Università degli Studi di Milano

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea Magistrale in Fisica

Performing PDF fits on GPU: a preliminary search for the best solution

Relatore: Dott. Stefano Carrazza

Elaborato di:
Emilio Villa
Matr. 903007

Anno Accademico 2018-2019

"When you ask what are electrons and protons, I ought to answer that this question is not a profitable one to ask and does not really have a meaning. The important thing about electrons and protons is not what they are but how they behave, how they move. I can describe the situation by comparing it to the game of chess. In chess, we have various chessmen, kings, knights, pawns and so on. If you ask what chessman is, the answer would be that it is a piece of wood, or a piece of ivory, or perhaps just a sign written on paper, or anything whatever. It does not matter. Each chessman has a characteristic way of moving and this is all that matters about it. The whole game of chess follows from this way of moving the various chessmen."

Paul A.M. Dirac

Contents

Chapter 1

QCD: an introduction

1.1 QCD Lagrangian

QCD is the theory of strong interactions between quarks and gluons, the elementary constituents of hadrons. QCD is a non-abelian gauge field theory with gauge group SU(3) [15]. Gluons are massless and quarks have fractional electric charge, either 2/3 or -1/3. QCD Lagrangian is a Yang-Mills Lagrangian invariant under local SU(3) transformations:

$$\mathcal{L} = -\frac{1}{4} F^a_{\mu\nu} F^{\mu\nu}_a + \sum_f \bar{\psi_f}^i \left(i\gamma^\mu D_{\mu,ij} - m_f \delta_{ij} \right) \psi^j_f \tag{1.1}$$

where ψ^i_f are the quark Dirac fields with the indexes i and f running respectively over colors and flavors and $D_{\mu,ij} = \delta_{ij}\partial_\mu - igt^a_{ij}A^a_\mu$ is the covariant derivative, where in turn standard notation is used to refer to the gluon field A^a_μ, to the generators of the symmetry group t^a_{ij} and to the bare coupling constant g. $F^a_{\mu\nu}$ is the gluon field strength and can be written in terms of the gluon field A^a_μ and the structure constants of the group f_{abc} as $F^a_{\mu\nu} = \partial_\mu A^a_\nu - \partial_\nu A^a_\mu + g f_{abc} A^b_\mu A^c_\nu$. Color indexes take respectively 3 or 8 values if they are attached to objects in the fundamental or in the adjoint representation.

Some features of the theory can directly be witnessed from the Lagrangian. First, as already mentioned, there is no gluon mass term which is forbidden by the gauge invariance of the theory. Secondly, the theory is *flavor blind*, i.e. there is no coupling between different flavor of quark. Finally, writing explicitly the interaction terms, one can easily see that the gluons self-interact and thus carry color charge.

QCD has its roots in the *quark model* introduced in 1963 by Gell-Mann and Zweig [18, 33] to explain the spectrum of strongly interacting particles in terms of elementary constituents: mesons and baryons were expected to be respectively bound states of a quark anti-quark pair and of three quarks. The existence of six different type (now called flavors) of quarks with fractional electric charge must be conjectured to take into account all the known hadrons [26]. The existence of color was also assumed in order to fulfill the requirements of Fermi-Dirac statistics with regards to the baryonic wavefunctions. Quarks were assigned to the fundamental representation of a SU(3) color global symmetry and hadrons were supposed to be singlets of color, in agreement with the experimental evidence. Such property of hadrons is known as *color confinement* and has not yet been proven analytically.

After studies on non-abelian gauge theories and the discovery of the property of asymptotic

freedom, QCD was eventually constructed as a non-abelian gauge theory and the color group was identified to be the gauge group of the theory.

1.2 The QCD coupling constant

The strong coupling, labeled as α_s, is defined as:

$$\alpha_s = \frac{g^2}{4\pi} \tag{1.2}$$

g being the coupling of eq. (1.1).

As long as the classical theory is taken into account, α_s is a constant, but when the quantum theory is considered, a dependence on the renormalization energy scale is assigned to the coupling to renormalize UV divergences in loop diagrams [26, 15]. The evolution of α_s with the energy is given by Callan-Symanzik equation:

$$\mu^2 \frac{d}{d\mu^2} \alpha_s \left(\mu^2\right) = \beta \left(\alpha_s \left(\mu^2\right)\right) \tag{1.3}$$

where $\beta\left(\alpha_s\right)$ is the so called β-function, which can be expanded in series of α_s:

$$\beta\left(\alpha_s\right) = -\alpha_s^2(\beta_0 + \alpha_s\beta_1 + \alpha_s^2\beta_2 + ...) \tag{1.4}$$

In 1973 Gross and Wilczek and independently Politzer [20, 28] evaluated the *beta*-function at leading order and found out that it was negative. More specifically, the expression of the first coefficient is:

$$\beta_0 = \frac{33 - 2n_f}{12\pi} \tag{1.5}$$

where n_f is the number of flavors light at the scale μ^2. Since for QCD $n_f < 17$ at any scale, the β-function is negative, implying that the coupling of the theory becomes less stronger as the energy (distance) scale of the interaction increases (decreases). This is the already mentioned property known as *asymptotic freedom*.

The running of the coupling constant can be given in terms of the value of the coupling itself at some arbitrary fixed scale Q_0^2:

$$\alpha_s\left(\mu^2\right) = \alpha_s\left(Q_0^2\right)\left(1 - \alpha_s\left(Q_0^2\right)\beta_0 \log\frac{\mu^2}{Q_0^2} + o\left(\alpha_s^2\right)\right) . \tag{1.6}$$

Measuring the value of the coupling constant at a certain scale is then enough to determine it at any other scale, the two values being related by the renormalization equation.

At leading order, the solution to the Callan-Symanzik equation can be written also in term of the *fundamental QCD scale* $\Lambda \sim 200$ MeV, that is a rough approximation of the scale at which the perturbative description of the theory breaks down[1]:

$$\alpha_s\left(\mu^2\right) = \frac{1}{\beta_0 \log\frac{\mu^2}{\Lambda^2}} . \tag{1.7}$$

The scaling of the strong coupling constant has been measured in the range of around $1 - 200$ GeV and found to agree with the experimental data [25].

[1] More specifically, Λ is the scale at which the coupling diverges to infinity.

Asymptotic freedom is what made QCD so hard to be understood during the earliest stages of its discovery. Indeed, quark confinement implied that free quarks could not be observed directly but at the same time experimental results of proton-proton scattering and deep inelastic scattering (DIS) experiments suggested the idea that hadrons were made up of point-like non-interacting particles. Such constituents, initially referred to as *partons*, were eventually identified with the quarks and gluons of QCD.

The famous parton model proposed by Feynman and Bjorken [16, 7] will be discussed in the next section.

1.3 The parton model

Historically, the first evidence that at high energy the strong force has an unexpected weak behavior came from proton-proton scattering experiments carried out in the late 1950s [22, 26]. At high energy of about 10 GeV, bunches of pions were produced in the collision, but mostly with momenta collinear to the collision axis, whilst the production of pions with large transverse momenta was exponentially suppressed in the value of the transverse momentum itself. To explain this "soft" scattering and the phase space distribution of the final pions, hadrons were thought to be ensembles of weakly bounded elementary constituents. In this theoretical framework, the two colliding protons would be torn into small pieces with momenta roughly collinear with the momenta of the original protons and would eventually build up into new hadrons moving along the same axis.

In the late 1960s, the SLAC-MIT collaboration performed deep inelastic scattering experiments to challenge this hypotheses about the inner structure of the hadrons [22, 26]. A 20 GeV electron beam was scattered from a hydrogen target and the scattering rate was measured for large deflection angles. The available information suggested that hadrons would yield primarily distributions of scattered electrons reflecting diffuse charge and magnetic moment distributions. The large momentum transfer, delivered by electromagnetic interaction, could be computed measuring the momentum of the scattered electron.

Against all the odds, a significant rate for hard scattering of electrons from protons was observed. The same outcome would have been observed if the proton were a point-like particle behaving simply according to QED. However, the major of the hard scattering had as final state not the proton but a jet of hadrons.

To account for the experimental evidence, Bjorken and Feynman introduced the so called parton model [16, 7]: hadrons were composed of almost free elementary constituents, including fermions carrying electric charge, i.e. the quarks, and possibly other neutral particles that bind them together. At high energy, the masses of the partons and the hadrons could be neglected compared to the scale Q of the hard scattering process.

That was enough to explain the experimental outcomes. Indeed, even if the quarks could not exchange large momentum interacting strongly, they could still do it by means of electromagnetic interaction. The electrons were then able to hard scatter from quarks and kick them out from the proton. The observed outgoing hadron jets were caused by subsequent hadronization processes taking place after the collision.

The main physical concepts supporting the parton model may be summarized as follow. First of all, the hadron consists of a set of partons in some virtual state with lifetime τ in the rest frame of the hadron[2] [9]. In the center-of-mass frame the hadron is contracted and the lifetime of its

[2] It is assumed that there is an effective lower bound, so that the hadron is made up primarily of virtual states of nonzero lifetime in its own rest frame

constituents are dilated according to Lorentz transformations. The more the center-of-mass energy increases, the less time the electron takes to cross the hadron.

From the electron perspective then, during the collision the hadron looks like a sea of frozen point-like particles. To exchange a large momentum q^μ with a parton, according to the uncertainty principle, the electron must come as close to it as $o\left(1/Q\right)$ with $Q^2 = -q^2$ the scale of the hard scattering. The probability for the scattering to occur is then suppressed by the geometrical factor $\frac{1/Q^2}{\pi R_0^2}$, with R_0 "radius" of the hadron, assuming that the partons are randomly distributed inside the hadron. A fundamental consequence of this argument is that the cross section for the DIS of a lepton from an hadron can be written as the probability of finding a given parton inside the hadron with a certain momentum fraction times the partonic cross section for the scattering of the lepton from the parton itself. Writing it schematically, one has:

$$\sigma_{l+h\to l'+X}\left(p,q\right) \sim \sum_i \int_0^1 dx\ \hat{\sigma}_{l,i\to l'+X}(p,xq) f_i\left(x\right) \tag{1.8}$$

where p and q are respectively the momenta of the incoming lepton l and hadron h, the sum runs over the partons i inside the hadron, $f_i\left(x\right)$ is the parton distribution function (PDF) giving the probability of finding the parton i in the hadron h with fractional momentum xq and finally $\hat{\sigma}_{l,i\to l'+X}\left(p,xq\right)$ is the partonic cross section where, as in the total cross section, $l^{'}$ and X are the outgoing lepton and the final arbitrary hadronic state.

In the next paragraph, infinities arising in QCD are discussed along with the introduction of the factorization scale and the famous DGLAP or Altarelli-Parisi equations.

1.4 Factorization scale and DGLAP equations

When it comes to divergences, higher order corrections QCD presents different type of infinities, arising both in loop diagrams (virtual corrections) and when new partons are emitted (real corrections).

Both UV and IR singularities affect loop diagrams. The first ones are dealt with renormalization procedures, while the second ones cancel out when combined with real corrections. This result holds for all the Standard Model and it is known as Kinoshita-Lee-Nauenberg theorem [23, 24].

Apart from these kinds of divergences, real emission diagrams are affected also by the so called collinear singularities, namely emissions of partons with transverse momenta approaching zero. These infinities are treated in a renormalization procedure fashion by reabsorbing them into a redefinition of a bare quantity. In this case, the bare quantities are the PDFs that now depend also on an energy scale (like α_s in the renormalization procedure), called *factorization scale* and are finite quantities. This result, known as *collinear factorization theorem* [15], is far from trivial since the divergent part should factorize in a way that is process independent. In other words, writing the renormalized (physical) parton density f in terms of the bare one $f^{(o)}$, one has:

$$f\left(z,\mu_F^2\right) = \Gamma\left(\alpha_s\left(Q^2\right),\mu_F^2/Q^2\right) f^{(o)}\left(z\right) \tag{1.9}$$

where $\Gamma\left(\alpha_s\left(Q^2\right),Q^2/\mu_F^2\right)$ must be a universal factor independent of the hard scattering process. In the previous expression, z is the fraction of the hadron momentum, μ_F is the factorization energy scale and Γ is the factor relating the bare with the physical PDF and depends on $\alpha_s\left(Q^2\right)$ and the ratio between the two energy scales.

Before moving to the DGLAP equations, it is worth noticing that there are different ways in which the divergences can be absorbed into the redefinition of the PDFs. To make things clear, let

us consider again the expression of the DIS cross section:

$$\sigma_{l+h\to l'+X}\left(p,q\right) \sim \sum_i \int_0^1 dx\ \hat{\sigma}_{l,i\to l'+X}(p,xq,\mu_F^2) f_i\left(x,\mu_F^2\right) \tag{1.10}$$

where now both the partonic cross section and the PDFs show a dependence on the μ_F. Once the factorization scale is introduced, there are both finite and infinite contributions depending on μ_F^2. The divergent terms must be reabsorbed in the PDFs, but the finite ones can be put arbitrarily in the PDFs or in the partonic part of the computation. The different ways of dealing with finite terms are called *factorization schemes* [9]. The most common is the modified minimal subtraction scheme, referred to as $\overline{\text{MS}}$, where the finite counterterms are kept in the partonic part.

Now, another feature of the expression above, is that the l.h.s. of the equation does not depend on the factorization scale, even if the r.h.s. does as it is explicitly written. However, this is perfectly reasonable since $\sigma_{l+h\to l'+X}$ is a physical observable and so it cannot depend on an arbitrary energy scale introduced in order to cure divergences. This simple consideration is fundamental in the derivation both of the Callan-Symanzik equations (for renormalization) and of the DGLAP or the Altarelli-Parisi equations (for factorization).
Taking the logarithmic derivative of the hadron structure functions with respect to the factorization scale, one obtains the DGLAP equations [2, 14, 19]:

$$\mu_F \frac{d}{d\mu_F} f_i\left(x,\mu_F^2\right) = \sum_j P_{ij}\left(x,\alpha_s\left(\mu_F^2\right)\right) \otimes f_j\left(x,\mu_F^2\right) \tag{1.11}$$

where $P_{ij}\left(x,\alpha_s\left(\mu_F^2\right)\right)$ are the Altarelli-Parisi splitting functions and the $\otimes$ symbol is the convolution operator, defined by:

$$f\left(x\right) \otimes g\left(x\right) \equiv \int_x^1 \frac{dy}{y} f\left(\frac{x}{y}\right) g\left(y\right)\ . \tag{1.12}$$

The Altarelli-Parisi splitting functions are known up to NNLO.

Chapter 2

Parton Distributions Fits

2.1 Introduction

A precise knowledge of Parton Distribution Functions is essential in order to make predictions for the Standard Model and beyond the Standard Model processes at hadron colliders [27]. However, it is not possible to determine analytically from first principles the functional form of PDFs since they belong to the non-perturbative regime. For this reasons, the shapes of PDFs are determined by a fit to data from experimental observables in various processes, using the DGLAP evolution equation to evolve PDFs from the fitting scale to the scale at which data are available.

The determination of PDFs is carried out by several groups, namely:

- MSTW
- CT10
- NNPDF
- HERAPDF
- AB(K)M
- GJR

Roughly speaking, the main differences between the groups rely on the choice of PDFs parameterization and on the method adopted to treat uncertainties, i.e. Monte Carlo method or the Hessian approach. For more details on each research group, see [27, 1].

In the following, a brief history of the development and improvement of PDFs determination will be given, along with an introduction to the methodology adopted by the Neural-Net Collaboration (NNPDF).

2.2 Historical outlines

PDFs determination has developed during the years in conjunction with the improvements in the phenomenological and theoretical understanding of the theory of strong interactions[1]. At a first

[1]This paragraph follows step by step the introduction to PDFs determination given in reference [3]

stage, PDFs were determined to show the compatibility between the data and the partonic interpretation of hard processes. They were determined on the basis of sum rules and of the first available experimental data. Although PDFs were semi-quantitative, they could be used to compare the observed scaling violations with those predicted by perturbative QCD, testing for the first time the theory of strong interaction.

Over the years, the accuracy of the data and the confidence in perturbative QCD improved, leading to the extraction of the gluon distribution function and to the performance of first parton set based on global fits. Even if next-to-leading (NLO) order evolution tools were available, these analysis were carried out at leading order.

When QCD started to be considered as precision physics and an integral part of the Standard Model, thanks to new data coming from high-precision deep-inelastic scattering an hadron collider experiments, a new approach to PDFs determination, able to account for next to leading order theory, were required. Moreover, PDFs should be determined using a wide and varied set of data in order to reduce at minimum the theoretical bias on PDFs functional form at initial scale.

Next-to-leading order parton sets evolved into standard analysis tools. Over time, the accuracy of PDFs sets improved till becoming comparable to that of NLO QCD computations, making PDFs suitable for the determination of most hard processes at collider energies. As already mentioned in the previous paragraph (2.1), these PDFs sets were determined by a fit to data: a parametrization of PDFs were assumed, based on the functional form $f(x) \sim x^{\alpha}(1-x)^{\beta}$, and the parameters were tuned so that computed observables fit the experimental data.

The employment of PDFs for high precision physics gave rise to the problem of the determination of reliable uncertainties for PDFs. Indeed, this task is far from trivial and presents subtle difficulties, the most obvious of which is the appropriate treatment of the correlated uncertainties of the data. At the beginning, when the problem was not completely understood, PDFs uncertainties were determined comparing the results obtained with different PDFs sets. However, such approach was unsatisfactory because PDFs sets determination were likely to share the same sources of systematic errors. Then, some first determinations of PDFs with uncertainties were obtained by only fitting to restricted data sets, but retaining all the information on the correlated uncertainties in the underlying data, and propagating it through the fitting procedure.

Despite PDFs with uncertainties are now available, benchmark comparisons performed between different PDFs sets have shown how difficult it is to give a statistical interpretation to the uncertainties, given that they are considerably linked to theoretical or phenomenological expectations. Indeed, uncertainty bands for PDFs based on global fits are given on the basis of a tolerance evaluated studying the compatibility of the data and the underlying theory [29], which has an effect comparable to multiplying experimental uncertainties by a factor ranging from four to six.

A completely new approach aimed at determining objectively both the value and the uncertainty of a function from a discrete set of many independent experimental measurements, was proposed for the first time in [17] and improved over the years by the NNPDF collaboration. Such approach will be describe in the next paragraph.

2.3 NNPDF approach

The NNPDF collaboration approach to PDFs determination is based on two main features: the generation of an ensemble of Monte Carlo replicas of the experimental data and the use of neural networks as "unbiased" interpolating functions. The procedure can be summarized as follow: firstly N_{rep} replicas of the N_{data} data are generated distributed accordingly to a N_{data}-dimensional multigaussian distribution, then N_{rep} set of N_{pdf} neural networks (i.e. the PDFs to be determined) are

trained over the data [13]. The result of the process consist in a final ensemble of best fit PDFs (N_{rep} best fit PDF for each flavor, i.e. for each one of the N_{pdf} PDFs). In what follows, the main steps of the procedure are treated in more details.

2.3.1 Monte Carlo generation of pseudodata

The central value of each Monte Carlo "pseudodata", equals the experimental one, and the same applies to uncertainty and covariance. Of course this result is perfectly achieved only in the limit of infinite replicas. However it can be shown that it stands already for $N_{rep} = 100$ or $N_{rep} = 1000$ [4]. So, the statistics of the data can be reproduced to arbitrary accuracy by generating the proper number of Monte Carlo replicas (and an a posteriori check can be performed evaluating averages, variance and covariance of the pseudodata and comparing them with the data counterparts).

For each data point $D_i^{(exp)}$, $k = 1, ..., N_{rep}$ artificial points $D_i^{(art)(k)}$ are generated according to:

$$D_i^{(art)(k)} = \left(1 + r_N^{(k)}\sigma_N\right)\left(D_i^{(exp)} + \sum_{p=1}^{N_{sys}} r_p^{(k)}\sigma_{i,p} + r_i^{(k)}\sigma_{i,s}\right) \tag{2.1}$$

where $\sigma_{i,p}$ are the N_{sys} correlated systematic errors, σ_N is the total normalization uncertainty, $\sigma_{i,s}$ is the statistical uncertainty and, for each independent error source, independent univariate gaussian random numbers $r^{(k)}$ are used. The covariance matrix is given by:

$$cov_{ij}^{(k)} = \left(\sum_{p=1}^{N_{sys}} \overline{\sigma}_{i,p}^{(k)}\overline{\sigma}_{j,p}^{(k)}\right) + \delta_{ij}\overline{\sigma}_{i,s}^{(k)2} \tag{2.2}$$

where $\overline{\sigma}_{i,p}^{(k)}$ and $\overline{\sigma}_{i,s}^{(k)}$ are defined as:

$$\overline{\sigma}_{i,s}^{(k)} = \left(1 + r_N^{(k)}\sigma_N\right)\sigma_{i,s} \tag{2.3}$$

$$\overline{\sigma}_{i,p}^{(k)} = \left(1 + r_N^{(k)}\sigma_N\right)\sigma_{i,s} \qquad p = 1, ..., N_{sys}\ . \tag{2.4}$$

Normalization error are included in the covariance matrix by rescaling the statistical and systematic errors as shown in the previous equation (2.3). Indeed, it has been shown in [12] that including normalization error on the same footing of the other sources of uncertainties would bias the fit.

2.3.2 Neural network parametrization

In the second stage of the procedure, a set of PDFs is constructed from each replica of the data, parametrizing each PDF at a given scale with a neural network. In short, neural networks are non linear maps between input $\xi_i^{(1)}$ and output $\xi_i^{(L)}$ variables (see figure (2.1)). Like orthogonal polynomials, in the limit of infinite size, they can reproduce any continuous function. However, upon truncation, neural networks and polynomials behave in a different way. Indeed, while the second ones introduce a specific bias on the form of the fitted function, this effect is avoided with neural networks. In particular, it is possible to show that the fit is stable under changes in the size of the neural network. This feature do not belong to standard fits, where eventually, increasing the size of the fitting function, no stable fit can be achieved. This characteristic is what makes neural networks a particularly convenient set of unbiased interpolating functions.

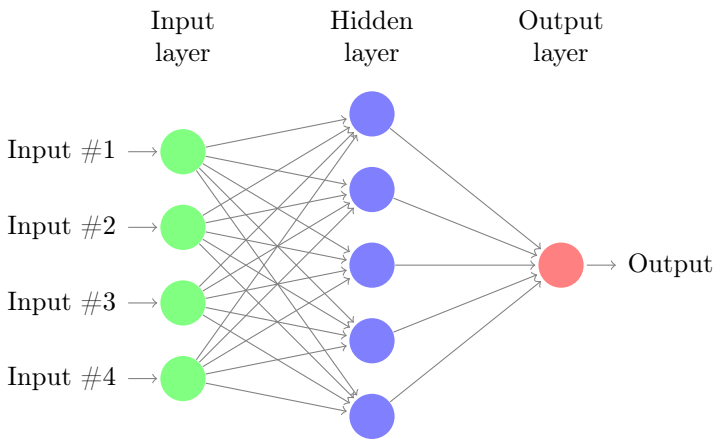

Figure 2.1: a schematic representation of a neural network with one hidden layer. Reprinted from TEXexample.net

The neural networks used in the fitting procedure are multi-layer feed-forward neural networks with architecture 2-5-3-1, where neurons are organized in layers $l = 1, ..., L$ with $j = 1, ..., n_l$ neurons per layer. The first layer provides the input and the neurons in a layer feed forward to the next one till the last one is reached. The last neuron contains the output variables. The output ξ_j^l of each neuron (j-th neuron of the l-th layer) is given by a non-linear activation function $g(x)$:

$$\xi_i^{(l)} = g\left(h_i^{(l)}\right) \ , \quad i = 1, ..., n_l \ , \quad l = 2, ..., L \tag{2.5}$$

where $h_i^{(l)}$ is given by:

$$h_i^{(l)} = \sum_j^{n_l - 1} \omega_{ij}^{(l)} \xi_j^{(l-1)} - \theta_i \tag{2.6}$$

where, in turn, ω_{ij} and θ_i are called *weights* and *thresholds* respectively and are the free parameters that are determined by the fitting procedure. The activation function used for the inner layers is a sigmoid, defined by:

$$g(x) = \frac{1}{1 + exp(-x)} \tag{2.7}$$

while for the last layer, the linear function $g(x) = x$ is adopted. Given the input and the output, the training of the neural networks consists in the determination of the best fit values of the weights and the thresholds.

Despite the fact that PDFs shapes are unknown, they are constrained by sum rules [10] and Regge theory [30], which suggest respectively that for $x \to 1$ the PDFs vanish and instead blow up for $x \to 0$. So, it is convenient to factor out from the neural network this known behavior and fit just the deviation from it. The result is the following functional form for PDFs parametrization at a given scale Q_0:

$$f_i(x, Q_0) = x^{-\alpha_i} (1 - x)^{\beta_i} NN_i(x) \tag{2.8}$$

where the index i refers to the parton kind and $NN_i(x)$ refers to the neural network.

2.3.3 Neural network training

The training of the neural networks, i.e. the fitting procedure, relies upon the minimization of an appropriate figure of merit [13]. The fitting to each replica is performed by minimizing the error function:

$$E^{(k)}[\omega] = \frac{1}{N_{data}} \sum_{i,j=1}^{N_{data}} \left(D_i^{(art)(k)} - D_i^{(net)(k)}\right) \left(cov^{-1}\right)_{ij} \left(D_j^{(art)(k)} - D_j^{(net)(k)}\right) \tag{2.9}$$

where $D_i^{(net)(k)}$ are the prediction of the data evaluated by means of the neural-network PDFs. $E^{(k)}$ defines the quality of the fit for each replica. The quality of the global fit is addressed by the χ^2 computed comparing the experimental data and the values of the observables averaged over the ensemble of best fit PDFs, i.e.:

$$\chi^2 = \frac{1}{N_{data}} \sum_{i,j=1}^{N_{data}} \left(D_i^{(exp)} - \langle D_i^{(net)} \rangle_{rep}\right) \left(\hat{cov}^{-1}\right)_{ij} \left(D_j^{(exp)} - \langle D_j^{(net)} \rangle_{rep}\right) \tag{2.10}$$

where $\hat{cov}_{ij}$ is the covariant matrix which includes normalization uncertainties (see [13] for more details).

The minimization of the error function $E^{(k)}[\omega]$ is carried out using a genetic algorithm. The "state" of the neural network can be identified by a vector, called *weight vector*, whose components are the weights ω_{ij} and thresholds θ_i of the neural networks and that can be written for simplicity as:

$$\left(\omega_1, \omega_2, ..., \omega_{N_{par}}\right) \tag{2.11}$$

where ω_i refers to a weight or a threshold and N_{par} is the total number of weights plus thresholds. A number N_{mut} of copies of the weight vector is produced and then the minimization is achieved in steps (named *generations*), performing for each step two operations. Firstly, the vectors are modified changing one of the components according to:

$$\omega_k \to \omega_k + \eta \left(r - \frac{1}{2}\right) \tag{2.12}$$

where r is a random number between 0 and 1 and η is an hyperparameter called *mutation rate*. This first stage is referred to as *mutation*. Then, the error function is computed for each copy and the original weight vector is replaced by the one with the lowest value of the error function. This second stage is known as *selection*. The procedure is iterated until a suitable convergence criterion is satisfied.

The stopping criterion is related to a famous feature of neural networks fits, known as *overlearning*. The main idea is quite simple and can be summarized as follow: if the minimization procedure is not stopped, the network will learn perfectly the data and the error function will tend to 0; however in such a way, being the data characterized by uncertainties, what is going on is that the network is learning also the noise, i.e. the data and their particular uncertainties. If the same neural network is then used to fit, for instance, another set of measurements of the same data but with different uncertainties (remember that the experimental uncertainites are gaussianly distributed), it will completely fail. In order to avoid this result and maintain the flexibility of the network, an appropriate stopping criterion should be adopted.

The problem is solved by splitting randomly[2] the data in two subsets called *training* and *validation* set. The first set is used to perform the fit while the second one is used to check the ability of

[2]This is not always the case: if a subset of the data, for example, presents common features, it is clear that choosing randomly the training and validation set is not the best solution. For more details see [21].

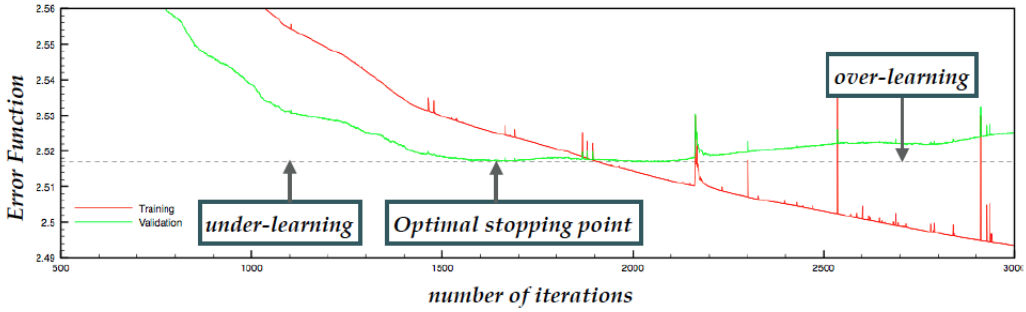

Figure 2.2: the error function computed for the validation set (green line) and for the training set (red line) as a function of the number of iterations in the fitting procedure. While the training set error function decreases indefinitely, the validation set error function reaches a minimum and then sees a rebound. The optimal stopping point corresponds to the minimum. Reprinted from the NNPDF website (http://nnpdf.mi.infn.it/research/cross-validation/).

the neural networks to describe also this subset of data. In practice, the error function is calculated for the two subsets. While the error function for the training set will always decrease, the one for the validation set firstly will decrease, but then, after reaching a minimum, will start to increase as shown in figure (2.2). The change in the slope of the validation set error function corresponds to the moment in which the neural network starts to overlearn. So, the fitting procedure should be stop when the minimum is reached.

Chapter 3

PDFs covolution: a numerical approach

3.1 Introduction

The minimization of the figure of merit $E^{(k)}[\omega]$ during the fitting procedure requires, as shown in eq. (2.9), the neural networks predictions of the observables at each step of the genetic algorithm for each replica of the data. Now, consider, for example, that the data are $pp \to X$ cross section measurements, whose analytic expression is given by a convolution between PDFs and the partonic cross section like in eq. (1.8). The amount of experimental data made available for use in PDFs fits by LHC collaboration is increasing over the years and a standard fit may already include thousands of data points. Moreover, for all these data points, predictions have to be computed thousands of time during the minimization procedure. It is clear that performing PDFs convolution with partonic cross sections for so many times represents a real computational challenge [5]. Indeed, it can be extremely time consuming. For this reason, this task should be optimized at best in order to reduce at minimum such negative effect.

The strategy developed over the years to address this issue consists, generally speaking, in reducing the effective number of calculations needed during the fit, precomputing the partonic hard cross sections in such a way that the standard numerical convolution with any set of PDFs can be reliably approximated by means of interpolation techniques [5]. Quantities of interest are stored in precomputed look up table ready to be used when necessary. The main idea is that it is possible to write the convolution in an especially convenient form, allowing to perform a huge amount of calculations "offline" and not during the fit itself.

This approach has been implemented on one hand in packages such as `APPLgrid` [11] and `FastNLO` [32], which carry out the precomputation of the partoninc hard cross sections, and, on the other hand, in `APFEL` [6], `HOPPET` [31] and `QCDNUM` [8], which instead perform PDFs evolution by analogous interpolation techniques. Later on, `APPLgrid` and `APFEL` have been combined togheter in `APFELgrid` [5], where the convolution has been reduced to the simplest possible form, i.e. a matrix product between a so called *fast kernel* table (FK table) and the PDFs, enhancing considerably the performances.

In what follows, an introduction to `APPLgrid`, `APFEL` and `APFELgrid` is given and a time benchmark of the performance achieved with different methods is presented.

3.2 Fast Kernel method

3.2.1 APPLgrid interpolating technique

The expression of a general cross section $pp \to X$ with a set of PDFs $\{f\}$ is given by:

$$\sigma_{pp\to X} = \sum_s \sum_p \int dx_1 dx_2 \hat{\sigma}^{(p)(s)} \alpha_s^{p+p_{LO}} \left(Q^2\right) F^{(s)} \left(x_1, x_2, Q^2\right) \tag{3.1}$$

where Q^2 is the typical hard scale of the process, the index s sums over the active partonic subprocesses in the calculation, p sums over the perturbative orders used in the expansion, p_{LO} is the leading-order power of α_s for the process and $\sigma^{(p)(s)}$ is the N^pLO contribution to the cross section for the partonic subprocess scattering $(s) \to X$. $F^{(s)}$ represents the subprocess parton density:

$$F^{(s)} \left(x_1, x_2, Q^2\right) = \sum_{i,j} C_{ij}^{(s)} f_i \left(x_1, Q^2\right) f_j \left(x_2, Q^2\right) \tag{3.2}$$

where the $C_{ij}^{(s)}$ matrix enumerates the combinations of PDFs contributing to the s-th subprocess.

Using a set of interpolating functions, spanning Q^2, x_1 and x_2, it is possible to factorize the PDFs and α_s dependence out of the convolution. This approach is implemented in the `APPLgrid` package [5]. If a basis of Lagrange polynomials $\mathcal{I}_\tau \left(Q^2\right)$, $\mathcal{I}_\alpha \left(x_1\right)$ and $\mathcal{I}_\beta \left(x_2\right)$ is considered, the subprocess PDFs and α_s can be written as:

$$\alpha_s^{p+p_{LO}} \left(Q^2\right) F^{(s)} \left(x_1, x_2, Q^2\right) = \sum_{\alpha,\beta,\tau} \alpha_s^{p+p_{LO}} \left(Q_\tau^2\right) F_{\alpha\beta,\tau}^{(s)} \mathcal{I}_\tau \left(Q^2\right) \mathcal{I}_\alpha \left(x_1\right) \mathcal{I}_\beta \left(x_2\right) \tag{3.3}$$

where the shorthand $F_{\alpha\beta,\tau}^{(s)} = F^{(s)} \left(x_\alpha, x_\beta, Q_\tau^2\right)$ is used. Inserting these expressions in the convolution, the hadronic cross section is eventually written in terms of a simple product:

$$\sigma_{pp\to X} = \sum_s \sum_p \sum_{\alpha\beta\tau} \alpha_s^{p+p_{LO}} \left(Q_\tau^2\right) W_{\alpha\beta,\tau}^{(p)(s)} F_{\alpha\beta,\tau}^{(s)} \tag{3.4}$$

where

$$W_{\alpha\beta,\tau}^{(p)(s)} = \mathcal{I}_\tau \left(Q^2\right) \int dx_1 dx_2 \hat{\sigma}^{(p)(s)} \mathcal{I}_\alpha \left(x_1\right) \mathcal{I}_\beta \left(x_2\right) \tag{3.5}$$

consists of the convolution of the hard cross section with the interpolating polynomials. $W_{\alpha\beta,\tau}^{(p)(s)}$ may be stored in a precomputed look-up table, making the final expression for the cross section a considerably simpler task to perform inside a fit than the direct evaluation of the double convolution (3.1).

3.2.2 APFELgrid interpolating technique

An even better results can be achieved performing PDFs evolution via analogous interpolating techniques and then combining the two results. In `APFEL`, for instance, PDFs at a general energy scale Q_τ are expressed as a product between PDFs at some initial fitting scale Q_0 and an *evolution* operator obtained solving the DGLAP equations [5]:

$$f_i \left(x_\alpha, Q_\tau^2\right) = \sum_k \sum_\beta A_{\alpha\beta,ik}^\tau f_k \left(x_\beta, Q_0^2\right) \tag{3.6}$$

where latin indices run over PDF flavour, greek indices run over points in an initial-scale interpolating x-grid and the evolution operator A may be accessed directly in the `APFEL` package. Given this operator, it is possible to substitute in eq. (3.2) the general-scale PDFs with the ones at the fitting scale:

$$F^{(s)}_{\alpha\beta,\tau} = \sum_{i,j}\sum_{k,l}\sum_{\delta,\gamma} C^{(s)}_{ij}\left[A^{\tau}_{\alpha\delta ik} f_k\left(x_\delta, Q_0^2\right) A^{\tau}_{\beta\gamma jl} f_l\left(x_\gamma, Q_0^2\right)\right] \tag{3.7}$$

$$= \sum_{k,l}\sum_{\delta,\gamma} \tilde{C}^{(s),\tau}_{kl,\alpha\beta\gamma\delta} f_k\left(x_\delta, Q_0^2\right) f_l\left(x_\gamma, Q_0^2\right) \tag{3.8}$$

where

$$\tilde{C}^{(s),\tau}_{kl,\alpha\beta\gamma\delta} = \sum_{i,j} C^{(s)}_{ij} A^{\tau}_{\alpha\delta ik} A^{\tau}_{\beta\gamma jl} \ . \tag{3.9}$$

Now, substituting in the total cross section, the expression for subprocess parton densities in eq. (3.7), one gets:

$$\sigma_{pp\to X} = \sum_{k,l}\sum_{\delta,\gamma}\sum_{p}\sum_{s}\sum_{\alpha,\beta}\sum_{\tau} \alpha_s^{p+p_{LO}}\left(Q_\tau^2\right) W^{(p)(s)}_{\alpha\beta,\tau} \tilde{C}^{(s),\tau}_{kl,\alpha\beta\gamma\delta} f_k\left(x_\delta, Q_0^2\right) f_l\left(x_\gamma, Q_0^2\right) \tag{3.10}$$

and performing further contractions:

$$\sigma_{pp\to X} = \sum_{k,l}\sum_{\delta,\gamma} \tilde{W}_{kl,\delta\gamma} f_k\left(x_\delta, Q_0^2\right) f_l\left(x_\gamma, Q_0^2\right) \ , \tag{3.11}$$

where $\tilde{W}_{kl,\delta\gamma}$ is defined as:

$$\tilde{W}_{kl,\delta\gamma} = \sum_{p}\sum_{s}\sum_{\alpha,\beta}\sum_{\tau} \alpha_s^{p+p_{LO}}\left(Q_\tau^2\right) W^{(p)(s)}_{\alpha\beta,\tau} \tilde{C}^{(s),\tau}_{kl,\alpha\beta\gamma\delta} \tag{3.12}$$

and it is referred to as an *FK table*. Eq. (3.11) is an extremely compact expression for the cross section, given in terms of a simple scalar product between the FK table and the PDFs at a given initial scale, where the sums are carried out respectively over the initial scale interpolating x-grid and the incoming partons.

3.2.3 A comparison of the two methods

The main difference between eq. (3.4) and eq. (3.11) relies upon the different treatment of PDFs evolution [5]. In tools such as `APPLgrid` or `FastNLO`, PDFs evolution is performed at the fitting stage, while in `APFELgrid`, PDFs evolution is pre-cached at the stage of FK table generation, and so only PDFs at the given initial scale are required at the time of fitting. In addition, the number of calculations required at the time of fitting is reduced, given that the sums over hard scale and perturbative order are performed when the FK is generated. Moreover, the simple form (a dot product) now assumed by the convolution, presents the advantageous possibility to apply standard computational tools such as multi-threading through e.g. OpenMP or Single Instruction Multiple Data (SIMD) operations such as SSE3 and AVX, to further improve time performances.

However, despite these convenient features, pre-caching the evolution of PDFs in the FK table limits the possibility to perform fits where there is the need to change the theoretical parameters inherently embedded in the FK, like the perturbative order, the strong coupling or the factorization/renormalization scales. For this reason, the Fast Kernel method is not suitable as a complete replacement for tools such as `APPLgrid` [5].

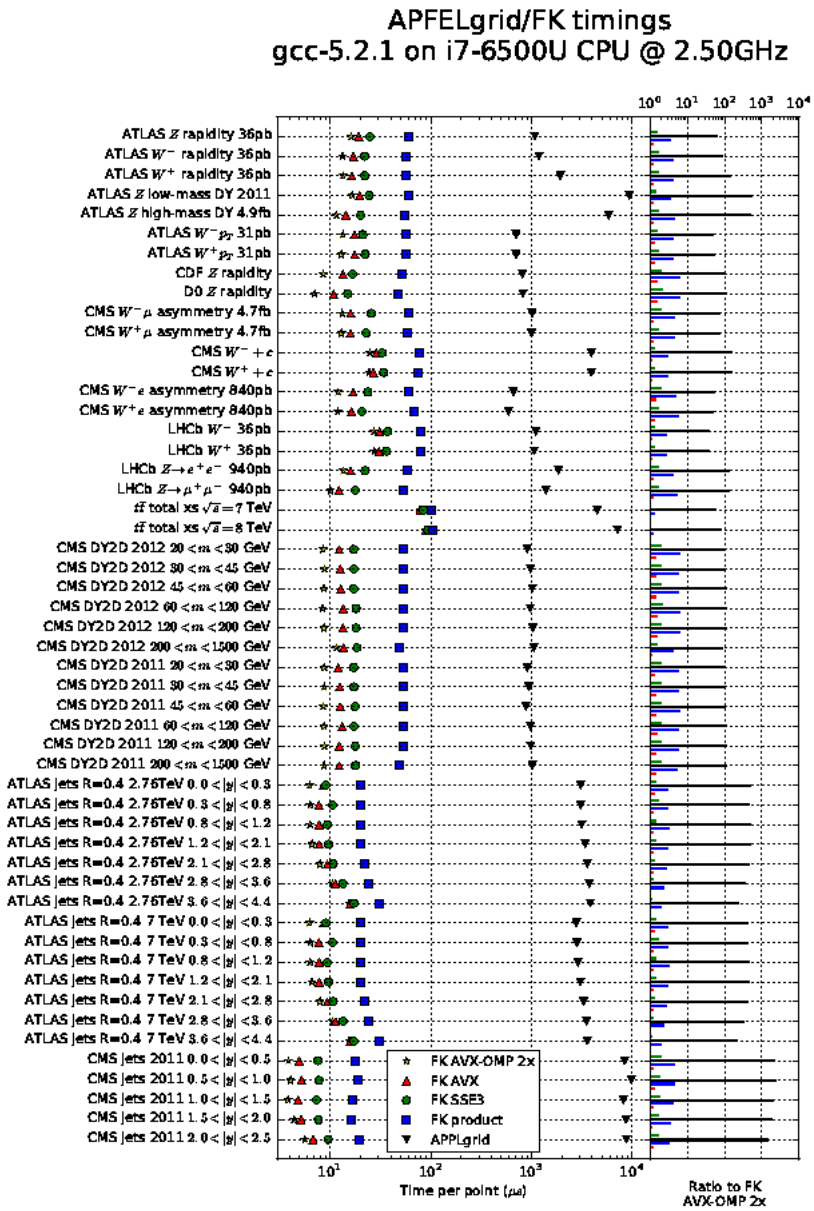

Figure 3.1: performance comparisons between FK with AVX-OpenMP 2x, AVX, SSE3, double precision convolution and APPLgrid convolution time per point and process. Reprinted from *APFELgrid: a high performance tool for parton density determinations*, by Bertone Valerio, Carrazza Stefano and Hartland Nathan P., 2017, retrieved from http://cds.cern.ch/record/2151510/plots.

3.3 Performance benchmarks

In figure (3.1), a comprehensive benchmark between the performance of `APPLgrid` and `APFELgrid` is shown. A wide range of processes including LHC and Tevatron electroweak vector boson production measurements , $t\bar{t}$ total cross sections, double-differential Drell-Yan cross sections and inclusive jet data, are taken into account. A total of 52 source `APPLgrid` is considered to perform predictions over a wide range of kinematics. For the purpose of comparison, FK tables are generated with 30 points in x and an initial scale below the charm threshold, with seven active partonic species [5].

The average time taken for datapoint by the two methods is compared for all the 52 `APPLgrid` table. FK calculations are performed with four different settings: AVX-OpenMP 2x (2 CPU cores), AVX, SSE3 and the standard double precision product.

Figure (3.1) shows that the FK method is systematically faster than the `APPLgrid` one. Indeed, comparing the fastest configuration of the FK calculations, i.e. the AVX-OpenMP 2x setting, it can be seen that the improvement in time performances ranges from a minimum factor of 10 for electroweak vector boson production to a maximum factor of 2000 for inclusive jet data. A significant improvements in timings for the FK is observed even without the use of SIMD or multi-threading.

For more details about other important features to perform a significant comparison between tools such as `APPLgrid` and the FK method, i.e. the table size in the filesystem and memory and the computational cost of pre-computing the FK itself, the reader is referred to the exhaustive discussion presented in [5].

Chapter 4

PDFs covolution: going further

4.1 Overview of the present work

As explained in the previous chapter, over the years PDFs convolution has been reduced to the simple form given by eq. (3.11) and a considerable enhancement in time performances has been achieved. However, this may not be enough: in the future, an always increasing amount of data, coming from experiments at colliders, will be available for PDFs fit and the performances of the fitting procedure should be kept "updated" in order to deal with them. Boosting continuously the time performances of PDFs convolution is thus necessary in order to make full usage of experimental data points.

While it is hard to think of an even simpler form for PDFs convolution, compared to the expression (3.11), it is natural to look at the problem from the *IT side* and investigate further what kind of hardware and technology is best suited to perform this task. Indeed, till now PDFs fits have been carried out on CPU devices but with the advent of GPUs and parallel computing (see paragraph (4.2)), it is spontaneous to ask ourselves if such devices may be more performing.

The work presented in the thesis comes within this context and aims to answer such question. Its main purpose is to improve further the time performances of PDFs convolution achieved in [5]. This is done by running and profiling PDFs convolutions on two different devices, i.e. an Intel(R) Core(TM) i9-9980XE CPU 3.00GHz and a GPU Nvidia Titan V (detailed specs are given at the end of the thesis), with different implementation of the convolution itself. The issue is addressed exclusively from the numerical point of view, maintaining the expression of the convolution given in eq. (3.11). The FK table used to perform the convolutions is generated by `APFELgrid` from the file `atlas-Z0-rapidity.root` (downloadable at https://applgrid.hepforge.org/downloads/), while PDFs are taken from `NNPDF30_nlo_as_0118.LHgrid` (for more details visit: http://nnpdf.mi.infn.it/nnpdf3-0/).

The work has been organized as follow: firstly, the convolution between the FK table and a single PDF has been implemented by means of external libraries such as `Eigen` and `OpenBLAS` (more details in paragraph (4.3)) on CPU and an overall time benchmark of the performances on CPU has been produced (see paragraph (4.5)). Subsequently, many parallel implementations of the convolution using `OpenCL` (see paragraph (4.3)) have been put into practice and the convolution has been run on GPU. The performances achieved with `OpenCL` has been compared to the previous ones and a significant memory overhead has been observed, making `OpenCL` the slowest method (see paragraph (4.5)).

This result showed that the convolution was not suited to be parallelized. This led to a change

in the adopted strategy: the aim of the research became to find a convenient way to run PDFs convolution on GPU, in order to take advantage from the computing power of such devices. Hence, a toy-model representing the convolution has been realized and various setups have been tested. The dimensions of the FK table and the PDF have been changed and the performances on CPU and GPU recorded for various sizes (see paragraph (4.8)). Moreover, the convolution has been carried out maintaining the FK table fixed and reloading at the end of each convolution the PDF to be convoluted. This setup has been tested increasing the number of the PDFs to be convoluted and changing the size of the FK table (see paragraph (4.9)). Finally, a last comparison has been made between `OpenCL` and an implementation of the convolution with `TensorFlow` (see paragraph (4.10)), a famous library for machine learning (see paragraph (4.3)).

In what follows, a brief introduction to parallel computing and to the adopted computational tools is given. Next, the steps described above are presented in more details and results are shown. At the end, a discussion of the results achieved is presented.

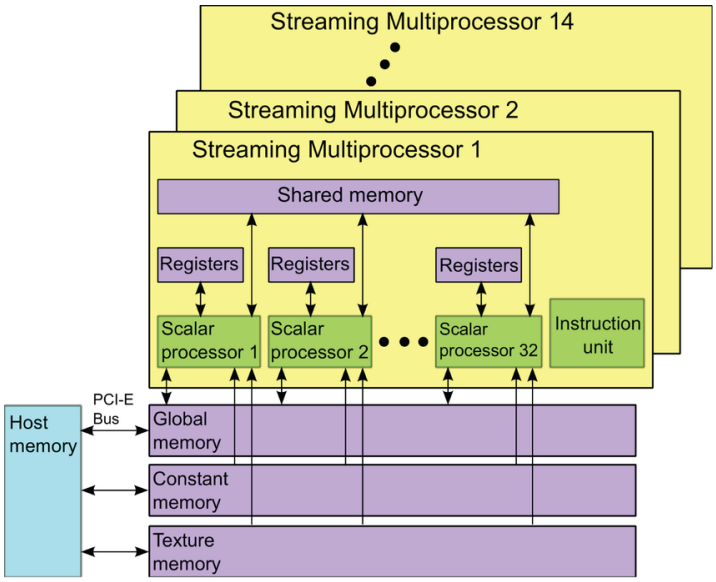

Figure 4.1: a schematic representation of a GPU. Reprinted from *Accelerating Image Reconstruction in Dual-Head PET System by GPU and Symmetry Properties*, by Chou et al., 2012, retrieved from http://www.researchgate.net .

4.2 GPUs and parallel computing[1]

Graphical Processing Units (GPUs) are specialized electronic circuits designed to rapidly manipulate and alter memory to accelerate the creation of images. Even though they have been developed in the context of game consoles[2], they are now used in a wide range of fields. The most interesting

[1]This chapter and the next one are intended to be brief and not technical introductions to the subjects presented. The purpose is to give just the main ideas and motivate the employment of such tools in the present work. References are given to look for more details.

[2]The term was first used by Sony in 1994 with the launch of the PlayStation 1.

feature of GPUs, at least for application in Physics, is their highly parallel structure that makes them suitable for algorithms that process large blocks of data in parallel.

A schematic representation of a GPU is given in figure (4.1). As shown, the GPU is composed of a series of multiprocessor, each of which contains a given number of processors along with their registers and a memory shared between them. The shared memory is accessible only from the processors inside a single multiprocessor. The global and constant memory, instead, are accessible from all the processors in the device. While the global memory is used to read and write data, constant memory is used to store read-only data. The light blue block represents the memory of the host, i.e. typically the CPU connected to the GPU, that communicates with device memory by means of buses. The texture memory and the instruction unit can be neglected for the scope of the present work.

This representation of a GPU closely follows the one given in computational tools for parallel computing such as `CUDA` and `OpenCL` where the processors are respectively referred to as *Cuda Cores* and *Compute Units*. The main idea of parallel computing is to make full usage of the computational power of GPU devices by dividing, when possible, the computational burden between the various processors of the GPU.

To fix the ideas, consider the simple example of a dot product between two vectors $\vec{a}$ and $\vec{b}$ with N components, that in pseudo-code can be written as:

```
for (int i = 0; i < N; i++)
        result += a[i]*b[i];
```

The calculation of each product between the i-th components of $\vec{a}$ and $\vec{b}$ is *independent* from the others and so they can all be performed at once by N distinct processors. The same argument does not hold for the sum which, in turn, must be carried out sequentially.

This is the essence of parallel computing. When parts of a calculation can be performed independently from the others, it may be convenient to execute them in parallel (i.e. simultaneously) to boost the performances of an algorithm. This procedure is known as *parallelization*.

This process should not be confused with *vectorization*, implemented, for example, in SIMD instructions. Vectorization takes place inside one core while parallelization interests many cores.

Returning to the dot product example, vectorization can be explained as follow: instead of storing vector $\vec{a}$ and $\vec{b}$ in memory without a proper alignment, they can be stored in appropriate registers that allows to perform operations on more components at a time, as shown in figure (4.2). The procedure results in a speed up of the computation.

In summary, both parallelization and vectorization results in the simultaneous performance of more operations, but by means of two completely different mechanisms.

4.3 APIs and libraries

The complete list of tools and libraries used in the present work is listed below.

1. **SSE3**, i.e. Streaming SIMD Extensions 3, is the third iteration of the SSE instruction set for the IA-32 (x86) architecture. SSE originally added eight new 128-bit registers known as XMM0 through XMM7, using only a single data type for XMM registers, i.e. four 32-bit single-precision floating point numbers. With SSE2, the available data-types increase. With SSE instructions it is possible to perform operations over 4 floats in an array at a time. For more details see: https://www.intel.co.uk

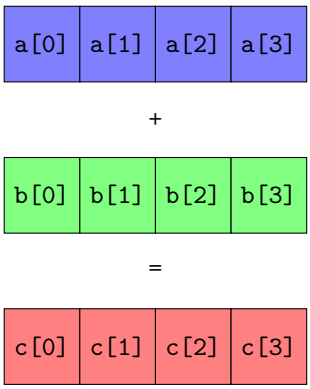

Figure 4.2: a schematic representation of three 128-bit SIMD registers where the sum of the four components is performed at once by means of SIMD instructions.

2. **AVX**, i.e. Advanced Vector Extension, are extensions to the x86 instruction set architecture for microprocessors from Intel and AMD proposed by Intel in March 2008. AVX uses sixteen YMM registers, each of which contains eight 32-bit single-precision floating point numbers. Using AVX is then possible to perform vectorization over 8 components. For more details see: https://www.intel.co.uk

3. **Eigen** is a high-level C++ library of template headers for linear algebra, matrix and vector operations. By means of a cost model of floating point operations, Eigen performs vectorization on its own. For more details see: http://eigen.tuxfamily.org

4. **OpenBLAS** is an open source implementation of the BLAS (Basic Linear Algebra Subprograms) API with many optimizations for specific processor architectures, including Intel Sandy Bridge. For more details see: https://www.openblas.net/

5. **GSL**, i.e. the GNU Scientific Library, is a numerical library, written in C, suited to perform numerical computations in applied mathematics and science. It is free software under the GNU General Public License. For more details see: https://www.gnu.org/software/gsl/

6. **MKL**, i.e. the Intel Math Kernel Library, is a library of optimized math routines for science and engineering. The routines in MKL are optimized specifically for Intel processors. For more details see: https://software.intel.com/en-us/mkl

7. **OpenCL**, i.e. Open Computing Language, is a framework for writing programs that execute across heterogeneous platforms consisting both of CPUs and GPUs. It provides a standard interface for parallel computing. For more details see: https://www.khronos.org/opencl/

8. **TensorFlow** is a free and open-source symbolic math library, used especially for machine learning applications such as neural networks. TensorFlow was developed by the Google Brain team and was released at the of 2015. For more details see: https://www.tensorflow.org/

4.4 More on the FK table

Before moving to the first time benchmark on CPU, it is important to explain in more details how the FK table exactly looks like at code level. In the `APFELgrid` code, the "standard" implementation of the convolution of equation (3.11), i.e. the one written with simple `for` loops, reads:

```
for (int i = 0; i < Ndata; i++)
  for (int n = 0; n < Npdf; i++){
    C[i*Npdf + n] = 0;
          for (int j = 0; j < N; i++)
            C[i*Npdf + n] += (FK+N*i)[j]*(pdf+N*n)[j];
  }
```

that, in mathematical formulas, reads:

$$C_{in} = \sum_{j=0}^{N} (FK)_{ij} \, (pdf)_{jn} \ , \qquad i = 0, ..., N_{data} \ , \qquad n = 0, ..., N_{pdf} \ . \tag{4.1}$$

where the matrix $(FK)_{ij}$ is the FK table and $(pdf)_{jn}$ is a matrix containing the PDFs to be convoluted. `Ndata`, `Npdf` and `N` will be defined shortly.

Now, this can be quite confusing. How exactly eq. (3.11) reduces to eq. (4.1) is not trivial. It is useful to rewrite here a slightly modified version of eq. (3.11) in order to make the opportune assignments and identifications:

$$\sigma^{(i)}_{pp\to X} = \sum_{k,l} \sum_{\delta,\gamma} \tilde{W}^{(i)}_{kl,\delta\gamma} f_k \left(x_\delta, Q_0^2\right) f_l \left(x_\gamma, Q_0^2\right) \ , \tag{4.2}$$

where the index (i) runs over the available experimental measurements (data points) of the cross section, e.g. at different energy scales Q^2 of the process. Consider now the FK table alone:

$$\tilde{W}^{(i)}_{kl,\delta\gamma} \ , \tag{4.3}$$

where the index (i) runs over the data points, k and l runs over active parton flavors in the process and δ and γ runs over the `APPLgrid` x-grid points. This FK can be written as a matrix:

$$(FK)_{ij} \tag{4.4}$$

identifying the index i with the index (i) over the data points and the index j with the multi-index (k, l, δ, γ). PDFs now should be reorganize in a matrix with a number of rows equal to the number of column of the matrix $(FK)_{ij}$ to perform the convolution in the matrix per matrix product fashion. The PDFs matrix should contain the values of the different flavor PDFs k and l, at certain point x_γ and x_δ at a given energy scale Q_0^2. Given this matrix, namely $(pdf)_{jn}$, the convolution can finally be written in the form of eq. (4.1), where the index n runs over the number of PDFs included in the convolution[3].

With this identification is now easy to understand the meaning of `Ndata`, `Npdf` and `N`: `Ndata` is the number of available data points, `Npdf` is the number of PDFs convoluted with the FK table and `N` is given by $N_k N_l N_\delta N_\gamma$ where N_k is the number of possible values for k, N_l is the number of possible values for l and so on.

[3] If $n = 1$, the FK is convoluted with one PDF, if $n = 2$ with two PDFs and so on.

For the FK table used in the present work, the number of x_γ and x_δ grid points considered is 27, so $N_\delta = 27$ and $N_\gamma = 27$; The number of active flavor is 7, so $N_k = 7$ and $N_l = 7$; finally, the number of data points is 8. The matrix $(FK)_{ij}$ is then a matrix of size 8 x $(27 \cdot 27 \cdot 7 \cdot 7)$, i.e. a matrix 8 x 35721. The matrix $(pdf)_{jn}$ is a matrix 35721 x `Npdf`, where `Npdf` is set each time to the desired value.

4.5 First time benchmarks

4.5.1 Without OpenCL

In the first stage of the analysis, the convolution (4.1) has been rewritten with all the tools listed in paragraph (4.3) except `OpenCL` and `TensorFlow`. The parameter `Npdf` has been set to 1, and the convolution has been performed 2000 times for each method (i.e. `AVX`, `SSE3` and so on), measuring the elapsed time. Then, the average elapsed time has been evaluated for each method. Results achieved are shown in figure (4.3).

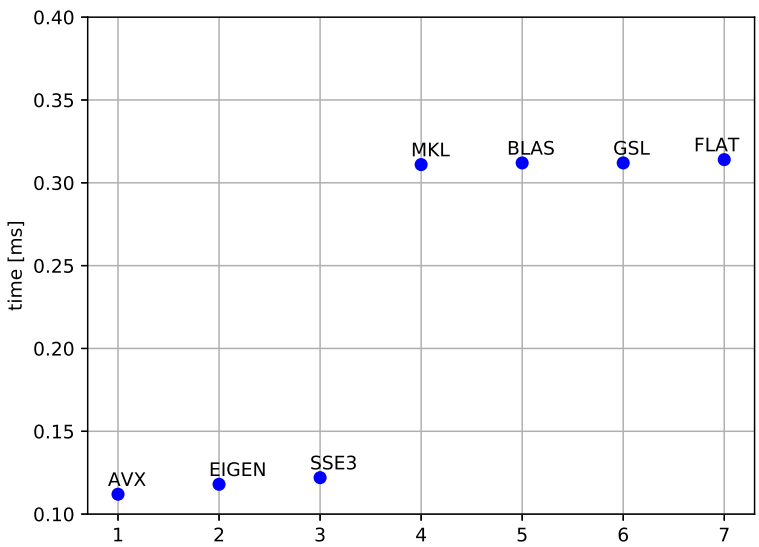

Figure 4.3: mean values of the elapsed time for each method. `FLAT` refers to the standard convolution with simple `for` loops shown in the previous chapter. As shown, the `AVX` version of the convolution is the fastest one.

As it is possible to see, `AVX` instructions allows to perform the convolution in approximately one third of the time needed by the `FLAT` method, i.e. the standard convolution with simple `for` loops.

4.5.2 With OpenCL

After the time benchmark of figure (4.2) has been produced, the convolution has been implemented also in `OpenCL` and has been run on CPU and GPU maintaining `Npdf` = 1. There are more ways in

which eq. (4.1) can be parallelize and so more version of the `OpenCL` convolution. However, for this first time benchmark, this is irrelevant since the different implementations took approximately the same amount of time to perform the calculation. The various way of parallelizing eq. (4.1) will be discussed in the next paragraph.

Results achieved with `OpenCL` are compared to the previous ones in figure (4.4), where results of various `OpenCL` implementations are reported as a single value, being very close to each other.

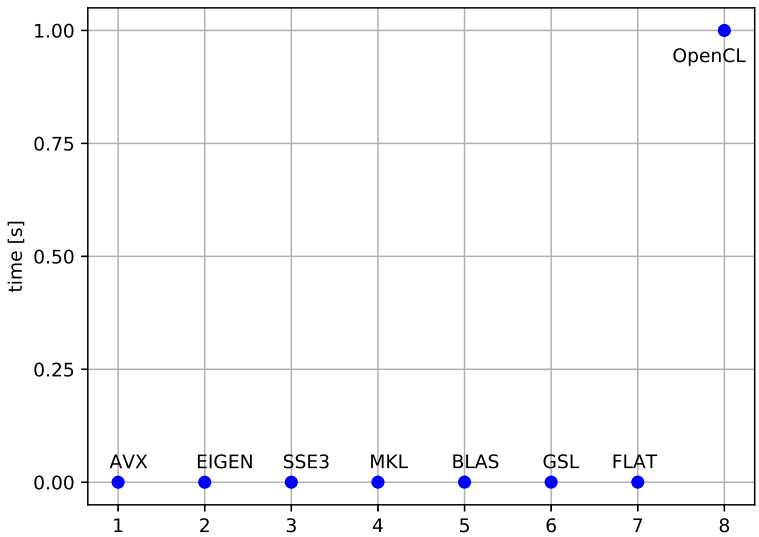

Figure 4.4: mean values of the elapsed time for each method, including `OpenCL`. As shown, `OpenCL` presents a significant (of approximately 3 order of magnitude) worsening of the time performances.

The figure shows that `OpenCL` takes about 1 second to perform the convolution, both on CPU and GPU. So, the time performances of `OpenCL` are approximately 3 order of magnitude worse than the performances achieved with the other methods.

This result can be explained profiling with more accuracy the `OpenCL` implementation of the convolution. Indeed, a typical `OpenCL` program is composed of six fundamental components, called *platform*, *device*, *context*, *program*, *kernel* and *queue*. This elements are necessary in order to interface the host with the device and execute the program on it. In particular, the code that will be executed by the single compute units of the device is written in *kernel*.

Describing in details the feature of all these six components is out of the scope of the present work. However, it should be pointed out that building such environment, necessary to subsequently execute the program, requires a significant amount of time, that could not be neglected in a time performance analysis.

Another "bottleneck" in time performances is related to the creation of `OpenCL` *buffers*, i.e. functions aimed at interfacing the memory of the host with the device memory, making "visible" to the device data stored in host memory.

After a better profiling of the `OpenCL` performances, it has been shown that the building of the `OpenCL` environment requires more time than the calculation itself and, more specifically, it takes

approximately 1 second. So, the bad results achieved can be addressed to such feature of `OpenCL` programs, common also to other parallel programming tools (e.g. `CUDA`).

This result shows that, in order to benefit from the advantages of GPU and parallel computing, the amount of calculations to be performed after the environment has been built, should be increased considerably. This is the strategy adopted in the next steps of the analysis.

4.6 OpenCL kernels

Before moving to the next time benchmarks, consider again the matrix product of equation eq. (4.1) or, even better, its `C++` counterpart. As mentioned in the dot product example of paragraph (1.12), the products between the j-th components of the FK and of the PDFs arrays are independent from each other, and the same holds for the two `for` loops over indexes `i` and `n`. Indeed, these can be performed in parallel, evaluating all the components of `out[i*Npdf+n]` simultaneously. The only operation that cannot be performed in parallel is the sum[4].

So, there are three operations that can be parallelized, i.e. the `for` loop over `i`, the `for` loop over `n` and the component-wise multiplication between the FK and the PDFs arrays. This results in seven possible ways in which the `OpenCL` kernel can be written, i.e. parallelizing just one of the operations (3 possibilities), parallelizing two of them (3 possibilities) or even all of them (1 possibility).

In what follows, the different `OpenCL` implementations of the convolution will be referred to as:

- **CL1**: where the component-wise multiplication is performed in parallel,
- **CL2**: where the two `for` loops are performed in parallel,
- **CL3**: where the `for` loop over `i` is performed in parallel,
- **CL4**: where the `for` loop over `i` and the component-wise multiplication are performed in parallel,
- **CL5**: where the `for` loop over `n` and the component-wise multiplication are performed in parallel,
- **CL6**: where the `for` loop over `n` is performed in parallel.

The implementation with all the operations performed in parallel has been discarded for memory and performance issues.

A feature common to the `OpenCL` versions of the convolution where the component-wise multiplication is performed in parallel, i.e. CL1, CL4 and CL5, is the need for an auxiliary array to store the results of the multiplications. This involves more memory traffic and a slowdown of the performances, as the results will show.

4.7 Testing different setup

The first results achieved with `OpenCL` showed that the convolution between the given FK table an a single PDF was not suited to be parallelized and that a new strategy should be developed in order to address the problem. So, a toy-model representing the convolution of eq. (4.1) was developed,

[4]However it is possible to split the sum of, say, N components in n sums of N/n components and perform the n sums in parallel. At the end the partial results should be summed in order to obtain the result of the original sum.

where the FK and the *pdf* matrices were represented by two dense matrices respectively with sizes `Ndata` x `N` and `N` x `Npdf`.

Then, the convolution was performed, changing `Ndata`, `N` and `Npdf`, with all the `OpenCL` methods and with `AVX`, in order to find a setup (combination of the three parameters) for which the parallel implementation achieves better time performances than the fastest sequential implementation of the convolution.

In doing so, one has to keep in mind what actually means to increase the sizes of the two matrices from the perspective of a PDFs fit. Remember that `Ndata` is the number of experimental data points, `Npdf` is the number of PDFs convoluted with the FK and `N` is given by the product of the points in the x_γ and x_δ grids and the number of active flavors. So, when `Ndata` is increased, we are saying that more data are available for the fit, which is perfectly reasonable given that the number of available data from collider experiments is increasing. When `Npdf` is increased, more PDFs are convoluted with the FK. While the number of physical PDFs (i.e. the number of active flavours) is fixed by the process under study, the number of PDFs replicas can be arbitrarily increased, making again this scenario feasible. Indeed, we can imagine to organise all the PDFs replicas in the PDFs matrix and, for instance, compute the convolution for each replica at the same time in parallel (more details in chapter 5). Finally, increasing `N` has a less straightforward meaning, being `N` the multi-index given by (k, l, δ, γ) (see paragraph (4.4)). An increase in `N` may correspond to an improvement of our knowledge of PDFs in a wider kinematic region, to an increase in the number of active flavors (i.e. the number of physical PDFs) or to both of them. This last option is process-dependent and so the results achieved with the toy-model can be reproduce in a real fit only for those processes with the same physical conditions.

4.8 GPU vs CPU

In this paragraph, the results achieved running the toy-model convolution with `AVX` on CPU and with `OpenCL` on GPU are presented. The graphs of figure (4.5), (4.6) and (4.7) show time performance trends obtained changing one parameter between `N`, `Ndata` and `Npdf` while keeping the other fixed. Graph (4.8) instead shows time performances obtained varying simultaneously `Npdf` and `Ndata` while keeping `N` fixed. A comprehensive summary of all the case studies is given in table (4.1). Trend obtained changing at the same time the values of `N` and `Ndata` or `Npdf` are shown only in table for practical reasons.

As it is possible to see in figure (4.5), when only `N` is increased, the time needed to perform the convolution with `AVX` remains significantly lower than the time needed by `OpenCL`. Moreover, the different `OpenCL` implementations take approximately the same time apart from CL6 that suffers from very bad performances because it parallelizes over `Npdf`, but `Npdf` $= 1$.

A similar trend is observed in figure (4.6), when only `Ndata` is increased. As might be expected, better results are achieved with CL2 and CL3, that parallelizes over `Ndata`. The bad performances of CL6 can be addressed to the same cause mentioned above.

Something new appears in figure (4.7). When `Npdf` is increased, time performances of `AVX` gets worse and becomes comparable to the ones achieved with CL2 and CL6. However, even though this result hints to the possibility that PDFs fits can take advantages of parallel computing, it does not justify alone the employment of such devices to perform PDFs fits and convolutions.

A considerable improvement in time performances instead is observed when both `Npdf` and `Ndata` are increased. Here, CL2, CL3 and CL6 respectively perform the convolution in approximately 1/4, 1/6 and 1/6 of the time needed by `AVX`. So, this setup is particularly suited to be parallelize. The bad performances achieved with the other `OpenCL` implementations can be addressed to the additional

N	Npdf	Ndata	AVX [s]	CL1 [s]	CL2 [s]	CL3 [s]	CL4 [s]	CL5 [s]	CL6 [s]
35712	1	8	0.000157	0.9	0.9	0.9	0.9	0.9	0.9
10^6	1	8	0.005	1	0.9	0.9	0.9	0.9	1.2
10^7	1	8	0.057	1.47	1.49	1.51	1.38	1.47	3.97
10^8	1	8	0.575	7.13	7.1	7.17	6.31	6.71	32.07
35712	10^2	8	0.016	1.02	0.9	1.04	0.948	0.96	0.908
35712	10^3	8	0.169	2.384	0.928	2.342	1.868	1.786	0.996
35712	10^4	8	1.731	16.35	1.757	15.655	11.0271	10.18	1.8
35712	1	10^2	0.00243	0.873	0.866	0.874	0.865	0.88	1.033
35712	1	10^3	0.0214	1.081	0.94	0.96	1.085	1.114	2.209
35712	1	10^4	0.216	3.23	1.759	1.761	3.14	3.43	14.377
35712	10^2	10^2	0.186	2.815	0.895	1.234	1.922	1.908	1.252
35712	10^3	10^3	21.61	193.237	5.252	8.46	101.552	101.598	8.413
35712	$2 \cdot 10^3$	$2 \cdot 10^3$	86.13	$\gg$	25.004	16.052	$\gg$	$\gg$	15.967
10^5	10^3	8	0.486	4.6	1.097	5	3.68	3.517	1.283
10^5	10^4	8	4.863	38.634	3.409	42.294	26.785	27.19	3.504
10^6	10^3	8	4.903	34.093	3.425	44.236	25.941	28.316	4.977
10^5	1	10^3	0.0611	1.49	1.114	1.172	1.488	1.488	4.646
10^5	1	10^4	0.606	6.917	3.331	3.332	7.287	7.243	38.725
10^6	1	10^3	0.615	6.108	3.529	3.536	7.425	6.622	39.557

Table 4.1: time performances achieved with `AVX` on CPU and `OpenCL` on GPU for the reported setups. Time is given in seconds.

memory traffic coming from the parallelization of the component-wise multiplication.

With regards to setups in which `N` is increased along with one of the other two parameters, it is possible to observe from table (4.1), that, when `Ndata` is increased, `AVX` performs the convolution approximately one order of magnitude faster than every `OpenCL` implementation; when `Npdf` is increased instead, results comparable to the `AVX` ones are achieved by CL2 and CL6.

In summary, the toy-model suggests that only when the number of operations grows considerably, due to an increase in the number of iterations in the two for loops, parallelize over these loops can be convenient and can result in a significant improvement in time performances.

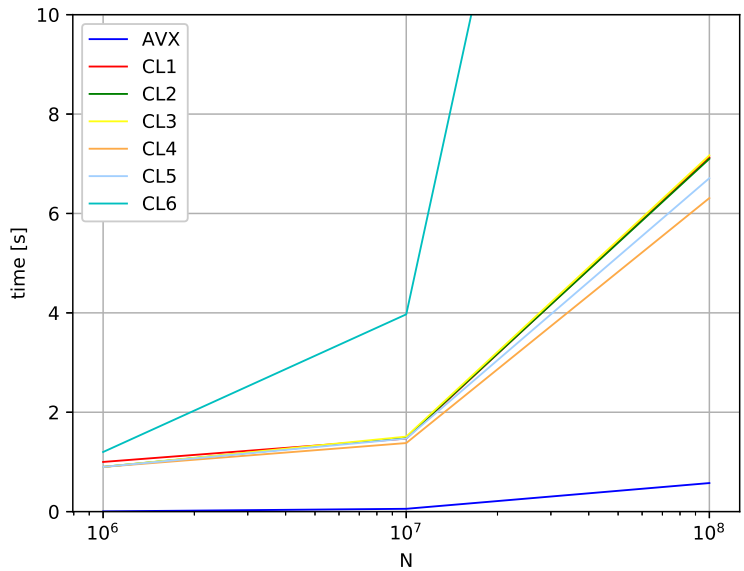

Figure 4.5: time performances achieved with `AVX` and `OpenCL`; `Ndata` = 8 and `Npdf` = 1.

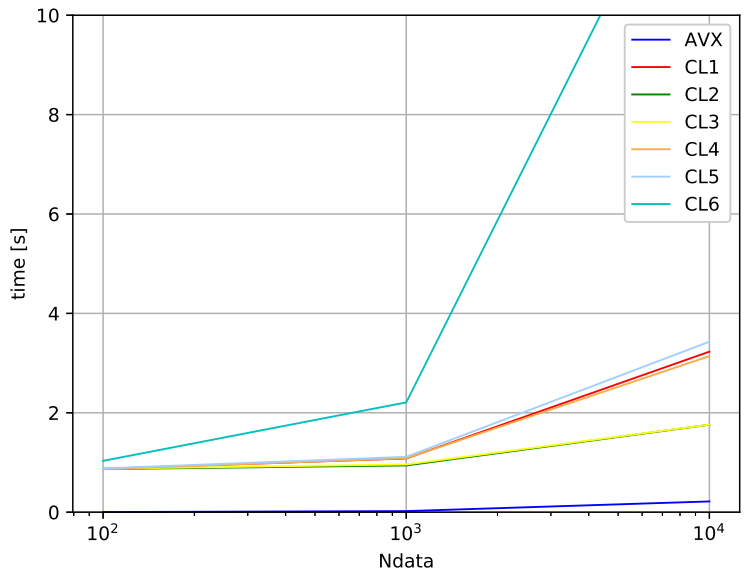

Figure 4.6: time performances achieved with `AVX` and `OpenCL`; `fDSz` = 35721 and `Npdf` = 1.

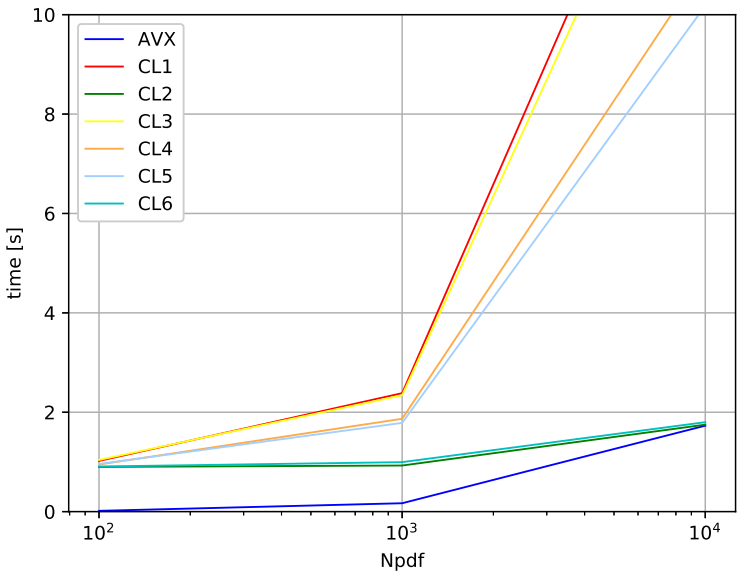

Figure 4.7: time performances achieved with `AVX` and `OpenCL`; `Ndata` = 8 and `fDSz` = 35721.

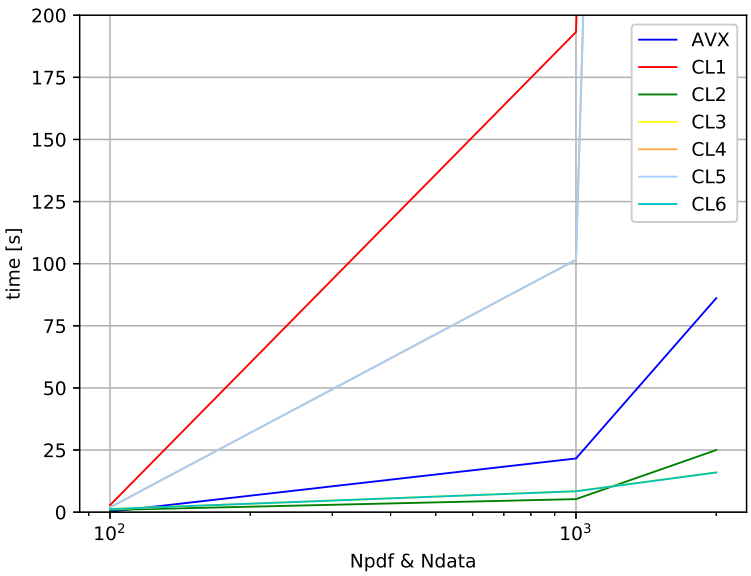

Figure 4.8: time performances achieved with `AVX` and `OpenCL`; `fDSz` = 35721.

fDSz	Npdf	fNData	NumPDFs	Avx [s]	fk3 [s]
35712	1	10^2	10^2	0.0018 ± 0.0001	0.0049 ± 0.0001
35712	1	10^2	10^3	0.00187 ± 0.00003	0.00350 ± 0.00003
35712	1	10^2	10^4	0.00186 ± 0.00003	0.00330 ± 0.00003
35712	1	10^3	10^2	0.0210 ± 0.0002	0.0101 ± 0.0002
35712	1	10^3	10^3	0.0210 ± 0.0002	0.0080 ± 0.0002
35712	1	10^3	10^4	0.0210 ± 0.0002	0.0076 ± 0.0002
35712	1	10^4	10^2	0.213 ± 0.001	0.0212 ± 0.0002
35712	1	10^4	10^3	0.213 ± 0.001	0.0090 ± 0.0002
35712	1	10^4	10^4	0.213 ± 0.001	0.00785 ± 0.0002

Table 4.2: time performances achieved with `AVX` on CPU and CL3 on GPU for the reported setups. Time is given in seconds.

4.9 Reloading the PDF

Apart from the matrix per matrix product, there is another way to perform the convolution between a fixed FK table and a given number of PDFs. Indeed, instead of storing the PDFs in a matrix `N` × `Npdf`, a single array PDF can be convoluted with the FK table and, at the end of the convolution, the array can be loaded with a new PDF. The convolution with a given number of PDFs, say `NumPDFs`, can then be carried out convoluting the FK table with a single PDF for `NumPDFs` times, reloading the PDF values at the end of each convolution.

Such approach has been tested for different values of `NumPDFs` and `Ndata`. The convolution has been implemented in parallel using the CL3 method, given that now it is no more possible to parallize over `Npdf`, being it equal to 1, and that the methods CL1, CL4, CL5 have shown to be slower than the CL3 implementation. Results obtained are shown in figures (4.9), (4.10) and (4.11) and summarized in table (4.2).

Figure (4.9) shows the average time per PDF employed by `AVX` and CL3 to perform the convolution. The number of data points `Ndata` is set equal to 10^2 while `NumPDFs` is varied between 10^2 and 10^4. As it is possible to see, `AVX` is more performing, even if the time difference is not so big. Also, while `AVX` curve is a constant, average time per PDF decreases with `NumPDFs` for the `OpenCL` implementation. This feature is common to all the three figures.

In figure (4.10) the situation is reversed: `OpenCL` is more performing than `AVX` but the difference is little. Here `Ndata` is equal to 10^3 and the time is given in seconds, as in figure (4.11).

A considerable improvement in time performances is gained when `Ndata` is set to 10^4. The average time per PDF taken by `OpenCL` reduces to about 0.01 $[s]$, i.e. one order of magnitude lower than the `AVX` counterpart.

This result is notable for two reasons: firstly because of the great saving of time achieved with such configuration and secondly because a configuration with `Ndata` $= 10^4$ and `NumPDFs` $= 10^2$ is absolutely "realistic" and no doubt can occur in real PDFs fits.

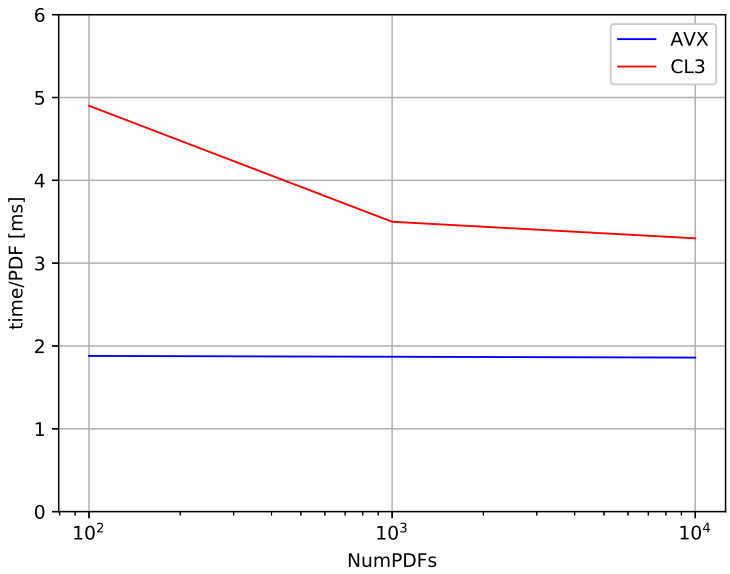

Figure 4.9: comparison between time performances achieved with `AVX` and CL3; `Ndata` is set to 10^2.

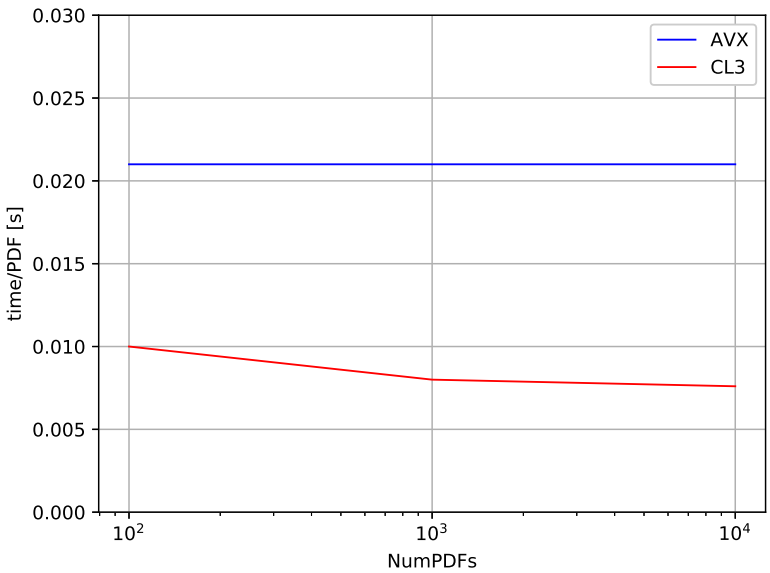

Figure 4.10: comparison between time performances achieved with `AVX` and CL3; `Ndata` is set to 10^3.

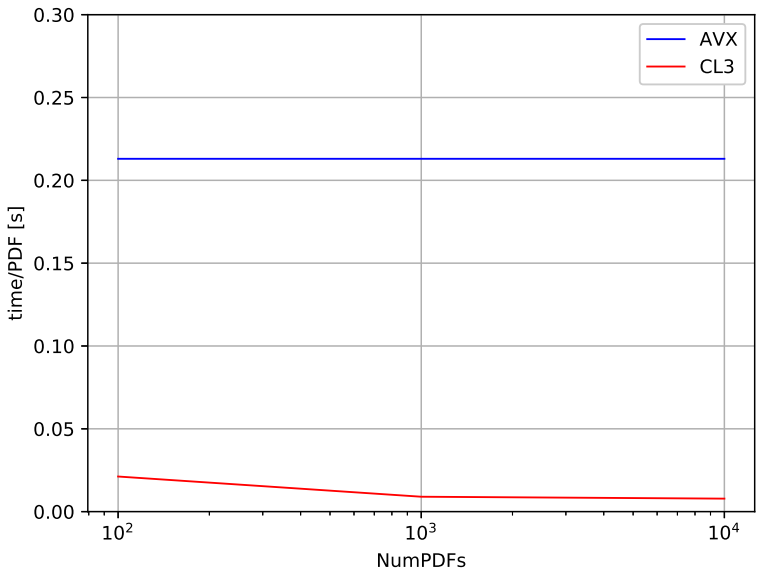

Figure 4.11: comparison between time performances achieved with `AVX` and CL3; `Ndata` is set to 10^4.

4.10 Adding `TensorFlow`

Results illustrated in the previous paragraphs are encouraging. Indeed, they show that there are some configurations and setups for which a parallel implementation of the convolution is considerably more performing than the `AVX` one. Now, the analysis can be pushed forward investigating if there is a more efficient way of running the convolution in parallel. For the previous benchmarks, `OpenCL` has been used because it executes on heterogeneous platforms and so it is particularly suited for applications that in principle can happen to be executed over different machines. Though, other parallel programming framework, such as `CUDA`, can be employed and their performances can be tested against the ones achieved with `OpenCL`.

In the present work, the toy-convolution has been rewritten using `TensorFlow`, a library commonly used in Neural Networks applications (see paragraph (4.3)). Compared to `OpenCL`, `TensorFlow` automatically performs the matrix per matrix product in parallel (when a GPU is connected to the host) and use all the memory of the GPU by default. Everything is done by calling just the single high level function `tf.matmul`. On the other hand however, what happens at low level remains hidden.

Results for the `TensorFlow` implementation of the convolution has been produced both for the case in which the three parameters `N`, `Ndata` and `Npdf` are varied and for the case in which a fixed FK is convoluted with a number `NumPDFs` of PDFs.

As it is the case for `OpenCL`, `TensorFlow` is significantly more performing than `AVX` when both `Ndata` and `Npdf` are increased, as shown in figure (4.12). With regards to the other setups, `TensorFlow` results are comparable to `OpenCL` ones and are not reported for the sake of simplicity.

fDSz	Npdf	fNData	NumPDFs	Avx [s]	fk3 [s]	TF [s]
35712	1	10^2	10^2	0.00188	0.0049	0.00457
35712	1	10^2	10^3	0.00187	0.0035	0.00166
35712	1	10^2	10^4	0.00186	0.0033	0.00143
35712	1	10^3	10^2	0.021	0.01	0.0133
35712	1	10^3	10^3	0.021	0.008	0.00278
35712	1	10^3	10^4	0.021	0.0076	0.00167
35712	1	10^4	10^2	0.213	0.0212	0.0925
35712	1	10^4	10^3	0.213	0.009	0.0125
35712	1	10^4	10^4	0.213	0.00785	0.00445

Table 4.3: time performances achieved with `AVX` on CPU, CL3 and `TensorFlow` on GPU for the reported setups. Time is given in seconds.

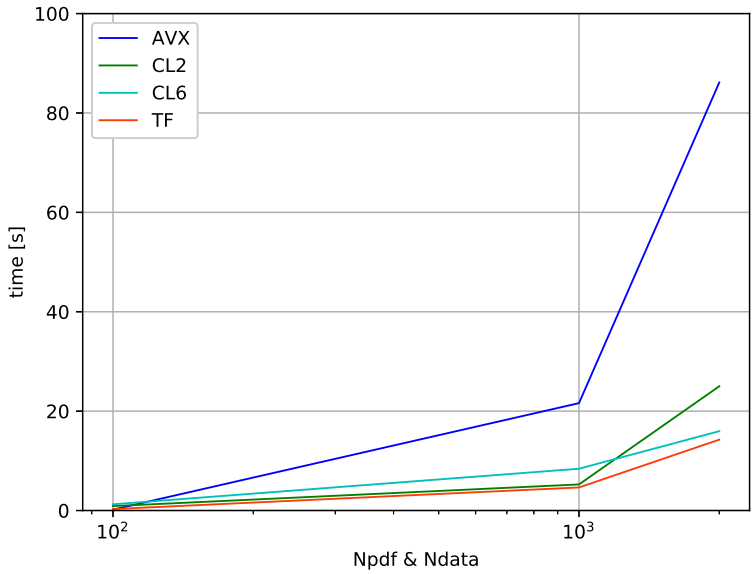

Figure 4.12: time performances achieved with `AVX`, `OpenCL` and `TensorFlow`; `fDSz` = 35721.

Moving to the single PDF analysis, again it is possible to observe in figures (4.13), (4.14) and (4.15) that `OpenCL` and `TensorFlow` time performances are comparable to each other. Values are given in table (4.3).

These results show that there are no significant differences between the two methods in terms of time performances. Still, it is worth noticing that `OpenCL` implementations such as CL2, CL3 and CL6 works in a totally different way with respect to `TensorFlow`. Indeed, while `TensorFlow` uses by default all the memory of the GPU, such `OpenCL` implementations use just a certain amount of it.

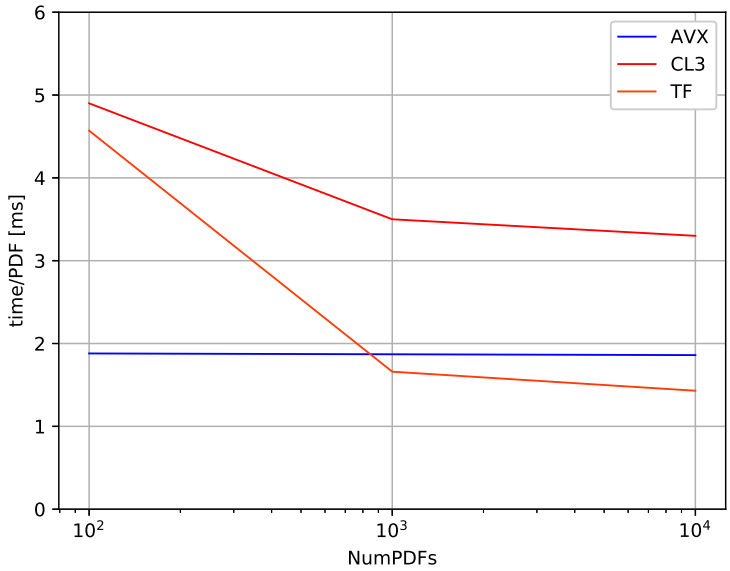

Figure 4.13: comparison between time performances achieved with `AVX`, CL3 and `TensorFlow`; `Ndata` is set to 10^2.

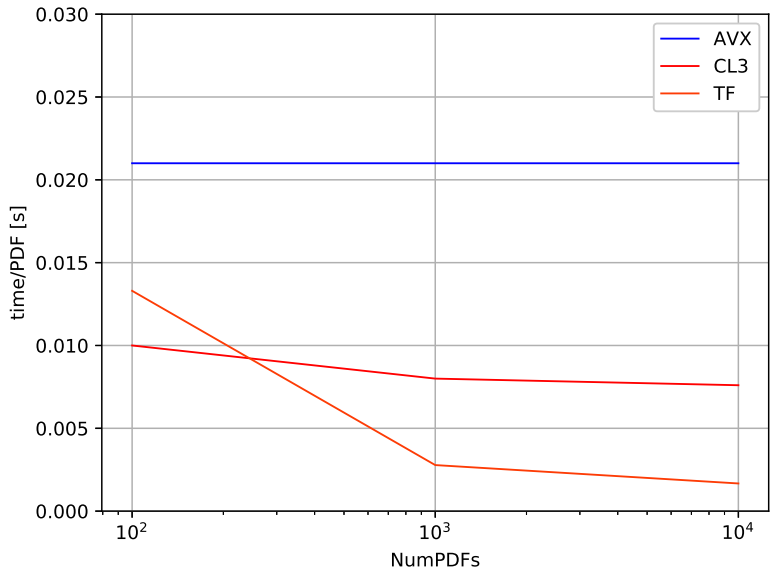

Figure 4.14: comparison between time performances achieved with `AVX`, CL3 and `TensorFlow`; `Ndata` is set to 10^3.

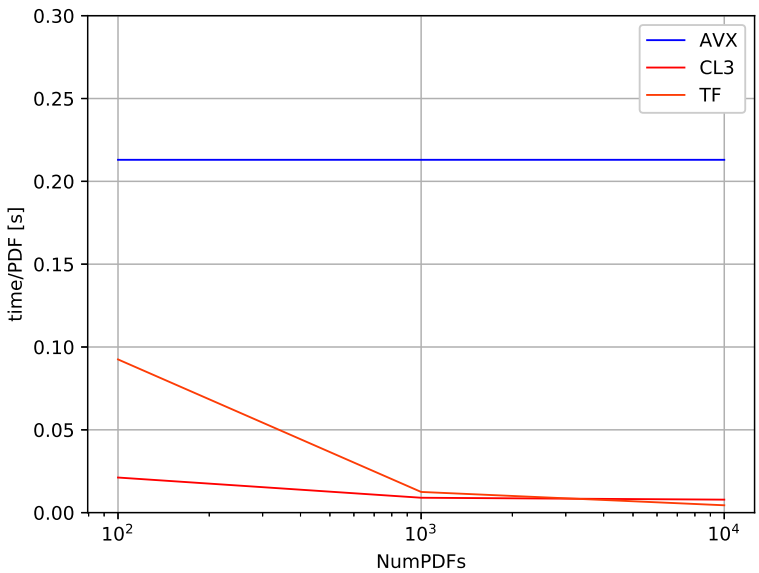

Figure 4.15: comparison between time performances achieved with `AVX`, CL3 and `TensorFlow`; `Ndata` is set to 10^4.

Chapter 5

Conclusions and outlook

The amount of data coming from experiments at colliders is increasing over time and the standard techniques to perform PDF fits are becoming inadequate to deal with them. Indeed, during the fitting procedure, the convolution between the partonic cross sections and the PDFs is computed thousands of time until the minimization algorithm converges. From a computational point of view, these operations are extremely energy and time consuming.

A great result in term of time performances was first achieved with the introduction of tools such as `APPLgrid` and `APFEL`, that make usage of interpolating techniques to precompute part of the convolution in order to reduce the amount of operations that has to be performed during the fit. An even greater improvement of the time performances was gained by joining together the two methods into `APFELgrid`, where the FK table was introduced and the convolution reduced to a simple matrix per matrix product.

After the introduction of the FK method, the further speed up of the convolution was addressed in this thesis studying the time performances achievable on GPU with parallel computing. The first attempt to simply move the convolution introduced in section (4.1) on GPU using `OpenCL` was unsuccessful, so the search for a convenient way of performing PDF fits on GPU was carried out by means of a toy-model, i.e. using fictitious FK, data-points and PDFs of chosen dimensions.

Results illustrated in the previous chapters suggest that there are ways to perform PDF fits in parallel on GPU that are more performing, in terms of time saving, than the currently adopted fitting procedure. Indeed, they show that when the number of calculations increases considerably, i.e. the number of data points and the number of convoluted PDFs increase, `AVX` take more time than `OpenCL` and `TensorFlow` to perform the toy-model convolution. More specifically, results show that CL2, CL3 and CL6 kernels are significantly more performing than `AVX` when it comes to perform a convolution with `Ndata` and `Npdf` of orders ranging from 10^2 to 10^4. CL3 is one order of magnitude more performing than `AVX` also in the case of a single PDF reloaded at the end of each convolution, the same result holding also for `TensorFlow`.

In summary, there are two convenient ways to perform PDF fits in parallel on GPU, given that the number of data-points is of the order of 10^2 or more. The first one is to perform, at each step of the genetic algorithm, the N_{rep} convolutions at the same time in parallel, organising the PDFs matrix in order to contain the N_{rep} replicas of the PDFs. This situation corresponds to the case in which both `Ndata` and `Npdf` are large. The second one instead take advantage of the parallelization over the number of data-points. This is applicable both to the previous case in which the replicas of the PDFs are organized in the PDFs matrix and to the case in which a single PDF at the time is convoluted with the FK. This second scenario corresponds to `NumPDFs` ranging from 10^2 to 10^4.

With regards to the parallelization over `N` instead, results show that there is no gain in time performances with respect to `AVX`, that is considerably faster. Moreover, a solution based on `N` would be unlikely applicable to general PDF fits, given that in `N` are embedded the characteristics of the FK, strictly related to the process under study.

In the future, the analysis could be refined investigating more deeply which type of parallel computing tool is best suited to perform the fit. Moreover, real PDF fits should be carried out with the proposed solutions to test the actual time performances against the ones predicted by the toy-model.

Bibliography

[1] Sergey Alekhin et al. The PDF4LHC Working Group Interim Report. 2011.

[2] Guido Altarelli and G. Parisi. Asymptotic Freedom in Parton Language. *Nucl. Phys.*, B126:298–318, 1977.

[3] Richard D. Ball, Luigi Del Debbio, Stefano Forte, Alberto Guffanti, Jose I. Latorre, Andrea Piccione, Juan Rojo, and Maria Ubiali. A Determination of parton distributions with faithful uncertainty estimation. *Nucl. Phys.*, B809:1–63, 2009. [Erratum: Nucl. Phys.B816,293(2009)].

[4] Richard D. Ball, Luigi Del Debbio, Stefano Forte, Alberto Guffanti, Jose I. Latorre, Andrea Piccione, Juan Rojo, and Maria Ubiali. Precision determination of electroweak parameters and the strange content of the proton from neutrino deep-inelastic scattering. *Nucl. Phys.*, B823:195–233, 2009.

[5] Valerio Bertone, Stefano Carrazza, and Nathan P. Hartland. APFELgrid: a high performance tool for parton density determinations. *Comput. Phys. Commun.*, 212:205–209, 2017.

[6] Valerio Bertone, Stefano Carrazza, and Juan Rojo. APFEL: A PDF Evolution Library with QED corrections. *Comput. Phys. Commun.*, 185:1647–1668, 2014.

[7] J. D. Bjorken and Emmanuel A. Paschos. Inelastic Electron Proton and gamma Proton Scattering, and the Structure of the Nucleon. *Phys. Rev.*, 185:1975–1982, 1969.

[8] M. Botje. QCDNUM: Fast QCD Evolution and Convolution. *Comput. Phys. Commun.*, 182:490–532, 2011.

[9] Raymond Brock et al. Handbook of perturbative QCD: Version 1.0. *Rev. Mod. Phys.*, 67:157–248, 1995.

[10] Stanley J. Brodsky and Glennys R. Farrar. Scaling laws at large transverse momentum. *Phys. Rev. Lett.*, 31:1153–1156, Oct 1973.

[11] Tancredi Carli, Dan Clements, Amanda Cooper-Sarkar, Claire Gwenlan, Gavin P. Salam, Frank Siegert, Pavel Starovoitov, and Mark Sutton. A posteriori inclusion of parton density functions in NLO QCD final-state calculations at hadron colliders: The APPLGRID Project. *Eur. Phys. J.*, C66:503–524, 2010.

[12] G. D'Agostini. *Bayesian reasoning in data analysis: A critical introduction.* 2003.

[13] Luigi Del Debbio, Stefano Forte, Jose I. Latorre, Andrea Piccione, and Joan Rojo. Neural network determination of parton distributions: The Nonsinglet case. *JHEP*, 03:039, 2007.

[14] Yuri L. Dokshitzer. Calculation of the Structure Functions for Deep Inelastic Scattering and e+ e- Annihilation by Perturbation Theory in Quantum Chromodynamics. *Sov. Phys. JETP*, 46:641–653, 1977. [Zh. Eksp. Teor. Fiz.73,1216(1977)].

[15] R. Keith Ellis, W. James Stirling, and B. R. Webber. QCD and collider physics. *Camb. Monogr. Part. Phys. Nucl. Phys. Cosmol.*, 8:1–435, 1996.

[16] R. P. Feynman. The behavior of hadron collisions at extreme energies. *Conf. Proc.*, C690905:237–258, 1969.

[17] Stefano Forte, Lluis Garrido, Jose I. Latorre, and Andrea Piccione. Neural network parametrization of deep inelastic structure functions. *JHEP*, 05:062, 2002.

[18] Murray Gell-Mann. A Schematic Model of Baryons and Mesons. *Phys. Lett.*, 8:214–215, 1964.

[19] V. N. Gribov and L. N. Lipatov. Deep inelastic e p scattering in perturbation theory. *Sov. J. Nucl. Phys.*, 15:438–450, 1972. [Yad. Fiz.15,781(1972)].

[20] David J. Gross and Frank Wilczek. Ultraviolet Behavior of Nonabelian Gauge Theories. *Phys. Rev. Lett.*, 30:1343–1346, 1973. [,271(1973)].

[21] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R.* Springer Publishing Company, Incorporated, 2014.

[22] Henry W. Kendall. Deep inelastic scattering: experiments on the proton and the observation of scaling. *Nobel Lecture.*

[23] T. Kinoshita. Mass singularities of Feynman amplitudes. *J. Math. Phys.*, 3:650–677, 1962.

[24] T. D. Lee and M. Nauenberg. Degenerate Systems and Mass Singularities. *Phys. Rev.*, 133:B1549–B1562, 1964. [,25(1964)].

[25] C. Patrignani et al. Review of Particle Physics. *Chin. Phys.*, C40(10):100001, 2016.

[26] Michael E. Peskin and Daniel V. Schroeder. *An Introduction to quantum field theory.* Addison-Wesley, Reading, USA, 1995.

[27] Ringaile Placakyte. Parton distribution functions. In *Proceedings, 31st International Conference on Physics in collisions (PIC 2011): Vancouver, Canada, August 28-September 1, 2011*, 2011.

[28] H. David Politzer. Reliable Perturbative Results for Strong Interactions? *Phys. Rev. Lett.*, 30:1346–1349, 1973. [,274(1973)].

[29] J. Pumplin, D. R. Stump, J. Huston, H. L. Lai, Pavel M. Nadolsky, and W. K. Tung. New generation of parton distributions with uncertainties from global QCD analysis. *JHEP*, 07:012, 2002.

[30] T. Regge. Introduction to complex orbital momenta. *Il Nuovo Cimento (1955-1965)*, 14(5):951–976, Dec 1959.

[31] Gavin P. Salam and Juan Rojo. A Higher Order Perturbative Parton Evolution Toolkit (HOPPET). *Comput. Phys. Commun.*, 180:120–156, 2009.

[32] M. Wobisch, D. Britzger, T. Kluge, K. Rabbertz, and F. Stober. Theory-Data Comparisons for Jet Measurements in Hadron-Induced Processes. 2011.

[33] G. Zweig. An SU(3) model for strong interaction symmetry and its breaking. Version 2. In D.B. Lichtenberg and Simon Peter Rosen, editors, *DEVELOPMENTS IN THE QUARK THEORY OF HADRONS. VOL. 1. 1964 - 1978*, pages 22–101. 1964.