

COMPUTER ORGANIZATION AND ARCHITECTURE

COA LAB 2023

GROUP NUMBER: 13

GPGPU-Sim provides a detailed simulation model of contemporary NVIDIA GPUs running CUDA and/or OpenCL workloads.

This documentation is meant for installing GPGPU Simulator and executing the applications using GPGPU-Simulator.

The version of GPGPU-Sim has been tested with a subset of CUDA version 4.2, 5.0, 5.5, 6.0, 7.5, 8.0, 9.0, 9.1, 10, and 11.

The commands and references used in the following stages are based on the usage of ***Ubuntu 20.04.06 LTS and CUDA 11.1 ToolKit only.***

- Install linux in a dual-boot format (prefer not to use virtual-box, as it may crash sometimes) using the above ubuntu version.
- **STEP 1: INSTALLING DEPENDENCIES**
 - First install all the dependencies using the following commands:
 - Install gcc, g++, git using the following commands:
 - ◆ `sudo apt-get install gcc-7 g++-7 git`
 - ◆ `cd /bin`
 - ◆ `sudo mv gcc gcc-9.4`
 - ◆ `sudo mv g++ g++-9.4`
 - ◆ `sudo mv gcc-7 gcc`
 - ◆ `sudo mv g++-7 g++`
 - GPGPU-Sim Dependencies
 - ◆ `sudo apt-get install build-essential xutils-dev bison zlib1g-dev flex libglu1-mesa-dev`
 - GPGPU-Sim documentation dependencies:
 - ◆ `sudo apt-get install doxygen graphviz`
 - AerialVision dependencies:
 - ◆ `sudo apt-get install python-pmw python-ply python-numpy libpng-dev python3-matplotlib`
 - CUDA SDK dependencies:

◆ `sudo apt-get install libxi-dev libxmu-dev freeglut3-dev`

- **STEP 2: INSTALLATION OF CUDA TOOLKIT**

➤ Now download the CUDA ToolKit using the following commands in order:

◆ `wget`
https://developer.download.nvidia.com/compute/cuda/11.1.0/local_installers/cuda_11.1.0_455.23.05_linux.run

◆ `sudo apt-get install nvidia-cuda-toolkit`

◆ `sudo sh cuda_11.1.0_455.23.05_linux.run`

◆ During the execution of this code

- Wait for a while unless a EULA acceptance screen appears.
- After that ensure that you deselect the NVIDIA driver if your system has a NVIDIA Graphic Card of whatever size may be.
- Now proceed with installation.

- **STEP 3: RUN THE APPLICATION**

➤ Now build the GPGPU-Sim

➤ Enter the following command in order:

➤ Path Variable Changes

◆ `sudo nano ~/.bashrc`

◆ ***Add the following two lines at the end the file:***

◆ `export CUDA_INSTALL_PATH="/usr/local/cuda-11.1"`

◆ `export PATH="/usr/local/cuda-11.1/bin:$PATH"`

◆ ***Exit the file and run the following command:***

◆ `source ~/.bashrc`

◆ Re-start the terminal

➤ **Enter the following commands in the terminal present in the root directory:**

◆ `git clone https://github.com/gpgpu-sim/gpgpu-sim_distribution.git`

◆ `cd gpgpu-sim_distribution/`

◆ `source setup_environment`

- ◆ make
- ◆ mkdir run_code
- ◆ cd run_code

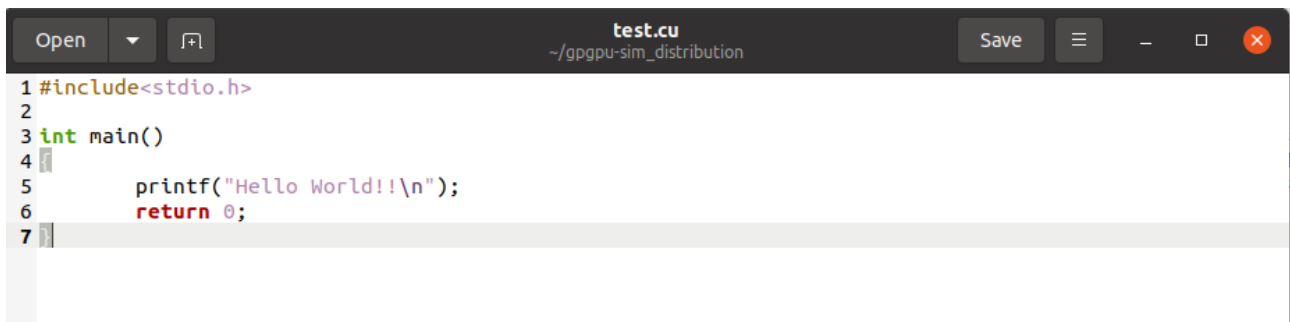
- **Now add your test file which you want to run using GPGPU-Sim.
(extension of file name should be .cu)**

Now copy any one your required GPU Hardware into the current working directory using the following command:

In the below line go to the directory gpgpusim_distribution/configs/tested-cfgs and choose the hardware and then write that name in the command.

- ◆ **cp -r ~/gpgpusim_distribution/configs/tested-cfgs/{choose your required hardware and replace it over here}/* ./**
- ◆ **source ~/gpgpu-sim_distribution/setup_environment**
- ◆ **nvcc -lcudart {Enter your file_name}.cu**
- ◆ **./a.out**

INPUT:

A screenshot of a code editor window titled 'test.cu' with the path '~/gpgpu-sim_distribution' shown below the title. The editor contains the following C code:

```
1 #include<stdio.h>
2
3 int main()
4 {
5     printf("Hello World!!\n");
6     return 0;
7 }
```

OUTPUT:

- ***The following is the output when the following commands are run in the terminal from the directory where the test file is present:***
- ◆ **source ~/gpgpu-sim_distribution/setup_environment**
 - ◆ **nvcc -lcudart test.cu**
 - ◆ **./a.out**

```
anish@anish-Inspiron-15-3511: ~/gpgpu-sim_distribution

anish@anish-Inspiron-15-3511:~/gpgpu-sim_distribution$ source setup_environment release
GPGPU-Sim version 4.0.0 (build gpgpu-sim_git-commit-90ec3399763d7c8512cfe7dc193473086c38ca38-modified_1.0) configured with GPUWatch.

-----
INFO - If you only care about PTX execution, ignore this message. GPGPU-Sim supports PTX execution in modern CUDA.
If you want to run PTXPLUS (sm_1x SASS) with a modern card configuration - set the environment variable
$PTXAS_CUDA_INSTALL_PATH to point a CUDA version compatible with your card configurations (i.e. 8+ for PASCAL, 9+ for VOLTA etc..)
For example: "export $PTXAS_CUDA_INSTALL_PATH=/usr/local/cuda-9.1"

The following text describes why:
If you are using PTXPLUS, only sm_1x is supported and it requires that the app and simulator binaries are compiled in CUDA 4.2 or less.
The simulator requires it since CUDA headers describe struct sizes in the exec which change from gen to gen.
The apps require 4.2 because newer versions of CUDA tools have dropped parsing support for generating sm_1x
When running using modern config (i.e. volta) and PTXPLUS with CUDA 4.2, the $PTXAS_CUDA_INSTALL_PATH env variable is required to get proper register usage
(and hence occupancy) using a version of CUDA that knows the register usage on the real card.

-----
setup_environment succeeded
anish@anish-Inspiron-15-3511:~/gpgpu-sim_distribution$
```

```
anish@anish-Inspiron-15-3511:~/gpgpu-sim_distribution$ nvcc -lcudart test.cu
anish@anish-Inspiron-15-3511:~/gpgpu-sim_distribution$ ./a.out

*** GPGPU-Sim Simulator Version 4.0.0 [build gpgpu-sim_git-commit-90ec3399763d7c8512cfe7dc193473086c38ca38-modified_1.0] ***

****
GPGPU-Sim PTX: simulation mode 0 (can change with PTX_SIM_MODE_FUNC environment variable:
1=functional simulation only, 0=detailed performance simulator)
GPGPU-Sim PTX: overriding embedded ptx with ptx file (PTX_SIM_USE_PTX_FILE is set)
GPGPU-Sim: Configuration options:
GPGPU-Sim: Configuration options:
- save_embedded_ptx 0 # saves ptx files embedded in binary as <no> ptx
- keep 0 # keep intermediate files created by GPGPU-Sim when interfacing with external programs
- gpgpu_ptx_save_converted_ptxplus 0 # saved converted ptxplus to a file
- gpgpu_occupancy_sm_number 20 # The SM number to pass to ptxas when getting register usage for computing GPU occupancy. This parameter is required in the config.
- ptx_opcode_latency_int 4,13,4,5,145,32 # Opcode latencies for integers <ADD,MAX,MUL,MAD,DIV,SHFL>Default 1,1,19,25,145,32
- ptx_opcode_latency_fp 4,13,4,5,39 # Opcode latencies for single precision floating points <ADD,MAX,MUL,MAD,DIV>Default 1,1,1,1,39
- ptx_opcode_latency_dp 0,19,0,0,330 # Opcode latencies for double precision floating points <ADD,MAX,MUL,MAD,DIV>Default 0,0,0,0,335
- ptx_opcode_latency_sfu 8 # Opcode latencies for SFU instructionsDefault 8
- ptx_opcode_latency_tensor 64 # Opcode latencies for Tensor instructionsDefault 64
- ptx_opcode_initiation_int 1,2,1,0,4 # Opcode initiation intervals for integers <ADD,MAX,MUL,MAD,DIV,SHFL>Default 1,1,4,4,32,4
- ptx_opcode_initiation_fp 1,2,1,1,4 # Opcode initiation intervals for single precision floating points <ADD,MAX,MUL,MAD,DIV>Default 1,1,1,1,5
- ptx_opcode_initiation_dp 8,16,8,0,130 # Opcode initiation intervals for double precision floating points <ADD,MAX,MUL,MAD,DIV>Default 8,8,8,0,130
- ptx_opcode_initiation_sfu 8 # Opcode initiation intervals for sfu instructionsDefault 8
- ptx_opcode_initiation_tensor 64 # Opcode initiation intervals for tensor instructionsDefault 64
- cdp_latency 7200,8000,100,12000,1000 # CDP API latency <cudaStreamCreateWithFlags, cudaGetParameterBufferV2_int_perWarp, cudaGetParameterBufferV2_perKernel, cudaLaunchDeviceV2_int_perWarp, cudaLaunchDeviceV2_perKernel>Default 7200,8000,100,12000,1000
- network_mode 1 # Interconnection network mode
- inter_config_file config_ferm_islip.icnt # Interconnection network config file
- icnt_in_buffer_limit 64 # in_buffer_limit
- icnt_out_buffer_limit 64 # out_buffer_limit
- icnt_subnets 2 # subnets
- icnt_arbiter_algo 1 # arbiter_algo
- icnt_verbose 0 # icnt_verbose
- icnt_grant_cycles 1 # grant_cycles
- gpgpu_ptx_use_cuobjdump 1 # Use cuobjdump to extract ptx and sass from binaries
- gpgpu_experimental_llb_support 0 # Try to extract code from cuda libraries [Broken because of unknown cudaGetExportable]
- checkpoint_option 0 # checkpointing flag (0 = no checkpoint)
- checkpoint_kernel 1 # checkpointing during execution of which kernel (1= 1st kernel)
- checkpoint_CTA 1 # checkpointing after # of CTA (< less than total CTA)
- resume_option 0 # resume flag (0 = no resume)
- resume_kernel 0 # Resume from which kernel (1= 1st kernel)
- resume_CTA 0 # resume from which CTA
- resume_CTA_t 0 # resume from which CTA
- checkpoint_insn_V 0 # resume from which CTA
- gpgpu_ptx_convert_to_ptxplus 0 # Convert SASS (native ISA) to ptxplus and run ptxplus
- gpgpu_ptx_force_max_capability 20 # Force maximum compute capability
- gpgpu_ptx_inst_debug_to_file 0 # Dump executed instructions' debug information to file
- gpgpu_ptx_inst_debug_file inst_debug.txt # Executed instructions' debug output file
```

```
GPGPU-Sim uArch: Interconnect node reverse Map (lcntID to shaderID+MemID)
GPGPU-Sim uArch: Memory nodes start from ID: 15
GPGPU-Sim uArch: 0 1 2 3 4
GPGPU-Sim uArch: 5 6 7 8 9
GPGPU-Sim uArch: 10 11 12 13 14
GPGPU-Sim uArch: 15 16 17 18 19
GPGPU-Sim uArch: 20 21 22 23 24
GPGPU-Sim uArch: 25 26
GPGPU-Sim uArch: performance model initialization complete.
self exe links to: /home/anish/gpgpu-sim_distribution/a.out
self exe links to: /home/anish/gpgpu-sim_distribution/a.out
11.0
GPGPU-Sim PTX: __cudaRegisterFatBinary, fat_cubin_handle = 1, filename=default
self exe links to: /home/anish/gpgpu-sim_distribution/a.out
Running md5sum using "md5sum /home/anish/gpgpu-sim_distribution/a.out "
60a3dc382517068aec070fa7f309215a /home/anish/gpgpu-sim_distribution/a.out
self exe links to: /home/anish/gpgpu-sim_distribution/a.out
Extracting specific PTX file named a.1.sm_52.ptx
Extracting PTX file and ptxas options 1: a.1.sm_52.ptx -arch=sm_52
Hello World!!
GPGPU-Sim: *** exit detected ***
anish@anish-Inspiron-15-3511:~/gpgpu-sim_distribution$
```

• We can see the
“Hello World!” Printed through
GPGPU-Sim.