hw4_109062320

> - 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

1. Identify how UCP Objects (`ucp_context` , `ucp_worker` , `ucp_ep`) interact through the API, including at least the following functions:
    - `ucp_init`： `ucp_context_t` 是負責這個application的 global context的object, 而ucp_init則會基於傳入的ucp_params_t去創建並初始化UCP application context,所以在這裡透過該API來initialize `ucp_context_t` object。
    - `ucp_worker_create`： `ucp_worker` 是負責處理communication的object，而在這裡ucp_worker_create透過傳入在 `ucp_init` 初始化好的 `ucp_context` 還有另外設定好用來指定thread mode的worker_params去初始化 `ucp_worker` object。
    - `ucp_ep_create`： `ucp_ep` 是代表連接到遠端的連結, 而在這裡透過先前設置好的ep_params(裡面存放client的UCX address以及相關資料)，還有先前初始化的 `ucp_worker` 來初始化 `ucp_ep` object。
      而在透過上面三個API去初始化完該application所需要的資訊過後，就可以透過 `ucp_worker` 的objects去基於不同的 `ucp_ep` objects和client去做溝通，其中可以把 `ucp_worker` 的objects設想為每一個要和別人做溝通的process/thread，而 `ucp_ep` objects則代表該process/thread和其他process溝通的每一個connection。
2. What is the specific significance of the division of UCP Objects in the program? What important information do they carry?
    - `ucp_context` ：在這個object中，它存放了該application所需的一些共用資料。其中，`ucp_tl_resource_desc_t` 負責管理communication resource，這對於幫助所有process/thread一致地選擇合適的network device和transport的類型（如 InfiniBand, Ethernet 等）。此外，在記憶體管理方面， `ucp_tl_md_t` 儲存了與記憶體資源使用相關的資訊，以確保application使用的記憶體能夠正確地對應到具體的硬體資源。總而言之，`ucp_context` 物件中儲存的資料是該program下所有process/thread所共同需要的數據，它們為高效的溝通和記憶體操作提供了必要的支持和配置。
    - `ucp_worker` ：在這個物件中，我們知道一個 ucp_worker 可以對應到一個 process 或 thread，這代表著它是一個單一的 communication entity。每個 process 或 thread 可能會有多個與 remote process/thread 的連接，這些連接都是透過 `ucp_ep` （endpoint）物件來管理和記錄的。為了有效管理這些連接，以及確保溝通的準確性， `ucp_worker` 中會包含用於識別自身的 identifier（如 client_id）。這個 identifier對於在建立連接時能夠明確辨識出發起溝通的 process 或 thread 是非常重要的。此外， `ucp_worker` 物件的主要用途是為了記錄和管

理該 process 或 thread 建立的每一個 `ucp_ep` 連接，並提供進行這些連接所需的相關數據。這包括了連接的狀態、資源管理、以及相關communication的參數設定。藉由 `ucp_worker` ，UCX能夠有效地管理每個 process 或 thread 的多個connection，並確保這些連接的正確和有效率的執行。

- `ucp_ep` ：如同先前在 `ucp_worker` 的描述中提到的，ucp_ep 負責記錄和管理每個 process 或 thread 與其他remote entity的連接資訊。它包含了下列幾個element來幫助 UCX framework 高效管理這些connection：
  - 所屬的 ucp_worker： `ucp_ep` 明確指出每個連接所屬的 `ucp_worker` ，因為一個 `ucp_worker` 可能會管理多個 `ucp_ep` ，每一個 `ucp_ep` 代表與一個remote entity的獨立connection。
  - Connection Lifecycle： `ucp_ep` 包含了關於該連接生命週期的資訊，例如連接的建立、維護，以及如何結束該連接。
  - Identifier： `ucp_ep` 中還會包含用於識別該連接的identifier，例如sequence number for remote connection（conn_sn） ，這能夠確保正確的連接至remote entity。
    基於上述所提到的各個 object 的功能，我們可以總結出 `ucp_context` 是用來存放所有 `ucp_worker` 共同需要的資料。這包括了Communication Resources, Memory Domain Information, and Global Configuration等。每個 `ucp_worker` 代表一個獨立的 Communication Entity，它們各自維護與其他 workers 的連結以及所使用的記憶體。因此， `ucp_worker` 存放的是與特定process/thread相關的資料，例如該 worker 管理的 `ucp_ep` （endpoint）列表。
    每個 `ucp_ep` 則代表一個具體的connection，包含了與特定remote communitcation的詳細資料。這包括連接的identifier、狀態等等。
    這樣的分層架構使得 UCX 能夠有效避免混淆不同 workers 或 connections 的資料。每個層級都專注於管理其專屬範疇的資訊，從而提高了整體系統的管理效率和存取便利性。這種結構不僅有助於提高communication的效率，也使得在High-Performance Computing環境中的資源管理和調節變得更加靈活和高效。

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

   - `UCX_TLS` :TLS代表有被啟用的Transport method，而我認為這項資訊是 `ucp_context` 的部分被load進來的，因為對於該context底下的所有worker，被enable的Transport method都應該是一樣的。
   - TLS selected by UCX:因為實際上儲存connection資料的是 `ucp_ep` ，而每一個connection不見得需要使用一樣的transport method，所以我認為在這個部分是由 `ucp_ep` 這裡更新的。

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?
   - `src/ucp/core/ucp_worker.c` :在這裡因為在打開log info之後發現我們需要在Line2印出的資訊基本上就包含在印出來的log裡面，所以就去找了他處理log info資訊的code，然後發現他

是在ucp_worker_print_used_tls這邊把UCX選擇的Transport protocol找出來然後把相對應的資訊丟到string裡面再印出來，所以在這裡可以直接擷取該字串然後依據spec的指示去call `ucp_config_print` 來把東西印出來。而在這邊因為已知在 `ucp_context` 這裡就會把被user enable的transport method load到 `config->tls.array` ，所以只要去check他就可以把line1需要的內容印出來了，因此為了方便性，我就一併在這裡把line1和line2印出來。其中在line1需要額外處理的部分是因為他可能會enable all, 所以要先檢查 `config->tls.mode` 是否為 `UCS_CONFIG_ALLOW_LIST_ALLOW_ALL` 是的話就把 `UCX_TLS=all` 丟到字串，不是的話就traverse `config->tls.array` 然後把裡面的內容丟到字串。在處理好line1, line2的字串之後就分別照順序傳進 `ucp_config_print` 讓他call到 `ucs_config_parser_print_opts` 把東西印出來。在這裡值得一提的部分就是要指定config print flag為 `UCS_CONFIG_PRINT_TLS` 。

- `src/ucs/config/parser.c` :在這裡依據todo所給的提示在 `ucs_config_parser_print_opts` 這邊把傳進來的字串印出來。
- `/src/ucs/config/types.h` :在這裡，因為在 `PP-HW4` 的 todo 提示中有提到 `UCS_CONFIG_PRINT_TLS` 這個enum constant，但它實際上還沒有在 ucs_config_print_flags_t 中被定義，因此需要在該enum中新增他的定義。

2. How do the functions in these files call each other? Why is it designed this way? 正如我在前面所提到的，我在 `ucp_worker_print_used_tls` 這邊去擷取line1, line2要印出來的資訊，然後call `ucp_config_print` , 而在這裡他會call到 `ucs_config_parser_print_opts` 去把東西印出來。而在這裡正如助教所說的，UCS是負責最上層的服務的，而在UCX裡面涵蓋了多個層面的操作，因此需要一個統一的方式來管理和顯示配置。UCS提供的功能能夠統一展示各種配置選項，無論它們來自 UCX 的哪個部分。這有助於用戶和開發者更好地理解和調整communication的行為。
3. Observe when Line 1 and 2 are printed during the call of which UCP API?
   - line1:他在 `[ucp_add_tl_resources](https://github.com/openucx/ucx/blob/f93e4d582add344e4134891a017396c3ff2cee1f/src/ucp/core/ucp_context.c#L1137C2-L1137C2)` 這邊就有traverse `tls.array` 去把被啟用的transport method印出來了。
   - line2:正如我在前面實作的部分所提到的，他在 `ucp_worker_print_used_tls` 會去把line2所印的那些資訊印出來。
4. Does it match your expectations for questions **1-3**? Why? 在這裡和我預想的幾乎是一模一樣的。
   - `UCX_TLS` :TLS是 `ucp_context` 在ucp_add_component_resources的時候去call `ucp_add_tl_resources` 來load進來的，並透過ucp_get_aliases_set去update alias的。
   - TLS selected by UCX:在這個部分我們可以看到在 `[ucp_ep_create_to_worker_addr](https://github.com/openucx/ucx/blob/f93e4d582add344e4134891a017396c3ff2cee1f/src/ucp/core/ucp_ep.c#L815)` 的部分會去create新的endpoint, 然後去initialize lane，並透過 `[ucp_wireup_select_lanes](https://github.com/openucx/ucx/blob/f93e4d582add344e4134891a017396c3ff2cee1f/src/ucp/wireup/select.c#L2399)` 去計算並選出分數最高的transport method。
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:這個variable儲存的是有關於connection的相關訊息，像是resource在context的tl_rsc table中相對應的index
- tl_rsc:他存放UCT那層當中相對應的資源，例如 `Transport name`, `Hardware device name` 等等
- tl_name:這個variable存放transport的名字，而他也是 `tl_rsc` 當中會存放的元素。
- tl_device:這個variable中會存放transport layer所使用的device的名字。
- bitmap:bitmap是用於快速檢索的一個map，比如說在 `alloc_mem_types` 這個bitmap當中，就可以使用相對應的index去bitmap當中尋找被allocate的memory的data type。
- iface:iface為interface的縮寫，他代表的是communication interface的內容。

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?

以最直觀的角度上來看，我們可以藉由更改 `TLS` 來達到加速的作用，這也是為什麼在UCX裡面他會對每一種transport method進行評分並選擇最高分的那一個。

```
-------------------------------------------------------------------
/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
```

在這裡我使用 `UCX_TLS=all` 還有 `UCX_TLS=sm` 來進行比較，因為在很多情況之下，shared memory的溝通速度都能有很好的表現，所以我想要拿它來跟UCX自己選的來做比較。 (因為用對數刻度有些會變成負的所以在這裡直接貼表格的數據)

latency	bandwidth

首先是latency的部分，我們可以看到shared memory的latency一直都比較低，某種程度來說其實很符合預期，但是從他跟UCX自己選的來看就可以發現UCX最後選的其實不是shared memory，而看完bandwidth的部分大概就可以為這個部分解答，因為shared memory為了要避免race condition所以每個人存取同一個部分可能都要排隊，而在這個時候就可能導致bandwidth下降，某種程度來說這就是一個tradeoff，而因為UCX在選transport method的時候是有自己在計算分數的，而他考量到的面相也比較多元，所以即使他的速度在data size很大的時候會變得非常慢，但是他的bandwidth卻是shared memory的兩倍之多。

4. Experience & Conclusion

What have you learned from this homework?

從這次的作業當中我覺得我學到最多的部分就是有關於資料結構還有模組化的重要性，因為在這整個package裡面都可以看到developer為了很有效的模組化，在每一層都定義了很多不同的資料結構來實現這件事情，同時也讓整個code的可讀性變得更高（雖然我在一開始看的時候還是很痛苦就是了哈哈）。除此之外，對於framework底下會自己進行評分並選擇評分最高的狀況我也是第一次見到，感覺真的越底層的東西會越注重這些，同時也真的很佩服那些寫底層的大佬，在trace code的時候差點沒跪著看...