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
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** - Nuclotron based Ion Collider fAcility, Joint Institute for Nuclear Research, Dubna, Russian Federation.
- **BM@N** – Baryon Matter at Nuclotron 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.
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.

<table>
<thead>
<tr>
<th>Function / Call Stack</th>
<th>CPU Time</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRandom::Gaus</td>
<td>137.187s</td>
<td>libG4Core.so.6.11.0.0</td>
</tr>
<tr>
<td>_deadZone0StripLayer_Inside</td>
<td>90.153s</td>
<td>libG4Core.so.6.11.0.0</td>
</tr>
<tr>
<td>TRandom3::Rndm</td>
<td>80.814s</td>
<td>libG4Core.so.6.11.0.0</td>
</tr>
<tr>
<td><em>sim</em>.fma</td>
<td>77.506s</td>
<td>libG4Core.so.6.11.0.0</td>
</tr>
<tr>
<td>_deflate</td>
<td>74.185s</td>
<td>libG4Core.so.6.11.0.0</td>
</tr>
<tr>
<td>FairMCApplication::Stepping</td>
<td>67.573s</td>
<td>libG4Core.so.18.1.2.0</td>
</tr>
<tr>
<td>BmnGemStripModule::AddRealPointF</td>
<td>46.392s</td>
<td>libG4Core.so.6.11.0.0</td>
</tr>
<tr>
<td>BmnGemStripLayer::ConvertPointToGsi</td>
<td>41.570s</td>
<td>libG4Core.so.6.11.0.0</td>
</tr>
<tr>
<td>std::map&lt;std::pair&lt;int, int&gt;, std::list&lt;...&gt;</td>
<td>41.495s</td>
<td>libG4Core.so.6.11.0.0</td>
</tr>
<tr>
<td>StripCluster::AddStrip</td>
<td>41.276s</td>
<td>libG4Core.so.6.11.0.0</td>
</tr>
<tr>
<td>BmnGemStripLayer::IsPointInsideStrip</td>
<td>31.413s</td>
<td>libG4Core.so.6.11.0.0</td>
</tr>
<tr>
<td>BmnGemStripLayer::ConvertNormalPt</td>
<td>27.492s</td>
<td>libG4Core.so.6.11.0.0</td>
</tr>
<tr>
<td>std::map&lt;std::pair&lt;int, int&gt;, std::list&lt;...&gt;</td>
<td>20.330s</td>
<td>libG4Core.so.6.11.0.0</td>
</tr>
<tr>
<td>std::vector&lt;int, int&gt;</td>
<td>18.805s</td>
<td>libG4Core.so.6.11.0.0</td>
</tr>
<tr>
<td>BmnGemStripLayer::IsPointInsideDea</td>
<td>18.630s</td>
<td>libG4Core.so.6.11.0.0</td>
</tr>
<tr>
<td>BmnNewFieldMap::FieldInterpolate</td>
<td>18.403s</td>
<td>libG4Field.so.0.0</td>
</tr>
<tr>
<td>std::vector&lt;int, int&gt;</td>
<td>16.207s</td>
<td>libG4Field.so.0.0</td>
</tr>
<tr>
<td>TAmgF::At</td>
<td>14.685s</td>
<td>libG4Field.so.0.0</td>
</tr>
<tr>
<td>BmnNewFieldMap::Isotrope</td>
<td>13.857s</td>
<td>libG4Field.so.0.0</td>
</tr>
<tr>
<td>std::map&lt;int, bool, std::less&lt;int&gt;, std::stable�</td>
<td>12.740s</td>
<td>libG4Field.so.0.0</td>
</tr>
<tr>
<td>BmnFieldMap::interpolate</td>
<td>12.701s</td>
<td>libG4Field.so.0.0</td>
</tr>
</tbody>
</table>

Testbench

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

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

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

GEM – Gas Electron Multiplier.
SI – Silicon detector.

Si

GEM

0 1 2
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 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.

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)

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

Example of the code

```cpp
Double_t BmnFieldMap::Interpolate(Double_t dx, Double_t dy, Double_t dz) {
    /**// Interpolate in x coordinate
    fHx[0][0] = fHa[0][0][0] + (fHa[1][0][0] - fHa[0][0][0]) * dx;
    fHx[1][0] = fHa[0][1][0] + (fHa[1][1][0] - fHa[0][1][0]) * dx;
    fHx[0][1] = fHa[0][0][1] + (fHa[1][0][1] - fHa[0][0][1]) * dx;
    fHx[1][1] = fHa[0][1][1] + (fHa[1][1][1] - fHa[0][1][1]) * dx;

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

    // Interpolate in z coordinate
    return fHz[0] + (fHz[1] - fHz[0]) * dz; // NEW
}
```
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 iizz = 0;

  if (IsInside(x, y, z, iix, iiy, iizz, dx, dy, dz)) {
    iix = Int_t(Nint((x - xMin) / xStep));
    iiy = Int_t(Nint((y - yMin) / yStep));
    iizz = Int_t(Nint((z - zMin) / zStep));
    Nh = fcomp->At(iix * fNy + iiy * fNz + iizz);

    //**fNz[i][i][i] = fcomp->At(iix * fNy + iiy * fNz + iy + fNz + iz);**/
    fNz[0][i][i] = fcomp->At(iix * fNy + iiy * fNz + iy + fNz + iz);
    fNz[1][i][i] = fcomp->At(iix * fNy + iiy * fNz + iy + fNz + (iz + 1));
    fNz[0][i][1] = fcomp->At(iix * fNy + iiy * fNz + iy + fNz + (iz + 1));
    fNz[1][i][1] = fcomp->At(iix * fNy + iiy * fNz + (iy + 1) + fNz + (iz + 1));
    fNz[1][1][1] = fcomp->At(iix * fNy + (iy + 1) + fNz + (iz + 1));
    return Interpolate(dx, dy, dz);
  }
  return 0.;
}

Optimized code of FieldInterpolate method of BmnNewFieldMap class.

- 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!