format (`set` *`variable value`*), one just needs to source the files in order to read their content into variables. Clearly one should never do this with foreign configuration files, because the configuration file could execute anything with the permissions of the current user.

Script searchnodes.sh

Supporting scripts: `checkip.expect`, `setnodeip.py`

The script first obtains the own IP address in the network. In order to find all possibe IP addresses, the script picks the IP address of the node, where it is running, let say 192.168.1.10. Then it cuts off the last number and inserts all numbers from 0 to 255. Thus it iterates from 192.168.1.0 to 192.168.1.255.

Each IP address is tested by `checkip.expect`. It tries to login to the requested IP address as the common user (remember that all nodes use the same user name and password). In case of success it produces a line „`host=` *`(user)`*“, e.g. „`host=bezier`“. This line is then put into a file in the folder `~/FAH/ip`, whose name is the IP address. In case that the script can not login at the IP address, the file is empty.

The final step is to process the information in the folder `~/FAH/ip`. The script `setnodeip.py` reads the content of all files. If the content is „`host=` *`(node)`*“, e.g. „`host=yosida`“, then the file is copied into the according node folder.

Script deploy.sh

Supporting script: `deploynode.sh`

The scrip `deploy.sh` reads the IPv4 addresses of the nodes from the node folders. It loops through all other nodes and calls `deploynode.expect` in order to deploy `~/FAH` to that node.

The script `deploynode.expect` opens a FTP session to the requested IP address and puts the current working folder to the according folder on the target node.

Script fah.sh

Supporting scripts: `searchnodes.sh`, `logfile.expect`, `temperature.expect`, `report.py`, `upload.expect`.

The script `fah.sh` is executed as a cron job and drives the whole report generation process. First it runs `searchnodes.sh` to get the current IP addresses of the nodes. Then it removes the old data from the node folders and loops through all nodes twice in order to populate the data again. In the loop it calls `logfile.expect` and `temperature.expect` in order to get the F@H logfile and the CPU temperature of the node. After the loop it calls `report.py`, which creates the report

file `index.html`. Finally it calls `upload.expect`, which uploads the file to the web space.

The first script file `searchnodes.sh` is discussed in detail above.

The script `logfile.expect` logs into a node with `sftp` and gets the file `log.txt` from the folder `/var/lib/fahclient` of the node.

The script `temperature.expect` logs into a node with `ssh` and executes the command `vcgencmd measure_temp`, which outputs the CPU temperature. This output is catched with a `grep` command in `fah.sh` and piped into the file `temperature.txt` in the node folder.

The Python script `report.py` scans all collected data and generates the HTML report by sticking together the file `~/FAH/html/head.html`, the content generated within this script and the file `~/FAH/html/tail.html`. It walks through each node. If there is no log file `log.txt` in the node folder, then the node is `offline`. Else it is either `idle` (available but not working) or `running`. If there is a log file, then the status is initially `idle`.

The format of log files is typically not a promised property in a software project. Analysis of logfiles should be reduced as much as possible. Here only two types of lines are used. The first one tells us the current project:

```
07:44:27:WU00:FS00:0xa8:Project: 16933 (Run 25, Clone 9, Gen 0)
```

This is the only appearance of the string „`Project:` “ in the log files. This line is used to set the project ID (here 16933). The other type of lines looks like this:

```
07:44:58:WU00:FS00:0xa8:Completed 1 out of 500000 steps (0%)
07:53:51:WU00:FS00:0xa8:Completed 5000 out of 500000 steps (1%)
...
22:02:23:WU00:FS00:0xa8:Completed 495000 out of 500000 steps (99%)
22:10:56:WU00:FS00:0xa8:Completed 500000 out of 500000 steps (100%)
```

These are the only occurrences of the string „`:Completed` “. The end of the line determines the progress in percent. It is printed for all intergers from 0 to 100. If such a line is found, then the status is `running`, except if it ends with „`(100%)`“. Then the status is `idle`. The time stamp of the progress is taken from the start of the last line of this type. The estimated time left is calculated from the last two time stamps of such lines. So if the progress is still 0%, then no time left is given, because there is no previous time stamp. One needs to add 24 hours, if the time left is negative. In this case the last time stamp was taken just after midnight and the previous just before midnight.

Traps and Tricks

`searchnodes.sh` In Raspberry Pi OS the command `hostname -I` prints the IPv4 and IPv6 address in one line:

```
192.168.1.9 2a01:c23:6c5d:d600:2356:1d1:63cb:56e1
```

So it is easy to get it from there. Though I want to use the script also on my Mac, because the rack is not connected to display, keyboard and mouse. In Darwin this command just produces an error. With a little help

from Google I found a way to extract the IPv4 address from the much more complex output of `ifconig`, using `grep`, `awk` and `cut`.

`searchnodes.sh` The trick using `ifconfig` works well when run manually, but it failed when running in a cron job. The reason is that cron jobs are run without the folder `/sbin` in the `PATH`. So this needs to be called with the full path `/sbin/ifconfig`. It works in in Darwin and Raspberry Pi OS.

`searchnodes.sh` 1. It may take some seconds until the script checkip.expect finishes. In order to speed up the process, the call of `checkip.expect` is sent to the background by appending `&`. After the loop the script sleeps for 60 seconds waiting for the output.

2. This process fails sometimes. So it tries twice to get the IPv4 addresses.

Both tricks are used in in the script `fah.sh` as well.

`checkip.expect` When connecting to another computer with `ssh` or `sftp` for the first time, one needs to confirm, that this computer is to be added to the known hosts. The next time this question will not appear any more. This is the reason, why one needs to login to the ftp server of the web space manually in order to set up a new node, so that all logins look the same. This is no issue, because typically these FTP servers do not change their ssh finger print.

It gets more complicated in a local WLAN with no fixed IP addresses assigned. Whenever a node is rebooted or the router is restarted, the node may get a different IP address. This also happens from time to time without restarting. So the Expect script needs to answer according to the response of the host. Two nodes might even swap their IPv4 addresses, so that one needs to remove the finger print from the knon hosts list. Since this may happen frequently, I decided to remove the IP address from the known hosts list every time in order to keep the script simple.

`checkip.expect` Expect echoes the input. So in order to grep the intended line „`host=`*`(node)`*“, one can not send a line like „`echo host=$h`“ to the node. So this command needed to be hidden behind a variable.

`fah.sh` As the script `searchnodes.sh`, this sends calls of `logfile.expect` and `temperature.expect` to the background for each node in order to speed up the process. Then it sleeps for 60 seconds. The process is repeated once in order to avoid issues with network hiccups.

Setting Up the Cron Job

To run a job on a regular schedule, add it to your cron tab:

```
crontab -e
```

You may delete all comment lines with a hash (#) at the beginning. I prefer to keep the one with the format description. Then save it and check by listing your crontab:

```
pi@banach:~ $ crontab -l
# m h  dom mon dow   command
10  *   *   *   *    /home/pi/FAH/fah.sh
```

This runs the script `/home/pi/FAH/fah.sh` at minute (`m`) 10 of every (`*`) hour (`h`), every (`*`) day of month (`dom`), every (`*`) month (`mon`) and every (`*`) day of week (`dow`). For more complex schedules see the crontab help. You don't need to set this up on every node. You can select any subset of nodes for generating the report and define their schedule.

Adding a New Node

Setup the Raspberry Pi for folding and prepare it for collaboration as described in earlier chapters.

Login to another node, create a folder with the name of the new node in `~/FAH/nodes`. Open a terminal and run the script `searchnodes.sh`, then run `deploy.sh`.

If you want the new node also to upload reports on a schedule, then create a folder `~/FAH` on it. Add the script `/home/pi/FAH/fah.sh` to the crontab according to your desired schedule. FTP to the web server once in order to add it to the known hosts.

Performance

I do not have data for a work package of the same project on a Raspberry Pi and on my iMac. Actually I had, but I lost it. Looking at the estimated points per day: If you are after credit points, try something else. On my iMac (3,4 GHz quad core i5) I get around 40,000 points per day using 3 cores. On the Raspberry Pi 4B with a 1,5 GHz quad core ARM processor I get around 1700 points per day on 4 cores. The credit points at F@H are non-linear. One should compare the time for executing a workpackage of the same project. As mentioned above, I did this, but I lost my data. If I remember right, then a workpackage for the same project needs 4x more time on a Raspberry Pi (ARM architecture) than on my iMac (Intel architecture) after normalizing by the number of cores and the CPU clock speed. So the Intel architecture would have an advantage of 4 over the ARM architecture for the kind of jobs, which Folding@Home does. And if I remember right.

Selecting the Names of the Nodes

Selecting names of the nodes is the hardest part. The selection tells a lot about you. In my local netwok I used to name everything after the Peanuts. I am writing this right now on Rerun. However here I needed a different and more serious approach. I started with one Raspberry Pi, because I wanted to do a proof of concept first. I named the first Raspberry Pi after Andrey Markov, because Markov chains play an important role in the Folding@Home method (they say).

So the standard was set: All of the nodes need to be named after mathematicians, because I have no further insight into biochemistry. Also the names must be 6 letters long, so that the report looks good. Here is my choice:

Stefan **Banach** is the father of my favorite area in mathematics: Functional analysis. Also he worked on very weird things.

Pierre Bézier actually is not a mathematician, but an engineer. But the people at the Université Pierre-et-Marie-Curie thought, that his work on computer aided geometric design is good enough to grant him a Ph.D. in mathematics. And so do I. His Bézier curves play a big role in my work in gas turbine engineering for design of turbine airfoil profiles.

Kōsaku Yosida wrote a great text book on functional analysis, plus the Hille-Yosida theorem plays an important role in my Ph.D. thesis.

Binomi, the inventor of the binomial formula, is an old joke at universities in Germany.

Customization

In my environment the user is the standard user `pi` in Raspberry Pi OS and the home folder is `/home/pi/FAH`. One can select another user in the configuration file `local.login`. One can select another folder in the user's home (or elsewhere) in the script `fah.sh`. The other scripts either need to be executed in the right folder, or they are clever enough to know where they are (`*.expect` scripts). I actually tried to use a different user, but the new user could not read the CPU temperature. So I gave up this idea and went ahead with the default user `pi`.

As of now I let all four nodes in the rack create reports one after the other, such that a new report is uploaded every 15 minutes. The fifth node `binomi` resides on my desk and is only turned on from time to time. It does not create reports on a schedule. if you see an asterisk next to it, then you know that I was just trying something. Anyway, you need to deploy the scripts only to those nodes, who need to create reports. Even if the scripts are on a node, it will not run automatically until the cron tab is set up accordingly.

Complete Listings of Files

With the complete listing of the files I provide a software. Just for the case that some tries to use it (which I don't believe) and that it causes heavy damage (maybe more likely, but …), I protect my self by applying a license, which excludes any liability. I picked the MIT license from the Open Source Initiative, because it is quite short:

Code in *italics* is a placeholder and needs to be replaced with your actual data.

local.login

```
#
# local.login - user and password of nodes (must be the same for all)
#

set user YOUR_USER_NAME
set password YOUR_PASSWORD
```

webftp.login

```
#
# webftp.login - user, host, password and remote path on web ftp server
#

set user YOUR_USER_NAME
set host YOUR_HOST
set password YOUR_PASSWORD
set path "YOUR_PATH_ON_HOST"
```

searchnodes.sh

```
#!/bin/sh

#
# searchnodes.sh - Searches all nodes by scanning IP addresses.
#                  Must be run in the project's home folder.
#

# base is the IP address until the last period.
ip=`/sbin/ifconfig | grep "inet " | grep -v 127.0.0.1 |awk '{$1=$1};1'|cut -d \  -f 2`
base=`echo $ip|awk '{split($0,a,"."); print a[1]"."a[2]"."a[3]"."}'`

home=`pwd`
rm -rf $home/nodes/*/${base}*

# try twice to get the IP addresses, because it fails occasionally

for count in 1 2
do

    cd $home/ip
    i=0
    while [ $i -lt 256 ]
    do
        ip=${base}${i}
        $home/checkip.expect $ip | grep "host=" > $ip &
        i=`expr $i + 1`
    done

    sleep 60
    cd $home

    $home/setnodeip.py

done

cd $home/ip
rm -rf *.*.*.*
```

checkip.expect

```
#!/usr/bin/env expect

#
# checkip.expect - checks whether IP address is of a host where user can login
#
#                  The IP address is provided as argument
#                  Reads user and password from file local.login
#

set ip [lindex $argv 0]
set home [file normalize [file dirname [info script]]]

# read user and password
source [file join $home local.login]
```

```
# delete host from ssh known hosts. hosts may even swap IP addresses over night

set cmd "ssh-keygen -f \"$env(HOME)/.ssh/known_hosts\" -R \"${ip}\""
catch { eval exec $cmd }

set address ${user}@${ip}
spawn ssh $address

expect "*connecting (yes/no)?"
send "yes\n"
expect "*password:"
send "${password}\n"

# hide "host=" in a variable, so that only one line matches in 'grep "host="'

expect "${user}@*:~ \$ "
send "export h=`hostname`\n"
expect "${user}@*:~ \$ "
send "export a=\"host\"\n"
expect "${user}@*:~ \$ "
send "echo \"\$a=\$h\"\n"
expect "${user}@*:~ \$ "
send "exit\n"
```

setnodeip.py

```
#!/usr/bin/env python3

#
# setnodeip.py - sets the nodes' IP addresses by copying the right files to the
#                nodes' folders
#

# subprocess to get nodes and IP files with "ls"

import subprocess

def getNodes():
    nodes = []
    command = subprocess.run(["ls","nodes"],stdout=subprocess.PIPE,text=True)
    for node in command.stdout.split("\n"):
        if len(node) > 0:
            nodes.append(node)
    return nodes

def getIPfiles():
    IPfiles = []
    command = subprocess.run(["ls","ip"],stdout=subprocess.PIPE,text=True)
    for filename in command.stdout.split("\n"):
        if len(filename) > 0:
            IPfiles.append(filename)
    return IPfiles

#
# MAIN
#
```

```
nodes   = getNodes()
IPfiles = getIPfiles()

for filename in IPfiles:
    filename="ip/"+filename
    ipfile = open(filename,"r")
    content = ipfile.read()
    if len(content) > 5:
        host = content[5:-1]
        if host in nodes:
            command = subprocess.run(["cp",filename,"nodes/"+host],text=True)
    ipfile.close()
```

deploy.sh

```
#!/bin/sh

#
# deploy.sh - Deploy FAH installation from here to all nodes
#
#             This assumes, that the IP addresses in the node
#             folders are correct and that the nodes are added to
#             known_hosts. Must be run in the project's home folder.

home=`pwd`

# get nodes by listing their folders

cd $home/nodes
nodes=`ls`

host=`hostname`

# loop nodes and deploy

for node in $nodes
    do
        if [ $node != $host ]
            then
                cd $home/nodes/$node
                address=`ls *.*.*.*`
                $home/deploynode.expect $address &
        fi
    done
```

deploynode.expect

```
#!/usr/bin/env expect

#
# deploynode.expect - Deploy FAH installation to one node via sftp
#
#                     The node's IP address is provided as argument
#                     Reads user and password from file local.login
#

set ip [lindex $argv 0]
```

```
set home [file normalize [file dirname [info script]]]

# read user and password
source [file join $home local.login]

set address ${user}@${ip}
spawn sftp $address
expect "${address}'s password:"
send "${password}\n"
expect "sftp>"
send "cd ${home}\n"
expect "sftp>"
send "cd ..\n"
expect "sftp>"
send "put -r ${home}\n"
expect "sftp>"
send "exit\n\n"
```

fah.sh

```
#!/bin/sh

#
# fah.sh - update web report with status of all nodes
#

home=~/FAH

# update IP addresses

cd $home
$home/searchnodes.sh

# get nodes by listing their folders

cd $home/nodes
nodes=`ls`

# delete the report files

for node in $nodes
do
    cd $home/nodes/$node
    rm -f *.txt
done

# loop nodes twice to get temperatures and logfiles, because this fails occasionally

for j in 1 2
do

    for node in $nodes
    do
        cd $home/nodes/$node
        address=`ls *.*.*.*`
        if [ ! -e log.txt -o ]
            then $home/logfile.expect $address &
```

```
            sleep 1
        fi
        if [ ! -e temperature.txt -o ! -s temperature.txt ]
            then $home/temperature.expect $address | grep "temp=" >> temperature.txt &
            sleep 1
        fi
    done
    sleep 60

done

# update html report

cd $home
rm -f html/index.html
$home/report.py
$home/upload.expect
```

logfile.expect

```
#!/usr/bin/env expect

#
# logfile.expect - copies F@H logfile from node to current directory
#
#                  The IP address of the node is provided as argument
#                  Reads user and password from file local.login
#

set ip [lindex $argv 0]
set home [file normalize [file dirname [info script]]]

# read user and password
source [file join $home local.login]

set address ${user}@${ip}
spawn sftp $address
expect "${address}'s password:"
send "${password}\n"
expect "sftp>"
send "cd /var/lib/fahclient\n"
expect "sftp>"
send "get log.txt\n"
expect "sftp>"
send "exit\n"
```

temperature.expect

```
#!/usr/bin/env expect

#
# temperature.expect - gets the temperature of a node
#
#                  The IP address of the node is provided as argument
#                  Reads user and password from file local.login
#
```

```
set ip [lindex $argv 0]
set home [file normalize [file dirname [info script]]]

# read user and password
source [file join $home local.login]

set address ${user}@${ip}
spawn ssh $address
expect "${address}'s password:"
send "${password}\n"
expect "${user}@*:~ \$ "
send "vcgencmd measure_temp\n"
expect "${user}@*:~ \$ "
send "exit\n"
```

report.py

```
#!/usr/bin/env python3

#
# report.py - generate HTML report for all nodes
#

# subprocess to get time stamp  with "date" and nodes with "ls"
import subprocess

def getTemperature(node):
    try:
        tempFile = open("nodes/"+node+"/temperature.txt")
    except:
        temp = "N/A"
    else:
        temp = tempFile.read()
        tempFile.close()
        length = len(temp)
        if length == 0:
            temp = "N/A"
        else:
            temp = temp[5:length-3]
    return temp

def getDateTimeStamp():
    command = subprocess.run(["date","-u","+\"%Y-%m-%d\""],stdout=subprocess.PIPE,text=True)
    dateStamp = command.stdout[1:-2]
    command = subprocess.run(["date","-u","+\"%H:%M UTC\""],stdout=subprocess.PIPE,text=True)
    timeStamp = command.stdout[1:-2]
    return dateStamp,timeStamp

def getNodes():
    nodes = []
    command = subprocess.run(["ls","nodes"],stdout=subprocess.PIPE,text=True)
    for node in command.stdout.split("\n"):
        if len(node) > 0:
            nodes.append(node)
    return nodes

def getHost():
    command = subprocess.run(["hostname"],stdout=subprocess.PIPE,text=True)
    host = command.stdout[0:-1]
    return host

def getProgress(node):
```

```
    status   = "N/A"
    project  = "N/A"
    progress = "N/A"
    lastTime = "N/A"
    timeLeft = "N/A"
    try:
        logFile = open("nodes/"+node+"/log.txt","r")
    except:
        status   = "offline"
    else:
        status   = "idle"
        timeOld  = "N/A"
        for row in logFile:
            # "Project: " is indicator for start of workpackage
            index = row.find("Project: ")
            if (index >= 0):
                status = "running"
                row = row[index+9:]
                index = row.index(" ")
                project = row[0:index]
            # (100%) is indicator for finished work package
            if row.find(":Completed ") >=0:
                index = row.find(" steps (100%)")
                if index >= 0:
                    status   = "idle"
                    project  = "N/A"
                    progress = "N/A"
                    lastTime = "N/A"
                    timeLeft = "N/A"
                    timeOld  = "N/A"
                else:
                    index1 = row.index("(")
                    index2 = row.index(")")
                    progress = row[index1+1:index2]
                    timeNew  = 3600 * int(row[0:2]) + 60 * int(row[3:5]) + int(row[6:8])
                    lastTime = row[0:5]
                    if timeOld != "N/A":
                        time = timeNew - timeOld
                        if time < 0:
                            time += 86400
                        try:
                            timeLeft = time * (100 - int(progress[:-1]))
                        except:
                            timeLeft = "N/A"
                    timeOld = timeNew
        logFile.close()
    return status,project,progress,lastTime,timeLeft

#
# MAIN
#

host = getHost()
nodes = getNodes()

# read HTML head and write to HTML report

htmlFile = open("html/index.html","w")

headFile = open("html/head.html","r")
header   = headFile.read()
htmlFile.write(header)
htmlFile.write("\n")
htmlFile.write("    <table>\n")
```

```
# loop nodes, get data and append to HTML report

for node in nodes:
    temp      = getTemperature(node)
    status,project,progress,lastTime,timeLeft = getProgress(node)
    print("node =",node,\
          "status =",status,\
          "project =",project,\
          "progress =",progress,\
          "time =",lastTime,\
          "Time left = ",timeLeft,\
          "Temperature =",temp)
    try:
        nodeUrlFile = open("nodes/"+node+"/node.url","r")
        nodeUrl=nodeUrlFile.read()
        nodeText="<a href=\""+nodeUrl+"\">"+node+"</a>"
        nodeUrlFile.close()
    except:
        nodeText = node
    if node == host:
        nodeText = nodeText + "*"
    # space to align table with date / time stamp in next table
    count = (10-len(node))
    if count < 0:
        count = 0
    space = " " * count
    htmlFile.write("      <tr><td><b>"+nodeText+space+"</b></td><td><b>"+status+"</b></td></tr>\n")
    if status != "offline":
        project = "<a href=\" https://stats.foldingathome.org/project?p="
                   +project+"\">"+project+"</a>"
        if status != "idle":
            htmlFile.write("      <tr><td>project</td><td>"+project+"</td></tr>\n")
            if progress != "N/A":
                htmlFile.write("      <tr><td>progress</td><td>"+progress+"</td></tr>\n")
                htmlFile.write("      <tr><td>time</td><td>"+lastTime+" UTC</td></tr>\n")
            if timeLeft != "N/A":
                timeLeft = int(timeLeft)
                hours    = timeLeft // 3600
                timeLeft = timeLeft - 3600 * hours
                minutes  = timeLeft // 60
                timeLeft = str(hours)+"h "+str(minutes)+"m"
                htmlFile.write("      <tr><td>time left</td><td>"+timeLeft+"</td></tr>\n")
        htmlFile.write("      <tr><td>CPU</td><td>"+temp+"&deg;C</td></tr>\n")
    htmlFile.write("      <tr><td> </td><td> </td></tr>\n")

htmlFile.write("    </table>\n")

# time stamp

dateStamp,timeStamp = getDateTimeStamp()
htmlFile.write("    <hr>\n")
htmlFile.write("    <table>\n")
htmlFile.write("    <tr><td>"+dateStamp+" </td><td>"+timeStamp+"</td></tr>\n")
htmlFile.write("    </table>\n")

# append HTML tail and close HTML report

tailFile = open("html/tail.html","r")
tail     = tailFile.read()
htmlFile.write(tail)

htmlFile.close()
```

upload.expect

```
#!/usr/bin/env expect

#
# upload.expect - uploads HTML report to web server
#
#                  Reads user, host, password and remote path from file webftp.login
#

set home [file normalize [file dirname [info script]]]

# read user, host, password and path
source [file join $home webftp.login]

set address ${user}@${host}
spawn sftp $address
expect "${address}'s password:"
send "${password}\n"
expect "sftp>"
send "cd ${path}\n"
expect "sftp>"
send "put html/index.html\n"
expect "sftp>"
send "exit\n"
```

html/head.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>F@RasPi</title>
    <style type = "text/css">
      <!--
      HTML  {font-size: calc(1.5em + 0.1vw) }
      P     {font-size: calc(1.5em + 0.1vw) }
      TABLE {font-size: calc(1.5em + 0.1vw);font-family:"Courier New", Courier, fixed}
      TT    {font-size: calc(1.5em + 0.1vw);font-family:"Courier New", Courier, fixed}
      -->
    </style>
  </head>
  <body>
    <h1>F@RasPi</h1>
    <hr>
```

html/tail.html (trimmed to minimum)

```
  </body>
</html>
```

nodes/*(node)*/node.url

```
https://en.wikipedia.org/wiki/Stefan_Banach
```

This is just an example. Provide any URL, which is understood by web browsers.

About the Author

Born in Augsburg 1968, grew up in Oberhausen. Still living there. Studied mathematics and later worked as research associate at the University of Duisburg. Working at Siemens Energy AG and its predecessors since 1998, designing engineering software. You can contact me via email to fah@anne-emscher.net.

Further Reading and Links

This project's home page: http://www.anne-emscher.net/fah/

Folding@Home project web page: http://www.foldingathome.org

Raspberry Pi project web page: https://www.raspberrypi.org

Download Raspberry Pi Imager: https://www.raspberrypi.org/software/

ICY BOX casing: https://www.raidsonic.de/products/accessories/raspberry/index_en.php.php?we_objectID=5885

fantec multi-charger https://www.fantec.de/produkte/smartphone-zubehoer/ladegeraete-adapter/produkt/details/artikel/1954_fantec_qc3_a51/

Configurable acrylic dust casing: https://www.sora.de/massanfertigungen/acrylhauben/124/plattenspielerhaube?c=164

Heathcote, P.M. (2020): Learning to Program in Python, reprint of 1st ed., Tolpuddle: PG Online