Compiling LAMMPS
because sometimes you just have to

Chris MacDermaid
26-March-2014
First things first

• If you’re using the lammps-ictp virtual machine:
  
  • Increase the VM processor-count to 4
  
  • you’ll need to install fftw, openmpi and libjpeg
    
    su -c ‘yum install fftw-static openmpi-devel libjpeg-turbo-static’
  
  • LAMMPS source: ~ictp/Downloads/lammps-vanilla

• If you’re using your own machine, you’ll need to download LAMMPS, and if you want to compile with fftw/openmpi enabled binaries, you’ll need to install those packages as well. You’ll probably need to install gcc/g++ as well.
LAMMPS Directory Structure

ictp@localhost~/Downloads/lammps-vanilla
├── bench
├── doc
├── examples
├── lib
│   └── reax
├── potentials
├── python
├── regress
├── src
│   ├── MAKE
│   │   ├── MOLECULE
│   │   └── USER-OMP
│   └── tools
Top-level Make

ictp@localhost ~/Downloads/lammps-vanilla/src $ make

make clean-all     delete all object files
make clean-machine delete object files for one machine
make purge         purge obsolete copies of package sources
make tar           create lmp_src.tar.gz of src dir and packages
make makelib       create Makefile.lib for static library build
make makeshlib     create Makefile.shlib for shared library build
make makelist      create Makefile.list used by old makes
make -f Makefile.lib machine build LAMMPS as static library for machine
make -f Makefile.shlib machine build LAMMPS as shared library for machine
make -f Makefile.list machine build LAMMPS from explicit list of files
make stubs         build dummy MPI library in STUBS
make install-python install LAMMPS wrapper in Python

...
Package Installation

make yes-user-omp
make yes-kspace

Standard packages: asphere body class2 colloid dipole fld gpu granular kim kspace manybody mc
meam misc molecule mpiio opt peri poems reax replica rigid shock srd voronoi xtc

User-contributed packages: user-atc user-awpmd user-cg-cmm user-colvars user-cuda user-eff
user-lb user-misc user-omp user-molfile user-phonon user-reaxc user-sph

make package list available packages
make package-status (ps) status of all packages
make yes-package install a single package in src dir
make no-package remove a single package from src dir
make yes-all install all packages in src dir
make no-all remove all packages from src dir
make yes-standard install all standard packages
make no-standard remove all standard packages
make yes-user install all user packages
make no-user remove all user packages
make no-lib remove all packages with external libs
Machine

/src/MAKE/Makefile.machine

src> make -j4 serial

# altix = SGI Altix, Intel icc, MPI, FFTs from SGI SCSL library
# bgl = LLNL Blue Gene Light machine, xlC, native MPI, FFTW
# chama = Intel sandybridge with dual socket/eight core nodes, mpic++, openmpi, no FFTW
# cygwin = Windows Cygwin, mpicxx, MPICH, FFTW
# encanto = NM cluster with dual quad-core Xeons, mpicxx, native MPI, FFTW
# fink = Mac OS-X w/ fink libraries, c++, no MPI, FFTW 2.1.5
# g++ = RedHat Linux box, g++4, MPICH2, FFTW
# g++3 = RedHat Linux box, g++ (v3), MPICH2, FFTW
# openmpi = Fedora Core 6, mpic++, OpenMPI-1.1, FFTW2
# serial = RedHat Linux box, g++4, no MPI, no FFTs
# xe6 = Cray XE6, Cray CC, native MPI, FFTW
# xt3 = PSC BigBen Cray XT3, CC, native MPI, FFTW
# xt5 = Cray XT5, Cray CC, native MPI, FFTW
The naive first attempt:

- `make yes-all`
  `make -j4 machine`

- What you’ve just done: Installed **ALL** packages, some requiring libraries that need to be compiled **before** building LAMMPS.

- What can I do? Install only the packages you need, and compile the necessary libraries under `lib` **before** building.
Serial

• The serial version uses mpi stubs (dummy MPI library) and kiss-fft (included)

• You’ll need to compile the mpi stubs first
  `src> make stubs`

• `src> make -j4 serial`

• run lj.melt benchmark
  `bench> ../src/lmp_serial < in.lj`

• **Note the loop time**
Cray XC30 Sandybridge based compute node. 2x8-core.

../src/lmp_serial < in.lj

1 task - 1 thread

Cray XC30 Sandybridge based compute node. 2x8-core.
Anatomy of Makefile.serial

# serial = RedHat Linux box, g++4, no MPI, no FFTs

SHELL = /bin/sh

# compiler/linker settings
# specify flags and libraries needed for your compiler

CC = g++
CCFLAGS = -O
SHFLAGS = -fPIC
DEPFLAGS = -M

LINK = g++
LINKFLAGS = -O
LIB =
SIZE = size

ARCHIVE = ar
ARFLAGS = -rc
SHLIBFLAGS = -shared

Description

- src/MAKE/Makefile.serial
Anatomy of Makefile.serial

# LAMMPS-specific settings
# specify settings for LAMMPS features you will use
# if you change any -D setting, do full re-compile after "make clean"

# LAMMPS ifdef settings, OPTIONAL
# see possible settings in doc/Section_start.html#2_2 (step 4)

LMP_INC = -DLAMMPS_GZIP

# MPI library, REQUIRED
# see discussion in doc/Section_start.html#2_2 (step 5)
# can point to dummy MPI library in src/STUBS as in Makefile.serial
# INC = path for mpi.h, MPI compiler settings
# PATH = path for MPI library
# LIB = name of MPI library

MPI_INC = -I../STUBS
MPI_PATH = -L../STUBS
MPI_LIB = -lmpi_stubs

Preprocessor Defines

• src/MAKE/Makefile.serial
Anatomy of Makefile.serial

FTTs

• src/MAKE/

Makefile.serial

Image Processing

• src/MAKE/

JPEGs

• src/MAKE/

PNGs

Anatomy of Makefile.serial

FTTs

• src/MAKE/

Makefile.serial

Image Processing
Serial: Optimized

• edit src/MAKE/Makefile.serial

• Adjust compiler flags to include additional optimizations:
  CCFLAGS = -O2 -funroll-loops -fno-exceptions -fno-rtti

• Recompile serial binary
  src> make clean-serial && make -j4 serial

• Rerun in.lj benchmark
  bench> ../src/lmp_serial < in.lj
1 task - 1 thread

..src/lmp_serial < in.lj
External Libraries

• To write out JPEG and PNG format files, you must build LAMMPS with support for the corresponding JPEG or PNG library. Note you must have the libjpeg/libpng library installed. To convert images into movies, LAMMPS has to be compiled with the -DLAMMPS_FFMPEG flag.

```bash
LMP_INC = -DLAMMPS_GZIP -DLAMMPS_JPEG

...#
# JPEG and/or PNG library, OPTIONAL
# see discussion in doc/Section_start.html#2_2 (step 7)
# only needed if -DLAMMPS_JPEG or -DLAMMPS_PNG listed with LMP_INC
# INC = path(s) for jpeglib.h and/or png.h
# PATH = path(s) for JPEG library and/or PNG library
# LIB = name(s) of JPEG library and/or PNG library

JPG_INC =
JPG_PATH =
JPG_LIB = -ljpeg
```
Packaged Libraries

• If you want to include a packaged library, you **MUST** compile that library first before compiling LAMMPS.

• `lib/reax> make -f Makefile.gfortran`

• `src> make yes-reax`

• `src> make -j4 serial`
Packaged Libraries

- In each library folder, you’ll find a “Makefile.lammps” file which defines the additional flags required by the compiler and linker. LAMMPS automatically includes this file when you install the corresponding package.

- Examine the “src/Makefile.package” and “src/Makefile.package.settings” files. Examine them after you’ve added or removed the reax package, for example.
Parallel: OpenMP

- Axel has implemented openmp parallelism (share everything) into LAMMPS. It's available as the ‘user-omp’ package.

- a) Make a new low-level makefile by copying Makefile.serial to Makefile.omp (cp Makefile.serial Makefile.omp)

- b) Change your compiler and linker flags to include openmp (edit src/MAKE/Makefile.omp). Change the comment at the top to reflect your new makefile.
  
  #omp = Fedora20, g++4.8, MPI-stubs, KISS-FFT
  CC = g++ -fopenmp
  LINK = g++ -fopenmp

- c) Install the openmp package
  
  src> make yes-user-omp

- src> make -j4 omp
Running with OpenMP

• To use your OpenMP enabled LAMMPS binary, do the following:

• To run on N processors:
  `OMP_NUM_THREADS=N lmp_omp -sf omp < in.file`

• Try running the melt example with different numbers of processors
  `bench> OMP_NUM_THREADS=2 ../src/lmp_omp -sf omp < in.lj`
  `bench> OMP_NUM_THREADS=4 ../src/lmp_omp -sf omp < in.lj`
OMP_NUM_THREADS=4 ..../src/lmp_omp -sf omp

1 task - 4 threads
OMP_NUM_THREADS=16 ../src/lmp_omp -sf omp

1 task - 16 threads
Biological Systems

• If you want to simulate biological systems you’ll need a few additional packages: rigid (for fix shake), molecule and kspace:

• src> make yes-molecule yes-kspace yes-rigid

• src> make clean-omp && make -j4 omp

• bench> OMP_NUM_THREADS=4 ../src/lmp_omp -sf omp < in.lj
bench> OMP_NUM_THREADS=4 ../src/lmp_omp -sf omp < in.rhodo
Rather than use the built-in kiss-fftw library, most prefer to use the FFTW3 library. In the case of our VM, it’s as easy as specifying a preprocessor directive and the library:

```
# FFT library, OPTIONAL
# see discussion in doc/Section_start.html#2_2 (step 6)
# can be left blank to use provided KISS FFT library
# INC = -DFFT setting, e.g. -DFFT_FFTW, FFT compiler settings
# PATH = path for FFT library
# LIB = name of FFT library

FFT_INC = -DFFT_FFTW3 -DFFT_SINGLE
FFT_PATH =
FFT_LIB = -lfftw3f
```

Makefile.omp
FFTWF3

• Edit src/Make/Makefile.omp to include fftw3

• src> make clean-omp && make -j4 omp

• bench> OMP_NUM_THREADS=4 ../src/lmp_omp -sf omp < in.lj
bench> OMP_NUM_THREADS=4 ../src/lmp_omp -sf omp < in.rhodo

• **Compare the loop times and FFT times, what do you notice?**
Parallel: OpenMPI

• LAMMPS’ primary inter-process communication method is openMPI (share nothing). If you want to run in parallel across multiple machines, you must use openMPI.

CC = mpic++
...
LINK = mpic++

# MPI library, REQUIRED
# see discussion in doc/Section_start.html#2_2 (step 5)
...

MPI_INC =
MPI_PATH =
MPI_LIB = -lmpi

> mpicc --showme
gcc -I/usr/include/openmpi-x86_64 -pthread -m64 -L/usr/lib64/openmpi/lib -lmpi
Parallel: OpenMPI

- edit src/MAKE/Makefile.openmpi
  Add support for fftw3f

- Load openmpi environment variables
  module load mpi/openmpi

- src> make no-user-omp
  src> make -j4 openmpi

- To run on N processors:
  mpiexec -np N lmp_openmpi < in.file

- Try running the melt example with different numbers of processors
  bench> mpiexec -np 2 ../src/lmp_openmpi < in.rhodo
  bench> mpiexec -np 4 ../src/lmp_openmpi < in.rhodo
8 MPI tasks per socket = 16 MPI Tasks per node

mpiexec -np 16 ../src/lmp_openmpi
```bash
mpiexec -np 64 ../src/lmp_openmpi
```
Parallel: OpenMP-OpenMPI Hybrid

- You can run LAMMPS as an OpenMP-OpenMPI hybrid, where each MPI process controls a specified number of OpenMP threads. Construction of a Makefile is straightforward:

  - a) Make a new low-level makefile by copying Makefile.openmpi to Makefile.openmpi-omp (cp Makefile.omp Makefile.openmpi-omp)

  - b) Change your compiler and linker flags to include openmp (edit src/MAKE/Makefile.openmpi-omp). Change the comment at the top to reflect your new makefile.
    
    ```
    #openmpi-omp = Fedora20, g++4.8, OpenMPI-1.7.3, FFTW3
    CC =    mpic++ -fopenmp
    LINK =  mpic++ -fopenmp
    ```

  - c) Install the openmp package
    ```
    src> make yes-user-omp
    ```

- src> make -j4 openmpi-omp
Running with OpenMP-OpenMPI Hybrid

• To use your OpenMPI-OpenMP enabled LAMMPS binary, do the following:

• To run N MPI tasks with M OpenMP threads:
  \[ \text{OMP\_NUM\_THREADS}=M \text{ mpiexec -np N lmp\_openmpi-omp -sf omp < in.file} \]

• Try running the melt and rhodo examples with different combinations of MPI tasks and omp threads
  \[ \text{bench}\text{>} \text{OMP\_NUM\_THREADS}=1 \text{ mpiexec -np 4../src/lmp\_openmpi-omp -sf omp < in.lj} \]
  \[ \text{bench}\text{>} \text{OMP\_NUM\_THREADS}=2 \text{ mpiexec -np 2../src/lmp\_openmpi-omp -sf omp < in.lj} \]
  \[ \text{bench}\text{>} \text{OMP\_NUM\_THREADS}=4 \text{ mpiexec -np 1../src/lmp\_openmpi-omp -sf omp < in.lj} \]
OMP_NUM_THREADS=8 mpiexec -np 2 ../src/lmp_openmpi-omp -sf omp

1 MPI task per socket, 8 threads each
OMP_NUM_THREADS=4 mpiexec -np 4 ..src/lmp_openmpi-omp -sf omp

Core 2 MPI tasks per socket, 4 threads each
OMP_NUM_THREADS=8 mpiexec -npersocket 1 ../src/lmp_openmpi-omp
OpenMP-OpenMPI Hybrid Considerations

• The best parallel efficiency from omp styles is typically achieved when there is at least one MPI task per physical processor, i.e. socket or die.

• Using multi-threading is most effective under the following circumstances:

  • Individual compute nodes have a significant number of CPU cores but the CPU itself has limited memory bandwidth

  • The interconnect used for MPI communication is not able to provide sufficient bandwidth for a large number of MPI tasks per node.

http://lammps.sandia.gov/doc/Section_accelerate.html#acc_5
OpenMP-OpenMPI Hybrid Considerations

• The input is a system that has an inhomogeneous particle density which cannot be mapped well to the domain decomposition scheme that LAMMPS employs.

• Finally, multi-threaded styles can improve performance when running LAMMPS in "capability mode", i.e. near the point where the MPI parallelism scales out.

http://lammps.sandia.gov/doc/Section_accelerate.html#acc_5