MPC Laboratories

donn

Last updated : May, 2022

More documents are available at github.zhaw.ch

Table of contents

Chapter 1

01 - Setup Laboratory

1.1 1. Introduction

In this exercise we want to setup a computing platform environment such that we are able to code programs associated with various laboratories and run them on the computing platform. A number of platforms are in use for the module and each has its own set-up peculiarities. The list of relevant/available computing platforms is in section 3.2.

No points will be awarded for completion of this laboratory exercise. You can do this exercise in groups. One laboratory period (1.5 hrs.) is considered adequate for this exercise.

1.2 2. Learning Aims

There are no learning aims associated with this laboratory exercise.

1.3 3. Task: Setup a Computing Platform

1.3.1 3.1. Prerequisites

1. MPC-device
    1. uSD card with at least 8GB capacity (included with the board)
    2. u96 power supply
    3. microUSB cable
2. Personal Laptop
    1. SSH client installed (ssh, PuTTY, . . .)
    2. Byte level copy software (dd, cat, BalenaEtcher)
    3. Decompression utility such as (unzip, 7-zip, zip, . . .)
    4. Webbroser (Firefox, Opera, Chrome, . . .)

Basic knowledge of command line usage in Linux is expected. Tutorials are available online, e.g. BASH-Grundkurs, Basic commands, or similar.

1.3.2 3.2. Computing Platform Specific Instructions

Ultra96-V2

Chapter 2

02 - Process, Threads, Task Decomposition

2.1 1. Introduction

In this laboratory we wish to explore the operating system (OS) primitives available to hand-craft a parallel program. Operating Systems can be categorised into multi-threading and multi-process operating systems and both will tend to offer some form of encapsulation of both and scheduling of both. This can often be leveraged to facilitate distribution of processes and/or threads across multiple executing cores. It is this domain we wish to explore with this laboratory.

We shall start by understanding the OS – in this case Linux - interface to threads and processes. Then we convert a piece of code for a single-core processor to be able to run on multiple cores.

There are three lab sessions dedicated to this laboratory. Group work is recommended.

2.2 2. Learning Aims

1. The difference between a process and a thread and their resource-allocations
2. The articulation of that difference in a rich OS like Linux
3. The POSIX system-calls to instantiate threads and processes, specifically under Linux.
4. The use of the task decomposition pattern on a computationally expensive algorithm
5. Pining threads and processes to a core
6. The interaction of threads and processes with the scheduler
7. The run-time behaviour of threads and processes

2.3 3. Task 1: Parallelisation with Threads

Recommended ET, optional for IT

2.3.1 3.1 Part 1: Threads, understanding, instantiation, manipulation, visualisation

In this section we shall be looking at creating a multi-threading program and the effects of processor scheduling, if any. Inspect the code in the file **L2P1_threads.c** and answer the following questions

1. Predict what it does.
2. What is necessary to integrate threading in a piece of c-code? (hint – don't forget to check the makefile)
3. Build the code and let it run (`make ...  ./out.e`)
4. Does it do what you predicted?
5. How many units of execution are run?
6. You see the print of "Thread 1" and "Thread 2" in seeming random fashion – this is correct behaviour as the threads are scheduled by the Linux scheduler. What do you learn from this behaviour?
   Uncomment the line
   `selectCPU(0)`
7. What does `selectCPU(0)` do?
8. Any behavioural change noticeable?
9. Do you think there is a temporal difference?
10. Why is a separate variable required to identify thread number?
    We posit that pthreads under Linux are kernel level threads, that is the kernel is responsible for scheduling the threads. We also posit that the process is the unit of resource ownership and a unit of scheduling, and that a thread is "only" a unit of scheduling. This implies that the threads run under the same Process ID (pid) and that each thread has its own Thread ID (tid).
    The process ID can be gotten via the call
    `pid_t pid = getpid();`
    and the thread ID can be gained by the call
    `pid_t tid = syscall(SYS_gettid)`
11. Prove the above posits by using these calls and printing out the results in appropriate positions in the code. Draw a call graph. What is the value of the TID if there is no thread running?

2.3.2 3.2 Part 2: Task Decomposition using Threads

Recommended ET & IT

We strongly recommend you keep the code you write for reference in the next section and, especially, for further use in the next labs

Examine the naïve Sobel implementation. Download it onto your board and let it run. It takes a image.png file and generates a edge_image.png file. You can upload the image file to your PC to view it. You should be able to substitute any (8-bit) colour .png file as the input. It's not well tested – don't overdo it.

Your task is to convert the code to a task-decomposition based architecture using threads – before you do so consider the following

1. How many units of execution do you really need? How many would be useful?
2. Draw the CFG of your solution
3. Where, when and why are you going to fork?
4. What do you intuit about the performance – the cost of thread creation versus the cost of the function?

https://computing.llnl.gov/tutorials/pthreads/ is an excellent resource.

2.4 4. Task 2: Parallelisation with Processes

2.4.1 4.1 Processes, understanding, instantiation, manipulation, visualisation

Recommended ET

Check out the code for part 1. Examine the code, download it, let it run, look at the output.

1. How is the code organised?
2. What do you notice on the output?
3. What effect are you looking at? Shared memory is one solution to the problem we shall handle this in theory later but by adding the following code before the fork

```
// pointer to shared memory
void *shmr;
// file handler and length (size) of shared memory - you need to allocate a value to len
int fd, len
// open a shared memory handle
fd = shm_open("/shared_memory", O_CREAT | O_RDWR, 0700);
// allocate bytes
ftruncate(fd, len);
// map into memory map of process
shmr = mmap(NULL, len, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
```

   you can treat the memory pointed to by `shmr` as if it were memory allocated by `malloc()`. This method only works for related processes. Note the position of the -lrt switch in the linker command (makefile). Note the child process inherits the handlers and shared memory region.
4. What is the output now?
5. What do you have to ensure in a read-write scenario where the entire array can be written to by the processes?
6. Printout the PIDs of the two processes

2.4.2 4.2 Task Decomposition using Processes

Recommended ET & IT

Implement the sobel algorithm as task decomposition implemented as a multi-process program. If you haven't done the first section make sure your output looks exactly like those generated by the base code. If you have done the first part, there should be no difference between the output of this task and the output of your code for thread-based task-decomposition.

2.5 5. Task 3: Time, Scheduling, Pinning

Recommended ET & IT

2.5.1 5.1 Cores

Find out what the system calls

```
get_nprocs_conf();
get_nprocs();
```

do and what your system tells you when you use them. Is there any difference in the information when you use the (keyboard) system call

```
lscpu
```

?

2.5.2 5.2 Time

Place the system calls, read the `man` pages first,

```
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &start);
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &stop);
```

initalising all the variables etc. in appropriate positions in the code to measure the runtime of the process with and without threading and the process-based task-decomposition code you have just written. Let the code run in a loop ~ten times.

7. What does the `CLOCK_PROCESS_CPUTIME_ID` flag mean?
8. What do you notice about the times?

2.5.3 5.3 Scheduling

What does the following system call do?

```
int policy = sched_getscheduler(0);
```

9. Put it into your code – what scheduler is being used?
10. What system call is used to set the scheduler? Write the code to set a specific scheduler and a task priority use it to change the scheduler from whatever to `SCHED_FIFO`
11. Set the scheduler at the beginning of the main process – what scheduler do the spawned threads and processes use, why?
12. Do the execution times of your program change?

2.5.4 5.3 Pinning

Examine the code in the file `setAffinity.c` use the calls to examine the current affinity at the start of the code and to change the affinity to set the core for specific processes/threads.

13. Do the runtimes times of your program change? Add a worker thread performing an endless loop. Open another terminal and run `htop` and then in the first terminal let your program run.
14. Which core is fully occupied and which threads are still running?
15. Why is one of your threads running most of the time and the other sleeping? You can kill the thread by opening a third terminal and issuing the

    ```
    kill <thread_id>
    ```

    command.
16. If you kill the thread with the higher TID, what happens?
17. If you kill the thread with the lower TID what happens?

2.6 6. Evaluation

This laboratory is not evaluated - you will almost certainly need it to be able to complete the marked laboratories. I **welcome** in-depth discussions on the contents of the lab and your solutions.

2.7 7. Version

Version	Date	By	Comments	Class Level	Module
V1.1	02.2022	donn	Added some sections	6S	MPC
V1.0	02.2021	donn	First version	6S	MPC

Chapter 3

03 - Data Decomposition

3.1 1. Introduction

In this laboratory we wish to explore the application of the data-decomposition pattern on the Sobel edge-detection problem encountered in the previous lab. You will need to decide on a geometric decomposition of the image and then implement the parallelisation of the Sobel algorithm.

There are three lab sessions dedicated to this laboratory. The laboratory will be marked. Group work is highly recommended.

3.2 2. Learning Aims

1. The difference between a process and a thread and their resource-allocations
2. The articulation of that difference in a rich OS like Linux
3. The POSIX system-calls to instantiate threads and processes, specifically under Linux.
4. The use of the task decomposition pattern on a computationally expensive algorithm
5. Pining threads and processes to a core
6. The interaction of threads and processes with the scheduler
7. The run-time behaviour of threads and processes

3.3 3. Task 1: Parallelisation with Threads

Recommended ET & IT

Convert the Sobel algorithm base code into a data-decomposition based program.

We have supplied you with a couple of base utilities, namely code for processing pixels - which you can decide upon yourselves and combine these routines to create, manage and process chunks of data. There is a main program available but you need to ask yourself does this conform to the pattern I require? Change at will.

In its current instantiation `process_chunk()` processes the entire picture into black and white by pixel. It also illustrates the max indices and general usage of the helper functions. Feel free to create your own.

You task is to decide what geometrical pattern you want to use and why, the reasoning behind the size of the chunks, the implementation pattern etc. and then implement it and characterise it

Be prepared to explain the following:

---

1. What was the process of deciding on the chunks?
2. How did you handle boundary data?
3. What other design constraints did you consider?
4. Would you do it that way again?

3.4 4. Evaluation

This laboratory is evaluated on the basis of the discussions with you on your code/solution. You can receive a maximum of 5 marks for this laboratory.

A perfect solution is good. An imperfect solution is acceptable. If your solution doesn't work, then it an identification of the problems and mistakes made will be accepted for grading.

I **welcome** in-depth discussions on the contents of the lab and your solutions **during** the implementation process.

3.5 5. Version

Version	Date	By	Comments	Class Level	Module
V1.1	02.2022	donn	Added some sections	6S	MPC
V1.0	02.2021	donn	First version	6S	MPC

Chapter 4

04 - OpenMP

4.1 1. Introduction

In this laboratory we wish to explore the application of the OpenMP framework. We shall be using the results of the data decomposition lab (lab_03) as a code base.

There is one lab session dedicated to this laboratory. The laboratory will be marked. Group work is highly recommended.

4.2 2. Learning Aims

1. The basics of openMP
2. The conceptual similarities of OpenMP as compared to the direct use of processes and threads as units of execution (UE)
3. The (syntatic) simplifications that OpenMP provides as opposed to direct use of threads and processes as units of execution.
4. The operational similarities of OpenMP as compared to the direct use of processes and threads

---

4.3 3. Task 1: Application of OpenMP

Recommended ET & IT

Convert your data decomposition program into an openMP variant.

You must be careful to ensure that your program acttually works in parallel. Use htop, for instance, to show this.

Be prepared to explain the following:

1. What was the process of converting to OpenMP?
2. How is boundary data handled?
3. What other design constraints did you consider?
4. Would you do it that way again?

4.4 4. Evaluation

This laboratory is evaluated on the basis of the discussions with you on your code/solution. You can receive a maximum of 5 marks for this laboratory.

A perfect solution is good. An imperfect solution is acceptable. If your solution doesn't work, then it an identification of the problems and mistakes made will be accepted for grading.

I **welcome** in-depth discussions on the contents of the lab and your solutions **during** the implementation process.

4.5 5. Version

Version	Date	By	Comments	Class Level	Module
V1.1	02.2022	donn	Added some sections	6S	MPC
V1.0	02.2021	donn	First version	6S	MPC

Chapter 5

05 - Incremental computation of moments

5.1 1. Introduction

In this exercise we want to begin getting a working knowledge of NEON SIMD inmstructions. We do this using a simple data reduction example, an online-calculation of moments of a statistical distribution.

No points will be awarded for completion of this laboratory exercise. You can do this exercise in groups. One laboratory period (1.5 hrs.) is considered adequate for this exercise

---

5.2 2. Learning Aims

1. The format and usage of ARM-NEON intrinsic instructions
2. Online statistical calculations

5.3 3. Task 1: Compute the Third Moment

Recommended ET & IT

5.3.1 Moments

Central moments are widely used in descriptive statistics. Standard approaches for computing them require two passes over the data. In applications were were data is streamed incremental results are needed after each new value is observed.

The first moment (mean) is given by

$$M_1 = M_1^A + \frac{(y - M_1^A)}{N}$$

The second moment (variance) of a distrubtion is given by

$$M_2 = M_2^A + \frac{N-1}{N}(y - M_1^A)^2$$

The third moment (skew) is given by

$$M_3 = M_3^A + \left[\frac{N-1}{(-N)^3} + \left(\frac{N-1}{N}\right)^3\right] \cdot (y - M_1^A)^3 + M_2^A \cdot \frac{(M_1^A - y)}{N}$$

The nth moment (kurtosis ...) of a distrubtion is given by

$$M_p = M_p^A + \left[\frac{N-1}{(-N)^p} + \left(\frac{N-1}{N}\right)^p\right] \cdot (y - M_1^A)^p + \sum_{k=1}^{p-2} \binom{p}{k} M_{p-k}^A \cdot \left(\frac{M_1^A - y}{N}\right)^k$$

5.3.2 Task

The example generates samples with a given Gaussian distribution. For each sample the new moments are calculated normally (`compute_moments_neon`) and with SIMD (`compute_moments_neon`). The `compute_moments_neon` function is incomplete as the third moment is not yet implemented.

Examine the given source code and answer the following questions:

1. How many dataset are calculated in parallel when using SIMD?
2. Is the number of datasets given when using SIMD (check the ARM Neon reference)?
3. The example can be build using `make'
4. Currently only the first two moments are calculated
5. Compare the calculation of the first two moments to the implementation in `compute_moments` and identify the used intrinsics.
6. Write the calculation of the third moment in the `computes_moments_neon` function. You can use a `compute_moments` function as a blueprint. Don not forget to check if you have to add load and store instruction at the start or the end of the function.
7. Compare your results of the two implementations. Are there differences?
8. Is there any run-time improvement to be expected by using openMP?

5.4 4. Evaluation

This laboratory is not evaluated Nevertheless I **welcome** in-depth discussions on the contents of the lab and your solutions **during** the implementation process.

5.5 5. References

https://developer.arm.com/architectures/instruction-sets/simd-isas/neon/intrinsics

5.6 6. Version

Version	Date	By	Comments	Class Level	Module
V1.1	04.2022	donn	Added some sections	6S	MPC
V1.0	02.2021	donn	First version	6S	MPC

Chapter 6

06 - Real-Time with SIMD

6.1 Introduction:

In this laboratory we wish to explore optimising designs using SMID instructions. We do this by getting you to implement a naïve implementation of a sound FIR-filter. As you should have completed the previous lab the task to master becomes one of streamlining data access.

This lab is marked and will score a maximum of 5 points. You have three lab sessions for this task (4.5 hours.)

6.1.1 Prerequisites

This lab requires additional software on the Ultra96 plattform and so you must install dependencies:

6.1.1.1 Directly install the dependencies (recommended)

Using bash:

```
sudo apt install alsa-utils portaudio19-dev libsndfile1-dev linux-tools-common valgrind␣
↪libne10-dev
```

6.1.1.2 Offline Installation

If you have not setup Internet access on the u96-ultra you can attempt an offline installation. For this a script is provided. Important, this method can not ensure a functioning installation as dependencies may not be installed. If this happens the method above must be used.

Using bash

```
cd ./offline_install
chmod +x ./offline_install.sh
./offline_install.sh
```

6.2 Learning aims

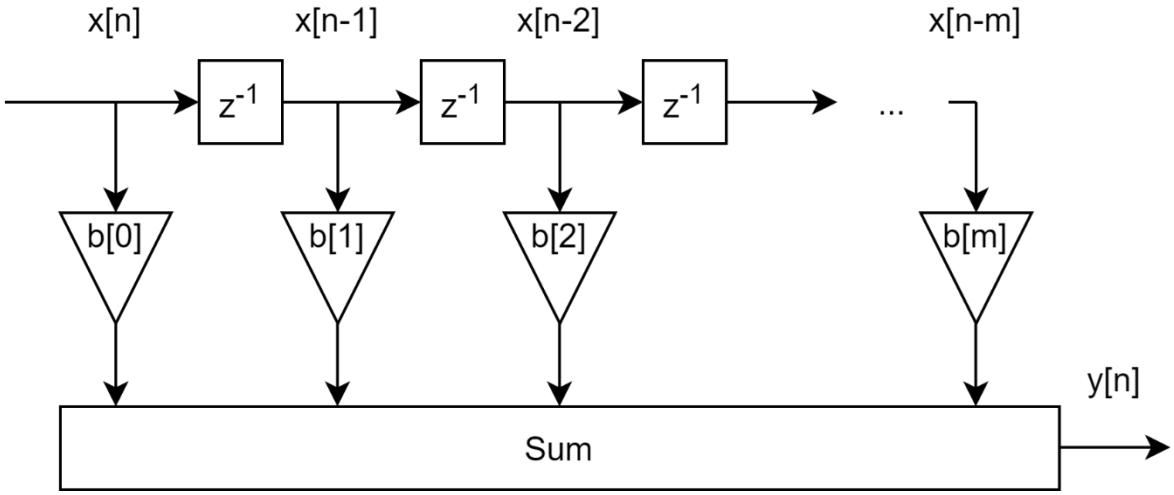
- Practical design using SIMD instructions
- Typical real-world, real-time signal processing problem

6.3 Task

In the laboratory an audio file is read, filtered in the time domain to apply an echo effect, and stored in a new audio file. Students which have attended a digital signal processing class will know that there are better options to achieve this, but these methods are out of scope for this laboratory.

6.3.1 Introduction

In this application a FIR filter is used to change the sound as if it were recorded in another room, to do this the impulse response of the room is convoluted with the input sound. By changing the impulse response, the audio effect can be modified. The block diagram below illustrates the computational process. The weights **b** are the samples of the impulse response used, while **x[n]** represents the input samples. The delay line (represented by the **z-1** elements) is essentially a FIFO buffer, in which the newest sample is **x[n]** and the oldest sample is **x[n-m]**.



The process consists of multiplications and additions, and each output sample **y[n]** requires **m+1** multiplications between input samples **x** and weights **b**, as well as **m+1** additions.

$$y[n] = \sum_{i=0}^{N} b_i \cdot x[n-i]$$

6.4 FIR Filter Implementation

6.4.1 Audio file processing

In order to access audio files, the sound file library is used (available in the libsndfile1-dev package). All interaction with this library are provided with the base code.

Depending on the implementation the runtime of the program can be relatively long. It is recommended to limit the number of input samples to 100'000 samples using the `#define LENGTH_LIMIT` and use the 'test_input.wav'.

The correct working of the filter can easily be checked by ear as wrong implementations tend to sound wrong and/or noisy. For the same reason it is advised to reverse the filter coefficients, in this way the effects of the lower-order coefficients are better taken into account.

6.4.2 Task 1: Data Preparation

When working with an application like this efficient data transport is required. Digital signal processors have hardware architectural constructs to move data efficiently from memory to registers and back again, including acceleration of multiply and accumulate (MAC) operations. In our processors we do not have this luxury and so must ensure that the SIMD instructions are efficiently fed with data by carefully considering the code.

Ensuring that data transport is efficient in code bears similarity to efficient programming in assembler. One starts at an instruction, a SIMD one in this case, and work backwards to make sure the data has the least possible number of formatting and alignment steps before being put into the operand register. As we work in loops the loops have a minimum number of instructions which makes the loops efficient and hence, hopefully, accelerate the computation time.
Further Hints:

- You can use `#define FILTERSIZE_MOD` to ensure that the length of your filter is dividable by the chosen number.
- You can use `#define DELAYLINE_OVERSIZE` to extend the size of your delay line beyond the filter size.
- During the data preparation phase, SIMD or OpenMP are not necessary as this phase only takes up a small fraction of the total runtime.
-

6.4.3 Task 2: FIR Filter

Implement your FIR filter in the `filter_neon` function. The given code frame already includes a variable for the `delayline (float *reg)` and the current sample position `(uint32_t position)` which you can use in your filter function. Your filter implementation should use both SIMD and OpenMP.

Hints:

- To use SIMD and OpenMP in an efficient manner a nested loop can be advisable.

6.5 Evaluation

This laboratory is evaluated on the basis of the discussions with you on your code/solution. You can receive a maximum of 5 marks for this laboratory.

A perfect solution is good. An imperfect solution is acceptable. If your solution doesn't work or is not particularly performant, then it an identification of the problems and mistakes made will be accepted for grading.

I welcome in-depth discussions on the contents of the lab and your solutions during the implementation process.

6.6 Appendix

6.6.1 References

http://www.portaudio.com/

https://libsndfile.github.io/libsndfile/

https://developer.arm.com/architectures/instruction-sets/simd-isas/neon/intrinsics

Chapter 7

07a - Cluster Computing

7.1 1. Introduction

Message Passing Interface (MPI) really comes into its own in scientific computing. In cluster systems using tightly bound multiple CPUs and/or GPUs or in networked loosely bound computing centres, MPI supports computation complexity and massively parallel systems. The usefulness of the MPI environment is that it is interface agnostic. It can run on a local multicore or on a high performance computing machine made up of distributed computing machines.

As a simple example, we want to calculate the value of PI in a floating-point approximation on a cluster of Raspberry Pi boards. Using such a cluster we can demonstrate MPI parallelisation and OpenMP parallelisation. A cluster of commercial computers networked together to form a massed array of computing power is known as a Beowulf cluster.

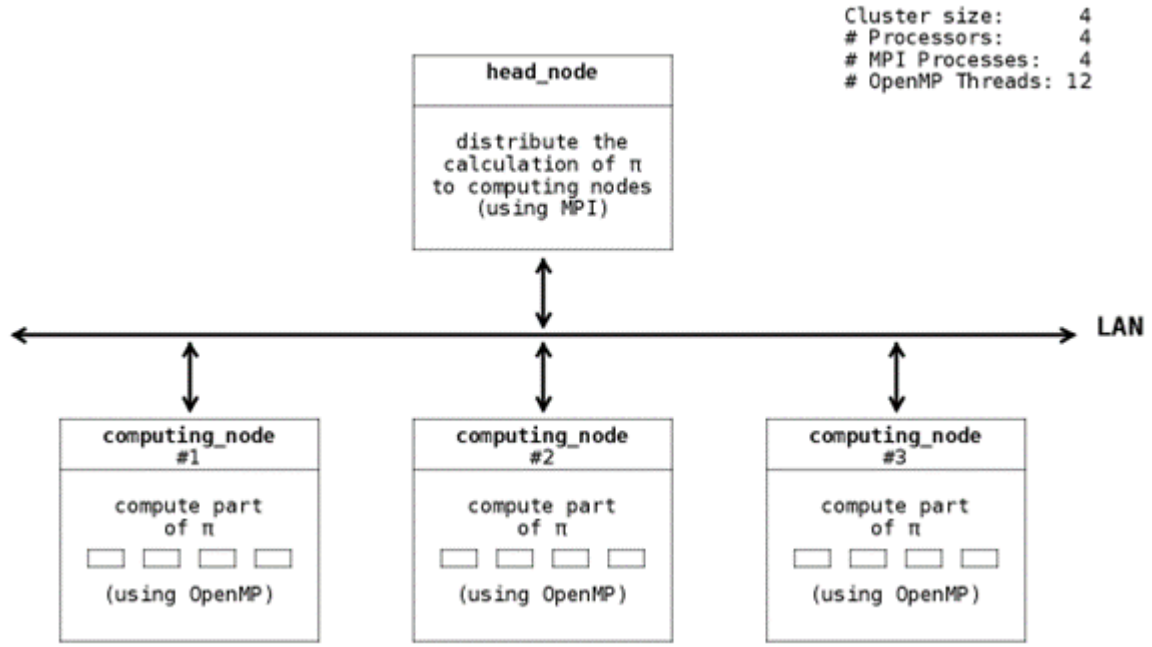
There are two lab sessions dedicated to this laboratory. The laboratory will be marked. Group work is highly recommended.

7.1.1 1.1 Calculating PI on a cluster

There are several reasons to calculate PI. One is to provide an example of scientific and parallelisable computing of mathematical numbers. A second reason is to find algorithms for efficient computing of numbers to large numbers of decimal places. A third reason is to serve as a benchmark for parallel programs and architectures.

For the purposes of this lab, we want to calculate PI on a computing cluster of four Raspberry Pi boards connected in a LAN. The calculation of PI will be distributed among three computing nodes and one node serving as coordinator. Therefore, we want to parallelise the execution on four individual processes – each on one Raspberry Pi board of the cluster. The four processes will be executed by one `mpirun` runtime.

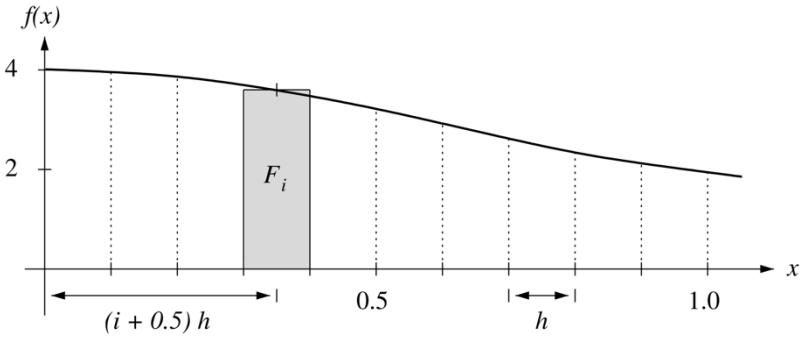
7.1.2 1.2 Cluster architecture



The number PI can be approximated using the following integral:

$$F = \int_a^b f(x)dx = \pi = \int_0^1 \frac{4}{1+x^2}dx$$

Numerically, the integral is determined by dividing the curve under the area f(x) into N rectangles of the area Fi and summing up the partial areas:



The following formula can be used implementing an approximation of PI:

$$\pi \approx F = \sum_{i=0}^{N-1} F_i = \sum_{i=0}^{N-1} h \cdot f(\,(i+0.5)\cdot h\,) \qquad \text{where} \quad h = \frac{1}{N}$$

7.2 2. Learning Aims

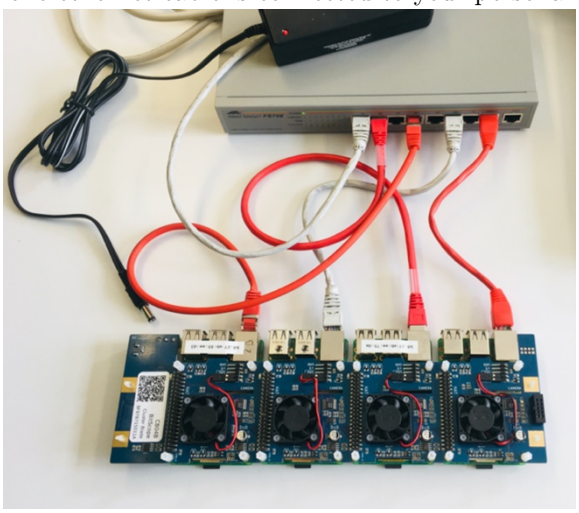
1. First steps with scientific high performance computing (HPC) computing
2. First steps with MPI
3. Handling a Beowulf cluster
4. Calcualtion-time differences on different architectures

7.3 3. Task

7.3.1 3.1 Setup the cluster of Raspberry Pi boards

Follow these steps:

1. Setup the cluster of Raspberry Pi boards according to the instructions given during the lab session. In the end, it should look like in the picture below (where one ethernet cable is connected to your personal laptop and the cluster blade not yet connected to power):



2. Flash the images provided on jupiter, the FTP server from previous labs, on your four SD cards (same procedure as with the U96 board). The image mpc_rpi_head_fs2022.img will be used once for the head node, the image mpc_rpi_child_fs2022.img will be used three times for each computing node.
3. After flashing the images on your SD cards, insert them into your Raspberry Pi boards and connect the cluster blade to the power source. The Raspberry Pi boards should now boot.

7.3.2 3.2 Setup the LAN

Now we will set up the network. The head node has the static IP *192.168.42.100* which was defined beforehand in the file */etc/network/interfaces*. The computing nodes will receive an IP address from the head node via DHCP. On the head node, a software package named *dnsmasq* was installed prior. This package runs as service which is started automatically during the boot process. This service includes a DHCP server which is responsible for the address distribution. Since every Raspberry Pi board is in the same subnet, we can retrieve the other IP addresses of each of the boards serving as computing nodes from the *leases* file of the DHCP server.

In a second step, we will give each node we want to communicate with in the network a dedicated name, i.e. every Raspberry Pi board is assigned a hostname. This is also helpful when we don't want to type in the nodes' IP addresses every time. The *hosts* file is provided by the device's operating system and used to map hostnames to IP addresses. **Note, we need to configure the hosts file of each board with the assigned names and mapping them to the correct IP addresses.**

Follow these steps:

1. Open a terminal and connect to your Raspberry Pi which serves as head node via SSH (Use the standard password "raspberry" to log in and answer the security question regarding authenticity with "yes"):

```
ssh pi@192.168.42.100
```

2. Retrieve the IP addresses of the other boards (and write them down somewhere):

```
cat dnsmasq.leases
```

3. Edit the host file /etc/hosts and add the hostname mapping to the IP addresses you just retreived:

```
#MPI Cluster Setup
192.168.42.100                head-node
"IP_of_Pi1"                 computing-node-0
"IP_of_Pi2"                 computing-node-1
"IP_of_Pi3"                 computing-node-2
```

These hostnames will be later used for mpirun.

4. Open three more terminals and connect to the other boards via SSH, using the retrieved IPs from above.
5. For each computing node, edit the host file /etc/hosts and add the hostname mapping to the IP addresses. Here we will need the mapping for the head node and the mapping for the current computing node.

```
#MPI Cluster Setup
 192.168.42.100               head-node
 "IP_of_PiX"                computing-node-X
```

7.3.3 3.3 Setting up SSH

Our nodes will be communicating over the network via SSH. We will establish passwordless SSH in order to enable an easier login and communication between the nodes.

Follow these steps:

On the head node:

1. Generate keys, in our case RSA keys:

```
ssh-keygen -t rsa
```

(In the prompt, just press enter, no need to specify a file or passphrase.)

2. Copy the keys to the other nodes' list of authorized_keys and connect to each computing node via ssh:

```
ssh-copy-id -i /home/pi/.ssh/id_rsa pi@"IP_of_PiX"
ssh@"IP_of_PiX"
exit
```

(Note, we are still on the head node, repeat each step for every computing node X.)

3. For the communication to work with the assigned hostnames, you need to execute the following commands for each computing node:

```
ssh computing-node-X
exit
```

On each of the computing nodes:

1. Generate keys, in our case RSA keys:

```
ssh-keygen -t rsa
```

(In the prompt, just press enter, no need to specify a file or passphrase.)

2. Copy the keys to the head node's list of authorized_keys and connect to the head node via ssh:

```
ssh-copy-id -i /home/pi/.ssh/id_rsa pi@192.168.42.100
ssh@192.168.42.100
exit
```

(Note, repeat these steps on every computing node X.)

7.3.4 3.4 Shared directory via NFS

We will use a shared directory via NFS in the head node to exchange data which each computing node mounts. The directory is called mpc_cloud and will be used for the MPI application that runs on every node on the cluster. This shared directory via NFS has already been set up on the head node. However, we need to check if it is in fact mounted on each computing node. It is possible, that a computing node was not able to mount due to the simultaneous booting of all Raspberry Pi boards in the cluster blade. In this case, the head node was not able to start the NFS-kernel-server in advance.

Follow these steps:

1. Check if mount was successful:

```
df -h
```

2. If you do not see 192.169.42.100:/home/pi/mpc_cloud in the list, execute the following command to mount:

```
sudo mount -t nfs 192.168.42.100:/home/pi/mpc_cloud          /home/pi/mpc_cloud
```

3. If successful, you should see it now in:

```
df -h
```

4. Check if you have the same contents in the folder mpc_cloud on the head node and on each of the computing nodes.

7.3.5 3.5 Calculate PI on a cluster of Pis

When we are all set up, we want to calculate PI on our cluster of Raspberry Pi boards. MPI is used for distributing the calculation among the boards and for the communication between them. OpenMP is used to parallelise the calculation within each calculating node.

Follow these steps:

1. On the head node, switch to the folder *mpc_cloud* and edit the *cluster_cpmputing_template.c* file according to the following steps.
2. Implement the calculation of PI in OpenMP with the formula above using the template given. **(See *calc_subsum_pi*(...) for instructions.)**
3. Fill in where necessary the MPI communication for the cluster network.
4. Compile as known and don't forget to add *-fopenmp*:

```
mpicc -o cluster_computing cluster_computing_template.c -fopenmp
```

5. Execute *mpirun* with the previously determined hostnames. Note the value calculated and the process time.
6. Change the number of rectangles used to calculate the total area beneath the curve. Use several different values. How does it affect the approximated value of PI and why?

7.4 4. Evaluation

This laboratory is evaluated on the basis of the discussions with you on your code/solution. You can receive a maximum of 5 marks for this laboratory.

A perfect solution is good.

I **welcome** in-depth discussions on the contents of the lab and your solutions **during** the implementation process.

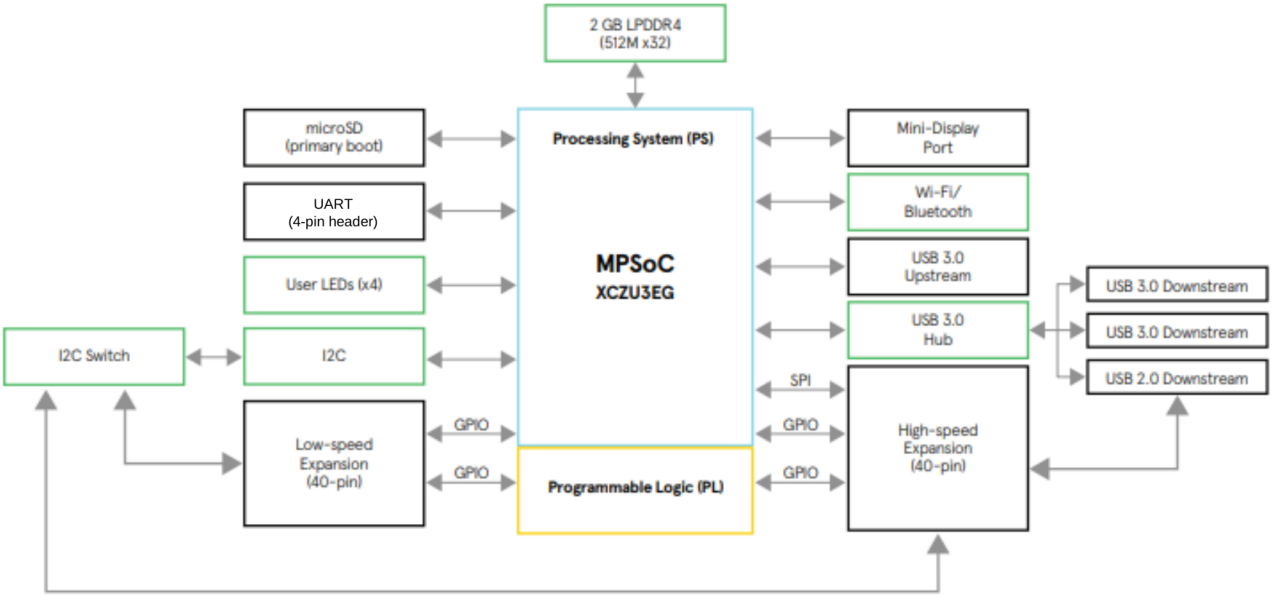
7.5 5. Version

Version	Date	By	Comments	Class Level	Module
V1.1	05.2022	donn	Ported to Pi-Blades	6S	MPC
V1.0	02.2021	donn	First version	6S	MPC

Chapter 8

Ultra96-V2

8.1 1. Introduction Ultra96-V2



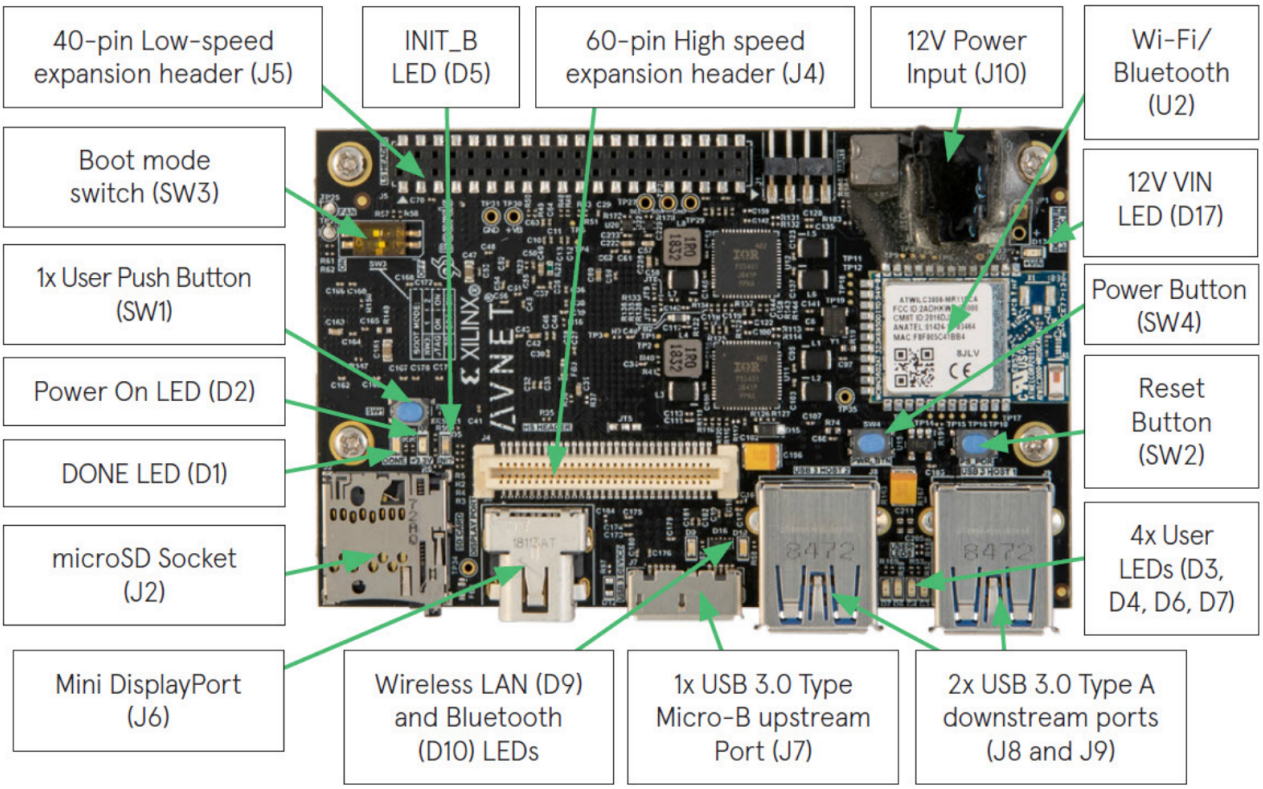
Component	Description
SoC	Xilinx Zynq UltraScale+ MPSoC ZU3EG A484
RAM	Micron 2 GB (512M x32) LPDDR4 Memory
SD-Flash	Delkin 16 GB microSD card + adapter
GUI	Downloadable SD card image for Linux OS with desktop environment
Networking	Microchip Wi-Fi / Bluetooth
Connections	1x USB 3.0 Type Micro-B upstream port 2x USB 3.0, 1x USB 2.0 Type A downstream ports
Expansion	40-pin 96Boards Low-speed expansion header 60-pin 96Boards High-speed expansion header
Graphics	Mini DisplayPort (MiniDP or mDP)

8.1.1 Xilinx Zynq UltraScale+ MPSoC

Xilinx Zynq UltraScale+ MPSoC ZU3EG

- **CPU**: Quad-core ARM **Cortex-A5**
- **CPU**: Dual-core ARM **Cortex-R5F, TCM**
- **GPU**: Mali-400 MP2
- **FPGA**: 154K Programmable Logic Cells
- 256KB On-Chip Memory w/ECC
- UART
- CAN
- USB 2.0
- I2C
- SPI
- 32b GPIO
- Real Time Clock
- WatchDog Timers
- Triple Timer Counter

8.1.2 Ultra96 Block Diagram



8.2 2. Setup

8.2.1 2.1 Prepare SD card

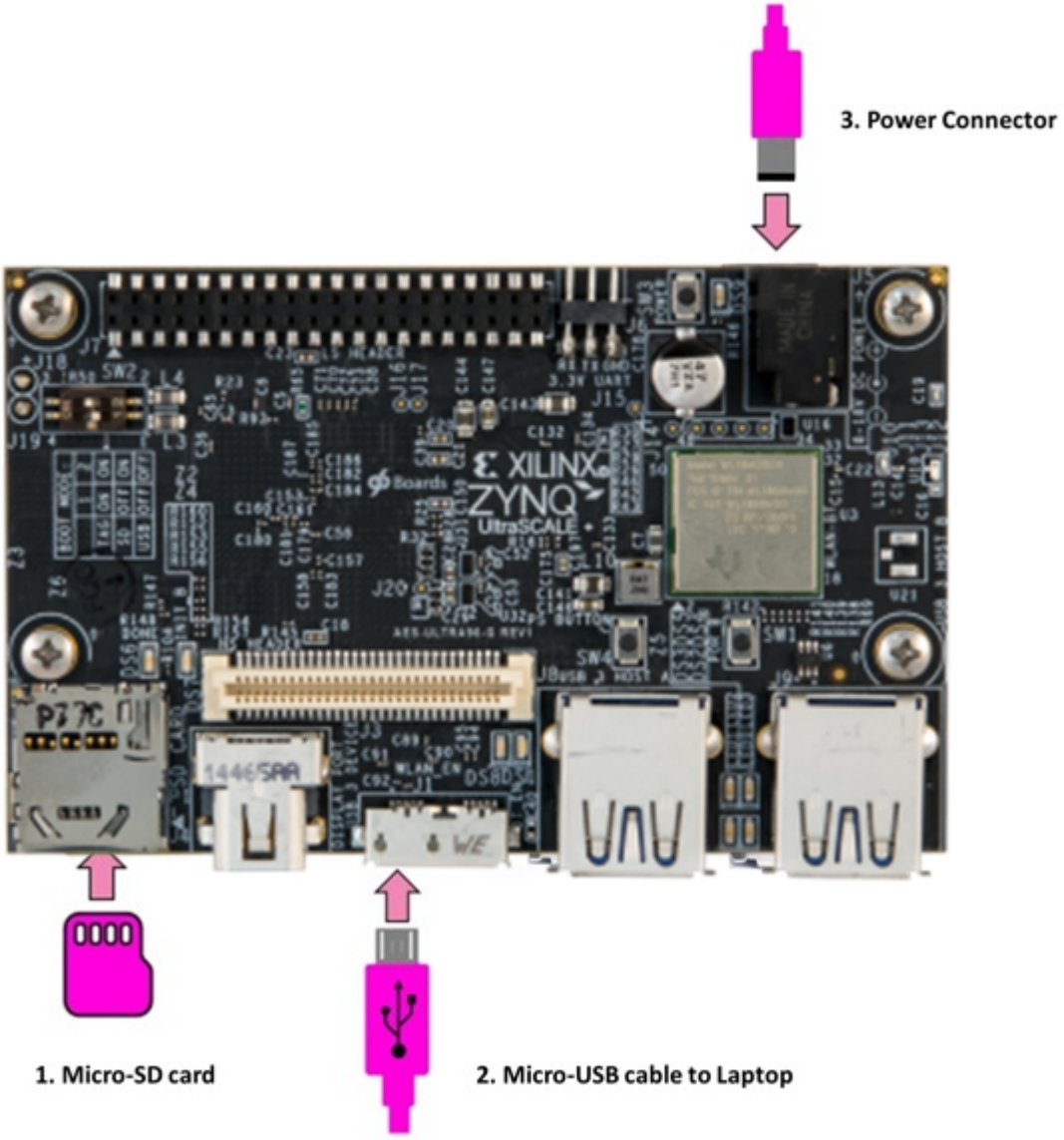
1. Download sd card image **ultra96v2_v2.6.0.img** from **ftp://jupiter-win.zhaw.ch/shared/** (using an FTP client or a browser, must be connected to ZHAW network) with the credentials:
   - Username: mpcro
   - Password: mpcro
2. Write image to the SD card, you can either use:
   - Etcher.io utility
   - **dd** on unix based systems:

```
# byte copy image to /dev/sdX
dd if=./ultra96v2_v2.6.0.img of=/dev/sdX bs=4M
# Bytecopy with progress bar
dd if=./ultra96v2_v2.6.0.img | pv -s 8G | dd of=/dev/mmcblk0 bs=4M
```

The image contains the PYNQ distribution for the Ultra96, it is an Ubuntu-based Linux OS.

8.2.2 2.2 Connect the Ultra96

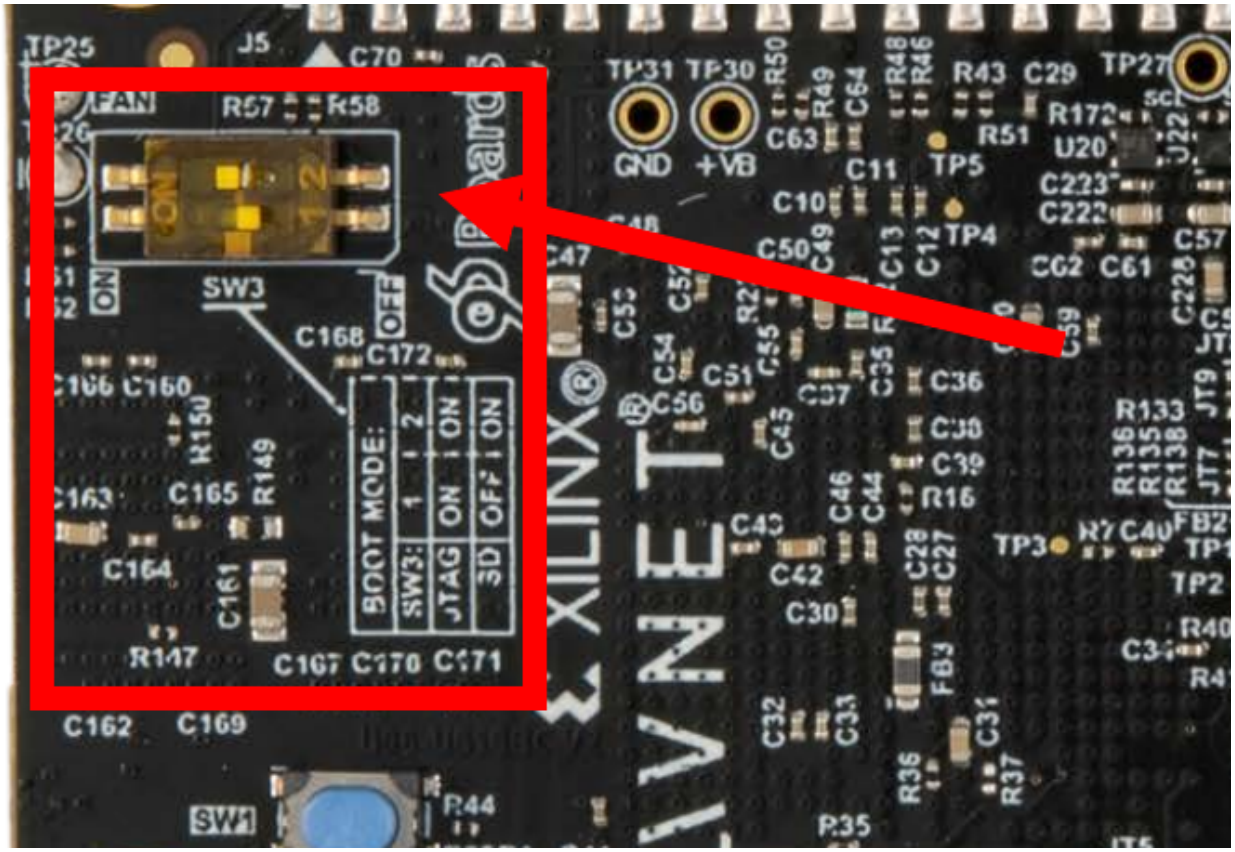
Plug in the cables as shown in the figure below.



8.2.3 2.3 Boot Linux from SD-card

The SoC boot mode is configured by the **SW3** DIP switch. Verify that the switch is set accordingly:

Switch	Position
1	OFF
2	ON



Insert SD-Card and boot by pressing the power-on switch. When done one user led blinks (Linux heartbeat).

8.2.4 2.4 Access the Ultra96-V2

The PYNQ image provides ethernet over USB and a serial connection. Access the board either via serial:

```
- 115200 baud
- 8 data bits
- 1 stop bit
- No Parity
- No Flow Control
```

Or using SSH with the credentials:

```
- Username: xilinx
- Password: xilinx
```

The IP address is either 192.168.3.1 (when using ethernet over USB), or can be found by issuing the `ip address` command in a console.

Configure the WiFi in order to have a connection to the internet by modifying the file */boot/boot.py* (e.g. by using `sudo nano /boot/boot.py`), add the following lines and insert your WiFi information.

```
from pynq.lib import Wifi

port = Wifi()
port.reset()
port.connect('your_ssid', 'your_password', auto=True)
```

To reload the configuration a reboot is required `sudo reboot`, after that the board should automatically connect to your network.

8.2.5 2.5 Access the Ultra96 files

We will use the samba share to edit the files and will compile programs using the console.
To connect using Windows, type `\\192.168.3.1\xilinx` in the address bar of the file explorer and access using the same credentials as above (xilinx:xilinx). Then you can navigate the folders in the home directory (*/home/xilinx/*) and edit files using a text editor or IDE.
For Linux use the URL `smb://192.168.3.1/xilinx` with your file manager.
A Jupyter Notebook is availabe at http://192.168.3.1 (password xilinx), from here Python development is possible.

8.3 3. First steps

8.3.1 3.1 Hello, World!

To compile programs we use the gcc compiler available in the distribution, in this section an example program is created and compiled. To automate compilation the make system is used by creating a *Makefile*.

1. Connect to the U96 via USB or WiFi
2. Open a terminal either via a serial connection or via SSH
3. Connect to the samba share to edit files with the host
4. Create a folder in the home directory and add the source file as *hello.c*, example code provided below
5. Add the Makefile, an example is provided below
6. In the console run `make hello`
7. Execute the program with `./hello`

8.3.1.1 Example code

```c
#include <stdio.h>

int main(){
        printf("Hello, World!\n");
        return 0;
}
```

8.3.1.2 Example Makefile

```
CC=gcc

.PHONY: hello

hello:
        $(CC) -Wall hello.c -o hello
```

Further reading:

- Makefiles GNU Make
- GCC GCC Doc

8.3.2 3.2 Useful programs

The running processes as well as the system usage can be shown with the command `htop`.
Information about the CPU can be printed with `lscpu`.
To install programs and libraries the *apt* package system is used:

- Update catalog of available programs: `sudo apt update`
- Install a program/library: `sudo apt install program`

For further information refer to APT Guide or the man page.

8.4 4. References

Ultra96-V2
PYNQ getting started