

LAMMPS

- LAMMPS
 - Tips
 - Setup for build
 - Paths
 - <.bashrc>
 - Dependencies
 - Zlib 1.2.11
 - HDF5 1.8.18
 - PNETCDF 1.12.3
 - NETCDF-C 4.8.1
 - NETCDF-fortran 4.6.1
 - ADIOS 2.8.0
 - Visualization tools
 - Others
 - Build LAMMPS
 - Run LAMMPS
 - Lennard Jones Simulation
 - LAMMPS.out
 - Visualization
 - JPG
 - FFMPEG
 - Performance
 - Number of MPI ranks (np).
 - Number of OpenMP threads per MPI rank
 - Modify <in.lj> script

Tips

1. 最重要的：所有人都要用一樣的**compiler**！！
2. 每新安裝一個dependency 記得 export path
3. 如果發現是路徑有問題直接開新的terminal重裝
4. 有用cmake開一個folder用來build · 失敗就刪掉重來
5. configure 和 install 的 folder 分開
 - build: for configuration
 - opt: for installation

Setup for build

```
mkdir lammps
cd lammps
mkdir build opt
```

Paths

```
ROOT=$HOME/lammps
BUILD=$HOME/lammps/build
OPT=$HOME/lammps/opt

HDF5=hdf5-1.8.18
PNETCDF=pnetcdf-1.12.3
NETCDF=netcdf-c-4.8.1
NETCDF_FORTRAN=netcdf-fortran-4.6.1
PYTHON=Python-3.12.0
ADIOS=ADIOS-2.8.0
VORO=voro++-0.4.6
LAMMPS=lammps-2Aug2023
```

<.bashrc>

Compiler: GCC 11.4.0

```
export PATH=/home/scsteam06/openmpi/bin:$PATH
export LD_LIBRARY_PATH=/home/scsteam06/openmpi/lib:$LD_LIBRARY_PATH

export HDF5_DIR=/home/scsteam06/lammps/opt/hdf5-1.8.18
export PNETCDF_DIR=/home/scsteam06/lammps/opt/pnetcdf-1.12.3
export NETCDF_DIR=/home/scsteam06/lammps/opt/netcdf-c-4.8.1
export NETCDF_FORTRAN_DIR=/home/scsteam06/lammps/opt/netcdf-fortran-4.6.1
export ADIOS_DIR=/home/scsteam06/lammps/build/ADIOS-2.8.0
export VORO_DIR=/home/scsteam06/lammps/opt/voro++-0.4.6
export ZFP_DIR=/home/scsteam06/lammps/opt/zfp

export PATH=$HDF5_DIR/bin:$PNETCDF_DIR/bin:$NETCDF_DIR/bin:$ADIOS_DIR/bin:$VORO_DIR/bin:$ZFP_DIR/bin:$PATH
export LD_LIBRARY_PATH=$HDF5_DIR/lib:$PNETCDF_DIR/lib:$NETCDF_DIR/lib:$ADIOS_DIR/lib:$VORO_DIR/lib:$ZFP_DIR/lib:$LD_LIBRARY_PATH
export C_INCLUDE_PATH=$HDF5_DIR/include:$PNETCDF_DIR/include:$NETCDF_DIR/include:$ADIOS_DIR/include:$VORO_DIR/include:$ZFP_DIR/include:$C_INCLUDE_PATH
export CPLUS_INCLUDE_PATH=$HDF5_DIR/include:$PNETCDF_DIR/include:$NETCDF_DIR/include:$ADIOS_DIR/include:$VORO_DIR/include:$ZFP_DIR/include:$CPLUS_INCLUDE_PATH
export LIBRARY_PATH=$HDF5_DIR/lib:$PNETCDF_DIR/lib:$NETCDF_DIR/lib:$ADIOS_DIR/lib:$VORO_DIR/lib:$ZFP_DIR/lib:$LIBRARY_PATH

export PATH=/home/scsteam06/lammps/build/lammps-2Aug2023/bin:$PATH
export LD_LIBRARY_PATH=/home/scsteam06/lammps/build/lammps-2Aug2023/lib:$LD_LIBRARY_PATH
```

```
source ~/.bashrc
```

Dependencies

Zlib 1.2.11

```
cd $BUILD
wget https://nchc.dl.sourceforge.net/project/libpng/zlib/1.2.11/zlib-1.2.11.tar.gz
tar -xvf zlib-1.2.11.tar.gz
cd zlib-1.2.11

CC=mpicc CXX=mpicxx \
./configure \
--prefix=$HOME/opt/zlib-1.2.11 #zlib路徑不影響


make -j
sudo make install
```

HDF5 1.8.18

```
cd $BUILD
wget https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.8/hdf5-1.8.18/src/hdf5-1.8.18.tar.gz
tar xf $HDF5.tar.gz
cd $HDF5

CC=mpicc CXX=mpicxx FC=mpifort \
./configure \
--prefix=$OPT/$HDF5 \
--enable-shared \
--enable-parallel

make -j
sudo make install
```



```
Libraries have been installed in:
/home/scteam06/lampps/opt/hdf5-1.8.18/lib
```

```
If you ever happen to want to link against installed libraries
in a given directory, LIBDIR, you must either use libtool, and
specify the full pathname of the library, or use the '-LLIBDIR'
flag during linking and do at least one of the following:
- add LIBDIR to the 'LD_LIBRARY_PATH' environment variable
  during execution
- add LIBDIR to the 'LD_RUN_PATH' environment variable
  during linking
- use the '-Wl,-rpath -Wl,LIBDIR' linker flag
- have your system administrator add LIBDIR to '/etc/ld.so.conf'
```

```
See any operating system documentation about shared libraries for
more information, such as the ld(1) and ld.so(8) manual pages.
```

PNETCDF 1.12.3

```
cd $BUILD
wget https://parallel-netcdf.github.io/Release/pnetcdf-1.12.3.tar.gz
tar xf $PNETCDF.tar.gz
cd $PNETCDF

CC=mpicc CXX=mpicxx FC=mpifort \
./configure \
--enable-shared \
--enable-subfilong \
--prefix=$OPT/$PNETCDF

make -j
sudo make install
```

```
PnetCDF has been successfully installed under
/home/scteam06/lammps/opt/pnetcdf-1.12.3

* PnetCDF header files have been installed in
  /home/scteam06/lammps/opt/pnetcdf-1.12.3/include
* PnetCDF library files have been installed in
  /home/scteam06/lammps/opt/pnetcdf-1.12.3/lib
* PnetCDF utility programs have been installed in
  /home/scteam06/lammps/opt/pnetcdf-1.12.3/bin
* PnetCDF man pages have been installed in
  /home/scteam06/lammps/opt/pnetcdf-1.12.3/share/man

To compile your PnetCDF programs, please add the following to the command
line, so the compiler can find the PnetCDF header files:
  -I/home/scteam06/lammps/opt/pnetcdf-1.12.3/include

Add the following line to link your program to PnetCDF library:
  -L/home/scteam06/lammps/opt/pnetcdf-1.12.3/lib -lpnetcdf

Add the following to your run-time environment variable LD_LIBRARY_PATH,
when linking your executable with the PnetCDF shared libraries.
  /home/scteam06/lammps/opt/pnetcdf-1.12.3/lib
```

NETCDF-C 4.8.1

```
cd $BUILD
wget https://github.com/Unidata/netcdf-c/archive/refs/tags/v4.8.1.tar.gz
tar xf v4.8.1.tar.gz
cd $NETCDF

CC=mpicc CXX=mpicxx FC=mpifort \
./configure \
--enable-pnetcdf \
--enable-hdf5 \
--enable-shared \
--prefix=$OPT/$NETCDF

make -j
sudo make install
```

Congratulations! You have successfully installed netCDF!

You can use script "nc-config" to find out the relevant compiler options to build your application. Enter

```
nc-config --help
```

for additional information.

CAUTION:

If you have not already run "make check", then we strongly recommend you do so. It does not take very long.

Before using netCDF to store important data, test your build with "make check".

NetCDF is tested nightly on many platforms at Unidata but your platform is probably different in some ways.

If any tests fail, please see the netCDF web site:
<http://www.unidata.ucar.edu/software/netcdf/>

NETCDF-fortran 4.6.1

```
cd $BUILD
wget https://downloads.unidata.ucar.edu/netcdf-fortran/4.6.1/$NETCDF_FORTTRAN.tar.gz
tar xf $NETCDF_FORTTRAN.tar.gz
cd $NETCDF_FORTTRAN

CC=mpicc CXX=mpicxx FC=mpifort \
./configure \
--prefix=$OPT/$NETCDF_FORTTRAN \
--enable-shared \

make -j
sudo make install
```

Congratulations! You have successfully installed the netCDF Fortran libraries.

You can use script "nf-config" to find out the relevant compiler options to build your application. Enter

```
nf-config --help
```

for additional information.

CAUTION:

If you have not already run "make check", then we strongly recommend you do so. It does not take very long.

Before using netCDF to store important data, test your build with "make check".

NetCDF is tested nightly on many platforms at Unidata but your platform is probably different in some ways.

If any tests fail, please see the netCDF web site:
<https://www.unidata.ucar.edu/software/netcdf/>

ADIOS 2.8.0

```
cd $BUILD
git clone -b v2.8.0 https://github.com/ornladios/ADIOS2.git $ADIOS
cd $ADIOS
mkdir build
cd build
```

```
CCC=mpicc CXX=mpicxx FC=mpifort \
cmake .. \
-DMPI_Fortran_COMPILER=mpifort \
-DCMAKE_Fortran_COMPILER=mpifort \
-DCMAKE_C_COMPILER=mpicc \
-DCMAKE_CXX_COMPILER=mpicxx \
-DCMAKE_CXX_FLAGS="-O3 -fpic" \
-DADIOS2_USE_MPI=ON \
-DADIOS2_USE_Endian_Reverse=ON \
-DADIOS2_USE_Fortran=ON \
-DADIOS2_RUN_INSTALL_TEST=OFF \
-DCMAKE_FIND_ROOT_PATH_MODE_LIBRARY=ONLY \
-DCMAKE_FIND_ROOT_PATH_MODE_INCLUDE=ONLY \
-DCMAKE_INSTALL_PREFIX=$BUILD/$ADIOS \
-DADIOS2_INSTALL_GENERATE_CONFIG=OFF \
-DHDF5_DIR=/home/scsteam06/lammps/opt/hdf5-1.8.18
-DADIOS2_BUILD_HDF5=ON #不要用 make -j
```

```
sudo make install
```

Visualization tools

- JPEG

```
sudo apt-get install libjpeg-dev
```

- FFMPEG

```
sudo apt-get install ffmpeg
```

Others

- ZFP

```
cd $BUILD
wget https://github.com/LLNL/zfp/archive/refs/tags/0.5.5.tar.gz
tar -xzf 0.5.5.tar.gz
cd zfp-0.5.5
mkdir build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=$OPT/zfp
make -j
make install
```

- LAPACK

```
sudo apt-get install liblapack-dev
```

Build LAMMPS

Version: 2Aug2023

```
OPTFLAGS="-march=cascadelake -O2"
```

```
CC=mpicc
```

```
CXX=mpicxx
```

```
FC=mpifort
```

```
CFLAGS="-fPIC -march=cascadelake -fopenmp -Wrestrict -DLMP_INTEL_USELRT -DLMP_USE_MKL_RNG
```

```
CXXFLAGS="-fPIC -march=cascadelake -fopenmp -Wrestrict -DLMP_INTEL_USELRT -DLMP_USE_MKL_RNG
```

```
FCFLAGS="-fPIC -march=cascadelake -Wrestrict -DLMP_INTEL_USELRT -DLMP_USE_MKL_RNG $OPTFLA
```

```
LDFLAGS="$OPTFLAGS -L$MKLR00T/lib/intel64 -ltbbmalloc -lmkl_intel_ilp64 -lmkl_sequential
```

```
cmake ../cmake \
```

```
-DCMAKE_INSTALL_PREFIX=$BUILD/$LAMMPS \
```

```
-DBUILD_SHARED_LIBS=yes \
```

```
-DINTEL_ARCH=cpu \
```

```
-DFFT=MKL \
```

```
-DFFT_MKL_THREADS=on \
```

```
-DWITH_GZIP=yes \
```

```
-DPKG_ADIOS=yes \
```

```
-DADIOS2_DIR=$ADIOS_DIR \
```

```
-DPKG_NETCDF=yes \
```

```
-DBUILD_MPI=yes \
```

```
-DBUILD_OMP=yes \
```

```
-DLAMMPS_MACHINE=mpi \
```

```
-DPKG_OPENMP=yes \
```

```
-DPKG_OPT=yes \
```

```
-DPYTHON_EXECUTABLE=$(which python3) \
```

```
-DPKG_ML-QUIP=yes \
```

```
-DPKG_ML-HDNNP=yes \
```

```
-DPKG_VORONOI=yes \
```

```
-DWITH_JPEG=yes \
```

```
-DLAMMPS_FFMPEG=yes
```

```
make -j
```

```
make install
```

- Get result.txt (lmp_mpi 就在 /lammps-2Aug2023/build)

```
lmp_mpi -help > result.txt
```

```
Installed packages:
```

```
ADIOS ML-HDNNP ML-QUIP NETCDF OPENMP OPT VORONOI
```

Run LAMMPS

Lennard Jones Simulation

- lammps/bench/in.lj : set *run* = 10000
- nproc = 8


```
mpirun -np 8 lmp_mpi -in in.lj
```

```
scteam06@head:~/lammps/build/lammps-2Aug2023/bench$ mpirun -np 8 lmp_mpi -in in.lj_op
LAMMPS (2 Aug 2023 - Update 3)
OMP_NUM_THREADS environment is not set. Defaulting to 1 thread. (src/comm.cpp:98)
using 1 OpenMP thread(s) per MPI task
Lattice spacing in x,y,z = 1.6795962 1.6795962 1.6795962
Created orthogonal box = (0 0 0) to (33.591924 33.591924 33.591924)
  2 by 2 by 2 MPI processor grid
Created 32000 atoms
  using lattice units in orthogonal box = (0 0 0) to (33.591924 33.591924 33.591924)
  create_atoms CPU = 0.001 seconds
Generated 0 of 0 mixed pair_coeff terms from geometric mixing rule
Neighbor list info ...
  update: every = 100 steps, delay = 0 steps, check = yes
  max neighbors/atom: 2000, page size: 100000
  master list distance cutoff = 2.8
  ghost atom cutoff = 2.8
  binsize = 1.4, bins = 24 24 24
  1 neighbor lists, perpetual/occasional/extra = 1 0 0
  (1) pair lj/cut, perpetual
    attributes: half, newton on
    pair build: half/bin/atomonly/newton
    stencil: half/bin/3d
    bin: standard
Setting up Verlet run ...
Unit style      : lj
Current step    : 0
Time step       : 0.005
Per MPI rank memory allocation (min/avg/max) = 3.725 | 3.725 | 3.725 Mbytes
Step      Temp      E_pair      E_mol      TotEng      Press
   0      1.44      -6.7733681      0      -4.6134356      -5.0197073
 10000     0.56854136     -5.8170392      0     -4.9642538     -0.10625207
Loop time of 26.5222 on 8 procs for 10000 steps with 32000 atoms

Performance: 162882.717 tau/day, 377.043 timesteps/s, 12.065 Matom-step/s
96.1% CPU use with 8 MPI tasks x 1 OpenMP threads

MPI task timing breakdown:
Section | min time | avg time | max time | %varavg | %total
-----|-----|-----|-----|-----|-----
Pair    | 21.079   | 21.27    | 21.47    | 3.1      | 80.20
Neigh    | 0.54558  | 0.55441  | 0.56211  | 0.9      | 2.09
Comm     | 3.9106   | 4.087    | 4.294    | 6.9      | 15.41
Output   | 0.00050116 | 0.00051907 | 0.00062915 | 0.0      | 0.00
Modify   | 0.28537  | 0.29538  | 0.30393  | 1.2      | 1.11
Other    |          | 0.3145   |          |          | 1.19

Nlocal:          4000 ave          4012 max          3986 min
Histogram: 1 1 0 1 0 1 0 3 0 1
Nghost:          5472.5 ave          5497 max          5453 min
Histogram: 1 0 3 0 1 0 1 1 0 1
Neighs:          149582 ave          151245 max          147941 min
Histogram: 1 1 1 0 1 1 0 1 1 1

Total # of neighbors = 1196654
Ave neighs/atom = 37.395437
Neighbor list builds = 100
Dangerous builds = 100
Total wall time: 0:00:26
```

LAMMPS.out

Output with time command

```
/usr/bin/time -v mpirun -np 4 lmp_mpi -in in.lj > LAMMPS.out 2>&1
```

```
Total # of neighbors = 1200062
Ave neighs/atom = 37.501937
Neighbor list builds = 500
Dangerous builds not checked
Total wall time: 0:00:27
  Command being timed: "mpirun -np 8 lmp_mpi -in in.lj"
  User time (seconds): 215.66
  System time (seconds): 7.27
  Percent of CPU this job got: 796%
  Elapsed (wall clock) time (h:mm:ss or m:ss): 0:27.98
  Average shared text size (kbytes): 0
  Average unshared data size (kbytes): 0
  Average stack size (kbytes): 0
  Average total size (kbytes): 0
  Maximum resident set size (kbytes): 29744
  Average resident set size (kbytes): 0
  Major (requiring I/O) page faults: 2318
  Minor (reclaiming a frame) page faults: 17929
  Voluntary context switches: 2245
  Involuntary context switches: 5667
  Swaps: 0
  File system inputs: 0
  File system outputs: 16536
  Socket messages sent: 0
  Socket messages received: 0
  Signals delivered: 0
  Page size (bytes): 4096
  Exit status: 0
```

Visualization

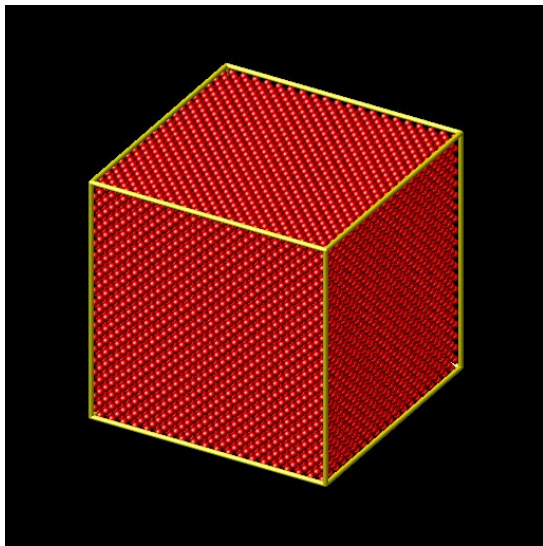
JPG

In <in.lj>, add:

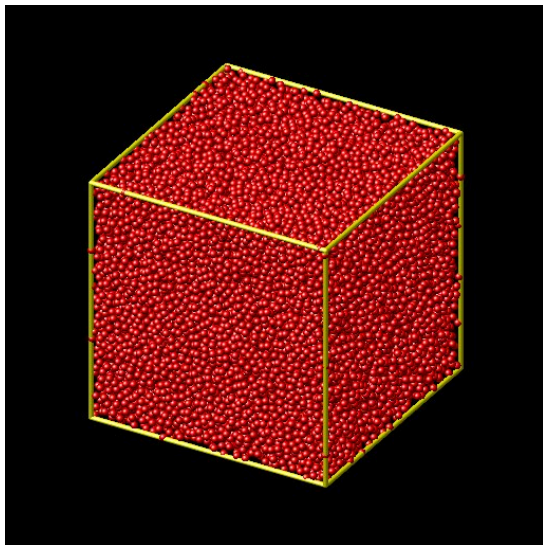
```
dump          1 all image 100 image.*.jpg type type &
              zoom 1.6 adiam 1.0
dump_modify   1 pad 5
```

Since the simulation runs for 10000 timesteps, and an image will be created every 100 timesteps, we should get 100 images in total (one for each of the following timesteps: 0, 100, 200, ..., 9900).

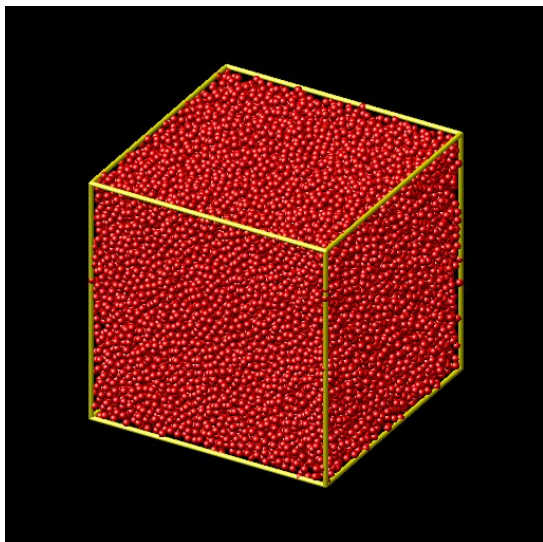
- **image.00000**



- image.05000



- image.10000



FFMPEG

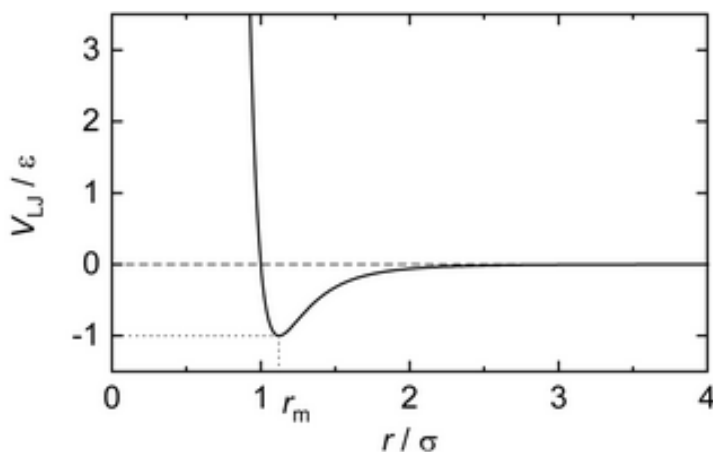
In <in.lj>, add:

```
dump          1 all movie 100 movie.mp4 type type &
              zoom 1.0 adiam 1.0
dump_modify   1 pad 5
```

This script creates [movie.mp4](https://drive.google.com/uc?export=download&id=1HHHyYj2UqOjjoehxxrv7bymTyAomf4jC) (<https://drive.google.com/uc?export=download&id=1HHHyYj2UqOjjoehxxrv7bymTyAomf4jC>), capturing frames every 100 timesteps of the simulation.

Performance

因為LAMMPS跑出來就會提供滿詳盡的performance數據，且較為直覺，所以雖然我中間也有用vtune測試過，後來還是以它本來的結果為主。



Lennard-Jones(LJ) potential這個數學模型是用來描述兩個電中性的分子或原子間交互作用位能，因此我們主要關心的結果就是圖中的的幾個物理量：

Step	Temp	E_pair	E_mol	TotEng	Press
0	1.44	-6.7733681	0	-4.6134356	-5.0197073
10000	0.70101409	-5.6728019	0	-4.6213136	0.72163282

Loop time of 28.1206 on 8 procs for 10000 steps with 32000 atoms

這是最開始的<in.lj>得到模擬結果(10000 steps)的，可以發現：

1. 溫度(Temp)明顯降低，表示系統趨於穩定(平衡)。
2. 由位能(E_pair)變大可知原子之間的距離變遠了。
3. 分子能量維持0，因為過程中沒有分子交互作用。
4. 總能量稍微降低(幾乎不變)，可能是平衡過程難免有能量散失。
5. 壓力由負的轉正，表示系統壓縮的狀態放鬆了。

因此接下來的優化過程中，要注意：

- 不能調整LJ相關的參數 pair_style 和 pair_coeff 。

```
pair_style lj/cut 2.5
pair_coeff 1 1 1.0 1.0 2.5
```

$$V(r) = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right]$$

- $\epsilon = 1.0$ (depth of the potential well)

- $\sigma = 1.0$ (finite distance where potential is zero)
- 2.5 : cutoff distance for the potential
- 結果得到的數值應保持一致，以確保在提升效能的同時模擬是精確的。

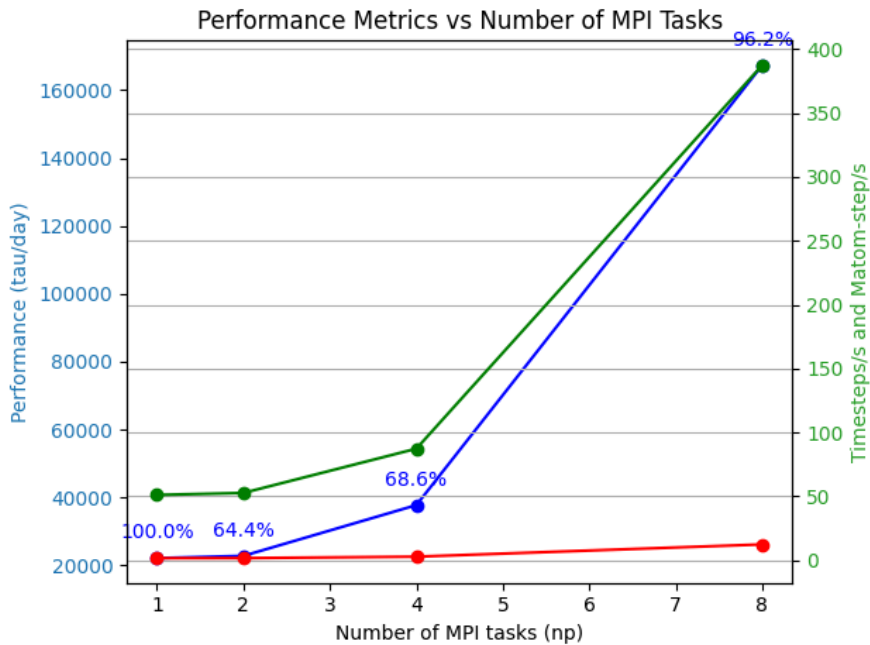
Performance: 153624.028 tau/day, 355.611 timesteps/s, 11.380 Matom-step/s
96.7% CPU use with 8 MPI tasks x 1 OpenMP threads

MPI task timing breakdown:

Section	min time	avg time	max time	%varavg	%total
Pair	20.974	21.176	21.438	3.7	75.31
Neigh	2.7127	2.754	2.9362	4.2	9.79
Comm	3.1904	3.6156	3.8694	11.4	12.86
Output	8.985e-05	0.000112	0.00015391	0.0	0.00
Modify	0.27982	0.28319	0.28674	0.4	1.01
Other		0.2913			1.04

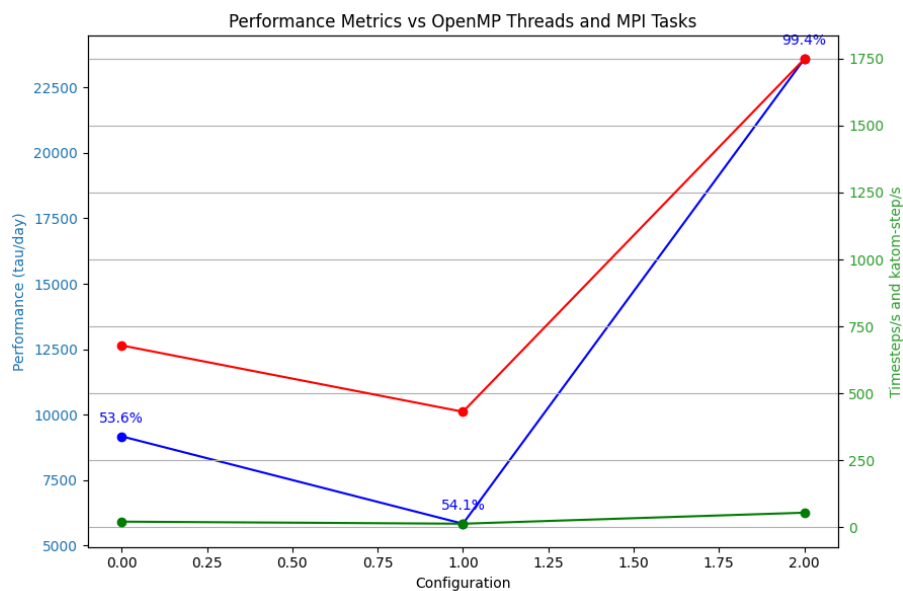
Performance主要看的是"tau/day", "timesteps/s", "Matom-step/s"CPU使用率。而由圖可看出該模擬最大的瓶頸就是在"pairing"這部份，"communication"也有進步空間。了解這些情況後就可以開始優化了。

Number of MPI ranks (np)



np = 8 時整體效率最好

Number of OpenMP threads per MPI rank



$n = 1$, $np = 8$ 時整體效率最好

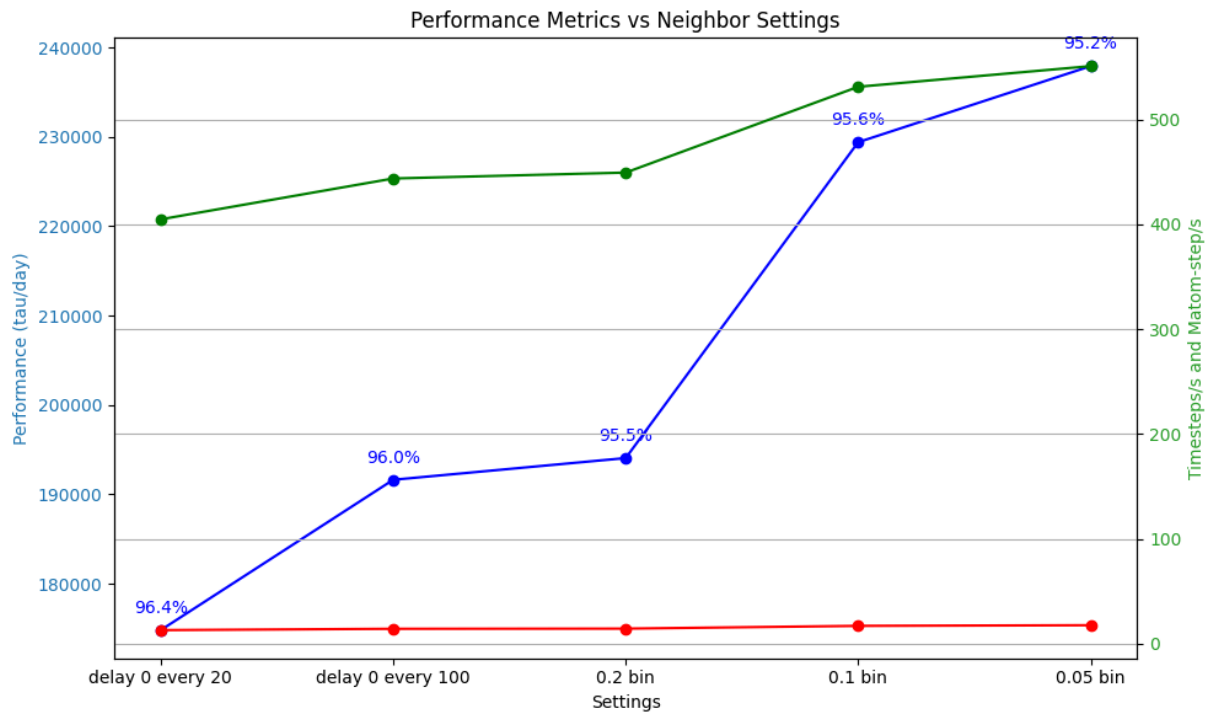
Modify <in.lj> script

- Atom sorting on/off

Sort Interval	Performance (tau/day)	Timesteps/s	Matom-step/s	CPU Usage (%)
0 1.0	157222.225	363.940	11.646	96.7
0 2.0	161417.419	373.651	11.957	95.9
50 2.0	170291.700	394.194	12.614	96.4
100 2.0	168046.347	388.996	12.448	96.2

- Performance (tau/day) 隨著 sort interval 的增加而提高，特別是在 50 2.0 時達到最高值。
 - Timesteps/s 和 Matom-step/s 也在 50 2.0 時達到最高值，這表明此配置在這些條件下性能最佳。
 - CPU Usage 在各種配置下變化不大，保持在 95-97% 之間。
- Newton flag on/off (預設是on)：發現off之後效能退步了。
- processors * * * : LAMMPS可以自動偵測最佳的processor分配方式，但實驗發現影響不大。
- Neighbor list:

```
neighbor      0.05 bin
neigh_modify  delay 0 every 100 check yes
```



- `neigh_modify delay 0 every 100 check yes`：提高更新頻率可以減少neighbor的重建次數，從而減少計算開銷，提升性能。然而，更新頻率過低可能導致鄰居列表過時，增加不必要的計算。從結果看來，`every 100` 的性能優於 `every 20`，說明在這個模擬中，減少鄰居列表的更新次數更有利於性能提升。
- `neighbor 0.05 bin`：bin 尺寸影響鄰居查找的效率。較小的 bin 尺寸可以減少每個 bin 中的原子數量，從而加快鄰居查找速度。從數據看來，將 bin 尺寸從 0.2 減少到 0.05，性能顯著提升。這說明較小的 bin 尺寸在這個模擬中更加有效，能夠顯著減少計算時間。
- 綜合考慮，最佳的配置是 `delay 0 every 100 check yes` 和 `neighbor 0.05 bin`，該配置下，性能指標（如 `tau/day`、`timesteps/s` 和 `Matom-step/s`）均達到最高。

```
Performance: 237914.742 tau/day, 550.729 timesteps/s, 17.623 Matom-step/s
95.2% CPU use with 8 MPI tasks x 1 OpenMP threads
```

```
MPI task timing breakdown:
```

Section	min time	avg time	max time	%varavg	%total
Pair	13.187	13.462	14.183	9.7	74.14
Neigh	0.4373	0.45022	0.49764	2.9	2.48
Comm	2.943	3.659	3.967	19.4	20.15
Output	9.6792e-05	0.00010524	0.00013584	0.0	0.00
Modify	0.27889	0.28283	0.28729	0.6	1.56
Other		0.3033			1.67