RIOPA.jl

Release 0.0

Philip Fackler, William Godoy

Dec 13, 2022

INTRODUCTION

Funded by the Exascale Computing Project (ECP), U.S. Department of Energy

CHAPTER

ONE

INTRODUCTION

RIOPA.jl: Reproducible Input Ouput (I/O) Pattern Application

CHAPTER

TWO

GETTING STARTED

It's helpful to set the following environment variables:

- `JULIA_MPI_PATH=<MPI-installation-prefix>`
- `JULIA_MPI_BINARY=system`
- `JULIA_HDF5_PATH=<path-to-HDF5-binaries>` (may be necessary in order to use HDF5 in parallel see [HDF5.jl docs](https://juliaio.github.io/HDF5.jl/stable/#Setting-up-Parallel-HDF5))

In addition, if `mpiexec` is not the proper run command for your system, set the environment variable `JULIA_MPI_EXEC` to the desired run command (such as `srun` or `jsrun`). See the MPI package [configuration](https://juliaparallel.github.io/MPI.jl/stable/configuration/) page for more options if necessary.

From top-level RIOPA directory run

```
julia --project[=.]
```

```
julia> ]
(RIOPA) pkg> instantiate
(RIOPA) pkg> build
(RIOPA) pkg> <Ctrl-D>
```

2.1 Test Suite

```
julia --project ./test/runtests.jl
```

2.2 Minimal Functionality ("hello") Mode

```
julia --project riopa.jl [(-c | --config) <config-file>] hello
```

Using the default configuration:

```
julia --project riopa.jl hello
```

or in parallel:

```
mpirun -n 4 julia --project riopa.jl hello
```

2.3 Generate a Configuration File

```
julia --project riopa.jl [(-c | --config) <config-filename>] generate-config
```

If no filename is given, the generated file will be given a generic name.

CHAPTER

THREE

GETTING STARTED ON SPECIFIC SYSTEMS

3.1 Summit (OLCF)

1. Load necessary modules

```
module load git hdf5 julia
```

2. Set environment variables

```
export JULIA_HDF5_PATH="$OLCF_HDF5_ROOT/bin"
export JULIA_MPI_PATH=$OLCF_SPECTRUM_MPI_ROOT
export JULIA_MPI_BINARY=system
```

3. Build package

```
julia --project -e 'using Pkg; Pkg.build(verbose=true)'
```

3.2 Crusher (OLCF)

1. Load necessary modules

```
module load git hdf5 julia
```

2. Set environment variables

```
export JULIA_HDF5_PATH="$OLCF_HDF5_ROOT/bin"
export JULIA_MPI_PATH=$MPICH_DIR
export JULIA_MPI_BINARY=system
```

3. Build package

```
julia --project -e 'using Pkg; Pkg.build(verbose=true)'
```

CHAPTER

FOUR

CONFIGURATION

A RIOPA simulation is specified by a YAML configuration file. This file is a multi-level hierarchy, since we want to flexibly model I/O patterns in terms of <dataset, step, [level, ...], rank>. Below is listed an example config file that demonstrates most of the features of RIOPA, followed by a reference describing each of the configuration keywords.

example.yaml

```yaml
datasets:
  - type: "output"
    name: "plot"
    basename: "plt"
    io_backend: "HDF5"
    nsteps: 10
    step_conversion_factor: 10
    compute_seconds: 1.0
    data_streams:
      - name: "Level_0"
        initial_size_range:
          - 3000
          - 3600
        evolution:
          function: "GrowthFactor"
          params: [ 1.15 ]
        proc_payload_groups:
          - size_ratio: "1/3"
            proc_ratio: 0.5
          - size_ratio: "2/3"
            proc_ratio: 0.5
      - name: "Level_1"
        initial_size_range:
          - 6000
          - 7200
        evolution:
          function: "Polynomial"
          params: [ 0.0 1.0 ]
        proc_payload_groups:
          - size_ratio: "1/3"
            proc_ratio: "1/4"
          - size_ratio: "2/3"
            proc_ratio: "3/4"
  - type: "output"
    name: "checkpoint restart"
```

(continues on next page)

(continued from previous page)

```
    basename: "chk"
    io_backend: "IOStream"
    nsteps: 3
    step_conversion_factor: 30
    compute_seconds: 3.0
    data_streams:
      - name: "Level_0"
        initial_size_range:
          - 3000
          - 3600
        proc_payload_groups:
          - size_ratio: "1/3"
            proc_ratio: 0.1
          - size_ratio: "2/3"
            proc_ratio: 0.9
      - name: "Level_1"
        initial_size_range:
          - 6000
          - 7300
        proc_payload_groups:
          - size_ratio: "1/3"
            proc_ratio: "1/8"
          - size_ratio: "2/3"
            proc_ratio: "7/8"
```

4.1 Reference

```
datasets:
  - ...
 [- ...]
```

4.1.1 Datasets

The `datasets` keyword introduces a list of dataset configurations (with list elements denoted by the `-`). Each is a separate series of I/O operations that may have completely different rules for when and how data is written (or read).

- **`type:`** Must be either `"input"` or `"output"`
- **`name:`** Full name for what the dataset represents
- **`basename:`** Literal string base for sequences of files or directories
- **`io_backend:`** Choose which file format to use:
  - `"IOStream"`
  - `"HDF5"`
- **`nsteps:`** How many times this I/O operation will take place in sequence
- **`step_conversion_factor:`** For some models if the number of I/O steps is not the same as the number of steps (*e.g.*, time steps) that meaningful to the application, you can use this to convert. This allows for the `evolution`

`function` to be expressed in terms of the application step. So, for example, if the application you are modeling writes output every 10 time steps, you would have `step_conversion_factor: 10` and specify your `evolution function` with respect to time steps.

- **`compute_seconds:`** Time (in seconds) for simulating application compute time between I/O steps for this dataset. RIOPA will use a "sleep" operation to simulate compute time between I/O steps.
- **`data_streams:`** List of data stream configurations.

4.1.2 Data Streams

All the data streams for a given output dataset are written in the same operation, but each stream represents a distinct aspect of the output and may have different rules for how data grows and is distributed among process ranks.

- **`name:`** Literal string used for naming files or directories. You may arbitrarily nest stream directories using a `/`, like `name: "a/b/c"`
- **`initial_size_range:`** List of two (integer) values specifying the total size of the output at the first I/O step. This is specified as a range to allow for variablity in output. Note that the evolution function below is applied to both limits in the range so that for each stream (and then for each rank executing the stream) there is a range of sizes that can be used. If you want to avoid this variability, just use the same number twice.
- **`evolution:`** Group specifying how the data size grows from step to step
  - **`function:`** Name of function type to use. Must be one of the following:
    - `"GrowthFactor"`
    - `"Polynomial"`
  - **`params:`** List of coefficients that fully specify the function
    - For `"GrowthFactor"`, there must only be one parameter, which is the growth factor
    - For `"Polynomial"`, there can be one or more parameters, which act as $[a_1, a_2, ...]$. That is, the constant term a_0 is left out. This term is already provided, derived from the `initial_size_range`

4.1.3 Payload Groups

Process payload groups are specified with size ratio and process ratio. These ratios may be given as fractions (or Julia Rationals) in quotes or as floating point values.

- **`proc_payload_groups:`** List of payload group configurations for distributing the payload across groups of processes (ranks). This allows for more fine-grained control of how I/O operations behave. If this is not needed, simply specify one group with both the `size_ratio` and the `proc_ratio` set to 1.0
  - **`size_ratio:`** Portion of stream output size executed by this group of processes
  - **`proc_ratio:`** Portion of processes belonging to this group

Note that the `size_ratio` entries must sum to 1.0. Likewise for the `proc_ratio` entries.