GROMACS - Feature #1181
Implementing asymmetric offload to MIC (Xeon Phi Co-processor)
03/07/2013 04:37 PM - Mikhail Plotnikov

**Status:** Closed
**Priority:** Normal
**Assignee:** Roland Schulz
**Category:** mdrun
**Target version:**
**Difficulty:** uncategorized

**Description**
There is highly optimized GPU offload scheme implemented in GROMACS with auto-detection of GPU and dynamic balancing between CPU<->GPU. It offloads PP computations to GPU and computes PME part on CPU. The same asymmetric parallelization approach can be implemented for MIC co-processor and is highly appreciated as MIC has been already launched and is accessible for users.

**Related issues:**
- Related to GROMACS - Feature #1187: Optimized Verlet SIMD kernels for native ...
  - Closed
- Related to GROMACS - Feature #1394: 16-wide MIC SIMD bonded interaction kernels
  - Closed
- Related to GROMACS - Feature #1420: Fast force reduction for large number of ...
  - Closed

**Associated revisions**
Revision d28336a2 - 01/09/2014 10:12 AM - Roland Schulz
Add SIMD support for Intel MIC

Only single precision is supported so far.

To compile in native mode:
CFLAGS="-mmic" CXXFLAGS="-mmic" cmake

Regression tests pass with ICC 14.

Instructions are not updated because only offload mode (#1181) is useful to the user.

Part of #1187

Change-Id: i81a2022cfecf6341dfaff5ce63ad820a5d4dee

**History**

#1 - 03/08/2013 02:51 PM - Szilárd Páll
- Category set to mdrun

To clarify things a bit, what we'd need for this is:

- either the full task-parallelization framework;
- or separate "PP nodes" similar to the PME ones that do nothing but calculate non-bonded (and perhaps bonded) forces and possibly do reduction on them.

The latter is probably considerably easier to implement. However, with the latter we might run into issues related to the (current) lack of task splitting and balancing meaning that e.g. the non-bonded workload can not be flexibly split between CPU and GPU and therefore, especially if the accelerator is relatively slow (compared to the CPU), the CPU can end up idling a lot (especially with non-PME runs).

#2 - 03/10/2013 08:03 AM - Roland Schulz
Why is either of this required? Can't we simply do an asynchronous offload of the kernel? This should be easier than adding PP-nodes and have the same effect. And wouldn't some very simple task splitting for the force kernel very simple? Also we would need to have kernels with the MIC instruction set to get good performance (either MIC intrinsics or ispc).

#3 - 03/11/2013 11:52 AM - Szilárd Páll
@Roland, Mikhail is working on the Verlet kernels for MIC, so that part of the issue is being addressed. Ideally, we should get LRNbi kernels for bondeds as well (@Mikhail: added you to the reviewers of patch set 2206 relevant in this context).
The reason why I think PP nodes would be a far better solution is that offloading is pragma-driven, hence rather inflexible and ugly as well.

Workload splitting is not hard, but splitting without any load balancing is not very useful, I think and the load-balancing part can be a bit more tricky.

All in all, I think the tasks are not difficult, but they require deep familiarity with the code and right now it is not sure who will have time to collaborate with Mikhail on these features in the near future.

#4 - 03/11/2013 12:58 PM - Mikhail Plotnikov
Roland Schulz wrote:

Why is either of this required? Can't we simply do an asynchronous offload of the kernel? This should be easier than adding PP-nodes and have the same effect.

Do you mean pragma offload in OpenMP loop? I don't think it will be efficient to make offload at this level due to overhead. Task decomposition is required with balancing between PP<->PME tasks.

#5 - 03/11/2013 03:13 PM - Szilárd Páll
Mikhail Plotnikov wrote:

Roland Schulz wrote:

Why is either of this required? Can't we simply do an asynchronous offload of the kernel? This should be easier than adding PP-nodes and have the same effect.

Do you mean pragma offload in OpenMP loop? I don't think it will be efficient to make offload at this level due to overhead. Task decomposition is required with balancing between PP<->PME tasks.

That's what I thought, but I am not very familiar with the offloading mechanisms on MIC. Are there any ways to implement efficient asynchronous execution similar to what one would do with GPUs? How about _Cilk_offload? Does that only work with the implicit memory copy model? I assume the implicit memory copy model is not very efficient, is it?

In its most simple form offloading for the non-bonded calculations would involve executing the following operations asynchronously:

- (transfer pair-list every pair search step)
- transfer coordinates to accelerator
- launch non-bonded kernel
- transfer forces from accelerator
- wait for forces to arrive

If the above operations can be launched asynchronously and implementing this is simple enough, even if it is not optimal, it could be advantageous to try it.

#6 - 03/11/2013 10:02 PM - Mikhail Plotnikov
Szilárd Páll wrote:

That's what I thought, but I am not very familiar with the offloading mechanisms on MIC. Are there any ways to implement efficient asynchronous execution similar to what one would do with GPUs?

Pragma offload should work fine. I just meant that offload should be done at the highest possible level, not at the main computational loop. There is a possibility to emulate GPU offload with environment variable documented in manual.

GMX_EMULATE_GPU: emulate GPU runs by using algorithmically equivalent CPU reference code instead of GPU-accelerated functions.

This one should enable all the mechanism that used for GPU offload. I think it worth to start with offloading such reference code. Do you agree?

#7 - 03/14/2013 12:29 PM - Berk Hess
The GPU emulation code does exactly the opposite. It produces a pair list optimized for the GPU, but does not do the off-loading.

Off-loading only involves the 5 calls Szilard mentioned above, which are all called in src/mdlib/sim_util.c: nbnxn_cuda_... I assume the off-loading should not be slower than MPI communication with MIC. Implementing off-loading will be less work than implementing separate non-bonded only nodes similar to the PME-only nodes. The latter could be somewhat more general, but it goes against our planned task parallelization scheme, where we would like to have one MPI rank per physical node.

#8 - 04/03/2013 03:01 AM - Szilárd Páll
Mikhail Plotnikov wrote:

This one should enable all the mechanism that used for GPU offload. I think it worth to start with offloading such reference code. Do you agree?
All GPU offload code is (NVIDIA) GPU/CUDA specific and consists of:

- GPU device and data management/initilization functionalities;
- the actual offload of the non-bonded computation including asynchronously executed explicit data copies to/from the GPU and non-bonded kernel.

The asynchronous CPU-GPU execution is achieved by enqueueing the aforementioned tasks in the GPU's work queue (through the CUDA runtime API). The GPU "emulation" mode doesn't do anything fancy, it simply omits call to any CUDA API functions (including kernel invocation) and executes a plain C kernel algorithmically equivalent to the CUDA GPU kernels. Hence, the GPU acceleration does not require separate PP ranks as the GPU tasks are handled by the GPU's own task scheduler and therefore there isn't much to (re)use for the MIC offload. Additionally, it really does not matter which plain C reference kernels are used, the GPU 8x8x8 or "normal CPU" 4x4 ones.

If it is reasonable to implement the CPU-MIC data/control flow in a similar fashion as the current GPU acceleration, the starting point could be implementing the MIC offload module by mirroring the GPU offload functionality, see:

- module API: src/mdlib/nbnxn_cuda/nbnxn_cuda.h
- module source: src/mdlib/nbnxn_cuda/nbnxn_cuda.cu

I assume none of the complicated explicit data- and device-management functionality that GPUs require is needed for MIC, right?

#9 - 10/08/2013 12:40 AM - Roland Schulz
- Target version changed from future to 5.0

#10 - 10/12/2013 01:55 AM - Szilárd Páll
Have you started working on the pragma-based offload?

This summer I've looked at the code and it seemed that the current coupling between the number of non-bonded tasks and force buffers needs to be broken and the buffers and reduction should probably be moved "closer" to the force kernel (perhaps thread-local storage + reduction immediately after the kernel and before transferring back a single array).

Additionally, it could be worth considering:

- a binary reduction could be more efficient on MIC;
- overlapping x/f transfers by splitting up the coordinates (we're working on this for GPUs);
- allowing the non-local kernels to start before all local work is done, but this may not be possible without asynchronous MPI.

#11 - 10/14/2013 08:11 PM - Roland Schulz
No we haven't started on this. We (mostly John Eblen in our group) have started with #1187. Both Mikhail and Chris Neale said they might be interested with doing this. If no one has time John will do it after #1187 is done.

With buffer you mean memory buffer (before reduce) not verlet buffer (rlist-rcut), correct? Thanks for the comments. I think whoever starts this work should have a chat with you before.

#12 - 10/17/2013 06:43 PM - Chris Neale
I am currently looking into this to see if I am capable of doing it. I will let you know if I make progress or decide it's beyond my abilities. I am available for discussion if anybody has suggestions.

#13 - 10/18/2013 07:55 PM - Szilárd Páll
Roland Schulz wrote:

> With buffer you mean memory buffer (before reduce) not verlet buffer (rlist-rcut), correct? Thanks for the comments. I think whoever starts this work should have a chat with you before.

I was referring to the force accumulation buffers which are currently allocated early in the pair-list setup phase. As we will (most likely) use asymmetric offload, we will need a different number of buffers than than the number of non-bonded pair lists (= number of cores the pair search uses). Even if these two numbers would match we'd anyway want to reduce on the MIC and transfer back a single force array (or two with DD) instead of sending back k*61 accumulation arrays for reduction on the CPU.

#14 - 10/18/2013 08:22 PM - Szilárd Páll
It is probably a good idea to discuss things before taking concrete action; please contact me (and/or Berk) for scheduling a meeting.

#15 - 01/13/2014 08:10 PM - Roland Schulz
- Status changed from New to In Progress
- Assignee set to Roland Schulz
#16 - 01/13/2014 09:12 PM - Roland Schulz
- Related to Feature #1420: Fast force reduction for large number of threads added

#17 - 05/13/2014 03:50 PM - Roland Schulz
- Target version changed from 5.0 to 5.x

#18 - 10/14/2015 06:08 PM - Mark Abraham
- Status changed from In Progress to Closed

I think this is never going to happen.

#19 - 10/14/2015 06:15 PM - Roland Schulz
- Status changed from Closed to In Progress

We are working on it. And if it gives a speedup it has a chance to get in right?

#20 - 10/16/2015 08:33 AM - Mark Abraham
Maybe, but if KNL won't work in offload mode it doesn't seem like a good focus for effort

#21 - 06/01/2016 02:11 PM - Mark Abraham
- Target version deleted (5.x)

Are we still working on this?

#22 - 03/06/2018 11:33 PM - Mark Abraham
- Status changed from In Progress to Closed

Apparently not.

#23 - 03/07/2018 12:03 AM - Roland Schulz
Correct. If anyone is interested https://github.com/rolandschulz/gromacs/commits/offload-knl contains the latest working version.

#24 - 03/09/2018 05:48 PM - Mark Abraham
Roland Schulz wrote:

Correct. If anyone is interested https://github.com/rolandschulz/gromacs/commits/offload-knl contains the latest working version.

great