Hadronic Cross Section Speedup

A. Dotti ; SLAC SD/EPP/Computing

https://indico.cern.ch/event/332535/contribution/2/material/slides/0.pdf

GEANT4 Hadronic Cross Section Optimizations

Robert Fowler and Paul Ruth
RENCI / UNC Chapel Hill

Pedro Diniz
ISI / USC

USC Viterbi School of Engineering
Information Sciences Institute

renci
RESEARCH ENGAGEMENT INNOVATION

Hadronic interactions and cross-sections

Consider a G4Track being propagate, for all hadronic processes we need to interrogate cross-sections twice: to calculate PIL and, when/if interaction happens we need to select the nucleus (and possibly isotope) on which interaction happens (sampleZandA method)

Note that to calculate (total) cross-section needed in first step, we anyway need to sum over all nucleus cross-sections

An optimization has always been present: iff triplet {particle,energy,material} is used twice in a row, no need to recalculate. Note that this happen only in some special conditions (i.e. neutral particle)

Proposal

ASCR proposed two optimizations:

- observation that triplets {particle,material,energy} tend to repeat, create a real cache that holds a set of these triplets: an extension of what has been always there
- build a surrogate model based on the Douglas-Peucker method (create histogram of the total cross-section), to be used when total cross-section is needed (for PIL calculation)

Motivation

How can we reduce CPU time?

Better caching to re-use calculations

Second strategy: provide a "fast surrogate model"

Potentially change physics results. Thus provide API to turn on/of selectively

- Hadronic CrossSections
  - ~10% of total wall clock time*
  - Deep call chain with no hot spots
    - Reduce call chain length
    - Reduce time spent in calculation

*51 real events simulated in ~2 hours (provided by Soon)

Scope	WALLCLOCK (us).[0,0] (E)	WALLCLOCK (us).[0,0] (I)
Experiment Aggregate Metrics	1.29e+09 100 %	1.29e+09 100 %
main		1.29e+09 100 %
131: G4UImanager::ApplyCommand(char const*)		1.28e+09 99.3%
422: G4UIcommand::DoIt(G4String)		1.28e+09 99.3%
210: G4UIcontrolMessenger::SetNewValue(G4UIcommand*, G4String)		1.28e+09 99.3%
277: G4UImanager::ExecuteMacroFile(char const*)		1.28e+09 99.3%
234: G4UIbatch::SessionStart()		1.28e+09 99.3%
215: G4UIbatch::ExecCommand(G4String const&)		1.28e+09 99.3%
170: G4UImanager::ApplyCommand(char const*)		1.28e+09 99.3%
422: G4UIcommand::DoIt(G4String)		1.28e+09 99.3%
210: G4RunMessenger::SetNewValue(G4UIcommand*, G4String)		1.28e+09 99.3%
287: G4RunManager::BeamOn(int, char const*, int)		1.28e+09 99.3%
155: G4RunManager::DoEventLoop(int, char const*, int)		1.26e+09 97.1%
237: G4RunManager::ProcessOneEvent(int)		1.26e+09 97.1%
264: G4EventManager::DoProcessing(G4Event*)	9.34e+05 0.1%	1.26e+09 97.1%
185: G4TrackingManager::ProcessOneTrack(G4Track*)	2.98e+06 0.2%	1.25e+09 96.6%
125: G4SteppingManager::Stepping()	5.61e+06 0.4%	1.17e+09 90.8%
180: G4VProcess::AlongStepGPIL(G4Track const&, double, double, double&, G4GPILSelection*)	2.27e+05 0.0%	5.64e+08 43.6%
458: G4Transportation::AlongStepGetPhysicalInteractionLength(G4Track const&, double, double, double&, G4GPILSelection*)	9.47e+06 0.7%	5.20e+08 40.2%
321: G4PropagatorInField::ComputeStep(G4FieldTrack&, double, double&, G4VPhysicalVolume*)	1.00e+07 0.8%	3.45e+08 26.7%
286: G4ChordFinder::AdvanceChordLimited(G4FieldTrack&, double, double, CLHEP::Hep3Vector, double)	2.51e+06 0.2%	2.29e+08 17.7%
202: G4MagInt_Driver::AccurateAdvance(G4FieldTrack&, double, double, double)	2.40e+06 0.2%	1.17e+08 9.1%
185: G4ChordFinder::FindNextChord(G4FieldTrack const&, double, G4FieldTrack&, double&, double, double*, CLHEP::He	4.28e+06 0.3%	9.35e+07 7.2%
323: G4MultiLevelLocator::EstimateIntersectionPoint(G4FieldTrack const&, G4FieldTrack const&, CLHEP::Hep3Vector const	2.29e+06 0.2%	1.02e+08 7.9%
210: G4ChordFinder::ApproxCurvePointV(G4FieldTrack const&, G4FieldTrack const&, CLHEP::Hep3Vector const&, doubl	1.02e+06 0.1%	6.01e+07 4.6%
298: G4VIntersectionLocator::IntersectChord(CLHEP::Hep3Vector const&, CLHEP::Hep3Vector const&, double&, double	8.82e+05 0.1%	1.52e+07 1.2%

renci

USC Viterbi School of Engineering
Information Sciences Institute

Cross Section Usage

- Particle/Material/Process Triples
  - 50% of cycles in ~10 triples
  - 90% of cycles in ~85 triples
  - Total ~18k triples

Existing CrossSection Calculation

```
G4double G4CrossSectionDataStore::GetCrossSection(part,mat){
  ...
    if(mat == currentMaterial && part->GetDefinition() == matParticle
     && part->GetKineticEnergy() == matKinEnergy)
    { return matCrossSection; }

   //Calculate CrossSection the regular way (including xsecelm)
  ...
}

G4double G4CrossSectionDataStore::SampleZandA(part,mat){
  ...
    G4double cross = GetCrossSection(part, mat);
  ...
}
```

Be as general as possible:
intercept call at highest level
G4CrossSectionDataStore::GetCrossSection(part,mat)
Reminder: each process has its own instance

renci

USC Viterbi School of Engineering
Information Sciences Institute

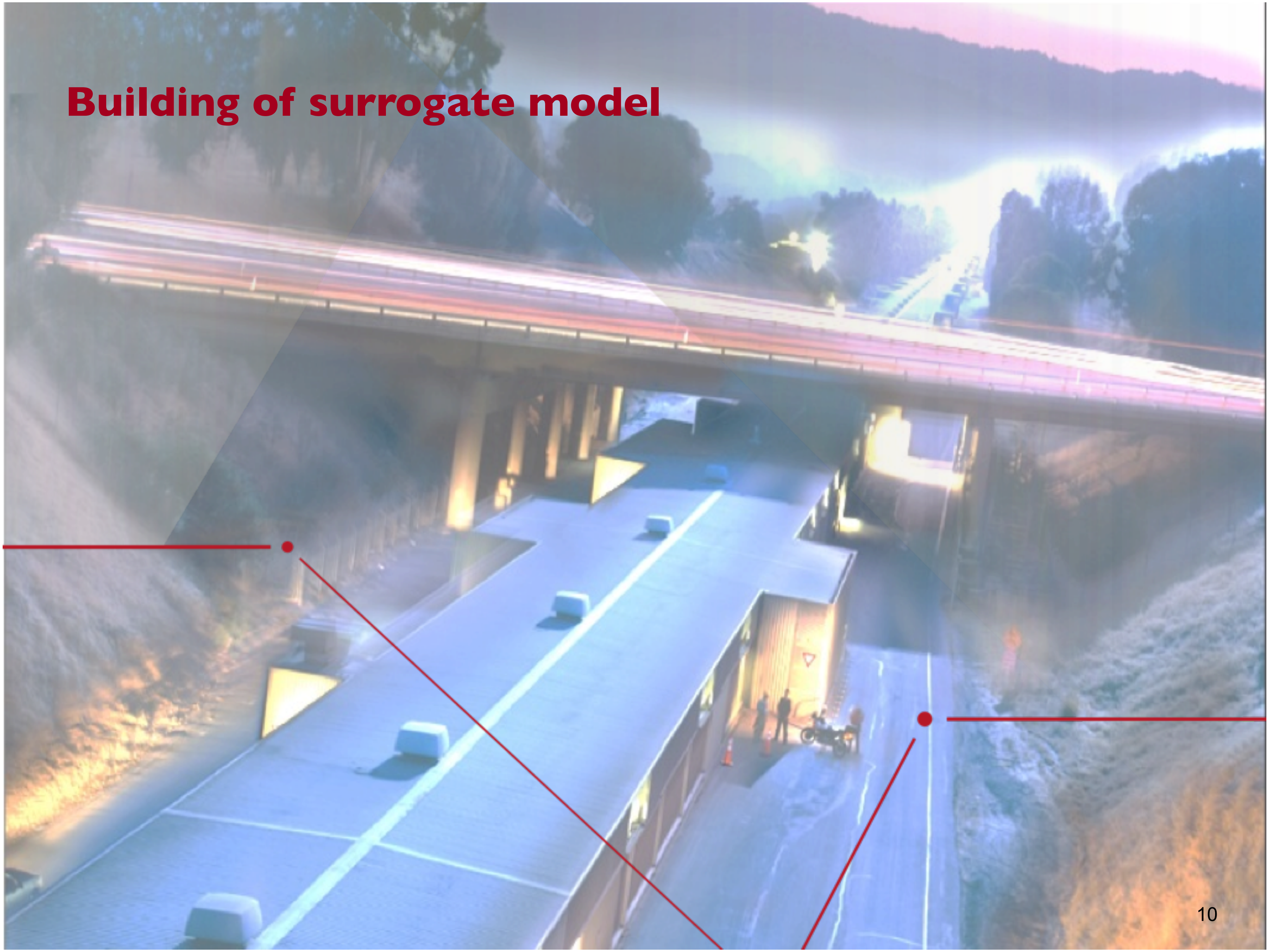
Caching CrossSection Results

- Observation
  - Multiple calls to GetCrossSection with exactly the same particle, material, process, and energy
  - Results in same cross section value
  - True even though energy is a double! (the physics is causing this)
- Optimization
  - Cache recent cross section for particle, material, process triple.
- Measurements
  - 17% of calls would benefit from this cache
  - 29% of GetCrossSection cycles are from these calls.
  - ~18k triples total
  - ~3k triples would benefit

Modified CrossSection Calculation

```
G4double G4CrossSectionDataStore::GetCrossSection(part,mat){
  ...
    entry = process_cache_map[(part,mat)];
    if(entry->energy == part->GetKineticEnergy()){
       xsecelm = entry->xsecelm;
       crossSection = entry->crossSection;
    } else
       //Calculate CrossSection the regular way (including xsecelm)
       ...
       entry->xsecelm = xsecelm;
       entry->crossSection = crossSection;
    }
    return crossSection;
}
```

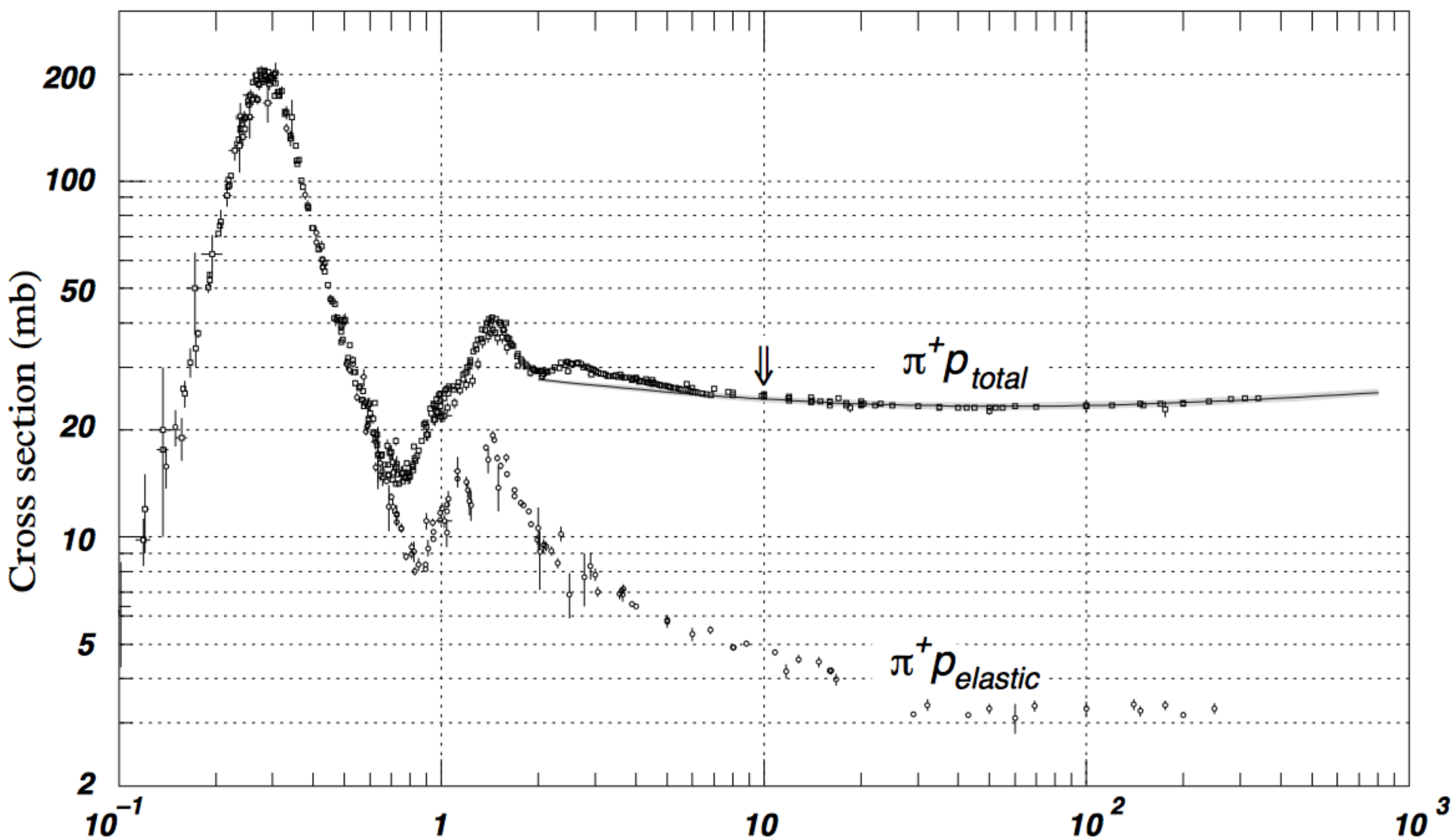
Building of surrogate model



How to build XS table

Define a "precision target"

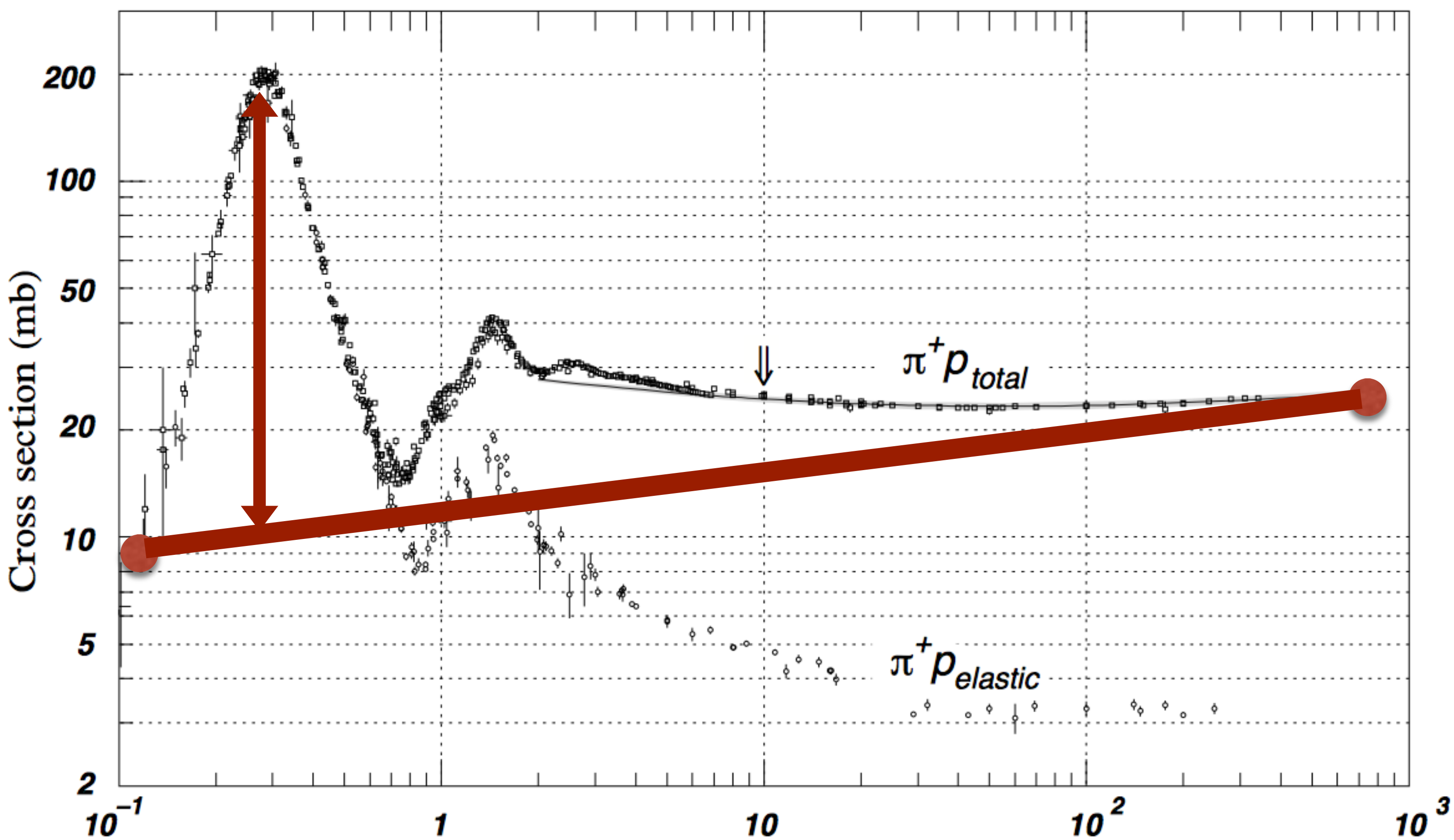
Oversimplification: please excuse my imprecision!



How to build XS table

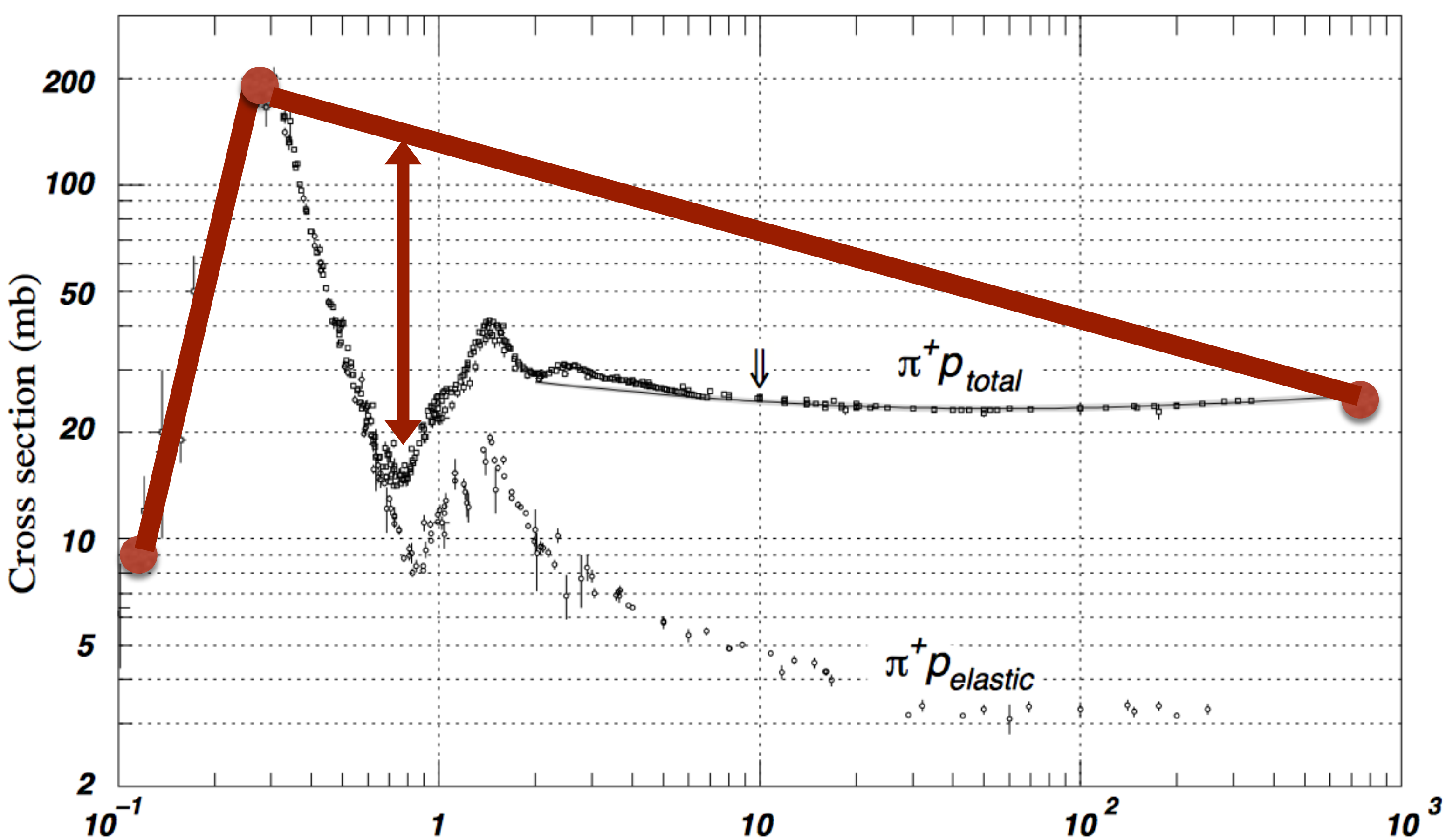
Interpolate and search maximum distance > precision target

Add a point there



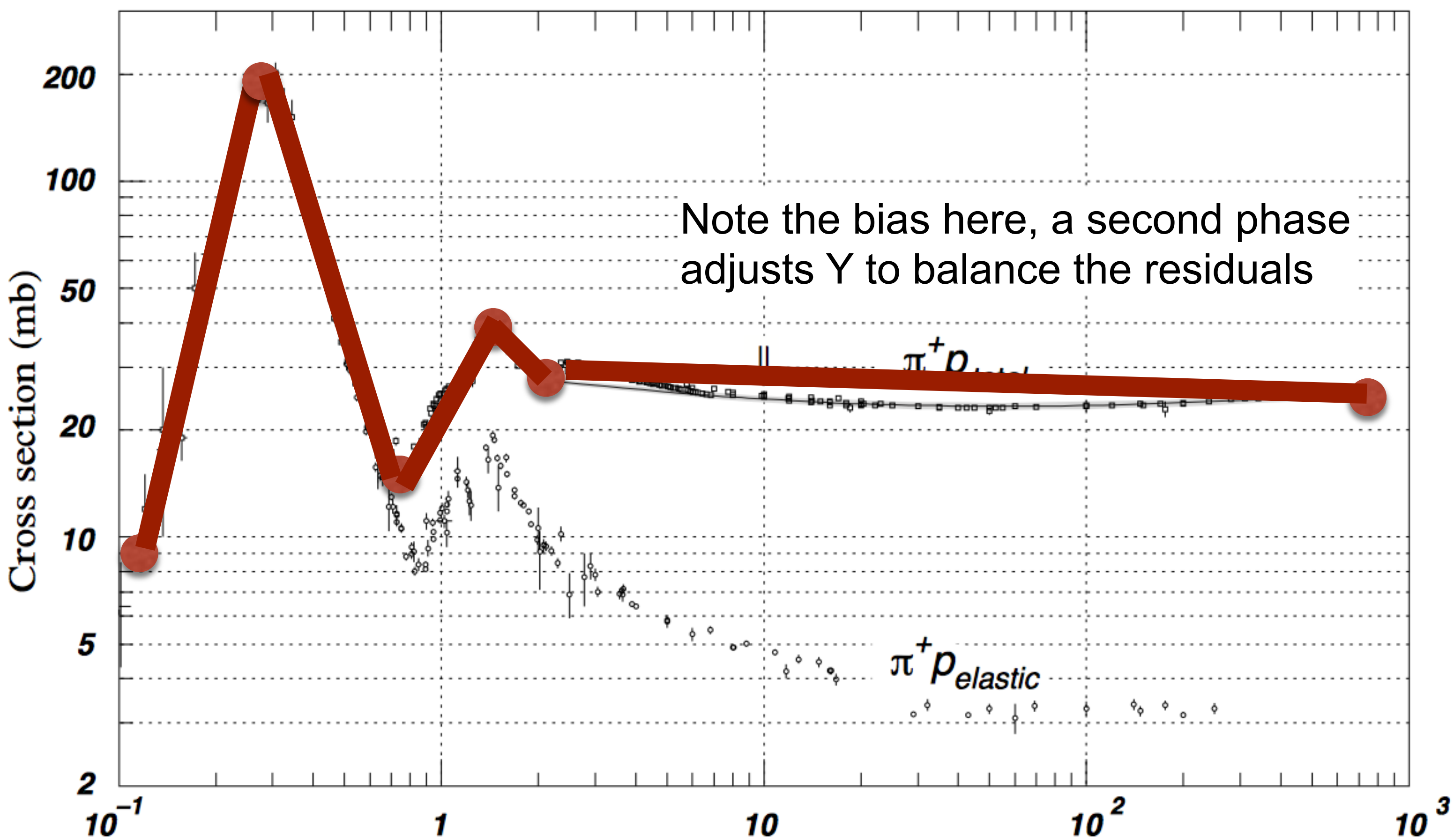
How to build XS table

Repeat



How to build XS table

Repeat until all added points distance < precision target



Fast Path Usage

Particle: neutron
Material: materials_StainlessSteel
Process: G4Neutron InelasticXS

Slow path only:

	Cycles	Calls	Cycles/Call
Slow Path	6,133,110,476	6,278,517	977

Fast path with lazy computation of slow path:

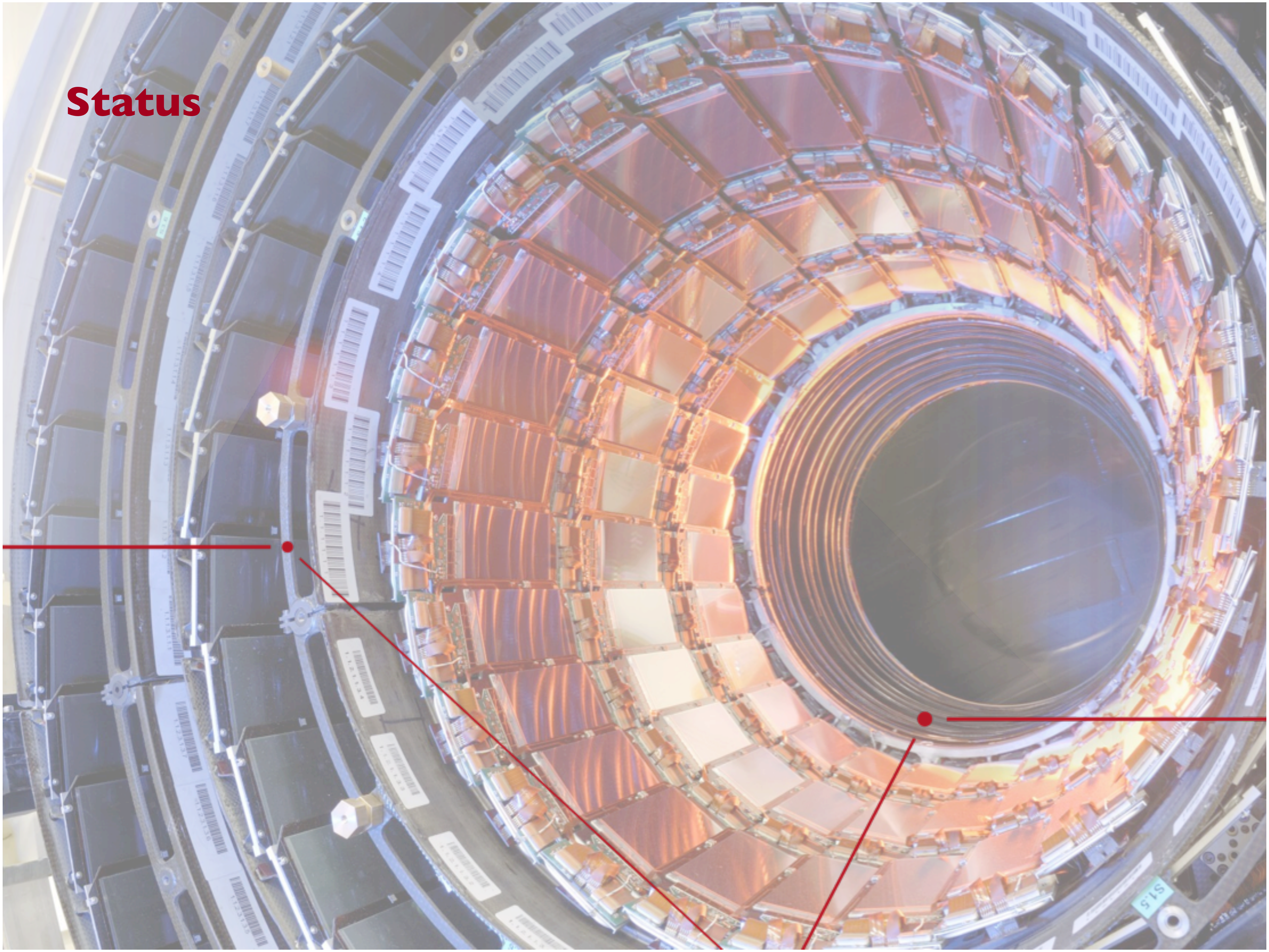
	Cycles	Calls	Cycles/Call
Slow Path	223,362,860	94,876	2,354
Fast Path	1,059,541,332	5,887,001	179
Total	1,282,904,192	5,981,877	214

Possible ~5x speed up of cross section calculation

renci

USC Viterbi School of Engineering Information Sciences Institute

Status



Inclusion in Geant4

Prototype code from ASCR provided before the summer

Integration in Geant4 source code is underway:

- code APIs has been rewritten to remove "C-style" functions
- integration w/ Geant4 structure done: code will be added to processes/hadronic/cross_sections
  - one new class added: G4FastPathCrossSection
  - G4CrossSectionDataStore modified

Details on algorithm

To minimize memory usage, only the total cross-section for a given G4Material is stored in the fast-path

- the same method G4CrossSectionDataStore::GetCrossSection(const G4DynamicParticle* , const G4Material*) is called both to calculate total cross-section for PIL calcualtion and to sample target nucleus (from sampleZandA)
- in second case we cannot avoid the slow-path unless previous call to the method was already a slow-path

Logic: call to G4CrossSectionDataStore::GetCrossSection(…) iff functionality is enabled and call does not come from sampleZandA, fast path is possible:

1. Step 0: super-fast-path (old): iff {particle,material,energy} is exactly the same as previous call, return immediately, else
2. Step 1: check if {particle,material,energy} is in cache, return, else
3. Step 2: use energy for fast-path corresponding {particle,material}, and add result to cache else
4. Step 3: slow-path

Missing functionalities

Preparation/filling of fast-path surrogate

By default this functionality is turned off, can be activated for a given triplet {processName,particleType,material}

- UI commands and C++ API will be created

At initialization, for active triplets the surrogate model is built: sample cross-section for given energy and build surrogate model

Require modifications to G4HadronicProcess interface

SimplifiedCalorimeter will be used for initial physics validation:

- compare results with feature on and off

FullCMS will be used for performance evaluation and profiling

Tentative schedule

Goal is to provide feature for 10.2

Time is tight for validation, in case move to 2016

- ATLAS has expressed interested in the feature

[redacted]