GROMACS - Feature #1972

external potential modules for refinement against experimental data

05/27/2016 01:55 PM - Christian Blau

Status:	New	
Priority:	Normal	
Assignee:		
Category:	core library	
Target version:	future	
Difficulty:	uncategorized	

Description

Forces, potential and virial contributions from external potentials.

Generalise the treatment of external potentials from external input data to make it easier, cleaner and safer to implement refinement protocolls in gromacs.
Create a modular infrastructure for external potentials similar to the trajectoryanalysis framework that bundles code contributions for external potentials in exactly one place.

Removing the need to

- ~~change code that reads mdp files~~
- ~~read/write tpr data~~
- ~~change the inputrecord~~
- ~~change mdrun~~

The module will in one place

- Declare its required input and name
- provide the force per atom, its contribution to the potential energy and virial

Implementation challenges

Parameter relay

Adding single parameters

- New parameters (like force-constants and field strength) will have to be relayed to the external potential through mdp options or an additional external file
- How does grompp know which mdp parameters are expected, which are misspelled?

For a first suggestion see Teemu Murtolas comment on the gromacs-dev list:

> "Just making the validated input string queryable for all seems to achieve the exact opposite: it makes a single, global instance of the whole configuration that everyone can read and parse as they see fit, which makes it possible for people to (mis)use data from other modules and even parse it differently in different places in the code. And the input string has even less compile-time checking or structure than the current t_inputrec.
>
> To achieve proper isolation, we would need a pattern where the module can declare their mdp options, and then it gets access to exactly those values. Preferably without knowing anything about the underlying storage format being JSON or having to declare the options in more than one place. Everything else would need to be passed through explicit coding. One option would be to use the options/ module for declaring the options, but that would require some work to support structured data (in particular, arrays of JSON objects would need some non-trivial work). This has the disadvantage that it will be difficult to have the mdp file have structure that does not closely reflect the structure of the code, but for most cases, mdp options provided by different modules would probably fit within their own dedicated sections... And it will be strict in enforcing that isolation, for better or worse.
>
> For reading data for analysis, or for pdb2gmx, the requirements are probably different, as the module(s) that read the data can have that as their main responsibility. So there it should be fine to just write classes to extract certain things from the parsed data and expose that as useful data structures (probably in a format where you can query it by atom/residue/whatever name)."

- ExternalPotential should have a check_input_consistency

Adding generic, force-field like parameters

- Parameters that are constant to an external potential, like scattering factors, should be read from one library file
- How to add properties not only to atom types, but also to single atoms?
- How to treat larger entities with the same property, e.g., the number of electrons of a coarse-grained bead?

External input files

Often experimental data comes in vanilla input file formats, like a scattering curve or a ccp4/mrc map.

- Dump everything in the tpr file, or keep only file name and a check sum in the tpr file?
- Where should tools for parsing these file formats reside - within the modules, within fileio of gromacs or a third "parse experimental data" place?

External Potential Group indexing

- Generate a Group class, that allows easy looping over indexed atoms
- Provide group class decorators that allow to make molecules whole - which will require domain decomposition updates to the groups

Efficient coordinate and property communication

No communication is required when external potential depends on atom coordinates only (e.g. pulling all atoms to 0,0,0). The other trivial, but inefficient solution is to communicate all external potential atom coordinates to the master node, perform the calculation there and report back forces.

However, most commonly we want a structure that calculates some intermediate result M_i from all local external potential atoms, then calculates some property from this intermediate.

M_1 <- x_1 .. x_n
M_2 <- x_n+1 .. x_m
--
M <- M_1 .. M_i

then calculate V(M,x_j), F(M,x_j), T_virial(M,x_j)

Hooks to the external potential

- in the mdp file read
- print ir routines
- runner

Logging / Output

What should appear in the md.log file, what goes to seperate output files?

Checkpointing

To ensure simulations with an external potential are correctly continued, write to checkpoint file what is needed for a continuation from the external potentials.

Parallelization / Tasking / Load balancing

Account for the time spent for the external potentials.
Implement some tasking scheme for external potentials.

Related issues:

Related to GROMACS - Task #2017: Modularize simulation option storage and rea...	**In Progress**
Related to GROMACS - Feature #2585: Infrastructure supporting external API	**Resolved**
Related to GROMACS - Task #2623: Allow extensible MDModules and forceProviders.	**Resolved**

Associated revisions

Revision 41e0f9cc - 06/27/2016 07:41 PM - Teemu Murtola

Factory for creating t_inputrec

Add gmx::MDModules that encapsulates initialization of t_inputrec in all
places where it gets created. This provides a single place that needs
to change with the introduction of a more modular approach for mdrun
modules. Child changes will demonstrate how this can be used for the
external electric field code. Split into a separate change to keep the
main change smaller and more focused.

For now, put the code in mdrunutility/, as it needs to be quite high in
the dependency stack, but a better home could be found in the future.

Related to #1972.

Change-Id: I0d86fe1a8bb8f8f872f5b37bc4e7931dceae9f09

Revision 13e5b63a - 10/07/2016 12:54 PM - David van der Spoel

Modularize electric field handling

Move external electric field code to a C++ class where only a single
place in the code is responsible for initializing it, and everything
else operates only on relatively general interaces. Details of the
electric field stuff is encapsulated within the class.

Moved the files into new directory applied-forces.

Added manual section for electric fields.

The interfaces may need more generalization to support more complicated
modules like the pull code, but that is better done when actually moving
those parts to this approach. Various considerations are documented in
mdmodules.h and inputrect.h.

Part of #1972.

Change-Id: I513632c74a8fae28b5f1087a3b5781791a2627bd

Revision 7075773c - 07/16/2018 08:28 PM - Christian Blau

Add LocalAtomSet for handling atom indices.

Moves reading and updating indices for a set of atoms to a single place.
Pulling, IMD, rotation enforcement, essential dynamics etc. all use
their own methods triggered in domain
decompostion to handle atom indices in parallel simulations.
This class works towards removing the exposed hooks to all these modules
in domain decomposition.

Part of casting groupcoord routines to c++.

External potentials will use LocalAtomSet to access atom subsets to make
handling of atom indices not the task of individual
external potentials, but of an parallel-Atom API instead.

Related to #1972.

Change-Id: I69460963927249c7d713e76f166e6c87122c68de

Revision cdbc0e9c - 08/07/2018 09:21 AM - Christian Blau

Local atom sets in compEL / ion swapping code.

Local atom sets remove depency of domain decomposition on the
computational electrophysiology modules and handle atom index updates
outside of this module.

Additional refactoring, because swap_group has to be a C++ struct now.

Refers to #1972

Change-Id: Ic6ee2a71f5c5c11e161fa26aafbed2764e014daa

Revision 9864b201 - 08/29/2018 02:32 PM - Eric Irrgang

Allow extensible MDModules and forceProviders.

supports gmxapi milestone 6, described at #2585.

MDModules::Impl gets a std::vector for (shared) ownership of objects providing the IMDModule interface. An add() method is added to the MDModules public member functions, but the binding protocols are separated into separate issues to allow minimal changes and varying dependencies on other pending changes.

Relates to #1972, #2229, #2492, #2574, #2590.

Refs #2623

Change-Id: Ibb16d1453003213a49622810ed8bad4ed4b06e2d

History

#1 - 05/30/2016 10:38 AM - Carsten Kutzner

Christian Blau wrote:

> # Efficient coordinate and property communication
>
> No communication is required when external potential depends on atom coordinates only (e.g. pulling all atoms to 0,0,0). The other trivial, but inefficient solution is to communicate all external potential atom coordinates to the master node, perform the calculation there and report back forces.

There is an intermediate solution which works surprisingly well: Communicate all external potential atom positions to ALL nodes, then let each node evaluate the forces on its LOCAL atoms only. This works surprisingly well when the part of the atoms subject to the external potential is not too large (if the water is not part of the external potential this is the case). The enforced rotation code does that (there are even benchmarks with the performance impact of that approach in the paper), and also the essential dynamics code. The routines are in groupcoord.cpp, where is also code that can take care of always having the external potential atoms 'whole' in PBC (given a 'whole' starting configuration'.)

#2 - 06/23/2016 04:54 PM - Christian Blau

Find an up to date sketch of the interface here:
externalpotential.dot.svg

#3 - 06/28/2016 09:20 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #1972.
Uploader: Teemu Murtola (teemu.murtola@gmail.com)
Change-Id: I0d86fe1a8bb8f8f872f5b37bc4e7931dceae9f09
Gerrit URL: https://gerrit.gromacs.org/5995

#4 - 06/29/2016 05:28 PM - Gerrit Code Review Bot

Gerrit received a related patchset '48' for Issue #1972.
Uploader: Teemu Murtola (teemu.murtola@gmail.com)
Change-Id: I513632c74a8fae28b5f1087a3b5781791a2627bd
Gerrit URL: https://gerrit.gromacs.org/5372

#5 - 06/30/2016 09:16 PM - Teemu Murtola

https://gerrit.gromacs.org/5372 introduces an initial interface like requested here that works for the electric field code. There are several things missing from there, as those are not necessary for the electric field case, but if we move additional functionality behind the same interface and generalize it in the process, we should end up with something usable. I would advocate this kind of an incremental approach that works on real code, since otherwise it will be difficult to get anywhere.

Once more code is behind this kind of interfaces, we can also see how to best manage the modules, as the current implementation is for sure not final.

#6 - 07/02/2016 10:01 PM - Peter Kasson

As we discussed in the meeting, I **strongly** believe we need the API specified (i.e. the external interface) and kept simple. API's aren't made to be broken at will. They need to be designed carefully, might go through an alpha period, and then need to be kept stable once published. So rather than doing a "move fast and break things", we need to specify a simple API (which I think we did on the board snapshot) and then back it up with code. Throwing code at a problem is developing backwards. /soapbox

#7 - 07/03/2016 05:54 AM - Teemu Murtola

Peter Kasson wrote:

> As we discussed in the meeting, I **strongly** believe we need the API specified (i.e. the external interface) and kept simple. API's aren't made to be broken at will. They need to be designed carefully, might go through an alpha period, and then need to be kept stable once published. So rather than doing a "move fast and break things", we need to specify a simple API (which I think we did on the board snapshot) and then back it up with code. Throwing code at a problem is developing backwards.

On a high level, I agree, but the problem is about arriving at that API specification. Realistically, in an environment where you are lucky to have a 0.1 FTE working on any problem, creating an API specification will take years. Doing that as a pure blackboard exercise is very useful to capture high-level needs, but you need to put a **lot** of effort into that if you want to get every single detail correct and usable. And very likely, you will still find something to correct once you start implementing things. If you rush things with designing an API without having enough code to back it up before you publish it (both on the implementation and using side), and want to be strict about maintaining compatibility, you will completely kill progress since no one can then improve things any longer when they realize that something was not considered from the start (in particular in a resource-constrained environment where you don't have the resources to write the backwards-compatibility glue, either).

Also, if you start with a world-class API first, you will also spend a lot of time implementing that API on top of the existing code, before you can implement any code that for real uses the API and you can verify all the details of your design. Not to mention that the current structure of the code may not support implementing your API vision, so you either need to rework it a lot to get anywhere, or spend a lot of time writing temporary wrappers.

A high-level/target vision is good to have, but I think we can arrive at a useful, working, and realistic API specification much faster if we implement it in smaller steps, and have some code that uses it from the start. We already have several instances of code in mdrun that would benefit from this kind of an interface, and the API must be able to support those as well to be useful. Once we get in the minimal changes to have a few realistic modules to run under this kind of a structure, we can implement the details of the final API and verify that everything works, again one piece at the time. With the reality of the availability of resources, it will still take years, no matter what approach we use, but the steps are possible to test, code review, and evaluate in isolation, and merge to the development version in manageable chunks. This is not a greenfield project where you can write all the code from scratch to support from your API, but it needs to work in the context of the existing mdrun.

#8 - 07/03/2016 01:31 PM - Peter Kasson

At the meeting we had, we put a starting point spec on the board that everyone there (Erik, David, Berk, Mark, Christian, and I) could agree on (that was my impression, at least. speak up if you were there and don't agree). I think if we can implement that clean interface, we'll be on a good start.

Regarding the resources question, the US NIH believes a good high-level API is a Good Thing (TM) for Gromacs and has decided to put money behind it. In a recently awarded grant, I have developer resources for this task (focusing on a multi-language high-level API, which of course also synergizes with development elsewhere in the Gromacs project). So we can put person-time on the project. We're currently hiring. If you know a good candidate for a postdoc (or a fractional software engineer FTE; engineers are a lot more expensive, of course), please encourage them to contact me.

#9 - 07/04/2016 10:58 AM - Christian Blau

My impression is that we had a pretty cohesive idea about our target-vision API, as the IExternalPotential in the sketch above which reflects the "board"-snapshot;

For we have a good set of cases that span a use space for external potentials,

- electric field handling
- pull code
- interactive molecular dynamics
- saxs refinement
- fitting against densities (above graph is based on the actual code for this),

my preferred route would be to generate an alpha-API by moving them one-by-one into above framework, and after each case moved reiterate the design questions.

#10 - 07/04/2016 11:05 AM - Christian Blau

Looking at the electric field implementation, I was wondering if we should split IExternalPotential into IForceProvider, IEnergyProvider, IVirialProvider?

This might be beneficial for cases where Energy, or Virial are not imlemented and instead of just returning zero for these values, this would make it explicit in the code that these are not calculated; otherwise we might want to extend the ExternalPotentialResult with something like hasEnergy, etc.

#11 - 07/05/2016 09:09 PM - Teemu Murtola

Christian Blau wrote:

> Looking at the electric field implementation, I was wondering if we should split IExternalPotential into IForceProvider, IEnergyProvider, IVirialProvider?
>
> This might be beneficial for cases where Energy, or Virial are not imlemented and instead of just returning zero for these values, this would make it explicit in the code that these are not calculated; otherwise we might want to extend the ExternalPotentialResult with something like hasEnergy, etc.

I'm not sure whether that separation makes much sense, since in most cases, you will evaluate everything that is needed in one go, and it will create a complicated code structure if you need to return that from different functions for different properties.

I think one thing that would allow simplifying the API significantly would be to work on the mdp and tpr formats. I think that it should be possible to just make the external potential module declare the options it supports, but get rid of the code to print or compare the values. And the way to do this would be to parse the mdp file to an internal, generic representation (for example, a JSON-like structure), write this structure instead of t_inputrec/other binary stuff to the tpr file, and then use the same structure when reading either mdp or tpr file to actually assigning the values to the module. The structure could also then be used for printing and comparison without any actual knowledge of the external potential modules.

I'm working on the options module to see what kind of an interface it could provide for declaring the options, but we also need decisions on the needed backwards compatibility for mdp and tpr files, on the new MDP format (JSON or something else), and if JSON, on the library to use for parsing. And on whether we need the current functionality in grompp that writes out an mdp file with defaults intertwined with the user-provided values.

#12 - 07/07/2016 04:25 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #1972.
Uploader: Christian Blau (cblau@gwdg.de)
Change-Id: I69460963927249c7d713e76f166e6c87122c68de
Gerrit URL: https://gerrit.gromacs.org/6020

#13 - 07/28/2016 09:27 AM - Teemu Murtola

- Related to Task #2017: Modularize simulation option storage and reading from mdp files added

#14 - 03/14/2018 01:34 PM - Gerrit Code Review Bot

Gerrit received a related DRAFT patchset '21' for Issue #1972.
Uploader: Christian Blau (cblau@gwdg.de)
Change-Id: gromacs~master~I69460963927249c7d713e76f166e6c87122c68de
Gerrit URL: https://gerrit.gromacs.org/6020

#15 - 03/14/2018 01:34 PM - Gerrit Code Review Bot

Gerrit received a related DRAFT patchset '1' for Issue #1972.
Uploader: Christian Blau (cblau@gwdg.de)
Change-Id: gromacs~master~Ic6ee2a71f5c5c11e161fa26aafbed2764e014daa
Gerrit URL: https://gerrit.gromacs.org/7680

#16 - 04/13/2018 06:18 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #1972.
Uploader: Christian Blau (cblau@gwdg.de)
Change-Id: gromacs~master~I497604f2f3293b62a214bdb16fbf82ef382102cf
Gerrit URL: https://gerrit.gromacs.org/7761

#17 - 07/30/2018 05:47 PM - Eric Irrgang

- Related to Feature #2585: Infrastructure supporting external API added

#18 - 08/22/2018 05:03 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #1972.
Uploader: M. Eric Irrgang (ericirrgang@gmail.com)
Change-Id: gromacs~master~Ibb16d1453003213a49622810ed8bad4ed4b06e2d
Gerrit URL: https://gerrit.gromacs.org/8219

#19 - 08/22/2018 05:03 PM - Eric Irrgang

- Related to Task #2623: Allow extensible MDModules and forceProviders. added