CS542200 Parallel Programming

Homework 4: Observe the behavior of UCX yourself

Due: Sun. Jan 7 23:59, 2024

1 Goal

Develop observability code within the UCX framework to gain a comprehensive understanding of its architecture and enhance our knowledge in modern parallel computing frameworks, with the following objectives:

1. **Comprehend the UCX Framework**: Delve into the UCX framework to understand the roles and interconnections of its subcomponents, thereby gaining a deeper insight into its overall functionality.
2. **Examine TLS Impact**: Investigate how Transport Layer Security (TLS) is influenced by the interactions and dependencies between UCP, UCS, and UCT within the UCX framework.
3. **Learn Program Structure**: Acquire knowledge about the program structure of modern parallel computing frameworks, focusing on how they are designed, implemented, and optimized for performance.

2 Requirements

- In this assignment, you are required to modify and recompile the UCX system (the underlying communication protocol for MPI) to add the following two features:
  - Display the UCX transport protocols currently configured in the system.
    - You are required to insert code at `// TODO: PP-HW4` (`src/ucs/config/parser.c`) to ensure your program is compatible with the UCX framework.
  - Output the UCX transport protocols actively utilized by the program.

3 Run your Progress

1. Recompile UCX and install it in the `$HOME/hw4` directory, ensuring that a lib folder is present that contains the required dynamic link files. (`libucp.so.0`, `libuct.so.0`, `libucm.so.0`, `libucs.so.0`).

Note that every time you modify the UCX code, you need to re-execute the last line of commands to recompile.

```
git clone -b pp-hw4 https://github.com/NTHU-LSALAB/UCX-lsalab
cd UCX-lsalab
mkdir build && cd build
../configure --prefix=$HOME/hw4/ --with-go=no
srun -n 1 -c 12 make -j12 install
```

2. We provide a custom command `mpiucx`, designed to replace `mpirun`, which will utilize your self-compiled UCX as the underlying communication framework for running MPI parallel programs. You could find the test cases in `UCX-lsalab/test/`

```
module load openmpi/4.1.5
mpiucx -x UCX_LOG_LEVEL=info -np 2 ./mpi_hello.out
```

3. Alter the internal code of UCX to enable printing of:
    - The UCX_TLS information is currently specified by the system.
    - The final TLS transport method was selected by UCX.
4. You may need to trace the files within `src/ucp`, `src/uct`, and `src/ucs` directories and modify at least the following two files to complete the homework.
    - `ucp/core/ucp_worker.c`
        - Invoke `ucp_config_print` to print `UCX_TLS`.
        - Print Line 2 by identifying a key variable.
    - `ucs/config/parser.c`
        - Implement the `ucs_config_parser_print_opts` function to be invoked by `ucp_config_print` here.

4 Result

For each transport, the output comprises 2 lines.

- Line 1: The information from `UCX_TLS` needs to be exactly the same.
- Line 2: The information should include only the key strings of transport protocols selected by UCX. The example must contain strings `cfg#0 tag(sysv/memory cma/memory)`

Sample input & output

- For the Apollo31 configuration, the default transport protocol is set to utilize `ud_verbs`.

```
[willian@apollo31 mpi]$ mpiucx -np 1 ./mpi_hello.out
UCX_TLS=ud_verbs
0x56544467c8f0 self cfg#0 tag(ud_verbs/ibp3s0:1)
Hello world from processor apollo31, rank 0 out of 1 processors
```

- We have enabled UCX to leverage all available transport protocols by setting up `UCX_TLS`.

```
[willian@apollo31 mpi]$ mpiucx -n 2 -x UCX_TLS=all ./send_recv.out
UCX_TLS=all
0x5580604e1a90 self cfg#0 tag(self/memory cma/memory)
UCX_TLS=all
0x559a90bb7a00 self cfg#0 tag(self/memory cma/memory)
UCX_TLS=all
0x5580604e1a90 intra-node cfg#1 tag(sysv/memory cma/memory)
UCX_TLS=all
0x559a90bb7a00 intra-node cfg#1 tag(sysv/memory cma/memory)
Process 0 sent message 'Hello from rank 0' to process 1
Process 1 received message 'Hello from rank 0' from process 0
```

5 Report

Here is the report template https://hackmd.io/lRFyciGGQYS4yxwaInteVw?both

Please copy the markdown into your markdown editor and submit the PDF after converting it at https://md2pdf.netlify.app/.

6 Grading

1. **[30%] Correctness**
   - The pass rate of test data for hw4-judge.
2. **[20%] Demo**
   - A demo session will be held remotely. You'll be asked questions about the homework.
3. **[50%] Report**

- Grading is based on your evaluation, discussion and writing. If you want to get more points, design or conduct more experiments to analyze your implementation.

7 Submission

Upload the files below to eeclass. (**DO NOT COMPRESS THEM**)

- hw4.diff
  - You can obtain this file by
  - `git add -A && git commit -m "Observed the behavior of UCX."`
  - `git diff --color remotes/origin/pp-hw4 > hw4.diff`
- hw4_{student_ID}.pdf
- **KEEP `$HOME/hw4/` CORRECTLY**

8 Hints

- Type `hw4-judge` on **apollo** to run the test cases.
- Scoreboard:
  - https://apollo.cs.nthu.edu.tw/pp23/scoreboard/hw4/
- You can use `-x UCX_LOG_LEVEL=info` in `mpirun` to obtain debug information during UCX runtime, which will help you understand the progress of UCX and gain insights into the transport protocol.
- Refer to slide **p37** of the presentation to determine which component is accountable for the `UCX_TLS` configuration in the system.
- Consult slides **p41** to **p42** of the presentation for insights into UCP's architecture, which will aid in navigating your code to pinpoint the transport protocols actually in use.
- Contact TA via pp@lsalab.cs.nthu.edu.tw or eeclass if you find any problems with the homework specification, judge scripts, example source code or the test cases.
- You are allowed to discuss and exchange ideas with others, but you are required to write the code on your own. You'll get **0 points** if we found you cheating.