PP HW 4 Report

> Please include both brief and detailed answers.The report should be based on the UCX code.Describe the code using the 'permalink' from GitHub repository.

1. Overview

> In conjunction with the UCP architecture mentioned in the lecture, please read ucp_hello_world.c

1. Identify how UCP Objects (`ucp_context`, `ucp_worker`, `ucp_ep`) interact through the API, including at least the following functions:
    - `ucp_init`
        - 對ucp_context做初始化，包含config的讀取設定、分配記憶體、遇到錯誤的對應訊息處理等
        - upc_context會被當作一個參數餵入ucp_init中，透過param和ucp_params對其初始化。
    - `ucp_worker_create`
        - 此function用於創建UCX對應的工作線程，包含分配ucp_worker_t結構的memory，初始化多個hash table，創建底層uct worker等
        - 此function會為ucp_context根據ucp_params創建對應worker
    - `ucp_ep_create`
        - ucp_ep分別有sever_ep以及client_ep，分別在run_ucx_client以及run_ucx_server被宣告，代表client以及server worker會創建對應的end points，在此之前server/ client會透過socket programming傳送指定address
        - ucp_ep_create會為指定worker創建對應end points，會根據flag以及params的不同創建對應類型的end point

2. What is the specific significance of the division of UCP Objects in the program? What important information do they carry?
    - `ucp_context`
        - 一個應用程序通常只應該創建一個UCP context，它管理著整個communication environment。
        - 其決定了transport resource、獲取transport component並根據程序提供的params和config初始化ucp context
    - `ucp_worker`
        - 負責處理和管理特定的communication task。
        - 其管理了thread mode、初始化多個linked list用來存放end point、rkey pointer等資訊，並創建了對應的底層uct worker
    - `ucp_ep`
        - end point代表通信的目標，可以是另一個application的worker等
        - 透過檢查 params 中的 flags 和 field_mask(CLIENT_SERVER、CONN_REQUEST、REMOTE_ADDRESS)，end point有不同的初始化方式
3. Based on the description in HW4, where do you think the following information is loaded/created?
    - `UCX_TLS`
        - 我認為在create context時就會透過讀取config將所有可用的TLS加入環境變數中，供未來create worker/ endpoint時可用。
    - TLS selected by UCX
        - 不同worker可以有不同的TLS，但同一個worker的TLS有相同的TLS，因此我認為TLS selected by UCX是在create worker時被建立的。

2. Implementation

> Describe how you implemented the two special features of HW4.

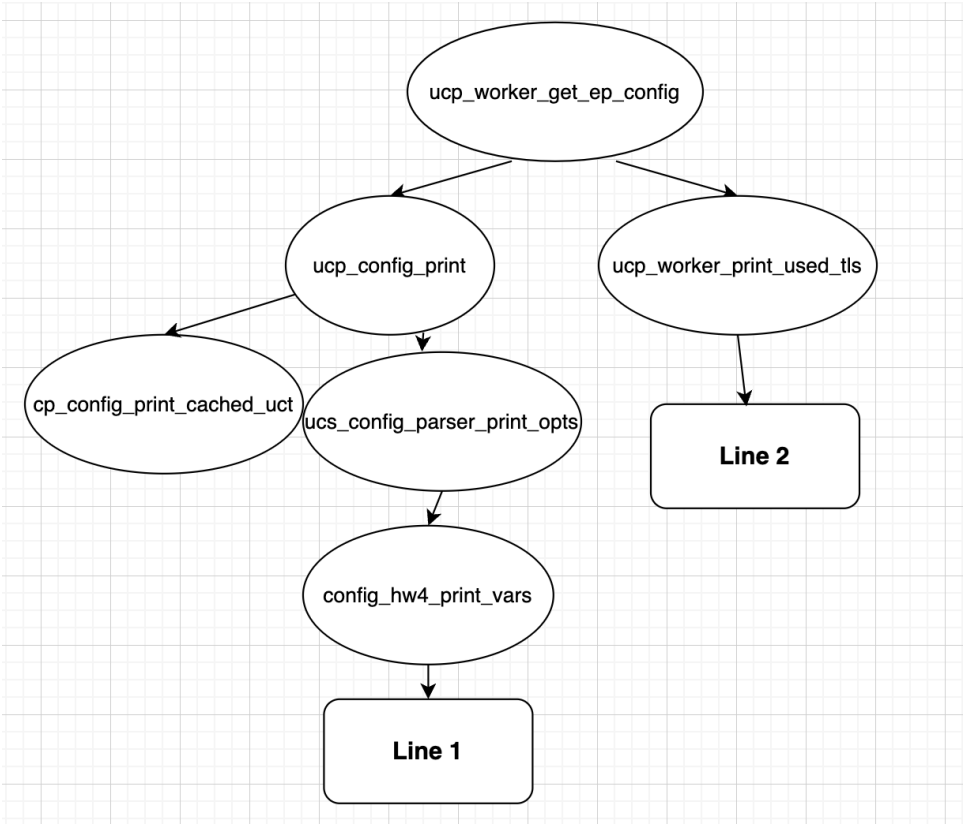
1. Which files did you modify, and where did you choose to print Line 1 and Line 2?
    - 為了輸出Line 1和Line 2，我修改了多個檔案的function，在下面一一說明

- ucp_worker.c中的ucp_worker_get_ep_config function: 其用於獲取endpoints的相關資訊，我在此function的最後判斷若其未使用簡短協議則讀取其config並輸出TLS協議的config information，再透過ucp_worker_print_used_tls 印出已使用的 TLS。
- parser,c中的ucs_config_parser_print_optsfunction: ucp_config_print會呼叫此function，因此我修改了ucs_config_parser_print_opts，在TODO: PP-HW4中呼叫自定義的config_hw4_print_vars("UCX_TLS");

```
1  void config_hw4_print_vars(const char *prefix){
2      char **envp, *envstr;
3      size_t prefix_len;
4      char *var_name;
5      khiter_t iter;
6      char *saveptr;
7      prefix_len      = strlen(prefix);
8      pthread_mutex_lock(&ucs_config_parser_env_vars_hash_lock);
9      for (envp = environ; *envp != NULL; ++envp) {
10         envstr = ucs_strdup(*envp, "env_str");
11         if (envstr == NULL) {
12             continue;
13         }
14         var_name = strtok_r(envstr, "=", &saveptr);
15         if (!var_name || strncmp(var_name, prefix, prefix_len)) {
16             ucs_free(envstr);
17             continue; /* Not UCX */
18         }
19
20         iter = kh_get(ucs_config_env_vars, &ucs_config_parser_env_vars, var_name);
21         if (iter == kh_end(&ucs_config_parser_env_vars)) {
22             if (ucs_global_opts.warn_unused_env_vars) {
23             }else{
24                 printf("%s \n", *envp);
25             }
26         }
27         ucs_free(envstr);
28     }
29     pthread_mutex_unlock(&ucs_config_parser_env_vars_hash_lock);
30 }
```

- 此function是根據ucs_config_parser_print_env_varsfunction修改而成，根據prefix調整搜尋特定的環境參數並輸出，本次我們需要輸出的參數為"UCX_TLS"，透過此function可輸出Line 1
- ucp_worker.c 中的ucp_worker_print_used_tls: 此function會將line2所需要之info加入strb中，原本透過ucs_info將其輸出，我們也透過printf即可印出line2內容

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



- 因為每一個endpoint都會有其傳輸設定，transport protocol、UCP資源配置等，因此只要trace 此程式如何在UCX_LOG_LEVEL=info印出即可透過類似方式將其輸出。

3. Observe when Line 1 and 2 are printed during the call of which UCP API?
    - Line 1是在自定義的config_hw4_print_vars印出，透過事先選好的prefix輸出指定資訊
    - Line 2透過ucp_worker_print_used_tls，將所需資訊加入strb後印出
4. Does it match your expectations for questions **1-3**? Why?
    - 與預期相符，只是要注意的是本次作業要求每個endpoint輸出自己對應的TLS，因此不能在create worker時就印出TLS資訊，需等init endpoint再進行對應的資訊輸出。
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

    - lanes中的每個成員代表一個通信通道，其描述了通信通道的相關資訊，如對應index、通信類型、segment 大小等
- tl_rsc
    - UCT的resource descriptor，包含Transport name、Hardware device name等資訊
- tl_name
    - tl_name指的是UCP中的transport layer name，在UCP中多個傳輸層中每種傳輸層都有一個名稱，用來標識該傳輸層。此名稱可供UCP識別在不同應用場景的需求中選擇和配置不同的傳輸層。
- tl_device
    - 此feature儲存一個傳輸層的裝置資源，描述關於底層以及通信設備的基本資訊，包含設備名稱，裝置類型等
- bitmap
    - 在此framework中，bitmap用來表示UCP哪些transport layer正在被使用，每個bit對應到一個資源，透過bitmap來追蹤所有layer使用情況。
- iface
    - iface是UCT中的一個介面，用來實現兩個端點之間的通信，擁有許多關於通信方面的資訊，而iface_attr則包含了關於通信interface的各種屬性，包括bandwidth、device priority、max eps等

3. 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?

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

module-whatis   {Sets up environment for OpenMPI located in /opt
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
-------------------------------------------------------------------
```

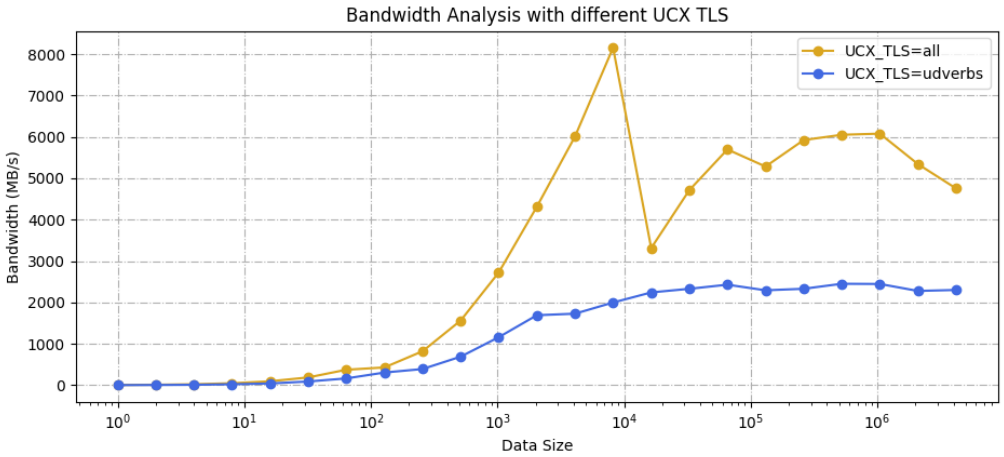
> Please use the following commands to test different data sizes for latency and bandwidth, to verify your ideas:

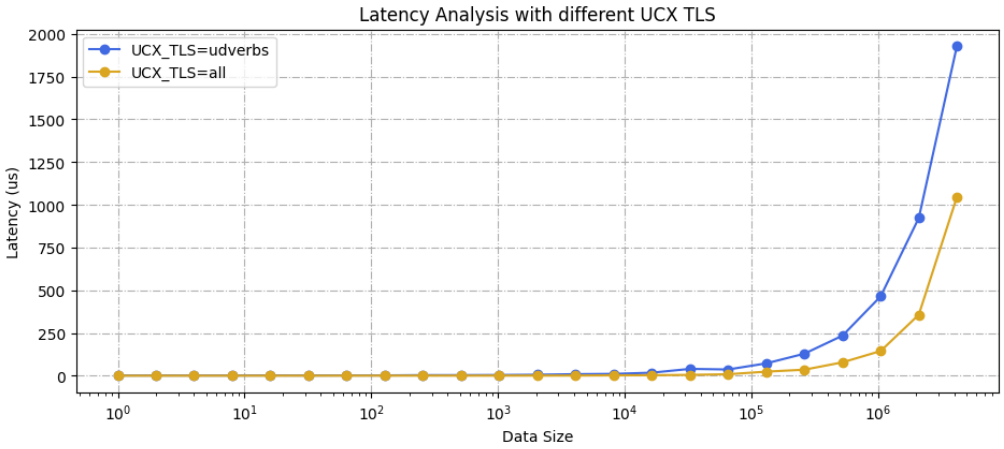
```
module load openmpi/4.1.5
mpiucx -n 2 $HOME/UCX-lsalab/test/mpi/osu/pt2pt/standard/osu_lat
mpiucx -n 2 $HOME/UCX-lsalab/test/mpi/osu/pt2pt/standard/osu_bw
```

- 透過上述指令，UCX會選擇default TLS，在這邊是ud_verbs，透過InfiniBand進行傳輸。若使用以下指令會使用不同的TLS

```
module load openmpi/4.1.5
mpiucx -n 2 -x UCX_TLS=all $HOME/UCX-lsalab/test/mpi/osu/pt2pt/s
mpiucx -n 2 -x UCX_TLS=all $HOME/UCX-lsalab/test/mpi/osu/pt2pt/s
```

- 透過上述指令，UCX會尋找所有TLS中最合適的傳輸方式，最大程度地利用系統及硬體效能，以此case為例UCX會使用shared memory進行傳輸。





- 上圖分別是使用UCX_TLS=udverbs以及all在不同data size下的bandwidth以及Latency表現，可以看到透過設定UCX_TLS=all，其會更妥善地利用硬體資源，單位時間可傳輸的資料量有明顯提升，在小的testcase下可能看不太出來，但在大測資下兩者Latency有明顯差異。

4. Experience & Conclusion

1. What have you learned from this homework?

   這次作業帶我比較深入的去探索我可能不會有機會碰到的程式碼，在只需要call API就能夠方便完成平行化課題的今天，我們不太有機會去接觸如此底層的核心實作，但

透過trace這些code有讓我更了解這些API是如何被完成的，雖然trace code的過程需要很有耐心，但比在課堂上聽的內容又多了幾份記憶點，也十分佩服完成如此龐大framework背後所需付出的努力。

2. Feedback (optional)

   這次作業讓我獲益良多，也辛苦教授和助教辛苦出此次作業，要在如此龐大的架構中尋找合適的作業題目真的很辛苦，最後祝大家新年快樂～