GROMACS - Task #1411

Future of thread_mpi

12/27/2013 06:28 AM - Teemu Murtola

Status:	New
Priority:	Normal
Assignee:	Mark Abraham
Category:	core library
Target version:	future
Difficulty:	uncategorized

Description

It is quite unclear what are the plans for thread_mpi in the future. General plans would be good to have to know where to direct refactoring efforts and what approach to take, e.g., when moving stuff away from legacyheaders/. Issues to discuss:

- Is there some replacement planned for some or all of its functionality? If so, on what timeframe will such a replacement realistically materialize? Generally, I think the following is the list of functionality it currently provides:
  - Basic thread support (wrapping both pthreads and Win32 threads behind a pthreads-like API). Also includes some thread affinity support.
  - Atomics support for several compilers/architectures.
  - MPI wrappers.
- Do we want to keep it as a separate library, with no dependencies to other Gromacs code?
- Do we want to keep it under a separate license/copyright when bundled with Gromacs?
- Do we want to keep maintaining it in a separate repository that is synced with the Gromacs code?
  - Where is that separate repository, who have access to it etc.? Should the separate repository be under Gerrit?
  - Is there some useful content (e.g., tests) in the current separate repository that should be merged into the main Gromacs repository if we are no longer maintaining the separate repository?
  - How is code going to get synced between the repositories? Are there some limitations on who is allowed to modify and what in the copy that resides in the Gromacs repository?
  - Is there someone willing to, interested in, and time available to take the long-term maintenance responsibility for the separate repository?
- Should we split the basic thread support from the MPI wrappers more explicitly?
  - How do we do this if the code is in a separate repository? Have subdirectories in that separate repository, and copy them to the Gromacs repository directly under src/gromacs/? Or just have separate subdirectories under src/gromacs/thread_mpi/ or similar?
- If the plan is to at some point release the library separately, do we need to play nice and never install anything related to the library outside, e.g., include/gromacs/? Do we need to support linking against a thread_mpi instance already on the system?

Possible approaches forward (from the Gromacs side of things):

1. Reorganize the thread_mpi code in the Gromacs source tree according to the new Gromacs approach (everything under src/gromacs/thread_mpi/, or possibly having a separate directory for the basic thread support and atomics).
   - This is somewhat difficult to keep in sync with the external repository if we want to keep it, since that probably should not have the same layout.
   - If we want to keep the code separate, this needs some explicit policy so that people don't inadvertently add unwanted dependencies.
2. Move the thread_mpi code to src/gromacs/external/thread_mpi/, and maintain it there.
   - Makes it clear that the code is not really Gromacs code, and allows keeping a layout that is closer to the external repository if we ever want to release the code.
   - This requires some extra trickery to get working correctly as long as we have installed headers (most notably, commrec.h) that depend on thread_mpi and want to keep installing the headers as include/gromacs/thread_mpi/ instead of include/thread_mpi/.
3. Make the thread_mpi a submodule under src/gromacs/external/, like is planned for the TNG library in https://gerrit.gromacs.org/#/c/2704/.
   - This way, the repositories automatically stay in sync.
   - Probably not worth the hassle, unless there are more submodules than just thread_mpi.
   - This raises the barrier for anyone except dedicated people to contribute to thread_mpi.

Related issues:

Related to GROMACS - Task #662: Reformat existing Doxygen comments	**Closed**	**01/12/2011**
Related to GROMACS - Task #1530: Offer binary downloads	**New**	**06/24/2014**
Related to GROMACS - Feature #1500: Post-5.0 feature clean-up plan	**New**	**01/20/2014**

Related to GROMACS - Task #1828: Exception handling in mdrun	**New**
Related to GROMACS - Task #1829: Future of thread level parallelism	**New**
Related to GROMACS - Task #1588: Future of single-node parallel coding	**Closed**

Associated revisions

Revision 8c1a0b04 - 03/27/2014 04:15 AM - Teemu Murtola

Move thread_mpi to src/external/

Now that thread-MPI headers are no longer installed, the whole thing can
be easily moved to src/external/, which better resembles its status
until it is decided what we really want to do with it. There, it is
excluded from checks that are done for other parts of the source, as
well as from Doxygen, removing the need to treat it specially in all
these cases.

This also removes a large number of headers from legacyheaders/ with a
relatively simple change, advancing this effort.

Related to #1411.

Change-Id: I64d202eb61dc2c61668ad5858d551add6578cb97

History

#1 - 01/03/2014 05:31 PM - Szilárd Páll

Teemu Murtola wrote:
Issues to discuss:

- Is there some replacement planned for some or all of its functionality? If so, on what timeframe will such a replacement realistically materialize?

I personally don't know of any replacement planned. I personally don't know of any major deficiencies of thread_mpi; are there any?

- Do we want to keep it as a separate library, with no dependencies to other Gromacs code?
- Do we want to keep maintaining it in a separate repository that is synced with the Gromacs code?

As far as I understood Sander was planning to publicly release the code as it could be rather useful for many projects. (Partly?) Because of this, the development has been completely separate from GROMACS. If this is still the case, the answer to both above questions is "yes", but even if it's not the case, it would be useful on the long run to keep at least the MPI implementation as separated as possible.

- Is there some useful content (e.g., tests) in the current separate repository that should be merged into the main Gromacs repository if we are no longer maintaining the separate repository?

Note that AFAIK the in-gmx-tree code differs a bit from the one in the repo in that there are some GROMACS-specific default/tweaks in the former.

- Do we want to keep it under a separate license/copyright when bundled with Gromacs?

Could be re-licensed just to keep things simple.

- Should we split the basic thread support from the MPI wrappers more explicitly?

That would certainly be useful as the upcoming tasking implementation will need feature that overlap with the thread support code.

#2 - 01/06/2014 07:06 AM - Teemu Murtola

- Description updated

Updated the description to include more background and some alternatives for the way forward.

#3 - 01/14/2014 05:55 AM - Teemu Murtola

- Description updated

Szilárd Páll wrote:

> Teemu Murtola wrote:

> Issues to discuss:
>
> - Is there some replacement planned for some or all of its functionality? If so, on what timeframe will such a replacement realistically materialize?
>
> I personally don't know of any replacement planned. I personally don't know of any major deficiencies of thread_mpi; are there any?

I raised this, because in various contexts, people have mentined, e.g., hwloc and TBB, when discussing the future of threading. They don't provide all the same features, but if we want to move towards task parallelism, then we either need to use an existing framework, or implement our own. And thread_mpi doesn't currently provide one. I have also seen, e.g., you complaining about potential portability issues with the atomics support and the requirement that we currently have on either pthread or Win32 threads support.

In general, if we move towards task parallelism, how many different threaded parallelization schemes we need to support concurrently? That would make three: emulated MPI, task parallelism, and OpenMP.

> - Do we want to keep it as a separate library, with no dependencies to other Gromacs code?
> - Do we want to keep maintaining it in a separate repository that is synced with the Gromacs code?
>
> As far as I understood Sander was planning to publicly release the code as it could be rather useful for many projects. (Partly?) Because of this, the development has been completely separate from GROMACS. If this is still the case, the answer to both above questions is "yes", but even if it's not the case, it would be useful on the long run to keep at least the MPI implementation as separated as possible.

Yes, but that comes with a maintenance cost. Someone needs to take the responsibility to maintain the code and make it fit into the new Gromacs layout. And it is extra effort (but doable) if we want to reduce the dependencies of existing code on, e.g., config.h to reduce rebuild times.

> - Is there some useful content (e.g., tests) in the current separate repository that should be merged into the main Gromacs repository if we are no longer maintaining the separate repository?
>
> Note that AFAIK the in-gmx-tree code differs a bit from the one in the repo in that there are some GROMACS-specific default/tweaks in the former.

My understanding was that Sander was manually synchronizing the repositories from time to time (at least, after adding new functionality into the separate repository and copying that over to Gromacs). Unless there is a maintainer for the separate repository who can act relatively quickly on issues noticed, I don't think we have any alternative except to allow modifications directly in the Gromacs tree, and synchronize those lazily (if at all).

#4 - 01/14/2014 07:41 PM - Roland Schulz

thread-mpi is missing support for MPI datatypes which we require for #966. Do we still need the contained MPI? For most users who don't have MPI the OpenMP support should be sufficient. For others it wouldn't be hard to auto-build MPI (mpich or OpenMPI) the same way as we do for FFTW. If we could deprecate the MPI part of tMPI, then the rest could be covered by TBB.
The advantage of removing tMPI would be: 1) less maintenance and 2) we could use all of MPI (not just datatypes but also e.g. MPI-IO).

#5 - 01/14/2014 07:54 PM - Erik Lindahl

We absolutely need thread_MPI until we have our own thread layer (or OpenMP) support for all parts of Gromacs. The problem is that MPI installation requires root permission on Windows, so it will not work in distributed computing environments. The other part is that we need to focus on making the build system successively smaller/simpler, rather than getting to the point where we automatically download and compile the latest Linux version for the user :-)

#6 - 01/14/2014 10:06 PM - Mark Abraham

Erik Lindahl wrote:

> We absolutely need thread_MPI until we have our own thread layer (or OpenMP) support for all parts of Gromacs. The problem is that MPI installation requires root permission on Windows, so it will not work in distributed computing environments.

I'd be surprised if we couldn't finesse that - we don't need to install mpirun or anything anywhere other than our own build tree. Kitware does this kind of thing all the time in their projects, so there is probably CMake support for anything we might want.

But I'd also be surprised if we'd want to finesse that! :-)

> The other part is that we need to focus on making the build system successively smaller/simpler, rather than getting to the point where we automatically download and compile the latest Linux version for the user :-)

Writing and maintaining automated download and build of a controlled version of hwloc and TBB sounds to me like a lot more fun than writing and maintaining custom

- CPUID stuff
- cache structure detection
- GPU and network locality & detection
- atomics support & detection
- affinity support & detection
- timing support
  Some of our most complicated pieces of CMake-ry are... CPUID and atomics support detection! The one thing in favour of the status quo is that it pretty much works. Our perennial problem is that our needs are ahead of the support for them in the tools we have available - but that should mean being prepared to get rid of our hand-rolled stuff when support has caught up (e.g. no assembly kernels). TBB looks refreshingly useful out of the box!

Heck, I'm even tempted to bootstrap compiling CMake on *nix ;-)

On Teemu's points: I definitely think separating the low-level thread-MPI machinery into a separate module (src/gromacs/threading?) from the MPI layer that uses it is a good idea. (So Teemu's option 1, second flavour.) There's already a two-part structure with its CMake support.

If the point of working on good task parallelism is to maximize intra-node performance for strong scaling, then it should make the single-node MPI layer redundant as a side effect. This will be a massive simplification to mdrun setup. If TBB as a task-parallelism solution provides all the low-level threading-type functionality we need, then I won't shed tears over losing the other chunk of thread-MPI either. Atomics support is guaranteed by C++11 (and C11, I think), which is another thing from which we will eventually benefit.

Thread-MPI was great stuff in its day, but its edge in the market (roughly, portable atomics and affinity support) is decaying as others catch up.

My dream setup in a year is that we use something like TBB+CUDA+MPI in mdrun (plus either our own or hwloc detection stuff). TBB and OpenMP can co-exist, but there's not a great deal in mdrun that makes OpenMP a clear winner in usability - unless you restrict the innermost loops to C-only and work only on regular arrays. But there's a lot of bridge to build, never mind water to pass under it!

GROMACS tools probably make the most sense with OpenMP only. I've been asking, and I haven't heard of much in the way of a compute-bound analysis task yet. (Though g_hbond apparently supports something useful in that line, and local pressure calculations might fit)

#7 - 01/14/2014 11:56 PM - Roland Schulz

Mark Abraham wrote:

> Erik Lindahl wrote:
>
> > We absolutely need thread_MPI until we have our own thread layer (or OpenMP) support for all parts of Gromacs. The problem is that MPI installation requires root permission on Windows, so it will not work in distributed computing environments.
>
> I'd be surprised if we couldn't finesse that - we don't need to install mpirun or anything anywhere other than our own build tree. Kitware does this kind of thing all the time in their projects, so there is probably CMake support for anything we might want.
>
> But I'd also be surprised if we'd want to finesse that! :-)

I agree. It shouldn't be any problem to bundle the required dll of the MPI library without relying on an installer. If that is the only reason we have to keep tMPI, I would be glad to do that for the FAH project. Certainly much easier than adding MPI IO and datatype support to tMPI.

#8 - 02/16/2014 10:40 AM - Teemu Murtola

- Related to Task #662: Reformat existing Doxygen comments added

#9 - 03/25/2014 05:16 AM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #1411.
Uploader: Teemu Murtola (teemu.murtola@gmail.com)
Change-Id: I64d202eb61dc2c61668ad5858d551add6578cb97
Gerrit URL: https://gerrit.gromacs.org/3291

#10 - 03/25/2014 09:50 AM - Erik Lindahl

I chatted a bit with Szilard & Mark over coffee yesterday related to thread stuff. It's not a firm schedule yet, but once 5.0 is out I realized it makes more sense to start with the multithreading rather than modularizing neighborsearching and nonbonded kernels better, otherwise we'll just have to rewrite them a second time with the task scheduling.

So, my quite personal plan is to start getting my feet wet by first writing lower-level modules for threads, mutexes, atomics, and a thread workpool. Once this works we can start hacking more of a task/scheduler class. In the mean time, Mark might try to work in parallel on taskifying some of our current code by using TBB. Even if we likely need our own scheduler already for 6.0, this will allow us to make progress on both fronts, and we'll hopefully learn better what we need from our own task class.

#11 - 06/20/2014 07:46 PM - Roland Schulz

I think it would be great if we could add the functionality missing in TBB by extending TBB (can include replacing parts), instead of directly using our own scheduler without TBB. If we indeed need our own scheduler, than this approach would mean we make a scheduler which is replacing the default TBB scheduler - but is used by the code through the standard TBB constructs. I think this would have a couple of advantages:

- Writing the scheduler and taskifying can progress in parallel, without requiring changes to taskified code after the scheduler is changed
- Our own scheduler doesn't has to be used. E.g. people can experiment with other schedulers, and we might be able to stop supporting our own if TBB incorporates our requirements in the future.

#12 - 06/23/2014 05:20 PM - Szilárd Páll

I agree, it's been a challenge to just start task encapsulation, let alone write an entire threading library and scheduler, so I think at first the effort should go into task encapsulation and simply using a TBB backend. The only caveat with trying to fully rely on TBB (with a custom scheduler) is that again, our license conflicts with TBB's. As the multi-threading and tasking will be a rather performance-sensitive and probably fragile, I'm not sure we want to rely on using whatever TBB version the user has on their machine - especially if we want to modify/extend TBB.

#13 - 06/23/2014 07:16 PM - Roland Schulz

The licensing shouldn't be any problem. Just because the code contains TBB source doesn't change the Gromacs license (and if we want to be extra sure we can do things like the auto-download we use for FFTW). And it can be continued to be used without GPL code because the user can use the commercial version of TBB and also by disabling TBB (if it is optional the same way as OpenMP is now). But even when used with the TBB GPL it doesn't effect the usage of Gromacs within a non-GPL program. The runtime exception is sufficient for that.

#14 - 06/23/2014 07:31 PM - Szilárd Páll

As far as I know, rather than just replacing OpenMP, the (quite ambitious) plan was to to restrict the use of MPI (or equivalent) to inter-node parallelization and intra-node use only our tasking - i.e. single rank/compute node by design. The same MPI-only SPMD we have now could still work, but it may take quite some effort to keep it performant for rank counts 2+ orders of magnitude apart (assuming ~100 cores/hw threads per node). If this direction remains, I do you think GROMACS can afford to have an optional library as its main means of parallelization? Reverting to truly serial if downloading TBB is not possible would be a quite strong limitation.

#15 - 06/24/2014 05:57 PM - Roland Schulz

What is the specific license problem you see if we don't make it optional?

#16 - 06/24/2014 07:39 PM - Szilárd Páll

Roland Schulz wrote:

> What is the specific license problem you see if we don't make it optional?

Unless I'm mistaken, the same reason that prevented us from bundling fftw3 is valid for TBB too, isn't it? The runtime exception TBB's license explicitly states makes no difference in this context, I guess.

#17 - 06/24/2014 08:22 PM - Roland Schulz

The problem with FFTW was that Erik didn't want binaries build and offered for download on servers paid by EU research funds to be GPL. Building against FFTW makes that binary GPL. That problem wouldn't be there for TBB even if TBB isn't optional. The source would still be LGPL and it if we ever want to do binaries we could use the commercial version of TBB, so that the binary is LGPL too. If FFTW would give us a license that we can distribute Gromacs+FFTW as LGPL (maybe if we give them a guarantee that it is linked in a way that FFTW isnt accessible from outside so it can be used by other, they might actually do it) then there wouldn't have had the license problem with FFTW either.

#18 - 06/24/2014 08:27 PM - Erik Lindahl

Just to set the record straight, it isn't that I care about how we use a particular server, but if we started shipping Gromacs binaries with FFTW built-in we would have to state clearly that the Gromacs binary distribution is strictly GPL, rather than LGPL. That in turn would (a) be confusing, and (b) it could get us in trouble if we want to claim that Gromacs is LGPL and business-friendly ;-)

#19 - 06/24/2014 09:32 PM - Roland Schulz

Put my answer to FFTW to #1530 (this was getting to off-topic)

#20 - 08/24/2014 12:24 PM - Mark Abraham

I would again ask people to do their homework before launching discussions based on licensing hearsay. I realize there is not much in the way of definitive legal precedent, but please at least read the licenses!

Licensing 101: GPL-type licenses cover the right to distribute, modify and/or create the original or any derived works, and the conditions under which that can take place. The right to use any such work is unencumbered.

TBB is dual licensed, one commercial and one GPLv2 + runtime exception (https://www.threadingbuildingblocks.org/licensing). The exception is relevant only when distributing binaries (but would permit us to do so).

Bundling the TBB source code (modified or not) would have the same problem as FFTW - it is unclear whether bundling the TBB distribution and #including TBB header files makes our source code a derivative work. Mere aggregation does not, simple use of the headers probably does not. If bundle+#include it does make a derivative work, then we would have to distribute that bundle under the GPLv2.

So, if we are able to use TBB without modifications, then we can distribute our source under LGPLv2.1 and use the same kind of approach we use for FFTW - link with a version provided by the system, or download and build a standard TBB the way we like it.

If we need to modify TBB source (e.g. to support new hardware), then issues become less clear - we can modify and re-distribute a standalone TBB under GPLv2, but it may become harder to demonstrate that GROMACS source code is no longer a derivative work, if we care. Obviously, we would seek to contribute our modifications back to TBB, and either wait for a release or negotiate something. If needed, a realistic option in that case might be to distribute LGPLv2.1-licensed binaries based on TBB, which is possible thanks to the runtime exception. Not sure what impact CUDA has in that case. However, weirdo hardware is probably not supported by MKL, and we can't distribute binaries linked against FFTW (yes, I've asked for FFTW under LGPL license, and Mateo Frigo said no), so we'd have to seek another solution for binary-distributable FFT support. (As Erik noted recently, our FFT vectors are so short that we might be able to roll our own, which might even be faster!)

Note that TBB is already modular with respect to schedulers, so we ought not to have to modify simply to implement a custom scheduler (but in any case, I think the right approach is to taskify first, then optimize scheduling, rather than achieve nothing by trying to do it all in one shot).

#21 - 08/24/2014 12:26 PM - Mark Abraham

- Related to Task #1530: Offer binary downloads added

#22 - 08/24/2014 01:00 PM - Mark Abraham

Erik suggested the other day that we rip out the parallelism side of thread-MPI, since CPU-only single-node machines are no longer a key optimization target. There are multiple ways for someone who still has need for performance on such machines to mitigate that (run GROMACS 5, use OpenMP, use real MPI, and later use TBB/whatever). That will likely lead to lower single-node performance on some hardware for some period, but I like the idea because it involves serious simplifications of mdrun initialization, and Jenkins testing matrix.

#23 - 08/24/2014 01:01 PM - Mark Abraham

- Related to Feature #1500: Post-5.0 feature clean-up plan added

#24 - 08/24/2014 02:32 PM - Erik Lindahl

Yeah, although it saddens me a bit, it's likely time to replace FFTW. It's not just the licensing being more restrictive, but it still doesn't come with eg AVX2 acceleration (and I expect FMA to be great for FFTs), AVX-512 might take even longer, and it might never support some other SIMD platforms.

My idea for single nodes was to rely on OpenMP once we have removed the group kernels. However... When I said that I had forgotten that Clang on OS X still doesn't do OpenMP, and this is still a very common target, not to mention that many developers use it on their laptops.

As a temporary solution, we might be able to hack something using Apple's APIs, or even a small thread wrapper to start kernels in many threads.

#25 - 08/24/2014 03:25 PM - Mark Abraham

Erik Lindahl wrote:

> Yeah, although it saddens me a bit, it's likely time to replace FFTW. It's not just the licensing being more restrictive, but it still doesn't come with eg AVX2 acceleration (and I expect FMA to be great for FFTs), AVX-512 might take even longer, and it might never support some other SIMD platforms.

FFTW has a generic --enable-fma. Does that not do anything useful for x86? IIRC that was all that is required for BG/Q.

> My idea for single nodes was to rely on OpenMP once we have removed the group kernels. However... When I said that I had forgotten that Clang on OS X still doesn't do OpenMP, and this is still a very common target, not to mention that many developers use it on their laptops.

We do need to bite some bullet at some point. Once we've added tabulated interactions to the Verlet scheme, and decided what to do with QM/MM, GB, TPI, and adress (which all have to support the Verlet scheme in some form, or die, which on present form means several will die), then I'm keen for killing group kernels, then killing thread-MPI parallelism.

We will probably need to maintain a task-parallelism branch for some time (merging master into it regularly). One option is to convert enough stuff to C++ in master, do trivial encapsulation of OpenMP-region functionality (which I have started in some private branches), fork a task-parallelism branch, add TBB support, and copy-paste tbb::parallel_for instead of #pragma omp. Then measure some performance and compatibilty and take stock. Depending what we learn, killing thread-MPI and OpenMP might already be reasonable.

> As a temporary solution, we might be able to hack something using Apple's APIs, or even a small thread wrapper to start kernels in many threads.

Huh? :-) There is an Intel-supported clang repo that has OpenMP, if people are keen to install it to get clang+openmp. That sounds like less work overall for devs.

#26 - 08/30/2014 03:36 AM - Roland Schulz

OpenMP is available if using gcc on Mac. Of course the gcc-4.2 by apple doesn't have avx. And by default the gcc from MacPorts also has problems (and HomeBrew seems to have the same issue with the old binutils). But according to http://stackoverflow.com/a/19342603/1385788 all one needs to do is add "-Wa,-q" to the compiler flags. And we could even do that automatically in our cmake. Don't have access to any Mac other than the Jenkins host and that doesn't seem to have AVX. Could someone try this? It seems to me that it would be acceptable to require a new gcc (from MacPorts, HomeBrew, GentooPrefix, ...) for users who want thread support, given that in my experience most MacOS & Gromacs users have MacPorts (or equivalent) installed anyhow.

#27 - 08/30/2014 09:13 AM - Erik Lindahl

We already enable that option automatically in Gromacs-5.0. However, while some developers and advanced users might have installed MacPorts, but they are a very small minority, even of our users - even I don't have it to avoid polluting my harddisk with lots of stuff (although I have gcc installed separately). Since Macs are by far the easiest option for a non-expert to use Gromacs, it will make life a lot more complicated for the if we would have to require them to start by installing a new build system just to use Gromacs.

However, it appears that Apple's GCD-extensions are quite similar in scope to OpenMP, and since the only critical part is mdrun I think we might be able to add that in a handful of places so normal simulations still work in parallel, if we don't get the tasks working well before then. Since thread_mpi isn't really holding up anything I don't see the point in crippling one platform just to do this slightly sooner (I simply forgot OpenMP when I suggested it), so I think it's better to postpone that a couple of months until the replacement stuff is in place.

#28 - 08/31/2014 08:31 PM - Roland Schulz

Currently our install-guide recommends Intel. Which also needs to be installed separately. A separate installed gcc (Macports, Gentoo, ...) is as good, correct? Then I think we should add that to the guide.

If we plan to replace OpenMP with TBB why do we need GCD?

#29 - 09/07/2014 09:17 PM - Peter Kasson

Can I weigh in with a dumb question here? How well does TBB (or similar platforms that we might use to build task parallelism) deal with hierarchical structure on a single node? (e.g. cores are grouped into units where intra-unit communication is much faster than inter-unit communication.) If one imagines a rather fat node, then a "flat" model where every core is just as close as any other may cause trouble for us. It seems that the ability to have MPI or threads-MPI among units on a fat node might be helpful for encoding this hierarchical structure.

(Disclaimer: there might be better ways to solve this, but I wanted to raise the question.)

#30 - 09/08/2014 12:01 AM - Erik Lindahl

It doesn't, but neither does MPI nor OpenMP. That comes down to the design of the parallelization, and the main reason for this change is that we'll eventually be able to scale further with advanced thread-level parallelism that uses tasks.

We'll always have support for MPI, but as an external library.

#31 - 09/08/2014 12:13 AM - Mark Abraham

What Erik said, plus that the restriction of one PP domain per MPI rank adds a large amount of overhead. Mapping a domain to a NUMA/whatever region requires that there **is** a domain decomposition (creates organizational overhead, and restricts those algorithms that don't work with more than one domain) and communication (which ought to be cheap in memory, but we still copy a lot more buffers, which library-MPI probably copies again). We know we can't program for a flat model (cf current sucky performance of our use of OpenMP across a NUMA boundary), but whatever we come up should (by design) offer a better chance of (e.g.) keeping the cost down to L3-L3 copies over QPI.

#32 - 09/17/2015 09:41 PM - Roland Schulz

- Related to Task #1828: Exception handling in mdrun added

#33 - 10/05/2015 10:21 PM - Roland Schulz

- Related to Task #1829: Future of thread level parallelism added

#34 - 11/17/2016 04:02 PM - Mark Abraham

- Related to Task #1588: Future of single-node parallel coding added

#35 - 02/26/2018 04:56 PM - Mark Abraham

Note that thread-MPI recently got converted to compile as C++ in 3a46c31dc041cb863394c993fac537db573d23e4