GROMACS - Feature #951

Multiple versions of Gromacs (e.g., single and double) in the same library/binary

05/31/2012 12:32 PM - Teemu Murtola

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

Description

One issue that surfaces now and then is to get rid of separate libraries for single and double precision. I haven't yet seen a concrete plan for it, so I'm creating this issue to keep the discussion (since it may be a long time since it actually gets implemented). I'll list some of the problems that need to be solved for this to happen. For the sake of clarity, I'll focus only on the single vs. double precision, MPI vs. no-MPI (or others) may have different problems (although some will likely be the same).

I've only seen people proposing some kind of dual compilation of the source files. Done naively (without any changes to the source), this leads to every function etc. appearing as a duplicate symbol, and things can't be linked. Alternatives to fix this while still keeping everything in one library:

1. Add #ifdefs around each function declaration (proposed in #685). This has a major issue in that it would need to be done for *every* function declaration, as well as for *every* call site. Changing the #ifdef to a macro (e.g., GMX_FUNCTION(func_name)) doesn't change the need to change *every* instance of each function.
2. Use a similar approach as is done on some Microsoft libraries with respect to Unicode/non-Unicode, i.e., #define each class and function name to a different name based on the precision (e.g., #define do_md do_mdS for single and #define do_md do_mdD for double). Requires quite a lot of maintenance, but may be possible to automate...
3. Don't know if there are some linker tricks that could be used to rename the symbols, but that is not going to be portable.
4. If everything is compiled as C++ without extern "C", then functions with a real argument will get different mangled names based on the precision. But there will still be duplicate symbols for functions that don't have real arguments, as well as for class names. Additionally, anonymous namespaces can cause additional issues (I think they may result in symbols that are visible to the linker).
5. With C++, we could perhaps also use different namespaces for the single and double precision versions, which might work. This would require only changing the top-level namespace declarations in headers and source files based on precision, but everything would first need to be put into these namespaces.
6. We could also use C++ templates, where real is a template argument. This solves most of the problems of the dual compilation approach, as we don't need to compile everything twice, and templates can be used also for class names. This is only manageable if we require compiler support for extern template, which is a C++11 feature (but there is hope for compiler support also on older compilers, as I think gcc has had it from 3.4 or something); without it, we would need to make most of Gromacs header-only...

If we only want to have both precisions in the same binary, but separate libraries is ok, we may get away without renaming the symbols. If the binary only needs a few entry points into the library (which will be the case if we do #685, see source:src/programs/g_ana/g_ana.cpp), then we could load dynamically load only one of the precision-dependent libraries. But this may get quite complex in practice, in particular if it needs to be portable...

The title is on purpose a bit vague to make room for discussing the different alternatives.

Related issues:

Related to GROMACS - Feature #1165: Multi-SIMD binaries	**Accepted**

History

#1 - 08/03/2012 01:08 PM - Roland Schulz

I think the best solution would be to have only multiple versions of the same function for those functions that require multiple versions. If we want to have double/single, 4 accelerations and gpu yes/no in one binary, it becomes unmanageable to create a version of every function for every combination (if nothing because of the library size). Thus I think we would want to use templates for all functions which require multiple versions.

#2 - 08/03/2012 08:09 PM - Teemu Murtola

- Description updated

Ah, forgot about templates when I originally wrote the description; was too focused on the dual compilation approach that had just popped up again in

some discussion. Updated the description to also discuss them as an alternative. Also reordered the items such that C++ solutions come after others.

To add template parameters that are not possible to deduce from the function arguments (like GPU on/off), I think that we would want to avoid function templates as much as possible and have template classes instead; otherwise, every function call must specify the template arguments. Also, different acceleration settings still require us to compile different parts of the code with different flags, and templates are not going to make this easier. And to create class templates from the current code is a lot of work, but may pay off in the long run. But I think we should avoid template parameters that alter the behavior of the function too much (like GPU on/off), since it may lead to very messy or hard-to-understand code (if the code is full of partially specialized templates for the different alternatives, I don't think it classifies as "simple C++").

#3 - 12/30/2012 07:17 AM - Teemu Murtola

- Target version set to future

#4 - 01/30/2013 08:08 PM - Roland Schulz

For double/single it would be good to know if it had any performance impact if most variables (80-90%) were always double and only the minority of code existed both in single and double versions. This could simplify the problem significantly.

#5 - 01/07/2019 02:10 AM - Mark Abraham

Current enthusiasm revolves around compiling gmx in a neutral way, and having it dynamically load the correct libgromacs to then do the main work. This could mean we have a forest of libraries for precision, GPU runtime, SIMD support, MPI runtime. With a bit of refactoring of config.h, this would not be a monstrous compilation issue in the minority of cases where one might choose it (e.g. to support a packaged build for a Linux distro). Precision and the SIMD flags affect very many files, but the MPI and GPU runtimes don't affect very many.