P. Koumoutsakos, D. Rossinelli, M. Troyer
ETH Zentrum, CAB H 69.2
CH-8092 Zürich

Unix 101

We strongly recommend that you use a Unix-based system like Linux or Mac OS X. Of course you are also allowed to use your own Windows-based machine and it should work just fine, but we can only give limited support for it. Also, the code-skeletons we provide will come with Makefiles that we will test on the student lab computers in CAB H56 and H57 and on Mac OS X. They should be easily adapted to any other machine though.

The core of any Unix-based system is its command line interface called the Terminal. We suppose that you can find relevant information on how to use it on the web, but if you never used it before, the following intro (in German) from another course might be useful:
`http://www.ti.inf.ethz.ch/ew/courses/Info1_10/unix-intro.pdf`.

The most relevant commands that you will have to use are man, ls, pwd, cd, locate and make. It will also be useful for you to know how to set environment variables. For a variable `VAR`, you can check the value it has with the command `echo $VAR`. In the shells sh, ksh and bash variables can be set using the command `export VAR="..."`, while for the shells csh and tcsh you would use `setenv VAR "..."` (the quotation marks are only needed if ... contains spaces). You can check the used shell by typing `echo $SHELL`. Compilers can usually be influenced with env. variables to define search paths for libraries (`LIBRARY_PATH`) and headers (`CPATH`) as colon-separated list of directories (just like for `PATH`).

Makefiles are a useful tool to compile programs when you want it to be portable between different Unix-based systems like your own machine and a supercomputing cluster. We will use them to provide with a simple way to compile the code we provide to you. Usually, you will only need to adapt a few variables pointing to the folders where CUDA etc are installed. For further information on makefiles, please consult the reference manual:
`https://www.gnu.org/software/make/manual/`

CUDA and OpenCL

`http://developer.nvidia.com/cuda-toolkit-40`
The webpage above includes everything you need to know or have about CUDA. The "GPU Computing SDK" is a collection of examples which you should all get and compile either for CUDA (subfolder C) or OpenCL (subfolder OpenCL). Usually, just typing `make` in one of those subfolders should do the job (binaries are put in `bin/...`). If `make` cannot find your CUDA or OpenCL files you might try to set the env. variables `LIBRARY_PATH` and `CPATH` (see above).

Besides the examples, the SDK also includes the extremely useful tools `deviceQuery` to get specs of your GPU and `bandwidthTest` to test the bandwidth.

Public Student Labs (CAB H56 or H57)

If you cannot or do not want to use your own machine to do the exercises, you are welcome to use the student labs in the CAB. They have Linux (and Windows) installed. Note that you have to restart the machine to change operating system. The Linux home folder is stored within your ETHZ home folder (see link below) in the subfolder `infk/Linux`. The ETHZ home folder on the other side can be found in `/dfs/LOGIN` (where LOGIN is your nethz login) on the Linux boxes (if you cannot find it, try executing `ls /dfs/LOGIN`). The machines have a quad core Intel i5-2400 (Sandy-Bridge, 198 SP GLOPs peak performance with AVX) with a nVidia GeForce GT 520 (1GB of RAM, 155 SP GFLOPs peak performance, compute capability 2.1). The CUDA runtime 3.2 with CUDA driver 4.0 is installed. The runtime can be found in `/usr/cuda`. The driver and OpenCL 1.0 are in `/usr/lib64/nvidia`.

How to get and compile the SDK examples?

1. Download the SDK for CUDA 3.2 (4.0 will not compile there!)
   `http://developer.nvidia.com/cuda-toolkit-32-downloads`
2. Execute the SDK installer and set the CUDA path to `/usr/cuda` (instead of `/usr`)
3. Execute `setenv LIBRARY_PATH "/usr/lib64/nvidia"` (needed for libcuda and libOpenCL)
4. Execute `make` in subfolder C (CUDA) and/or OpenCL to generate the examples

CUDA 4 includes the "CUDA-STL" library thrust. It is just a collection of headers and therefore does not require any installation. You can just download it, extract it and include it in your source folder. `http://code.google.com/p/thrust/`

Recently, the Intel C++ compiler was installed (in `/opt/intel/composerxe-2011.3.174`) which provides better performance than gcc and includes the libraries IPP, MKL and TBB to help you create high-performance parallel code. To use it you should execute
`source /usr/bin/iccvars.csh intel64`
in the terminal to get all needed environment variables. You can then use `icc` to compile your C++ code. The Makefiles we provide will automatically use appropriate settings for icc when `CC=icc` is set there.

References:
`https://www.isg.inf.ethz.ch/ServicesStudentLabsPublicLinux`
`https://www1.ethz.ch/id/services/list/comp_raum_stud/homeverzeichnis/index_EN`

Visualization using VMD

In order to visualize the moving particles, you are going to use VMD, a visualization program for displaying, animating, and analyzing large biomolecular systems using 3-D graphics. You can download VMD at `http://www.ks.uiuc.edu/Research/vmd/` or use the version installed on the ETH computers. If you are logged on remotely, enable forwarding of the graphical output by executing the following command before you launch VMD:

```
$ export LIBGL_ALWAYS_INDIRECT=yes
```

In order to launch the program, open a terminal and enter:

```
$ vmd
```

At this point, you should see the main VMD application window and the output window (See Figure 1). To load a dataset (e.g. output.xyz created by your n-body code), open the *Molecular File Browser* dialog, locate the test file on your local file system and make sure the file type is set tp to XYZ (usually done automatically, see Figure 2).

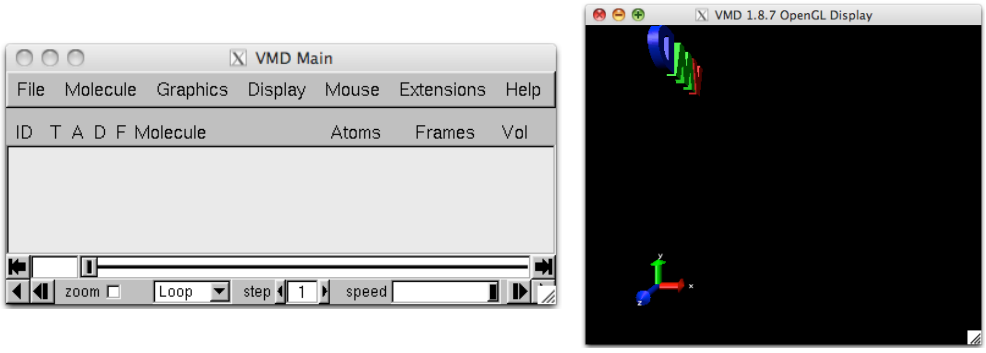

Figure 1: VMD: Main window (left) and output window (right)

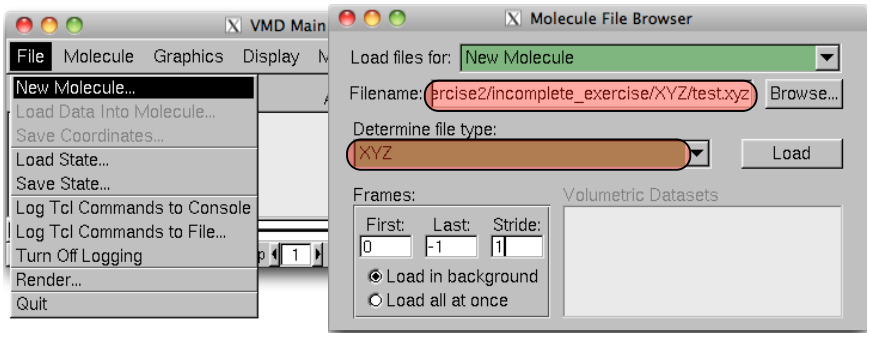

Figure 2: VMD: Loading a data set.

In order to adjust the graphical representation of the data, open the *Graphical Representation* dialog as indicated in Figure 3. Before you start changing the graphical representation, make sure to reset the timeline to its initial position on the very left (Figure 3, left). Set the *Coloring Method* of the particles to position->radial and the *Drawing Method* to VDW. To adjust the radius of the displayed spheres, play with the *Sphere Scale* parameter (Figure 3, bottom).

In the output window, you should now observe an image similar to the one displayed in Figure 4. Use your mouse wheel on the output window to zoom on the image. By clicking and dragging the image in the output window, you can change the view angle. Use the time line (Figure 3, left) to advance the simulation in time. After successfully completion of the visualization of the test dataset, repeat these steps for the simulation output data (`output.xyz`). Set the *Coloring Method* of the particles to position->X . Use the built in functionality of VMD to dump movies or images that can be combined into a movie with some external tool. Open the VMD

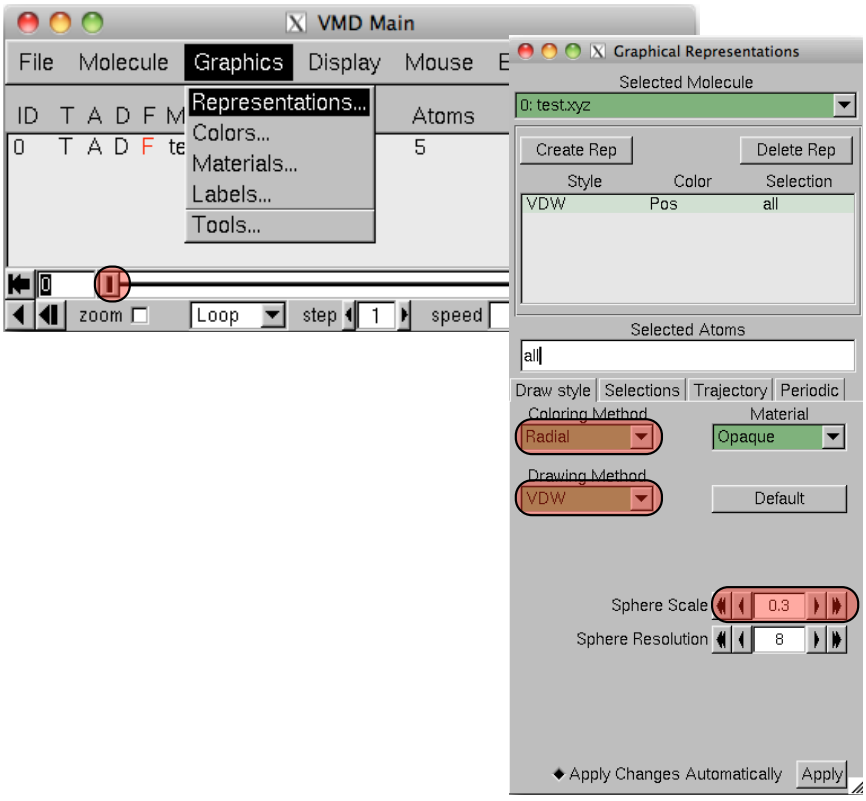

Figure 3: VMD: Main window (left) and output window (right)

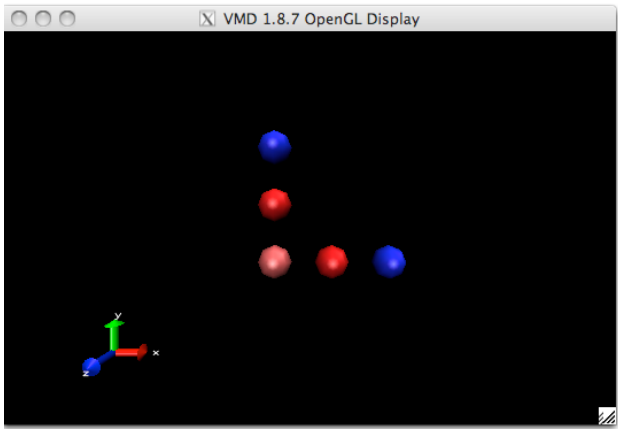

Figure 4: VMD: Main window (left) and output window (right)

Movie Generator (Extensions->Visualization->Movie Maker). In the *Movie Generator* dialog, set the *Movie Settings* to Trajectory, the *Format* to your favorite format and define the working directory and movie name. Click the *MakeMovie* Button to render the images. You might need to install external tools like ImageMagick or ffmpeg the use a specific output format.

Common issues

- Unable to find shared library when executing code.
  Set the env. variable `LD_LIBRARY_PATH` (Linux) or `DYLD_LIBRARY_PATH` (Mac) to the path of the non-found library (use `locate` to find it).