Parallelization and vectorization of the Fit

Xavier Valls

ROOT

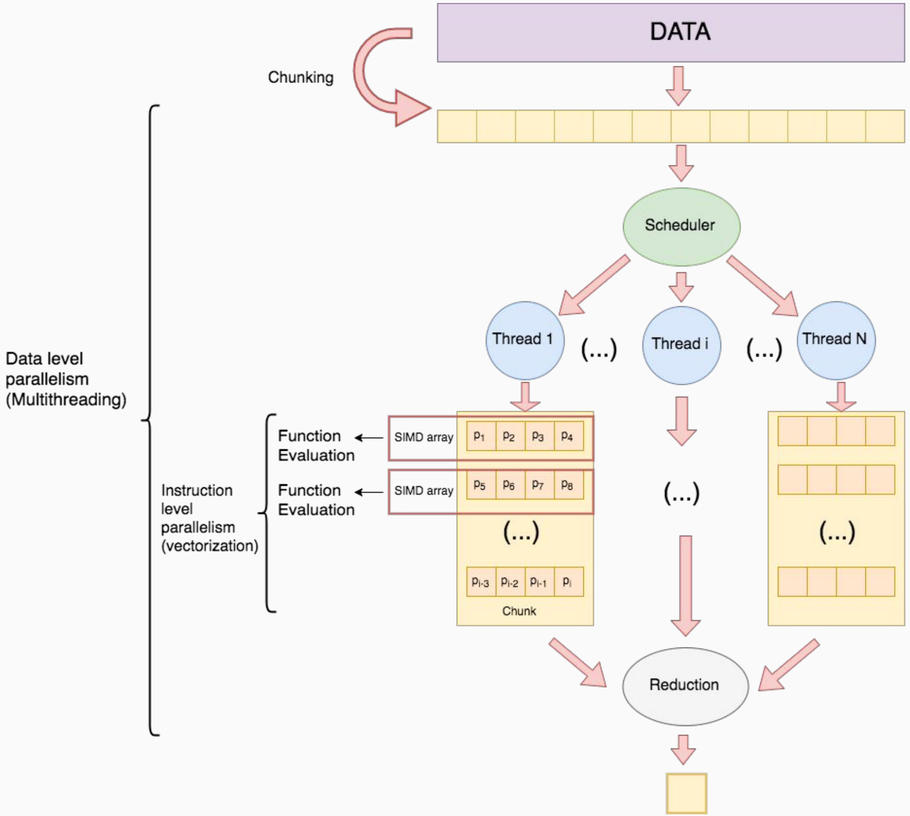
Data Analysis Framework

https://root.cern

My poster


Parallelization and Vectorization of ROOT Fitting classes
ROOT
Data Analysis Framework
X. Valls, 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 analysis, parallelization and vectorization have been introduced in the ROOT mathematical and statistical libraries, requiring minimal changes in user code.
CERN Computing Needs Keep Growing (Images: CERN)
As part of this effort, new generic classes supporting a task based parallelization mode have been defined 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.
Tools for parallelism
Task level parallelism: TThreadExecutor
• Task-oriented, multithreaded MapReduce for ROOT
• Provides operations Map, Reduce, Foreach and even chunked mapping with partial reduction.
• Used in math fitting, TMVA(Boosted Decision Tree evaluation, Deep Neural Networks processing), Implicit multithreading operations in I/O (reading, deserialization and decompression of tree branches in parallel, parallel writing) and for parallel execution of functional chains in TDataFrame.
Instruction level parallelism: VecCore
• Provides efficient vectorization on all platforms by writing abstract, architecture-generic code that will map to each of its optional backends' concrete types, methods or instructions. Includes a scalar backend for the case when SIMD operations are not available.
• See the poster "Speeding up software with VecCore, a portable SIMD library" by Guilherme Amadio.
Fitting Parallelization
DATA
Chunking
Scheduler
Data level parallelism (Multithreading)
Instruction level parallelism (vectorization)
Function Evaluation
Reduction
References
1. VecCore Library https://github.com/root-project/veccore
2. Vc https://github.com/VcDevel/Vc
3. UME::SIMD https://github.com/edanor/umesimd
4. ROOT Data Analysis Framework https://root.cern
CASE EXAMPLE: HIGGS FIT
VECTORIZATION
~3.3X
MAX 4
PARALLELIZATION
~3.8X
MAX 4
PARALLELIZATION + VECTORIZATION
~11X
MAX 16
Higgs Fit
Current implementation
Vectorized plus parallelized implementation
Performance of the Higgs Fit on your laptop
Vectorizing the Higgs Fit (4 physical cores)
Higgs Fit Scaling With Number of Cores
Scaling of the least squares fit (14 cores + AVX2)
Autovectorization in ICC
Compiler performance
Compiler performance for the Unbinned. Likelihood Fit (4 cores + SSE2)

Fitting parallelization



Tools for parallelism

Task level: **ROOT::TThreadExecutor**

Instruction level: **VecCore**

Backwards compatibility!

Spot the differences

```
//Higgs Fit: Implementation of the scalar function
double func(const double *data, const double *params)
{
   return params[0] * exp(-(*data + (-130.)) * (*data + (-130.)) / 2) +
          params[1] * exp(-(params[2] * (*data * (0.01)) - params[3] *
          ((*data) * (0.01)) * ((*data) * (0.01))));
}

TF1 *f = new TF1("fScalar", func, 100, 200, 4);
f->SetParameters(1, 1000, 7.5, 1.5);
TH1D h1f("h1f", "Test random numbers", 12800, 100, 200);
h1f.FillRandom("fvScalar", 1000000);
h1f.Fit(f);
```

```
//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) +
          params[1] * exp(-(params[2] * (*data * (0.01)) - params[3] *
          ((*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");
```

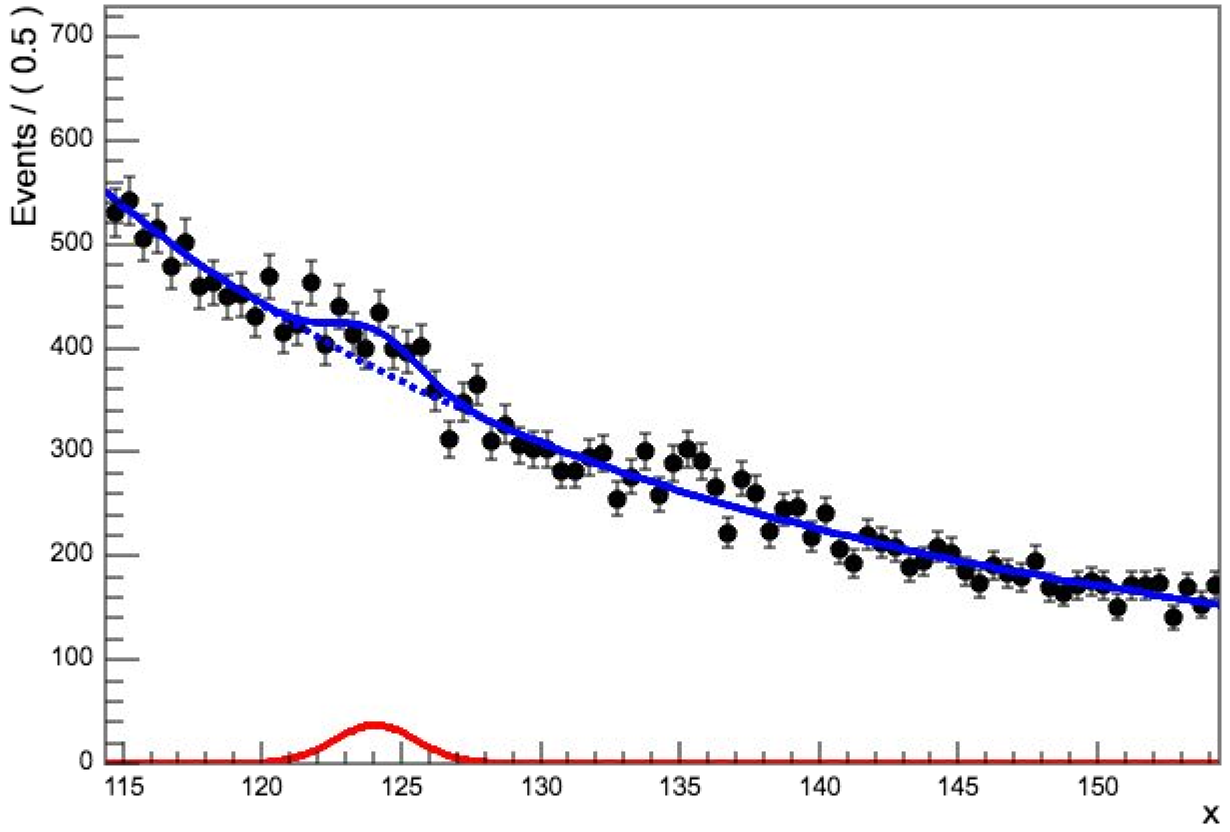
Backwards compatibility!

```
//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) +
          params[1] * exp(-(params[2] * (*data * (0.01)) - params[3] *
          ((*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!

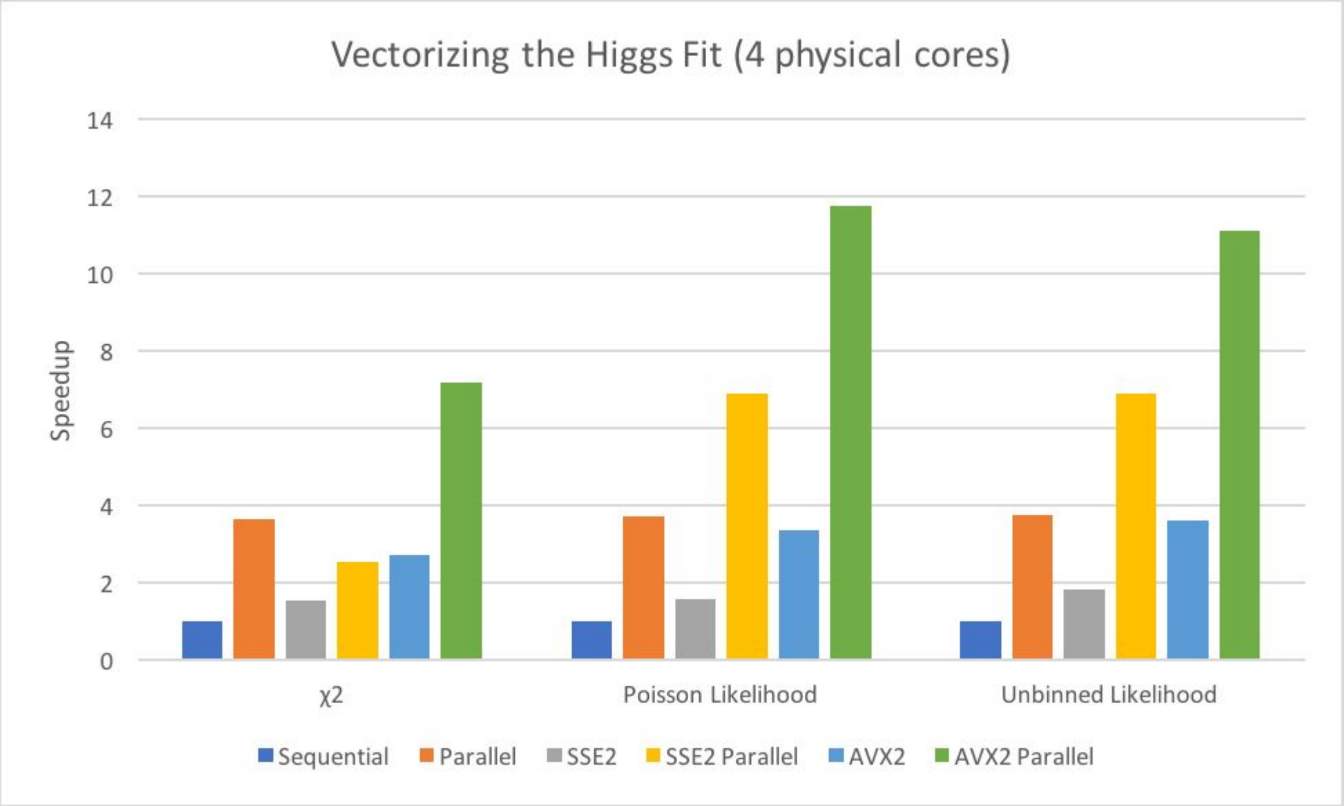


Events / (0.5)
700
600
500
400
300
200
100
0
115
120
125
130
135
140
145
150
x

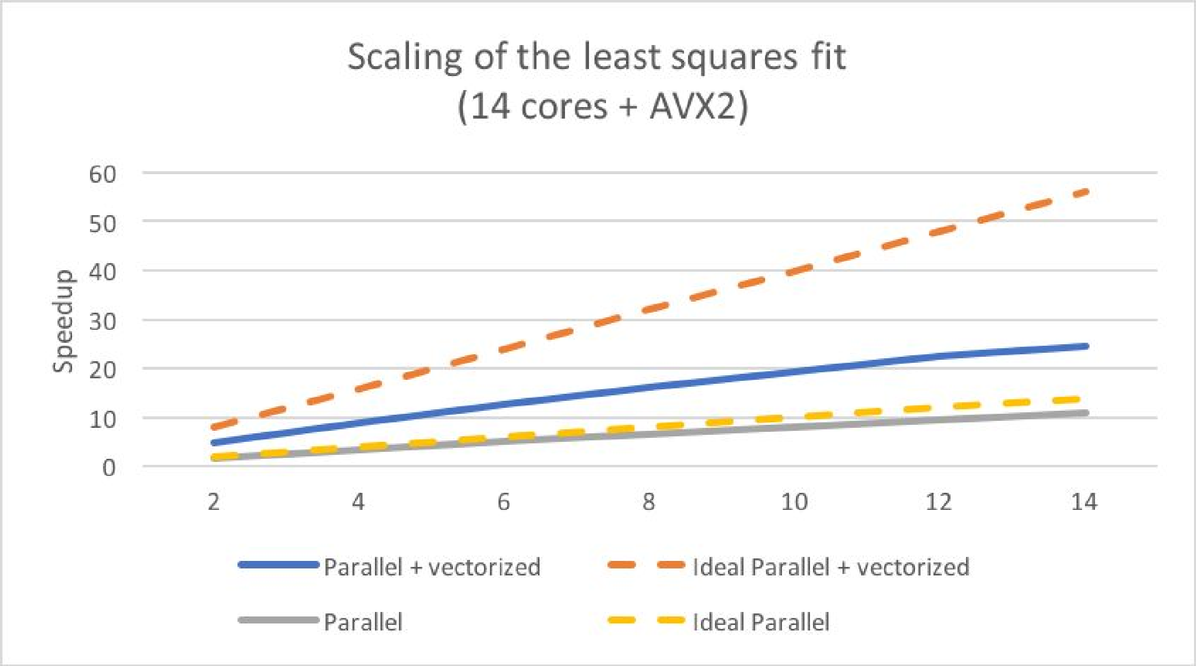
VECTORIZATION	PARALLELIZATION	PARALLELIZATION + VECTORIZATION
~3.3X	~3.8X	~11X
MAX 4	MAX 4	MAX 16

On your PC!

Big images, big results



Multithreaded Speed up



Compiler performance