GROMACS - Bug #2255

nstlist override stopped affecting the input parameter listing

09/19/2017 05:33 PM - Szilárd Páll

Status:	New		
Priority:	Normal		
Assignee:			
Category:	mdrun		
Target version:			
Affected version - extra info:		**Difficulty:**	uncategorized
Affected version:	git master		

Description

As a result of some recent refactoring, the master branch code no longer overrides nstilist early enough for the input parameter listing to correctly specify the nstlist parameter.

Not sure if this is an oversight or a certain interpretation of what the "input parameter" listing should contain.

History

#1 - 09/20/2017 02:22 PM - Berk Hess

I noticed this as well. But I don't know if the old or new behavior is better.
A better approach for the future might be to set both nstlist and rlist to unset, e.g. -1, instead, so it's clear they will be optimized at run time.

#2 - 09/20/2017 03:34 PM - Szilárd Páll

I'm not sure either which one is better, but inconsistency and silent changes in behavior are certainly not great! Free parameters that can and are sometimes changed by the user (or heuristic tuning) should happen in a predictable manner.

I'm leaning toward the opinion that the input parameter list should be consistent and self-contained so that a user does not have to look at override messages in the log just to know what nstlist was used.

Perhaps we should add some kind of test that checks (parts of?) the log file output against a reference so we can be certain we don't miss log output changes?

#3 - 09/20/2017 09:31 PM - Berk Hess

For nstlist this has worked. But rlist can be changed dynamically during the run by PME tuning. We can also imagine dynamic nstlist tuning.

#4 - 09/21/2017 12:05 AM - Szilárd Páll

> For nstlist this has worked. But rlist can be changed dynamically during the run by PME tuning. We can also imagine dynamic nstlist tuning.

That is a good point, but having the listing contain the initial overrides -- especially when there is only a one-time override and no tuning -- still seems to make sense.

Berk Hess wrote:

> A better approach for the future might be to set both nstlist and rlist to unset, e.g. -1, instead, so it's clear they will be optimized at run time.

Command-line overrides are also omitted, I think.

#5 - 09/22/2017 08:39 PM - Mark Abraham

I think the most useful thing for a user is to know what the simulation engine is actually doing ie after defaults and overrides are applied; what they chose is available to them in other ways (mdp file, shell history, job script, tpr, etc. plus perhaps log file). Secondarily, that information is often what we want to know in bug reports or user assistance. Given our implementation decisions over the years, probably it is best to also acknowledge whether the chosen value arose from the action of our defaults, or a user's choice.

Attempting to verify that the reporting has not changed in unanticipated ways is somewhat tricky, given that it will be highly specific to the combination of inputs, and that we have not designed for a seam point where everything has its (initial) value for the simulation. We shouldn't attempt capture log files and assert they haven't changed without a lot of attention to making the reference data agnostic with respect to the current configuration of tests and build slaves. Doing a not-very-good job of that will become a source of irritation to developers - I avoid changing the text of warning messages, because if I do so I have to bump the regressiontests suite and deal with that pain. Tests within the source repo would work more smoothly, but I still don't think the result (log file doesn't change except as intended) is not worth the effort.

#6 - 11/12/2017 06:49 PM - Szilárd Páll

Mark Abraham wrote:

> I think the most useful thing for a user is to know what the simulation engine is actually doing ie after defaults and overrides are applied; what they chose is available to them in other ways (mdp file, shell history, job script, tpr, etc. plus perhaps log file). Secondarily, that information is often what we want to know in bug reports or user assistance. Given our implementation decisions over the years, probably it is best to also acknowledge whether the chosen value arose from the action of our defaults, or a user's choice.

First you seem to suggest that all overrides should happen early so that we present the "Input Parameters" listing with the options as used at runtime. That is exactly the behavior that the code reorganization broke, so do I understand correctly that you suggest reverting the order?

Additionally, as Berk pointed out, we have two kinds of overrides: early and late. For some options the heuristic switch happens much later (rlist, nstlist, PME grid) and for these we inherently can not display the final setting at the point where the "Input Parameters" table is printed. We could still decide to apply early overrides on the listing and do only the runtime tuning later, but what's important, I think is to do this consistently (which happens to be an improvement now as the nsteps override is done after the parameters listing) and document the behavior.

> Attempting to verify that the reporting has not changed in unanticipated ways is somewhat tricky, given that it will be highly specific to the combination of inputs, and that we have not designed for a seam point where everything has its (initial) value for the simulation. We shouldn't attempt capture log files and assert they haven't changed without a lot of attention to making the reference data agnostic with respect to the current configuration of tests and build slaves. Doing a not-very-good job of that will become a source of irritation to developers - I avoid changing the text of warning messages, because if I do so I have to bump the regressiontests suite and deal with that pain. Tests within the source repo would work more smoothly, but I still don't think the result (log file doesn't change except as intended) is not worth the effort.

We should be able to identify well-defined parts of the log file that we want to keep consistent and we want to avoid changing in an incidental or ad-hoc manner. I do think consistency is important, but it may well be that this will be considered more effort than what it's worth, I admit. I still think it's easily doable, though.

For a given fixed set of inputs (tpr and command line) everything between "Input Parameters:" and "Started mdrun on rank..." should stay unchanged. Additionally, given a fixed set of build parameters (same compilers, libraries) even the initial part of the log (from line 3 until "Input Parameters:") should stay the same with the exception of the "Built on:" and "Build OS/arch:" -- which are not hard at all to either omit printing (which BTW could be useful for package maintainers because some distros strive to make reproducible builds which are prevented by these two fields) for this very test build or remove before comparing.

#7 - 12/15/2017 11:45 AM - Erik Lindahl

I wouldn't primarily blame the code reorganization for breaking it - the traditional setup was that input values were simply fixed, and did not change. Since then we have started to change various parameters in a few different places, without really deciding what can be changed where and when.

I think most of this will be solved by better modularization. We should have one clear place where we read all data and (early) change global settings that might influence lots of things. After this, hierarchical modules (e.g. nonbonded forces, including short- and long-range) should gradually initialize themselves based on the default settings, and potentially print some things that are overridden inside each module, and after that sub-modules.

The one thing we need to find a way to avoid is doing late global overrides that influence lots of things. Then the only choice will be to reinitialize all modules.

#8 - 12/17/2017 06:53 AM - Mark Abraham

Erik Lindahl wrote:

> I wouldn't primarily blame the code reorganization for breaking it - the traditional setup was that input values were simply fixed, and did not change. Since then we have started to change various parameters in a few different places, without really deciding what can be changed where and when.

Indeed. With hindsight, we didn't realise the full scope of the problem of attempting to re-use nstlist and rlist. The scope of the changes we made about the time we introduced the Verlet scheme meant nobody took the time to focus on the wisdom of changing the .mdp UI, or not.

> I think most of this will be solved by better modularization. We should have one clear place where we read all data and (early) change global settings that might influence lots of things. After this, hierarchical modules (e.g. nonbonded forces, including short- and long-range) should gradually initialize themselves based on the default settings, and potentially print some things that are overridden inside each module, and after that sub-modules.

IMO there should be no such thing as a global setting. For example, domain decomposition is required to satisfy the needs of several other modules. rlist is merely the one that the short-ranged scheme cares about. As the Verlet scheme is constructed, it will register itself with the DD manager, and at DD time it will be asked what it cares about, and now whatever the current dynamic value of nstlist and rlist are should flow. The DD module probably needs to know about no .mdp parameters whatsoever, but does have some mdrun command-line parameters.

> The one thing we need to find a way to avoid is doing late global overrides that influence lots of things. Then the only choice will be to reinitialize all modules.

Explicitly expressing such initialization dependencies is essential. Even we don't care about our own sanity, you can't have an API-driven simulation where the user can change a parameter and not get what they chose.

#9 - 12/17/2017 08:05 AM - Mark Abraham

Szilárd Páll wrote:

> Mark Abraham wrote:
>
>> I think the most useful thing for a user is to know what the simulation engine is actually doing ie after defaults and overrides are applied; what they chose is available to them in other ways (mdp file, shell history, job script, tpr, etc. plus perhaps log file). Secondarily, that information is often what we want to know in bug reports or user assistance. Given our implementation decisions over the years, probably it is best to also acknowledge whether the chosen value arose from the action of our defaults, or a user's choice.
>
> First you seem to suggest that all overrides should happen early so that we present the "Input Parameters" listing with the options as used at runtime. That is exactly the behavior that the code reorganization broke, so do I understand correctly that you suggest reverting the order?

I said that I thought the output should reflect what the simulation is doing. If "Input parameters" is that output, then it should not happen until after the short-ranged code has finished changing things. But given that I had a reason for moving the point where the change happens, it would be far simpler and likely best to move the point where that output is made, not to revert what was probably a complex decision involving multiple execution flows.

Each module should eventually be responsible for its own reporting, and that should happen when the owner of the module handles executes a loop over all of them, and that point will naturally be after all of the initialization and any inter-module interaction.

> Additionally, as Berk pointed out, we have two kinds of overrides: early and late. For some options the heuristic switch happens much later (rlist, nstlist, PME grid) and for these we inherently can not display the final setting at the point where the "Input Parameters" table is printed. We could still decide to apply early overrides on the listing and do only the runtime tuning later, but what's important, I think is to do this consistently (which happens to be an improvement now as the nsteps override is done after the parameters listing) and document the behavior.

I agree with Berk that we should make the defaults for nstlist and rlist clearly auto, e.g. -1. Now there is no question of calling anything an override. (We will change variables during tuning - whether or not we call that an override.) Either we do what the user explicitly asked for, or we do what we think best if the user asked for auto. But to get to that point, we need to change the names of such options, because otherwise we have to deal with old mdp files that we used to override, but now produce something other than the user nominally asks for. We can't dig ourselves out of the hole without changing the names, which I now think we should have done originally. (That kind of approach would also have stopped people blindly using old twin-range .mdp files with the new Trotter implementation...)

>> Attempting to verify that the reporting has not changed in unanticipated ways is somewhat tricky, given that it will be highly specific to the combination of inputs, and that we have not designed for a seam point where everything has its (initial) value for the simulation. We shouldn't attempt capture log files and assert they haven't changed without a lot of attention to making the reference data agnostic with respect to the current configuration of tests and build slaves. Doing a not-very-good job of that will become a source of irritation to developers - I avoid changing the text of warning messages, because if I do so I have to bump the regressiontests suite and deal with that pain. Tests within the source repo would work more smoothly, but I still don't think the result (log file doesn't change except as intended) is not worth the effort.
>
> We should be able to identify well-defined parts of the log file that we want to keep consistent and we want to avoid changing in an incidental or ad-hoc manner. I do think consistency is important, but it may well be that this will be considered more effort than what it's worth, I admit. I still think it's easily doable, though.

Doing so on a per-module basis might be easier - even if for no other reason than to be able to proof-read the effects of changes while reviewing.

> For a given fixed set of inputs (tpr and command line) everything between "Input Parameters:" and "Started mdrun on rank..." should stay unchanged.

It looks to me like there's a lot of output that also depends on the number of ranks, duty split, probably GPUs, rlist and nstlist, so there'd have to be work also to create the seams to call those with mock data, or to use pattern matching to work with anything that Jenkins does / might reasonably produce. The friction from the way regressiontests does exact matching of warning strings inhibits us from working on those messages, and that is something I think we should avoid.

> Additionally, given a fixed set of build parameters (same compilers, libraries) even the initial part of the log (from line 3 until "Input Parameters:") should stay the same with the exception of the "Built on:" and "Build OS/arch:" -- which are not hard at all to either omit printing (which BTW could be useful for package maintainers because some distros strive to make reproducible builds which are prevented by these two fields) for this very test build or remove before comparing.

Those are already parametrized by CMake variables that set src/buildinfo.h, so anybody can go ahead and do that. But now we'd have to update the source repo every time we bump the relevant Jenkins compiler, libraries, etc. (I'm not volunteering for that.) And if you make all the reported output fully reproducible, then that test is merely a rehash of the order of the printf statements, and friction against changing any of them. I don't see where the value is for that work.