GROMACS - Task #2983

Task # 2675 (In Progress): bonded CUDA offload task

better suited data-types for bonded GPU kernels

06/18/2019 08:49 PM - Szilárd Páll

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

Description

The data types as used now, due to the simplistic port from CPU, are ill-suited for efficient access on GPUs.
E.g. the type and atomic indices can not be loaded in a vectorized manner and as a result a huge amount of L2 transactions become a major bottleneck to performance. We should develop better data structures and employ mechanisms to mitigate the overhead (like vectorized prefetching of data).

Associated revisions

Revision 5854b8da - 07/03/2019 04:02 PM - Magnus Lundborg

Vectorize GPU bonded data.

Loads data for bonds, angles, UB angles and pairs in blocks
of 3, 4, 4 and 3 ints respectively.

Refs: #2983

Change-Id: I37605395bd0c375b2eb4b94e76f9b30f6cb15e8d

History

#1 - 06/18/2019 09:33 PM - Magnus Lundborg

I'll be happy to try to help with this as well, but I must admit I don't know how to prefetch the data in a suitable way. I guess this should be done when the data is copied to the GPU already, right? Do we do this anywhere (for GPU calculations) already?

#2 - 06/19/2019 02:15 PM - Szilárd Páll

We are discussing alternatives; the way to do prefetching is to declare a __shared__ memory region that is per-block i.e. 256 threads (btw we should reconsider that ad-hoc chosen number anyway); then load into this memory with vectorized sequential access then read forceatoms from this shared memory buffer. This is a bit tricky because we might have to make the granularity of bonded type mapping from warpSize to blockSize because shared memory is per bock. As an example, we do prefetching e.g. here: source:src/gromacs/nbnxm/cuda/nbnxm_cuda_kernel.cuh#L389

It seems it might be a better approach (though slightly more work) to instead tweak the data structures to directly store parameters instead of looking them up via an index. Then we might still want to do a prefetch to make variable length loads more efficient, but we won't have the indirection we have now. This latter approach would allow possible slight simplification/improvement in the CPU bonded kernels too, but of course CPU kernels would have to change as well.

Thoughts?

#3 - 06/20/2019 12:53 PM - Magnus Lundborg

Since I'm far from an expert at GPU programming (I just try to learn as I go along) I'm afraid I can't say what alternative I think is best or how to implement them easily.

#4 - 07/03/2019 03:51 PM - Szilárd Páll

So having looked briefly at some profiles briefly, my impression is that there is a lot of overhead in both the type and the index loads as well as the parameter loads -- where t_iparam is the main culprit, it is simply not a good data structure for efficient data access (and that is IMO true for CPUs too).

On the short run, for the former I suggest:
- we try to prefetch -- this will prevent running at full occupnacy, but based on my tests even at 50%occupancy the non-virial kernel runs only a few % slower, so the gain from efficient access will outweigh that loss. Also note that while int3 loads do "vectorize", these are still *two* loads (and 8- and a 4-byte load).
- consider devising an improved version of t_iparam that allows efficient loads of parameters, i.e. vectorized loads from adjascent memory locations rather than non-vectorized loads from a "gappy" union representation.

#5 - 07/04/2019 08:14 AM - Magnus Lundborg

I made an attempt at preloading the data, but I just shifted the problem earlier. The problem was that I did not come up with a good way to read it sequentially and still store it in a convenient way.

#6 - 07/09/2019 03:36 PM - Magnus Lundborg

It seems to me like it would be best to split iatoms into std::vector<int> InteractionTypes and std::vector<int4> InteractionAtoms both of the same length as the number of interactions. But it would require quite large changes.

#7 - 07/11/2019 04:32 PM - Szilárd Páll

Magnus Lundborg wrote:

> It seems to me like it would be best to split iatoms into std::vector<int> InteractionTypes and std::vector<int4> InteractionAtoms both of the same length as the number of interactions. But it would require quite large changes.

I agree, that would be better for all interaction types that have 3 or 4 atoms involved. For pairs it would be quite wasteful, but it might be worth the tradeoff.

Side-note: I still feel it may be worth storing all coordinates instead of atom indices (although I'm not certain how much difference it will make). While this would waste bandwidth it would save instructions and latency: 4x float3 can be loaded with 3x 16-byte loads whereas we now do (up to) four 4-byte loads, then (up to) four dependent 16-byte loads. While the latter will often be cached, bandwidth is not the primary issue, but rather memory instruction throughput and instruction latency, both of which would be reduced with this solution, I think.