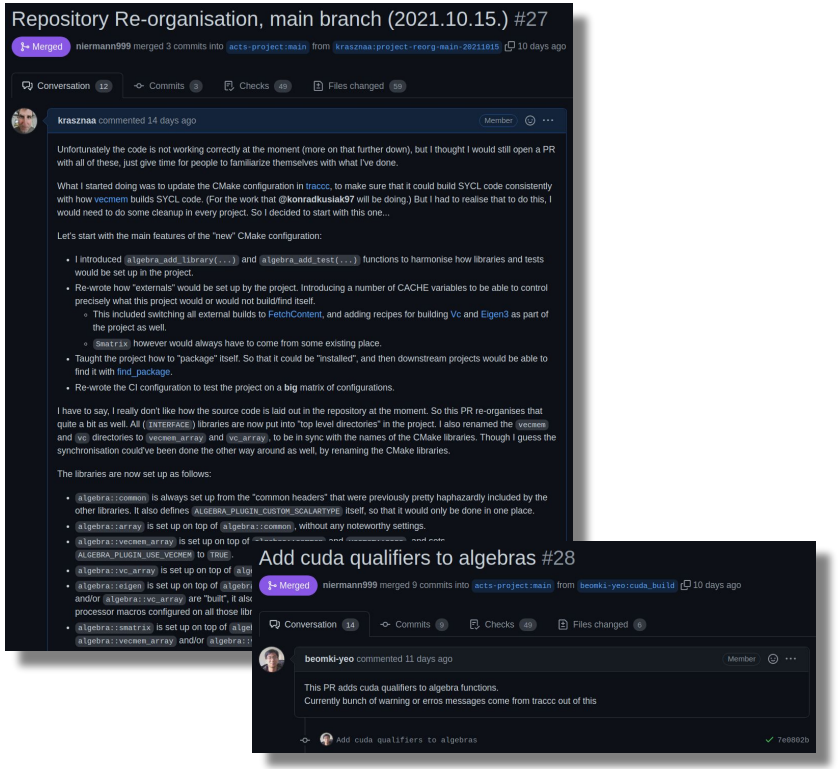
algebra-plugins rewrite

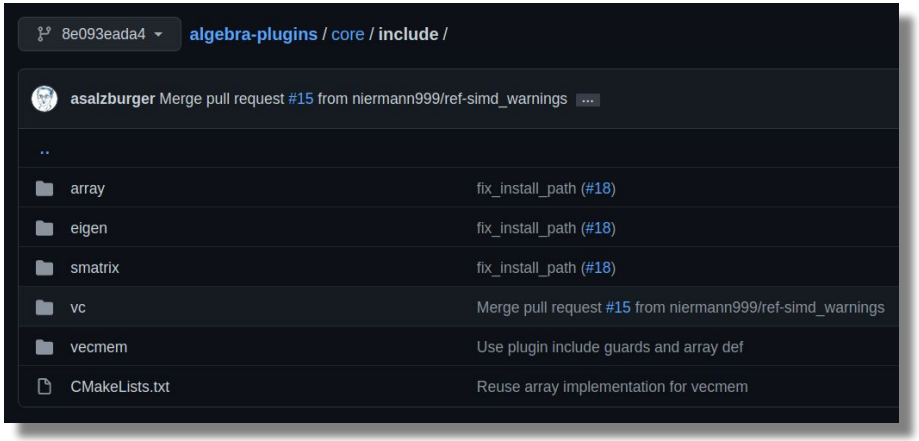
Attila Krasznahorkay

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



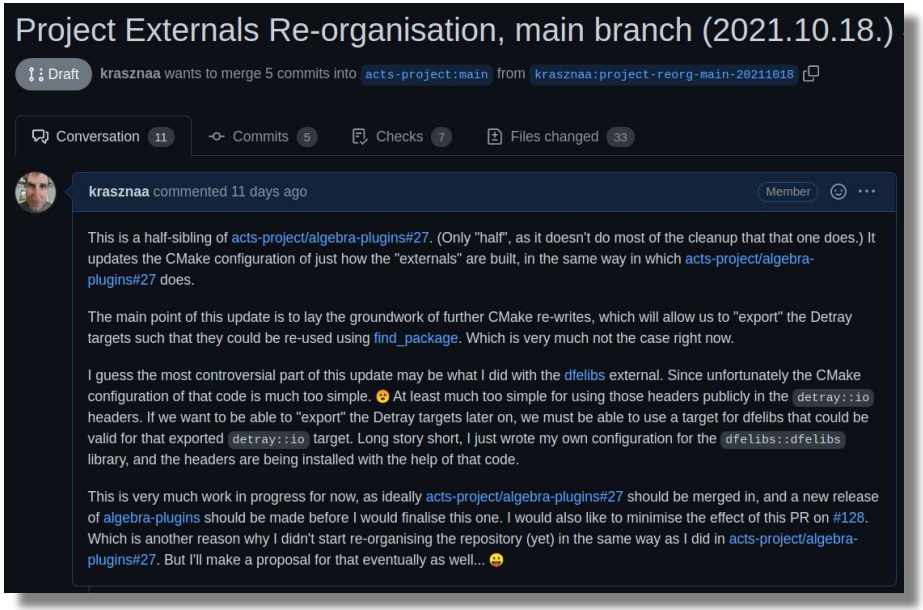
Current Code Organisation



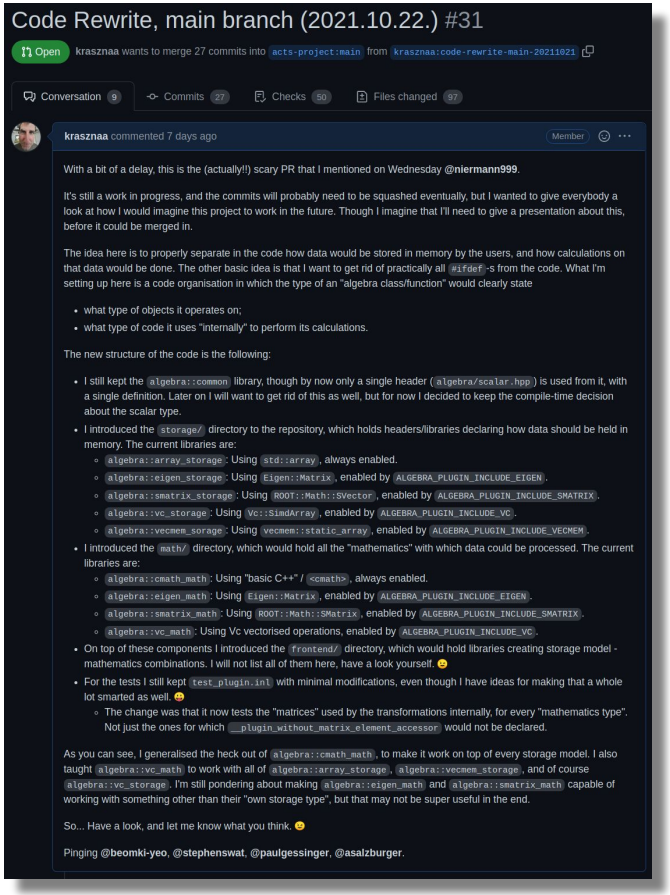
- The current code used in detray, which still precedes algebra-plugins#27, provides the following libraries:
  - algebra::array, algebra::eigen, algebra::smatrix, algebra::vc: Providing `std::` or `vecmem::` vector definitions, based on what was enabled
  - algebra::vecmem: Always providing `vecmem::` 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::SimdArray`, 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 smarted 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

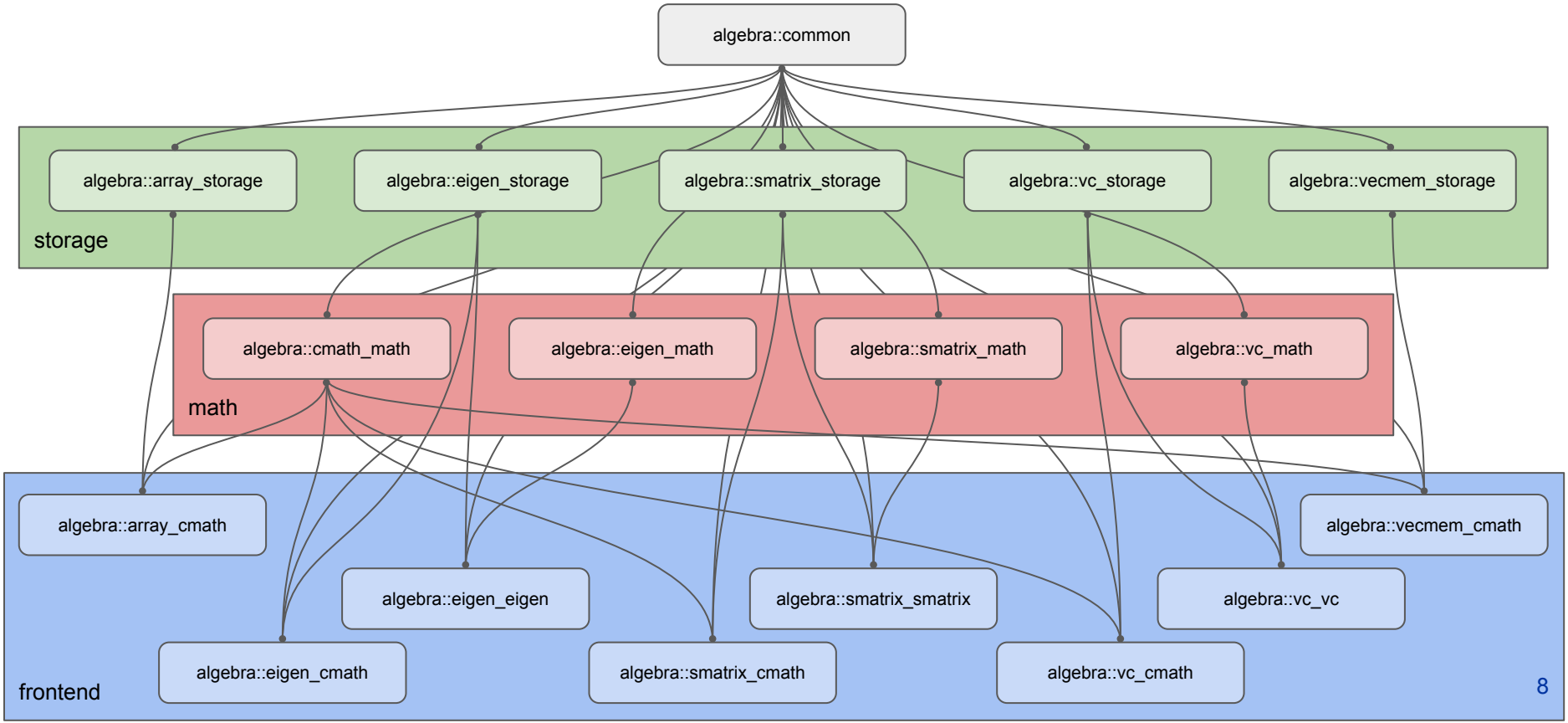
```
// Project include(s).
#include "algebra/common/algebra_qualifiers.hpp"
#include "algebra/math/impl/cmath_getter.hpp"
#include "algebra/math/impl/cmath_vector.hpp"

namespace algebra::cmath {

/** Transform wrapper class to ensure standard API within differnt plugins
 **/
template <template <typename, auto> class array_t, typename scalar_t,
          typename matrix44_t = array_t<array_t<scalar_t, 4>, 4>,
          class element_getter_t = element_getter<array_t, scalar_t>,
          class block_getter_t = block_getter<array_t, scalar_t>,
          typename vector3_t = array_t<scalar_t, 3>,
          typename point2_t = array_t<scalar_t, 2> >
struct transform3 {

  /// @name Type definitions for the struct
  /// @{

  /// Array type used by the transform
  template <typename T, auto N>
  using array_type = array_t<T, N>;
  /// Scalar type used by the transform
  using scalar_type = scalar_t;

  /// 3-element "vector" type
  using vector3 = vector3_t;
  /// Point in 3D space
  using point3 = vector3;
  /// Point in 2D space
  using point2 = point2_t;

  /// 4x4 matrix type
  using matrix44 = matrix44_t;

  /// Function (object) used for accessing a matrix element
  using element_getter = element_getter_t;
  /// Function (object) used for accessing a sub-matrix of a matrix
  using block_getter = block_getter_t;

  /// @}

  /// @name Data objects
  /// @{

  matrix44 _data;
  matrix44 _data_inv;

  /// @}
```

“Frontend” Libraries

```
// Project include(s).
#include "algebra/common/scalar.hpp"
#include "algebra/math/cmath.hpp"
#include "algebra/storage/array.hpp"

namespace algebra {

/// @name Operators on @c algebra::array::storage_type
/// @{

using cmath::operator*;
using cmath::operator-;
using cmath::operator+;

/// @}

namespace array {

/// @name cmath based transforms on @c algebra::array::storage_type
/// @{

using transform3 = cmath::transform3<array::storage_type, scalar>;
using cartesian2 = cmath::cartesian2<transform3>;
using polar2 = cmath::polar2<transform3>;
using cylindrical2 = cmath::cylindrical2<transform3>;

/// @}

}  // namespace array

namespace getter {

/// @name Getter functions on @c algebra::array::storage_type
/// @{

using cmath::eta;
using cmath::norm;
using cmath::perp;
using cmath::phi;
using cmath::theta;

/// @}
```

- 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

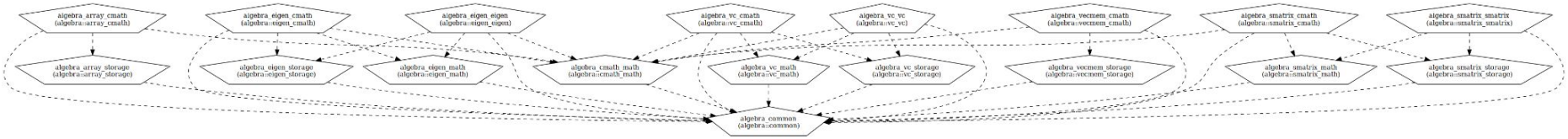
- Since the algebra-plugins unit tests do not test every aspect of the code (something to improve upon later on), I was testing my changes in detray#129.
  - The biggest question lately was not whether the re-written code would work, but how it would affect the performance of everything.
  - This is 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


algebra_array_cmath
(algebra::array_cmath)
algebra_eigen_cmath
(algebra::eigen_cmath)
algebra_eigen_eigen
(algebra::eigen_eigen)
algebra_vc_cmath
(algebra::vc_cmath)
algebra_vc_vc
(algebra::vc_vc)
algebra_vecmem_cmath
(algebra::vecmem_cmath)
algebra_smatrix_cmath
(algebra::smatrix_cmath)
algebra_smatrix_smatrix
(algebra::smatrix_smatrix)
algebra_array_storage
(algebra::array_storage)
algebra_eigen_storage
(algebra::eigen_storage)
algebra_eigen_math
(algebra::eigen_math)
algebra_cmath_math
(algebra::cmath_math)
algebra_vc_math
(algebra::vc_math)
algebra_vc_storage
(algebra::vc_storage)
algebra_vecmem_storage
(algebra::vecmem_storage)
algebra_smatrix_math
(algebra::smatrix_math)
algebra_smatrix_storage
(algebra::smatrix_storage)
algebra_common
(algebra::common)

http://home.cern