GROMACS - Task #2766

Improve hardware option selection

11/18/2018 03:35 AM - Kevin Boyd

<table>
<thead>
<tr>
<th>Status:</th>
<th>In Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>Kevin Boyd</td>
</tr>
<tr>
<td>Category:</td>
<td></td>
</tr>
<tr>
<td>Target version:</td>
<td>2021</td>
</tr>
<tr>
<td>Difficulty:</td>
<td>uncategorized</td>
</tr>
</tbody>
</table>

Description

MDrunner currently uses a single struct to determine hardware usage options such as number of thread-mpi and openmp ranks. The struct gets passed to a number of functions that can modify it, and the values are overwritten. The two main issues from this are that A) we can't tell if a value was decided by the user or by the program, and B) it's impossible to guarantee that user-defined options are always honored.

These issues could potentially be addressed by expanding the hw_opt_t struct into a more full-fledged class with some construction-time evaluation of whether the user set any of the options, and making the members private and having a check in setters to make sure no overriding of user-defined parameters occurs.

History

#1 - 11/19/2018 02:12 AM - Gerrit Code Review Bot
Gerrit received a related DRAFT patchset '5' for Issue #2766.
Uploader: Kevin Boyd (kevin.boyd@uconn.edu)
Change-Id: gromacs~master~Ia55d39be1559c52422669d6cde8cc21ae72a82ff
Gerrit URL: https://gerrit.gromacs.org/8719

#2 - 11/19/2018 12:54 PM - Eric Irrgang
There are a couple of policy issues that maybe should be explicit somewhere:

If a user explicitly chooses a value for a parameter, and that chosen value is the default value, what is recorded?
I think the choice to override should be recorded, because

- the default may change in a future version and artifacts should be distinguishable
- some of the more complicated set-up logic may override / change defaults depending on other values

What parameter state is recorded and checkable?

Things to distinguish:

1. Has a field in a structure been initialized?
2. Is the current value provided by user input?
3. Is the current value generated by heuristic logic?

Do all of these questions need to be answerable at the same time, all of the time?
I think the issue becomes simpler if we can answer "no". For instance,

1. user-provided values are held in a version of the structure with fields of (a type compatible with https://en.cppreference.com/w/cpp/utility/optional) that is passed from the client code to the next level where
2. a new structure is populated (through a builder) from some combination of user input and defaults / generated values.

If more initialization stages are needed, then step 2 gets duplicated and the structures passed from one stage to the next use the "optional" pattern.

This has the added benefit of decoupling the structures used for user input from the structures used in API calls or within the library implementation code.

#3 - 11/21/2018 12:29 PM - Mark Abraham
Eric Irrgang wrote:

There are a couple of policy issues that maybe should be explicit somewhere:
If a user explicitly chooses a value for a parameter, and that chosen value is the default value, what is recorded?

If they chose a value, we record that. It's not significant that they chose something that happens to be the default right now.

I think the choice to override should be recorded, because

- the default may change in a future version and artifacts should be distinguishable
- some of the more complicated set-up logic may override / change defaults depending on other values

I think it's simpler than that - record what they chose. Everything else is an implementation detail of the module being initialized. Part of the current confusion is that we use the same object to do both right now. The simplest thing we could do would be to have two copies of that - one of which is clearly never changed.

What parameter state is recorded and checkable?

The user input, and the state of the module once prepared to work.

Things to distinguish:

1. Has a field in a structure been initialized?
2. Is the current value provided by user input?
3. Is the current value generated by heuristic logic?

Do all of these questions need to be answerable at the same time, all of the time?

I think the issue becomes simpler if we can answer "no". For instance,

1. user-provided values are held in a version of the structure with fields of (a type compatible with 'https://en.cppreference.com/w/cpp/utility/optional') that is passed from the client code to the next level where
2. a new structure is populated (through a builder) from some combination of user input and defaults / generated values.

If more initialization stages are needed, then step 2 gets duplicated and the structures passed from one stage to the next use the "optional" pattern.

This has the added benefit of decoupling the structures used for user input from the structures used in API calls or within the library implementation code.

I like the idea of std::optional (or something similar, we have gmx::Variant also).

Something like ir->nsteps is expressed well as a variant. Either it's an integer that the user chose, or it's a special choice they made (e.g. in the .mdp, we accept -1 to mean infinite), or it's "default." The Integrator setup can then interpret default however makes sense (and now that can be different for MD and EM), and internally it can use a type that properly expresses the user's expectations about completion, because an integer doesn't do the job. Now when we get a checkpoint that has already done too many steps we can't just accidentally misuse the < operator.

#4 - 12/20/2018 06:40 PM - Kevin Boyd
- Status changed from New to In Progress

#5 - 12/20/2019 02:34 PM - Artem Zhmurov
- Target version changed from 2020 to 2021

Not feasible in 2020