PP HW 5 Report Template

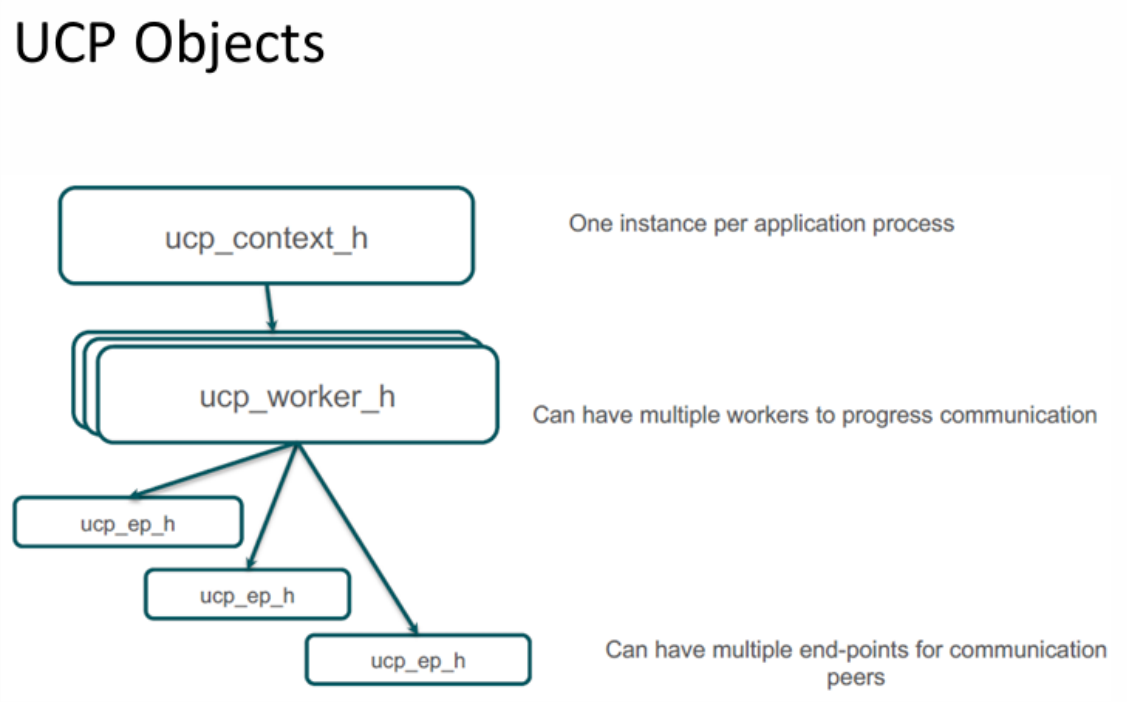
> - Please include both brief and detailed answers.
> - The report should be based on the UCX code.
> - Describe the code using the 'permalink' from GitHub repository.

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`
        - function定義在 `ucx-pp/include/ucp/api/ucp.h` 中會呼叫位在 `UCX-lsalab/src/ucp/core/ucp_context.c` 的ucp_init_version()，該function 建立並初始化 ucp_context、確認API版本、找到可用的network interfaces並初始化。
    - `ucp_worker_create`
        - 呼叫在 `UCX-lsalab/src/ucp/core/ucp_worker.c` 中ucp_worker_create()建立並初始化worker object，一個worker對應一個application context，一個application context可以建立多個worker以便並行存取communication resources
    - `ucp_ep_create`
        - 呼叫在 `UCX-lsalab/src/ucp/core/ucp_ep.c` 中的ucp_ep_create()，creates and connects an endpoint on a local worker for a destination address that identifies the remote worker.This function is non-blocking, and communications may begin immediately after it returns. If the connection process is not completed, communications may be delayed. The created endpoint is associated with one and only one worker.
2. UCX abstracts communication into three layers as below. Please provide a diagram illustrating the architectural design of UCX.
    - `ucp_context`
    - `ucp_worker`

- `ucp_ep`



- 當process需要透過ucp使用ucx時。需要建立context，一個context可以擁有多個worker負責傳輸資料，worker必須透過指定的transport方式去和另一端做溝通，每個worker只能有一種transport方式，不同的worker就能有多種傳輸方式。而一個worker可以有多個endpoint，與多個remote process建立connection做溝通，例如:MPI_Allreduce,MPI_Bcast。

> Please provide detailed example information in the diagram corresponding to the execution of the command `srun -N 2 ./send_recv.out` or `mpiucx --host HostA:1,HostB:1 ./send_recv.out`

3. Based on the description in HW5, where do you think the following information is loaded/created?
    - `UCX_TLS`
    - 該變數是用來指定傳輸層安全性 (TLS) 的參數，會由context create記錄所有可能的TLS，以利後續創建worker。
    - TLS selected by UCX
        - 由ucp_ep負責，當創建一個端點 (ucp_ep_create) 時，UCX 會根據配置和可用的 TLS 資源選擇最合適的傳輸層來建立與遠程端點的連接。這個選擇過程包括決定使用哪一個傳輸層（如 TCP、InfiniBand 等）來進行具體的通信操作。因此，TLS 的選擇和初始化是在 ucp_ep 的創建過程中完成的

2. Implementation

> Please complete the implementation according to the spec
> Describe how you implemented the two special features of HW5.

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

- `UCX-lsalab/src/ucs/config/types.h`
  - 根據提示，增加了一個 UCS_CONFIG_PRINT_TLS 的 flag 因此需要到 types.h 檔案中新增一行

```
/**
 * Configuration printing flags
 */
typedef enum {
    UCS_CONFIG_PRINT_CONFIG          = UCS_BIT(0),
    UCS_CONFIG_PRINT_HEADER          = UCS_BIT(1),
    UCS_CONFIG_PRINT_DOC             = UCS_BIT(2),
    UCS_CONFIG_PRINT_HIDDEN          = UCS_BIT(3),
    UCS_CONFIG_PRINT_COMMENT_DEFAULT = UCS_BIT(4),
    UCS_CONFIG_PRINT_TLS = UCS_BIT(5)
} ucs_config_print_flags_t;
```

- `UCX-lsalab/src/ucp/core/ucp_worker.c`
  - 經由原始版本所印出資訊我們可以很快地找到parser.c `ucs_config_parser_print_env_vars()` 中ucs_string_buffer_cstr(&used_vars_strb)藏有我們需要的UCX_TLS=ud_verbs資訊
  - 經由trace code我們可以發現 `ucp_worker_print_used_tls()`呼叫 `ucp_ep_config_name()` 中strb藏有我們需要的self cfg#資訊
  - 因此在ucp_worker_print_used_tls()中新增以下幾行，將 `UCS_CONFIG_PRINT_TLS` 以及帶有self cfg#資訊的 `ucs_string_buffer_cstr(&strb)` 傳入 `ucp_config_print` ，讓 `ucp_worker.c` 可以成功 call 到 parser.c中 `ucs_config_parser_print_opts` 用於印出答案所新增的條件判斷。
  - `ucp_worker_print_used_tls()`

```
// HW4
    ucp_config_t *config;
    ucs_status_t status;
    status = ucp_config_read(NULL, NULL, &config);
    if (status == UCS_OK) {
        ucp_config_print(config, stdout, ucs_string_buffer_cstr(&strb), UCS_
        ucp_config_release(config);
    }
```

- `UCX-lsalab/src/ucs/config/parser.c`
  - 使用 strtok_r 將環境變數字符串分割為名稱 (var_name) 和值，使用 `strcmp(var_name, "UCX_TLS") == 0` 檢查變數名稱是否與 "UCX_TLS" 完全匹配。如果匹配，則印出整個環境

變數字符串。

- title變數來自於 `void ucp_config_print(const ucp_config_t *config, FILE *stream,const char *title, ucs_config_print_flags_t print_flags)` 中第三個參數 &strb 會記錄 transport protocols selected by UCX
- `ucs_config_parser_print_opts()`

```
if (flags & UCS_CONFIG_PRINT_TLS){
            char **envp;
            char *envstr;
            char *var_name;
            char *saveptr;
            for (envp = environ; *envp != NULL; ++envp) {
                envstr = ucs_strdup(*envp, "env_str");
                if (envstr == NULL) {
                    continue; // 記憶體分配失敗，跳過當前環境變數
                }

                // 分割環境變數為名稱和值
                var_name = strtok_r(envstr, "=", &saveptr);
                if (var_name != NULL && strcmp(var_name, "UCX_TLS") == 0) {
                    printf("%s\n", *envp); // 印出原始環境變數字符串
                }

                ucs_free(envstr);
            }

        // 2. print second line
        printf("%s\n", title);

    }
```

3. How do the functions in these files call each other? Why is it designed this way?
   - `ucp_worker.c` 中的 `ucp_config_print` 會去 call `parser.c` 中的 `ucs_config_parser_print_opts`
   - ucp_worker 主要負責管理和處理通信過程中的各種參數和操作，但不需要直接處理用戶希望打印的具體配置細節。當需要打印配置時，ucp_worker 只需設置適當的標誌（flag），然後將具體的打印任務交由 parser.c 完成就好。
4. Observe when Line 1 and 2 are printed during the call of which UCP API?
   - 根據所印出的資訊可以發現在 `ucp_context` 時就有出現 `"UCX_TLS=ud_verbs"` ，而具體的位址則是在 `ucp_worker.c` 才出現。

```
[pp24s036@apollo31 mpi]$ mpiucx -x UCX_LOG_LEVEL=info -np 2 ./mpi_hello.out
[1734794075.362281] [apollo31:102396:0]     ucp_context.c:2119 UCX  INFO  Ver
[1734794075.370083] [apollo31:102395:0]     ucp_context.c:2119 UCX  INFO  Ver
[1734794075.383359] [apollo31:102396:0]          parser.c:2075 UCX  INFO  UCX
[1734794075.388985] [apollo31:102395:0]          parser.c:2075 UCX  INFO  UCX
[1734794075.389888] [apollo31:102395:0]     ucp_context.c:2119 UCX  INFO  Ver
[1734794075.389901] [apollo31:102396:0]     ucp_context.c:2119 UCX  INFO  Ver
UCX_TLS=ud_verbs
0x555e7bcc51b0 self cfg#0 tag(ud_verbs/ibp3s0:1)
[1734794075.393822] [apollo31:102395:0]      ucp_worker.c:1866 UCX  INFO    (
```

5. Does it match your expectations for questions **1-3**? Why?
   - TLS位址似乎在`ucp_worker_print_used_tls()`就已經決定好，而不是在ucp_ep

6. 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.

   1. lanes（通道）
      定義：表示端點（endpoint）與遠端之間的多個通信通道。
      內容：包括不同的傳輸協議，如 TCP、InfiniBand 等。
      作用：允許在多個傳輸層之間進行並行通信，提升通信效率和可靠性。例如，一個端點可能同時使用 TCP 和 InfiniBand 進行數據傳輸，以實現高性能和冗餘。
   2. tl_rsc（傳輸層資源）
      定義：代表 UCX 中的傳輸層資源。
      內容：包括硬體名稱（如網路介面卡）、協議名稱（如 TCP、InfiniBand）、相關配置等。
      作用：管理和配置底層的傳輸資源，以優化通信性能。這些資源決定了通信的具體方式和效率，確保數據能夠通過最佳的路徑傳輸。
   3. tl_name（傳輸層名稱）
      定義：儲存傳輸層協議的名稱。
      內容：例如 "tcp"、"ib"（InfiniBand）等。
      作用：用於識別和選擇具體的傳輸協議，便於配置和調試。根據名稱，系統可以選擇合適的傳輸層進行通信。
   4. tl_device（傳輸層裝置）
      定義：儲存用於通信的具體裝置信息。
      內容：如網路介面卡的名稱或標識符。
      作用：確定通信所使用的硬體設備，管理設備資源。這有助於系統了解哪些硬體可用，並根據需求分配資源。
   5. bitmap（位圖）
      定義：用來標記傳輸層資源的使用狀態。
      內容：以位元的形式表示不同傳輸層資源是否被選中或正在使用。
      作用：快速檢查和管理可用的傳輸層資源，支持高效的資源選擇和分配。例如，使用位圖可以快速判斷某個傳輸層是否已被使用，避免資源衝突。

6. iface（介面）
   定義：負責處理底層網路或硬體的通信協議。
   內容：包括接口的使用次數、回調 ID、未完成請求數量等。
   作用：管理具體的通信接口，處理傳輸層的細節操作，確保數據能夠正確傳輸和接收。這包括管理回調函數以處理異步事件，以及跟踪未完成的請求以確保通信的完整性。

3. Optimize System

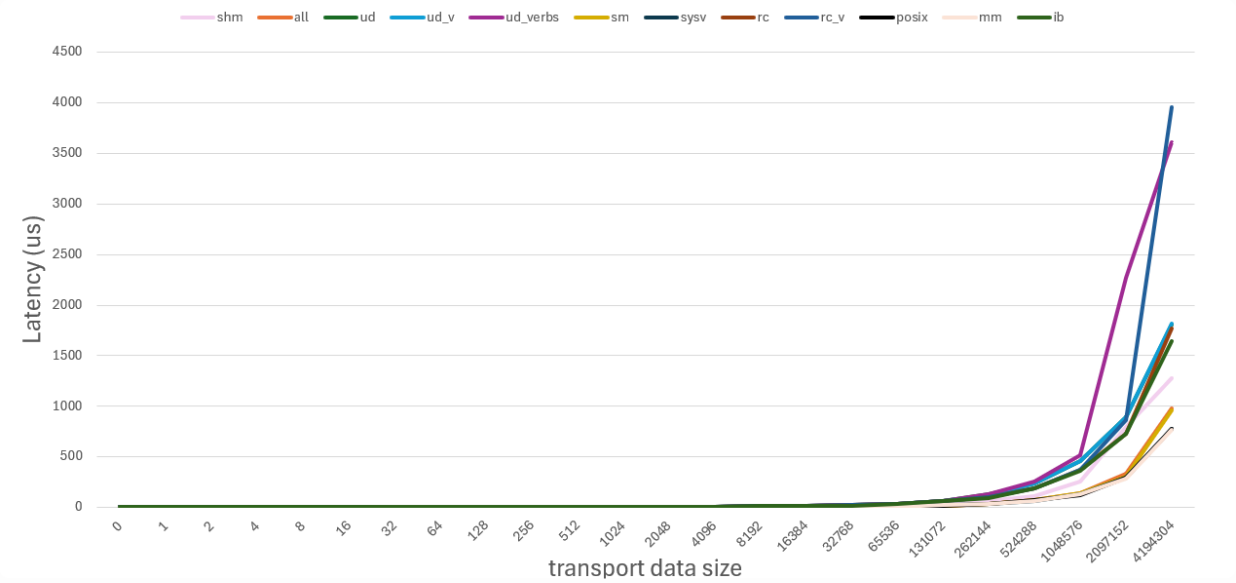
1. Below are the current configurations for OpenMPI and UCX in the system. Based on your learning, what methods can you use to optimize single-node performance by setting UCX environment variables?

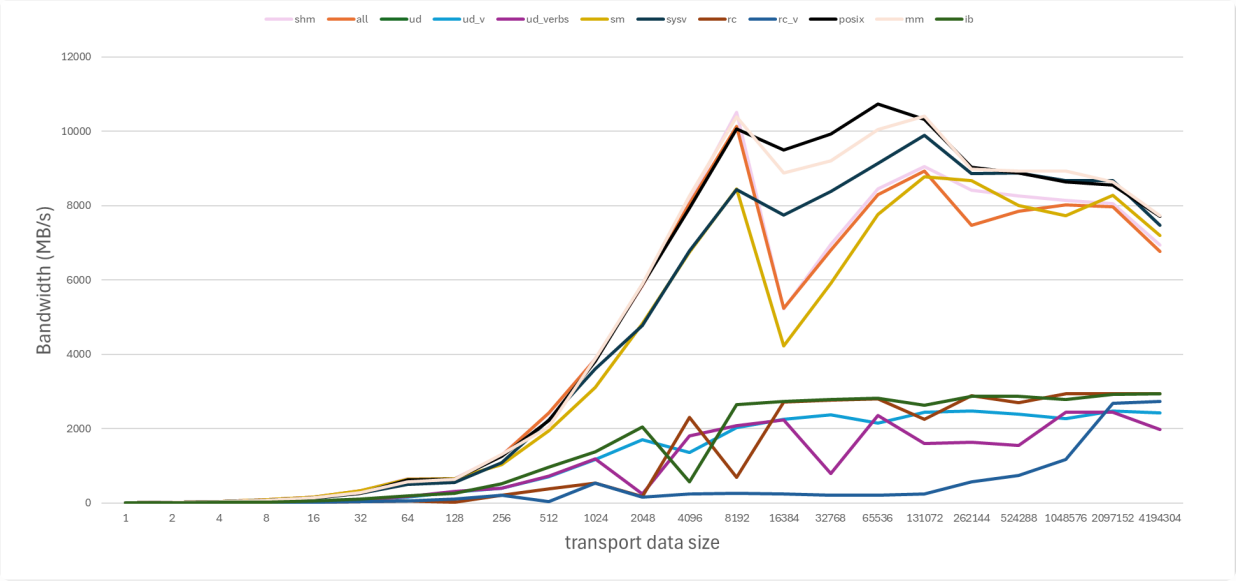
```
-------------------------------------------------------------------
/opt/modulefiles/openmpi/ucx-pp:

module-whatis   {OpenMPI 4.1.6}
conflict        mpi
module          load ucx/1.15.0
prepend-path    PATH /opt/openmpi-4.1.6/bin
prepend-path    LD_LIBRARY_PATH /opt/openmpi-4.1.6/lib
prepend-path    MANPATH /opt/openmpi-4.1.6/share/man
prepend-path    CPATH /opt/openmpi-4.1.6/include
setenv          UCX_TLS ud_verbs
setenv          UCX_NET_DEVICES ibp3s0:1
-------------------------------------------------------------------
```

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

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

2. Please create a chart to illustrate the impact of different parameter options on various data sizes and the effects of different testsuite.





4. Based on the chart, explain the impact of different TLS implementations and hypothesize the possible reasons (references required).

- 我對每項TLS協議測試過他的latency 和 bandwidth，發現是mm和posix協議能有較低的latancy，較高的bandwidth，sm和all則是排名第二。
    - mm（Memory-Mapped）可以直接訪問內存，不需要透過網路傳輸，降低了數據複製和傳輸延遲。
    - posix 協議利用 POSIX 標準的共享內存接口進行數據傳輸，這種方式也能實現高效的內存直接操作。
    - all 代表 UCX 中的所有可用協議，當 UCX 使用 all 配置時，系統需要根據情境選擇合適的協議，但也因為協議選擇和混合資源的開銷導致延遲略高。
    - sm（Shared Memory）共享內存但增加了同步機制，所以增加了延遲

Advanced Challenge: Multi-Node Testing

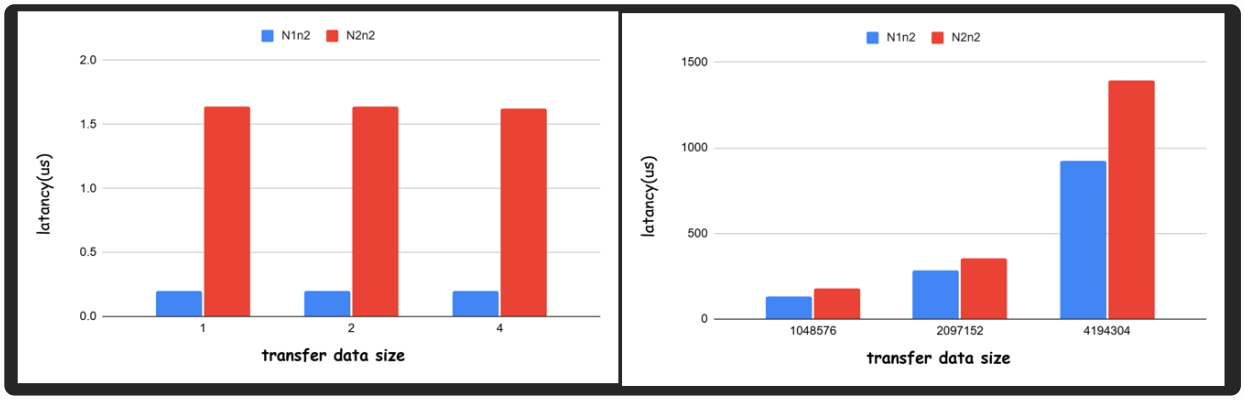
This challenge involves testing the performance across multiple nodes. You can accomplish this by utilizing the sbatch script provided below. The task includes creating tables and providing explanations based on your findings. Notably, Writing a comprehensive report on this exercise can earn you up to 5 additional points.

- For information on sbatch, refer to the documentation at Slurm's sbatch page.
- To conduct multi-node testing, use the following command:

```
cd ~/UCX-lsalab/test/
sbatch run.batch
```

OSU的latency benchmark和HW1寫的Odd-Even Sort來做實驗

測試資料:HW3 30.txt n=64123483



由圖表可以得知當使用越多node、傳輸越大的資料量時，latancy會有顯著的提升

4. Experience & Conclusion

1. What have you learned from this homework?

- 透過本次作業讓我更了解ucx中uxp的運作，也更了解不同的TLP協議的優缺。能夠trace code大型library專案，並完成作業的過程感覺很有成就感。

2. How long did you spend on the assignment?

- 兩天