GROMACS - Task #948

C++ thread synchronization primitives

05/30/2012 05:33 PM - Teemu Murtola

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

Description

For 5.0, we will need C++ implementations of mutexes etc. I see the following alternatives:

1. Use those that C++11 provides these by default. Compiler support may be a problem.
2. Use something from boost. There has to be something there. :) But I haven't checked what are it's prerequisites, how it works etc.
3. Write our own wrappers on top of thread_mpi (or as part of thread_mpi).
4. Use some other external cross-platform library.
5. Write our own cross-platform implementation from scratch.

With the amount of resistance I'm seeing for external dependencies, I bet that the third could be a reasonable alternative.

Some requirements that I'd like to see for the C++ implementation:

- Even if/when we don't use the C++11 versions, it would be nice to model our interface similar to that. It should be quite well thought out.
- RAII, it's a must with exceptions.
- No need for conditional compilation in using code. If thread synchronization is not necessary for a build, the implementation should provide no-op versions of all classes/functions. If necessary for performance reasons, there could be some conditional compilation in the headers to inline the no-op versions.

Related issues:

Related to GROMACS - Feature #868: Implement parallelization support to analy...	**In Progress**	
Related to GROMACS - Bug #936: ThreadMPI on VS2008 doesn't compile	**Rejected**	**04/29/2012**

Associated revisions

Revision d39f6dad - 07/04/2012 01:24 AM - Teemu Murtola

Always build thread_mpi threading into libgromacs.

This allows using basic thread synchronization primitives without
conditional compilation.
Adjusted the build system to build only the basic thread synchronization
files instead of full thread_mpi if GMX_THREAD_MPI is off.
This required splitting tMPI_Malloc and friends from tmpi_init.c into a
separate file (since tmpi_init.c depends on most other files), and also
moving TMPI_COMM_WORLD to the new file.
Also removed a few unused variables from atomic.h to avoid cppcheck
warnings.

Part of #948.

Change-Id: I3c6bd2cf084acd4a4e881b99a07bc3a4c878727e

Revision cb6f4849 - 11/26/2013 09:27 PM - Teemu Murtola

Remove #ifdef GMX_THREAD_MPI for basic thread safety.

Simplifies the code, and at least most of this should have zero impact
on performance. Basic thread support from thread-MPI is already
required for compilation anyways even with GMX_THREAD_MPI off.
This removes about 70% of the #ifdefs, making it clearer what is
different between MPI and thread-MPI implementation in Gromacs.

Also set MPI_IN_PLACE_EXISTS with GMX_THREAD_MPI instead of checking for GMX_THREAD_MPI separately each time. Adjust one #ifdef in network.c to replace an GMX_LIB_MPI made unnecessary by this with GMX_MPI.

Related to #948 and #1320.

Change-Id: I03a54b0bffde090100afe5ed2e91c184af36c189

History

#1 - 05/30/2012 07:32 PM - Roland Schulz

1. E.g. HP doesn't support either atomic or thread local: http://wiki.apache.org/stdcxx/C%2B%2B0xCompilerSupport .
2. Boost has thread. It depends on move for lock move semantic (boost/thread/locks.hpp). And that it turn depends on MPL which is large. Not sure whether it makes sense to make a subset with out the move semantic to remove the MPL dependency. (MPL dependency is also intruduced through date_formatting but that be easy to remove). It doesn't depend on move with C++11 Rvalue support but that isn't better than 1.

How important is move semantic for locks. Especially given Teemu's very sensible requirement that the interface should be similar to C++11? If we need move semantic an own C++ wrapper will require boost::move (for non C++11 compilers) even for options 3-5. If we can do without it might be an option to make a boost/thread subset.

#2 - 05/30/2012 07:39 PM - Erik Lindahl

One thing to consider is that mutexes are usually quite slow since they do full memory fence synchronization and yield the CPU. This is important for applications where one is running many more threads than processors, and although it requires much more code it is the safe and therefore typically the default option for thread classes.

However, for performance-sensitive code where we have one core per thread we frequently prefer busy spinlocks. This has worked very well with threadmpi, so even if we support normal mutexes too, I think it is important that we keep that functionality in any C++ interface too.

#3 - 05/30/2012 08:06 PM - Teemu Murtola

I don't now have the time to look it up, but it may be difficult to implement exact C++11 semantics without move, but something similar should be possible (at least if we give up on some features). As for actual move support (moving the ownership of the lock between contexts), I hope that we could live without it.

As for how things are implemented internally, I don't really have any opinion. But at some point in the C++ conversion, we will need to start touching those parts of the code that currently use mutexes, and for that we need a C++ alternative that doesn't require jumping through hoops to make things work correctly with exceptions (some of the mutexes can probably also be refactored away, but some are still necessary).

#4 - 05/30/2012 10:07 PM - Sander Pronk

There are two (actually three if we count OpenMP) entirely different thread synchronization types in Gromacs:
- synchronization through mutexes, condition variables (used when performance isn't critical) are currently done through thread_mpi which implements a pthreads-style API. This is an obvious candidate for somewhat improved C++ interfaces.
- atomic operations for high-performance lock-free synchronization. I've looked at what C++11 has to offer (and it thankfully does offer an API for this): it is basically the same functionality we already have in thread_mpi, with pretty much the same function-based programming API - this is basically because the supported operations are by definition low-level atomic operations that operate on primitive types. Honestly, I don't see any case for switching to C++11.

For the mutex/condvar synchronization there would be a slight advantage in using a C++11-style approach, but the gains are not great. When designing object-oriented code that is thread-safe, one has to think mostly about the interface and the guarantees that it makes anyway. I guess we can provide thin object wrappers around mutexes and condvars without too much work.

#5 - 05/31/2012 09:07 AM - Teemu Murtola

I was not implying that we should reimplement the C++11 standard library just for the sake of it. Instead, since there is still *some* C++ code that we just need to write, my suggestion was to use C++11 as a guideline whenever there is a design/naming decision to make etc.

And I really think that we need a C++ approach for locks, at the minimum. Or which one of these would you prefer to read/write/maintain (just a simple example with not so much control flow)?

Based on our current C interface:

```
bool f()
{
#ifdef GMX_THREAD_MPI
    tMPI_Thread_mutex_lock(&mutex);
#endif
    try {
        if (check_some_condition()) {
            do_something();
#ifdef GMX_THREAD_MPI
            tMPI_Thread_mutex_unlock(&mutex);
#endif
```

```
            return true;
        }
    }
    catch (...) {
#ifdef GMX_THREAD_MPI
        tMPI_Thread_mutex_unlock(&mutex);
#endif
        throw;
    }
#ifdef GMX_THREAD_MPI
    tMPI_Thread_mutex_unlock(&mutex);
#endif
    return false;
}
```

C++11:

```
bool f()
{
    lock_guard<mutex> lock(our_mutex);
    if (check_some_condition()) {
        do_something();
        return true;
    }
    return false;
}
```

#6 - 05/31/2012 09:20 AM - Sander Pronk

First of all, we should get rid of the
#ifdef GMX_THREAD_MPI
for basic initialization thread safety, and only keep them for cases where there's actual thread_mpi-specific code.
There's essentially no cost associated with having the mutexes in case of no threads, and the thread_mpi pthreads api works for all platforms single-threaded Gromacs runs on.

That being said, the second example you put is still a pretty thin wrapper around posix threads. It shouldn't be too much work to put one together that is a subset of the C++11 thread, mutex, condition_variable classes.

#7 - 05/31/2012 09:27 AM - Teemu Murtola

I agree on both points (the removal of the #ifdefs is even included in the original description). We can simply implement our C++ classes only for cases we actually need, but to get started, at least the basic mutex and lock implementations should be in place.

#8 - 05/31/2012 09:32 AM - Sander Pronk

OK - I've looked at the C++11 interface and I'll use that to create a mutex, condition_variable and thread class in thread_mpi. I'll think about how to do thread-local storage but that seems a little more difficult to do well (and is of lower priority).

#9 - 05/31/2012 11:09 AM - Teemu Murtola

- Assignee set to Sander Pronk

Great! I think that mutex and something like lock_guard would be the highest-priority items. For example, if/when #950 needs to touch the behavior of Program()/ShortProgram(), it would be nice to convert those into C++ internally (in particular since I've already added a C++ class that has similar functionality), but it would be nice to keep the mutex around the global program information.

Assigned to you, Sander, since you said you'll do at least some part of it.

#10 - 05/31/2012 12:05 PM - Sander Pronk

Check out
https://gerrit.gromacs.org/1060
for a C++11 compatible mutex & lock_guard implementation in thread_mpi/mutex.h

#11 - 06/04/2012 12:49 PM - Teemu Murtola

Tried now to commit some code that actually uses the new classes without conditional compilation (https://gerrit.gromacs.org/#/c/1068/), but it seems that thread_mpi does not build with VS2008 or with ICC on Windows (https://gerrit.gromacs.org/#/c/1069/). cppcheck also gives a few warnings: http://jenkins.gromacs.org/job/cppcheck_Gerrit_master/284/.

#12 - 06/04/2012 04:10 PM - Sander Pronk

See http://redmine.gromacs.org/issues/948 for a new bug report on this issue.

#13 - 05/13/2014 10:04 AM - Mark Abraham

- Target version changed from 5.0 to 5.x

#14 - 07/11/2016 08:24 PM - Mark Abraham

- Target version deleted (5.x)

I'm in favour of using C++11 things wherever possible