Modification of simulation, reconstruction and quality assurance modules of the BmnRoot software

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

K.I.Mashitsin, A.A.Yufryakova

Saint-Petersburg State University

Joint Institute for Nuclear Research

5th Collaboration Meeting of the BM@N Experiment at the NICA Facility. 20-21 April 2020. VBLHEP







Outline

- 1. Optimization of simulation and new version of reconstruction in BmnRoot.
- 2. First experience with PROOF.
- 3. QA porting on JSROOT.





Optimization of simulation and new version of reconstruction in BmnRoot



Analysis and parallelization of simulation



Analysis Configuration Collection Log	Summary B	ottom-up Caller/Callee	Top-down Tree Platform				
Grouping: Function / Call Stack			▼ (%) Q Q _a				
Function / Call Stack	CPU Time 🔻 🔌	Module	Function (Full)				
TRandom::Gaus	145.867s	libMathCore.so.6.16.00	TRandom::Gaus(double, double)				
DeadZoneOfStripLayer::IsInside	137.187s	libGem.so.0	DeadZoneOfStripLayer::IsInside(double, double)				
▶cos_fma	98.153s	libm.so.6	cos_fma				
TRandom3::Rndm	80.814s	libMathCore.so.6.16.00 TRandom3::Rndm(void)					
▶sin_fma	77.566s	libm.so.6	sin_fma				
▶ deflate	74.185s	libz.so.1	deflate				
FairMCApplication::Stepping	67.573s	libBase.so.18.2.0	FairMCApplication::Stepping(void)				
BmnGemStripModule::AddRealPointFi	46.391s	libGem.so.0	BmnGemStripModule::AddRealPointFull(double, double, do				
BmnGemStripLayer::ConvertPointToSt	43.867s	libGem.so.0	BmnGemStripLayer::ConvertPointToStripPosition(double, double)				
std::map <std::pair<int, int="">, int, std::les</std::pair<int,>	41.465s	libBmnData.so.0	std::map <std::pair<int, int="">, int, std::less<std::pair<int, int="">>, std::allocator<s< td=""></s<></std::pair<int,></std::pair<int,>				
StripCluster::AddStrip	41.270s	libGem.so.0	StripCluster::AddStrip(int, double)				
BmnGemStripLayer::IsPointInsideStrip	31.419s	libGem.so.0	BmnGemStripLayer::IsPointInsideStripLayer(double, double)				
BmnGemStripLayer::ConvertNormalPc	27.492s	libGem.so.0	BmnGemStripLayer::ConvertNormalPointToStripX(double, double)				
std::map <std::pair<int, int="">, int, std::les</std::pair<int,>	20.336s	libBmnData.so.0	std::map <std::pair<int, int="">, int, std::less<std::pair<int, int="">>, std::allocator<s< td=""></s<></std::pair<int,></std::pair<int,>				
std::operator< <int, int=""></int,>	18.805s	libBmnData.so.0	bool std::operator< <int, int="">(std::pair<int, int=""> const&, std::pair<int, int=""> const</int,></int,></int,>				
BmnGemStripLayer::IsPointInsideDea	18.630s	libGem.so.0	BmnGemStripLayer::IsPointInsideDeadZones(double, double)				
BmnNewFieldMap::FieldInterpolate	18.493s	libBmnField.so.0	BmnNewFieldMap::FieldInterpolate(TArrayF*, double, double, double)				
std::_Rb_tree_iterator <std::pair<int cor<="" p=""></std::pair<int>	16.267s	libBase.so.18.2.0	std::_Rb_tree_iterator <std::pair<int const,="" fairvolume*="">>::operator++(int)</std::pair<int>				
▶ TArrayF::At	14.695s	libBmnField.so.0	TArrayF::At(int) const				
BmnNewFieldMap::IsInside	13.857s	libBmnField.so.0	BmnNewFieldMap::IsInside(double, double, double, int&, int&, int&, double{				
std::map <int, bool,="" std::less<int="">, std::a</int,>	12.740s	libBmnData.so.0	std::map <int, bool,="" std::less<int="">, std::allocator<std::pair<int bool="" const,="">>></std::pair<int></int,>				
BmnFieldMap::Interpolate	12.701s	libBmnField.so.0	BmnFieldMap::Interpolate(double, double, double)				
	44 700-	üher an C					
	200	s 400s	600s 800s 1000s				
root.exe (TID: 2534)			and the second				
E							

Dynamic analysis displays hotspots of the BmnRoot simulation modules.





OpenMP parallelization. DGSM generator, 1000 events. Testbench: IntelXeonE-2136 @ 4.5GHzTurbo (6 cores with HT). RAM: 32 Gigabytes. OS: Ubuntu. Not committed to git.

bmnroot/gem/BmnGemStripDigitizer.cxx

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





Analysis and parallelization of reconstruction



OpenMP parallelization. Reconstruction. 10000 events. Experimental data. Testbench Intel Xeon CPU E5-2650 v3 @ 2.30 GHz chips (24 cores, no hyperthreading), 128 GB of RAM, 4 x GK110BGL, OS Debian 10.2. Committed to git.

bmnroot/gem/BmnCellAutotracking.cxx













⊘ Effective CPU Utilization Histogram

Idle

700s = 600s = 500s = 400s = 300s = 200s = 100s =

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.

	Analysis Configuration	Collection Log	Summary	Bottom-up	Caller/Callee	Top-do	wn Tree	Platform	
G	Grouping: Function / Call Stack								
Function / Call Stack		CPU Time 🔻 📄		Instructions Retire	ed Mid	Microarchitecture Usage 🖹			
►	[vmlinux]		34.466s		68,907,300,	000		17.0%	
►	BmnNewFieldMap::Fiel	dinterpolate	12.549s		123,796,200,	000		55.5%	
►	func@0x7ec0		9.843s		58,119,600,	000		33.9%	
►	libc_malloc		8.255s		69,270,300,	000		48.7%	
▶ func@0x7fd50		7.212s		60,776,100,	000		45.5%		
►	TGeoVoxelFinder::GetN	VextCandidates	6.538s		62,492,100,	000		51.6%	
►	BmnKalmanFilter::RK4	Order	6.290s		55,677,600,	000		53.8%	
[Kernel dynamic code]		6.094s		4,973,100,	000		14.0%		
BmnFieldMap::Interpolate		5.693s 📒		80,873,100,	000		86.1%		
TGeoVoxelFinder::GetCheckList		5.147s 📒		45,177,000,	000		45.5%		
▶ func@0x81b80		4.027s 📒		20,552,400,	000	31			
BmnInnerTrackingRun7::MatchHit		3.492s 📒		18,539,400,	000		30.2%		
_logf_finite		3.265s 📒		24,987,600,	000		51.0%		
BmnKalmanFilter::RK4TrackExtrapolate		3.252s		20.898.900	000		50.1%		
H		1							
	p: 🕇 —	⊯r <u>⊯</u> r 0s 2	0s 40s	60s 80	ls 100s 120	Os 140	s 160s	185.568s	
ad	root.exe (TID: 16365)								
Thre									

Dynamic analysis displays hotspots of the BmnRoot reconstruction modules. Optimization is in progress.

Ana	ysis Configuration Collection Log Summary Bottom-up Caller/Callee Top-down Tree Platform BmnKalmanFilter.cxx × BmnNewFieldMap.cxx ×		
s	Assembly III = b^{\mp} b^{+} b_{\pm}		
	Source	🖕 CPU Time: Total 🕑	CPU Time: Self 🗵
60	dx = dy = dz = 0;		
51	return kFALSE;		
52	}		
63			
34	// Determine grid cell		
35	<pre>ix = Int_t((x1 - fXmin) / fXstep);</pre>		
36	<pre>iy = Int_t((yl - fYmin) / fYstep);</pre>		
67	<pre>iz = Int_t((zl - fZmin) / fZstep);</pre>		
58		1	
69	// Relative distance from grid point (in units of cell size)		
70	<pre>dx = (x1 - fXmin) / fXstep - Double_t(ix);</pre>		
71	dy = (yl - fYmin) / fYstep - Double_t(iy);		
72	dz = (zl - fZmin) / fZstep - Double_t(iz);		
73			
74	return kTRUE;		
75	}		
76			
77	Double_t BmnNewFieldMap::FieldInterpolate(TArrayF* fcomp, Double_t x, Double_t y, Double_t z) {	0.2%	1.242s
78	$Int_t ix = 0;$	0.0%	0.312s
79	Int_t iy = 0;	0.0%	0.028s
30	$Int_t iz = 0;$	0.0%	0.036s
31	Double_t $dx = 0.;$	0.0%	0.244s
32	Double_t dy = 0.;	0.0%	0.056s
33	$Double_t dz = 0.;$	0.0%	0.280s
34			
35	if (IsInside(x, y, z, ix, iy, iz, dx, dy, dz)) {	0.3%	2.368s
86	fHa[0][0][0] = fcomp->At(ix * fNy * fNz + iy * fNz + iz);	0.3%	2.152s
37	fHa[1][0][0] = fcomp->At((ix + 1) * fNy * fNz + iy * fNz + iz);	0.2%	1.924s
38	fHa[0][1][0] = fcomp->At(ix * fNy * fNz + (iy + 1) * fNz + iz);	0.2%	1.690s
89	fHa[1][1][0] = fcomp->At((ix + 1) * fNy * fNz + (iy + 1) * fNz + iz);	0.2%	1.900s
90	fHa[0][0][1] = fcomp->At(ix * fNy * fNz + iy * fNz + (iz + 1));	0.2%	1.422s
91	fHa[1][0][1] = fcomp->At((ix + 1) * fNy * fNz + iy * fNz + (iz + 1));	0.2%	1.732s
92	fHa[0][1][1] = fcomp->At(ix * fNy * fNz + (iy + 1) * fNz + (iz + 1));	0.2%	1.668s
93	fHa[1][1][1] = fcomp->At((ix + 1) * fNy * fNz + (iy + 1) * fNz + (iz + 1));	0.3%	2.6365
94			
95	return Interpolate(dx, dy, dz);	0.2%	1.464s
96	}		
97	return 0.;	0.1%	0.648s
98)	0.2%	1.322s
99			







First experience with PROOF





PROOF (Parallel ROOT) – distributed parallel analysis framework based on ROOT.

PROOF-Lite is a version of PROOF optimized for multicore desktops and laptops.



In progress.





QA porting on JSROOT



Development and implementation of modified version of QA (Quality Assurance).

Previous version of QA is based on static html-pages.

New version is based on more functional and convenient JSROOT.

JSROOT (JavaScript ROOT). The JSROOT project intends to implement ROOT graphics for web browsers. Reading of binary ROOT files is supported. Successor of the JSRootIO project.

Source and short User Guide are available.

Committed to git.



BM@









Thank you for attention!

Sergei Nemnyugin is snemnyugin@mail.ru