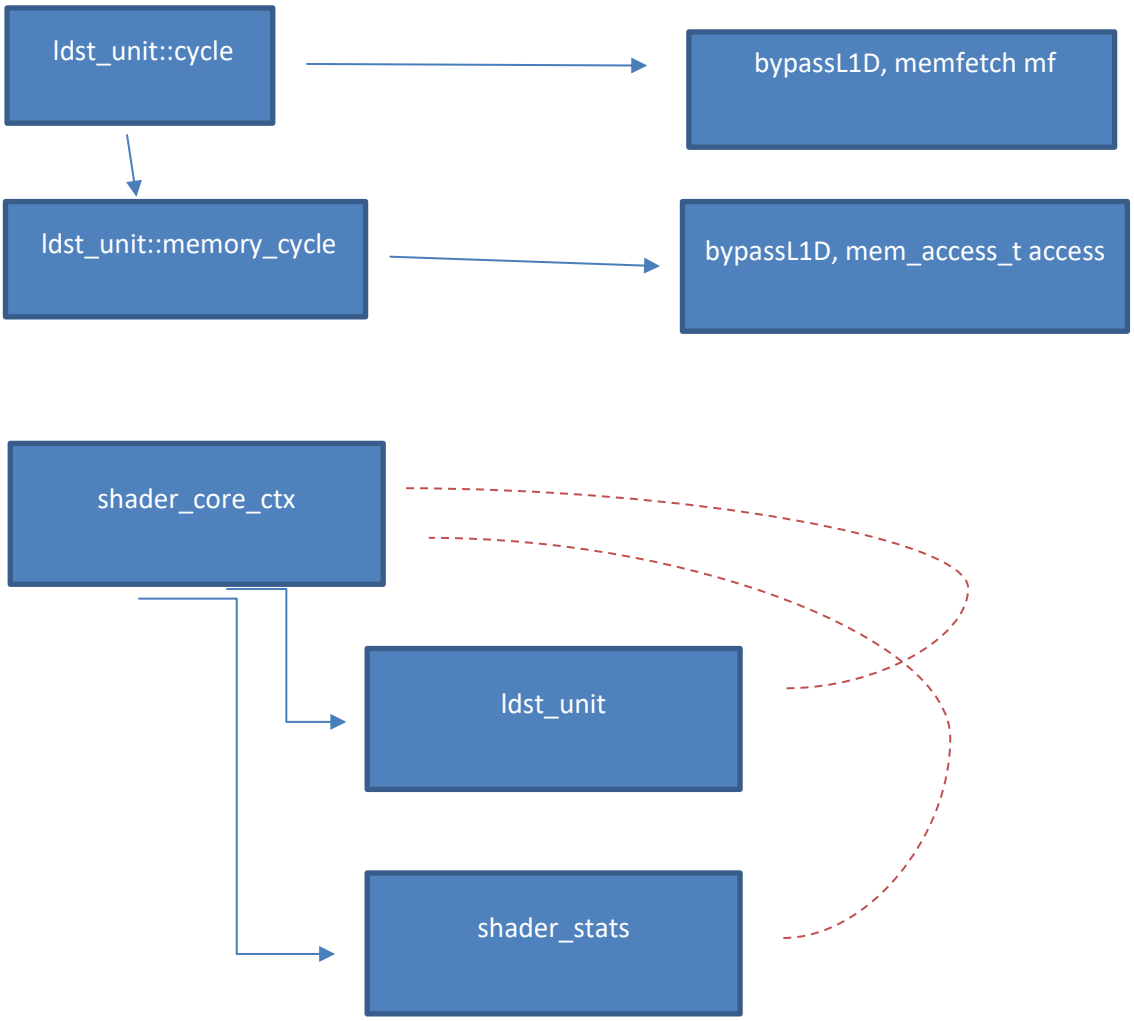
ECE786: PROGRAMMING ASSIGNMENT THREE - A

TASK: Load Bypass Mechanism

CODE ANALYSIS DIAGRAM



CODE ANALYSIS DISCUSSION

- The task requires to enforce L1 Data Cache bypass for loads with specific addresses. The load store part of gpgpu-sim is handled by the ldst_unit class that manages global, local, shared and texture memory accesses.
- The load instructions are issued for global and local memory. Investigating the methods of the ldst_unit class reveals two methods namely, memory_cycle() and cycle() that manage accesses to the L1 Data Cache within the simulator.
- The above diagram depicts the specific variables and class objects (bypassL1D, access, mf) that have been utilized to bypass the L1 Data Cache for the address range given in the assignment. This handles the first part of the assignment.
- It is also required to print the L1 bypasses for each kernel. Within a gpu each kernel is allocated to one SM. The SM is modelled by the shader_core_ctx class in the simulator.
- This SM class also contains its own instances of ldst_unit (m_ldst_uni)t and shader_stats (m_shader_stats). These classes also have a pointer to the SM object for which they are instantiated as depicted in the above diagram,

- Based on the prior information, the counters for each kernel can be modelled in the following two ways:
    - First, a counter can be declared within the shader_core_ctx class which will be updated in the memory_cycle and cycle methods of ldst_unit. This requires building public methods to access the counter as the shader_core_ctx pointer is protected member in ldst_unit. In a similar way, shader_stats class can access this counter in its print method to print the bypasses for each kernel.
    - Second, a cumulative counter and a static counter declared in the shader_stats class. The cumulative counter is updated in the ldst_unit methods. The print method of shader_stats will access the cumulative counter and subtract it with static counter to get the bypasses for each kernel. The static counter stores the prior cumulative counter value.
- Because of the simplicity of the code, the second method has been utilized to implement the count mechanism for the bypasses.

CODE SOLUTION SNIPPETS

```
for (unsigned i = 0; i < m_config->num_shader(); i++) {
  thread_icount_uarch += m_num_sim_insn[i];
  warp_icount_uarch += m_num_sim_winsn[i];
}
// ***********************************************************
// print the L1 bypasses - Aditya
// ***********************************************************

fprintf(fout, "******** L1 D$ Bypasses ******** : %d\n", this->count_cum_load_bypasses - sh
fprintf(fout, "******** L1 D$ Cum Bypasses ****** : %d\n", this->count_cum_load_bypasses);
shader_core_stats::count_load_bypasses_per_core= this->count_cum_load_bypasses;
fprintf(fout, "gpgpu_n_tot_thrd_icount = %lld\n", thread_icount_uarch);
fprintf(fout, "gpgpu_n_tot_w_icount = %lld\n", warp_icount_uarch);
```

```
class shader_core_stats : public shader_core_stats_pod {
 public:

  // ***************************************************************
  // counter to count load bypasses for each shader_core_ctx - Aditya
  // ***************************************************************
  long count_cum_load_bypasses;


  // ***************************************************************************
  // counter to keep count of previous load bypasses for each shader core ctx - Aditya
  // ***************************************************************************
  static long count_load_bypasses_per_core;
  shader_core_stats(const shader_core_config *config) {
  // ***************************************************************
  // Initialize counter to count load bypasses for each shader_core_ctx - Aditya
  // ***************************************************************
  count_cum_load_bypasses = 0;

    m_config = config;
    shader_core_stats_pod *pod = reinterpret_cast<shader_core_stats_pod *>(
        this->shader_core_stats_pod_start);
    memset(pod, 0, sizeof(shader_core_stats_pod));
    shader_cycles = (unsigned long long *)calloc(config->num_shader(),
                                                 sizeof(unsigned long long));
    m_num_sim_insn = (unsigned *)calloc(config->num_shader(), sizeof(unsigned));
```

```
// ************************************************************
// make and increment the counter for L1 Data Cache Load bypasses - Aditya
// ************************************************************
if(inst.is_load())
{
    new_addr_type addr = access.get_addr();
    if((addr >= 0xc0000000) && ( addr <= 0xc00fffff)){
      bypassL1D = true;
      this->m_stats->count_cum_load_bypasses++;
    }


  }
```

RESULT

Cumulative Load Bypasses	Load Bypasses for each kernel
26	26
163	137
963	800
5884	4921
34108	28224
181692	147584
609201	427509
671956	62755
674391	2435
674406	15

Note: Sample count for one of the kernels

```
distro:
995, 721, 1039, 795, 806, 807, 935, 795, 827, 911, 1085, 1156, 923, 1505, 692, 798, 851,
935, 828, 922, 998, 1069, 848, 829, 283, 204, 304, 40, 241, 241, 272, 294,
******** L1 D$ Bypasses ******** : 15
******** L1 D$ Cum Bypasses ****** : 674406
gpgpu_n_tot_thrd_icount = 69452416
gpgpu_n_tot_w_icount = 1888513
gpgpu_n_stall_shd_mem = 1284090
gpgpu_n_mem_read_local = 0
gpgpu_n_mem_write_local = 0
```