GROMACS - Feature #1123

binary incompatibility

01/17/2013 11:27 AM - Mark Abraham

Status:	Rejected	
Priority:	Normal	
Assignee:		
Category:		
Target version:		
Difficulty:	uncategorized	

Description

In 4.6, if you build with (say) AVX and run on a non-AVX machine machine, all GROMACS tools exit with "illegal instruction (core dumped)". That's unavoidable if we tell the compiler it can optimize for AVX.

Is there a way to use our CPU detection (from code compiled without hardware optimization flags) to warn the user that the tool might be about to crash, because their execution machine (e.g. cluster front end) does not match the configure target (e.g. cluster back end)?

Related issues:

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

History

#1 - 01/22/2013 11:19 AM - Berk Hess

We have been discussing this issue.
The problem is that the compiler can insert SIMD instructions anywhere. We might have a reasonably chance of succeeding if this check is the very first thing we call in any program, but there is no guarantee. Also we might need to compile the check routine itself with -O0, no SIMD, etc.

#2 - 01/22/2013 08:14 PM - Teemu Murtola

I don't know exactly what the -mavx flag changes in the produced binaries (except for allowing to generate AVX instructions and changing the instruction encoding scheme). But if it doesn't change the calling convention for C functions (and it probably doesn't), it should be perfectly possible to compile some parts of the binary without -mavx. I don't think that compilers will generate any AVX instructions or such if not given explicit permission to do so. With C++, there can be some added complexity from stuff that comes from headers (there are several cases where the C++ linker is required to discard duplicate symbols (most notably with templates), and it can cause problems if these duplicate symbols come from different files with different instruction sets enabled), but don't know whether this is a real problem.

So in principle, we could compile main() and the check routine (and any other general-purpose code needed) without -mavx, and it should work, assuming that the compiler would not generate any AVX instructions in the code that we can't influence and that runs before main(). But this is very easy to break, since, e.g., some initialization code that is called before main() (like global variables of class types in C++) that reside in an AVX-enabled source file could still cause the error.

To do the above, it requires some additional complexity in the build system, and changing every main() function. As there are quite a few in 4.6, it is a bit of work, although mostly straightforward. In master, it could be simpler (in particular if #685 is implemented fully, to have only a single binary).

Just my two cents...

#3 - 01/22/2013 08:23 PM - Erik Lindahl

Right now this checking usually goes on fairly deep in the call sequence with routines the function and all three libraries involved. We thought about this earlier, but I still think that change is beyond what we want to change for 4.6 - and then we still have problems with things like FMA support or FMA4 that will still break compatibility. I think a reasonable compromise for 4.6 is that we explicitly warn the user if we detect that they are running on hardware that doesn't support the compiled acceleration, but also accept that the execution might not reach that far in some cases.

For 5.0 I would prefer to start and think if there is any way we can reintroduce the ability to have multiple accelerated codepaths at least for the kernels - if we anyway introduce several file-specific flags just for the warnings the real solution should not be a whole lot harder.

#4 - 01/24/2013 02:46 AM - Szilárd Páll

I've suggested earlier what I think is the solution, but never had the time to try it out. We could compile the current CPU detection code with lower acceleration (AFAIK for cpuid anyway *no optimization* is recommended) and call it first in the very beginning of mdrun, before anything else is called. The reason why we need to call it *as early as possible* is that the illegal instruction occurs already during the first couple of prints to stderr. We could either call it again later or would have to propage the detection results down in the call tree.

So I believe it is perfectly doable and would requires quite little code - especially if we re-detect later or provide a specific functionality to the cpuid module which only detects and checks the max supported acceleration against the one mdrun is compiled with.

#5 - 04/29/2013 07:54 PM - Mark Abraham

- Target version deleted (4.6.1)

#6 - 06/26/2013 01:40 AM - Mark Abraham

- Target version set to 4.6.x

Szilard's suggestion seems worth investigating for 4.6, but I agree with Erik we should consider catering for more portable binaries in 5.0

#7 - 06/19/2014 12:05 PM - Rossen Apostolov

so remove 4.6.x target?

#8 - 06/19/2014 12:06 PM - Erik Lindahl

- Target version changed from 4.6.x to future

#9 - 07/11/2016 08:52 PM - Mark Abraham

- Target version deleted (future)

#10 - 01/07/2019 02:05 AM - Mark Abraham

- Status changed from New to Rejected

Not going to be fixed, unless via #1165