CERN

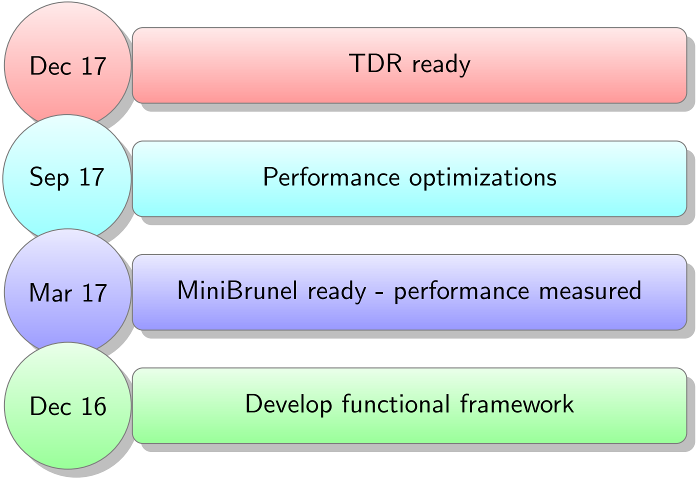
Software upgrade status

Sébastien Ponce
sebastien.ponce@cern.ch

Context

- TDR
  - due by end 2017
  - needs numbers for taking decisions
- Internal milestone + Review
  - End of March / April / May
  - current status
  - evaluation of the strategy

The master plan



Dec 17 TDR ready

Sep 17 Performance optimizations

Mar 17 MiniBrunel ready - performance measured

Dec 16 Develop functional framework

Areas concerned

- LHCb core framework
- Event model
- Conditions
- Detector Description
- Collaboration training

Outline

Current Status

- Conditions
- Detector description
- Event model
- Core framework

Next / Current Steps

Collaboration training

Conclusions

Outline

- Current Status
  - Conditions
  - Detector description
  - Event model
  - Core framework
- Next / Current Steps
- Collaboration training
- Conclusions

Conditions - The goals

- Adapt our conditions' interfaces to multithreaded environment
  - multiple concurrent events
  - may not have same conditions
- Change our transient representation
  - triggered by the move to DD4Hep
- Change our persistent representation
  - from XML to something simpler to parse
- Change our tools to manage condition files
  - from COOL/Coral to something simpler and maintained

Conditions - Status

- A prototype of thread safe conditions have been proposed for Gaudi
  - under discussion at Gaudi level
  - looks promising from LHCb point of view
  - challenge is to make it happen
- Conditions Management switched to git
  - faster, smaller, easier than COOL/Coral
  - allows to drop a lot of code
  - ready to be used, being commissionned for 2017 run
- Representation changes not yet started

Detector Description

Current implementation

- is not thread safe
- is not maintained

Plans

- Investigate DD4Hep as replacement
- Use a minimal geometry
  - to speed up tracking

DetDesc - Status

- Full geometry has been converted to DD4Hep
    - validation is almost finished
    - integration to LHCb is the next step
- The minimal geometry has been defined
    - is default in MiniBrunel
- Efficiency of the code has been reviewed
    - not optimal, opportunity of optimizations

Event model : what changes ?

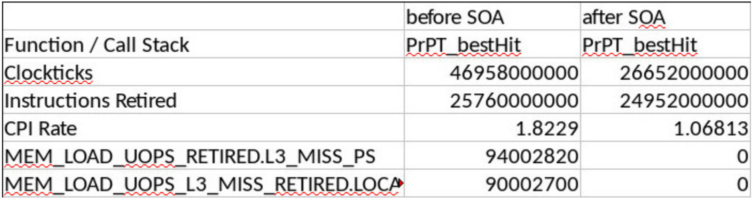
- Change access pattern to the Event Store
  - write once, read-only after write
  - imposed by multi-threading
- Introduce Structures of Arrays (SoA)
  - to boost gains due to auto vectorization
- Review usage of doubles
  - replace with floats when possible

Event Model - Status

- TES is now read-only
  - less impact than foreseen
  - most code can be adapted by splitting objects
- Composition can be achieved using “range v3”
  - often enough for transient data
  - used successfully to port the RICH code
- SoA components have been developed
  - now tested in HLT1 context
  - working as expected
  - small improvements to be done
- Switching double to floats has been tested
  - Vectorized Kalman filter goes 2x faster

Example of SoA on PixelTracking

```
const PrPixelHit* bestHit(const PrPixelModuleHits& modulehits, ...)
class PrPixelModuleHits final {
  std::vector<PrPixelHit> m_hits;
};
class PrPixelHit final {
  float m_x;
  float m_y;
  float m_z;
};
```



	before SOA	after SOA
Function / Call Stack	PrPT_bestHit	PrPT_bestHit
Clockticks	46958000000	26652000000
Instructions Retired	25760000000	24952000000
CPI Rate	1.8229	1.06813
MEM_LOAD_UOPS_RETIRED.L3_MISS_PS	94002820	0
MEM_LOAD_UOPS_L3_MISS_RETIRED.LOCA	90002700	0

In this test, SOA was crafted manually

SOAContainer & SOAView

```
// AOS - style object
struct Hit {
  float m_x;
  float x() const noexcept { return m_x; }
};

// SOA - style
struct HitFields {  // fields defined as types
  typedef struct : public SOATypelist::wrap_type<float> {} f_x;
}
// Skin decorating HitFields
template ... struct HitSkin : ... , HitFields {
  auto & x() const noexcept { return this->template get<f_x> (); }
}
SOAContainer <std::vector, HitSkin, HitFields::f_x> hits;
hits.reserve(...);
hits.emplace_back(...);
```

Event Model - Status

- TES is now read-only
  - less impact than foreseen
  - most code can be adapted by splitting objects
- Composition can be achieved using “range v3”
  - often enough for transient data
  - used successfully to port the RICH code
- SoA components have been developed
  - now tested in HLT1 context
  - working as expected
  - small improvements to be done
- Switching double to floats has been tested
  - Vectorized Kalman filter goes 2x faster

Event Model

Challenges

- Do we need transparent composition ?
  - for non transient data, the Packing step may save us
  - may be complex, not easy to use for the end user
- Validation of physics with floats
  - where can we use them ?
- ~~Usage of structure of arrays~~

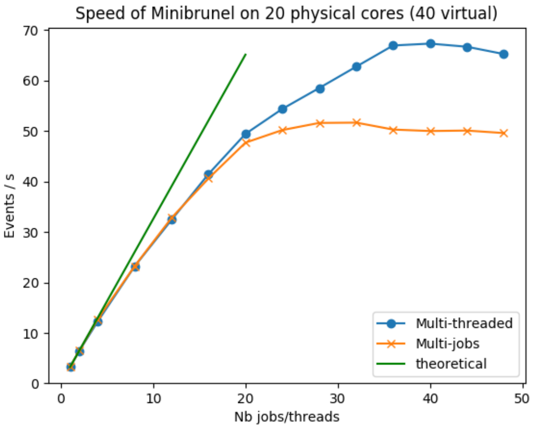
Risks

- Missing person power
  - has to come from the subsystems

Framework - Status

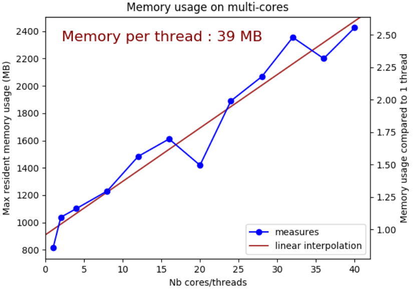
- The functional framework is functional
    - used in many algorithms (close to 100)
    - used in 2017 production
- "MiniBrunel"
    - is not so mini !
        - includes Kalman filter, full Rich reconstruction
    - it has an HLT1 version
- Performance extensively measured
    - very good behavior of multithreading
    - coherent with upgrade performance document

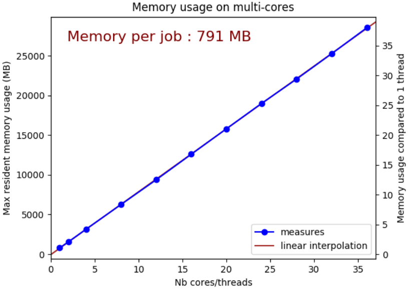
Acceleration with multithreading



Disclaimer : full MiniBrunel is used, in non optimized mode.
This is only measuring scalability, the absolute performance is to be ignored

Memory usage comparison





Framework

Challenges

- merge back future work into master branch
  - aggressively now that 2017 production is branched
- adapt the framework to an online usage
  - started and ongoing
- adapt the framework to simulation usage
  - multi event algorithms

Outline

- Current Status
  - Conditions
  - Detector description
  - Event model
  - Core framework
- Next / Current Steps
- Collaboration training
- Conclusions

Main goals

- merge master branch of git into future
  - done, to be redone regularly
  - failing merges to be handled by their author
- stabilize HLT1 MiniBrunel
  - dead locks and crashes under pressure
  - ongoing and almost done
- setup nightly performance tests
  - PR2 in place, including per algo performance
  - not yet giving satisfactory numbers
- optimize HLT1 code and measure speedup

Main goals

- merge master branch of git into future
  - done, to be redone regularly
  - failing merges to be handled by their author
- stabilize HLT1 MiniBrunel
  - dead locks and crashes under pressure
  - ongoing and almost done
- setup nightly performance tests
  - PR2 in place, including per algo performance
  - not yet giving satisfactory numbers
- **optimize HLT1 code and measure speedup**

HLT1 optimization targets

Identified thanks to extensive measurements

- usage of SOA
- optimization of objects in TES
- cut order
- better vectorization (AVX2)
- optimized binary search in PrPixelTracking
- drop monitoring
- optimized allocations
- optimize geometry code

MiniBrunel time distribution

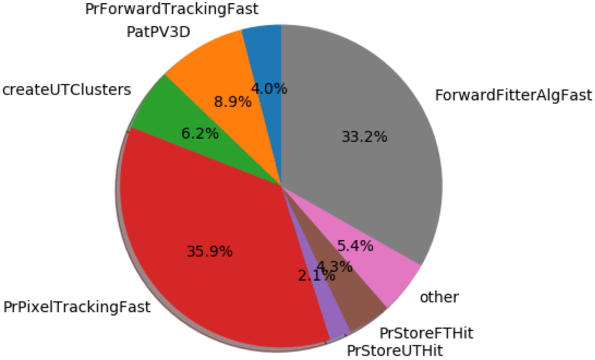


Figure: HLT1 on Minimum bias

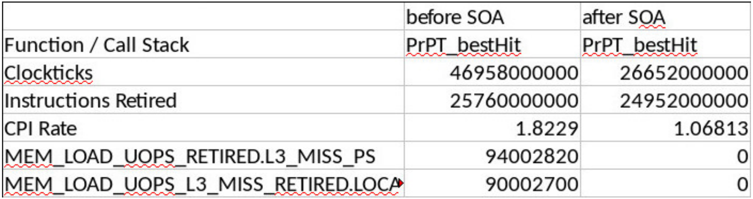
HLT1 optimization targets

Identified thanks to extensive measurements

- usage of SOA
- optimization of objects in TES
- cut order
- better vectorization (AVX2)
- optimized binary search in PrPixelTracking
- drop monitoring
- optimized allocations
- optimize geometry code

Example of SoA on PixelTracking

```
const PrPixelHit* bestHit(const PrPixelModuleHits& modulehits, ...)
class PrPixelModuleHits final {
  std::vector<PrPixelHit> m_hits;
};
class PrPixelHit final {
  float m_x;
  float m_y;
  float m_z;
};
```



	before SOA	after SOA
Function / Call Stack	PrPT_bestHit	PrPT_bestHit
Clockticks	46958000000	26652000000
Instructions Retired	25760000000	24952000000
CPI Rate	1.8229	1.06813
MEM_LOAD_UOPS_RETIRED.L3_MISS_PS	94002820	0
MEM_LOAD_UOPS_L3_MISS_RETIRED.LOCA	90002700	0

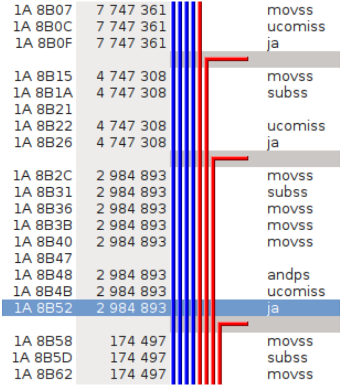
In this test, SOA was crafted manually

HLT1 optimization targets

Identified thanks to extensive measurements

- usage of SOA
- optimization of objects in TES
- cut order
- better vectorization (AVX2)
- optimized binary search in PrPixelTracking
- drop monitoring
- optimized allocations
- optimize geometry code

PrPixel cut efficiency



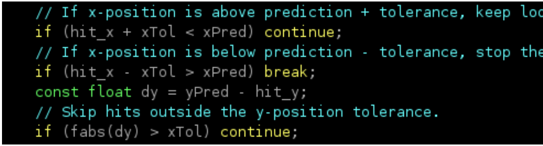
```
1A 8B07   7 747 361   movss
1A 8B0C   7 747 361   ucomiss
1A 8B0F   7 747 361   ja

1A 8B15   4 747 308   movss
1A 8B1A   4 747 308   subss
1A 8B21
1A 8B22   4 747 308   ucomiss
1A 8B26   4 747 308   ja

1A 8B2C   2 984 893   movss
1A 8B31   2 984 893   subss
1A 8B36   2 984 893   movss
1A 8B3B   2 984 893   movss
1A 8B40   2 984 893   movss
1A 8B47
1A 8B48   2 984 893   andps
1A 8B4B   2 984 893   ucomiss
1A 8B52   2 984 893   ja

1A 8B58     174 497   movss
1A 8B5D     174 497   subss
1A 8B62     174 497   movss
```

About ordering of the cuts...



```
// If x-position is above prediction + tolerance, keep loo
if (hit_x + xTol < xPred) continue;
// If x-position is below prediction - tolerance, stop the
if (hit_x - xTol > xPred) break;
const float dy = yPred - hit_y;
// Skip hits outside the y-position tolerance.
if (fabs(dy) > xTol) continue;
```

...are we optimal ?

HLT1 optimization targets

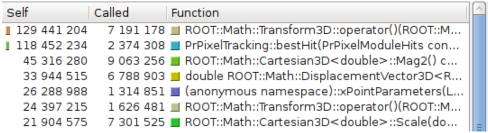
Identified thanks to extensive measurements

- usage of SOA
- optimization of objects in TES
- cut order
- better vectorization (AVX2)
- optimized binary search in PrPixelTracking
- drop monitoring
- optimized allocations
- optimize geometry code

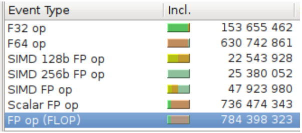
Flops usage and vectorization

Extension of callgrind allowing to count flops

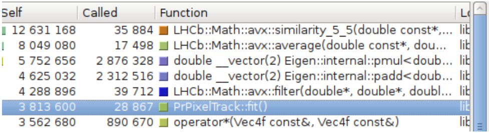
- and differentiate scalar from SIMD
- but also different vector widths
- and floats from doubles



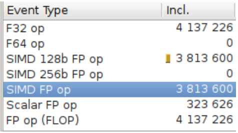
Self	Called	Function
129 441 204	7 191 178	🟫 ROOT::Math::Transform3D::operator()(ROOT::M...
118 452 234	2 374 308	🟦 PrPixelTracking::bestHit(PrPixelModuleHits con...
45 316 280	9 063 256	🟩 ROOT::Math::Cartesian3D<double>::Mag2() c...
33 944 515	6 788 903	🟨 double ROOT::Math::DisplacementVector3D<R...
26 288 988	1 314 851	🟪 (anonymous namespace)::xPointParameters(L...
24 397 215	1 626 481	🟫 ROOT::Math::Transform3D::operator()(ROOT::M...
21 904 575	7 301 525	🟩 ROOT::Math::Cartesian3D<double>::Scale(do...



Event Type	Incl.
F32 op	153 655 462
F64 op	630 742 861
SIMD 128b FP op	22 543 928
SIMD 256b FP op	25 380 052
SIMD FP op	47 923 980
Scalar FP op	736 474 343
FP op (FLOP)	784 398 323



Self	Called	Function	L
12 631 168	35 884	🟧 LHCb::Math::avx::similarity_5_5(double const*,...	lib
8 049 080	17 498	🟩 LHCb::Math::avx::average(double const*, dou...	lib
5 752 656	2 876 328	🟪 double __vector(2) Eigen::internal::pmul<doub...	lib
4 625 032	2 312 516	🟪 double __vector(2) Eigen::internal::padd<doub...	lib
4 288 896	39 712	🟦 LHCb::Math::avx::filter(double*, double*, doubl...	lib
3 813 600	28 867	🟩 PrPixelTrack::fit()	lib
3 562 680	890 670	🟨 operator*(Vec4f const&, Vec4f const&)	lib



Event Type	Incl.
F32 op	4 137 226
F64 op	0
SIMD 128b FP op	3 813 600
SIMD 256b FP op	0
SIMD FP op	3 813 600
Scalar FP op	323 626
FP op (FLOP)	4 137 226

HLT1 optimization targets

Identified thanks to extensive measurements

- usage of SOA
- optimization of objects in TES
- cut order
- better vectorization (AVX2)
- optimized binary search in PrPixelTracking
- drop monitoring
- optimized allocations
- optimize geometry code

PrPixel timing

```
const PrPixelHit *PrPixelTracking::bestHit() {
  // Do a binary search through the hits.
  unsigned int hit_start = ...
  // Find the hit that matches best.
  for (unsigned int i = hit_start; ...) { ... }
}
```

Where do we spend time ?

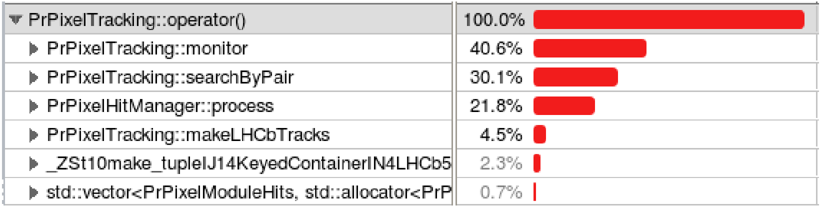
- 80% of instructions are in the loop
- but 60% of the time is spent in binary search

HLT1 optimization targets

Identified thanks to extensive measurements

- usage of SOA
- optimization of objects in TES
- cut order
- better vectorization (AVX2)
- optimized binary search in PrPixelTracking
- drop monitoring
- optimized allocations
- optimize geometry code

PrPixel timing



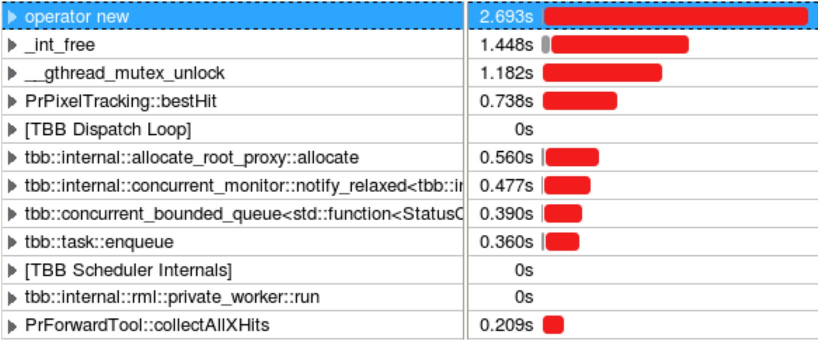
▼ PrPixelTracking::operator()	100.0%
▶ PrPixelTracking::monitor	40.6%
▶ PrPixelTracking::searchByPair	30.1%
▶ PrPixelHitManager::process	21.8%
▶ PrPixelTracking::makeLHCbTracks	4.5%
▶ _ZSt10make_tupleIJ14KeyedContainerIN4LHCb5	2.3%
▶ std::vector<PrPixelModuleHits, std::allocator<PrP	0.7%

HLT1 optimization targets

Identified thanks to extensive measurements

- usage of SOA
- optimization of objects in TES
- cut order
- better vectorization (AVX2)
- optimized binary search in PrPixelTracking
- drop monitoring
- optimized allocations
- optimize geometry code

Top time consumers in HLT1



operator new	2.693s
_int_free	1.448s
__gthread_mutex_unlock	1.182s
PrPixelTracking::bestHit	0.738s
[TBB Dispatch Loop]	0s
tbb::internal::allocate_root_proxy::allocate	0.560s
tbb::internal::concurrent_monitor::notify_relaxed<tbb::ir	0.477s
tbb::concurrent_bounded_queue<std::function<StatusC	0.390s
tbb::task::enqueue	0.360s
[TBB Scheduler Internals]	0s
tbb::internal::rml::private_worker::run	0s
PrForwardTool::collectAllXHits	0.209s

HLT1 optimization targets

Identified thanks to extensive measurements

- usage of SOA
- optimization of objects in TES
- cut order
- better vectorization (AVX2)
- optimized binary search in PrPixelTracking
- drop monitoring
- optimized allocations
- optimize geometry code

Outline

Current Status

- Conditions
- Detector description
- Event model
- Core framework

Next / Current Steps

Collaboration training

Conclusions

Training - Why ? Who ? What ?

- We are changing considerably the framework
  - the language used (C^{++}17 now)
  - the common practices (e.g. TES, functional)
  - the tools around it (e.g. git)
- Everyone is impacted
  - many will have to convert code
  - others will write new algorithms
- We need a substantial training effort
  - on the languages and tools
  - on the best practices

Training - What was done so far

- Development kit
- Workshops and Hackathons
  - user oriented hackathons
  - extended C^{++}courses
  - tutorials on framework and tools
- Practical courses
  - converting code to new framework
  - vectorization
  - efficient cache usage (to come)

LHCb C^{++}courses

LHCb hackathons

Training - Effort need to continue

- improve development kit
- other hackathons
  - week of 19^{th} of June
- more courses
  - tutorial on tools for optimization this afternoon

Outline

- Current Status
  - Conditions
  - Detector description
  - Event model
  - Core framework
- Next / Current Steps
- Collaboration training
- Conclusions

Summary

- The framework is under control
- Event Model, Conditions and DetDesc are progressing well
    - but more effort will be needed
- We master performance measurement
    - and found many opportunities of optimizations
- Optimizations are ongoing
    - we will finally get numbers for the TDR !
- Training of subsystem developers is essential
    - as we need all of them to be involved
    - as most of the person power has to come from them

www.cern.ch