GROMACS - Feature #1665

Feature # 742 (New): Enhancing the performance of the free energy code

improve free energy non-bonded kernel performance

12/19/2014 11:41 PM - Szilárd Páll

Status:	New
Priority:	Normal
Assignee:	
Category:	core library
Target version:	future
Difficulty:	uncategorized

Description

The performance of the free energy kernels with the Verlet scheme is particularly poor with gcc, the icc performance can can be up to 1.5x better. As the difference does not vary much with the number of threads used, this is likely not caused by the atomic efficiency.

The performance numbers below show ms/step for the non-bonded free energy evaluation (regular non-bondeds offloaded to GPU) on different number of threads (both OpenMP and tMPI) using gcc 4.8, 4.9 and icc 15. The processor was an SB i7 3930K, but the same has been observed on Haswell too.

```
     icc 15    gcc 4.9    gcc4.8
 1x3    1.55     2.33    2.41
 1x6    0.83     1.22    1.26
1x12    0.65     0.91    0.94 (HT)

 3x1    1.39     2.20    2.18
 6x1    0.71     1.09    1.11
12x1    0.48     0.75    0.73 (HT)
```

Related issues:

Related to GROMACS - Bug #2014: GROMACS free energy memory footprint	**Closed**
Related to GROMACS - Feature #1666: new approach for Verlet-scheme kernel gen...	**New**

History

#1 - 12/19/2014 11:41 PM - Szilárd Páll

- File topol.tpr added

Added reproducer tpr.

#2 - 12/21/2014 12:36 PM - Mark Abraham

- Parent task set to #1500

The kernels themselves are those from the group-scheme implementation, so it is not suprising that one compiler might do better than another. It might be illuminating to find out why, though. We should probably upgrade the implementation to proper Verlet scheme kernels once we have resolved how we're going to do kernel generation in the future.

#3 - 12/23/2014 07:05 PM - Szilárd Páll

Mark Abraham wrote:

> The kernels themselves are those from the group-scheme implementation, so it is not suprising that one compiler might do better than another.

I know that the code in question uses the old 1x1 lists, but the performance discrepancy is unlikely to be related to this. I think it's related more to the implementation which does not seem to be particularly performance-oriented. Such a large difference I would attribute to either missed or mis-vectorization by gcc.

In any case, given that this discrepancy combined with the fact that we recommend gcc can lead to quite some performance loss, it could be worth investigating.

#4 - 12/24/2014 10:01 PM - Mark Abraham

Szilárd Páll wrote:

> Mark Abraham wrote:
>
> > The kernels themselves are those from the group-scheme implementation, so it is not suprising that one compiler might do better than another.
>
> I know that the code in question uses the old 1x1 lists, but the performance discrepancy is unlikely to be related to this. I think it's related more to the implementation which does not seem to be particularly performance-oriented. Such a large difference I would attribute to either missed or mis- vectorization by gcc.

I'm not sure what distinction you are drawing between the use of 1x1 lists and "the implementation." The use of the same branch-filled FEP kernel used in the group scheme is oriented to convenience and not performance, given that these kernels tend to be a small fraction of the run time. It is not suprising that one compiler would do differently from another. I'd be suprised if there was any vectorization going on at all. Different default implementations of switch statements seem to me more likely to lead to the observed performance differences.

> In any case, given that this discrepancy combined with the fact that we recommend gcc can lead to quite some performance loss, it could be worth investigating.

I think solving the underlying problem of kernel generation, and then applying those is more profitable. We wrote pre-vectorized non-FEP kernels for a reason ;-)

#5 - 12/25/2014 04:48 PM - Szilárd Páll

Sorry, I think I edited this post by Szilard instead of replying to it. Comment 6 is now what I said, but I don't have the original text that went here any more. Hopefully someone has it backed up on email somewhere. Sorry again.

#6 - 12/26/2014 10:24 PM - Mark Abraham

Actually, it has since occurred to me that the branchy and non-SIMD nature of this kernel makes it a perfect test example for templating over run-time constants. It's non-trivial code and simply cannot run any slower than it does! We can do this independently of any efforts on the SIMD kernels.

There are probably all the normal short-range kernel specializations, plus some new ones (soft-core alpha, off the top of my head). We probably don't want to add compiling 50-100 kernels just for FEP, but specializations for RF+cutoff, PME+cutoff, PME+PME cover most of the relevant cases. To handle those, we could have grompp- and mdrun-time warnings about enabling a conditional-compilation feature to get all the specializations compiled... but since we probably have zero test cases for most of that space, we could just tell them to ping gmx-developers to discuss.

#7 - 12/28/2014 03:58 PM - Szilárd Páll

I've done some tests with disabling vectorization and also dropping optimization level to O2/O1/O0, but strangely, except O0 the other options die not change much the performance of the free energy non-bonded code; disabling vectorization (-fno-tree-vectrorize and -no-vec -no simd for gcc/icc, resp.) does improve performance a bit (~3%) with gcc 4.9 and 7-8% with icc 15.

In any case, you are right, these kernels could be the perfect testing ground for templated kernels - although with the disadvantage that the performance aspect can not be assessed.

#8 - 12/28/2014 04:12 PM - Szilárd Páll

Szilárd Páll wrote:

> I've done some tests with disabling vectorization and also dropping optimization level to O2/O1/O0, but strangely, except O0 the other options die not change much the performance of the free energy non-bonded code; disabling vectorization (-fno-tree-vectrorize and -no-vec -no simd for gcc/icc, resp.) does improve performance a bit (~3%) with gcc 4.9 and 7-8% with icc 15.

BTW, I just realized that the lovely hidden compiler flags could have interfered with these experiments, although not very likely as the ones used with gcc only control frame pointer omission, unrolling, and precision.

#9 - 06/18/2015 06:59 PM - Erik Lindahl

- Tracker changed from Bug to Feature

- Target version set to future

Changing to feature, not because it's not important, but because it's not something incorrect we have introduced along the way. Previously we have never cared about the performance of the free energy kernels (that is the whole reason it was never SIMD), and we have simply called standard libraries even for expensive functions like exp() and pow().

Thus, I don't see any obvious way where we could just "fix" the performance, but it's possible due to better math libraries with commercial compilers. I think we'll have to revisit the entire free energy kernels after we have done the new version of the verlet kernels, and then consider whether we should

formulate it as a SIMD verlet kernel.

... or Michael might have some progress on their new form that won't require separate free-energy versions of the nonbonded evaluation by then.

#10 - 06/23/2016 04:01 PM - Mark Abraham

- Related to Feature #742: Enhancing the performance of the free energy code added

#11 - 07/11/2016 10:43 PM - Mark Abraham

- Target version changed from future to 2018

#12 - 10/19/2016 03:32 PM - Szilárd Páll

- Related to Bug #2014: GROMACS free energy memory footprint added

#13 - 04/23/2017 11:59 AM - Mark Abraham

- Target version changed from 2018 to 2019

#14 - 04/23/2017 12:07 PM - Mark Abraham

- Related to Feature #1666: new approach for Verlet-scheme kernel generation added

#15 - 09/19/2018 03:07 PM - Mark Abraham

- Target version changed from 2019 to future

#16 - 08/26/2019 03:47 PM - Mark Abraham

- Parent task changed from #1500 to #742

Files

topol.tpr	831 KB	12/19/2014	Szilárd Páll