HW4 UCX

109062101 許佳綺

Overview

1. Identify how UCP Objects (`ucp_context`, `ucp_worker`, `ucp_ep`) interact through the API, including at least the following functions:
    - `ucp_init` : 負責UCP context initialization，一個context對應一個application，`ucp_init` 會檢查API version和network interface，初始ucp_context_t和一些需要的資源，並設定一些Transportation Protocols跟feature
    - `ucp_worker_create` : 為context create `ucp_worker`，`ucp_worker` 是context中實際用來communication的對象，其負責處理及記錄旗下的endpoint資訊和底層UCT的worker，一個UCP context可以有多個independent worker，每個worker可以在不同thread運作，達到 concurrent communication。
    - `ucp_ep_create`: 根據worker提供的資訊創建`ucp_ep`，endpoint將實際與remote endpoint連接並溝通。一個worker可以有多個endpoint，每個 Endpoint 可以與不同的遠端通信對象建立連接，實現與多個遠端的通信。
2. What is the specific significance of the division of UCP Objects in the program? What important information do they carry?
    - `ucp_context` : 保存global的配置和信息，用於管理整個communication環境。包含了有關communication環境的global configuration details, memory types, device information等。在initialization library 時，`ucp_context` 被創建，用於全局配置和資源管理。
    - `ucp_worker` : 實際communication的工人，即一個thread或workflow中的communication實體，負責維護endpoint跟具體的communication操作，包括Communication 所需要的 Initialize Connection，Resource Management (e.g. `uct_iface`, `ucp_ep`, etc)，Event Handling。
    - `ucp_ep` : communication的端點，即communication的目標或發起點。每個 `ucp_ep` 對象代表一個與他方建立的通信連接。包括對方的address、lanes、transport protocols、其他communication資料等。在`ucp_worker`在使用UCP communication時，會創建和操作 `ucp_ep` 。
3. Based on the description in HW4, where do you think the following information is loaded/created?
    - `UCX_TLS`: 在 **context 中被 loaded跟維護 的 environment variable**，再一路進去其他function add TL Resource
        - `ucp_init()` → `ucp_init_version()` → `ucp_fill_resources()` → `ucp_add_component_resources()` → `ucp_add_tl_resources()`
    - TLS selected by UCX: 在 **endpoint**
        - `ucp_ep_create()` → `ucp_ep_create_to_worker_addr()` → `ucp_wireup_init_lanes()` → `ucp_wireup_select_lanes()` → `ucp_wireup_search_lanes()` (為不同lane選擇protocols)
        - 在`ucp_wireup_search_lanes()`中會針對不同功能的lane (RMA / AMO / AM / TAG / RMA_BW / AM_BW 等等) 選擇transport protocol，例如:

          `ucp_wireup_add_rma_lanes()` → `ucp_wireup_add_rma_lanes()` → `ucp_wireup_add_memaccess_lanes()` → `ucp_wireup_select_transport()`
        - 最後走進 select.c 中的`ucp_wireup_select_transport()` 去決定

Implementation

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

   `ucp_worker.c`: 在 `ucp_worker_print_used_tls()` 中 invoke `ucp_config_print()`

   在進入 `ucp_worker_print_used_tls()` 後，由於這個function就是來parse系統使用的TLS (即我們要的Line2資訊)，因此將這個function處理完的strb傳入 `ucp_config_print()` 。

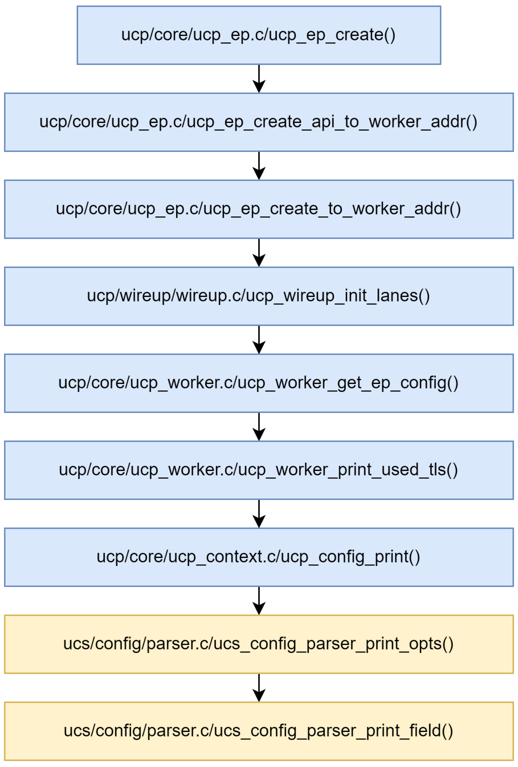
   ```
   // worker.c
   ucp_config_read(NULL, NULL, &config);
   ucp_config_print(config, stdout,
                                   ucs_string_buffer_cstr(&strb), UCS_CONFIG_
   ucp_config_release(config);
   ```

   接著進入 `ucp_config_print()` 後會再進入到 `ucs_config_parser_print_opts()` (也就是TODO: PP-HW4的部分)，在這邊新增一種 if-else 情況並呼叫 `ucs_config_parser_print_field()` 去print特定的欄位，也就是TLS欄位(即為我們需要的Line1資訊)。

   ```
   // parser.c
   if (flags & UCS_CONFIG_PRINT_TLS) {
       table_prefix_elem.prefix = table_prefix ? table_prefix : "";
       ucs_list_add_tail(&prefix_list, &table_prefix_elem.list);
       // field[4] records the TLS variables
       ucs_config_parser_print_field(stream, opts, prefix, &prefix_list,
                                   fields[4].name, &fields[4], flags, NULL);
       fprintf(stream, "%s\n", title);
   }
   ```

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

   以下的圖說明了這些function是如何呼叫彼此的，藍色表示都在ucp folder下，橘色表示都在ucs folder底下。這樣的呼叫模式驗證了UCX 設計切分成UCS(應用端), UCP(High Level API), UCT(Low Level API)的功能，print out 東西本身就是在應用端，因此UCP部分的endpoint跟worker若想要輸出東西，則呼叫UCS中的parser去parse並輸出。


ucp/core/ucp_ep.c/ucp_ep_create()
ucp/core/ucp_ep.c/ucp_ep_create_api_to_worker_addr()
ucp/core/ucp_ep.c/ucp_ep_create_to_worker_addr()
ucp/wireup/wireup.c/ucp_wireup_init_lanes()
ucp/core/ucp_worker.c/ucp_worker_get_ep_config()
ucp/core/ucp_worker.c/ucp_worker_print_used_tls()
ucp/core/ucp_context.c/ucp_config_print()
ucs/config/parser.c/ucs_config_parser_print_opts()
ucs/config/parser.c/ucs_config_parser_print_field()

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

   由上圖可知其中間經過哪些function而print出來的，而追本朔源是 `ucp_ep_create()` 這個 API被呼叫時被print的。

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

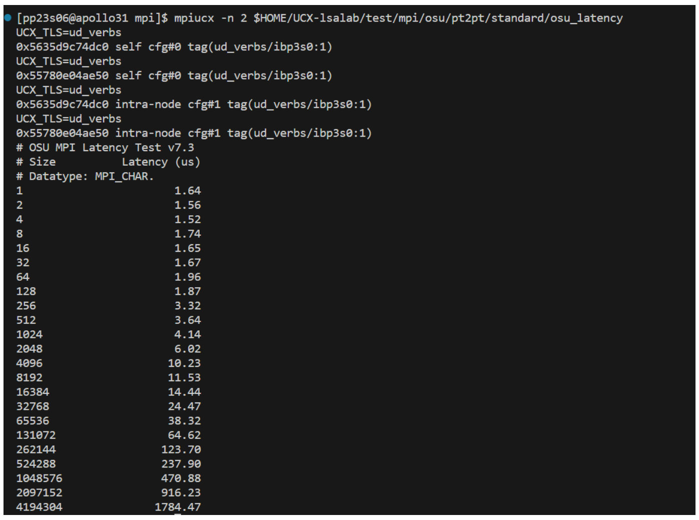
   確實有符合上述的過程。

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.

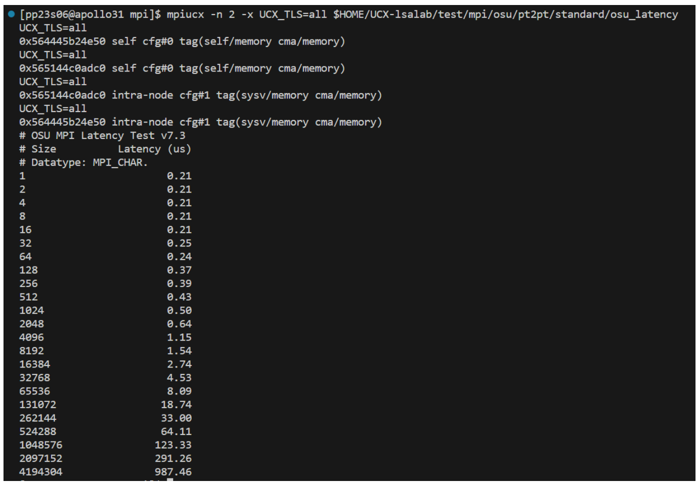
   - **lanes (通信通道)**: 一種抽象概念，讓兩個endpoint溝通的通道，該struct記錄像是bandwidth, Memory Domain, Transport Protocol等資訊，而lanes又分成以下好幾種功能的lane
     - AM（Active Message）lane / RMA / AMO / TAG / RMA_BW / AM_BW / Keepalive
   - **tl_rsc (Transport Resource):** 紀錄系統上 actual and available transport resources 的資訊 (包含名字，硬體type等等)，像是context中紀錄的tl_rsc實際上他的struct是記錄這些東西
   - **tl_name (Transport Name):** 紀錄傳輸協議的名字
   - **tl_device (Transport Device)**: 紀錄處理傳輸層功能的具體硬體或軟體，例如網卡
   - **bitmap**: 用0或1來標記array中每個位置是否可支援某些特定的功能或屬性或是否可使用，該array可以是一個TL resourses array, device arrray等等。
   - **iface (interface):** 紀錄跟connection相關的interface，這個interface是相對於最底層的device, TL resource等，那些傳輸層接口的資料 (包括API的屬性、能力等)，例如get, put, connection establishment等等功能。

Optimize System

1. 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?
   - 以single node來說，將UCX_TLS設置成shared memory(sm)跟self會更快，此外我也將其設置成all的話，側了幾次後發現都都是差不多的加速效果，而其中又以all平均上來說最快，因此以下放all的測試結果。
   - 設置UCX_TLS=all讓其找出最佳的TLS，如此一來latency會可以比原本設置udverbs降低2~8倍，並且bandwidth也可以上升3~4倍左右。
     - latency (上:原本，下:=all)

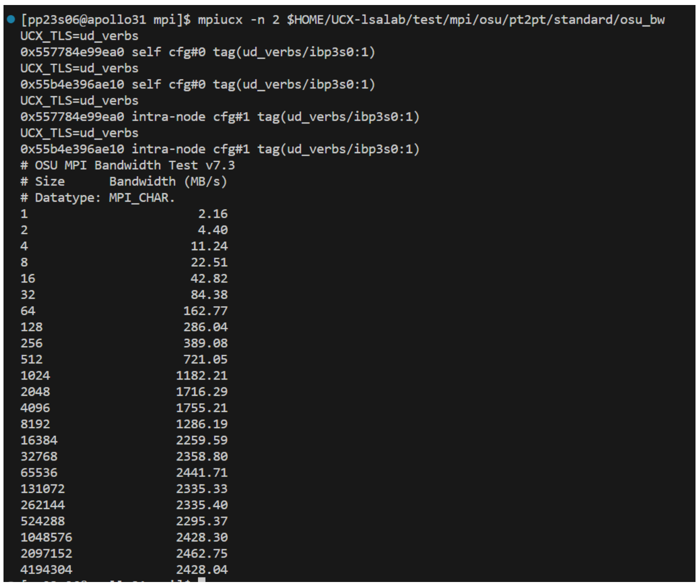


```
[pp23s06@apollo31 mpi]$ mpiucx -n 2 $HOME/UCX-lsalab/test/mpi/osu/pt2pt/standard/osu_latency
UCX_TLS=ud_verbs
0x5635d9c74dc0 self cfg#0 tag(ud_verbs/ibp3s0:1)
UCX_TLS=ud_verbs
0x55780e04ae50 self cfg#0 tag(ud_verbs/ibp3s0:1)
UCX_TLS=ud_verbs
0x5635d9c74dc0 intra-node cfg#1 tag(ud_verbs/ibp3s0:1)
UCX_TLS=ud_verbs
0x55780e04ae50 intra-node cfg#1 tag(ud_verbs/ibp3s0:1)
# OSU MPI Latency Test v7.3
# Size          Latency (us)
# Datatype: MPI_CHAR.
1                       1.64
2                       1.56
4                       1.52
8                       1.74
16                      1.65
32                      1.67
64                      1.96
128                     1.87
256                     3.32
512                     3.64
1024                    4.14
2048                    6.02
4096                   10.23
8192                   11.53
16384                  14.44
32768                  24.47
65536                  38.32
131072                 64.62
262144                123.70
524288                237.90
1048576               470.88
2097152               916.23
4194304              1784.47
```

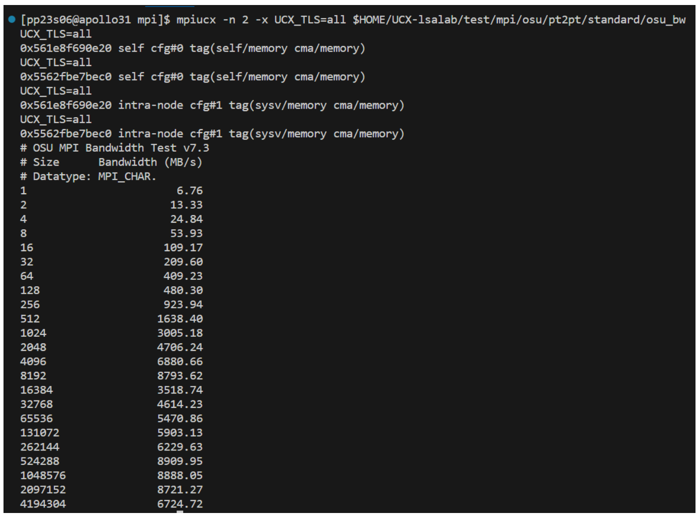


```
[pp23s06@apollo31 mpi]$ mpiucx -n 2 -x UCX_TLS=all $HOME/UCX-lsalab/test/mpi/osu/pt2pt/standard/osu_latency
UCX_TLS=all
0x564445b24e50 self cfg#0 tag(self/memory cma/memory)
UCX_TLS=all
0x565144c0adc0 self cfg#0 tag(self/memory cma/memory)
UCX_TLS=all
0x565144c0adc0 intra-node cfg#1 tag(sysv/memory cma/memory)
UCX_TLS=all
0x564445b24e50 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                       0.21
4                       0.21
8                       0.21
16                      0.21
32                      0.25
64                      0.24
128                     0.37
256                     0.39
512                     0.43
1024                    0.50
2048                    0.64
4096                    1.15
8192                    1.54
16384                   2.74
32768                   4.53
65536                   8.09
131072                 18.74
262144                 33.00
524288                 64.11
1048576               123.33
2097152               291.26
4194304               987.46
```

     - bandwidth (上:原本，下:=all)



```
[pp23s06@apollo31 mpi]$ mpiucx -n 2 $HOME/UCX-lsalab/test/mpi/osu/pt2pt/standard/osu_bw
UCX_TLS=ud_verbs
0x557784e99ea0 self cfg#0 tag(ud_verbs/ibp3s0:1)
UCX_TLS=ud_verbs
0x55b4e396ae10 self cfg#0 tag(ud_verbs/ibp3s0:1)
UCX_TLS=ud_verbs
0x557784e99ea0 intra-node cfg#1 tag(ud_verbs/ibp3s0:1)
UCX_TLS=ud_verbs
0x55b4e396ae10 intra-node cfg#1 tag(ud_verbs/ibp3s0:1)
# OSU MPI Bandwidth Test v7.3
# Size      Bandwidth (MB/s)
# Datatype: MPI_CHAR.
1                       2.16
2                       4.40
4                      11.24
8                      22.51
16                     42.82
32                     84.38
64                    162.77
128                   286.04
256                   389.08
512                   721.05
1024                 1182.21
2048                 1716.29
4096                 1755.21
8192                 1286.19
16384                2259.59
32768                2358.80
65536                2441.71
131072               2335.33
262144               2335.40
524288               2295.37
1048576              2428.30
2097152              2462.75
4194304              2428.04
```



```
[pp23s06@apollo31 mpi]$ mpiucx -n 2 -x UCX_TLS=all $HOME/UCX-lsalab/test/mpi/osu/pt2pt/standard/osu_bw
UCX_TLS=all
0x561e8f690e20 self cfg#0 tag(self/memory cma/memory)
UCX_TLS=all
0x5562fbe7bec0 self cfg#0 tag(self/memory cma/memory)
UCX_TLS=all
0x561e8f690e20 intra-node cfg#1 tag(sysv/memory cma/memory)
UCX_TLS=all
0x5562fbe7bec0 intra-node cfg#1 tag(sysv/memory cma/memory)
# OSU MPI Bandwidth Test v7.3
# Size      Bandwidth (MB/s)
# Datatype: MPI_CHAR.
1                       6.76
2                      13.33
4                      24.84
8                      53.93
16                    109.17
32                    209.60
64                    409.23
128                   480.30
256                   923.94
512                  1638.40
1024                 3005.18
2048                 4706.24
4096                 6880.66
8192                 8793.62
16384                3518.74
32768                4614.23
65536                5470.86
131072               5903.13
262144               6229.63
524288               8909.95
1048576              8888.05
2097152              8721.27
4194304              6724.72
```

Experience & Conclusion

這次的 trace 龐大的UCX project真的花了好多時間，理解每個function間的呼叫花了不少功夫。透過這次作業，我更了解老師上課提及的UCX架構，也更理解底層通訊方式的一些設計思路，對於實際溝通流程到底是怎麼建立的也更加清楚。一開始要implement時會不太確定到底哪些variable是應該拿來用的，但漸漸trace更深入後會了解每個variable的含意跟儲存的資訊，真的覺得從設計到實作出這樣的project的過程讓人嘆為觀止，也非常感謝助教跟教授的耐心教導!