Student Cluster Competition 2022

Reproducibility Challenge

Introduction

For the SC22 Reproducibility Challenge, your team is tasked with reproducing results from the SC21 paper titled “Productivity, Portability, Performance: Data-Centric Python,” which will hereafter be referred to as the “Data-Centric Python” paper. Your team’s score in the Reproducibility Challenge will be based on the written report and accompanying artifact. You will find below instructions on how to prepare them. Please read these instructions multiple times thoroughly and ask questions (for example, through the Discourse channel) **before** the competition begins.

General Instructions

Main Task

Your main task is to reproduce Figs. 7, 8, and 12, but only for the CPython/NumPy reference version and DaCe:

- **Do not** reproduce anything related to CuPy, Dask, Legate, Numba, and Pythran.
- You still need a valid installation of CuPy to properly execute the GPU tests (NPBench requirement).
- The CPython/NumPy reference execution time **is exactly the same** on CPU and GPU, and CPython/NumPy runs **only** on CPU.

Basic Rules

You must obtain your shared-memory (CPU and GPU) results using NPBench: https://github.com/spcl/npbench. Specifically, you must use branch "scc22", which includes the *secret task* (more details below). You must obtain your distributed-memory results using the following script (you are allowed to edit it, but you may not change the output format):

https://github.com/spcl/dace/blob/310ff52653bea6146fa50222fbb2c0342cb12902/samples/distributed/polybench.py
This script is essentially the same as the one given before:
https://github.com/spcl/dace/blob/master/samples/distributed/polybench.py
The only difference is that the new script records timestamps. It follows that you must use a recent DaCe version from GitHub or a recent release: https://github.com/spcl/dace. However, if you have already prepared using an earlier version that still works properly with NPBench and the distributed-memory script, you may continue with this version. You are also **free to modify DaCe** any way you see fit.

You must follow these rules:

- For NPBench CPU and GPU results, you **must use the "paper" preset** (see https://github.com/spcl/npbench#presets).
- You must obtain the results shown in your **figures during the competition**. You **cannot** use result files (e.g., the "npbench.db" file or the CSV files created by the distributed-memory script) that you obtained **before the competition**.
- You **can** refer to any results you obtained **before the competition** (with any DaCe and NPBench versions, including the original artifact) in your report to answer **qualitative questions**.
- You **cannot** modify NPBench's CPython/NumPy reference code samples.

- You **can** modify NPBench's DaCe reference code samples.
- You **can** modify NPBench's infrastructure, but you **must still provide** the "npbench.db" file produced by the unmodified version with your CPU and GPU results.
- You may (but we do not recommend) alter the distributed-memory codes.
- You can adapt the sizes used in the distributed-memory benchmarks **only if** they consume too much memory in your machine.
- You are free to choose the design of your figures. Produce figures that present the results properly and use sound statistics.

Handling Reproducibility Issues

Before the competition, you were given an early draft of the instructions to prepare and iron out any issues running the software on your machines and the cloud infrastructure. However, if any new issues arise during the competition, just document them in detail (if this cannot fit your report, you can add a file in the artifact and point to it in your report). If some benchmarks that should run and validate failed to do so due to, e.g., some unforeseen bug, your report will be taken into account, and you may still get points.

Some of the points allocated for your report will be assigned based on how many results you were able to reproduce successfully. To get the maximum number of those points, you must succeed in the following (**excluding** the secret task):

- 51 CPU results. Please note that NPBench currently has 52 samples; "arc distance" is the extra one, which was not in the paper but will still count toward the 51 results.
- 30 GPU results. This is how many GPU results were shown in the paper. More benchmarks may run correctly with a new DaCe version, and they will count toward the 30 results.
- 11 distributed-memory results. For each result, you need at least 3 different MPI rank counts: single-node, full-scale, and middle-ground. If you cannot produce 3 different rank counts because of your machine's architecture, e.g., you have too few nodes, then explain this explicitly in your report. In such a case, the scoring formula will be adapted, and you will not be penalized for not having enough nodes. If you can produce results for more rank counts and you have time, you should probably do so because it will assist you with making a better figure and commentary on the results.

Other

For reproducibility reasons, we note that the following parameters were set for the benchmarks shown in the paper:

- OMP_NUM_THREADS was set to the number of physical cores of the machine.

- DaCe can utilize OpenMP sections to run different parts of the program in parallel if they are independent. This functionality was **disabled**. You can do the same by setting the following environment variable to zero: DACE_compiler_openmp_section=0

Secret Task Instructions

In the NPBench branch “scc22” that you are asked to use, we have added 3 new samples: “lstsqr,” “specialconvolve,” and “wdist.” We have already added all the necessary files to run these samples with both CPython/NumPy and DaCe (CPU). However, the DaCe versions, which you can find in the following files, will not run as-is:

- npbench/benchmarks/lstsqr/lstsqr_dace.py
- npbench/benchmarks/specialconvolve/specialconvolve_dace.py
- npbench/benchmarks/wdist/wdist_dace.py

These benchmarks will run automatically when executing the “run_framework.py” script. You can run them individually with the “run_benchmark.py” scripts passing the benchmark name using the “-b” flag.

Your task is to adapt the DaCe versions of the above benchmarks so that at least one of the optimization schemes (fusion, parallel, auto_opt) executes and validates. **You must provide results for the reference CPython/Numpy execution and DaCe on CPU. GPU execution is not asked.** Please note that the above benchmarks **will not count** towards the pure reproducibility task (see Handling Reproducibility Issues). However, **do include them in your reproduction of Fig. 7**. If any of the DaCe codes also run on GPU, it is fine if they also appear on Fig. 8 due to how your plotting script works. You must also describe your changes (see Report Instructions) and include your code in the artifact (see Artifact Description). If you manage to make all three benchmarks work, you will be awarded 5 points. Partial points will be awarded if you manage to make one or two benchmarks work.

Report Instructions

The written report should follow the outline presented in this document. We will grade your submission according to how well and thoroughly you address each section. Once you have finished your report, use the following naming scheme, where “team##” is your team number, e.g., team01:

scc22_team##_p3dcp_reproducibility_report.pdf

For the SCC, the reproducibility challenge application must be run on the cluster your team has deployed at SC22 in person in Dallas and your assigned cloud resources.

Instructions

The report must be at most five (5) pages long with 10 pt font. It must be written in English and submitted in PDF format. A LaTeX template that follows the journal style guide with section headings prepopulated will be provided to you. You may also use a collaborative editing tool such as Overleaf. You must indicate your team name and number on the report.

Report Structure

Abstract

Provide an overview of the structure and findings of your report. Include a high-level description of your hardware and software infrastructure at a high level, the primary goals, and the main results.

Introduction

Describe the Data-Centric Python paper with specific attention to the main goals of your report and the parts of the Data-Centric Python paper that you are attempting to reproduce. Give an outline of your testing methodology and any considerations of your cluster that are important concerning reproducibility. Provide an overview of the sections to follow.

Experimental Setup

Describe your cluster in the state you used it to run the application for the challenge. You should include:

- The hardware configuration, including relevant processors, memory, accelerators, interconnect, etc.
- The operating system and its version.
- The compiler(s) and their version.
- The libraries and other dependencies and their versions (e.g., OpenMPI v4.1, pEVSL SHA 7bbf93b1cf).
- Any changes made for the reproducibility challenge, such as changes to your baseline experimental setup, the application codebase, libraries, etc.

Description of Experimental Run

Your goal for designing experiments should be to successfully run as many benchmarks as possible in the time allotted and be able to identify differences compared with the paper's results and explain the underlying cause of any discrepancies. Describe the design of your experiments, how many benchmarks you ran, and how DaCe (or you) optimized the codes. Which components were essential to reproducing the results of the Data-Centric Python paper?

CPU runtime and speedup over NumPy

Describe your study's design on CPU runtime and speedup over Python/NumPy. Teams should reproduce a similar figure (in content) to Fig. 7 in the Data-Centric Python paper, but only with the DaCe and NumPy columns. The figure's design may be adapted. The NPBench benchmark source code for the NumPy tests should remain unchanged. If needed, you may modify the DaCe versions by altering the source code and the resulting data-centric intermediate representation of the code as long as the results remain numerically correct. **Ensure that the figure includes the results for the secret task** (if you managed to get at least one of the secret benchmarks to work).

Compare, comment on, and explain any differences in runtimes, especially speed-ups for the benchmarks you ran. Describe the workflow you used to optimize the benchmarks, detailing any manual code alterations and transformations of the data-centric intermediate representation.

GPU runtime and speedup over NumPy

Describe your study's design on GPU runtime and speedup over Python/NumPy. Teams should reproduce a similar figure (in content) to Fig. 8 in the Data-Centric Python paper but without CuPy. The figure's design may be adapted. Please note that the source codes for the NumPy tests are the same as for the CPU, and you can reuse your results from the previous task. If needed, you may modify the DaCe versions by altering the source code and the resulting data-centric intermediate representation of the code as long as the results remain numerically correct.

Compare, comment on, and explain any differences in runtimes, especially speed-ups for the benchmarks you ran. Describe the workflow you used to optimize the benchmarks, detailing any manual code alterations and transformations of the data-centric intermediate representation.

Distributed results

Describe the design of your distributed runtime study. Teams should reproduce a similar figure to Fig. 12 in the Data-Centric Python paper, only for DaCe, using as many distributed MPI processes as available. The figure's design may be adapted. Teams must use the scaling factors described in the paper but can adjust initial problem sizes to fit their hardware resources if necessary. However, such cases must be documented and explained thoroughly. If needed, you may modify the DaCe versions by altering the source code and the resulting data-centric intermediate representation of the code as long as the results remain numerically correct.

Compare, comment on, and explain any differences in runtimes, especially speed-ups for the benchmarks you ran. Describe the workflow you used to optimize the benchmarks, detailing any manual code alterations and transformations of the data-centric intermediate representation.

Secret task

Describe the changes you made to the secret benchmark codes so that they work with DaCe.
Ensure that Fig. 8 (CPU) includes the results for the secret task.

Conclusion

Summarize your experiments, the CPU, GPU, and distributed runtime studies results, and how they compare to those in the Data-Centric Python paper. Teams can comment on issues faced, lessons learned, and possible improvements. Furthermore, you should also frame your observations and results from the perspective of the reproducibility initiative. How does this challenge contribute to the reproducibility field?

Artifact Description

Your team must submit a digital artifact with the scripts (shell, bash, Python, Julia, gnuplot, etc.) used for post-processing and plotting the output data to produce the figures for your report and a short readme describing these scripts and their usage. If you used a different plotter, such as Microsoft Excel, please include the input files (in this case, the .xls file). It is not required to use the same styling and color schemes that are used in the Data-Centric Python paper. While it will not be factored during grading, you are encouraged to use a similar style (type of chart, coloring, etc.) for the figures you submit. Similar stylings will aid the reviewers in directly comparing your results with the ones presented in the Data-Centric Python paper. This document will guide how to package the artifact your team will submit.

Outline

Your digital artifact should be packaged as a tarball or zip file that will have a single directory and several subdirectories (detailed below). Populate the subdirectories with the necessary scripts, data, figures, and tables created during the competition, along with a description of what is contained in the artifact and instructions on how to run the code provided. The artifact will contain your reproducibility report and must be turned in before the end of the competition.
The digital artifact should contain the folder structure and files shown below. Please, use the exact same names. The asterisk "*" is a placeholder, you may substitute it with whatever you find appropriate. The double hash "##" is a placeholder for your team's ID, e.g., 07:

- Top-level directory
    - scc22_team##_p3dcp_reproducibility_report.pdf
    - scc22_team##_p3dcp_artifact (directory)
        - codes (directory)
            - lstsqr_dace.py (file)
            - specialconvolve_dace.py (file)
            - wdist_dace.py (file)
        - compile (directory)
            - compile* (one or more files)
            - README (file)
            - system.txt (file)
        - doc (directory)
            - README (file)
        - figures (directory)
            - output (directory)
                - figure_cpu* (file)
                - figure_gpu* (file)

        - figure_distr* (file)
    - scripts (directory)
        - README (file)
        - plot* (one or more files)
- run (directory)
    - output (directory)
        - npbench.db (file)
        - dace_cpu_* (one file per MPI rank count)
    - scripts (directory)
        - README (file)
        - run* (one or more files)

The codes directory

The codes directory should contain the files with your DaCe implementations of the three benchmarks from the secret task.

The doc directory

The doc directory should contain a README in Markdown or plain text. This document will serve as a guide to understanding the artifact's contents and how to use them. It should include the following:

- A description of the artifact's contents.
- Instructions on how to run/view/use each part of the artifact.
- Instructions on how to compare the artifact's output with the output presented in the Data-Centric Python paper. The doc directory should also contain your final reproducibility report in .pdf format.

The compile directory

The compile directory should contain a “compilation” script or set of scripts. These scripts should produce the binaries and any other libraries used by the following “run” script(s). The scripts should build all major dependencies that are required by your build of NPBench. The output of this script should be the binaries used in the Data-Centric Python paper.

- Add a README in the compile directory that gives a brief explanation of how the script(s) work, including the following:
    - The compiler information.
    - Which software dependencies are being built by the script vs. supplied by the OS or .rpms?
    - Any comments on what changes or modifications were done to compile on your cluster?

- Include in the main README in the doc directory how to compile the application (i.e., invoke the script(s)).
- In a separate file, include the output from a script to gather system information. Use https://github.com/SC-Tech-Program/Author-Kit (or a similar script) with appropriate modifications.

The run directory

The run directory should contain two subdirectories: a “scripts” subdirectory and an output subdirectory.

Scripts subdirectory

The scripts subdirectory should contain a “run” script. This script runs NPBench in a given setup and creates the logs/output and results used in the “figures” script. The README should describe how the “run” script is used, including the following:

- Options and environment variables, etc.
- The environment at run time (e.g., env output).
- Differences in the experiment design compared to the Data-Centric Python paper.
- Indicate how you measured timings. For example, indicate if you used the NPBench integrated method, modified it, or used a custom script made by your team.

Output subdirectory

The output subdirectory should contain the logs/output resulting from the “run” script. These files are:

- The “npbench.db” file automatically generated by NPBench. It should include your CPU and GPU results.
- The “dace_cpu_*” CSV files generated by the distributed-memory script. These files should include your distributed-memory results.

These result files are those used by the “figures” script. You could, for example, use the output subdirectory as the target for the output of your run scripts in the run/scripts subdirectory.

The figures directory

The figures directory should contain two subdirectories: a “scripts” subdirectory and an output subdirectory.

Scripts subdirectory

The script subdirectory should contain a “figures” script. This script takes the logs and output from running the “run” script to create your cluster’s version of Fig. 7 (DaCe and NumPy entries only), Fig. 8, and Fig. 12 (DaCe entries only) in the Data-Centric Python paper. The README should detail how the “figures” script is run and what tools and software were utilized.

Output subdirectory

The figures/output directory should contain tables, charts, and figures from running the “figures” script. Please, name the figures as follows:

- “figure_cpu*” for Fig. 7
- “figure_gpu*” for Fig. 8
- “figure_distr*” for Fig. 12

The README should include the following:

- A brief description of the results in the output plots (this commentary can be taken mainly from the report text).
- Comment on the speedups/performance on the team’s machine and compare them to the results in the Data-Centric Python paper (this can also largely be taken from the report text).

Tools to prepare your submission

Your team may write your scripts in a shell scripting language or Python. Please check with the competition organizers if you need to use another scripting language. You can also optionally use automation tools to help you prepare your submission. If you use one of these, document your settings and turn them in with your report and artifact. Feel free to try them and give us your feedback:

- Spack: A flexible package manager that supports multiple versions, configurations, platforms, and compilers.
- EasyBuild: Software build and installation framework to manage scientific software on HPC systems in an efficient way.
- CK: Guide to help you create, run and pack your SCC workflow.
- Popper: Guide on creating a workflow from an existing set of scripts.