GooFit 2.0

Henry Schreiner[1], Christoph Hasse[2], Bradley Hittle[3], Himadri Pandey[1], Michael Sokoloff[1] and Karen Tomko[3]

[1] University of Cincinnati, 2600 Clifton Ave, Cincinnati, OH 45220, USA
[2] Technical University of Dortmund, Emil-Figge-Straße 50, 44227 Dortmund, Germany
[3] Ohio Supercomputer Center, 1224 Kinnear Rd, Columbus, OH 43212, USA

E-mail: henry.schreiner@uc.edu

Abstract. The GooFit package provides physicists a simple, familiar syntax for manipulating probability density functions and performing fits, and is highly optimized for data analysis on NVIDIA GPUs and multithreaded CPU backends. GooFit was updated to version 2.0, bringing a host of new features. A completely revamped and redesigned build system makes GooFit easier to install, develop with, and run on virtually any system. Unit testing, continuous integration, and advanced logging options are improving the stability and reliability of the system. Developing new PDFs now uses standard CUDA terminology and provides a lower barrier for new users. The system now has built-in support for multiple graphics cards or nodes using MPI, and is being tested on a wide range of different systems. GooFit also has significant improvements in performance on some GPU architectures due to optimized memory access. Support for time-dependent four-body amplitude analyses has also been added.

1. Introduction

Multidimensional fits to large datasets, with tens of millions of events, are becoming increasingly common in High Energy Physics (HEP). The models can be computationally intense, with hundreds of parameters. In preparing a model and fitting the data, the most natural description is usually to prepare components,[1] and then combine these components for a specific fit. This is the approach taken by RooFit [1], one of the most commonly used physics fitting packages. With the advent of highly parallel computing architectures, the need arose for a system that would make implementing fitting efficiently in parallel as intuitive to physicists as RooFit.

Graphical Processing Units (GPUs) provide an alternative to traditional computing, with the ability to do computations in a massively parallel compute engine, with thousands of floating point operations processed at the same time (see Table 1). Since most of the time spent in fitting a probability distribution to a dataset is composed of independently computing the value of a function at each point in the dataset, this maps well to the processing capabilities of GPUs.

There are several complications however; programming on a graphics card uses a different language (such as CUDA for NVIDIA) and requires data to be managed between memory in the host and the device. GPUs have a simpler instruction set, no branch prediction, and an emphasis on single precision compute, especially for the cheaper "gamer"-class cards. They also must compute the exact same instruction across groups of data at the same time. Even a

[1] Although commonly called PDFs, these components include complex quantum mechanical amplitudes whose magnitude-squared values represent PDFs.

Table 1. The advertised floating point operations per second (FLOP/s) for several common graphics cards, for both single precision (SP) and double precision (DP), and the number of Streaming Multi-processors (SMPs). Some approximate FLOPs of Intel CPUs for comparison include 0.0864 GFLOP/s for a 2015 i5 dual core processor or 0.461 GFLOP/s for a 12 core Xeon from a similar year.

				GFLOP/s		
	Name	**Cores**	**Clock**	**SP**	**DP**	**Cost**
Gamer	GTX 1050 Ti	768	1290 Mhz	1980	62	$150
	GTX 1080 Ti	3584	1596 Mhz	11300	330	$850
Server	Tesla K40	2880	745 Mhz	4290	1430	$3000
	Tesla P100	3584	1329 Mhz	9300	4700	$10000

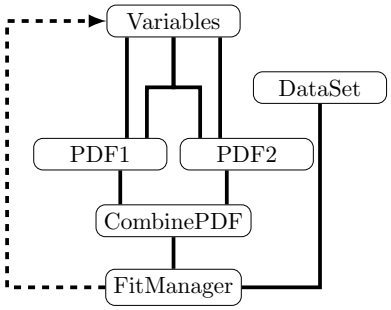

Figure 1. Illustration of the structure of a GooFit program. The `Variable`s represent both observables and parameters, and are input to PDFs. The PDFs can be combined using combination PDFs. The final PDF is fed into a `FitManager`, which also takes a binned or unbinned `DataSet` with data to attach to an observable. The `FitManager` performs a fit, and sets the parameter variables accordingly.

simple if statement will force a mask to be applied and both sides of the if statement must be computed separately. These factors combine to make writing GPU code substantially different from traditional CPU code.

One of the first fitting systems designed for parallel fits on a GPU was the GooFit package, first released in 2013 by Rolf Andreassen [2]. The project was designed using the CUDA language from NVIDIA, and used their Thrust library [3] to manage kernel launches. By the end of the year, GooFit was expanded with an Thrust supported OpenMP backend to enable parallel fitting on CPU devices "with the flip of a (compiler) switch".

GooFit 1.0 had an impressive list of built-in PDFs, including a specialized system for amplitude analyses of three body particle decays (often called Dalitz plot analyses). Combination PDFs provided ways to build more complex PDFs out of simpler building blocks. Several examples were provided. Simple composition of PDFs using existing building blocks was GooFit's design target (see Figure 1), and it succeeded in that. The comparative performance for different systems can be seen in Table 2, and scalability when changing the number of OpenMP threads can be seen in Figure 2 and Figure 3.

However, providing new PDFs was non-trivial, and many of the more advanced Physics PDFs had large sets of custom code that was not generalized or shared with the rest of GooFit, such as complex return values or signal generation. GooFit was hard to build and had specific system requirements, and the development was fragmented across several Git repositories on GitHub. The GooFit 2.0 project was undertaken to make GooFit easier to build and develop with, and to combine the development effort. Future work is addressing other aspects of the design to make PDFs easier to author and simpler to maintain.

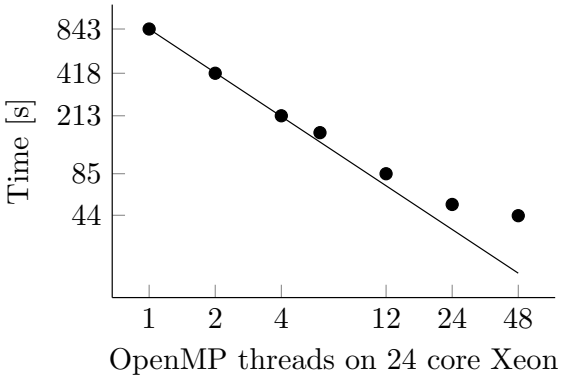

Figure 2. Performance of $\pi^+\pi^-\pi^0$ with 16 time dependent amplitudes on an Intel Xeon E5-2680 dual-chip system (24 cores). The line illustrates ideal scaling.

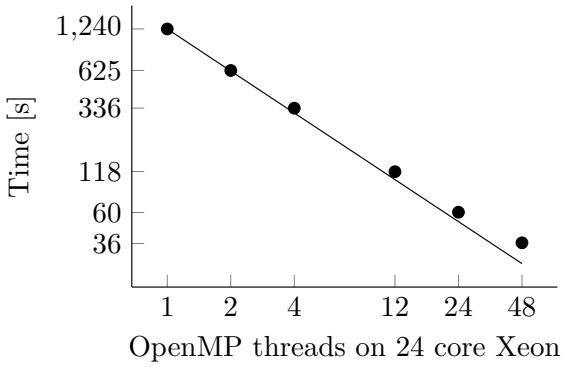

Figure 3. Performance of the Zach fit (D^{*+} and D^0 mass difference measurement with BaBar data) with 16 time dependent amplitudes.

Table 2. Comparison of performance of example code from the GooFit 2.0 examples on different systems. $\pi^+\pi^-\pi^0$ on the left, Zach fit on the right.

Type	Device	Time
2 Cores	Core 2 Duo	1159. s
GPU	GeForce GTX 1050 Ti	86.4 s
GPU	Tesla K40	64.0 s
MPI	Tesla K40 ×2	39.3 s
GPU	Tesla P100	20.3 s

Type	Device	Time
2 Cores	Core 2 Duo	738. s
GPU	GeForce GTX 1050 Ti	60.3 s
GPU	Tesla K40	96.6 s
MPI	Tesla K40 ×2	54.3 s
GPU	Tesla P100	23.5 s

2. New build system

The build system has been completely redesigned using CMake, the most popular cross-platform build system. This was done in a three step procedure, allowing an evolutionary change to the traditional build system to support new folder structures and code reorganization while the new system was being built (described in more detail in [4]).

The new CMake-powered build added a plethora of new features to GooFit. Support for IDEs, such as XCode and QtCreator is now available; IDE-centric features such as header listings and folder organization are included. More backends are now supported, such as the single threaded "CPP" backend, and a limited form of support for Intel's Threaded Building Blocks (TBB). Support for Intel and LLVM compilers was also added, including the Apple LLVM compiler on macOS for the first time. Data files are automatically downloaded from GitHub by the build system for the examples. All of the required libraries are now downloaded through Git submodules if not discovered, including several new header-only libraries. The helper files that make these features possible are available in a separate Git repository, for reuse by other non-GooFit packages. A new packaging system was added as well, allowing a user to easily develop code without forking the GooFit project.

An initial set of unit tests were added, primarily focusing on parts of GooFit that were refactored during the 2.0 development process, such as the `FitManager` and `Variable`s. The examples now have a script that runs all of them sequentially, and checks the validity of the results. Continuous integration for the OpenMP backend builds, tests, and runs the examples on the Travis CI service. This system also builds the Doxygen comments in the source into documentation for every version. Code coverage was added with GCov and the CodeCov service

to estimate the percentage of the code base covered by the test cases, and to report changes on new pull requests.

The Travis CI system presented several difficulties. The GooFit library required the ROOT library to install [5], but building ROOT takes more than the alloted time on a Travis worker node utilizing both available threads. This was overcome by using a prebuilt binary and caching it on Travis. CMake is trivial to download and run for any system, so a recent copy is obtained on Travis and used as well. A standalone copy of Minuit 2 was made available for GooFit 2.0, so there is longer a requirement that ROOT be available, although since it is still used in most of the examples and a few optional tests, ROOT remains part of the standard Travis build.

GooFit Docker images for both OpenMP and CUDA (using NVIDIA-Docker) were added to provide a simple way for new users to start using GooFit. An exact set of commands to prepare a basic GooFit environment, developed and tested using pristine operating system Docker containers, was developed for CentOS7, OpenSUSE, Alpine, and Ubuntu systems.

3. Modernization

GooFit was originally designed for CUDA 4.0 and NVIDIA CUDA compute architecture 2.0. Several of the GooFit forks were already using C++11 features available in CUDA 7+, so the decision was made to target CUDA 7 and higher for the GooFit 2.0 upgrade. Code cleanup and modernization was initially performed manually, but was slow and laborious for such a large code base. Some changes, such as renames, were done using regular expressions in Python using the `ModernizeGooFit.py` script. This was instrumental in removing custom terminology that would be unfamiliar to new developers. Where possible, standard CUDA or Thrust terminology was used for all backends, and several spellings were normalized to match ROOT. To facilitate rewriting GooFit using newer language constructs, the Clang-Tidy program was used. It processed all of the source code using the built-in CMake 3.6+ support, and changed a specified set of features. One feature was processed at a time, using Git to view the changes. Some of the most useful changes were the use of override for all overriding virtual functions, the use of range-style `for` loops, and the use of `nullptr` vs. `0` or `NULL`.

The usage of a custom GooFit class for complex numbers was replaced with the new Thrust complex number class. This also required supplying an external addition that enhanced the `ldg` CUDA intrinsic[2] to support non-intrinsic types to retain recent performance gains on midrange NVIDIA hardware [6]. Logging is now under a unified interface using the fmt library to provide simple Python-like formatted messages [7]. Compile time settings in CMake allow debug and trace logging to be added to GooFit with no runtime cost if disabled. Unified errors are provided with a custom exception subclass that also utilizes the fmt library. Timing statistics have been added to the standard output. A modified version of the small FeatureDetector library checks for missed compiler optimization and prints warnings as needed [8].

One of the features commonly needed for the examples and for analyses was the addition of a Command Line Interface parser, CLI11 [9]. This library was designed to provide a completely general utility for creating command line interfaces for complex and performance dependent applications, with the ability to provide customized behavior for specific toolkits. GooFit provides a customized subclass for the main application, adding standard GooFit specific options, backend information, color printing through the Rang library [10], checks for missing compiler optimization, and a few other features. The GooFit version sets defaults that are designed to provide a natural one-to-one mapping of a command line interface and the standard models described by fits, while CLI11 defaults remain true to standard Unix programs.

[2] In CUDA, `__ldg()` reads from global memory using the texture-path, which is a read-only memory path providing faster global memory access.

4. New software features

One of the key features of GooFit is the caching of partial results inside PDFs to be reused in the computations, taking advantage of the fact that many parameters remain the same between calls. The `Variable` change detection system was improved to support multiple datasets. The caching system for specific PDFs was modified to support a "Structure of Arrays" format which gives better performance.

Another recent addition is preliminary MPI support. This is included as an option in the CMake build, and the standardized CLI11 application class allows setup and teardown to conditionally be included. The provided MPI support divides the dataset for the application by the number of processes involved in the calculation. The compilation and execution of MPI is supported for both OpenMP and CUDA. If multiple GPU processes run on the same node, they will each select a different GPU, if available.

The default fitter in GooFit has been changed to the newly supported Minuit 2 fitter, with the Minuit 1 fitter still provided if ROOT is present. The Minuit 2 fitter was factored out of ROOT, and a new CMake build was added. If ROOT is not found, GooFit will default to the standalone Minuit 2. As part of this change, the Minuit 1 fitter was completely redesigned to provide a more consistent interface. The new version avoids global variables and has automated variable synchronization. The Minuit 2 fitter provides direct access to the FCN and Minuit 2 variables, improved logging output, and direct access to all the Minuit 2 controls and output.

The other non-PDF core classes in GooFit, `DataSet`s and `Variable`s, were also rewritten. Input to a dataset is much simpler and faster internally, improving code transparency and maintainability. The inheritance design was improved; it is now possible to write generic code with `DataSet` and select the binned or unbinned variations at runtime, simplifying several examples. `Variable`s now have a much more tightly controlled API, allowing GooFit to catch errors more reliably at compile time. Further runtime checks were added to warn unsuspecting users who load data out of range.[3] Several operator overloads were added to make manipulation of the `Variable` value as convenient as direct access.

5. New physics features

Several physics analyses have used new physics features from the various GooFit forks that were merged into GooFit 2.0.

Support for three-body time-dependent amplitude analyses was added. This was used to measure mixing in the decay $D^0 \to \pi^+\pi^-\pi^0$ [11]. The code for this analysis has been provided in both GooFit as an example, and as the original RooFit based package. The data for this example were made public for the first time by the *BABAR* collaboration just before the release of GooFit 2.0. Another use of this feature can be seen in the LHCb mixing and CP violation search in $D^0 \to K^0_S\pi^-\pi^+$ [12].

Four-body time-integrated and time-dependent amplitude analyses support was added, as well. This was developed for and used in a mixing parameter search in $D^0 \to K^+\pi^-\pi^+\pi^-$ [13].

Toy Monte Carlo generation for multi-body systems was added using the MCBooster library [14]. This is used in the three- and four-body amplitude analysis PDFs for integration, signal generation, and coordinate transformations. Physics analyses that use this include the mixing parameter search previously mentioned as well as a model independent partial wave analysis (MIPWA) of $D^+ \to h^-h'^+h'^+$ [15].

6. GooFit future developments

One of the most requested features, Python bindings, has been developed as a proof-of-principle for an example, a simple exponential. Only the minimal set of tools needed for that one example

[3] The author was one such unsuspecting user.

is provided in GooFit 2.0, but it is relatively straight forward to extend to the rest of GooFit. The bindings were constructed with pybind11 [16], and work with OpenMP or GPU backends. The Python interface is disabled by default, but is included in the test builds on Travis.

In development for GooFit 2.1 are drastically expanded Python bindings, with support for most of GooFit's features and all of the PDFs. The current development version is available from the PyPI system through pip (Package Installer for Python).

GooFit development continues at a rapid pace. A new indexing system is being designed and implemented across all PDFs, which will simplify PDF authoring and enable new optimizations to the backend in the future. The Hydra package is being considered for inclusion, a new framework for data analysis with massively multi-threaded platforms, developed by the same author as MCBooster as a replacement [17].

7. Conclusions

GooFit has undergone a major code structure transformation. The changes behind the 2.0 design is enabling new features to be added, is reducing the burden on analysts building and using GooFit, and is encouraging contributions to a unified code base. The code runs faster and is better at catching a user's mistakes. It runs on more systems than ever and supports IDEs and debuggers. It provides a set of tools to assist analysts in writing GooFit code.

The future of GooFit looks bright. The Python bindings will make PDF composition even easier to access. The PDF indexing and redesign efforts will make developing in GooFit easier, and could potentially improve GooFit's already exciting performance. Hydra inclusion may further improve the performance and abilities of analysts.

References

[1] Verkerke W and Kirkby D P 2003 *eConf* **C0303241** MOLT007 (*Preprint* physics/0306116)

[2] Andreassen R E, de Silva W M, Meadows B T, Sokoloff M D and Tomko K A 2014 *IEEE Access* **2** 160–176

[3] Hoberock J and Bell N 2010 *Thrust* Version 1.8.2, accessed 2017-10-20 URL http://thrust.github.io/

[4] Schreiner H 2017 Modernizing GooFit: A Case Study *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact* PEARC17 (ACM) pp 30:1–30:5 ISBN 978-1-4503-5272-7 URL http://doi.acm.org/10.1145/3093338.3093364

[5] Brun R and Rademakers F 1997 *Nucl. Instrum. Meth.* **A389** 81–86

[6] NVIDIA 2013 *Generics* Accessed 2017-10-20 URL https://github.com/bryancatanzaro/generics

[7] Zverovich V 2012 *fmt* Version 4.0.0, accessed 2017-10-20 URL https://github.com/fmtlib/fmt

[8] Yee A 2015 *FeatureDetector* Accessed 2017-10-20 URL https://github.com/Mysticial/FeatureDetector

[9] Schreiner H 2017 *CLI11* Version 1.2, accessed 2017-10-20 URL https://github.com/CLIUtils/CLI11

[10] Gauniyal A 2016 *Rang* Version 2.1, accessed 2017-10-20 URL https://agauniyal.github.io/rang

[11] Lees J P *et al.* (BaBar) 2016 *Phys. Rev.* **D93** 112014 (*Preprint* 1604.00857)

[12] Reichert S 2015-12-18 *Measurement of the mixing parameters of neutral charm mesons and search for indirect CP violation with* $D^0 \to K^0_S\pi^+\pi^-$ *decays at LHCb* Ph.D. thesis Dortmund U. URL http://inspirehep.net/record/1503629/files/CERN-THESIS-2015-348.pdf

[13] Hasse C, Albrecht J, Alves Jr A A, d'Argent P, Evans T D, Rademacker J and Sokoloff M D 2017 Amplitude analysis of four-body decays using a massively-parallel fitting framework *22nd International Conference on Computing in High Energy and Nuclear Physics (CHEP 2016) San Francisco, CA, October 14-16, 2016* (*Preprint* 1702.06735)

[14] Alves Jr A A and Sokoloff M D 2017 (*Preprint* 1702.05712)

[15] Sun L, Aoude R, dos Reis A C and Sokoloff M D 2017 Model-independent partial wave analysis using a massively-parallel fitting framework *22nd International Conference on Computing in High Energy and Nuclear Physics (CHEP 2016) San Francisco, CA, October 14-16, 2016* (*Preprint* 1703.03284)

[16] Jakob W 2015 *pybind11* Version 2.2.1, accessed 2017-10-20 URL https://github.com/pybind/pybind11

[17] Alves Jr A A 2017 Hydra: A Framework for Data Analysis in Massively Parallel Platforms (San José: GTC2017) Presentation ID S7340