Vectorization of the VELO-UT tracking

Short internal report

Péter Kardos

Please check out the *Design of the improved algorithm* section of my thesis report for a detailed description of the algorithm itself. All code is on branch pkardos_PrVeloUtOpt.

1 Artifacts

1.1 New VELO-UT algorithm & hit handler

Rec/Pr/PrVeloUT/src/
- *Constants.h*
- *Constants.cpp*
- *PrVeloUtOpt.h*
- *PrVeloUtOpt.cpp*
- *PrStoreUtHitOpt.h*
- *PrStoreUtHitOpt.cpp*
- *UtHitSpacePartition.h*
- *UtHitSpacePartition.cpp*
- *UtUtil.h*
- *UtUtil.cpp*

Next to the original `PrVeloUT` algorithm, I added another one called `PrVeloUtOpt`. The new algorithm, `PrVeloutOpt`, is mostly the same, but its code is in line with clean coding practices (as much as was possible) and it is vectorized and optimized. The `HitHandler` is replaced by the `UtHitSpacePartition` which uses uniform grid space partitioning instead of per-sector partitioning. The `PrStoreUtHitOpt` algorithm writes hits in this new scheme. `Contants` are hardcoded properties of the detector geometry, `UtUtil` contains solution to common problems such as accessing B dL values.

1.2 SOA data model "framework"

Rec/Pr/PrVeloUT/src/
SoaContainerify.hpp

Adds `resize`, `reserve` and `size` methods to SOA struct declarations using the boost::PFR library to discover and manipulate individual member variables at compile time.

Example:

```
struct PointsBase {
    std::vector<float> x;
    std::vector<float> y;
    std::array<std::vector<int>, 3> meta;
};
using Points = SoaContainerify<PointsBase>;

Points points;
points.resize(10);
assert(points.x.size() == 10);
assert(points.y.size() == 10);
assert(points.meta[0].size() == 10);
assert(points.size() == 10);
```

1.3 Supporting code

Rec/Pr/PrVeloUT/src/
SimdUtil.h
StackAllocator.h
AlignedAllocator.hpp

`SimdUtil` contains some stuff to wrap Vc and give matching vector sizes for the VELO-UT algorithm. It also contains SIMD pruning helper function that are interoperable with Vc.

The `StackAllocator` is a simple STL allocator that acquires aligned chunks of memory from a large upfront memory pool.

The `AlignedAllocator` is an STL allocator that wraps STL' s `aligned_alloc` functions to force different alignment than `std::allocator<T>` would give. Disables zero-initialization of primitive arithmetic types such as `float`.

1.4 Boost PFR library

*Rec/Pr/PrVeloUT/src/boost/**

A template metaprogramming helper library that provides a tuple-like interface for plain structs. Not officially part of boost.

1.5 Auxiliary algorithms

Rec/Pr/PrMCTools/src/PrLHCbID2MCParticle.cpp

As the new algorithm no longer uses the `HitHandler`, the `PrLHCbID2MCParticle` had to be modified to extract the LHCbIDs from an `UtHitSpacePartition` object.

2 Implementation details

2.1 Overallocation

Note that the code relies on underlying allocators allocating space with one more entire SIMD vector available at the end of the requested block. This is because the loops don't mask the last element, just read it regardless. The allocators giving the exact number of bytes allocated likely leads to segfaults and nasty things with SIMD loops. To fix this, masked reads and writes or at least documentation to the code could be added. The code is memory bound anyways, adding a mask calculation may not have any adverse effects on performance.

3 Results

In isolated benchmarks, on busy events, the optimized version is about 2.5 times faster than the original. In the framework, it is somewhat less. The old version produces 12300 evt/s whereas the new does 12600 with this and this configuration.
TODO: add exact number and exact config
TODO: insert efficiency table from checker

4 How to integrate, properly

4.1 Current state

All it takes to get the new algorithm working is changing three lines in the configuration files to start using them. The algorithm runs properly and produces the expected results.

4.2 Arranging files

I added all new code to *Rec/Pr/PrVeloUT/src/*, including support files and third party libs. That is obviously not correct. Boost PFR needs to be merged with boost in LCG. The helper files, such as the SOA framework and UT tools may be useful in other contexts, and should be moved accordingly.

4.3 CMake dependencies

Pr/PrMCTools/CMakeLists.txt is forced to depend on a sibling library, *Pr/PrVeloUT/CMakeLists.txt*, because it requires the definitions of the `UtHitSpacePartition`. This is not good, and it can be solved by moving `UtHitSpacePartition` (and all its dependencies) to LHCb, which is also not good because there is no other reason to move these files.

Pr/PrVeloUT/CMakeLists.txt was hacked to emit a library to be consumed by *Pr/PrMCTools/CMakeLists.txt*, which is not good either – this is because MC matching (`PrLHCbID2MCParticle.` needs to definition of the `UtHitSpacePartition`. Additionally, it now depends on Vc libraries, that's perfectly fine.

4.4 Contants.h

Most information in this file should be retrieved from geometry services instead of being hardcoded. The file should be deleted and the rest of the code updated accordingly.

4.5 Namespaces, names

Naming and namespaces are off, as there is not a clear convention in the current software. A clear convention should be decided and code adjusted to it.

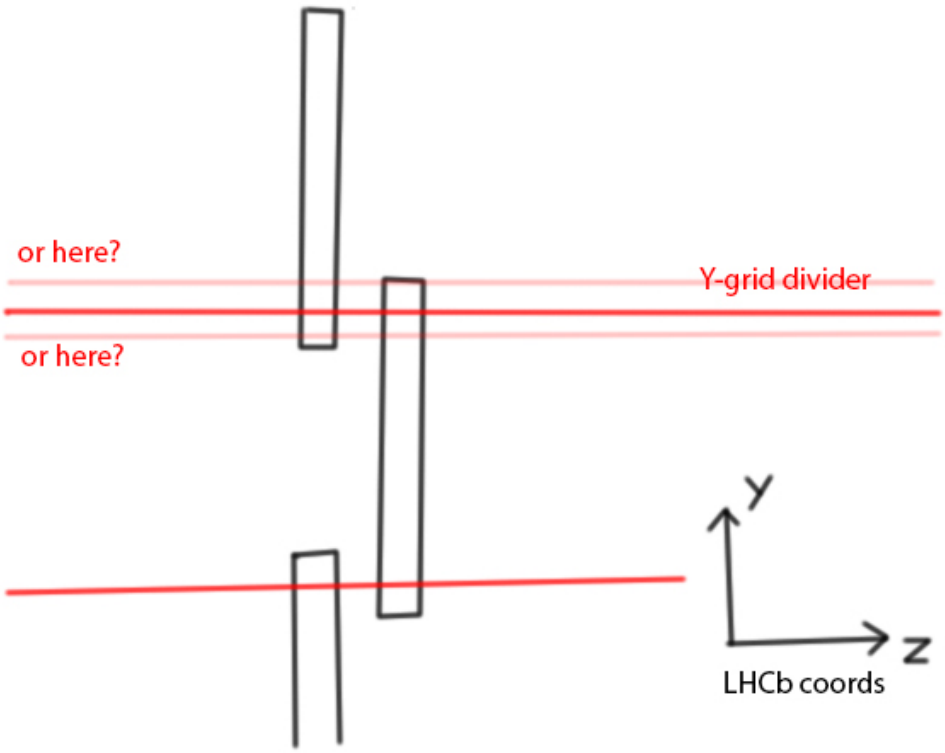
4.6 Compilation

Flags AVX2, BMI and BMI2 need to be specified. This basically means Haswell.

5 Future work and known issues

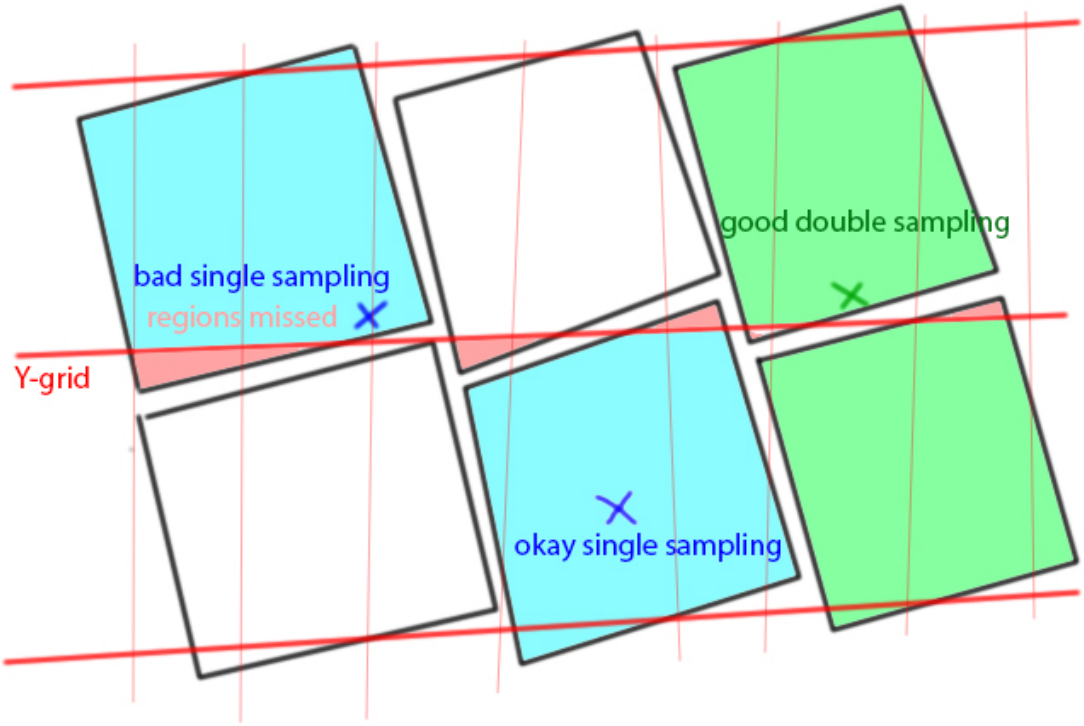
5.1 Align space partitioning properly on Y

The uniform grid space partitioning should be tightly aligned to UT sensor rows. The grid's Y ticks should be exactly where the non-tilted sensors of the UT meet, possibly shifted a few millimeters up or down to make the most of the overlap of the UT sensors when the track hits at an angle. Currently I just took the upper and lower limits of the UT from the TDR an divided it into 14 equal rows.



5.2 Add double binning on Y

Efficiencies are slightly lower compared to the original PrVeloUT, this is probably because there is no double binning on the Y axis - that is, only fiber hits from a single row are considered. In case the track hits between two rows, the overlap of the rows should take care of the problem, but for U and V layers hits may still be missed due to the tilt. By looking up both adjacent rows, one burns some more CPU cycles but it may help with efficiency. To be tried.



5.3 Vectorize aligned hit searching

`FindLinesForTrack` is a heavy CPU consumer yet it is not vectorized at all. The loops are too small and branching makes it impractical. Maybe there is a clever way? Maybe a completely different algorithm? Maybe vectorizing it over tracks instead?

5.4 AVX-512

Should work without problems, where the algorithm needs the vector width it is acquired from a global constexpr `SimdWidth` which is currently acquired from Vc. The AVX2 emulation of the compressed store should be replaced with native intrinsics. Unfortunately, Vc does not support AVX-512, and the code is written with Vc.