



Running Aerial cuPHY

Table of contents

Building Aerial cuPHY

Running the cuPHY Examples

Running cuPHY Performance Testing Scripts

List of Figures

Figure 0. Pucch Outcome 1

Figure 1. Pucch Outcome 2

Figure 2. Cubb Gpu Test Bench

Aerial cuPHY provides the cuPHY library and several examples that link with the library. Here we include instructions on using MATLAB to generate TVs. Please refer to [Generating TV and Launch Pattern Files](#) for using Aerial Python mcore Module to generate TVs.

Building Aerial cuPHY

Prerequisites

The following instructions assume the system configuration and Aerial cuBB installation are done. If not, see the [cuBB Install Guide](#) to complete the installation or upgrade process.

After powering on the system, use the following commands to verify that the GPU and NIC are in the correct state:

```
# Verify GPU is detected and CUDA driver version matches the release manifest. $ nvidia-smi
```

Verify that the NIC is in the correct state on the host (this is only required to run cuBB end-to-end):

```
# Verify NIC is detected: Example CX6-DX $ sudo lshw -c network -businfo Bus info
Device Class Description
=====
pci@0000:05:00.0 eno1 network I210 Gigabit Network Connection pci@0000:06:00.0
enp6s0 network I210 Gigabit Network Connection pci@0000:b5:00.0 ens6f0 network
MT2892 Family [ConnectX-6 Dx] pci@0000:b5:00.1 ens6f1 network MT2892 Family
[ConnectX-6 Dx] # Verify the link state is right. Assuming NIC port 0 is connected. $
sudo mlxlink -d b5:00.0 Operational Info ----- State : Active Physical state :
LinkUp Speed : 100G Width : 4x FEC : Standard RS-FEC - RS(528,514) Loopback Mode
: No Loopback Auto Negotiation : ON Supported Info ----- Enabled Link Speed
(Ext.) : 0x000007f2 (100G_2X,100G_4X,50G_1X,50G_2X,40G,25G,10G,1G) Supported
Cable Speed (Ext.) : 0x000002f2 (100G_4X,50G_2X,40G,25G,10G,1G) Troubleshooting
Info ----- Status Opcode : 0 Group Opcode : N/A Recommendation : No
issue was observed.
```

Set Up the Host Environment

Set up the environment on both servers. You must run `aerial-init.sh` once each time the system reboots. Refer to the “System Initialization Script” section in the cuBB Installation Guide for more details.

```
$ sudo ./aerial-init.sh
```

Launch the cuBB Container

Use the following command to launch the cuBB container:

```
$ sudo docker exec -it cuBB /bin/bash
```

Build Aerial cuPHY in the Container

Build cuPHY in the cuBB container using the following commands:

```
$ cd /opt/nvidia/cuBB/cuPHY $ mkdir build $ mkdir install $ cd build $ cmake .. -  
DCMAKE_INSTALL_PREFIX=../install $ make -j $(nproc --all)
```

cuPHY is, by default, built in `Release` mode. The option `BUILD_DOCS=ON` is also enabled by default to allow the make to generate the Doxygen documentation for the cuPHY library API. To disable this option, pass `-DBUILD_DOCS=OFF` to the CMake command line. The output directory is `cuPHY/install/docs`.

To put the built cuPHY headers and libraries into an installation directory so that other applications using the cuPHY library can compile and link with cuPHY, use the commands from the current build directory:

```
$ make install
```

This creates the `include` and `lib` directories under the `cuPHY/install` directory.

Running the cuPHY Examples

This section describes how to run the Aerial cuPHY standalone example programs. They read test vector data files as input. Refer to the supported test vector configurations list in the [cuBB Release Notes](#) for what test vectors to use for all the different configurations. Do not use old test vectors from previous cuBB releases with the example programs of this release.

Generating Test Vectors using Matlab 5GModel

Run this Matlab command:

```
cd('nr_matlab'); startup; [nTC, errCnt] = runRegression({'TestVector'}, {'allChannels'},  
'compact', [0, 1]);
```

All the cuPHY test vectors are generated and stored under `nr_matlab/GPU_test_input`.

Instructions for Testing cuPHY Channels Manually

PUSCH

Test Vectors

Match test vector name with *PUSCH_gNB_CUPHY_*.h5*

How to Run

- Streams mode:

```
cuPHY/build/examples/pusch_rx_multi_pipe/cuphy_ex_pusch_rx_multi_pipe -i  
~/&lt;tv_name&gt;.h5
```

- Graphs mode:

```
cuPHY/build/examples/pusch_rx_multi_pipe/cuphy_ex_pusch_rx_multi_pipe -i  
~/&lt;tv_name&gt;.h5 -m 1
```

Expected Outcome

Test 1 (CRC test KPI): All test cases must have zero CRC errors (only CRC errors, not correct ones, are reported when the channel is run).

PUCCH

Test Vectors

Match test vector name with *PUCCH_F*_gNB_CUPHY_*.h5*

How to Run

PUCCH format 0/1/2/3:

```
cuPHY/build/examples/pucch_rx_pipeline/cuphy_ex_pucch_rx_pipeline -i
<tv_name>
```

Expected Outcome

- `cuphy_ex_pucch_Fx_receiver` checks if the test vector includes PFX UCI first.
- If the test-vector UCI format is not expected, it displays “No PFX UCI received”.

```
./build/examples/pucch_F0_receiver/cuphy_ex_pucch_F0_receiver -i
./GPU_test_input/TVnr_6109_PUCCH_gNB_CUPHY_s0p126.h5
***** COMPUTE-SANITIZER

No PF0 UCI received.
***** LEAK SUMMARY: 0 bytes leaked in 0 allocations
***** ERROR SUMMARY: 0 errors
```

- If the test-vector UCI format is expected, it compares UCI output.xzsd.

```
./build/examples/pucch_F0_receiver/cuphy_ex_pucch_F0_receiver -i
./GPU_test_input/TVnr_6026_PUCCH_gNB_CUPHY_s0p1.h5
***** COMPUTE-SANITIZER

comparing cuPHY F0 UCI output to reference output: 0 mismatches out of 1 UCIs
***** LEAK SUMMARY: 0 bytes leaked in 0 allocations
***** ERROR SUMMARY: 0 errors
```

PRACH

Test Vectors

Match test vector name with *PRACH_gNB_CUPHY_*.h5*

How to Run

```
cuPHY/build/examples/prach_receiver_multi_cell/prach_receiver_multi_cell -i
<tv_name> -r <num_iteration> -k
```

Expected Outcome

- `prach_receiver_multi_cell` compares against the reference measurements in the test vector.
- Measured values are displayed and if they are within tolerance the message is displayed:

```
===== > Test PASS
```

PDSCH

Test Vectors

Match test vector name with `PDSCH_gNB_CUPHY_*.h5`

How to Run

- PDSCH in non-AAS mode, streams:

```
cuPHY/build/examples/pdsch_tx/cuphy_ex_pdsch_tx ~/<tv_name>.h5 2 0 0
```

- PDSCH in non-AAS mode, graphs:

```
cuPHY/build/examples/pdsch_tx/cuphy_ex_pdsch_tx ~/<tv_name>.h5 2 0 1
```

Expected Outcome

Test 1 (correctness against reference model): Channel reports correct match with reference model

PDCCH

Test Vectors

Match test vector name with `PDCCH_gNB_CUPHY_*.h5`

How to Run

- Streams mode:

```
cuPHY/build/examples/pdcch/embed_pdcch_tf_signal -i ~/<tv_name>.h5 -m 0
```


- Graphs mode:

```
cuPHY/build/examples/pdcch/embed_pdcch_tf_signal -i ~/&lt;tv_name&gt;.h5 -m 1
```

Expected Outcome

Test 1 (correctness against reference model): Test PASS

SSB

Test Vectors

Match test vector name with *SSB_gNB_CUPHY_*.h5*

How to Run

- Streams mode:

```
cuPHY/build/examples/ss/testSS -i ~/&lt;tv_name&gt;.h5 -m 0
```
- Graphs mode:

```
cuPHY/build/examples/ss/testSS -i ~/&lt;tv_name&gt;.h5 -m 1
```

Expected Outcome

Test 1 (correctness against reference model): Test PASS

CSI-RS

Test Vectors

Match test vector name with *CSIRS_gNB_CUPHY_*.h5*

How to Run

- Streams mode:

```
cuPHY/build/examples/csi_rs/nzp_csi_rs_test -i &lt;tv_name&gt; -m 0
```
- Graphs mode:

```
cuPHY/build/examples/csi_rs/nzp_csi_rs_test -i &lt;tv_name&gt; -m 1
```

Expected Outcome

Test 1 (correctness against reference model): Test PASS

SRS

Test Vectors

Match test vector name with *SRS_gNB_CUPHY_*.h5*

How to Run

- Streams mode:

```
cuPHY/build/examples/srs_rx_pipeline/cuphy_ex_srs_rx_pipeline -i &lt;tv_name&gt;  
-r &lt;num_iteration&gt; -m 0
```

- Graphs mode:

```
cuPHY/build/examples/srs_rx_pipeline/cuphy_ex_srs_rx_pipeline -i &lt;tv_name&gt;  
-r &lt;num_iteration&gt; -m 1
```

Expected Outcome

Test 1 (correctness against reference model): SRS reference check: PASSED! ; Timing results are provided

BFC

Test Vectors

Match test vector name with *BFW_gNB_CUPHY_*.h5*

How to Run

- Streams mode:

```
cuPHY/build/examples/bfc/cuphy_ex_bfc -i &lt;tv_name&gt; -r  
&lt;num_iteration&gt; -m 0
```

- Graphs mode:

```
cuPHY/build/examples/bfc/cuphy_ex_bfc -i &lt;tv_name&gt; -r  
&lt;num_iteration&gt; -m 1
```

- Add `-c` to enable reference check (default disabled)

Expected Outcome

Test 1 (measure latency without reference check): Timing results are provided

Test 2 (correctness against reference model using `-c`): `Test PASS`; Timing results are provided

Instructions for LDPC Performance Test

The `ldpc_perf_collect.py` Python script from the cuPHY repository can be used to perform error rate tests for the cuPHY LDPC decoder. There are test input files defined for $Z = [64, 128, 256, 384]$, $BG = [1, 2]$. The current tests check whether the block error rate (BLER, also sometimes referred to as Frame Error Rate or FER) is less than 0.1.

From the build directory, the following commands run the tests:

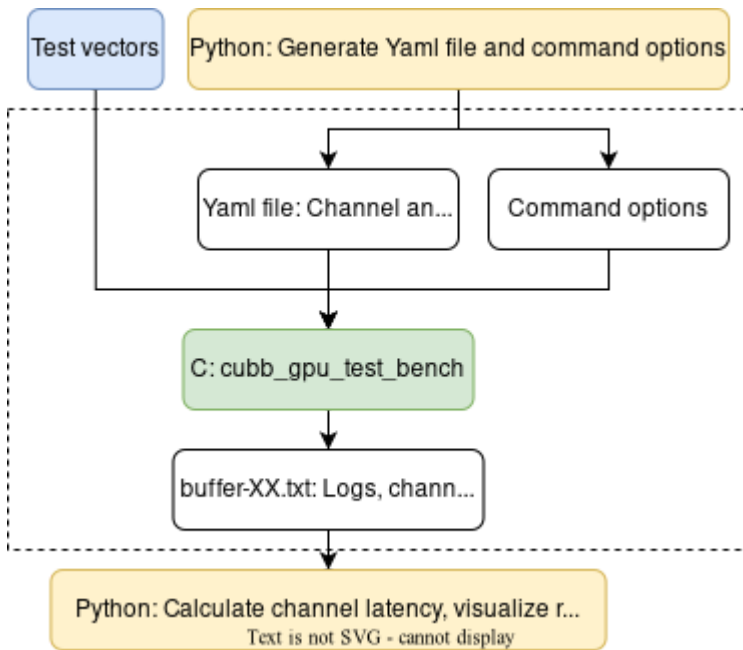
```
../util/ldpc/ldpc_perf_collect.py --mode test -i
../util/ldpc/test/ldpc_decode_BG1_Z64_BLER0.1.txt -f -w 800 -P
../util/ldpc/ldpc_perf_collect.py --mode test -i
../util/ldpc/test/ldpc_decode_BG1_Z128_BLER0.1.txt -f -w 800 -P
../util/ldpc/ldpc_perf_collect.py --mode test -i
../util/ldpc/test/ldpc_decode_BG1_Z256_BLER0.1.txt -f -w 800 -P
../util/ldpc/ldpc_perf_collect.py --mode test -i
../util/ldpc/test/ldpc_decode_BG1_Z384_BLER0.1.txt -f -w 800 -P
../util/ldpc/ldpc_perf_collect.py --mode test -i
../util/ldpc/test/ldpc_decode_BG2_Z64_BLER0.1.txt -f -w 800 -P
../util/ldpc/ldpc_perf_collect.py --mode test -i
../util/ldpc/test/ldpc_decode_BG2_Z128_BLER0.1.txt -f -w 800 -P
../util/ldpc/ldpc_perf_collect.py --mode test -i
../util/ldpc/test/ldpc_decode_BG2_Z256_BLER0.1.txt -f -w 800 -P
../util/ldpc/ldpc_perf_collect.py --mode test -i
../util/ldpc/test/ldpc_decode_BG2_Z384_BLER0.1.txt -f -w 800 -P
```

Each test input file contains multiple tests for different code rates, as specified by the number of parity nodes.

Running cuPHY Performance Testing Scripts

`aerial_sdk/testBenches` provides a multi-cell multi-channel test bench to test cuPHY standalone performance. It relies on [NVIDIA Multi-Process Service \(MPS\)](#) to share the GPU

among multiple channels. Specifically, there are two folders and their relationship can be summarized as follows:



- *cubb_gpu_test_bench*: a C test bench that runs the multi-cell multi-channel cuPHY standalone GPU workload (that is, without I/O to and from NIC or layer 2). The input of *cubb_gpu_test_bench* are test vectors, a Yaml file, and some command options to run the GPU workload. The output is a *buffer-XX.txt* file that has the logs, channel start/end times, debug info, etc. Here *XX* is the number of cells used in testing.
- *perf*: a set of Python scripts to automate performance testing using *cubb_gpu_test_bench*. The Python scripts can help generate the Yaml file and command options, config GPU and MPS before running *cubb_gpu_test_bench*; collect the test results by reading the output *buffer-XX.txt* from *cubb_gpu_test_bench*.

Generating Test Vectors using Matlab 5GModel

Run this Matlab command:

```

cd <5GModel root>/nr_matlab startup genCfgTV_perf_ss('performance-avg.xlsm');
genCfgTV_perf_ss_bwc('performance-avg.xlsm'); genCfgTV_perf_pucch();
genCfgTV_perf_pdcch(); genCfgTV_perf_prach(); genCfgTV_perf_csirs();
genCfgTV_perf_ssb(); genCfgTV_perf_srs();
  
```

All the cuPHY Performance test vectors are generated and stored under `nr_matlab/GPU_test_input`.

Measuring cuPHY Performance using `cubb_gpu_test_bench`

Requirements:

- The performance measurements can be run using a Linux environment making one of more GPU available. Such environment is here assumed to have:
 - bash or zsh as default shell
 - Python 3.8+ and the following packages: numpy, pyCUDA, pyYAML
 - CUDA toolkit 11.4 or above properly configured so that `nvidia-cuda-mps-control` and `nvidia-smi` are in PATH
 - The executable `cubb_gpu_test_bench` is located in the `<testBenches>/build` folder.

There are three steps when measuring cell capacity using `cubb_gpu_test_bench`. The `perf` folder provides some pre defined test cases. Below is an example of 4T4R (F08) using TDD pattern DDDSUUDDDD.

1. Generate the JSON file that defines the use case (e.g., 8~16 peak or average cells)

```
python3 generate_avg_TDD.py --peak 8 9 10 11 12 13 14 15 16 --avg 0 --exact --case F08
```

2. Measure the latency of all channels based on predefined patterns

```
python3 measure.py --cuphy <testBenches>/build --vectors <test_vectors> --config testcases_avg_F08.json --uc uc_avg_F08_TDD.json --delay 100000 --gpu <GPU_ID> --freq <GPU_freq> --start <cell_start> --cap <cell_cap> --iterations 1 --slots <nSlots> --power <budget> --target <sms_prach> <sms_pdcch> <sms_pucch> <sms_pdsch> <sms_pusch> <sms_ssb> --2cb_per_sm --save_buffer --priority --prach --prach_isolate --pdcch --pdcch_isolate --pucch --pucch_isolate --tdd_pattern
```

```
dddsuudddd --pusch_cascaded --ssb --csirs --groups_dl --pack_pdsch --groups_pusch
--ldpc_parallel <--graph>
```

where:

- `<GPU_ID>` is the ID of the GPU on which the measurements are to be run; e.g., **0** for single GPU systems
 - `<GPU_freq>` is the GPU clock frequency in MHz
 - `<cell_start>` is the minimum number of cells to be tested
 - `<cell_cap>` is the maximum number of cells to be tested. The Python scripts will run `cubb_gpu_test_bench` for a range of [`<cell_start>`, `<cell_cap>`] cells and collect the latency results
 - `<budget>` is the power budget in Watts
 - `<sms_channelName>` is the number of streaming multiprocessors used per MPS sub-context for each channel during the run, where `channelName` can be PRACH, PDCCH, PUCCH, PDSCH, PUSCH, SSB
 - `<--graph>` add this option to run in graph mode, otherwise in stream mode
 - Notes: use `--test` to see what yaml file and command options the Python scripts generated without running the tests on GPU
3. Visualize the latency of each channel (this step requires Python library `matplotlib`). We generate `compare-<date>.png` showing the CDF of the latency for all tested channels:

- if run in stream mode:

```
python3 compare.py --filename
<sms_prach>_<sms_pdcch>_<sms_pucch>_<sms_pdsch>_<sms_pusch>_<sms_ssb>_sw
--cells <nCell>+0
```

- if run in graph mode:

```
python3 compare.py --filename  
<sms_prach>_<sms_pdcch>_<sms_pucch>_<sms_pdsch>_<sms_pusch>_<sms_ssb>_sw  
--cells <nCell>+0
```

where:

- *<nCell>* is the number of cells we would like to visualize the latency results

It is possible to compare latency results of different number of cells in one figure. For instance, we can compare the latency of 8 cells and 9 cells:

```
python3 compare.py --filename  
<sms_prach>_<sms_pdcch>_<sms_pucch>_<sms_pdsch>_<sms_pusch>_<sms_ssb>_sw  
<sms_prach>_<sms_pdcch>_<sms_pucch>_<sms_pdsch>_<sms_pusch>_<sms_ssb>_sw  
--cells 8+0 9+0
```

In all cases, Aerial CUDA-Accelerated RAN offers the possibility of measuring the latency of all workloads including:

- Dynamic and heterogeneous traffic (meaning that each cell is stimulated with different test vectors and every slot sees a different allocation of the test vectors to the considered cells)
- Specific traffic models

© Copyright 2024, NVIDIA.. PDF Generated on 06/06/2024