PP HW 4 Report

109062113 葛奕宣

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` Initializes the UCP context. It takes in 3 parameters, a user defined parameter for configuring context, a config for configuring the library, and a pointer that retrieve the created context. Inside the function, it will init everything context need, including allocating memory, filling config, generating uuid, rcache, vfs.....etc.
   - `ucp_worker_create` Create the ucp_worker, and return the created worker in the provided *worker parameter. We provided the context we created and parameters for configuring the worker.
   - `ucp_ep_create` This function create and connect the specific EP we want, EP is the representation of connection. This functnion resides in *run_ucx_client()* and *run_ucx_server()*, this make sense because this two function is responsible for the communication between server and client, thus EP(connection) is created inside them. Because ep resides in worker, so we need to provide a specific worker in the parameter, just like we provide context to worker creation.
2. What is the specific significance of the division of UCP Objects in the program? What important information do they carry? we can see the information they carry in each of there header files.
   - `ucp_context` Context carry imformation about **Communication Components and memory domain(MD)** resource, including many kinds of map like memory registration map, Map of MDs requiring caching. MD index.....etc. There's also *ucp_tl_resource_desc_t*, which is **UCP communication resource**.
   - `ucp_worker` Because it manage list of EPs. There's data storing **information about EPs** including list of internal/all endpoint, EP storage.....etc. **Keep alive information** is another important information that manage the activity of worker
   - `ucp_ep` There's no much information store in ucp_ep itself. It only stores **information about connection** like the **cache value, connection sequence**.....etc. However, there's a **Endpoint extention field** *ext of type ucp_ep_ext_t. It stores additional information about the **connection status** like remote/local EP ID, endpoint match context and remote completion status.
3. Based on the description in HW4, where do you think the following information is loaded/created?
   - `UCX_TLS`

- TLS selected by UCX Based on the discription, TLS is related to transport layer. I think it will be loaded/created during the worker creation function because this worker manage each ep.

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?

- *ucp_worker_print_used_tls(src/ucp/core /ucp_worker.c)*
  - I print the two lines here. The first line I called *ucp_config_print(NULL,stdout,NULL,UCS_CONFIG_PRINT_TLS)*, the second line I use *fprintf* to print the result of *strb* transformed to str.

```
ucp_config_print(NULL,stdout,NULL,UCS_CONFIG_PRINT_TLS);
fprintf(stdout, "%s\n", ucs_string_buffer_cstr(&strb));
```

- *ucs_config_parser_print_opts(src/ucs/config /parser.c)*
  - I add a if for the message flag. Inside the if, I iterate through all the env using environ and find the desired UCX_TLS using strncmp.

```
if (flags & UCS_CONFIG_PRINT_TLS) {
    char **envp;
    for (envp = environ; *envp != NULL; ++envp) {
        if(!strncmp(*envp, "UCX_TLS", 7)) {
            fprintf(stream, "%s\n", *envp);
        }
    }
}
```

- *types.h(src/ucs/config /types.h)*
  - add a type for printing the desired content.

```
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)   //add
} ucs_config_print_flags_t;
```

3. How do the functions in these files call each other? Why is it designed this way? The reason why we need to call ucp_config_print to print the UCX_TLS is that env data resides in parser.c and also it's where most printing function locates.

4. Observe when Line 1 and 2 are printed during the call of which UCP API? trace the function stack, the 2 lines are printed in ***ucp_ep_create()***
5. Does it match your expectations for questions **1-3**? Why? No, the information are loaded/created in ep creation. I think it is becasuse TLS is selected in runtime, so we cannot choose TLS first when creating worker. Instead, when ep is created, it will select its own TLS during the process.
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.

- lanes
  - means the lane that the request is about to go
- tl_rsc
  - it is in type *uct_tl_resource_desc_t*, it stores transport name, device name/type.....etc. In general, it represent virtaul or actual communication resource like network interface, communication port.....etc.
- tl_name
  - inside tl_rsc, means transport name.
- tl_device
  - it is in type *uct_tl_device_resource_t*, it **represent the transport device**, containing type, name, system device.
- bitmap
  - tl_bitmap stores the used tl resource. It can be used for discovering network interface(iface) or other resource.
- iface
  - it is in type *iface_info_t*, it is the communication interface that stores informations like memory domain, allocated worker.....etc.

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/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
```

I **set the UCX_TLS=all**, to enable every transports so that UCX can choose the most suitable one. I show the change of transport method and the improvement in the following result.

- before optimized

```
UCX_TLS=ud_verbs
0x55ae51d6680 self cfg#0 tag(ud_verbs/ibp3s0:1)
```

- after optimized

```
UCX_TLS=all
0x55fbca333670 self cfg#0 tag(self/memory cma/memory)
```

From the result above, we can see the TLS changes to using cma/memory.

Evaluation

We can see from the table below, latency has improved at most 6.68x in 1 bit data and 1.9x in the biggest data.

	Before	After
1 bit	1.67s	0.25s
4194304 bit	1777.11s	933.13s

We can see from the figure below, bandwidth has improved at most 3.54x in 1 bit data and 3.02x in the biggest data.

	Before	After
1 bit	2.64/s	9.36 /s
4194304 bit	2380/s	7201/s

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.

In this experiment, I try to benchmark the latency difference bwtween single node and 2 nodes. I use **osu_latency** as the benchmark. We can see the **Transport Method** and **Latency** difference in the following figures. The left one is multi-node, right one is single-node. For single node, it choose to use **intra-node memory**. For multi-node, it use **inter-node rc_verbs**. In latency, for 1 bit data there's a **9x** speedup in single-node. For the biggest data, there's a **1.55x** speedup. The reason for the speedup lies on the difference of transport method and the cross node hardware bandwidth limitation.

	multi-node	**single-node**
1 bit	2.22s	0.22s
4194304 bit	1390s	895s

4. Experience & Conclusion

1. What have you learned from this homework? I get familiar with the whole UCX system through tracing codes. Learning each variables' meaning is really useful. The whole homework journey is fun.