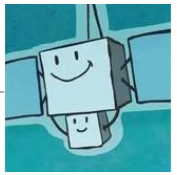
cross kalman

a fit and smooth kalman filter



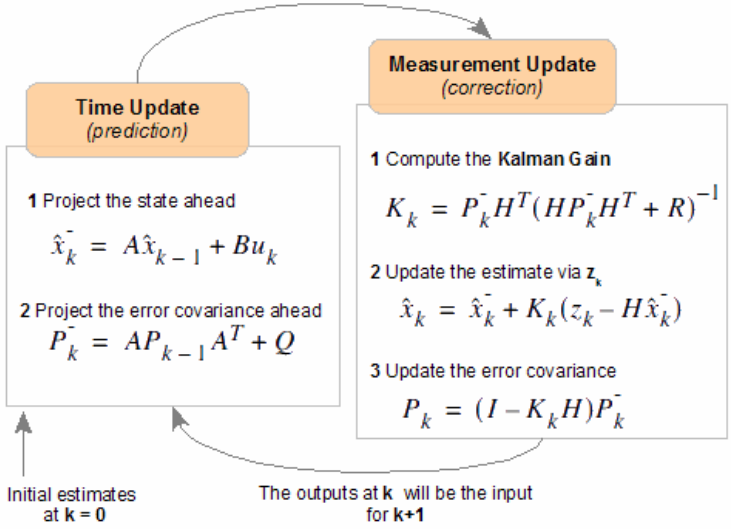
On previous episodes

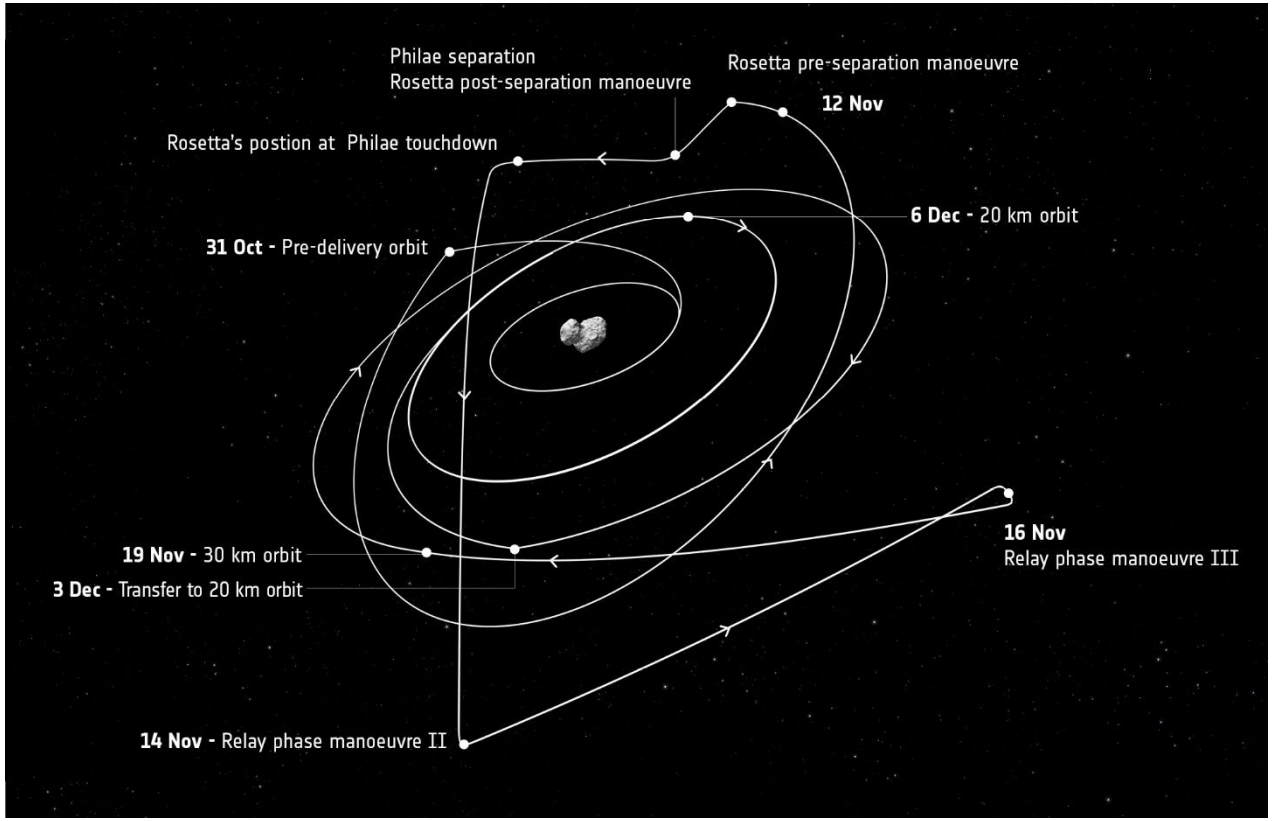
- Cross-architecture Kalman Filter first results, LHCb 7th Computing Workshop
- LHCb Kalman Filter cross architectures studies, CHEP 2016
- Kalman Filter update, T&A last week

On this episode

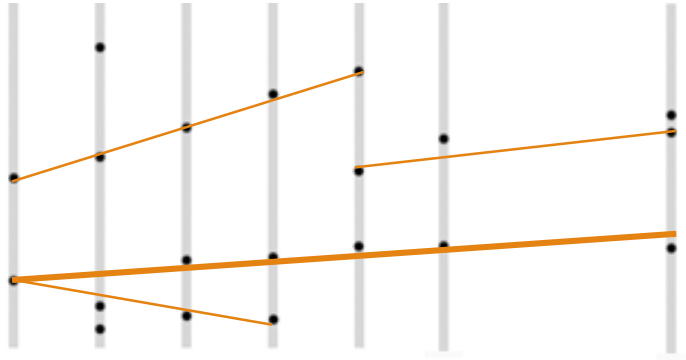
- Implementation details
- First physics

What is a Kalman Filter?





Kalman at LHCb



Forward → p_0u_0 p_1u_1 p_2u_2 … $p_{n-1}u_{n-1}$ — m at a time (at best)

Backward ← $bp_{n-1}bu_{n-1}$ … bp_0bu_0 — m at a time (at best)

Smoother ↓ s_0 s_1 s_2 … s_{n-1} — $\sum_{i=1}^{m} n_i$ at a time

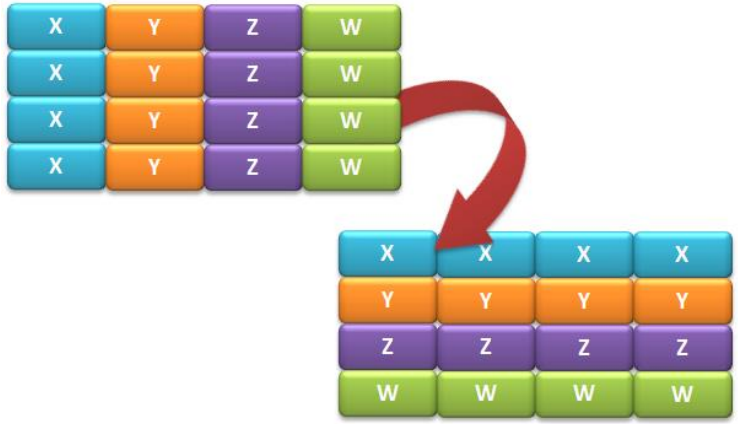
until converges + # for outlier removal

Diving into the details

- Event model targeting locality
  - AOSOA across processing tracks
  - Data alignment set according to precision and vector width
- Efficient scheduler
  - Lightweight static scheduler fills up exposed vector units efficiently
- Smooth algorithmy
  - Cross architecture
  - Fully vectorised
  - Configurable *precision* and *vector width* at compile time
  - Find the best price / performance / watt balance

Data layout is essential

```
struct State {
  PRECISION* m_basePointer = 0x0;
  TrackVector m_predictedState;
  TrackVector m_updatedState;
  TrackSymMatrix m_predictedCovariance;
  TrackSymMatrix m_updatedCovariance;
  PRECISION* m_chi2;

  ...
}
```



- AOSOA structured backend ensures data locality and alignment
- AOS view preserves usability with previous interface
- There is a performance benefit of reading and writing on the same location
  - Using the same memory location for the Predicted and Updated state

Scheduler to the rescue

Good scheduling is essential to benefit from vectorisation

- Keeps data movements to a minimum
- Simplifies the flow of the application

We opted for a static scheduler

- Preserves data structure ordering for posterior Smoother
  - Ie. Makes smoother **50 % faster**
- Low overhead
- Same structure reusage on backward

```
it    in   out  act  vector (#track-#node)
[...]
#540: 0000 0001 1111 { 112-9 80-11 81-11 113-10 }
#541: 0001 1110 1111 { 112-10 80-12 81-12 79-3 }
#542: 1110 0000 1111 { 107-2 109-1 108-2 79-4 }
#543: 0000 0000 1111 { 107-3 109-2 108-3 79-5 }
#544: 0000 0000 1111 { 107-4 109-3 108-4 79-6 }
#545: 0000 0000 1111 { 107-5 109-4 108-5 79-7 }
#546: 0000 0000 1111 { 107-6 109-5 108-6 79-8 }
#547: 0000 0000 1111 { 107-7 109-6 108-7 79-9 }
#548: 0000 0000 1111 { 107-8 109-7 108-8 79-10 }
#549: 0000 0000 1111 { 107-9 109-8 108-9 79-11 }
#550: 0000 0001 1111 { 107-10 109-9 108-10 79-12 }
#551: 0001 1110 1111 { 107-11 109-10 108-11 111-1 }
#552: 1110 0000 1111 { 114-1 78-3 77-3 111-2 }
#553: 0000 0000 1111 { 114-2 78-4 77-4 111-3 }
#554: 0000 0000 1111 { 114-3 78-5 77-5 111-4 }
#555: 0000 0000 1111 { 114-4 78-6 77-6 111-5 }
#556: 0000 0000 1111 { 114-5 78-7 77-7 111-6 }
#557: 0000 0000 1111 { 114-6 78-8 77-8 111-7 }
#558: 0000 0000 1111 { 114-7 78-9 77-9 111-8 }
#559: 0000 0000 1111 { 114-8 78-10 77-10 111-9 }
[...]
```

cross kalman scheduler

Clear flow, complex backend

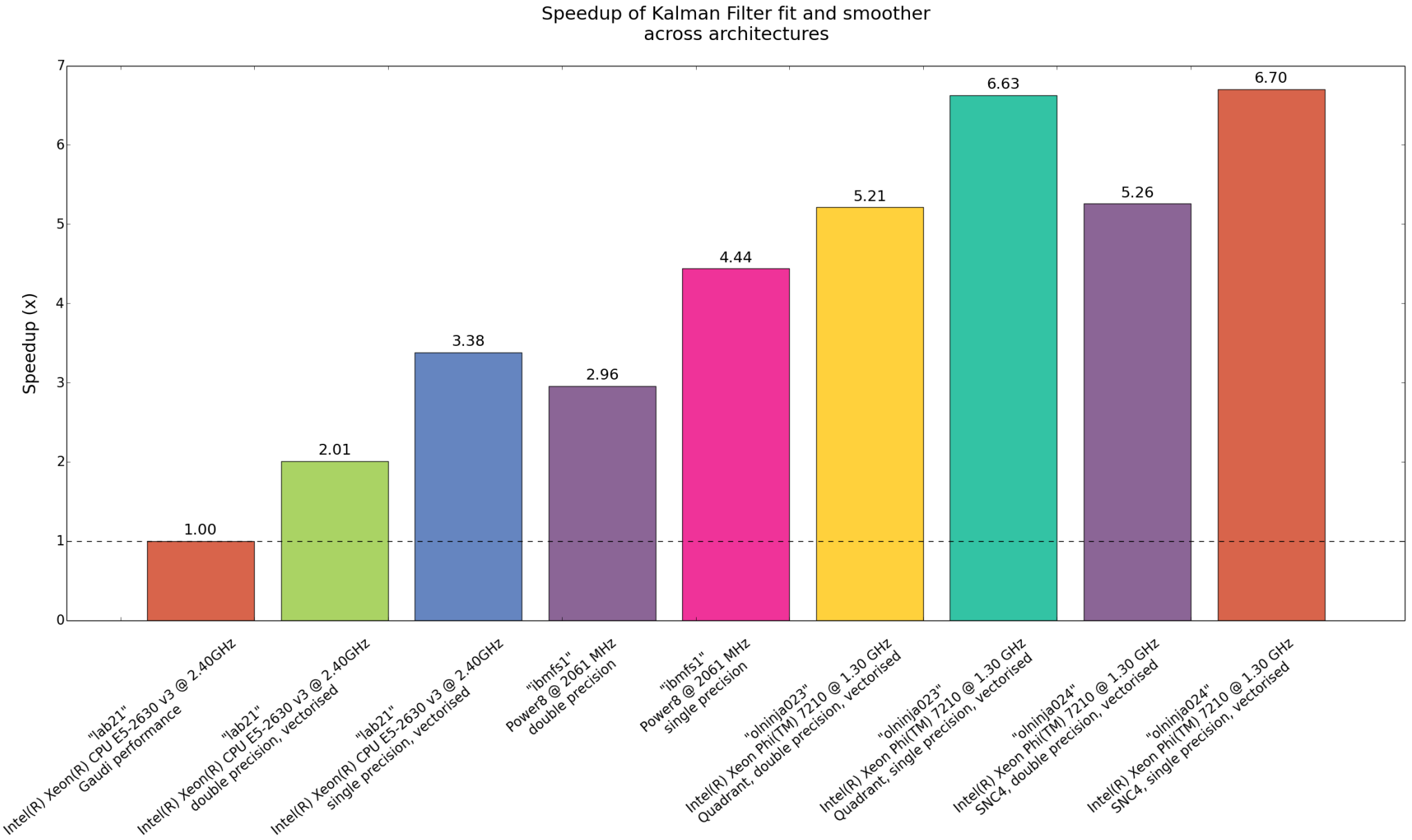
```
std::for_each (scheduler_main.begin(), scheduler_main.end(), [&]
(decltype(scheduler_main[0])& s) {
  const auto& in  = s.in;
  const auto& out = s.out;
  const auto& action = s.action;
  auto& pool = s.pool;

  // Feed the data we need in our vector
  const auto& lastVector = memvecforward.getLastVector();
  if (in.any()) {
    for (unsigned i=0; i<VECTOR_WIDTH; ++i) {
      if (in[i]) {
        VectorFit::States memslot (lastVector + i);
        memslot.m_updatedState.copy(pool[i].prevnode->getState<VectorFit::Op::Forward,
VectorFit::Op::Update, VectorFit::Op::StateVector>());
        memslot.m_updatedCovariance.copy(pool[i].prevnode-
>getState<VectorFit::Op::Forward, VectorFit::Op::Update, VectorFit::Op::Covariance>());
      }
    }
  }

  // predict and update
  VectorFit::States last (lastVector);
  VectorFit::States current (vector);

  VectorFit::fit_vec<VectorFit::Op::Forward>::op<VECTOR_WIDTH> (
    pool,
    last.m_updatedState.m_basePointer,
    last.m_updatedCovariance.m_basePointer,
    current.m_updatedState.m_basePointer,
    current.m_updatedCovariance.m_basePointer,
    current.m_chi2
  );
```

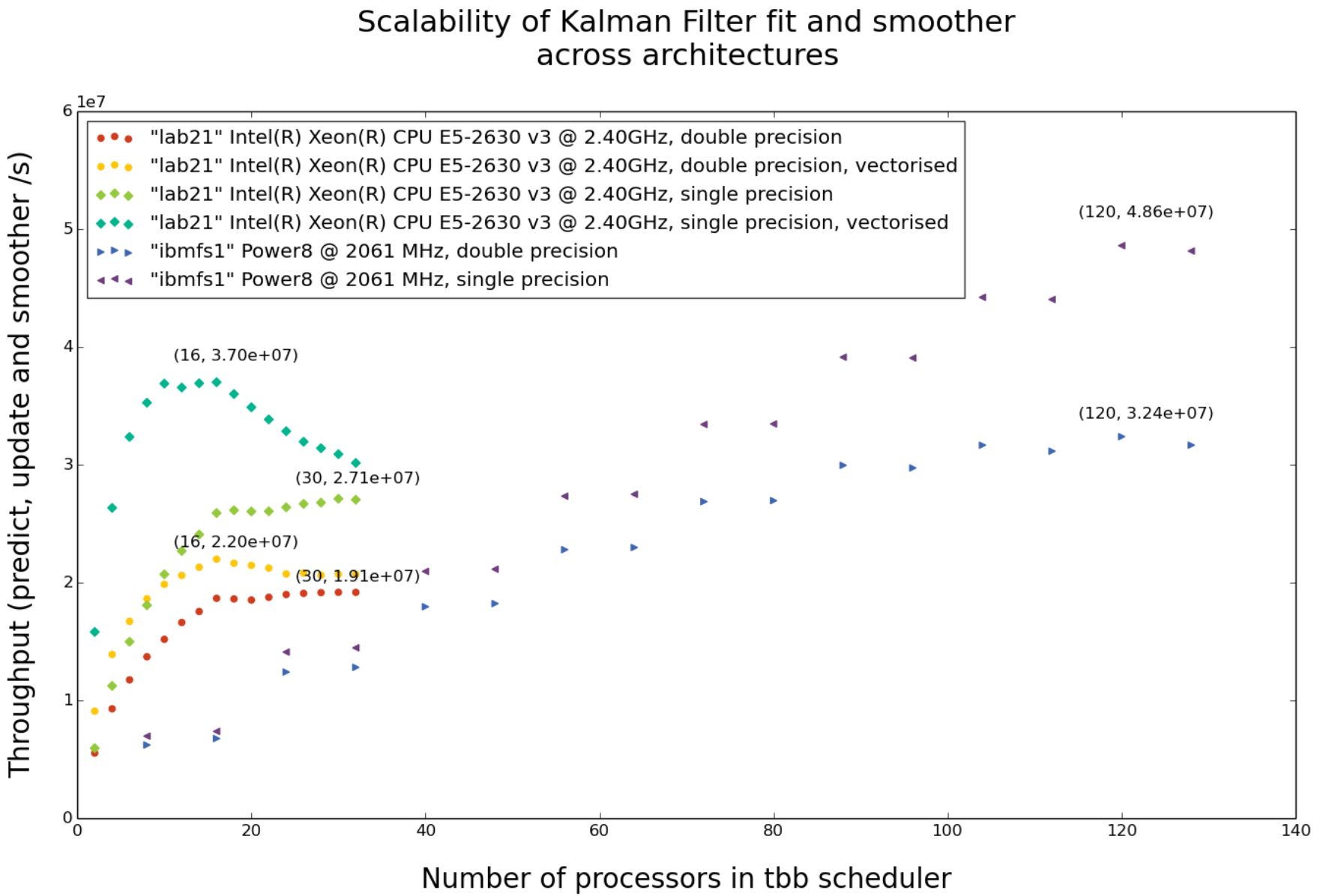
```
/**
 * Transposition in AVX of 4 PRECISION vectors
 */
#define ___MM256_TRANSPOSE4_PD(in0, in1, in2, in3, out0, out1,
out2, out3, __in0, __in1, __in2, __in3, __out0, __out1, __out2,
__out3, __tmp0, __tmp1, __tmp2, __tmp3) \
  do { \
    __m256d __in0 = (in0), __in1 = (in1), __in2 = (in2), __in3
= (in3); \
    __m256d __tmp0, __tmp1, __tmp2, __tmp3; \
    __m256d __out0, __out1, __out2, __out3; \
    __tmp0 = _mm256_shuffle_pd(__in0, __in1, 0x0); \
    __tmp1 = _mm256_shuffle_pd(__in0, __in1, 0xf); \
    __tmp2 = _mm256_shuffle_pd(__in2, __in3, 0x0); \
    __tmp3 = _mm256_shuffle_pd(__in2, __in3, 0xf); \
    __out0 = _mm256_permute2f128_pd(__tmp0, __tmp2, 0x20); \
    __out1 = _mm256_permute2f128_pd(__tmp1, __tmp3, 0x20); \
    __out2 = _mm256_permute2f128_pd(__tmp0, __tmp2, 0x31); \
    __out3 = _mm256_permute2f128_pd(__tmp1, __tmp3, 0x31); \
    (out0) = __out0, (out1) = __out1, (out2) = __out2, (out3) =
__out3; \
  } while (0)
```

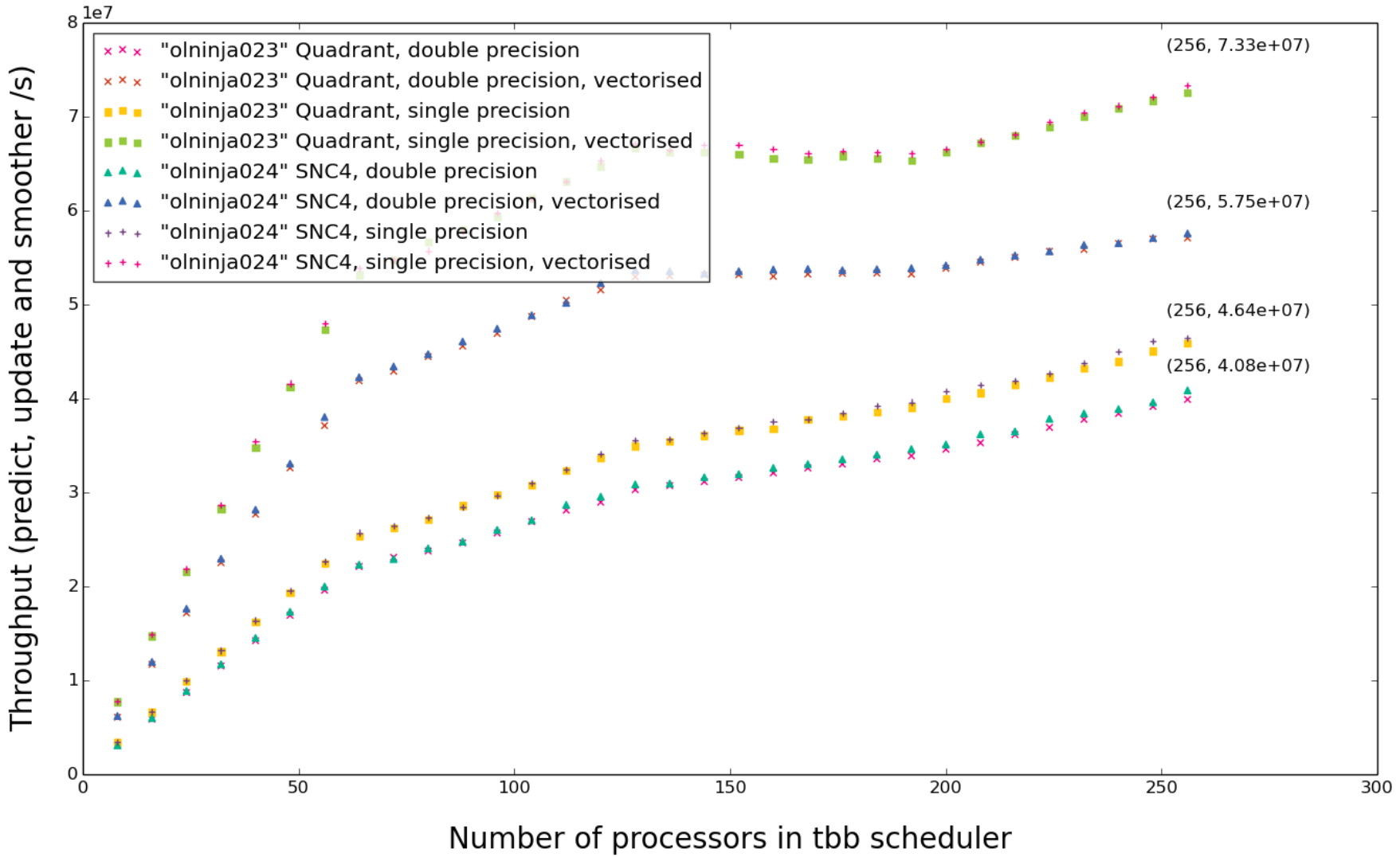
Results

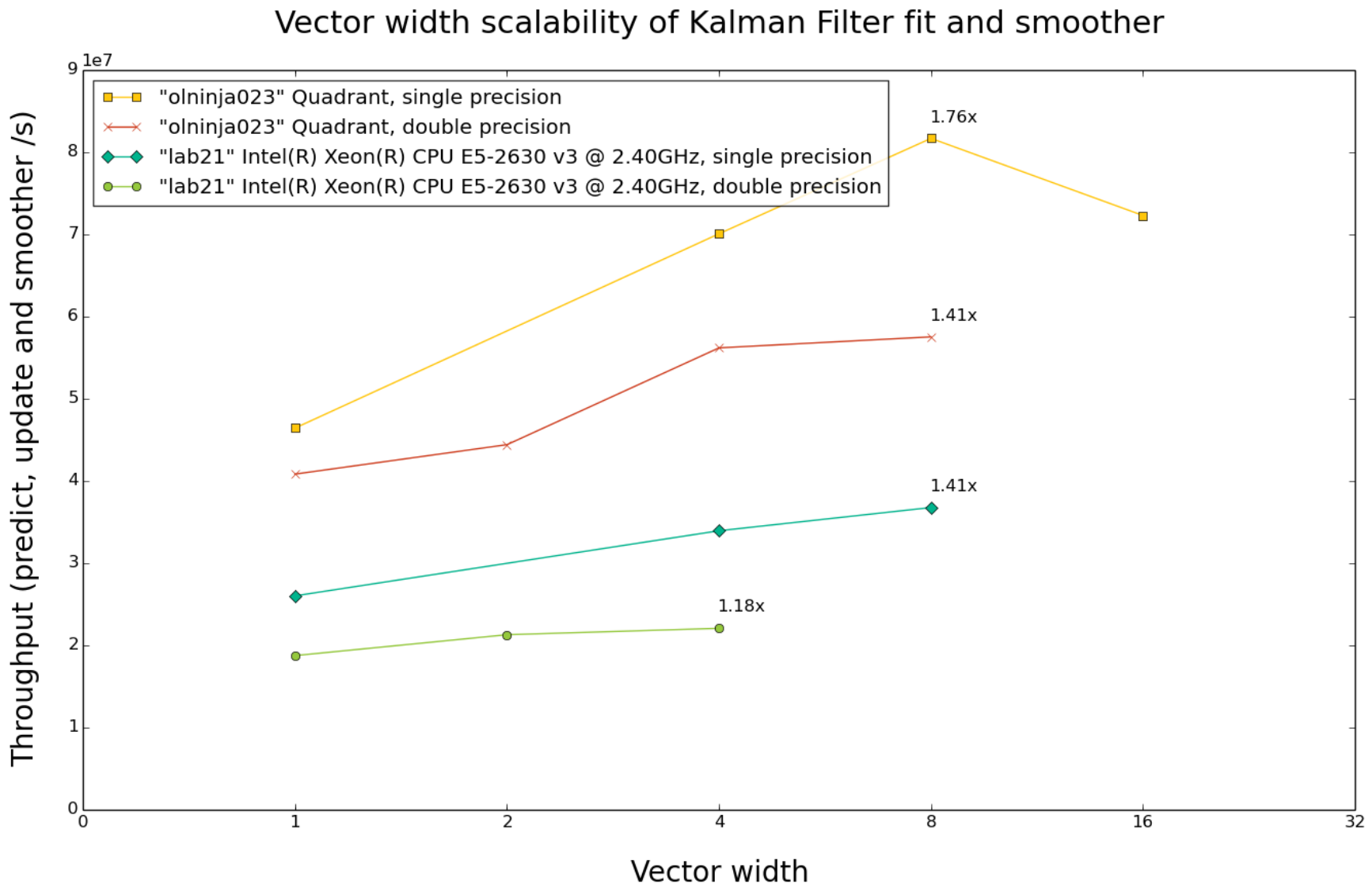
Running conditions

- Compiled with gcc 6.2.0
- Ran 500.000 experiments (events), each event is a *task*
- Used Montecarlo events from the LHCb Upgrade, checked against Gaudi implementation results
- Spawned one process per numa domain, with as many TBB threads as cores in domain and pinned to its memory

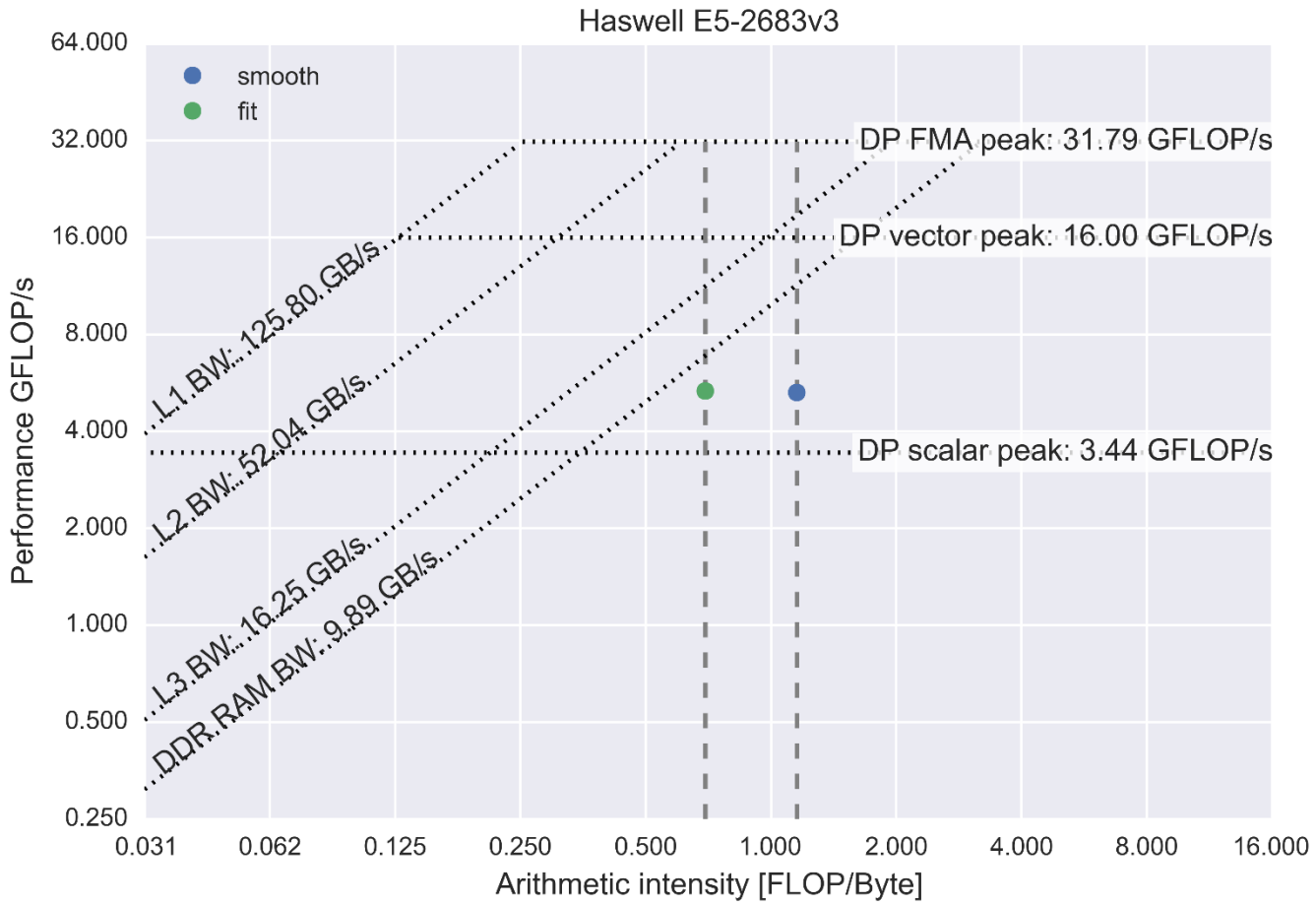

Speedup of Kalman Filter fit and smoother
across architectures
Speedup (x)
0
1
2
3
4
5
6
7
1.00
2.01
3.38
2.96
4.44
5.21
6.63
5.26
6.70
"lab21"
Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz
Gaudi performance
"lab21"
Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz
double precision, vectorised
"lab21"
Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz
single precision, vectorised
"ibmfs1"
Power8 @ 2061 MHz
double precision
"ibmfs1"
Power8 @ 2061 MHz
single precision
"olninja023"
Intel(R) Xeon Phi(TM) 7210 @ 1.30 GHz
Quadrant, double precision, vectorised
"olninja023"
Intel(R) Xeon Phi(TM) 7210 @ 1.30 GHz
Quadrant, single precision, vectorised
"olninja024"
Intel(R) Xeon Phi(TM) 7210 @ 1.30 GHz
SNC4, double precision, vectorised
"olninja024"
Intel(R) Xeon Phi(TM) 7210 @ 1.30 GHz
SNC4, single precision, vectorised


Scalability of Kalman Filter fit and smoother across architectures
1e7
"lab21" Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz, double precision
"lab21" Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz, double precision, vectorised
"lab21" Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz, single precision
"lab21" Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz, single precision, vectorised
"ibmfs1" Power8 @ 2061 MHz, double precision
"ibmfs1" Power8 @ 2061 MHz, single precision
(120, 4.86e+07)
(16, 3.70e+07)
(120, 3.24e+07)
(30, 2.71e+07)
(16, 2.20e+07)
(30, 1.91e+07)
Throughput (predict, update and smoother /s)
Number of processors in tbb scheduler
0
1
2
3
4
5
6
0
20
40
60
80
100
120
140


Scalability of Kalman Filter fit and smoother
on Intel(R) Xeon Phi(TM) 7210 @ 1.30 GHz
Throughput (predict, update and smoother /s)
1e7
Number of processors in tbb scheduler
"olninja023" Quadrant, double precision
"olninja023" Quadrant, double precision, vectorised
"olninja023" Quadrant, single precision
"olninja023" Quadrant, single precision, vectorised
"olninja024" SNC4, double precision
"olninja024" SNC4, double precision, vectorised
"olninja024" SNC4, single precision
"olninja024" SNC4, single precision, vectorised
(256, 7.33e+07)
(256, 5.75e+07)
(256, 4.64e+07)
(256, 4.08e+07)
0
1
2
3
4
5
6
7
8
0
50
100
150
200
250
300



- *Speedup* shown is versus the respective scalar version



- **Roofline model** of our main processes
  - Roofline: An Insightful Visual Performance Model for Multicore Architectures
- We have improved our performance by combining *predict* and *update* (into *fit*)
- The requirements of the Kalman Filter, in terms of memory operations, caps the attainable performance

A word about precision

- Single precision is worth exploring

  - It shows from 20 to 70 % performance gain, depending on architecture
  - Memory required is nominally half
  - Some architectures naturally favor single precision (ie. GPUs)

  - Mathematical error estimation hard to get
  - Cholesky decomposition can fail due to numerical imprecisions
    - It does indeed fail for about 1 % of the nodes in our tests

- Still need to check back in Gaudi what is the impact of enabling single precision
- Mixed precision?
- Configurable precision upon track properties?

Physics results

Work in progress – this is not the final product!

Fit is identical on double precision

- Verified both as a feature of cross_kalman, as well as within the framework
- Therefore, divergences in results come from FSM rewrite, as well as parallel track processing

```
Check results succeeded!
All of the experiments gave the expected results

Fit timers mean: 2.32358e-07 sum: 4.64716e-07 min: 1.97111e-07 max: 2.67605e-07 stddev: 3.52468e-08 rawsum: 0.00317773
Smoother timers mean: 1.51774e-08 sum: 3.03548e-08 min: 1.39819e-08 max: 1.63729e-08 stddev: 1.19553e-09 rawsum: 0.000103783
Overhead rawsum: 3.96e-07
Total time: 0.00328072

tbb default_num_threads reports execution with 32 threads
Throughput per processor, estimated total throughput:
 Fits and smoother combined: 2.0843e+06 / s, 6.66975e+07 / s
```

Best efficiencies

```
[dcampora@lab21 ~/projects/gaudi/kalmanfit/cross_kalman/output]$ grep "PrChecker.Best " *
0_vanilla:PrChecker.Best      INFO **** Best                          22926 tracks including         2373 ghosts [10.4 %], Event average   8.3 % ****
0_vanilla:PrChecker.Best      INFO                          long :    5855 from     5910 [ 99.1 %]    339 clones [ 5.5 %], purity: 99.35 %, hitEff: 90.54 %
0_vanilla:PrChecker.Best      INFO                    long>5GeV :    3771 from     3795 [ 99.4 %]    184 clones [ 4.7 %], purity: 99.37 %, hitEff: 92.59 %
0_vanilla:PrChecker.Best      INFO                long_strange :     309 from      311 [ 99.4 %]     18 clones [ 5.5 %], purity: 99.11 %, hitEff: 88.44 %
0_vanilla:PrChecker.Best      INFO          long_strange>5GeV :     155 from      155 [100.0 %]      8 clones [ 4.9 %], purity: 99.51 %, hitEff: 92.31 %
0_vanilla:PrChecker.Best      INFO                 long_fromB :      10 from       11 [ 90.9 %]      2 clones [16.7 %], purity: 99.56 %, hitEff: 77.10 %
0_vanilla:PrChecker.Best      INFO           long_fromB>5GeV :       3 from        3 [100.0 %]      1 clones [25.0 %], purity:100.00 %, hitEff: 65.71 %

1_doublep:PrChecker.Best      INFO **** Best                          23330 tracks including         2408 ghosts [10.3 %], Event average   8.4 % ****
1_doublep:PrChecker.Best      INFO                          long :    5865 from     5910 [ 99.2 %]    389 clones [ 6.2 %], purity: 99.44 %, hitEff: 89.78 %
1_doublep:PrChecker.Best      INFO                    long>5GeV :    3775 from     3795 [ 99.5 %]    215 clones [ 5.4 %], purity: 99.46 %, hitEff: 91.91 %
1_doublep:PrChecker.Best      INFO                long_strange :     309 from      311 [ 99.4 %]     24 clones [ 7.2 %], purity: 99.27 %, hitEff: 86.88 %
1_doublep:PrChecker.Best      INFO          long_strange>5GeV :     155 from      155 [100.0 %]     11 clones [ 6.6 %], purity: 99.26 %, hitEff: 90.45 %
1_doublep:PrChecker.Best      INFO                 long_fromB :      10 from       11 [ 90.9 %]      2 clones [16.7 %], purity: 99.56 %, hitEff: 75.44 %
1_doublep:PrChecker.Best      INFO           long_fromB>5GeV :       3 from        3 [100.0 %]      1 clones [25.0 %], purity:100.00 %, hitEff: 60.71 %

2_singlep:PrChecker.Best      INFO **** Best                          22675 tracks including         1379 ghosts [ 6.1 %], Event average   4.8 % ****
2_singlep:PrChecker.Best      INFO                          long :    5886 from     5910 [ 99.6 %]    478 clones [ 7.5 %], purity: 99.25 %, hitEff: 87.49 %
2_singlep:PrChecker.Best      INFO                    long>5GeV :    3786 from     3795 [ 99.8 %]    255 clones [ 6.3 %], purity: 99.22 %, hitEff: 90.10 %
2_singlep:PrChecker.Best      INFO                long_strange :     310 from      311 [ 99.7 %]     28 clones [ 8.3 %], purity: 99.30 %, hitEff: 84.33 %
2_singlep:PrChecker.Best      INFO          long_strange>5GeV :     155 from      155 [100.0 %]     13 clones [ 7.7 %], purity: 99.27 %, hitEff: 88.16 %
2_singlep:PrChecker.Best      INFO                 long_fromB :      10 from       11 [ 90.9 %]      2 clones [16.7 %], purity: 99.56 %, hitEff: 72.18 %
2_singlep:PrChecker.Best      INFO           long_fromB>5GeV :       3 from        3 [100.0 %]      1 clones [25.0 %], purity:100.00 %, hitEff: 50.95 %
```

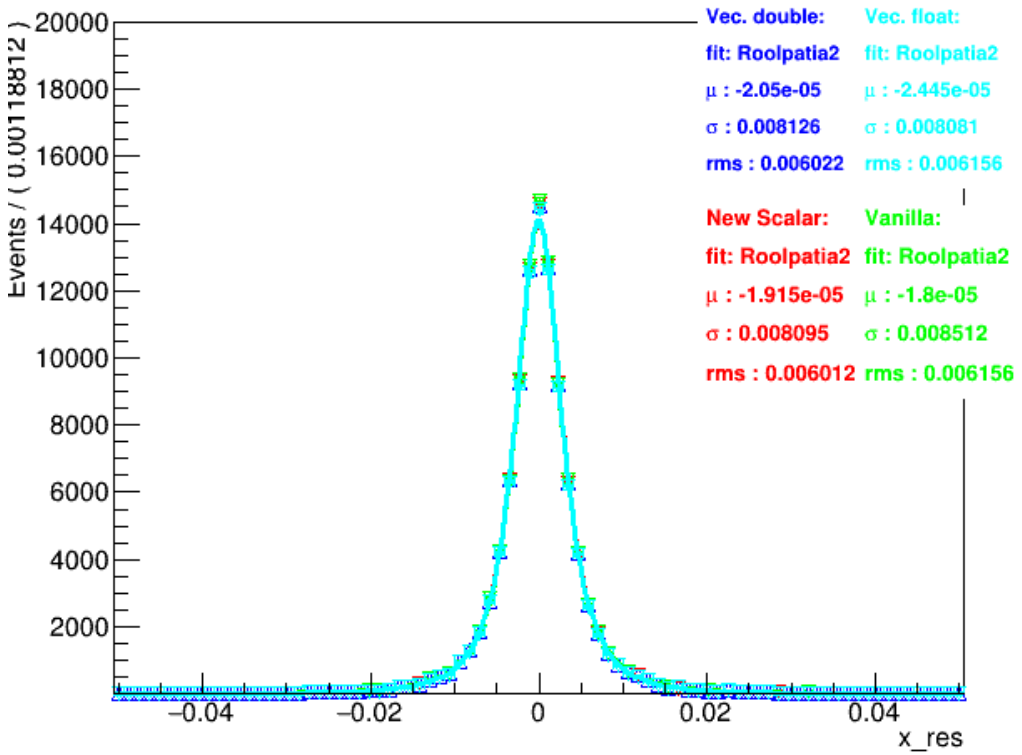
BestLong efficiencies

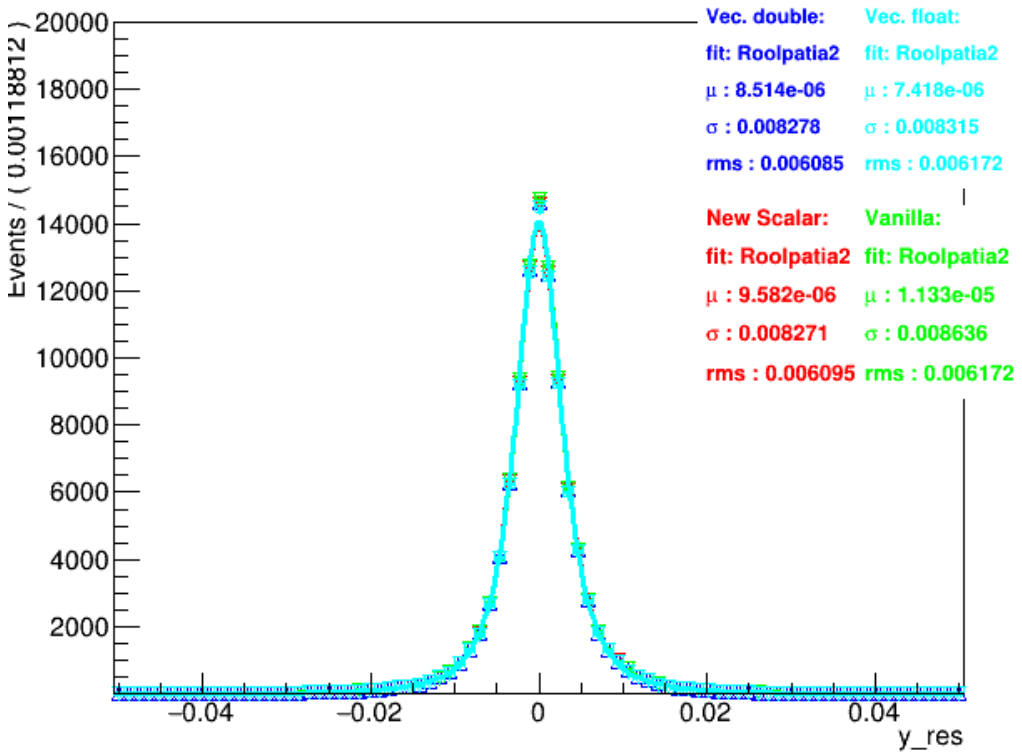
```
[dcampora@lab21 ~/projects/gaudi/kalmanfit/cross_kalman/output]$ grep PrChecker.BestLong *
0_vanilla:PrChecker.BestLong   INFO **** BestLong                          6937 tracks including          1020 ghosts [14.7 %], Event average  11.2 % ****
0_vanilla:PrChecker.BestLong   INFO                              long :    5411 from     5910 [ 91.6 %]     89 clones [ 1.6 %], purity: 99.51 %, hitEff: 96.29 %
0_vanilla:PrChecker.BestLong   INFO                        long>5GeV :    3572 from     3795 [ 94.1 %]     58 clones [ 1.6 %], purity: 99.58 %, hitEff: 96.67 %
0_vanilla:PrChecker.BestLong   INFO                     long_strange :     275 from      311 [ 88.4 %]      2 clones [ 0.7 %], purity: 99.34 %, hitEff: 96.49 %
0_vanilla:PrChecker.BestLong   INFO               long_strange>5GeV :     144 from      155 [ 92.9 %]      2 clones [ 1.4 %], purity: 99.46 %, hitEff: 96.76 %
0_vanilla:PrChecker.BestLong   INFO                     long_fromB :       7 from       11 [ 63.6 %]      0 clones [ 0.0 %], purity: 99.25 %, hitEff: 95.65 %
0_vanilla:PrChecker.BestLong   INFO               long_fromB>5GeV :       1 from        3 [ 33.3 %]      0 clones [ 0.0 %], purity:100.00 %, hitEff:100.00 %

1_doublep:PrChecker.BestLong   INFO **** BestLong                          6812 tracks including           961 ghosts [14.1 %], Event average  10.9 % ****
1_doublep:PrChecker.BestLong   INFO                              long :    5365 from     5910 [ 90.8 %]     89 clones [ 1.6 %], purity: 99.56 %, hitEff: 96.33 %
1_doublep:PrChecker.BestLong   INFO                        long>5GeV :    3546 from     3795 [ 93.4 %]     58 clones [ 1.6 %], purity: 99.61 %, hitEff: 96.70 %
1_doublep:PrChecker.BestLong   INFO                     long_strange :     267 from      311 [ 85.9 %]      2 clones [ 0.7 %], purity: 99.44 %, hitEff: 96.45 %
1_doublep:PrChecker.BestLong   INFO               long_strange>5GeV :     140 from      155 [ 90.3 %]      2 clones [ 1.4 %], purity: 99.51 %, hitEff: 96.65 %
1_doublep:PrChecker.BestLong   INFO                     long_fromB :       7 from       11 [ 63.6 %]      0 clones [ 0.0 %], purity: 99.25 %, hitEff: 95.65 %
1_doublep:PrChecker.BestLong   INFO               long_fromB>5GeV :       1 from        3 [ 33.3 %]      0 clones [ 0.0 %], purity:100.00 %, hitEff:100.00 %

2_singlep:PrChecker.BestLong   INFO **** BestLong                          6424 tracks including           663 ghosts [10.3 %], Event average   7.9 % ****
2_singlep:PrChecker.BestLong   INFO                              long :    5289 from     5910 [ 89.5 %]     80 clones [ 1.5 %], purity: 99.55 %, hitEff: 96.50 %
2_singlep:PrChecker.BestLong   INFO                        long>5GeV :    3511 from     3795 [ 92.5 %]     53 clones [ 1.5 %], purity: 99.60 %, hitEff: 96.85 %
2_singlep:PrChecker.BestLong   INFO                     long_strange :     263 from      311 [ 84.6 %]      2 clones [ 0.8 %], purity: 99.41 %, hitEff: 96.46 %
2_singlep:PrChecker.BestLong   INFO               long_strange>5GeV :     138 from      155 [ 89.0 %]      2 clones [ 1.4 %], purity: 99.51 %, hitEff: 96.64 %
2_singlep:PrChecker.BestLong   INFO                     long_fromB :       7 from       11 [ 63.6 %]      0 clones [ 0.0 %], purity: 99.25 %, hitEff: 95.65 %
2_singlep:PrChecker.BestLong   INFO               long_fromB>5GeV :       1 from        3 [ 33.3 %]      0 clones [ 0.0 %], purity:100.00 %, hitEff:100.00 %
```

BestDown efficiencies

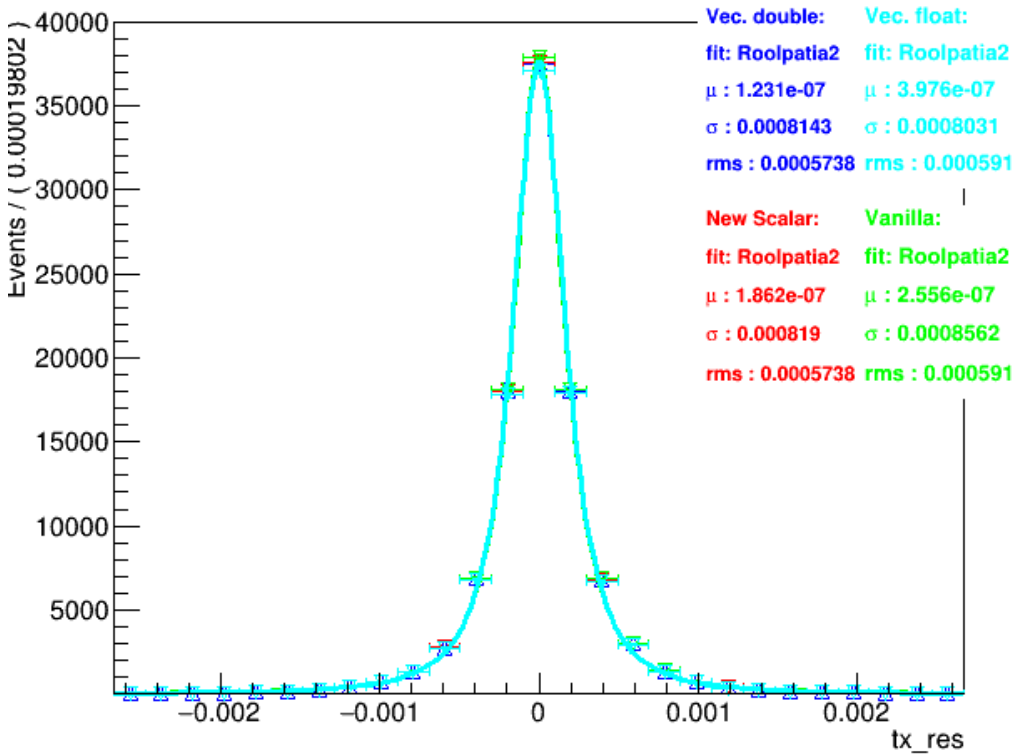
```
[dcampora@lab21 ~/projects/gaudi/kalmanfit/cross_kalman/output]$ grep PrChecker.BestDown * --color=always | grep -v fromD | grep -v fromB
0_vanilla:PrChecker.BestDown   INFO **** BestDown                          1226 tracks including          421 ghosts [34.3 %], Event average  32.2 % ****
0_vanilla:PrChecker.BestDown   INFO                              UT+T :     512 from     6568 [  7.8 %]      0 clones [ 0.0 %], purity: 99.76 %, hitEff: 96.84 %
0_vanilla:PrChecker.BestDown   INFO                          UT+T>5GeV :     270 from     4035 [  6.7 %]      0 clones [ 0.0 %], purity: 99.78 %, hitEff: 97.23 %
0_vanilla:PrChecker.BestDown   INFO                       UT+T_strange :     198 from      583 [ 34.0 %]      0 clones [ 0.0 %], purity: 99.68 %, hitEff: 96.79 %
0_vanilla:PrChecker.BestDown   INFO                  UT+T_strange>5GeV :     123 from      310 [ 39.7 %]      0 clones [ 0.0 %], purity: 99.72 %, hitEff: 96.77 %
0_vanilla:PrChecker.BestDown   INFO               noVelo+UT+T_strange :     187 from      276 [ 67.8 %]      0 clones [ 0.0 %], purity: 99.66 %, hitEff: 96.86 %
0_vanilla:PrChecker.BestDown   INFO          noVelo+UT+T_strange>5GeV :     116 from      158 [ 73.4 %]      0 clones [ 0.0 %], purity: 99.71 %, hitEff: 96.74 %

1_doublep:PrChecker.BestDown   INFO **** BestDown                          1537 tracks including          678 ghosts [44.1 %], Event average  38.6 % ****
1_doublep:PrChecker.BestDown   INFO                              UT+T :     556 from     6568 [  8.5 %]      0 clones [ 0.0 %], purity: 99.71 %, hitEff: 96.80 %
1_doublep:PrChecker.BestDown   INFO                          UT+T>5GeV :     301 from     4035 [  7.5 %]      0 clones [ 0.0 %], purity: 99.69 %, hitEff: 97.05 %
1_doublep:PrChecker.BestDown   INFO                       UT+T_strange :     208 from      583 [ 35.7 %]      0 clones [ 0.0 %], purity: 99.66 %, hitEff: 96.91 %
1_doublep:PrChecker.BestDown   INFO                  UT+T_strange>5GeV :     130 from      310 [ 41.9 %]      0 clones [ 0.0 %], purity: 99.68 %, hitEff: 96.89 %
1_doublep:PrChecker.BestDown   INFO               noVelo+UT+T_strange :     189 from      276 [ 68.5 %]      0 clones [ 0.0 %], purity: 99.63 %, hitEff: 96.86 %
1_doublep:PrChecker.BestDown   INFO          noVelo+UT+T_strange>5GeV :     119 from      158 [ 75.3 %]      0 clones [ 0.0 %], purity: 99.65 %, hitEff: 96.76 %

2_singlep:PrChecker.BestDown   INFO **** BestDown                            66 tracks including           18 ghosts [27.3 %], Event average  13.3 % ****
2_singlep:PrChecker.BestDown   INFO                              UT+T :      22 from     6568 [  0.3 %]      0 clones [ 0.0 %], purity:100.00 %, hitEff: 97.96 %
2_singlep:PrChecker.BestDown   INFO                          UT+T>5GeV :      13 from     4035 [  0.3 %]      0 clones [ 0.0 %], purity:100.00 %, hitEff: 99.52 %
2_singlep:PrChecker.BestDown   INFO                       UT+T_strange :      10 from      583 [  1.7 %]      0 clones [ 0.0 %], purity:100.00 %, hitEff:100.00 %
2_singlep:PrChecker.BestDown   INFO                  UT+T_strange>5GeV :       8 from      310 [  2.6 %]      0 clones [ 0.0 %], purity:100.00 %, hitEff:100.00 %
2_singlep:PrChecker.BestDown   INFO               noVelo+UT+T_strange :       9 from      276 [  3.3 %]      0 clones [ 0.0 %], purity:100.00 %, hitEff:100.00 %
2_singlep:PrChecker.BestDown   INFO          noVelo+UT+T_strange>5GeV :       8 from      158 [  5.1 %]      0 clones [ 0.0 %], purity:100.00 %, hitEff:100.00 %
```

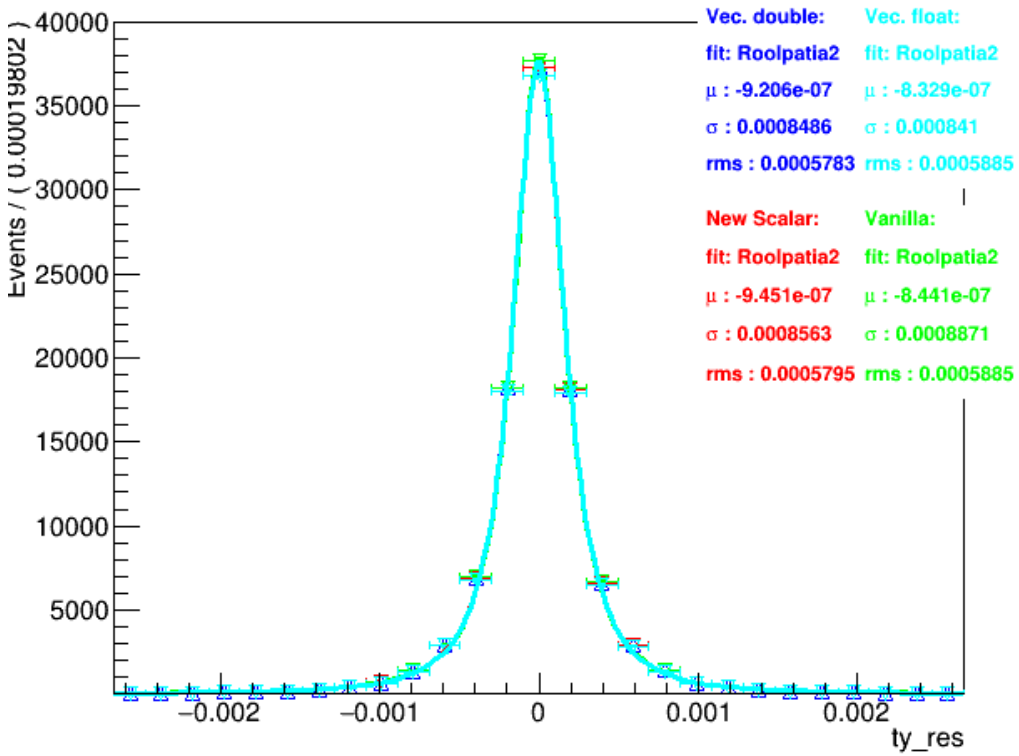
Detail for BestLong





Detail for BestLong





- More at https://lbevent.cern.ch/online/dcampora/kalman_filter/efficiencies_20161117

The *master* plan

- Integrate cross kalman with master
  - Currently integrated with Brunel v51r1, since yesterday!
- Check / fix reconstruction discrepancies
- Check single precision efficiencies

- Rewrite TrackBestTrackCreator from the ground up
  - Not necessarily rewrite all components, but redesign KalmanFitResult and FitNode to adapt to VectorFit
  - Interact with things like the material corrections and transport machinery, simply with an std::for_each, to leave room for future optimisations yet have something relatively fast

Concluding

We have achieved some good things

- Cross architecture, fully vectorised Kalman Filter
- Base 2x with respect to original implementation
- Up to 6.7x on KNL
- Our roofline model shows we are getting the most out of the machine
- Double precision vector fit gives exactly the same result as scalar fit
- Still some divergences in Gaudi, stemming from flow change
- Single precision requires more tweaking

There is more to come

- Integration with present and future Gaudi
- GPU implementation revamp
- Vectorisation for Power8
- Testing + vectorisation for ARM64

Thanks!

Thanks to all of these people, who greatly contributed to these results,

- O Awile
- O Bouizi
- S Harald
- F Lemaitre
- C Potterat
- The CERN Intel Openlab HTC Collaboration
- The RGNC at the University of Sevilla

Resources

- cross kalman repository
- Roofline: An Insightful Visual Performance Model for Multicore Architectures
- https://lbevent.cern.ch/online/dcampora/kalman_filter/efficiencies_20161117

Questions?