GROMACS - Task #2053

refine notation in GPU code

09/20/2016 04:31 PM - Szilárd Páll

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

Description

It is often beneficial to know the memory space a pointer points to without having to inspect the declaration (or when a pointer is passed to a function, the original declaration). This applies both to kernel and host-device interface code. In the former, I propose allowing easy identification of:
- shared, global, and constant memory pointers in CUDA (and equivalents in OpenCL) using prefixes (e.g. s_, g_, c_), no prefix for register space;
- host and device pointers prefixed for clarity when the two are mixed in interface code

Related issues:

Related to GROMACS - Feature #2054: PME on GPU	**Accepted**
Related to GROMACS - Task #2048: C++11: CUDA dependency on general headers	**New**

Associated revisions

Revision 56067ac8 - 03/28/2019 08:52 AM - Artem Zhmurov

GPU naming conventions

In GPU programing, it is convenient to indicate what memory space
the pointer points to. This is often done by adding prefixes to the
pointers, which is now indicated in the developers manual.

Refs #2053.

Change-Id: Id39ad0b9c5876e4362fa4e261d0c011125dc380a

History

#1 - 09/20/2016 04:39 PM - Aleksei Iupinov

I like the idea; the only confusing thing would be conditional compilation. Currently I have in my code:

```
#if PME_GPU_PARALLEL_SPLINE
__shared__
#endif
float data[dataSize * order];
```

What would I do with that in general case?
Granted, this is already slightly confusing code.

#2 - 09/20/2016 07:14 PM - Szilárd Páll

Good point, I don't think there is a good way to solve that issue without introducing branching at the point where the pointer is used.

#3 - 09/26/2016 08:37 AM - Mark Abraham

Szilárd Páll wrote:

> It is often beneficial to know the memory space a pointer points to without having to inspect the declaration (or when a pointer is passed to a function, the original declaration). This applies both to kernel and host-device interface code. In the former, I propose allowing easy identification of:
> - shared, global, and constant memory pointers in CUDA (and equivalents in OpenCL) using prefixes (e.g. s_, g_, c_), no prefix for register space;
> - host and device pointers prefixed for clarity when the two are mixed in interface code

Sounds like a good approach. Note that we already have some conventions at http://jenkins.gromacs.org/job/Documentation_Nightly_master/javadoc/dev-manual/naming.html#id1 and these would clash somewhat with how we

already use s_ and g_. Even though the words are the same, the meaning differs and I think this has the potential for creating confusion. How about simply sharedData, globalData, constantData for an identifier that is currently called data?

Aleksei Iupinov wrote:

> I like the idea; the only confusing thing would be conditional compilation. Currently I have in my code:
>
> #if PME_GPU_PARALLEL_SPLINE
> __shared__
> #endif
> float data[dataSize * order];

Off-topic, but if you need to do this kind of thing a lot, see src/gromacs/utility/basedefinitions.h for examples of how you can define gmx_shared in some PME header, so that the places where you use it have less clutter. But mostly that's useful for things that have to change when configuration does (e.g. this compiler needs this keyword variation that another will reject).

> What would I do with that in general case?

I suggest using
@#if PME_GPU_PARALLEL_SPLINE
typedef *shared* float SharedFloat;
#else
typedef float SharedFloat;
#endif

//...

SharedFloat sharedData[dataSize * order];
@

If you're reading/writing code that is intended to work in both cases regardless of the type of an object, then its notional type must be the union of the constraints on the two types, so give that a name. You still have the option to specialize upon PME_GPU_PARALLEL_SPLINE if you need to.

#4 - 10/03/2016 05:08 PM - Aleksei Iupinov

> Sounds like a good approach. Note that we already have some conventions at http://jenkins.gromacs.org/job/Documentation_Nightly_master/javadoc/dev-manual/naming.html#id1 and these would clash somewhat with how we already use s_ and g_. Even though the words are the same, the meaning differs and I think this has the potential for creating confusion.

Alright, adding "m":
-- gm_, sm_, cm_ in the kernels
-- h_, d_ in the host GPU code

> How about simply sharedData, globalData, constantData for an identifier that is currently called data?

This also works:
-- global, shared, constant in the kernels
-- host, device in the host GPU code
Really have no preference as my code is stuck between the two.
I think the NB code uses h_ and d_ but not the in-kernel prefixes.
How about we vote :-)

#5 - 10/04/2016 03:13 PM - Szilárd Páll

> -- gm_, sm_, cm_ in the kernels
> -- h_, d_ in the host GPU code

I should have realized the clash with e.g. "c_" already being used for constants. Adding "m" sounds like a reasonable trade-off.

> How about simply sharedData, globalData, constantData for an identifier that is currently called data?

I'd prefer not calling it "data" as that's is not really correct. The prefix is meant to refer to a memory space, so it's not the data pointed to by the pointer that's global or constant, but the memory space where the data resides.

Secondly, adding more than a few character long prefixes will, in my opinion make things worse not better for the code readability. I find kernels hard to read when many statements end up 100-150 characters long, algorithms and data flow end becomes much harder to follow. Additionally, in such kernel, especially the inner loops, it's really not the casual reader's perspective that I'd cater for, and I think it's more important to have compact, readable, but of course reasonably self-documenting code.

I'm not sure where the balance should be, but I'd rather keep kernel code compact.

One aspect that we have yet to factor in is OpenCL compatibility of the naming scheme. Unfortunately, due to naming differences, we'd have to compromise one way or another. For now, the only clash is shared vs local memory.

#6 - 10/06/2016 05:35 PM - Aleksei Iupinov

As I'm not familiar with it, what are the OpenCL naming peculiarities?

#7 - 11/17/2016 03:43 PM - Szilárd Páll

Aleksei Iupinov wrote:

> As I'm not familiar with it, what are the OpenCL naming peculiarities?

The differences are summarized on slide 30 of this talk

Most notable example is that what's what's called "shared memory" in CUDA is called "local memory" in OpenCL. (Both names make sense, but the represent a different pov).

Not sure if it's easy to reconcile these, but we need to make a decision asap and adapt new code accordingly.

#8 - 11/29/2016 03:49 PM - Aleksei Iupinov

Oh well, these do seem incompatible. I would suggest sticking to the already suggested CUDA-style prefixes in all the code, even OpenCL (for easier code comparison/portability). No strong reasoning.

#9 - 03/20/2017 05:48 PM - Aleksei Iupinov

- Related to Feature #2054: PME on GPU added

#10 - 12/18/2017 11:12 AM - Aleksei Iupinov

- Target version changed from 2018 to future

#11 - 03/19/2018 11:26 AM - Mark Abraham

- Related to Task #2048: C++11: CUDA dependency on general headers added

#12 - 03/07/2019 07:44 PM - Szilárd Páll

- Target version changed from future to 2020

This policy has unfortunately been ignored during the bonded GPU change and it is not being followed by current developments either. We need to make sure everyone is aware of it.

#13 - 03/08/2019 11:18 AM - Mark Abraham

Szilárd Páll wrote:

> This policy has unfortunately been ignored during the bonded GPU change and it is not being followed by current developments either. We need to make sure everyone is aware of it.

Indeed. The "m" suffixes and CUDA-centric naming seem reasonable (preferably documented). Can you upload a change to the coding style, please?

#14 - 03/08/2019 03:03 PM - Szilárd Páll

Mark Abraham wrote:

> Szilárd Páll wrote:
>
> > This policy has unfortunately been ignored during the bonded GPU change and it is not being followed by current developments either. We need to make sure everyone is aware of it.
>
> Indeed. The "m" suffixes and CUDA-centric naming seem reasonable (preferably documented). Can you upload a change to the coding style, please?

yes, working on it!

#15 - 03/11/2019 11:41 AM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #2053.
Uploader: Artem Zhmurov (zhmurov@gmail.com)
Change-Id: gromacs~master~Id39ad0b9c5876e4362fa4e261d0c011125dc380a
Gerrit URL: https://gerrit.gromacs.org/9334

#16 - 04/30/2019 05:19 PM - Artem Zhmurov

Suggest naming for common indexing:

```
// Thread Index in Block
int tib       = static_cast<int>(threadIdx.x);
// Total number of threads in the block
int blockSize  = static_cast<int>(blockDim.x);
// Thread Index on Device
int tid       = static_cast<int>(blockIdx.x*blockDim.x + threadIdx.x);
```

In case of multidimensional grid, one can use tibi or tibx, whatever is appropriate for the case.

#17 - 07/02/2019 12:17 PM - Artem Zhmurov

The reallocateDeviceBuffer uses two numbers too keep track of allocated memory:
1. What is the number of the active elements in the array.
2. What size was allocated.
Suggest the following naming convention for these variables for d_<bufferName>_:
numBufferNameCurrent_
numBufferNameAlloc_

#18 - 07/03/2019 11:21 AM - Artem Zhmurov

Update for the naming convention for these variables for d_<bufferName>_:
numBufferName_
numBufferNameAlloc_

#19 - 09/04/2019 11:46 AM - Mark Abraham

Artem Zhmurov wrote:

> Update for the naming convention for these variables for d_<bufferName>_:
> numBufferName_
> numBufferNameAlloc_

Since we already need devs to be familiar with std::vector's naming of size vs capacity, that would be useful to adopt here also.

#20 - 09/04/2019 01:47 PM - Artem Zhmurov

Mark Abraham wrote:

> Artem Zhmurov wrote:
>
> > Update for the naming convention for these variables for d_<bufferName>_:
> > numBufferName_
> > numBufferNameAlloc_
>
> Since we already need devs to be familiar with std::vector's naming of size vs capacity, that would be useful to adopt here also.

Good point. Suggest:
h_bufferName_
d_bufferName_
bufferNameSize_
bufferNameCapacity_