Ping Multi Port

Table of contents

List of Figures

In this section, we look at how to create an application with a more complex workflow where operators may have multiple input/output ports that send/receive a user-defined data type.

In this example we will cover:

- how to send/receive messages with a custom data type
- how to add a port that can receive any number of inputs

> **Note**
>
> The example source code and run instructions can be found in the examples directory on GitHub, or under `/opt/nvidia/holoscan/examples` in the NGC container and the debian package, alongside their executables.

Operators and Workflow

Here is the diagram of the operators and workflow used in this example.

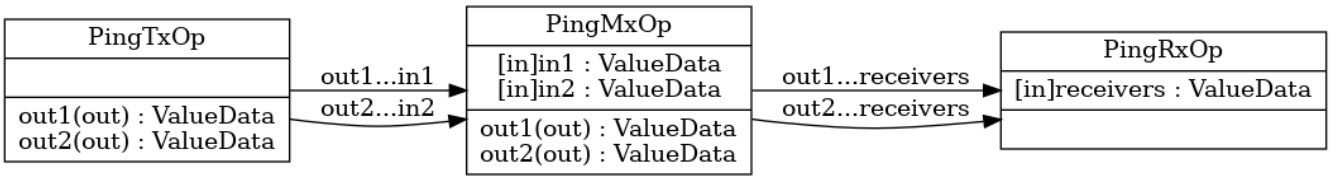

Fig. 7 *A workflow with multiple inputs and outputs*

In this example, `PingTxOp` sends a stream of odd integers to the `out1` port, and even integers to the `out2` port. `PingMxOp` receives these values using `in1` and `in2` ports, multiplies them by a constant factor, then forwards them to a single port - `receivers` - on `PingRxOp`.

User Defined Data Types

In the previous `ping` examples, the port types for our operators were integers, but the Holoscan SDK can send any arbitrary data type. In this example, we'll see how to configure operators for our user-defined `ValueData` class.

Ingested Tab Module

Defining an Explicit Number of Inputs and Outputs

After defining our custom `ValueData` class, we configure our operators' ports to send/receive messages of this type, similarly to the previous example.

This is the first operator - `PingTxOp` - sending `ValueData` objects on two ports, `out1` and `out2`:

Ingested Tab Module

We then configure the middle operator - `PingMxOp` - to receive that data on ports `in1` and `in2`:

Ingested Tab Module

`PingMxOp` processes the data, then sends it out on two ports, similarly to what is done by `PingTxOp` above.

Receiving Any Number of Inputs

In this workflow, `PingRxOp` has a single input port - `receivers` - that is connected to two upstream ports from `PingMxOp`. When an input port needs to connect to multiple upstream ports, we define it with `spec.param()` instead of `spec.input()`. The inputs are then stored in a vector, following the order they were added with `add_flow()`.

Ingested Tab Module

The rest of the code creates the application, operators, and defines the workflow:

Ingested Tab Module

- The operators `tx`, `mx`, and `rx` are created in the application's `compose()` similarly to previous examples.
- Since the operators in this example have multiple input/output ports, we need to specify the third, port name pair argument when calling `add_flow()`:
  - `tx/out1` is connected to `mx/in1`, and `tx/out2` is connected to `mx/in2`.
  - `mx/out1` and `mx/out2` are both connected to `rx/receivers`.

Running the Application

Running the application should give you output similar to the following in your terminal.

```
[info] [gxf_executor.cpp:222] Creating context [info] [gxf_executor.cpp:1531] Loading extensions from configs... [info] [gxf_executor.cpp:1673] Activating Graph... [info] [gxf_executor.cpp:1703] Running Graph... [info] [gxf_executor.cpp:1705] Waiting for completion... [info] [gxf_executor.cpp:1706] Graph execution waiting. Fragment: [info] [greedy_scheduler.cpp:195] Scheduling 3 entities [info] [ping_multi_port.cpp:80] Middle message received (count: 1) [info] [ping_multi_port.cpp:82] Middle message value1: 1 [info] [ping_multi_port.cpp:83] Middle message value2: 2 [info] [ping_multi_port.cpp:112] Rx message received (count: 1, size: 2) [info] [ping_multi_port.cpp:114] Rx message value1: 3 [info] [ping_multi_port.cpp:115] Rx message value2: 6 [info] [ping_multi_port.cpp:80] Middle message received (count: 2) [info] [ping_multi_port.cpp:82] Middle message value1: 3 [info] [ping_multi_port.cpp:83] Middle message value2: 4 [info] [ping_multi_port.cpp:112] Rx message received (count: 2, size: 2) [info] [ping_multi_port.cpp:114] Rx message value1: 9 [info] [ping_multi_port.cpp:115] Rx message value2: 12 ... [info] [ping_multi_port.cpp:114] Rx message value1: 51 [info] [ping_multi_port.cpp:115] Rx message value2: 54 [info] [ping_multi_port.cpp:80] Middle message received (count: 10) [info] [ping_multi_port.cpp:82] Middle message value1: 19 [info] [ping_multi_port.cpp:83] Middle message value2: 20 [info] [ping_multi_port.cpp:112] Rx message received (count: 10, size: 2) [info] [ping_multi_port.cpp:114] Rx message value1: 57 [info] [ping_multi_port.cpp:115] Rx message value2: 60 [info] [greedy_scheduler.cpp:374] Scheduler stopped: Some entities are waiting for execution, but there are no periodic or async entities to get
```

```
out of the deadlock. [info] [greedy_scheduler.cpp:403] Scheduler finished. [info]
[gxf_executor.cpp:1714] Graph execution deactivating. Fragment: [info]
[gxf_executor.cpp:1715] Deactivating Graph... [info] [gxf_executor.cpp:1718] Graph
execution finished. Fragment: [info] [gxf_executor.cpp:241] Destroying context
```

Note

Depending on your log level you may see more or fewer messages. The output above was generated using the default value of `INFO`. Refer to the Logging section for more details on how to set the log level.

© Copyright 2022-2024, NVIDIA.. PDF Generated on 06/06/2024