GROMACS - Task #1768

decide future of command-line options vs env vars

07/06/2015 10:33 AM - Mark Abraham

Status:	New	
Priority:	Normal	
Assignee:		
Category:	mdrun	
Target version:	future	
Difficulty:	uncategorized	

Description

At https://gerrit.gromacs.org/#/c/4838/, Erik said

> Isn't this bound to become another example of cases where we introduce too many ways of affecting behavior, and then it leads to bugs?
> It does not strike me as a wonderful balance to first introduce efficiency checks, we make them fatal errors just to make sure users cannot avoid them, but that causes problems, and then we need an environment variable to override the very behavior we introduced...

Later, Mark said

> agree we want to avoid dual control (but sometimes we are stuck with things like OMP_NUM_THREADS, so I don't think "only command line options" is a fully workable strategy). I would like to move in the direction of command-line options for things users might want / should use, and env vars for thigns we do not expect users to ever use. That should mean no duplication, and minimizes mdrun -h content.
> I understand there are ways to construct e.g. mpirun gmx_mpi mdrun -gpu_id differently on different nodes so the duality of GMX_GPU_ID is probably not needed, etc.

Then Erik said

> For the future, apart from standardized defines (such as for OpenMP - **IF** we stick with OpenMP) I think we should move completely to command-line options, but maybe have a bunch of them hidden.
> The problem with environment variables is that they might be set in the shell, so Gromacs execution will suddenly depend on settings that are not visible e.g. in the batch script. Even if we want to use environment variables, we should stop checking them in individual routines, but have one settings object that we initialize early and then propagate (so it is possible to check **ALL** settings in a single place).

Mark agrees with the desire for a central logically const user-choices object that keeps a description of the initial conditions, howsoever described. It's also obviously useful in testing. There would be no more calls to getenv (whether or not we keep the use of env vars). Such an object should certainly report the setting of any developer-level option. Doing so as an mdrun command line does make it easier to reproduce the run conditions.

Any further thoughts?

History

#1 - 07/09/2015 06:54 PM - Szilárd Páll

Mark Abraham wrote:

> agree we want to avoid dual control (but sometimes we are stuck with things like OMP_NUM_THREADS, so I don't think "only command line options" is a fully workable strategy).

+1

> I would like to move in the direction of command-line options for things users might want / should use, and env vars for thigns we do not expect users to ever use. That should mean no duplication, and minimizes mdrun -h content.

I don't see why is this an "mdrun -h" issue; the help page simply needs to be redone, sane *nix command line tools simply don't dump hundreds of lines of help output - essentially the entire man page.

Also, there are valid user use-cases of env. var. - see below.

> I understand there are ways to construct e.g. mpirun gmx_mpi mdrun -gpu_id differently on different nodes so the duality of GMX_GPU_ID is probably not needed, etc.

How?

> The problem with environment variables is that they might be set in the shell, so Gromacs execution will suddenly depend on settings that are not visible e.g. in the batch script. Even if we want to use environment variables, we should stop checking them in individual routines, but have one settings object that we initialize early and then propagate (so it is possible to check **ALL** settings in a single place).

This is just a reporting issue, I think. It's not difficult to "hide" command line options either - it's more a matter of what is one used to.

```
run.sh:
#/bin/bash
aprun -n 666 mdrun $MY_OPTS

$ MY_OPTS="-ntomp 1" sbatch ./run.sh
```

I actually do the above to not have to edit my submit scripts just to append an extra option and I simply rely on keeping the log file which contains the full command line. So we should simplyreport the env vars too (like MPI distros do if one requests it).

> Mark agrees with the desire for a central logically const user-choices object that keeps a description of the initial conditions, howsoever described. It's also obviously useful in testing. There would be no more calls to getenv (whether or not we keep the use of env vars). Such an object should certainly report the setting of any developer-level option. Doing so as an mdrun command line does make it easier to reproduce the run conditions.

Fully agree.

I see this more of an issue of complex and scattered option handling code, the lack of centralized, option handling/parsing/validation, and poor logging/reporting facilities (and the lack of reporting of env var set options) rather than a problem with the environment variables themselves. Nor do I see it as an issue that some options, when warranted, are exposed through both command line and environment variables (can simply read both OMP_NUM_THREADS and -ntomp and with 3 lines of code validate them).

Valid use-cases are:

- shell (session) persistent options, e.g. GMX_MAXBACKUP (or the removed GMX_MAX_THREADS)
- per-process options, e.g. GMX_PME_NUM_THREADS, GMX_GPU_ID
- libgromacs used by another program

Therefore, I would rather prefer to i) centralize and restructure the parsing of all options including command line, mdp, and env var, and do better validation ii) do better loggin/reporting.

#2 - 07/09/2015 11:40 PM - Mark Abraham

Szilárd Páll wrote:

> Mark Abraham wrote:
>
> > agree we want to avoid dual control (but sometimes we are stuck with things like OMP_NUM_THREADS, so I don't think "only command line options" is a fully workable strategy).
>
> +1
>
> > I would like to move in the direction of command-line options for things users might want / should use, and env vars for thigns we do not expect users to ever use. That should mean no duplication, and minimizes mdrun -h content.
>
> I don't see why is this an "mdrun -h" issue; the help page simply needs to be redone, sane *nix command line tools simply don't dump hundreds of lines of help output - essentially the entire man page.

Do compare gmx mdrun -h with mpirun -h and nvidia-smi -h ;-) With -h, they're showing much less than their man pages. Our -h is bigger, but not by crazy amounts.

We need fewer options(=features) overall, and to choose the most useful subset to show. e.g. there's DLB control flags that feel like no user should ever use them. e.g. docs for IMD flags make sense in the user guide, now that we have one.

We need to move even more of the "how to run mdrun fast" off to the mdrun performance section in the new user guide. I plan to move some more stuff there next week, for inclusion in 5.1

Big reductions in the number of mdrun options are available when we move the table input to grompp, and lots of the specialized output files to TNG fields, but people have to want this and to agree to reviewing code within a few weeks of it appearing on Gerrit, or such inglorious work won't happen at all.

Combined, I think that means we don't need to invest in choosing a subset for mdrun -h, and then implementing the division.

> Also, there are valid user use-cases of env. var. - see below.
>
> > I understand there are ways to construct e.g. mpirun gmx_mpi mdrun -gpu_id differently on different nodes so the duality of GMX_GPU_ID is probably not needed, etc.
>
> How?

Along the lines of https://www.open-mpi.org/faq/?category=running#mpmd-run. I'm assuming we can pass command line options on a per-process basis as well; the example doesn't show that. I seem to recall that it did work when I tried something like that when I wanted to run different numbers of ranks per node on a "PME node" while experimenting on beskow. OTOH constructing such inputs is perhaps more work than setting GMX_GPU_ID different in different places, and if we can construct a unified input-management-object then the duality of ways to set the value is only a localized problem there, and thus manageable.

#3 - 07/10/2015 10:40 PM - Mark Abraham

Szilárd Páll wrote:

> I see this more of an issue of complex and scattered option handling code, the lack of centralized, option handling/parsing/validation, and poor logging/reporting facilities (and the lack of reporting of env var set options) rather than a problem with the environment variables themselves. Nor do I see it as an issue that some options, when warranted, are exposed through both command line and environment variables (can simply read both OMP_NUM_THREADS and -ntomp and with 3 lines of code validate them).
>
> Valid use-cases are:
>
> - shell (session) persistent options, e.g. GMX_MAXBACKUP (or the removed GMX_MAX_THREADS)
> - per-process options, e.g. GMX_PME_NUM_THREADS, GMX_GPU_ID
> - libgromacs used by another program
>
> Therefore, I would rather prefer to i) centralize and restructure the parsing of all options including command line, mdp, and env var, and do better validation ii) do better loggin/reporting.

The question of how libgromacs ought to be used by a third party code is a good one to remember. I think the traditional answer is "configure it using the same API as its main client." I think this leads us back to "have a central settings object and call its methods." I think that means **mdrun** might respond to environment variables / whatever we decide, but **libgromacs** responds to the settings manager.

One of the criticisms of OpenMP as a parallelism solution is that it is non-composable - one OpenMP code has a hard time calling another. IIRC TBB is designed with composability in mind. But if/when we are seriously designing libgromacs for some other application to call, then we might need the ability to expose our detection API (e.g. wrapped from hwloc) so that the client of libgromacs can give reasonable instructions about how much hardware libgromacs can use. Separate modules for managing detection, gathering user settings, constructing defaults given all the inputs, and managing auto-tuning are needed in order to do a decent job of all the things.

#4 - 07/12/2015 10:01 PM - Szilárd Páll

Mark Abraham wrote:

> Szilárd Páll wrote:
>
> > Mark Abraham wrote:
> >
> > > agree we want to avoid dual control (but sometimes we are stuck with things like OMP_NUM_THREADS, so I don't think "only command line options" is a fully workable strategy).
>
> +1
>
> I would like to move in the direction of command-line options for things users might want / should use, and env vars for thigns we do not expect users to ever use. That should mean no duplication, and minimizes mdrun -h content.

> I don't see why is this an "mdrun -h" issue; the help page simply needs to be redone, sane *nix command line tools simply don't dump hundreds of lines of help output - essentially the entire man page.

> Do compare gmx mdrun -h with mpirun -h and nvidia-smi -h ;-) With -h, they're showing much less than their man pages. Our -h is bigger, but not by crazy amounts.

I think I was rushed and the message did not end up clear enough. What I meant to say is that the issue with "mdrun -h" output is not directly related to the question of environment variables the -h output IMO needs a complete redesign (which should inherently make it short). Incidentally, getting rid of env. var. will make the -h shorter, but most (if not all) of the this info should be in the man and not in the -h output to begin with.

I didn't mean to suggest that -h page should be long or the same as the man. On the contrary, the current style makes the -h output not very useful. As expressed previously (see #1687 note 8), I think GROMACS would benefit a help/man style that's close to what established OSS command line programs use. So, I do agree that -h should be shorter, but it's not size that matters, but the information density and formatting, and that should be different for the two.

Depending on what tools you pick as example you'll get a varying -h/man size ratios and varying amount of info in each depending on the complexity of the tool and its command line interface:

- mpirun: 222/1049
- nvidia-smi: 179/1775
- git clone: 28/311
- awk: 36/1695
- gcc: 60/16074
- gmx mdrun: 465/535

One can argue whether the help page should be super-brief or a thorough summary, but in any case it has a different scope and volume than the man page in all examples above except mdrun.

> We need fewer options(=features) overall, and to choose the most useful subset to show. e.g. there's DLB control flags that feel like no user should ever use them. e.g. docs for IMD flags make sense in the user guide, now that we have one.
>
> We need to move even more of the "how to run mdrun fast" off to the mdrun performance section in the new user guide. I plan to move some more stuff there next week, for inclusion in 5.1
>
> Big reductions in the number of mdrun options are available when we move the table input to grompp, and lots of the specialized output files to TNG fields, but people have to want this and to agree to reviewing code within a few weeks of it appearing on Gerrit, or such inglorious work won't happen at all.
>
> Combined, I think that means we don't need to invest in choosing a subset for mdrun -h, and then implementing the division.

I don't mean to question whether a specific option or feature should stay or go. However, I think the man page is a command line tool's manual and it's best if contains a fairly complete documentation of it's use. I understand that the latter overlaps in scope with the user guide and the line on what should be included where needs to be drawn. However, unless the man page gets severely reduced and users sent online to the guide (which IMO should be a decision of the community) I do not think options or description of important use-cases (e.g. see man git-rebase with ACII illustration) should be omitted from the man page just because those are described in the user guide.

I now realize that the man vs -h vs user guide discussion is off-topic, but let me still post it here and will move it elsewhere if there is a better context.

> Also, there are valid user use-cases of env. var. - see below.
>
> > I understand there are ways to construct e.g. mpirun gmx_mpi mdrun -gpu_id differently on different nodes so the duality of GMX_GPU_ID is probably not needed, etc.
>
> How?

> Along the lines of https://www.open-mpi.org/faq/?category=running#mpmd-run. I'm assuming we can pass command line options on a per-process basis as well; the example doesn't show that. I seem to recall that it did work when I tried something like that when I wanted to run different numbers of ranks per node on a "PME node" while experimenting on beskow.

Interesting, I did not know about that syntax. Still, it does not seem too elegant nor does it seem to allow to implement conditionality that depends on the nodes allocated or the order of the nodes.

> OTOH constructing such inputs is perhaps more work than setting GMX_GPU_ID different in different places, and if we can construct a unified input-management-object then the duality of ways to set the value is only a localized problem there, and thus manageable.

I agree.

#5 - 07/12/2015 10:10 PM - Szilárd Páll

Mark Abraham wrote:

> The question of how libgromacs ought to be used by a third party code is a good one to remember. I think the traditional answer is "configure it using the same API as its main client." I think this leads us back to "have a central settings object and call its methods." I think that means **mdrun** might respond to environment variables / whatever we decide, but **libgromacs** responds to the settings manager.

So you suggest that libgromacs would by design ignore environment variables? Isn't that somewhat confusing? If a user simply wants to implement a sequence of gmx tool commands via a python wrapper rather than bash script, it seems counter-intuitive to have GMX_GPU_ID respected in one case, but ignored in the other.

> One of the criticisms of OpenMP as a parallelism solution is that it is non-composable - one OpenMP code has a hard time calling another. IIRC TBB is designed with composability in mind. But if/when we are seriously designing libgromacs for some other application to call, then we might need the ability to expose our detection API (e.g. wrapped from hwloc) so that the client of libgromacs can give reasonable instructions about how much hardware libgromacs can use. Separate modules for managing detection, gathering user settings, constructing defaults given all the inputs, and managing auto-tuning are needed in order to do a decent job of all the things.

Good point. I sense a difference in philosophy here rather than a case of bad vs good design, but perhaps in practice it's often a mix of the two.

#6 - 11/03/2016 05:51 PM - Mark Abraham

More generally, we should avoid having more than one way to do things. Some aspects of #1942 seem likely to have been caused by the way mdrun -nsteps interacts with mdrun -cpi and multiple pieces of code treating the inputrec as a read-write object. (I can't see a bug in that so far, but that code is hard to follow.)

Eliminating gmx mdrun -nsteps seems like a no brainer, but will be highly unpopular with people with affected workflows, including some developers. Resolving #1781 with new support for performance benchmarking will eliminate much of this. I think it is clear that we should have gmx tpbconv -extend as the way to extend simulations, not hacking with gmx mdrun -nsteps. We need gmx grompp -t state.cpt available to modify .mdp parameters (whether to change output frequency, or physics or whatever). One option is to deprecate gmx mdrun -nsteps in 2017 release.