

algebra-plugins rewrite

Attila Krasznahorkay

Previously on algebra-plugins...



- Some much-needed changes went into [algebra-plugins](#) recently
 - In an effort to harmonise how each project builds/finds its externals, I rewrote how algebra-plugins would do this
 - Beomki added explicit CUDA qualifiers on the functions defined in the project, to make them usable from device code

Repository Re-organisation, main branch (2021.10.15.) #27

Merged niermann999 merged 3 commits into `acts-project:main` from `krasznaa:project-reorg-main-20211015` 10 days ago

Conversation 12 Commits 3 Checks 49 Files changed 59

krasznaa commented 14 days ago

Unfortunately the code is not working correctly at the moment (more on that further down), but I thought I would still open a PR with all of these, just give time for people to familiarize themselves with what I've done.

What I started doing was to update the CMake configuration in `traccc`, to make sure that it could build SYCL code consistently with how `veccmem` builds SYCL code. (For the work that @konradkuslak97 will be doing.) But I had to realise that to do this, I would need to do some cleanup in every project. So I decided to start with this one...

Let's start with the main features of the "new" CMake configuration:

- I introduced `algebra_add_library(...)` and `algebra_add_test(...)` functions to harmonise how libraries and tests would be set up in the project.
- Re-wrote how "externals" would be set up by the project. Introducing a number of `CACHE` variables to be able to control precisely what the project would or would not build/find itself.
 - This included switching all external builds to `FetchContent`, and adding recipes for building `Vc` and `Eigen3` as part of the project as well.
 - `smatrix` however would always have to come from some existing place.
- Taught the project how to "package" itself. So that it could be "installed", and then downstream projects would be able to find it with `find_package`.
- Re-wrote the CI configuration to test the project on a big matrix of configurations.

I have to say, I really don't like how the source code is laid out in the repository at the moment. So this PR re-organises that quite a bit as well. All `(INTERFACE)` libraries are now put into "top level directories" in the project. I also renamed the `veccmem` and `vc` directories to `veccmem_array` and `vc_array`, to be in sync with the names of the CMake libraries. Though I guess the synchronisation could've been done the other way around as well, by renaming the CMake libraries.

The libraries are now set up as follows:

- `algebra::common` is always set up from the "common headers" that were previously pretty haphazardly included by the other libraries. It also defines `ALGEBRA_PLUGIN_CUSTOM_SCALARATYPE` itself, so that it would only be done in one place.
- `algebra::array` is set up on top of `algebra::common`, without any noteworthy settings.
- `algebra::veccmem_array` is set up on top of `algebra::common` and `veccmem`.
- `algebra::plugin_sscvecmem` to `traccc`.
- `algebra::sycl_array` is set up on top of `algebra::array`.
- `algebra::eigen` is set up on top of `algebra::array` and/or `algebra::sycl_array` are "built", it also processor macros configured on all those libr
- `algebra::smatrix` is set up on top of `algebra::veccmem_array` and/or `algebra::eigen`.

Add cuda qualifiers to algebras #28

Merged niermann999 merged 9 commits into `acts-project:main` from `beomki-yeo:cuda_build` 10 days ago

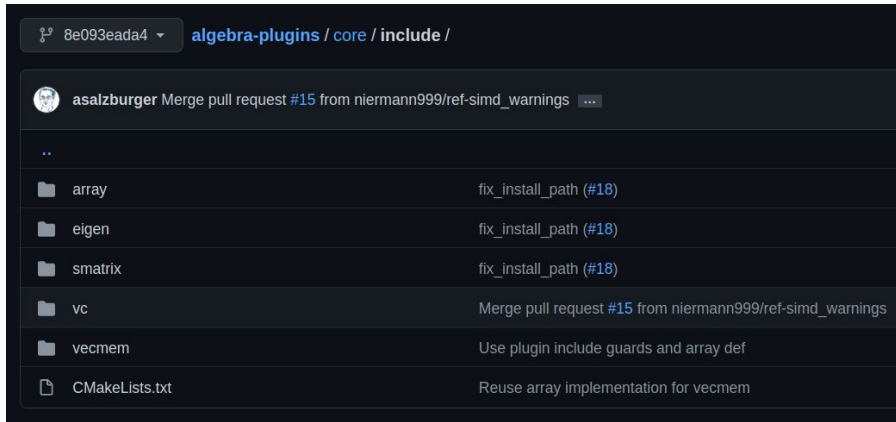
Conversation 14 Commits 3 Checks 49 Files changed 6

beomki-yeo commented 11 days ago

This PR adds cuda qualifiers to algebra functions. Currently bunch of warning or errors messages come from `traccc` out of this

Add cuda qualifiers to algebras ✓ 768802b

Current Code Organisation

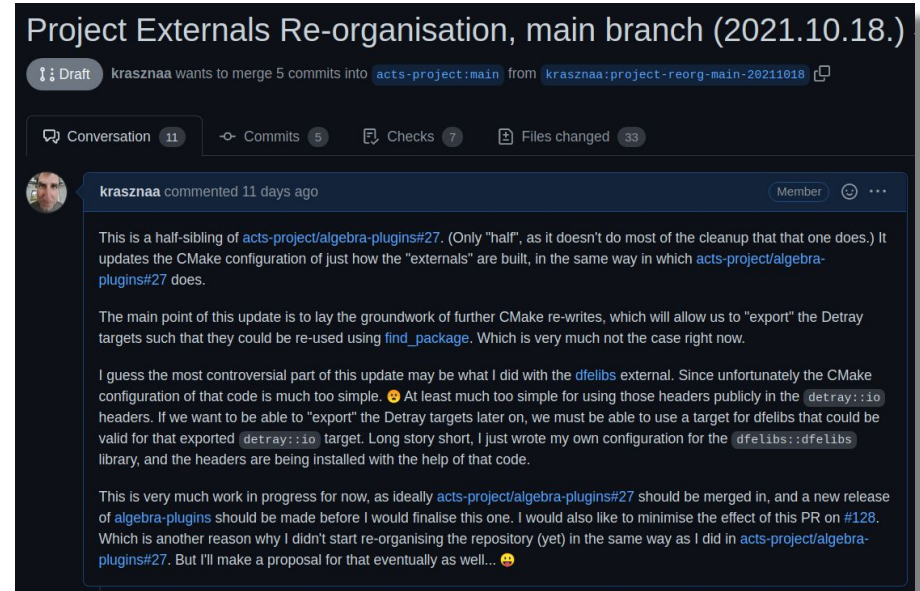


- The current code used in `detray`, which still precedes [algebra-plugins#27](#), provides the following libraries:
 - [algebra::array](#), [algebra::eigen](#), [algebra::smatrix](#), [algebra::vc](#): Providing `std::` or `vecmem::` vector definitions, based on what was enabled
 - [algebra::vecmem](#): Always providing `vecmem::` vector definitions
- It also shares a lot of headers between these libraries, using different preprocessor definitions to choose how those headers should behave
 - Making it impossible to use more than one of those libraries/headers at the same time

Detray Code Reorganisation



- After the cleanup in [algebra-plugins#27](#), I wanted to do something similar in [Detray](#) as well
 - At first just making it build its externals in the same way as I made algebra-plugins do it
 - Then, I would move on to cleaning up how the Detray libraries/executables/tests would be set up
- However I had to realise that [algebra-plugins#27](#) was not compatible with how the headers are currently being used in Detray 😞



algebra-plugins Rewrite



Code Rewrite, main branch (2021.10.22.) #31

Open krasznaa wants to merge 27 commits into `acts-project-main` from `krasznaa:code-rewrite-main-20211021`

Conversation 9 ← Commits 27 Checks 50 Files changed 97

krasznaa commented 7 days ago Member

With a bit of a delay, this is the (actually!) scary PR that I mentioned on Wednesday @niermann999.

It's still a work in progress, and the commits will probably need to be squashed eventually, but I wanted to give everybody a look at how I would imagine this project to work in the future. Though I imagine that I'll need to give a presentation about this, before it could be merged in.

The idea here is to properly separate in the code how data would be stored in memory by the users, and how calculations on that data would be done. The other basic idea is that I want to get rid of practically all `#ifdef`-s from the code. What I'm setting up here is a code organisation in which the type of an "algebra class/function" would clearly state

- what type of objects it operates on;
- what type of code it uses "internally" to perform its calculations.

The new structure of the code is the following:

- I still kept the `algebra::common` library, though by now only a single header (`algebra/scalar.hpp`) is used from it, with a single definition. Later on I will want to get rid of this as well, but for now I decided to keep the compile-time decision about the scalar type.
- I introduced the `storage/` directory to the repository, which holds headers/libraries declaring how data should be held in memory. The current libraries are:
 - `algebra::array_storage`: Using `std::array`, always enabled.
 - `algebra::eigen_storage`: Using `Eigen::Matrix`, enabled by `ALGEBRA_PLUGIN_INCLUDE_EIGEN`.
 - `algebra::smatrix_storage`: Using `ROOT::Math::SVector`, enabled by `ALGEBRA_PLUGIN_INCLUDE_SMATRIX`.
 - `algebra::vc_storage`: Using `Vc::SVecArray`, enabled by `ALGEBRA_PLUGIN_INCLUDE_VC`.
 - `algebra::vecmem_storage`: Using `vecmem::static_array`, enabled by `ALGEBRA_PLUGIN_INCLUDE_VECMEM`.
- I introduced the `math/` directory, which would hold all the "mathematics" with which data could be processed. The current libraries are:
 - `algebra::cmath_math`: Using "basic C++" `<cmath>`, always enabled.
 - `algebra::eigen_math`: Using `Eigen::Matrix`, enabled by `ALGEBRA_PLUGIN_INCLUDE_EIGEN`.
 - `algebra::smatrix_math`: Using `ROOT::Math::SMatrix`, enabled by `ALGEBRA_PLUGIN_INCLUDE_SMATRIX`.
 - `algebra::vc_math`: Using Vc vectorised operations, enabled by `ALGEBRA_PLUGIN_INCLUDE_VC`.
- On top of these components I introduced the `frontend/` directory, which would hold libraries creating storage model - mathematics combinations. I will not list all of them here, have a look yourself. 🤪
- For the tests I still kept `test_plugin.inl` with minimal modifications, even though I have ideas for making that a whole lot smarter as well. 🤪
 - The change was that it now tests the "matrices" used by the transformations internally, for every "mathematics type". Not just the ones for which `__plugin_without_matrix_element_accessor` would not be declared.

As you can see, I generalised the heck out of `algebra::cmath_math`, to make it work on top of every storage model. I also taught `algebra::vc_math` to work with all of `algebra::array_storage`, `algebra::vecmem_storage`, and of course `algebra::vc_storage`. I'm still pondering about making `algebra::eigen_math` and `algebra::smatrix_math` capable of working with something other than their "own storage type", but that may not be super useful in the end.

So... Have a look, and let me know what you think. 🤪

Pinging @beomki-yeo, @stephenswat, @paulgessinger, @asalzburger.

- At that point I decided to rewrite algebra-plugins in a quite fundamental way, so that it would not have to use preprocessor definitions to function
- That became [algebra_plugins#31](#), which this talk is about...

Storage / Math Separation

- I wanted to make the “data storage” and “math implementation” separated
 - To allow us to mix-and-match these as we’d like
 - Since the “math implementation” between the current libraries has a lot of copy-paste, this also allows us to have every function/class only defined once
- For now only the “cmath math implementation” can use vectors from all data storage implementations
 - But this could very well be extended later on

```

10 // Project include(s).
11 #include "algebra/common/algebra_qualifiers.hpp"
12 #include "algebra/math/impl/cmath_getter.hpp"
13 #include "algebra/math/impl/cmath_vector.hpp"
14
15 namespace algebra::cmath {
16
17 /** Transform wrapper class to ensure standard API within differnt plugins
18 **/
19 template <typename, auto> class array_t, typename scalar_t,
20         typename matrix44_t = array_t<array_t<scalar_t, 4>, 4>,
21         class element_getter_t = element_getter<array_t, scalar_t>,
22         class block_getter_t = block_getter<array_t, scalar_t>,
23         typename vector3_t = array_t<scalar_t, 3>,
24         typename point2_t = array_t<scalar_t, 2> >
25 struct transform3 {
26
27     /// @name Type definitions for the struct
28     /// @{
29
30     /// Array type used by the transform
31     template <typename T, auto N>
32     using array_type = array_t<T, N>;
33     /// Scalar type used by the transform
34     using scalar_type = scalar_t;
35
36     /// 3-element "vector" type
37     using vector3 = vector3_t;
38     /// Point in 3D space
39     using point3 = vector3;
40     /// Point in 2D space
41     using point2 = point2_t;
42
43     /// 4x4 matrix type
44     using matrix44 = matrix44_t;
45
46     /// Function (object) used for accessing a matrix element
47     using element_getter = element_getter_t;
48     /// Function (object) used for accessing a sub-matrix of a matrix
49     using block_getter = block_getter_t;
50
51     /// @}
52
53     /// @name Data objects
54     /// @{
55
56     matrix44 _data;
57     matrix44 _data_inv;
58
59     /// @}

```

“Frontend” Libraries

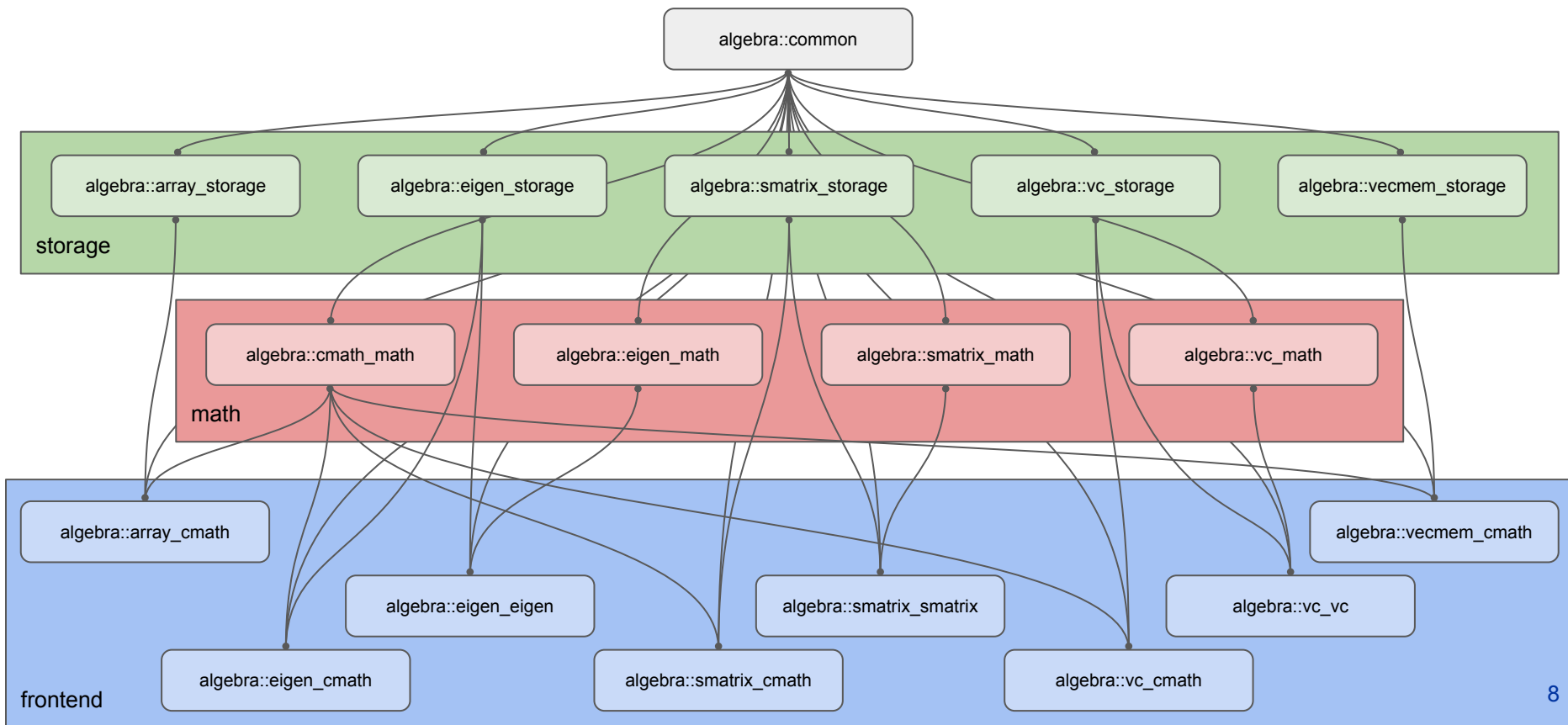
```

10 // Project include(s).
11 #include "algebra/common/scalar.hpp"
12 #include "algebra/math/cmath.hpp"
13 #include "algebra/storage/array.hpp"
14
15 namespace algebra {
16
17     /// @name Operators on @c algebra::array::storage_type
18     /// @{
19
20     using cmath::operator*;
21     using cmath::operator-;
22     using cmath::operator+;
23
24     /// @}
25
26     namespace array {
27
28         /// @name cmath based transforms on @c algebra::array::storage_type
29         /// @{
30
31         using transform3 = cmath::transform3<array::storage_type, scalar>;
32         using cartesian2 = cmath::cartesian2<transform3>;
33         using polar2 = cmath::polar2<transform3>;
34         using cylindrical2 = cmath::cylindrical2<transform3>;
35
36         /// @}
37     } // namespace array
38
39     namespace getter {
40
41         /// @name Getter functions on @c algebra::array::storage_type
42         /// @{
43
44         using cmath::eta;
45         using cmath::norm;
46         using cmath::perp;
47         using cmath::phi;
48         using cmath::theta;
49
50         /// @}
51     }

```

- Storage types would be combined with math implementations in “frontend libraries”
- These would define “the correct” math types/functions in the “global namespaces”
 - Client code currently assumes that separate namespaces are used for the vector type and transform declarations, but that “helper functions” would always be in `algebra::getter` and `algebra::vector`

Library Re-Organisation



Testing The Changes



- Since the algebra-plugins unit tests do not test every aspect of the code (something to improve upon later on), I was testing my changes in [de_tray#129](#).

- The biggest question lately was not whether the re-written code would work, but how it would affect the performance of everything.
- This is a bit hard to answer... 😞 I would argue that it does not affect performance in any significant way
- Some more details can be found [here](#)

<foo>_intersect_all times in milliseconds, on an AMD Threadripper 3970X

double precision			
Plugin Type	Current algebra-plugins	New algebra-plugins	New algebra-plugins with aggressive Vc optimisation
array	1169 ± 2	1200 ± 10	1141 ± 2
eigen	1373 ± 8	1229 ± 2	1183 ± 1
smatrix	2645 ± 14	2625 ± 6	2658 ± 3
vc	3691 ± 5	2678 ± 2	1355 ± 4
float precision			
Plugin Type	Current algebra-plugins	New algebra-plugins	New algebra-plugins with aggressive Vc optimisation
array	1042 ± 4	1061 ± 15	980 ± 1
eigen	1270 ± 5	1307 ± 3	1192 ± 2
smatrix	2493 ± 2	1568 ± 2	1529 ± 2
vc	1040 ± 1	1148 ± 1	1067 ± 5

Backup

Dot Graph





<http://home.cern>