Software contribution from SPbU: algorithmic and code optimization of the BmnRoot framework

S Nemnyugin, S Merts, A Driuk, A Iufryakova , N Kakhanovskaya, A Myasnikov

Saint-Petersburg State University

Joint Institute of Nuclear Research

Outline

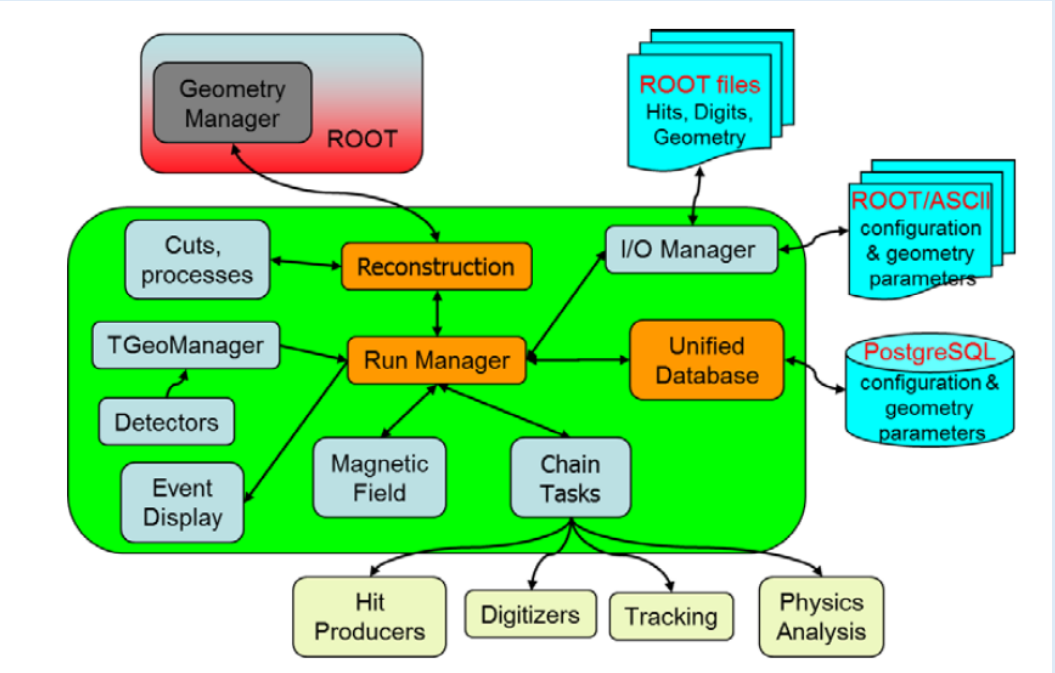
- Performance bottlenecks of the BmnRoot package
- PROOF in the BmnRoot
- Search and removal of memory errors in the BmnRoot package
- Optimization of the vertex finding algorithms
- Adding triggers description into geometry of SRC experiment

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

Performance bottlenecks of the BmnRoot package

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.

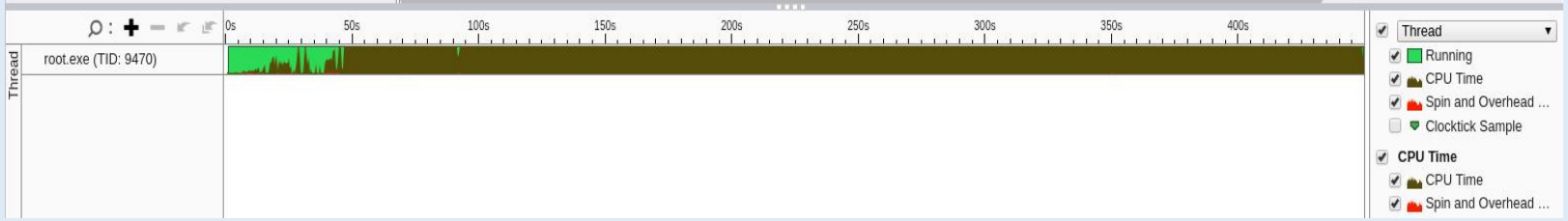


BmnRoot reconstruction modules

Hotspots of reconstruction in the BM@N experiment

Grouping: Function / Call Stack

Function / Call Stack	CPU Time ▼	Instructions Retired	Microarchitecture Usage	Module	Function (Full)
BmnDchTrackFinder::PrepareArraysToProcessEvent	61.724s	407,343,200,000	61.8%	libDch.so.0.0.0	BmnDchTrackFinder::PrepareArraysToProcessEvent(void)
BmnKalmanFilter::RK4Order	16.108s	68,686,200,000	42.5%	libBmnData.so.0.0.0	BmnKalmanFilter::RK4Order(std::vector<double, std::allocator<double>> const&, doub
BmnNewFieldMap::FieldInterpolate	11.925s	68,866,600,000	59.4%	libBmnField.so.0.0.0	BmnNewFieldMap::FieldInterpolate(TArrayF*, double, double, double)
TArrayF::At	10.234s	57,723,600,000	56.1%	libBmnField.so.0.0.0	TArrayF::At(int) const
BmnNewFieldMap::IsInside	8.900s	51,275,400,000	58.8%	libBmnField.so.0.0.0	BmnNewFieldMap::IsInside(double, double, double, int&, int&, int&, double&, double&,
BmnFieldMap::Interpolate	8.542s	40,937,600,000	51.6%	libBmnField.so.0.0.0	BmnFieldMap::Interpolate(double, double, double)
std::__fill_a1<double*, double>	8.381s	17,703,400,000	22.6%	libBmnData.so.0.0.0	__gnu_cxx::__enable_if<std::__is_scalar<double>::__value, void>::__type std::__fill_a
BmnKalmanFilter::TransportC	8.333s	36,082,200,000	46.0%	libBmnData.so.0.0.0	BmnKalmanFilter::TransportC(std::vector<double, std::allocator<double>> const&, std:
TArray::BoundsOk	7.689s	43,945,000,000	58.2%	libBmnData.so.0.0.0	TArray::BoundsOk(char const*, int) const
std::vector<double, std::allocator<double>>::operator[]	6.623s	30,780,200,000	50.1%	libBmnData.so.0.0.0	std::vector<double, std::allocator<double>>::operator[](unsigned long) const
fit_seg	6.140s	37,470,400,000	59.3%	libBmnData.so.0.0.0	fit_seg(double*, double*, double*, double*, int, int)
BmnMwpcHitFinder::SegmentFinder	5.414s	37,802,600,000	70.6%	libMwpc.so.0.0.0	BmnMwpcHitFinder::SegmentFinder(int, int**, double***, int***, int*, double***, double*
std::vector<double, std::allocator<double>>::operator[]	4.577s	20,633,800,000	51.8%	libBmnData.so.0.0.0	std::vector<double, std::allocator<double>>::operator[](unsigned long)
BmnKalmanFilter::RK4TrackExtrapolate	4.560s	19,564,600,000	45.2%	libBmnData.so.0.0.0	BmnKalmanFilter::RK4TrackExtrapolate(FairTrackParam*, double, std::vector<double,
TGeoVoxelFinder::GetCheckList	3.967s	14,449,600,000	33.4%	libGeom.so.6.16.00	TGeoVoxelFinder::GetCheckList(double const*, int&, TGeoStateInfo&)
__libc_malloc	3.582s	14,027,200,000	40.6%	libc-2.31.so	__libc_malloc
BmnKalmanFilter::CalcOut	3.453s	13,083,400,000	38.7%	libBmnData.so.0.0.0	BmnKalmanFilter::CalcOut(double, double const*)
TGeoVoxelFinder::GetNextCandidates	3.341s	12,909,600,000	31.0%	libGeom.so.6.16.00	TGeoVoxelFinder::GetNextCandidates(double const*, int&, TGeoStateInfo&)
BmnMaterialEffects::CalcThetaSq	2.605s	7,917,800,000	36.1%	libBmnData.so.0.0.0	BmnMaterialEffects::CalcThetaSq(FairTrackParam const*, BmnMaterialInfo const*) co
std::vector<BmnLink, std::allocator<BmnLink>>::vector	2.290s	8,852,800,000	48.5%	libSilicon.so.0.0.0	std::vector<BmnLink, std::allocator<BmnLink>>::vector(std::vector<BmnLink, std::alloc
BmnDchTrackFinder::BuildUVSegments	2.285s	12,949,200,000	49.6%	libDch.so.0.0.0	BmnDchTrackFinder::BuildUVSegments(int, int, int, int, int, int, int, int, int, double**, do
TObject::TObject	2.226s	9,352,200,000	42.3%	libBmnData.so.0.0.0	TObject::TObject(TObject const&)



Experimental data:

- Run 4665.7.
- Argon beam (3.2 GeV).
- Aluminium target.

List of most significant hotspots:

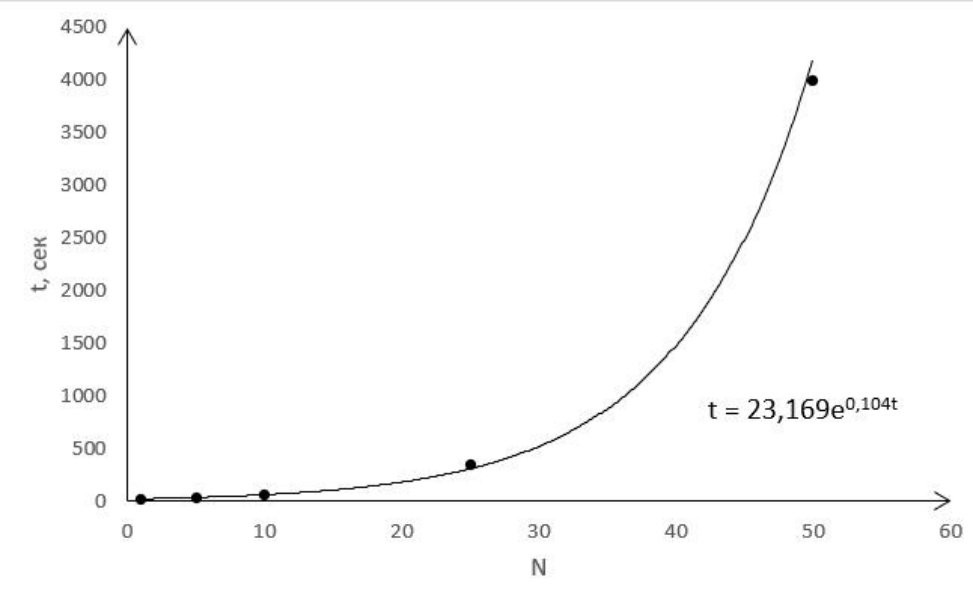
- BmnDchTrackFinder::PrepareArraysToProceedEvent
- BmnKalmanFilter::RK4Order
- BmnNewFieldMap::FieldInterpolate
- BmnNewFieldMap::IsInside
- BmnFieldMap::Interpolate

"Hotspot" is part of a program which consumes significant CPU time.
Hotspots should be optimized first of all.
List of hotspots depends on the input data, because a running program is parametrized DAG.

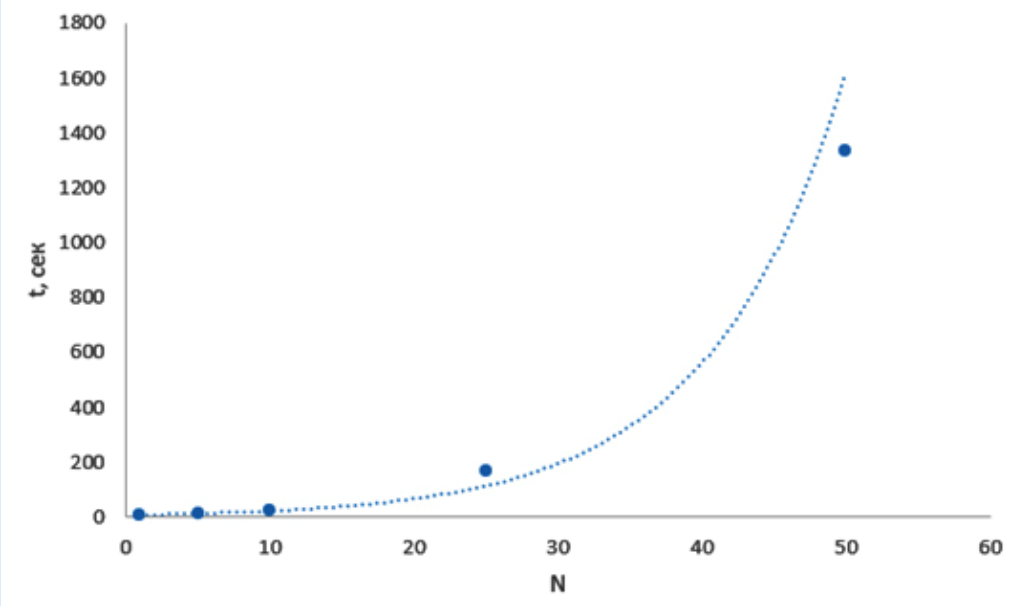
Questions:

1. What should be optimized in the code (if possible)?
2. Which features of experimental data define reconstruction hotspots?

Time of reconstruction



a)



b)

Monte Carlo simulated data.
BOX generator. N – multiplicity.
Two builds: a) Debug and b) Release

Most significant reconstruction hotspots. Multiplicity N = 1

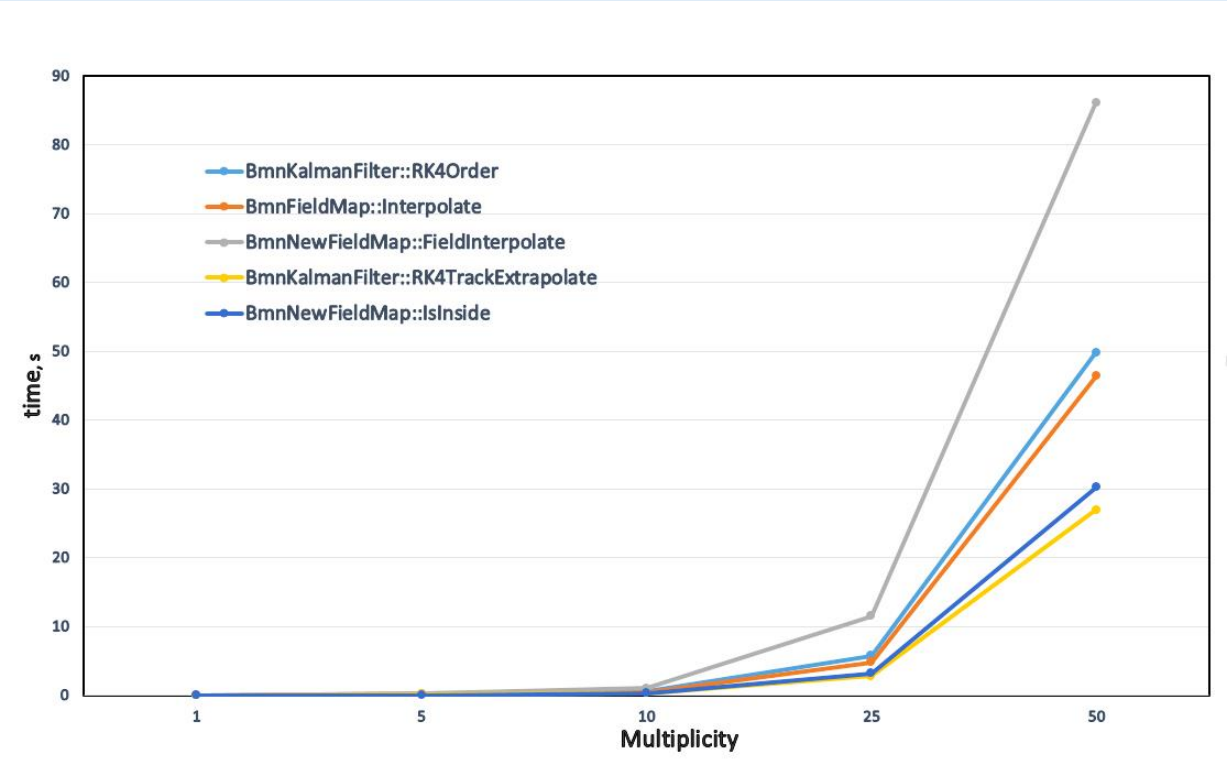
Function / Call Stack	CPU Time ▼	Module	Function (Full)
inflate	847.861ms	libz.so.1	inflate
__dlopen	660.839ms	libdl.so.2	__dlopen
std::vector<BmnMatch, std::allocator<BmnMatc	607.963ms	libSilicon.so.0.0.0	std::vector<BmnMatch, std::allocator<BmnMatch>>::_M_fill_insert(__
BmnMatch::~BmnMatch	364.048ms	libBmnData.so.0.0.0	BmnMatch::~BmnMatch(void)
TObject::~TObject	216.003ms	libCore.so.6.16	TObject::~TObject(void)
memcmp	187.982ms	libc-dynamic.so	memcmp
OS_BARESYSCALL_DoCallAsmIntel64Linux	118.758ms	libc-dynamic.so	OS_BARESYSCALL_DoCallAsmIntel64Linux
operator new	105.981ms	libstdc++.so.6	operator new(unsigned long)
TBranch::GetEntry	105.966ms	libTree.so.6.16	TBranch::GetEntry(long long, int)
TBranch::FillImpl	103.987ms	libTree.so.6.16	TBranch::FillImpl(ROOT::Internal::TBranchIMTHelper*)
clang::Lexer::LexTokenInternal	100.011ms	libCling.so	clang::Lexer::LexTokenInternal(clang::Token&, bool)
BmnGemStripLayer::ResetStripMatches	96.025ms	libGem.so.0.0.0	BmnGemStripLayer::ResetStripMatches(void)
BmnGemStripLayer::ResetStripDigitNumberMat	79.959ms	libGem.so.0.0.0	BmnGemStripLayer::ResetStripDigitNumberMatches(void)
BmnSiliconLayer::ResetStripMatches	76.071ms	libSilicon.so.0.0.0	BmnSiliconLayer::ResetStripMatches(void)
llvm::BitstreamCursor::readRecord	64.005ms	libCling.so	llvm::BitstreamCursor::readRecord(unsigned int, llvm::SmallVectorIm
__memmove_avx_unaligned_erms	62.027ms	libc.so.6	__memmove_avx_unaligned_erms
memmove	61.952ms	libc-dynamic.so	memmove
operator new	60.069ms	libpin3dwarf.so	operator new(unsigned long)
clang::Lexer::LexIdentifier	56.022ms	libCling.so	clang::Lexer::LexIdentifier(clang::Token&, char const*)
TObjArray::GetAbsLast	56.010ms	libCore.so.6.16	TObjArray::GetAbsLast(void) const
deflate	51.969ms	libz.so.1	deflate
clang::RedeclarableTemplateDecl::loadLazySpe	51.841ms	libCling.so	clang::RedeclarableTemplateDecl::loadLazySpecializationsImpl(llvm:

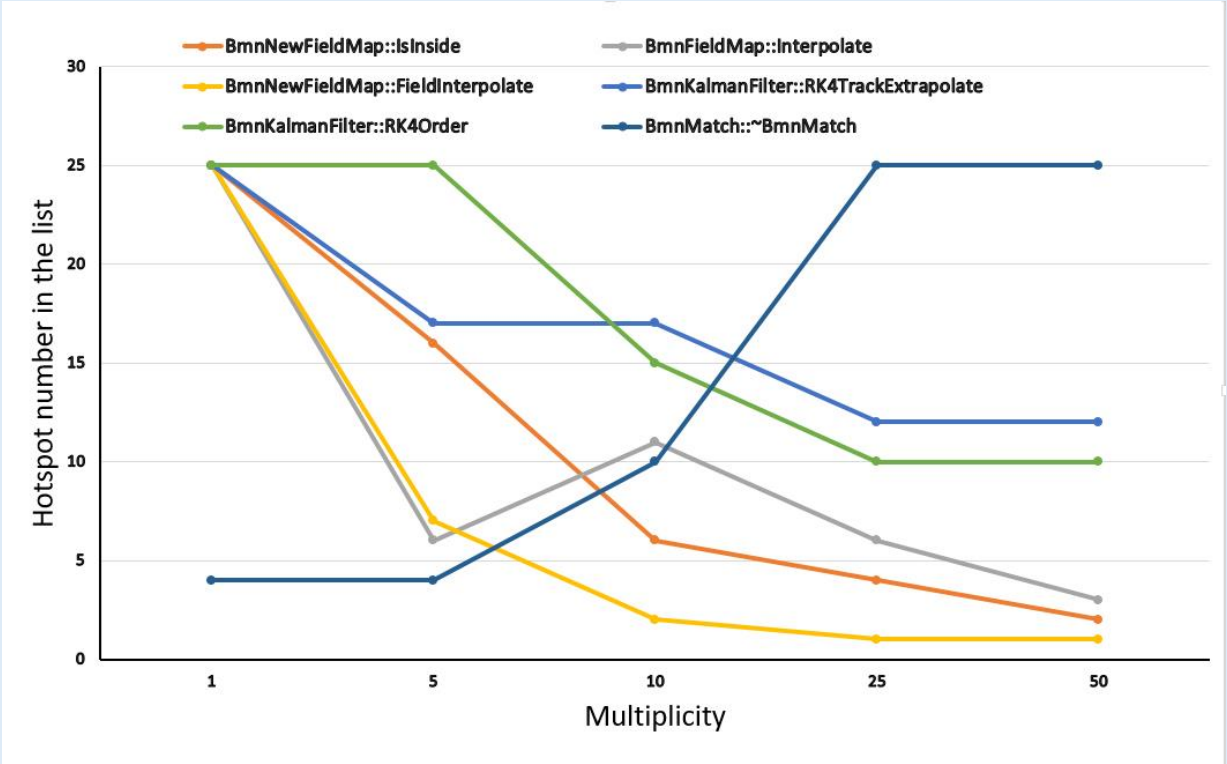
Dynamic hotspot analysis in the report is performed with Intel® VTune Amplifier 2020

Most significant reconstruction hotspots. Multiplicity N = 50

Function / Call Stack	CPU Time ▼	Module	Function (Full)
BmnNewFieldMap::FieldInterpolate	86.156s	libBmnField.so.0.0.0	BmnNewFieldMap::FieldInterpolate(TArrayF*, double, double, double
BmnKalmanFilter::RK4Order	49.820s	libBmnData.so.0.0.0	BmnKalmanFilter::RK4Order(std::vector<double, std::allocator<doubl
BmnFieldMap::Interpolate	46.439s	libBmnField.so.0.0.0	BmnFieldMap::Interpolate(double, double, double)
TGeoVoxelFinder::GetNextCandidates	44.739s	libGeom.so.6.16	TGeoVoxelFinder::GetNextCandidates(double const*, int&, TGeoStat
TGeoVoxelFinder::GetCheckList	43.731s	libGeom.so.6.16	TGeoVoxelFinder::GetCheckList(double const*, int&, TGeoStateInfo&
__pow	43.266s	libm.so.6	__pow
__GI_	38.839s	libc.so.6	float __GI_(long, int, bool, char)
TGeoShapeAssembly::DistFromOutside	35.199s	libGeom.so.6.16	TGeoShapeAssembly::DistFromOutside(double const*, double const
TGeoNavigator::SearchNode	32.411s	libGeom.so.6.16	TGeoNavigator::SearchNode(bool, TGeoNode const*)
BmnNewFieldMap::IsInside	30.296s	libBmnField.so.0.0.0	BmnNewFieldMap::IsInside(double, double, double, int&, int&, int&, d
operator new	27.874s	libstdc++.so.6	operator new(unsigned long)
BmnKalmanFilter::RK4TrackExtrapolate	26.987s	libBmnData.so.0.0.0	BmnKalmanFilter::RK4TrackExtrapolate(FairTrackParam*, double, st
TGeoVoxelFinder::GetIndices	26.436s	libGeom.so.6.16	TGeoVoxelFinder::GetIndices(double const*, TGeoStateInfo&)
TGeoVoxelFinder::SortCrossedVoxels	20.988s	libGeom.so.6.16	TGeoVoxelFinder::SortCrossedVoxels(double const*, double const*,
TObjArray::GetAbsLast	18.876s	libCore.so.6.16	TObjArray::GetAbsLast(void) const
BmnGeoNavigator::MakeStep	18.018s	libBmnData.so.0.0.0	BmnGeoNavigator::MakeStep(float) const
BmnMaterialEffects::CalcSigmaSqQp	17.069s	libBmnData.so.0.0.0	BmnMaterialEffects::CalcSigmaSqQp(FairTrackParam const*, BmnM
TGeoHMatrix::Multiply	16.986s	libGeom.so.6.16	TGeoHMatrix::Multiply(TGeoMatrix const*)
BmnKalmanFilter::TGeoTrackPropagate	15.234s	libBmnData.so.0.0.0	BmnKalmanFilter::TGeoTrackPropagate(FairTrackParam*, double, in
TGeoVoxelFinder::IsSafeVoxel	14.750s	libGeom.so.6.16	TGeoVoxelFinder::IsSafeVoxel(double const*, int, double) const
TGeoNode::MasterToLocal	14.454s	libGeom.so.6.16	TGeoNode::MasterToLocal(double const*, double*) const
TGeoShapeAssembly::Contains	14.294s	libGeom.so.6.16	TGeoShapeAssembly::Contains(double const*) const

Dependence of most significant reconstruction hotspots on multiplicity





Most significant hotspots for reconstruction of the Monte Carlo simulated data:

- BmnFieldMap
- BmnKalmanFilter
- BmnNewFieldMap

PROOF in the BmnRoot

Parallel ROOT Facility (**PROOF**) is a framework for parallel analysis of ROOT Ttrees.

PROOF is part of ROOT distribution, but it should be integrated into reconstruction macros to use with the BmnRoot package.

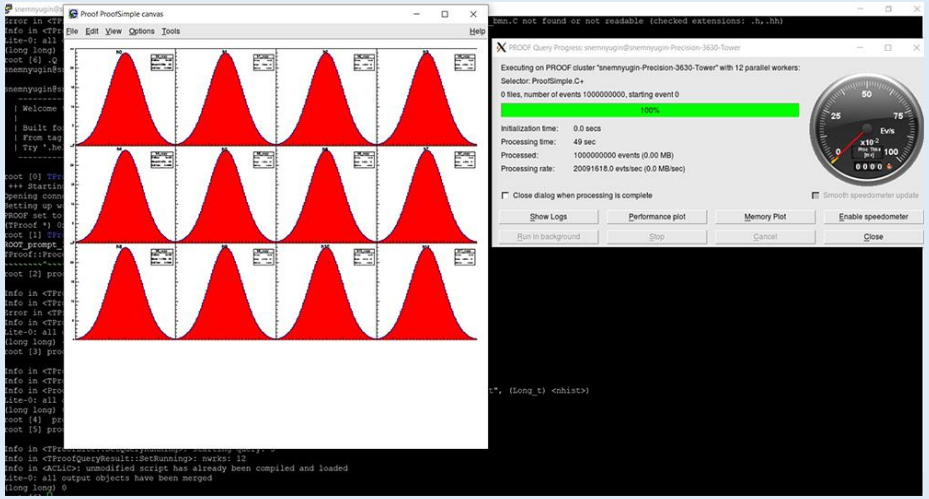
Work is in progress

PROOF is integrated into reconstruction macros with 2 kinds of execution:

1. parallel multi-core multi-threaded;
2. parallel cluster.

Classes of the BmnRoot inherited from FairTask are modified to remove segmentation fault errors.

Now some problems of incorrect serialization have to be solved.



Screenshot of the PROOF example.
Multithreaded execution.

Search and removal of memory errors in the BmnRoot package

BmnRoot is very complicated software package. Incorrect memory operations are unavoidable, but should be corrected.

Dynamic analysis with Valgrind 3.15.0  (https://valgrind.org).

Analysis configuration:

- OS Ubuntu 20.04.2 LTS x86_64;
- compiler gcc 9.3.

Modules of the BmnRoot which are used in reconstruction of simulated and experimental data (run_reco_src.C and run_reco_bmn.C) have been analyzed.

Memory leaks have been localized. Memory leaks are consequence of non-release of dynamically allocated memory both explicitly and implicitly. Some memory leaks are given in the Table (next slide).

Class with memory leak	Valgrind description of memory leak	Number of function calls with memory leak
BmnInnTrackerAlign.cxx	10,626 (24 direct, 10,602 indirect) bytes in 1 blocks	2
	384 (88 direct, 296 indirect) bytes in 1 blocks	2
	10,818 (24 direct, 10,794 indirect) bytes in 1 blocks	2
	8,000 bytes in 1 blocks are possibly lost	2
UniDbRun.cxx	4 bytes in 1 blocks	2
BmnMwpcHitFinder.cxx	1,008 bytes in 21 blocks	2
	128 bytes in 4 blocks	1
	64 bytes in 2 blocks	1
	32 bytes in 1 blocks	1
BmnFillDstTask.cxx	312 (144 direct, 168 indirect) bytes in 1 blocks	1
	24 bytes in 1 blocks	1
	16 bytes in 1 blocks	1
	8 bytes in 1 blocks	1
BmnFieldMap.cxx	24 bytes in 1 blocks	1
	48 (24 direct, 24 indirect) bytes in 1 blocks	1
BmnMwpcHitFinder.cxx	2,016 bytes in 42 blocks	2

Table. Examples of memory leaks

Examples of memory leaks are given for sample of 10 events (analysis is very time consuming: app. 7 sec without Valgrind and 330 sec with Valgrind).

Incorrect access to array elements have been localized. Example:

BmnSiliconTrackFinder.cxx

…

```
iModX[Sec_St]   = -1;
Xforglfit[Sec_St][iModX[Sec_St]]      = -999.;
```

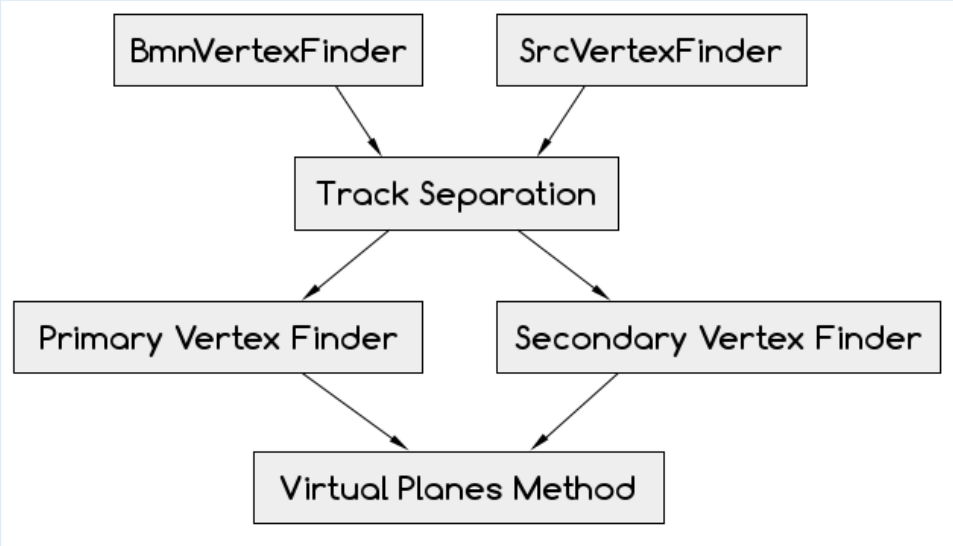
…

Work is in progress and in contact with Software Team of the BM@N experiment.

Optimization of the vertex finding algorithms

Finding of primary and secondary vertices

- ✓ Primary vertex finder is a part of standard event reconstruction chain.
- ✓ Secondary vertices are necessary for decays analysis.



Vertex finder algorithm

Track separation

Extrapolate all tracks to initial approximation of primary vertex position.
Check if track is in "beam region" or not and mark track with corresponding flag.

Primary vertex finder

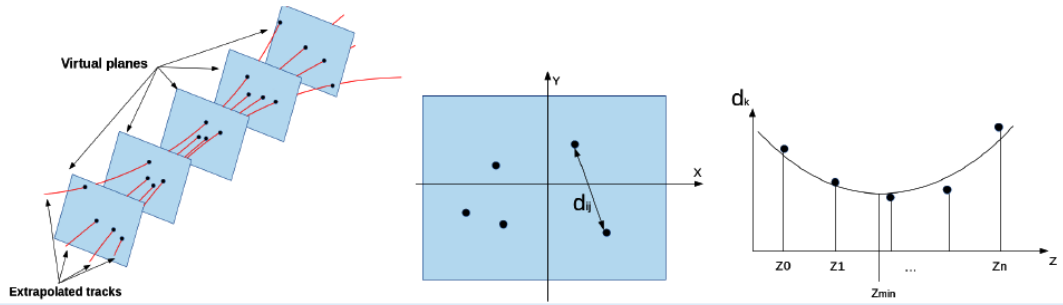
If there are less than 2 tracks marked as primary, return.
Reconstruct primary vertex for tracks marked as primary by virtual planes Method.
If Z position of found vertex is out of range (R), mark tracks as secondary and return.
Extrapolate tracks belonging this vertex to found Z_{PV} and calculate means for X and Y distributions (X_{PV} and Y_{PV})

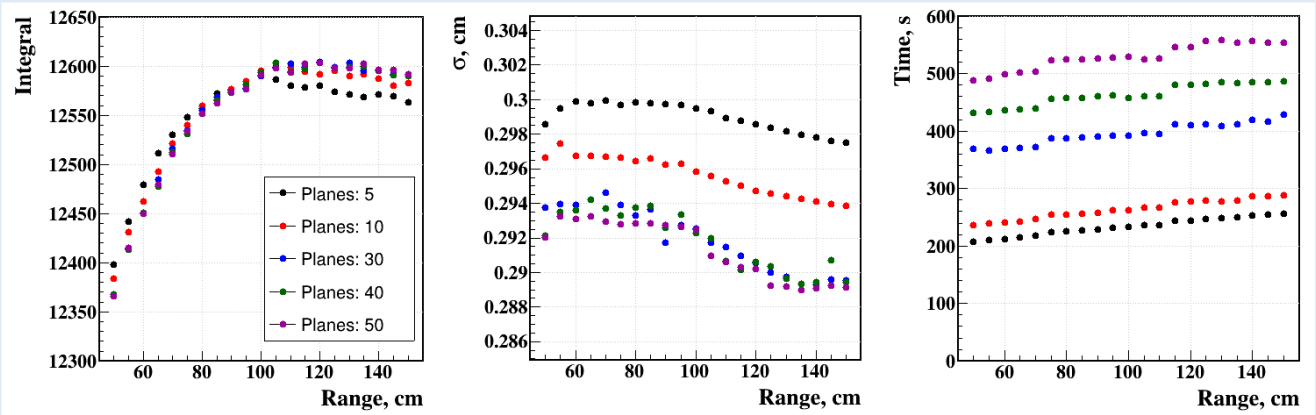
Secondary vertex finder

If there are less than 2 tracks marked as secondary, return
Reconstruct secondary vertex for tracks marked as secondary by virtual planes method in wide range
Extrapolate tracks belonging this vertex to found Z_{SV} and calculate mean for X and Y distributions (X_{SV} and Y_{SV})

Virtual planes method

1. Extrapolate reconstructed tracks to set of $\{z_k\}_0^{N_{planes}}$ by Kalman Filter around initial estimation: $Z_v^{init} - R < z_k < Z_v^{init} + R$
2. Calculate distance between each pair of points on plane k: $d_{ij}^k = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$
3. Calculate mean distance for each plane: $d^k = \sum d_{ij}^k / N_{pairs}$
4. Fit $d^k(z_k)$ by parabolic function and find z_{min}
5. Reduce R by factor speed: $R = R/speed$
6. Repeat 1-5 until required accuracy is achieved





Optimization of the algorithm

Input parameters

Range to search primary vertex in (Range).
Number of virtual planes (Planes).
Range reduction rate (Speed).

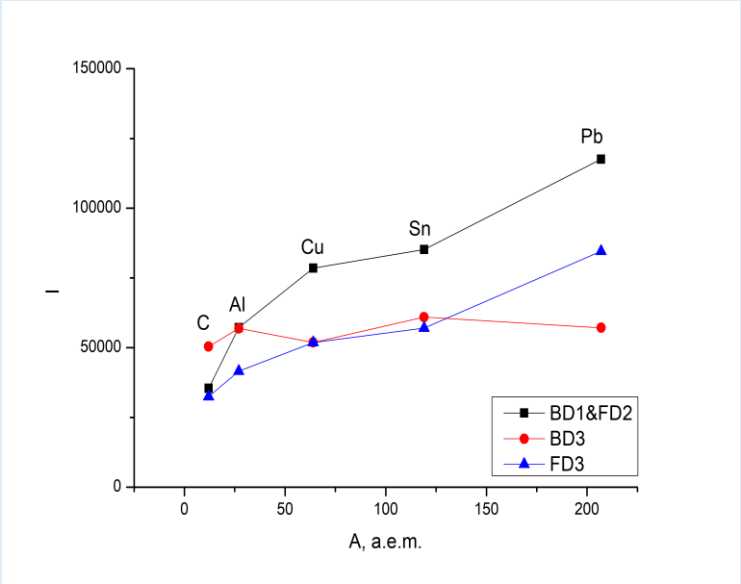
Control parameters

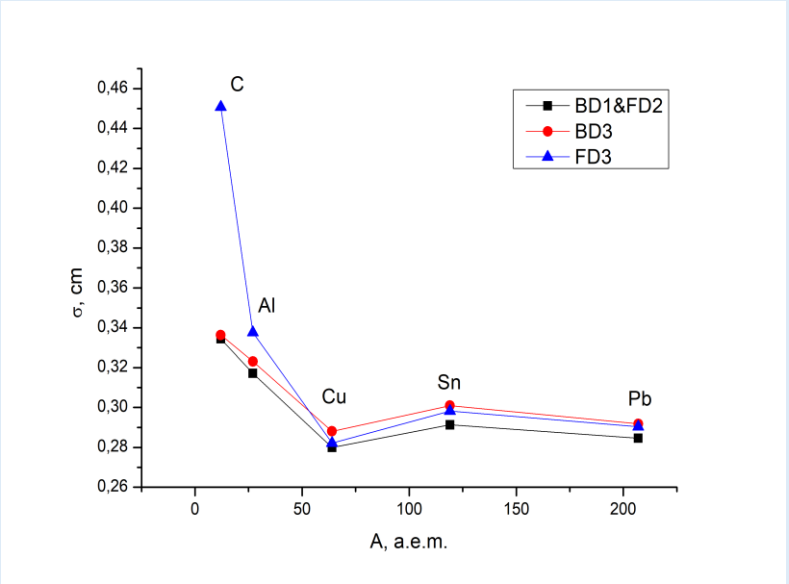
Number of found vertexes in -3cm < z < 3cm (Integral).
Width of Gaussian fit.
Work time (Time).

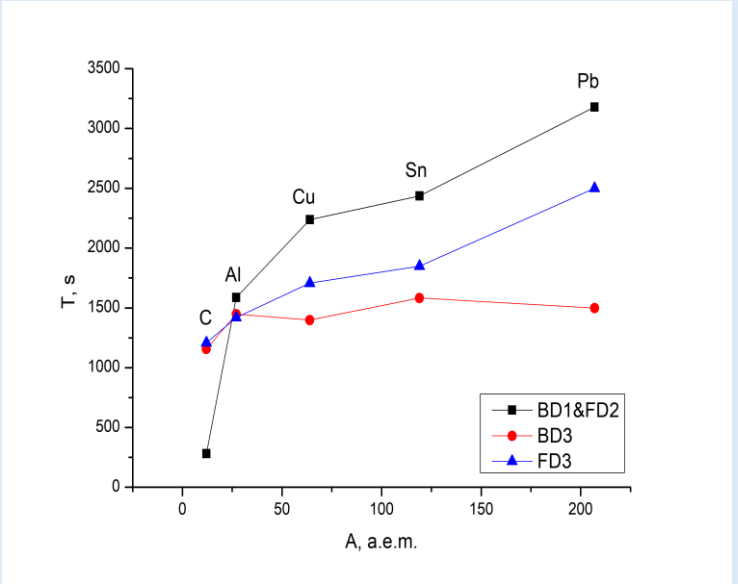
Main idea

Scan algorithm over input parameters to maximize Integral and minimize and Time.

Output parameters dependencies on number of virtual planes and search range for range reducing time 1.5
Sample: 10^6 events of Ar+Sn (BD > 3)

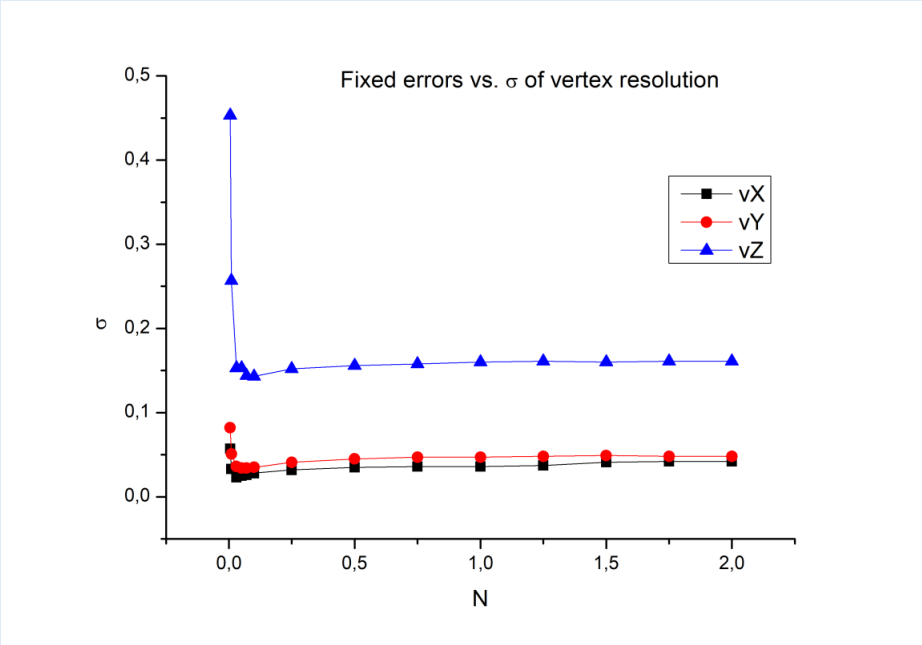


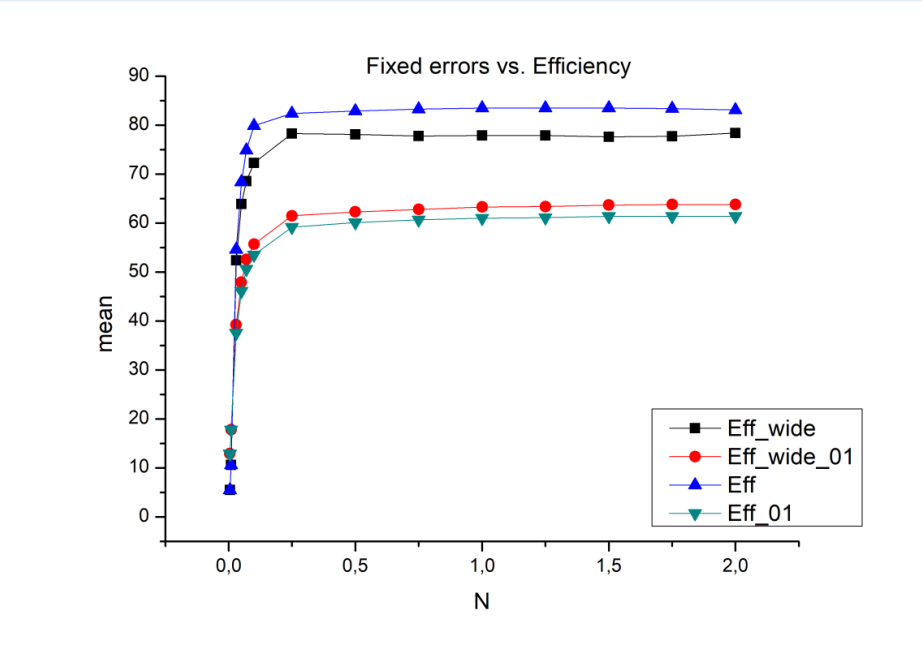




Number of reconstructed vertices, distribution width for primary vertex and time of reconstruction for various targets and triggers (Run 7)

Optimization of vertex finding efficiency with respect to hits errors

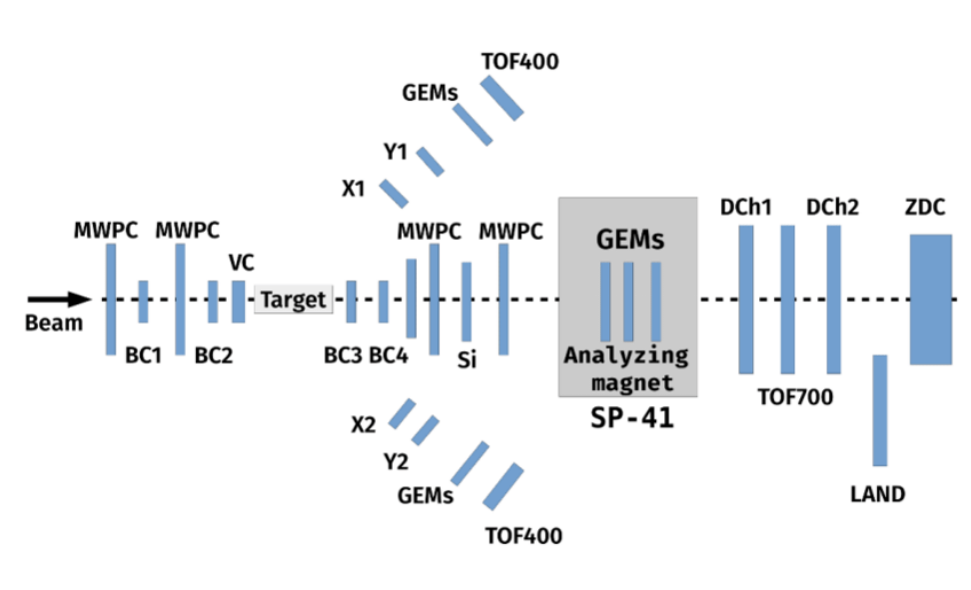




Dependence of vertex resolution precision and efficiency on hits errors (simulated data).

- Efficiency is maximum for N = 0.25
- Precision of vertex resolution is minimum for N = 0.04

Adding triggers description into geometry of SRC experiment



The SRC (Short Range Correlation) experiment is an extension of the physics program of the BM@N experiment.

- Multiwire proportional chambers (MWPC).
- Liquid-hydrogen target.
- Beam counters (BC1, BC2, VC, BC3, BC4).
- Scintillation-based trigger detectors in the arms (X1, X2, Y1, Y2).
- Three silicon planes (Si).
- Gas electron multiplier stations (GEMs).
- Drift chamber stations (DCh1, Dch2).
- Time-of-flight detectors (TOF-400 in the arms and TOF-700).
- Zero degree calorimeter.
- Analyzing magnet SP-41.

Carbon beam with energy 4 GeV/c/nucleon + liquid-hydrogen target.

The arm trigger detectors (X1, Y1, X2, Y2) are important part of the experimental setup. They are used to select events of corresponding SRC reaction.

The work consisted of adding geometry of X1, X2, Y1, Y2 triggers into geometry of the experiment and developing classes for digitization of Monte-Carlo data. Geometry of BC triggers was refined. Digitizer has been developed and normalization of signals has been performed. Results are uploaded into repository.

When the information from the arm triggers was integrated in reconstruction procedure the experimental and simulation data could be compared correctly, as it provided us with an ability to select the events in which the left or/and the right arm triggers worked.

Summary

- Study of dependence of the BmnRoot reconstruction performance bottlenecks on the input data has been performed.
- Integration of the PROOF into the BmnRoot reconstruction macros is in progress.
- Hunting for memory errors of the BmnRoot is in progress.
- Vertex finding algorithm based on virtual planes approach has been studied from the point of view of performance. Dependence of vertex finding efficiency on hits errors is explored.
- Triggers description is added in the geometry of SRC. Committed to Gitlab.