Tracking on Xeon Phi Prototype Progess

Software and Computing R&D – Oct. 12, 2015

G.Cerati, M.Tadel, F.Würthwein, A.Yagil (UCSD)
S.Lantz, K.McDermott, D.Riley, P.Wittich (Cornell)
P.Elmer (Princeton)

Introduction

- R&D project exploring parallel computing architectures for HEP data processing
- Tracking is natural starting point as it's the most CPU intensive step
- Xeon-phi is a good candidate because of common features with Xeon processors and because of DOE investments in supercomputers
  - but not only possibility we will consider (also GPUs, etc.)

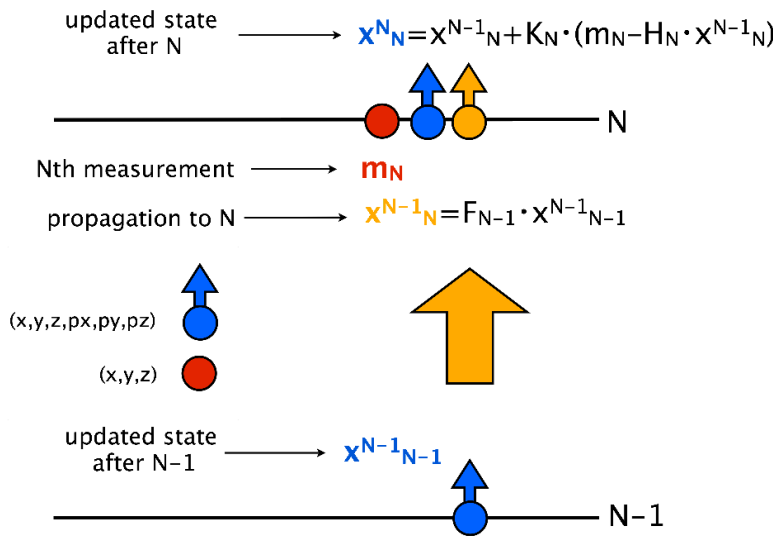
- We started from a semi-realistic standalone setup for simulation and reconstruction with the goal of understanding assets and bottlenecks of the technology
  - simple barrel geometry, reduced material effects, gaussian smeared hits, no jets nor decays
  - Kalman Filter-based combinatorial algorithm, inspired by CMS version

Kalman Filter Tracking

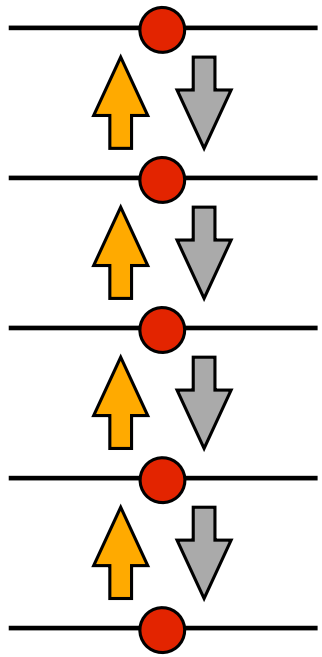
Kalman Filter tracking widely used in HEP, outstanding performance in LHC environment.

It consists in the reiteration of a **basic logic unit** for each tracker layer.

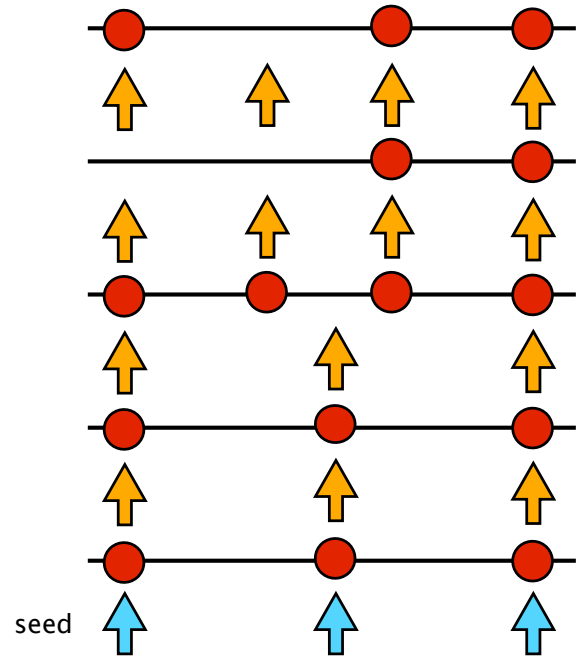
basic logic unit



track fit



track building



The track reconstruction process can be divided in 3 steps: track seeding, building and fitting.

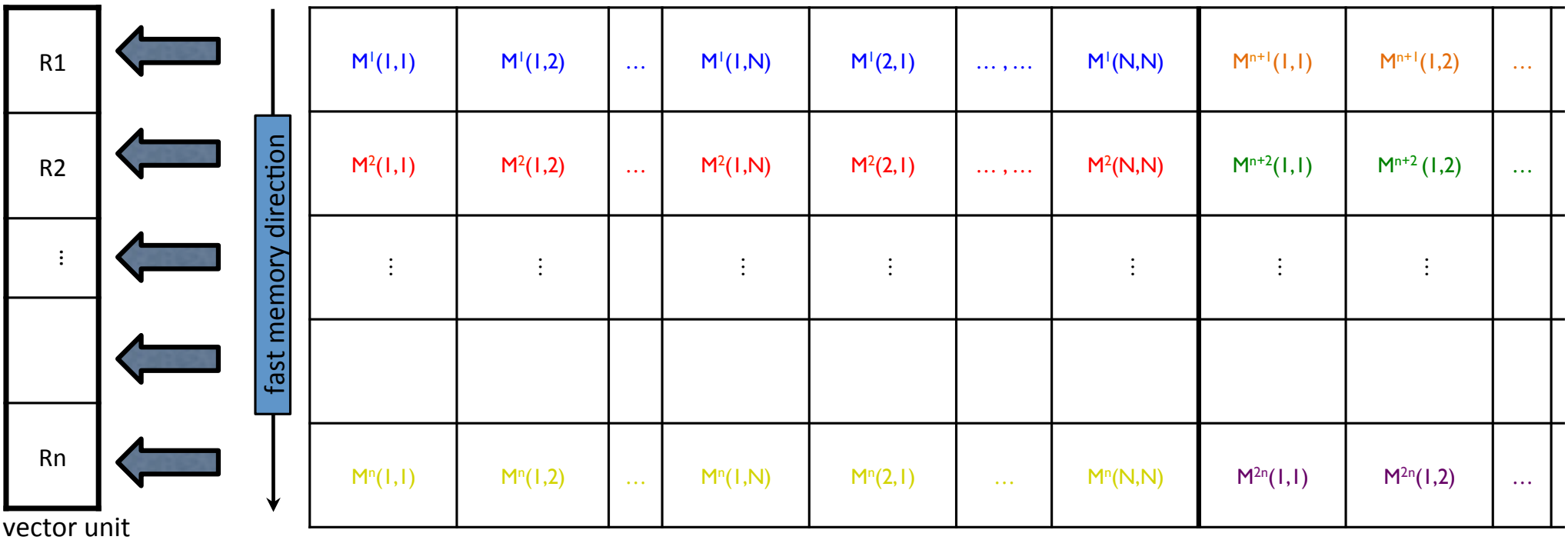
The **track fit** is the bare repetition of the basic unit, ideal as a **starting point**.
Track **building is the most time consuming part** – it involves branching points of variable size, with the simplest version degenerating into the track fit case.
Track **seeding** under development but not included yet, for now seeds are defined using MC info.

Dedicated data structures: Matriplex

Approach successful only if data structures are optimized for the specific architecture.

Kalman filter calculations based on small matrices.
Intel Xeon and Xeon Phi have **vector units** with size 8 and 16 floats respectively.
How can we efficiently exploit them?

Matriplex is a "matrix-major" representation, where vector units elements are separately filled by a different matrix: **n matrices work in sync**.



Matrix size **NxN**, vector unit size **n**

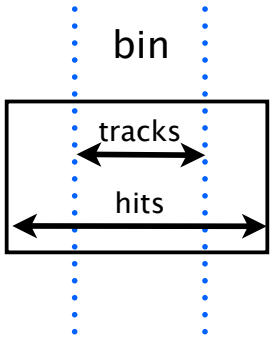
Strategy for Track Building

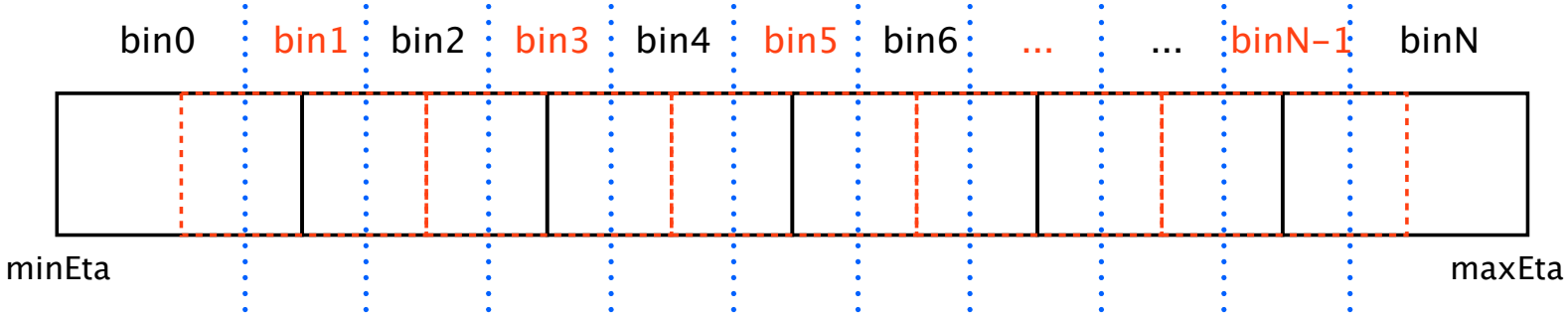
- Same core calculations as in track fitting but adding two big **complications**
  - **Hit set is not defined**: hit on next layer to be chosen between O(10k) hits
  - For >1 compatible hit, combinatorial problem requires **cloning of candidates**

- The two issues can be **factorized** by dividing the development in two stages
  - first develop a simplified algorithm choosing only the best hit on next layer
    - deal with large number of hits, not with cloning – study vectorization in this case first
  - then full implementation with combinatorial expansion
    - parallelization already using this version!

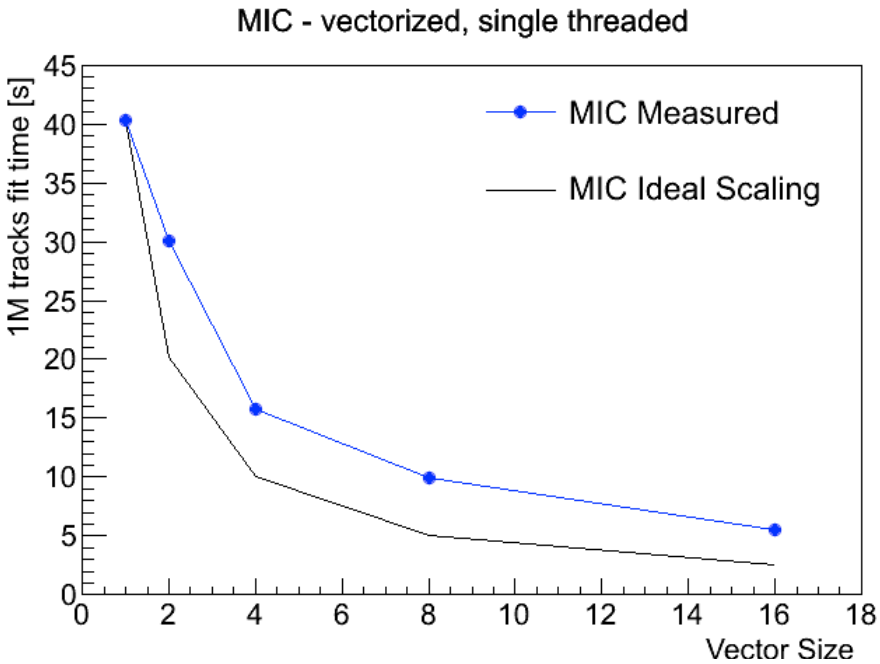
- **Data locality** is the key for reducing the Nhits problem
  - **eta partitions** are **self consistent** (no bending)
    - bins redundant in terms of hits, track candidates never search outside their eta bin
    - natural **boundary for thread definitions**
  - phi partitions give fast lookup of hits in compatibility window

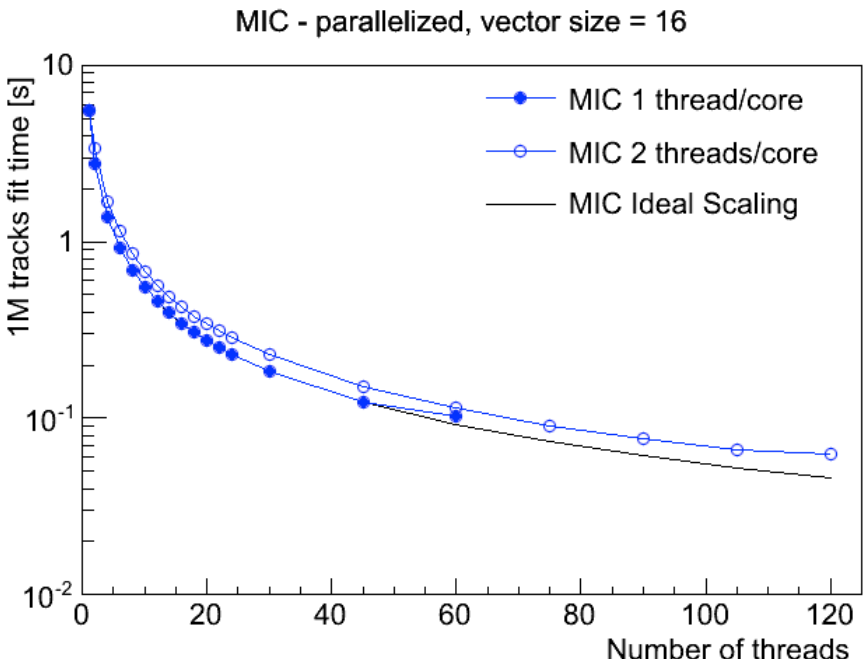
- Seeding is not fully implemented yet, get seeds from sim tracks (cheating...)





Track Fitting Results



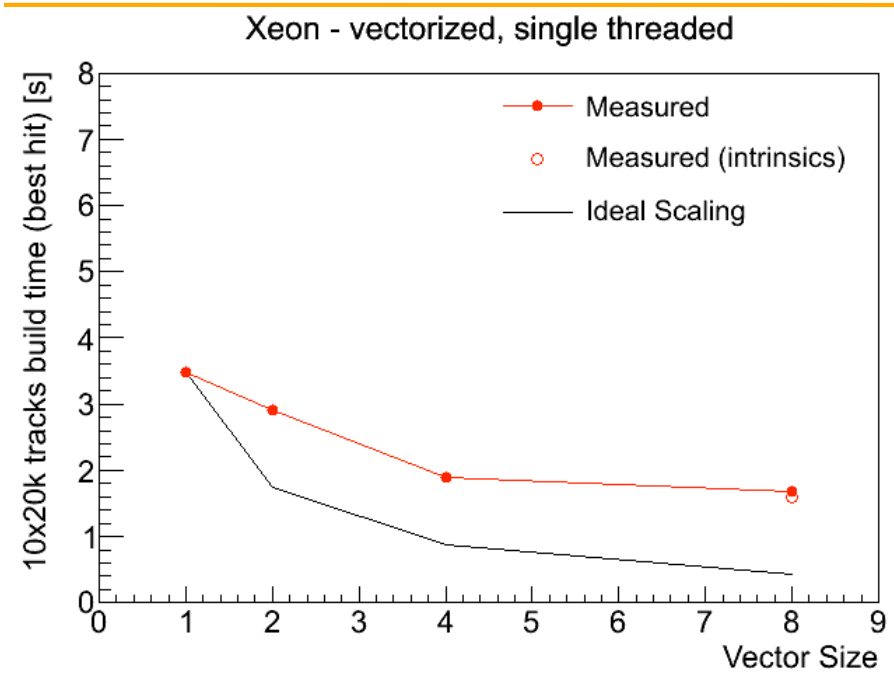


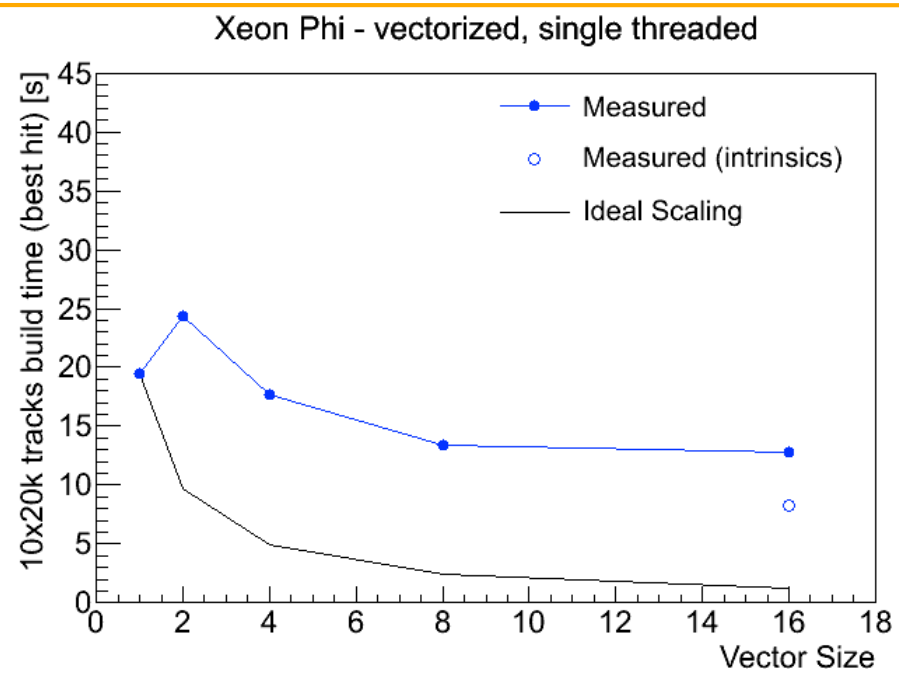
- **Track fit** implemented using **Matriplex**
  - same physics results and faster than SMatrix even in serial case
  - tested both on **Intel Xeon and Xeon Phi** (native application) with OpenMP, **similar qualitative results**

- Observe **large speedup** both from vectorization and parallelization.
  - Effective performance of vectorization is about 50% utilization efficiency.
  - Parallelization performance is close to ideal in case of 1 thread/core
    - some overhead with 2 threads/core

- Both **issues related to L1 cache**
  - Data availability and data packing in matriplex format

Track Building Vectorization Results



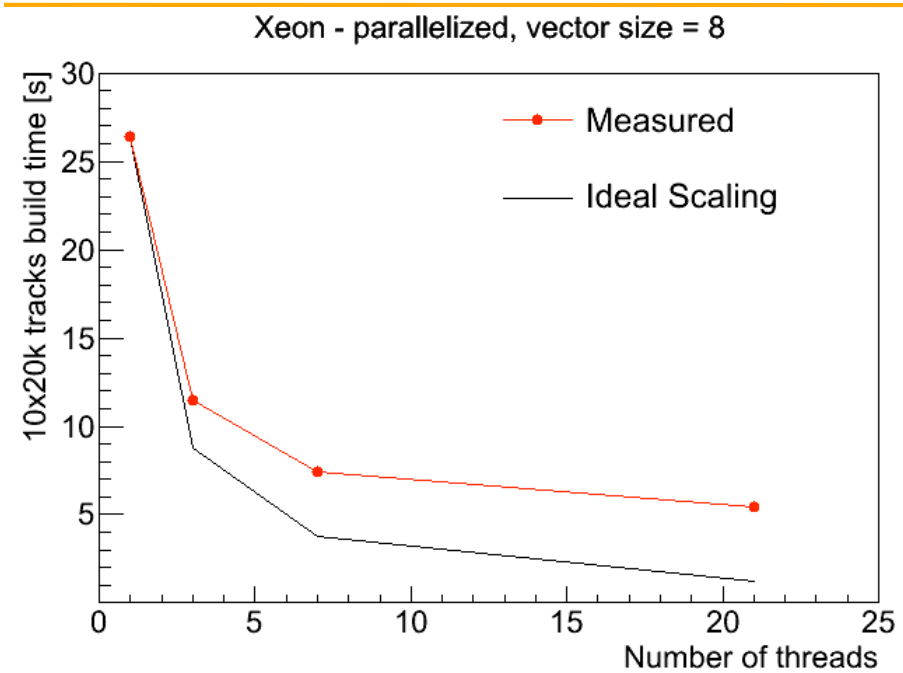


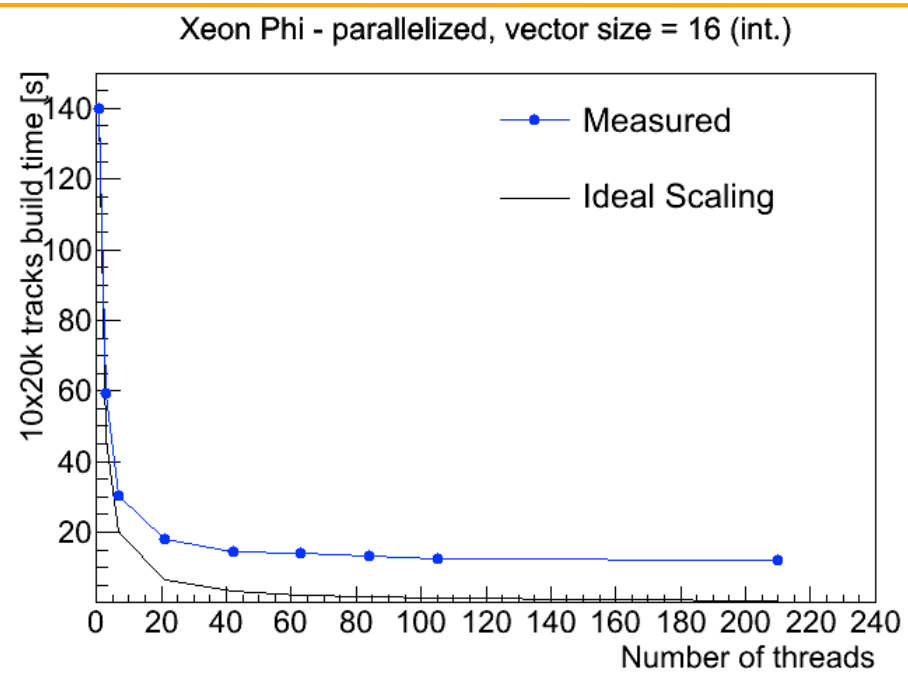
- Run **simplified track building** (best hit option) on 10 events with 20k tracks each
  - pick hit in compatibility window with lowest chi2 at each layer
  - 70% (93%) of tracks found with ≥90% (60%) of the hits

- Already much **more difficult than fitting case**, expect worse results:
  - test multiple (non pre-determined) hits per track
    - compatibility window and hits to process are not fully defined until propagation to layer

- Results show a **maximum speedup of >2x** both on Xeon and Xeon Phi
  - reasonable scaling on Xeon
  - overhead observed when enabling vectorization on Xeon Phi, then speedup
    - further **gain from using prefetching and gathering** instrinsics, but data input still takes a large fraction of the time!

Track Building Parallelization Results





- Run **full track building** with combinatorial expansion of candidates
  - ultimate physics performance, slower
  - 85% (95%) of tracks found with ≥90% (60%) of the hits

- Parallelization is implemented by **distributing threads across 21 eta bins**
  - for nEtaBin multiple of nThreads, split eta bins in threads
  - for nThreads multiple of nEtaBin, split seeds in bin across nThreads/nEtaBin threads

- Large **speedup** achieved, both on Xeon and Xeon Phi
  - up to **~5x on Xeon and >10x Xeon Phi**
  - speedup saturates above nThreads=42

Bottlenecks of single threaded running

VTune Hotspots Analysis on CHEP code

Advanced Hotspots Hotspots viewpoint (change)

Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up | Caller/Callee | Top-down Tree | Tasks and Frames

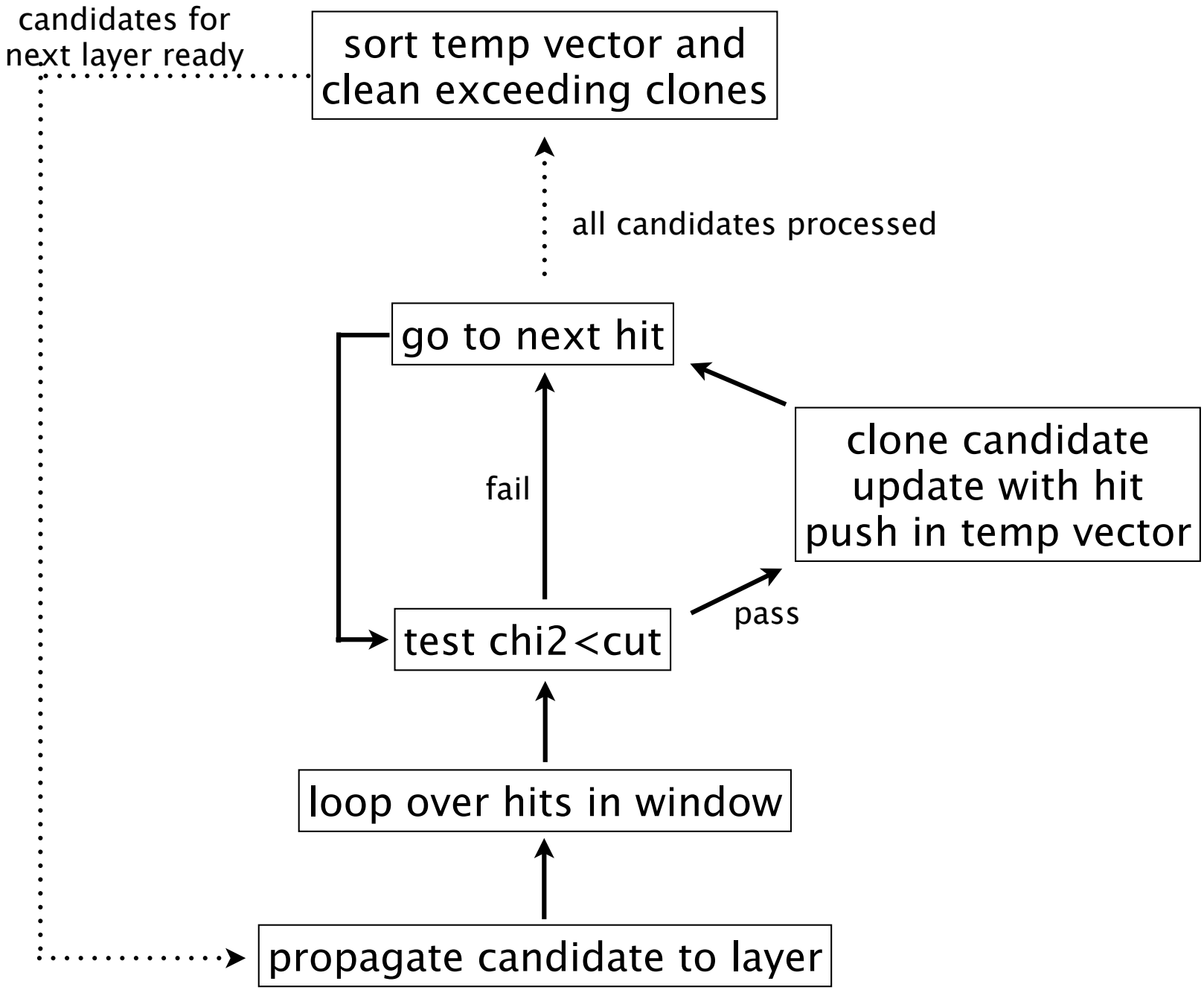
Grouping: Function / Call Stack

Function / Call Stack	CPU Time: Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)	CPU Time: S. Ti.	CPU Time: O. Ti.	Instructions Retired	Estimated Call Count	Total Iteration Count
std::vector<int, std::allocator<int>>::vector	40.772s	0s	0s	114,991,736,536	728,825,808	0
_int_free	39.751s	0s	0s	136,359,038,066	0	1,125,954,207
operator new	32.712s	0s	0s	86,154,002,942	0	0
atan2f	30.187s	0s	0s	96,263,571,713	0	0
brk	14.193s	0s	0s	2,656,096,078	0	0
Matriplex::MatriplexSym<float, (int)3, (int)8>::SlurpIn	13.738s	0s	0s	27,254,784,743	0	0
std::vector<Hit, std::allocator<Hit>>::vector	13.491s	0s	0s	48,368,155,014	1,447,206,650	6,041,737
Matriplex::CramerInverterSym<float, (int)3, (int)8>::Invert	8.327s	0s	0s	15,279,940,773	0	0
std::__unguarded_linear_insert<__gnu_cxx::__normal_iterator<Track*, std::vector<Track, std::allocator<T	6.851s	0s	0s	40,713,325,132	59,662,888	888,022,699
ROOT::Math::MatRepSym<float, (unsigned int)6>::operator=	6.092s	0s	0s	12,600,131,879	0	467,391,832
__intel_ssse3_rep_memmove	5.754s	0s	0s	14,338,306,198	0	0
std::vector<std::vector<Track, std::allocator<Track>>, std::allocator<std::vector<Track, std::allocator<T	4.927s	0s	0s	8,850,791,643	17,446	13,912,039
std::vector<EtaBinOfCombCandidates, std::allocator<EtaBinOfCombCandidates>>::~vector	4.838s	0s	0s	5,514,436,399	0	34,567,836
MkFitter::FindCandidates	4.508s	0s	0s	11,976,985,333	7,887,339	187,147,759
std::vector<Track, std::allocator<Track>>::reserve	4.334s	0s	0s	7,961,238,732	14,178,785	0
free	3.918s	0s	0s	12,843,035,454	0	0
std::vector<int, std::allocator<int>>::_M_emplace_back_aux<int const&>	3.012s	0s	0s	24,161,489,523	394,041,601	0
Matriplex::MatriplexSym<float, (int)6, (int)8>::operator=	2.818s	0s	0s	9,673,130,099	0	1,350,384,733
Track::Track	2.786s	0s	0s	7,584,629,305	93,542,787	463,911,688
_IO_file_write	2.592s	0s	0s	435,958,384	0	0
propagateHelixToRMPlex	2.203s	0s	0s	3,122,056,392	0	0
std::__insertion_sort<__gnu_cxx::__normal_iterator<Track*, std::vector<Track, std::allocator<Track>>>, I	2.164s	0s	0s	7,990,728,691	5,356,129	62,442,951

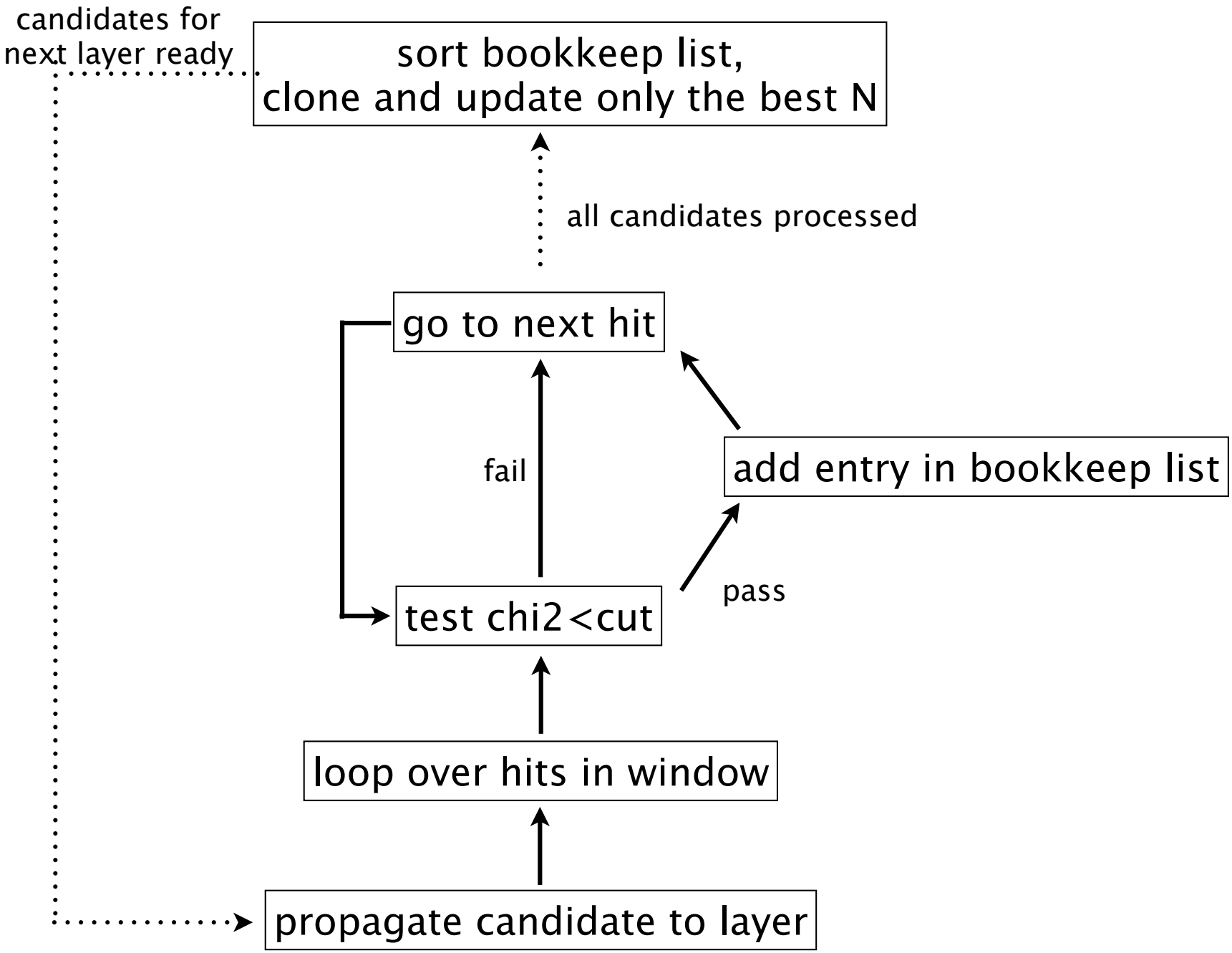
Leading functions are all memory operations!
Cloning of candidates and loading of hits in cache are the bottlenecks.

(note that atan2f is mainly in event preparation – not counted in timing tests)

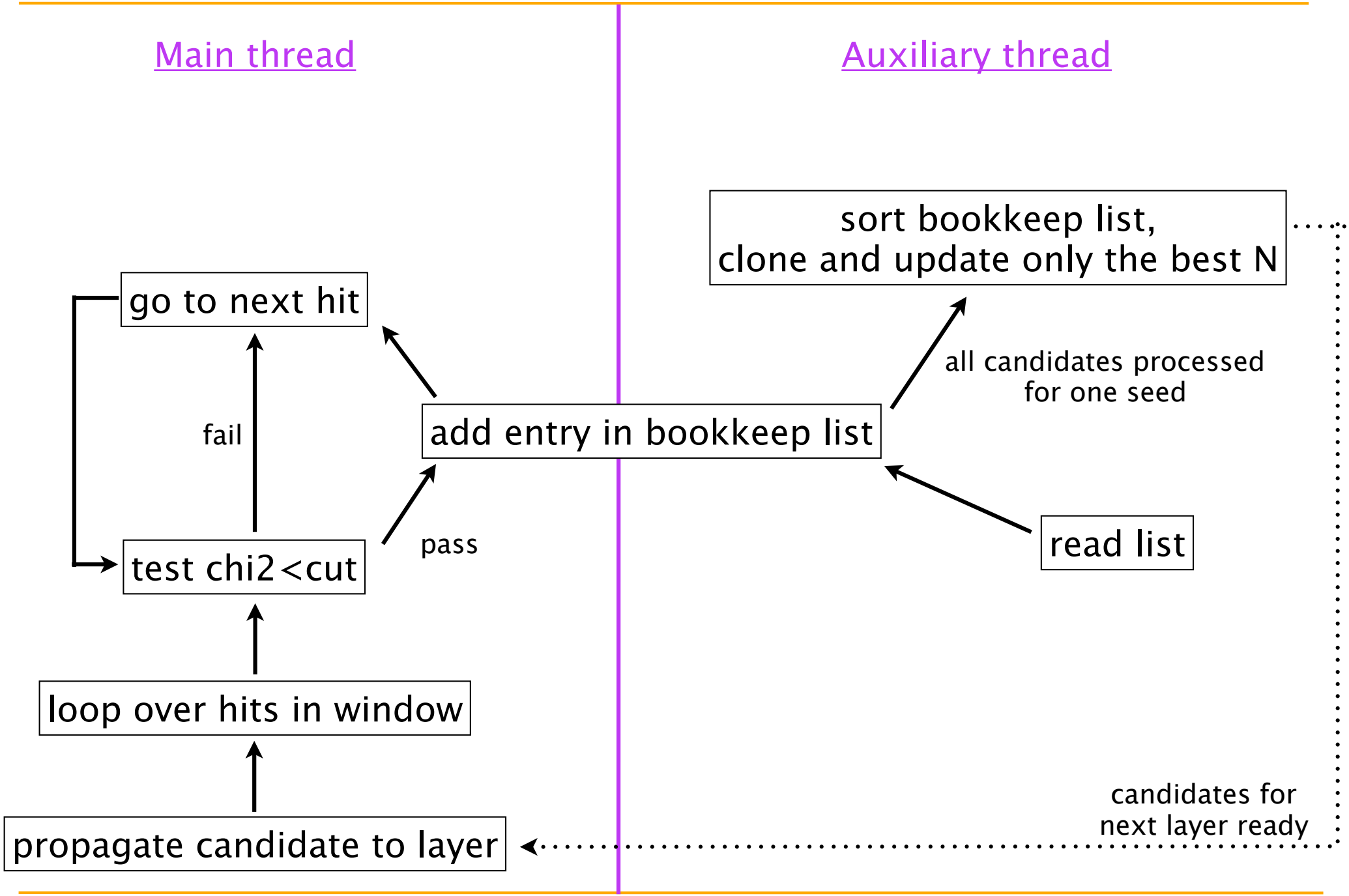
Cloning: CHEP approach



Cloning engine approach



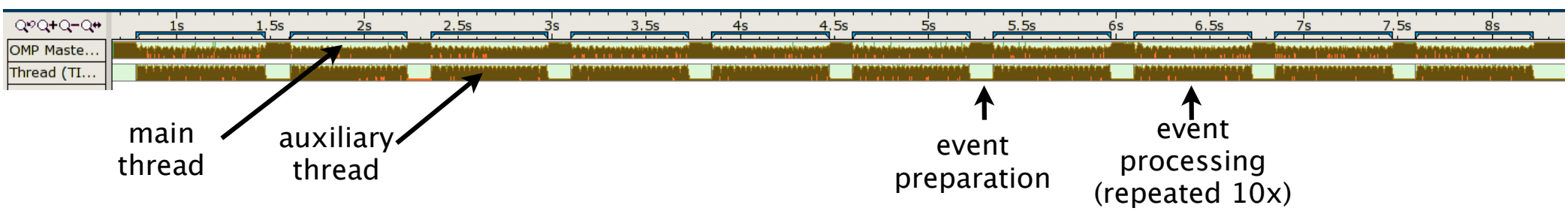
Threaded cloning engine approach



Performance

20k tracks per event, report time for 10 events

	VU01 phiphi [s/10 evt]	VU08 phiphi [s/10 evt]	VU01→08 speedup [%]	VU01 mic0 [s/10 evt]	VU16 mic0 [s/10 evt]	VU01→08 speedup [%]
CHEP	17.40	12.40	29	94.31	70.76	25
cloning engine	17.87	8.20	54	93.00	50.00	46
threaded cloning engine	11.60	6.13	47	64.50	38.00	41
speedup [%] th. c.e. vs c.e.	35	25	-	31	24	-



Cloning engine gives large speedup from vectorization.
Threaded cloning engine gives significant speedup over serial cloning engine: 25-35%
(full utilization of parallel threads would be 50%)

Reducing size of data formats

- Size of Hit and Track objects is crucial since they have heaviest impact on memory
- Current versions carry data members that are not necessarily needed
  - MC truth information, copy of hit vector
  - parameters and errors stored as SMatrix objects which are heavier than just the array of floats
- Reduce size of Track by 20% and size of Hit by 40%
  - change SMatrix members to plain arrays (only for hits for now)
  - move MC truth information in separate structure (with same indexing as main object)

	VU01 phiphi [s/10 evt]	VU08 phiphi [s/10 evt]
CHEP	17.6	12.3
CHEP (r.d.f)	13.9	8.7
cloning engine	19.3	9.7
cloning engine (r.d.f)	18.3	8.5
threaded cloning engine	13.0	8.3
threaded cloning engine (r.d.f)	13.8	8.1

Large speedup wrt CHEP, smaller impact for cloning engine case.
Or in other terms, cloning engine more relevant when memory issues are bigger.

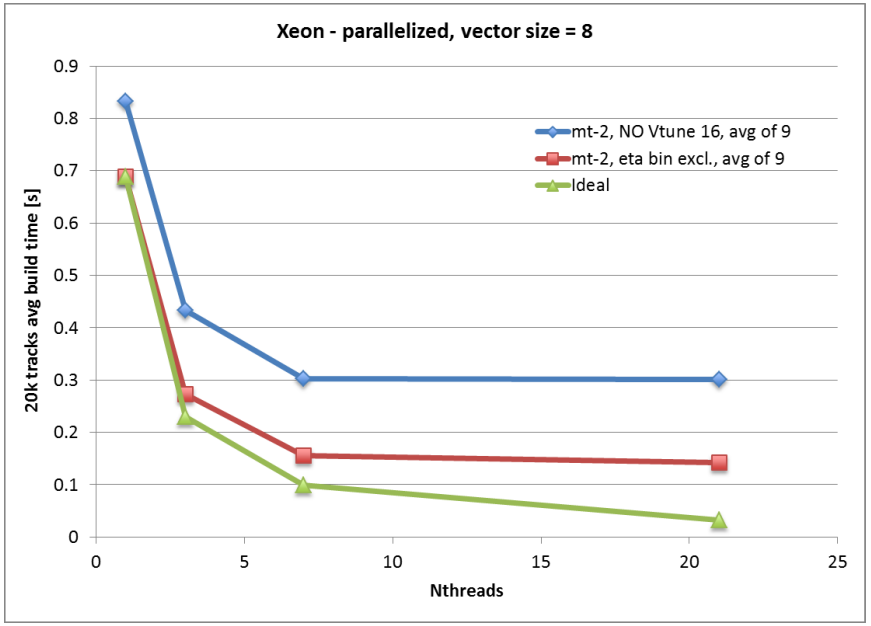
Further ideas to improve vectorization

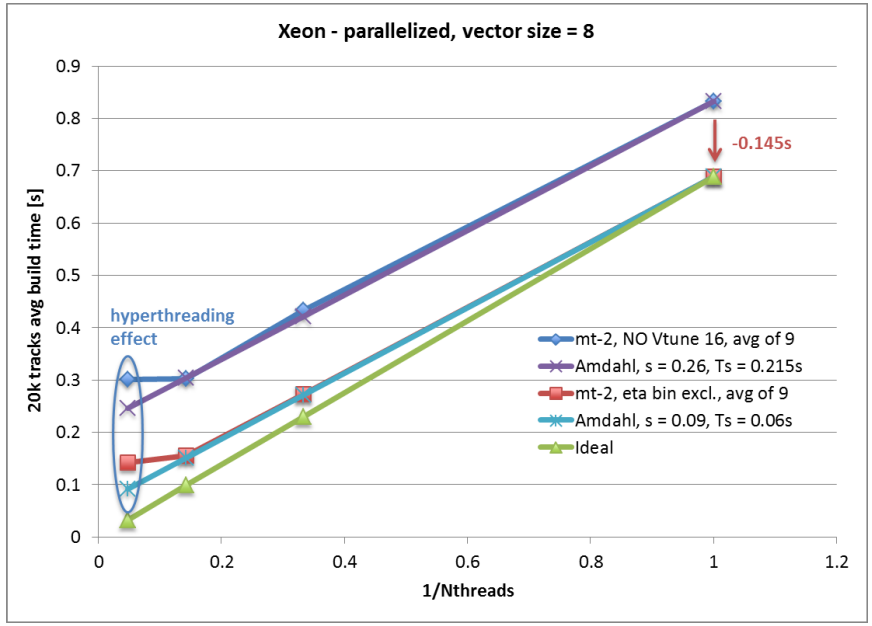
- Detailed analysis revealed where is the current bottleneck for non ideal vectorization performance

- We process 8/16 candidates in the same vector unit on Xeon/XeonPhi
- Different number of probed hits per candidate lead to dead time
  - in case the search window is very different
  - in case the local occupancy is very different
  - in case there is a track that goes crazy

- Currently tracks are simply split in eta bins
- Main idea for further improvement is to sort track candidates in a smart way
  - sort by position on next layer, sort by curvature, ...

Understanding parallelization issues





- CHEP results on Xeon consistent with a serial workload of ~25% of T1 execution
  - Fit to Amdahl's Law: T = T1 * (0.74/Nthreads + 0.26)
- Largest contribution coming from re-instantiation of data structure at each event
- Replacing deletion/creation with simple reset gave large improvement
  - Amdahl still fits: T = T1 * (0.91/Nthreads + 0.09)

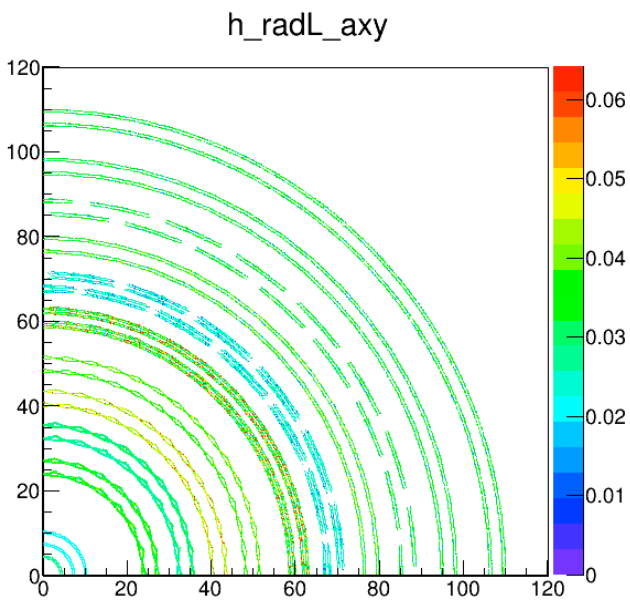
- Significant residual contribution to non-ideal scaling is due to non-uniformity of occupancy within threads, i.e. some threads take longer than others
  - clear limitation of distributing the thread work among eta bins
- Work ongoing to define strategies for an efficient 'next in line' approach or a dynamic reallocation of thread resources

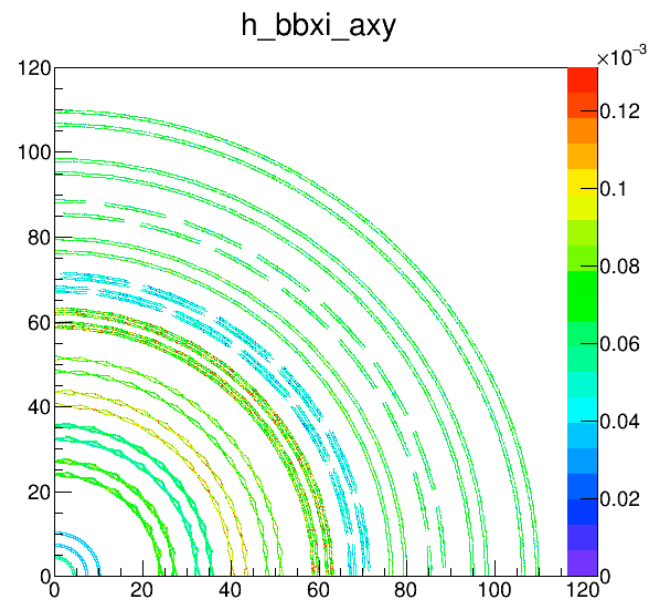
Full simulation results

- We are working towards reconstructing tracks from a full simulation or from real detector data
  - non ideal geometry, material effects
  - detector inefficiencies, non-gaussian tails in hit position
  - particle clustering in jets, particle decays

- The simplest way to do this is to interface with CMSSW
  - indirect way, dumping and reading from an ntuple
  - save and link information from all tracking-related collections
    - hits, seed, tracks - both simulated and reconstructed
  - maximal flexibility:
    - use simulation, local reconstruction or steps in global reconstruction as our starting point
    - allow direct comparison of same events with CMSSW reconstruction (but this is not so straightforward)
  - can be useful for all sort of tracking studies in CMS: https://github.com/cerati/tracking-tests/tree/master/TrackingNtuple

Material in CMSSW reconstruction

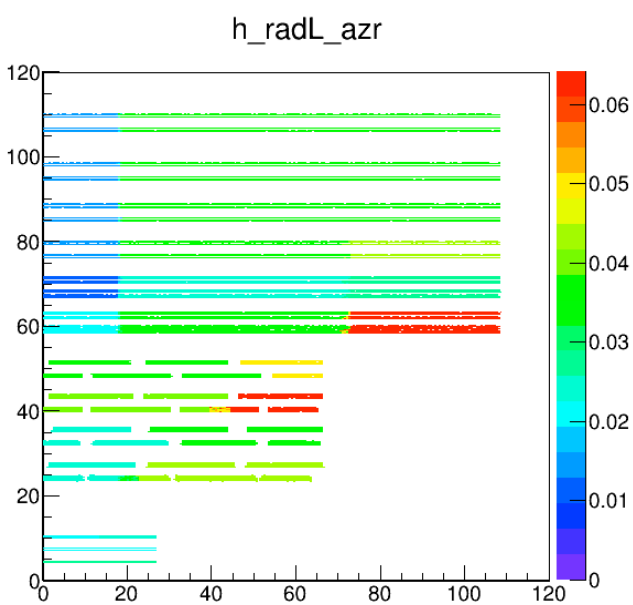


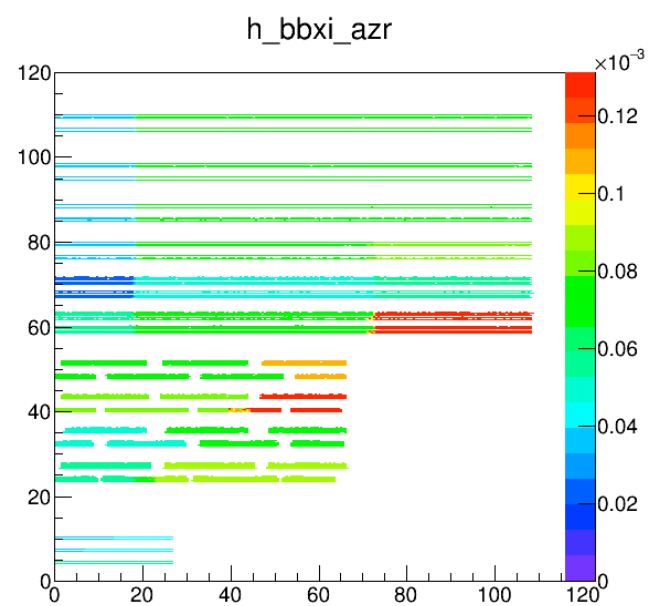


In CMSSW the tracker material is parametrized in terms for radiation length (radL) and energy loss ($\xi = Kz^2Z/A$ term in Bethe-bloch formula)

All material (including services) is assumed on the detector module, so the effect is that the parameters are flat in phi but vary significantly vs r and z

Maps made from ntuple, using simHit position

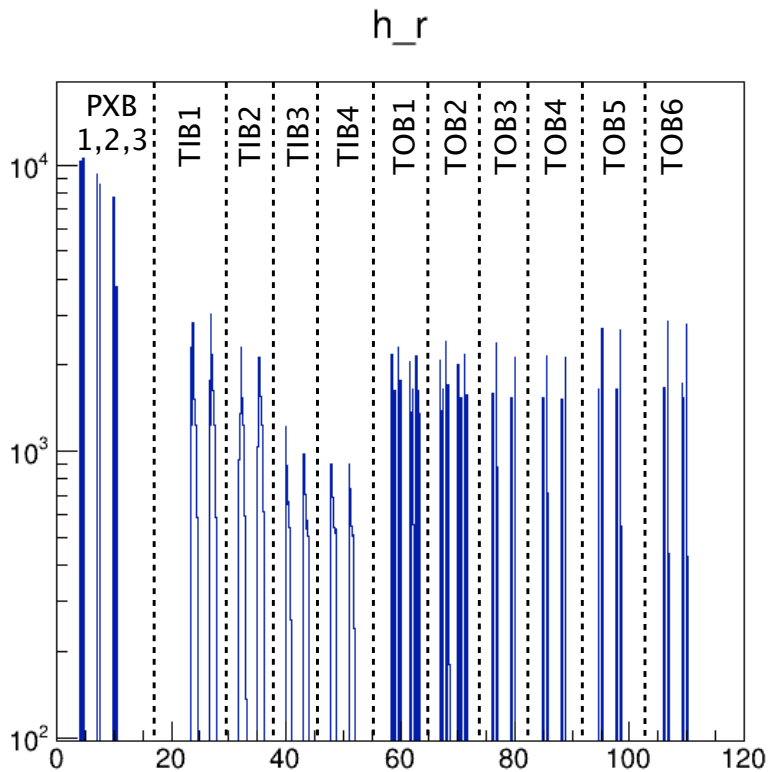




Material Effects in mkFit

- Add two float data members to Hit class: radl and xi
  - filled when reading simulation txt file from cmssw ntuple
- The PDG formulas are used when propagating to the hit position
  - multiple scattering uses radl to inflate the momentum errors
  - energy loss uses xi to reduce |p| (no change in uncertainty, assume negligible)
  - not when propagating to the average radius (we do not know the hit yet):
    - pros: effects are applied only once, included in chi2 calculation and trajectory update;
    - cons: they are not included in window search, not applied to invalid hits
- Thoughts on a possibly better approach
  - parametrize the radl and xi values as a function of layer and z
  - avoid increasing the size of the Hit class
  - can include material effects in window search calculation and for invalid hits
- Overall a small effect in the barrel:
  - increase the covariance by $\leq O(10^{-6})$ GeV^2, i.e. O(MeV) on momentum error

Layer radii



Average radii [cm]:

```
PXB1 =   4.42
PXB2 =   7.31
PXB3 =  10.17
TIB1 =  25.65
TIB2 =  33.81
TIB3 =  41.89
TIB4 =  49.67
TOB1 =  60.95
TOB2 =  69.11
TOB3 =  78.19
TOB4 =  86.84
TOB5 =  96.78
TOB6 = 108.10
```

Build reco geometry with cylindric barrel layer at average radii.
First propagation step to average radii, find hits in compatibility window.
For hits in window, perform second propagation step:
compute chi2 and update parameters at exact hit radius.

Two main issues:
Spread within strip layers comparable to distance to next layer.
Large gap from PXB3 to TIB1.

Results on CMSSW simulation

Single event with 100 tracks merging from single muon simulation events.
Hits obtained from smearing of SimHit positions.

	Tot found	with $\Delta p_T<10\%$	Found with $N^{hits}\geq 8$	with $\Delta p_T<10\%$
p_T=1 GeV, χ^2 cut=30	100	100	78	78
p_T=10 GeV, χ^2 cut=30	100	100	94	94

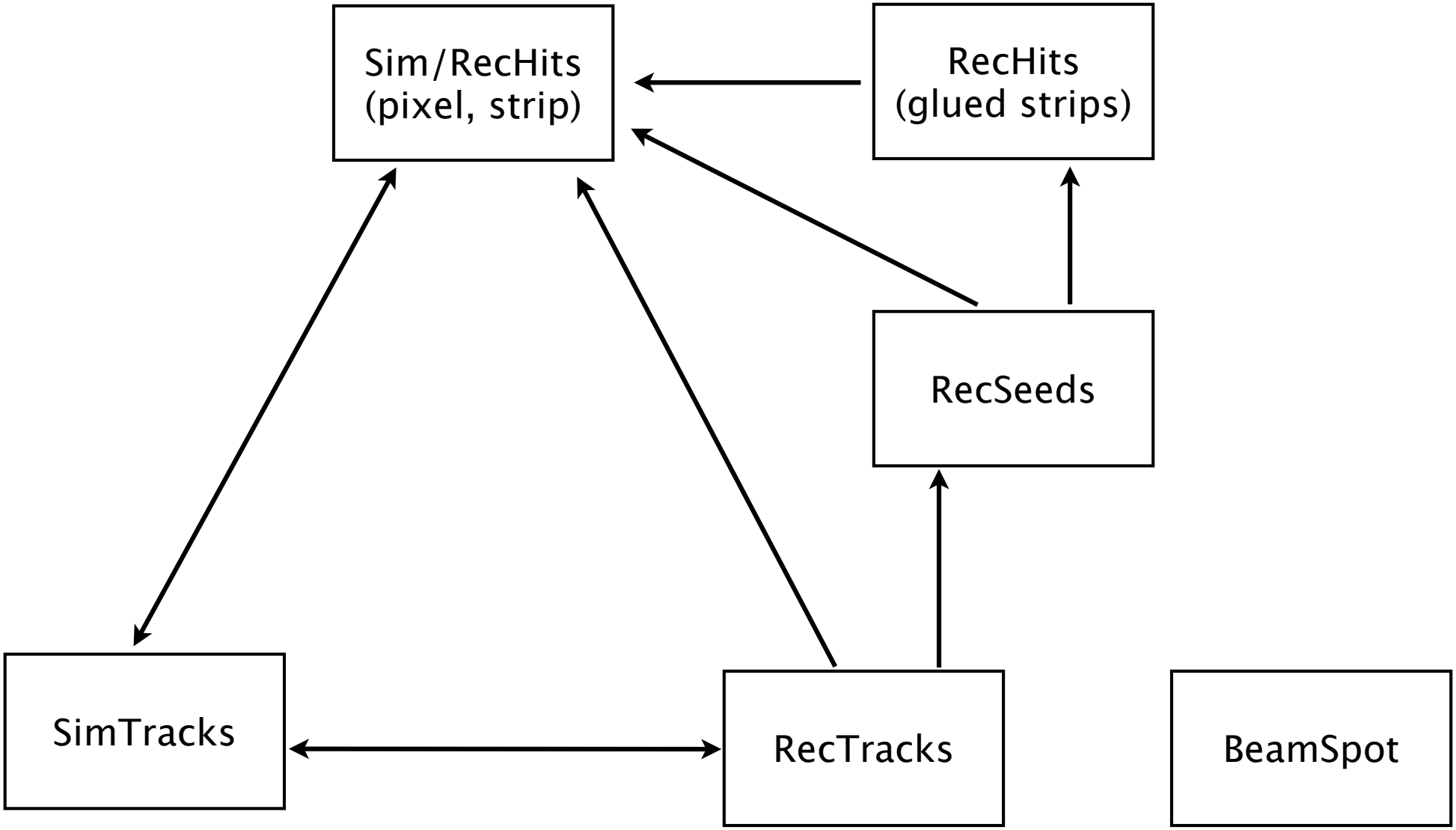
Performance not far from full efficiency.
Detailed debugging of problematic cases in ongoing.

Conclusions

- Significant progress in parallelized and vectorized tracking R&D on Xeon/Xeon Phi
- Good understanding of bottlenecks and limitations, new versions of the code faster and closer scaling to ideal
  - ideas to further improve performance
- Setup to process fully realistic data (CMSSW), with encouraging preliminary results
- The project is solid and promising but we still have a long way to go

backup

TrackingNtuple



https://github.com/cerati/tracking-tests/tree/master/TrackingNtuple

Neglecting material effects

Just for curiosity, let's neglect material effects in reconstruction...

	Tot found	with $\Delta p_T<10\%$	Found with $N^{hits}\geq 8$	with $\Delta p_T<10\%$
p_T=1 GeV, χ^2 cut=15	100	100	21	21
p_T=10 GeV, χ^2 cut=15	100	100	94	94

No effect on 10 GeV tracks, large reduction in hit collection efficiency for 1 GeV.

This is exactly what we expected, looks like our setup in now in good shape!