HEIG-VD | REDS Institut Reconfigurable and Embedded Digital Systems

Calcul Numérique et Accélération Matérielle (CNM)

Prof Marina Zapater
Assistant: Mehdi Akeddar

Introduction to the environment

lab00 (26.09.2024)

Objectives of the laboratory

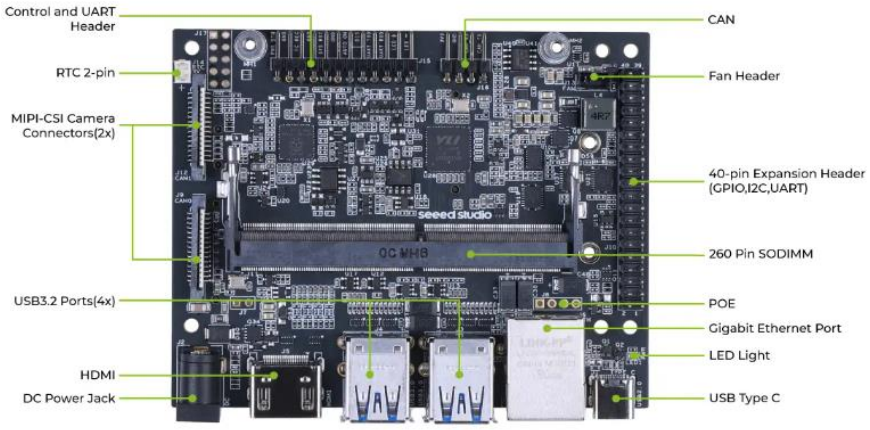
This is an introductory lab to the working environment for CNM laboratories. In this context, each student will have the opportunity to become familiar with the hardware, the development environment, and the basic tools that we will use in the laboratory.

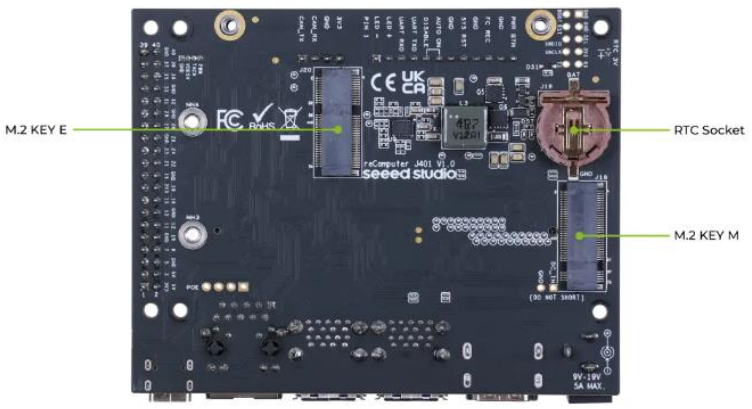
Laboratory validation

This lab will not be graded but we require you to write a README file with the answers to some parts of this lab.

Stage 1 – Working with the device

We will explain two ways to connect to the device so we can work with it. The device runs GNU/Linux including a desktop environment and an SSH server.



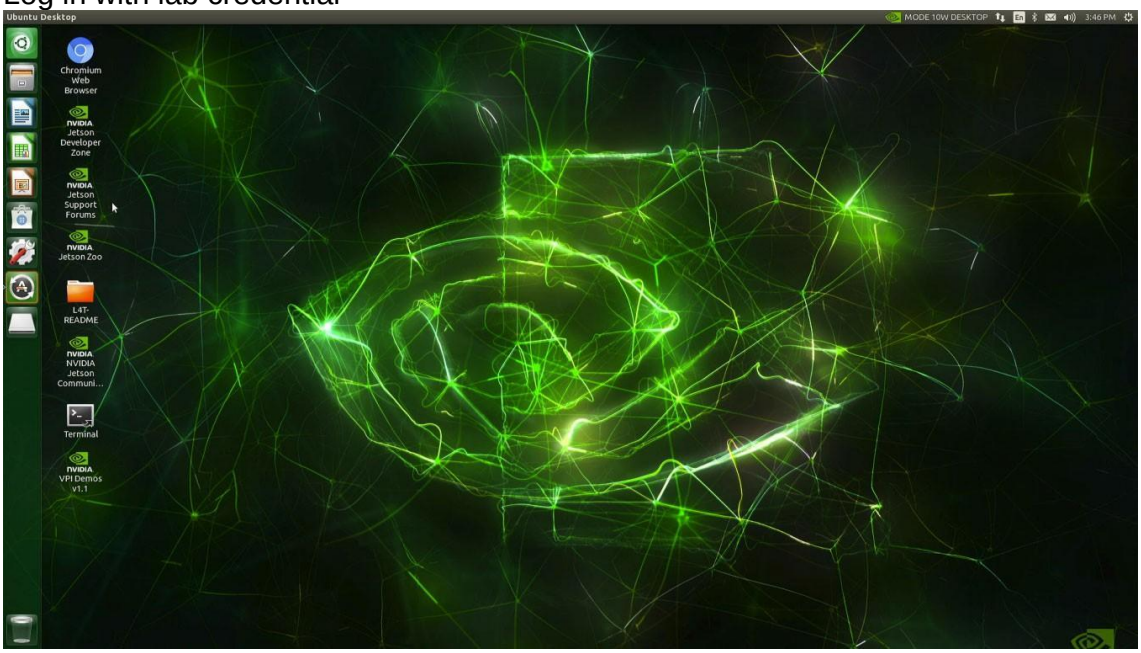


The device is flashed and pre-configured. We have created the following user for the lab:

- User: **cnm**
- Password: **cnm**

a) Physical connection. The device has multiple interfaces that allow us to use it as a regular computer.

1. Connect all peripherals (monitor, mouse, keyboard)
2. Connect DC power
3. Wait for boot-up
4. Log in with lab credential



b) **Network connection:** We connect to the device over SSH. The default IP address of the device is **192.168.55.1**

1. Connect to micro USB.
2. Connect the device's DC power.
3. Wait boot up.
4. Connect over SSH using lab credentials. To connect over SSH you will need an SSH client.

```
ssh cnm@192.168.55.1
```

Stage 2 – Device information

Once we can work with the device, we can use the terminal to get information about its hardware. CPU characteristics are relevant to the course. Use the command **lscpu** to get the following CPU information:

- CPU architecture.
- Number of cores.
- Number of threads per core.
- Max and min frequencies.
- Different cache sizes.

We already saw that the cache line size and the cache associativity are relevant. Find a way to obtain these two parameters for the L1d (for data) cache.

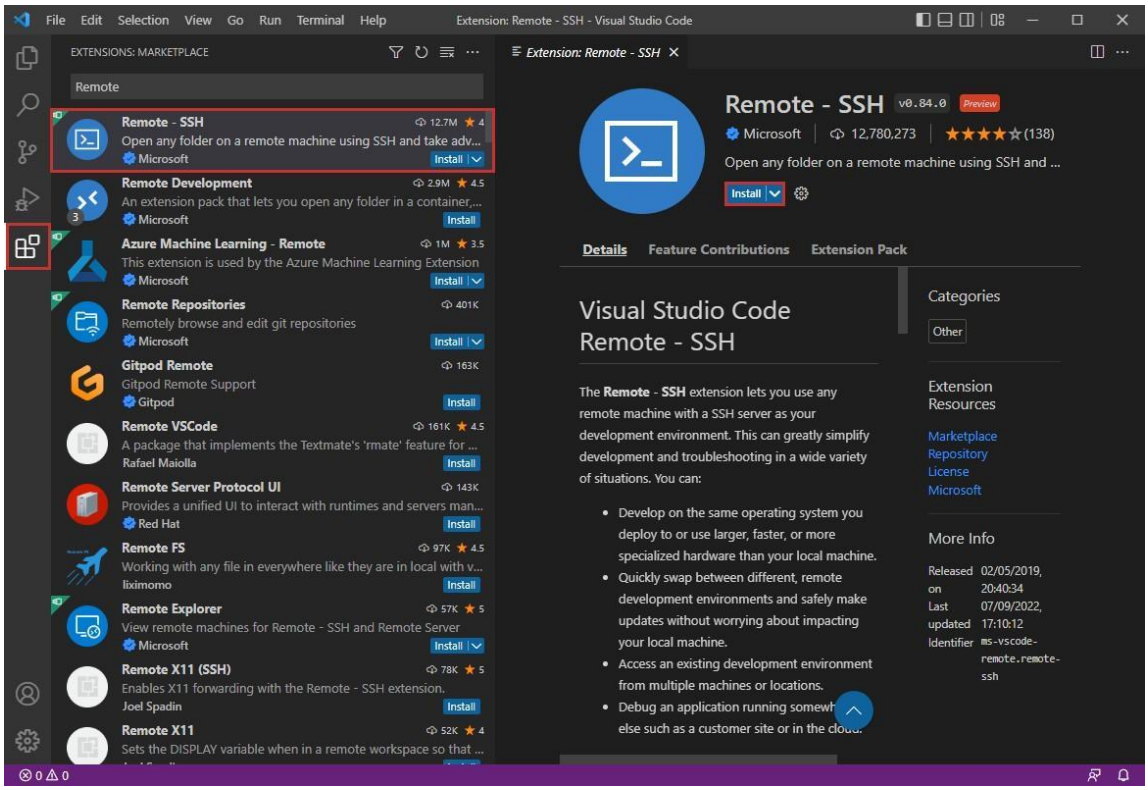
Note the obtained information obtained. You will be asked to fill-in this information on a README file and to push it to a git repository in Stage 4.

Stage 3 – Developing on the device

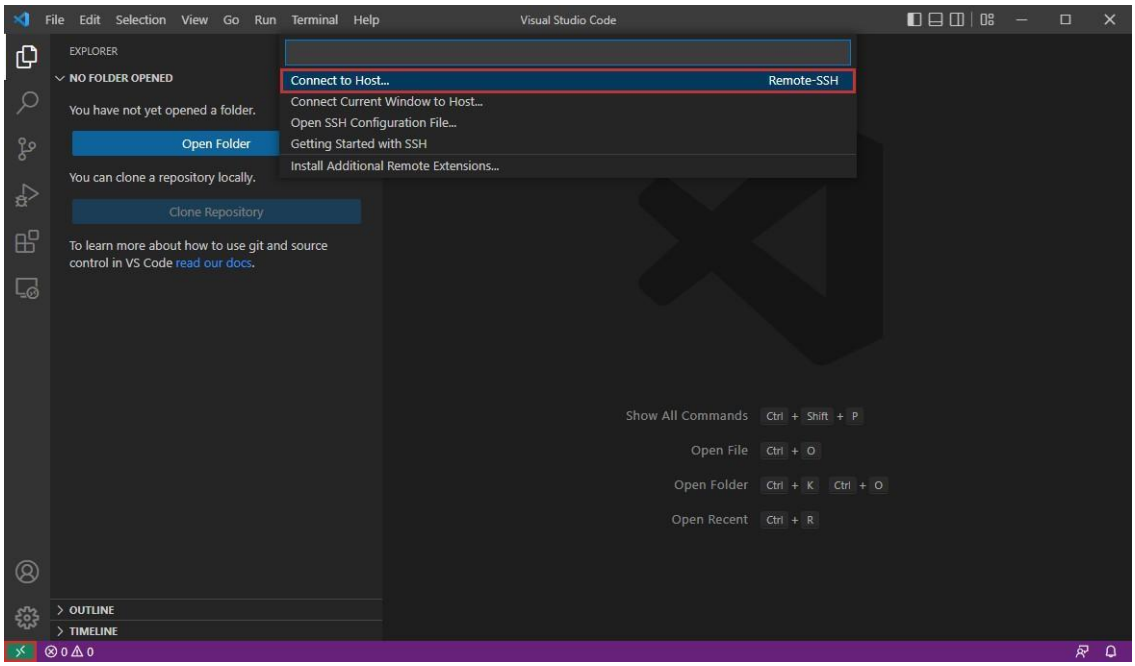
We use Visual Studio Code as IDE to develop on the device. It is pre-installed in the devices so if you have chosen to work directly in the device you just need to launch it. If you have chosen to work from your computer (network connection), you will need to install the remote extension.

a) Installing remote extension.

1) Go to the extension panel (left) and search for **remote**. Then install **Remote – SSH** extension

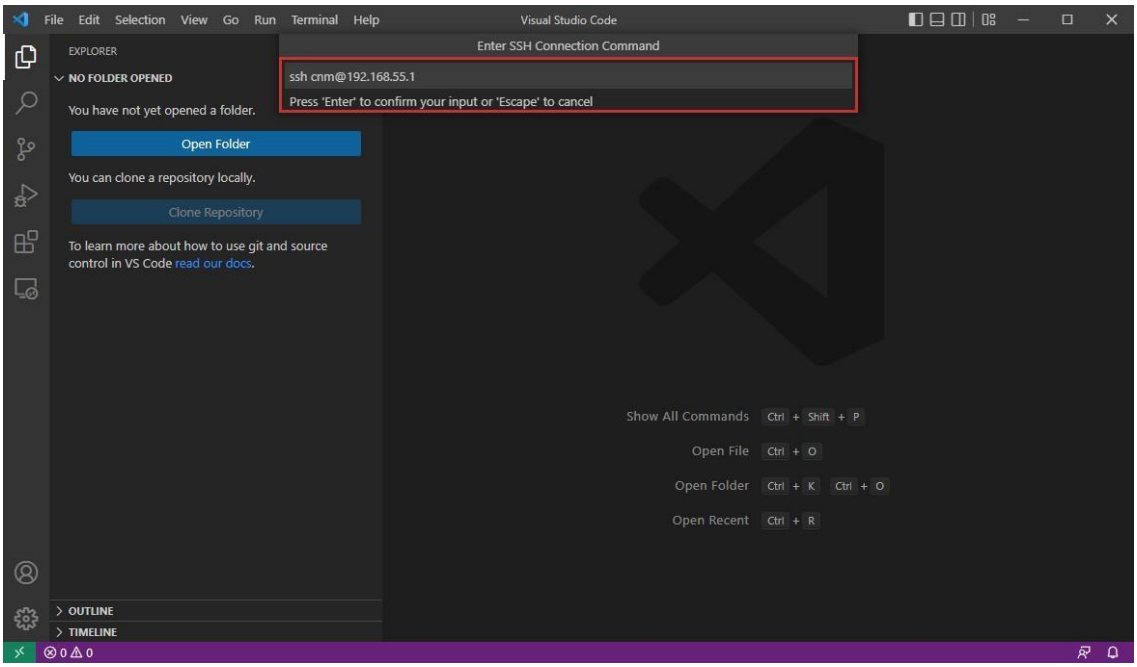


2) Then we need to connect to the device. Open the remote menu by clicking on the remote indicator of the status bar (green bottom left) or search **connect to host** in the command palette (`Ctrl+Shift+P`).

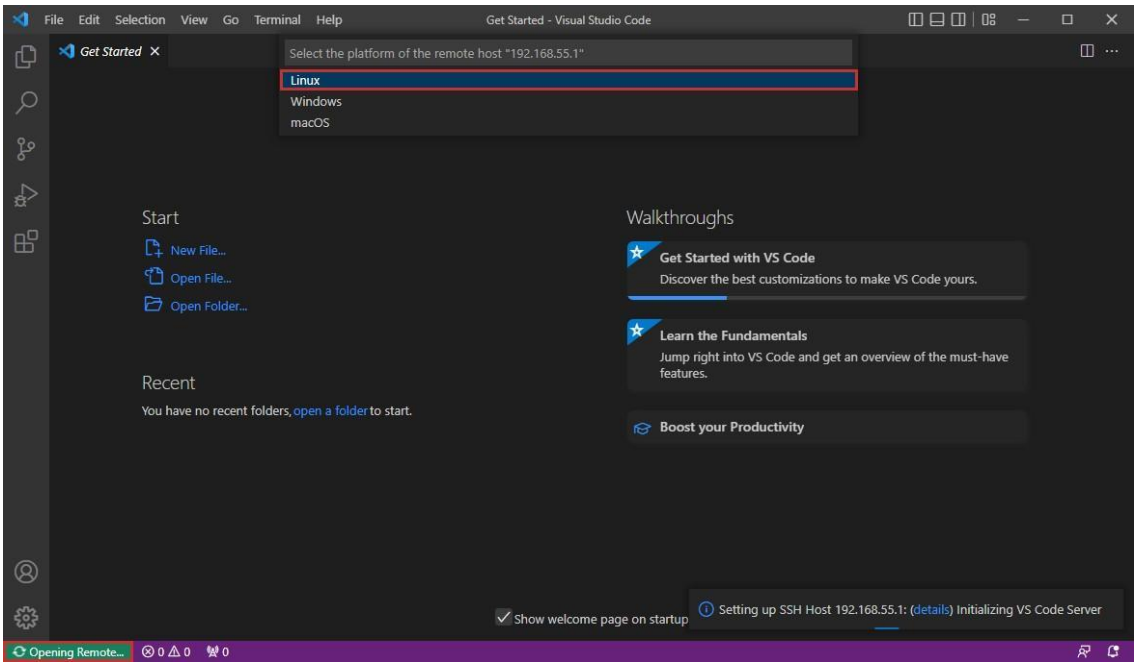


3) Add a new host and then introduce the ssh connection command (see network connection in stage 1).

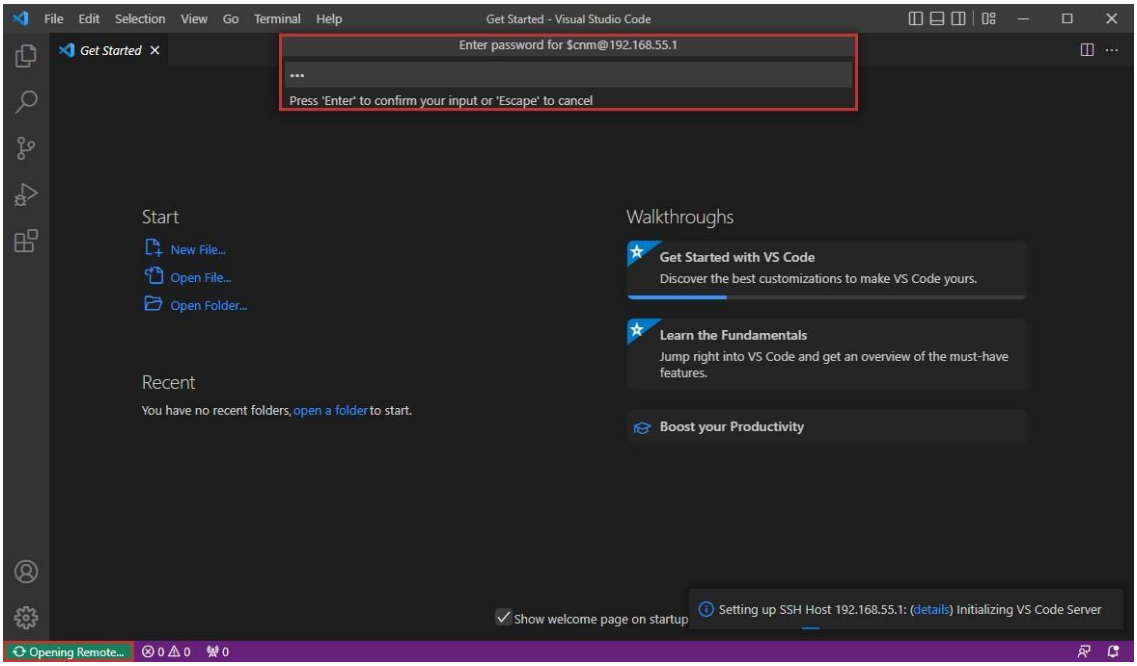
   Optional: You could change the SSH config file to add an alias to the SSH connection.



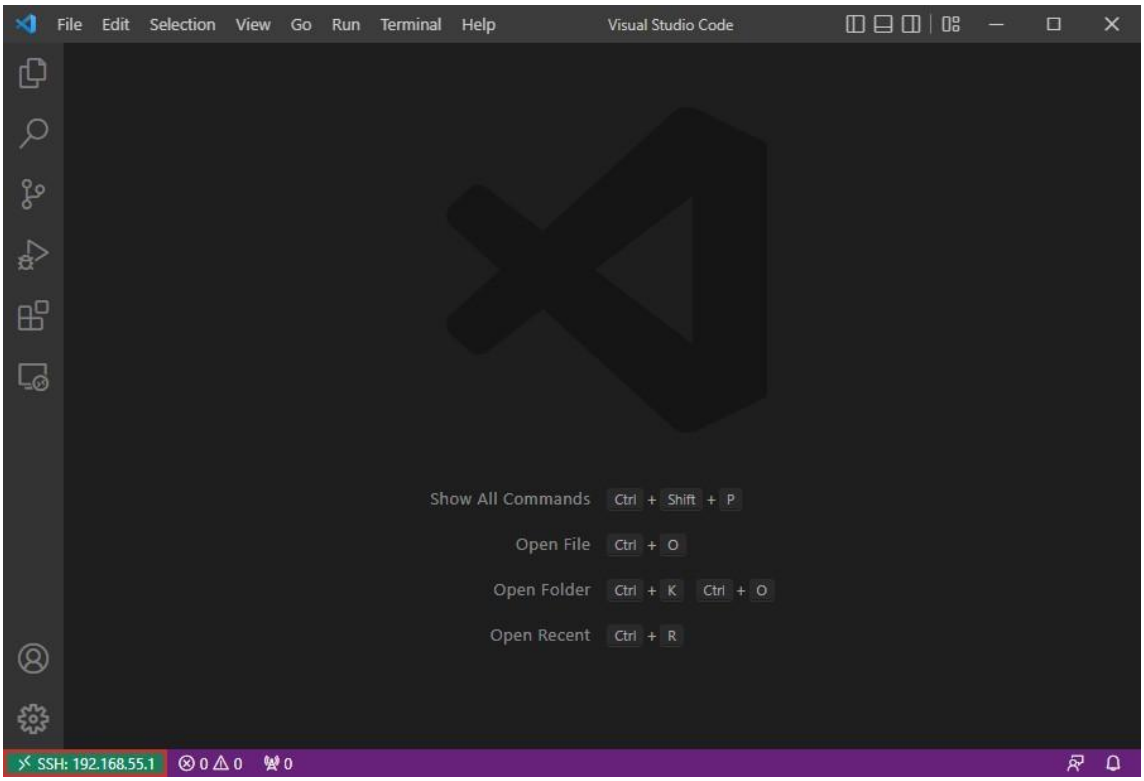
4) Once we have added our board as a new host we can connect to it. The first time it will ask for the type of host OS. We have to choose **Linux**.



5) Next step it will ask for the password (the first time it could ask for the password several times). Use the password used in stage 1.



6) Then it will start to install the VS Code Server on the device and if there are no errors, you will be connected to the device (see status bar).

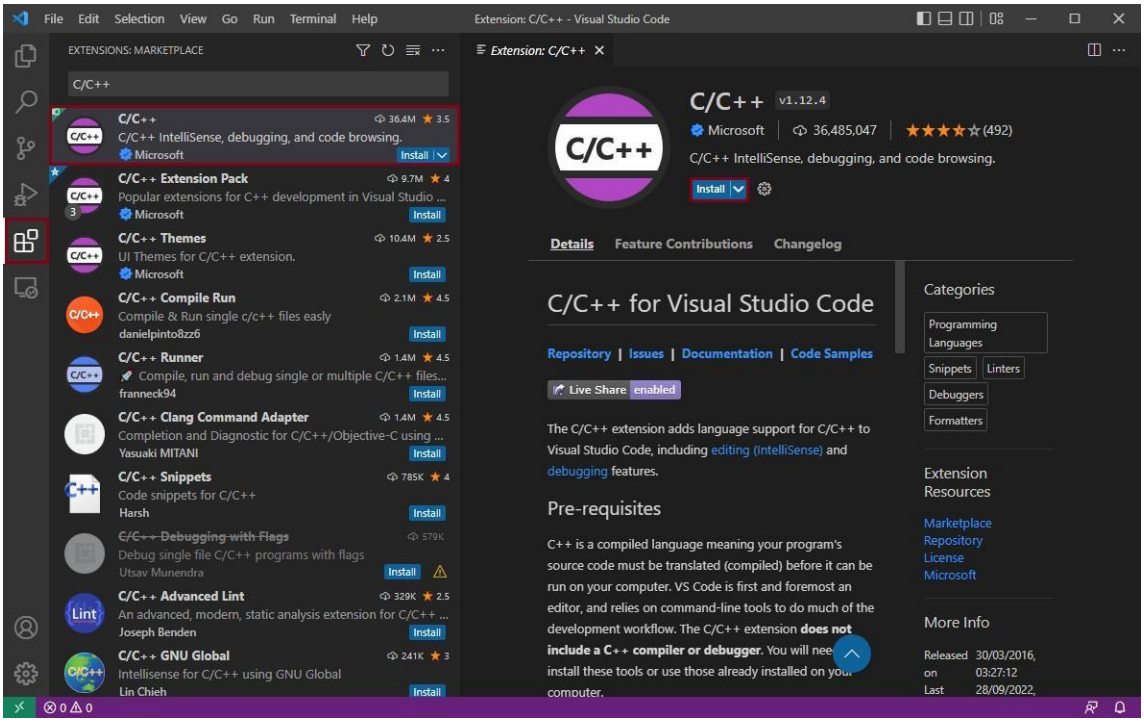


7) Once you are connected you will be able to open a folder as if you were connected to the device (it will require a password) or open a new terminal.

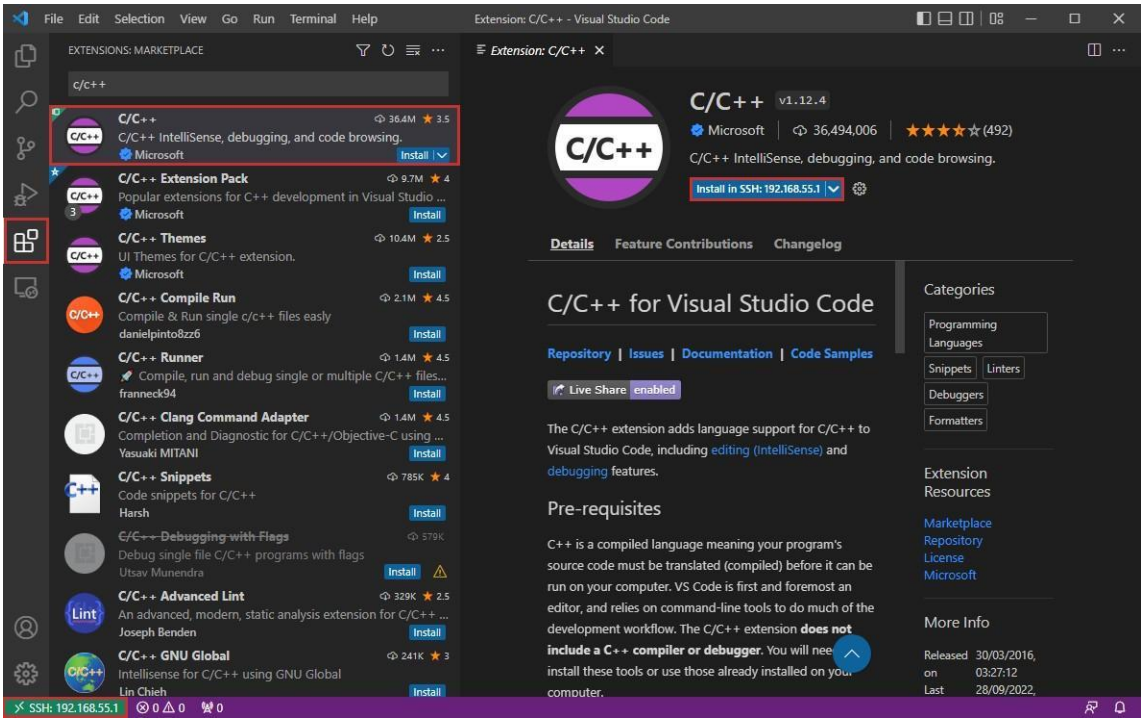
Once we are connected to the device we can drag and drop files to Visual Studio Code and it will get uploaded to the device. To download a file or folder from the device we need to right-click and choose download.

b) Installing the C/C++ extension

1) Go to the extension panel (left) and search for **C/C++**. Then install the **C/C++** extension. Optional, you could also install the **C/C++ Themes** extension.



If you are connected via SSH you will need to install this extension in the device following the same steps but once you are already connected. You can check it.



Once we have Visual Studio Code configured for our device we can start developing code on it.

Stage 4 – Git setup

It is advised to have knowledge of how to use git in the command line. We will use GitLab as our repo manager. The workflow used for the CNM labs is to configure our repo (https://reds-gitlab.heig-vd.ch/reds-public/cnm22_student) as upstream from which you will pull the changes that we will provide as the laboratory progresses. **Your work will be pushed to your personal repository cnm24-yourname that you have to create. You will also need to add us (users: maa-reds and mzs-reds) as Maintainers.**

1. If you have not created your GitLab repo or account yet, create an account and a private project named cnm24-yourname.
2. Clone your repository to your computer

```
git clone https://gitlab.com/<username>/cnm24-yourname.git
```

3. Add our repo as remote

```
git remote add cnm_base https://gitlab.com/reds-public/cnm24_student.git
```

4. Configure git repository

```
git config --local push.default simple
git config --local user.name “your
git config --local user.mail “your email”
```

5. Once you have configured it, you can use these commands to update the modifications for the laboratory progress.

```
 # Command to retrieve content from the base
repo
 # To be done at each new lab
 git pull cnm_base main
 # or
 git fetch cnm_base
 git merge cnm_base/main
```

6. To stage your changes to your local repository

```
git add <my_files>

git commit –m “commit message explaining all changes”
```

7. Push to and pull from your remote repository

```
# Push to your remote repo
git push
# Pull from your remote repo
git pull
```

Your repository should be updated regularly so we can track the status of your work. During the evaluation, only the last commit pushed to your *master* branch before the lab submission deadline will be considered. You can manage your repository as you wish (create and remove branches, for example) but in the end, you **must have everything in the *master* branch**.

Stage 5 – Compile, run and debug

We have provided a simple C program so you can test how to compile, run, and debug it. You will find it in the file **main.c** inside the lab folder **20220930_lab01_intro**.

a) Compile the program in a terminal. The simplest to compile your code with **gcc** is to type:

```
gcc –o main main.c
```

If we want to compile it with debug information, we need to use the flag **-g** with **gcc**.

```
gcc -g –o main main.c
```

b) Generate assembler code in the terminal. To stop after the stage of compilation, we need to use the flag **–S** with **gcc**.

```
gcc –S –o main.s main.c
```

The output can be open as a text file.

c) Profiling. To generate the extra code to write profile information suitable for the analysis with **gprof** we need to use the flag **–gp** with **gcc**.

```
gcc -o main main.c -pg
```

Then you can open the profiling information with **gprof** using the binary name. You can read more in the GNU gprof documentation.

```
gprof main
```

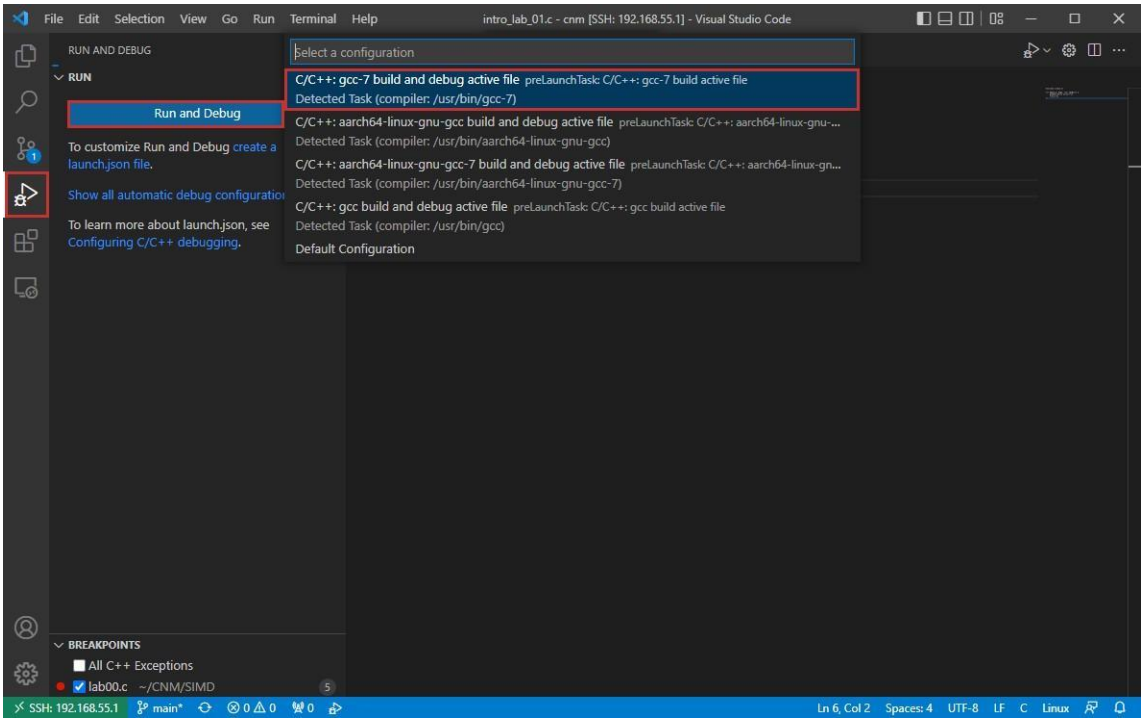
Check the output provided by gprof. You may want to double-check the gprof documentation to see the different flags that can be used.

According to the profiling information, what takes longer: vector initialization or the dot product? You will need to reply to this question on the README file of the repo.

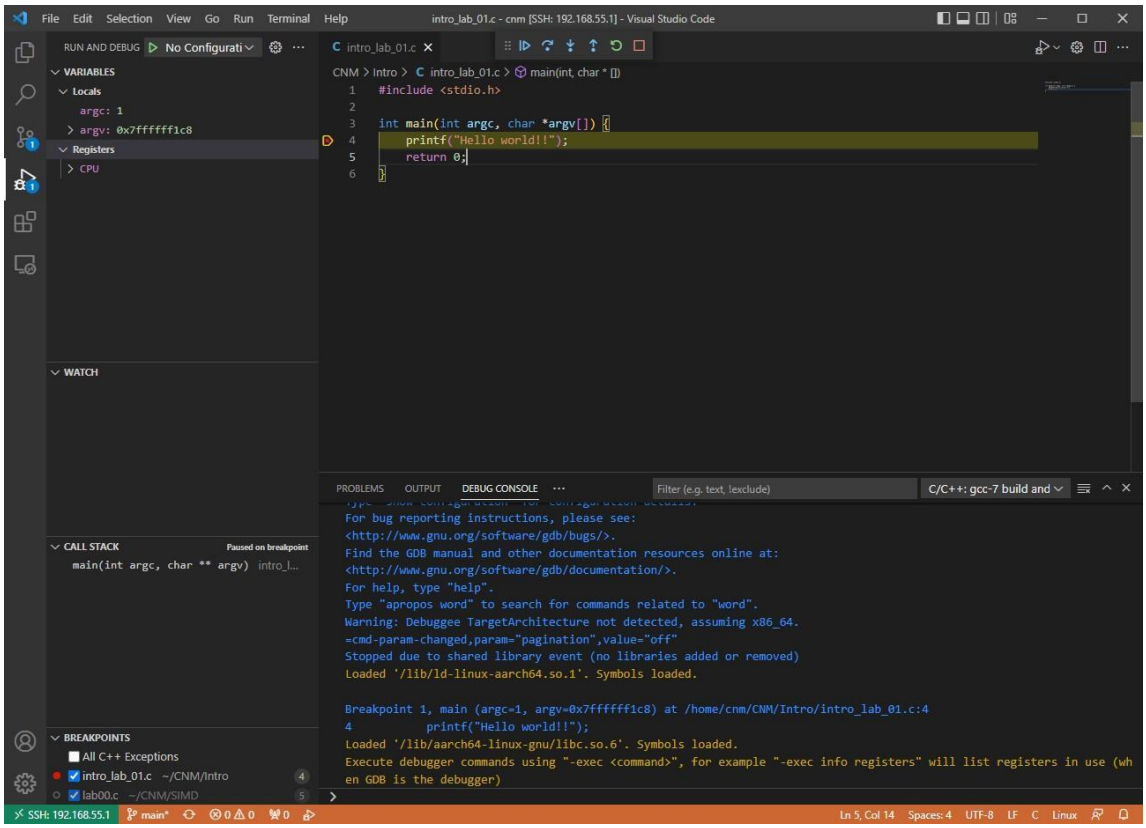
d) Debugging in Visual Studio Code. An easy way to debug our programs in the device is using the debugging capabilities of Visual Studio Code and the C/C++ extension.

1) Go to the Run and Debug panel in the left panel. Click on **Run and Debug** button. If it is the first time, you use the debugging you need to choose a compiler.

Find the differences between the compilers and write a short explanation in the README file of the repo



2) If there were no errors, the debugging process should start after compilation.



Internally, Visual Studio Code has created a file called **task.json** inside the folder .**vscode** that contains the building task configuration.

You can read more about debugging with Visual Studio Code in the official documentation.

Before you leave…

Make sure you have replied to all the questions *in light blue* of this manual and that you have filled- in the answers in the README file in the repo. You need to complete:

- The Stage 2 information about CPU and cache architecture
- The Stage 3 information using perf command
- Reply to the Stage 5 questions regarding profiling information and compilers.

Change the README file and commit changes to your own “cnm24-yourname” repository.