GROMACS - Bug #1190

Use of FORCE in setting cached variables

03/13/2013 07:09 PM - Mark Abraham

Status:	New		
Priority:	Normal		
Assignee:	Mark Abraham		
Category:			
Target version:			
Affected version - extra info:		**Difficulty:**	uncategorized
Affected version:	4.6.x		

Description

Roland said in https://gerrit.gromacs.org/#/c/2230/2 in the context of BLAS & LAPACK settings management:

> I don't think there is anything wrong with overwriting GMX_EXTERNAL_*. As with all other variables (e.g. GMX_OPENMP) the user can set what she wants. But if it isn't available or required we change it.

The fact that we have a bunch of CMake code that over-writes user settings so that we know the result of some detection/testing doesn't make that a correct approach.

IMO, the cache is where the user should be able to set things to trigger behaviour. We might use it for values that have to persist from one invocation of cmake to the next (e.g. test results). Whether we are going to do a compilation that uses OpenMP does not need to persist. The underlying calls made by find_package(OpenMP) cache the results of their tests, so there's no reason we need to cache them too. We do need a variable that controls whether to emit code and compilation options for OpenMP, but it does not need to have the same name as the one that lets the user control whether we even try. IMO, HAVE_OPENMP should serve the former role.

If some user setting doesn't make sense, then we have to cope somehow. We should do that by issuing a fatal error, or falling back on a strategy that will work while emitting a message. I don't think changing a user setting is a valid way to give feedback. Someone using non-GUI cmake won't get that feedback anyway. The fact that we do these kinds of changes to cache variables has prompted people in the past to want to write code to see whether the value in a variable is the result of a CMake default, a user setting, or a GROMACS setting. If we use a sensible interface, then we can have a much better idea what is going on. Someone wanting to know if we're going to do an OpenMP build should query the result of the testing.

For example, immediately following the OpenMP checks is the code that sets our compiler flags. The latter depends on whether we are going to do an OpenMP build, and checks the value of GMX_OPENMP. But someone maintaining code like that might have to wonder whether they need to check whether that value is a sanitized user input or not. If the variable's name is HAVE_OPENMP, they know it's not a stupid value the user supplied to GMX_OPENMP. Maybe they want to move the checks to a different place. If the variable is named HAVE_OPENMP, then they know they don't want to move this code before set(HAVE_OPENMP) is called.

For another example, the GPU code emits a warning if there's no OpenMP. Currently it checks GMX_OPENMP. That warning should be emitted on the first pass of CMake. Szilard has made it so with gmx_gpu_setup(). I expect he'd have had a easier time knowing he wanted to defer emitting that warning until after the OpenMP checks if he'd been testing HAVE_OPENMP. Now when he observes HAVE_OPENMP is not yet defined, he can realise that this check belongs after both GPU and OpenMP detection have been resolved. If we get in the habit of using HAVE_XXX then now it becomes reasonable to write a warning-emitting function that notices the dependent uncached variable is undefined and flags that to the developer before it becomes a bug. That's not available if we're reusing a variable the user might have set.

Szilard also has to have a forest of variables for GPU detection. He caches GMX_GPU_DETECTION_DONE, which is analogous to how the find_xxx() routines work. That's great. The first thing gmxManageGPU does is to set GMX_GPU_AUTO to see whether the user made an active selection of GPU usage, precisely because an earlier pass of the detection routine could have set GMX_GPU, and he can only tell the difference early on. My general recommendation would be for us to use GMX_GPU as the user input, and HAVE_GPU as the "yes, it's good to use" output, but Szilard's solution has converged on that functionality by a different name.

People don't write C code that does

```
void maybe_change_x(int *x) {
  if(complex_condition) (*x) += 5;
}
```

```
x=strtol(argc[1],NULL,10);
maybe_change_x(&x);
```

if they're later going to wonder whether x got changed or not. They'd write y = operation(x) if later x and y might play semantically different roles.

A good configuration summary needs to be able to report to the user that "you asked for OpenMP (even if it was by default), we know you can't have it". But we can't use the same variable to do that.

> Because of KISS I think it is better to have less variables if possible. Given that I don't see how this improves anything I prefer the original.

For example, https://gerrit.gromacs.org/#/c/2230/2 means that we have the non-cache variable HAVE_EXTERNAL_BLAS. Now code can query that and know that the value is the result of a test - we will link to an external BLAS and that will work. Before this patch, GMX_EXTERNAL_BLAS could have been set by the user and never used because HAVE_LIBMKL was true. So, (for example) a configuration summary would have to reproduce half the logic of the BLAS setting in order to report what was going on. It couldn't just look at GMX_EXTERNAL_BLAS. You could argue the previous behaviour was buggy, but if it takes organizing the code into one place to see there is a possible bug, then that's a no-brainer.

Also we now have the option to tag HAVE_EXTERNAL_BLAS with information about that external BLAS (MKL, Accelerate, GMX_USER_BLAS, detected it) by setting a CMake property. A configuration summary can make use of that. Now it can be a dumper and does not have to have logic to maintain. Having to work around convoluted logic is why I am not yet keen on a configuration summary - it'd just be plastering over problems (even if it could be made to work well). Admittedly, we could tag GMX_EXTERNAL_BLAS also, but it's already doing too much work.

History

#1 - 03/13/2013 07:44 PM - Teemu Murtola

Just commenting on the general aspects for now.

I would strive for as simple an approach as possible, which in my mind would be that we don't try to change anything the user provides. If the user provides a mutually conflicting set of options, then that is a fatal error. If the user does not provide all the options, we can then make a guess for what they probably would want, but don't try to be overly smart about it: if they later change the other options that would influence the default, then I don't think we need to try to reiterate our guess.

With this approach, in nearly all cases it doesn't really matter whether we use GMX_X for multiple purposes in the cmake code, because we know that it is not going to change, ever, unless the user changes it. I think this makes maintenance even simpler than a separate HAVE_X flag. In particular, it reduces the need to do any complex reporting to the user, since they always get what they asked for.

Convoluted dependencies between the input variables can still require some trickery, but we could also try to get rid of some of these dependencies (e.g., GMX_MPI and GMX_THREAD_MPI don't probably want to ever coexist, so they could be merged into one variable).

PS. The last time I tried (with cmake 2.8.10, IIRC), it was not possible to set any custom properties on cache variables (this one I tried), and no properties at all on non-cached variables (this one is based on reading documentation only), so that does not solve anything.

#2 - 03/13/2013 08:27 PM - Roland Schulz

Mark Abraham wrote:

> If some user setting doesn't make sense, then we have to cope somehow. We should do that by issuing a fatal error, or falling back on a strategy that will work while emitting a message. I don't think changing a user setting is a valid way to give feedback. Someone using non-GUI cmake won't get that feedback anyway.

Yes, in general, we should issue a message. I think in this special case it wasn't done because the blas/lapack lib doesn't really matter (and would be shown often because we try to use an external if possible). And someone who really cared could look it up.

> The fact that we do these kinds of changes to cache variables has prompted people in the past to want to write code to see whether the value in a variable is the result of a CMake default, a user setting, or a GROMACS setting.

The one case I remember was in the particular weird case of CFLAGS. Because it is done partly in cmake and in a somewhat odd way. Apart from that I don't remember us having that problem.

> For example, immediately following the OpenMP checks is the code that sets our compiler flags. The latter depends on whether we are going to do an OpenMP build, and checks the value of GMX_OPENMP. But someone maintaining code like that might have to wonder whether they need to check whether that value is a sanitized user input or not. If the variable's name is HAVE_OPENMP, they know it's not a stupid value the user supplied to GMX_OPENMP. Maybe they want to move the checks to a different place. If the variable is named HAVE_OPENMP, then they know they don't want to move this code before set(HAVE_OPENMP) is called.

The sanitizing of user input is not an issue because cmake automatically converts everything to true/false. If we split it into GMX_... (meaning "user wants") and HAVE_... (wants+is available), and it is not available, than the user gets a warning/status every time she runs cmake. I think getting only once a warning makes more sense. Especially for something unimportant as the blas/lapack library. How do you avoid printing the warning more than once if you have these 2 independent variables?

> For another example, the GPU code emits a warning if there's no OpenMP. Currently it checks GMX_OPENMP. That warning should be emitted on the first pass of CMake. Szilard has made it so with gmx_gpu_setup(). I expect he'd have had a easier time knowing he wanted to defer emitting that warning until after the OpenMP checks if he'd been testing HAVE_OPENMP. Now when he observes HAVE_OPENMP is not yet defined, he can realise that this check belongs after both GPU and OpenMP detection have been resolved. If we get in the habit of using HAVE_XXX then now it becomes reasonable to write a warning-emitting function that notices the dependent uncached variable is undefined and flags that to the developer before it becomes a bug. That's not available if we're reusing a variable the user might have set.

That makes sense. But isn't really an issue for blas/lapack. So I'm not sure it is the best place to try out this new approach. Also I'm wondering whether changing all of cmake to have two variables for each option is something we want to do for 4.6. It might be better to only put this into 5.0. Or do you still want to add a configuration summary to the 4.6 cmake?

> For example, https://gerrit.gromacs.org/#/c/2230/2 means that we have the non-cache variable HAVE_EXTERNAL_BLAS. Now code can query that and know that the value is the result of a test - we will link to an external BLAS and that will work. Before this patch, GMX_EXTERNAL_BLAS could have been set by the user and never used because HAVE_LIBMKL was true.

That is bug. If MKL/Accel is used those should be forced to true. Or give an error if they are not true. Fixing that would make it as easy to write a configuration summary as with 2 variables.

#3 - 03/14/2013 11:29 AM - Mark Abraham

Teemu Murtola wrote:

> Just commenting on the general aspects for now.
>
> I would strive for as simple an approach as possible, which in my mind would be that we don't try to change anything the user provides. If the user provides a mutually conflicting set of options, then that is a fatal error. If the user does not provide all the options, we can then make a guess for what they probably would want, but don't try to be overly smart about it: if they later change the other options that would influence the default, then I don't think we need to try to reiterate our guess.

One aspect of trying not to change things the user inputs is that we need to be sure we upgrade messages when we can't comply to at least warnings. Currently, there are probably some messages that are only "status" messages, and ccmake users don't see most of those. One advantage of the old approach of changing the value of variables like GMX_OPENMP is that a ccmake user does get some feedback when they see their variable change value - but a cmake user only gets something when they scroll through their output.

> With this approach, in nearly all cases it doesn't really matter whether we use GMX_X for multiple purposes in the cmake code, because we know that it is not going to change, ever, unless the user changes it. I think this makes maintenance even simpler than a separate HAVE_X flag. In particular, it reduces the need to do any complex reporting to the user, since they always get what they asked for.

That works only if we issue a fatal error every time we can't comply with a specific request. If the compiler doesn't support a request (e.g. use OpenMP) then we can't proceed without setting some variable that controls/describes how the code is emitted and compiled. If we don't change user inputs and don't have a second variable then we're stuck. That could be a feature, but only if we have machinery that lets us know that GMX_OPEN=on was a user request, not a default. In turn, that requires that option(GMX_OPENMP...) comes after the management code, so that we can tell the difference between a silent default and a choice somebody made. This is similar to how Szilard manages the GPU detection at the moment.

If the user made no request about OpenMP, and the default is on, and the compiler can't do it, then it makes sense to switch it off and move on (probably with a warning).

> Convoluted dependencies between the input variables can still require some trickery, but we could also try to get rid of some of these dependencies (e.g., GMX_MPI and GMX_THREAD_MPI don't probably want to ever coexist, so they could be merged into one variable).

Sort of. ThreadMPI is no longer just a parallelism layer, but rather a low-level detection library also. The code that manages affinity settings calls ThreadMPI routines even from real-MPI builds. I expect this will cause a lot of pain some time this year.

> PS. The last time I tried (with cmake 2.8.10, IIRC), it was not possible to set any custom properties on cache variables (this one I tried), and no properties at all on non-cached variables (this one is based on reading documentation only), so that does not solve anything.

Hmmm, I'll have to test that, then.

#4 - 03/14/2013 12:25 PM - Mark Abraham

Roland Schulz wrote:

> Mark Abraham wrote:
>
>> The fact that we do these kinds of changes to cache variables has prompted people in the past to want to write code to see whether the value in a variable is the result of a CMake default, a user setting, or a GROMACS setting.
>
> The one case I remember was in the particular weird case of CFLAGS. Because it is done partly in cmake and in a somewhat odd way. Apart from that I don't remember us having that problem.

I think I recall it from Teemu's draft configuration summary. I could be wrong.

>> For example, immediately following the OpenMP checks is the code that sets our compiler flags. The latter depends on whether we are going to do an OpenMP build, and checks the value of GMX_OPENMP. But someone maintaining code like that might have to wonder whether they need to check whether that value is a sanitized user input or not. If the variable's name is HAVE_OPENMP, they know it's not a stupid value the user supplied to GMX_OPENMP. Maybe they want to move the checks to a different place. If the variable is named HAVE_OPENMP, then they know they don't want to move this code before set(HAVE_OPENMP) is called.
>
> The sanitizing of user input is not an issue because cmake automatically converts everything to true/false.

Sorry, bad choice of words by me. I meant that setting the compiler flags should only happen after GMX_OPENMP reflects what we're going to do. In all cases it should follow the OpenMP management code, but the variable change I suggest makes that dependency explicit.

> If we split it into GMX_... (meaning "user wants") and HAVE_... (wants+is available), and it is not available, than the user gets a warning/status every time she runs cmake. I think getting only once a warning makes more sense. Especially for something unimportant as the blas/lapack library. How do you avoid printing the warning more than once if you have these 2 independent variables?

Good point, sort of. If the user asked for external BLAS and there isn't one, then a repeated warning serves a purpose. We know we haven't been able to comply and the user should make a decision about it. If the user asked for default behaviour (which is external BLAS if we find one) then the warning should not repeat. This requires that the code be sensitive to whether GMX_EXTERNAL_* was set by someone (user or us), or is just the default (per how I interpret Teemu's suggestion above).

>> For another example, the GPU code emits a warning if there's no OpenMP. Currently it checks GMX_OPENMP. That warning should be emitted on the first pass of CMake. Szilard has made it so with gmx_gpu_setup(). I expect he'd have had a easier time knowing he wanted to defer emitting that warning until after the OpenMP checks if he'd been testing HAVE_OPENMP. Now when he observes HAVE_OPENMP is not yet defined, he can realise that this check belongs after both GPU and OpenMP detection have been resolved. If we get in the habit of using HAVE_XXX then now it becomes reasonable to write a warning-emitting function that notices the dependent uncached variable is undefined and flags that to the developer before it becomes a bug. That's not available if we're reusing a variable the user might have set.

Teemu's suggestion also helps solve this one. If the GPU+OpenMP warning was being emitted too early, then GMX_OPENMP might be undefined(=default) which can emit a developer-level warning. However if GMX_OPENMP has a value (whether from user input or the result of subsequent detection in an earlier iteration of CMake) then the value should be used. That would mean defining gmx_check_option(option result), so that there's a hook for the warning. The actual logic would need to use the value returned in the result variable.

> That makes sense. But isn't really an issue for blas/lapack. So I'm not sure it is the best place to try out this new approach. Also I'm wondering whether changing all of cmake to have two variables for each option is something we want to do for 4.6. It might be better to only put this into 5.0. Or do you still want to add a configuration summary to the 4.6 cmake?

I'd like a 4.6 configuration summary based on a good CMake design that doesn't change the user interface ;-) But I'm not going to get all of that!

Various aspects of the design already have the two-variable approach. For example,

- GMX_X11 triggers find_package to set X11_FOUND, which sets the (unused) HAVE_X11 variable. src/ngmx/CMakeLists.txt checks X11_FOUND directly. If detection fails, the default behaviour is to silently accept that and not change GMX_X11. This should get some more verbosity.
- In master, GMX_XML triggers find_package to set LIBXML2_* (which get used various places). If detection fails, the default behaviour is to silently accept that and not change GMX_XML. Likewise should be more verbose.
- GMX_MPI triggers various tests that end up with GMX_LIB_MPI being set, or a fatal error.

The first two would be fine in the model of "don't ever change explicit user settings" implemented by being sensitive to whether the initial value is undefined. If the user explicitly asked for X11 and it isn't there, it should not be possible for the user to configure, build and only then find that ngmx is missing. This would mean that HAVE_X11, X11_FOUND and LIBXML2_FOUND disappear in favour of using the GMX_* variable.

As far as configuration summary goes, implementing one that reports the result is not too difficult. I'm hoping there's a good way to trigger writing it out at "generation" time. If we can tag the variable each time we set it (e.g. when setting a default, or because of a failed test) then we can also report to the user why the variable has that value. This will hopefully make the configuration process less mysterious. This would probably mean defining a custom gmx_set_option() function that maintained the tags.

> > For example, https://gerrit.gromacs.org/#/c/2230/2 means that we have the non-cache variable HAVE_EXTERNAL_BLAS. Now code can query that and know that the value is the result of a test - we will link to an external BLAS and that will work. Before this patch, GMX_EXTERNAL_BLAS could have been set by the user and never used because HAVE_LIBMKL was true.
>
> That is bug. If MKL/Accel is used those should be forced to true. Or give an error if they are not true. Fixing that would make it as easy to write a configuration summary as with 2 variables.

#5 - 03/23/2013 01:46 PM - Teemu Murtola

I'm still not convinced that we would need to know the origin of the value of some GMX_XXX with my proposal. If we organize the code such that for each option, there is

1. first, a bunch of code that determines the default guess (if the option is not present in the cache, or if the default is required for some other purpose),
2. then, a set(GMX_XXX ${GMX_XXX_DEFAULT} CACHE TYPE "Description")
3. followed by any code that depends on GMX_XXX,

and have a policy that everything that would require changing GMX_XXX after the set(GMX_XXX) is a fatal error, then any warnings about the defaults can be emitted in the code that determines the default (and that even easily behaves the "natural" way, only issuing the warning once). Similarly, if we want to avoid our default causing issues (e.g., the compiler not supporting OpenMP), then checking for those issues should be part of determining the default. We should decide whether we want, e.g., OpenMP so badly that they need to consciously disable it to avoid a fatal error, or whether the default should be off (with a possible warning) in those cases that we can detect it won't work.

For the types of messages, in addition to "status" and "warning", CMake has a (very unintuitive, as it is produced by a simple message("Text")) "important" message category that is not a warning, but still is presented to the user when running ccmake. This is what I was using the configuration summary draft (in addition to real warnings and errors).

#6 - 04/12/2013 06:52 PM - Mark Abraham

Teemu Murtola wrote:

> I'm still not convinced that we would need to know the origin of the value of some GMX_XXX with my proposal. If we organize the code such that for each option, there is
>
> 1. first, a bunch of code that determines the default guess (if the option is not present in the cache, or if the default is required for some other purpose),
> 2. then, a set(GMX_XXX ${GMX_XXX_DEFAULT} CACHE TYPE "Description")
> 3. followed by any code that depends on GMX_XXX,
>
> and have a policy that everything that would require changing GMX_XXX after the set(GMX_XXX) is a fatal error, then any warnings about the defaults can be emitted in the code that determines the default (and that even easily behaves the "natural" way, only issuing the warning once). Similarly, if we want to avoid our default causing issues (e.g., the compiler not supporting OpenMP), then checking for those issues should be part of determining the default. We should decide whether we want, e.g., OpenMP so badly that they need to consciously disable it to avoid a fatal error, or whether the default should be off (with a possible warning) in those cases that we can detect it won't work.
>
> For the types of messages, in addition to "status" and "warning", CMake has a (very unintuitive, as it is produced by a simple message("Text")) "important" message category that is not a warning, but still is presented to the user when running ccmake. This is what I was using the configuration summary draft (in addition to real warnings and errors).

Sounds good. I have updated https://gerrit.gromacs.org/2230 to follow this approach, rather than my original one. I experimented with whether the main logic should

- set the default or
- post-manage the main variable,
  and decided the former works best.

Note the subtle way we can trigger internal linear algebra without having to use FORCE on a cached variable where the user set GMX_EXTERNAL_*=ON. A configuration summary can report that. We'd have to do something else if we also wanted to report the things the user explicitly set (since they might differ).

Assuming things work out, I think we should move in the direction of the policy in Teemu's post 5 above.

#7 - 06/15/2014 05:05 AM - Roland Schulz

- Affected version set to 4.6.x

Anything left to do for this?

#8 - 06/17/2014 06:05 AM - Teemu Murtola

At least the OpenMP management code still has bunch of

```
  set(GMX_OPENMP OFF CACHE ... FORCE)
```

calls (while at it, the code that unsets various OpenMP variables at the end of the main CMakeLists.txt should probably be moved to gmxManageOpenMP.cmake, since at least that currently seems to be the only place where GMX_OPENMP is getting changed).

I couldn't spot any other obvious places with git grep -C1 FORCE -- CMakeLists.txt '*.cmake', but not all the issues mentioned here are actually associated with a FORCE. At least GMX_LOAD_PLUGINS is also getting silently ignored if some detection fails.

#9 - 06/17/2014 06:09 AM - Teemu Murtola

And there is a lot of CMake code that does not follow the pattern from my comment 5; at least the GMX_GPU code could probably be simplified using this.