Performance optimization of the BmnRoot reconstruction modules

S NEMNYUGIN, S MERTS, A IUFRYAKOVA

SAINT-PETERSBURG STATE UNIVERSITY

JOINT INSTITUTE OF NUCLEAR RESEARCH

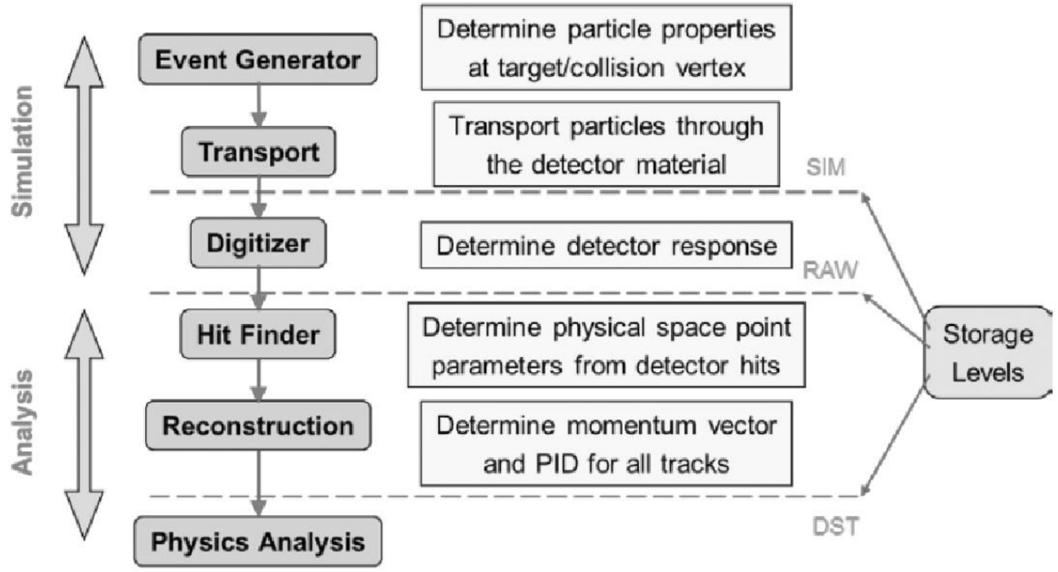
Outline

- Performance study and performance bottlenecks of the BmnRoot reconstruction modules.
- Algorithmic optimization of the BmnNewFieldMap.
- Compiler optimizations and code improvements of the BmnRoot reconstruction modules.
- Adaptation of the BmnRoot reconstruction modules to multicore.

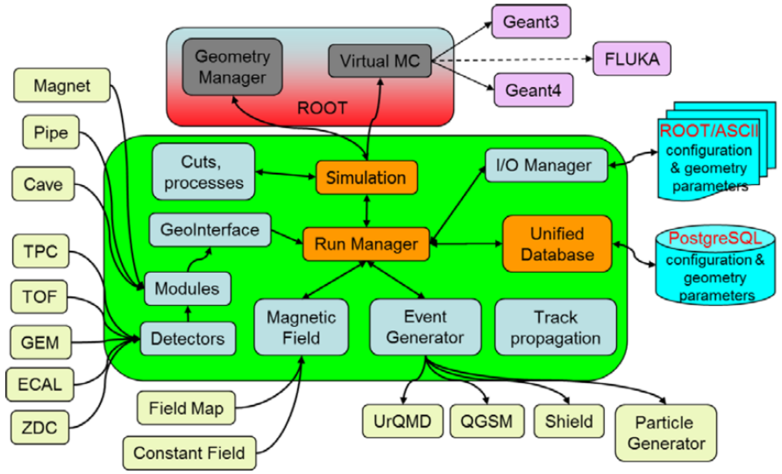
Work is supported by Russian Foundation for Basic Research grant 18-02-40104 mega.

BmnRoot framework

- ✓ BmnRoot framework is based on the FairRoot and FairSoft software packages (GSI, Darmstadt).
- ✓ Complex structure (simulation/analysis) with a lot of modules, hundreds of thousands lines of code.
- ✓ Simulation: setup configuration and geometry, beam parameters, Monte-Carlo event generators (BOX, DQGSM, UrQMD, SHIELD), Virtual Monte-Carlo, transport codes (Geant3, Geant4, Fluka), magnetic field maps, digitizers etc.
- ✓ Simulation performance should be improved.



Reasonable Monte-Carlo simulation needs for large sampling size.
Realistic simulations are time-consuming.



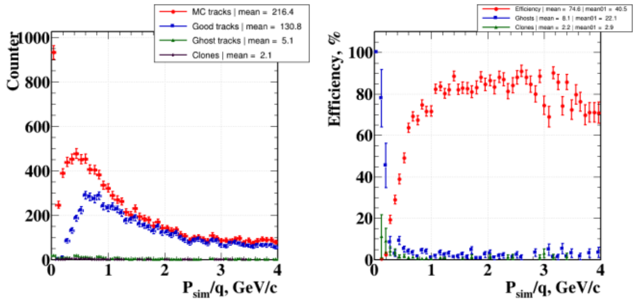
BmnRoot simulation modules

Optimization of the BmnRoot simulation modules

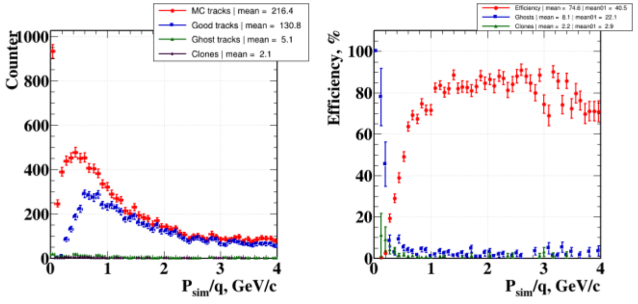
- ✓ Performance optimization (parallelization of most time-consuming hotspots).
- ✓ Tests of correctness and scalability of optimized code (Quality Assurance).

BmnGemStripDigitizer

```
…
FairMCPoint* GemStripPoint;
Int_t NNotPrimaries = 0;
#pragma omp parallel
#pragma omp for schedule(dynamic)
for (UInt_t ipoint = 0; ipoint < fBmnGemStripPointsArray->GetEntriesFast();
ipoint++) {
GemStripPoint = (FairMCPoint*) fBmnGemStripPointsArray->At(ipoint);
…
```



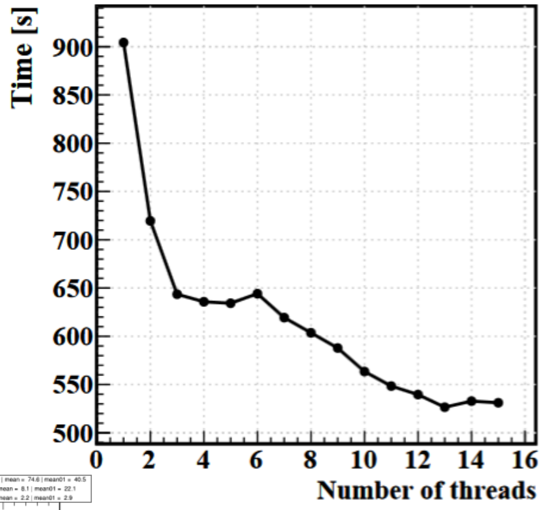
QA - 1 thread

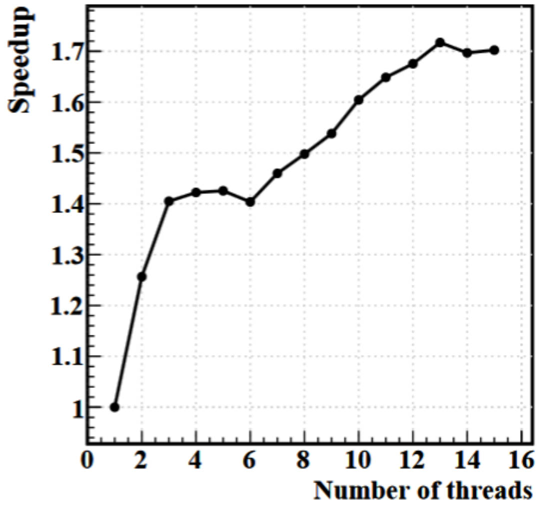


QA - 4 thread

Quality Assurance for simulation

OpenMP parallelization

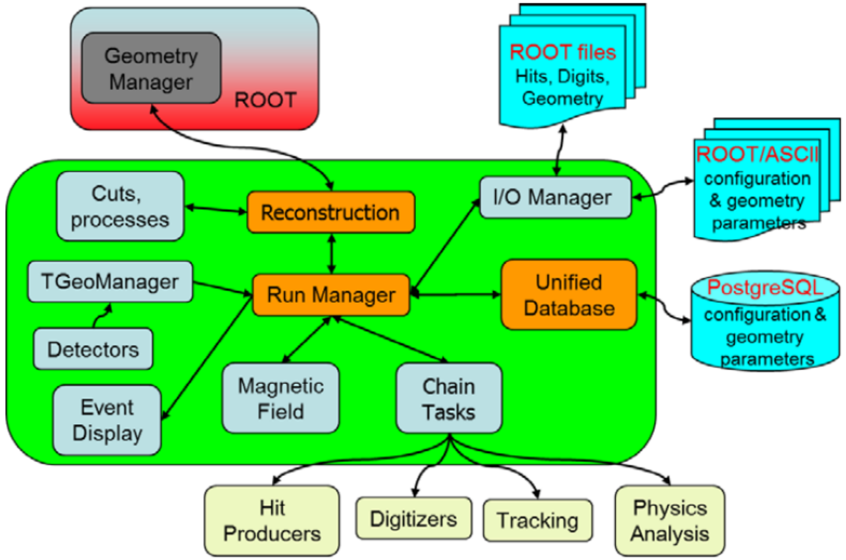




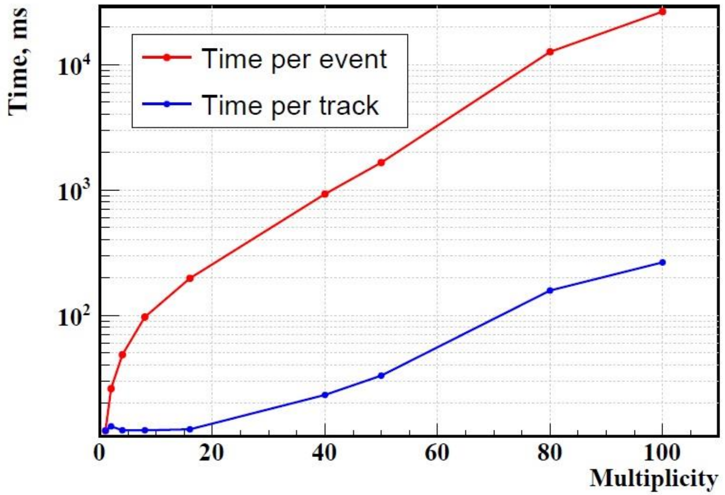
Scalability of the BmnRoot simulation parallelized with OpenMP

BmnRoot framework

1. Reconstruction: setup configuration and geometry, all detector subsystems (GEMs, multiwired chambers, drift chambers, silicon detector modules, zero-degree calorimeters, TOF, two arms for the SRC experiment etc.), beam parameters, magnetic field accounting, digitizers, matching (local/global) etc.
2. Reconstruction performance should be improved.

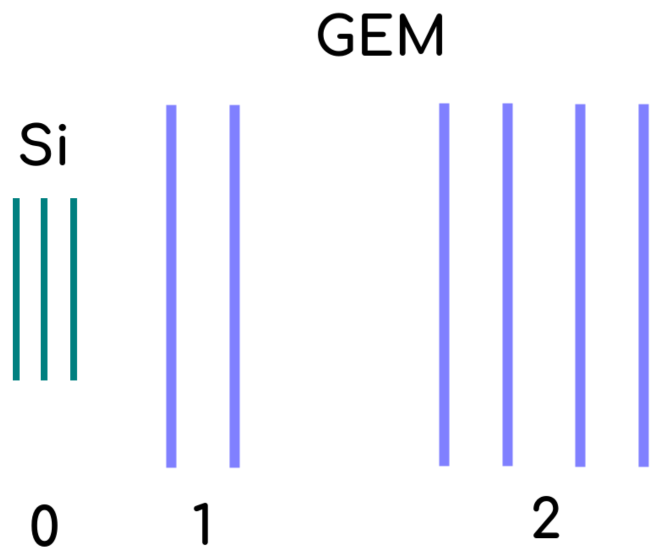


BmnRoot reconstruction modules

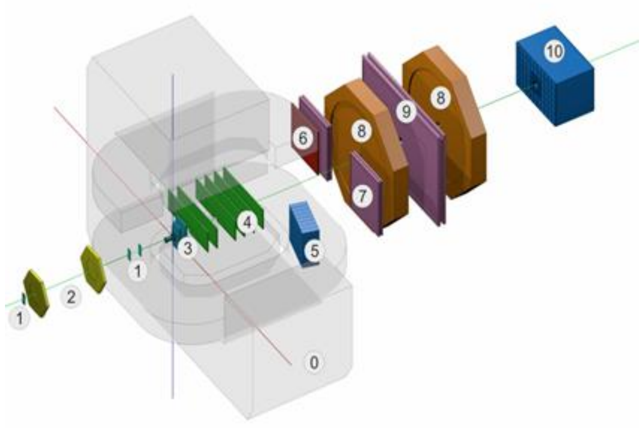


BmnRoot reconstruction time vs events multiplicity

BmnRoot reconstruction (new version)



GEM – Gas Electron Multiplier.
SI – Silicon detector.



Track reconstruction algorithm

1. Search for high momentum tracks.
Construct 4-hits candidates and estimate their parameters in zone 2. Propagate each candidate to hits in zone 1 and zone 0 by Kalman Filter (KF) etc.

2. Search for high momentum tracks with low efficiency.
Construct 3-hits candidates and estimate their parameters in zone 2 for UNUSED hits. Propagate each candidate to hits in zone 1 and zone 0 by KF etc.

3. Search for low momentum tracks with inefficiency.
Construct 2-hits candidates in zone 1 for UNUSED hits. Propagate each candidate to hits in zone 0 by straight line in ZY plane etc.

BmnInnerTrackingRun7::Exec()

```
…
FindTracks_4of4_OnLastGEMStations();
FindTracks_3of4_OnLastGEMStations();
fNHitsCut = 5;
FindTracks_2of2_OnFirstGEMStationsDownstream();
FindTracks_2of2_OnFirstGEMStationsUpstream();
…
```

Performance bottlenecks of the BmnRoot reconstruction modules

- ✓ **Monte Carlo data 1 sec/event**
- ✓ **Experimental data 6 sec/event**
- ✓ **One file (200 000 event) up to 2 weeks**

Testbench

CPU: Intel Xeon E-2136 @ 4.5GHz Turbo (6C 2xHT, L3 Cache 8MB)
RAM: 32GB 2666MHz DDR4
OS: Ubuntu 16.04.6 LTS

Testcase

Simulation data with DQGSM generator
1000-5000 events.
Experimental data: Run 7 at BM@N, Argon beam, Al target.
Macro run_reco_bmn.C

Details of CPU Time Consumption
Si+GEM Track Finder: 45%
Global Matching: 21%
Vertex Finder: 19%

Analysis summary

A lot of hotspots belong to the BmnField module – load of the analyzing magnet field:

- 3D Cartesian lattice;
- piecewise linear interpolation between lattice nodes;
- extrapolation outside known values.

HOTSPOTS

Function / Call Stack	CPU Time	Module
clock	66.450s	libc.so.6
BmnKalmanFilter::RK4Order	34.481s	libBmnData.so.0.0.0
BmnNewFieldMap::FieldInterpolate	23.124s	libBmnField.so.0.0.0
TArrayF::At	22.950s	libBmnField.so.0.0.0
inflate	20.673s	libz.so.1
BmnKalmanFilter::TransportC	17.638s	libBmnData.so.0.0.0
BmnNewFieldMap::IsInside	15.992s	libBmnField.so.0.0.0
TArray::BoundsOk	15.566s	libBmnData.so.0.0.0
BmnFieldMap::Interpolate	15.226s	libBmnField.so.0.0.0
std::vector<double, std::allocator<doub	13.862s	libBmnData.so.0.0.0
operator new	12.704s	libstdc++.so.6
std::vector<double, std::allocator<doub	12.222s	libBmnDst.so.0.0.0
BmnKalmanFilter::RK4TrackExtrapolat	11.896s	libBmnData.so.0.0.0
std::vector<double, std::allocator<doub	11.128s	libBmnData.so.0.0.0
TGeoVoxelFinder::GetNextCandidates	10.982s	libGeom.so.6.16
__pow	10.944s	libm.so.6
std::__fill_n_a<double*, unsigned long	10.074s	libBmnData.so.0.0.0
TGeoVoxelFinder::GetCheckList	9.832s	libGeom.so.6.16
__GI_	8.701s	libc.so.6
std::vector<double, std::allocator<doub	8.420s	libBmnData.so.0.0.0
std::vector<BmnLink, std::allocator<Bn	7.352s	libSilicon.so.0.0.0
TGeoNavigator::SearchNode	6.766s	libGeom.so.6.16

Hotspots of the BmnRoot reconstruction modules by Intel Parallel Studio - Intel® VTune™ Profiler

Optimization of the BmnRoot reconstruction modules

Compiler (GCC) optimization

DEBUG➔RELEASE
(O2 level optimization)

Tracking parameters selection

Before optimization	After optimization
Monte Carlo 1 sec/event	Monte Carlo 0.3 sec/event
Experimental 6 sec/event	Experimental 0.7 sec/event
One file (200 000 events) 2 weeks	One file (200 000 events) 39 hours

Source code improvements

1. More efficient addressing.
2. Replacement of small arrays to variables.
3. More efficient programming of arithmetical expressions etc.

Low effect (CPU time reduction by percent's)

Compiler (GCC) optimization

Aggressive vectorization.

Autoparallelization of loops.

Profile-guided optimization.

Data alignment.

Various kinds of loops optimization

etc

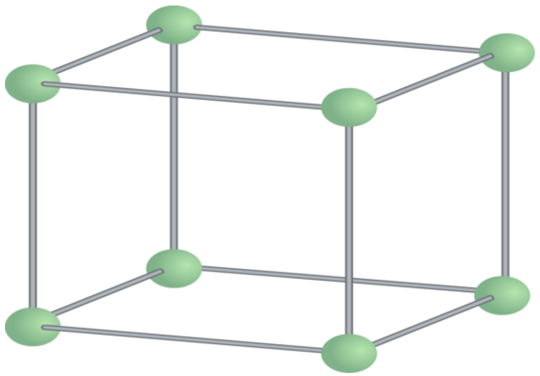
➔

not efficient

Optimization of the BmnRoot reconstruction modules

Algorithmic optimization of BmnFieldMap

1. Second hotspot next to the Kalman Filter.
2. Measured values of the analyzing magnet's field are saved at discrete set of 3-dimensional cubic lattice.
3. Proposal for optimization – replace linear-piecewise interpolation by constant-piecewise interpolation. Calculation for 8 vertices of cube elementary cell is not necessary => reduction of number of floating point operations.



$\Delta x = 0.25$ cm
$\Delta y = 0.45$ cm
$\Delta z = 0.17$ cm

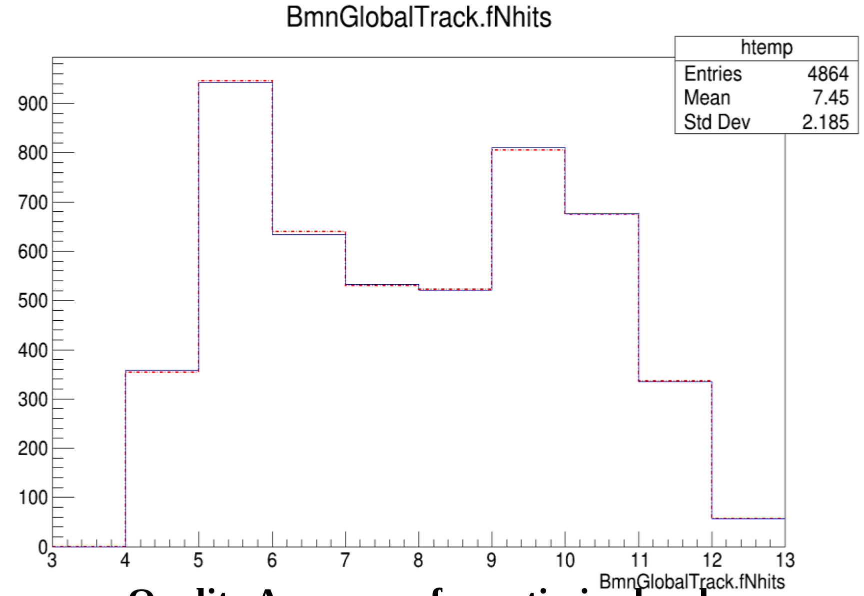
```
Double_t BmnFieldMap::Interpolate(Double_t dx, Double_t dy, Double_t dz) {

 /**  // Interpolate in x coordinate
    fHb[0][0] = fHa[0][0][0] + (fHa[1][0][0] - fHa[0][0][0]) * dx;
    fHb[1][0] = fHa[0][1][0] + (fHa[1][1][0] - fHa[0][1][0]) * dx;
    fHb[0][1] = fHa[0][0][1] + (fHa[1][0][1] - fHa[0][0][1]) * dx;
    fHb[1][1] = fHa[0][1][1] + (fHa[1][1][1] - fHa[0][1][1]) * dx;

    // Interpolate in y coordinate
    fHc[0] = fHb[0][0] + (fHb[1][0] - fHb[0][0]) * dy;
    fHc[1] = fHb[0][1] + (fHb[1][1] - fHb[0][1]) * dy;

    // Interpolate in z coordinate
    return fHc[0] + (fHc[1] - fHc[0]) * dz; **/
     return Hh; // (NEW)


}
```

Example of the code (linear interpolation of magnetic field)

Optimization of the BmnRoot reconstruction modules

```
Double_t BmnNewFieldMap::FieldInterpolate(TArrayF* fcomp, Double_t x, Double_t y, Double_t z) {
    Int_t ix = 0;
    Int_t iy = 0;
    Int_t iz = 0;
    Double_t dx = 0.;
    Double_t dy = 0.;
    Double_t dz = 0.;

    Int_t iix = 0;
    Int_t iiy = 0;
    Int_t iiz = 0;


    if (IsInside(x, y, z, ix, iy, iz, dx, dy, dz)) {

        iix = Int_t(Nint((x - fXmin) / fXstep));
        iiy = Int_t(Nint((y - fYmin) / fYstep));
        iiz = Int_t(Nint((z - fZmin) / fZstep));

        Hh = fcomp->At(iix * fNy * fNz + iiy * fNz + iiz);

       /**fHa[0][0][0] = fcomp->At(ix * fNy * fNz + iy * fNz + iz);
        fHa[1][0][0] = fcomp->At((ix + 1) * fNy * fNz + iy * fNz + iz);
        fHa[0][1][0] = fcomp->At(ix * fNy * fNz + (iy + 1) * fNz + iz);
        fHa[1][1][0] = fcomp->At((ix + 1) * fNy * fNz + (iy + 1) * fNz + iz);
        fHa[0][0][1] = fcomp->At(ix * fNy * fNz + iy * fNz + (iz + 1));
        fHa[1][0][1] = fcomp->At((ix + 1) * fNy * fNz + iy * fNz + (iz + 1));
        fHa[0][1][1] = fcomp->At(ix * fNy * fNz + (iy + 1) * fNz + (iz + 1));
        fHa[1][1][1] = fcomp->At((ix + 1) * fNy * fNz + (iy + 1) * fNz + (iz + 1));**/

        return Interpolate(dx, dy, dz);
    }
    return 0.;
}
```

Optimized code of FieldInterpolate method of BmnNewFieldMap class.



Quality Assurance for optimized code.
Histogram for the number of reconstructed hits

- ✓ Build in Debug mode (compiler optimization switched off) reduced total execution time by 10%.
- ✓ Build in O2 optimization mode reduced execution time by 4%.
- ✓ Execution time of the BmnField is 7% from total reconstruction time.
- ✓ Quality Assurance methods used in BM@N demonstrates very small difference between non-optimized and optimized results.

Parallelization of the BmnRoot reconstruction modules

Track finders OpenMP parallelization

```
BmnInnerTrackingRun7::FindTracks_4of4_OnLastGEMStations() {
const Int_t nxRanges = 8;
const Int_t nyRanges = 5;
...
vector<BmnTrack> candidates;
vector<BmnTrack> sortedCandidates;
Int_t nThreads = THREADS_N;
vector<vector<BmnTrack>> candsThread(nThreads);
clock_t t0 = 0;
Int_t threadNum;
Int_t sH8 = sortedHits[8].size();
#pragma omp parallel if(sH8 > 100)  num_threads(nThreads)
#pragma omp for  // schedule(static,1)
for (Int_t ii = 0; ii < sH8; ++ii) {
BmnHit*  hit8;
hit8 = sortedHits[8].at(ii);
...
```

Also OpenMP tasks have been tried. Not efficient

Optimization for shared memory systems (multicore) with OpenMP.
Restrictions:
- ✓ Syntax – **range-based for loops** are not supported by OpenMP.
- ✓ Program’s flow – breaks inside loops are not allowed, loop dependencies are not allowed.
- ✓ Overhead costs - only computationally “heavy” code fragments (loops) should be parallelized.

Reasonable scalability is not yet received.
Possible reasons of low efficiency:
- ✓ In many cases number of loops iterations is zero so efficiency of OpenMP-parallelization is low.
- ✓ Most significant hotspot relates to the Kalman filter, so it should be optimized first.

What next?
- ✓ Study of hotspots. Their reasons and ways of elimination.
- ✓ Kalman filter optimization?
- ✓ Vectorization.
- ✓ Hybrid computing.
- ✓ Distributed computing etc.

Conclusion

- Performance studies are performed and performance bottlenecks of the BmnRoot reconstruction modules are revealed.
- Performance bottlenecks of the BmnRoot reconstruction modules are localized.
- Various approaches to optimization of the BmnRoot tracks reconstruction modules were tested and estimates of their efficiency are obtained.

Thank you!