High-performance optimization of event simulation and track reconstruction software in the BM@N NICA experiment

NEMNYUGIN S.A., MERTS S.P., YUFRYAKOVA A.A., DRIUK A.V., ROUDNEV V.A., STEPANOVA M.M.

SAINT-PETERSBURG STATE UNIVERSITY

JOINT INSTITUTE OF NUCLEAR RESEARCH
Outline

• Baryon Matter at Nuclotron (BM@N) experiment.
• BmnRoot framework.
• Performance study and performance bottlenecks of the BmnRoot simulation modules.
• Optimization of the BmnRoot simulation modules.
• Performance bottlenecks of the BmnRoot tracks reconstruction modules.
• 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 - Nuclotron based Ion Collider fAcility, Joint Institute for Nuclear Research, Dubna.

BM@N – Baryon Matter at Nuclotron experiment. Heavy ion collisions with fixed targets.
Beam: Ar, Kr, C, d, energy 2.9 – 4.5 GeV.
Target: Al, Cu, C, Pb, Sn, H2 (liquid) etc.
Setup includes detector subsystems, magnet, electronics.
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 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
Performance problems of the BmnRoot simulation modules

- Dynamic hotspot analysis of the BmnRoot simulation modules is most reasonable (various possible execution paths).
- Hotspots localization is performed.

**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 DQGSM generator
1000-5000 events.
Macro run_sim_bmn.C

### Hotspots of the BmnRoot simulation modules

<table>
<thead>
<tr>
<th>Function / Call Stack</th>
<th>CPU Time</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRandom::Gaus</td>
<td>145.867s</td>
<td>libMathCore.so.6.16.00</td>
</tr>
<tr>
<td>DeadZoneOfStripLayer::IsInside</td>
<td>137.187s</td>
<td>libGen.so.0</td>
</tr>
<tr>
<td>__cos_fma</td>
<td>98.153s</td>
<td>libm.so.6</td>
</tr>
<tr>
<td>TRandom3::Rndm</td>
<td>80.814s</td>
<td>libMathCore.so.6.16.00</td>
</tr>
<tr>
<td>__sin_fma</td>
<td>77.566s</td>
<td>libm.so.6</td>
</tr>
<tr>
<td>deflate</td>
<td>74.185s</td>
<td>libz.so.1</td>
</tr>
<tr>
<td>FairMCApplication::Stepping</td>
<td>67.573s</td>
<td>libBase.so.18.2.0</td>
</tr>
<tr>
<td>BmnGemStripModule::AddRealPointPhi</td>
<td>46.391s</td>
<td>libGem.so.0</td>
</tr>
<tr>
<td>BmnGemStripLayer::ConvertPointToSi</td>
<td>43.867s</td>
<td>libGem.so.0</td>
</tr>
<tr>
<td>std::map&lt;std::pair&lt;int, int&gt;, int, std::less&lt;int&gt;&gt;</td>
<td>41.465s</td>
<td>libBmnData.so.0</td>
</tr>
<tr>
<td>StripCluster::AddStrip</td>
<td>41.270s</td>
<td>libGem.so.0</td>
</tr>
<tr>
<td>BmnGemStripLayer::IsPointInsideStrip</td>
<td>31.419s</td>
<td>libGem.so.0</td>
</tr>
<tr>
<td>BmnGemStripLayer::ConvertNormalFrom</td>
<td>27.492s</td>
<td>libGem.so.0</td>
</tr>
<tr>
<td>std::map&lt;std::pair&lt;int, int&gt;, int, std::less&lt;int&gt;&gt;</td>
<td>20.336s</td>
<td>libBmnData.so.0</td>
</tr>
<tr>
<td>std::operator&lt;int, int&gt;</td>
<td>18.805s</td>
<td>libBmnData.so.0</td>
</tr>
<tr>
<td>BmnGemStripLayer::IsPointInsideDead</td>
<td>18.630s</td>
<td>libGem.so.0</td>
</tr>
<tr>
<td>BmnNewFieldMap::FieldInterpolate</td>
<td>18.493s</td>
<td>libBmnField.so.0</td>
</tr>
<tr>
<td>std::_Rb_tree_iterator&lt;std::pair&lt;int, int&gt;&gt;</td>
<td>16.267s</td>
<td>libBase.so.18.2.0</td>
</tr>
<tr>
<td>TArrayF::At</td>
<td>14.695s</td>
<td>libBmnField.so.0</td>
</tr>
<tr>
<td>BmnNewFieldMap::IsInside</td>
<td>13.857s</td>
<td>libBmnField.so.0</td>
</tr>
<tr>
<td>std::map&lt;int, bool, std::less&lt;int&gt;, std::auto&gt;</td>
<td>12.740s</td>
<td>libBmnData.so.0</td>
</tr>
<tr>
<td>BmnFieldMap::Interpolate</td>
<td>12.701s</td>
<td>libBmnField.so.0</td>
</tr>
</tbody>
</table>
Optimization of the BmnRoot simulation modules

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

OpenMP parallelization

```
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);

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

Scalability of the BmnRoot simulation parallelized with OpenMP

Quality Assurance for simulation

QA - 1 thread

QA - 4 thread
```
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 problems of the BmnRoot reconstruction modules

Very slow
✓ 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

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.

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

Hotspots of the BmnRoot reconstruction modules

<table>
<thead>
<tr>
<th>Function / Call Stack</th>
<th>CPU Time</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock</td>
<td>66.450s</td>
<td>libc.so.6</td>
</tr>
<tr>
<td>BmnKalmanFilter::RK4Order</td>
<td>34.481s</td>
<td>libBmnData.so.0.0.0</td>
</tr>
<tr>
<td>BmnNewFieldMap::FieldInterpolate</td>
<td>23.124s</td>
<td>libBmnField.so.0.0.0</td>
</tr>
<tr>
<td>TArrayF::At</td>
<td>22.950s</td>
<td>libBmnField.so.0.0.0</td>
</tr>
<tr>
<td>inflate</td>
<td>20.673s</td>
<td>libz.so.1</td>
</tr>
<tr>
<td>BmnKalmanFilter::TransportC</td>
<td>17.638s</td>
<td>libBmnData.so.0.0.0</td>
</tr>
<tr>
<td>BmnNewFieldMap::IsInside</td>
<td>15.992s</td>
<td>libBmnField.so.0.0.0</td>
</tr>
<tr>
<td>TArray::BoundsOk</td>
<td>15.566s</td>
<td>libBmnData.so.0.0.0</td>
</tr>
<tr>
<td>BmnFieldMap::Interpolate</td>
<td>15.226s</td>
<td>libBmnField.so.0.0.0</td>
</tr>
<tr>
<td>std::vector&lt;double, std::allocator&lt;double&gt;</td>
<td>13.862s</td>
<td>libBmnData.so.0.0.0</td>
</tr>
<tr>
<td>operator new</td>
<td>12.704s</td>
<td>libstdc++.so.6</td>
</tr>
<tr>
<td>std::vector&lt;double, std::allocator&lt;double&gt;</td>
<td>12.222s</td>
<td>libBmnData.so.0.0.0</td>
</tr>
<tr>
<td>BmnKalmanFilter::RK4TrackExtrapolate</td>
<td>11.896s</td>
<td>libBmnData.so.0.0.0</td>
</tr>
<tr>
<td>std::vector&lt;double, std::allocator&lt;double&gt;</td>
<td>11.128s</td>
<td>libBmnData.so.0.0.0</td>
</tr>
<tr>
<td>TGeoVoxelFinder::GetNextCandidates</td>
<td>10.982s</td>
<td>libGeom.so.6.16</td>
</tr>
<tr>
<td>_pow</td>
<td>10.944s</td>
<td>libm.so.6</td>
</tr>
<tr>
<td>std::__fill_n_a&lt;double*, unsigned long</td>
<td>10.074s</td>
<td>libBmnData.so.0.0.0</td>
</tr>
<tr>
<td>TGeoVoxelFinder::GetCheckList</td>
<td>9.832s</td>
<td>libGeom.so.6.16</td>
</tr>
<tr>
<td><em>GI</em></td>
<td>8.701s</td>
<td>libc.so.6</td>
</tr>
<tr>
<td>std::vector&lt;double, std::allocator&lt;double&gt;</td>
<td>8.420s</td>
<td>libBmnData.so.0.0.0</td>
</tr>
<tr>
<td>std::vector&lt;BmnLink, std::allocator&lt;Bn&gt;</td>
<td>7.352s</td>
<td>libSilicon.so.0.0.0</td>
</tr>
<tr>
<td>TGeoNavigator::SearchNode</td>
<td>6.766s</td>
<td>libGeom.so.6.16</td>
</tr>
</tbody>
</table>
Optimization of the BmnRoot reconstruction modules

Compiler (GCC) optimization

DEBUG ➔ RELEASE
(O2 level optimization)

Tracking parameters selection

<table>
<thead>
<tr>
<th>Before optimization</th>
<th>After optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monte Carlo 1 sec/event</td>
<td>Monte Carlo 0.3 sec/event</td>
</tr>
<tr>
<td>Experimental 6 sec/event</td>
<td>Experimental 0.7 sec/event</td>
</tr>
<tr>
<td>One file (200 000 events) 2 weeks</td>
<td>One file (200 000 events) 39 hours</td>
</tr>
</tbody>
</table>

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)
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.

\[ \Delta x = 0.25 \text{ cm} \]
\[ \Delta y = 0.45 \text{ cm} \]
\[ \Delta z = 0.17 \text{ cm} \]
Optimization of the BmnRoot reconstruction modules

Double_t BmnNewFieldMap::FieldInterpolate(TH2zayF* 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 llx = 0;
    Int_t lly = 0;
    Int_t llz = 0;

    if (IsInside(x, y, z, ix, iy, iz, dx, dy, dz)) {
        llx = Int_t(Mint((x - fmin) / fstep));
        lly = Int_t(Mint((y - fmin) / fstep));
        llz = Int_t(Mint((z - fmin) / fstep));

        Hh = fcomp->At(llx * fNy + lly * fNx + lly * fNx + llx);
        /**fHa[1][1][0] = fcomp->At((x + 1) * fNy + fNx + iy + fNx + iz);**/
        fHa[1][1][0] = fcomp->At((ix + 1) * fNy + fNx + iy + fNx + iz);
        fHa[1][1][0] = fcomp->At((ix + 1) * fNy + fNx + (iy + 1) * fNx + iz);
        fHa[1][1][0] = fcomp->At((ix + 1) * fNy + fNx + (iy + 1) * fNx + (iz + 1));
        fHa[0][0][1] = fcomp->At(ix * fNy + iy * fNx + fx + (iz + 1));
        fHa[0][0][1] = fcomp->At((ix + 1) * fNy + fNx + iy * fNx + (iz + 1));
        fHa[0][0][1] = fcomp->At((ix + 1) * fNy + fNx + iy * fNx + fNx + (iz + 1));
        fHa[0][1][1] = fcomp->At(fy * fNx + (iy + 1) * fNx + fNx + (iz + 1));
        fHa[0][1][1] = fcomp->At(fy * fNx + (iy + 1) * fNx + fNx + fNx + (iz + 1));
        fHa[1][1][1] = fcomp->At((x + 1) * fNy + fNx + (iy + 1) * fNx + fNx + fNx + (iz + 1));

        return Interpolate(dk, dy, dz);
    }
    return 0.;
}

Optimized code of FieldInterpolate method of BmnNewFieldMap class.

✓ 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.
Optimization 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);
        ...
    }
}

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.
Conclusion

• Performance studies are performed and performance bottlenecks of the BmnRoot simulation and reconstruction modules are revealed.

• Proposed method of optimization of the BmnRoot simulation modules displays good scalability on multicore workstation.

• Performance bottlenecks of the BmnRoot tracks 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 for attention!