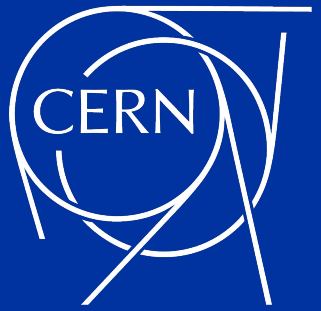


Coding Patterns for Better Performance

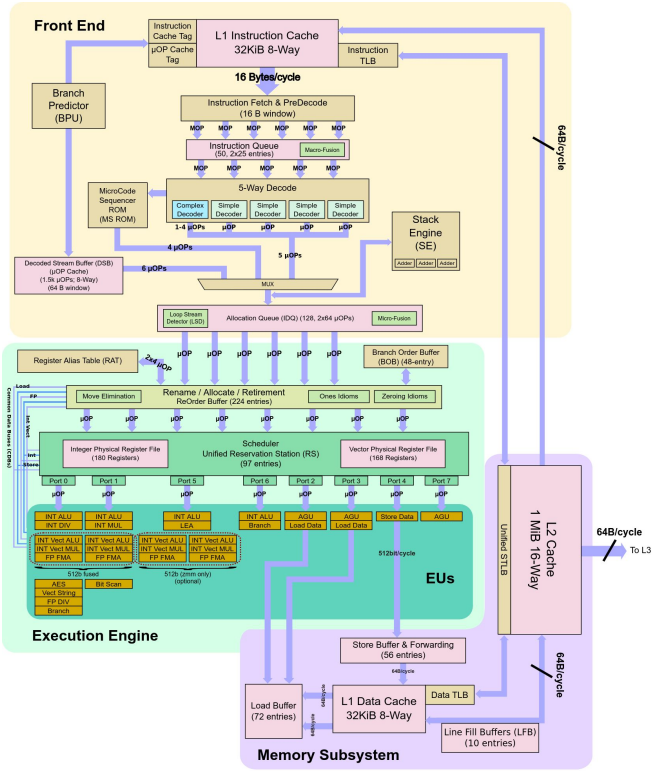
G. Amadio (EP-SFT), for the Geant4 Collaboration

21 Sep 2021

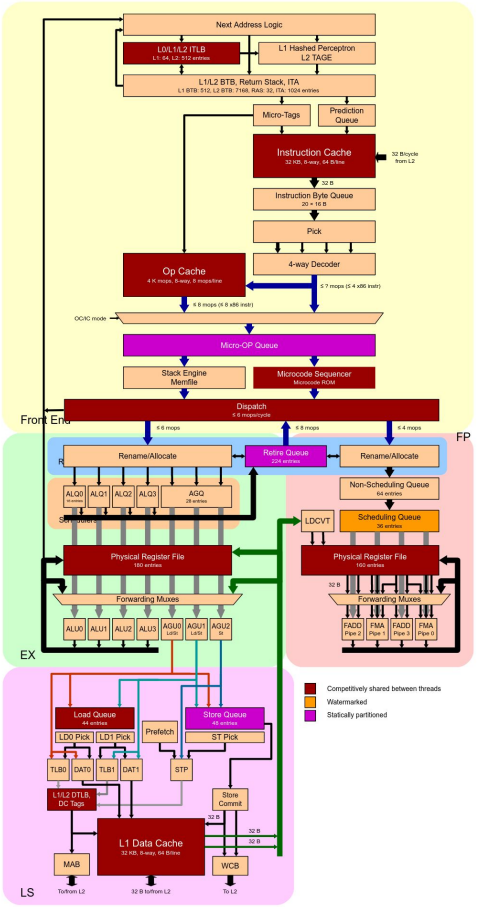
Intel and AMD Microarchitectures

- Front End
  - Instruction Fetch and Decode
  - Branch Predictor Unit
  - L1 Instruction Cache
  - Instruction TLB
- Back End
  - Execution Engine
    - Register Renaming
    - Move Elimination
  - Memory Subsystem
    - Load/Store Units
    - L1 Data Cache
    - L2 Shared Cache
    - Data TLB

Intel Skylake

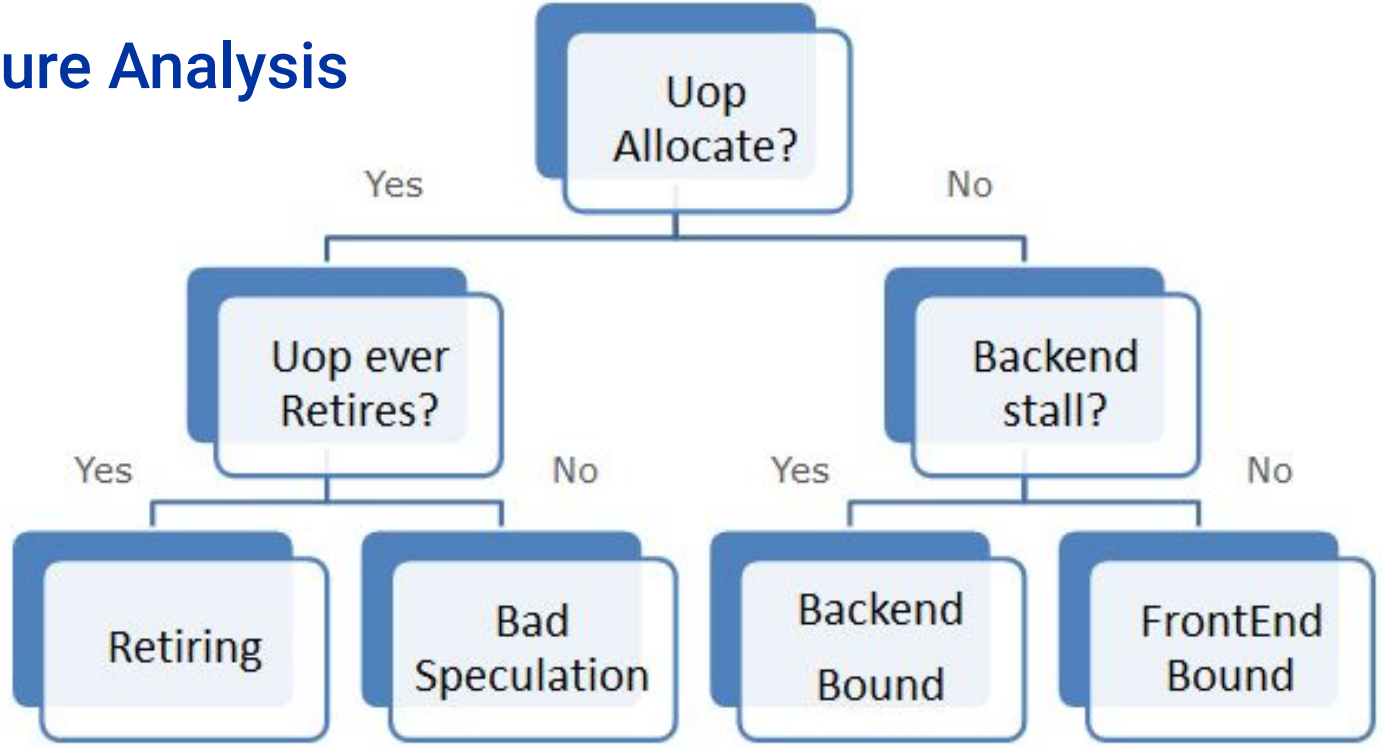


AMD Zen 2



source: https://en.wikichip.org

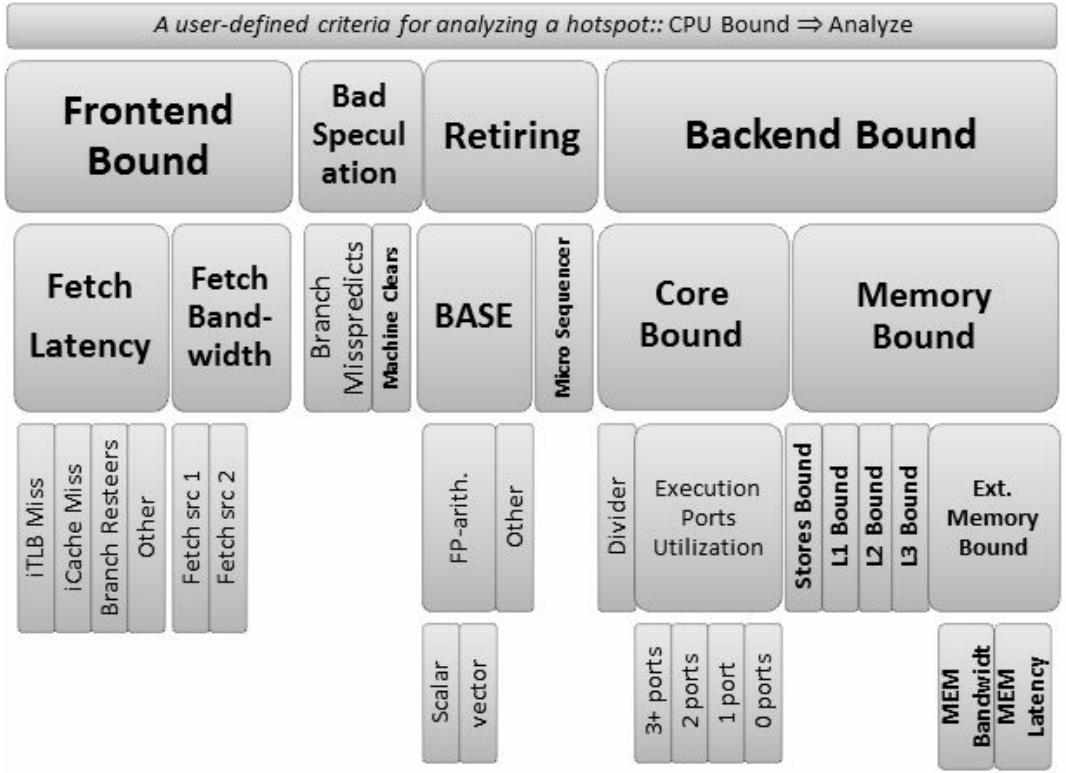
Microarchitecture Analysis



A. Yasin, "A Top-Down method for performance analysis and counters architecture," *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Monterey, CA, 2014, pp. 35-44, doi: 10.1109/ISPASS.2014.6844459.

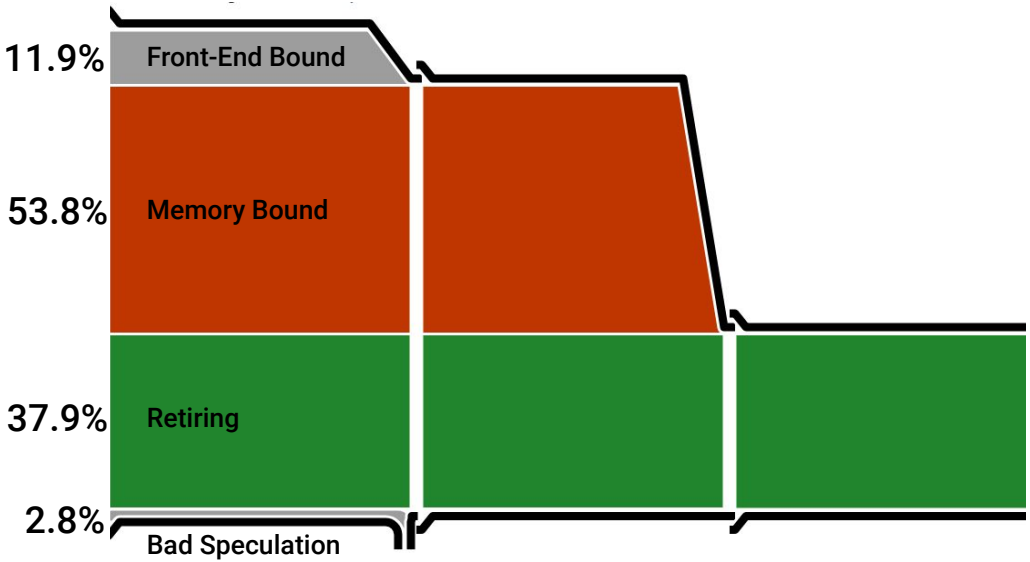
Top-Down Analysis of Microarchitecture

- Front End Bound
  - Code Duplication
  - Code Layout (Locality)
  - Frequent Branching
  - Unnecessary Work
- Back End Bound
  - Core Bound
    - Data Dependencies
    - Divisions and Special Functions
  - Memory Bound
    - False Sharing
    - Frequent DRAM Accesses
    - Scattered Memory Accesses
    - Unnecessary Work



A. Yasin, "A Top-Down method for performance analysis and counters architecture," *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Monterey, CA, 2014, pp. 35-44, doi: 10.1109/ISPASS.2014.6844459.

Geant4 Microarchitecture Usage on Haswell

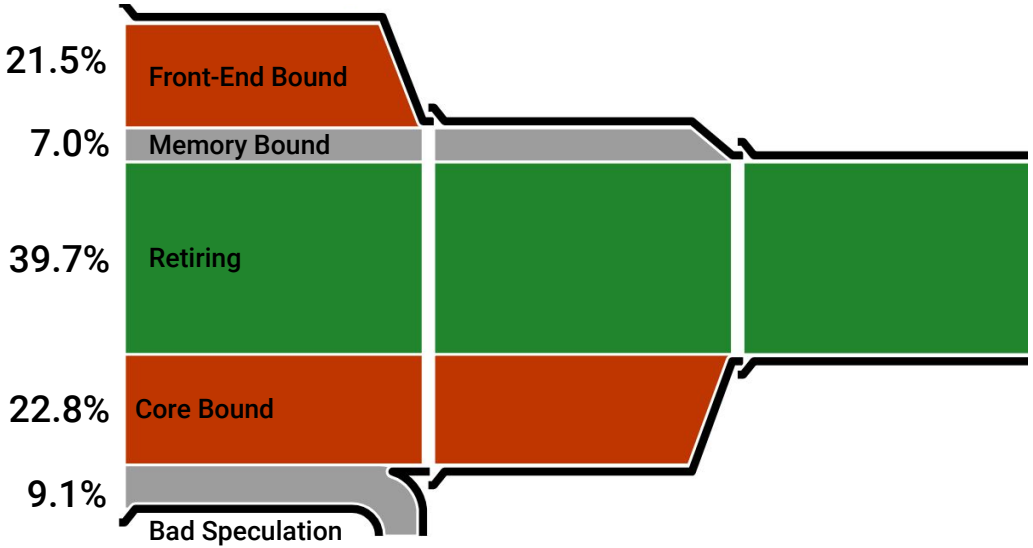


Retiring:	**37.9%**	**of Pipeline Slots**
Front-End Bound:	**11.9%**	**of Pipeline Slots**
Bad Speculation:	**2.8%**	**of Pipeline Slots**
Back-End Bound:	**47.3%**	**of Pipeline Slots**
Memory Bound:	**53.8%**	**of Pipeline Slots**
L1 Bound:	**48.9%**	**of Clockticks**
DTLB Overhead:	13.3%	of Clockticks
Loads Blocked by Store Forwarding:	0.0%	of Clockticks
Lock Latency:	0.0%	of Clockticks
Split Loads:	0.0%	of Clockticks
4K Aliasing:	0.3%	of Clockticks
FB Full:	0.0%	of Clockticks
L2 Bound:	0.0%	of Clockticks
L3 Bound:	**5.6%**	**of Clockticks**
Contested Accesses:	0.0%	of Clockticks
Data Sharing:	0.0%	of Clockticks
L3 Latency:	6.9%	of Clockticks
SQ Full:	0.0%	of Clockticks
DRAM Bound:	**0.0%**	**of Clockticks**
Store Bound:	**0.0%**	**of Clockticks**
Core Bound:	**0.0%**	**of Pipeline Slots**

Mostly memory bound on Haswell

https://software.intel.com/content/www/us/en/develop/documentation/vtune-cookbook/top/methodologies/top-down-microarchitecture-analysis-method.html

Geant4 Microarchitecture Usage on Skylake



Retiring:	**39.7%**	**of Pipeline Slots**
Front-End Bound:	**21.5%**	**of Pipeline Slots**
Bad Speculation:	**9.1%**	**of Pipeline Slots**
Back-End Bound:	**29.8%**	**of Pipeline Slots**
Memory Bound:	**7.0%**	**of Pipeline Slots**
Core Bound:	**22.8%**	**of Pipeline Slots**
Divider:	10.3%	of Clockticks
Port Utilization:	**24.5%**	**of Clockticks**
Cycles of 0 Ports Utilized:	**1.9%**	**of Clockticks**
Cycles of 1 Port Utilized:	17.1%	of Clockticks
Cycles of 2 Ports Utilized:	18.7%	of Clockticks
Cycles of 3+ Ports Utilized:	**38.5%**	**of Clockticks**
Vector Capacity Usage (FPU):	12.5%	

Mostly frontend and core bound on Skylake, quite different than Haswell

https://software.intel.com/content/www/us/en/develop/documentation/vtune-cookbook/top/methodologies/top-down-microarchitecture-analysis-method.html

Top 20 classes in Geant4 10.5.1

Class / Source Function / Call Stack	CPU Time ▼	Instructions Retired	Microarchitecture Usage	
			Microarchitecture Usage	CPI Rate
CLHEP::Hep3Vector	9.7%	10.7%	37.3%	0.589
LArWheelCalculator_Impl::DistanceCalculatorSaggingOff	8.2%	11.3%	55.8%	0.473
G4PhysicsVector	5.4%	2.9%	21.3%	1.219
[Not part of any known object class]	5.2%	4.4%	33.5%	0.777
G4PolyconeSide	4.6%	7.1%	57.8%	0.431
G4VoxelNavigation	4.0%	1.6%	14.2%	1.661
MagField::AtlasFieldSvc	3.6%	3.7%	40.6%	0.633
G4SteppingManager	2.8%	3.0%	39.7%	0.611
LArWheelSolid	2.2%	1.7%	31.2%	0.860
LArWheelCalculator	2.2%	3.6%	51.4%	0.394
G4ProductionCutsTable	1.6%	0.7%	14.8%	1.555
CLHEP::MixMaxRng	1.6%	2.1%	50.0%	0.493
BFieldCache	1.5%	2.5%	60.3%	0.385
G4CrossSectionDataStore	1.4%	1.6%	39.7%	0.548
G4LogicalVolume	1.4%	1.1%	28.3%	0.893
G4Navigator	1.4%	1.4%	32.7%	0.649
G4Tubs	1.4%	1.2%	34.3%	0.751
G4UrbanMscModel	1.3%	0.8%	24.1%	1.023
G4VProcess	1.3%	0.7%	26.2%	1.159
G4VCSGfaceted	1.3%	1.4%	49.4%	0.607

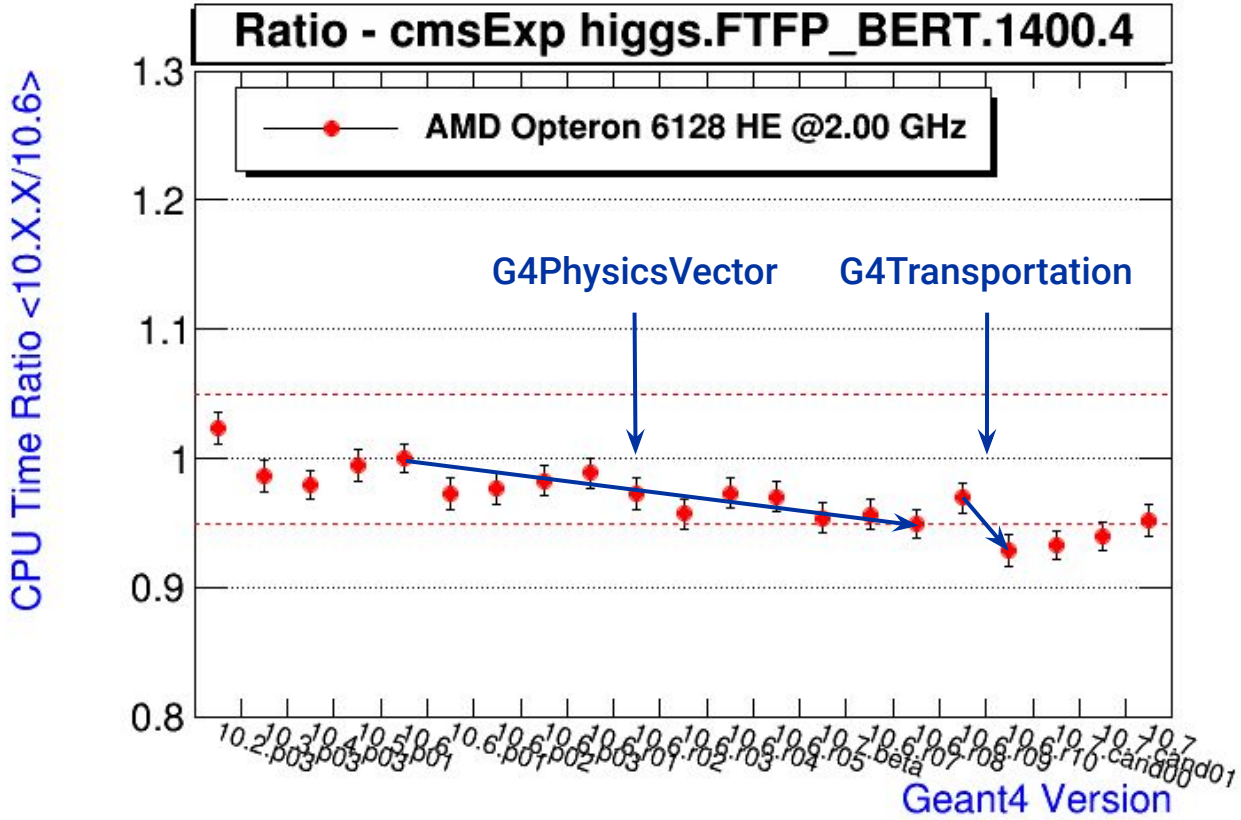
Top 20 classes in Geant4 10.7

Class / Source Function / Call Stack	CPU Time ▼	Instructions Retired	Microarchitecture Usage	
			Microarchitecture Usage	CPI Rate
CLHEP::Hep3Vector	9.4%	10.8%	40.7%	0.564
LArWheelCalculator_Impl::DistanceCalculatorSaggingOff	8.9%	11.9%	54.2%	0.490
G4PolyconeSide	5.3%	7.5%	51.0%	0.466
G4VoxelNavigation	4.4%	1.8%	14.2%	1.661
[Not part of any known object class]	4.2%	3.4%	30.7%	0.797
G4PhysicsVector	4.0%	2.5%	25.1%	1.055
MagField::AtlasFieldSvc	3.8%	3.8%	40.0%	0.643
G4SteppingManager	2.8%	3.2%	40.2%	0.575
LArWheelCalculator	2.7%	3.9%	46.3%	0.452
LArWheelSolid	2.7%	1.8%	29.2%	0.943
G4VCSGfaceted	1.8%	1.5%	39.7%	0.820
G4Navigator	1.7%	1.8%	38.4%	0.612
G4ProductionCutsTable	1.7%	0.7%	13.9%	1.605
BFieldCache	1.6%	2.7%	61.0%	0.393
G4LogicalVolume	1.5%	1.1%	27.6%	0.908
G4Tubs	1.5%	1.2%	31.8%	0.827
G4UrbanMscModel	1.5%	0.9%	21.4%	1.127
G4CrossSectionDataStore	1.5%	1.6%	37.8%	0.591
CLHEP::MixMaxRng	1.3%	1.6%	48.4%	0.529
G4AffineTransform	1.3%	1.2%	38.7%	0.654

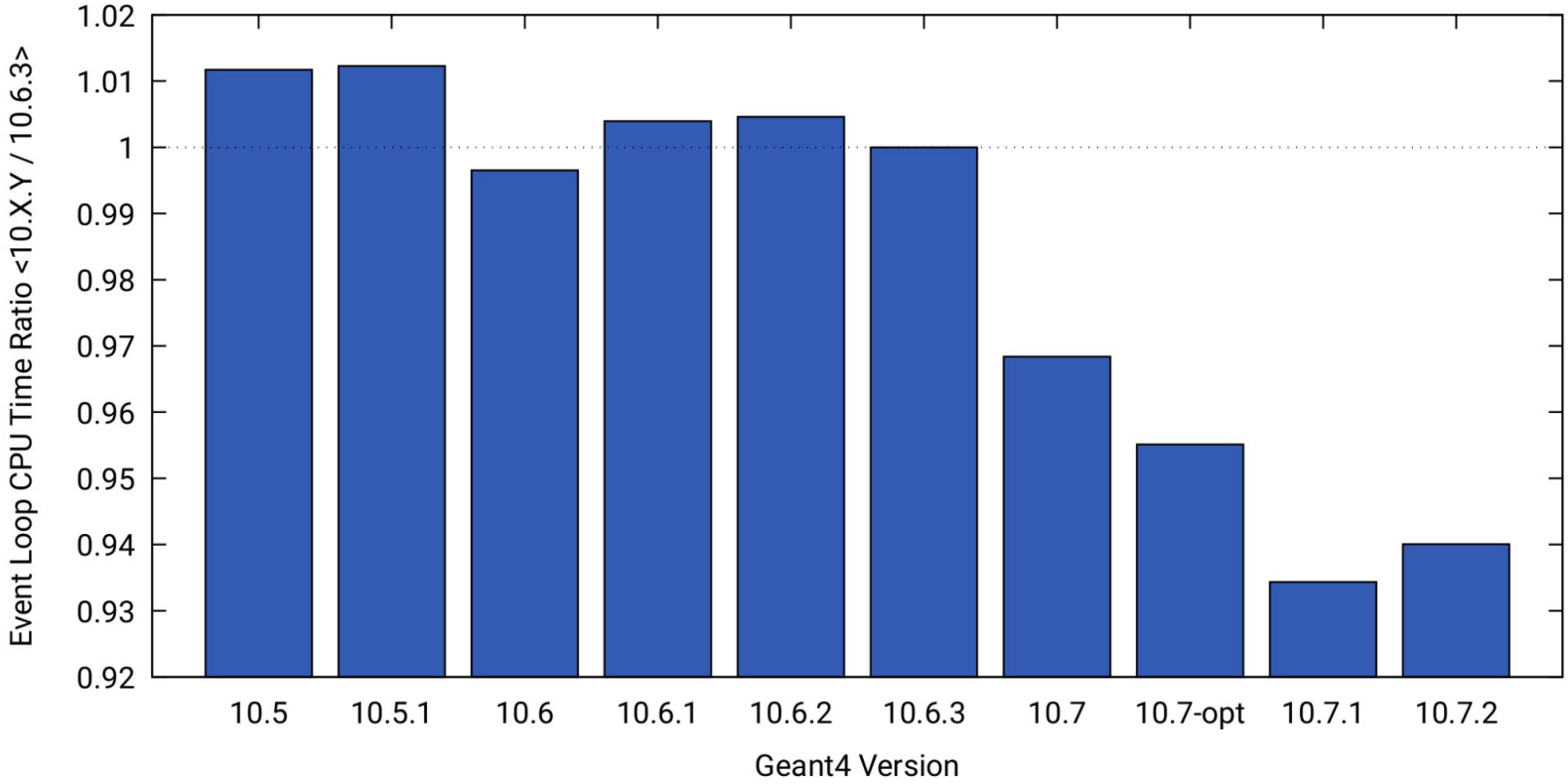
Top 20 classes in Geant4 10.7 + Optimizations

Class / Source Function / Call Stack	CPU Time ▼	Instructions Retired	Microarchitecture Usage	
			Microarchitecture Usage	CPI Rate
CLHEP::Hep3Vector	9.6%	10.9%	40.4%	0.565
LArWheelCalculator_Impl::DistanceCalculatorSaggingOff	8.9%	11.8%	55.7%	0.485
G4PolyconeSide	5.2%	7.5%	55.0%	0.439
G4VoxelNavigation	4.6%	1.7%	14.7%	1.706
[Not part of any known object class]	4.3%	3.5%	31.0%	0.796
MagField::AtlasFieldSvc	3.9%	3.9%	40.7%	0.649
G4PhysicsVector	2.9%	2.0%	27.3%	0.944
G4SteppingManager	2.7%	3.1%	41.4%	0.554
LArWheelSolid	2.6%	1.8%	28.3%	0.892
LArWheelCalculator	2.4%	3.9%	51.1%	0.399
G4Navigator	1.8%	1.8%	35.3%	0.647
G4ProductionCutsTable	1.7%	0.7%	14.6%	1.596
BFieldCache	1.7%	2.7%	61.4%	0.398
G4UrbanMscModel	1.6%	0.9%	20.5%	1.115
G4VEmProcess	1.6%	1.1%	27.9%	0.883
G4LogicalVolume	1.5%	1.1%	29.5%	0.902
G4Tubs	1.5%	1.2%	33.9%	0.785
G4VCSGfaceted	1.5%	1.4%	46.7%	0.650
CLHEP::MixMaxRng	1.3%	1.6%	47.6%	0.527
G4AffineTransform	1.3%	1.3%	37.2%	0.645
G4CrossSectionDataStore	1.3%	1.4%	41.7%	0.580

Two Major Optimizations in Geant4 10.6 and 10.7



FullSimLight Performance vs Geant4 Version



Optimization Examples

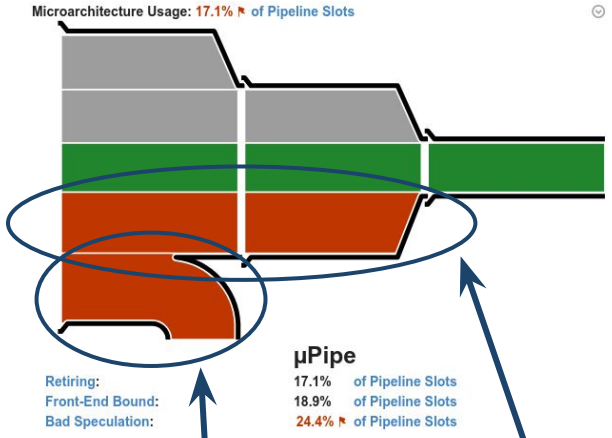
Reduce Code Duplication

```
-#include "G4PhysicalConstants.hh"
-#include "G4Log.hh"
-#include "G4Exp.hh"
-
-using namespace std;

 G4hPairProductionModel::G4hPairProductionModel(const G4ParticleDefinition* p,
                                                const G4String& nam)
@@ -65,114 +60,3 @@ G4hPairProductionModel::~G4hPairProductionModel()
 {}

 //....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......
-
-G4double G4hPairProductionModel::ComputeDMicroscopicCrossSection(
-                                        G4double tkin,
-                                        G4double Z,
-                                        G4double pairEnergy)
-//  differential cross section
-{
-  static const G4double bbbtf= 183. ;
-  static const G4double bbbh = 202.4 ;
-  static const G4double g1tf = 1.95e-5 ;
-  static const G4double g2tf = 5.3e-5 ;
-  static const G4double g1h  = 4.4e-5 ;
-  static const G4double g2h  = 4.8e-5 ;
-
-  G4double totalEnergy  = tkin + particleMass;
-  G4double residEnergy  = totalEnergy - pairEnergy;
-  G4double massratio    = particleMass/electron_mass_c2 ;
lines 511-539
```

This was a copy of G4MuPairProductionModel::ComputeDMicroscopicCrossSection. We can keep only the copy from the base class.

Example of Bad Speculation

Be careful with what you assume the compiler can optimize for you.



Source Line ▲	Source	Clockticks	Instructions Retired	CPI Rate	Locators					
					Retiring	Front-End Bound	Bad Speculation	Back-End Bound		
								Memory Bound	Core Bound	
									Divider	Port Utilization
40										
41	inline double Hep3Vector::operator () (int i) const {									
42	switch(i) {	20,232,000,000	18,900,000,000	1.070	16.7%	18.9%	23.3%	18.5%	0.0%	20.0%
43	case X:									
44	return x();									
45	case Y:									
46	return y();	72,000,000	36,000,000	2.000	0.4%	0.0%	0.0%	0.0%	0.0%	0.0%
47	case Z:									
48	return z();	144,000,000	0		0.0%	0.0%	1.5%	0.0%	0.0%	1.3%
49	}									
50	return 0.;									
51	}									

Example of Bad Speculation

Be careful with what you assume the compiler can optimize for you.

```
1   class vector {
2       public:
3       double operator[](int i) {
4           switch(i) {
5               case 0: return x;
6               case 1: return y;
7               case 2: return z;
8           }
9       }
10
11      double operator()(int i) { return (&x)[i]; }
12
13      private:
14      double x, y, z;
15  };
```

```
1   vector::operator[](int):
2     cmp esi, 1
3     je .L2
4     cmp esi, 2
5     jne .L6
6     vmovsd xmm0, QWORD PTR [rdi+16]
7     ret
8   .L2:
9     vmovsd xmm0, QWORD PTR [rdi+8]
10    ret
11  .L6:
12    vmovsd xmm0, QWORD PTR [rdi]
13    ret
14  vector::operator()(int):
15    movsx rsi, esi
16    vmovsd xmm0, QWORD PTR [rdi+rsi*8]
17    ret
```

Branching Efficiently

```
    Reorder condition in G4CrossSectionDataStore::ComputeCrossSection()

diff --git a/source/processes/hadronic/cross_sections/src/G4CrossSectionDataStore.cc b/source/processes/hadronic/cross_sections/src/G4CrossSectionData
Store.cc
index d81d3239ba..06e446f192 100644
--- a/source/processes/hadronic/cross_sections/src/G4CrossSectionDataStore.cc
+++ b/source/processes/hadronic/cross_sections/src/G4CrossSectionDataStore.cc
@@ -271,10 +271,10 @@ G4double
 G4CrossSectionDataStore::ComputeCrossSection(const G4DynamicParticle* part,
                                              const G4Material* mat)
 {
-  if(mat == currentMaterial && part->GetDefinition() == matParticle
-     && part->GetKineticEnergy() == matKinEnergy) {
+  if(part->GetKineticEnergy() == matKinEnergy && mat == currentMaterial &&
+     part->GetDefinition() == matParticle)
     return matCrossSection;
-  }
+
   currentMaterial = mat;
   matParticle = part->GetDefinition();
   matKinEnergy = part->GetKineticEnergy();

commit 3a2c4714fddc62e6ea31b6c5f074f60461ac05f4
Author: Guilherme Amadio <amadio@cern.ch>

    Update History file for G4CrossSectionDataStore

diff --git a/source/processes/hadronic/cross_sections/History b/source/processes/hadronic/cross_sections/History
lines 3391-3418/3673 93%
```

When a branching condition has several terms, order it from the most discriminant term to the least. Here, the energy is different much more frequently than material or particle type, so this order leads to more early decisions and better performance.

Probabilities:

mat == currentMaterial ⇒ 98.9%
matParticle == part->GetDefinition ⇒ 71.6%
matKinEnergy == part->GetKineticEnergy() ⇒ 32.1%

Note that these are independent. They are equal together only about 6.9% of the time.

Unnecessary Work: Caching Data

```
@@ -297,38 +300,30 @@ G4CrossSectionDataStore::GetCrossSection(const G4DynamicParticle* part,
                                         const G4Element* elm,
                                         const G4Material* mat)
 {
-  if(mat == elmMaterial && elm == currentElement &&
-     part->GetDefinition() == elmParticle &&
-     part->GetKineticEnergy() == elmKinEnergy)
-    { return elmCrossSection; }
-
-  elmMaterial = mat;
-  currentElement = elm;
-  elmParticle = part->GetDefinition();
-  elmKinEnergy = part->GetKineticEnergy();
-  elmCrossSection = 0.0;
-
-  G4int i = nDataSetList-1;
+  G4int i = nDataSetList-1;
    G4int Z = elm->GetZasInt();
+
    if (elm->GetNaturalAbundanceFlag() &&
        dataSetList[i]->IsElementApplicable(part, Z, mat)) {

     // element wise cross section
-    elmCrossSection = dataSetList[i]->GetElementCrossSection(part, Z, mat);
-  } else {
-    // isotope wise cross section
-    size_t nIso = elm->GetNumberOfIsotopes();
-
-    // user-defined isotope abundances
lines 3282-3310/3673 91%
```

This method is always called with each element of each material in a loop, so the element is never the same, and the cache was missed 100% of the time.

Unnecessary Work: Return Early If Possible

```
commit 1c0b893bef0fe6f32fd8593989882239628262a9
Author: Guilherme Amadio <amadio@cern.ch>

    G4MuPairProductionModel: make early return condition more explicit

diff --git a/source/processes/electromagnetic/muons/src/G4MuPairProductionModel.cc b/source/processes/electromagnetic/muons/src/G4MuPairProductionMode
l.cc
index f7ba48dc7d..e0b7c550f0 100644
--- a/source/processes/electromagnetic/muons/src/G4MuPairProductionModel.cc
+++ b/source/processes/electromagnetic/muons/src/G4MuPairProductionModel.cc
@@ -321,6 +321,9 @@ G4double G4MuPairProductionModel::ComputeDMicroscopicCrossSection(
   static const G4double g1h  = 4.4e-5 ;
   static const G4double g2h  = 4.8e-5 ;

+  if (pairEnergy <= 4.0 * electron_mass_c2)
+    return 0.0;
+
   G4double totalEnergy  = tkin + particleMass;
   G4double residEnergy  = totalEnergy - pairEnergy;
   G4double massratio    = particleMass/electron_mass_c2;
@@ -334,7 +337,6 @@ G4double G4MuPairProductionModel::ComputeDMicroscopicCrossSection(
   G4double c8 = 6.*particleMass*particleMass;
   G4double alf = c7/pairEnergy;
   G4double a3 = 1. - alf;
-  if (a3 <= 0.) { return cross; }

   // zeta calculation
   G4double bbb,g1,g2;

lines 685-712/3673 18%
```

Moving the early return up reduces unnecessary divisions.
Also a3 <= 0 is harder to understand than the new form.

Unnecessary Work: Avoid Expensive Function Calls

```
    G4MuPairProductionModel: potentially avoid call to log in zeta calculation

diff --git a/source/processes/electromagnetic/muons/src/G4MuPairProductionModel.cc b/source/processes/electromagnetic/muons/src/G4MuPairProductionMode
l.cc
index ae040db976..55d514719c 100644
--- a/source/processes/electromagnetic/muons/src/G4MuPairProductionModel.cc
+++ b/source/processes/electromagnetic/muons/src/G4MuPairProductionModel.cc
@@ -350,14 +350,15 @@ G4double G4MuPairProductionModel::ComputeDMicroscopicCrossSection(
   if( Z < 1.5 ) { bbb = bbbh ; g1 = g1h ; g2 = g2h ; }
   else          { bbb = bbbtf; g1 = g1tf; g2 = g2tf; }

-  G4double zeta = 0;
-  G4double zeta1 =
-    0.073*G4Log(totalEnergy/(particleMass+g1*z23*totalEnergy))-0.26;
-  if ( zeta1 > 0.)
+  G4double zeta = 0.0;
+  G4double z1exp = totalEnergy / (particleMass + g1*z23*totalEnergy);
+
+  // 35.221047195922 is the root of zeta1(x) = 0.073 * log(x) - 0.26, so the
+  // condition below is the same as zeta1 > 0.0, but without calling log(x)
+  if (z1exp > 35.221047195922)
   {
-    G4double zeta2 =
-      0.058*G4Log(totalEnergy/(particleMass+g2*z13*totalEnergy))-0.14;
-    zeta  = zeta1/zeta2 ;
+    G4double z2exp = totalEnergy / (particleMass + g2*z13*totalEnergy);
+    zeta = (0.073 * G4Log(z1exp) - 0.26) / (0.058 * G4Log(z2exp) - 0.14);
   }
lines 826-853/3673 22%
```

We can avoid calling G4Log by replacing the condition with an equivalent one for the input argument of G4Log.

Unnecessary Work: Avoid Distant Data Accesses

```
   // Check if the particle has a force, EM or gravitational, exerted on it
   //
-  G4FieldManager* fieldMgr = 0;
-  G4bool fieldExertsForce  = false;

-  fieldMgr = fFieldPropagator->FindAndSetFieldManager(track.GetVolume());
   G4bool eligibleEM =
-    (particleCharge != 0.0) || (fUseMagneticMoment && (magneticMoment != 0.0));
-  G4bool eligibleGrav = fUseGravity && (restMass != 0.0);
+    (particleCharge != 0.0) || ((magneticMoment != 0.0) && fUseMagneticMoment);
+  G4bool eligibleGrav = (restMass != 0.0) && fUseGravity;

-  if((fieldMgr != nullptr) && (eligibleEM || eligibleGrav))
+  fFieldExertedForce = false;
+
+  if(eligibleEM || eligibleGrav)
   {
-    // User can configure the field Manager for this track
-    fieldMgr->ConfigureForTrack(&track);
-    // Called here to allow a transition from no-field pointer
-    // to finite field (non-zero pointer).
-
-    // If the field manager has no field ptr, the field is zero
-    //   by definition ( = there is no field ! )
-    const G4Field* ptrField = fieldMgr->GetDetectorField();
-    if(ptrField)
+    if(G4FieldManager* fieldMgr =
+         fFieldPropagator->FindAndSetFieldManager(track.GetVolume()))
     {
lines 2502-2530/3673 69%
```

Finding the field manager is expensive. It requires accessing distant pieces of data like the fFieldPropagator class member to call its method, and the track's current volume. However, we can avoid checking the field for neutral and/or massless particles, as the field has no effect on them.

Data Access Patterns: Group Nearby Reads & Writes

```
    G4Transportation: Move changes to fParticleChange closer together

diff --git a/source/processes/transportation/src/G4Transportation.cc b/source/processes/transportation/src/G4Transportation.cc
index 1fc97e1595..ba7a621299 100644
--- a/source/processes/transportation/src/G4Transportation.cc
+++ b/source/processes/transportation/src/G4Transportation.cc
@@ -195,12 +195,6 @@ G4double G4Transportation::AlongStepGetPhysicalInteractionLength(
    //
    *selection = CandidateForSelection;

-   fFirstStepInVolume = fNewTrack || fLastStepInVolume;
-   fLastStepInVolume  = false;
-   fNewTrack          = false;
-
-   fParticleChange.ProposeFirstStepInVolume(fFirstStepInVolume);
-
    // Get initial Energy/Momentum of the track
    //
    const G4DynamicParticle* pParticle       = track.GetDynamicParticle();
@@ -510,6 +504,11 @@ G4double G4Transportation::AlongStepGetPhysicalInteractionLength(
      }
    }

+   fFirstStepInVolume = fNewTrack || fLastStepInVolume;
+   fLastStepInVolume  = false;
+   fNewTrack          = false;
+
+   fParticleChange.ProposeFirstStepInVolume(fFirstStepInVolume);
    fParticleChange.ProposeTrueStepLength(geometryStepLength);
lines 2361-2389/3673 65%
```

If a class member needs to be accessed multiple times inside a function or method, prefer keeping these accesses close together to avoid unnecessary cache misses.

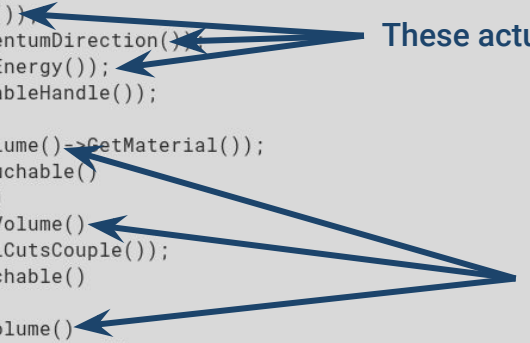
Data Access Patterns: Avoid Indirections

```
Author: Guilherme Amadio <amadio@cern.ch>

    G4SteppingManager: reuse value of fCurrentVolume to find current region

diff --git a/source/tracking/src/G4SteppingManager.cc b/source/tracking/src/G4SteppingManager.cc
index 0e00aca0d3..1108ef957d 100644
--- a/source/tracking/src/G4SteppingManager.cc
+++ b/source/tracking/src/G4SteppingManager.cc
@@ -257,9 +257,10 @@ G4StepStatus G4SteppingManager::Stepping()
   {
     fUserSteppingAction->UserSteppingAction(fStep);
   }
-  G4UserSteppingAction* regionalAction = fStep->GetPreStepPoint()
-        ->GetPhysicalVolume()->GetLogicalVolume()
-        ->GetRegion()->GetRegionalSteppingAction();
+
+  G4UserSteppingAction* regionalAction =
+    fCurrentVolume->GetLogicalVolume()->GetRegion()->GetRegionalSteppingAction();
+
   if(regionalAction)
     regionalAction->UserSteppingAction(fStep);


commit 3c70a7e614d885364905bb5208f6cefbff94c2d9
Author: Guilherme Amadio <amadio@cern.ch>
```

Chained accessors via pointers require several memory accesses to retrieve a single piece of data, with similar cost to traversing a linked list. Here we can avoid 3 access indirections by reusing the value of fCurrentVolume.

Data Access Patterns: Avoid Indirections

```
  // Initialize G4StepPoint attributes.
  // To avoid the circular dependency between G4Track, G4Step
  // and G4StepPoint, G4Step has to manage the copy actions.
  fpPreStepPoint->SetPosition(fpTrack->GetPosition());
  fpPreStepPoint->SetGlobalTime(fpTrack->GetGlobalTime());
  fpPreStepPoint->SetLocalTime(fpTrack->GetLocalTime());
  fpPreStepPoint->SetProperTime(fpTrack->GetProperTime());
  fpPreStepPoint->SetMomentumDirection(fpTrack->GetMomentumDirection());
  fpPreStepPoint->SetKineticEnergy(fpTrack->GetKineticEnergy());
  fpPreStepPoint->SetTouchableHandle(fpTrack->GetTouchableHandle());
  fpPreStepPoint->SetMaterial(
    fpTrack->GetTouchable()->GetVolume()->GetLogicalVolume()->GetMaterial());
  fpPreStepPoint->SetMaterialCutsCouple(fpTrack->GetTouchable()
                                        ->GetVolume()
                                        ->GetLogicalVolume()
                                        ->GetMaterialCutsCouple());
  fpPreStepPoint->SetSensitiveDetector(fpTrack->GetTouchable()
                                       ->GetVolume()
                                       ->GetLogicalVolume()
                                       ->GetSensitiveDetector());
  fpPreStepPoint->SetPolarization(fpTrack->GetPolarization());
  fpPreStepPoint->SetSafety(0.);
  fpPreStepPoint->SetStepStatus(fUndefined);
  fpPreStepPoint->SetProcessDefinedStep(0);
  fpPreStepPoint->SetMass(fpTrack->GetDynamicParticle()->GetMass());
  fpPreStepPoint->SetCharge(fpTrack->GetDynamicParticle()->GetCharge());
  fpPreStepPoint->SetWeight(fpTrack->GetWeight());

  // Set Velocity
"track/include/G4Step.icc" 306 lines --54%--                    166,1         59%
```



These actually return fDynamicParticle->Get...()!

Not good! Traverses pointer chains multiple times, and G4TouchableHandle is actually reference counted, so this has branches and is incrementing and decrementing counters multiple times too!

Data Access Patterns: Avoid Indirections

Source Line ▲	Source
166	// Initialize G4StepPoint attributes.
167	// To avoid the circular dependency between G4Track, G4Step
168	// and G4StepPoint, G4Step has to manage the copy actions.
169	fpPreStepPoint->SetPosition(fpTrack->GetPosition());
170	fpPreStepPoint->SetGlobalTime(fpTrack->GetGlobalTime());
171	fpPreStepPoint->SetLocalTime(fpTrack->GetLocalTime());
172	fpPreStepPoint->SetProperTime(fpTrack->GetProperTime());
173	fpPreStepPoint->SetMomentumDirection(fpTrack->GetMomentumDirection());
174	fpPreStepPoint->SetKineticEnergy(fpTrack->GetKineticEnergy());
175	fpPreStepPoint->SetTouchableHandle(fpTrack->GetTouchableHandle());
176	fpPreStepPoint->SetMaterial(
177	fpTrack->GetTouchable()->GetVolume()->GetLogicalVolume()->GetMaterial());
178	fpPreStepPoint->SetMaterialCutsCouple(fpTrack->GetTouchable()
179	->GetVolume()
180	->GetLogicalVolume()
181	->GetMaterialCutsCouple());
182	fpPreStepPoint->SetSensitiveDetector(fpTrack->GetTouchable()
183	->GetVolume()
184	->GetLogicalVolume()
185	->GetSensitiveDetector());
186	fpPreStepPoint->SetPolarization(fpTrack->GetPolarization());
187	fpPreStepPoint->SetSafety(0.);
188	fpPreStepPoint->SetStepStatus(fUndefined);
189	fpPreStepPoint->SetProcessDefinedStep(0);
190	fpPreStepPoint->SetMass(fpTrack->GetDynamicParticle()->GetMass());
191	fpPreStepPoint->SetCharge(fpTrack->GetDynamicParticle()->GetCharge());
192	fpPreStepPoint->SetWeight(fpTrack->GetWeight());
193	
194	// Set Velocity
195	// should be placed after SetMaterial for preStep point
196	fpPreStepPoint->SetVelocity(fpTrack->CalculateVelocity());
197	
198	(*fpPostStepPoint) = (*fpPreStepPoint);
199	}
200	

Address ▲	Source Line	Assembly	Clockticks	Instructions Retired	CPI Rate	Retiring
0x2bd92	176	callq 0xf580	1,674,000,000	1,872,000,000	0.894	12.6%
0x2bd97		**Block 8:**				
0x2bd97	176	movq %rax, 0x60(%r15)				
0x2bd9b	178	movq 0x28(%rbx), %rax				
0x2bd9f	178	movq 0x10(%rbx), %r12				
0x2bda3	178	movq 0x38(%rax), %rax				
0x2bda7	178	test %rax, %rax				
0x2bdaa	178	jz 0x2bfb3 <Block 40>				
0x2bdb0		**Block 9:**				
0x2bdb0	178	movq 0x8(%rax), %rdi				
0x2bdb4	178	xor %esi, %esi	0	18,000,000	0.000	0.0%
0x2bdb6	179	movq (%rdi), %rax				
0x2bdb9	178	callq 0x20(%rax)	0	54,000,000	0.000	0.0%
0x2bdbc		**Block 10:**				
0x2bdbc	178	movq 0x10(%rax), %rdi	36,000,000	0		0.0%
0x2bdc0	178	callq 0xf690	684,000,000	504,000,000	1.357	3.1%
0x2bdc5		**Block 11:**				
0x2bdc5	178	movq %rax, 0x68(%r12)				
0x2bdca	182	movq 0x28(%rbx), %rax				
0x2bdce	182	movq 0x10(%rbx), %r12				
0x2bdd2	182	movq 0x38(%rax), %rax				
0x2bdd6	182	test %rax, %rax				
0x2bdd9	182	jz 0x2bfb3 <Block 40>				
0x2bddf		**Block 12:**				
0x2bddf	182	movq 0x8(%rax), %rdi				
0x2bde3	182	xor %esi, %esi				
0x2bde5	183	movq (%rdi), %rax				
0x2bde8	182	callq 0x20(%rax)	54,000,000	36,000,000	1.500	0.0%
0x2bdeb		**Block 13:**				
0x2bdeb	182	movq 0x10(%rax), %rdi	0	0	0.000	0.0%
0x2bdef	182	callq 0xf9e0	720,000,000	540,000,000	1.333	0.0%
0x2bdf4		**Block 14:**				
0x2bdf4	182	movq 0x28(%rbx), %rdi	0	0	0.000	0.0%
0x2bdf8	182	movq %rax, 0x70(%r12)				
0x2bdfd	186	movq 0x50(%rdi), %r12				

Arithmetics: Avoid Data Dependencies

```
for (G4int i=0; i<8; ++i)
{
  G4double a4 = G4Exp(tmn*xgi[i]);     // a4 = (1.-asymmetry)
  G4double a5 = a4*(2.-a4) ;
  G4double a6 = 1.-a5 ;
  G4double a7 = 1.+a6 ;
  G4double a9 = 3.+a6 ;
  G4double xi = xi0*a5 ;
  G4double xii = 1./xi ;
  G4double xi1 = 1.+xi ;
  G4double screen = screen0*xi1/a5 ;
  G4double yeu = 5.-a6+4.*bet*a7 ;
  G4double yed = 2.*(1.+3.*bet)*G4Log(3.+xii)-a6-a1*(2.-a6) ;
  G4double ye1 = 1.+yeu/yed ;
  G4double ale = G4Log(bbb/z13*sqrt(xi1*ye1)/(1.+screen*ye1)) ;
  G4double cre = 0.5*G4Log(1.+2.25*z23*xi1*ye1/massratio2) ;
  G4double be;

  if (xi <= 1.e3) {
    be = ((2.+a6)*(1.+bet)+xi*a9)*G4Log(1.+xii)+(a5-bet)/xi1-a9;
  } else {
    be = (3.-a6+a1*a7)/(2.*xi);
  }
  G4double fe = (ale-cre)*be;
  if ( fe < 0.) fe = 0. ;

  G4double ymu = 4.+a6 +3.*bet*a7 ;
  G4double ymd = a7*(1.5+a1)*G4Log(3.+xi)+1.-1.5*a6 ;
  G4double ym1 = 1.+ymu/ymd ;
```

363,26 57%

Data dependencies between arithmetic operations can create execution latency even without cache misses.

Breaking up long loops into smaller parts makes it possible to hide some of the latency from divisions and math function calls with instruction level parallelism.

Arithmetics: Avoid Data Dependencies

```
-  for (G4int i=0; i<8; ++i)
+  G4double rho[8];
+  G4double rho2[8];
+  G4double xi[8];
+  G4double xi1[8];
+  G4double xii[8];
+
+  for (G4int i = 0; i < 8; ++i)
   {
-    G4double rho = G4Exp(tmn*xgi[i]) - 1.0; // rho = -asymmetry
-    G4double rho2 = rho * rho;
-    G4double xi = xi0*(1.0-rho2) ;
-    G4double xii = 1./xi ;
-    G4double xi1 = 1.+xi ;
-    G4double screen = screen0*xi1/(1.0-rho2) ;
-    G4double yeu = 5.-rho2+4.*bet*(1.0+rho2) ;
-    G4double yed = 2.*(1.+3.*bet)*G4Log(3.+xii)-rho2-a1*(2.-rho2) ;
+    rho[i] = G4Exp(tmn*xgi[i]) - 1.0; // rho = -asymmetry
+    rho2[i] = rho[i] * rho[i];
+    xi[i] = xi0*(1.0-rho2[i]);
+    xi1[i] = 1.0 + xi[i];
+    xii[i] = 1.0 / xi[i];
+  }
+
+  for (G4int i = 0; i < 8; ++i)
+  {
+    G4double screen = screen0*xi1[i]/(1.0-rho2[i]) ;
+    G4double yeu = 5.-rho2[i]+4.*bet*(1.0+rho2[i]) ;
+    G4double yed = 2.*(1.+3.*bet)*G4Log(3.+xii[i])-rho2[i]-a1*(2.-rho2[i]) ;
lines 1088-1116
```

Data dependencies between arithmetic operations can create execution latency even without cache misses.

Breaking up long loops into smaller parts makes it possible to hide some of the latency from divisions and math function calls with instruction level parallelism.

G4PhysicsVector::Interpolation()

Source Line ▲	Source	Clockticks	Instructions Retired	CPI Rate	Locators				
					Retiring	Front-End Bound	Bad Speculation	Back-End Bound	
								Memory Bound	Core Bound
184	`// ------------------------------------------------------------------`								
185									
186	`inline G4double G4PhysicsVector::Interpolation(const std::size_t idx,`								
187	`                                               const G4double e) const`								
188	`{`								
189	`  // perform the interpolation`								
190	`  const G4double x1 = binVector[idx];`	0.2%	0.1%	1.011	3.8%	6.0%	3.4%	3.9%	3.8%
191	`  const G4double d1 = binVector[idx + 1] - x1;`								
192	`  // note: all corner cases of the previous methods are covered and eventually`								
193	`  //       gives b=0/1 that results in y=y0\y_{N-1} if e<=x[0]/e>=x[N-1] or`								
194	`  //       y=y_i/y_{i+1} if e<x[i]/e>=x[i+1] due to small numerical errors`								
195	`  const G4double b = std::max(0., std::min(1., (e - x1) / d1));`	0.0%	0.0%	0.199	0.3%	0.0%	0.0%	0.0%	0.0%
196	`  G4double res;`								
197	`  if(useSpline)  // spline interpolation`	0.0%	0.0%	1.000	0.4%	0.0%		0.4%	
198	`  {`								
199	`    const G4double os = 0.166666666667;  // 1./6.`								
200	`    const G4double a  = 1.0 - b;`	0.1%	0.0%	3.851	1.4%	0.0%	0.7%	2.3%	2.3%
201	`    const G4double c0 = (a * a * a - a) * secDerivative[idx];`	0.1%	0.0%	4.105	0.5%	0.0%	0.9%	1.9%	1.1%
202	`    const G4double c1 = (b * b * b - b) * secDerivative[idx + 1];`	0.1%	0.0%	5.371	1.3%	0.0%	2.0%	3.6%	3.8%
203	`    res =`								
204	`      a * dataVector[idx] + b * dataVector[idx + 1] + (c0 + c1) * d1 * d1 * os;`	0.6%	0.3%	1.240	9.7%	0.0%	3.7%	17.4%	23.9%
205	`  }`								
206	`  else  // linear interpolation`								
207	`  {`								
208	`    const G4double y1 = dataVector[idx];`								
209	`    const G4double y2 = dataVector[idx + 1];`								
210	`    res               = y1 + b * (y2 - y1);`	0.0%	0.0%	7.667	0.0%	1.5%	0.0%	0.1%	0.0%
211	`  }`								
212	`  return res;`								
213	`}`								

Arithmetics: Instruction Level Parallelism

```
+++ b/source/global/management/include/G4PhysicsVector.icc
@@ -129,10 +129,10 @@ inline G4double G4PhysicsVector::Interpolation(const std::size_t idx,
 {
   // perform the interpolation
   const G4double x1 = binVector[idx];
-  const G4double dl = binVector[idx + 1] - binVector[idx];
+  const G4double dl = binVector[idx + 1] - x1;

   const G4double y1 = dataVector[idx];
-  const G4double dy = dataVector[idx + 1] - dataVector[idx];
+  const G4double dy = dataVector[idx + 1] - y1;

   // note: all corner cases of the previous methods are covered and eventually
   //       gives b=0/1 that results in y=y0\y_{N-1} if e<=x[0]/e>=x[N-1] or
@@ -143,10 +143,9 @@ inline G4double G4PhysicsVector::Interpolation(const std::size_t idx,

   if(useSpline)  // spline interpolation
   {
-    const G4double a  = 1.0 - b;
-    const G4double c0 = (a * a * a - a) * secDerivative[idx];
-    const G4double c1 = (b * b * b - b) * secDerivative[idx + 1];
-    res += (c0 + c1) * dl * dl * (1.0/6.0);
+    const G4double c0 = (2.0 - b) * secDerivative[idx];
+    const G4double c1 = (1.0 + b) * secDerivative[idx + 1];
+    res += (b * (b - 1.0)) * (c0 + c1) * (dl * dl * (1.0/6.0));
   }

   return res;
lines 3626-3654/3673 100%
```

Refactoring terms saves some multiplications, but note also the parenthesis. Floating point arithmetics is not associative. Parenthesizing ensures that each of the independent multiplications can be performed in parallel. This alone reduces estimated execution from 60 to 51 cycles (llvm-mca).

Without parenthesis

```
vmovsd xmm5, QWORD PTR .LC0[rip]
vmovsd xmm4, QWORD PTR .LC2[rip]
vsubsd xmm2, xmm1, xmm5
vsubsd xmm4, xmm4, xmm1
vmulsd xmm2, xmm2, xmm1
vaddsd xmm1, xmm1, xmm5
vmulsd xmm1, xmm1, QWORD PTR [rcx+8+rdi]
vfmadd231sd xmm1, xmm4, QWORD PTR [rcx+rax*8]
vmulsd xmm1, xmm2, xmm1
vmulsd xmm1, xmm1, xmm3
vmulsd xmm1, xmm1, xmm3
vfmadd231sd xmm0, xmm1, QWORD PTR .LC1[rip]
```

same register ⇒ sequential

With parenthesis

```
vmovsd xmm5, QWORD PTR .LC0[rip]
vmovsd xmm4, QWORD PTR .LC2[rip]
vsubsd xmm3, xmm1, xmm5
vsubsd xmm4, xmm4, xmm1
vmulsd xmm2, xmm2, xmm2
vmulsd xmm3, xmm3, xmm1
vaddsd xmm1, xmm1, xmm5
vmulsd xmm1, xmm1, QWORD PTR [rcx+8+rdi]
vmulsd xmm2, xmm2, QWORD PTR .LC1[rip]
vfmadd132sd xmm4, xmm1, QWORD PTR [rcx+rax*8]
vmulsd xmm3, xmm3, xmm4
vfmadd231sd xmm0, xmm3, xmm2
```

different registers ⇒ parallel

Arithmetics: Instruction Level Parallelism

```
                    0123456789          0123456789          0123456789
Index     0123456789          0123456789          0123456789

[0,0]     DeER .    .    .    .    .    .    .    .    .    .    .   .   movslq %edi, %rax
[0,1]     D=eER.    .    .    .    .    .    .    .    .    .    .   .   leaq   (,%rax,8), %rdi
[0,2]     D=eeeeeER .    .    .    .    .    .    .    .    .    .   .   vmovsd (%rsi,%rax,8), %xmm1
[0,3]     D==eeeeeER.    .    .    .    .    .    .    .    .    .   .   vmovsd 8(%rsi,%rdi), %xmm3
[0,4]     D=eeeeeE-R.    .    .    .    .    .    .    .    .    .   .   vmovsd (%rdx,%rax,8), %xmm2
[0,5]     D=======eeeeER .    .    .    .    .    .    .    .    .   .   vsubsd %xmm1, %xmm3, %xmm3
[0,6]     .D=====eeeeE-R .    .    .    .    .    .    .    .    .   .   vsubsd %xmm1, %xmm0, %xmm1
[0,7]     .D=eeeeeE----R .    .    .    .    .    .    .    .    .   .   vmovsd 8(%rdx,%rdi), %xmm0
[0,8]     .D==========eeeeeeeeeeeeER  .    .    .    .    .    .   .   vdivsd %xmm3, %xmm1, %xmm1
[0,9]     .D======eeeeE-------------R  .    .    .    .    .    .   .   vsubsd %xmm2, %xmm0, %xmm0
[0,10]    .D========================eeeeER   .    .    .    .    .   .   vfmadd132sd    %xmm1, %xmm2, %xmm0
[0,11]    .DeE--------------------------R   .    .    .    .    .   .   testb  %r8b, %r8b
[0,12]    . DeE-------------------------R   .    .    .    .    .   .   je     .L1
[0,13]    . D=eeeeeE--------------------R   .    .    .    .    .   .   vmovsd .LC0(%rip), %xmm5
[0,14]    . D=eeeeeE--------------------R   .    .    .    .    .   .   vmovsd .LC1(%rip), %xmm4
[0,15]    . D======================eeeeER   .    .    .    .    .   .   vsubsd %xmm5, %xmm1, %xmm2
[0,16]    . D=========================eeeeER  .    .    .    .    .   .   vsubsd %xmm1, %xmm4, %xmm4
[0,17]    . D===========================eeeeER    .    .    .    .   .   vmulsd %xmm1, %xmm2, %xmm2
[0,18]    .  D=======================eeeeE---R    .    .    .    .   .   vaddsd %xmm5, %xmm1, %xmm1
[0,19]    .  D=========================eeeeeeeeeER .    .    .    .   .   vmulsd 8(%rcx,%rdi), %xmm1, %xmm1
[0,20]    .  D============================eeeeeeeeeER  .    .    .   .   vfmadd231sd    (%rcx,%rax,8), %xmm4, %xmm1
[0,21]    .  D======================================eeeeER   .    .   .   vmulsd %xmm1, %xmm2, %xmm1
[0,22]    .   D=======================================eeeeER    .    .   vmulsd %xmm3, %xmm1, %xmm1
[0,23]    .   D================================================eeeeER.   .   vmulsd %xmm3, %xmm1, %xmm1
[0,24]    .   D=============================================eeeeeeeeeER   vfmadd231sd    .LC2(%rip), %xmm1, %xmm0
[0,25]    .    DeeeeeeeE-----------------------------------------R   retq
titanx ~ $ g++-11.2.0 -march=native -O3 -S test.cc -DV2=1 -o - | llvm-mca -iterations=1 -timeline 2>/dev/null | sed -n 97,125p
```

Legend

```
D : Instruction dispatched.

e : Instruction executing.

E : Instruction executed.

R : Instruction retired.

= : Instruction already dispatched, waiting to be executed.

- : Instruction executed, waiting to be retired.
```

Arithmetics: Instruction Level Parallelism

```
                    0123456789          0123456789          0
Index     0123456789          0123456789          0123456789

[0,0]     DeER .    .    .    .    .    .    .    .    .    .   movslq  %edi, %rax
[0,1]     D=eER.    .    .    .    .    .    .    .    .    .   leaq    (,%rax,8), %rdi
[0,2]     D=eeeeeER .    .    .    .    .    .    .    .    .   vmovsd  (%rsi,%rax,8), %xmm1
[0,3]     D==eeeeeER.    .    .    .    .    .    .    .    .   vmovsd  8(%rsi,%rdi), %xmm2
[0,4]     D=eeeeeE-R.    .    .    .    .    .    .    .    .   vmovsd  (%rdx,%rax,8), %xmm3
[0,5]     D=======eeeeER .    .    .    .    .    .    .    .   vsubsd  %xmm1, %xmm2, %xmm2
[0,6]     .D=====eeeeE-R .    .    .    .    .    .    .    .   vsubsd  %xmm1, %xmm0, %xmm1
[0,7]     .D=eeeeeE----R .    .    .    .    .    .    .    .   vmovsd  8(%rdx,%rdi), %xmm0
[0,8]     .D==========eeeeeeeeeeeeeER  .    .    .    .    .   vdivsd  %xmm2, %xmm1, %xmm1
[0,9]     .D======eeeeE-------------R  .    .    .    .    .   vsubsd  %xmm3, %xmm0, %xmm0
[0,10]    .D========================eeeeER   .    .    .    .   vfmadd132sd     %xmm1, %xmm3, %xmm0
[0,11]    .DeE--------------------------R   .    .    .    .   testb   %r8b, %r8b
[0,12]    . DeE-------------------------R   .    .    .    .   je      .L1
[0,13]    . D=eeeeeE--------------------R   .    .    .    .   vmovsd  .LC0(%rip), %xmm5
[0,14]    . D=eeeeeE--------------------R   .    .    .    .   vmovsd  .LC1(%rip), %xmm4
[0,15]    . D======================eeeeER   .    .    .    .   vsubsd  %xmm5, %xmm1, %xmm3
[0,16]    . D==========================eeeeER  .    .    .    .   vsubsd  %xmm1, %xmm4, %xmm4
[0,17]    . D=========eeeeE--------------R  .    .    .    .   vmulsd  %xmm2, %xmm2, %xmm2
[0,18]    .  D=========================eeeeER    .    .    .   vmulsd  %xmm1, %xmm3, %xmm3
[0,19]    .  D=======================eeeeE---R    .    .    .   vaddsd  %xmm5, %xmm1, %xmm1
[0,20]    .  D=========================eeeeeeeeeER .    .    .   vmulsd  8(%rcx,%rdi), %xmm1, %xmm1
[0,21]    .  D=========eeeeeeeeeE--------------R .    .    .   vmulsd  .LC2(%rip), %xmm2, %xmm2
[0,22]    .   D=========================eeeeeeeeeER  .    .   vfmadd132sd     (%rcx,%rax,8), %xmm1, %xmm4
[0,23]    .   D=======================================eeeeER   .   vmulsd  %xmm4, %xmm3, %xmm3
[0,24]    .   D=======================================eeeeER   vfmadd231sd     %xmm2, %xmm3, %xmm0
[0,25]    .    DeeeeeeeE-----------------------------------R   retq
titanx ~ $ g++-11.2.0 -march=native -O3 -S test.cc -DV3=1 -o - | llvm-mca -iterations=1 -timeline 2>/dev/null | sed -n 97,125p_
```

Legend

```
D : Instruction dispatched.

e : Instruction executing.

E : Instruction executed.

R : Instruction retired.

= : Instruction already dispatched, waiting to be executed.

- : Instruction executed, waiting to be retired.
```

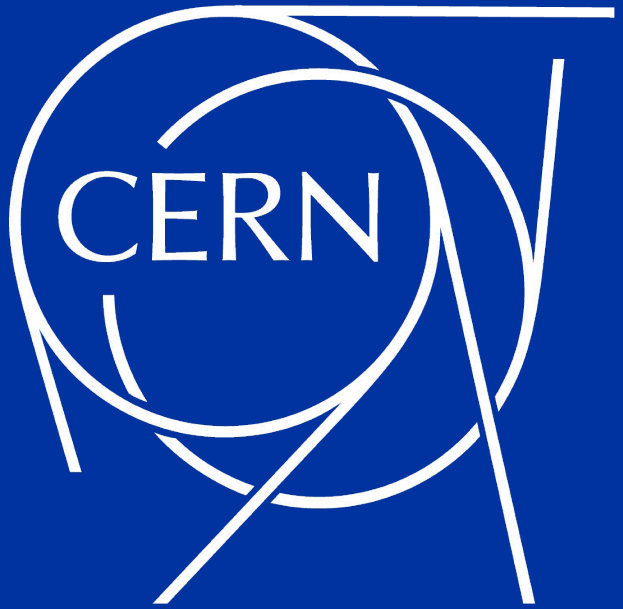
Summary

Frontend Optimizations

- Reduce code size, code duplication
- Check for excessive inlining
- Avoid unnecessary functions calls
- Avoid frequent calls to distant code
  - Merge libraries that call each other frequently
  - Place functions that call each other nearby
- Optimize conditionals
  - Order by true/false probability
  - Replace conditions by arithmetics
- Use SIMD vectorization (less instructions)

Backend Optimizations

- Group nearby data accesses
- Prefer regular data members to pointers
- Avoid indirections from accessor chains
- Break up long for loops into smaller ones
- Avoid data dependencies in arithmetics
- Check cache performance, hit/miss rates
- Be conservative with your assumptions about what the compiler can optimize


CERN