PROCEEDINGS OF SCIENCE

MINUIT Package Parallelization and applications using the RooFit Package

Alfio Lazzaro[*ab] **and Lorenzo Moneta**[a]
[a] *CERN, Geneve*
[b] *Universita' degli Studi and INFN, Milano*
E-mail: alfio.lazzaro@mi.infn.it, lorenzo.moneta@cern.ch

The fitting procedures are based on numerical minimization of functions. The MINUIT package is the most common package used for such procedures in High Energy Physics community. The main algorithm in this package, MIGRAD, searches the minimum of a function using the gradient information. It requires the calculation of the derivative for each parameter of the function to be minimized. This procedure can be parallelized on multi-cores architectures and multiple nodes in a cluster. We present in this paper an implementation of such parallelization using OpenMP and MPI techniques, respectivately. Furthermore we will show applications using the RooFit packages for Maximum Likelihood fits, and possibility of hybrids between MPI and multi-threads (taking advantage from multi-cores architecture).

XII Advanced Computing and Analysis Techniques in Physics Research
November 3-7, 2008
Erice, Italy

[*] Speaker.

PoS(ACAT08)083

1. Introduction

In general all the methods used in data analysis are based on optimization problems. Depending on the particular method, the evaluation of a function is required, like the maximum likelihood function or the expected prediction error function, which has to be optimized as function of several free parameters to be determined [1]. In the last years many complex techniques are being used in the High Energy Physics (HEP) community, like maximum likelihood fits, Neural Networks, Boosted Decision Tree [2]. These techniques have several advantages with respect to the simple *cut and count* analysis method, such as better discrimination between signal and background events, the possibility to take in account errors with a better precisions, and to consider correlations between the discriminating variables used in the analysis.

Increasing the samples of data analyses and using complex algorithms require high CPU performance. In general the requirement on CPU-time depends on:

- number of free parameters to be determined
- number of input events in the process
- complexity of the model to be evaluated

In the last years, vendors like Intel and AMD have not incremented the performance of single CPU unit as in the past, but they are working on multi-core CPU. Currently we have up to 6 cores implemented on one single chip. This fact represents a possible revolution in the development of new programs. Indeed we can parallelize the code using a shared memory paradigm (such as OpenMP) obtaining great benefits from new multi-core architectures. So we have to reformulate some algorithms generally used for HEP data analyses. It is also possible to have faster execution of the code using the Message Passing Interface (MPI) paradigm, spreading the execution of the code over several CPUs in a cluster. These techniques of High Performance Computing (HPC) are well established in other fields, like computational chemistry and astrophysics. In HEP community there is not such a large use, but in the future it can be an elegant solution in all the cases where the data analyses will get more and more complicated.

In the work described in this paper we focus on the parallelization of maximum likelihood (ML) fitting code, focusing on the likelihood function calculation based on the RooFit package [3], and optimization of the ML function using the MINUIT package [4]. We will present the techniques adopted for the parallelization and some speed-up examples.

2. Maximum Likelihood Technique

In this section we briefly introduce the maximum likelihood technique. More details can be found elsewhere [5].

We consider a random variable x (or a multidimensional random vector $\hat{x} = (x_1, \ldots, x_n)$) distributed with a distribution function $f(x;\theta)$. We assume $f(x;\theta)$ to be well known except for the parameter θ (or parameters $\hat{\theta} = (\theta_1, \ldots, \theta_p)$). So, $f(x;\theta)$ expression represents, after normalizing it, hypothesized probability density function (PDF) for the x variable. Then, we suppose to perform an experiment where a measurement has been repeated N times, supplying $x_1, \ldots, x_N$ values. The

maximum likelihood method is a technique to estimate the parameter value from this data sample. Defining the *likelihood function* $\mathscr{L}$ as

$$\mathscr{L}(\theta) = \prod_{i=1}^{N} f(x_i;\theta), \tag{2.1}$$

to estimate the parameter value we have to maximize this function (*i. e. maximum likelihood*). We should underline that x_i are measured and the $f(x;\theta)$ function is well-known, so $\mathscr{L}$ only depends on the parameter we want to fit. The evaluation of maximum for likelihood $\mathscr{L}$ as function of the unknown parameter can be done in a numeric way. Usually, it is used to minimize the equivalent function $-\ln(\mathscr{L})$, the *negative log-likelihood* (*NLL*), or the function $\chi^2 = -2\ln(\mathscr{L})$. So the *NLL* has the form:

$$NLL = -\sum_{i=1}^{N} \left(\ln f(x_i;\theta)\right), \tag{2.2}$$

that is, a sum of logarithms. The evaluation of the *NLL* used in this paper is done in the ROOT framework using the RooFit package [6].

3. Optimization procedure: MINUIT

There are several algorithms to find the minimum of a function [1]. Among them, the most common method used in the HEP community is based on the MIGRAD algorithm inside the MINUIT package. MIGRAD performs the minimization of a function using the *variable metric* method [7]. This method involves the calculation of the derivatives of the *NLL* for each free parameter. Since very often we are faced with minimizing a function for which no derivatives are provided, MIGRAD is able to estimate the derivatives of the function by finite differences. Details of the implementation of the method used in MIGRAD can be found elsewhere [4]. Here we simply say that for the first derivative we can use the formula

$$\left.\frac{\partial NLL}{\partial \hat{\theta}}\right|_{\hat{\theta}} \approx \frac{NLL(\hat{\theta}+\hat{\mathrm{d}}) - NLL(\hat{\theta}-\hat{\mathrm{d}})}{2\hat{\mathrm{d}}}, \tag{3.1}$$

where $\hat{\theta}$ is the set of free parameters and $\hat{\mathrm{d}}$ is a "small" displacement (step-size $\hat{\mathrm{d}}$). The value for $\hat{\mathrm{d}}$ should be chosen as small as possible, but still large enough so that the rounding error in the computation of *NLL* does not become larger than the error introduced by the approximation. This approximation requires $2p$ function calls for p free parameters to estimate the first derivative. Then, of course, the whole procedure has to be repeated for each point $\hat{\theta}$ of the minimization procedure until the minimum is reached. Depending of the complexity of the *NLL* function, this can be very time-consuming. Furthermore, we should consider that in the *NLL* function we use PDFs that need to be normalized for each iteration of the minimization procedure, in case the integral of the PDFs depends on the free parameters. This requires to calculate the integral of the function, which is also time-consuming. Specific examples of maximum likelihood fits can require several hours [8].

4. Parallelization

Summarizing what we described in the previous sections, we can distinguish three parts of the maximum likelihood fit procedure:

1. *NLL* calculation (implemented in RooFit): the likelihood function is calculated over the input events. From formula 2.2, this is a sum of terms over the N events (scaling as N);

2. PDFs normalization (implemented in RooFit): it depends on the complexity of the function, in particular the calculation can be very slow if we do not have an analytical expression of the integral;

3. minimization procedure (implemented in MIGRAD): it requires the gradient calculation, *i. e.* the calculation of the derivatives for several different free parameters values, which depends on the number of free parameters p to be determined (scaling as $2p$).

In principle all these parts can be simply parallelized, using a combination of shared-memory and MPI paradigms.

Currently there is no implementation of parallelization of the point 2.

For the point 1 the parallelization can be easily achieved splitting the sum term in the *NLL*. This is done in RooFit using *fork* calls. It allows a sensible reduction of the duration of maximum likelihood fits in case we run on multi-core machines, requiring one core for each thread. In this way the speed-up scales almost with a factor proportional to the number of threads. However, this parallelization used in RooFit is not a shared-memory solution and is therefore limited by the total number of memory available in the multi-core machine.

The work presented in this paper concerns the point 3, using either a shared-memory solution (using OpenMP) or MPI on multiple nodes of a cluster. In the latter we also show the hybrid solution with the parallelization of point 1 on the single node. The parallelization is done splitting the derivative calculation over several *processes* (OpenMP or MPI processes), balancing the number of derivatives (which are equal to the number of free parameters in the *NLL* function) for each process. This means that the maximum number of processes is equal to the number of free parameters p. For example with 10 free parameters and 3 processes, we have four derivatives for one process, and three each for the other two. Each process has a common serial initialization part. Then we have the splitting of the derivative calculation, where each process takes care of his group of derivatives. At the end of this stage, all results are scattered between the processes, so that they can conclude the minimization iteration (also this part is in common between all the processes, *i. e.* each process proceeds in the same way in the minimization). This represents an iteration of the minimization procedure, which is repeated until the minimum of the function is reached. Figure 1 shows an illustrative schema of this procedure.

It is important to note that the time spent for each iteration depends on the slowest process, *i. e.* the process with the most derivatives to calculate. This fact suggests the maximum number of processes where we can see an effective speed-up. For example, with 10 free parameters, there is at least one process with 2 parameters when we require between 5 and 9 processes in the parallelization. Of course increasing the number of processes means overhead for the synchronization and communication between the processes. Therefore a good speed-up can be reached in case of large number of free parameters and reasonable low number of processes.

In the following example reported, we used MINUIT2, which is just the C++ object-oriented version of the original MINUIT [9], and RooFit implemented in ROOT v5.20.

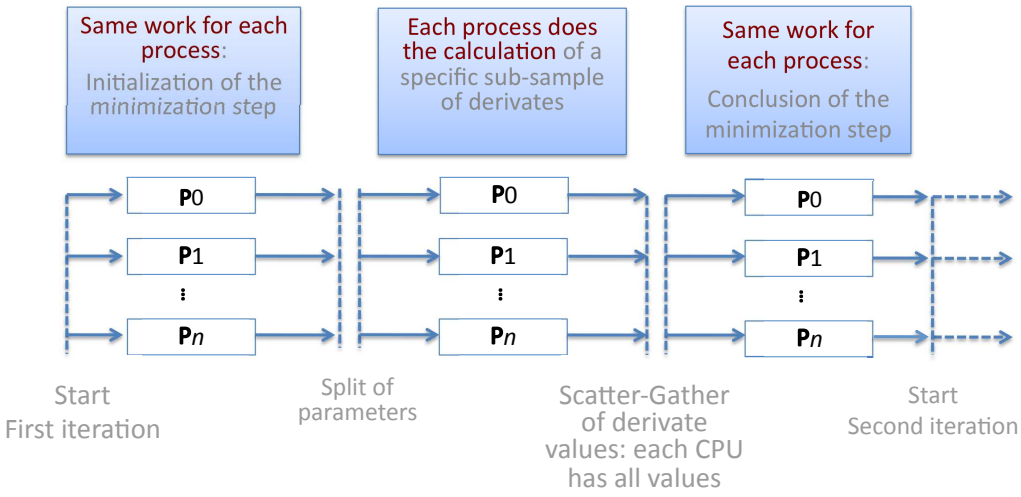

Figure 1: Scheme of the parallelization of the minimization procedure (see text for details).

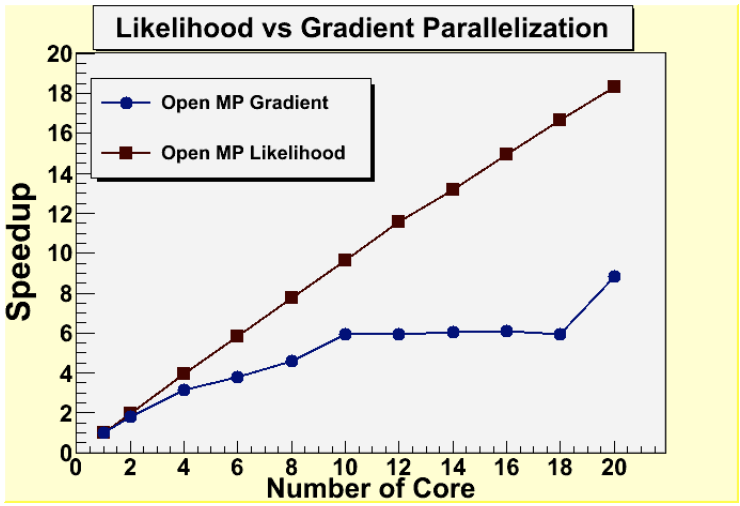

Figure 2: Speed-up plot when using OpenMP parallelization of the minimization procedure (blue circles) and of the *NLL* calculation (brown squares). Note that the plateau when using 10–18 cores corresponds to at least one process with 2 derivatives to calculate, so no increase in the speed-up is expected (see text for more details).

In the figure 2 we show the speed-up using OpenMP for a test done with 50 Gaussian uncorrelated variables, 20 free parameters and 10,000 events, running on a machine with 24 cores and 64 GB shared memory [1]. For comparison in the same plot we show the speed-up that we can reach using the parallelization of the *NLL* calculation. There is a better response for this second parallelization, which is reasonable since the parallelization of the minimization procedure has more sequential part of the code and does not scale well in case of small number of parameters (as mentioned above).

[1] This machine has been provided by OpenLab at CERN.

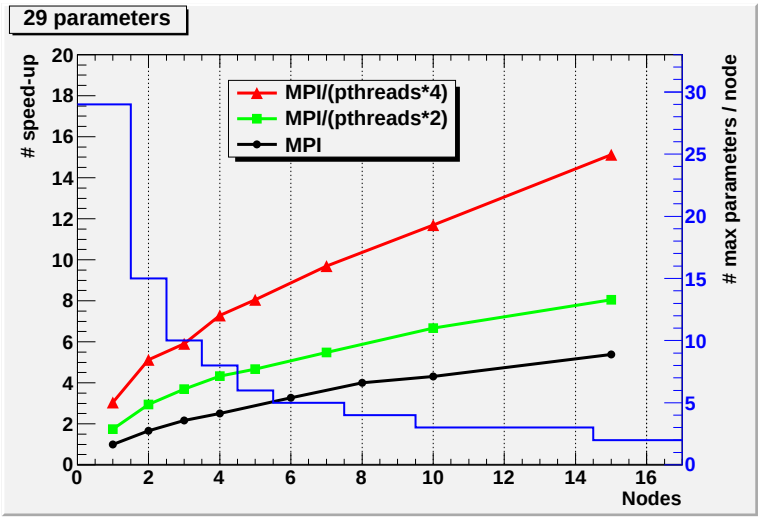

Figure 3: Speed-up plot when using MPI parallelization of the minimization procedure (black circles) for a function with 29 free parameters. Blue line (referring to right y-axis) refers to the maximum number of derivatives calculate for each node. The other two lines refer to the hybrid case of MPI and RooFit *NLL* parallelization requiring 2 (green squares) and 4 (red triangles) cores for each node. Note the 4 cores is the maximum number available for each node in the cluster used for the test.

Concerning the parallelization using MPI, the tests have been done on an IBM cluster with dual-core CPUs and 2 slots per node [2], *i. e.* 4 cores per node, with 8 GB/node of shared memory. In total the cluster has 1280 nodes, which means 5120 cores in total. The internal network for the communication is provided by Infiniband (5 GB/s). All the MPI calls are declared in a specific class of the MINUIT2 package. In particular for the communication and synchronization we use the MPI `Allgatherv` call. In figure 3 we show the speed-up for a test with 10,000 events, 4 variables, and 29 free parameters. In the same plot we show the case when we also apply the hybrid parallelization for *NLL* calculation in each node.

5. Conclusion

The solutions adopted for paralellization give good results for the tests done. Further improvements can be the parallelization of the integral calculation of the PDFs and the possibility to have also the *NLL* calculation done on multiple nodes using MPI. The work presented in this paper have required changes in the RooFit and MINUIT2 packages, which will be part of the future release of ROOT. The parallelization of the minimization procedure implemented in MINUIT2 can be used in all domains where it is required such a procedure, *i. e.* not only *NLL* fits, hence in general in data analysis code based on MINUIT.

6. Acknowledgments

Authors thank CINECA HPC group for their kind support during the tests done on his cluster

[2] The cluster has been provided by CINECA HPC group at Bologna, Italy.

and OpenLab group to provide the possibility to use his test machine. Alfio Lazzaro thanks Filippo Spiga for his suggestions about the MPI parallelization.

References

[1] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, Third Edition, Cambridge University Press (2007).

[2] J. Friedman, T. Hastie, R. Tibshirani, *The Elements of Statistical Learning*, Springer (2001).

[3] See the web page of the RooFit package: `URL:http://roofit.sourceforge.net/`.

[4] F. James, *MINUIT - function minimization and error analysis*, CERN Program Library Long Writeup D506 (1972).

[5] G. Cowan, *Statistical Data Analysis*, Clarendon Press (1998).

[6] See the web page of the ROOT project: `URL:http://root.cern.ch/`.

[7] W. C. Davidon, SIAM J. Optim. **1**, 1, pp. 1-17 (1991).

[8] B. Aubert *et al.*, *BABAR* Collaboration, Phys. Rev. Lett. **98**, 211802 (2007).

[9] M. Hatlo *et al.*, *IEEE Transactions on Nuclear Science* **52-6**, 2818 (2005).