Masters Project
January 8, 2022

Opening the Door for the Use of Herwig in LHCb

Zachary Harper
10241694
Project work completed in collaboration with
Oliver Hay
10299033

University of Manchester

Abstract

This project is an attempt to integrate the Herwig7 generator into the Gauss Simulation Application used by LHCb to produce simulated LHC data. This is an ambitious goal which significant progress has been made towards over the course of the project, specifically in operating Herwig7 with external code and operating external libraries using Gauss. Immediate next steps will be to combine this progress to allow Gauss to initialise Herwig7.

1 Introduction

The ultimate aim of this project is to fully integrate the the Herwig7 Monte Carlo Event Generator (MCEG) into the Gauss System Framework, which is a computer program used by LHCb to generate event data. This is quite an ambitious goal, and this project expects to make significant progress towards it. This report will discuss what has been achieved between September and December of 2021, along with providing background information for project and discuss its importance. It also aims to provide sufficient information to aid in the completion of a similar process by persons of a similar level of knowledge and expertise.

Herwig7 is version 3 release of Herwig++ [1]. Herwig++ is a complete re-write of the original Herwig generators 1 through 6, moving it from Fortran to C++ [1, 2]. As of version 3, it has been deemed complete enough to become the 7th version of the Herwig generator [1]. Previous versions of Herwig have had significant use, including by LHCb [2, 3], supporting the use of Herwig7 by LHCb.

2 Background

2.1 The Monte Carlo Method

The phrase Monte Carlo is commonly used when referring to data created by a simulation rather than experiment, and this often leads to its exact meaning becoming quite nebulous.

Monte Carlo does not by definition require the use of a computer [4]. The technique originated during the 1930s , but did not gain its name until the final years of World War II [5, 6]. It was originally known as 'Statistical Sampling', [5, 6] a descriptive name, and could be computed by hand. For example, one wishing to find the approximate area of a polygon could place it under a grid of regular squares of a known area. By randomly selecting squares, by rolling dice or dropping items onto the grid or similar, and then determining what portion of those squares are half or more filled by the shape, one can determine the total area of the whole grid the shape is expected to take up, and therefore the area of the shape. Figure 1 demonstrates this method.

Using

$$\frac{N_p}{N_t} = \frac{A_s}{A_t} \tag{1}$$

where N_t total sample size (30 here), N_p is positive samples (7 here), A_t is the total area in unit squares (900 here), and A_s is polygon area in unit squares, we can calculate an estimate for the shape area to be 210 squares. Comparing this to the size of the marked out black square, this seems to be sensible. Taking a 5% error, an area of 210 ± 10 unit squares is an accurate estimate.
Now consider that the sample size was about 3.3% of the total area. It should be plain to see why this technique can be a very powerful way to solve much more complex problems where a complete calculation is not viable or simply not possible. Take a coin toss.

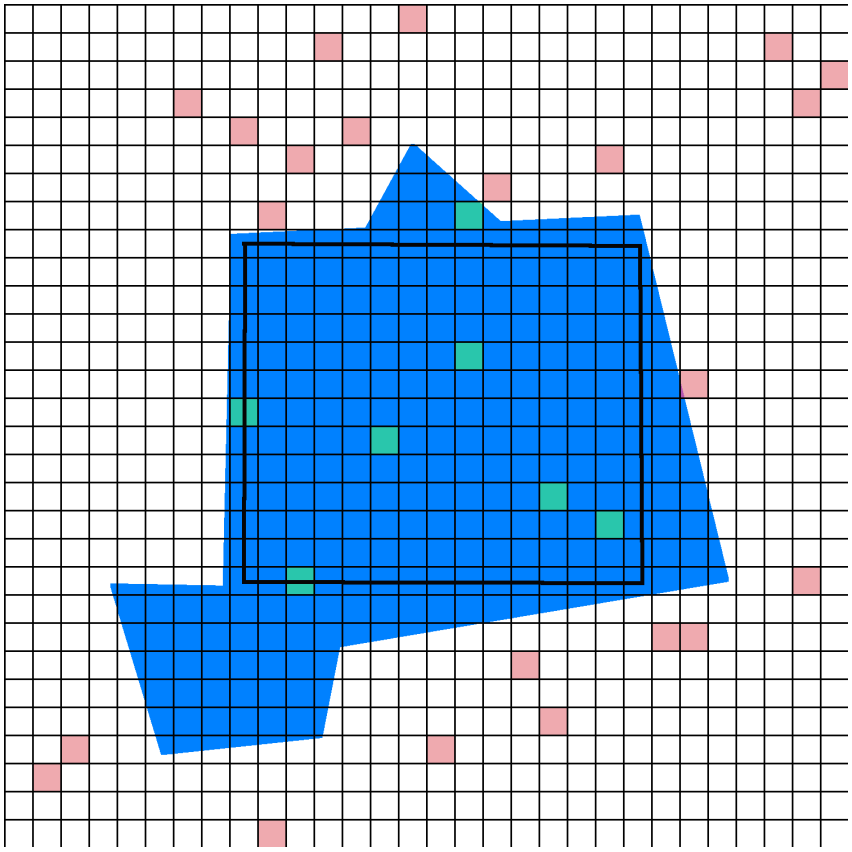

Figure 1: An unusual polygon is set on a 30x30 unit square grid. A sample of 30 randomly selected squares has been taken. There are 7 positive results (green squares) and 23 negative results (red squares). The black rectangle outlines a 15x13 square area, totalling 195 squares.

The outcome of this is not known to be 50/50 through calculation of dynamics, this is excessively difficult to complete, especially when compared to the ease of taking a sample - that is, tossing a coin. Instead, it is known simply through practice, and a very large sample of results.

Now, referring back to Figure 1, as one can expect, as the grid size tends to zero and the sample size tends the infinity, the accuracy of the method increases. And, as these factors change, the time taken to complete the calculation increases too. Thus, the technique never really achieve much of note before the invention of the computer.

2.2 Development of the Monte Carlo Method

At the end of World War II, a collection of physicists and mathematicians, including Stanislaw Ulam, John von Neumann and Nicholas Metropolis, were working with the ENIAC, the first electrical computer, in Pennsylvania and Los Alamos [5]. As should be unsurprising for the time, nuclear research was of particular importance. As the ENIAC was completed, it was tested using models of nuclear reactions. The idea of using a statistical sampling method for this model was thanks to Stanislaw Ulam. Ulam had an interest in random processes, and those that contain branching. This was particularly useful in the case of their nuclear models, as the nuclear interactions develop into branching

chains [5]. This branching behaviour is complex to calculate, with subtle differences in branching chains resulting in very different looking outcomes. Using a computer to calculate this would speed up the calculation of a single model greatly [5], and this would allow a large sample of outcomes to be collected. This sample of outcomes of the nuclear model can be compared to the sampled squares demonstrated in Section 2.1.

However, in order to produce varying outcomes, some of the variables used need to be altered. This typically requires the use of random numbers, again referring back to Section 2.1, the random selection of coordinates for sample squares demonstrates this.

This method earned the name Monte Carlo, thanks to the Casino Stanislaw Ulam's Uncle used to gamble at [5].

2.3 More Complex Techniques

The use of the electrical computer allows for much more complex problems to be solved this way, and also allows for improved techniques.

The problem described in Section 2.1, applying to a simple polygon, can be altered to now describe integration. The area of the polygon becomes the area under a curve, and the grid area can be used to define limits. The shape of the polygon now becomes the shape of the curve. Thus the Monte Carlo technique can now be used to solve complex integrals [6].

However, there is one issue to point out here. For an integral far from an axis, or with very wide limits, an approximation of a trapezium or other shape may acquire similarly accurate results much quicker, as the deviations in area from the curve become a small fraction of the total area. Therefore, there are changes that can be made to the Monte Carlo technique [6].

The 'randomly' selected points can be biased, so that they now more commonly select points around the curve, vastly improving the accuracy of the estimate in important areas, while paying little attention the solid blocks of area far below the curve. Providing this bias is taken into account in the final calculation, this technique can hugely improve accuracy at no expense of computation time [6].

2.4 Terminology

Due to the common vagueness related to the meanings and methods relating to 'Monte Carlo', clear definitions for terms used here are provided below.

The term 'Monte Carlo' here is being used to refer to any "use of stochastic techniques ... to solve a deterministic problem" [4].
As discussed, Monte Carlo techniques are best performed with the use of a computer. However, the nature of this use is also important. Most Monte Carlo techniques are performed using a model, or a simulation such as that used for the nuclear reactions, in order to collect a sample. This is a Monte Carlo simulation. It bears no physical

resemblance to the scenario it is modelling. The simulation is a collection of mathematical models and computer bits.
A 'Monte Carlo method' refers to the generally physical act of directly replicating the examined event, something like flipping a coin repeatedly in order to best ascertain the probability of getting a heads. Equally this also applies to the demonstration in **??**.

2.5 Use of Monte Carlo Simulations in Particle Physics

The LHC is currently the largest particle collider on the planet. Collisions inside the LHC are extremely energetic, and produce large and complex events. Simply storing the quantity of data produced by the detectors required specialised work [7]. Having the ability to simulate these collisions brings a few clear and immediate benefits. It should be possible to track the production of every particle produced during a collision, and follow the direct path from the proton-proton collision to the output from the detectors [3, 8]. This also doesn't require the running of the LHC and all the resources that involves. Additionally, the vast majority of events in the LHC do not actually produce 'interesting' or useful to study data. Using data generators, the generated events a can be biased, as discussed in Section 2.3, in order to more commonly or only produce interesting data [8].

The MCEGs used in particle physics are Monte Carlo for two primary reasons. They are generally used to create a large sample size, for example, Herwig7's default quantity of events for a single generation is ten thousand. This large sample size may then be used to study specific physics. Secondly, in order for these generators to produce varying results, they make use of random (or more accurately 'pseudorandom' when used with a computer [5]) numbers in order to introduce variation into the samples. After producing these numbers, Parton Distribution Functions (PDFs) [2], are consulted in order to determine an outcome.

Further, more technical reasons for the use of MCEGs in particle physics are that the accuracy of a Monte Carlo method is independent from the its dimensionality [2]. Therefore, integration via Monte Carlo Methods is useful due to the complexity of these equations, commonly featuring large scopes and multidimensionality across phase space, and the importance of flavour and spin labels of final state particles [2]. Adding further, parton showers can be accurately simulated using Monte Carlo methods, and hadronisation is already being simulated using them [2].

2.6 The Physics of an MCEG

A brief overview of the physics involved in an event generation of a single a proton-proton collision in the Large Hadron Collider (LHC) will be given, as a full in-depth discussion is beyond the scope of this report. For further reading please see the excellent review of the physics used in MCEGs by Michael Seymour and Marilyn Marx at https://arxiv.org/abs/1304.6677.

An event generation can be split up into five distinct sections, hard processes, parton shower, hadronisation, Underlying events, and unstable particle decays [8].

Hard processes refer to the main collision, the collision of two protons, and the initial and subsequently produced partons. This is quite easy to compute through the use of

PDFs and low order perturbation theory [8].

The produced partons then begin the parton shower, as the initial, still high energy partons produce further partons and bosons. This commonly produces a large amount of gluons. This is also reasonably simple to simulate as a step-by-step process [8].

Then comes hadronisation, as these individual partons are part of groups that comprise hadrons. This stage essentially collects the partons together sensibly, and is particularly important as the hadrons are what the LHC experiments detect [8].

Underlying events refer to additional, secondary events that occur during the collision, due to how close the proton partons become during the collision [8].

The final step is decays, as a large part of the hadrons produced by the initial collisions are quite heavy and unstable, and therefore decay into lighter hadrons before reaching the detectors [8].

Once all of these processes have been completed, the event has been properly generated.

2.7 The Herwig7 MCEG

Herwig7 is the newest version of the Herwig MCEG. It is the same as version 3.0 of Herwig++. Herwig++ is a project to re-write the old Herwig generator in the C++ programming language, as opposed to Fortran which it has previously been written in [1,2,9]. The generator was originally developed in 1986, named "Herwig" as an acronym for "Hardron Emission Reactions With Interfering Gluons, with the last major release written in Fortran being 6.5 in 2000 [9]. In particular, Herwig features angular parton showers, colour coherence effects, and a cluster hadronisation model. Herwig ++ was first released in 2003, [9], with the most recent version before version 3.0 being 2.4.2 released in 2009. Herwig++ has updated some of the physics and features found in the original, along with updating the code itself. The features that now set Herwig++ apart from other MCEGs are [2]:

- The automatic generation of decays and hard processes with full spin correlations. This extends to a lot of Beyond the Standard Model (BSM) Models.
- The matching of hard processes at Next Leading Order (NLO) sing the POWHEG method
- Ordered Angular Parton Showers
- Cluster Hadronisation
- Complex decay model for hadrons, particularly for those involving bottom quarks and τ leptons
- Soft and Hard multiple partonic interactions

Herwig has hand coded matrix elements for common sub processes, and can calculate new matrices for BSM physics where it is necessary. The Specific BSM Models it is capeable of processing for are Minimal Supersymmetric Standard Model (MSSM), Next-to-Minimal Supersymetric Standard Model (NMSSM), and Supersymmetric Les Houches Accord (SLHA) [2]. Herwig's specialisation with bottom quarks makes it especially useful to LHCb, as this is what the majority of its work involves.

3 Project Work

The main purpose of this project is to integrate Herwig7 into the Gauss Simulation Application that is used by LHCb for event generation. This required a good understanding of Gauss and Herwig7's code bases.

3.1 The Gauss Simulation Application

The Gauss Simulation Application and System Framework is an extensive computer program developed by LHCb for LHCb. This project has primarily used Gauss v55r2. It is comprised of several component parts and makes use of several other frameworks, as detailed in Figure 2. Gauss is built on Gaussino, which uses Gaudi framework to integrate Geant4 and MCEGs [10].

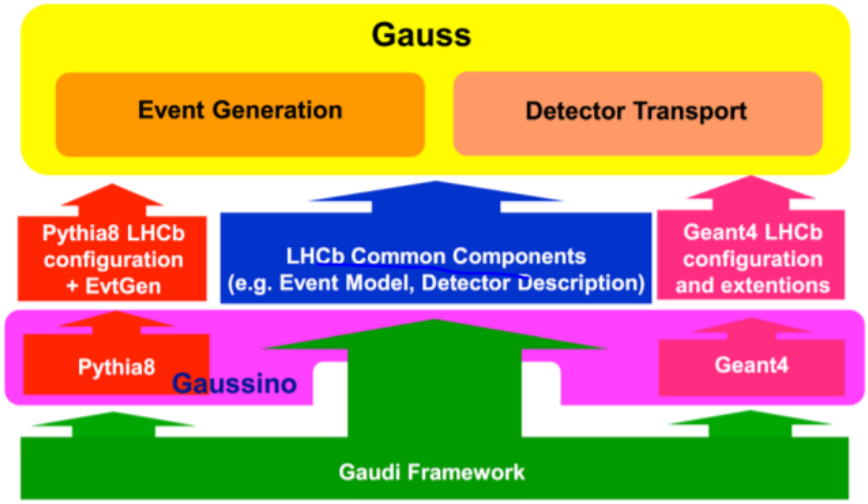

Figure 2: A diagram detailing the components and supporting parts of Gauss. This project is primarily working with the left hand side of the diagram, being work on event generators. Credited to [10].

Gauss uses external libraries for event generation. Currently, Gauss primarily uses Pythia as its event generator, and this project is hoping to enable the use of Herwig7 as another option. Gauss has used previous versions of Herwig in the past [3].
Gauss is used for pure generator studies and for full simulations. This is because Gauss works in two stages, the generation phase and the simulation phase. The generation phase uses MCEGs for event generation. The simulation phase then uses the data created by the generation phase to track particle position through detectors. Subsequently, this data can then be passed into another program named Boole, which will created a simulated LHC response to the generated event and simulated particle motion through detectors [3]. This data can then be used for study in a similar manner to real LHC data.

The aim of this project has been to integrate Herwig7 into Gauss, allowing it to use Herwig7 as an MCEG. This requires a reasonable understanding of low level Gauss architecture. As Herwig7 is purely an MCEG, this means that generator phase is all that

matters. This requires that Gauss be able to locate Herwig7, initiate the MCEG, and then receive data from it.

Gauss uses HepMC formatted events internally, and the Transient Event Store (TES) holds data between the generation phase and the simulation phase.

It is possible to make use of Gauss as a direct user, or through central productions. Central production use appears more common, and it what this project has been working on. This means Gauss is installed on a central computer system that is generally accessible to multiple users. Due to this project working with LHCb, the CERN CVMFS file system was used.

A single use of Gauss for data generation is known as a 'job'. In order for Gauss to successfully complete a job, it must be run with a specific code file written in the programming language Python. This file is used to specify the parameters for the 'job', including which generator is used. In order for this to be successful, this must correctly point to the 'options' file for the generator, another Python file. This also points Gauss to the Production files for a the generator, which are C++ code and header files. Providing Gauss can successfully locate these and the generator itself functions properly, the 'job' can run successfully, at least for the generation stage.

3.2 Herwig7 Architecture

Alongside the low-level structure of Gauss, knowledge of the architecture of Herwig is also extremely important to the project. This section assumes a reasonable technical understanding of computer programming and the C++ language.

The Herwig7 MCEG is a considerably sized computer program now written in C++. It is primarily built around ThePEG, with the Herwig Generator inheriting and extending classes from this module. Alongside ThePEG, Herwig7 also makes use of other modules, these being primarily BOOST, gsl, fastjet, LHAPDF, HepMC, and RIVET. Herwig7 is also capable of integrating multiple other modules for further functions, however these are being ignored for the purposes of this project. A full list can be found here: https://herwig.hepforge.org/tutorials/installation/prerequisite.html.

Herwig7 functions by initialising an event generator based on ThePEG's classes. This generator is initialised, or run, from a '.run' file, which is in turn individually produced by running the program with a '.in' file. This '.in' file is essentially a list of options for how the program will operate, similar to a 'job' in Gauss. One the generator is initialised, it will operate as an MCEG and under normal running conditions produce an output file containing particle information.

This generator can also be accessed using HerwigAPI protocols, which are part of the Herwig Library. This allows the generator to be initialised as a pointer-referenced object and events can be individually requested from this object.

This is important, as Gauss' structure requires only single events to be delivered to the TES at a time. Gauss also requires HepMC formatted outputs, and this can be specified in the '.in' files used by Herwig.

3.3 Process

So far, this project has primarily achieved three things. An understanding of Herwig7's dependencies, and how these interact with the CVFMS file system. A successful call from the Gauss v55r2 Development version to the Herwig7 files. The successful use of the Herwig7 libraries to produce single HepMC format particles using external code.

The initial weeks of this project were spent examining the dependencies of Herwig7. These are quite extensive, and in order for Herwig to be successfully installed on a central Gauss system, these must all be installed and accessible. Due to the constant development of all of Herwig7's prerequisite modules, along with that of Herwig7 and Gauss themselves, version compatibility is a large issue here. In order for Herwig7's dependencies and the installation process to be best understood, a small manual and local install using a Conda environment was completed on the Manchester HEP computer cluster. This was then extended to locate and use the compatible prerequisite versions from the CVFMS file system.

As this neared completion work began on examining Herwig7's low level architecture for compatibility with Gauss. This began with looking into its generator code, especially how this could be initialised. Herwig7's output was then also examined, determining how to use Herwig7 to output HepMC format files, and ensure this output data was understood. It was discovered that the Herwig7 event generator can be accessed and initialised with external code in a pointer form. This could then be called using the pointer to generate an event and deposit particle data into another pointer.

With the installation examination completed, a local installation of the Gauss v55r2 Development version was made on the CERN Andrew File System (AFS). This was set up to completely generate results using Pythia8, and also with the current unfinished development for Herwig++. Replicating the file system used for Pythia8 and Herwig++, with local variable renamed to function for Herwig7, allowed for the Gauss system to correctly call the Herwig7 options and production files in order to initialise the generator.

3.4 Results

The project has progressed significantly, and is at a good point to move further.

The information gained from the Herwig manual installation process has been useful to the project. Please see Appendix A for recommendations about how best to go about this process.

Using HerwigAPI and HerwigUI from the Herwig libraries, external code can be used to complete setup of an instance of a Herwig7 event generator using '.run' configuration files. This generator is referenced by pointer, and using this pointer, event data can be produced in the form of another pointer. The information contained in this data can be produced in HepMC format according to specifications at https://herwig.hepforge.org/tutorials/faq. Currently this has only been implemented for a standalone instance of Herwig7.

In the Development version of Gauss being used for the project, a new directory

specific to Herwig7 has been created. This is modelled on the Herwig++ implementation, renamed for Herwig7. This contains the options and production files used to initialise the generator. It also contains a CMake file, CMakeLists.txt, which is required to properly build Gauss with the new directories. The current version of this file is available in Appendix B. Once Gauss has been successfully made with this directory, Gauss can be set to initialise with the library for the generator. Set an environment variable with the absolute path to the folder containing the Herwig7 library, and then in the job file, import options should be set to point from this environment variable to the Herwig7 options file. Providing this is completed, Gauss will call the specified options file and subsequently the production file the options file specifies.

It seems this approach can be extended to nearly any generator, providing it is of a format involving option and production files. The environment variables are still specific to the specific directory Gauss is being used in, so will need to be set for each individual Gauss instance.

4 Assessment

4.1 Evaluation

The project progressed over the course of twelve weeks, with sixteen hours a week dedicated to it. Weekly meetings were held during this time, to set goals for the week and to review progress made in the previous weeks. This also often involved discussions with supervisor(s) about technical issues or specifications. This project involved a large amount of technical knowledge of varying computer languages. Python and C++ were already known to a good extent, but the use of code specific to the production of MCEGs in these languages had to be learned for the project. This was made particularly difficult to approach due to a general lack of easily accessibly and digestible documentation, often resulting to searching through code files with varying quality commenting.

Additionally, the use of the CERN and Manchester HEP computers required the setup of computing accounts to obtain computer access, a necessary but unfortunately time consuming process. These computers were accessed by SSH, remote online access, and run Linux-based operating systems. This meant most interactions were completed through command line using the Bash command language. Having no prior experience with Bash this required learning as well. Additionally, the command line is quite a slow and unwieldy method for interacting with a computer. Typing out full directories locations to navigate file libraries is more time consuming than file navigation in modern Graphical User Interface oriented Operating Systems. Some of this could be prevented by using GitLab pages for extensive programs such as Gauss (https://gitlab.cern.ch/lhcb/Gauss), and the majority of computer programming was completed on personal computers using Integrated Development Environments (IDEs) such as VSCode. However, this is not without its own issues, as testing any code that required file libraries installed on AFS/CVMFS/Manchester HEP requires the desired file to be uploaded to the file system every time testing is required, again increasing time taken to develop code.

As mentioned previously, compatibility issues with Herwig7 were a major issue to

begin with. The varying modules are generally installed via a 'tarball'. These can be downloaded directly from their appropriate online sites onto the file systems and extracted according to installation instructions discussed in Appendix A. However, several of these programs can be quite hard to locate, especially HepMC. As for technical issues with Gauss, there were a few issues involving setup that were solved by the main supervisor due to their knowledge of the program.

In order to run Herwig, it has to produce a '.run' file from an initial '.in' file. There are issues with identifying this '.run' file, due it having a currently unpredictable naming convention. This is causing issues when combining the stand alone Herwig generator manipulation with the calls from Gauss. in order to correctly use it to initialse the generator. This issue will need to be fixed in order to allow Gauss to run Herwig with customised options, as the '.in' file is how settings for the event generator class are specified.

These were majority technical issues that anyone attemping to follow similar

4.2 Potential Improvements

Should a similar project be undertaken, recommendations would be advanced reading on the structures of Gauss and Herwig7. A significant amount of time with this project has been dedicated to research on high and low level program structures.

Whilst not a direct improvement to the project, improved documentation of the code found in Gauss and Herwig systems would be greatly useful. What currently exists is either aimed at surface level users of the project or at programmers currently working on the projects. There appears to be little consideration of external users looking at low level code and how this creates the general system of the program. This is undoubtedly useful for adapting programs to each other, but is understandably lacking due to this likely being the smallest demographic of use for documentation, alongside the constant development of the programs involved in this process.

4.3 Next Steps

This project will continue to progress from February through to May 2022, and potentially beyond this if required. The logical immediate next step is to consolidate the work done already, combining the standalone work on the Herwig7 generator with the successful production initialisation in Gauss to complete a run of the generation phase using Herwig7.

Further recommendations will be to smooth the attachment of Herwig7 to Gauss. The majority of this code is as of yet hardcoded, and softening this will increase the versatility of the program. For example, because of the wide range of parameters that can be modified when using Herwig, establishing a set of default run settings would be a good idea, then progressing to the ability to set these within a Gauss job file.

Finally tuning specific parameters within the Herwig generator to bring it more in line with what LHCb will be using it for would be greatly useful.

5 Conclusion

To conclude, the project has been a success so far. With a solid understanding of how Gauss and Herwig function gained, the project should progress somewhat more quickly over the coming months. Two important goals of the project have been achieved, those being to get Gauss to recognise and run the Herwig7 production file, and writing external code capable of initialising the Herwig7 MCEG and producing HepMC formatted results. The next immediate goal is to combine these and overcome the issue with the '.run' file naming convention, in order to allow Gauss to fully run Herwig7. After this, moving on to softening some of the hardcoded areas of the project and tuning the Herwig7 generator to better fit LHCb's needs seem like natural next steps.

References

[1] J. Bellm *et al.*, *Herwig 7.0/herwig++ 3.0 release note*, The European Physical Journal C **76** (2016) .

[2] A. Buckley *et al.*, *General-purpose event generators for lhc physics*, Physics Reports **504** (2011) 145–233.

[3] M. Clemencic *et al.*, *The LHCb simulation application, gauss: Design, evolution and experience*, Journal of Physics: Conference Series **331** (2011) 032023.

[4] S. S. Sawilowsky, *You think you've got trivials?*, Journal of Modern Applied Statistical Methods **2** (2003) .

[5] N. Metropolis, *The beginning of the monte carlo method*, Los Alamos Science **Special Issue** (1987).

[6] D. P. L. . K. Binder, *A Guide to Monte Carlo Simulations in Statistical Physics, Third Edition*, Cambridge University Press, 2009.

[7] X. Espinal *et al.*, *Disk storage at CERN: Handling LHC data and beyond*, Journal of Physics: Conference Series **513** (2014) 042017.

[8] M. H. Seymour and M. Marx, *Monte carlo event generators*, 2013.

[9] G. Corcella *et al.*, *Herwig 6: an event generator for hadron emission reactions with interfering gluons (including supersymmetric processes)*, Journal of High Energy Physics **2001** (2001) 010–010.

[10] Müller, Dominik, *Adopting new technologies in the lhcb gauss simulation framework*, EPJ Web Conf. **214** (2019) 02004.

Appendices

A Installing Herwig7

The website https://herwig.hepforge.org/index.html is the main resource for Herwig7. If a standalone version of Herwig is required, is recommended to install Herwig using a

bootstrap file according to these instructions:
https://herwig.hepforge.org/tutorials/installation/bootstrap.html
Should a manual installtion be required, following these:
https://herwig.hepforge.org/tutorials/installation/manual.html
Recommended Versions for modules agree with those listed at:
https://herwig.hepforge.org/tutorials/installation/prerequisite.html.
Note, please use gcc compilers with a version before version 11. Additionally, absolute directory paths to pre-installed modules is best recommended. This approach allows for manual installation of Herwig using modules located on the CVMFS system, for example.

B Project Code

This appendix contains code samples written during the project so far that may be useful to the reader.

```
###############################################################################
# (c) Copyright 2000-2020 CERN for the benefit of the LHCb Collaboration      #
#                                                                             #
# This software is distributed under the terms of the GNU General Public      #
# Licence version 3 (GPL Version 3), copied verbatim in the file "COPYING".   #
#                                                                             #
# In applying this licence, CERN does not waive the privileges and immunities #
# granted to it by virtue of its status as an Intergovernmental Organization  #
# or submit itself to any jurisdiction.                                       #
###############################################################################
###############################################################################
# Package: LbHerwig7
###############################################################################
gaudi_subdir(LbHerwig7 v0r1)

gaudi_depends_on_subdirs(Gen/Generators)

find_package(ThePEG)
find_package(Herwig++)

if (HERWIG++_FOUND)

  find_package(Boost)
  find_package(HepMC)
  include_directories(SYSTEM ${Boost_INCLUDE_DIRS} ${HEPMC_INCLUDE_DIRS} ${THEPEG_INCLUDE_DIRS} ${HERWIG++_INCLUDE_DIRS})

  gaudi_add_module(LbHerwig7
                   src/component/*.cpp
                   INCLUDE_DIRS ThePEG
                   INCLUDE_DIRS Herwig++
                   LINK_LIBRARIES ThePEG GeneratorsLib)

  if(TARGET GSL::gsl)
    set(gsl_target GSL::gsl)
  endif()
  string(REPLACE "-Wl,--no-undefined" "" CMAKE_SHARED_LINKER_FLAGS "${CMAKE_SHARED_LINKER_FLAGS}")
  gaudi_add_library(LbHerwig7Random
                    src/Lib/*.cpp
                    NO_PUBLIC_HEADERS
                    LINK_LIBRARIES ThePEG GaudiKernel ${gsl_target})

else()
  message(WARNING "missing Herwig++: cannot build LbHerwig7")
endif()
```

Figure 3: The current of time of writing CmakeLists.txt file found in the Herwig7 library in the Gauss Gen folder.

```
#include "Herwig/API/HerwigUI.h"
#include <iostream>

namespace Herwig {

/**
 * HerwigInterface is a class to store
 */
class HerwigInterface : public HerwigUI {
public:

    /// Constructor from the arguments provided by main()
    HerwigInterface(int argc, char * argv[]);


    // new Constructor
    HerwigInterface(RunMode::Mode runMode,
    bool resume, bool tics, std::string tag,
    std::string inputfile, std::string repository, std::string setupfile,
    std::string integrationList,
    std::vector<std::string> prependReadDirectories, std::vector<std::string> appendReadDirectories,
    long N, int seed, int jobs, unsigned int jobsize, unsigned int maxjobs);

    /// Destructor to leave a clean ThePEG::Repository behind
    ~HerwigInterface();

    /// Requested Herwig run mode
    RunMode::Mode runMode() const { return runMode_; }

    /// Try to resume execution from an earlier interrupted run.
    bool resume() const { return resume_; }

    /// Require verbose progress markers
    bool tics() const { return tics_; }
```

```
/// A user-defined tag to append to the run name.
std::string tag() const { return tag_; }

/// Name of the file to be read
std::string inputfile() const { return inputfile_; }

/// Repository name to operate on
std::string repository() const { return repository_; }

/// Name of the setup file to be read, to modify the repository
std::string setupfile() const { return setupfile_; }

std::string integrationList() const { return integrationList_; }


const std::vector<std::string> &
prependReadDirectories() const { return prependReadDirectories_; }

const std::vector<std::string> &
appendReadDirectories() const { return appendReadDirectories_; }

long N() const { return N_; }
int seed() const { return seed_; }
int jobs() const { return jobs_; }
unsigned int jobSize() const { return jobsize_; }
unsigned int maxJobs() const { return maxjobs_; }

void quitWithHelp() const;

void quit() const;

/// Return the standard out stream to be used
virtual std::ostream& outStream() const { return std::cout; }
```

```
    /// Return the standard err stream to be used
    virtual std::ostream& errStream() const { return std::cout; }

    /// Return the standard in stream to be used
    virtual std::istream& inStream() const { return std::cin; }

private:

    RunMode::Mode runMode_;

    bool resume_;
    bool tics_;
    std::string tag_;

    std::string inputfile_;
    std::string repository_;
    std::string setupfile_;

    std::string integrationList_;

    std::vector<std::string> prependReadDirectories_;
    std::vector<std::string> appendReadDirectories_;

    long N_;
    int seed_;
    int jobs_;
    unsigned int jobsize_;
    unsigned int maxjobs_;

};

}
```

```
#include "Herwig/API/RunDirectories.h"
#include <ThePEG/Utilities/DynamicLoader.h>
#include <ThePEG/Utilities/Debug.h>
#include <ThePEG/Repository/Repository.h>
#include <ThePEG/Handlers/SamplerBase.h>

namespace Herwig {

void HerwigInterface::quitWithHelp() const {
    //std::cerr << "quit with help" << '\n';
    quit();
}

void HerwigInterface::quit() const {
    ThePEG::Repository::cleanup();
    exit( EXIT_FAILURE );
}

HerwigInterface::~HerwigInterface() {
    ThePEG::Repository::cleanup();
}
```

```
//Need to examine required parameters etc.
HerwigInterface::HerwigInterface(RunMode::Mode runMode,
    bool resume, bool tics, std::string tag,
    std::string inputfile, std::string repository, std::string setupfile,
    std::string integrationList,
    std::vector<std::string> prependReadDirectories, std::vector<std::string> appendReadDirectories,
    long N, int seed, int jobs, unsigned int jobsize, unsigned int maxjobs)
    : runMode_(runMode),
      resume_(resume), tics_(tics), tag_(tag),
      inputfile_(inputfile), repository_(repository), setupfile_(setupfile),
      integrationList_(integrationList),
      N_(N), seed_(seed), jobs_(jobs),
      jobsize_(jobsize), maxjobs_(maxjobs)
{
    // Define runMode of program
    if(runMode_ == RunMode::ERROR)
    {
        quitWithHelp();
    }


}

}
```

```
#include "Herwig/API/HerwigAPI.h"
#include <cstdlib>
#include "ThePEG/Utilities/Exception.h"
#include "ThePEG/Utilities/ColourOutput.h"



int main(int argc, char * argv[]) {

    try {



        Herwig::RunMode::Mode runMode{Herwig::RunMode::READ};
        bool resume{0};
        bool tics{true};
        std::string tag{""};
        std::string inputfile{"Test.in"};
        std::string repository{"HerwigDefaults.rpo"};
        std::string setupfile; //might be useful for specifying output type?
        std::string integrationList;
        std::vector<std::string> prependReadDirectories;
        std::vector<std::string> appendReadDirectories;
        long N{10};
        int seed{0};
        int jobs{1};
        unsigned int jobsize{0};
        unsigned int maxjobs{0};
```

```
    // read in options
    const Herwig::HerwigInterface cl = Herwig::HerwigInterface(runMode,
    resume, tics, tag,
     inputfile,  repository, setupfile,
    integrationList,
    prependReadDirectories, appendReadDirectories,
    N, seed, jobs, jobsize, maxjobs);
//std::cout << cl.inputfile();
    // Call program switches according to runMode
    switch ( cl.runMode() ) {
    case Herwig::RunMode::INIT:        Herwig::API::init(cl);       break;
    case Herwig::RunMode::READ:        Herwig::API::read(cl);       break;
    case Herwig::RunMode::BUILD:       Herwig::API::build(cl);      break;
    case Herwig::RunMode::INTEGRATE:   Herwig::API::integrate(cl);  break;
    case Herwig::RunMode::MERGEGRIDS:  Herwig::API::mergegrids(cl); break;
    case Herwig::RunMode::RUN:         Herwig::API::run(cl);        break;
    case Herwig::RunMode::ERROR:
        std::cout << ThePEG::ANSI::red
                  << "Error during read in of command line parameters.\n"
                     << "Program execution will stop now."
                  << ThePEG::ANSI::reset;
        return EXIT_FAILURE;
    default:
        cl.quitWithHelp();
    }
    //return 0;

    return EXIT_SUCCESS;
```

```
catch ( ThePEG::Exception & e ) {
    std::cout << ThePEG::ANSI::red
                  << argv[0] << ": ThePEG::Exception caught.\n"
                     << e.what() << '\n'
                     << ThePEG::ANSI::yellow
                     << "See logfile for details.\n"
                     << ThePEG::ANSI::reset;
    return EXIT_FAILURE;
}
catch ( std::exception & e ) {
    std::cout << ThePEG::ANSI::red
                  << argv[0] << ": " << e.what() << '\n'
                  << ThePEG::ANSI::reset;
    return EXIT_FAILURE;
}
catch ( const char* what ) {
    std::cout << ThePEG::ANSI::red
                  << argv[0] << ": caught exception: "
           << what << "\n"
                  << ThePEG::ANSI::reset;
    return EXIT_FAILURE;
}
catch ( ... ) {
    std::cout << ThePEG::ANSI::red
                  << argv[0] << ": Unknown exception caught.\n"
                  << ThePEG::ANSI::reset;
    return EXIT_FAILURE;
}
```

Figure 4: As of time of writing external C++ code for running the Herwig7 Generator using HerwigAPI