Assignment 1: Memory Management and Command Submission

Question: Modifying *gem_exec_basic.c* according to the given instructions.

Solution: The following is the overall edited code for the *gem_exec_basic.c* code provided with required additions mentioned in the assignment.

(1) For part-1 (*line 33*), I have added 1000/4000 MI_NOOP instructions to the batch buffer before the end instruction.

- I created a batch buffer called test_batchbuffer_1 and created another batch buffer object noop_bbe which adds 1000/4000 MI_NOOP instructions to the previous batch buffer object.
- To execute 1000/4000 instructions, select noop_inst_1/noop_inst_2 in the void loop accordingly.

(2) For part-2 (*line 114*), I added a similar batch buffer to execute it again after gem_sync command.

(3) For part-3 (*line 146*), to measure the time from first batch create to last sync, I used the gem_quiescent_gpu command.

Note (1): In particular places in the code, I have commented the question from the assignment which it refers to.

Note (2): I worked up the compilation process but there were some errors due to which it was unsuccessful. I have added the final repository with this pdf.

```
/*
 * Copyright    2016 Intel Corporation
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and associated documentation files (the "Software
   "),
 * to deal in the Software without restriction, including without
   limitation
 * the rights to use, copy, modify, merge, publish, distribute, sublicense
   ,
 * and/or sell copies of the Software, and to permit persons to whom the
 * Software is furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice (including the
   next
 * paragraph) shall be included in all copies or substantial portions of
   the
 * Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
    OR
```

```
16  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY
      ,
17  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT
      SHALL
18  * THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
      OTHER
19  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
20  * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
      DEALINGS
21  * IN THE SOFTWARE.
22  */
23
24 #include "igt.h"
25 #include "igt_collection.h"
26
27 #include "i915/gem_create.h"
28
29 IGT_TEST_DESCRIPTION("Basic sanity check of execbuf-ioctl rings.");
30
31 static uint32_t batch_create(int fd, uint32_t batch_size, uint32_t region)
32 {
33   //(For Q.1) Creating a batch buffer object
34   struct test_batchbuffer_1 bbe = {
35     .handle = gem_create( fd, batch_size),
36   };
37
38   //Inputting the # of MI_NOOP instructions
39   noop_inst_1 = 1000
40   noop_inst_2 = 4000
41   //We can replace noop_inst_1 by noop_inst_2 for executing 4000
       instructions before the end
42
43     //Now, we will create another batch buffer object which adds 1000/4000
       instructions to the previous batch buffer object
44   void noop_bbe(struct test_batchbuffer_1 *batch, noop_inst_2)
45     {
46    int loop;
47
48    igt_assert(batch);
49
50    BEGIN_BATCH(noop_inst_2 + 1, 0);
51    for(loop = 0; loop < noop_inst_2; loop++)
52     OUT_BATCH(MI_NOOP);
53    OUT_BATCH(MI_BATCH_BUFFER_END);
54
55    ADVANCE_BATCH();
56     }
57   const uint32_t bbe = MI_BATCH_BUFFER_END;
58   uint32_t handle;
59
60   handle = gem_create_in_memory_regions(fd, batch_size, region);
61   gem_write(fd, handle, 0, &bbe, sizeof(bbe));
```

```
62
63    return handle;
64  }
65
66  igt_main
67  {
68    const struct intel_execution_engine2 *e;
69    struct drm_i915_query_memory_regions *query_info;
70    struct igt_collection *regions, *set;
71    uint32_t batch_size;
72    const intel_ctx_t *ctx;
73    int fd = -1;
74
75    igt_fixture {
76      fd = drm_open_driver(DRIVER_INTEL);
77      ctx = intel_ctx_create_all_physical(fd);
78
79      /* igt_require_gem(fd); // test is mandatory */
80      igt_fork_hang_detector(fd);
81
82      query_info = gem_get_query_memory_regions(fd);
83      igt_assert(query_info);
84
85      set = get_memory_region_set(query_info,
86                  I915_SYSTEM_MEMORY,
87                  I915_DEVICE_MEMORY);
88    }
89
90    igt_subtest_with_dynamic("basic") {
91      for_each_combination(regions, 1, set) {
92        char *sub_name = memregion_dynamic_subtest_name(regions);
93        struct drm_i915_gem_exec_object2 exec;
94        uint32_t region = igt_collection_get_value(regions, 0);
95
96        batch_size = gem_get_batch_size(fd, MEMORY_TYPE_FROM_REGION(region))
      ;
97        memset(&exec, 0, sizeof(exec));
98        exec.handle = batch_create(fd, batch_size, region);
99
100       for_each_ctx_engine(fd, ctx, e) {
101         igt_dynamic_f("%s-%s", e->name, sub_name) {
102           struct drm_i915_gem_execbuffer2 execbuf = {
103             .buffers_ptr = to_user_pointer(&exec),
104             .buffer_count = 1,
105             .flags = e->flags,
106             .rsvd1 = ctx->id,
107           };
108
109           gem_execbuf(fd, &execbuf);
110         }
111       }
112       gem_sync(fd, exec.handle); /* catch any GPU hang */
```

```
113
114             //(For Q.2) Creating another batch buffer object
115           struct test_batchbuffer_2 bbe = {
116         .handle = gem_create( fd, batch_size),
117           };
118
119             //Now, we will create another batch buffer object which adds
120           void noop_bbe(struct test_batchbuffer_2 *batch, noop_inst_2)
121             {
122            int loop;
123
124           igt_assert(batch);
125
126           BEGIN_BATCH(noop_inst_2 + 1, 0);
127           for(loop = 0; loop < noop_inst_2; loop++)
128         OUT_BATCH(MI_NOOP);
129           OUT_BATCH(MI_BATCH_BUFFER_END);
130
131           ADVANCE_BATCH();
132             }
133
134       gem_close(fd, exec.handle);
135       free(sub_name);
136         }
137           }
138
139  igt_fixture {
140    free(query_info);
141    igt_collection_destroy(set);
142    igt_stop_hang_detector();
143    intel_ctx_destroy(fd, ctx);
144    close(fd);
145    }
146    //(For Q.3) To get the present clock times
147     gem_quiescent_gpu(fd);
148   clock_gettime(CLOCK_MONOTONIC, &start);
149   intel_batchbuffer_flush_on_ring(bbe, ringid);
150   clock_gettime(CLOCK_MONOTONIC, &end);
151 }
```

****

Soham Kulkarni, EE19BTECH11053, IIT Hyderabad