GROMACS - Bug #1968

V-rescale thermostat and replica exchange broken

05/23/2016 02:42 PM - Berk Hess

Status:	Closed		
Priority:	High		
Assignee:			
Category:	mdrun		
Target version:	2016		
Affected version - extra info:	master	**Difficulty:**	uncategorized
Affected version:	2016		

Description

Commit 2d0247f6a60c810d883b0ed1ad42ef326e87dbf8 made the normal distributed random number for the V-rescale thermostat always use the same "random" number. The number if different for different ld-seeds.
This leads to the temperature being off. The magnitude of the error is inversely proportional to the size of the system. For large systems results might still be usable, since the thermostat still coupling off drift and the temperature is only off by a small amount. For small systems the results are useless. Check you temperature.

Associated revisions

Revision 3a9e67dd - 05/24/2016 12:36 AM - Berk Hess

Fix bug in v-rescale thermostat & replica exchange

Commit 2d0247f6 made the random normally distributed number for the
v-rescale thermostat constant for a run only depending on ld-seed,
and moved a reset for replica exchange inside the multi-exchange
branch.

Fixes #1968.

Change-Id: Iadc38ccadb99c6c1232756fd595843a01e5f3ce8

History

#1 - 05/23/2016 02:45 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #1968.
Uploader: Berk Hess (hess@kth.se)
Change-Id: Iadc38ccadb99c6c1232756fd595843a01e5f3ce8
Gerrit URL: https://gerrit.gromacs.org/5885

#2 - 05/23/2016 02:47 PM - Berk Hess

- Status changed from New to Fix uploaded

PS All our tests didn't catch this, since the reference data had to be updated for the new random number framework (and apparently no one checked their temperatures).

#3 - 05/23/2016 05:51 PM - Mark Abraham

- Subject changed from V-rescale thermostat broken to V-rescale thermostat and replica exchange broken

Standard replica exchange is also broken, because the restart got put into the bMultiEx == TRUE branch, and belongs one level up because normal exchange also needs random numbers that change every step.

Berk Hess wrote:

> PS All our tests didn't catch this, since the reference data had to be updated for the new random number framework (and apparently no one checked their temperatures).

I will see later this week if the ensemble quality test infrastructure I am building this week would have found it... but perhaps for this kind of case we should also validate the velocity distribution of some trivial systems in e.g. the upcoming post-submit configs (depending on speed of the tests). But these aren't likely to find the bugs we introduced in replica exchange...

Perhaps orthogonally, after this fix, all of the uses of ThreeFry2x64 call restart(step, someValue) some time after construction with the two key values, perhaps multiple times per MD step. The practice of the constructor calling seed(key0,key1) calling restart(0,0) seems rather risky in itself, and clearly it is prone to misuse such as this. Options include

- adding a struct field for Debug builds that permits us to assert that restart() got called again (but this doesn't address the case where e.g. we might forget to restart for each temperature-coupling group), and
- since we already think restart() is efficient / optimizable enough (the first block_ that is generated is perhaps provably unused) that we can call it an extra time per RNG per MD step, we could replace most calls to rng.restart() by a call to a four-parameter constructor in the scope of the relevant inner loop, or similar (and thus eliminate the bug-prone two-parameter constructors if/until there's a use case for them).

I think it is clear that the four-parameter constructor is least likely to be implemented or refactored wrongly e.g. gmx::ThreeFry2x64<N> rng(seed, type, step, atomOrGroup), and it seems plausible that compilers will do a good job of the code generation because the data members of ThreeFry2x64 are almost trivial. Thoughts?

#4 - 05/23/2016 07:02 PM - Erik Lindahl

I'm looking through all uses (ETA 1h) and I'd suggest we merge them all into one patch (since it might involve updating reference values).

Note that the default random generator (forgot the class name) is not using restarts, so we probably cannot just require it for all of them. In principle we could create a separate class, or add an optional field to the constructor to make it required, but I'm not sure how useful that will be for new code; this condition was a bit special since we had to clean up & edit a ton of old code, but for any new algorithm I would expect the mere debugging during writing an algorithm would find it. We can decide that post release-2016, though.

#5 - 05/23/2016 09:45 PM - Mark Abraham

Erik Lindahl wrote:

> I'm looking through all uses (ETA 1h) and I'd suggest we merge them all into one patch (since it might involve updating reference values).

Replica exchange code doesn't contribute to any regressiontests, so that consideration doesn't apply (we had a segfault in 5.1 that went undetected because the trivial end-to-end tests were not being run in practice). But I will agree if there's any other issues that do impact regressiontests.

> Note that the default random generator (forgot the class name) is not using restarts, so we probably cannot just require it for all of them. In principle we could create a separate class, or add an optional field to the constructor to make it required, but I'm not sure how useful that will be for new code; this condition was a bit special since we had to clean up & edit a ton of old code, but for any new algorithm I would expect the mere debugging during writing an algorithm would find it. We can decide that post release-2016, though.

I don't know what you mean by "not using restarts." If the user re-starts the MD step count, then the consequences (if any) are surely on them (but we could document that somewhere).

My objective is to avoid the anti-pattern of the partly-constructed object. If we make an object that can return a random number, then it should not be easy to return a random number that lacks obvious qualities (like independence between MD steps). Two-phase construction creates this kind of usability problem...

#6 - 05/24/2016 12:58 AM - Erik Lindahl

> I don't know what you mean by "not using restarts." If the user re-starts the MD step count, then the consequences (if any) are surely on them (but we could document that somewhere).

ThreeFry is used for all of our random numbers (through gmx::DefaultRandomEngine). Only some of them rely on restarts, since it is a perfectly normal RNG that will return a beautiful stream of random numbers even without a restart. Thus, we cannot just require that the default class must always call restart without complicating lots of code that has no idea about the restart concept, and often no natural counter.

> If we make an object that can return a random number, then it should not be easy to return a random number that lacks obvious qualities (like independence between MD steps). Two-phase construction creates this kind of usability problem...

One could certainly create a separate object that requires restart() to be called, or even have an object that requires restart to be used for every call. However, part of the problem is also that we use the calls in routines with hundreds of lines of code, we use quite different values (not just step) for the counter, and because we're worried about performance we wanted to avoid generating extra random numbers. I would recommend creating a derived class that requires restart() to be used, but to really find future bugs it's probably more important that we start testing for sequences returned from modules that in turn use random engines.

#7 - 05/24/2016 01:18 AM - Roland Schulz

Why did this not throw an exception? Or in other words why did the internal counter not overflow when the restart was not called?

#8 - 05/24/2016 12:23 PM - Mark Abraham

Erik Lindahl wrote:

> I don't know what you mean by "not using restarts." If the user re-starts the MD step count, then the consequences (if any) are surely on them (but we could document that somewhere).

> ThreeFry is used for all of our random numbers (through gmx::DefaultRandomEngine). Only some of them rely on restarts, since it is a perfectly normal RNG that will return a beautiful stream of random numbers even without a restart.

If we take the four-parameter constructor approach, then a use case that only has two natural key values can use a derived class that defaults the other parameters to whatever we like. But that does not mean that the base class should have only two parameters and always call restart(0,0) for all use cases, particularly if that means a user who has more natural key values can misuse them.

> Thus, we cannot just require that the default class must always call restart without complicating lots of code that has no idea about the restart concept, and often no natural counter.

Sure, but that's not necessary for either of my approaches. That code currently calls restart() via the call to seed() in the base-class constructor. Instead, it can call restart() in its own constructor, to suit that use case. Convenience of some clients shouldn't dictate creating opportunities for problems for others. Or we name it gmx::ToolsRandomEngine and gmx::MdrunRandomEngine to make the use case clear to the coder and reviewer.

> If we make an object that can return a random number, then it should not be easy to return a random number that lacks obvious qualities (like independence between MD steps). Two-phase construction creates this kind of usability problem...

> One could certainly create a separate object that requires restart() to be called, or even have an object that requires restart to be used for every call. However, part of the problem is also that we use the calls in routines with hundreds of lines of code, we use quite different values (not just step) for the counter, and because we're worried about performance we wanted to avoid generating extra random numbers. I would recommend creating a derived class that requires restart() to be used,

Plausible, but I think it is slightly superior to have the special case be the one that sets parameters to default values. Also, the size of client code that calls the RNGs doesn't have anything to with this decision. :-)

> but to really find future bugs it's probably more important that we start testing for sequences returned from modules that in turn use random engines.

That also, which would have found these. But patches that add tests get ignored by people to the point that I will propose that anybody adding a test case that passes Jenkins can submit to master after a week or two, unless comment has arisen. For example, https://gerrit.gromacs.org/#/c/5435/10 and its parent. I do not want people's hands being tied by the same standards we strive to have for review for changes to high-value code.

#9 - 05/24/2016 01:44 PM - Mark Abraham

Roland Schulz wrote:

> Why did this not throw an exception? Or in other words why did the internal counter not overflow when the restart was not called?

restart gets called with default counter values of (0,0) by the base ThreeFry constructor, which produces an rng that you can misuse by calling it every MD step without specifying the step index as a counter. It's not a running-out-of number issue, but rather an invalid initialization issue.

Since we know that mdrun will always have at least three values to use (user seed key, algorithm identifier key, MD step), and often a fourth (atom or group index), then we should have a type that makes using four explicit values the only legal construction. Since there's also reasonable uses for a two-parameter constructor, we can have another type. I'll make a patch that makes this concrete.

#10 - 05/25/2016 05:27 PM - Berk Hess

- Status changed from Fix uploaded to Resolved

Applied in changeset 3a9e67dd55949e7daaa666e5bc4de95bcfece2c4.

#11 - 05/25/2016 10:48 PM - Erik Lindahl

Roland: There was nothing incorrect with the initialization of the RNG; the restart() is an optional feature that allows to go to some arbitrary index directly, not seeding.

Mark: Please don't modify the constructor to take those arguments, but provide a separate type (or constructor with an optional enum setting) that makes it possible to create an RNG that in a debug build will throw if restart has not been called before drawing numbers.

The whole architecture of ThreeFry is that construction is allowed to be slow (so we should not introduce changes that prevents extracting constructions from the routines where the step/atom is known), while the restart() method must be extremely fast.

#12 - 05/25/2016 10:48 PM - Erik Lindahl

- Status changed from Resolved to Closed