MAX PLANCK
COMPUTING & DATA FACILITY

Bits & Bytes

No.210, August 2022

Computer Bulletin of the Max Planck Computing and Data Facility (MPCDF)*
https://docs.mpcdf.mpg.de/bnb

High-performance Computing

Cobra successor procurement

Earlier this year, MPCDF in collaboration with administrative headquarters of the Max-Planck Society (MPG) launched a Europe-wide call for tenders for procuring a successor for the HPC system *Cobra* of the MPG. The corresponding proposal to the president of the MPG is supported by 37 Max-Planck Institutes of all three sections and expresses the need for significant CPU-only and GPU-accelerated computing capacity and storage in the time frame of the next five years. Bids from different vendors and with different technical characteristics will be gauged primarily by the compute performance and energy efficiency the offered system delivers for a representative set of HPC applications of the MPG. This benchmark suite comprises a mix of major simulation codes and machine-learning applications developed or extensively used in the MPG on both, CPU-only and GPU-accelerated HPC platforms. Delivery of the new system is expected by the end of next year, or early 2024.

Erwin Laure, Hermann Lederer, Markus Rampp

CO_2 footprint of MPCDF

With more and more Max-Planck Institutes requesting the CO_2 "footprint" associated with their use of MPCDF computing resources, we now provide a web page[1] which documents the total annual power consumption of MPCDF, together with the fraction used by the central HPC systems (currently, *Raven* and *Cobra*) and the corresponding CO_2 generation, as communicated by the electricity provider. In addition, the web page shows the electricity mix of nuclear, fossile, and "green" sources.

Andreas Schott

Software news

MPI compiler wrappers consolidation

Starting with the Intel MPI module version `impi/2021.6`, only MPI compiler wrappers named according to a scheme with the prefix `mpi` followed by the name of the underlying (non-MPI) compiler executable are available on MPCDF systems (e.g., `mpiicc`, `mpiicpc`, `mpiifort` for Intel compilers and `mpigcc`, `mpig++`, `mpigfortran` for GNU compilers). Compiler wrappers named differently, e.g. `mpicc`, `mpicxx`, `mpif77`, ..., which are unsupported and have been deprecated by MPCDF before will no more be available under Intel MPI. The command-line option `-show` informs about the underlying compiler and the options actually used by the MPI wrapper.

COMSOL multiphysics available

Upon request by a few Max-Planck Institutes, MPCDF now provides the commercial software package COMSOL multiphysics[2] on the HPC system *Raven*. The licensing is based on the software licensing service[3] of the Max-Planck Digital Library, and currently includes a few of the commonly used "add-ons" with specialized COMSOL functionality. Additional functionality or an extension of the pool of concurrent licenses can be accommodated on request. A general overview of free and commercial scientific-software packages available on MPCDF systems is provided in our documentation[4]. Specific software packages can be located on MPCDF machines with the help of the command `find-module <package name or simple search string>`

Tobias Melson, Sebastian Ohlmann, Markus Rampp

*Editors: Dr. Renate Dohmen & Dr. Markus Rampp, MPCDF
[1] https://www.mpcdf.mpg.de/about/co2-footprint
[2] https://www.comsol.com/
[3] https://www.soli.mpdl.mpg.de/en/
[4] https://docs.mpcdf.mpg.de/doc/computing/software/hpc-application-packages.html

GitLab CI Distributed Cache

Introduction

GitLab's continuous integration (CI) infrastructure supports the caching of files and directories for reuse between different jobs of a pipeline and also between subsequently launched pipelines of the same repository. For certain CI setups that rely on a specific (e.g. static or rarely changing) set of files the GitLab CI cache should be used to speed up the pipeline execution considerably.

The MPCDF shared runners (for both CPU and GPU) are configured to use distributed caching, enabling them to access the same caches from different runners consistently. After a pipeline is done, the created caches will not be deleted but stay available for further runs of the same pipeline.

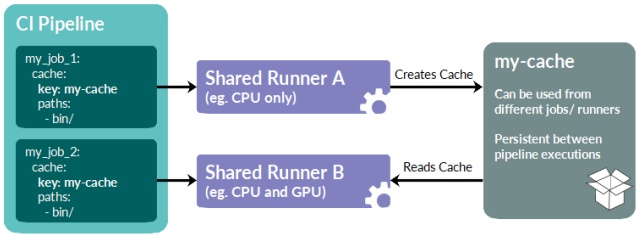

Figure 1: Schema distributed cache

Adding a cache to your CI configuration

For a CI job caching is enabled by adding a `cache` section to its configuration in the `.gitlab-ci.yml` file:

```
my_job:
  cache:
    key: my-cache-key
    paths:
      - bin/
```

The `cache:key` keyword can be used to give each cache a unique identifier. Note that all jobs that use the same cache key access the same cache, which applies to the same pipeline and also to subsequent pipelines of the same branch of a project. The `cache:paths` section lists paths to be included into the cache. In order to enforce a rebuild of the cache, the corresponding key needs to be changed in the pipeline definition.

In the following, two example use cases are discussed. The first use case demonstrates the advantages of creating a cache only once and then reusing it in subsequent jobs and further pipeline runs. The second one demonstrates the use of a distributed cache over different shared runners during a pipeline run. A GitLab project[5] contains the setups for both examples.

CI of software that depends on third-party packages

Consider a Python package that requires a certain third-party package as a dependency. A naive (but frequently adopted) approach would be to `pip install` such a dependency within the CI script right before the actual tests of the primary package are run. Clearly, this creates unnecessary overhead and Internet traffic when done repeatedly. The GitLab CI cache can be used to store the third-party packages between invocations of the pipeline such that `pip` does not repeatedly download them. Once the cache has been created it will stay on the storage back end, being available for further runs of the same pipeline. For compiled codes an analogous use case would be to cache the download and build of a third-party library your primary code relies on.

CI of a complex HPC code that requires CPU and partly GPU resources

Consider the prototypical CI pipeline of an HPC code that first builds the code and then runs tests on it. Often, the build takes a considerable amount of time whereas the tests are comparably quick. On the GPU runners this becomes inefficient because during the build the GPU is almost always idle. Using the GitLab CI cache the build can be performed on the (less expensive and more available) CPU runners, and only the tests can be executed on GPU runners, thereby improving the overall throughput of CI jobs. For some codes it may also be possible to use a single build for both CPU and GPU tests.

Concluding remarks

The `cache:key` keyword allows to cover a variety of use cases, in particular in combination with predefined variables. Several examples[6] are provided in the comprehensive GitLab cache documentation[7].

Unlike GitLab artifacts which are exposed per pipeline run and are downloadable by users via the GitLab web interface for a predefined amount of time, the GitLab cache is considered a mere optimization. A cache is not guaranteed to exist and might need to be regenerated (automatically). In practice, this would be the case e.g. after some time (e.g. weeks) when the cache was cleaned automatically on the server side and the user would launch the pipeline once more.

Thomas Zastrow, Klaus Reuter

[5] https://gitlab.mpcdf.mpg.de/khr/gitlab-ci-cache-usage-examples
[6] https://docs.gitlab.com/ee/ci/caching/#common-use-cases-for-caches
[7] https://docs.gitlab.com/ee/ci/caching

Globus Flows

Introduction

In recent years Globus Online has continued to evolve, adding functionality such as data sharing and publishing as well as automation services. One of the latest additions is Globus Flows which we will introduce in this article.

Globus Flows is a service which allows users to define and execute data workflows using the Globus Subscription attached to servers at MPCDF, such as DataHub. These workflows consist of Actions which can be combined into a single logical operation, be that as simple as staging data across several servers or more complex, for instance processing data on ingest with the need for a human in the loop to review and confirm results before the data is finally published to end users.

Flows in detail

Each Action in a Flow is enabled by an Action Provider which is a REST resource. Globus provides a wide selection of general Action Providers such as transfer, delete and Datacite minting, which can be used by all authorized users to form Flows. In addition, a Python SDK exists to allow projects to create and register their own Action Providers, thus they can wrap their own analysis or infrastructure services to make them available to Flows.

The Flows themselves are written using the Amazon State Language which allow Actions and States to be linked to form complex pipelines which involve both data transfer and processing. Once a Flow has been defined it can be deployed to the Globus Automate platform where it can be made visible to other Globus Online users or kept private so that it is only visible to the author. The Flow can be run from the Globus Online Web app and a form for input values can be generated for users as an interface to the Flow (see example below).

In most cases users will not need to write Flows themselves, but instead they will either use standard Flows defined already by Globus or specific Flows defined by their project or community. A library of Flows[8] already exists and can be found via the Globus Online Web app.

Example

A simple, and yet often useful, example is that of a two-stage data transfer. In this case data is to be transferred between source and destination via an intermediate server. This workflow is often used when users stage data to/from the MPCDF HPC system via DataHub.

The flow encompasses the following steps and can be found online[9]:

1. Copy data from source to intermediate
2. Copy data from intermediate to destination
3. Delete data on intermediate server

Once a user clicks “Start Flow” they are presented with a form where they can fill in the details about the source, intermediate and destination. The screenshot (Figure 2) shows this form where the intermediate has already been selected as the DataHub. Note that the intermediate step is highlighted in green to show it has been defined.

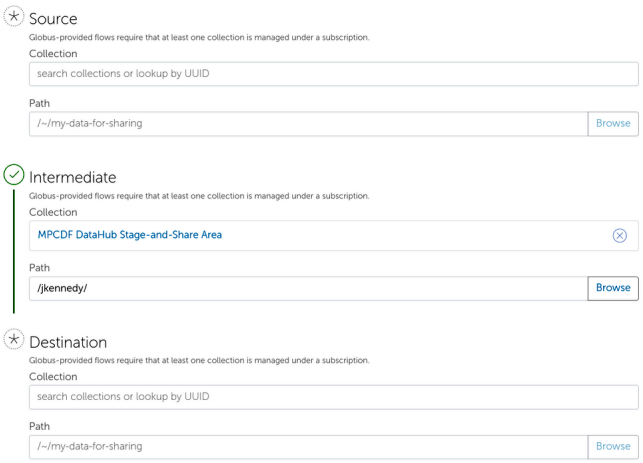

Figure 2: Two-stage Globus Flow

Without Globus Flows these steps need to be taken manually by the user, which in the best case leads to a need to “babysit” transfers and in the worst case can lead to human error and the need to re-transfer data. Using a Flow allows this to be wrapped up into a single logical operation. Moreover, similar to transfers in Globus Online, Flows may be given labels and the results from previous Flows can be viewed in the Web app. This means users can debug Flows and also see their history of previously run Flows.

Summary

Globus Flows provide a means to automate data transfers, simplifying this process for users while also making multi-stage operations more reliable. Many useful Flows, which can be used or taken as templates, already exist in the Flows library. In addition, custom Flows can be defined which can even include project-specific Action Providers.

John Alan Kennedy, Frank Berghaus

[8] https://app.globus.org/flows/library
[9] https://app.globus.org/flows/99791f7d-6c2c-4675-af4b-b927db68bad0

News & Events

For latest training events, course material, as well as for more details and updates on the courses listed below, please visit the MPCDF training website[10].

Introduction to MPCDF services

The next issue of our semi-annual workshop "Introduction to MPCDF services" will be held on October 13th, 14:00-16:30 on zoom. Topics comprise login, file systems, HPC systems, the Slurm batch system, and the MPCDF services remote visualization, Jupyter notebooks and datashare, together with a concluding question & answer session. Basic knowledge of Linux is required. Registration[11] is open.

Meet MPCDF

The next seminar in our monthly online-lecture series "Meet MPCDF" will be held on September 1st, 15:30-16:30 on zoom. During this event our GitLab instance will be introduced, addressing its known and less-well-known features.

Also the subsequent issue of "Meet MPCDF" (on October 6th, 15:30-16:30) is related to GitLab, when we will present a talk about "Going public with your code". The seminar will address various questions and decisions that may become relevant in preparation of sharing your own code with other developers and the public, such as licensing, software management and quality assurance (automatic tests, continuous integration, etc.).

For "Meet MPCDF" events, no registration is necessary, the connection details can be found on the MPCDF training website.

Advanced HPC workshop

MPCDF will again organize an advanced HPC workshop for developers and users of the MPG and of the EU Centre of Excellence NOMAD[12] from Tuesday, November 22nd until Wednesday, November 23rd, 2022, with an additional hands-on day (participation is optional and by application) on Thursday, November 24th. If the pandemic permits, we will have the hands-on day in presence in Garching and the lectures in a hybrid fashion onsite and online. The main topics of the lectures are

- Debugging and profiling of CPU and GPU codes
- Porting codes to GPU-accelerated systems

As a prerequisite, we require participants to have an account for the HPC machines of the MPCDF and to be already familiar with accessing, building and running their codes there.

If you are interested in bringing in your own code to work "hands-on" with the experts and to use the techniques and tools taught in the lectures, please apply by adding a short description of your code and your specific goals in the registration form. The entire Thursday, November 24th is dedicated to working on the selected code projects.

The lectures will be given by members of the application group of the MPCDF, together with experts from Intel and Nvidia. Registration[13] is open until November 12th. Applicants for the hands-on day are kindly asked to register as early as possible and to prepare a representative test case for their code on *Raven* (ideally the test can be run on a single *Raven* node, either GPU-accelerated or CPU-only) already in advance of the workshop. Assistance by MPCDF is provided on request.

Python for HPC

From November 15th to November 17th, 2022, MPCDF offers another iteration of the course "Python for HPC" to participants of the MPG. The course will be given online via zoom with lectures in the morning (9:00-12:00) and exercises in the late afternoon (16:00-17:00).

The course teaches approaches to use Python efficiently and appropriately in an HPC environment, covering performance-related topics such as NumPy, Cython, Numba, compiled C and Fortran extensions, profiling of Python and compiled code, parallelism using multiprocessing and mpi4py, and efficient I/O with HDF5. In addition, topics related to software engineering will be addressed, such as packaging, publishing, testing, and the semi-automated generation of documentation.

The lectures will be given based on Jupyter notebooks which include many reusable code examples. For each topic, hands-on exercises will be provided and discussed in separate sessions. On the last day, there will be time for a general question & answer session. Registration[14] is open.

Tilman Dannert

[10] https://www.mpcdf.mpg.de/services/training
[11] https://events.gwdg.de/e/intro-mpcdf-2022-10-13
[12] https://www.nomad-coe.eu/
[13] https://events.gwdg.de/e/hpc-workshop-2022
[14] https://events.gwdg.de/e/python4hpc-2022