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.

This documentation is part of the EuroHPC Competence Center in Denmark, managed by DeiC.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 and testing

Every time we code, testing is a concern and is usually performed by the coder(s) at some point during the project. However, this is not always the best solution, as a project can scale to be quite large and complex, with a lot of things to test. It is a good practice to implement simple but representative tests that cover all the code, so that bugs can be spotted automatically 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.

Code integration

Code styling

Containerized applications

Docker, Singularity

Documentation

Markdown-style coding

Package/Environment management systems

When coding, it is essential that all the project is 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, new algorithmic implementations will not create conflicts difficult to trace back under development.

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`.

Software design

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.