There are a number of things that I think we should deprecate for 5.0. The reasons vary:

- if we've known for years they've been broken and we haven't bothered to fix them, we may as well throw them out and re-implement if/when we care again
- if they don't scale well
- if their author has abandoned them

People still have the option of using old code versions, or doing the work to update/re-implement. We cannot afford to deliver a high-performance, high-quality, kitchen-sink code base. In particular, we can't afford to create all the test cases that have a chance of showing whether these things still work now, never mind whether they continue to work as we refactor key data structures. Neither do we want to invest time into keeping code compiling, or compromising the design of data structures to support code that is broken (and we maybe don't even know what is broken under what conditions).

Here's what Mark thinks should go, and some detail on why. Edit feedback so far looks like this.

1) Particle decomposition
Doesn't support checkpointing. Doesn't scale because it does not balance load. Removing it will simplify a lot of conditionally executed code paths. The main caveat is whether there are important algorithms whose implementations still require it (see below). Erik is all for killing PD

2) Fancy distance restraints
Peter and David are keen to retain most/all of the restraint machinery
a) Ensemble-averaged distance restraints
Has been disabled since 2009. If/when it did work before that, it required PD. Its implementation and that of REMD seem to compromise each other a bit. Its implementation doesn't look like it would work with multiple ranks per simulation.

b) Time-averaged distance restraints
Disabled with DD since 2008. I can't see any reason why it should require PD if simple distance restraints work with DD. If the issue is that DD only works with distance restraints that happen to be inside the cutoff (so that the positions are available), then it should be not too painful to add a mechanism to keep track of the handful of atom pairs involved. However, the normal DD mechanism for communicating positions is probably unsuited to communicating between domains on arbitrary processors, so we'd have to add a special round of communication. That'd scale poorly, but that's the choice of the person using such restraints. But I'd prefer to remove time-averaged distance restraints than to keep PD just to support it.

c) Writing of pair distance and/or time-averaged pair distance to energy files
Seems more like debugging output than anything you'd use in post-processing (and whatever it does support could be implemented via a trajectory-analysis tool).

3) Fancy orientation restraints
Pretty much as above. Stuff's been disabled with DD for years.
As above, Peter and David are keen to retain

4) Current polarization support?
Is anything here known to be useful enough that we'd want to keep it? David and Michael think Drude (a.k.a. shell) model should definitely be retained, but perhaps via re-implementation.

5) QM support
No real beef here, but unless someone is willing to construct at least some simple test cases, we won't know whether it is broken. My experience of asking for test cases from the authors of specialist features has been mixed :-) Existing QMMM to be removed and re-implemented. WIP in qmmm branch.

6) Energy minimizers
I’d lose L-BFGS if it causes any pain, since its implementation has never worked in parallel or with constraints. *Useful for deep minimization before NMA. Consult with Erik if there are issues.*

7) *Generalized Born*
   It worked in 4.5, but various aspects of implicit solvation got broken during the 4.6 cycle, and I see no pressure to fix any of it. The method is anachronistic. Its implementation will not benefit from anything else we are doing. Its existence leads people to assume that all the other things GROMACS can do work with GB (e.g. free-energy calculations), and I suspect that all that ever worked was the stuff that was wanted for the Larsson and Bjelkmar publications. Preference from David and Michael to retain to support niche market. Erik torn.

8) *MTTK pressure coupling*
   Per Michael’s intentions. *Retain, but remove support for topologies with constraints.*

9) *Reaction-field-nec*
   Was only ever a backward-compatibility feature?

10) *Encad-shift*
    We deprecated its force field, do we need its shift?

11) *mdrun -ionize*
    *David to remove*

12) *GCT*
    *Believed broken. David to remove.*

13) *mdrun -seppot*
    *NEW* Looks to Michael, Teemu and Mark like debugging output, rather than typical .log file output (see [1294](#1294)). If so, we should either write output only in response to -debug, or not write the output.

Please let your thoughts be known - but we can’t take more than a few weeks over this. Code has to change! In particular, there are things above that I am unsure about, and I’ll update the main list as I learn from you.

If you disagree with wanting to remove something and expect some debate, please fork a new Redmine, note your disagreement here, and link the two issues. Otherwise we might drown in the cross-discussion :-(

<table>
<thead>
<tr>
<th>Subtasks:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug # 1429: shell code issues (broken with DD+grid; broken with Verlet scheme)</td>
</tr>
<tr>
<td>Feature # 1500: Post-5.0 feature clean-up plan</td>
</tr>
<tr>
<td>Feature # 1054: Improve implicit solvent parallelization</td>
</tr>
<tr>
<td>Bug # 1117: ensemble-averaged distance restraints is probably broken</td>
</tr>
<tr>
<td>Task # 1424: remove mdrun -testverlet option</td>
</tr>
<tr>
<td>Feature # 1665: improve free energy non-bonded kernel performance</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Related issues:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Related to Gromacs - Bug #1416: 2-D Ewald summation and DD is broken</td>
</tr>
<tr>
<td>Related to Gromacs - Feature #753: Use of GB in parallel and/or with all-vs-a...</td>
</tr>
<tr>
<td>Related to Gromacs - Bug #973: Energies from implicit solvent calculations</td>
</tr>
</tbody>
</table>

**Associated revisions**

**Revision 8e2028cd - 09/18/2013 05:57 PM - Mark Abraham**

Remove forcefield-scan feature

Refs #1292

Change-Id: i05d8bdec45665d5dc3b72b72d312240abbd2983c7

**Revision 864de6d4 - 09/26/2013 10:30 AM - Mark Abraham**

Remove General Coupling Theory stuff

Also update mdp_opt.html and src/contrib to no longer refer to defunct xmdrun machinery. This commit will prevent compilation of
src/contrib/prfn.c, but that is slated for removal in I842a92ec41.

Refs #1292
Change-Id: I8e06d3395787545ae5aba5334acfc57b7d8683c3

Revision 4282773f - 09/26/2013 02:40 PM - Mark Abraham
Remove mdrun -ionize feature
Refs #1292
Change-Id: lc3b8bf33265304f903tcba749fe41d4f00386d1d

Revision 30113ccc - 10/18/2013 12:11 AM - Mark Abraham
Move mdrun trajectory writing into wrapper function
Refs #1292, #1193, #1137
Change-Id: I3f19e0995ff7ab465184d5bab8c2683260a1853

Revision 8df8c14d - 10/28/2013 06:36 PM - Mark Abraham
Create fileio module

This patch contains only code motion. There are no functional code changes. Moves lots of I/O code into src/gromacs/fileio. This means lots of changes to avoid having everything as a compile-time dependency of everything else because everything is pulled in via typedefs.h, etc. Note that src/gromacs/legacyheaders/filenm.h and src/gromacs/legacyheaders/types/filenm.h have been consolidated into src/gromacs/fileio/filenm.h.

I/O code in files named stat[ch] now lives in various new files in fileio.

Files outside of the module now #include its header files in a proper way, e.g. #include "../fileio/filenm.h" or "gromacs/fileio/filenm.h" according to whether they are an installed header, or not. Files within the module are blessed and do not need that qualifier.

This module installs most of its headers (because they're almost all inter-dependent; gmxfio_int.h is not installed because it is only useful internally, vmdio.h is not installed because it relies on a header from src/external)

Files in new module

- conform to preferred include-guard format.
- have up-to-date copyright headers thanks to Teemu's automatic script
- that are installed headers refer to other GROMACS include files via relative paths

Moves mdrun trajectory writing into wrapper function.

Removes small pieces of I/O code that was hiding behind "#if 0".

Some pieces of I/O code specific to the gmxpreprocess module have remained there.
Moved a cppcheck suppression to follow matio.cpp to its new home.

Minor fix to xdrf.h logic, since it is now the subject of a CMake test.

Refs #1292, #1193, #1137

Change-Id: i820036298d574966d596bab9e258ed8676e359184

Revision 20fa2cda - 02/20/2014 07:03 PM - Mark Abraham

Remove particle decomposition

The paths that are eliminated are those for which MD_PARTDECOMP was needed, those triggered by PARTDECOMP, and any for which PAR && !cr->dd.

Reviewers, please note

- that for the purposes of this patch, OpenMP is not a form of parallelism,
- the definition of DOMAINDECOMP in commrec.h,
- that TPI and NM can run in parallel, but use neither PD nor DD,
- multi-simulations run in parallel but need not use DD, so
- we still need both PAR and DOMAINDECOMP, and
- I have generally left the indenting alone to make for easy review, but we will uncrustify before merging

Summary of changes in this patch:

Removed

- code triggered by MD_PARTDECOMP
- code triggered by PARTDECOMP
- anything decomposition-related with "pd" in the name (but the name clash with x86 vector intrinsics is unfortunate)
- t_mdatoms field called start (DD does atom number remapping)
  Note that bPDvsitecomm was never set anyway!
- manual section

Renamed two functions with "pd" in their name changed to "serial" because they are still needed there (and moved them?)

Deleted files, functions and function parameters that only supported PD code paths.

Cleaned up

- use of DOMAINDECOMP
- a bit of Generalized Born code that should have been static or was unused
- bcast_state machinery can now be much simpler; still does the right thing, but does it further behind the scenes and easier to understand
- made explicit the assumption of replica exchange that the integrator is dynamical, which makes for some minor simplifications

Refs #1292

Change-Id: If029f16e6b4b06d58d465afe072a3cede6481479e
Revision d17542a3 - 03/17/2014 06:46 PM - Mark Abraham

House-keeping for MTTK

Leapfrog + MTTK silently produced a .tpr that is probably broken; certainly the documentation only states support for Velocity-verlet integrators.

Added note about deprecation and planned removal of MTTK + constraints.

Refs #1292

Change-Id: lec2cf0dd866242735ce04a954e585b24616e701

Revision 8b550151 - 08/17/2014 08:24 AM - Mark Abraham

Remove mdrun -seppot

This output of rank-local energies is not needed for any routine use, and isn't much use in mdrun -debug either. Removing it simplifies a few code paths, including removing some dependencies on writing sane things to the log file. Eliminated some now-unused parameters.

Refs #1292

Change-Id: i7628b7862144ae2478b6f42a6818915e3a22fe3

Revision 7c6185b4 - 12/13/2014 11:41 AM - Mark Abraham

Remove heuristic group-scheme neighbour-list updates

-1 == nstlist used to trigger the use of an algorithm that did neighbour searching only when particles had moved far enough that there might be a need to update the list. This supported obtaining better energy conservation with the group cut-off scheme. It has been superseded by the Verlet cut-off scheme.

Part of #1292

Change-Id: i074790c1c5670e874b8b587a1d46e62e2edc8961

History

#1 - 06/26/2013 04:11 PM - Mark Abraham
In a week or so, I'll post on gmx-developers and gmx-users so we catch a wider audience who might want to start screaming/coding.

#2 - 06/26/2013 04:15 PM - Peter Kasson

Depends on what you mean by "fancy" orientation restraints, but we use a number of the restraint features. There are some important Gromacs features that aren't nicely compatible with DD, but people do use them. I would argue not to drop (options include disable DD/run in serial if necessary, print nasty warning messages, etc.). Some of this by the scream volume on gmx-users as you suggest. :) Thanks!

#3 - 06/26/2013 05:02 PM - David van der Spoel

Thanks for bringing this up Mark. The following features are some we are actively working on, and should be retained:

All of 2/3 should be retained unless we are to give the NMR refinement/analysis completely. We are currently in the process of adding other refinement tools to GROMACS and they have similar requirements, of having all particles coordinates available on all nodes. The code for computing restraints is also used in analysis tools, which is very useful for force field validation and protein folding analysis. Summary here should be according to me that the code should be updated to work with time averaged restraints using DD.

Polarization support (4) should be retained and updated. A new CHARMM polarizable FF for biomolecules will be released shortly and we want to support that along with other developments. Note also that a number of publications with polarizable models have used gromacs recently.

QM support (5) we are in the process of rewriting in Uppsala. This is being done from scratch in the qmmm branch. The present code will be removed once the new code works.

Generalized born (7) I noticed that it didn't work in master, crashing with LINCS errors. I guess others would like to use it for special applications, although we should discourage it in general.

#4 - 06/26/2013 05:19 PM - Michael Shirts

8) MTTK pressure coupling
Per Michael's intentions.

MTTK should remain, but it should be disabled with constraints. There's a posted fix, but still needs to be validated a bit more (4.6 issues have taken precedence).

4) Current polarization support?
Is anything here known to be useful enough that we'd want to keep it?

David vdS talked about CHARMM, but they don't use a shell model. I support including polarizability for drude models especially, but I think it may need to be built in from scratch, not necessarily from existing code. multistep may be our friend here.

7) Generalized Born
It worked in 4.5, but various aspects of implicit solvation got broken during the 4.6 cycle, and I see no pressure to fix any of it. The method is anachronistic. Its implementation will not benefit from anything else we are doing. Its existence leads people to assume that all the other things GROMACS can do work with GB (e.g. free-energy calculations), and I suspect that all that ever worked was the stuff that was wanted for the Larsson and Bjelkmar publications.

Implicit solvent is very useful for a lot of large scale applications. I'd like to see it continued.
Just a small correction: the Drude model is just another name for the shell model. We may need to implement additional stuff to make Charmm work though.

I agree that features that are not supported anymore should be deprecated, but to deprecate features because they don't scale well (when activated by the user I suppose) kind of violates the 'flexible' of the fast flexible and free policy. I'd like to keep QM support, but based on the new qmmm branch of David vd S. We will provide some essential tests also.

Hi!

I would suggest reversing the question, so rather than thinking what "should" be retained, the question is who is willing to help keep each feature supported :-)

Second, rather than making it absolute, I would suggest that we consider disabling some of these features temporarily. Otherwise ALL the efforts to maintain them will fall on the people doing the hard work to update the parallelization, etc. If the rest of us think a feature is important, it is then up to us to make sure we keep it working.

I wrote our L-BFGS code once upon a time, and it is very good when you need an accurate minimum (read: for normal mode analysis), so I can probably help fix it again.

I also have a long-term interest in polarization and QM/MM, so that too is something I'll try to contribute to.

For implicit solvent I'm torn. We could certainly fix it, but the question is whether it's worth it long-term?

PS: I'm all for killing particle decomposition. It will hurt some parts of the code slightly, but that's a signal we need to fix those parts of the code.

Updated top post with feedback so far. People who have expressed interest in retaining/using a feature can expect to be asked to help out with the work, e.g. at least "what is this code doing," "how do we test this is right," "please review the code." :-)
c) Writing of pair distance and/or time-averaged pair distance to energy files
Seems more like debugging output than anything you'd use in post-processing (and whatever it does support could be implemented via a trajectory-analysis tool).

Is anybody using this? What is it good for? Code to support this stuff is in lots of places. Aspects of time averaging require checkpoint support, but if there are aspects that only support the .edr output, that would be a candidate for simplification.

Another thing we should consider removing is installing the soft links from old tools to the new gmx program.

I agree with Michael and David with respect to keeping GB and implementing Drude polarization. Both are actually elements related to what I will be working on in my new postdoc position, so I will probably be able to lend a hand there, too.

Two points to consider:

- Even if we don't want to keep all these features, I think it would make sense to treat them as a sample of how people would want to extend Gromacs. It may not be a representative sample, but it may be biased towards "simple" features (extensive modifications probably never made it to mainline Gromacs), but it is still better than nothing. If we want to make the code easy to extend for this kind of applications, this is useful information. Even if we don't implement all of it now, we could still put just a bit of thought into the design to avoid creating something where it is impossible to implement these things in the future.
- As Mark says, many such features should still be straightforward to implement in serial (or probably more generally, with just one DD domain). If it is explicit that these features are not supposed to work in parallel, maintaining them and keeping them functional is much less effort. And it would avoid removing things completely, allowing parallelization support to be added separately later if someone wants to put that effort in.

Some more features of mdrun that may be included in the list:

- -ionize: doesn't seem to have been touched in a long time; does it still work, is anyone using it?
- GCT (General Coupling Theory?): I seem to recall a post on gmx-users quite a while back that this is broken. Also doesn't seem to have been touched in a long time.

I'm all for killing PD (unless there's some dependency I'm missing). Doing some combination of serial runs and domain "pinning" for the restraint features sounds good. IIRC we've done restraints for some large systems where we hacked the code to essentially do your option #3 (fatal error if there's a problem), and it worked well. But pinning would be more robust in that we'd take a performance hit rather than throw fatal errors.
I will throw out ionize and GCT. In the worst case I can re-implement it later based on older versions.

I think Gerrit does have a point. Just because an algorithm can't run on 1000s of cores because it is not compatible with DD, it may still be important, and may used by a considerable number of people. Among other things, such aspects should contribute to the decision of whether to drop a feature.

IMHO, the sheer complexity of the DD code represents a significant barrier in making features compatible with it. As far as I know, Berk is the main author and maintainer of all DD-related code and features that need to integrate with DD typically (have to?) go through him.

PD should work fine with multi-threading and should run quite well on multi-core platforms (e.g on Intel OpenMP multi-threading is quite efficient with up 24-32 threads). Moreover, if the features of interest have/gain support with GPU acceleration, given the pretty good single-node performance (without DD) on small to medium sized systems, DD and multi-node scaling is actually not a need, but more of a nice to have! Hence, I'd (also) suggest that, especially during the 5.x transition, we could keep many advanced features with no DD support, but with multi-threading and/or GPU acceleration supported.

It's not a matter about deprecating PD because it doesn't scale well, but because it complicates code. As long as we have PD, every single routine that does anything with parallelization has to implement two completely separate partitioning schemes. Similarly, some algorithms only support PD, and then we get a mess where users get errors about parallelization not working, and when they instead enable PD they get other errors.

Had this just been a matter of not actively removing working code it would of course have been fine, but do we really want to invest the limited resources we have into making sure particle decomposition still works when we change things? If we do want to keep it, what other features should we consider axing instead to reduce the maintenance load?

I regret the order of the reasons I wrote in support for removing PD - removing code complexity is indeed the primary objective there. I'm perfectly happy to have code and algorithms that cater only to single-thread, single-node or low-parallelism cases.

But if they're important to have/keep, then they need to be implemented in the same parallelism framework. If that means fatal errors and nasty warnings, that's the price of availability until someone values parallelism enough to do the job properly. That's like what we've been doing for the last 5 years - PD is still around and DD issues fatal errors because nobody has valued parallelism in these algorithms enough to convert to DD. (I still don't know whether real parallelism support for distance restraints exists now for PD.)

Preserving PD to cater for low-parallelism cases does not seem to offer enough return to pay the development and maintenance cost; DD can do that job, too. Maintaining two decomposition frameworks will get even more untenable when we move to task parallelism.
Another thing we should consider removing is installing the soft links from old tools to the new gmx program.

I'm all for this, if it is acceptable to break backwards compatibility in such a drastic way (other changes will likely require changing users' existing scripts anyway). It is already possible to disable installing the links by turning a CMake option; it is enabled by default (both in the binary directory and for installation) to avoid breaking all kinds of stuff (in particular, Jenkins builds). For example, the regression tests, the manual build and the online manual require the links to be present to build.

#22 - 07/02/2013 02:45 PM - Mark Abraham
Teemu Murtola wrote:

David van der Spoel wrote:

Another thing we should consider removing is installing the soft links from old tools to the new gmx program.

I'm all for this, if it is acceptable to break backwards compatibility in such a drastic way (other changes will likely require changing users' existing scripts anyway). It is already possible to disable installing the links by turning a CMake option; it is enabled by default (both in the binary directory and for installation) to avoid breaking all kinds of stuff (in particular, Jenkins builds). For example, the regression tests, the manual build and the online manual require the links to be present to build.

When we agreed on the creation of a unified binary (#685), keeping such symlinks was thought useful. There's a case for eliminating them (we're not going back, and it is a major release). In my heart, I always want to throw out the old and have new and better toys :-). However, updating our documentation will take time that I don't think we have, so I'd prefer a transition period. And users and third-party tutorial/tool providers will appreciate a gradual transition, too. So, I'm happy with announcing the old links are deprecated in 5.0 for subsequent removal.

It probably does make sense to change the names of some of the tools (partial discussion in #685), and we should probably do that for 5.0. For example, have gmx solvate and symlink genbox to it. (And banish genbox -nmol to gmx insert-molecule, for violating "One Tool, One Function." Perhaps leave a hint in the fatal error from wrongly-invoked gmx solvate pointing at gmx insert-molecule. Doing some such refactoring will start prompting users to get with the times, without necessarily breaking everything at once.)

#23 - 07/02/2013 03:08 PM - David van der Spoel
I think the transition period is now during development. Maybe we can post a mail to the gmx-developers to warn them this is going to happen. It is not a big deal I would think.

Splitting the likes of genbox into separate programs is another matter of course: I have nothing against it, but it is real work.

#24 - 07/02/2013 03:18 PM - Peter Kasson
Hmm--I think it would be good to have **some** period of time where both the old and the new methods work, so we don't fall into the well-trodden situation of shiny new way is beta and old way is deprecated. My group certainly has a decent-sized codebase that drives the old tools for various tasks. It will be great when we have python bindings and we can eliminate the annoying command-line stuff. But it would be good to be able to test a single set of code both ways during the transition period. It's hard to update code cleanly when way #1 and way #2 don't have the same behavior (i.e. work on different versions of the software).

#25 - 07/02/2013 03:39 PM - David van der Spoel
This is why I said the transition period is **NOW**. It's at least a year. Awk is your friend here. Are you working on python bindings by the way? If you develop that then you're independent of whether or not there are soft links, and you can develop this NOW as well, since all the infrastructure for at least the analysis is in place.

#26 - 07/02/2013 03:44 PM - Peter Kasson
I'm sorry, but I would explicitly direct my group **not** to use 5.0 draft code for production simulations and analysis now. The whole idea of the unstable master at this point is to develop new code. Transitions for "third-party" links need to happen after we have stable code. This isn't an issue I'm going to go to the mat on, but I think it's poor practice, and the likely result is that more people will stick with old versions longer rather than upgrading. Witness python 3.0. How many of us use it?

#27 - 07/02/2013 03:48 PM - Mark Abraham
David van der Spoel wrote:

> Splitting the likes of genbox into separate programs is another matter of course: I have nothing against it, but it is real work.

Sure. And so is trying to understand whether one of the options is intended to work in concert with the other options, and whether (and what) ordering of operation is implied. trjconv is an egregious offender here. There are some things that work if you combine their command lines, but you'd have to read the source to know the order of the operations, and no sane person should ever want to read gmx_trjconv.c! So we answer all the questions on gmx-users with "do one thing at a time." So we should enforce that, somehow :-) Sometimes that should be a check within the same tool, sometimes a split will make sense.

Oops, we're getting off topic. The point is that having some such splits will break some uses of the proposed symlinks, so that we get access to

- a period of decent-but-not-100% backward compatibility,
- developers have an easier time maintaining and debugging tools, and
- users get some hints that the lunch is not going to be always free.

I mailed gmx-developers back in 2011 that tool names were up for change in 5.0. Nobody responded (http://lists.gromacs.org/pipermail/gmx-developers/2011-February/005026.html). So there can't be a huge complaint, whatever we do.

In practice, if I were a third-party user of GROMACS tools (i.e. everyone), I wouldn't put any effort into porting my stuff until the 5.0 feature set was frozen. We pretty much ensure we won't get any user-space beta testing if nobody's scripts will work "out of the box." That sounds like a mighty fine reason for symlink support in 5.0.
Peter Kasson wrote:

I'm sorry, but I would explicitly direct my group not to use 5.0 draft code for production simulations and analysis now. The whole idea of the unstable master at this point is to develop new code. Transitions for "third-party" links need to happen after we have stable code. This isn't an issue I'm going to go to the mat on, but I think it's poor practice, and the likely result is that more people will stick with old versions longer rather than upgrading. Witness python 3.0. How many of us use it?

I did not say production runs. However, when my group develops code most group members help to test it.

That said I can buy Mark's argument for the development phase. But getting rid of unnecessary backward compatibility was the main theme in the thread, right ;).

Mark Abraham wrote:

From what I remember of the PD implementation, the reason it is more friendly for supporting the fancy restraints is that particles cannot change home processor mid-run. Was/is there support for a restraint across a PD slab? Alternatively, if there is only "accidental" support for such restraints working in parallel, then supporting that in DD is probably much easier than a fully general implementation.

Still haven't seen anybody report that they're using the NMR-modelling restraints in parallel (i.e. with PD). So we will plan for that to be supported only in serial, until somebody complains / volunteers some work.

Similarly, at least some kinds of parallel implicit solvent runs have been broken for years (#777), and the all-vs-all acceleration in 4.6.x is still broken now. So I propose that implicit solvation will only be supported in serial, and probably only with the Verlet code (since nobody knows what other kinds of things were broken in the pre-4.6 group kernel patches, and we have fixed too many bugs in those already). Of course, that situation can change when the method is important enough for someone to do the necessary work.

"Supported in serial" probably means single-domain DD. That might mean we can remove "simple" neighbour searching.

Michael Shirts wrote:

I'd propose eliminating the force field scan (bFFscan functionality). It seems like it would be sufficient to just generate force field parameters in new .tpr's and "mdrun -rerun" each new set of parameters.
OK, my post on gmx-users and gmx-developers generated a little bit of traffic to me off-list, but nothing that changes what we've said here.

So, the first order of business for me is removing particle decomposition. The main issue there is doing minimal damage to time- and/or pair-average distance- and orientation restraints. I have no specific relevant experience with these algorithms, or example input files. I hope those of you who do value these features can help me out with something I can use to see whether and when I introduce any regression. Tarball of .mdp+.top+.gro that works with 4.5 or 4.6 would be perfect.

... coz otherwise breaking something that right now requires PD to work correctly happens to be right near the top of the list of things that won't get me fired ;-)
David van der Spoel wrote:

Peter Kasson wrote:

I'm sorry, but I would explicitly direct my group **not** to use 5.0 draft code for production simulations and analysis now. The whole idea of the unstable master at this point is to develop new code. Transitions for "third-party" links need to happen after we have stable code. This isn't an issue I'm going to go to the mat on, but I think it's poor practice, and the likely result is that more people will stick with old versions longer rather than upgrading. Witness python 3.0. How many of us use it?

I did not say production runs. However, when my group develops code most group members help to test it.

I strongly support this. If not even the groups affiliated with the development are willing to test development code than who is supposed to do that? Unless a number of volunteering testers show up (which as we saw it did not really happen, at least not for 4.6), I don't think there is any other reasonable option than eating our own dogfood.

Regarding PD and other features that some may use, I strongly suggest publicly announcing the plans for the widest audience possible **and** offering an option for volunteers to pitch in with test cases and/or code if they would like to keep it. I see room for "contrib" non-core functionality to be accepted and maintained a bit more seriously than it has been, but of course this requires decentralization and volunteers.

Additionally, I'm all for removing complexity, but I still think that runs with a single-node limit (which we should stop calling "serial") should not be downplayed as much as they seem to be. Multi-threading already now provides pretty good scaling to 16+ cores and if we factor in accelerators, the lack of DD support with a certain mdrun feature does not mean that the feature has very limited scope due to low performance (one can easily get up to 50-200 ns/day on a single node with mid-sized protein systems). Moreover, the future tasking scheme will greatly improve on the non-DD multi-threading scalability.

Szilárd Páll wrote:

David van der Spoel wrote:

Peter Kasson wrote:

I'm sorry, but I would explicitly direct my group **not** to use 5.0 draft code for production simulations and analysis now. The whole idea of the unstable master at this point is to develop new code. Transitions for "third-party" links need to happen after we have stable code. This isn't an issue I'm going to go to the mat on, but I think it's poor practice, and the likely result is that more people will stick with old versions longer rather than upgrading. Witness python 3.0. How many of us use it?
I did not say production runs. However, when my group develops code most group members help to test it.

I strongly support this. If not even the groups affiliated with the development are willing to test development code than who is supposed to do that? Unless a number of volunteering testers show up (which as we saw it did not really happen, at least not for 4.8), I don't think there is any other reasonable option than eating our own dogfood.

Yes. The caveat is that the branch for the next release needs to stay nearly beta quality, which so far is the case.

Regarding PD and other features that some may use, I strongly suggest publicly announcing the plans for the widest audience possible and offering an option for volunteers to pitch on with test cases and/or code if they would like to keep it. I see room for "contrib" non-core functionality to be accepted and maintained a bit more seriously than it has been, but of course this requires decentralization and volunteers.

I've already emailed on gmx-users and gmx-developers, as reported above. I invited contributions and suggested providing test case, etc. I got only two responses, and neither was actually perturbed by the proposed removal of features. I can think of no other appropriate forum, unless there are suggestions. I'll bump the gmx-* threads and announce the issue is regarded as resolved.

Additionally, I'm all for removing complexity, but I still think that runs with a single-node limit (which we should stop calling "serial") should not be downplayed as much as they seem to be. Multi-threading already now provides pretty good scaling to 16+ cores and if we factor in accelerators, the lack of DD support with a certain mdrun feature does not mean that the feature has very limited scope due to low performance (one can easily get up to 50-200 ns/day on a single node with mid-sized protein systems). Moreover, the future tasking scheme will greatly improve on the non-DD multi-threading scalability.

We should be careful to distinguish serial from single-node; there are algorithms that do not (or have never been shown to) work in parallel. That's likely to stay that way until someone cares enough about some science they want to do to put in the required effort. Other algorithms can (accidentally?) benefit from things like OpenMP and/or with an accelerator, even if they are not able to work with multi-domain DD.

These algorithms are all fine to have, but we owe it to the users to prevent wrong results from unsupported use cases, and to ourselves to make clear what cases are not important when designing code. My inclination is to be rather more explicit about warning users when we have no idea whether something works. The problem is that 99% of the combination space is untested and is likely to stay that way. How can we best instill a sense of caution? No serious experimentalist would leap into using a new technique on a problem with an unknown answer, but I think that is distressingly prevalent in computational fields. Even "my professor said to do it this way" is often wrong - the professor last read the manual two major versions ago...
Mark Abraham wrote:

We should be careful to distinguish serial from single-node; there are algorithms that do not (or have never been shown to) work in parallel.
That's likely to stay that way until someone cares enough about some science they want to do to put in the required effort.

Not necessarily. OpenMP parallelization can be moderately easy to accomplish. On the other hand, the tasking framework can ensure that even if a certain algorithm is not multi-threaded, it does not become a major bottleneck (if it does not pose inconvenient task-dependencies and as long as it does not take much longer than everything else).

These algorithms are all fine to have, but we owe it to the users to prevent wrong results from unsupported use cases, and to ourselves to make clear what cases are not important when designing code. My inclination is to be rather more explicit about warning users when we have no idea whether something works. The problem is that 99% of the combination space is untested and is likely to stay that way. How can we best instill a sense of caution? No serious experimentalist would leap into using a new technique on a problem with an unknown answer, but I think that is distressingly prevalent in computational fields. Even "my professor said to do it this way" is often wrong - the professor last read the manual two major versions ago...

I agree. However, we should be very careful with the wording because being to honest about the large number of untested combinations of options/features, if documented, can be yet another point to criticize GROMACS for -- even though most academic/scientific software is probably worse in this respect, just that they are typically not open about it.

Justin Lemkul wrote:

I just wanted to chime in quickly here regarding polarization. I am willing to contribute to the efforts to possibly re-implement the way we do polarization. Polarizable force fields are now the main focus of my postdoc in Alex MacKerell's lab and he has asked me to try to get things working in Gromacs like they do in CHARMM. This is something I would like to discuss during the upcoming workshop, so I can get a feel for where things are and what the path forward is. Count me in for development and testing.

Mark Abraham wrote:

Also on the hit list for 5.0 is heuristic group-kernel neighbour-list updates. Berk and I suspect nobody is using them, and most of what they are good for are available now/soon with the Verlet kernels. And they create a lot of messy signalling requirements in the MD loop that we'd rather not have to deal with.

Mark Abraham wrote:

Erik just pointed out we can probably just warn the user that what they really want are the Verlet kernels, and move them over - but we would want the Verlet kernels to support a few more things yet.
Mark Abraham wrote:

Erik just pointed out we can probably just warn the user that what they really want are the Verlet kernels, and move them over - but we would want the Verlet kernels to support a few more things yet.

After further thought, I don't think automatic switch to Verlet is correct. The user should be more involved in making Verlet-kernel-specific decisions than that. See https://gerrit.gromacs.org/#/c/2665/

---

Mark Abraham wrote:

Another scalp for my collection: constant acceleration NEMD has probably been broken for over four years since a6ee084bcf removed the only call to update_ekindata(). I vote we remove it.

While we're there, there's another non-equilibrium MD feature that uses the cos-acceleration .mdp setting. It looks like Berk implemented it in about 2000 and it's had only cursory attention since. I vote we remove it. Does anybody still care about it?

---

David van der Spoel wrote:

We definitely want to keep cosine acceleration. This is the best way to compute viscosities as far as I know. I have used this in papers as well since Berk implemented it and am planning to use it again. Apart from being useful it is not particularly in the way is it? And the same with the constant acceleration: these things do not interfere with parallelism. I would rather vote to strengthen this kind of functionality such that the liquid community can see gromacs as an alternative to lammps or dилopoly.

---

Mark Abraham wrote:

Great - we should keep things that are in use if at all possible. Can I have some sample input for it, please? I would like to show that it is surviving the transitions going on around it.

Note that parallelism is not really the concern with these NEMD algorithms. The reason both are a bit of a pain is that both have their implementations living in a gmx_ekindata_t that also maintains collecting the output of all the different kinds of temperature coupling, and various other intermediate outputs for computing things that relate to kinetic energies. This kind of thing makes it super hard to reason about or improve the control flow in the MD loop - and that means either paralysis or bugs for everyone. It also makes it harder to notice things like that constant acceleration is broken because its update routine is never called.

There's also a struct called gmx_ekinstate_t, which seems to just be a bucket to hold things while they go to and from checkpoints. It swaps data at various points with these algorithms, but now that they're not all in the same struct it is clear that constant acceleration is not checkpointed, either.

I already have a draft patch that breaks gmx_ekindata_t into what I think are its separable parts, and it seems to work. Being able to make a test case that shows cosine acceleration working would add confidence that it is done correctly.
Is the support for architectures which don't have a 64bit integer deprecated for 5.0? I don't mean the support for a 32bit binary, but only that we require that it provides a 64bit type so that gmx_large_int_t is guaranteed to be 64bit. This would simplify a few things and removes one compile condition which is hardly (/not at all?) tested.

I'm happy to require a 64-bit int in the interests of simplification.

I can only see that being a potential problem for Folding@Home, running new code on old OS. That doesn't sound like a tail that should wag the dog. Thoughts, Peter?

The OS shouldn't matter (Windows XP has support for 64bit). C99 requires support for 64bit. And hardware support is available for all current CPUs (for ARM version>8 2011).

I agree with Roland--at least from a F@H perspective, this should be fine. Current versions of the Folding@Home build shouldn't have a problem with this. I think support for 64-bit ints (as Roland says, not necessarily a 64-bit OS, but the ability to specify 64-bit int) is a reasonable requirement. Is anyone aware of target platforms that would have an issue with this? e.g. some of the exotic SPARC-based ones?

- Related to Bug #1416: 2-D Ewald summation and DD is broken added

I have finally completed the patch that removes PD. Details on gerrit, but to the extent that I can test that things still work according to the above discussion, I have. Nobody has helped me with how to construct test cases for time- or ensemble-averaged restraints, so I do not consider the lack of testing those algorithms a barrier to progress. This situation can still change, but "maybe it used to work in parallel and if I wish hard enough it'll keep working" is not really an excuse for leaving toys all over the floor ;-)
#57 - 04/16/2014 01:19 PM - Rossen Apostolov
- Related to Feature #753: Use of GB in parallel and/or with all-vs-all kernels needs a mention in the manual added

#58 - 04/16/2014 01:48 PM - Rossen Apostolov
- Related to Bug #973: Energies from implicit solvent calculations added

#59 - 05/13/2014 10:04 AM - Mark Abraham
Ongoing discussion in #1500

#60 - 05/13/2014 10:04 AM - Mark Abraham
- Status changed from New to Closed

#61 - 06/01/2014 11:55 PM - Gerrit Code Review Bot
Gerrit received a related patchset '1' for Issue #1292.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: I7628b78862144ae2478b6f42a68189f15e3a22fe3
Gerrit URL: https://gerrit.gromacs.org/3533