Home

Welcome to the knowledge pool for the danish EuroHPC competence center. Here you will find best practices, material for HPC, HPDA and AI software and hardware, Analysis of the danish national HPCs infrastructure, relevant resources and literature. All the material can be found in pdf form for offline consultation here.

If you desire a specific topic or material to be covered, or need any type of assistance related with the topics of the knowledge pool, contact the HPC facilitator at samuele@chem.au.dk.

Best practices for HPC

This page lists some useful best practices to keep in mind when coding and running applications and pipelines on an HPC.

Code coverage, testing, continuous integration

Every time we code, testing is a concern and is usually performed by the coder(s) regularly during the project. One can identify some basic main types of test:

Regression test

Given an expected output from a specific input, the code is tested to reproduce that same output.

Unit test

Tests the smallest units of the software (e.g. single functions) to identify bugs, especially in extreme cases of inputs and outputs

Continuous integration

A set of tests for the software runs automatically everytime the software code is updated. This is useful to spot bugs before someone even uses the code.

More things one might need to test are the performance/scalability of the code, usability, response to all the intended types of input data.

Unit and regression test can be useful, but at some point not really feasible, since the code can scale to be quite large and complex, with a lot of things to control. It is thus a good practice to use continuous integration, and implement simple but representative tests that cover all the code, so that bugs can be spotted often before the final users do that. Code coverage tools to implement such tests exists for several programming languages, and also for testing code deployed on Github version control.

Links

pyTest, a package to test `python` code

Cmake to test both C, C++ and Fortran code

Travis CI tool for continuous integration in most of the used programming languages. Works on Git version control.

covr, test coverage reports for R

Code styling

Containerized applications

Docker, Singularity

Documentation

When creating a piece of software, it is always a good idea to create a documentation explaining the usage of each element of the code. For packages, there are softwares that create automatically a documentation by using functions´ declarations and eventually some text included into them as a string.

Documents with live code

Programming languages like `python` and `R` allows users to write documents that contain text, images and equations together with executable code and its output. Text is usually written using the very immediate markdown `language`. Markdown files for `R` can be created in the GUI `Rstudio`, while `python` uses `jupyter notebooks`.

Links

Introduction to Markdown for `R` in `Rstudio`.

Jupyter notebooks to create interactive code with `python`. You can write `R` code in a jupyter notebook by using the `python` package rpy2.

Package/Environment management systems

When coding, it is essential that all the projects are developed under specific software conditions, i.e. the packages and libraries used during development (dependencies) should not change along the project's lifetime, so that variations in things such as output formats and new algorithmic implementations will not create conflicts difficult to trace back under development. An environment and package manager makes the user able to create a protected framework where to install specific packages that will not influence other softwares outside the specific environment in use.

Conda is an easy to use and very popular environment manager.

Many short jobs running

Everytime a job is submitted to the job manager (e.g. SLURM) of a computing cluster, there is an overhead time necessary to elaborate resource provision, preparation for output, and queue organization. Therefore it is wise to create, when possible, longer jobs. One needs to find the correct balance for how to organizing jobs: if these are too long and fail because of some issue, than a lot of time and resources have been wasted, but such problem can be overcome by tracking the outputs of each step to avoid rerunning all computations. For example, at each step of a job outputting something relevant, there can be a condition checking if the specific output is already present.

Massive STDOUT outputs

Try to avoid printing many outputs on the standard output STDOUT. This can be problematic when a lot of parallel jobs are running, letting STDOUT filling all the home directory up, and causing errors and eventual data loss. Use instead an output in software-specific data structures (such as `.RData` files for the `R` language) or at least simple text files.

Packaging a coding project

When coding a piece of software in which there are multiple newly implemented function, it can be smart to organize all those functions as a package, that can be reused and eventually shared with ease. Such a practice is especially easy and can be mastered very quickly for coding projects in `python` and `R`.

pyPA `python` packaging user guide

R package development using `Rstudio`

Version control

Version control is the tracking of your development history for a project. This allows multiple people working on the same material to keep changes in sync without stepping over each other's contributions. Version control tools allow to commit changes with a description, set up and assign project objectives, open software issues from users and contributors, test automatically the code to find bugs before users step into them. Version control is useful for both teams and single users, and it is a good practice to have version control as a standard for any project.

GitHub is the most used tool for version control. Here is a quick introduction to get started on Github for new users.

Alternatives to Github are GitLab and BitBucket.