Parallelization and Vectorization of ROOT Fitting classes

X. Vallis, L. Moneta for the ROOT Team

Introduction

In order to take full advantage of new computer architectures and to satisfy the requirement of maximizing the CPU usage with increasing amount of data to analyze, parallelization and vectorization have been introduced in the ROOT mathematical and statistical libraries.

As part of this effort, new generic classes supporting a task based parallelization mode have been introduced in ROOT, which can be used for a wide range of computational tasks in the field of High Energy Physics. The support for different SIMD’s libraries has also been included.

All these different tools for parallelism come together when parallelizing the fitting. As a result of this work, vectorization and parallelization have been introduced in ROOT requiring minimal changes in user code.

The Higgs fit is a good case example to report on the improvements obtained in the function evaluation, used for data modelling, by adding the support for SIMD vectorization and multithreaded parallelization. We will display for this use case how the different evaluations of the likelihood and the least square functions used for fitting ROOT histograms, graphs and trees perform.

Tools for parallelism

ROOT provides several generic classes for the expression of parallelism at different levels. Some of them play a central role when parallelizing ROOT fitting classes.

Task level parallelism: TThreadExecutor

TThreadExecutor is a task-oriented, multithreaded MapReduce for parallelism at different levels. Some of them play a central role when parallelizing ROOT fitting. As a result of this work, vectorization and parallelization have been introduced in ROOT requiring minimal changes in user code.

In order to report on the performance of the parallelization described we decided to apply it to the well known Higgs Fit.

Case example: Higgs Fit

In order to report on the performance of the parallelization described we decided to apply it to the well known Higgs Fit.

Fitting

The figure below describes in detail the parallelization process of the fitting functions in both task level and instruction level.

Vectorization will be applied for evaluating the function several points at a time in the multithread Map stage, which will generate several partial results in each thread to combine in a final reduction step.

The figure above describes the speed up obtained fitting o a typical mass-consumption general purpose computer, a 4-core machine with different sets of vectorization instructions and different evaluation functions.

Below, we study the scaling of multithreaded operations on a 14 cores Haswell processor with SSE2 as SIMD instruction set. While it scales in the multithreaded scalar case, vectorization limits the speedup obtained on the multithreaded vectorized case.

Higgs Fit Scaling With Number of Cores

It is also interesting to show how different compilers behave when vectorizing.

Higgs Fit

The pieces of code below compare the current code used for fitting the case example with the code needed for a fully parallelized implementation of the same fit.

Current implementation

```
//Higgs Fit: Implementation of the vectorized function
ROOT::TSeq<ROOT::TBlend<ROOT::Double_v, data, data>> return_params(const std::vector<h1f>& data, params);

double f(const std::vector<h1f>& data, params); //Higgs Fit: Implementation of the scalar function
```

Vectorized plus parallelized implementation

```
//Higgs Fit: Implementation of the vectorized function
ROOT::TSeq<ROOT::TF1<ROOT::Double_v, data, data>> return_params(const std::vector<h1f>& data, params); //Higgs Fit: Implementation of the scalar function
```

The only changes required are make the fitting function vectorized (change the data parameter type and the return type in this case) and to specify the "MULTITHREAD" option to the fitting. This makes most of the existing user code eligible to vectorize with very little effort.

Below we present some of the performance measurements made while working on this case example. All the fitting times have been normalized to the number of function calls made by the fitter, as the nature of minimization problems will make the number of function calls fluctuate between examples and influence the times.

Note that the compiler will try to auto-vectorize operations in the scalar case, and our current implementation for the function evaluation will provide help in that direction. This may make the vectorization times not look so good.

References

1. VecCoreLibrary (https://github.com/Root-project/veccore)
2. Vc (https://github.com/VcDevel/Vc)
4. ROOT Data Analysis Framework (https://root.cern)