Parallelization and vectorization of the Fit

Xavier Valls
Parallelization and Vectorization of ROOT Fitting classes

Introduction

In order to enhance the performance of data analysis in ROOT framework, we propose parallelization and vectorization techniques for fitting classes. This approach allows for significant speedup in processing large datasets, especially in high-energy physics applications. The implementation involves utilizing the ROOT framework's capabilities for parallel execution and leveraging vector operations to accelerate fitting processes.

Fitting Parallelization

Task load parallelism: TThreadExecutor
- A thread executor framework designed to distribute tasks across multiple threads.
- Enables efficient execution of fitting tasks in parallel, reducing the overall processing time.

Task-driven parallelism: TVectorized
- Utilizes vectorized operations for fitting tasks, optimizing performance on modern processors.
- Enhanced performance through the utilization of SIMD (Single Instruction, Multiple Data) instructions.

CASE EXAMPLE: HIGGS FIT

Performance of the Higgs fit

References

My poster
Fitting parallelization

DATA

Chunking

Scheduler

Thread 1

Thread i

Thread N

Function Evaluation

Instruction level parallelism (vectorization)

Data level parallelism (Multithreading)

Function Evaluation

SIMD array

SIMD array

Chunk

Reduction

Input
Task level: **ROOT::TThreadExecutor**

Instruction level: **VecCore**
Backwards compatibility!

Spot the differences

//Higgs Fit: Implementation of the scalar function

```c
double func(const double *data, const double *params) {
    return params[0] * exp(-(*data + (-130.)) * (*data + (-130.)) / 2) +
    (*data) * (0.01)) * (*data) * (0.01))) ;
}
```

TF1 *f = new TF1("fScalar", func, 100, 200, 4);
f->SetParameters(1, 1000, 7.5, 1.5);
TH1 h1f("h1f", "Test random numbers", 12800, 100, 200);
h1f.FillRandom("fScalar", 1000000);
h1f.Fit(f);

//Higgs Fit: Implementation of the vectorized function

```c
ROOT::Double_v func(const ROOT::Double_v *data, const double *params) {
    return params[0] * exp(-(*data + (-130.)) * (*data + (-130.)) / 2) +
    (*data) * (0.01)) * (*data) * (0.01))) ;
}
```

//This code is totally backwards compatible
TF1 *f = new TF1("fCore", func, 100, 200, 4);
f->SetParameters(1, 1000, 7.5, 1.5);
TH1 h1f("h1f", "Test random numbers", 12800, 100, 200);
h1f.FillRandom("fCore", 1000000);
```

//Added multithreaded fit option
h1f.Fit(f, "MULTITHREAD");
Backwards compatibility!

```cpp
//Higgs Fit: Implementation of the vectorized function
ROOT::Double_v func(const ROOT::Double_v *data, const double *params)
{
    return params[0] * exp(-(data + (-130.)) * (data + (-130.)) / 2) +
                             (**data) * (0.01)) * (**data) * (0.01));
}

//This code is totally backwards compatible
TF1 *f = new TF1("fvCore", func, 100, 200, 4);
f->SetParameters(1, 1000, 7.5, 1.5);
TH1D h1f("h1f", "Test random numbers", 12800, 100, 200);
h1f.FillRandom("fvCore", 1000000);

//Added multithreaded fit option
h1f.Fit(f, "MULTITHREAD");
```

Won't be needed with ImplicitMT!
On your PC!

- Vectorization: ~3.3X
  - MAX 4

- Parallelization: ~3.8X
  - MAX 4

- Parallelization + Vectorization: ~11X
  - MAX 16
Vectorizing the Higgs Fit (4 physical cores)
Multithreaded Speed up

Scaling of the least squares fit
(14 cores + AVX2)
Compiler performance

Times for the Max. Likelihood Fit (default, SSE2)

- Sequential
- Vectorized

Times for the Max. Likelihood (no autovectorization, SSE2)

- Sequential
- Vectorized

Compiler performance for the Unbinned Likelihood Fit (4 cores + SSE2)

Graph showing speedup vs. number of physical CPUs:
- GCC
- Clang
- ICC

Graph showing times for different compilation methods and libraries.