CS542200 Parallel Programming

Homework 4: Observe the behavior of UCX yourself

108062373 蘇勇誠

一、Overview

1. Identify how UCP Objects (ucp_context, ucp_worker, ucp_ep) interact through the API, including at least the following functions:

   a. ucp_init：Initialize Global Configuration。在 ucp_init 會將傳入的 paramenter 以及 ucp_context_config_table 中的 configuration 資訊存入 ucp_context，如：傳輸時所使用的 tag 為何、estimated end points 的數量、request size…等。此外，ucp_init 還會 intialize 所有可用的 TLS (transport layer)與 memory domain 之相關資訊至 ucp_context。

   b. ucp_worker_create : initialize ucp_worker object，將 user 設置的 thread mode 參數存入 ucp_worker，如：使用 single thread 或 multiple thread。若 user 指定的 mode 系統沒有支援，則會使用 default mode。接著，ucp_woker 會 initialize endpoint allocation 有關的資訊。此外，會根據 bitmap 決定使用哪個 TLS 進行傳輸。若沒有設置 bitmap，則會在所有 TLS 中找到最好的 TLS 並用該 TLS 進行傳輸。ucp_worker 亦會 initialize 可使用的 memory pool 之資訊。之後，ucp_worker 也會 initialize 傳輸時之 tag 資訊以及 active message。

   c. ucp_ep_create : 建立與其他 worker 的連線，連線方式可以是 client-server 的連線（ucp_ep_create_to_sock_addr 為 client 端之 client-server 連線建立，而 ucp_ep_create_api_conn_request 為 server 端之 client-server 連線建立），也可以是 remote memory acces 的方式進行連線（ucp_ep_create_api_to_worker_addr）。

2. What is the specific significance of the division of UCP Objects in the program? What important information do they carry?

   a. ucp_context：ucp_context 中存了 global resource 之資訊，以下列出 carry 重要的 information。如：TLS 之資訊、可使用的 memory 之 resouce 相關資訊、可以使用的 communication resource、可使用的 protocol。

   b. ucp_worker：ucp_context 存可用來 communication 的 resource，而 ucp_worker 中存的資訊為該 worker 要用來進行 communication 的 resource 與資訊。如：end point 之分配與連線建立相關資訊、track 可用的 atomic operation、該 worker 可以供其他 endpoint request 的 memory pool、該 worker 之 asynchronous communication handler、該 worker 之 event handler…等。

c. ucp_ep：存與 remote worker 連線相關資訊，如：與哪個 worker 建立連線、該 end point 是否要被 destory、end point 的 status、紀錄或 cache 在 network 所傳輸的 data…等。

3. Based on the description in HW4, where do you think the following information is loaded/created?

UCX_TLS：由於 UCX_TLS 為 global 的 configuration 資訊，因此可以推測該資訊會在 ucp_context 中被讀入。接著，在 trace code 時，看到了 ucp_init 時會呼叫 ucp_config_read 將 UCX_TLS 之資訊讀入 ucp_config_t 之中，證實了該想法。

TLS selected by UCX：基於原本的認知 TLS 在 end point 要建立連線時，才會根據傳輸 configuration 進行選擇適合的 TLS。因此，推測是在 ucp_ep 要建立連線時 UCX 才會選擇要用哪個 TLS。

二、Implementation

1. Which files did you modify, and where did you choose to print Line 1 and Line 2?

首先是 print line1 的部分，我在 ucp_worker.c 的 ucp_worker_get_ep_config 中，加入 ucp_config_read 取得 UCX TLS 之資訊(如下圖之 2078 行)。接著，會 call ucp_config_print（如下圖之 2079 行），透過 ucs/config/parser.c 中的 ucs_config_parser_print_opts 將取得的 TLS 資料印出來。

```
2034 ucs_status_t ucp_worker_get_ep_config(ucp_worker_h worker,
2035                                        const ucp_ep_config_key_t *key,
2036                                        unsigned ep_init_flags,
2037                                        ucp_worker_cfg_index_t *cfg_index_p)
2038 {
2039     ucp_context_h context = worker->context;
2040     ucp_worker_cfg_index_t ep_cfg_index;
2041     ucp_ep_config_t *ep_config;
2042     ucp_memtype_thresh_t *tag_max_short;
2043     ucp_lane_index_t tag_exp_lane;
2044     unsigned tag_proto_flags;
2045     ucs_status_t status;
2046+    ucp_config_t *config;
2047
2048     ucs_assertv_always(key->num_lanes > 0,
2049                        "empty endpoint configurations are not allowed");
2050
2051     /* Search for the given key in the ep_config array */
2052     ucs_array_for_each(ep_config, &worker->ep_config) {
2053         if (ucp_ep_config_is_equal(&ep_config->key, key)) {
2054             ep_cfg_index = ep_config - worker->ep_config.buffer;
2055             goto out;
2056         }
2057     }
2058
2059     /* Create new configuration */
2060     ucs_array_append(ep_config_arr, &worker->ep_config,
2061                      return UCS_ERR_NO_MEMORY);
2062     if (ucs_array_length(&worker->ep_config) >= UCP_WORKER_MAX_EP_CONFIG) {
2063         ucs_array_pop_back(&worker->ep_config);
2064         ucs_error("too many ep configurations: %d (max: %d)",
2065                   ucs_array_length(&worker->ep_config),
2066                   UCP_WORKER_MAX_EP_CONFIG);
2067         return UCS_ERR_EXCEEDS_LIMIT;
2068     }
2069
2070     ep_config = ucs_array_last(&worker->ep_config);
2071     status    = ucp_ep_config_init(worker, ep_config, key);
2072     if (status != UCS_OK) {
2073         return status;
2074     }
2075
2076     ep_cfg_index = ucs_array_length(&worker->ep_config) - 1;
2077+
2078+    ucp_config_read(NULL, NULL, &config);
2079+    ucp_config_print(config, stdout, NULL, UCS_CONFIG_PRINT_TLS);
2080
2081     if (ep_init_flags & UCP_EP_INIT_FLAG_INTERNAL) {
```

接著，在 ucs/config/parser.c 的 ucs_config_parser_print_opts 中，加入下左圖 1882～1884

行印出 UCX_TLS。由於 1882 行會需要用到 UCS_CONFIG_PRINT_TLS，因此我在 ucs/config/types.h 中的 ucs_config_print_flags_t 中加入 UCS_CONFIG_PRINT_TLS，如下右圖所示。1883 行會 call ucs_config_parser_get_value 將 TLS 之資訊讀出。1884 行會將 TLS 之資訊印出來

```
1853 void ucs_config_parser_print_opts(FILE *stream, const char *title, const void *opts,
1854                                    ucs_config_field_t *fields, const char *table_prefix,
1855                                    const char *prefix, ucs_config_print_flags_t flags)
1856 {
1857     ucs_config_parser_prefix_t table_prefix_elem;
1858     UCS_LIST_HEAD(prefix_list);
1859+    char str[100];
1860
1861     if (flags & UCS_CONFIG_PRINT_DOC) {
1862         fprintf(stream, "# UCX library configuration file\n");
1863         fprintf(stream, "# Uncomment to modify values\n");
1864     }
1865
1866     if (flags & UCS_CONFIG_PRINT_HEADER) {
1867         fprintf(stream, "\n");
1868         fprintf(stream, "#\n");
1869         fprintf(stream, "# %s\n", title);
1870         fprintf(stream, "#\n");
1871         fprintf(stream, "\n");
1872     }
1873
1874     if (flags & UCS_CONFIG_PRINT_CONFIG) {
1875         table_prefix_elem.prefix = table_prefix ? table_prefix : "";
1876         ucs_list_add_tail(&prefix_list, &table_prefix_elem.list);
1877         ucs_config_parser_print_opts_recurs(stream, opts, fields, flags,
1878                                             prefix, &prefix_list);
1879     }
1880
1881     // TODO: PP-HW4
1882+    if (flags & UCS_CONFIG_PRINT_TLS) {
1883+        ucs_config_parser_get_value(opts, fields, "TLS", str, 100 * sizeof(char));
1884+        fprintf(stream, "UCX_TLS=%s\n", str);
1885+    }
1886
1887     if (flags & UCS_CONFIG_PRINT_HEADER) {
1888         fprintf(stream, "\n");
1889     }
1890 }
```

```
85  /**
86   * Configuration printing flags
87   */
88  typedef enum {
89      UCS_CONFIG_PRINT_CONFIG          = UCS_BIT(0),
90      UCS_CONFIG_PRINT_HEADER          = UCS_BIT(1),
91      UCS_CONFIG_PRINT_DOC             = UCS_BIT(2),
92      UCS_CONFIG_PRINT_HIDDEN          = UCS_BIT(3),
93+     UCS_CONFIG_PRINT_COMMENT_DEFAULT = UCS_BIT(4),
94+     UCS_CONFIG_PRINT_TLS             = UCS_BIT(5)
95  } ucs_config_print_flags_t;
```

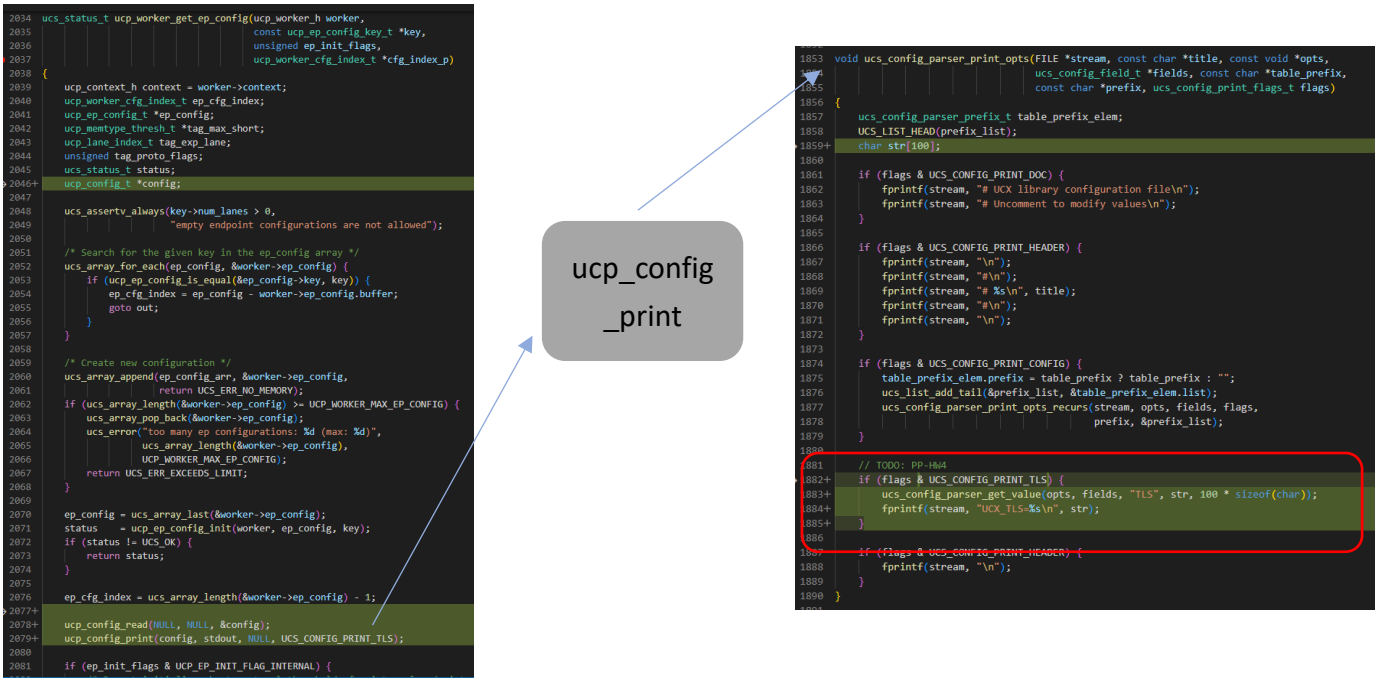
再來是 print line 2 的部分，下圖為由於 ucp_worker_print_used_tls function 之截圖。由於 ucp_worker_get_ep_config 會呼叫 ucp_worker_print_used_tls function，且 ucp_worker_print_used_tls function 中有計算出目前所使用的 tls 之資訊，因此直接將該資訊印出即可，如下右圖之 1856 行。

```
1763 static void
1764 ucp_worker_print_used_tls(ucp_worker_h worker, ucp_worker_cfg_index_t cfg_index)
1765 {
1766     const ucp_ep_config_key_t *key = &ucs_array_elem(&worker->ep_config,
1767                                                      cfg_index).key;
1768     ucp_context_h context          = worker->context;
1769     UCS_STRING_BUFFER_ONSTACK(strb, 256);
1770     ucp_lane_map_t tag_lanes_map    = 0;
1771     ucp_lane_map_t rma_lanes_map    = 0;
1772     ucp_lane_map_t amo_lanes_map    = 0;
1773     ucp_lane_map_t stream_lanes_map = 0;
1774     ucp_lane_map_t am_lanes_map     = 0;
1775     ucp_lane_map_t ka_lanes_map     = 0;
1776     int rma_emul                    = 0;
1777     int amo_emul                    = 0;
1778     int num_valid_lanes             = 0;
1779     ucp_lane_index_t lane;
1780
1781     ucp_ep_config_name(worker, cfg_index, &strb);
1782
1783     for (lane = 0; lane < key->num_lanes; ++lane) {
1784         if (key->lanes[lane].rsc_index == UCP_NULL_RESOURCE) {
1785             continue;
1786         }
1787
1788         if (key->am_lane == lane) {
1789             ++num_valid_lanes;
1790         }
1791
1792         if ((key->am_lane == lane) || (key->rkey_ptr_lane == lane) ||
1793             (ucp_ep_config_get_multi_lane_prio(key->am_bw_lanes, lane) >= 0)  ||
1794             (ucp_ep_config_get_multi_lane_prio(key->rma_bw_lanes, lane) >= 0)) {
1795             if (context->config.features & UCP_FEATURE_TAG) {
1796                 tag_lanes_map |= UCS_BIT(lane);
1797             }
1798
1799             if (context->config.features & UCP_FEATURE_AM) {
1800                 am_lanes_map |= UCS_BIT(lane);
1801             }
1802         }
1803
1804         if (key->tag_lane == lane) {
1805             /* tag_lane is initialized if TAG feature is requested */
1806             ucs_assert(context->config.features & UCP_FEATURE_TAG);
1807             tag_lanes_map |= UCS_BIT(lane);
1808         }
1809
1810         if ((key->am_lane == lane) &&
```

```
1811             (context->config.features & UCP_FEATURE_STREAM)) {
1812             stream_lanes_map |= UCS_BIT(lane);
1813         }
1814
1815         if (key->keepalive_lane == lane) {
1816             ka_lanes_map |= UCS_BIT(lane);
1817         }
1818
1819         if ((ucp_ep_config_get_multi_lane_prio(key->rma_lanes, lane) >= 0)) {
1820             rma_lanes_map |= UCS_BIT(lane);
1821         }
1822
1823         if ((ucp_ep_config_get_multi_lane_prio(key->amo_lanes, lane) >= 0)) {
1824             amo_lanes_map |= UCS_BIT(lane);
1825         }
1826     }
1827
1828     if (num_valid_lanes == 0) {
1829         return;
1830     }
1831
1832     if ((context->config.features & UCP_FEATURE_RMA) && (rma_lanes_map == 0)) {
1833         ucs_assert(key->am_lane != UCP_NULL_LANE);
1834         rma_lanes_map |= UCS_BIT(key->am_lane);
1835         rma_emul       = 1;
1836     }
1837
1838     if ((context->config.features & UCP_FEATURE_AMO) && (amo_lanes_map == 0) &&
1839         (key->am_lane != UCP_NULL_LANE)) {
1840         amo_lanes_map |= UCS_BIT(key->am_lane);
1841         amo_emul       = 1;
1842     }
1843
1844     ucp_worker_add_feature_rsc(context, key, tag_lanes_map, "tag", &strb);
1845     ucp_worker_add_feature_rsc(context, key, rma_lanes_map,
1846                                !rma_emul ? "rma" : "rma_am", &strb);
1847     ucp_worker_add_feature_rsc(context, key, amo_lanes_map,
1848                                !amo_emul ? "amo" : "amo_am", &strb);
1849     ucp_worker_add_feature_rsc(context, key, am_lanes_map, "am", &strb);
1850     ucp_worker_add_feature_rsc(context, key, stream_lanes_map, "stream", &strb);
1851     ucp_worker_add_feature_rsc(context, key, ka_lanes_map, "ka", &strb);
1852
1853     ucs_string_buffer_rtrim(&strb, "; ");
1854
1855     ucs_info("%s", ucs_string_buffer_cstr(&strb));
1856+    printf("%s\n", ucs_string_buffer_cstr(&strb));
1857 }
```

2. How do the functions in these files call each other? Why is it designed this way?

首先，在 ucp_worker.c 的 ucp_worker_get_ep_config 中，會先 call ucp_config_read 取得 UCX TLS 之資訊。接著會 call ucp_config_print，ucp_config_print 會 call ucs/config/parser.c 的 ucs_config_parser_print_opts 將 TLS 之資訊印出。之後，在 ucp_worker.c 的 ucp_worker_get_ep_config 會 call ucp_worker_print_used_tls，將目前所使用的 tls 之資訊印出。



由於透過 mpiucx -x UCX_LOG_LEVEL=info -np 2 ./mpi_hello.out 之指令觀察出 ucp_worker_print_used_tls 會印出目前所使用的 TLS 之資訊。接著，透過 trace code 發現 ucp_worker_get_ep_config 會 call ucp_worker_print_used_tls，並計算出目前所使用的 TLS 之資訊，因此可以透過 ucp_worker_print_used_tls function 用該 function 所計算出的資訊直接印出 line 2。有了 line 2 資訊之後，就只差 line 1 之資訊，因此我在 ucp_worker_get_ep_config call ucp_worker_print_used_tls 之前，將全部 TLS 之資訊印出就完成了此次 SPEC 的要求。

3. Observe when Line 1 and 2 are printed during the call of which UCP API?

在 ucp_worker_get_ep_config function call 時，會 print line 1 與 line 2，而 ucp_worker_get_ep_config 是在要建立與連線(wireup)前被 call 的，以取得連線時所需的 resource（ucp_worker_get_ep_config 被 ucp_ep_init_create_wireup function call 的）。

4. Does it match your expectations for questions 1-3? Why?

TLS selected by UCX：先前，推測是在 ucp_ep 要建立連線時 UCX 才會選擇要用哪個 TLS。Trace 完 code 之後，發現亦是如此，ucp_ep 在建立 connection 前，會 call ucp_worker_get_ep_config function 以取得被 UCX 選用的 TLS。

UCX_TLS：先前推測由於 UCX_TLS 為 global 的 configuration 資訊，因此該資訊會在 ucp_context 中被讀入，在 trace code 之後也確實是如此，在 ucp_init function 中會透過 ucp_config_read 將 UCX_TLS 讀到 ucp_context 之中。但由於 ucp_worker_get_ep_config 才會取得被 UCX 選用的 TLS，因此 UCX_TLS 印出的時機須在 ucp_worker_get_ep_config function 中，故在 ucp_worker_get_ep_config function 中透過 ucp_config_read API 取得所有的 TLS。

5. In implementing the features, we see variables like lanes, tl_rsc, tl_name, tl_device, bitmap, iface, etc., used to store different Layer's protocol information. Please explain what information each of them stores.

   lane 為目前傳輸的 state 為何，如下圖所示 state 可能是正在 remote memory access，可能是正在 check 是否可連線，可能是正在進行 atomic memory access…等。

```
16  typedef enum {
17      UCP_LANE_TYPE_FIRST,                  /* First item in enum */
18      UCP_LANE_TYPE_AM = UCP_LANE_TYPE_FIRST, /* Active messages */
19      UCP_LANE_TYPE_AM_BW,                  /* High-BW active messages */
20      UCP_LANE_TYPE_RMA,                    /* Remote memory access */
21      UCP_LANE_TYPE_RMA_BW,                 /* High-BW remote memory access */
22      UCP_LANE_TYPE_RKEY_PTR,               /* Obtain remote memory pointer */
23      UCP_LANE_TYPE_AMO,                    /* Atomic memory access */
24      UCP_LANE_TYPE_TAG,                    /* Tag matching offload */
25      UCP_LANE_TYPE_CM,                     /* CM wireup */
26      UCP_LANE_TYPE_KEEPALIVE,              /* Checks connectivity */
27      UCP_LANE_TYPE_LAST
28  } ucp_lane_type_t;
29
```

   tl_rsc 為 transport layer 相關資訊，如：tl_name、此 tl_rsc 可用的 device。
   tl_name 為 transport layer 的 name，如：ud_verbs。
   tl_device 為 transport layer 可用的 device，如：ethernet、infiniband。
   Bitmap 的功用為來檢視是否有該個 resource。
   Iface 為與網卡的 interface，管理 message 收送。

三、Optimize System

Below are the current configurations for OpenMPI and UCX in the system. Based on your learning, what methods can you use to optimize single-node performance by setting UCX environment variables?

```
-------------------------------------------------------------------
/opt/modulefiles/openmpi/4.1.5:

module-whatis     {Sets up environment for OpenMPI located in /opt/openmpi}
conflict          mpi
module            load ucx
setenv            OPENMPI_HOME /opt/openmpi
```

```
prepend-path      PATH /opt/openmpi/bin
prepend-path      LD_LIBRARY_PATH /opt/openmpi/lib
prepend-path      CPATH /opt/openmpi/include
setenv             UCX_TLS ud_verbs
setenv             UCX_NET_DEVICES ibp3s0:1
-------------------------------------------------------------------
```

Please use the following commands to test different data sizes for latency and bandwidth, to verify your ideas:

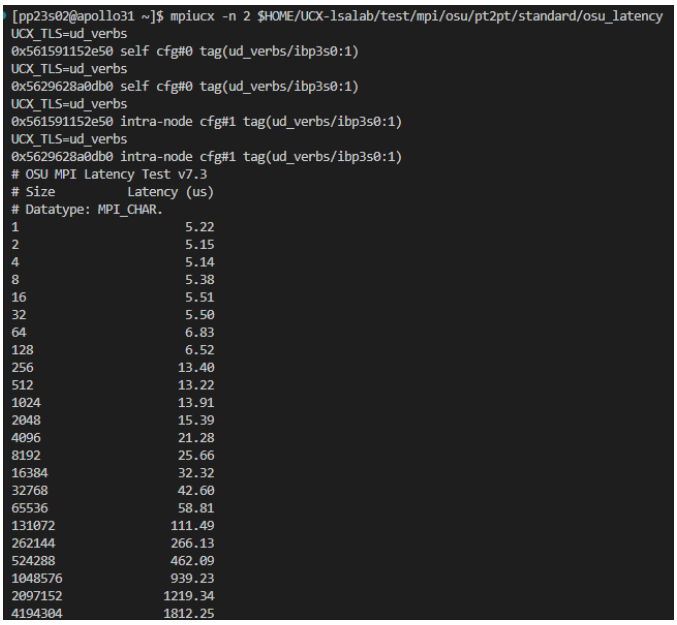
```
module load openmpi/4.1.5
mpiucx -n 2 $HOME/UCX-lsalab/test/mpi/osu/pt2pt/standard/osu_latency
mpiucx  -n  2  $HOME/UCX-lsalab/test/mpi/osu/pt2pt/standard/osu_bw
```

由於 ud_verbs 會使用 RDMA，而使用 RDMA 還是需要到 remote 將 memory copy 過來，因此會有 memory copy overhead，隨著 data size 增加，latency 會大幅度上升。而在 single node 可以使用 shared memory access，以減少 memory copy 的 overhead。
以下左圖為使用 shared memory 之 latency 結果，使用的 command 為 mpiucx -n 2 -x UCX_TLS=sm $HOME/UCX-lsalab/test/mpi/osu/pt2pt/standard/osu_latency。下右圖為使用 shared memory 之 bandwidth 結果，使用的 command 為 mpiucx -n 2 -x UCX_TLS=sm $HOME/UCX-lsalab/test/mpi/osu/pt2pt/standard/osu_bw。

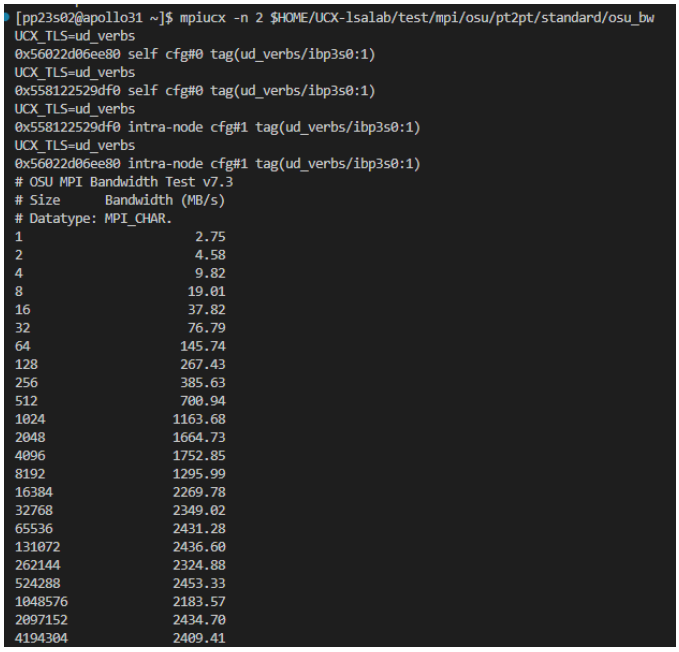
```
[pp23s02@apollo31 ~]$ mpiucx -n 2 -x UCX_TLS=sm $HOME/UCX-lsalab/test/mpi/osu/pt2pt/standard/osu_latency
UCX_TLS=sm
0x558ca4a12e30 self cfg#0 tag(sysv/memory cma/memory)
UCX_TLS=sm
0x558ca4a12e30 intra-node cfg#1 tag(sysv/memory cma/memory)
UCX_TLS=sm
0x557f0834ddb0 self cfg#0 tag(sysv/memory cma/memory)
UCX_TLS=sm
0x557f0834ddb0 intra-node cfg#1 tag(sysv/memory cma/memory)
# OSU MPI Latency Test v7.3
# Size          Latency (us)
# Datatype: MPI_CHAR.
1                    0.21
2                    0.22
4                    0.22
8                    0.21
16                   0.22
32                   0.26
64                   0.25
128                  0.38
256                  0.40
512                  0.43
1024                 0.51
2048                 0.67
4096                 0.99
8192                 1.71
16384                2.99
32768                7.08
65536                8.75
131072              18.22
262144              45.90
524288              70.69
1048576            131.00
2097152            303.46
4194304           1066.81
```

```
[pp23s02@apollo31 ~]$ mpiucx -n 2 -x UCX_TLS=sm $HOME/UCX-lsalab/test/mpi/osu/pt2pt/standard/osu_latency
UCX_TLS=sm
0x558ca4a12e30 self cfg#0 tag(sysv/memory cma/memory)
UCX_TLS=sm
0x558ca4a12e30 intra-node cfg#1 tag(sysv/memory cma/memory)
UCX_TLS=sm
0x557f0834ddb0 self cfg#0 tag(sysv/memory cma/memory)
UCX_TLS=sm
0x557f0834ddb0 intra-node cfg#1 tag(sysv/memory cma/memory)
# OSU MPI Latency Test v7.3
# Size          Latency (us)
# Datatype: MPI_CHAR.
1                    0.21
2                    0.22
4                    0.22
8                    0.21
16                   0.22
32                   0.26
64                   0.25
128                  0.38
256                  0.40
512                  0.43
1024                 0.51
2048                 0.67
4096                 0.99
8192                 1.71
16384                2.99
32768                7.08
65536                8.75
131072              18.22
262144              45.90
524288              70.69
1048576            131.00
2097152            303.46
4194304           1066.81
```

以下左圖為使用 default（ud_verbs)之 latency 結果，而以下右圖為使用 default（ud_verbs）之 bandwidth 結果。



```
[pp23s02@apollo31 ~]$ mpiucx -n 2 $HOME/UCX-lsalab/test/mpi/osu/pt2pt/standard/osu_latency
UCX_TLS=ud_verbs
0x561591152e50 self cfg#0 tag(ud_verbs/ibp3s0:1)
UCX_TLS=ud_verbs
0x5629628a0db0 self cfg#0 tag(ud_verbs/ibp3s0:1)
UCX_TLS=ud_verbs
0x561591152e50 intra-node cfg#1 tag(ud_verbs/ibp3s0:1)
UCX_TLS=ud_verbs
0x5629628a0db0 intra-node cfg#1 tag(ud_verbs/ibp3s0:1)
# OSU MPI Latency Test v7.3
# Size          Latency (us)
# Datatype: MPI_CHAR.
1                       5.22
2                       5.15
4                       5.14
8                       5.38
16                      5.51
32                      5.50
64                      6.83
128                     6.52
256                    13.40
512                    13.22
1024                   13.91
2048                   15.39
4096                   21.28
8192                   25.66
16384                  32.32
32768                  42.60
65536                  58.81
131072                111.49
262144                266.13
524288                462.09
1048576               939.23
2097152              1219.34
4194304              1812.25
```



```
[pp23s02@apollo31 ~]$ mpiucx -n 2 $HOME/UCX-lsalab/test/mpi/osu/pt2pt/standard/osu_bw
UCX_TLS=ud_verbs
0x56022d06ee80 self cfg#0 tag(ud_verbs/ibp3s0:1)
UCX_TLS=ud_verbs
0x558122529df0 self cfg#0 tag(ud_verbs/ibp3s0:1)
UCX_TLS=ud_verbs
0x558122529df0 intra-node cfg#1 tag(ud_verbs/ibp3s0:1)
UCX_TLS=ud_verbs
0x56022d06ee80 intra-node cfg#1 tag(ud_verbs/ibp3s0:1)
# OSU MPI Bandwidth Test v7.3
# Size      Bandwidth (MB/s)
# Datatype: MPI_CHAR.
1                       2.75
2                       4.58
4                       9.82
8                      19.01
16                     37.82
32                     76.79
64                    145.74
128                   267.43
256                   385.63
512                   700.94
1024                 1163.68
2048                 1664.73
4096                 1752.85
8192                 1295.99
16384                2269.78
32768                2349.02
65536                2431.28
131072               2436.60
262144               2324.88
524288               2453.33
1048576              2183.57
2097152              2434.70
4194304              2409.41
```

根據以上結果採用 shared memory 確實可以減少 memory copy 的時間，以減少 memory access 的 latency，以及提升 memory 的 bandwidth。

四、What have you learned from this homework?

透過本次這次 trace UCX code 的作業，讓我更了解 UCX 每個 layer 之功能與傳輸的架構。也透過這次作業更了解老師上課所教的 UCX 內容。也更加體會到硬體要有好的軟體搭配。雖然之前有 trace open source code 經驗，不過 UCX 的架構分明，讓人可以容易理解 UCX 之實作內容。