Develop MW2

Developer documentation

Reference developer documentation is hosted on the MW2 Gitlab's wiki

https://gitlab.maisondelasimulation.fr/amarinla/mw2/wikis/home

Git and GitLab

Git's basics

- Distributed version control system
- Main repository hosted on Maison de la Simulation's gitlab git clone git@gitlab.maisondelasimulation.fr:amarinla/mw2.git
- Code in your local repository
- Push your changes to distant repository

Git 4 stages

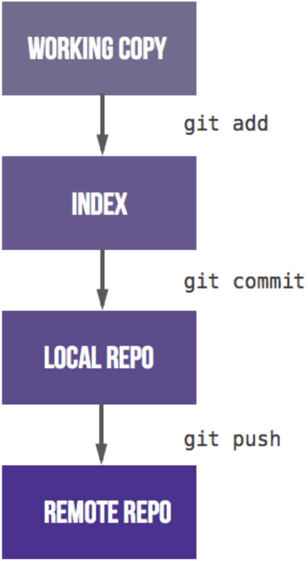

Figure 1: code-add-commit-push

A simple Git Flow to implement a feature

1. Create a branch
   - new branch must be created off of *up-to-date* master
   - branch name should be descriptive
2. Add commits
   - commit often
   - write clear and descriptive commit message
3. Create a merge request
   - push your local branch to the GitLab's repository
4. Discuss and review code
5. Tests
6. Merge into master

GitLab features

- Wiki
- Issue tracker
- Repository Graph
- Continuous Integration
- Merge request discussion board

MW2 Code Structure

MW2 project tree

- *doc/*: documentation
- *make/*: machine specific Makefile configuration and rules
- *scripts/*: user and developer scripts
- *src/*: Fortran source code and unit tests
- *tests/*: Test input and reference data
- *.gitignore*: Set rules for git to ignore some files/directories
- *.gitlab-ci.yml*: CI configuration file
- *Makefile*: Main Makefile
- *README.md*: README file

Coding Style

- The code is written in Fortran 95.
- There should be only *modules* and *program* source files.
- Indent 3 spaces for each block structure.
- Use lowercase for Fortran keywords.
- Use explicit variable names, using underscore ('_') to separate words.
- Use uppercase for variables with the 'parameter' attribute.
- Use save attribute for module variables which are not parameters
- Don't forget to comment what you are coding.
- subroutines defined inside a submodule should have a clear and concise name

Modules

- module names are prepended with MW_
- group related subroutine into the same module
- derived datatype
  - name: MW_module_t
  - define_type : allocate arrays and initialize values in the data type
  - void_type : deallocate arrays and reset values in the data type
  - print_type : print content of the data type to a unit
- importing from other module with `use module , only:`

Modules - Import subroutine from another module

```
module MW_foo
implicit none
contains
   subroutine sub2()
      use MW_bar, only: MW_bar_sub1 => sub1
      implicit none
      ...
      call MW_bar_sub1()
      ...
   end subroutine sub2
end module MW_foo
```

Handling 2DPBC and 3DPBC at the same time

- Difference only in distance computation
- `if` statement prevents vectorization
- Avoid code duplication
- Write algorithm for 2DPBC and let a sed script change 2DPBC to 3DPBC

Makefile

Main Makefile

- Targets
  - *all* [default]: compile the mw binary
  - *check*: compile the mw_tests binary and run the unit tests
  - *clean*: remove object and mod files from build directory
- Includes
  - make/config.mk: environment dependent compilation variable
  - src/module.mk: dependency list
  - make/rules.mk: define generic target rules
- build directory:
  - object files are created in the build directory to keep src clean
  - `$(build_dir)` make variable

Makefile configuration variables

- *F90*: fortran compiler or mpiwrapper
- *F90FLAGS*: fortran compilation flags
- *FPPFLAGS*: fortran preprocessor flags
- *LDFLAGS*: Linker flags
- *J*: command to specify mod file output dir (-J for gfortran)
- *PFUNIT*: Path to pFUnit installation directory

```
# Compilation options
F90 := mpif90
F90FLAGS := -g -O0 -Wall
FPPFLAGS :=
LDFLAGS :=
J := -J

# Path to pFUnit (Unit testing Framework)
PFUNIT := /opt/pfunit/pfunit-parallel
```

Modify the module.mk to add a new file

- File a.f90 defines module mw_a

  ```
  local_srcs_f90 += $(current_dir)/a.f90
  ```

- File b.f90 defines module mw_b and use mw_a

  ```
  local_srcs_f90 += $(current_dir)/b.f90
  $(current_build_dir)/b.o: $(current_build_dir)/a.o
  ```

- File c.f90 defines module mw_c, includes c_inc.inc and use module a

  ```
  local_srcs_f90 += $(current_dir)/c.f90
  $(current_build_dir)/c.o: $(current_dir)/c_inc.inc
  $(current_build_dir)/c.o: $(current_build_dir)/a.o
  ```

MPI parallelisation

MPI parallelisation strategy

- Distribute work only
- Each rank has a copy of all the dataset ($\approx 30\mathrm{MB}$ per rank)
- `MPI_Allreduce` to sum up contribution from all ranks

Short range interactions kernels

- Distribute pairs by block
- Block size: 32×32

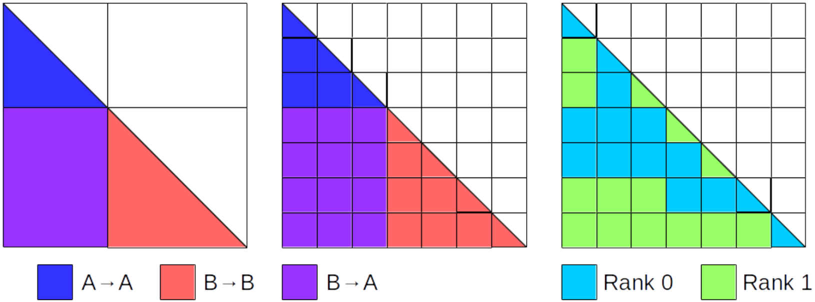

Figure 2: pair block decomposition

Reciprocal space ewald summation kernels

- Distribute k-points

```
l = lstart_local
m = mstart_local
n = nstart_local
do imode = 1, num_modes_local
   compute contribution for each mode
   update l, m, n
end do
call mpi_allreduce
```