GROMACS - Feature #2840

Test that functionality does not compile that is not supposed to.

01/25/2019 12:34 PM - Christian Blau

<table>
<thead>
<tr>
<th>Status:</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td></td>
</tr>
<tr>
<td>Category:</td>
<td></td>
</tr>
<tr>
<td>Target version:</td>
<td>future</td>
</tr>
<tr>
<td>Difficulty:</td>
<td>uncategorized</td>
</tr>
</tbody>
</table>

Description

Classes/templates yield essential compiler errors to prevent misuse.

However testing that code changes do not unintentially allow code to compile that was explicitly forbidden from it requires testing the compilation for which we don't have functionality at the moment.

The following patch has a solution for this from Roland Schulz that ended up not being used, [[https://gerrit.gromacs.org/#/c/8192/14/src/gromacs/gpu_utils/tests/CMakeLists.txt]]

```cpp
function(gmx_compile_test)
    if(NOT GMX_DEVELOPER_BUILD)
        return() #Users might run tests on a different machine
    endif()
    set(NAME ${ARGV0})
    set(SOURCE ${ARGV1})
    set(ERRORMSG ${ARGV2})
    add_library(${NAME} OBJECT EXCLUDE_FROM_ALL ${SOURCE})
    add_test(NAME ${NAME} COMMAND ${CMAKE_COMMAND} --build ${CMAKE_BINARY_DIR} --target ${NAME})
    set_tests_properties(${NAME} PROPERTIES PASS_REGULAR_EXPRESSION ${ERRORMSG})
endfunction()

gmx_compile_test(HostAllocatorCompileTest "hostallocator_compile.cpp"
             "This allocator policy doesn't support copy construction")
```

History

#1 - 01/25/2019 02:34 PM - Mark Abraham

There are a few other approaches out there.

Those at [https://stackoverflow.com/questions/30155619/expected-build-failure-tests-in-cmake](https://stackoverflow.com/questions/30155619/expected-build-failure-tests-in-cmake) leverage that ctest can be taught to understand that a test would fail, but it seems to require making subdirectories with source files, which would get messy for using in multiple places.

That of [https://petriconi.net/?p=118](https://petriconi.net/?p=118) looks good. We can have have the helper types in a common header in src/testutils, and hopefully have test code that looks like

```cpp
#include "foo.h"
#include "testutils header.h"

// The specialization for int shall be forbidden
template <>
struct Foo<int> : public disable_usage<Foo>
{
};

EXPECT_TRUE(is_disabled_from_usage<Foo<int>>::value);
```

#2 - 01/25/2019 02:35 PM - Mark Abraham

- Description updated