MIDSEM PROJECT

GROUP-5

GROUP MEMBERS:

Prisha Srinidi (21CS01057)
Kankanala Siva Sai Amrutha (21CS02005)
Bharati Pradhan (21CS01009)
Meruva Yashna (21CS01039)
Meesala Aakanksha (21CS01056)

TYPES OF WARP STATES

1. Waiting- Warps that are waiting for an instruction so that further dependent instructions can be issued to the pipeline in this category.

2. Others- Warps that are waiting for a synchronisation instruction, and warps that do not have any instruction present in their instruction buffer are included in this category.

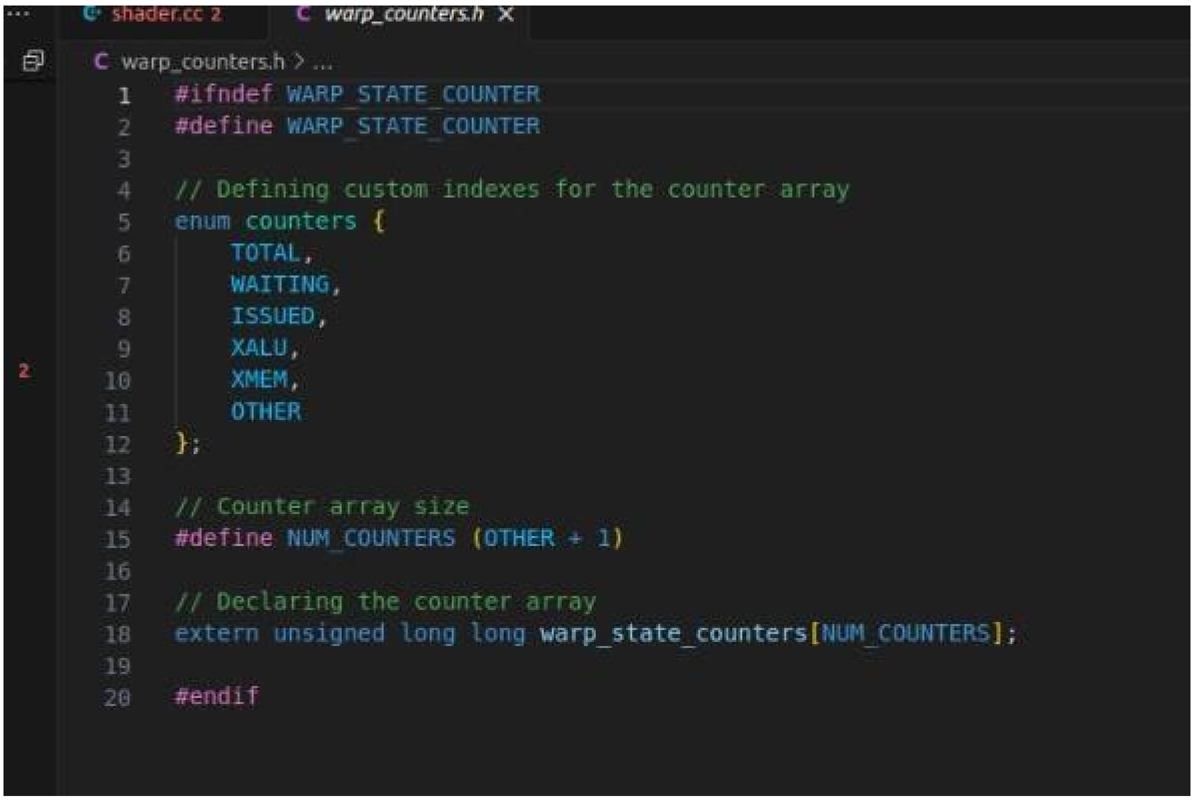
3. Excess ALU- Warps that are ready for the execution of arithmetic operations, but cannot do so due to unavailability of resources (pipelines).

4. Excess MEM- Warps that are ready for the execution of memory related operations, but are unable to do so due to unavailability of resources. These warps are restricted if the pipeline is stalled due to back pressure from memory or if the maximum number of instructions that can be issued to the pipeline has already been issued.

5. Issued- Warps that have already issued an instruction to the pipeline come under this category. These warps contribute toward the IPC count of the streaming multiprocessor.

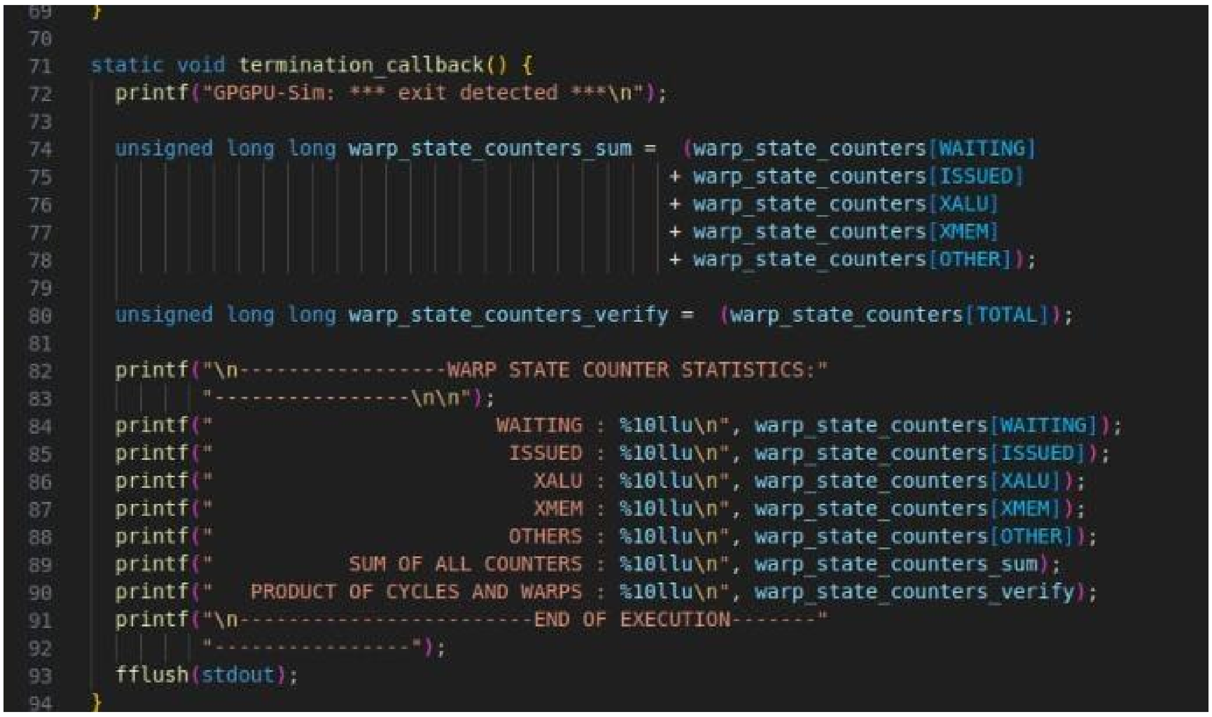
PROCEDURE

1. Include a header file “warp_counter.h” that contains the enumeration of the different types of warps in the src folder in the gpgpu-sim_distribution folder.



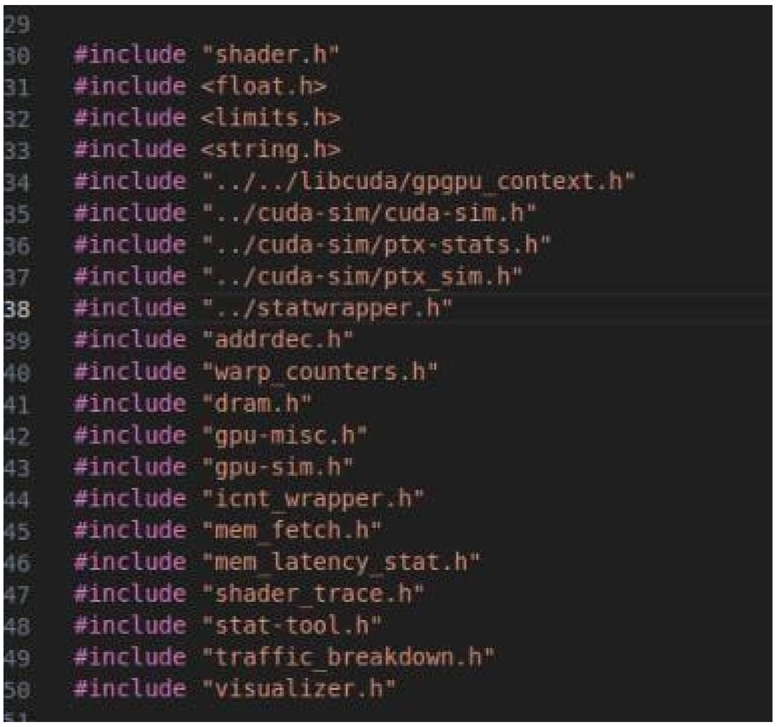
```
#ifndef WARP_STATE_COUNTER
#define WARP_STATE_COUNTER

// Defining custom indexes for the counter array
enum counters {
    TOTAL,
    WAITING,
    ISSUED,
    XALU,
    XMEM,
    OTHER
};

// Counter array size
#define NUM_COUNTERS (OTHER + 1)

// Declaring the counter array
extern unsigned long long warp_state_counters[NUM_COUNTERS];

#endif
```

2. Modify the gpgpusim_entrypoint.cc file to include the following lines of code in the terminal_callback() function, so that we are able to print the final statistics of the warps in different states.



```c
}

static void termination_callback() {
  printf("GPGPU-Sim: *** exit detected ***\n");

  unsigned long long warp_state_counters_sum =  (warp_state_counters[WAITING]
                                               + warp_state_counters[ISSUED]
                                               + warp_state_counters[XALU]
                                               + warp_state_counters[XMEM]
                                               + warp_state_counters[OTHER]);

  unsigned long long warp_state_counters_verify =  (warp_state_counters[TOTAL]);

  printf("\n-----------------WARP STATE COUNTER STATISTICS:"
         "----------------\n\n");
  printf("                       WAITING : %10llu\n", warp_state_counters[WAITING]);
  printf("                        ISSUED : %10llu\n", warp_state_counters[ISSUED]);
  printf("                          XALU : %10llu\n", warp_state_counters[XALU]);
  printf("                          XMEM : %10llu\n", warp_state_counters[XMEM]);
  printf("                        OTHERS : %10llu\n", warp_state_counters[OTHER]);
  printf("           SUM OF ALL COUNTERS : %10llu\n", warp_state_counters_sum);
  printf("   PRODUCT OF CYCLES AND WARPS : %10llu\n", warp_state_counters_verify);
  printf("\n-----------------------END OF EXECUTION-------"
         "----------------");
  fflush(stdout);
}
```

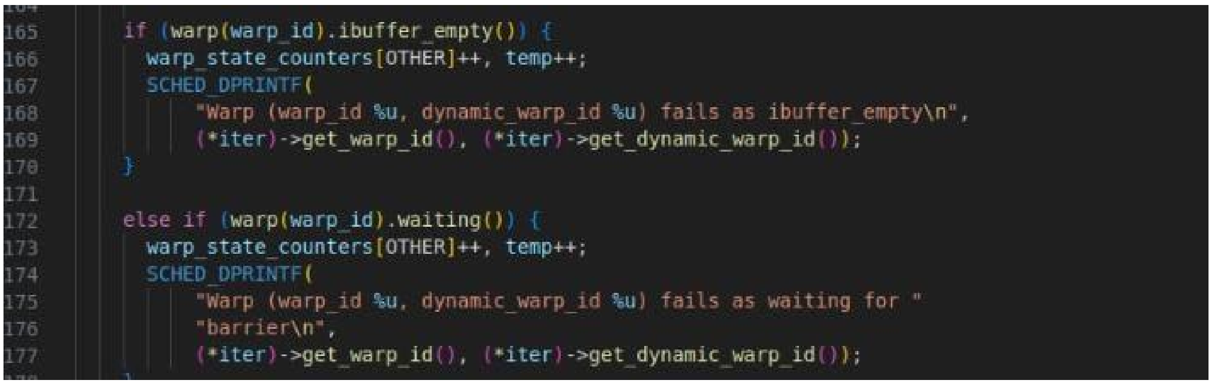
3. Add the warp_counters.h file to the shader.cc file as a header file



```c
#include "shader.h"
#include <float.h>
#include <limits.h>
#include <string.h>
#include "../../libcuda/gpgpu_context.h"
#include "../cuda-sim/cuda-sim.h"
#include "../cuda-sim/ptx-stats.h"
#include "../cuda-sim/ptx_sim.h"
#include "../statwrapper.h"
#include "addrdec.h"
#include "warp_counters.h"
#include "dram.h"
#include "gpu-misc.h"
#include "gpu-sim.h"
#include "icnt_wrapper.h"
#include "mem_fetch.h"
#include "mem_latency_stat.h"
#include "shader_trace.h"
#include "stat-tool.h"
#include "traffic_breakdown.h"
#include "visualizer.h"
```

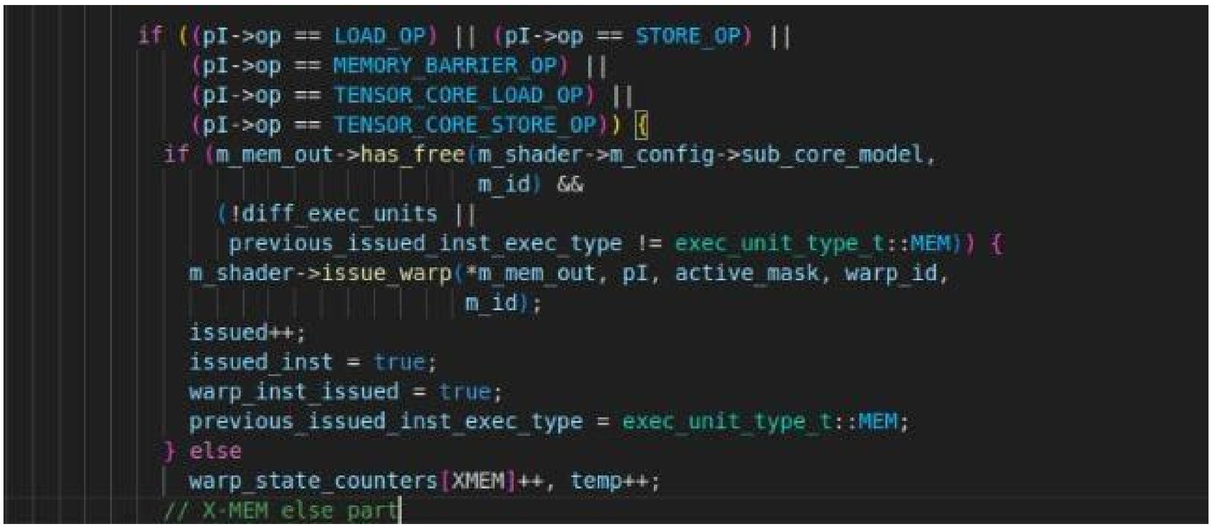
4. Modify the void scheduler_unit::cycle() function in the shader.cc file to include counters for different states of warps.

5. A warp can be categorised into the “Others” category, it must have its instruction buffer to be empty, or it must wait for some synchronisation instruction to get executed. So, increment the waiting_warp counter in both the if cases.



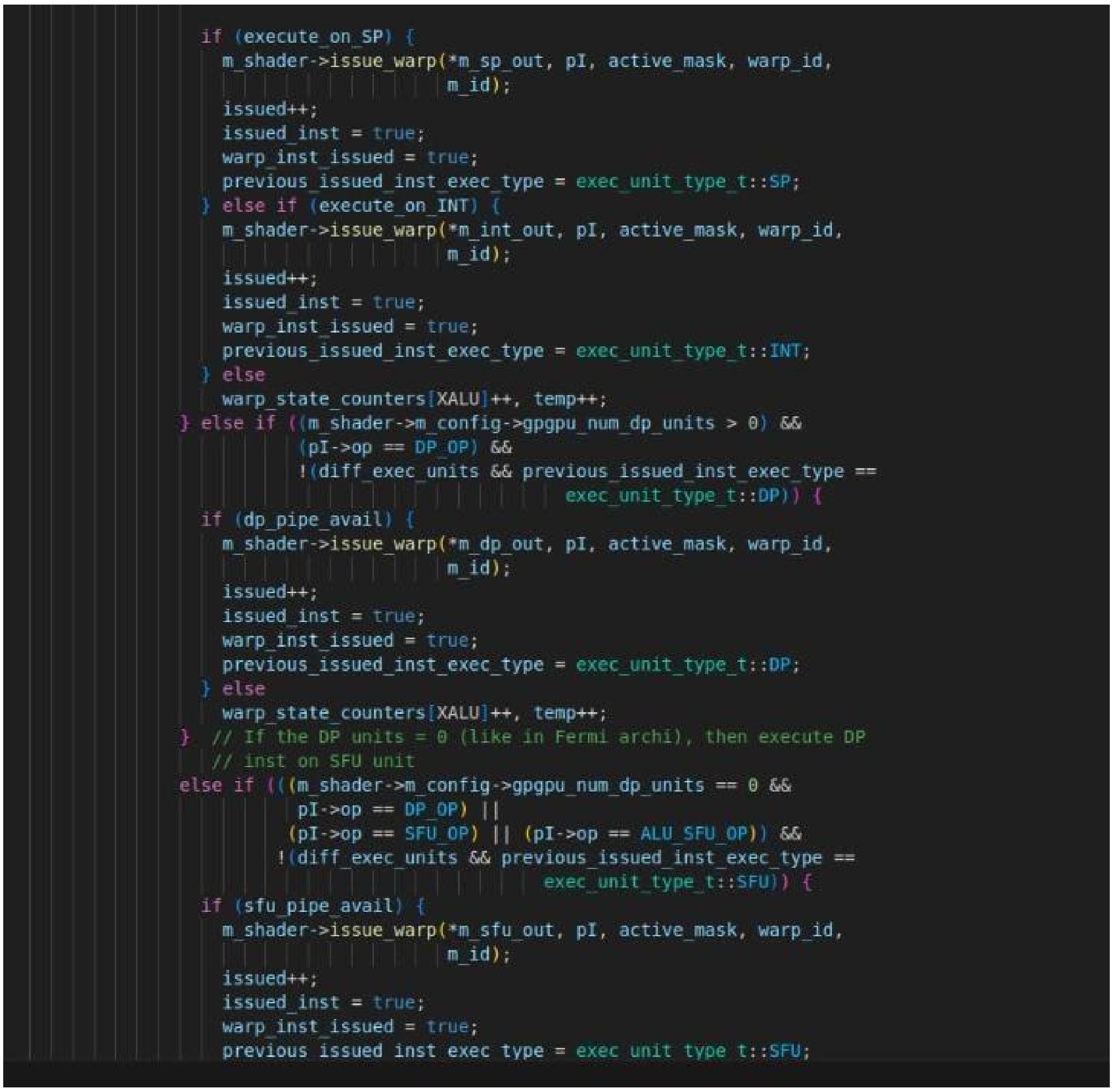
```
    if (warp(warp_id).ibuffer_empty()) {
      warp_state_counters[OTHER]++, temp++;
      SCHED_DPRINTF(
          "Warp (warp_id %u, dynamic_warp_id %u) fails as ibuffer_empty\n",
          (*iter)->get_warp_id(), (*iter)->get_dynamic_warp_id());
    }

    else if (warp(warp_id).waiting()) {
      warp_state_counters[OTHER]++, temp++;
      SCHED_DPRINTF(
          "Warp (warp_id %u, dynamic_warp_id %u) fails as waiting for "
          "barrier\n",
          (*iter)->get_warp_id(), (*iter)->get_dynamic_warp_id());
```

6. A warp can be categorised in the X_MEM state if it is ready to execute memory related instructions, but is not able to due to unavailability of resources. Thus, add another else condition to the if condition, and then increment the X_MEM counter.

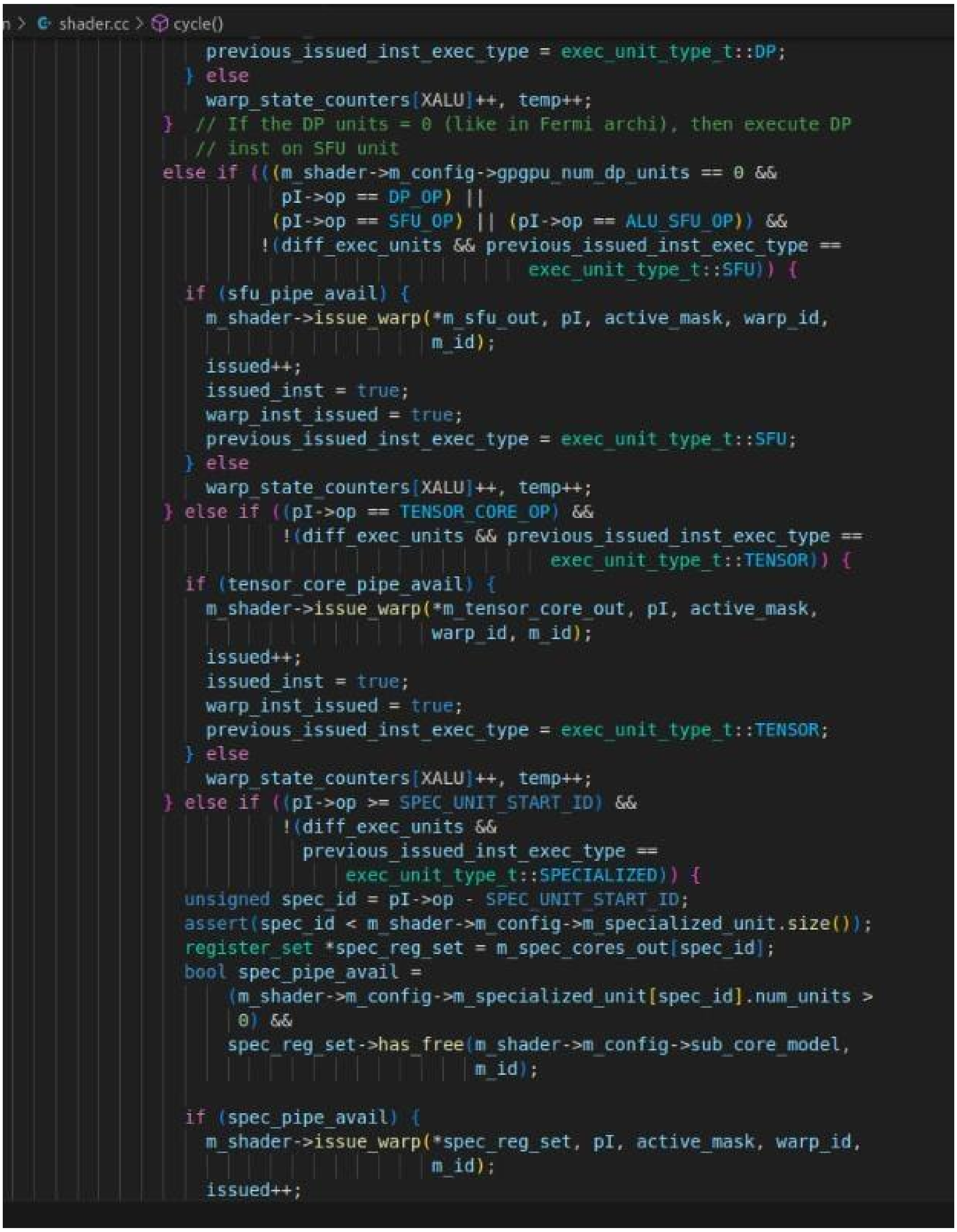


```
if ((pI->op == LOAD_OP) || (pI->op == STORE_OP) ||
    (pI->op == MEMORY_BARRIER_OP) ||
    (pI->op == TENSOR_CORE_LOAD_OP) ||
    (pI->op == TENSOR_CORE_STORE_OP)) {
  if (m_mem_out->has_free(m_shader->m_config->sub_core_model,
                          m_id) &&
      (!diff_exec_units ||
       previous_issued_inst_exec_type != exec_unit_type_t::MEM)) {
    m_shader->issue_warp(*m_mem_out, pI, active_mask, warp_id,
                         m_id);
    issued++;
    issued_inst = true;
    warp_inst_issued = true;
    previous_issued_inst_exec_type = exec_unit_type_t::MEM;
  } else
    warp_state_counters[XMEM]++, temp++;
  // X-MEM else part
```

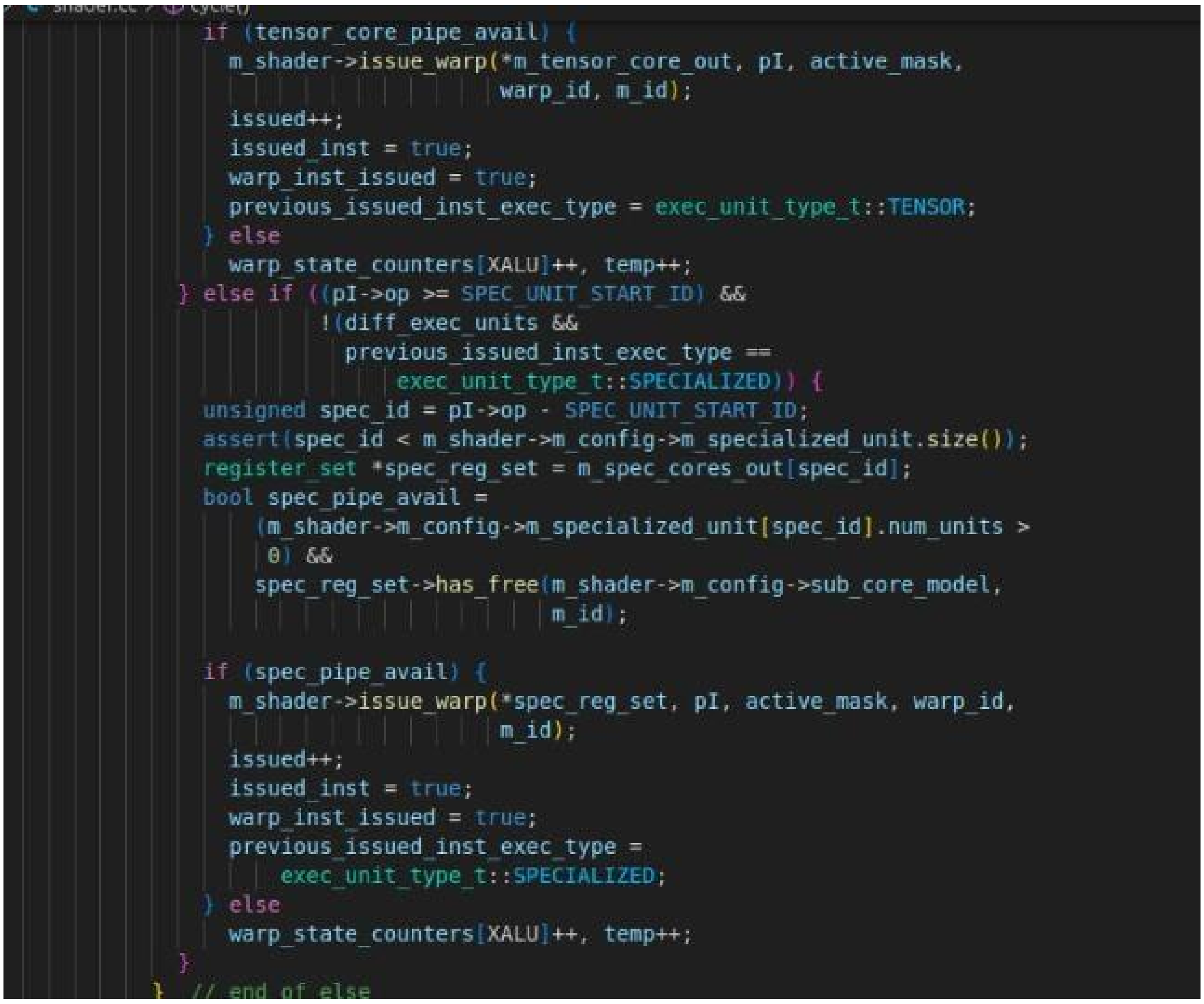
7. For a warp to be categorised in the X_ALU state, it is ready to execute arithmetic operations, but is unable to do so due to unavailability of resources(pipeline). So, to all the available(pipeline that executes arithmetic instructions) conditions, add another corresponding else part and increment the X_ALU counter.



```
  if (execute_on_SP) {
    m_shader->issue_warp(*m_sp_out, pI, active_mask, warp_id,
                         m_id);
    issued++;
    issued_inst = true;
    warp_inst_issued = true;
    previous_issued_inst_exec_type = exec_unit_type_t::SP;
  } else if (execute_on_INT) {
    m_shader->issue_warp(*m_int_out, pI, active_mask, warp_id,
                         m_id);
    issued++;
    issued_inst = true;
    warp_inst_issued = true;
    previous_issued_inst_exec_type = exec_unit_type_t::INT;
  } else
    warp_state_counters[XALU]++, temp++;
} else if ((m_shader->m_config->gpgpu_num_dp_units > 0) &&
           (pI->op == DP_OP) &&
           !(diff_exec_units && previous_issued_inst_exec_type ==
                                     exec_unit_type_t::DP)) {
  if (dp_pipe_avail) {
    m_shader->issue_warp(*m_dp_out, pI, active_mask, warp_id,
                         m_id);
    issued++;
    issued_inst = true;
    warp_inst_issued = true;
    previous_issued_inst_exec_type = exec_unit_type_t::DP;
  } else
    warp_state_counters[XALU]++, temp++;
}  // If the DP units = 0 (like in Fermi archi), then execute DP
   // inst on SFU unit
else if (((m_shader->m_config->gpgpu_num_dp_units == 0 &&
           pI->op == DP_OP) ||
          (pI->op == SFU_OP) || (pI->op == ALU_SFU_OP)) &&
         !(diff_exec_units && previous_issued_inst_exec_type ==
                                   exec_unit_type_t::SFU)) {
  if (sfu_pipe_avail) {
    m_shader->issue_warp(*m_sfu_out, pI, active_mask, warp_id,
                         m_id);
    issued++;
    issued_inst = true;
    warp_inst_issued = true;
    previous_issued_inst_exec_type = exec_unit_type_t::SFU;
```

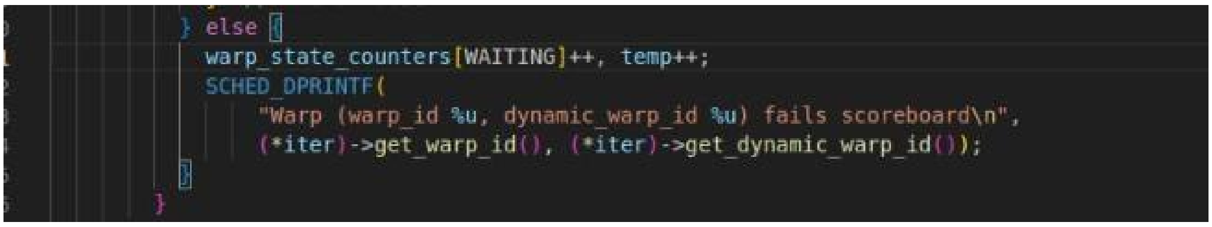


```
        previous_issued_inst_exec_type = exec_unit_type_t::DP;
      } else
        warp_state_counters[XALU]++, temp++;
    }  // If the DP units = 0 (like in Fermi archi), then execute DP
      // inst on SFU unit
    else if (((m_shader->m_config->gpgpu_num_dp_units == 0 &&
               pI->op == DP_OP) ||
              (pI->op == SFU_OP) || (pI->op == ALU_SFU_OP)) &&
             !(diff_exec_units && previous_issued_inst_exec_type ==
                                      exec_unit_type_t::SFU)) {
      if (sfu_pipe_avail) {
        m_shader->issue_warp(*m_sfu_out, pI, active_mask, warp_id,
                             m_id);
        issued++;
        issued_inst = true;
        warp_inst_issued = true;
        previous_issued_inst_exec_type = exec_unit_type_t::SFU;
      } else
        warp_state_counters[XALU]++, temp++;
    } else if ((pI->op == TENSOR_CORE_OP) &&
               !(diff_exec_units && previous_issued_inst_exec_type ==
                                        exec_unit_type_t::TENSOR)) {
      if (tensor_core_pipe_avail) {
        m_shader->issue_warp(*m_tensor_core_out, pI, active_mask,
                             warp_id, m_id);
        issued++;
        issued_inst = true;
        warp_inst_issued = true;
        previous_issued_inst_exec_type = exec_unit_type_t::TENSOR;
      } else
        warp_state_counters[XALU]++, temp++;
    } else if ((pI->op >= SPEC_UNIT_START_ID) &&
               !(diff_exec_units &&
                 previous_issued_inst_exec_type ==
                     exec_unit_type_t::SPECIALIZED)) {
      unsigned spec_id = pI->op - SPEC_UNIT_START_ID;
      assert(spec_id < m_shader->m_config->m_specialized_unit.size());
      register_set *spec_reg_set = m_spec_cores_out[spec_id];
      bool spec_pipe_avail =
          (m_shader->m_config->m_specialized_unit[spec_id].num_units >
           0) &&
          spec_reg_set->has_free(m_shader->m_config->sub_core_model,
                                 m_id);

      if (spec_pipe_avail) {
        m_shader->issue_warp(*spec_reg_set, pI, active_mask, warp_id,
                             m_id);
        issued++;
```



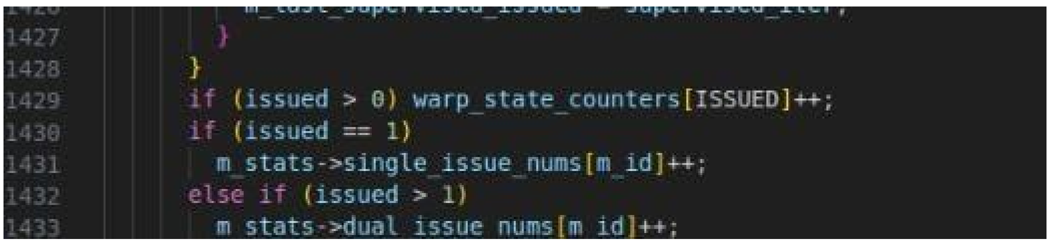
```
          if (tensor_core_pipe_avail) {
            m_shader->issue_warp(*m_tensor_core_out, pI, active_mask,
                                 warp_id, m_id);
            issued++;
            issued_inst = true;
            warp_inst_issued = true;
            previous_issued_inst_exec_type = exec_unit_type_t::TENSOR;
          } else
            warp_state_counters[XALU]++, temp++;
        } else if ((pI->op >= SPEC_UNIT_START_ID) &&
                   !(diff_exec_units &&
                     previous_issued_inst_exec_type ==
                         exec_unit_type_t::SPECIALIZED)) {
          unsigned spec_id = pI->op - SPEC_UNIT_START_ID;
          assert(spec_id < m_shader->m_config->m_specialized_unit.size());
          register_set *spec_reg_set = m_spec_cores_out[spec_id];
          bool spec_pipe_avail =
              (m_shader->m_config->m_specialized_unit[spec_id].num_units >
               0) &&
              spec_reg_set->has_free(m_shader->m_config->sub_core_model,
                                     m_id);

          if (spec_pipe_avail) {
            m_shader->issue_warp(*spec_reg_set, pI, active_mask, warp_id,
                                 m_id);
            issued++;
            issued_inst = true;
            warp_inst_issued = true;
            previous_issued_inst_exec_type =
                exec_unit_type_t::SPECIALIZED;
          } else
            warp_state_counters[XALU]++, temp++;
        }
      }  // end of else
```

8. For a warp to be categorised into the waiting state, it must have failed the scoreboard test.



```
        } else {
          warp_state_counters[WAITING]++, temp++;
          SCHED_DPRINTF(
              "Warp (warp_id %u, dynamic_warp_id %u) fails scoreboard\n",
              (*iter)->get_warp_id(), (*iter)->get_dynamic_warp_id());
        }
      }
```

9. For a warp to be categorised into the issued state, it must have already issued an instruction to the pipeline, so that it contributes towards the IPC count of the SM.



```
        m_last_supervised_issued = supervised_iter;
      }
    }
    if (issued > 0) warp_state_counters[ISSUED]++;
    if (issued == 1)
      m_stats->single_issue_nums[m_id]++;
    else if (issued > 1)
      m_stats->dual_issue_nums[m_id]++;
```

10. Thus, these counter values for different state of warps get updated for each cycle.

11. Save the shader.cc file, and go to the terminal and build the files in the gpgpu-sim_distribution directory.

12. Run benchmark applications in the terminal by navigating to the respective file location and by giving ”nvcc file.cu -lcudart“ in the terminal.

13. Note down the values that indicate the total number of warps in different states.

14. Plot a graph showing the different state of warps for different benchmark applications.

WARP STATE COUNTER STATISTICS:

For Mmul_new.cu:

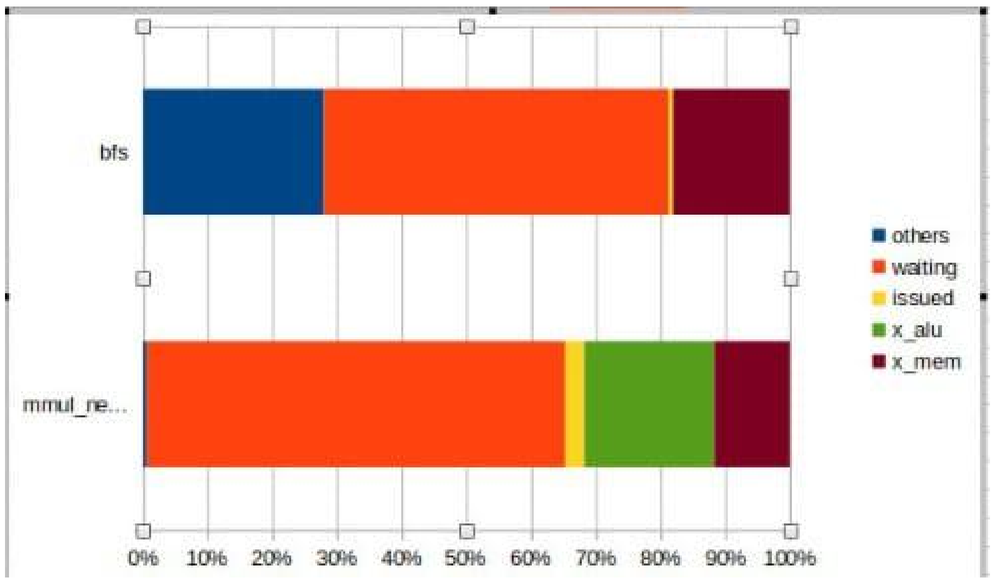
WAITING: 9994138
ISSUED: 460800
XALU: 3113953
XMEM: 1808769
OTHERS: 90340
SUM OF ALL COUNTERS:15468000
PRODUCT OF CYCLES AND WARPS: 15468000

WARP STATE COUNTER STATISTICS:

For BFS Benchmark:

WAITING: 189950915
ISSUED: 2324006
XALU: 5822
XMEM: 64481453
OTHERS: 99630940
SUM OF ALL COUNTERS: 356393136
PRODUCT OF CYCLES AND WARPS: 359393136

GRAPH TO DEPICT THE WARP STATE BREAKDOWN:


bfs
mmul_ne...
others
waiting
issued
x_alu
x_mem