PARALLEL FRAMEWORK FOR PARTIAL WAVE ANALYSIS AT BES-III EXPERIMENT

V.A. Tokareva[a], I.I. Denisenko

Laboratory of Nuclear Problems, Joint Institute for Nuclear Research, 6 Joliot-Curie, Dubna, Moscow region, 141980, Russia

E-mail: [a] tokareva@jinr.ru

The most common approach to the partial wave analysis in modern experiments is the event-by-event maximum likelihood fit. Within this approach the analysis of the collected so far statistics at BESIII (more than one billion of J/ψ events) typically requires huge amount of a computational time. Fortunately, the event-by-event analysis can be naturally parallelized. We developed the parallel cross-platform software architecture that can run calculations at various high-performance computing platforms, such as multi-core CPUs, Intel Xeon Phi co-processors, and GPUs. The software supports switching between different minimization algorithms like MINUIT or FUMILI. The wave functions amplitudes are constructed using covariant tensor formalism. Currently, analysis is developed for the $J/\psi \rightarrow K^+K^-\pi^0$ decay channel. The algorithm for caching the intermediate results has been developed, that minimizes the amount of calculations performed in each iteration. Besides, a number of software optimizations has been used, including vectorization, memory access linearization, and data alignment. In future we plan to add the analysis for new reaction channels, the option for combined analysis of several reaction or decay channels and opportunity to adapt our software for use in other experiments.

Keywords: partial wave analysis, big data, heterogeneous computing, high-performance computing, Xeon Phi, GPU, minimization algorithms.

1. Introduction

Partial wave analysis (PWA) is the technique of studying scattering or decay of hadrons based on the decomposition of the process amplitude to partial waves. This method is widely used nowadays in HEP (high energy physics) experiments such as BESIII, LHCb, COMPASS etc. in order to search for new and exotic resonances, that can not be seen directly as a peak in the mass spectrum.

The main objective of the BES-III experiment, which takes place at Beijing Electron–Positron Collider II (China), is investigating the properties of charmonia and charmed mesons. The physics program of BESIII includes light hadron spectroscopy in charmonia decays. The typical process that can be studied in BESIII experiment is the $J/\psi \rightarrow K^+K^-\pi^0$ decay. For the moment, $(1310.6 \pm 7.0) \times 10^6$ billions [1] J/ψ events have been accumulated and the number is expected to rise up to 10 billions.

From the other side, the unbinned likelihood PWA method is a computationally expensive one (see the discussion of PWA cost issues in [2]), and has a number of restrictions related to minimization issues, which will be considered further. The third point is that models tend to be complicated, and the larger number of parameters we have, the more time-consuming the calculations are.

Thus, performing PWA for the typical J/ψ process mentioned above provides us with a number of challenges to solve because the expensive calculations have to be performed under the big data amounts in appropriate time. The widely-used nowadays approach to solution of such kind of problems is optimization and refactoring of existing algorithms for effective usage on massive-parallel hardware (GPU, multi-core CPU, Xeon Phi processors, etc.).

In this article we present the current state of the parallel PWA framework being developed at DLNP JINR.

2. Issues of the task and our optimization approach

In our studies used covariant tensor formalism [3] to described partial waves and assumed that decay amplitude A can be described with the isobar model (i.e. decay amplitude is given by a sum of resonances for each pair of our final mesons). Our fit model consisted of set of resonances with masses, width and couplings to be determined in the fit. Given the fit model we can calculate probability to observe data event i with measured momenta of final particles:

$$p_i = |A_i|^2/\sigma, \quad (1)$$

where σ is a normalization integral over the phase space ($\sigma = \int |A|^2 \, d\Phi$) . In following this integral will be approximated as MC-sum : $\int |A|^2 \, d\Phi = C \sum_k |A_k|^2$ (index k runs over sample of MC phase space distributed events). Finally log-likelihood F to be minimized is given by

$$F = -lnL = -ln(\prod_i p_i) = -\sum_i ln\, p_i, \quad (2)$$

where the product and the sum are taken over data events. An important feature of this approach is that detector data selection efficiency can be trivially introduced.

The classical approach to solving such tasks include using gradient minimization methods where the values of parameters are being varied on the sequential basis and the numerical value of the log-likelihood function is being recalculated, respectively. At the end of every step the direction of next step (i.e. the direction where the function gradient is decreased in the fastest way) is chosen. In this work, we use the realizations of gradient minimization methods included in the ROOT framework (see further).

Since the contributions to the log-likelihood function could be estimated independently for different events, the task can be naturally parallelized. The data access linearization provides the decreasing of execution time too, especially for the multi-core devices supporting vector calculations (modern multi-core CPUs, Intel Xeon Phi coprocessors, etc.). The main formula could be separated in parts, which can be calculated independently and do not need to be recalculated in case, if there were no changes in this part parameters on the current step.

Thus, the properly organized data caching is helpful for decreasing computational costs. The speed up even increases with the increasing the model complexity (as more parameters mean more

cached factors that do not need to be recalculated) and the number of events being analyzed (as the relative importance of overheads for creating the parallel regions diminishes).

Performance has been measured with 10^5 data events and a PHSP sample of data for normalization consisting of 10^6 events.

3. Minimization notes

Minimization of a log likelihood function is a narrow place for any PWA framework development. Various minimization algorithms have different convergence and demand various computation times per iteration. It was noticed in [2] that it is exponentially hard to determine whether the likelihood fit (usually with a large number of free parameters) has found the global minimum or one of the local ones.

Using the approved and broadly known minimization tools, besides obvious advantages, do not provide the opportunity to fine-tune the inner work of a minimizer.

Such issues are being solved in different ways at the others PWA projects [4]-[8]. It is quite typical for the frameworks in this domain to support minimization employing one or two minimization frameworks. For these and other essential details about PWA frameworks supported nowadays, see Table 1.

Table 1. Great variety of PWA frameworks being developed

Framework	*Language*	*Support of parallel computations*	*Minimization*	*Actual release*
Fortran PWA	Fortran	—	Fumili [9]	80's
GPUPWA	C++	GPU (OpenCL)	Minuit2 [10]	2011
PyPWA	Python	—	PyFit [11]	2015
ComPWA	C++	CPU	Minuit2, Geneva [12]	—
ROOTPWA	C++, Python	GPU (CUDA), CPU (MPI)	Minut2	2015
Our framework	C++	CPU (OpenMP), Xeon Phi (OpenMP), GPU (CUDA)	Minuit, Fumili	—

In our framework we support minimization with both Minuit [9] and Fumili [10] packages.

In Fig. 1 measurements of the convergence times are shown for the $J/\psi \rightarrow K^+K^-\pi^0$ decay reaction, employing Fumili, MIGRAD, and SIMPLEX minimization algorithms with 2 and 4 resonances taken as initial conditions.

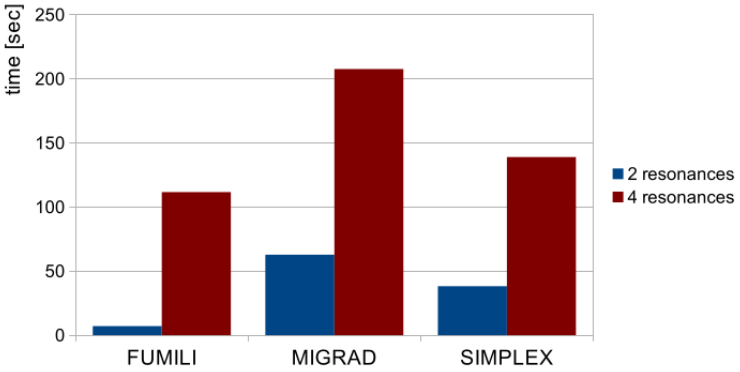

Figure 1. Convergence time for the different minimization methods

4. Algorithm and realizations

Presently, the industry of multi-core devices is rapidly developing and almost every computing device around us already has several cores inside. It is reasonable to assume that proper

use of all computing cores allows one to speed up the execution of the code without losing the quality of calculations. However, any implementation of the framework aimed at parallel computations inevitably should deal with the volumes of simultaneously available RAM, the bandwidth of accessing it, and the resolution of collisions in case of sequential atomic operations. Individual approach of the developers to solving these issues is implemented through the choice of general software architecture of the framework, as well as the tools and the platforms for parallelization.

In our framework we implemented support for two popular solutions for heterogeneous clusters: either of control CPU and executing GPU or control CPU and executing coprocessor Intel Xeon Phi of the first generation. Our framework also supports performing multi-threaded computations on a multi-core CPU — this category, besides CPU nodes present on most modern computing clusters, includes also the majority of user-end PCs sold nowadays, potentially expanding the audience of users for our framework.

Fig. 2 shows the results of comparative testing of the framework on various hardware platforms: CPU (represented by 2 ×Intel Xeon E5-2695v2), GPU (NVIDIA Tesla K80), and Intel Xeon Phi (model 5110P, denoted as PHI).

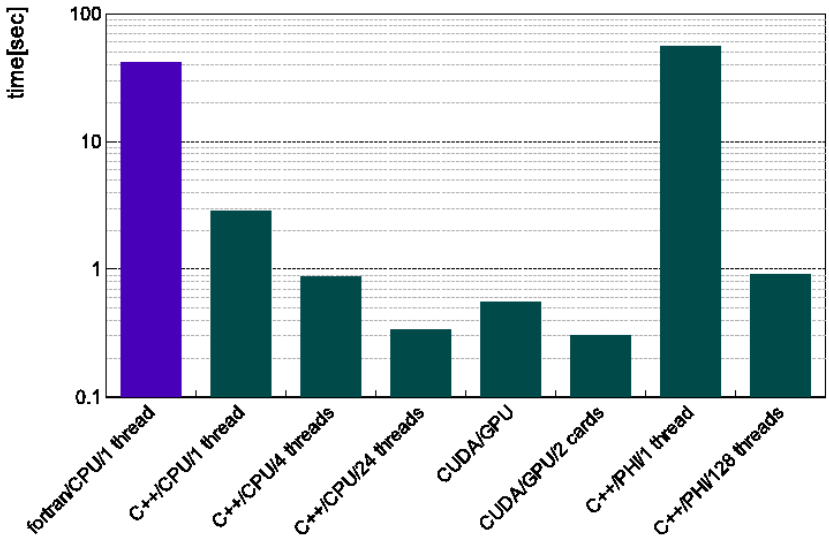

Figure 2. Average call time for test function with gradients, 5 resonances

GPU is deservedly the most popular tool for solving massively-parallel tasks. We can see in our case it shows the best performance as well. CPU performance is comparable to that of GPU. Xeon Phi looks like not as effective, probably because it is more sensitive to vectorization issues like memory alignment. Fine-tuning of the code for a better performance on Phi coprocessors is included in our future plans.

For CPU and Phi processors we have taken the numbers of threads, where the system reaches its maximum performance and/or a plateau. More details on the performance of the parallel algorithm depending on the number of cores can be seen in Fig. 3-4. We see that the speedup at low amounts of threads increases almost linearly and reaches plateau at the number of threads that equals approximately the number of physical CPU cores, meaning that for this task hyper-threading brings almost no effect since there are almost no I/O operations.

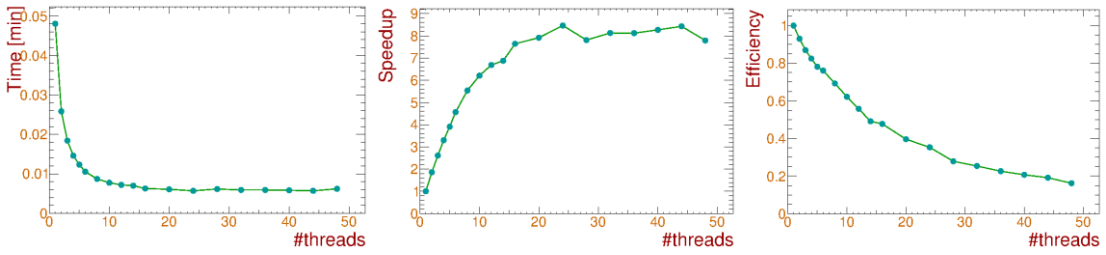

Figure 3. CPU benchmarks: time, speedup, efficiency (2 Intel Xeon E5-2695v2 CPUs)

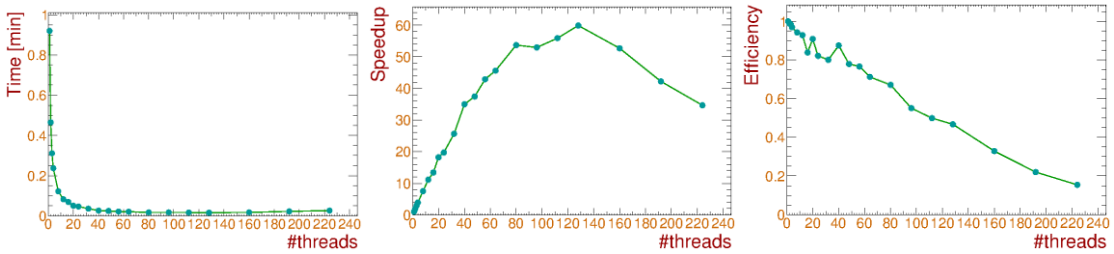

Figure 4. Co-processor benchmarks: time, speedup, efficiency (Intel Xeon Phi 5110P)

5. Future plans

Our priorities involve employing other models and reactions which are typically used in PWA to support our users with a wide range of predefined analysis patterns.

We are going as well to increase framework cross-platform stability, providing a big deal of benchmarks, unit tests and user documentation.

We are planning to make a comparative study of performance and features for the currently available PWA frameworks, which is going to bring massive benefits for the community of PWA users and product developers and advance community searches for the general PWA framework solution to a new discussion level.

Additionally speeding up for the calculations could be discussed in terms of lower level code specifications and searches for new applicable architecture patterns.

Our priorities also include an implementation of ROOT-independent PWA framework version (on the base of the stand-alone versions of Minuit and Fumili minimizers).

6. Acknowledgements

Authors express their gratitude to the team of HybriLIT heterogeneous cluster for support, discussions and provided resources.

We are thankful to the Fumili minimizer authors for the fruitful discussions on optimization theory and applications.

7. Conclusion

The framework for the partial wave analysis, which we are working on, employes data-level parallelisation of calculations event-by-event maximum likelihood estimation using the benefits of brand-new high-performance computing hardware (such as fast parallel processing of big amounts of data and good task scalability). Since during the technical revolution such kind of computational devices became cheaper and easier-available not only for big organizations, but for personal users as well, our framework is organized in such a way individual researchers could use it with a maximum profit.

The framework architecture is intended for calculations performing on a wide range of popular multi-core devices, which are multi-core CPUs, Intel Xeon Phi coprocessors, and GPUs. Minimization is provided on the basis of both Minuit and Fumili minimizers.

In this research, sensitive architecture principles were discussed, performance benchmarks for calculations on different types of devices and for different minimization tools were provided for predefined analysis of the $J/\psi \rightarrow K^{+}K^{-}\pi^{0}$ decay reaction employing the data samples accumulated by the BES-III experiment.

References

[1] Ablikim M. et al. Determination of the number of J/ψ events with inclusive J/ψ decays. // Chinese Phys. C, 2017, vol. 41, No. 1 (013001). DOI: 10.1088/1674-1137/41/1/013001.

[2] Berger N. Partial Wave Analysis using Graphics Cards. // Proceedings of the XIV International Conference on Hadron Spectroscopy (hadron2011), Munich, 2011, edited by B. Grube, S. Paul, and N. Brambilla, eConf C110613 (2011) [arXiv:1108.5882v1], pp. 810-817.

[3] Zou B.S., Bugg D.V. Covariant tensor formalism for partial wave analyses of ψ decay to mesons. // arXiv:hep-ph/0211457v2. Web. 26 Dec 2002.
[4] GPUPWA framework, available at http://sourceforge.net/projects/gpupwa/
[5] ROOTPWA framework, available at http://sourceforge.net/projects/rootpwa/.
[6] PyPWA framework, available at https://github.com/JeffersonLab/PyPWA
[7] ComPWA framework, available at https://github.com/ComPWA/ComPWA
[8] Model-independent partial wave analysis using a massively-parallel fitting framework. //submitted to the proceedings of the 22nd International Conference on Computing in High Energy and Nuclear Physics, CHEP 2016, arXiv:1703.03284v1. Web.19 Feb 2017.
[9] James F., Roos M. Minuit - a system for function minimization and analysis of the parameter errors and correlations. //Comp. Phys. Communications, vol. 10 (1975), pp. 343 - 367.
[10] Dymov S. N. et al. Constrained minimization in C++ environment. // Nucl. Instrum. Meth. A440, 431-437 (2000).
[11] PyFit, available at http://www.stsci.edu/institute/software_hardware/pyfits/Download
[12] Geneva optimisation tools, available at https://www.gemfony.eu/index.php?id=geneva