Merlin++
A flexible and feature-rich library for accelerator simulations

Roger Barlow
Sam Tygier and Scott Rowan
The University of Huddersfield

13th November 2020
What it’s all about

- What is Merlin++?
  - A short history
  - Some examples
  - Where Merlin belongs
- The big picture
  - A library not a program
  - A truly Object Oriented program
  - Using the C++ compiler
  - Writing solid software
- Performance and Features
- Getting started
What it’s all about
A short history

History of Merlin++, formerly Merlin,

First developed at DESY, circa 2000, by Nick Walker for ILC studies
Extended by Andy Wolski to include linac and damping rings
Added Twiss parameter calculations and symplectic integrators
More features including wakefields, collimation and synchrotron radiation
Handed on to Manchester/Huddersfield in 2009
Developed including advanced scattering models and Hollow Electron Lens for LHC and HL-LHC collimation studies
Tidied up and renamed Merlin++
What it’s all about

Some examples

Particles in the LHC collimators

Beam hitting the edge of a collimator

$\beta$ functions around ATLAS from MAD and Merlin

Beam through a copper slab: Comparison with 2xGeant4
What it’s all about
Some more examples

Sextupole at the end of a FODO lattice

\[ \mu_x \sim 2\pi \times 0.25 \]

\[ x \text{ [mm]} \]

\[ 10^9 p_x \]

\[ \mu_x \sim 2\pi \times 0.2 \]

\[ x \text{ [mm]} \]

\[ 10^9 p_x \]

Losses in the LHC

![Graph showing losses in the LHC](image)

Roger Barlow (Huddersfield)

Merlin++

13th November 2020
What it’s all about
Merlin, MAD and others

MAD
Many more features

Geant4
Does not do detailed collisions and cascades
Does do beam optics and particle bunches – and much faster

FLUKA
Same as Geant4

SixTrack
Similar purpose – but cleaner

All the rest
Merlin++ is general purpose, specific aspects can be added
The big picture
Why Merlin++ is a library, not a stand-alone not a program

For
No need to write parser
Full flexibility of C++ language
User can do what they want with results. Developer does not have to anticipate
User can easily add their own classes

Against
Have to compile program
User can do stupid things

For the user
A library rather than a stand-alone program brings power and responsibility

A lot more power
and only a little more responsibility
The big picture
A full Object Oriented design

Accelerator (or beamline) comprises many components
Magnets (dipoles, quads, sextupoles), drifts, collimators...

Also applies to: particle distributions, trackers, scattering models ...

C-style solution
enumeration and switch

C++ solution
Inheritance: a quad is a magnet which is a component, and a particle is transported through it by its own member function

Extending Functionality
Makes it easy for the user to include a new process
Add child class with new feature - no need to change the core of Merlin++
If useful, can add to the library for other users
The big picture
Using new C++ features

You can write FORTRAN programs in any language

Writing C++ code involves a continual battle not to write in C
Design philosophy from the start was to use not just inheritance but all
features of C++, e.g. Templates
Continue this philosophy as C++ develops (C++11, 14, 17...)
Some features of Merlin got included in later C++ versions. Discard and
adapt new ones

Random Numbers
Nick wrote Merlin with its own random number generator, as the standard
that came with C++ at the time wasn’t good enough for long-cycle
simulations.
C++11 included a proper Mersenne Twister random number generator
So we use it, and drop the old one

For Merlin++, backwards compatibility is not an argument
Good code is **fast, usable and sustainable** – code quality can be measured!

With help of CS colleagues (Colin Venters’ group), analysed Merlin Scott Rowan et al, *Sustainability of the Merlin++ particle tracking code*, CHEP2018 https://doi.org/10.1051/epjconf/201921405028

Criteria e.g. from UK Software Sustainability Institute

Some are just tickboxes: licensing

Some use tools: github, uncrustify, doxygen, cmake tests

Some are providing material: website, tutorials, documentation

Some are more profound e.g. meaningful names

*(PointInside() became CheckWithinApertureBoundaries())*
Cleaning up the code

McCabe value: measures complexity (e.g. lots of if and switch statements means complex (=hard to read) code). Measured by Metriculator package

Cleaning up code also improves speed, through improving look-ahead (measured by valgrind)

The bad guys:

Long Methods, Large Classes, Long Parameter Lists, Switch Statements, Alternative Classes With Different Interfaces, Parallel Inheritance Hierarchies, Duplicate Code, Dead code and Middle Man classes
Cleaning up the code - decoupling

Use ArchDiaDV8 tool to measure Propagation Cost (if you change one thing, how much else needs changing?) and Decoupling Level (are code modules independent?)

Overall figures generally good. Looking at history: got a lot better, then gradually worse, then better again thanks to Scott’s clean-up efforts.

Not just cosmetic: changes also increase speed
Implemented on

- Linux (obviously). Ubuntu and CentOS
- Windows because some people do use it
- MacOS because laptops can be useful as well as beautiful
- Raspberry Pi just because we can
- HTC condor for high-volume work. Random number seeding needed

Testing it on different architectures and operating systems has brought odd issues to light, and forced us to conform to standards

Runs standalone or from Eclipse
Physics Features

- Scattering: various models including the ‘Practical Pomeron’ implementation for elastic and diffractive scattering
- Synchrotron radiation - from a cooling point of view. Doesn’t track the SR photons, though it could
- Spin tracking - haven’t tested this but it’s there
- High-order wakefields - geometric and resistive, for circular beampipes
- Hollow Electron Lens - Haroon Rafique’s thesis
- Heavy Ions - Sam Tygier working on this for RHIC
Performance

For a collimation study (horizontal halo) with scattering

- Tracking 10,000 particles for 10 LHC turns takes 110 secs on a desktop
- Tracking 1,000,000 particles for 100 LHC turns takes 13782 seconds (~ 4 hours)

Can use multiple cores with openmp (results shown for 16-core Xeon)
Getting started and keeping going

1. Read the paper, arXiv 2011.04345. Tells you enough about Merlin++ to decide whether it’s going to be useful for you.

2. Go to the website. http://merlinpp.org

3. Click on ‘Quickstart guide’ and ‘Installation’ and follow instructions

4. Try the tutorials, then start writing your own code - either from scratch or by adapting one of the examples.
Finally

We have come through a fairly major re-write and clean-up of Merlin++, now we want to widen the user base

So do give it a try...

and get in touch