EPJ Web of Conferences **214**, 06028 (2019) https://doi.org/10.1051/epjconf/201921406028
*CHEP 2018*

# Deployment of a Matrix Element Method code for the ttH channel analysis on GPU's platform

*Gilles* Grasseau[1,*], *Florian* Beaudette[1,**], *Cristina* Martin Perez[1,***], *Alexandre* Zabi[1,****], *Arnaud* Chiron[1,†], *Thomas* Strebler[2,‡], and *Gabriel* Hautreux[3,§]

[1]Laboratoire Leprince-Ringuet, Ecole polytechnique, Palaiseau, France
[2]Laboratoire Leprince-Ringuet, Ecole polytechnique, Palaiseau, France. Now at Centre de Physique des Particules de Marseille, Aix-Marseille Université, CNRS/IN2P3, Marseille, France
[3]GENCI "Grand Equipement de Calcul Intensif", 6 bis rue Auguste Vitu, Paris, France

**Abstract.** The observation of the associated production of the Higgs boson with two top quarks in proton-proton collisions is one of the highlights of the LHC Run 2. Driven by the theoretical description of the physics processes, the Matrix Element Method (MEM) consists in computing a probability that an event is compatible with the signal hypothesis (ttH) or with one of the background hypotheses. It is a powerful classifying tool requiring high dimensional integral computations. The deployment of our MEM production code on GPU's platform will be described. What follows will focus on the adaptation of the main components of the computations in OpenCL kernels, namely the Magraph matrix element code generator, VEGAS, and LHAPDF. Finally, the gain obtained on GPU's platforms compared with classical CPU's platforms will be assessed.

## 1 Introduction

The recent observation of the associated production of the Higgs boson with two top quarks [1] is one of the highlights of the Run 2 at the LHC. Since its observation in 2012 [2, 3], the properties of the new Higgs boson have been extensively tested, and in particular, its couplings to other particles. In the Standard Model, the fermions couple to the Higgs boson through a Yukawa interaction with a strength proportional to the fermion mass. Consequently, the coupling of the Higgs boson to the $b$ and $\tau$ light fermions have been inferred from the decay rate of the Higgs boson in $bb$[4] and $\tau\tau$[5] respectively. However, this strategy does not apply to the coupling with the top quark, $y_t$, as the decay of the Higgs boson into a top quark pair is kinematically forbidden. Instead, the $y_t$ coupling can directly be measured from the rare associated $ttH$ production process.

The combination of Run 1 and 2016 Run 2 data from LHC collisions allowed the first observation[1] of the simultaneous production of a Higgs boson with a $t\bar{t}$ pair in April 2018.

*e-mail: grasseau@llr.in2p3.fr
**e-mail: beaudette@llr.in2p3.fr
***e-mail: mperez@llr.in2p3.fr
****e-mail: zabi@cern.ch
†e-mail: chiron@llr.in2p3.fr
‡e-mail: thomas.strebler@cern.ch
§e-mail: hautreux@genci.fr

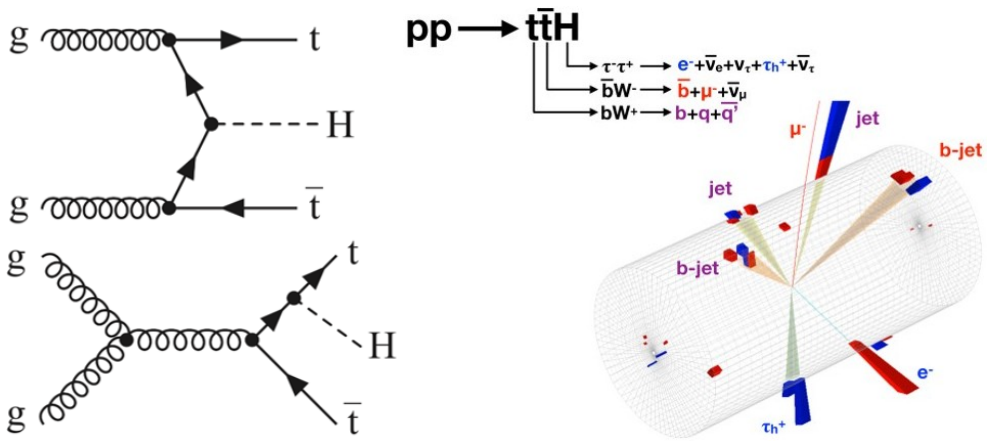


**Figure 1.** *ttH* production process: (*Left*) Feynman diagrams of the Higgs boson production in association with top quarks, resulting from top quark fusion (top) or Higgs radiation from a top quark (bottom); (*Right*) an event candidate for the production of a top quark and top anti-quark pair in addition with a Higgs Boson in the CMS detector[6].

## 2 The Matrix Element Method

Unlike supervised methods (neural networks, decision trees, support vector machines, ...), the Matrix Element Method[7] allows events to be classified by computing the probability that a final state corresponds to a given physics process solely thanks to the physics laws involved. Among the new emerging computing models based on Machine or Deep Learning, it is essential to have a theory-driven model which plays the role of computational reference. This sophisticated method is however very CPU time consuming due to the investigation of all the main physical cases and requires huge powerful computing platforms to perform the CMS analyses carried out at our laboratory in a reasonable time. Thanks to the pioneering works of [9, 10] on the implementation of the MEM on a single GPU, we were confident that the multi-GPU implementation was achievable.

### 2.1 Principle

Given the various possible decay channels of Higgs boson and of the $W$ resulting from the $t \to Wb$ decay, the *ttH* signal can result in a large variety of final states. In the analysis considered here, one of the $W$ bosons is required to decay leptonically and the Higgs boson is required to decay into $\tau^+\tau^-$ (Fig. 2). This final state offers a good compromise between the total branching ratio and the purity. To further reduce the background, the charged leptons produced in the $\tau$ and $W$ decay are required to have the same electric charge.

For each event of the data set to analyze, the probabilities or weights are evaluated for a given theory model (*signal* or *background*). This probability expression is shown below. An integration is performed over all the possible values of the generator-level variables $\boldsymbol{x}$ and the Bjorken fractions of the incoming partons $x_a$ and $x_b$. The response of the detectors is modelled through Transfer Functions (TF). In particular, gaussian response functions turn out to be very well adapted to mimick the jet response, as well as the missing transverse momentum (x,y) components. For the tau leptons, the semi-analytic TF have been used.

They predominantly account for the emissions of neutrino(s) and therefore depend on the tau decay channel.

$$w_i(y) = \frac{1}{\sigma_i} \sum_p \int \boldsymbol{dx} dx_a dx_b \frac{f(x_a, Q).f(x_b, Q)}{x_a x_b S} \delta^4(x_a P_a + x_b P_b - \Sigma P_k) \, |M_i(\boldsymbol{x})|^2 \, W(\boldsymbol{y}||\boldsymbol{x}) \quad (1)$$

This probability $w_i(y)$ sums all possible contributions of the matrix element $|M_i(\boldsymbol{x})|^2$ (evaluated in the rest frame of the outgoing particles) for a process $i$ and is convolved by the Transfer Function $W(\boldsymbol{y}||\boldsymbol{x})$, the probability to have the observed state described by the variables $\mathbf{y}$, given the kinematic variables $\mathbf{x}$. A second ponderation $\frac{f(x_a,Q).f(x_b,Q)}{x_a x_b S}$ gives the probability that the two incoming proton particles or partons (proton quarks or gluons) interact, in which $f(x_a, Q)$ is the Parton Density Function (or PDF) for the parton $a$. All the parton combinations are described by the $\Sigma_p$ term. Finally, the $\delta^4(x_a P_a + x_b P_b - \Sigma P_k)$ term corresponds to the kinematic constraints between the p-p incoming partons and the outgoing particles considered in the Matrix Element.

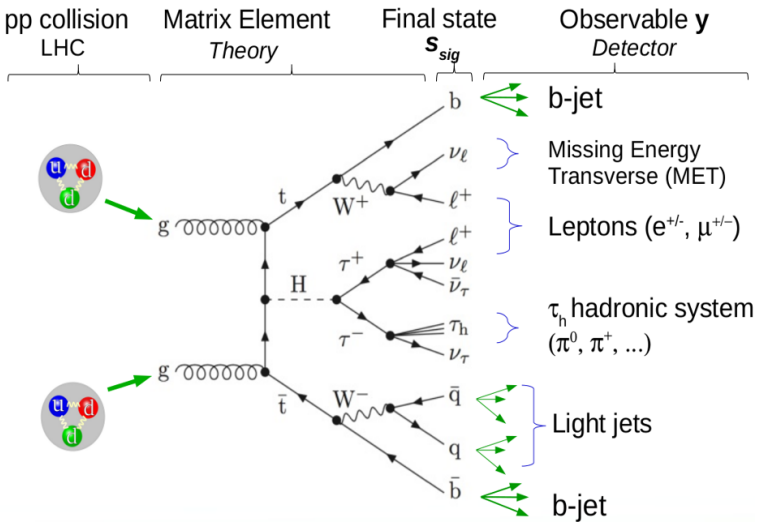


**Figure 2.** The final state considered in our analysis: $b\bar{b}$, $q\bar{q}$ pairs, $\tau_h$, 2 same sign leptons and 3 undetectable neutrinos $\nu's$. The $\tau^-$ decays in a hadronic system referred to as $\tau_h$ is constituted by pions particles $\pi^{\pm}(\pi^0)^n, \pi^{\pm}\pi^{\mp}\pi^{\pm} \ldots$ together with a $\nu_\tau$ neutrino. The CMS reconstruction software, CMSSW, reconstructs the jets, identifies the $b$ and light quarks, the hadronic $\tau_h$, leptons and the Missing Transverse Energy (MET) collecting the transverse energy of undetectable neutrinos.

### 2.2 Background processes

The same final state can be obtained from several background processes and only the dominant ones are considered for the matrix element computations. For the irreducible $ttZ$ background where the $H$ boson of the signal is simply replaced by a $Z$ boson, the MEM computation is the same as with the signal, except for the matrix element component of the integral. For the reducible backgrounds, $(ttZ, Z \rightarrow ll)$ and $t\bar{t}$, where one of the objects is mis-identified, the matrix element and the Transfer Functions as well as the integration strategies are modified.

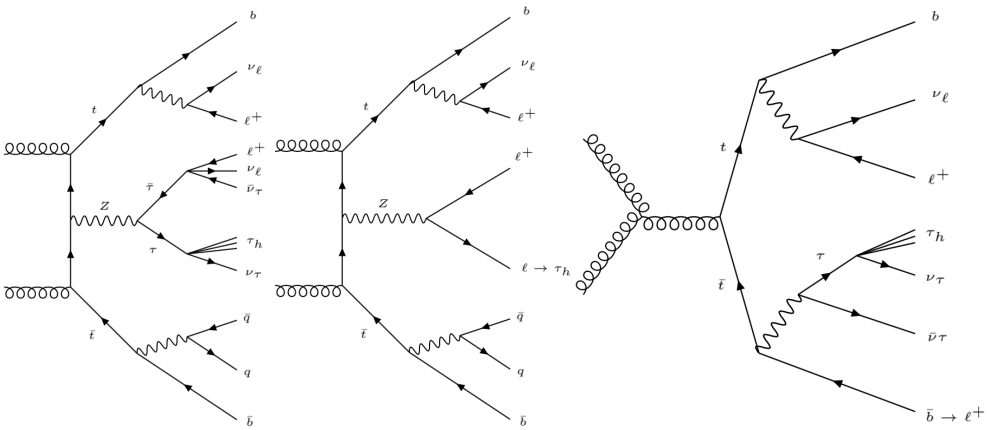


**Figure 3.** Main *background* processes leading to the observation of the same final state.
(*Left*) $ttZ, Z \to \tau\tau$, the $Z$ boson replaces the Higgs boson in this diagram.
(*Middle*) $ttZ, Z \to ll$, with one of the leptons misidentifed, in this case the $Z$ boson decays in two leptons, and one them is misidentified by the reconstruction software and associated with an hadronic tau particle $\tau_h$.
(*Right*) $t\bar{t}$, with an additional lepton produced in the decay of $B$ hadrons. Additional light jets can be produced due to some QCD radiation, leading to the final state observation, but are not taken into account in the matrix element.

### 2.3 Permutations and integration multiplicity

Up to now, we have seen that we have to compute 4 integrals per event: 1 to compute the probability that belongs to the signal class and 3 to belong each of the considered background classes. However, we have to consider that an observed b-jet can be assigned to the $b$ quark or the $\bar{b}$ of the $ttH$ process. Similarly, several different configurations give rise to two same-sign leptons: their observations in the detector cannot be assigned uniquely to the one coming from the $\tau^+$ decay and the other coming from the $W^+$ decay. As a result, we have to take into account the 4 lepton-bottom permutations for each of the 4 signal/background probabilities to compute. Besides, the possibility that a jet falls out of the detector acceptance is also taken into account and requires a specific integration.

There is another multiplicative factor that increases the number of integrals to compute. It occurs for events in which only one quark has been identified, leading to a missing $q\bar{q}$ pair in the final state. In this special case, integrals are computed by assigning a light-jet from the light jet list identified in the event to the missing quark of the final state. For this kind of event, $4 \times 4 \times n_{light-jets}$ integrals, where $n_{light-jets}$ is the number of light-jets observed in the event, must be considered.

In practice, to reduce the number of integrals to compute, a filter is applied. It selects only the lepton-bottom permutations and the light-jet permutations which are physically realistic, based on invariant mass criteria. Concerning the integration space, the integral dimensions vary from 3 to 5 in the case of the $q\bar{q}$ pair is identified, and 5 to 7 if not. More details on the integration procedure can be found in [8].

## 3 MEM code implementation

Because the MEM is very time consuming, we started our developments with High Perfomance Computing (HPC) in mind, and to avoid large waiting times or elapsed time to obtain the results, we started to build a parallel release MPI-MEM with the MPI library[15]. The common way to compute high-dimension integrals is to use an adaptive Monte-Carlo algorithm, in this case the commonly-known VEGAS[12], implemented in the GSL[20]. Running the code gave already reasonable waiting time. To analyze a data set of around 2500 events (around 30k integrals to compute), it requires typically 14 hours of computation deployed on 96 cores (6 nodes with 2 Intel Xeon E5-2640 processors, 16 physical cores rates at 2.6 Ghz), thanks to MPI. This means that the analysis would take 55 days on one core.

Once this reference implementation was achieved, rather than optimizing the MPI-MEM code we focused our efforts on GPU's platform, with the idea to gather as many GPU devices as possible to have at our disposal a huge computing power to run the MEM analysis[9]. We benefited from a first experience that had been implemented within the $H \to \tau\tau$ channel analysis [18].

### 3.1 OpenCL/CUDA implementation

Considering the importance of the portability of our developments, we selected the OpenCL standard[14] to handle the GPU's. The parts which have been selected to run on GPU's are the most computationally intensive: the integral evaluations. To achieve this goal we transformed the integration parts from the initial MPI-MEM C++ implementation to the C language to build the GPU kernels. In addition, to minimize memory traffic between the node memory and the GPU memory, all the integrals of a same event are computed in the same GPU. The main components used in the GPU-MPI-MEM implementation are:

**LHAPDF library:** this library takes care of the computation of the PDF function seen in subsection 2.1. We translated it from Fortran to C and we numerically validated it before implementing it in the integration part.

**ROOT utilities:** a small subset of ROOT framework[16] (such as vector, geometric, Lorentz arithmetic) has been rewritten in the C-kernel language. In addition, basic operations in the ROOT suite, used for instance in the Transfer Functions, have been transformed to C-kernel functions.

**MadGraph:** the Matrix Element (ME) $|M_i(\boldsymbol{x})|^2$ in the expression (1) is given by the MadGraph5_aMC@NLO framework[17]. To be efficient on CPU, MadGraph generates C++ code to compute the ME of a given process. We have extended this generative part of MadGraph to provide C-sources to be directly integrated into our GPU kernels.

**MC Integration:** the integration computation is the part which is easily parallelized (or vectorized) thanks to the numerous and independent function evaluations [13]. Nevertheless, the adaptive part of the VEGAS is a little bit more tricky with the GSL implementation.

Moreover, keeping in mind the uncertain future of OpenCL for NVidia devices, we developed a bridge between OpenCL 1.2 and CUDA able to manage CPU devices as well as NVidia GPU, AMD GPU, FPGA devices. This bridge is equivalent to an OpenCL implementation with CUDA primitives, mapping the OpenCL paradigm to the CUDA one, taking into account all mandatory features to run efficiently on NVidia devices: events, asynchronous mechanisms, ... This development takes advantage of the latest releases of efficient CUDA compilers, which

is generally not the case with OpenCL provided by NVidia. The portability has been tested on pre-exascale architectures: IBM OpenPOWER processor and NVidia P100 GPU's[11], especially to validate the bridge OpenCL / CUDA.

### 3.2 Performance

For this performance analysis, we used a production platform which offers an important number of GPU's for our application : the National Computing Center CC-IN2P3 at Lyon. Before moving to finer optimizations, it was important to understand the global behavior of the application and, to focus on two points first:

- How does the application scale in function of the GPU number? The application uses several asynchronous mechanisms, one at the MPI level (because the computing time varies a lot between two events), and the other at the OpenCL level which distributes the integral computation to the different devices attached to the node. Consequently, we want to make sure that the concurent computations are not blocking each other.
- The acceleration ratio can be obtained. In other terms, what is the gain comparing the GPU's MEM application with our MPI/CPU version? Can we enhance the performance of the GPU-MPI-MEM code?

Each node of our GPU platform has 2 Intel Xeon E5-2640 processors, providing 2 x 8 physical cores rated at 2.6 GHz. In addition, each node is equipped with 4 Kepler GK-210 GPU's on 2 NVidia K80 boards. The applications are compiled with gcc 5.3 compiler with the -O 3 optimization option and, for the GPU's kernels, with nvcc compiler from the CUDA 8.0 software environment.

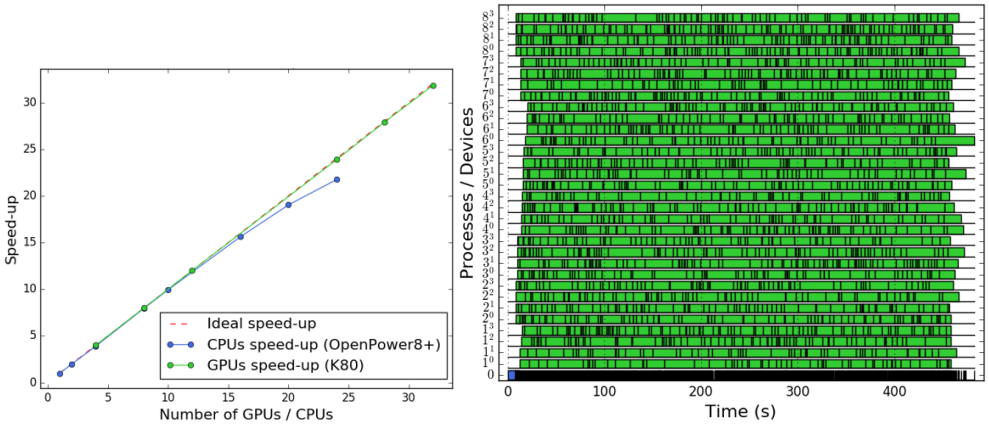


**Figure 4.** (*Left*) Speed-up of the application on a single IBM OpenPower8+ node with 20 physical cores (in blue) and speed-up on 8 nodes with 4 NVidia K80 GPU's each (in green); (*Right*) Work distribution in time of our dataset (2395 events), in this case, to the 32 available GPU's. The label $P^d$ identifies a GPU, with $P$ the MPI process number and $d$ the GPU local identifier in the node. There is one MPI process per node, except in the case of node 0 which is the *master* MPI process. It takes in charge the I/O aspects and the event distribution. One box in one timeline represents the computation of all the integrals of a same event.

For all the performance analyses we add a time log at the MPI level to profile the moments when the event processing starts and finishes. The figure 4 (left) shows a excellent speed-up

(the ratio between the execution time of one code instance *ie* one MPI process, divided by the execution time of N instances *ie* N MPI processes) up to 8 nodes i.e. 32 GPU's. This can be explained by the very small time overhead of the communications and the event scheduling, compared to the average time to process an event. The figure 4 (right) shows that there is no delay between two event processing and that all the dataset is performed in impressive computing time: less than 500 seconds. Because the number of integrals per event to compute, as well as their kind (dimensionality) vary, the processing time for a single event fluctuates in a large range of values.

This remarkable gain is not only attributable to the GPU's. A 3-4 factor is commonly conceded for the gain between an HPC node and a GPU device but also the kernel recoding in "flat" C avoiding the hidden overheads of the method calls in cascade ways. The other point is that the OpenCL or CUDA programming paradigm, based on data-parallel model, naturally vectorizes and is easily parallelized by the compilers.

## 4 Conclusion

The MEM is a powerful classification tool used for signal extraction in the *ttH* analysis. It however requires a significant amount of computations. Its deployment on GPU's platform has remarkably accelerated the computing time. We demonstrate in this paper that there is a high added value in using simultaneously a great number of GPU's for our *ttH* analysis: 55 days with one core, 14 hours with 96 cores and less than 500 seconds on 32 GPU's platform. Thanks to this computing time speed-up, the analyzers can now run the analysis multiple times. More checks can thus be carried out, and more analyses improvements be tested. This improvement however comes with a price: the OpenCL or CUDA programming paradigm forces to write the code or the kernels which are easy to parallelize/vectorize by the compiler, explaining this important gain.

Nevertheless, it is a tedious and cumbersome task to transform C++ code to C kernels. In the future we will focus on this problem avoiding to penalize the extensibility of this implementation to other models. Using the MadGraph code-generation capability could be a way.

Today, the GPU-MPI-MEM version is used to analyze the new data-taken at LHC. In the future with the High-Luminosity LHC upgrade, the computing power required will drastically increase, due to the larger data taking rate and the larger complexity of the events. Our GPU's-based MEM analysis is ready to assimilate this data increase, thus freeing the LHC Computing Grid resources from such heavy computating analyses.

## 5 Acknowledgements

This work has been funded by the P2IO LabEx (ANR-10-LABX-0038) in the framework "Investissements d'Avenir" (ANR-11-IDEX-0003-01) managed by the French National Research Agency (ANR).

## References

[1] Sirunyan A. M. *et al.* (CMS Collaboration), *Observation of $t\bar{t}H$ Production*, Phys. Rev. Lett. **120**, 231801 (2018)

[2] Aad G. *et al.* (ATLAS Collaboration), *Observation of a new particle in the search of the Standard Model Higgs boson with the ATLAS detector at the LHC*, Phys. Lett. B **716**, 1 (2012)

[3] Chatrchyan S. *et al.* (CMS Collaboration), *Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC*, Phys. Lett B **716** 30 (2012)

[4] CMS Collaboration, *Evidence for the Higgs boson decay to a bottom quark-antiquark pair*, Phys. Lett. B **780** 501 (2018)

[5] CMS Collaboration, *Observation of the Higgs boson decay to a pair of tau leptons*, Phys. Lett. B **779** 283 (2017)

[6] CMS Collaboration, *An event candidate for the production of a top quark and top anti-quark pair in conjunction with a Higgs Boson in the CMS detector*, https://cds.cern.ch/record/2621446, (2018)

[7] Abazov V.M. *et al.* (DØ Collaboration), *A precision measurement of the mass of the top quark*, Nature **429**, 638–642 (2004)

[8] T. Strebler, thesis *Probing the Higgs coupling to the top quark at the LHC in the CMS experiment*, chap. 4 & 5 (2017)

[9] Schouten D., DeAbreu A. and Stelzer B. *Accelerated Matrix Element Method with Parallel Computing*, Computer Physics Communications, Volume 192, pages 54-59 (2015)

[10] Hagiwara K., Kanzaki J., Li Q., Okamura N. and Stelzer T. *Fast computation of MadGraph amplitudes on graphics processing unit (GPU)*, European Physical Journal **73** 2608 (2013)

[11] Hautreux G. *et al.*, *Pre-exascale Architectures: OpenPOWER Performance and Usability Assessment for French Scientific Community*, ISC International Workshops 2017: 309-324 (2017)

[12] Lepage G. P., *A New Algorithm for Adaptive Multidimensional Integration*, J.Comput.Phys. **27**:192 (1978)

[13] Kanzaki J., *Monte Carlo integration on GPU*, European Physical Journal C, **71**:1559 (2011)

[14] Stone J E, Gohara D, and Shi G. *Opencl: A parallel programming standard for heterogeneous computing systems IEEE, Computing in Science & Engeneering*, **12** (3), p. 66-73 (2010).

[15] Gropp W., Lusk E. and Skjellum A., *Using MPI: portable parallel programming with the message-passing interface*, MIT Press Cambridge MA, USA (1994)

[16] Brun R. and Rademakers F., *ROOT - An Object Oriented Data Analysis Framework*, Nucl. Inst. & Meth. in Phys. Res. A, **389**:1-2 p. 81-86 (1997)

[17] Alwall J., Herquet M., Maltoni F., Mattelaer O. and Stelzer T., *MadGraph 5 : Going Beyond*, JHEP 1106 128. arXiv:1106.0522, (2011)

[18] Grasseau G. *et al.*, *Matrix element method for high performance computing platforms*, CHEP 2015, J. Phys.: Conf. Ser. 664, p. 92009 (2015)

[19] Whalley M. R., Bourilkov D. and Group R. C., *The Les Houches accord PDFs (LHAPDF) and Lhaglue*, arXiv:hep-ph/0508110 (2005)

[20] Galassi M. *et al.*, *GNU Scientific Library Reference Manual*, Network Theory Ltd third edition (2009)