GROMACS - Feature #2239

split libgromacs into base and full

08/29/2017 12:53 AM - Roland Schulz

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

Description

It would be nice if testutils would not depend on the full libgromacs but only on the small subset which is needed. This would speed-up compiling of some of the unit tests a lot. An example is simd-test. One option would be to split libgromacs into two parts. One part being the fast to compile utility modules. And the other part being all other.

History

#1 - 11/12/2017 07:46 PM - Szilárd Páll

Is the issue only the simd tests or a more general one? There were plans to spin off out the SIMD module into a separate library which would also help here (as the unit tests would inherently be independent) but I'm not sure if those plans were postponed or abandoned.

BTW, I proposed the same a few years ago although the reasoning was somewhat different: to have a mdrun-only build that's leaner and to avoid going wild with C++11 only outside of the core mdrun. There was not a lot of enthusiasm for it (and given that most if not all of the conservative C++ use has mostly been ditched all over the code part of it is moot point). One of the objections was, I think, that defining an API is too much effort -- not sure to what extent does that apply here?

#2 - 12/30/2017 11:58 PM - Erik Lindahl

I would still like to split off the SIMD code, but I think the history there is a good example why it's complicated to have many libraries: Originally there was not a single dependency, but as we have implemented various features there have been more and more dependencies on other parts of the code.

I see a huge maintenance advantage with a simple common language code base and coding standards for everything. Of course it's also possible to have advanced setups with different libraries using different internal utilities and different coding requirements in the different parts, but that would potentially come at a fairly high effort cost - in particular for defining APIs.

Similarly, there has also been discussions that it could be convenient to only have the core MD parts as the separate library (i.e., not the utility modules). That too could of course be useful in some cases (portability), but would lead to even more complex dependencies if we now have three different libraries.

At the end of the day, unless somebody has tons of spare time and would like to do this, I think we should stick to the present setup - I don't think slightly faster compiles would justify the investments required.

#3 - 03/06/2018 11:41 PM - Mark Abraham

Roland Schulz wrote:

> It would be nice if testutils would not depend on the full libgromacs but only on the small subset which is needed. This would speed-up compiling of some of the unit tests a lot. An example is simd-test.

simd-test seems like it takes ages to compile because changes to it force recompilation of the SIMD kernels, which are not tested by simd-test. That aspect could be fixed (to save time for people hacking on the SIMD module) by a developer-facing CMake option to disable compilation of the SIMD kernels (at least the NB ones).

But otherwise I agree that the cost of such a change is very high for the value, and particularly so while the code base has many related deficiencies.

#4 - 01/07/2019 01:47 AM - Mark Abraham

Structuring libgromacs as a collection of cmake object libraries would permit testutils to not depend on the whole of libgromacs. Individual test binaries would have to depend on their own subset of the object libraries, so that e.g. simd-test would probably depend on testutils, simd, math, options, and utility, but perhaps nothing else, which would be a bit improvement for developer iteration time.

That's not a big change - instead of appending files to a CMake variable that has everything in libgromacs, we add the files to the module-specifc object library, and libgromacs depends on all of them. This also gives us the leverage we might want to compile modules differently, which would be useful also for building multi-configuration binaries.