High-performance optimization of simulation and reconstruction modules in the BM@N software at the NICA

Driuk A.V., Merts S.P., Nemnyugin S.A., Roudnev V.A., Stepanova M.M., Yufryakova A.A.

Saint-Petersburg State University

5th International Conference on Particle Physics and Astrophysics, 9 October 2020

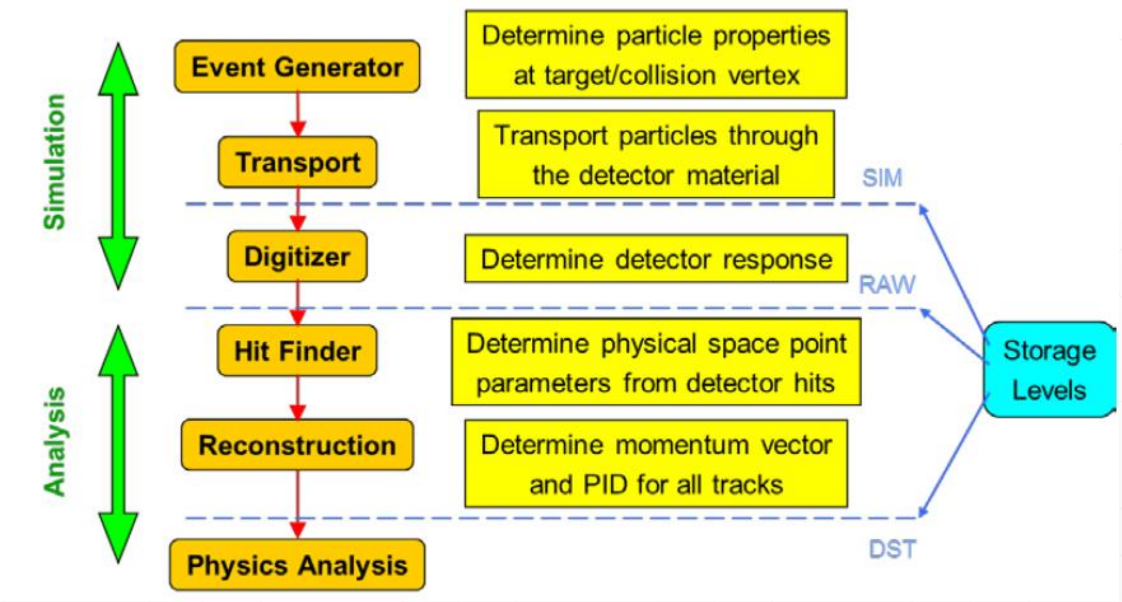
Outline

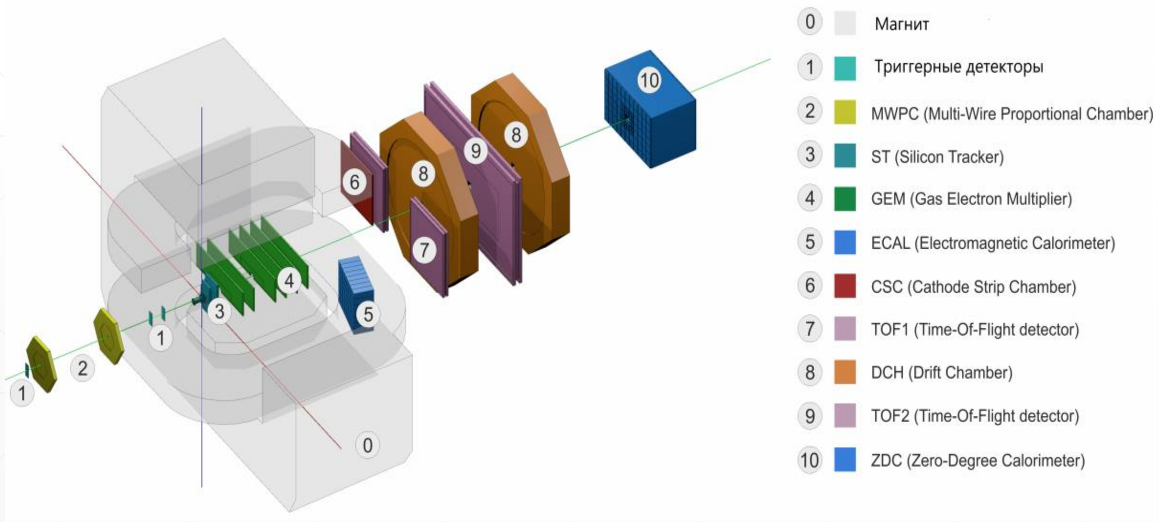
- Software for the Baryon Matter at Nuclotron (BM@N) experiment.
- Performance study and optimization of the BmnRoot simulation modules.
- Performance bottlenecks and optimization of the BmnRoot tracks reconstruction modules.

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

BM@N NICA experiment

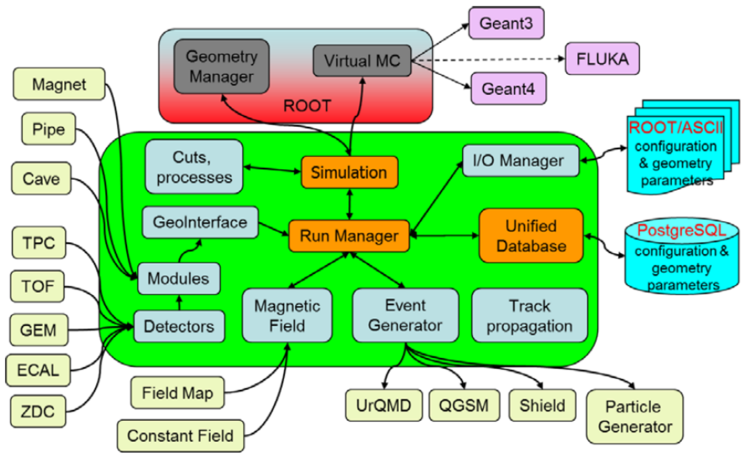
- **NICA - N**uclotron based **I**on **C**ollider f**A**cility, Joint Institute for Nuclear Research, Dubna, Russian Federation.
- **BM@N** – **Baryon M**atter at **N**uclotron experiment. Heavy ion collisions with fixed targets.
- Setup includes detector subsystems, magnet, electronics.
- Software: BmnRoot package for simulation and event reconstruction.



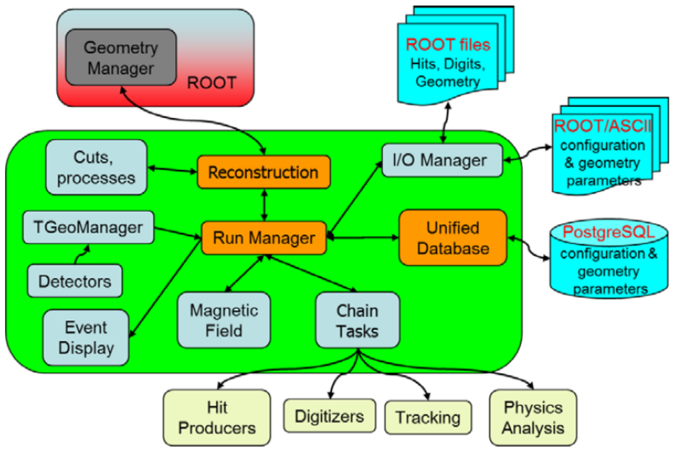


BmnRoot software

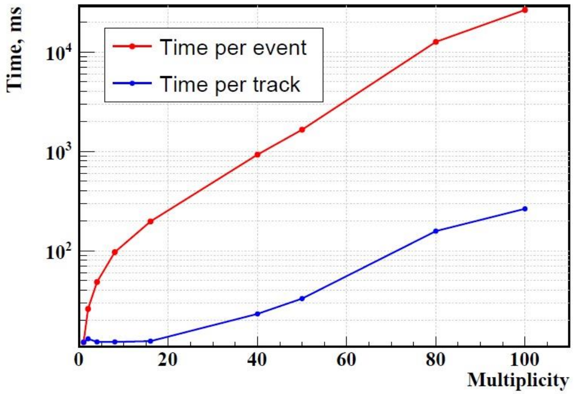
1. Is based on the FairRoot and FairSoft software packages (GSI, Darmstadt).
2. Simulation: setup configuration and geometry, beam parameters, variety of Monte-Carlo event generators, etc.
3. Reconstruction: setup configuration and geometry, beam parameters, hit producers, digitizers etc.
4. Complex structure with a lot of modules, tens (hundreds) of thousand lines of code.
5. Both simulation and reconstruction performance should be improved.



BmnRoot simulation modules



BmnRoot reconstruction modules



BmnRoot reconstruction time vs events multiplicity

Optimization of the BmnRoot simulation modules

1. Static or/and dynamic hotspot analysis of the BmnRoot simulation modules.
2. Hotspots localization.
3. Performance optimization (parallelization of most time-consuming hotspots).
4. Tests of correctness and scalability of optimized code.

Function / Call Stack	CPU Time ▼	Module
TRandom::Gaus	145.867s	libMathCore.so.6.16.00
DeadZoneOfStripLayer::IsInside	137.187s	libGem.so.0
__cos_fma	98.153s	libm.so.6
TRandom3::Rndm	80.814s	libMathCore.so.6.16.00
__sin_fma	77.566s	libm.so.6
deflate	74.185s	libz.so.1
FairMCApplication::Stepping	67.573s	libBase.so.18.2.0
BmnGemStripModule::AddRealPointFu	46.391s	libGem.so.0
BmnGemStripLayer::ConvertPointToSt	43.867s	libGem.so.0
std::map<std::pair<int, int>, int, std::les	41.465s	libBmnData.so.0
StripCluster::AddStrip	41.270s	libGem.so.0
BmnGemStripLayer::IsPointInsideStrip	31.419s	libGem.so.0
BmnGemStripLayer::ConvertNormalPo	27.492s	libGem.so.0
std::map<std::pair<int, int>, int, std::les	20.336s	libBmnData.so.0
std::operator<<int, int>	18.805s	libBmnData.so.0
BmnGemStripLayer::IsPointInsideDea	18.630s	libGem.so.0
BmnNewFieldMap::FieldInterpolate	18.493s	libBmnField.so.0
std::_Rb_tree_iterator<std::pair<int co	16.267s	libBase.so.18.2.0
TArrayF::At	14.695s	libBmnField.so.0
BmnNewFieldMap::IsInside	13.857s	libBmnField.so.0
std::map<int, bool, std::less<int>, std::a	12.740s	libBmnData.so.0
BmnFieldMap::Interpolate	12.701s	libBmnField.so.0

Hotspots of the BmnRoot simulation modules

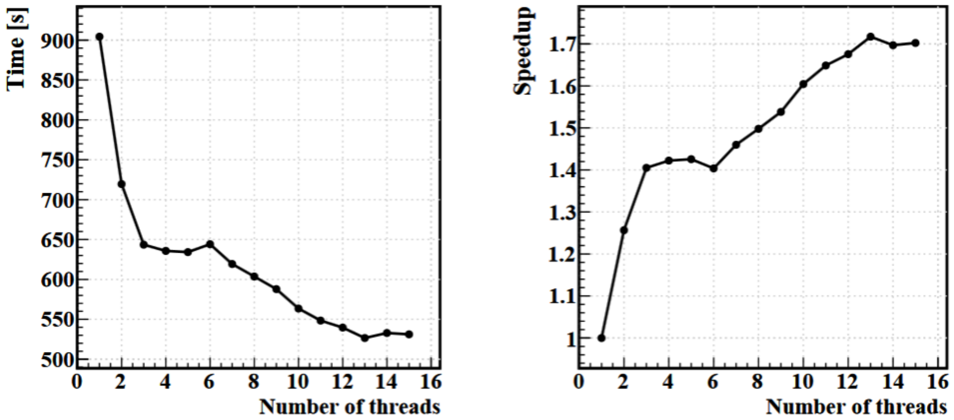
Testbench

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 with QGSM generator
1000-5000 events.
Macro run_sim_bmn.C

OpenMP parallelization

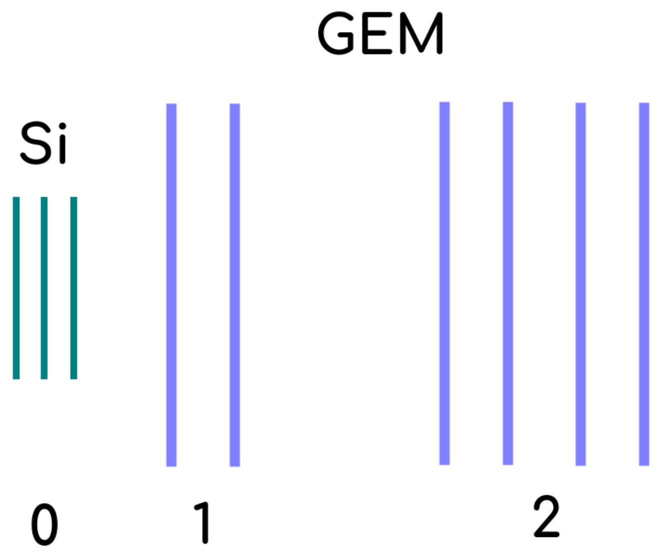


Scalability of the BmnRoot simulation parallelized with OpenMP

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);
…
```

Optimization of the BmnRoot tracks reconstruction modules



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 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.

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

Optimization of the BmnRoot tracks reconstruction modules

Compiler (GCC) optimization

DEBUG➔RELEASE
(O2 level optimization)

Tracking parameters selection

Before optimization
Monte Carlo 1 sec/event
Experimental 6 sec/event
One file (200 000 event) 2 weeks

After optimization
Monte Carlo 0.3 sec/event
Experimental 0.7 sec/event
One file (200 000 event) 39 hours

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 tracks reconstruction modules

1. Static or/and dynamic hotspot analysis of the BmnRoot reconstruction modules.
2. Hotspots localization.
3. Performance optimization (algorithmic and parallelization of most time-consuming hotspots).
4. Tests of correctness and scalability of optimized code.

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

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

Analysis summary

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

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

Details of CPU Time Consumption

Si+GEM Track Finder: 45%
Global Matching: 21%
Vertex Finder: 19%

Optimization of the BmnRoot tracks reconstruction modules

Small code improvements

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

Small effect (CPU time reduction by percent's)

```
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

Algorithmic optimization of BmnFieldMap

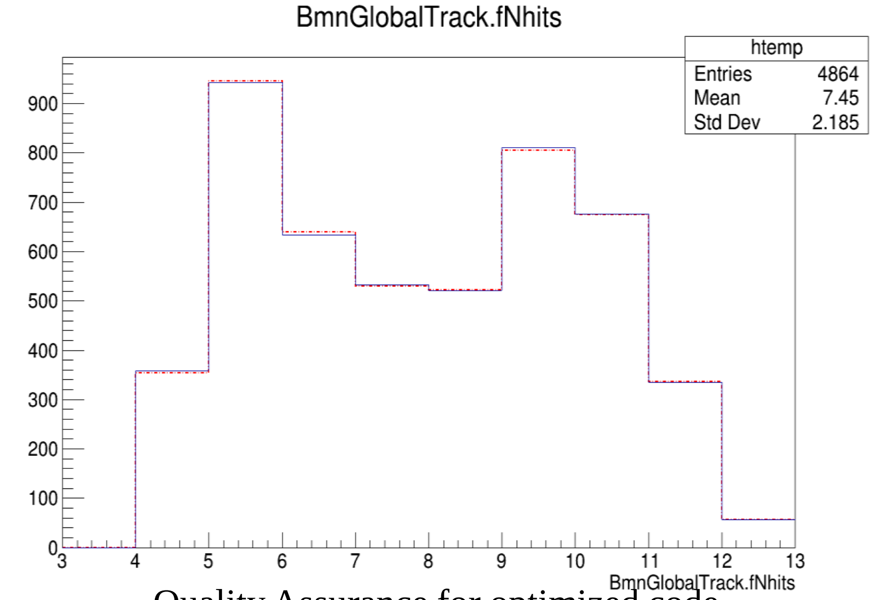
1. Replacement of linear-piecewise interpolation by constant-piecewise interpolation. Calculation for 8 vertices of cube elementary cell is not necessary.
2. Interface of existing classes must be preserved.

Tests of correctness are required

Optimization of the BmnRoot tracks 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. Number of reconstructed hits

- ✓ Build in Debug mode (without compiler optimization) reduced total execution time by 10%.
- ✓ Build in O2-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 small difference between non-optimized and optimized results (see figure).

Optimization of the BmnRoot tracks reconstruction modules

Track finders OpenMP parallelization

```
BmnInnerTrackingRun7::FindTracks_4of4_OnLastGEMStations() {
//Fit of dX vs X for different stations {p0, p1, sigma} (from qgsm simulation)    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);
...
```

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.

Thank you for your kind attention!