Hindawi Publishing Corporation
Advances in High Energy Physics
Volume 2014, Article ID 507690, 13 pages
http://dx.doi.org/10.1155/2014/507690

Research Article

High Performance Numerical Computing for High Energy Physics: A New Challenge for Big Data Science

Florin Pop

Computer Science Department, Faculty of Automatic Control and Computers, University Politehnica of Bucharest, Splaiul Independentei 313, Bucharest 060042, Romania

Correspondence should be addressed to Florin Pop; florin.pop@cs.pub.ro

Received 20 September 2013; Revised 23 December 2013; Accepted 26 December 2013

Academic Editor: Carlo Cattani

Modern physics is based on both theoretical analysis and experimental validation. Complex scenarios like subatomic dimensions, high energy, and lower absolute temperature are frontiers for many theoretical models. Simulation with stable numerical methods represents an excellent instrument for high accuracy analysis, experimental validation, and visualization. High performance computing support offers possibility to make simulations at large scale, in parallel, but the volume of data generated by these experiments creates a new challenge for Big Data Science. This paper presents existing computational methods for high energy physics (HEP) analyzed from two perspectives: numerical methods and high performance computing. The computational methods presented are Monte Carlo methods and simulations of HEP processes, Markovian Monte Carlo, unfolding methods in particle physics, kernel estimation in HEP, and Random Matrix Theory used in analysis of particles spectrum. All of these methods produce data-intensive applications, which introduce new challenges and requirements for ICT systems architecture, programming paradigms, and storage capabilities.

1. Introduction

High Energy Physics (HEP) experiments are probably the main consumers of High Performance Computing (HPC) in the area of e-Science, considering numerical methods in real experiments and assisted analysis using complex simulation. Starting with quarks discovery in the last century to Higgs Boson in 2012 [1], all HEP experiments were modeled using numerical algorithms: numerical integration, interpolation, random number generation, eigenvalues computation, and so forth. Data collection from HEP experiments generates a huge volume, with a high velocity, variety, and variability and passes the common upper bounds to be considered Big Data. The numerical experiments using HPC for HEP represent a new challenge for Big Data Science.

Theoretical research in HEP is related to matter (fundamental particles and Standard Model) and Universe formation basic knowledge. Beyond this, the practical research in HEP has led to the development of new analysis tools (synchrotron radiation, medical imaging or hybrid models [2], wavelets-computational aspects [3]), new processes (cancer therapy [4], food preservation, or nuclear waste treatment), or even the birth of a new industry (Internet) [5].

This paper analyzes two aspects: the computational methods used in HEP (Monte Carlo methods and simulations, Markovian Monte Carlo, unfolding methods in particle physics, kernel estimation, and Random Matrix Theory) and the challenges and requirements for ICT systems to deal with processing of Big Data generated by HEP experiments and simulations.

The motivation of using numerical methods in HEP simulations is based on special problems which can be formulated using integral or differential-integral equations (or systems of such equations), like quantum chromodynamics evolution of parton distributions inside a proton which can be described by the Gribov-Lipatov-Altarelli-Parisi (GLAP) equations [6], estimation of cross section for a typical HEP interaction (numerical integration problem), and data representation using histograms (numerical interpolation problem). Numerical methods used for solving differential

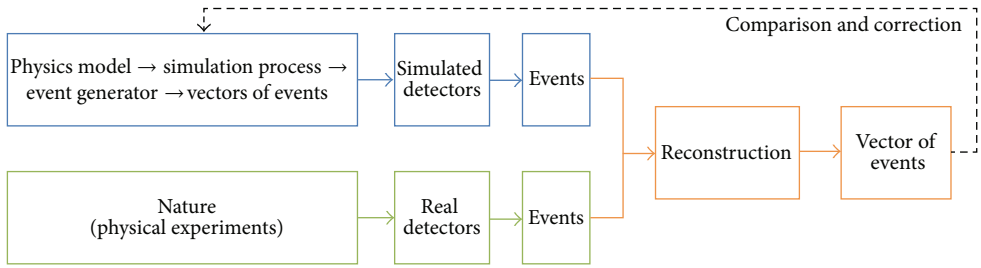

Figure 1: General approach of event generation, detection, and reconstruction.

equations or integrals are based on classical quadratures and Monte Carlo (MC) techniques. These allow generating events in terms of particle flavors and four-momenta, which is particularly useful for experimental applications. For example, MC techniques for solving the GLAP equations are based on simulated Markov chains (random walks), which have the advantage of filtering and smoothing the state vector for estimating parameters.

In practice, several MC event generators and simulation tools are used. For example, HERWIG (http://projects.hepforge.org/herwig/) project considers angular-ordered parton shower, cluster hadronization (the tool is implemented using Fortran), PYTHIA (http://www.thep.lu.se/torbjorn/Pythia.html) project is oriented on dipole-type parton shower and string hadronization (the tool is implemented in Fortran and C++), and SHERPA (http://projects.hepforge.org/sherpa/) considers dipole-type parton shower and cluster hadronization (the tool is implemented in C++). An important tool for MC simulations is GATE (GEANT4 Application for Tomographic Emission), a generic simulation platform based on GEANT4. GATE provides new features for nuclear imaging applications and includes specific modules that have been developed to meet specific requirements encountered in SPECT (Single Photon Emission Tomography) and PET (Positron Emission Tomography).

The main contributions of this paper are as follows:

(i) introduction and analysis of most important modeling methods used in High Energy Physics;

(ii) identifying and describing of the computational numerical methods for High Energy Physics;

(iii) presentation of the main challenges for Big Data processing.

The paper is structured as follows. Section 2 introduces the computational methods used in HEP and describes the performance evaluation of parallel numerical algorithms. Section 3 discusses the new challenge for Big Data Science generated by HEP and HPC. Section 4 presents the conclusions and general open issues.

2. Computational Methods Used in High Energy Physics

Computational methods are used in HEP in parallel with physical experiments to generate particle interactions that are modeled using vector of events. This section presents general approach of event generation, simulation methods based on Monte Carlo algorithms, Markovian Monte Carlo chains, methods that describe unfolding processes in particle physics, Random Matrix Theory as support for particle spectrum, and kernel estimation that produce continuous estimates of the parent distribution from the empirical probability density function. The section ends with performance analysis of parallel numerical algorithms used in HEP.

2.1. General Approach of Event Generation. The most important aspect in simulation for HEP experiments is event generation. This process can be split into multiple steps, according to physical models. For example, structure of LHC (Large Hadron Collider) events: (1) hard process; (2) parton shower; (3) hadronization; (4) underlying event. According to official LHC website (http://home.web.cern.ch/about/computing): "*approximately 600 million times per second, particles collide within the LHC. . .Experiments at CERN generate colossal amounts of data. The Data Centre stores it, and sends it around the world for analysis*." The analysis must produce valuable data and the simulation results must be correlated with physical experiments.

Figure 1 presents the general approach of event generation, detection, and reconstruction. The physical model is used to create simulation process that produces different type of events, clustered in vector of events (e.g., the fourth type of events in LHC experiments).

In parallel, the real experiments are performed. The detectors identify the most relevant events and, based on reconstruction techniques, vector of events is created. The detectors can be real or simulated (software tools) and the reconstruction phase combines real events with events detected in simulation. At the end, the final result is compared with the simulation model (especially with generated vectors of events). The model can be corrected for further experiments. The goal is to obtain high accuracy and precision of measured and processed data.

Software tools for event generation are based on random number generators. There are three types of random numbers: truly random numbers (from physical generators), pseudorandom numbers (from mathematical generators), and quasirandom numbers (special correlated sequences of numbers, used only for integration). For example, numerical integration using quasirandom numbers usually gives faster convergence than the standard integration methods based on

```
(1)  procedure RANDOM_GENERATOR_POISSON(μ)
(2)     number ← −1;
(3)     accumulator ← 1.0;
(4)     q ← exp{−μ};
(5)     while accumulator > q do
(6)        rnd_number ← RND() ;
(7)        accumulator ← accumulator ∗ rnd_number;
(8)        number ← number + 1;
(9)     end while
(10)    return number;
(11) end procedure
```

Algorithm 1: Random number generation for Poisson distribution using many random generated numbers with normal distribution (RND).

quadratures. In event generation pseudorandom numbers are used most often.

The most popular HEP application uses Poisson distribution combined with a basic normal distribution. The Poisson distribution can be formulated as

$$P[X = k] = \frac{\mu^k}{k!} \exp\{-\mu\}, \quad k = 0, 1, \ldots, \tag{1}$$

with $E(k) = V(k) = \mu$ (V is variance and E is expectation value). Having a uniform random number generator called RND() (Random based on Normal Distribution) we can use the following two algorithms for event generation techniques.

The result of running Algorithms 1 and 2 to generate around 10^6 random numbers is presented in Figure 2. In general, the second algorithm has better result for Poisson distribution. General recommendation for HEP experiments indicates the use of popular random number generators like TRNG (True Random Number Generators), RANMAR (Fast Uniform Random Number Generator used in CERN experiments), RANLUX (algorithm developed by Luscher used by Unix random number generators), and Mersenne Twister (the "industry standard"). Random number generators provided with compilers, operating system, and programming language libraries can have serious problem because they are based on system clock and suffer from lack of uniformity of distribution for large amounts of generated numbers and correlation of successive values.

The art of event generation is to use appropriate combinations of various random number generation methods in order to construct an efficient event generation algorithm being solution to a given problem in HEP.

2.2. Monte Carlo Simulation and Markovian Monte Carlo Chains in HEP. In general, a Monte Carlo (MC) method is any simulation technique that uses random numbers to solve a well-defined problem, P. If F is a solution of the problem P (e.g., $F \in R^n$ or F has a Boolean value), we define $\widehat{F}$, an estimation of F, as $\widehat{F} = f(\{r_1, r_2, \ldots, r_n\}, \ldots)$, where $\{r_i\}_{1 \le i \le n}$ is a random variable that can take more than one value and for which any value that will be taken cannot be predicted in advance. If $\rho(r)$ is the probability density function, $\rho(r)dr = P[r < r' < r + dr]$, the cumulative distributed function is

$$C(r) = \int_{-\infty}^{r} \rho(x)\, dx \Longrightarrow \rho(r) = \frac{dC(r)}{dr}. \tag{2}$$

$C(r)$ is a monotonically nondecreasing function with all values in $[0, 1]$. The expectation value is

$$E(f) = \int f(r)\, dC(r) = \int f(r)\, \rho(r)\, dr. \tag{3}$$

And the variance is

$$V(f) = E[f - E(f)]^2 = E\left(f^2\right) - E^2(f). \tag{4}$$

2.2.1. Monte Carlo Event Generation and Simulation. To define a MC estimator the "Law of Large Numbers (LLN)" is used. LLN can be described as follows: let one choose n

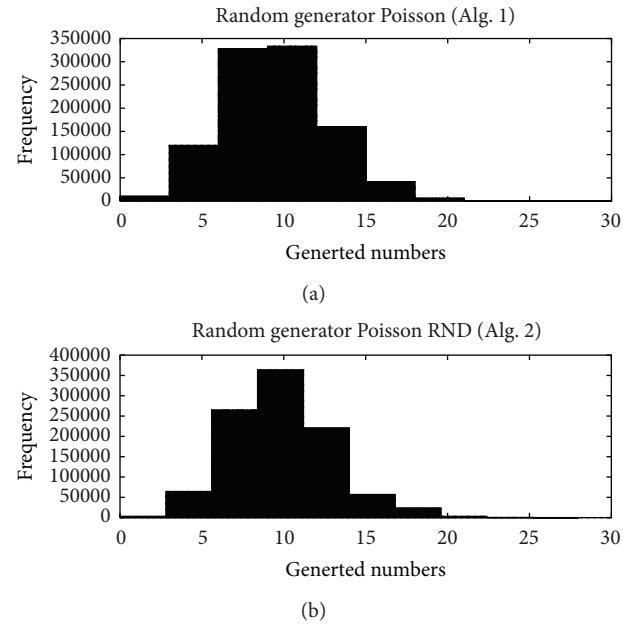

Figure 2: General approach of event generation, detection, and reconstruction.

```
(1)  procedure RANDOM_GENERATOR_POISSON_RND(μ, r)
(2)      number ← 0;
(3)      q ← exp{−μ};
(4)      accumulator ← q;
(5)      p ← q;
(6)      while r > accumulator do
(7)          number ← number + 1;
(8)          p ← p ∗ μ/number;
(9)          accumulator ← accumulator + p;
(10)     end while
(11)     return number;
(12) end procedure
```

Algorithm 2: Random number generation for Poisson distribution using one random generated number with normal distribution.

numbers r_i randomly, with the probability density function uniform on a specific interval (a, b), each r_i being used to evaluate $f(r_i)$. For large n (consistent estimator),

$$\frac{1}{n}\sum_{i=1}^{n} f(r_i) \longrightarrow E(f) = \frac{1}{b-a}\int_a^b f(r)\,dr. \tag{5}$$

The properties of a MC estimator are being normally distributed (with Gaussian density); the standard deviation is $\sigma = \sqrt{V(f)/n}$; MC is unbiased for all n (the expectation value is the real value of the integral); the estimator is consistent if $V(f) < \infty$ (the estimator converges to the true value of the integral for every large n); a sampling phase can be applied to compute the estimator if we do not know anything about the function f; it is just suitable for integration. The sampling phase can be expressed, in a stratified way, as

$$\int_a^b f(r)\,dr = \int_a^{r_1} f(r)\,dr + \int_{r_1}^{r_2} f(r)\,dr + \cdots + \int_{r_n}^{b} f(r)\,dr. \tag{6}$$

MC estimations and MC event generators are necessary tools in most of HEP experiments being used at all their steps: experiments preparation, simulation running, and data analysis.

An example of MC estimation is the Lorentz invariant phase space (LIPS) that describes the cross section for a typical HEP process with n particle in the final state.

Consider

$$\sigma_n \sim \int |M|^2 dR_n, \tag{7}$$

where M is the matrix describing the interaction between particles and dR_n is the element of LIPS. We have the following estimation:

$$R_n(P, p_1, p_2, \ldots, p_n) = \int \delta^{(4)}\left(P - \sum_{k=1}^{n} p_k\right)\prod_{k=1}^{n}\left(\delta\left(p_k^2 - m_k^2\right)\Theta\left(p_k^0\right)d^4p_k\right), \tag{8}$$

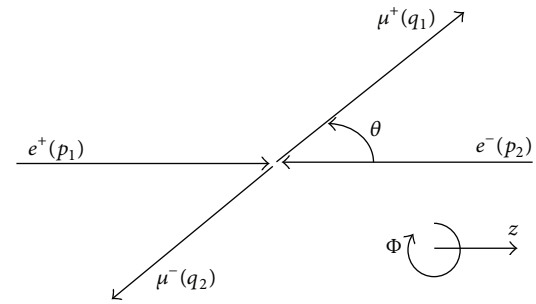

Figure 3: Example of particle interaction: $e(p_1) + e^-(p_2) \rightarrow \mu^+(q_1)\mu^-(q_2)$.

where P is total four-momentum of the n-particle system; p_k and m_k are four-momenta and mass of the final state particles; $\delta^{(4)}(P - \sum_{k=1}^{n} p_k)$ is the total energy momentum conservation; $\delta(p_k^2 - m_k^2)$ is the on-mass-shell condition for the final state system. Based on the integration formula

$$\int \delta\left(p_k^2 - m_k^2\right)\Theta\left(p_k^0\right)d^4p_k = \frac{d^3p_k}{2p_k^0}, \tag{9}$$

obtain the iterative form for cross section:

$$R_n(P, p_1, p_2, \ldots, p_n) = \int R_{n-1}(P - p_n, p_1, p_2, \ldots, p_{n-1})\,\frac{d^3p_n}{2p_n^0}, \tag{10}$$

which can be numerical integrated by using the recurrence relation. As result, we can construct a general MC algorithm for particle collision processes.

Example 1. Let us consider the interaction: $e^+e^- \rightarrow \mu^+\mu^-$ where Higgs boson contribution is numerically negligible. Figure 3 describes this interaction (Φ is the azimuthal angle, θ the polar angle, and p_1, p_2, q_1, q_2 are the four-momenta for particles).

The cross section is

$$d\sigma = \frac{\alpha^2}{4s}\left[W_1(s)\left(1 + \cos^2\theta\right) + W_2(s)\cos\theta\right]d\Omega, \tag{11}$$

where $d\Omega = d\cos\theta d\Phi$, $\alpha = e^2/4\pi$ (fine structure constant), $s = (p_1^0 + p_2^0)^2$ is the center of mass energy squared, and $W_1(s)$ and $W_2(s)$ are constant functions. For pure processes we have $W_1(s) = 1$ and $W_2(s) = 0$, and the total cross section becomes

$$\sigma = \int_0^{2\pi} d\Phi \int_{-1}^{1} d\cos\theta \frac{d^2\sigma}{d\Phi d\cos\theta}. \tag{12}$$

We introduce the following notation:

$$\rho(\cos\theta, \Phi) = \frac{d^2\sigma}{d\Phi d\cos\theta}, \tag{13}$$

and let us consider $\widetilde{\rho}(\cos\theta, \Phi)$ an approximation of $\rho(\cos\theta, \Phi)$. Then $\widetilde{\sigma} = \iint d\Phi d\cos\theta \widetilde{\rho}$. Now, we can compute

$$\begin{aligned}\sigma &= \int_0^{2\pi} d\Phi \int_{-1}^{1} d\cos\,\theta \rho(\cos\,\theta, \Phi) \\ &= \int_0^{2\pi} d\Phi \int_{-1}^{1} d\cos\,\theta w(\cos\,\theta, \Phi)\,\widetilde{\rho}(\cos\,\theta, \Phi) \\ &\approx \langle w\rangle_{\widetilde{\rho}} \int_0^{2\pi} d\Phi \int_{-1}^{1} d\cos\,\theta \widetilde{\rho}(\cos\,\theta, \Phi) = \widetilde{\sigma}\langle w\rangle_{\widetilde{\rho}},\end{aligned} \tag{14}$$

where $w(\cos\,\theta, \Phi) = \rho(\cos\,\theta, \Phi)/\widetilde{\rho}(\cos\,\theta, \Phi)$ and $\langle w\rangle_{\widetilde{\rho}}$ is the estimation of w based on $\widetilde{\rho}$. Here, the MC estimator is

$$\langle w\rangle_{\mathrm{MC}} = \frac{1}{n}\sum_{i=1}^{n} w_i, \tag{15}$$

and the standard deviation is

$$s_{\mathrm{MC}} = \left(\frac{1}{n(n-1)}\sum_{i=1}^{n}\left(w_i - \langle w\rangle_{\mathrm{MC}}\right)^2\right)^{1/2}. \tag{16}$$

The final numerical result based on MC estimator is

$$\sigma_{\mathrm{MC}} = \widetilde{\sigma}\langle w\rangle_{\mathrm{MC}} \pm \widetilde{\sigma} s_{\mathrm{MC}}. \tag{17}$$

As we can show, the principle of a Monte Carlo estimator in physics is to simulate the cross section in interaction and radiation transport knowing the probability distributions (or an approximation) governing each interaction of elementary particles.

Based on this result, the Monte Carlo algorithm used to generate events is as follows. It takes as input $\widetilde{\rho}(\cos\,\theta, \Phi)$ and in a main loop considers the following steps: (1) generate $(\cos\,\theta, \Phi)$ peer from $\widetilde{\rho}$; (2) compute four-momenta p_1, p_2, q_1, q_2; (3) compute $w = \rho/\widetilde{\rho}$. The loop can be stopped in the case of unweighted events, and we will stay in the loop for weighted events. As output, the algorithm returns four-momenta for particle for weighted events and four-momenta and an array of weights for unweighted events. The main issue is how to initialize the input of the algorithm. Based on $d\sigma$ formula (for $W_1(s) = 1$ and $W_2(s) = 0$), we can consider as input $\widetilde{\rho}(\cos\,\theta, \Phi) = (\alpha^2/4s)(1 + \cos^2\theta)$. Then $\widetilde{\sigma} = 4\pi\alpha^2/3s$.

In HEP theoretical predictions used for particle collision processes modeling (as shown in presented example) should be provided in terms of Monte Carlo event generators, which directly simulate these processes and can provide unweighted (weight = 1) events. A good Monte Carlo algorithm should be used not only for numerical integration [7] (i.e., provide weighted events) but also for efficient generation of unweighted events, which is very important issue for HEP.

2.2.2. Markovian Monte-Carlo Chains. A classical Monte Carlo method estimates a function F with $\widehat{F}$ by using a random variable. The main problem with this approach is that we cannot predict any value in advance for a random variable. In HEP simulation experiments the systems are described in states [8]. Let us consider a system with a finite set of possible states $S_1, S_2, \ldots$, and S_t the state at the moment t. The conditional probability is defined as

$$P\left(S_t = S_j \mid S_{t_1} = S_{i_1}, S_{t_2} = S_{i_2}, \ldots, S_{t_n} = S_{i_n}\right), \tag{18}$$

where the mappings $(t_1, i_1), \ldots, (t_n, i_n)$ can be interpreted as the description of system evolution in time by specifying a specific state for each moment of time.

The system is a Markov chain if the distribution of S_t depends only on immediate predecessor S_{t-1} and it is independent of all previous states as follows:

$$\begin{aligned}&P\left(S_t = S_j \mid S_{t-1} = S_{i_{t-1}}, \ldots, S_{t_2} = S_{i_2}, S_{t_1} = S_{i_1}\right) \\ &\quad = P\left(S_t = S_j \mid S_{t-1} = S_{i_{t-1}}\right).\end{aligned} \tag{19}$$

To generate the time steps $(t_1, t_2, \ldots, t_n)$ we use the probability of a single forward Markovian step given by $p(t \mid t_n)$ with the property $\int_{t_n}^{\infty} p(t \mid t_n)dt = 1$ and we define $p(t) = p(t \mid 0)$. The 1-dimensional Monte Carlo Markovian Algorithm used to generate the time steps is presented in Algorithm 3.

The main result of Algorithm 3 is that $P(t_{\max})$ follows a Poisson distribution:

$$\begin{aligned}P_N &= \int_0^{t_{\max}} p\left(t_1 \mid t_0\right) dt_1 \times \int_{t_1}^{t_{\max}} p\left(t_2 \mid t_1\right) dt_2 \\ &\quad \times \cdots \times \int_{t_{N-1}}^{t_{\max}} p\left(t_N \mid t_{N-1}\right) dt_N \\ &\quad \times \int_{t_{\max}}^{\infty} p\left(t_{N+1} \mid t_N\right) dt_{N+1} = \frac{1}{N!}\left(t_{\max}\right)^N e^{-t_{\max}}.\end{aligned} \tag{20}$$

We can consider the 1-dimensional Monte Carlo Markovian Algorithm as a method used to iteratively generate the systems' states (codified as a Markov chain) in simulation experiments. According to the Ergodic Theorem for Markov chains, the chain defined has a unique stationary probability distribution [9, 10].

Figures 4 and 5 present the running of Algorithm 3. According to different values of parameter s used to generate the next step, the results are very different, for 1000 iterations. Figure 4 for $s = 1$ shows a profile of the type of noise. For $s = 10, 100, 1000$ profile looks like some of the information is filtered and lost. The best results are obtained for $s = 0.01$

(1) Generate t_1 according with $p(t_1) = p(t_1 \mid t_0 = 0)$
(2) **if** $t_1 < t_{\max}$ **then** ▷ Generate the initial state.
(3) $P_{N\geq 1} = \int_0^{t_{\max}} p(t_1 \mid t_0)\, dt_1$; ▷ Compute the initial probability.
(4) **Retain** t_1;
(5) **end if**
(6) **if** $t_1 > t_{\max}$ **then** ▷ Discard all generated and computed data.
(7) $N = 0$; $P_0 = \int_{t_{\max}}^{\infty} p(t_1 \mid t_0)\, dt_1 = e^{-t_{\max}}$;
(8) **Delete** t_1;
(9) **EXIT**. ▷ The algorithm ends here.
(10) **end if**
(11) $i = 2$;
(12) **while** (1) **do** ▷ Infinite loop until a successful EXIT.
(13) Generate t_i according with $p(t_i \mid t_{i-1})$
(14) **if** $t_i < t_{\max}$ **then** ▷ Generate a new state and new probability.
(15) $P_{N\geq i} = \int_{t_i}^{t_{\max}} p(t_i \mid t_{i-1})\, dt_i$;
(16) **Retain** t_i;
(17) **end if**
(18) **if** $t_i > t_{\max}$ **then** ▷ Discard all generated and computed data.
(19) $N = i - 1$; $P_i = \int_{t_{\max}}^{\infty} p(t_i \mid t_{i-1})\, dt_i$;
(20) **Retain** $(t_1, t_2, \ldots, t_{i-1})$; **Delete** t_i;
(21) **EXIT**. ▷ The algorithm ends here.
(22) **end if**
(23) $i = i + 1$;
(24) **end while**

Algorithm 3: 1-Dimensional Monte Carlo Markovian Algorithm.

and $s = 0.1$ and the generated values can be easily accepted for MC simulation in HEP experiments.

Figure 5 shows the acceptance rate of values generated with parameter s used in the algorithm. And parameter values are correlated with Figure 4. Results in Figure 5 show that the acceptance rate decreases rapidly with increasing value of parameter s. The conclusion is that values must be kept small to obtain meaningful data. A correlation with the normal distribution is evident, showing that a small value for the mean square deviation provides useful results.

2.2.3. Performance of Numerical Algorithms Used in MC Simulations. Numerical methods used to compute MC estimator use numerical quadratures to approximate the value of the integral for function f on a specific domain by a linear compilation of function values and weights $\{w_i\}_{1\leq i\leq m}$ as follows:

$$\int_a^b f(r)\, dr = \sum_{i=1}^{m} w_i f(r_i). \tag{21}$$

We can consider a consistent MC estimator a a classical numerical quadrature with all $w_i = 1$. Efficiency of integration methods for 1 dimension and for d dimensions is presented in Table 1. We can conclude that quadrature methods are difficult to apply in many dimensions for variate integration domains (regions) and the integral is not easy to be estimated.

Table 1: Efficiency of integration methods for 1 dimension and for d dimensions.

Method	1 dimension	d dimensions
Monte Carlo	$n^{-1/2}$	$n^{-1/2}$
Trapezoidal rule	n^{-2}	$n^{-2/d}$
Simpson's rule	n^{-4}	$n^{-4/d}$
m-points Gauss rule ($m < n$)	n^{-2m}	$n^{-2m/d}$

As practical example, in a typical high-energy particle collision there can be many final-state particles (even hundreds). If we have n final state particle, we face with $d = 3n - 4$ dimensional phase space. As numerical example, for $n = 4$ we have $d = 8$ dimensions, which is very difficult approach for classical numerical quadratures.

Full decomposition integration volume for one double number (10 Bytes) per volume unit is $n^d \times 10$ Bytes. For the example considered with $d = 8$ and $n = 10$ divisions for interval $[0, 1]$ we have, for one numerical integration,

$$n^d \times 10\,\text{Bytes} = \frac{10^8 \times 10}{1024^3}\, G\,\text{Bytes} \approx 0.93\, G\,\text{Bytes}. \tag{22}$$

Considering 10^6 events per second, one integration per event, the data produced in one hour will be ≈3197.4 P Bytes.

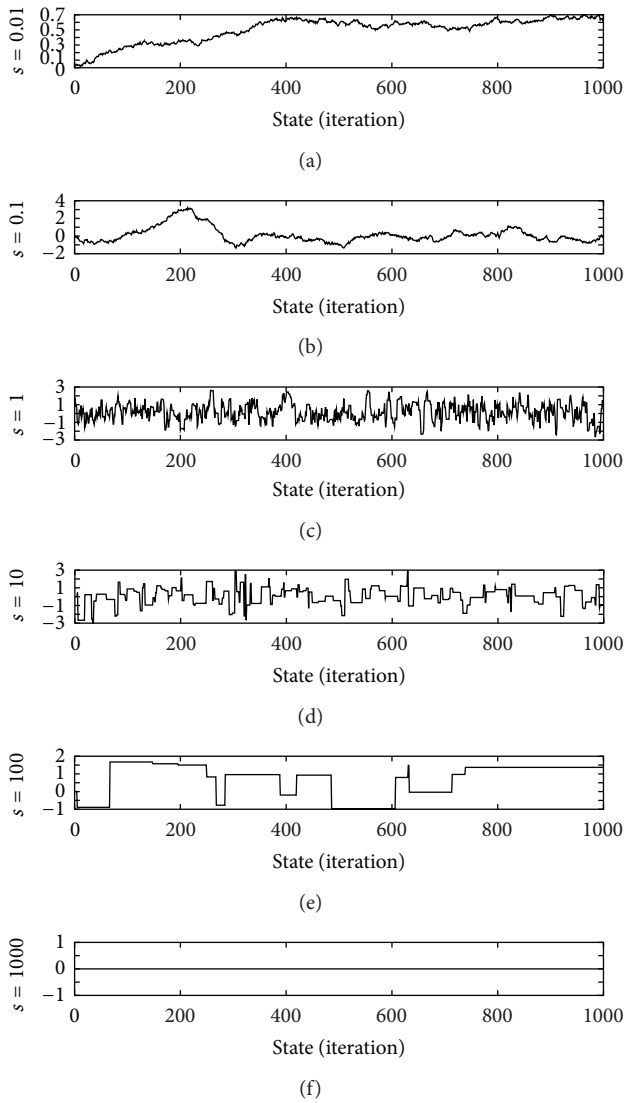

Figure 4: Example of 1-dimensional Monte Carlo Markovian algorithm.

The previous assumption is only for multidimensional arrays. But due to the factorization assumption, $p(r_1, r_2, \ldots, r_n) = p(r_1)p(r_2)\cdots p(r_n)$, we obtain for one integration

$$n \times d \times 10 \text{ Bytes} = 800 \text{ Bytes}, \tag{23}$$

which means ≈2.62 T Bytes of data produce for one hour of simulations.

2.3. Unfolding Processes in Particle Physics and Kernel Estimation in HEP. In particle physics analysis we have two types of distributions: true distribution (considered in theoretical models) and measured distribution (considered in experimental models, which are affected by finite resolution and limited acceptance of existing detectors). A HEP interaction process starts with a true knows distribution and generate a measured distribution, corresponding to an experiment of a well-confirmed theory. An inverse process starts with a measured distribution and tries to identify the true distribution. These unfolding processes are used to identify new theories based on experiments [11].

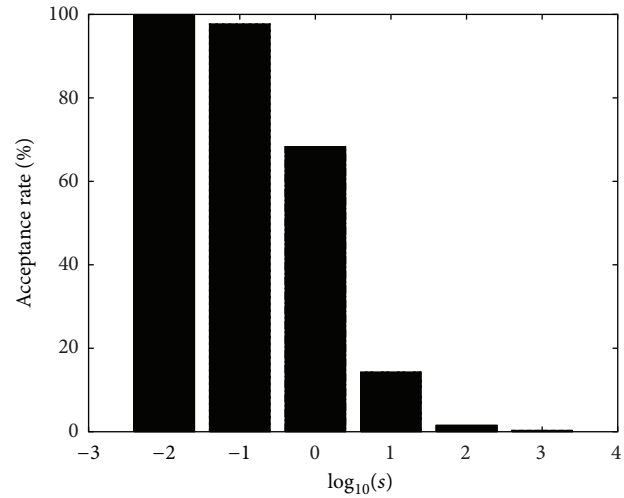

Figure 5: Analysis of acceptance rate for 1-dimensional Monte Carlo Markovian algorithm for different s values.

2.3.1. Unfolding Processes in Particle Physics. The theory of unfolding processes in particle physics is as follows [12]. For a physics variable t we have a true distribution $f(t)$ mapped in x and an n-vector of unknowns and a measured distribution $g(s)$ (for a measured variable s) mapped in an m-vector of measured data. A response matrix $A \in R^{m \times n}$ encodes a Kernel function $K(s,t)$ describing the physical measurement process [12–15]. The direct and inverse processes are described by the Fredholm integral equation [16] of the first kind, for a specific domain Ω,

$$\int_{\Omega} K(s,t)\, f(t)\, dt = g(s). \tag{24}$$

In particle physics the Kernel function $K(s,t)$ is usually known from a Monte Carlo sample obtained from simulation. A numerical solution is obtained using the following linear equation: $Ax = b$. Vectors x and y are assumed to be 1-dimensional in theory, but they can be multidimensional in practice (considering multiple independent linear equations). In practice, also the statistical properties of the measurements are well known and often they follow the Poisson statistics [17]. To solve the linear systems we have different numerical methods.

First method is based on linear transformation $x = A^{\#} y$. If $m = n$ then $A^{\#} = A^{-1}$ and we can use direct Gaussian methods, iterative methods (Gauss-Siedel, Jacobi or SOR), or orthogonal methods (based on Householder transformation, Givens methods, or Gram-Schmidt algorithm). If $m > n$ (the most frequent scenario) we will construct the matrix $A^{\#} = (A^T A)^{-1} A^T$ (called pseudoinverse Penrose-Moore). In these cases the orthogonal methods offer very good and stable numerical solutions.

Second method considers the singular value decomposition:

$$A = U\Sigma V^T = \sum_{i=1}^{n} \sigma_i u_i v_i^T, \quad (25)$$

where $U \in R^{m\times n}$ and $V \in R^{n\times n}$ are matrices with orthonormal columns and the diagonal matrix $\Sigma = \text{diag}\{\sigma_1, \ldots, \sigma_n\} = U^T AV$. The solution is

$$x = A^{\#} y = V\Sigma^{-1}\left(U^T y\right) = \sum_{i=1}^{n} \frac{1}{\sigma_i}\left(u_i^T y\right) v_i = \sum_{i=1}^{n} \frac{1}{\sigma_i} c_i v_i, \quad (26)$$

where $c_i = u_i^T y$, $i = 1, \ldots, n$, are called Fourier coefficients.

2.3.2. Random Matrix Theory. Analysis of particle spectrum (e.g., neutrino spectrum) faces with Random Matrix Theory (RMT), especially if we consider anarchic neutrino masses. The RMT means the study of the statistical properties of eigenvalues of very large matrices [18]. For an interaction matrix A (with size N), where A_{ij} is an independent distributed random variable and A^H is the complex conjugate and transpose matrix, we define $M = A + A^H$, which describes a Gaussian Unitary Ensemble (GUE). The GUE properties are described by the probability distribution $P(M)dM$: (1) it is invariant under unitary transformation, $P(M)dm = P(M')dM'$, where $M' = U^H MU$, U is a Hermitian matrix ($U^H U = I$); (2) the elements of M matrix are statistically independent, $P(M) = \prod_{i\le j} P_{ij}(M_{ij})$; and (3) the matrix M can be diagonalized as $M = UDU^H$, where $U = \text{diag}\{\lambda_1, \ldots, \lambda_N\}$, λ_i is the eigenvalue of M and $\lambda_i \le \lambda_j$ if $i < j$

$$\text{Propability (2): } P(M)\,dM \sim dM \exp\left\{-\frac{N}{2} Tr\left(M^H M\right)\right\};$$

$$\text{Propability (3): } P(M)\,dM \sim dU \prod_i d\lambda_i \prod_{i<j} \left(\lambda_i - \lambda_j\right)^2 \times \exp\left\{-\frac{N}{2}\sum_i \left(\lambda_i^2\right)\right\}. \quad (27)$$

The numerical methods used for eigenvalues computation are the QR method and Power methods (direct and indirect). The QR method is a numerical stable algorithm and Power method is an iterative one. The RMT can be used for many body systems, quantum chaos, disordered systems, quantum chromodynamics, and so forth.

2.3.3. Kernel Estimation in HEP. Kernel estimation is a very powerful solution and relevant method for HEP when it is necessary to combine data from heterogeneous sources like MC datasets obtained by simulation and from Standard Model expectation, obtained from real experiments [19]. For a set of data $\{x_i\}_{1\le i\le n}$ with a constant bandwidth h (the difference between two consecutive data values), called the smoothing parameter, we have the estimation

$$\widehat{f}(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right), \quad (28)$$

where K is an estimator. For example, a Gauss estimator with mean μ and standard deviation σ is

$$K(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}, \quad (29)$$

and has the following properties: positive definite and infinitely differentiable (due to the exp function), and it can be defined for an infinite supports ($n \to \infty$). The kernel is a nonparametric method, which means that h is independent of dataset and for large amount of normally distributed data we can find a value for h that minimizes the integrated squared error of $\widehat{f}(x)$. This value for bandwidth is computed as

$$h^* = \left(\frac{4}{3n}\right)^{1/5} \sigma. \quad (30)$$

The main problem in Kernel Estimation is that the set of data $\{x_i\}_{1\le i\le n}$ is not normally distributed and in real experiments the optimal bandwidth it is not known. An improvement of presented method considers adaptive Kernel Estimation proposed by Abramson [20], where $h_i = h/\sqrt{f(x_i)}$ and σ are considered global qualities for dataset. The new form is

$$\widehat{f}_a(x) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{h_i} K\left(\frac{x - x_i}{h_i}\right), \quad (31)$$

and the local bandwidth value that minimizes the integrated squared error of $\widehat{f}_a(x)$ is

$$h_i^* = \left(\frac{4}{3n}\right)^{1/5} \sqrt{\frac{\sigma}{\widehat{f}(x_i)}}, \quad (32)$$

where $\widehat{f}$ is the normal estimator.

Kernel estimation is used for event selection to confidence level evaluation, for example, in Markovian Monte Carlo chains or in selection of neural network output used in experiments for reconstructed Higgs mass. In general, the main usage of Kernel estimation in HEP is searching for new particle, by finding relevant data in a large dataset.

A method based on Kernel estimation is the graphical representation of datasets using advanced shifted histogram algorithm (ASH). This is a numerical interpolation for large datasets with the main aim of creating a set of *nbin* histograms $H = \{H_i\}$, with the same bin-width h. Algorithm 4 presents the steps of histograms generation starting with a specific interval $[a, b]$, a number of points n in this interval, and a number of bins and a number of values used for kernel estimation, m. Figure 6 shows the results of kernel estimation if function $f = -(1/2)x^2$ on $[0, 1]$ and graphical representation with a different number of bins. The values on vertical axis are aggregated in step 17 of Algorithm 4 and increase with the number of bins.

2.3.4. Performance of Numerical Algorithms Used in Particle Physics. All operations used in presented methods for particle physics (Unfolding Processes, Random Matrix Theory,

```
(1)  procedure ASH (a, b, n, x, nbin, m, f_m)
(2)    δ = (b − a) /nbin; h = mδ;
(3)    for k = 1 ... nbin do
(4)      v_k = 0; y_k = 0;
(5)    end for
(6)    for i = 1 ... n do
(7)      k = (x_i − a) /δ + 1;
(8)      if k ∈ [1, nbin] then
(9)        v_k = v_k + 1;
(10)     end if
(11)   end for
(12)   for k = 1 ... nbin do
(13)     if v_k = 0 then
(14)       k = k + 1;
(15)     end if
(16)     for i = max {1, k − m + 1} ... min {nbin, k + m − 1} do
(17)       y_i = y_i + v_k f_m (i − k);
(18)     end for
(19)   end for
(20)   for k = 1 ... nbin do
(21)     y_k = y_k/(nh)  ;
(22)     t_k = a + (k − 0.5) δ;
(23)   end for
(24)   return {t_k}_{1≤k≤nbin}, {y_k}_{1≤k≤nbin}.
(25) end procedure
```

Algorithm 4: Advanced shifted histogram (1D algorithm).

and Kernel Estimation) can be reduced to scalar products, matrix-vector products, and matrix-matrix products. In [21] the design of new standard for the BLAS (Basic Linear Algebra Subroutines) in C language by extension of precision is described. This permits higher internal precision and mixed input/output types. The precision allows implementation of some algorithms that are simpler, more accurate, and sometimes faster than possible without these features. Regarding the precision of numerical computing, Dongarra and Langou established in [22] an upper bound for the residual check for $Ax = y$ system, with $A \in R^{n\times n}$ a dense matrix. The residual check is defined as

$$r_\infty = \frac{\|Ax - y\|_\infty}{n\epsilon\left(\|A\|_\infty \|x\|_\infty + \|y\|_\infty\right)} < 16, \tag{33}$$

where ϵ is the relative machine precision for the IEEE representation standard; $\|y\|_\infty$ is the infinite norm of a vector: $\|y\|_\infty = \max_{1\le i\le n}\{|y_i|\}$; and $\|A\|_\infty$ is the infinite norm of a matrix $\|A\|_\infty = \max_{1\le i\le n}\{\sum_{j=1}^{n} |A_{ij}|\}$.

Figure 7 presents the graphical representation of Dongarras result (using logarithmic scales) for simple and double precision. For simple precision, $\epsilon_s = 2^{-24}$, for all $n \ge 1.05 \times 10^6$ the residual check is always lower than imposed upper bound, similarly for double precision with $\epsilon_d = 2^{-53}$, for all $n \ge 5.63 \times 10^{14}$. If matrix size is greater than these values, it will not be possible to detect if the solution is correct or not. These results establish upper bounds for data volume in this model.

In a single-processor system, the complexity of algorithms depends only on the problem size, n. We can assume $T(n) = \Theta(f(n))$, where $f(n)$ is a fundamental function ($f(n) \in \{1, n^\alpha, a^n, \log n, \sqrt{n}, \ldots\}$). In parallel systems (multiprocessor systems, with p processors) we have the serial processing time $T^*(n) = T_1(n)$ and parallel processing time $T_p(n)$. The performance of parallel algorithms can be analyzed using speed-up, efficiency, and isoefficiency metrics.

(i) The *speed-up*, $S(p)$, represents how a parallel algorithm is faster than a corresponding sequential algorithm. The speed-up is defined as $S(p) = T_1(n)/T_p(n)$. There are special bounds for speed-up [23]: $S(p) \le p\tilde{p}/(p + \tilde{p} - 1)$, where $\tilde{p} = T_1/T_\infty$ is the average parallelism (the average number of busy processors given unbounded number of processors). Usually $S(p) \le p$, but under special circumstances the speed-up can be $S(p) > p$ [24]. Another upper bound is established by the Amdahls law: $S(p) = (s + ((1 - s)/p))^{1/2} \le 1/s$ where s is the fraction of a program that is sequential. The upper bound is considered for a 0 time of parallel fraction.

(ii) The *efficiency* is the average utilization of p processors: $E(p) = S(p)/p$.

(iii) The *isoefficiency* is the growth rate of workload $W_p(n) = pT_p(n)$ in terms of number of processors to keep efficiency fixed. If we consider $W_1(n) - EW_p(n) = 0$ for any fixed efficiency E we obtain $p = p(n)$. This means that we can establish a relation between needed number of processors and problem size. For example for the parallel sum of n numbers using p processors we have $n \approx E(n + p\log p)$, so $n = \Theta(p\log p)$.

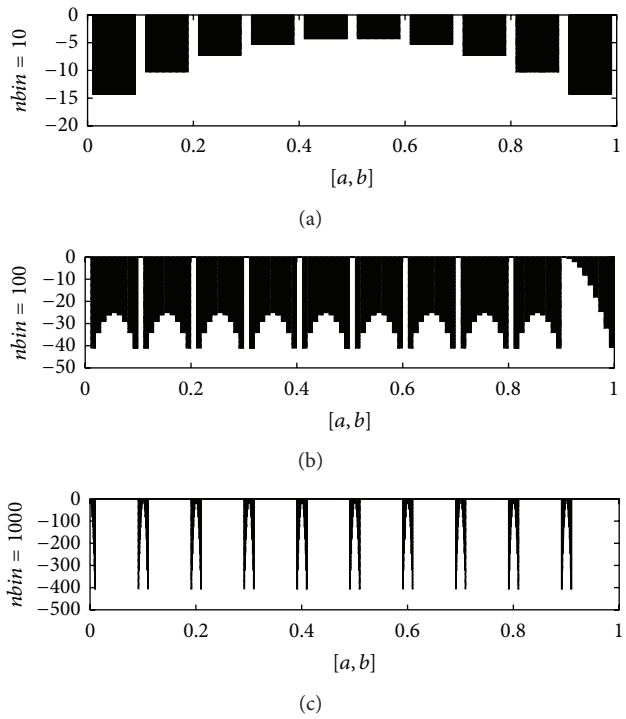

Figure 6: Example of advanced shifted histogram algorithm running for different bins: 10, 100, and 1000.

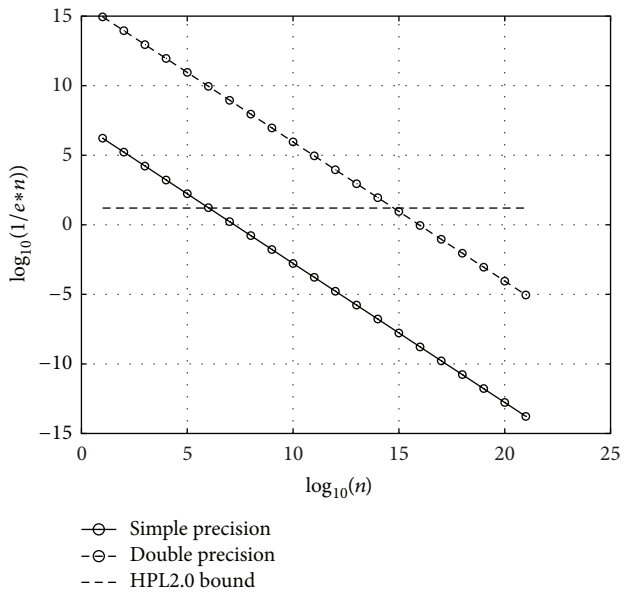

Figure 7: Residual check analysis for solving $Ax = y$ system in HPL2.0 using simple and double precision representation.

Numerical algorithms use for implementation a hypercube architecture. We analyze the performance of different numerical operations using the isoefficiency metric. For the hypercube architecture a simple model for intertask communication considers $T_{\text{com}} = t_s + Lt_w$ where t_s is the latency (the time needed by a message to cross through the network), t_w is the time needed to send a word ($1/t_w$ is called bandwidth), and L is the message length (expressed in number of words). The word size depends on processing architecture (usually it is two bytes). We define t_c as the processing time per word for a processor. We have the following results.

Table 2: Isoefficiency for a hypercube architecture: $n = \Theta(p \log p)$ and $n = \Theta(p(\log p)^2)$. We marked with $(*)$ the limitations imposed by Formula (33).

Scenario	Architecture size (p)	$n = \Theta(p \log p)$	$n = \Theta(p(\log p)^2)$
1	10^1	1.0×10^1	1.00×10^1
2	10^2	2.0×10^2	8.00×10^2
3	10^3	3.0×10^3	2.70×10^4
4	10^4	4.0×10^4	$6.40 \times 10^{5*}$
5	10^5	$5.0 \times 10^{5*}$	1.25×10^7
6	10^6	6.0×10^6	2.16×10^8
7	10^7	7.0×10^7	3.43×10^9
8	10^8	8.0×10^8	5.12×10^{10}
9	10^9	9.0×10^9	7.29×10^{11}

(i) *External product* xy^T. The isoefficiency is written as

$$t_c n \approx E\left(t_c n + \left(t_s + t_w\right) p \log p\right) \Longrightarrow n = \Theta\left(p \log p\right). \quad (34)$$

Parallel processing time is $T_p = t_c n/p + (t_s + t_w) \log p$. The optimality is computed using

$$\frac{dT_p}{dp} = 0 \Longrightarrow -t_c \frac{n}{p^2} + \frac{t_s + t_w}{p} = 0 \Longrightarrow p \approx \frac{t_c n}{t_s + t_w}. \quad (35)$$

(ii) *Scalar product* (internal product) $x^T y = \sum_{i=1}^{n} x_i y_i$. The isoefficiency is written as

$$t_c n^2 \approx E\left(t_c n^2 + \frac{t_s}{2} p \log p + \frac{t_w}{2} n\sqrt{p} \log p\right) \Longrightarrow n = \Theta\left(p(\log p)^2\right). \quad (36)$$

(iii) *Matrix-vector product* $y = Ax$, $y_i = \sum_{j=1}^{n} A_{ij} x_j$. The isoefficiency is written as

$$t_c n^2 \approx E\left(t_c n^2 + t_s p \log p + t_w n\sqrt{p} \log p\right) \Longrightarrow n = \Theta\left(p(\log p)^2\right). \quad (37)$$

Table 2 presented the amount of data that can be processed for a specific size. The cases that meet the upper bound $n \geq 1.05 \times 10^6$ are marked with $(*)$. To keep the efficiency high for a specific parallel architecture, HPC algorithms for particle physics introduce upper limits for the amount of data, which means that we have also an upper bound for Big Data volume in this case.

The factors that determine the efficiency of parallel algorithms are *task balancing* (work-load distribution between all used processors in a system $\rightarrow$ to be maximized); *concurrency* (the number/percentage of processors working simultaneously $\rightarrow$ to be maximized); and *overhead* (extra work for introduce by parallel processing that does not appear in serial processing $\rightarrow$ to be minimized).

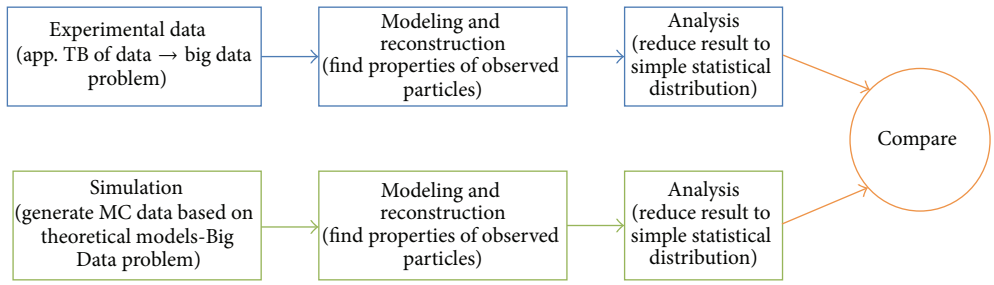

Figure 8: Processing flows for HEP experiments.

3. New Challenges for Big Data Science

There are a lot of applications that generate Big Data, like social networking profiles, social influence, SaaS & Cloud Apps, public web information, MapReduce scientific experiments and simulations (especially HEP simulations), data warehouse, monitoring technologies, and e-government services. Data grow rapidly, since applications produce continuously increasing volumes of both unstructured and structured data. The impact on the approach to data processing, transfer, and storage is the need to reevaluate the way and solutions to better answer the users' needs [25]. In this context, scheduling models and algorithms for data processing have an important role becoming a new challenge for Big Data Science.

HEP applications consider both experimental data (that are application with TB of valuable data) and simulation data (with data generated using MC based on theoretical models). The processing phase is represented by modeling and reconstruction in order to find properties of observed particles (see Figure 8). Then, the data are analyzed a reduced to a simple statistical distribution. The comparison of results obtained will validate how realistic is a simulation experiment and validate it for use in other new models.

Since we face a large variety of solutions for specific applications and platforms, a thorough and systematic analysis of existing solutions for scheduling models, methods, and algorithms used in Big Data processing and storage environments is needed. The challenges for scheduling impose specific requirements in distributed systems: the claims of the resource consumers, the restrictions imposed by resource owners, the need to continuously adapt to changes of resources' availability, and so forth. We will pay special attention to Cloud Systems and HPC clusters (datacenters) as reliable solutions for Big Data [26]. Based on these requirements, a number of challenging issues are maximization of system throughput, sites' autonomy, scalability, fault-tolerance, and quality of services.

When discussing Big Data we have in mind the 5 Vs: Volume, Velocity, Variety, Variability, and Value. There is a clear need of many organizations, companies, and researchers to deal with Big Data volumes efficiently. Examples include web analytics applications, scientific applications, and social networks. For these examples, a popular data processing engine for Big Data is Hadoop MapReduce [27]. The main problem is that data arrives too fast for optimal storage and indexing [28]. There are other several processing platforms for Big Data: Mesos [29], YARN (Hortonworks, Hadoop YARN: A next-generation framework for Hadoop data processing, 2013 (http://hortonworks.com/hadoop/yarn/)), Corona (Corona, Under the Hood: Scheduling MapReduce jobs more efficiently with Corona, 2012 (Facebook)), and so forth. A review of various parallel and distributed programming paradigms, analyzing how they fit into the Big Data era is presented in [30]. The challenges that are described for Big Data Science on the modern and future Scientific Data Infrastructure are presented in [31]. The paper introduces the Scientific Data Lifecycle Management (SDLM) model that includes all the major stages and reflects specifics in data management in modern e-Science. The paper proposes the SDI generic architecture model that provides a basis for building interoperable data or project centric SDI using modern technologies and best practices. This analysis highlights in the same time performance and limitations of existing solutions in the context of Big Data. Hadoop can handle many types of data from disparate systems: structured, unstructured, logs, pictures, audio files, communications records, emails, and so forth. Hadoop relies on an internal redundant data structure with cost advantages and is deployed on industry standard servers rather than on expensive specialized data storage systems [32]. The main challenges for scheduling in Hadoop are to improve existing algorithms for Big Data processing: capacity scheduling, fair scheduling, delay scheduling, longest approximate time to end (LATE) speculative execution, deadline constraint scheduler, and resource aware scheduling.

Data transfer scheduling in Grids, Cloud, P2P, and so forth represents a new challenge that is the subject to Big Data. In many cases, depending on applications architecture, data must be transported to the place where tasks will be executed [33]. Consequently, scheduling schemes should consider not only the task execution time, but also the data transfer time for finding a more convenient mapping of tasks [34]. Only a handful of current research efforts consider the simultaneous optimization of computation and data transfer scheduling. The big-data I/O scheduler [35] offers a solution for applications that compete for I/O resources in a shared MapReduce-type Big Data system [36]. The paper [37] reviews Big Data challenges from a data management respective and addresses Big Data diversity, Big Data reduction, Big Data integration and cleaning, Big Data indexing and query, and finally Big Data analysis and mining. On the opposite side, business analytics, occupying the intersection

of the worlds of management science, computer science, and statistical science, is a potent force for innovation in both the private and public sectors. The conclusion is that the data is too heterogeneous to fit into a rigid schema [38].

Another challenge is the scheduling policies used to determine the relative ordering of requests. Large distributed systems with different administrative domains will most likely have different resource utilization policies. For example, a policy can take into consideration the deadlines and budgets, and also the dynamic behavior [39]. HEP experiments are usually performed in private Clouds, considering dynamic scheduling with soft deadlines, which is an open issue.

The optimization techniques for the scheduling process represent an important aspect because the scheduling is a main building block for making datacenters more available to user communities, being energy-aware [40] and supporting multicriteria optimization [41]. An example of optimization is multiobjective and multiconstrained scheduling of many tasks in Hadoop [42] or optimizing short jobs [43]. The cost effectiveness, scalability, and streamlined architectures of Hadoop represent solutions for Big Data processing. Considering the use of Hadoop in public/private Clouds; a challenge is to answer the following questions: what type of data/tasks should move to public cloud, in order to achieve a cost-aware cloud scheduler? And is public Cloud a solution for HEP simulation experiments?

The activities for Big Data processing vary widely in a number of issues, for example, support for heterogeneous resources, objective function(s), scalability, coscheduling, and assumptions about system characteristics. The current research directions are focused on accelerating data processing, especially for Big Data analytics (frequently used in HEP experiments), complex task dependencies for data workflows, and new scheduling algorithms for real-time scenarios.

4. Conclusions

This paper presented general aspects about methods used in HEP: Monte Carlo methods and simulations of HEP processes, Markovian Monte Carlo, unfolding methods in particle physics, kernel estimation in HEP, Random Matrix Theory used in analysis of particles spectrum. For each method the proper numerical method had been identified and analyzed. All of identified methods produce data-intensive applications, which introduce new challenges and requirements for Big Data systems architecture, especially for processing paradigms and storage capabilities. This paper puts together several concepts: HEP, HPC, numerical methods, and simulations. HEP experiments are modeled using numerical methods and simulations: numerical integration, eigenvalues computation, solving linear equation systems, multiplying vectors and matrices, interpolation. HPC environments offer powerful tools for data processing and analysis. Big Data was introduced as a concept for a real problem: we live in a data-intensive world, produce huge amount of information, we face with upper bound introduced by theoretical models.

Conflict of Interests

The author declares that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The research presented in this paper is supported by the following projects: "*SideSTEP—Scheduling Methods for Dynamic Distributed Systems: a self-∗ approach*", (PN-II-CT-RO-FR-2012-1-0084); "*ERRIC—Empowering Romanian Research on Intelligent Information Technologies*," FP7-REGPOT-2010-1, ID: 264207; CyberWater Grant of the Romanian National Authority for Scientific Research, CNDI-UEFISCDI, Project no. 47/2012. The author would like to thank the reviewers for their time and expertise, constructive comments, and valuable insights.

References

[1] H. Newman, "Search for higgs boson diphoton decay with cms at lhc," in *Proceedings of the ACM/IEEE Conference on Supercomputing (SC '06)*, ACM, New York, NY, USA, 2006.

[2] C. Cattani and A. Ciancio, "Separable transition density in the hybrid model for tumor-immune system competition," *Computational and Mathematical Methods in Medicine*, vol. 2012, Article ID 610124, 6 pages, 2012.

[3] C. Toma, "Wavelets-computational aspects of sterian realistic approach to uncertainty principle in high energy physics: a transient approach," *Advances in High Energy Physics*, vol. 2013, Article ID 735452, 6 pages, 2013.

[4] C. Cattani, A. Ciancio, and B. Lods, "On a mathematical model of immune competition," *Applied Mathematics Letters*, vol. 19, no. 7, pp. 678–683, 2006.

[5] D. Perret-Gallix, "Simulation and event generation in high-energy physics," *Computer Physics Communications*, vol. 147, no. 1-2, pp. 488–493, 2002.

[6] R. Baishya and J. K. Sarma, "Semi numerical solution of non-singlet Dokshitzer-Gribov-Lipatov-Altarelli- Parisi evolution equation up to next-to-next-to-leading order at small x," *European Physical Journal C*, vol. 60, no. 4, pp. 585–591, 2009.

[7] I. I. Bucur, I. Fagarasan, C. Popescu, G. Culea, and A. E. Susu, "Delay optimum and area optimal mapping of k-lut based fpga circuits," *Journal of Control Engineering and Applied Informatics*, vol. 11, no. 1, pp. 43–48, 2009.

[8] R. R. de Austri, R. Trotta, and L. Roszkowski, "A markov chain monte carlo analysis of the cmssm," *Journal of High Energy Physics*, vol. 2006, no. 05, 2006.

[9] C. Şerbănescu, "Noncommutative Markov processes as stochastic equations' solutions," *Bulletin Mathématique de la Société des Sciences Mathématiques de Roumanie*, vol. 41, no. 3, pp. 219–228, 1998.

[10] C. Şerbănescu, "Stochastic differential equations and unitary processes," *Bulletin Mathématique de la Société des Sciences Mathématiques de Roumanie*, vol. 41, no. 4, pp. 311–322, 1998.

[11] O. Behnke, K. Kroninger, G. Schott, and T. H. Schorner-Sadenius, *Data Analysis in High Energy Physics. A Practical Guide to Statistical Methods*, Wiley-VCH, Berlin, Germany, 2013.

[12] V. Blobel, "An unfolding method for high energy physics experiments," in *Proceedings of the Conference on Advanced*

Statistical Techniques in Particle Physics, Durham, UK, March 2002, DESY 02-078 (June 2002).

[13] P. C. Hansen, *Rank-Deficient and Discrete Ill-Posed Problems*, SIAM, Philadelphia, Pa, USA, 1998.

[14] P. C. Hansen, *Discrete Inverse Problems: Insight and Algorithms*, vol. 7, SIAM, Philadelphia, Pa, USA, 2010.

[15] A. Höcker and V. Kartvelishvili, "SVD approach to data unfolding," *Nuclear Instruments and Methods in Physics Research A*, vol. 372, no. 3, pp. 469–481, 1996.

[16] I. Aziz and Siraj-ul-Islam, "New algorithms for the numerical solution of nonlinear Fredholm and Volterra integral equations using Haar wavelets," *Journal of Computational and Applied Mathematics*, vol. 239, pp. 333–345, 2013.

[17] A. Sawatzky, C. Brune, J. Muller, and M. Burger, "Total variation processing of images with poisson statistics," in *Proceedings of the 13th International Conference on Computer Analysis of Images and Patterns (CAIP '09)*, pp. 533–540, Springer, Berlin, Germany, 2009.

[18] A. Edelman and N. R. Rao, "Random matrix theory," *Acta Numerica*, vol. 14, no. 1, pp. 233–297, 2005.

[19] K. Cranmer, "Kernel estimation in high-energy physics," *Computer Physics Communications*, vol. 136, no. 3, pp. 198–207, 2001.

[20] I. S. Abramson, "On bandwidth variation in kernel estimates—a square root law," *The Annals of Statistics*, vol. 10, no. 4, pp. 1217–1223, 1982.

[21] X. Li, J. W. Demmel, D. H. Bailey et al., "Design, implementation and testing of extended and mixed precision BLAS," *ACM Transactions on Mathematical Software*, vol. 28, no. 2, pp. 152–205, 2002.

[22] J. J. Dongarra and J. Langou, "The problem with the linpack benchmark 1.0 matrix generator," *International Journal of High Performance Computing Applications*, vol. 23, no. 1, pp. 5–13, 2009.

[23] L. Lundberg and H. Lennerstad, "An Optimal lower bound on the maximum speedup in multiprocessors with clusters," in *Proceedings of the IEEE 1st International Conference on Algorithms and Architectures for Parallel Processing (ICAPP '95)*, vol. 2, pp. 640–649, April 1995.

[24] N. J. Gunther, "A note on parallel algorithmicspeedup bounds," *Technical Report on Distributed, Parallel, and Cluster Computing*. In presshttp://arxiv.org/abs/1104.4078.

[25] B. C. Tak, B. Urgaonkar, and A. Sivasubramaniam, "To move or not to move: the economics of cloud computing," in *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing (HotCloud '11)*, p. 5, USENIX Association, Berkeley, CA, USA, 2011.

[26] L. Zhang, C. Wu, Z. Li, C. Guo, M. Chen, and F. Lau, "Moving big data to the cloud: an online cost-minimizing approach," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 12, pp. 2710–2721, 2013.

[27] J. Dittrich and J. A. Quiane-Ruiz, "Efficient big data processing in hadoop mapreduce," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 2014–2015, 2012.

[28] D. Suciu, "Big data begets big database theory," in *Proceedings of the 29th British National conference on Big Data (BNCOD '13)*, pp. 1–5, Springer, Berlin, Germany, 2013.

[29] B. Hindman, A. Konwinski, M. Zaharia et al., "A platform for fine-grained resource sharing in the data center," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (NSDI '11)*, p. 22, USENIX Association, Berkeley, CA, USA, 2011.

[30] C. Dobre and F. Xhafa, "Parallel programming paradigms and frameworks in big data era," *International Journal of Parallel Programming*, 2013.

[31] Y. Demchenko, C. de Laat, A. Wibisono, P. Grosso, and Z. Zhao, "Addressing big data challenges for scientific data infrastructure," in *Proceedings of the IEEE 4th International Conference on Cloud Computing Technology and Science (CLOUDCOM '12)*, pp. 614–617, IEEE Computer Society, Washington, DC, USA, 2012.

[32] T. White, *Hadoop: The Definitive Guide*, O'Reilly Media, 2012.

[33] J. Celaya and U. Arronategui, "A task routing approach to large-scale scheduling," *Future Generation Computer Systems*, vol. 29, no. 5, pp. 1097–1111, 2013.

[34] N. Bessis, S. Sotiriadis, F. Xhafa, F. Pop, and V. Cristea, "Meta-scheduling issues in interoperable hpcs, grids and clouds," *International Journal of Web and Grid Services*, vol. 8, no. 2, pp. 153–172, 2012.

[35] Y. Xu, A. Suarez, and M. Zhao, "Ibis: interposed big-data i/o scheduler," in *Proceedings of the 22nd International Symposium on High-performance Parallel and Distributed Computing (HPDC '13)*, pp. 109–110, ACM, New York, NY, USA, 2013.

[36] S. Sotiriadis, N. Bessis, and N. Antonopoulos, "Towards inter-cloud schedulers: a survey of meta-scheduling approaches," in *Proceedings of the 6th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (PGCIC '11)*, pp. 59–66, Barcelona, Spain, October 2011.

[37] J. Chen, Y. Chen, X. Du et al., "Big data challenge: a data management perspective," *Frontiers in Computer Science*, vol. 7, no. 2, pp. 157–164, 2013.

[38] V. Gopalkrishnan, D. Steier, H. Lewis, and J. Guszcza, "Big data, big business: bridging the gap," in *Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications (BigMine '12)*, pp. 7–11, ACM, New York, NY, USA, 2012.

[39] R. van den Bossche, K. Vanmechelen, and J. Broeckhove, "Online coste fficient scheduling of deadline-constrained workloads on hybrid clouds," *Future Generation Computer Systems*, vol. 29, no. 4, pp. 973–985, 2013.

[40] N. Bessis, S. Sotiriadis, F. Pop, and V. Cristea, "Using a novel messageexchanging optimization (meo) model to reduce energy consumption in distributed systems," *Simulation Modelling Practice and Theory*, vol. 39, pp. 104–112, 2013.

[41] G. V. Iordache, M. S. Boboila, F. Pop, C. Stratan, and V. Cristea, "A decentralized strategy for genetic scheduling in heterogeneous environments," *Multiagent and Grid Systems*, vol. 3, no. 4, pp. 355–367, 2007.

[42] F. Zhang, J. Cao, K. Li, S. U. Khan, and K. Hwang, "Multi-objective scheduling of many tasks in cloud platforms," *Future Generation Computer Systems*, 2013.

[43] K. Elmeleegy, "Piranha: optimizing short jobs in hadoop," *Proceedings of the VLDB Endowment*, vol. 6, no. 11, pp. 985–996, 2013.