CUDA Pairlist for GROMOS XX

Matthew Breeze (mbreeze@gmail.com)

November 2009

1 Outline

GPU Acceleration has successfully been achieved both for the grid-based neighbour-list algorithm and for nonbonded Lennard-Jones interactions in the software HOOMD by Joshua A. Anderson, Chris D. Lorenz, and Alex Travesset (DOI: 10.1016/j.jcp.2008.01.047). The source code for HOOMD is released under a licence that maintains the Copyright, Disclaimer and Promotion rights of the HOOMD code with the original authors. See `hoomd-0.8.2/src/LICENSE` for more details. As such it can be used in GROMOS XX as long as the terms of the HOOMD license are included in the GROMOS XX license.

This document describes extracting the neighbour-list code from HOOMD and inserting it into GROMOS to be used alongside the existing code as an alternative through a simple additional block in the `md` program input file.

2 Extraction of Code from HOOMD

The HOOMD source code may be downloaded from http://codeblue.umich.edu/hoomd-blue/download.html. Extracted the source code is found in the directory `hoomd-0.8.2/src/`. There are unit test programs in the subdirectory `unit_tests/`. One of these (`neighborlist_test`.cc) was extracted from the source code along with the required files and compiled seperately from HOOMD. The extracted files had the Python interface removed and a function declaration put in the code occasionally of the style "extern cudaError_t cuda...;" to avoid the compiler error "no matching function for call to 'bind(...)'". This formed the basis for the HOOMD code used in GROMOSXX.

3 Insertion of Code into GROMOS XX

A directory called `HOOMD_CODE/` was created inside `contrib/` inside the GROMOS XX source directory. Inside were put the files required by the unit test (but not the unit test procedure itself, i.e. not `neighborlist_test`.cc as it is not needed). Two header files were also created inside the `HOOMD_CODE/` directory, `HOOMD_GROMOSXX_processor.h` and `HOOMD_GROMOSXX_interface.h`, the primary one being `HOOMD_GROMOSXX_interface.h`, which defines a new class HOOMD_Pairlist_Algorithm that has the same interface as the existing GROMOS XX pairlist algorithm classes but internally calls the HOOMD code. The most recent version of the development version of GROMOS XX was used (https://gromos.ethz.ch/svn/repos/gromosXXc++ revision 1329).

3.1 Classes

3.1.1 Class processor::Processor in `HOOMD_GROMOSXX_processor.h`

processor::Processor is a class in a newly-created processor namespace to represent the processor to use in the HOOMD code. The processor namespace additionally contains an new enumerated type named which that contains a number of options (CPU: CPU only; GPUs: all available CUDA-capable GPUs), to describe the processor to use. The processor::Processor constructor takes a single argument of type which and is a simple wrapper for the HOOMD class ExecutionConfiguration, which is used to represent the processor type to use in HOOMD.

3.1.2 Class interaction::HOOMD_Pairlist_Algorithm in `HOOMD_GROMOSXX_interface.h`

HOOMD_Pairlist_Algorithm is a child class of the GROMOS XX class interaction::Pairlist_Algorithm, and can be treated like a GROMOS XX pairlist algorithm. However, currently only short-range solvent-solvent pairs are generated. Internally, the class is a wrapper for four HOOMD classes, but only one is used, the choice of which being based on two criteria:

1. If the value of the processor variable in the object simulation::hoomd (set by io::In_Parameter::read_HOOMD) is simulation::cpu then a CPU-only code is used, otherwise a CUDA code is used that uses all the GPUs on the system.

2. If the value of the grid variable in the object simulation::pairlist (set by io::In_Parameter::read_PAIRLIST) is 0, then the $O(n^2)$ algorithm option is used, otherwise the $O(n)$ algorithm option is used.

The four possible classes are as follows:

NeighborList the neighbour-list base class, with CPU-only $O(n^2)$ (double-loop) algorithm

BinnedNeighbourList a child of NeighborList, with CPU-only $O(n)$ (grid-based) algorithm

NeighborListNsqGPU a child of NeighborList, with CUDA $O(n^2)$ (double-loop) algorithm

BinnedNeighbourListGPU a child of NeighborList, with CUDA $O(n)$ (grid-based) algorithm

The two interaction::Pairlist_Algorithm functions prepare and update are called sequentially each time a pairlist is updated by GROMOS XX. They are implemented in HOOMD_Pairlist_Algorithm as follows:

prepare

1. If the system box dimensions have changed, create a new ParticleData object, to specifiy the number of molecules and the box dimensions, then create a new HOOMD pairlist algorithm object (e.g. BinnedNeighbourList), setting the storage mode to half, if possible.
2. Obtain access to the coordinates storage array, and copy the first atom of each solvent molecule to the array, after centering each molecule so the first atom is inside the box.
3. Close access to the coordinates storage array.

update

1. The HOOMD NeighborList function compute is called to construct the pairlist.
2. The HOOMD functions getList is called to get access to the pairlist in a 2D vector format (same as that used by GROMOS XX). Internally this involves a transfer of the pairlist from Video RAM to RAM.
3. The pairs are read and inserted into the GROMOS XX pairlist container, as atomic pairs. Prior to insertion, if a full pairlist was generated by HOOMD, half the pairs are removed.

The CUDA algorithms can only build a full pairlist (one where reverse pairs are included), as this is the kind used on the GPU for the force calculations. As GROMOS XX uses a half pairlist ($(i,j)|i<j$), all the pairs extracted from the CUDA algorithms had to be processed by checking if the reverse pair was already in the pairlist.

Because the HOOMD pairlist algorithms do not have any functions that allow an efficient and reliable resizing of the system box, if the system box size changes a new HOOMD neighbour list algorithm object must be created each time (the ParticleData function setBox is not a reliable way to resize the system box). The box dimensions can change if pressure scaling is enabled in GROMOS XX, so this is checked during the prepare call.

HOOMD has no knowledge of chargegroups but it can potentially allow up to 16 exclusions per particle. Currently, the HOOMD_Pairlist_Algorithm interface only generates short-range solvent-solvent pairs and assumes a single solvent type.

HOOMD internally represents a buffer zone around the short-range cut-off that can be internally monitored to automatically rebuild the pairlist if particles drift beyond the cut-off. However this is unused and the buffer radius is set to 0.

3.2 Other Modifications

To access from GROMOS XX the code in `HOOMD_CODE/`, GROMOS XX was modified in a number of ways. The modifications are only enabled if HAVE_HOOMD is defined during compilation.

simulation::proc the shared pointer named proc that points to a processor::Processor object was added to the simulation namespace in `src/simulation/simulation.h`. The advantage of using a shared pointer is that the object will be deleted when the last reference to the shared pointer goes out of scope.

simulation::hoomd the enumerated type hoomd was added to the simulation namespace in `src/simulation/parameter.h` to contain possible options for using the HOOMD code. Currently just the kind of processor to use is supported.

simulation::hoomd the hoomd_struct struct was added to the simulation namespace in `src/simulation/parameter.h` to store the chosen options for using the HOOMD code. A single global object of this struct exists in the simulation namespace named hoomd. Currently just one variable of the enumerated type hoomd is held in the struct. The default choice for this variable is the unknown processor type, which has the effect that HOOMD code is not used.

io::In_Parameter::read_HOOMD the function read_HOOMD was defined in `src/io/parameter/in_parameter.h` and implemented in `src/io/parameter/in_parameter.cc` to read a HOOMD block in the `md` program input file and update the hoomd_struct object with the user's setting if provided. The expected format for a HOOMD block is as follows:

```
HOOMD
#       PROCESSOR: cpu gpus
#
# PROCESSOR
 gpus
END
```

This format currently only supports the kind of processor to use.

io::In_Parameter::read the function io::In_Parameter::read(...) in `src/io/parameter/in_parameter.cc` contains a list of calls to read blocks in the `md` program input file. A call to read_HOOMD was added to it.

io::read_input the function io::read_input in `src/io/read_input.cc` makes calls to read all the specified input files such as the `md` program input file and the configuration (coordinates file). This indirectly calls io::In_Parameter::read(...) described above. A few lines to construct the object pointed to by simulation::proc based on the settings of the hoomd_struct struct simulation::hoomd were added to the end of io::read_input to ensure that the GPU access is only initiated after all the input files are successfully read.

interaction::create_g96_nonbonded the function interaction::create_g96_nonbonded in `src/interaction/nonbonded/create_nonbonde`
builds and assembles the GROMOS XX Algorithm objects related to non-bonded interactions, such as the Pairlist_Algorithm objects, into a form used by the MD algorithm. This function was modified to build a HOOMD_Pairlist_Algorithm object instead of a conventional Pairlist_Algorithm object if the processor variable of the hoomd_struct simulation::hoomd was not unknown. In addition checks are made to ensure MPI and OpenMP are not used.

3.2.1 Building

A `Makefile` is provided in the `HOOMD_CODE/` directory that will compile and link the code within that directory into a .so file (`libhoomd.so`) if make all is run within that directory. This .so file and all the header files in `HOOMD_CODE/` should be copied manually to the desired install directory.

To build GROMOS XX in such a way as to link to these files, the `./configure` script must be run with an additional argument –with-hoomd=`DIRECTORY`, where `DIRECTORY` is the absolute path to the desired install directory. The use of this option also defines HAVE_HOOMD during compilation.

The –with-hoomd option was made a possible option by the following modifications:

`configure.in` AM_PATH_HOOMD was added after AM_PATH_FFTW3 so autoconf would also call that function in the `./configure` script

`acinclude.m4` AM_PATH_HOOMD was defined as in Algorithm 1.

The `Makefile` provided in the `HOOMD_CODE/` directory can be modified to build in debug mode by replacing -O3 with -g.

The expected device architecture has CUDA Compute Capability 1.0, as represented by the definition CUDA_ARCH=10 in both the `Makefile` and in `HOOMD_GROMOSXX_processor.h`. This can be optimised to 11, 12 or 13 if the hardware to be used has Compute Capability 1.1, 1.2 or 1.3 respectively.

4 Usage

To use the HOOMD code instead of the GromosXX code for whichever kind of pairlist algorithm is requested in the PAIRLIST block (either standard or grid), only an additional block "HOOMD", with the format described below needs to be added to the input file for the `md` program.

```
HOOMD
#       PROCESSOR: cpu gpus
#
# PROCESSOR
 gpus
END
```

Algorithm 1 Addition to `acinclude.m4`

```
dnl check for lib HOOMD
AC_DEFUN([AM_PATH_HOOMD],[
AC_ARG_WITH(hoomd,
[ –with-hoomd=DIR Enable HOOMD code and use the provided directory for .h & .so files],
[
[CXXFLAGS="$CXXFLAGS -DHAVE_HOOMD=1 -I/usr/local/cuda/include -I${withval}"]
[LDFLAGS="$LDFLAGS -L${withval}"]
[LIBS="$LIBS -lhoomd"]
],
[
AC_MSG_WARN([hoomd path was not specified.])
]
)
])
```

PAIRLIST.ALG/HOOMD.PROCESSOR	cpu	gpus
standard	CPU $O(n^2)$	CUDA $O(n^2)$
grid	CPU $O(n)$	CUDA $O(n)$

Table 1: HOOMD pairlist algorithm to use as determined by settings in PAIRLIST and HOOMD blocks in `md` program input file

The effect of the choice of the PROCESSOR argument of this block alongside the choice of pairlist algorithm is shown in Table 1. The HOOMD block is optional; without it HOOMD code is not used.

Also, HOOMD may run more efficiently if the Compute Capability setting during building corresponds to that of the GPUs available.

5 Testing

5.1 Systems

Three water (SPC) boxes were tested as shown in Table 2. The boxes were equilibrated at 300 K, 1 atm prior to testing.

5.2 Tests

The tests consisted of running short simulations and calculating the average time required to build the pairlist. Both NVT and NPT simulations were run. The HOOMD BinnedNeighbourListGPU pairlist algorithm via the HOOMD_Pairlist_Algorithm interface and the GROMOSXX extended grid-based pairlist algorithm (Extended_Grid_Pairlist_Algorithm) running on a single CPU thread were compared. As the HOOMD pairlist algorithm only generates a single-range pairlist, the GROMOS XX pairlist algorithm long-range cutoff was made identical to the short-range cutoff. A cutoff of 0.9 nm was used in all simulations. All code was compiled with -O3 optimisation. A 2.4 GHz AMD Athlon X2 4600 CPU and 4 GB DDR2-667 SDRAM (CAS: 5-5-5-15) and a NVIDIA GeForce 8800 GTS GPU (Compute Capability 1.0) with 320 MB GDDR3 VRAM connected over a 16x PCI-Express bus were used for the tests. Linux 2.6.27.37, g++ 4.3.2 and the NVIDIA 190.29 driver (built for OpenCL and CUDA) was the software environment used.

5.3 Results & Discussion

The average timings are shown in Table 3. Overall, the HOOMD_Pairlist_Algorithm class is slower than the Extended_Grid_Pairlist_Algor class. This is due to two factors that can easily be mitigated. The largest loss of performance arises from the cost of retriev-

Water box	Water molecules	Volume (nm^3)
A	2145	65.74
B	33532	1000
C	112776	3375

Table 2: Test systems: Systems of SPC water of various sizes were tested

Code/Test system	A (NVT Simulation)				B (NVT Simulation)				C (NVT Simulation)			
	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
HOOMD	0.0001	0.0035	0.0508	0.0544	0.002	0.021	0.723	0.746	0.007	0.074	2.344	2.425
	(i)	(ii)		(iii)	(i)	(ii)		(iii)	(i)	(ii)		(iii)
GROMOS XX	0.0005	0.0166		0.0216	0.009	0.304		0.313	0.031	1.026		1.057
	A (NPT Simulation)				B (NPT Simulation)				C (NPT Simulation)			
	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
HOOMD	0.1017	0.0906	0.0551	0.2474	0.063	0.291	0.828	1.182	0.108	0.877	2.610	3.595
	(i)	(ii)		(iii)	(i)	(ii)		(iii)	(i)	(ii)		(iii)
GROMOS XX	0.0005	0.0167		0.0217	0.009	0.308		0.317	0.031	1.034		1.065

Table 3: Average time to build pairlist for different test systems (seconds). HOOMD results show four times: (1) Time required by HOOMD_Pairlist_Algorithm::prepare() function, which includes passing the coordinates from GROMOS XX to HOOMD; (2) Time required to build pairlist on the GPU; (3) Time required to transfer full pairlist back to RAM, extract half pairlist and convert to GROMOS XX format; (4) Sum of times (1)-(3). GROMOS XX results show three times: (i) Time required by Extended_Grid_Pairlist_Algorithm::prepare() function; (ii) Time required by Extended_Grid_Pairlist_Algorithm::update() function; (iii) Sum of times (i) and (ii).

ing the pairlist from the GPU, extracting the half pairlist and converting it to GROMOS XX format, as seen in the third time "(3)" for each HOOMD pairlist timing result. The second largest loss of performance arises from the choice of ensemble: running a NPT simulation is significantly slower than a NVT simulation because HOOMD_Pairlist_Algorithm::prepare() function must rebuild all HOOMD classes every time the system box size changes, as seen in the first and second times, "(1)" and "(2)", for each NPT HOOMD pairlist result. These two drawbacks are easily overcome. First, as the pairlist itself is only used for making the non-bonded interaction calculations more efficient, if these calculations can also be performed on the GPU, the pairlist does not need to be retrieved and can be access directly by the force kernels. Such routines are already available in HOOMD and from Nathan Schmid of IGC. Second, as the relative magnitude of box size changes during a typical NPT simulation is around 1%, and the size of the grid cells used by HOOMD is much larger, it should require only minor modifications to the algorithm for it to work efficiently with box size changes. If these additions are implemented, it is expected that the HOOMD_Pairlist_Algorithm class will be approximately 13x faster than the Extended_Grid_Pairlist_Algorithm class for the test systems B & C.

The pairlists that were generated were compared and found to be identical between the HOOMD_Pairlist_Algorithm and Extended_Grid_Pairlist_Algorithm classes in all cases.

6 Conclusion and Future Work

GPUs are already a cost efficient HPC platform and can be used to accelerate Molecular Dynamics simulations. A number of tasks lie ahead and are given in order of importance:

1. The pairlist generated by HOOMD in CUDA format must be accessed from Nathan Schmid's SPC-SPC interaction code (http://code.google.com/p/gpugromos/). As far as I am aware the format generated by HOOMD is a simple 1D unsigned int array divided into sections, each section holding the pairlist for a given molecule (see `HOOMD_CODE/NeighbourList.cuh`)

2. If NPT simulations are the goal of this work, then the HOOMD pairlist algorithm must be modified slightly to handle slight changes in the box size efficiently. A simple modification that works for isotropic pressure coupling is as follows: rather than sending coordinates and box dimensions that are scaled according to the pressure coupling algorithm, simply reduce or increase the cut-off used by the HOOMD pairlist algorithm so that the pairlist generated would be limited or increased proportionately.

3. The HOOMD pairlist algorithm may run more efficiently on the GPUs available with a different CUDA block size. This can be set in HOOMD_Pairlist_Algorithm by calling the HOOMD function setBlockSize() once prior to calling compute(), e.g.

```
nlist->setBlockSize(64); // in prepare() after constructing nlist
...
nlist->compute(++timestep); // in update()
```

4. The HOOMD pairlist algorithm should be modified to return two pairlists, both short and long range.

5. The GROMOS XX code should be modified so that a solute-solute/solute-solvent pairlist is generated efficiently on the CPU, and optionally to run simultaneously alongside the CUDA code. Currently, the HOOMD_Pairlist_Algorithm interface does not support MPI or OpenMP. Note, HOOMD internally uses all available GPUs and only one CPU thread should access HOOMD functions.

6. The HOOMD pairlist algorithm should be modified to support non-rectangular boxes.

7 Code & Documentation Accessibility

The code is accessible from: https://gromos.ethz.ch/svn/repos/gromosXXc++ revision 1330 (see `trunk/gromosXX/contrib/HOOMD_`
The test systems are available from: lychee.md.smms.uq.edu.au:`/home/matt/HOOMD-Pairlist-waterboxes`. This file and its PDF are in the same revision at `trunk/gromosXX/doc/`.