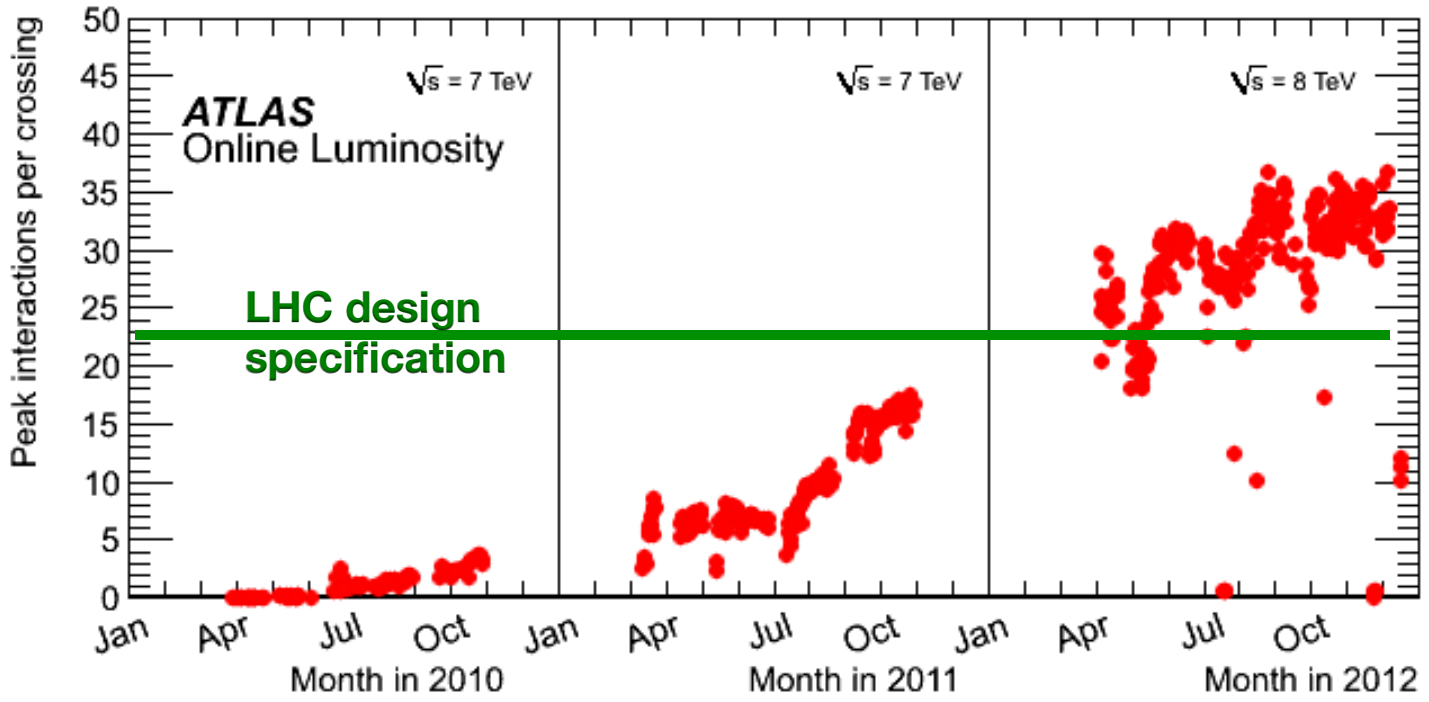
Preparing the Track Reconstruction in ATLAS for a high multiplicity future

Robert Langenberg (TU München, CERN),

For the ATLAS Collaboration

Outstanding LHC performance in run 1

- exceeded design specifications for pile-up
- put huge pressure and CPU & disk resources

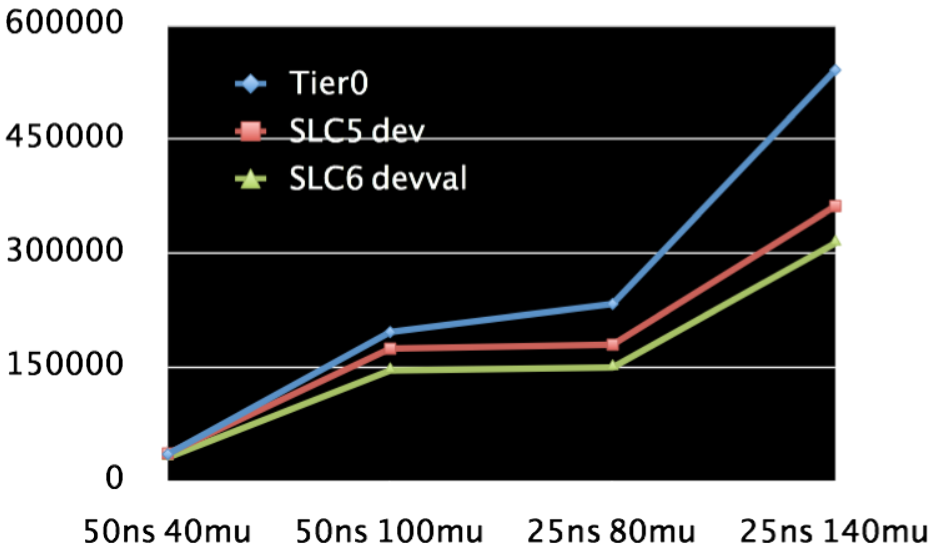


ID track reconstruction is main consumer

- Takes about 50 % of total time
- ~88 Mio channels in the Inner Detector only
- Pattern recognition is a combinatorial problem
- Reconstruction workload close to 100% of capacity
  - Memory usage is close to 4GB
  - Current production software doesn't allow parallel processing of events per program instance
  - Parallel CPU capabilities not exploited

Challenges for Run 2

- Pile-up will increase
  - depends on 25/50 ns
  - up to factor of 2.5



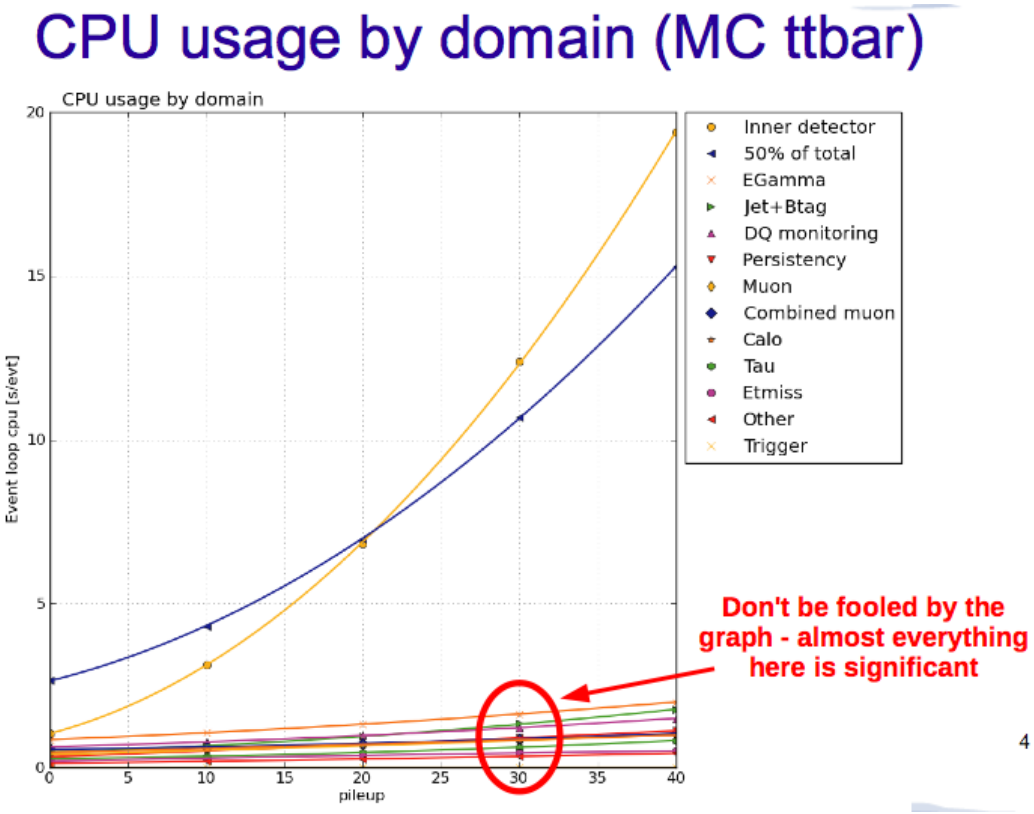
- Computing budget for Run 2 is constant
  - Electricity cost substantial factor
  - More machines or longer runtimes not affordable
  - Memory limited to below 3GB per core
- New machines don’t run faster nowadays but have more cores

Goals

- Increase computational throughput at least two-fold without decreasing physics performance
- Increase maintainability of code
- Show correctness of physics results of changed code
- Get it done before the end of LS1

Analysis Means

- Hotspot analysis
  - Wall time
    - I/O analysis
    - CPU utilization
  - Memory usage
    - Memory churn analysis
    - Heap/Stack analysis



Optimization Means

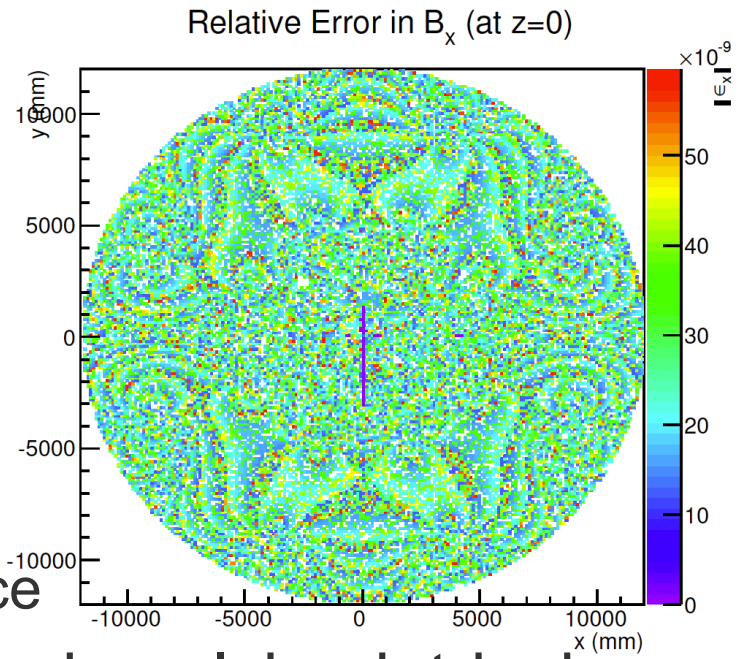
- Replacing algorithms
- Optimizing algorithms
  - High level (improving the algorithm, e.g. caching results)
  - Low level (tailoring the algorithm to suit hardware architecture)
- Change algorithms to run in parallel
  - By vectorizing
  - By multithreading

Applied Optimizations

- Replacing libraries
- Optimize memory usage
- Manually vectorizing algorithms
- Restructuring event data model (EDM)

Efforts: Magnetic Field

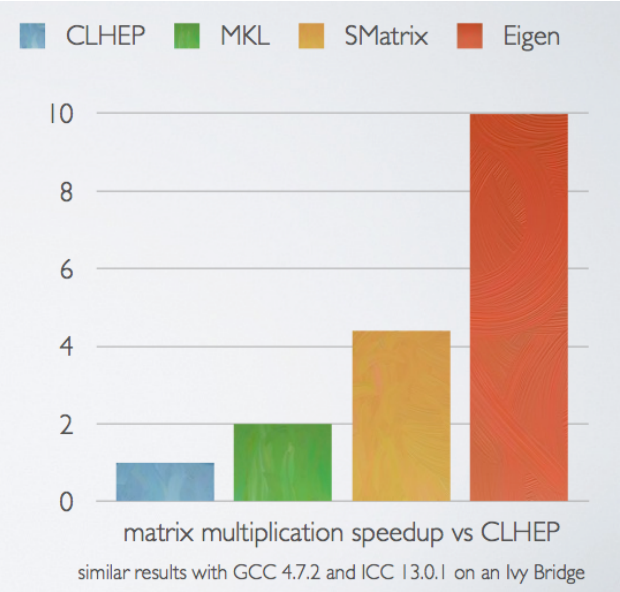
- Change from Fortran77 to C++
  - Code a lot more readable now
  - Reduced function call depth
- Adding field value cache
  - Greatly affects performance as particles are traced along their trajectory
- Unit Conversion Minimization
  - Affects accuracy and performance

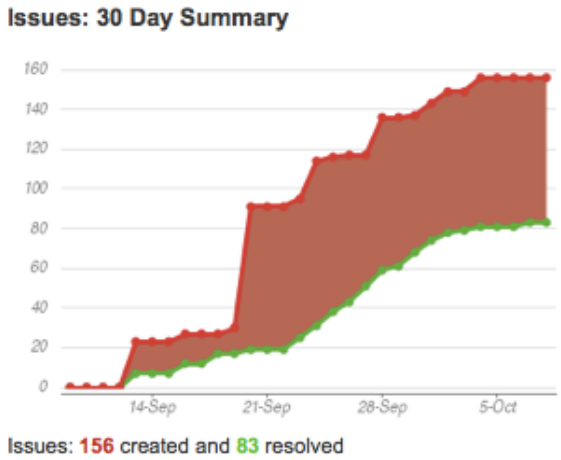


- Make code autovectorizable and applying intrinsics
- Speed-up of 20% (reco) up to 60 % (single particle simulation)

Efforts: Library Change

- Change from CLHEP to Eigen
  - Huge software migration
    - O(1000) packages affected
  - Eigen library functions can vectorize if compiled accordingly
- Exchanging the allocator
- Exchanging GNU libm
  - Under investigation: VDT, libimf





Eigen library extensions

- Extending Eigen by inline functions for a more CLHEP like API facilitating migration
  - Macro allows Eigen extension without inheritance
- Adding optimized functions for symmetric matrices
  - Eigen doesn't have symmetric matrices or optimized operations
  - Overhead free matrix data access as double*
  - E.g. Gauss-Jordan inversion implemented

Eigen library integration

- Creation of Amg:: namespace for Eigen
- Access unification to all Eigen types
  - Facilitates future changes
  - Simplifies access (before there were 3 different wrappers for e.g. CLHEP::Vector3D)
  - Direct access to Eigen discouraged
- All but equal interface allows the same tests running on both Amg and CLHEP
  - Any other library, e.g. Root's Smatrix can be wrapped in the same way to allow easy comparison

Eigen library findings

- Performance findings very promising:
  - Conversion of global to local coordinates 3x faster with Eigen
- Numerical differences to CLHEP
  - Comparison of conversion local -> global -> local shows Eigen to be closer to the original value than CLHEP
- Inlining greatly affects speed
  - Deactivating inlining for Eigen diminishes performance advantages
- Unified access through single interface improves readability and maintainability

Efforts: Tcmalloc and libm

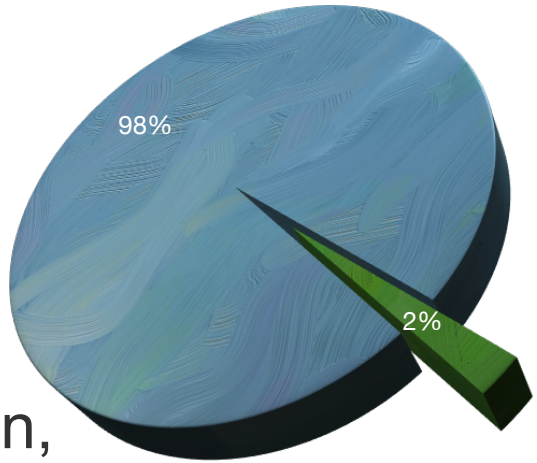
- Tcmalloc version in use is outdated
  - Uses less memory than all newer mallocs but provides unaligned memory
    - Cannot vectorize
- Many newer allocators faster
  - Facebook's jemalloc up to 10% faster
- GNU libm slow compared to other libs
  - Intel's Libimf and CMS' VDT provide up to 10x faster trigonometric functions
- Both libraries can be replaced with minimum effort using LD_PRELOAD

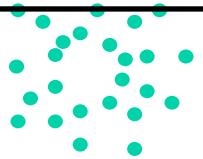
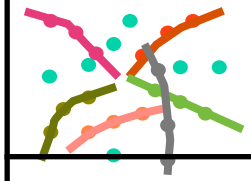
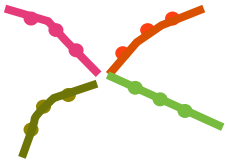
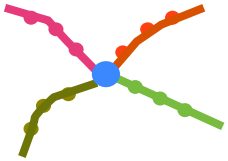
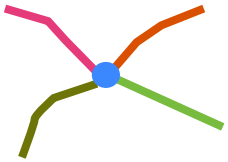
Tcmalloc and libm

- Tcmalloc and libm are both responsible for about 5% of total reconstruction time
- Number of mallocs will decrease after Eigen migration
  - Eigen uses less temporaries than CLHEP which now makes up more than 10% of all mallocs
- Increased speedup expected for vdt when migrating and therefore allowing inlining
- Final decision outstanding, short term measure: LD_PRELOAD for libimf

Efforts: Event Data Model (EDM)

- Initial Tracking EDM was highly polymorph
  - Design before data taking was focused on flexibility rather than CPU performance
  - Experiences of Run 1 now integrated, EDM reworked:
    - Flatter EDM structure
    - Avoidance of dynamic casts
    - Persistency considerations enter transient EDM
- Example:
  - New templated track parameterisation, removed 98 % of C++ code lines



	stage	migration status	ID CPU time	comments
	space point formation	done	-1%	fully validated
	space point seeded track finding	done	-20%	slightly increased track finding efficiency
	ambiguity solving & track fitting	done	- ? %	**biggest relative CPU saving potential,** currently crashing
	vertex reconstruction	ongoing	- ? %	Depends on ongoing changes in analysis EDM
	particle creation	ongoing	- ? %	

Future Efforts Necessary

- Goals require further action
  - Current projects expected to yield at least 20% reconstruction performance improvement
- Other options include
  - Preparing code for autovectorization
  - Manual vectorization of hot-spots w. intrinsics
  - Creating data pools to avoid frequent malloc/free calls
  - Parallelizing algorithms to run on multiple cores

Summary

- At least two fold performance improve required
- Optimization projects ongoing
  - Replacing various libraries (requiring different effort)
  - Optimizing algorithms
  - Optimizing memory
- Further efforts required to achieve goals
- Many other optimizations possible
  - E.g. parallelization, vectorization, data pools
- Cost-benefit analysis for future projects ongoing