GROMACS - Task #2017

Modularize simulation option storage and reading from mdp files

07/26/2016 05:02 PM - Christian Blau

Status:	In Progress
Priority:	Normal
Assignee:	Christian Blau
Category:	core library
Target version:	future
Difficulty:	uncategorized

Description

Replace the reading of all input parameters in one place (readir.cpp) with a structure, where each MD-module declares its parameters and is itself responsible for checking input consistency.

This should greatly help the implementation of modular external potential modules, as well as modularisation of the existing code.

Related to #1972 (External Potential Modules), #1842 (Replace XML with JSON), #1854 (Remove all cyclic dependencies), #1964 (pull="no" warnings), #1889 (checkpoint file writing) and #1806 ("ref-t" for temperature coupling not validated)

We had previous discussion about this on the gromacs-dev list, citing Teemu Murtola:

> 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)."

Related issues:

Related to GROMACS - Feature #1972: external potential modules for refinement...	**New**
Related to GROMACS - Feature #1842: Replace XML with JSON	**New**

History

#1 - 07/28/2016 09:27 AM - Teemu Murtola

The options topic currently in Gerrit has the basic infrastructure for using the options module for this. The next steps would be to make the electric field code use this in small steps (to make it clear how it would work), replacing most of IInputRecExtension, once everything is rebased into a single chain of commits (or something has been merged and accepted).

#2 - 07/28/2016 09:27 AM - Teemu Murtola

- Related to Feature #1972: external potential modules for refinement against experimental data added

#3 - 07/28/2016 09:27 AM - Teemu Murtola

- Related to Feature #1842: Replace XML with JSON added

#4 - 10/31/2016 03:42 PM - Mark Abraham

At https://gerrit.gromacs.org/#/c/6263/7, Erik asked

> Another question (not directly relates to this change per se): What's the intention of the relation between MDP options specified in the KVT vs. the usage of the options module for the settings? Right now both seem to be involved, but I'm not sure about their relation. Is the idea to move to a single set of options for modules (which would be a good idea - for the module I guess it's irrelevant whether something is from the command line or a file)?

I think there's some benefit in having a single options object that is available for each module, which permits us to decouple whether the source was mdp, command line or environment variable, particularly thinking of settings that some kind of user might choose. This makes it much easier to test a module in isolation (or a group of them). There would also be uniformity in how each module interacts with its inputs.

There is also the result of hardware introspection, but it seems a bit different, because it's a set of observed facts that have to be lived with, rather than a set of instructions to be followed as appropriate.

There will have to be high-level code that will interact with multiple modules, so I guess we need a mechanism to say "(if this module exists) read the value (tree) for this key" so that high level coordination can work.

#5 - 10/31/2016 07:33 PM - Teemu Murtola

Mark Abraham wrote:

> I think there's some benefit in having a single options object that is available for each module, which permits us to decouple whether the source was mdp, command line or environment variable, particularly thinking of settings that some kind of user might choose. This makes it much easier to test a module in isolation (or a group of them). There would also be uniformity in how each module interacts with its inputs.

That is true, provided that those input options really are uniform in the way they should be handled. This also means that they will all be stored in the tpr file, and that any command-line or environment variable mechanism that can be used to set them can be used to set any other option from such a set.

If there is some variability, it would be clearer to keep these separate. All of them can still be handled as Options, but options with different semantics should not be mixed in the same object. For testing, I don't think there is much difference between these cases.

> There is also the result of hardware introspection, but it seems a bit different, because it's a set of observed facts that have to be lived with, rather than a set of instructions to be followed as appropriate.

Yes, that is completely different, and should be handled by passing the data to the module where required (and not allowing the module to change any of it).

> There will have to be high-level code that will interact with multiple modules, so I guess we need a mechanism to say "(if this module exists) read the value (tree) for this key" so that high level coordination can work.

The current approach is intentionally restrictive: exactly one module can declare an option for a certain parameter, and there is no direct access to the raw tree data structure. This means that for every input parameter, there must be exactly one module that is responsible for declaring it and owning the value. If multiple modules need some values, then these need to be parsed in a common module, and that data passed to the other modules (preferably as immutable data). The parsing code can be in the higher-level code as well, for values that cannot be easily encapsulated.

The high-level code can also force a set of modules to put all their options in a common subsection that is declared in the high-level code, if that is desired (currently, declaring the same subsection in multiple modules also gives an error).

#6 - 12/20/2016 05:28 AM - Mark Abraham

Teemu's approach seems very good

#7 - 03/15/2017 06:03 PM - Gerrit Code Review Bot

Gerrit received a related patchset '2' for Issue #2017.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: gromacs~master~I7be0241d433f93c8af107a9fc1d4b4d51a045802
Gerrit URL: https://gerrit.gromacs.org/6485

#8 - 03/15/2017 06:04 PM - Mark Abraham

- Target version changed from 2018 to future

more work needed here

#9 - 04/06/2018 06:07 PM - Mark Abraham

- Status changed from New to In Progress

I made considerable progress on expanding https://gerrit.gromacs.org/#/c/6485/ but it will basically invalidate all tpr files in regressiontests, and keep doing so as we port modules to key-value format.

I think we should regard porting the regression test functionality into the source repo as a prerequisite for this task, and choose carefully which legacy .tpr file versions we want to continue testing that can be read and produce the right output.

The existing test functionality that verifies that gmx check -s1 reference_s -s2 topol is a barrier to progress. So long as the check functionality is

tested, and that we can still read legacy files is tested, it is not necessary to store a whole battery of .tpr files (requiring periodic updates) just to verify that we didn't inadvertently change the .tpr format. For example, as the key-value specification evolves, individual modules will be able to decide whether they can implement the simulation that the user's mdp or tpr input requires. That makes sense to test unit style - give .mdp inputs in whatever formats are still supported, and assert that the module sets the current parameters accordingly to the right values (or gives an error).