Tinker-HP : Readme/Quickstart (v1.2)

Louis Lagardère, Luc-Henri Jolly, Jean-Philip Piquemal
Sorbonne Université, Paris, France.
TinkerHP_Support@ip2ct.upmc.fr

I. Citing Tinker-HP

If you use Tinker-HP please cite the following reference :

> *Tinker-HP: a Massively Parallel Molecular Dynamics Package for Multiscale Simulations of Large Complex Systems with Advanced Polarizable Force Fields.* L. Lagardère, L.-H. Jolly, F. Lipparini, F. Aviat, B. Stamm, Z. F. Jing, M. Harger, H. Torabifard, G. A. Cisneros, M. J. Schnieders, N. Gresh, Y. Maday, P. Ren, J. W. Ponder, J.-P. Piquemal, Chem. Sci., 2018, 9, 956-972 (Open Access) DOI: 10.1039/C7SC04531J

If you use the AVX512 vectorized version of Tinker-HP 1.2, please also cite :

> *Raising the Performance of the Tinker-HP Molecular Modeling Package [Article v1.0].* L.-H. Jolly, A. Duran, L. Lagardère, J. W. Ponder, P. Y. Ren, J.-P. Piquemal, LiveCoMS, 2019, 1 (2), 10409 (Open Access) DOI: 10.33011/livecoms.1.2.10409

II. Prerequisites

A. Calculation Libraries

Tinker-HP requires the MKL library, a FFT library (such as FFTW) and a slightly modified 2DECOMP_FFT library (shipped with Tinker-HP) in order to run. The 2DECOMP_FFT library enables parallel 3D FFT computations based on 2d-pencils data distribution (see 2DECOMP_FFT site) based on a sequential implementation of FFTs such as the one provided by the FFTW library.

B. Parallel library

Tinker-HP also requires a recent enough MPI library supporting MPI 3.x standards such as non blocking collectives. The code has been extensively tested with recent IntelMPI versions (such as intel MPI 5.1) and better performances have been observed with this family of MPI implementation compared to other ones such as OpenMPI .

III. Installation

As Tinker-HP is shipped in source form, you need to compile it. This was not always an easy task in the previous releases. Tinker-HP now uses a `configure` script built with autotools packages from GNU to ease the compilation and installation process. Apart from the usual options available with all `configure` scripts, there are specific options for Tinker-HP.

```
Usage: ./configure [OPTION]... [VAR=VALUE]...

Optional Features:
  --enable-debug                Enable debug mode (check array bounds, implicit
                                none, etc...). Should not be active in normal
                                operations [default is no]
  --enable-skylake              Enable AVX512 Optimization for Skylake Processors
                                [default is no]
  --enable-knl                  Enable AVX512 Optimization for KNL (Xeon Phi)
                                Processors [default is no]
  --enable-fft-generic          Enable generic FFT mode [default is yes]
  --enable-fft-mkl              Enable MKL   FFT mode [default is no]
  --enable-fft-fftw3            Enable fftw3 FFT mode [default is no]
  --enable-fft-fftw3_f03        Enable fftw3_f03 FFT mode [default is no]

Optional Packages:
  --with-blaslib=<BLAS LIB>     Specify BLAS library [mkl, lapack or
                                /absolute/path/to/BLAS_library]
  --with-fftlib=<FFT LIB>       Specify a library for FFT called by 2decomp [mkl or
```

```
                                    fftw3 or /absolute/path/to/FFTW_library]
```

The ultimate goal of this script is to let you type

```
./configure ; make ; make install ; cd example ; ./ubiquitin2.run
```

and have everything compiled, installed and running.

A. List of Options

As for all the `configure` scripts, you can choose the directory in which the binaries will be copied. So, `configure` has `--prefix=<DIR>`.

Tinker-HP has a special interest to know if it will run on AVX-512 capable processors. So, `configure` has options for that:

```
--enable-slylake
--enable-knl
```

Recall that Tinker-HP needs to make a FFT decomposition with a modified version of the 2DECOMP_FFT library, which in turn needs a working FFTW library. This is why you can find `configure` options about FFT interface and library:

```
--enable-fft-generic
--enable-fft-mkl
--enable-fft-fftw3
--enable-fft-fftw3_f03
--with-fftlib=<FFT LIB>
```

Tinker-HP also needs some functions that resides in a working BLAS library. So, there is an option for that:

```
--with-blaslib=<BLAS LIB>
```

Finally, as there might be some execution problems, or compilation problems for the users who develop code, Tinker-HP has an `--enable-debug` option.

`configure` tries to find its path to reach a valid MPI compiler and a valid Fortran compiler by unsetting the environment variables `$FC` and `$F77`, and reading the environment variable `$PATH`. It also tries to find valid FFT and BLAS libraries by reading `$FFTW` and `$MKLROOT` or `$LAPACK` respectively. Most of the time, these environment variables are defined through the `module` framework. As a try, do a

```
module available 2>&1 | less
```

to see if you have the `module` framework installed on your machine, and to know what `module` you can load. `configure` then figures out how to build the correct Makefiles.

By default, `configure` chooses the MKL library from Intel as the BLAS and FFTW3 libraries, sets the `--enable-fft-mkl` option, does not make any processor optimization, and disables debugging. Thus, typing `./configure` give the same result as if you have typed `./configure --enable-fft-mkl --with-blaslib=mkl --with-fftlib=mkl`.

B. Using *configure*

If you want to have different settings than those used by default, you'll have to give `configure` more information. Be aware that `configure` cannot magically guess anything. So, the information you give must be precise and complete.

1) Install Directory: By default, this is where you have unzipped and untarred the distribution. If you want another place, use `--prefix=<DIR>`. You can choose any directory you want, providing that you have permission to create this directory and/or write in it.

2) Processor optimization: The machine on which you compile is not always the one on which Tinker-HP will run. If you know that Tinker-HP is going to run on AVX-512 capable processors, you are strongly encouraged to use one of :

`--enable-knl` for KNL processors (also known as Xeon-Phi)
`--enable-slylake` for Skylake processors.

as this will dramatically improve the execution speed. Otherwise, the optimization will be done using the capabilities of the compilation machine, as determined by the compiler.

3) FFT interface: You can choose the interface of FFTW you want to use. This has an effect on the 2DECOMP_FFT library. So :

`--enable-fft-generic`	gives the generic FFT, with no call to FFTW library
`--enable-fft-mkl`	gives the MKL FFT. It also automatically selects the MKL library as the FFTW library.
`--enable-fft-fftw3`	gives the fftw3 interface, and is designed to work with an external FFTW library.
`--enable-fft-fftw3_f03`	gives the fftw3 Fortran2003 interface, and is designed to work with an external FFTW library

4) FFT Library: You can choose the FFTW library you want to use. It can come from the MKL suite, or some FFTW3 package (either system installed, or compiled by you). So :

`--with-fftlib=mkl`	: selects the MKL library, but needs the variable `$MKLROOT` to be set to the absolute path of the MKL library
`--with-fftlib=/path/to/mkl/library`	: selects the MKL library by giving the absolute path of the MKL library
`--with-fftlib=fftw3`	: selects the FFTW3 library, but needs the variable `$FFTW` to be set to the absolute path of the FFTW3 library
`--with-fftlib=/path/to/fftw3/library`	: selects the FFTW3 library by giving the absolute path of the FFTW3 library

Here are typical commands you can type. If `$MKLROOT` has been correctly set :

```
./configure --enable-fft-mkl --with-fftlib=mkl
```

If you wish to give the absolute path of the library :

```
./configure --enable-fft-mkl   --with-fftlib=/path/to/mkl/library
./configure --enable-fft-fftw3 --with-fftlib=/path/to/fftw3/library
```

These last commands can also be written this way :

```
MKLROOT=/path/to/mkl/library ./configure --enable-fft-mkl   --with-fftlib=mkl
FFTW=/path/to/fftw3/library  ./configure --enable-fft-fftw3 --with-fftlib=fftw3
```

5) BLAS library: You can choose the BLAS library you want to use. It can come from the MKL suite or some LAPACK package (either system installed, or compiled by you). So :

`--with-blaslib=mkl`	: selects the MKL library, but needs the variable `$MKLROOT` to be set to the absolute path of the MKL library
`--with-blaslib=/path/to/mkl/library`	: selects the MKL library, by giving the absolute path of the MKL library
`--with-blaslib=lapack`	: selects the LAPACK library, but needs the variable `$LAPACK`to be set to the absolute path of the LAPACK library
`--with-blaslib=/path/to/lapack/library`	: selects the LAPACK library, by giving the absolute path of the LAPACK library.

Here are typical commands you can type. If `$MKLROOT` has been correctly set :

```
./configure --enable-fft-mkl --with-blas=mkl
```

If you wish to give the absolute path of the library :

```
./configure --enable-fft-mkl --with-blas=/path/to/mkl/library
./configure --enable-fft-mkl --with-blas=/path/to/lapack/library
```

These last commands can also be written this way :

```
MKLROOT=/path/to/mkl/library    ./configure --enable-fft-mkl --with-blas=mkl
LAPACK=/path/to/lapack/library  ./configure --enable-fft-mkl --with-blas=lapack
```

6) DEBUG mode: This mode is primarily intended for developers, but can also be useful if you experience errors while running Tinker-HP. Adding `--enable-debug` to the `configure` command turns on **boundary checking**, forces **implicit none**, sets the optimization level to **0** (the lowest value) and enables **backtracing** and **all warnings**. The compilation produces all the binaries and gives them the `.debug` extension, so that you know that these binaries are not optimized.

C. Output of configure

`configure` produces a final log to resume what will be done. It displays using colors (if available) all the information you gave, and everything it has been able to catch from the environment.

Here is the result of a successful run of the `configure` command :

```
FFTW=/usr/local/fftw-3.3.7/Intel/2018/impi/
./configure --enable-debug --enable-fft-fftw3 --with-fftlib=fftw3 --with-blas=lapack
```

where we give the absolute path of the FFTW3 library in the `$FFTW` variable, ask for the DEBUG mode, enable the fftw3 interface, use the FFTW3 library and want the lapack library for BLAS, assuming that the `$LAPACK` variable is already set.

```
configure:
configure: ************************************************************************
configure: **
configure: ** Running Mode          : DEBUG (binaries'extension is .debug)
configure: ** MPI Fortran Wrapper   : mpiifort
configure: ** Fortran Compiler      : ifort
configure: ** Fortran flags         : -O0 -g -u -warn all -check bounds -no-ipo
                                      -no-prec-div -inline -heap-arrays -traceback -xHost
configure: ** 2decomp Library       : -L ../2decomp_fft/src/ -l2decomp_fft
configure: ** FFTW3 Interface       : fftw3 of the FFTW3 library
configure: ** FFTW3 Path            : /usr/local/fftw-3.3.7/Intel/2018/impi//lib
configure: ** FFTW3 Includes        : -I /usr/local/fftw-3.3.7/Intel/2018/impi//include
configure: ** FFTW3 Library         : -lfftw3
configure: ** BLAS Type             : LAPACK
configure: ** BLAS Path             : /usr/local/Libraries/lapack-3.8.0/Intel/2018
configure: ** Prefix installation   : /home/lhj/neutron/Tinker/REL/PME/v1.2
configure: ** Binaries location     : /home/lhj/neutron/Tinker/REL/PME/v1.2/bin
configure: **
configure: ************************************************************************
configure:
```

This log confirms that we are in DEBUG mode and that we use the Intel compiler `ifort` and the `mpiifort` wrapper from IntelMPI. The installation directory where all binaries (with `.debug` extension) will be installed is shown as well.

D. Making binaries

Once you are happy with the option you selected, it's time to run the `make` command, or even the `make install` command, which will compile and install all at once.

As the compilation process takes care of the dependencies between subroutines and modules, you can safely use the `-j` flag of the `make` command to do parallel compilation. This would dramatically speedup the compilation process.

Anyway, on modern machines, the compilation is not very long, except for 2 or 3 subroutines that can take up to 5(!) minutes to compile, depending on the compiler you use and even on the fastest machines. Everything should compile and link gracefully. Using `make install` will copy the binaries into the directory you selected with the `--prefix=<DIR>` option. Don't forget to install the binaries you created, or you will not be able to run the examples.

IV. Note for developers

A. Writing new sources

We don't want to impose you a unique style of writing. Indeed, we don't have one. But we just want to give you some rules we believe are important for the consistency of Tinker-HP's code.

a) File format: We use FIXED FORM format throughout all the code, even though the code is written in FORTRAN90. This is mandatory. The compilation process would not work otherwise.

b) Editing: We always use lowercase letters for code (except for printing purposes). We indent all lines embedded in `do.....enddo`, `do.....while`, etc... statements, or in `if...else...endif` constructs.

c) Variable declarations: You are required to use `implicit none`. If you compile in debug mode, that will be enforced by the compiler.

The order we use to declare variables is:

1) `integer` (4 bytes sized)
2) `real` (8 bytes sized)

3) `logical`
4) `array` (in the same order)
5) `character` (single string or array)

We always try not to mix different types of variables in the same declaration line. This is not just because it is easier to read. That is also because it is more memory efficient, particularly for vectorization, where alignment in memory is crucial. `character` should be put at the very end, since they can have arbitrary lengths and almost never align to a memory boundary. We also try to choose significant names for the variables.

d) Comments: We always begin a comment line by the `c` character. The `!` character should only appear in the middle of a line. This is because the `!` character at the beginning is reserved to introduce compiler directives.

If ever you create modules, please comment all the new variables you create, like :

```
c     maxvalue        atoms directly bonded to an atom
c     maxgrp          user-defined groups of atoms
c     maxtyp          force field atom type definitions
c     maxclass        force field atom class definitions
```

In subroutines or functions, give as many comments as you believe is needed to understand what your code is doing. That would be precious for you, and for us as well.

B. Compiling new sources

If you make development on Tinker-HP, it is likely that you would need to add subroutines and modules in the source directory.

All modules should be put in files named `MOD_xxxxx.f`, even though they are written in FORTRAN90. All functions and routines should be put in files with names beginning by a lowercase letter and with `.f` extension. Please, try to find significant names (`epolar1tcg2shortreal.f` is far better than `ep1tc2shre.f`). You are required to follow this scheme as much as possible.

To compile your new sources, you should add them in the `Makefile.am` file of the `source` directory. We've put some comments in this file, to help you know where to put things. Search for the string `Add` in the file.

There are 3 different cases[1]:

1) You created a new main program (like `analyze` or `dynamic`). Add a line
   `bin_PROGRAMS += yourmain`
   (with no extension) below the line `bin_PROGRAMS += testgrad`. Then, add the lines
   `yourmain_SOURCES = yourmain.f`
   and
   `yourmain_DEPENDENCIES = libtinkermod.a libtinkercalc.a`
   after the similar lines concerning `testgrad`.
2) You created new module(s). Add lines
   `libtinkermod_a_SOURCES += MOD_xxxxx.f`
   just below[2] `libtinkermod_a_SOURCES += MOD_virial.f`
3) You created functions and subroutines. Add lines like
   `libtinkercalc_a_SOURCES += yourexplicitfilename.f`
   just below[3] the line `libtinkercalc_a_SOURCES += version.f`

You should now go in the main directory, where `configure.ac` resides, and type `autoconf` and `automake`. `autoconf` should not generate any message. `automake` will probably do, mainly because of a different version than the one we used to create the distribution. In this case, just type `aclocal` before running `automake` again. These 2 (or 3) commands will generate a new `configure` script that takes care of your new sources. You should then run `./configure`[4], compile, install and enjoy debugging your code.

V. EXECUTABLES

After having successfully compiled the code, five executable files should be present in the install directory: `analyze`, `bar`, `dynamic`, `testgrad` and `minimize`[5], which are the analogous of the binaries of the Tinker-8.4 release and require

[1] Of course, you can match all three at the same time!

[2] As the compilation process takes care of all the dependencies, the positions of the lines you add are not really significant. But putting the new lines at the end is just a way of remembering they are – well – new.

[3] Same remark as above.

[4] Presumably with the `--enable-debug` option flag.

[5] If you have ever compiled with `--enable-debug` before, you should have 5 more binaries.

similarly a geometry (given by a *.xyz file), a simulation setup (given by a *.key file) and possibly a restart (given by a *.dyn file) for the `dynamic` program.

All these executables must run in the same environment you had during the compilation phase. That means the same set of modules, or the correct `LIBRARY_PATH`. They should be launched with the `mpirun -np x` prefix in order to run in parallel with `x` MPI processes.

The only boundary conditions that are available in this release are periodic boundary conditions treated with Particle Mesh Ewald. Classical force fields such as AMBER, CHARMM and OPLS are available in Tinker-HP as well as polarizable force fields such as AMOEBA.

A. `analyze`

The `analyze` executable allows potential energy analysis. Compared to the Tinker-8.4 software, the only option compatible with this binary is "e".

For example the command line:

```
mpirun -np 16 ./analyze dhfr2 e
```

will give you as an output the potential energy terms of the geometry given by a dhfr2.xyz file and with the simulation setup given by the dhfr2.key file. Furthermore, this computation will run on 16 MPI processes.

B. `dynamic`

The `dynamic` executable allows to run molecular dynamics simulation. As for Tinker-8.4, the command line used to run the MD should give first the number of MD steps to make, then the size of each time step (in femtoseconds), then the time between each writing of geometry (in picoseconds), then the statistical ensemble to sample : 1 is NVE, 2 is NVT, 4 is NPT. For NVT and NPT simulations, this number should be followed by the temperature (in Kelvin) of the simulation, and for NPT simulation by the pressure (in Atmosphere) of the simulation.

For example the command lines:

```
mpirun -np 16 ./dynamic dhfr2 1000 1 1 1
```

will give you as an output 1000 MD steps in NVE for the dhfr2 system, with a 1 *fs* time step and a 1 *ps* frequency output.

```
mpirun -np 16 ./dynamic dhfr2 1000 1 1 2 300
```

will give you as an output 1000 MD steps in NVT at 300K for the dhfr2 system, with a 1 *fs* time step and a 1 *ps* frequency output.

```
mpirun -np 16 ./dynamic dhfr2 1000 1 1 4 300 1
```

will give you as an output 1000 MD steps in NPT at 300K and 1atm for the dhfr2 system, with a 1 *fs* time step and a 1 *ps* frequency output.

C. `testgrad`

The `testgrad` program is absolutely equivalent to the one of the Tinker-8.4 release: it allows the output of the components of the analytical and/or numerical gradients of the different energy terms.

For example, the command line:

```
mpirun -np 16 ./testgrad dhfr2 Y Y 0.0001 Y
```

will give you as an output all the analytical and numerical gradients (computed with an increment of 0.0001 Angstroms for the positions of the atoms) of all the energy terms of the dhfr2 system.

D. `minimize`

The `minimize` program computes energy minimization starting from a given structure, using a low memory quasi-newton BFGS algorithm as in Tinker-8.4. The command line used should give the numerical threshold for the convergence of the algorithm.

For example, the command line:

```
mpirun -np 16 ./minimize dhfr2 0.1
```

will compute energy minimization on the dhfr2 structure until the RMS on the gradient is inferior to 0.1. The new geometry will be written at each iteration of the algorithm in the file dhfr2.xyz_2.

VI. KEYWORDS

The main keywords of Tinker-8.4 are available in Tinker-HP. So, a description of these keywords can be found in the Tinker user guide. Let us review a few of these and some new ones which are specific to Tinker-HP.

A. Keywords specific to the `dynamic` program

- **Integrators:** As in Tinker, the integrator of a dynamic is imposed by the keyword **"integrator x"**, x being one of the available integrator:

  BEEMAN : The default one
  VERLET : Verlet
  BBK : Langevin Dynamics for constant temperature simulations
  BAOAB : Langevin Dynamics for constant temperature simulations
  RESPA : Bonded/non bonded respa-split with a velocity-verlet inner loop and with a 0.25 *fs* default timestep for the inner loop
  BAOABRESPA : Bonded/non bonded respa-split for Langevin dynamics with a **BAOAB** inner loop, the default time step for the inner loop is also 0.25 *fs*
  RESPA1 : (Bonded)/(short range non bonded)/(long range non bonded) three level respa1-split with a velocity verlet inner loop. The default timesteps are 0.25 *fs* for the inner loop and 2 *fs* for the intermediate one
  BAOABRESPA1 : (Bonded)/(short range non bonded)/(long range non bonded) three level respa1-split for Langevin dynamics with a **BAOAB** inner loop. The default timesteps are 0.25 *fs* for the inner loop and 2 *fs* for the intermediate one
  BAOABPISTON : Constant pressure **BAOAB** Langevin dynamics with a Langevin Piston pressure control and a **BAOAB** evolution of the volume extended variable. The default mass of the piston is $2e^{-5}$ atomic units and the default friction for the piston is 20.0 ps^{-1}.

  For all the Langevin integrators (**BBK**, **BAOAB**, **BAOABRESPA**, **BAOABRESPA1** and **BAOABPISTON**), the friction (in ps^{-1}) can be controlled by the keyword `friction x`, the default being 1 ps^{-1}.
  For **RESPA**, **BAOABRESPA**, **RESPA1** and **BAOABRESPA1**, the inner timestep can be imposed by the keyword `dshort x`, `x` being its desired value in *ps*.
  For **RESPA1** and **BAOABRESPA1**, the intermediate timestep can be imposed by the keyword `dinter x`, `x` being its desired value in *ps*.
  For **BAOABPISTON**, the mass of the piston (in atomic units) can be set by the keyword `masspiston x` and the friction of the piston (in ps^{-1}) can be set by the keyword `frictionpiston x`.

- **Thermostats and barostats**: Aside from the Langevin integrators, the thermostats available in Tinker-HP are Berendsen, Bussi (which is the default one) and Andersen. Aside from the Langevin Piston, the barostats available in Tinker-HP are the Berendsen (which is the default one) and the Monte-Carlo one. These option can be set by putting the keywords: `thermostat berendsen`, `thermostat bussi`, `thermostat andersen`, `barostat berendsen` and `barostat montecarlo` in the key file.

Tinker-HP deals with restart files for dynamic trajectories the same way as Tinker-8.4 does by creating a *.dyn file encompassing current positions, velocities and accelerations of the system.

B. New keywords specific to Tinker-HP

Some new keywords have been introduced in Tinker-HP. The first one concern the algorithm used to converge the polarization equations.

polar-alg `x` : choose the algorithm used to compute the dipoles solution of the polarization equations. `x` can be:
`1` : Conjugate Gradient with a diagonal preconditioner
`2` : Jacobi/DIIS
`3` : Truncated Conjugate Gradient (TCG)
`5` : Divide and Conquer Jacobi/DIIS (default)

TCG is a systematically improvable method with 4 tunable parameters that can be controlled by different keywords, each of them being prefixed by `tcg`:

tcgorder `x` : order of the TCG truncation, `x` can take the value 1 (TCG1) or 2 (TCG2), default is 2
tcgprec `x` : use of a diagonal preconditioner. `x` can take the value 1 (YES) or 0 (NO), default is 1
tcggues `x` : use of a "direct guess" ($polarizability \times permanent_electric_field$) as a guess. `x` can take the value 1 (YES) or 0 (NO), default is 0.
tcgpeek`x` : use of a peek step. `x` can take the value 1 (YES) or 0 (NO), default is 1. When a peek step is used, a Jacobi Over Relaxation (JOR) is applied to the TCG values of the dipoles with a parameter ω. By default, this value is $\omega = 1$. (regular Jacobi step) but three keywords can modify this:

tcgomega x : change the value of the ω parameter to x.
tcgomegafit : impose a regular fitting of the ω parameter to match at regular intervals the fully converged polarization energy.
tcgomegafitfreq x : number of timesteps between two updates of the fitted ω parameter. The default of x is 1000.

When the **RESPA1** and **BAOABRESPA1** integrators are used, one has to solve the short range real space polarization equations at the intermediate time steps. By default, this is done using the same algorithm as the one used to solve the complete polarization equations at the outer time steps. But one can chose a different algorithm to solve the short range polarization by using the keyword **polar-algshort** with the same possible values as for **polar-alg**. If TCG is chosen as a short range polarization solver, one can define all the related option for this solver by taking the same keywords defined above and adding the suffix **short**:

tcgordershort : order of the short range TCG truncation
tcgprecshort : use of diagonal preconditioner or not
tcgguessshort : use of a "direct guess"
tcgpeekshort : use of a peek
tcgomegashort ω : choice of the peek step parameter

With the introduction of Steered Molecular Dynamics come two keywords:

CVSMD for Constant Velocity Steered Molecular Dynamics
CFSMD for Constant Force Steered Molecular Dynamics.

The use of those two forms of steered molecular dynamics (SMD) is described in a dedicated tutorial which can be found in the `tutorials/SMD/` directory, along with two subdirectories (`CFSMD` and `CVSMD`) containing input files, running scripts and output files. The 2 Tinker Archives corresponding to **CVSMD** and **CFSMD** (178MB each) can be downloaded at `http://tinker-hp.ip2ct.upmc.fr/?Download-Process`.

Unlike Tinker-8.4, the neighbor-lists for non bonded interactions are computed every x steps (x=20 being the default for a 2 *fs* time step) and not adjusted dynamically at each time step. This frequency of update can be modified by using the keywords **nlupdate x**, 1 corresponding to a neighbor-list update at each time step (which can be useful during equilibration for example).

Other keywords have been introduced to specify parallel options when running Tinker-HP. The current distributed release is only available with the PME algorithm which involves direct and reciprocal space computations in the electrostatic and polarization interactions. As the reciprocal space interactions are known to have a less efficient parallel scaling (because of FFTs) it is possible to specify a lower number of MPI processes that will be dedicated to these computations for both the computation of electrostatic and polarization interactions. This can be done by using the keyword **pme-procs** x, corresponding to x MPI processes dedicated to reciprocal space computations.

To find the ideal x value of **pme-procs**, a good starting point, when a large number of cores is used, is usually to dedicate about $\frac{1}{4}$ of the total cores to reciprocal space computations. But this parameter depends greatly on the machine used and on the setup of your simulation so it should be adjusted manually by comparing the timings obtained with different values for **pme-procs**. During a dynamic, detailed timings are written when the **verbose** keyword is in the *.key file. In the future, the parameter **pme-procs** will be adapted heuristically by the program as it is done in popular MD packages.

As explained in the beginning of the document, Tinker-HP distributes the data on a grid used to run 3D FFT in 2d pencils which can be associated to a 2d processor grid as explained in 2DECOMP_FFT web page. By default, the library looks for an optimal 2d processor grid given the number of cores available but this decomposition can be imposed by the user by setting the keyword: **decompfft-grid n1 n2**, corresponding to a n1$\times$n2 processor grid.

Note that the user does not have to specify that he wants neighbor-lists to be used as it is the only available option in Tinker-HP.

VII. EXAMPLES

4 examples of systems with associated *.key files are given in the distribution: ubiquitin2, dhfr2, puddle and pond. The sizes of these systems are respectively: 9737, 23558, 96000 and 288000 atoms, making them good various benchmarks for the program.

4 different setups are given for the ubiquitin system with 4 different key files:

1) **ubiquitin2.key** : regular (langevin with BAOAB integration based) 2 *fs* respa (bonded/non-bonded split) computations with DC-JI/DIIS as a polarization solver
2) **ubiquitin2tcg.key** : 2 *fs* respa computations with TCG2 (with a diagonal preconditioner, no guess and a peek step with $\omega = 1$ as a polarization solver
3) **ubiquitin2respa1.key** : 6 *fs* respa1 (bonded/short range non-bonded/long range non-bonded split) langevin with BAOAB integration computations with DC-JI/DIIS as a short and total polarization solver
4) **ubiquitin2respa1tcg.key** : 10 *fs* respa1 langevin with BAOAB integration computations with heavy hydrogen, TCG1 (with a diagonal preconditioner, no guess and no peek step) as a short range polarization solver and DC-JI/DIIS as a total polarization solver

A fifth example is reserved for debug purposes. It's exactly the same as the first one. `./ubiquitin2.debug.run` runs the `dynamic.debug` binary.

VIII. Tutorials

Three tutorials can be found in the directory **tutorials** of the release:

A general tutorial to prepare systems for Tinker/Tinker-HP : **Tinker_preparation_tutorial.pdf**
A tutorial to use Umbrella Sampling with Tinker/Tinker-HP : **Umbrella_sampling_tutorial.pdf**
A tutorial to use Steered MD with Tinker-HP : **SMD/SMD_manual.pdf**

IX. Support

Tinker-HP is maintained by few people. That means we cannot promise you to answer in a minute to your requests. Anyway, if you have any question or need any support for Tinker-HP, feel free to send a mail to our team at `TinkerHP_Support@ip2ct.upmc.fr`. We will answer as soon as we can, providing that we can !