GROMACS - Task #2524

struct alignment/packing for OpenCL host & device code

05/26/2018 03:16 PM - Szilárd Páll

Status:	New
Priority:	Normal
Assignee:	
Category:	mdrun
Target version:	2020
Difficulty:	uncategorized

Description

trying to figure out the requirements and best practices for host/device OpenCL struct alignment/padding; e.g. questions that have come up:
- Should we always specify an alignment for structs? How can we know what that should be?
- Do you always have to use cl_-prefixed types or can we just use e.g. int (or intNN_t) and assert on the size of the struct for safety?

Other requirements that we missed?

Related issues:

Related to GROMACS - Feature #2054: PME on GPU	**Accepted**
Related to GROMACS - Task #2453: PME OpenCL porting effort	**Resolved**

History

#1 - 05/26/2018 03:17 PM - Szilárd Páll

- Related to Feature #2054: PME on GPU added

#2 - 05/26/2018 03:17 PM - Szilárd Páll

- Related to Task #2453: PME OpenCL porting effort added

#3 - 05/28/2018 11:04 AM - Aleksei Iupinov

A recap:

Without being instructed what to do, an OpenCL compiler can do whatever it wants with alignment/padding.
If the same structure is being used as a kernel launch parameter but has different size on host and device - one gets CL_INVALID_ARG_SIZE from clSetKernelArg() right away.
However, even if size is the same, that doesn't guarantee same structure member packing on host/device, so the kernel launches, and you might get unpredictable bugs, entirely depending on the compiler.
The options are
- using attribute aligned(n) on structs, where n is number of bytes
- using attribute packed on structs
- sorting struct members and hoping it works
- adding dummy char arrays for explicit padding

With PME, I've settled on just resorting the structs (smallest to largest) until it worked everywhere (AMD/CUDA/Intel). This is not a guarantee, but I am not able to prove that alignment attribute always solves the problem by itself. It doesn't hurt either, but I can't prove it's necessary at all :-)
Packed attribute doesn't work with PME because I use a templated typedef DeviceBuffer<T> for cl_mem, and compiler probably can't resolve that to the underlying _cl_mem *. This probably can be resolved by specializing is_pod for DeviceBuffer, but also feels like a hacky direction. I'm also not sure whether tightly packing the parameter structs with no alignment has any consequences for kernel performance.
Dummy char arrays I haven't tried much.

Oh yes, and the cl_ types themselves read to me like modern int32_t/... - they only seem to guarantee the size on the host, not offset.

Sources:
https://www.khronos.org/registry/OpenCL/sdk/1.2/docs/man/xhtml/attributes-variables.html
https://www.khronos.org/registry/OpenCL/sdk/1.2/docs/man/xhtml/scalarDataTypes.html
http://www.catb.org/esr/structure-packing/

#4 - 05/29/2018 06:51 PM - Roland Schulz

Aleksei Iupinov wrote:

> With PME, I've settled on just resorting the structs (smallest to largest) until it worked everywhere (AMD/CUDA/Intel).

Did you have a case where it was sorted by size of members and it still didn't work? I.e. did it ever matter in which order members of same size where (e.g. ptr and int64)? Or cases where no padding was necessary (e.g. 2 int32 followed by int64) and it didn't work? In other words if we compile the struct with Wpadded and don't get a warning do we know of a case where it doesn't work (I know there is no guarantee - just asking whether there is a known case where this isn't sufficient)?

Either way compiling structs which are used for offloading with Wpadded is probably a good idea.

#5 - 10/30/2018 12:06 PM - Mark Abraham

- Target version changed from 2019 to 2020