Vectorization techniques for probability distribution functions using VecCore

Oscar R. Chaparro Amaro [1], J. Martínez-Castro [1], Soon Yung Jun [2]

[1]Instituto Politécnico Nacional, Centro de Investigación en Computación †, Av. Juan de Dios Bátiz s/n CDMX C.P. 07738, México

[2] Fermi National Accelerator Laboratory ††, PO Box 500, Batavia, IL 60510, USA

E-mail: saga9211@hotmail.com

Abstract.
Probability distribution functions (PDFs) are very used in modeling random processes and physics simulations. Improving the performance of algorithms that generate many random numbers under complex PDFs is often a very challenging task when methods as direct functions are not available. In this work we present general strategies on how to vectorize some PDFs using VecCore library. We show the results for the Exponential, Gaussian, discrete Poisson and Gamma probability distributions.

1. Introduction

Probability distribution functions (PDFs) are statistical expressions of random variables and are very useful in physics simulation and modeling. In many cases, direct estimations of PDFs involve harder processes than numeric approximation algorithms, implying less performance in time and that more computational resources are required. Several state-of-the-art parallel pseudo-random number generator algorithms (PRNGs) such as MRG32k3a, Threefry and Philox are recently implemented in VecMath [1] for both SIMD and SIMT(GPU) architectures [2]. However, complex PDFs cannot be vectorized easily and efficiently as they usually involve conditional branches, and rejection and acceptance processes recursively.

VecCore [3, 4] is an open source SIMD (Single Instruction Multiple Data) library that expresses vector operations with architecture independent APIs. The portability between these algorithms and different vector architectures can be abstracted through backends, which supports explicit SIMD vectorization libraries such as Vc [5] and UME::SIMD [6] as well as CUDA specific extensions for NVIDIA GPUs [7]. More complex PDFs can be vectorized using strategies based on algorithms that use the i.i.d (independent and identically distributed) in (0,1], $U(0,1]$, implementations of VecMath [1]. In this paper we will describe strategies how to vectorize some common PDFs using VecMath and APIs of VeCore for two main categories: algorithms without branches and with branches.

2. Methods of PDFs

Algorithms for generating PDFs simulate random values using their function parameters instead of evaluating direct PDF's expressions due to the difficulty of dealing with expensive computations such as factorial calculations as in the Gamma function case. As long as

vectorization concerns, the simplest and most efficient method is the case without any branches for which the VecCore Backend can apply SIMD instructions easily.

2.1. Algorithms without Branches

These cases involve two different approaches, the first example is the case of the exponential PDF:

$$E(x,\tau) = \frac{e^{-x/\tau}}{\tau} \tag{1}$$

It uses the inverse-transform technique that generates random variates by using its inverse cumulative function with parameter τ and a uniform random value, $U(0,1]$: $-\tau * \ln(U) \sim E(\tau)$ [8]. For vectorization, operations without branches are suitable as the same operation is repeated by each processing unit with different input data. This method is efficient for vectorization because it uses a direct SIMD intrinsics.

The second example is the case of the normal or Gaussian PDF:

$$N(x,\mu,\sigma) = \frac{e^{-(x-\mu)^2/2\sigma^2}}{\sqrt{2\pi\sigma^2}} \tag{2}$$

for which there exists an alternative branch-free algorithm which generates an equivalent random variate. The generalized Box Müller algorithm is selected for vectorization due to the easy vector pipeline implementation over input data [9]. The Box Müller algorithm is more efficient for the vectorization since it takes two uniform random values as input, which are fully vectorizable using VecMath APIs. The algorithm requires the vector of the mean μ and standard deviation σ. Therefore, using SIMD instructions, each processing unit executes the fixed function $\left[\mu + \sigma * \sqrt{-2 * ln(U_1)} * cos(U_2 * 2\pi)\right] \sim N(\mu,\sigma)$ for multiple input data.

2.2. Algorithms with Branches

When PDFs cannot be generated by its inverse transformation efficiently or there is no known way to generate an alternative equivalent variate, it is common to use the well-known acceptance-rejection method [10]. Cases as Gamma and discrete Poisson PDFs belong to this category. The Gamma PDF is defined as:

$$G(x,\alpha,\beta) = \frac{\beta^\alpha x^{\alpha-1} e^{-\beta x}}{\Gamma(\alpha)} \tag{3}$$

where α and β are shape and scale parameters, respectively and $\Gamma(\alpha)$ is the Gamma function of α. Algorithms with branches are introduced since they use the rejection-acceptance technique. These conditional branches can be handled by vectorized techniques using few VecCore APIs. In general, SIMD instructions with mask selector methods and the combination of branches are good strategies to apply. The two general cases of the Gamma distribution are $\alpha > 1$ and $\alpha \leq 1$, but both cases are implemented with a common kernel. Considering performance and flexibility for the vectorization. We choose the acceptance-rejection algorithm of the Marsaglia and Tsang method [11]. For the $\alpha \leq 1$ case, an extended variant of the algorithm for $\alpha > 1$ was used [10]. The method exploits the similarity relationship between Normal and Gamma PDFs with two distribution thresholds defined as: $c = 1 - 0.0331 * N_1(0,1)^4$ and $d = \left(1 + N(0,1)/\sqrt{9\alpha - 3}\right)^3$. This algorithm involves a conditional branch since $d(\alpha - 1/3) \sim G(\alpha,1)$ is accepted only if some random value $U_1 \sim U(0,1]$ is less than c and d is positive [11]. VecCore provides APIs that can deal with these branches as the `MaskFull` method, which allows to perform a mask-based selection on a specific branch according to one conditional (Mask). Then, the vector operation is applied to the whole vector inputs, executing 'if' conditionals simultaneously. If the first

condition is rejected, new random value $N_2 \sim N(0,1)$ and a second conditional that depends on U_1, N_1, α and N_2 values are tested [11]. This double conditional can be managed with the same SIMD instructions nesting Mask conditions on input data, keeping the vectorization and the same operation over the whole vector shown in Example Code 1:

```
Mask<Type_variable_v> condition_name1(condition1);
if(MaskFull(condition1)) { Path1 } else {
{Mask<Type_variable_v> condition_name2(condition2);
if(MaskFull(condition1)) { Path2 } } else { scalar path }}
```

Example Code 1: An example of VecCore implementation for branches.

If the second condition is rejected, the scalar loop can be applied, which rarely happens because the Marsaglia and Tsang method has less than 10% of rejection rate. In the case when $\alpha \leq 1$, an alternative conditional test can be managed by `MaskEmpty`, which selects a branch inside vector operations if some condition is not accomplished by the vector inputs (when $\alpha < 1$). This second case assumes a random value $U_3 \sim U(0,1]$ and then calculates: $G(\alpha+1,1)U_3^{1/\alpha} \sim G(\alpha,1)$. The scale parameter β, was added assuming the linearity $G(\alpha,1) * 1/\beta \sim G(\alpha,\beta)$ [12]. If all lanes of the input vector are different values with $\alpha > 1$ and $\alpha \leq 1$, full vectorization is not possible (the mixed case), but is possible to increase the acceptance rate by regrouping data.

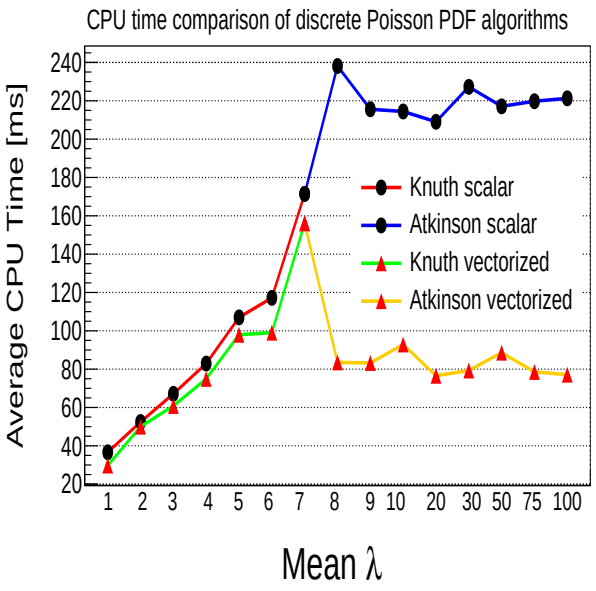

Figure 1: Execution time comparison between Discrete Poisson PDF algorithms that depends on λ.

The second example of these cases is the discrete Poisson PDF defined as:

$$P(k,\lambda) = \frac{e^{-\lambda}\lambda^k}{k!} \tag{4}$$

where λ is the average number of events per interval (mean). Algorithms by Knuth [13] and Atkinson [14] were used for vectorizing this PDF. The performances of both algorithms depend directly on λ. For the Knuth's algorithm, if λ increases, performance is worse. It states: $L{=}e^{-\lambda}$, $k{=}0$, $p{=}1$ and then increments $k+1$ and generates new $U(0,1]$ each time $p * U \leq L$. When the condition is satisfied, it returns the final count $k-1$, which involves the do-while loop. Unfortunately, this kind of algorithms are very difficult to vectorize efficiently since the condition diverges for each processing unit during the iterations and then is resolved individually at different stages.

In contrast, the second algorithm of Atkinson improves its performance for larger values of λ but only from $\lambda \geq 5$ [14]. This method uses double acceptance conditions; the first condition depends on λ, while the second condition depends on a function that uses a $U(0,1]$, two λ-dependent variables and $p = \ln(n!)$. For this case, p is approximated by the Lanczos method [15] of which implementation is efficiently vectorizable using VecCore APIs. Once the second condition is accepted, the function returns $\texttt{Floor}\left[\left(\pi * \lambda/\sqrt{3\lambda} - \ln((1-U)/U)\right)/\pi/\left(\sqrt{3\lambda}+0.5\right)\right]$, otherwise the process is repeated until these two conditions are hold. If conditionals do not diverge, then the Mask method can be applied as for the Gamma PDF case.

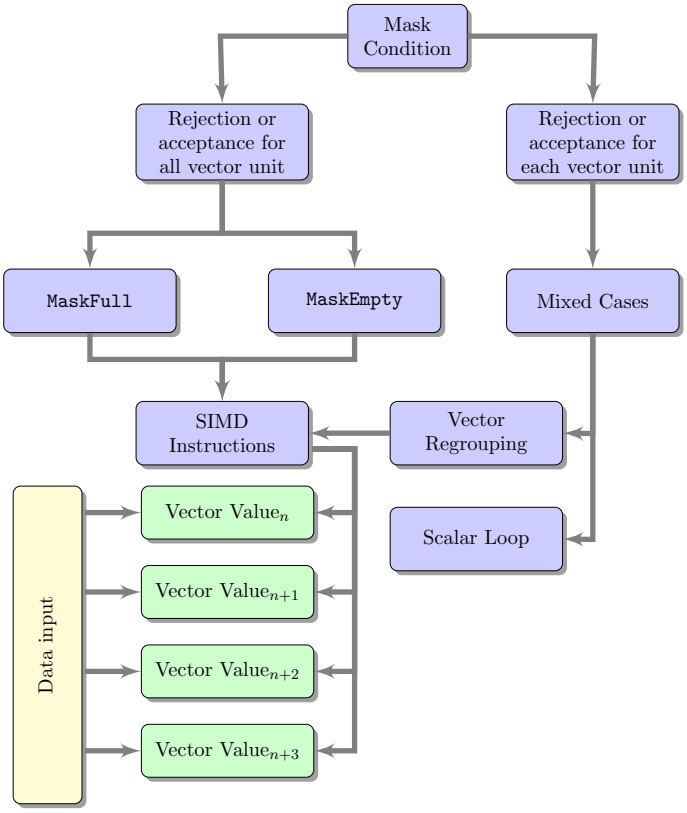

Figure 2: General strategies followed to use SIMD instructions for vectorizing PDFs.

Finally, due to the limitation of the second method at small values of λ , we combine these two methods using the first algorithm up to $\lambda = 8$, which keeps the good performance at large values of λ without affecting the generation of random variates with small λ values. Figure 1 shows performance comparisons (execution times) between scalar (red) and vector (green) methods for the two algorithms. Vectorized results of Gamma and discrete Poisson PDFs are shown in Figure 3 and Figure 4. As a summary, the execution time as the number of input for both scalar and vector methods are compared in Figure 5, showing improvements around 3.5 to 4 times by vectorization for fixed parameter cases following the strategies of the Figure 2.

3. Conclusions

Several common PDFs were vectorized successfully using VecCore. We demonstrate that sizable gains on CPU performance are achievable by vectorization techniques for implemented PDFs with VecMath. More complex PDFs can be vectorized exploring their generation algorithms.

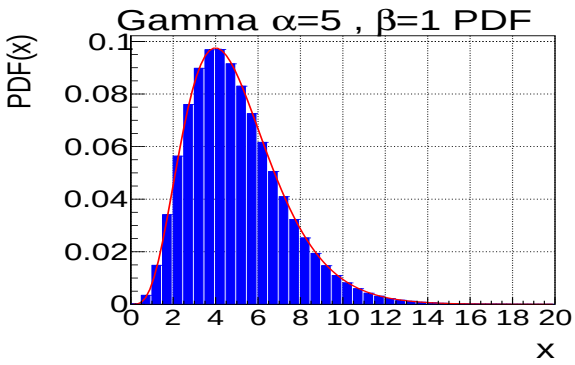

Figure 3: Simulated results of the vectorized Gamma density distribution with fitted parameters α=4.98 and β=1.001 using input vector size of 1×10^6.

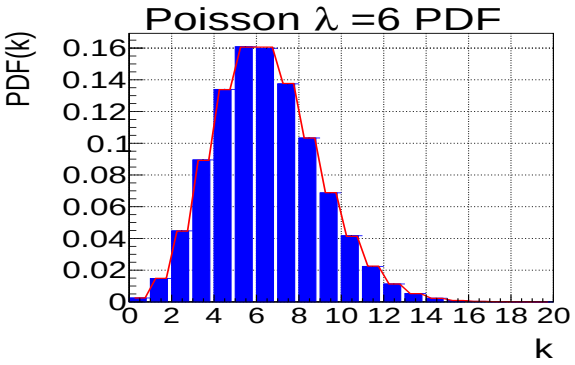

Figure 4: Simulated results of the vectorized discrete Poisson density distri- bution with fitted parameter λ=6.0009 using input vector size of 1×10^6.

The presented strategies can be used to deal with future vectorization of algorithms that generates other complex PDFs.

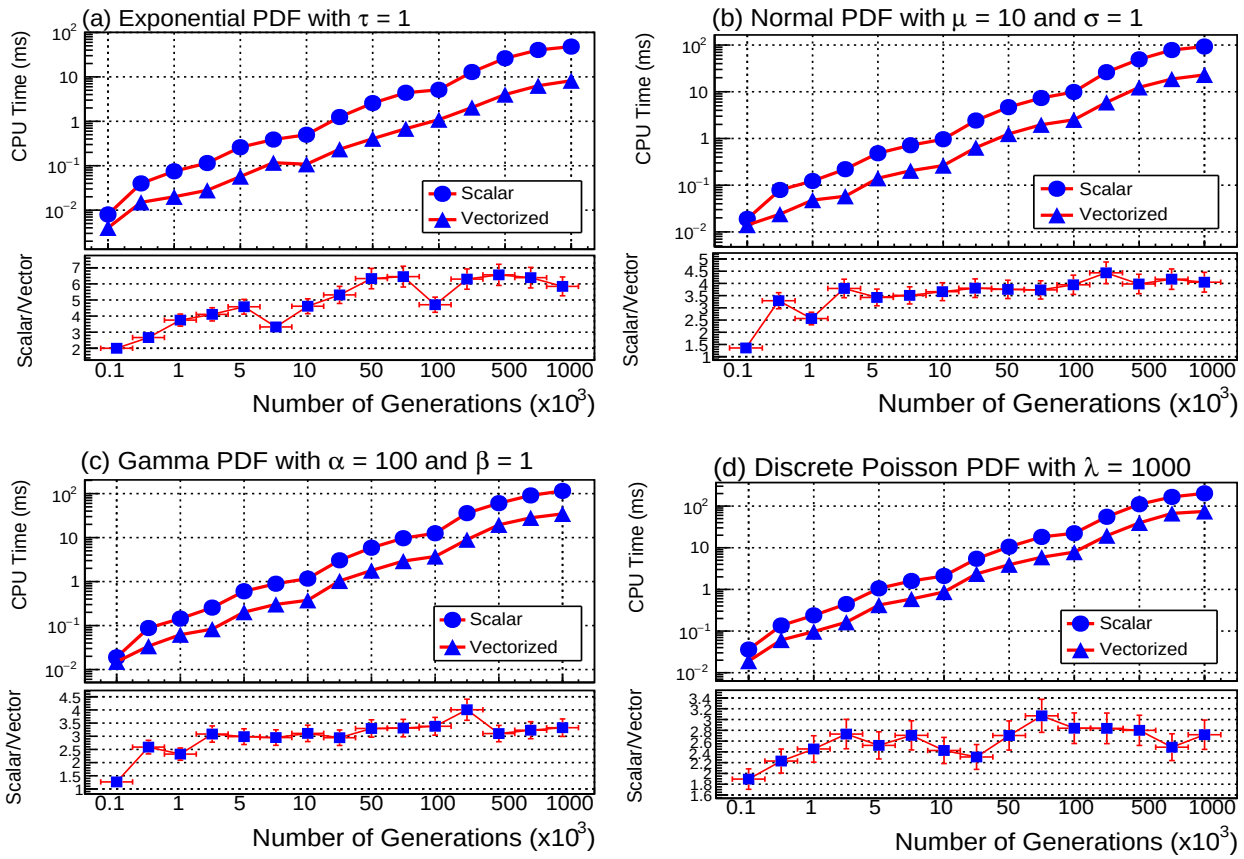

Figure 5: The average CPU time of 100 measurements as the number of samplings and the performance gain by vectorization (scalar/vector) on CPU, Intel® Core ™ i7-4510U 2.00GHz (4 cores), using the AVX2 instruction set.

Acknowledgments

† Grants supported by CONACYT and BEIFI Project No. 20195941.

†† Operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United States Department of Energy.

4. References

[1] VecMath [software], 2018. Available from https://github.com/root-project/vecmath [accessed 2018-02-15]

[2] Jun S Y *et al.* 2019. Vectorization of random number generation and reproducibility of concurrent particle transport simulation, the 19^{th} ACAT conference

[3] Amadio G *et al.* 2018 *Journal of Physics* **1085** 1

[4] VecCore project [software] . Available from https://github.com/root-project/veccore [accessed 2018-02-17]

[5] Vc [software], version 1.3.0, 2017. Available from https://github.com/VcDevel/Vc [accessed 2018-05-20]

[6] UME::SIMD [software], version 0.8.1, 2017. Available from https://github.com/edanor/umesimd [accessed 2018-05-15]

[7] Nickolls J, Buck I, Garland M and Skadron K 2008 *Queue-GPU Computing* **6** 40–53

[8] Micula S and Pop I D 2016 *Journal of Information Systems and Operations Management* **1** 435

[9] Murison M 2000 *Astronomical Applications Department U.S. Navy Observatory.* **1** 5

[10] Van N, Thi N and Chien T Q 2015 *Indian Journal of Computer Science and Engineering* **5** 199

[11] Marsaglia G and Tsang W W 2000 *ACM Transactions on Mathematical Software* **26** 363

[12] Martino L and Luengo D 2013 *Cornell University Library Statistics Computation* **3** 1

[13] Devroye L 1980 *Journal in Computing* **1** 197

[14] Atkinson A C 1979 *Journal of the Royal Statistical Society* **28** 29

[15] Lanczos C 1964 *Journal of Society for Industrial and Applied Mathematics B. and Numerical Analysis* **1** 86