GROMACS - Task #1828

Exception handling in mdrun

09/17/2015 08:18 PM - Teemu Murtola

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

Description

There are several changes now in Gerrit that introduce real C++ code deeper within mdrun (not just C code compiled as C++). And it is difficult to keep such code free from exceptions; both changes now in Gerrit (https://gerrit.gromacs.org/4734, https://gerrit.gromacs.org/4993) have code that can throw, and the latter also has explicit throws. Neither change does anything to handle those exceptions. We need to decide how to handle these exceptions. In https://gerrit.gromacs.org/4993 Erik said that requiring some action would hamper C++ adoption, but a decision to not do anything also needs to be explicitly made, with its consequences understood.

Options:

1. Do nothing. Let the code throw exceptions, and only have the current, single top-level catch-all (in gmx.cpp and/or mdrun_main.cpp). This will not catch exceptions thrown from other threads, and this will not do anything for the other threads that might be executing when the main thread throws. Also, it is not allowed to throw exceptions from OpenMP regions (I have no idea what happens if you do). If we do this, we really need to accept that anything can happen on errors:
   - You might get an error message from std::terminate or such, which most likely contains no information that would help in identifying what actually caused the issue.
   - mdrun might deadlock.
   - Some code might eat the exception, and execution would continue, but would silently skip some computation. Hopefully there is no such implementation out there, but who knows...
2. Wrap all C++ code in a try/GMX_CATCH_ALL_AND_EXIT_WITH_FATAL_ERROR as close as possible to the C++ call, such that all code within the block is exception-safe. This is what all other code does.
3. Do the above wrapping at each thread boundary instead. This requires extra care with, e.g., barriers or other communication that might get skipped because of the exception.
4. Do proper handling at the thread boundaries, throwing the exception after the other threads have ended, such that the exception comes all the way through to the top-level catch-all.

With the last two, we also need to decide that what level of exception safety is acceptable on the paths that are covered: Are memory leaks OK? How about deadlocks that would trigger if there weren't an immediate exit?

Note that if we plan to use TBB, it will do the last part a bit more automatically when compiled with C++11, but that is not going to solve any problems soon.

Related issues:

Related to GROMACS - Task #1411: Future of thread_mpi	**New**
Related to GROMACS - Task #1829: Future of thread level parallelism	**New**

Associated revisions

Revision 4fb0842d - 10/05/2015 09:10 PM - Erik Lindahl

Added try/catch statements in OpenMP regions and tMPI start

OpenMP cannot handle exceptions when they are thrown inside an
OpenMP region, but caught outside of it. Instead we catch these
exceptions inside the region and exit with a fatal error. Not
beautiful, but at least we will know what happened, and exceptions
should only occur in catastrophic scenarios in parallel regions
(where we already call the fata error handler). For trivial OpenMP
blocks that should never through there are comments noting this, so
we remember to add a try/catch pair if necessary in the future.
There is also a new catch-all in the tMPI entry point function.

Refs #1828.

Change-Id: I52103a076b7f0a5e8ac1f740160e4de0fa97f638

History

#1 - 09/17/2015 08:37 PM - Berk Hess

This looks like a pretty nasty issue.
I hope it's feasible to avoid throwing exceptions in any thread-parallel compute region, both for code simplicity and performance reasons. If we do memory allocation we probably can't avoid throwing, but it shouldn't be much effort to wrap the few thread parallel initialization/reallocation routines. Even if such a scheme is possible, it requires careful and tedious checking if any code can be executed thread parallel and throw exceptions, unless there are tools for this.

#2 - 09/17/2015 09:41 PM - Roland Schulz

- Related to Task #1411: Future of thread_mpi added

#3 - 09/17/2015 09:47 PM - Roland Schulz

- Related to Task #1829: Future of thread level parallelism added

#4 - 09/27/2015 11:34 PM - Erik Lindahl

This might actually be a strong argument for TBB, and against OpenMP. If we move to a task-parallel setup where pretty much all code runs as tasks, there is simply no way we'll be able to avoid exceptions entirely.
It sounds as if it might be worth doing some simple checks with OpenMP. What the standard says is one thing, but if it works in practice for the normal compilers we use, then we can probably live with it during a transition period.

Wrapping close to the call sounds like a good option when we can do it, but for cases where we e.g. draw tons of random numbers in tight loops it will have to go on a higher level.

#5 - 10/01/2015 08:42 PM - Erik Lindahl

Just tried it with a trivial test program and g++-4.9, and the problem is real.

Good news first: It seems to work fine to catch the exception inside an OpenMP region.

Bad news: If the try/catch clause is outside the OpenMP region and throw a single exception, it is not caught but terminate is called and at least we see the exception message.
Worse news: If exceptions are thrown from multiple OpenMP threads at roughly the same time, we get random errors about recursive calling of terminate, and don't even see the exception message.

Short-term, I can imagine two solutions:

1) We try to catch exceptions inside the openmp loops.
2) For code like the RNGs, we can have a check for _OPENMP and use fatal errors instead of exceptions for those rare cases.

Long-term, it seems pretty much impossible to avoid exceptions (or catch them all) if we move to extensive task parallelism. The core mdrun loop is one thing, but we'll probably be using tasks extensively in IO, maybe load balancing/communication, all our tools, etc - if we are to avoid exceptions entirely in all these parts of the code and their dependencies, we would pretty much be back to the alternative of never allowing any exceptions?

TBB just started to look a whole lot more tempting...

#6 - 10/01/2015 09:10 PM - Teemu Murtola

Erik Lindahl wrote:

> Wrapping close to the call sounds like a good option when we can do it, but for cases where we e.g. draw tons of random numbers in tight loops it will have to go on a higher level.

Does it? In principle, in modern implementations, the code to catch an exception is "free" unless the exception actually gets caught. The only thing where it might matter is for compiler optimization (where it may place some limitations on what code it can move where), but I can't really say how much. But after compilation, the normal execution path doesn't even know there is some code somewhere that will catch the exceptions. The cost of the exception infrastructure is elsewhere (in the executable size, and in the cost of actually throwing the exception).

#7 - 10/03/2015 05:31 PM - Erik Lindahl

I gave this some more thought (and code-looking), and based on that I would favor that we put the try-catch section where we have the openmp-pragmas.

The main reason for that is that with more and more C++ code it's going to be a nightmare to track not only every single statement that might throw. In principle any code in GROMACS might suddenly occur inside an openmp loop if somebody adds that type of parallelization or tasks a dozen function calls further up.

The advantage of doing it based on the openmp loops is that for many (most?) of these loops it will be trivial to see if exceptions cannot happen, and then we don't have to worry about it. That seems a whole lot easier than tracking all the rest of the code?

#8 - 10/03/2015 07:39 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #1828.
Uploader: Erik Lindahl (erik.lindahl@gmail.com)
Change-Id: I52103a076b7f0a5e8ac1f740160e4de0fa97f638
Gerrit URL: https://gerrit.gromacs.org/5168

#9 - 10/03/2015 07:49 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #1828.
Uploader: Erik Lindahl (erik.lindahl@gmail.com)
Change-Id: I79edc458f839db2363dc9ca371cdaddd5106b63b
Gerrit URL: https://gerrit.gromacs.org/5170

#10 - 10/06/2015 03:49 PM - Szilárd Páll

Erik Lindahl wrote:

> The advantage of doing it based on the openmp loops is that for many (most?) of these loops it will be trivial to see if exceptions cannot happen, and then we don't have to worry about it. That seems a whole lot easier than tracking all the rest of the code?

Looks like the question has been (implicitly) answered by a change that's already merged. Does this imply that the policy is to always use try-catch around OpenMP regions unless known to be safe to omit? It would be good to document this and inform people, wouldn't it?

#11 - 10/07/2015 11:36 AM - Mark Abraham

Szilárd Páll wrote:

> Erik Lindahl wrote:
>
> > The advantage of doing it based on the openmp loops is that for many (most?) of these loops it will be trivial to see if exceptions cannot happen, and then we don't have to worry about it. That seems a whole lot easier than tracking all the rest of the code?
>
> Looks like the question has been (implicitly) answered by a change that's already merged. Does this imply that the policy is to always use try-catch around OpenMP regions unless known to be safe to omit? It would be good to document this and inform people, wouldn't it?

Indeed. Sounds like a good first addition to the developer guide. ;-)