GROMACS - Feature #1994

explore using JIT for CUDA

06/21/2016 05:41 PM - Szilárd Páll

Status:	New	
Priority:	Low	
Assignee:		
Category:	mdrun	
Target version:	future	
Difficulty:	uncategorized	

Description

The CUDA Runtime Compilation library introduced in v7.0 provides JIT-ing of CUDA kernels. The use of this feature was expected to allow

1. potential performance benefits
2. reduced compilation time
3. flexibility through kernel customization at runtime.

Initial evaluation of 1. was done using gmx 5.0 by hardcoding kernel parameters. The results showed up to 4% performance decrease. The reasons were never explored in-depth, but NVIDIA has worked on the issue and based on their feedback, with 8.0 RC the kernels with hard-coded constants are now faster.

While point 2. has become less of an issue since as we have introduced cmake support for targeting a reduced set of CUDA arch in the build as well as split up the nbnxn_cuda module into multiple compilation units, there are still clear opportunity to do fast JIT compilation of only those few kernels that are actually needed as well as kernel on-line auto-tuning, etc. This relates to point 3. too which would both allow on-the-fly kernel (flavor) generation for auto-tuning as well as user customization of kernels (e.g. custom functional forms).

Proposed future tasks:
- evaluate CUDA 8.0-RC and the performance with fixed kernel constants in the new kernels (v2016 w/wo combination rules);
- evaluate the potential of this feature in the new PME kernels;
- when considering implementing the CUDA runtime compilation support, this functionally should mirror/be merged with the current OpenCL JIT support; experiments on the possible future use cases for JIT can (and should) first be done with OpenCL.

History

#1 - 07/11/2016 10:20 PM - Mark Abraham

Noted. I have been doing a bit of further cleanup on the OpenCL JIT off gerrit, with a view to eventually using some simple tasking across CPU cores to improve time-to-JIT, which you noted some time back could be a problem on future machines with weak CPUs (which might happen with any vendor's GPUs). However, I have identified that we should probably focus first on making sure some structs are defined in only one place (at least OpenCL duplicates a few things, a bunch of which looks unnecessary). Reducing the contents of and include dependencies of the CPU SIMD kernels would also help avoid compiling them more often than necessary during ongoing development of the NBNXN module. I have a preliminary patch in preparation that moves a bunch of code out of the catch-all nbnxn_internal.h and nbxnxn_pairlist.h includes, along with some other cleanup, but we'll want Berk's opinion on that.