MDBenchmark Documentation

Release 2.0.0

Written by the MDBenchmark development team

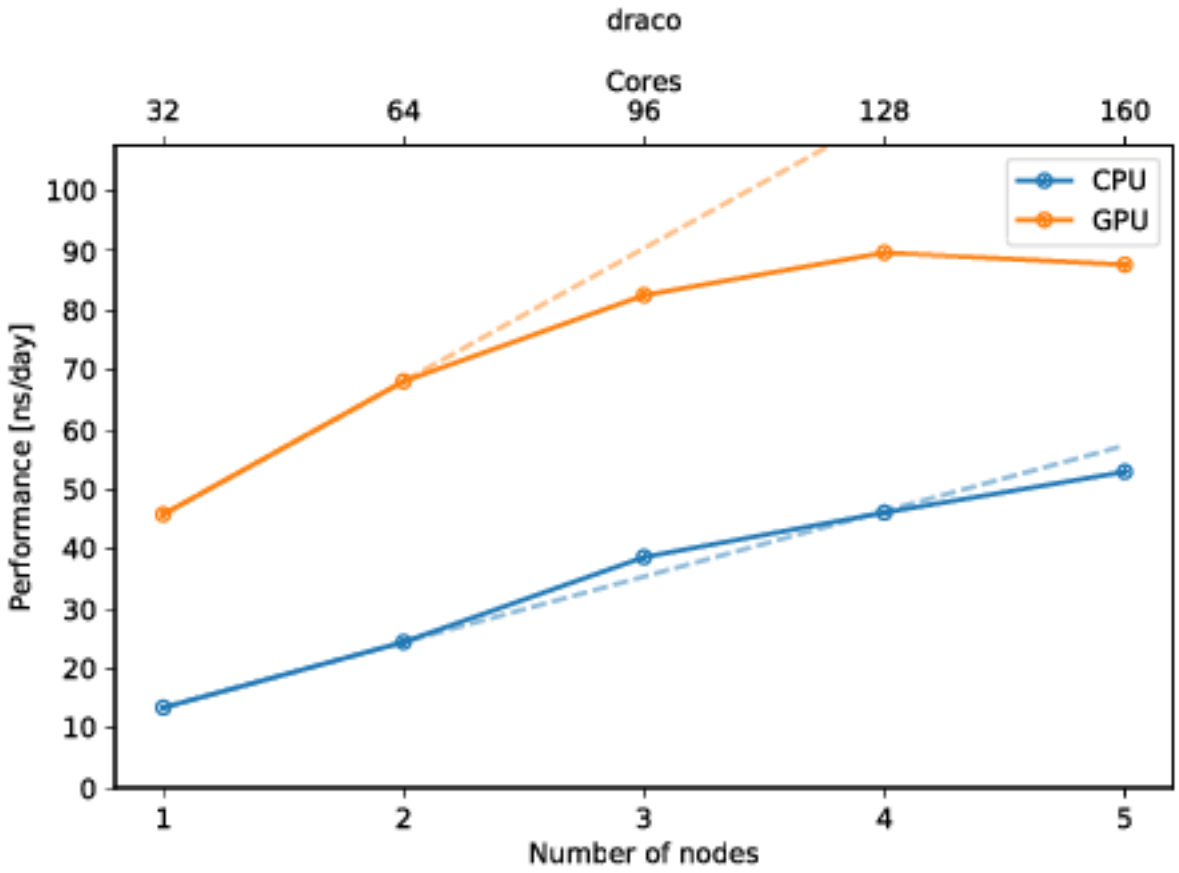
Nov 01, 2018

Contents

MDBenchmark — quickly generate, start and analyze benchmarks for your molecular dynamics simulations.

MDBenchmark is a tool to squeeze the maximum out of your limited computing resources. It tries to make it as easy as possible to set up systems on varying numbers of nodes and compare their performances to each other.

You can also create a plot to get a quick overview of the possible performance (and show of to your friends)! The plot below shows the performance of an molecular dynamics system on up to five nodes with and without GPUs.



CHAPTER 1

Quick start

Follow the next two paragraphs to get a quick start. Extended usage guides can be found below. You can install `mdbenchmark` with your favorite Python package manager. Afterwards you are ready to use `mdbenchmark`.

1.1 Install

If you are familiar with the usual way of installing python packages, just use `pip`:

```
pip install mdbenchmark
```

Anaconda users can install via `conda`:

```
conda install -c conda-forge mdbenchmark
```

Cutting-edge users may prefer `pipenv`:

```
pipenv install mdbenchmark
```

1.2 Usage

Now that the package is installed, you can generate benchmarks for your system. Assuming you want to benchmark a GROMACS 2018.3 simulation on up to 5 nodes, with the TPR file called `md.tpr`, run the following command:

```
mdbenchmark generate -n md --module gromacs/2018.3 --max-nodes 5
```

After generation benchmarks can be submitted:

```
mdbenchmark submit
```

Now, you can also monitor the status of your benchmark with `mdbenchmark`. This will show you the performance of all runs that have finished. If you only wish to view the data you can omit the `--save-csv` flag:

```
mdbenchmark analyze --save-csv data.csv
```

Finally, you can plot the data from your `data.csv` file with:

```
mdbenchmark plot --csv data.csv
```

CHAPTER 2

Content

2.1 Installation

2.1.1 Why isolated environments matter

Installing a new python package into the main python environment of your system can lead to unforeseen consequences. Python packages can have dependencies on different versions of the same package, i.e. `numpy`. If package `packageA` depends on `numpy==1.14.1` and you install `packageB`, which depends on `numpy==1.9.2`, then `packageA` may stop to work. Isolating packages into their own environments makes sure to provide the needed dependencies, while not disrupting the dependencies of other packages (in other environments).

Depending on your setup, there are different ways to create an isolated environment. In the normal Python world, one calls them virtual environment, while users of the Anaconda distribution know them as conda environment.

We recommend to install the package inside a conda environment, while the other ways are also supported.

2.1.2 Install via `conda`

Installation for Anaconda users is handled by `conda`. The following commands create an environment called `benchmark` and install `mdbenchmark` inside.

```
conda create -n benchmark
conda install -n benchmark -c conda-forge mdbenchmark
```

Before every usage of `mdbenchmark`, you need to first activate the conda environment via `source activate benchmark`. After doing this once, you can use `mdbenchmark` for the duration of your shell session.

```
source activate benchmark
```

2.1.3 Install via `pip`

Installation with `pip` should also be done inside a virtual environment.

```
python3 -m venv benchmark-env
```

This created a new directory called `benchmark-env`, if it did not exist before. Now you can activate the environment, as described above.

```
source benchmark-env/bin/activate
```

After activating the environment, you should be able to install the package via `pip`.

Note: The `--user` option leads to the installation of the package in your home directory `$HOME`. If you are not using the option, you may get errors due to missing write permissions.

```
pip install --user mdbenchmark
```

The method requires you to remember where you put the virtual environment and always specify the path when activating. `conda` makes this easier. Several python packages try to make up for this and provide some wrappers, like virtualenvwrapper.

2.1.4 Install via `pipenv`

The easiest way is to install the package is via `pipenv`. First install `pipenv` (refer to its documentation).

```
pip install --user pipenv
```

Now you can let `pipenv` take care of creating the virtual environment. The only downside here is, that you will always need to call `mdbenchmark` from the folder you installed it in.

```
pipenv install mdbenchmark
pipenv run mdbenchmark
```

You can also activate the virtual environment once and then visit different directories afterwards:

```
pipenv shell
cd ..
mdbenchmark
```

2.2 Upgrading

While we try to make upgrades from one version of MDBenchmark to another as easy as possible, we are sometimes forced to break things along the way. This page gives some guidelines on how to migrate to specific versions of MDBenchmark, if you have been using a different version before.

To update from a previous MDBenchmark version to a new release, run the following, if you are using `conda` and your conda environment is called `benchmark`:

```
conda update -n benchmark mdbenchmark
```

and if you use `pip`, run:

```
pip install mdbenchmark --upgrade
```

Refer to our *installation guide*, if you need help with conda/virtual environments.

2.2.1 Migrating to a newer version

Version 2.0.0

Note: **TL;DR:** You need to uninstall the unneeded `datreant.core`, `datreant.data` and `mdsynthesis` Python packages. Or just get rid of the current environment, e.g., `conda env remove -n benchmark` if you are using `conda`, and *create it from scratch*.

Starting from version 2.0.0, MDBenchmark migrated to a different dependency, than it used before. While the specialized `mdsynthesis` package was used in previous version, we migrated to the more general `datreant` package. This leads to two things:

1. Previously, every folder representing a specific number of nodes contained a `Sim.<uuid>.json` file, where `<uuid>` is a UUID. This file contained all information representing a specific benchmark with its parameters. With the newer version of `datreant`, the format was changed and now each of these folders contains a `.datreant` folder instead.
2. `datreant` version 1.0 changed its package layout and led to the necessity of uninstalling previous packages. Because we were using a previous `datreant` version, you as a user will also need to uninstall those packages.

Uninstalling old packages

For proper migration, you need to uninstall the following three packages:

1. `datreant.core`
2. `datreant.data`
3. `mdsynthesis`

If you are using `conda` and your environment is called `benchmark`, run the following command:

```
conda remove -n benchmark datreant.core datreant.data mdsynthesis
```

You can also decide to fully remove the environment (`conda env remove -n benchamrk`) and *create it from scratch*.

If you are using `pip` simply run:

```
pip uninstall datreant.core datreant.data mdsynthesis
```

2.3 Basic usage of MDBenchmark

Usage of MDBenchmark can be broken down into four points:

1. `generate`
2. `submit`
3. `analyze`
4. `plot`

We first generate benchmarks from an input file, e.g., `.tpr` in GROMACS. Afterwards we submit all generated benchmarks to the queuing system of your HPC. Finally, we analyze the performance of each run and generate a plot for easier readability.

MDBenchmark currently supports two MD engines: GROMACS and NAMD. Extensions for AMBER and LAMMPS is planned and help is appreciated. In the following, we will describe the usage of the supported MD engines.

2.3.1 GROMACS

Assuming your TPR file is called `protein.tpr` and you want to run benchmarks with the module `gromacs/2018.3`, run the following command:

```
mdbenchmark generate --name protein --module gromacs/2018.3
```

To run benchmarks on GPUs simply add the `--gpu` flag:

```
mdbenchmark generate --name protein --module gromacs/2018.3 --gpu
```

You can also create benchmarks for different versions of GROMACS:

```
mdbenchmark generate --name protein --module gromacs/2018.3 --module gromacs/2016.4 --
↪gpu
```

2.3.2 NAMD

Note: NAMD support is experimental. If you encounter any problems or bugs, we would appreciate to hear from you.

Generating benchmarks for NAMD follows a similar process to GROMACS. Assuming the NAMD configuration file is called `protein.namd`, you will also need the corresponding `protein.pdb` and `protein.psf` inside the same folder.

Warning: Please be aware that all paths given in the `protein.namd` file must be absolute paths. This ensures that MDBenchmark does not destroy paths when copying files around during benchmark generation.

In analogy to the GROMACS setup, you can execute the following command to generate benchmarks for a module named `namd/2.12`:

```
mdbenchmark generate --name protein --module namd/2.12
```

To run benchmarks on GPUs add the `--gpu` flag:

```
mdbenchmark generate --name protein --module namd/2.12-gpu --gpu
```

Be aware that you will need to use different NAMD modules when generating and running GPU and non-GPU benchmarks! To work with GPUs, NAMD needs to be compiled separately and will be probably named differently on the host of your choice. Using the `--gpu` option on non-GPU builds of NAMD may lead to poorer performance and erroneous results.

2.3.3 Usage with multiple modules

You can use this feature to compare multiple versions of one MD engine or different MD engines with each other. Note that the base name for the GROMACS and NAMD files (see above) must to be the same, e.g., `protein.tpr` and `protein.namd`:

```
mdbenchmark generate --name protein --module namd/2.12 --module gromacs/2018.3
```

2.3.4 Steps after benchmark generation

After you have generated your benchmarks, you can submit them to your queuing system:

```
mdbenchmark submit
```

When benchmarks have finished, you can retrieve the performance results:

```
mdbenchmark analyze
```

Finally, you can plot your benchmarks. For this you need to save your performance results to a CSV file via `mdbenchmark analyze --save-csv results.csv` and invoke the `plot` command on this file:

```
mdbenchmark plot --csv results.csv
```

2.4 Generation of benchmarks

We first need to generate benchmarks with MDBenchmark, before we can run and analyze these. All options for benchmark generation are accessible via `mdbenchmark generate`. The options presented in the following text can be chained together in no particular order in one single call to `mdbenchmark generate`. Before actually writing any files, you will be promted to confirm the action. You can skip this confirmation with the `--yes` option.

2.4.1 Specifying the input file

MDBenchmark requires one file to generate GROMACS benchmarks and three files for NAMD. The base name of the input file is provided via the `-n` or `--name` option to `mdbenchmark generate`. The following table lists all files required by the given MD engine.

MD engine	Required files
GROMACS	`.tpr`
NAMD	`.namd`, `.psf`, `.pdb`

If your input file is called `protein.tpr`, then the base name of the file is `protein` and you need to call:

```
mdbenchmark generate --name protein
```

2.4.2 Choosing a MD engine for the benchmark(s)

MDBenchmark assumes that your HPC uses the modules package to manage loading of MD engines. When given the name of a supported MD engine, it will try to find the specified version:

```
mdbenchmark generate --module gromacs/2018.3
```

It is also possible to specify two or more modules at the same time. MDBenchmark will generate the correct number of benchmark systems for the respective MD engines, sharing all other given options:

```
mdbenchmark generate --module gromacs/2018.3 --module gromacs/2018.2
```

Also it is possible to mix and match MD engines in a single `mdbenchmark generate` call, if the base name of the files is the same (see above):

```
mdbenchmark generate --module gromacs/2018.3 --module namd/2.12
```

2.4.3 Skipping module name validation

If MDBenchmark does not manage to determine the naming of your MD engine modules, it will warn you, but continue generating the benchmarks. Contrary, if it manages to determine the naming, but is unable to find the specified version, benchmark generation fails. If you are sure that the name is correct and MDBenchmark is wrong, you can force the generation of benchmark systems with the `--skip-validation` option:

```
mdbenchmark generate --skip-validation
```

2.4.4 Defining the number of nodes to run on

Benchmarks are especially helpful, if you want to figure out on how many nodes you should run your MD job on. You can provide MDBenchmark with a range of nodes to run benchmarks on. The two options defining the range are `--min-nodes` and `--max-nodes` for the lower and upper limit of the range, respectively. If you do not specify either of these two options, MDBenchmark will use the default values of `--min-nodes=1` and `--max-nodes=5`. This would generate a total of 5 benchmarks, running each benchmark on 1, 2, 3, 4 and 5 nodes.

2.4.5 Listing available hosts

MDBenchmark comes with two pre-defined templates for the MPCDF clusters draco and hydra. You can easily create your own job templates, as described *here*. You can list all available job templates via:

```
mdbenchmark generate --list-hosts
```

2.4.6 Defining the job template to run from

MDBenchmark will try to lookup the hostname of your current machine and search for a job template with the same name. If it cannot find the correct file or you want to use one you have written yourself, e.g., named `my_job_template`, simply use the `--host` option:

```
mdbenchmark generate --host my_job_template
```

2.4.7 Running on CPUs or GPUs

Depending on your setup you might want to run your simulations only on GPUs or CPUs. You can do so with the `--cpu/--no-cpu` and `--gpu/--no-gpu` flags, `-c/-nc` and ``-g/-ng` respectively. If neither of both options is given, benchmarks will be generated for CPUs only. The default template for the MPCDF cluster ``draco` showcases the ability to run benchmarks on GPUs:

```
mdbenchmark generate --gpu
```

This generates benchmarks for both GPU and CPU partitions. If you only want to run on GPUs this is easily achieved with:

```
mdbenchmark generate --gpu --no-cpu
```

2.4.8 Limiting the run time of benchmarks

You want your benchmarks to run long enough for the MD engine to stop optimizing the performance, but short enough not to waste too much computing time. We currently default to 15 minutes per benchmark, but think that common system sizes (less than 1 million atoms) can be benchmarked in 5-10 minutes on modern HPCs. To change the run time per benchmark, simply use the `--time` option:

```
mdbenchmark generate --time 5
```

This would run all benchmarks for a total of five minutes.

2.4.9 Changing the job name

If you want your benchmark jobs to have specific names, use the `--job-name` option:

```
mdbenchmark generate --job-name my_benchmark
```

2.5 Submission of benchmarks

After all benchmark systems are generated, you can also use MDBenchmark to submit these to the queuing system on your HPC. We currently support submission to Slurm, SGE and LoadLeveler.

2.5.1 Submitting all generated benchmarks

To submit all generated benchmarks that are recursively found starting in the current directory, use:

```
mdbenchmark submit
```

Note: `mdbenchmark submit` will ask for your confirmation, before submitting any benchmarks to the queing system. To skip the confirmation use the `--yes` option.

2.5.2 Submitting specific benchmarks separately

If you do not want to submit all benchmark systems at once, you can submit them separately with the `--directory` option. Simply define the relative path to the given directory:

```
mdbenchmark submit --directory draco_gromacs/2018.3
```

2.5.3 Force submitting jobs that were already submitted once

If your jobs were already submitted, but you want to resubmit them once more, you can do so with the `--force` option:

```
mdbenchmark submit --force
```

2.6 Analysis of benchmarks

As soon as the benchmarks have been submitted you can request a summary of the current performance. If a job has not yet finished, not yet started or crashed, MDBenchmark notifies you and marks the affected benchmarks accordingly.

2.6.1 Retrieving the results

The benchmark results can be retrieved immediately after they have been submitted, even if the jobs have not yet started. To do this, simply run:

```
mdbenchmark analyze
```

This will print a summary of your benchmarks. You can do this any time and check the status of your simulations even if they haven't completed yet. The printed results look like this:

```
+--------------------------+---------+----------+------------------+-------+--------+-
↪---------+
| module                   |   nodes |   ns/day |   run time [min] | gpu   | host   |␣
↪  ncores |
|--------------------------+---------+----------+------------------+-------+--------+-
↪---------|
| gromacs/2018.3           |       1 |   99.102 |               15 | True  | draco  |␣
↪       24 |
| gromacs/2018.3           |       2 |  161.454 |               15 | True  | draco  |␣
↪       48 |
| gromacs/2018.3           |       3 |        ? |               15 | True  | draco  |␣
↪       72 |
| gromacs/2018.3           |       4 |  181.614 |               15 | True  | draco  |␣
↪       96 |
+--------------------------+---------+----------+------------------+-------+--------+-
↪---------+
```

The results above showcases that MDBenchmark displays jobs that have not finished, started or crashed with a question mark (`?`).

2.6.2 Saving a CSV file

Once your runs have completed you can write an output CSV file for further processing and plotting. You can define the name of the output CSV file with the `-s` or `--save-csv` option:

```
mdbenchmark analyze --save-csv my_benchmark_results.csv
```

2.6.3 Narrow down results to a specific benchmark

Similar to the submission of benchmarks, you can use the `--directory` option to narrow down the performance analysis to a specific path of benchmarks or a single benchmark:

```
mdbenchmark analyze --directory draco_gromacs/2018.3
```

2.6.4 Plotting of benchmark results

> **Warning:** The function `mdbenchmark analyze --plot` was deprecated with version 2.0. You should migrate to the newer `mdbenchmark plot`, as it provides more functionality and the former version will be removed in the future.

MDBenchmark provides a quick and simple way to plot the results of the benchmarks, giving you a `.pdf` file as output. To generate a plot simply use the `--plot` option:

```
mdbenchmark analyze --plot
```

Plot the number of cores

You can customize the top of your plot with the `--ncores` option. It accepts an integer value, referring to the number of cores per node. If the option is not given, MDBenchmark will try to read this information from the log file.

2.7 Plotting of benchmarks

After generating a CSV file with `mdbenchmark analyze` you can plot the results. In the following we describe how to use `mdbenchmark plot`.

Note: Make sure you wrote a CSV file using `mdbenchmark analyze --save-csv`. Versions before `2.0` generated this file automatically, but this is no longer the default behavior.

2.7.1 Plotting a single CSV

You can plot your benchmarks from a single CSV file, if you saved the data beforehand:

```
mdbenchmark plot --csv data.csv
```

2.7.2 Plotting multiple CSV files

It is also possible to create one plot from multiple CSV files. To do this simply call the `--csv` option multiple times.:

```
mdbenchmark plot --csv data1.csv --csv data2.csv
```

This will plot all data from the benchmark results found in the given files. These can be filtered using the options detailed below. All filters are applied to data in all CSV files collectively.

2.7.3 Output options and file formats

You can set the output name for the generate plot using the `--ouput-name` option. If no name is given, the current date and time will be used as a file name. To generate a plot with the filename `my_benchmark_plot`, simply run:

```
mdbenchmark plot --output-name my_benchmark_plot
```

You can also specify a filetype with the `--output-format` option. If the option is not specified, a PNG file will be generated. Supports file formats are `png`, `pdf`, `svg` and `ps`. To set the file format to PDF, run:

```
mdbenchmark plot --output-format pdf
```

2.7.4 Filter what to plot

To filter your data from the given CSV file(s) you can use the following commands. These can be combined as you like.

Exclude CPU or GPU benchmarks from plots

By default both GPU and CPU data will be plotted. To create a plot without the GPU benchmarks run:

```
mdbenchmark plot --no-gpu
```

To create a plot without CPU benchmarks run:

```
mdbenchmark plot --no-cpu
```

Filter by MD engine

If you have run benchmarks for different MD engines, but want to only plot a subset of these, you can do so with the `--module` option. For example, if you wanted to plot all benchmarks for the two modules `gromacs/2018.3` and `gromacs/2016.4`, run:

```
mdbenchmark plot --module gromacs/2018.3 --module gromacs/2016.4
```

Filter by job template

It is possible to filter your benchmarks by the job template that was used for any given benchmark with the `--template` option. To only plot benchmarks that were run with the `draco` job template, run:

```
mdbenchmark plot --template draco
```

Note: Both `--template` and `--host` may be used interchangeably. While some users might think of job templates as one specific template for their cluster, others might think of them as a bundle of templates with different settings for the same cluster. Either view is correct, and thus we provide both options, but prefer `--template`.

2.7.5 Changing axis labels from nodes to cores

To change the x-axis label from number of nodes to number of cores you can run:

```
mdbenchmark plot --plot-cores
```

2.7.6 Hiding the linear fit

To create a plot without any linear fit, use the `--no-fit` option:

```
mdbenchmark plot --no-fit
```

2.7.7 Changing font size

Font size can be adjusted with the `--font-size` option. The default is `16pt`:

```
mdbenchmark plot --font-size 16
```

2.7.8 Adjusting plot resolution (DPI)

The plot resolution can be changed with the `--dpi` option. The default ist `300`:

```
mdbenchmark plot --dpi 300
```

2.7.9 Customizing ticks on the x-axis

It is possible to change the frequence of ticks on the x-axis. To do this, call the `--xtick-step` option:

```
mdbenchmark plot --xtick-step 1
```

The default value for `--xtick-step` depends on the data you want to plot:

- `--xtick-step=1`, if you plot less than 19 benchmarks
- `--xtick-step=2`, if you plot more than 18 benchmarks
- `--xtick-step=3` if you plot the number of cores and, more than 10 benchmarks or the first number of cores is bigger than 100

2.7.10 Removing the watermark

By default a small watermark will be placed in the top left corner of every plot. To disable this, use the `--no-watermark` option:

```
mdbenchmark plot --no-watermark
```

You are free to use your plots with and without the watermark, whereever you like, but we kindly ask you to cite our latest DOI from Zenodo. In this way, more people will notice MDBenchmark and start optimizing their use of high performance computing resources.

2.8 Defining host templates

You can create your own host templates in addition to the ones shipped with the MDBenchmark. We use the `jinja2` Python package for these host templates. Please refer to the official Jinja2 documentation for further information on formatting and functionality.

To be detected automatically, a template file must have the same filename as returned by the UNIX command `hostname`. If this is not the case, you can point MDBenchmark to a specific template by providing its name via the `--host` option.

Assuming you created a new host template in your home directory `~/.config/MDBenchmark/my_custom_hostfile`:

```
mdbenchmark generate --host my_custom_hostfile
```

2.8.1 Sun Grid Engine (SGE)

This example shows a HPC running SGE with 30 CPUs per node.

```
#!/bin/bash
### join stdout and stderr
#$ -j y
### change to currend work dir
#$ -cwd
#$ -N {{ job_name }}
# Number of nodes and MPI tasks per node:
#$ -pe impi_hydra {{ 30 * n_nodes }}
#$ -l h_rt={{ formatted_time }}

module unload gromacs
module load {{ module }}
module load impi

# Run gromacs/{{ version }} for {{ time - 5 }} minutes
mpiexec -n {{ 30 * n_nodes }} -perhost 30 mdrun_mpi -v -maxh {{ time / 60 }} -deffnm {
↪{ name }}
```

2.8.2 Slurm

The following showcases the job template for the MPCDF cluster `draco` using Slurm.

```
#!/bin/bash -l
# Standard output and error:
#SBATCH -o ./{{ job_name }}.out.%j
#SBATCH -e ./{{ job_name }}.err.%j
# Initial working directory:
#SBATCH -D ./
# Job Name:
#SBATCH -J {{ job_name }}
#
# Queue (Partition):
{%- if gpu %}
#SBATCH --partition=gpu
#SBATCH --constraint='gpu'
```

(continues on next page)

(continued from previous page)

```
{%- else %}
{%- if time is lessthan 30 or time is equalto 30 %}
#SBATCH --partition=express
{%- elif time is greaterthan 30 and time is lessthan 240 or time is equalto 240 %}
#SBATCH --partition=short
{%- else %}
#SBATCH --partition=general
{%- endif %}
{%- endif %}
#
# Number of nodes and MPI tasks per node:
#SBATCH --nodes={{ n_nodes }}
#SBATCH --ntasks-per-node=32
# Wall clock limit:
#SBATCH --time={{ formatted_time }}

module purge
module load impi
module load cuda
module load {{ module }}

# Run {{ module }} for {{ time  }} minutes
srun gmx_mpi mdrun -v -maxh {{ time / 60 }} -deffnm {{ name }}
```

2.8.3 LoadLeveler

Here is an example job template for the MPG cluster `hydra` (LoadLeveler).

```
# @ shell=/bin/bash
#
# @ error = {{ job_name }}.err.$(jobid)
# @ output = {{ job_name }}.out.$(jobid)
# @ job_type = parallel
# @ node_usage = not_shared
# @ node = {{ n_nodes }}
# @ tasks_per_node = 20
{%- if gpu %}
# @ requirements = (Feature=="gpu")
{%- endif %}
# @ resources = ConsumableCpus(1)
# @ network.MPI = sn_all,not_shared,us
# @ wall_clock_limit = {{ formatted_time }}
# @ queue

module purge
module load {{ module }}

# run {{ module }} for {{ time }} minutes
poe gmx_mpi mdrun -deffnm {{ name }} -maxh {{ time / 60 }}
```

2.8.4 Options passed to job templates

MDBenchmark passes the following variables to each template:

Value	Description
name	Name of the TPR file
job_name	Job name as specified by the user, if not specified same as name
gpu	Boolean that is true, if GPUs are requested
module	Name of the module to load
n_nodes	Maximal number of nodes to run on
time	Benchmark run time in minutes
formatted_time	Run time for the queuing system in human readable format (HH:MM:SS)

To ensure correct termination of jobs `formatted_time` is 5 minutes longer than `time`.

MDBenchmark will look for user templates in the xdg config folders defined by the environment variables `XDG_CONFIG_HOME` and `XDG_CONFIG_DIRS` which by default are set to `$HOME/.config/MDBenchmark` and `/etc/xdg/MDBenchmark`, respectively. If the variable `MDBENCHMARK_TEMPLATES` is set, the script will also search in that directory.

MDBenchmark will first search in `XDG_CONFIG_HOME` and `XDG_CONFIG_DIRS` for a suitable template file. This means it is possible to overwrite system-wide installed templates or templates shipped with the package.

CHAPTER 3

Usage reference

3.1 mdbenchmark

Generate, run and analyze benchmarks of molecular dynamics simulations.

```
mdbenchmark [OPTIONS] COMMAND [ARGS]...
```

Options

--version
Show the version and exit.

3.1.1 analyze

Analyze benchmarks and print the performance results.

Benchmarks are searched recursively starting from the directory specified in `--directory`. If the option is not specified, the working directory will be used.

Benchmarks that have not started yet or finished without printing the performance result, will be marked accordingly.

The benchmark performance results can be saved in a CSV file with the `--save-csv` option and a custom filename. To plot the results use `mdbenchmark plot`.

```
mdbenchmark analyze [OPTIONS]
```

Options

-d, --directory <directory>
Path in which to look for benchmarks. [default: .]

-p, --plot
DEPRECATED. Please use 'mdbenchmark plot'. Generate a plot of finished benchmarks.

--ncores, --number-cores <ncores>
DEPRECATED. Please use 'mdbenchmark plot'. Number of cores per node. If not given it will be parsed from the benchmarks' log file.

-s, --save-csv <save_csv>
Filename for the CSV file containing benchmark results.

3.1.2 generate

Generate benchmarks for molecular dynamics simulations.

Requires the `--name` option to be provided an existing file, e.g., `protein.tpr` for GROMACS and `protein.namd`, `protein.pdb` and `protein.psf` for NAMD. The filename `protein` will then be used as the job name, or can be overwritten with the `--job-name` option.

The specified module name will be validated and searched on the current system. To skip this check, use the `--skip-validation` option.

Benchmarks will be generated for CPUs per default (`--cpu`), but can also be generated for GPUs (`--gpu`) at the same time or without CPUs (`--no-cpu`).

The hostname of the current system will be used to look for benchmark templates, but can be overwritten with the `--template` option. Templates for the MPCDF clusters `cobra`, `draco` and `hydra` are provided with the package. All available templates can be listed with the `--list-hosts` option.

```
mdbenchmark generate [OPTIONS]
```

Options

-n, --name <name>
Name of input files. All files must have the same base name.

-c, --cpu, -nc, --no-cpu
Use CPUs for benchmark. [default: True]

-g, --gpu, -ng, --no-gpu
Use GPUs for benchmark. [default: False]

-m, --module <module>
Name of the MD engine module to use.

-t, --template, --host <host>
Name of the host template.

--min-nodes <min_nodes>
Minimal number of nodes to request. [default: 1]

--max-nodes <max_nodes>
Maximal number of nodes to request. [default: 5]

--time <time>
Run time for benchmark in minutes. [default: 15]

--list-hosts
Show available host templates.

--skip-validation
Skip the validation of module names.

--job-name <job_name>
Give an optional to the generated benchmarks.

-y, --yes
Answer all prompts with yes.

3.1.3 plot

Generate plots showing the benchmark performance.

To generate a plot, you must first run `mdbenchmark analyze` and generate a CSV file. Use this CSV file as the value for the `--csv` option in this command.

You can customize the filename and file format of the generated plot with the `--output-name` and `--output-format` option, respectively. Per default, a fit will be plotted through the first data points of each benchmark group. To disable the fit, use the `--no-fit` option.

To only plot specific benchmarks, make use of the `--module`, `--template`, `--cpu/--no-cpu` and `--gpu/--no-gpu` options.

A small watermark will be added to the top left corner of every plot, to spread the usage of MDBenchmark. You can remove the watermark with the `--no-watermark` option.

```
mdbenchmark plot [OPTIONS]
```

Options

--csv <csv>
Name of CSV file to plot.

-o, --output-name <output_name>
Filename for the generated plot.

-f, --output-format <output_format>
File format for the generated plot. [default: png]

-m, --module <module>
Name of the MD engine module(s) to plot.

-t, --template, --host <template>
Name of host templates to plot.

-g, --gpu, -ng, --no-gpu
Plot data of GPU benchmarks. [default: True]

-c, --cpu, -nc, --no-cpu
Plot data of CPU benchmarks. [default: True]

--plot-cores
Plot performance per core instead performance per node. [default: False]

--fit, --no-fit
Fit a line through the first two data points, indicating linear scaling. [default: True]

--font-size <font_size>
Font size for generated plot. [default: 16]

--dpi <dpi>
Dots per inch (DPI) for generated plot. [default: 300]

--xtick-step <xtick_step>
Override the step for xticks in the generated plot.

--watermark, --no-watermark
Puts a watermark in the top left corner of the generated plot. [default: True]

3.1.4 submit

Submit benchmarks to queuing system.

Benchmarks are searched recursively starting from the directory specified in `--directory`. If the option is not specified, the working directory will be used.

Requests a user prompt. Using `--yes` flag skips this step.

Checks whether benchmark folders were already generated, exits otherwise. Only runs benchmarks that were not already started. Can be overwritten with `--force`.

```
mdbenchmark submit [OPTIONS]
```

Options

-d, --directory <directory>
Path in which to look for benchmarks. [default: .]

-f, --force
Resubmit all benchmarks and delete all previous results.

-y, --yes
Answer all prompts with yes.

CHAPTER 4

Indices and tables

- search
- genindex

Index

Symbols

M