LAMMPS – Installation

Wojciech Rosiński
student of 3rd year of Materials Engineering at Warsaw Technical University
rosinskiws@gmail.com

This work has been performed in July 2016, at National Center of Nuclear Research, Materials Laboratory, Otwock-Świerk, Poland, as a part of student's professional practice.

Content

LAMMPS Overview

LAMMPS is basically code/program which enables us to conduct molecular dynamics simulations using Classical Mechanics mathematical formulations. It is distributed by Sandia National Laboratories (http://www.sandia.gov/), and what's more important, it is an open source code. Name is the acronym for Large-scale Atomic/Molecular Massively Parallel Simulator, which pretty much describes it all.

Let's begin by giving a concise overview of Molecular Dynamics. What is it? We can say that this is a method of computer simulation enabling us to see and study physical movements of atoms and bigger molecules, which are allowed to interact between themselves for a fixed period of time, so evolution of the system over time can be seen – we'll see behavior of molecules under conditions chosen for our simulation. LAMMPS is based on Classical Mechanics theory, so at lower level it integrates Newton's equations of motion of collection of molecules, of which we have built our model for exact simulation. Those particles will interact via short-range or long-range forces. Certain initial and boundary conditions can be provided for our simulation, which will govern motion of molecules.

In LAMMPS we can perform calculations on systems either with a few particles or billions of them. This provides us with a possibility of accurately reproducing models obtained with Quantum mechanical computations and then scaling them up, so we can gain a lot of information about predicted behavior of those systems in mesoscale.

In order to conduct a simulation we need information on initial positions and velocities of atoms and an interatomic potential. This potential is a function which describes terms by which particles in our model will interact. It calculates potential energy of a system with given initial positions. Having set both parameters of the particles (positions, velocities) and potential function, we can run a simulation. There are many types of those, but I'll come back to this later. Based on the function, interactions will be simulated and we'll be able to see effects of those interactions after a fixed time step, being set in the input file to LAMMPS.

Now, it would be appropriate to say, why exactly LAMMPS. And there are quite a few reasons for that. We should begin with the most obvious one – computation time. When we would like to perform a simulation for a bigger set of particles, for a few thousand or more timesteps, it can easily take a few hours, few days or even a few weeks. Therefore, scalability is very important. And LAMMPS is perfect for this – it has been designed for parallel computers, so with a few commands, using MPI libraries, you can scale the problem to a few or a few dozens of cores and reduce computation time significantly. There are also libraries enabling user to use CUDA or GPU acceleration (benchmarks: http://lammps.sandia.gov/bench.html).

Another advantage of LAMMPS is the fact that it can be used for simulations of different types of materials, not only for problems connected directly with Materials Science, i.e. indentation simulation, but also Biophysics or Chemistry ones (some of example problems can be checked in examples directory of your LAMMPS build – described below). There are many styles of certain parameters to choose from and a wide range of potentials covered, so there is high probability that you will be able to find those you need for your model. You may create dump files of your simulations, configure them in many ways, so that you'll be able to process and/or visualize the output in a way that is much easier to understand.

Because of the fact that LAMMPS is open source, it is being actively extended not only by it's developers but also by community. The community is very important, as sometimes it may be so that you'll encounter a truly difficult problem on your way, have no idea how to solve it and that's when the community may extend a helping hand. But first, always use search function and google your specific problems, as there's a possibility that they have been encountered and answered before!
In terms of basics and less difficult problems, it's a good idea to check out the documentation – http://lammps.sandia.gov/doc/Manual.html. There is a lot of information on over 1200 pages.

More reading:
http://lammps.sandia.gov/doc/Section_intro.html
http://lammps.sandia.gov/workshops/Aug15/PDF/tutorial_Thompson1.pdf
http://lammps.sandia.gov/tutorials/sor13/SoR_01-Overview_of_MD.pdf
https://en.wikipedia.org/wiki/Molecular_dynamics

Installing LAMMPS. Overall plan.

I. Choose packages to install
II. Make LAMMPS either normally or as shared library
III. Connect with Python wrapper

1. Downloading LAMMPS

First, we have to download the LAMMPS Molecular Dynamics Simulator. We can do this using their official site: http://lammps.sandia.gov/. There are a few ways to obtain the software. This instruction will be based on the source tarball version. It can be downloaded from: http://lammps.sandia.gov/download.html. We choose the version we want – I chose the stable one, from 14th May 2016.

Afterwards we open up terminal and write: `cd Downloads` in order to move to our Downloads directory (if you have different folder for default downloads location, you should move to this folder), where the program should be saved. Then we can use the `ls -al` command to see what the directory contains.
There should be a file named `lammps-stable.tar.gz`.

Being in the directory with the LAMMPS file, we use followind commands:

```
gunzip lammps*.tar.gz
tar xvf lammps*.tar
```

This way the directory is being unzipped and then unpacked from tarball file, so a `lammps-14May16` folder is obtained, in which all the LAMMPS files can be found.
Directories contained in the folder:

README	text file
LICENSE	the GNU General Public License (GPL)
bench	benchmark problems
doc	documentation
examples	simple test problems
potentials	embedded atom method (EAM) potential files
src	source files
tools	pre- and post-processing tools

2. Making LAMMPS

After downloading and unpacking LAMMPS files, we have to build it for our machines. First we have to move to our `lammps-14May16/src` folder using the `cd lammps-14May16/src` command. Then we execute the make command, which shows us list of default builds for different machines.
I'm using Kubuntu 16.04, so from those machines, I'll chose one made for my environment, which will be: `ubuntu = Ubuntu Linux box, g++, openmpi, FFTW3.`

Here we have listed libraries, of which build for our machine is made. But first, and this is very important, we need to install specific packages on our system, so the machine can be built properly. Needed packages can be found after we move to `lammps-14May16/src/MAKE/MACHINES` directory. There are Makefile files listed for different builds. Next step is to open the file for our machine (in my case it'll be `Makefile.ubuntu`). I use Midnight Commander, so I'll open it from the terminal using `mc` command,then select wanted file and open it using F3 button. Now, in the second line of the file, there'll be information about needed packages, in my case:

```
# you have to install the packages g++, mpi-default-bin,
# mpi-default-dev, libfftw3-dev, libjpeg-dev and libpng12-dev
# to compile LAMMPS with this makefile
```

Therefore I'll need to install needed packages and I'll do this using the terminal with `sudo apt-get install x` (where x is the name of package, eg. `sudo apt-get install g++`). It is likely that some other packages, mostly some development libraries, will need to be installed as well, depending on your initial Linux installation.

Having installed everything, we can then make the proper machine using `make x` command (where x is the name of the build, eg. `make ubuntu` in my case). The process can also be run in parallel, which should significantly speed it up, when there is possibility to make use of multiple cores. We do this by using this command: `make ubuntu -j x` (where x will stand for number of cores).

LAMMPS will start to build the machine on our system. If everything goes fine, after a few minutes if no error will occur, we'll have it ready for use – a new file lmp_x will be made, lmp_ubuntu for me.

In order to check the build, we can run one of example simulations, for example indentation. We copy the lmp_ubuntu file to the `lammps-14May16/examples/indent` directory and run a simple simulation. I'll use the mpirun to do this in parallel:
`mpirun -np 4 lmp_ubuntu -in in.indent`.
We'll get log shown in the console and if the making was successful, the log will be finished with "Total wall time:" line.

3. Extending LAMMPS with packages

There are specific packages which enable us to use many various features. Overview of the packages can be found here:
http://lammps.sandia.gov/doc/Section_packages.html.
If we would like to see list of all available packages, we should use the `make package` command in the `lammps-14May16/src/` directory.

Using packages is possible only if we include them before building LAMMPS. Therefore we are forced to make it once again, now including packages we'd like to use.

Command for inclusion if very simple, we just have to write `make yes-x` (where x will be package name, such as `make yes-asphere`) in the `lammps-14May16/src/` directory. I'll include the following packages:

make yes-asphere
make yes-body
make yes-class2
make yes-colloid
make yes-compress
make yes-coreshell
make yes-dipole
make yes-fld
make yes-granular
make yes-kspace
make yes-manybody
make yes-mc
make yes-misc
make yes-molecule
make yes-mpiio
make yes-peri
make yes-qeq
make yes-replica
make yes-rigid
make yes-shock
make yes-snap
make yes-srd
make yes-xtc
make yes-python
make yes-kim
make yes-meam
make yes-reax
make yes-poems
make yes-voronoi

These are official packages, which are supported by LAMMPS developers.
Packages which are written in bold caused problems during installation, as they need another portion of auxiliary libraries. These were deleted as of now, and I'll add further description on their installation some time from now.

We'll also include some of the user-contributed packages, such as:

make yes-user-atc
make yes-user-awpmd
make yes-user-cg-cmm
make yes-user-colvars
make yes-user-cuda
make yes-user-diffraction
make yes-user-dpd
make yes-user-drude
make yes-user-eff
make yes-user-fep
make yes-user-qtb
make yes-user-h5md
make yes-user-intel
make yes-user-lb
make yes-user-mgpt
make yes-user-misc
make yes-user-molfile
make yes-user-omp
make yes-user-phonon
make yes-user-qmmm
make yes-user-quip
make yes-user-reaxc
make yes-user-smd
make yes-user-smtbq
make yes-user-sph
make yes-user-tally

Now we'll make LAMMPS again and see if everything works. Making LAMMPS with all the packages without those which are in bold was succesful (In shared library mode).

4. Extending LAMMPS with Python wrapper

I have also included package Python, as I will want to use the Python-LAMMPS interface. In order to be able to do that, LAMMPS will have to be made as shared library and the package Python must also be installed.

In order to make LAMMPS as shared library now, we have to use the command: `make ubuntu mode=shlib`.

It's also important to use the `make install-python` command from the src directory, sometimes you'll be forced to use the SuperUser mode, adding `sudo` in front of the command.

In case of problems with building of the shared library in connection with Python, you may have to install additional package `python-dev`. Without it, it's possible that you'll encounter errors saying that no file for Python config has been found during making of the shared library.

You should also make sure that Python wrapper will be able to find 2 files:

- python/lammps.py
- src/liblammps.so

In my case, `sudo make install-python` from src directory did the proper work and connected the wrapper to my LAMMPS build.

Afterwards, to make sure that I can use LAMMPS from Python directly, I copied my machine file: lmp_ubuntu to the /lammps-14May16/examples/indent directory, from there I invoked Python from command line using `python` and did the check:

```
Python 2.7.12 (default, Jul  1 2016, 15:12:24)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from lammps import lammps
>>> lmp = lammps()
LAMMPS (14 May 2016)
>>> lmp.file("in.indent")
```

Make sure you use Python 2.7, as the LAMMPS module will not be imported into Python 3.5.

In next step, we can add the possibility to run LAMMPS in parallel from Python by downloading either `pypar` (installation instructions for pypar can be found in the LAMMPS documentation) or `mpi4py` Python packages. I will use mpi4py, so the instruction will be based on that one.

For this to be possible, we need to install mpi4py, this can be done using pip.
In order to get it first, we use `sudo apt-get install python-pip`
in the console. After pip is installed, we can download mpi4py using `pip install mpi4py`.

When it installs successfully, we run the indentation simulation once again, this time from Python. I have used code from LAMMPS documentation (section 11.5), only

changing the input file to `in.indent`.

```
from mpi4py import MPI
from lammps import lammps
lmp = lammps()
lmp.file("in.indent")
me = MPI.COMM_WORLD.Get_rank()
nprocs = MPI.COMM_WORLD.Get_size()
print "Proc %d out of %d procs has" % (me,nprocs),lmp
MPI.Finalize()
```

If you'd like to quickly build a machine with shared library, wrapped with Python and specific packages for your environment, you can do this making following steps:

- Specify which packages you'd like to have implemented in your build and install them with make x command in the src directory (where x stands for package name).
- Build you machine with make x mode=shlib -j y (where -j y uses multiple cores, y is the number of cores to be used during installation – described in the section).
- Use sudo make install-python in the src directory, this will connect Python wrapper to your LAMMPS interface.
- Install mpi4py or pypar for parallel Python processing using pip: pip install mpi4py for your default Python enviroment.

Just in case, try initializing a simulation, this will enable you to check if everything was installed properly. In order to verify everything at once, we'll run a LAMMPS simulation using parallel processing from Python:

Move to indent directory: `cd /lammps-14May16/examples/indent`

Then write `python` to run python from command line. For the simulation, either save the program to file, for example test.py and run it or just paste following lines into Python:

```
from mpi4py import MPI
from lammps import lammps
lmp = lammps()
lmp.file("in.indent")
me = MPI.COMM_WORLD.Get_rank()
nprocs = MPI.COMM_WORLD.Get_size()
print "Proc %d out of %d procs has" % (me,nprocs),lmp
MPI.Finalize()
```

If successful, you'll get a log in your console with all the information about the simulation itself.

Installation of SPPARKS

Installing SPPARKS on your machine is going to be pretty similar to the process of making LAMMPS. If we'd like to be able to use SPPARKS package within our LAMMPS build, first SPPARKS must be build and LAMMPS afterwards.

1. Downloading SPPARKS

SPPARKS can be downloaded from: http://www.sandia.gov/~sjplimp/download.html. You choose it from available packages and download. Then, a zipped tar file will appear in your default downloads folder. You'll have to unzip it and create a directory from tar file, which you can do with following commands in this download folder:

```
gunzip file.tar.gz
```

```
tar xvf file.tar
```

Afterwards, a directory containing SPPARKS files will be created. In my case it's called spparks-26Feb16, as it's a stable version from 26/02/2016. There will be a src folder, to which we'll navigate.

2. Making SPPARKS

In order to build a machine for your system and be able to connect it with LAMMPS, it should be built on libraries corresponding to those we used to build LAMMPS. In certain cases, you'll be able to use pre-built version of SPPARKS if there's one ready for your enviroment.

You can list those pre-built versions using `make` in `spparks-26Feb16/src`.

There isn't one made for Ubuntu, so another Makefile must be compiled. Mr. Kozioł has compiled one for SPPARKS, which enables us to build it for our Ubuntu systems.
`<code for the file>`

It has to be placed in the `src/MAKE folder`. If the build has been compiled correctly, after using `make` command in the `src` folder once again, a new machine to be made will appear.

Now, we can use `make x` command to build a machine we want (where x stands for machine name).

I'll be willing to connect SPPARKS with Python wrapper, so I'd like to make it as a shared library, so that Python will be able to invoke SPPARKS. In this case I'll use following commands in the `src` folder:

```
make makeshlib
```

```
make x mode=shlib
```

where `x` will stand for machine name.

If the process ran correctly and no errors occurred a file libspparks_x.so will be created in the src directory and libspparks.so file, which will be loaded by default by Python wrapper.

3. Installing Python wrapper

Python must know about two files in order to be able to invoke SPPARKS package:

python/spparks.py – Python wrapper on the SPPARKS interface

src/libspparks.so – shared SPPARKS library

After having build the shared library we go into `spparks-26Feb16/python` folder and use the `python install.py` command, which will create links between SPPARKS and Python. It's possible that we'll have to use the sudo mode.

SPPARKS can be run from Python using parallel processing. This addition can be installed in a process analogical to the one used with LAMMPS installation, which I've discussed in section 4. of LAMMPS part.

When it's also installed we invoke python from command line using python and check if it's wrapped using following commands:

```
from spparks import spparks
>>> spk = spparks()
```

If it has been installed properly, you should see following output:

```
Python 2.7.12 (default, Jul  1 2016, 15:12:24)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>> from spparks import spparks
>>> spk = spparks()
SPPARKS (26 Feb 2016)
```

If we'd like to check proper installation, we can run a simulation. I'll run one from examples/erbium folder. Here, I'll create testspk.py file:

```
from mpi4py import MPI

from spparks  import spparks

spk = spparks()

spk.file("in.erbium")

me = MPI.COMM_WORLD.Get_rank()
```

```
nprocs = MPI.COMM_WORLD.Get_size()

print "Proc %d out of %d procs has" % (me,nprocs),lmp

MPI.Finalize()
```

Using LAMMPS with CUDA support through KOKKOS packages

A) Installation of Nvidia drivers

We will start with installation of proper drivers for our GPU. To do this we have to start with adding Nvidia repository to our system with:

```
sudo add-apt-repository ppa:graphics-drivers/ppa
```

afterwards, we'll update: `sudo apt-get update` and install the drivers: `sudo apt-get install nvidia-x )`. You'll be asked if you'd like to install the driver – confirm this. A reboot will be needed to load the drivers, which can be done through terminal with: `sudo reboot.`

On Ubuntu 16.04 (the one i'm using), some of you may encounter errors such as system freeze or black screen, in order to solve those problems, look at those articles:

http://askubuntu.com/questions/691946/how-to-solve-black-screen-problem-after-installing-nvidia-drivers-on-ubuntu-15-1

https://elementaryforums.com/index.php?threads/howto-install-latest-nvidia-driver-on-linux-without-getting-black-screen.7/.

I've encountered this problem – my system freezed and I was unable to even start the terminal, following those instructions and using recovery mode made it possible to solve the issue and now the GPU runs properly.

B) Prerequisites for CUDA

Then, we have to check if we have all the needed prerequisites for CUDA:

http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-linux/#axzz4EZfBESUF. This proess. has been described in detail here.

C) Installation of CUDA

CUDA can be downloaded from here: https://developer.nvidia.com/cuda-toolkit. Two versions are available as of now (16.07.2016) – Toolkit 7.5 and Toolkit 8 RC, I'll chose 8 RC of those two, as I'm using GTX1080 and only this version supports Pascal Architecture. When a version is chosen, you'll be redirected to the next page, where you'll have to pick your environment, in my case it'll be: Linux > x86_64 > Ubuntu > 16.04 > runfile(local). A download will start, it's quite big (1,3GB).

There're also Installation Instructions on how to run the runfile, when the download is finished.

Installation Instructions:

1. Run `sudo sh cuda_8.0.27_linux-run`
2. Follow the command-line prompt

https://developer.nvidia.com/cuda-release-candidate-download

It'll also be a good idea to verify if it has downloaded without any problems with MD5 checksum. To do this, we head into the directory, where CUDA has been downloaded and from terminal use: `md5sum x` (where x will stand for CUDA file name).

After checking the pre-installation actions and downloading CUDA toolkit, we should disable Nouveau graphics drivers. This can be done with creation of `blacklist-nouveau.conf` file in modprobe.d directory, which we can accessed using `cd /etc/modprode.d directory`. Then, we'll have to create the conf file with text editor and super user privilege, for this I'll use Kate editor: `sudo kate blacklist-nouveau.conf` in terminal. We put:

```
blacklist nouveau

options nouveau modeset=0
```

in our `blacklist-nouveau.conf` file and save it. Afterwards, regeneration of kernel initrd is needed: `sudo update-initramfs -u`.

Then we'll need to reboot and run in text mode – CUDA must be installed when graphics drivers aren't being used. We can also do so using `sudo service sddm stop`, so our GUI will be disabled. Enabling it again is possible if we replace "`stop`" with "`start`" at the end of command. Running in pure text mode can be done if we edit our system's kernel boot parameters. We can do so following this tutorial: http://ubuntuhandbook.org/index.php/2014/01/boot-into-text-console-ubuntu-linux-14-04/ .

Ubuntu 16.04 isn't supported yet, so we'll have to use some additional tricks to get it installed on our system. Most of them will be based on: https://www.pugetsystems.com/labs/hpc/NVIDIA-CUDA-with-Ubuntu-16-04-beta-on-a-laptop-if-you-just-cannot-wait-775/ . This should enable us to get CUDA working. As we're going to install 8 RC version, Pascal architecture is officially supported now.

After you download the CUDA driver, head into directory, where the .run file is placed. Then, from terminal run: `chmod +x some-app.run –` this will add an option for the file to be executed. Afterwards, let's stop our graphics services with `sudo service sddm stop`. Now, we navigate to the directory, where executable CUDA toolkit file is placed. Then, we run it with: `sudo ./cuda* --override`, adding `–override` will enable us to install it properly, otherwise an error will be thrown, indicating that our version of gcc compiler isn't supported. During installation, a few questions will be asked – I chose default Toolkit location and changed Samples location to /usr/local/cuda-8.0/Samples. I confirmed creation of symbolic link. When the installation will finish, we have to set our enviroment for CUDA.

I used Kate editor for files creation:

- `sudo kate /etc/profile.d/cuda.sh`

and in the file: `export PATH=$PATH:/usr/local/cuda/bin`

- `sudo kate /etc/ld.so.conf.d/cuda.conf`

  and in the file: `/usr/local/cuda/lib64`

After creating those, those two commands should also be executed:

```
source /etc/profile.d/cuda.sh

sudo ldconfig
```

We have gcc 5 on our Ubuntu, so we'll have to force CUDA to work with it, this can be done through edition of /usr/local/cuda/include/host_config.h file with:

```
sudo kate /usr/local/cuda/include/host_config.h
```

" // " must be added at the beginning of 115th line:

line: 115 comment out error

```
//#error -- unsupported GNU version! gcc versions later than 4.9 are not
supported!
```

Let's copy CUDA samples, which were installed during our installation of Toolkit to a new directory, which will be one for experimentation, I'll name mine CUDA and in there, I'll create Projects directory, one for experimentation. Create it wherever you want and while being in it, do:

```
rsync -av /usr/local/cuda-8.0/Samples .
```

This will copy the Samples files to the desired directory. Afterwards, let's use command: `make` in this Samples directory in order to compile the bin files.

According to Nvidia guide, the best way to check if CUDA is properly installed and configured is to run deviceQuery.

Using `cd CUDA/Projects/Samples/NVIDIA_CUDA-8.0_Samples/bin/x86_64/linux/release/` we'll find ourselves in the directory containing deviceQuery sample. Then we use `./deviceQuery` and check the output. You'll be able to see all the information about your particular GPU, just like in my case. Now we'll see if the system and the GPU can communicate correctly, this will be done with running `./bandwidthTest` from the same directory as the deviceQuery program.

D) KOKKOS Installation

Now, we'll extend our LAMMPS build with KOKKOS, which will enable us to use CUDA-provided GPU acceleration on the GTX 1080.

For more details on KOKKOS itself:
http://lammps.sandia.gov/doc/accelerate_kokkos.html .

- First, head into your lammps*/src directory with `cd lammps*/src` used while being where the LAMMPS folder is. Being there, use `make yes-kokkos`, so the package enabling to extend LAMMPS with KOKKOS will be installed.

- Next step is to create a machine specifically for CUDA library. If we use command make in the src directory, we'll see description of many potential LAMMPS builds. For cuda, there are two versions:

  ```
  # kokkos_cuda = KOKKOS/CUDA package, MPICH with nvcc compiler, Kepler GPU
  # kokkos_cuda = KOKKOS/CUDA package, OpenMPI with nvcc compiler, Kepler GPU
  ```

  I'll choose the one based on OpenMPI. In order to build it, we'll now have to use `make kokkos_cuda_openmpi -j 4` (I'll add -j 4 in order to build it on using my 4 CPU cores) command. Makefile for this one can be found in: `lammps-14May16/src/MAKE/OPTIONS/.`

EDIT2: Let's try to edit the Makefile of `kokkos_cuda_openmpi` and change the KOKKOS_ARCH to Pascal61 from Kepler35, let's see if it enables us to use high Compute Capability. An error will be thrown, this will not work.

EDIT1: Let's install KOKKOS library independently. We download the branch: https://github.com/kokkos/kokkos, unzip it, which will create kokkos-master directory. Next, we'll create one for KOKKOS, in order to test it, I'll call it KOKKOS_test.

EDIT1A: After some experimentation with new build of KOKKOS, I wasn't able to properly connect it with LAMMPS.

As of now, I haven't been able to create a shared library with KOKKOS, as there was an error connected with my installed MRO in version 3.2.5.

- After the machine has been built, let's check if it works properly with our GPU. Copy created `lmp_kokkos_cuda_openmpi` file into, let's say, indent directory (or whichever you wish to try) with `cp lmp_kokkos_cuda_openmpi ../examples/indent/.`

- An important step, when using simulations accelerated by KOKKOS package is to change the parameters in the input file. In this case, we'll have to change the `atom_style` command in `in.indent` from standard `atomic` to `atomic/kk`.

- Now, let's try the simulation. `mpirun -np 1 lmp_kokkos_cuda_openmpi -k on g 1 -in in.indent`. This means that one MPI task will be run `(-np 1)` using `lmp_kokkos_cuda_openmpi` machine with KOKKOS on `(-k on)` and 1 GPU `(g 1)`. Unfortunately, there will be a Warning: `Kokkos::Cuda::initialize WARNING: running kernels compiled for compute capability 3.5 on device with compute capability 6.1 , this will likely reduce potential performance.`

  This is because Pascal architecture, on which GTX 1080 is based, is provided with Compute Capability 6.1, according to: https://developer.nvidia.com/cuda-gpus .

  Unfortunately, it seems that the performance is much worse as opposed to simulation ran on CPU using 4 cores. I'll have to dwelve deeper in order to find the cause.

E) GPU Package Installation

Unfortunately, didn't work, maybe it's because Pascal architecture isn't supported. I tried to edit Makefiles but I keep receiving an error.

F) USER-CUDA Package

USER-CUDA package is not being maintained for quite a some time, so no new architectures will be supported. Therefore, it's unlike that it'll provide us with a significant performance acceleration.

VMD Installation

1. Downloading VMD

VMD can be downloaded from: http://www.ks.uiuc.edu/Research/vmd/. You'll have to register an account or login to already existing one, so that you may download the program. Afterwards, you'll be prompted to choose version you'd like to have. I'm using Ubuntu Linux on 64bit processor and Nvidia graphics card, so I'll choose one with 64bit Linux and CUDA support.

Downloaded fill will be placed in the default download folder. You should go to this folder using your terminal and while being in the folder write: `tar -zxvf vmd*` so the archive will be unpacked and a folder with all the files created.

2. Installing VMD

Then, go into your created VMD folder, named `vmd-x` (where x will stand for version of downloaded VMD).

When you open configure executable file with an editor, you'll be able to see a list of possible compilations. Here, you can choose a compilation you'd like to have in your machine. The next step is to execute the file using `sudo ./configure X` (where X will stand for chosen compilation, in my case `LINUXAMD64`).

When the configuration is done, which won't create any output, you should move to the vmd/src directory with `cd src` and while there, type `sudo make install` in your terminal. This will create VMD for your machine.

In order to check if everything works properly, just write `vmd` in the terminal and the program should start.

VMD can use CUDA for acceleration, so if you have CUDA-enabled GPU, it will be supported by default - no action will be needed from you in order to use CUDA if you choose version of VMD with CUDA when downloading.

3. VMD visualization from LAMMPS

It is possible to see the output of your LAMMPS programs in VMD. I have created ZrO2 structures in Python, which can be read by LAMMPS and after the calculations are done, lattices will be created. If we'd like to visualize them using VMD, a proper LAMMPS dump will be needed, i.e. xyz trajectory file.

In the input file for LAMMPS, we add line:

dump 3 all atom 1000 trajmonoclinic.xyz

which will create a dump file with coordinates of all atoms every 1000 timesteps. Information about lattice itself will also be written. Afterwards, we open VMD, using vmd command in our terminal (preferably in the directory, where our damp file has been created – this will prevent us from searching manually for this particular directory using VMD.

When VMD is opened, we click File > New Molecule… , we browse for the Filename, which should be this .xyz dump file. When it's chosen, VMD will determine file type as XYZ – this will be wrong, as the molecule won't be read properly. We open "Determine file type" list and choose "LAMMPS Trajectory". Then, we click Load and if the dump file has been written by LAMMPS correctly, a molecule will be loaded, atoms will be shown as "Lines" in default mode. If we'd like to be able to assess our structure better, we open VMD Main window, choose Graphics > Representations and from "Drawing Method" in Graphical Representations window we choose "Beads". Then, we click Apply, so this mode of visualization will be chosen.

Bibliography

http://lammps.sandia.gov/doc/Manual.html

http://spparks.sandia.gov/doc/Manual.html

http://yangcha.github.io/GTX-1080/

http://blog.thismagpie.com/2012/11/how-to-install-vmd-on-linux-ubuntu-64.html

http://askubuntu.com/questions/691946/how-to-solve-black-screen-problem-after-installing-nvidia-drivers-on-ubuntu-15-1

https://elementaryforums.com/index.php?threads/howto-install-latest-nvidia-driver-on-linux-without-getting-black-screen.7/

http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-linux/#axzz4EZfBESUF

http://ubuntuhandbook.org/index.php/2014/01/boot-into-text-console-ubuntu-linux-14-04/

https://www.pugetsystems.com/labs/hpc/NVIDIA-CUDA-with-Ubuntu-16-04-beta-on-a-laptop-if-you-just-cannot-wait-775/

http://askubuntu.com/questions/799184/how-can-i-install-cuda-on-ubuntu-16-04

http://terryum.io/ml_practice/2016/05/15/UbuntuSetup/