GROMACS - Task #2294

Require identical hardware on nodes on parallel runs

11/13/2017 04:59 PM - Erik Lindahl

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

Description

Some Gromacs routines currently try to cater to nodes with different hardware.

However, as hardware has gotten more complex our simple tests are not sufficient. There is e.g. no longer any simple one-dimensional relation between SIMD instruction sets, and the SIMD instruction set we suggest to recompile Gromacs with might depend on the individual mix of nodes in each run. If the user recompiles, they might get another error/warning next time when running on a subset of the same nodes. The present code also causes us to confuse users since we do not separate the cases whether there was some local hardware feature or different node in the run that limited the SIMD set we could use.

Several parts of the code and detection would be less complex (and we avoid future work) if we can simply require all nodes in an MPI-parallel run to be identical. This will be true for any normal supercomputer and a very large majority of clusters with infiniband. The only exception is likely lower-end department/group clusters, but without good interconnects there is little or no reason for running large MPI-parallel single jobs on them.

Basically: If anybody wants to keep this feature after Gromacs 2018, it's time to volunteer and redo all the hardware checks and assignments in C++ in a robust way that hides the complexity :-)

History

#1 - 11/13/2017 10:34 PM - Berk Hess

I don't see a good reason for not allowing users to run on such a setup. I would suggest to print a warning that the user is running on a heterogeneous setup and this might negatively affect performance and note that we will not print any further performance related notes.

#2 - 11/14/2017 01:53 AM - Erik Lindahl

The main reason is of course that it adds complexity to the code, which costs time & effort.

Right now one problem is that we need to think about SIMD support on all nodes, and when we then start reporting problems we need to account for the fact that the "wanted" level might be based on a completely different node - but we can only to the AVX512 tests on the local nodes. That in turn means we must always test whether the local node only has a single FMA unit so we know that's why we want to use AVX2.

It also means our error messages are fuzzier because there can be many reasons for needing a lower level of SIMD.

Not printing things doesn't really solve the complexity: That still means everybody needs to consider whether we run on heterogeneous hardware and propagate that information to the routines writing or checking stuff.

I think the easier solution is to simply tell users in some documentation that it might be possible to run on heterogeneous hardware if they manually set things to the lowest common SIMD level, but that Gromacs itself will assume heterogeneous hardware and you might get some nonsensical warning messages, and that it's not supported. Then we save time & complexity because we don't have to care about it at all in the code, but it might still work.

#3 - 11/14/2017 01:18 PM - Berk Hess

But different hardware is an issue anyhow, independent of if we want to support it or not. Starting MPI ranks on multiple nodes starts code execution and illegal instructions can happen before we can even check or print a message. Furthermore, that is fundamentally not different from running a binary compiled for a different architecture on a single node.
As I said, I suggest writing a single message that a heterogeneous setup might negatively affect performance and then run without any more performance checks and messages.

The situation might get different if we support multiple targets in the same binary, but currently we don't.

#4 - 11/14/2017 02:41 PM - Erik Lindahl

We might actually have more similar opinions than we think. If we compare it to running a binary compiled for a different architecture, we simply don't care about it. We log the setup early on to help with debugging, and we try to issue a message if the SIMD is newer than the hardware, but then we just say "things might crash", don't care more about it, and leave it up to the user.

I think that would be a good solution for heterogeneous MPI runs too.

The problem with not doing any more performance checks or messages is that it pushes work onto programmers if all routines need to account for the fact that we might be running on heterogeneous hardware and not testing/warning about things.

So, my suggestion for release 2019 would be that we have one early message about detecting heterogeneity, warn that it might or might not work, and then say all the code does not have to take it into account.

#5 - 11/17/2017 08:10 AM - Mark Abraham

I'm all for simplicity. Is anyone aware of any other widely used MPI code that makes any attempt to run across heterogeneous resources? There's no need for us to push the norms in that direction, as well as all the other good things we try to do.

#6 - 11/17/2017 06:20 PM - Szilárd Páll

I'm not sure I see a great gain to be had by not allowing heterogeneity and yes, I know that many people do use heterogeneous nodes to launch multisim jobs (if nothing else). We have been encouraging the use of multisim for such purposes for years because there is no better job placement manager than mdrun itself. This works like a charm even over the crappiest Ethernet across a mix of desktop machines, even a mix of GPU and non-GPU hosts.

I don't think it's the right timing to debate it and it feels like the AVX2 vs AVX512 concern over a small performance benefit (it's really few %) seems to be far more than what the the coding, review, and discussion required warrants -- especially as an argument in the context of people running on mixed, often old and new hardware, etc. I'd be happy to approve a simple change that simply tells the user that they're running across different nodes (no details) or even code that simply omit such details and document it instead.
For this realease (and perhaps not only) there are use cases with significantly bigger impact and more performance benefit, so I'd suggest we to leave the AVX512 detection, tuning, and UI/reporting optimization headaches for later.

As Berk said, we're not even trying to make sure that the wrong binary does not exit with illegal instruction on hardware that does not support the instruction set. We can simply document (or possibly warn unless the reporting code gets removed) that in a heterogenous setup users have to use the lowest common SIMD support (gut or remove all the aggregate reporting code if somebody is so bothered, though I think it's nice), but leave the user to run on whatever mix of hardware they wish to run on.

> Is anyone aware of any other widely used MPI code that makes any attempt to run across heterogeneous resources? There's no need for us to push the norms in that direction, as well as all the other good things we try to do.

I'm confused where's the push? Wanting to handle an env var or the aggregate reporting? Compared to all the l'art pour l'art tuning and code beautification effort we all do, checking whether the content of and env. var. matches another string is really minimal overhead. Regarding the heterogeneity reporting, as I said before, if somebody is bothered by the aggregate/mismatch reporting code, remove it (though it might bite back as clusters are increasingly heterogenous and all it takes is a wrong SLURM argument to get a mixed set of nodes), but otherwise I don't think there is any other support that's is required, is there?

#7 - 11/17/2017 06:28 PM - Erik Lindahl

I was not intending this for release-2018, but I wanted to create an entry for the future while I remember.

Multisim was mainly implemented to enable runs where the queue system either cannot handle enough jobs, or because centers do not allow small jobs. I can't seriously imagine ANY center in the world that forces users to submit jobs over different types of nodes. It is also not true that this will be a better job placement, since multiruns will have to be limited to an instruction set supported by all nodes, while separate simulations could use different binaries.

However, it sounds like we have an agreement that we can just warn about it, and then ignore it in all other routines. Beyond that, anybody is welcome to work on it if they feel it's important.

#8 - 11/17/2017 11:47 PM - Szilárd Páll

Erik Lindahl wrote:

> I was not intending this for release-2018, but I wanted to create an entry for the future while I remember.

OK.

> Multisim was mainly implemented to enable runs where the queue system either cannot handle enough jobs, or because centers do not allow small jobs.

It was, but for the last few years the use-cases have greatly extended. As there is no better job placement manager (that does the affinity and GPU assignment just right -- not even copernicus does any of that), we have explicitly encouraged users to use it to manage their throughput/multi jobs to avoid pinoffset and GPU ID headaches. Additionally, he have also advertised mdrun-multi as the simple way to maximize throughput and get extra extra performance when placing multiple ranks on the same GPU (e.g. "Best bang for your buck" paper).

> I can't seriously imagine ANY center in the world that forces users to submit jobs over different types of nodes.

I did not claim "force", not even "suggest" or "recommend", did I?

My point was that there are an increasing number of clusters which have multiple generation and GPU/non-GPU nodes (e.g. Tegner, CSC Finland, Archer and Isambard in UK just to name a few "proper" clusters I know of and than we have not even talked about homebrew clusters which is the new raging trend given the migration of bio-MD away from HPC). All it takes is a wrong -gres or queue argument and you can get allocated an Ivy Bridge and a Haswell node or one with 2 GPUs and another with four. If your binary does not crash (e.g. because you wanted to submit to the Ivy Bridge but also got Haswell or the wanted 2-GPU nodes, but also got some with 3-4), users clearly do benefit from a reporting capable to let them know that they have mixed hardware.

> It is also not true that this will be a better job placement, since multiruns will have to be limited to an instruction set supported by all nodes, while separate simulations could use different binaries.

Most users won't care and we should let them not to have to care.

Also, if you're using a GPU, we discussed not long ago that I did measure the performance of using SSE4 binary on IIRC AVX hardware and the difference was surprisingly small -- especially compared to other aspects (like pinning vs not pinning). I expect that this continues especially with AVX vs AVX512 as you gain clock speed (perhaps even SSE4 on AVX512 may not be that bad).
For that reason, starting the 21 lambda points sims part of an FE calculation at once (which will anyway run with different performance so using AVX2 binary on AV512 will often matter less) is an option with real benefits and AFAIK some people do it.

#9 - 11/18/2017 01:53 AM - Erik Lindahl

As long as somebody volunteers to maintain it, I'm fine having it. However, arguing that it's an important feature and others should do it carries very little weight :-)

#10 - 11/19/2017 08:54 PM - Mark Abraham

Szilárd Páll wrote:

> > Is anyone aware of any other widely used MPI code that makes any attempt to run across heterogeneous resources? There's no need for us to push the norms in that direction, as well as all the other good things we try to do.
>
> I'm confused where's the push? Wanting to handle an env var or the aggregate reporting?

Wanting to prioritise on handling the major use cases with clear, simple code.

> Compared to all the l'art pour l'art tuning and code beautification effort we all do,

I prefer self-identify as a software engineer.

> checking whether the content of and env. var. matches another string is really minimal overhead. Regarding the heterogeneity reporting, as I said before, if somebody is bothered by the aggregate/mismatch reporting code, remove it (though it might bite back as clusters are increasingly heterogenous and all it takes is a wrong SLURM argument to get a mixed set of nodes), but otherwise I don't think there is any other support that's is required, is there?

For example, I'm not volunteering to write task placement code that can do a good job with a node with 14 Intel cores and 4 GPUs and another with 16 AMD cores and 2 GPUs. And until we can do a decent job of placement and auto-tuning in the more important homogeneous case, then considering the former is out of scope.

We are not obliged to write software that can maximize the outcome when the user makes an arbitrary error in their job-submission script on a cluster of arbitrary composition. That's on them. If we can run, we do so. If it's easy to warn them that this is a bad idea, that's fine. Otherwise, we're all agreed that we will make no attempt to handle such cases gracefully. If we notice something inconsistent, we might warn or error depending on the nature of the issue.