GROMACS - Feature #2224

Proposed feature: conditional stop

08/02/2017 07:35 PM - Vedran Miletic

Status:	New
Priority:	Normal
Assignee:	Vedran Miletic
Category:	mdrun
Target version:	2020
Difficulty:	simple

Description

(I planned to present this idea at the dev teleconf today, but couldn't due to Pexip not liking my setup. Apologies for that.)

In the recent months I have been independently asked twice whether GROMACS allows stopping a simulation when a certain condition is satisified, so I'm assuming there might be some interest in such a feature. The most obvious use case is to stop the simulation right before potential goes to infinity and save trajectories, but more generally, one can stop when two residues get too close to or too far from each other.

I have a prototype implementation, which I can post, but I would like to gauge whether people here consider such a feature useful or interesting. Rough sketch of the implementation is as follows.

mdp changes

The interface is designed in style of pull/rotational groups. A user specifies, nstepscondstop, n groups (indexed in the ndx file) and m conditions (each condition containing two groups, type, distance criterion, and distance threshold).

Types:

- minimum distance between atom pairs in the groups
- maximum distance between atom pairs in the groups

Distance criteria:

- is larger than the distance threshold
- is smaller than the distance threshold

tpxio/readir changes are very similar to pull/rotational groups.

mdrun changes

Added one if in the while loop that each nstepscondstop calls a function to check whether any of the user specified criteria is satisfied. I also had to change trajectory_writing.cpp to enable writing trajectory if simulation stops early (i.e. the current step number is not equal to the number of simulation steps requested).

seeking feedback

- Do you believe this feature would be useful?
- Any implementation suggestions?
- What other condition types could one include?

Related issues:

Related to GROMACS - Task #1781: re-design benchmarking functionality	**Accepted**
Related to GROMACS - Bug #2136: mdrun -noconfout checkpointing issue	**Closed**
Related to GROMACS - Task #1971: Removing buggy features vs. keeping workflows	**New**
Related to GROMACS - Feature #2585: Infrastructure supporting external API	**Resolved**

Associated revisions

Revision 806b4574 - 10/14/2018 10:00 AM - Eric Irrgang

Allow API access to simulation signals.

Refs #2620 for gmxapi milestone 10

Provide API client access to issue stop signals through simulation
Session resources. Stop conditions will be discovered by the simulation
when checked from within the MD loop.

Also relates to #2224 and various other works in progress.

Change-Id: I2815733d3d18bc04685dacbd8f6a3ba56e55b783

History

#1 - 08/02/2017 09:50 PM - Erik Lindahl

Hi,

I'm somewhat hesitant :-)

The main reason is that we have a history of repeatedly making the simple art of starting/stopping simulations way too complicated, which has led to bugs.

For instance:

1) What should be done when one of many simulations in a multi-run triggers the criterion?

2) If you want to stop the simulation when a distance criterion is reached, you likely want the frame where it happens stored, but if you want to stop the simulation after reaching infinite energy, you rather want the previous frame. Suddenly you might need to introduce extra code to decide what frame to write. Not rocket science, but it adds complexity.

3) Should the user be allowed to continue such simulations (i.e., using the checkpointing mechanism)? In that case, what should happen with the extra frame?

4) I can imagine a common condition would be to check e.g. if any molecule in a class is close to some other class. For large parallel simulations that can become very costly, so unless the checks are parallelized they could come to dominate the runtime. In general, just calculating arbitrary distances on the fly is not trivial for domain decomposition, since we need to check what coordinates should be communicated first.

So, it's not a hard "no", but for once I'm trying to consider the potential issues before we add code - it might not be quite trivial, and could add complexity.

#2 - 08/02/2017 09:59 PM - Roland Schulz

- Related to Task #1781: re-design benchmarking functionality added

#3 - 08/02/2017 09:59 PM - Roland Schulz

- Related to Bug #2136: mdrun -noconfout checkpointing issue added

#4 - 08/02/2017 10:00 PM - Roland Schulz

- Related to Task #1971: Removing buggy features vs. keeping workflows added

#5 - 08/02/2017 10:02 PM - Roland Schulz

I'm inclined to say: Yes, but please help first to clean-up the related code. I added the related issues I'm aware of. I think if the sum of complexity stays the same (reduction of mess ~= new complexity), I don't see an issue.

#6 - 08/02/2017 10:05 PM - Vedran Miletic

Erik Lindahl wrote:

> Hi,
>
> I'm somewhat hesitant :-)
>
> The main reason is that we have a history of repeatedly making the simple art of starting/stopping simulations way too complicated, which has led to bugs.

It's obvious from the moment one studies the code and sees bLastStep as a somewhat separate thing from (step - ir->nsteps) == 0. :-)

> For instance:
>
> 1) What should be done when one of many simulations in a multi-run triggers the criterion?

Good question, haven't thought of those. It's either stop all of them or stop just one of them and let the others run, but that would complicate things in case of replica exchange. I would have to think about that.

> 2) If you want to stop the simulation when a distance criterion is reached, you likely want the frame where it happens stored, but if you want to stop the simulation after reaching infinite energy, you rather want the previous frame. Suddenly you might need to introduce extra code to decide what frame to write. Not rocket science, but it adds complexity.

Due to precisely the problem you describe, I only implemented the distance criterion, not the energy. User should think what distance causes energy to become too large and adjust the distance threshold accordingly.

> 3) Should the user be allowed to continue such simulations (i.e., using the checkpointing mechanism)? In that case, what should happen with the extra frame?

Sure, why not? Though the user has to understand that, if after nstepscondstop steps the criterion is still satisfied, the simulation will stop again. A particular corner case is whether we should check at step 0, and I lean towards saying yes -- in that case restarting from checkpoint will make the run stop immediately.

> 4) I can imagine a common condition would be to check e.g. if any molecule in a class is close to some other class. For large parallel simulations that can become very costly, so unless the checks are parallelized they could come to dominate the runtime. In general, just calculating arbitrary distances on the fly is not trivial for domain decomposition, since we need to check what coordinates should be communicated first.

Exactly :-) I have considered in advance that GROMACS has to scale to exascale and I have such functionality *mostly* ready, including the selection code (i.e. I know which coordinates are checked and should be communicated), decision which process checks which condition, and collecting the results of the checks. I haven't implemented yet the broadcast of coordinates which should be used for checking, but, assuming I can reuse the global state (which is, according to my testing, allocated but zeroed on all processes except the master), that should be doable. I stopped at the point "hmmm, DD doesn't automatically broadcast global state, maybe we should do the checks only at the master MPI process after all, let's see how other devs look at the issue".

> So, it's not a hard "no", but for once I'm trying to consider the potential issues before we add code - it might not be quite trivial, and could add complexity.

Thank you so much for your feedback.

#7 - 05/16/2018 06:46 PM - Vedran Miletic

Hello everyone, after discussing this with Mark last month in Huenfeld I came up with a proposal that abstracts the bLastStep change: https://gerrit.gromacs.org/#/c/7856/

Apparently the Condition and Trigger classes are not there yet, but roughly I would go for something like:

```
class Condition
{
    virtual gmx_bool check_if_sastisfied() = 0;
};

class DistanceCondition : Condition
{
...
};

class Trigger
{
    virtual gmx_bool check_conditions();
    virtual void execute_actions();
    private:
        std::vector<Condition> m_conditions;
        std::vector<Action>    m_actions;
};
```

I'm flexible on the Trigger class API, but I would expect it to execute condition checking and actions at some point in the MD loop (or more points, depending on the trigger type, think EarlyTrigger, LateTrigger, PrePullTrigger, PostPullTrigger etc.).

Alternatively, one could have Condition class that executes action directly, but this complicates the multiple satisfied conditions executing one or actions case.

Furthermore, I would appreciate:

- feedback on the code that I posted on Gerrit, and
- some concrete suggestion on the cleanups that you would like to see this effort achieve. (I skimmed through the discussions in the bugs linked, but some pointers would help.)

On our side, if we could get the distance condition with stop action in GROMACS that would be good, and if we could get distance condition with flexible action that would be excellent (another group at HITS would is interested in this use case). The initial prototype code with distance condition stop action is here: https://github.com/HITS-MBM/gromacs-developments

#8 - 05/24/2018 02:35 PM - Eric Irrgang

I have some preliminary experimentation in another project at https://github.com/kassonlab/gromacs-gmxapi/pull/15/files that is unfortunately cluttered with other things and rather distantly forked from master, so I'll summarize.

- Ownership of the SimulationSignals container is moved up to Mdrunner.
- Access to the container is mediated by a new class to compartmentalize this sort of interaction and keep it out of an unconstrained "Mdrunner" interface, but for lack of a better idea, I stuck it in an object representing the "MD context" and added it to the integrator_t signature.
- Additional code is intended to abstract out direct access, and provide a mechanism to bind a "signal provider" to the resource owned by the runner

A module can then be bound to the signaler before being passed to the MD engine and provide a signal based on whatever logic it wants to employ.

I haven't tried to address any issue of a generic module to provide a canned set of configurable conditions because at this point I'm thinking in terms of the stop condition being a specific case of a more general boolean data event and I am working on more generic call-back infrastructure. Are the hooks provided by the current MDModules interface enough to get the conditional code evaluated in the right part of the MD loop? It would probably be better in the long run to add another type of module that can just consume trajectory information for analysis purposes, but I currently sneak extra code into the calculateForces call of an IForceProvider.

#9 - 07/30/2018 05:49 PM - Eric Irrgang

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

#10 - 08/21/2018 08:44 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #2224.
Uploader: M. Eric Irrgang (ericirrgang@gmail.com)
Change-Id: gromacs~master~I2815733d3d18bc04685dacbd8f6a3ba56e55b783
Gerrit URL: https://gerrit.gromacs.org/8216

#11 - 09/19/2018 03:06 PM - Mark Abraham

- Target version changed from 2019 to 2020