Transformed Lattice Rules for Feynman Loop Integrals on GPUs

E de Doncker¹, A Almulihi¹ and F Yuasa²

¹ Department of Computer Science, Western Michigan University, Kalamazoo MI 49008, U. S. A.
² High Energy Accelerator Research Organization (KEK), Oho 1-1, Tsukuba, Ibaraki, 305-0801, Japan

E-mail: {elise.dedoncker, ahmedhassan.almulihi}@wmich.edu, fukuko.yuasa@kek.jp

Abstract. We focus on the application of rank-1 lattice rules with tanh and sin\textsuperscript{m} type transformations for the integration of functions which may have singularities on the boundaries of the integration domain. Using this technique we compute Feynman loop integrals for classes of 2-loop box and 3-loop self-energy Feynman diagrams that have up to eight internal lines with masses. The \( d \)-dimensional rule approximation with \( n \) points relies on a generator vector of length \( d \) with integer components, which is computed once and for all using the Component-by-Component method by Nuyens and Cools (2006). The generator vector is incorporated in the (CUDA) integration program that implements the lattice integration for fast execution on GPUs. Results are obtained efficiently and without special attention to specific problem characteristics.

1. Introduction

It is our goal to evaluate Feynman loop integrals accurately and efficiently. Thousands of Feynman diagrams may be associated with a particle interaction, each representing a possible configuration of virtual processes for the interaction and contributing a corresponding part to the total amplitude and thus to the probability or cross section of the interaction. The amplitude is expanded as a perturbation series, where the leading (lowest) order of approximation corresponds to tree level Feynman diagrams, and loop diagrams are needed for the higher orders. The increasing precision of parameter measurements in collider experiments necessitates the computation of loop integrals, underlying the theoretical predictions for present and future high-energy experiments. After the discovery of the Higgs boson at the Large Hadron Collider in 2012 [1, 2], new physics beyond the Standard model (SM) is expected but has not yet emerged. The motivation for the discovery of derivations from SM predictions increases, and the computation of higher order corrections including loop integrals in perturbative quantum field theory is of great importance. Detailed discussions on the precision levels are presented in [3, 4].

Analytic methods are generally not feasible for multi-loop integrals with a variety of masses, and the integral dimension increases with the number of loops and internal lines. Numerical computations need to deal with the dimension as well as integrand singularities and the large scale of the computations. We have targeted singularities with reduction, transformation and extrapolation methods [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18], and with adaptive and parallel strategies as further reported in [19, 20, 21, 22, 23, 24, 25, 26]. Accelerator boards such as GPUs can be utilized to parallelize approaches that involve a uniform computational structure. Product type approximations such as a product trapezoidal rule (typically applied with a transformation to relieve singularities at the boundaries of the domain) fall in this category but suffer from the dimensional effect. In this paper we...
explore GPU power for lattice rule integration combined with transformations for dealing with boundary singularities. In this context we investigate the accuracy and efficiency of \( \tanh \) [27] and Sidi’s \( \sin^m \)-transformations [28, 29].

Lattice integration was used with Sidi’s transformations in programs for 2 and 3 dimensions in \( r2d2tri \) [30], \( elrint3d \) [31], with a product-wise implementation of the latter for 5-dimensional integration in \( print5d \) [32]. Lattice rules on GPU were recently applied to integrals resulting from sector decomposition in high energy physics [33]. In [34] we compared lattice rules using Sidi’s \( \sin^m \)-transformations with \( m = 2, 4, 6 \) (and without transformation, \( m = 0 \)) for Feynman integrals. We reported comparisons of lattice rules to Monte Carlo integration on GPUs in [35] for several classes of test functions in higher dimensions (up to dimension 25).

Importance sampling for variance reduction has been used extensively in Monte Carlo integration. The adaptive Monte Carlo integration and event generation package BASES/SPRING [36] uses a strategy similar to that of Vegas [37], and allows the user to select a set of important variables (coordinate directions) where the integrand function varies most. The implementation of Vegas in the Cuba package [38] incorporates sampling by Mersenne Twister and RANLUX pseudo-random numbers and by Sobol quasi-random (low discrepancy) sequences.

In subsequent sections we give some background for rank-1 lattice rules in Section 2, and outline transformations to treat singularities on the boundaries of the unit simplex domain in Section 3. Section 4 applies lattice rules with transformations to Feynman loop integrals for classes of 2-loop box and 3-loop self-energy diagrams, and compares GPU results for various numbers of points with respect to accuracy and time. Lattice generator vectors used for the application, as well as CUDA code for the integration kernel and for sample integrand device functions are given in appendices.

2. Lattice rules

A Monte Carlo approximation for the integral over the unit cube \( C_d = [0, 1]^d \) in \( \mathbb{R}^d \) takes the form

\[
If = \int_{C_d} f(x) \, dx \approx \frac{1}{n} \sum_{j=0}^{n-1} f(x_j),
\]

where the \( x_j \) are uniform random. We consider Quasi-Monte Carlo (QMC) approximations of the same form as in Eq (1), but where the function evaluation points lie on a lattice within the half-open cube \([0, 1)^d\). One direction to derive good rules is based on the theory of optimal coefficients or good lattice points introduced by Korobov [39, 40, 41] and Hlawka [42]. For overviews see, e.g., [43, 44, 45, 46].

A rank-1 lattice rule is represented as

\[
Q(z, n) f = \frac{1}{n} \sum_{j=0}^{n-1} f(\{z/n\}), \tag{2}
\]

where \( z \) is an integer generator vector with components \( z \in \mathbb{Z}_n = \{1 \leq z < n, \gcd(z, n) = 1\} \), and \( \{x\} \) denotes the vector in \([0, 1)^d\) obtained by taking the fractional part of each component of \( x \).

Assuming that \( f(x) \) is one-periodic and its Fourier expansion

\[
f(x) = \sum_{m \in \mathbb{Z}^d} c_m \exp(2\pi i \cdot m) \quad \text{with} \quad c_m = \int_{C_d} \exp(-2\pi i \cdot m \cdot x) \, f(x) \, dx, \quad m \in \mathbb{Z}^d
\]

is absolutely convergent, and applying the rule of Eq (2) to the Fourier expansion leads to [44, 46]

\[
Q(z, n) f - If = \sum_{m \in \mathbb{Z}^d-\{0\}} c_m \sum_{n \equiv 0 \pmod{n}} Q(z, n) \exp(2\pi i \cdot m \cdot x) = \sum_{m \neq 0} c_m.
\]

Here the prime on the sum indicates that the term for \( m = 0 \) is omitted.

The class \( E^\alpha (\kappa) \), for \( \alpha > 1 \) and some \( \kappa > 0 \) is defined as the class of functions whose Fourier coefficients are bounded as
In the dominant term of Eq (5), by

$$I[0, 1] f = \int_0^1 f(x) dx,$$

the transformation $x = \varphi(u)$ yields

$$I[0, 1] f = \int_0^1 f(\varphi(t)) \varphi'(t) dt.$$

**3. Transformations**

Convergence results of lattice rules for functions in $\mathcal{E}^\alpha(\kappa)$ as those listed above motivate the use of periodizing transformations, such that the resulting function can be extended periodically. For the one-dimensional integral $I[0, 1] f = \int_0^1 f(x) dx$, the transformation $x = \varphi(u)$ yields

$$I[0, 1] f = \int_0^1 f(\varphi(t)) \varphi'(t) dt.$$

Periodizing transformations include [44]

$$\varphi(t) = \varphi_m(t) = (2m + 1)\left( \frac{2m}{m} \right) \int_0^t u^m (1 - u)^m du \quad (\text{Korobov's transformation [41]})$$

and

$$\varphi(t) = 1 - 2|t - \frac{1}{2}|.$$

**Membership in the class $\mathcal{E}^\alpha(\kappa)$ characterizes the smoothness of the function $f$ and allows for a bound on its integration error as**

$$|Q(z, n) f - I f| \leq \kappa P_\alpha(z, n),$$

where $P_\alpha(z, n) = \sum_{m=0}^{n} r_m^\alpha$. A sufficient condition for membership $f \in \mathcal{E}^\alpha(\kappa)$ is given by conditions of bounded variation on its mixed partial derivatives in the sense of Hardy and Krause [47, 46]. It is further noted that the bound can be reached (with an equality) in Eq (3).

With Korobov’s definition [39], the components of the generator $z$ are called *optimal components* (or $z$ is called a *good lattice point* by Hlawka [42]) if, for $f \in \mathcal{E}^\alpha(\kappa)$,

$$|Q(z, n) f - I f| \leq \kappa \gamma(d, \alpha) \frac{(\log n)^{\beta(d, \alpha)}}{n^\alpha},$$

with $\gamma(d, \alpha)$ and $\beta(d, \alpha)$ independent of $n$ and $\alpha > 1$. Thus if $P_\alpha(z, n) \leq \gamma(d, \alpha) \frac{(\log n)^{\beta(d, \alpha)}}{n^\alpha}$ in Eq (3), then $z$ is a good lattice point according to Eq (4) (see [46]). Existence of good rank-1 rules was first proved by Korobov for $n$ prime. Results by Niederreiter [43, 46] are used to show the existence of $z \in \mathbb{Z}^d$, $d \geq 2$, $n \geq 2$ such that

$$P_\alpha(z, n) \leq \frac{(2 \log n)^{ad}}{n^\alpha} + \mathcal{O}\left(\frac{(\log n)^{ad-1}}{n^\alpha}\right).$$

Disney and Sloan [48] improved particularly on the factor $(2^{ad})$ in the dominant term of Eq (5), by proving the existence of $z$ such that $P_\alpha(z, n)$ is asymptotically bounded by $(\frac{2}{\pi})^{ad} (\log n)^{\beta(d, \alpha)}$ for $d \geq 3$. For the sake of comparison consider the error bound of Monte Carlo integration with convergence rate of $\mathcal{O}(1/\sqrt{n})$ as $n \to \infty$.

The component-by-component (CBC) algorithm by Sloan and Reztsov [49] for the computation of the generator vector $z$ runs in $\mathcal{O}(dn^2)$ time. For our implementation of the integration we pre-compute the generator vector with the fast CBC algorithm by Nuyens and Cools [50, 51], which runs in $\mathcal{O}(dn \log(n))$ time and $\mathcal{O}(n)$ space. Table A1 gives a list of the generator vectors we obtained for various numbers $n$ of points and dimensions $d$. In the construction, weights can be associated with the coordinate directions (variables) according to their importance. However, the rules generated for the computations in this paper assumed equal weights. No a-priori information is provided for the integration problems.

The summation of Eq (2) is implemented in the CUDA kernel of Appendix B, with a structure similar to that of the dot product in [52]. In the while loop we compute the $x$-components of the lattice evaluation points (in array $x$) from the generator given in array $z$ on input, and perform the integrand evaluations. The array cache is shared by the threads over a block, and the reduction after the while loop sums the contributions into the cache[0] element for the block. The partial results over the blocks are returned in the array $q$ and will be summed in the main program over the blockdim blocks. As the number of blocks is relatively small (e.g., blockdim = 64), this summation is carried out sequentially.
Figure 1. (a) Lattice points for $d = 2$, $n = 503$, $z = (1, 192)$; (b) Transformed lattice points with tanh; (c) Transformed lattice points with tanh in unit triangle

Figure 2. (a) Function $1/\sqrt{x_1 x_2(1 - x_1 - x_2)}$ over unit triangle; (b) Transformed integrand $1/\sqrt{x_1 x_2(1 - x_1 - x_2)} \times$ Jacobian with tanh transformation

We further consider the class of $\sin^m$-transformations by Sidi [28, 29],

$$\varphi(t) = \Psi_m(t) = \frac{\theta_m(t)}{\theta_m(t)}$$

where $\theta_m(t) = \int_0^t \sin^m(\pi u) \, du$, $m = 1, 2, \ldots$,

in particular,

$$\Psi_6(t) = t - \frac{1}{60} \sin(2\pi t) - \frac{9}{2} \sin(4\pi t) + \sin(6\pi t)$$

and the tanh transformation by Sag and Szekeres [27], given for $I[0, 1]f = \int_{-1}^{1} f\left(\frac{\varphi^{-1}}{2}\right) \, dt$ by

$$t = \varphi(u) = \tanh\left(\frac{c u}{1 - u^2}\right), \text{ so that } I[0, 1]f = \frac{c}{2} \int_{-1}^{1} f\left(\frac{\varphi(u)}{2} + \frac{1}{2}\right) \frac{1 + u^2}{(1 - u^2)^3} \, \sech^2\left(\frac{c u}{1 - u^2}\right) \, du$$

(where $c > 0$ is a constant). The Jacobian of the tanh transformation in Eq (7) and all its derivatives vanish at the end-points of the transformed integral. The latter property is also satisfied by the IMT [53] and DE (double exponential) [54, 55] type transformations. Korobov’s and Sidi’s transformations $\varphi(t) = \varphi_m(t)$ and $\varphi(t) = \Psi_m(t)$ satisfy that $\varphi'(t) = O(t^m)$ as $t \to 0+$, and $\varphi'(t) = O((1 - t)^m)$ as $t \to 1-$ (see [28]). Whereas a low order $\Psi_m$ such as $\Psi_1(t) = (1 - \cos(\pi t))/2$ may be used to transform a smooth nonperiodic function $f(x)$ into a smooth periodized function $f(\Psi_m(t))\Psi'_m(t)$, higher orders can help alleviate or smoothen boundary singularities more effectively through the occurrence of the Jacobian proportional with $\sin^m(\pi t)$.

Fig 1(a) illustrates the $n = 503$ lattice points of the 2D rule with $z = (1, 192)$. Fig 1(b) and (c) show the lattice points transformed with the tanh-transformation (the latter to the unit triangle). Note that the transformed points spread out from the middle of the region, moving toward the edges. Especially for
rules with a large number of points this may lead to computational problems with over- or underflows when the integrand is evaluated at points very close to the boundaries of the domain.

With respect to Fig 2 consider the integral of \( f(x_1,x_2) = \frac{1}{\sqrt{x_2-x_1(x_1-x_2)}} \) over the unit triangle \( S_2 \),

\[
I(S_2) = \int_0^1 dx_1 \int_0^{1-x_1} dx_2 f(x_1,x_2) = \int_0^1 dx_1 \int_0^{1-x_1} dx_2 f(x_1,(1-x_1)x_2) = \frac{1}{4} \int_{-1}^1 dt_1 \int_{-1}^1 dt_2 f(1+t_1/2,1-t_1/2-t_2/2)
\]

where \( \varphi(u) = \tanh(\frac{u}{1-u}) \), \( c = 3 \), and \( \alpha_1 = \frac{1}{2}(1+\varphi(u_1)) \), \( \alpha_2 = \frac{1}{4}(1-\varphi(u_1))(1+\varphi(u_2)) \). Then Fig 2(a) depicts the integrand \( f(x_1,x_2) \) of Eq (8) as a function of \( x_1 \) and \( x_2 \) over \( S_2 \), and Fig 2(b) plots the integrand of Eq (9) as a function of \( u_1 \) and \( u_2 \) over \([-1,1]^2\). Even though the original integrand has singularities at the edges of the triangle, the transformed integrand is zero at the boundaries of the transformed domain. A similar behavior is depicted for Sidi’s \( \Psi_4 \) transformation in [34]. The interest in the triangle region \( S_2 \) is motivated by the computation of Feynman integrals over simplex domains.

4. Application to Feynman Diagrams
4.1. Feynman loop integrals

We represent a Feynman integral with \( L \) loops and \( N \) internal lines by \( \mathcal{F} = (4\pi)^{-\nu L/2} \mathcal{F}_{L,N} \) with

\[
\mathcal{F}_{N,L} = \Gamma(N-\frac{\nu L}{2})(-1)^N \int_{\mathcal{C}_N} \prod_{r=1}^{N} dx_r \delta(1-\sum_{r} x_r) \frac{c^{N-\nu(L+1)/2}}{(D-igC)^{N-\nu L/2}},
\]

where \( \nu = 4 \) is the space-time dimension, and \( C \) and \( D \) are polynomials arising from determinants and depending on the topology of the Feynman diagram and physical parameters. \( C \) and \( D \) are homogeneous of degree \( L \) and \( L+1 \), respectively, in the integration variables \( x_r \). The \( \delta \)-function in Eq (10) can be eliminated by expressing one of the integration variables in terms of the other, leading to

\[
\mathcal{F}_{N,L} = \Gamma(N-\frac{\nu L}{2})(-1)^N \int_{\mathcal{S}_{N-1}} \frac{c^{N-\nu(L+1)/2}}{(D-igC)^{N-\nu L/2}} dx.
\]

Here the domain is the \( d = (N-1) \)-dimensional unit simplex, \( \mathcal{S}_d = \{ x \in \mathcal{C}_d \mid 0 \leq \sum_{k=1}^d x_k \leq 1 \} \). The integral over \( \mathcal{S}_{N-1} \) can then further be transformed to \( \mathcal{C}_{N-1} \).

Subsequently we give numerical results for the 2-loop box integrals of Fig 3 and the 3-loop self-energy diagrams of Fig 4. We compute the \( C \) and \( D \) functions as outlined in [12]. Sample CUDA code for the integrand of Fig 4(e) (with \( N = 8 \)) transformed from \( \mathcal{S}_7 \) with a \( \tanh \) type transformation to the cube \( \mathcal{C}_7 \) is given in Appendix C.1. The integral approximation will be multiplied in the main program with the factor \( \text{pow}(c+\text{half}, d) = (\frac{c}{2})^7 \) (from the Jacobian of the \( \tanh \) transformation in Eq (7)).

Similarly, Appendix C.2 lists the integrand for Fig 3(b) (with \( N = 6 \)) as a CUDA device function using \( \Psi_6 \) after a transformation from the simplex \( \mathcal{S}_5 \) to the unit cube \( \mathcal{C}_5 \). Here the constant \( \text{con} = (\frac{16}{2^3})^3 = (3.2)^3 \) arises from the Jacobian of the \( \Psi_6 \) transformation in Eq (6).
4.2. GPU results

The numerical calculations are performed on a cluster node of the thor cluster at WMU. The node has dual 8-core, 2.6 GHz Intel Xeon E5-2670 processors, and 128 GB of memory, and furthermore an NVIDIA Tesla K20m GPU, which has 2496 CUDA cores and 4800 MB global memory.

Tables 1 and 2 report results for the 2-loop box diagrams of Fig 3 using lattice rules with \( n = 10, 100, 200, 350 \) and 400M (million) points, and with the tanh and Sidi’s \( \Psi_6 \) transformation, respectively. We list the exact values from [56] – except for the crossed box box diagram of Fig 3(e), where the integral value resulted from an extensive computation with large numbers of integrand evaluations using the adaptive ParInt code in long double precision [16]. The integral dimensions for the diagrams [2lb](a-e) are \( d = 4, 5, 6, 6, 6 \). The lattice rule sums are generated from a \( z \)-vector for dimension \( d_R = d \) and a \( z \)-vector for a dimension \( d_R > d \); the rule for the latter is applied by using the first \( d \) components of the generator.

For each result the absolute error and the time in seconds are reported. Note that, where the accuracy is listed as \( E_n(\varepsilon_{d_R}) \) in a column labeled \( d_R \), the rule for dimension \( d_R \) was used. In a number of cases the accuracy of the higher \( (d_R) \)-dimensional rule is comparable to (and occasionally better than) that of the \( d \)-dimensional rule. For each integral, the times for the same number of points \( n \) are very close for different \( d_R \); in any case the average time is given. The timings for corresponding cases in Table 1 (with tanh) and Table 2 (with \( \Psi_6 \)) are generally very similar. Overall the accuracies are somewhat better with

![Figure 4. [3LS] Sample 3-loop self-energy diagrams with massive internal lines, cf., Laporta [56]: (a) \( N = 7 \), (b) \( N = 7 \), (c) \( N = 8 \), (d) \( N = 8 \), (e) \( N = 8 \).](image-url)
in milliseconds (ms), as well as the parallel speedup $S$ with generators for dimensions $n$ in Appendix C.1. For most of the integrals the accuracy improved from the 10M to the 350 or 400M rule.

Therefore $\Psi_6$ was not further considered.

Tables 3 and 4 give results for the 3-loop self-energy diagrams of Fig 4, for which the integrals are of dimensions $d = 6, 6, 7, 7, 7$. In the columns labeled $d_E$ of these tables, somewhat better results obtained with generators for dimensions $d_L$ are listed (as $E_{c}(d_E)$), which are more accurate than the corresponding $d$-dimensional approximations in some cases. For the tanh results in Table 3, two results appear with a superscript "*". In these cases the tanh transformation with constant $c = 3$ resulted in an overflow, and $c = 2.5$ was used instead. Otherwise, overflows were not problematic using the type of coding shown in Appendix C.1. For most of the integrals the accuracy improved from the 10M to the 350 or 400M rule.

Parallel performance results are given in Table 5 for the [2lb(e)] (crossed box) diagram, and lattice rules for dimension 6. In each case the sequential time $T_{seq}$ and the parallel (GPU) time $T$ are reported in millisecond (ms), as well as the parallel speedup $S = T_{seq}/T$, for rules with $n = 10M, 100M, 200M$ and 400M points. The speedup measures the improvement in time when the program is executed in parallel, versus the sequential running time.

For the different numbers of points it emerges that speedups are achieved of around 500 for the lattice rules applied with the $\Psi_6$ transformation, and around 400 with the tanh transformation. The reason for the difference in speedup lies in the sequential times, which are somewhat higher for the $\Psi_6$ transformation than for tanh, whereas the parallel times are similar. As a consequence the speedup ratio of the sequential to the parallel time is higher for the $\Psi_6$ transformation. Excellent speedups are also obtained for other diagrams of the problem sets considered.

The approach combining lattice rules and transformations for these loop integrals performs well compared to other numerical methods. For the 2-loop crossed box problem using parallel adaptive integration in [23] with 600M to 300B function evaluations, the absolute accuracy estimate ranged from $2.9e-08$ to $1.1e-11$, obtained in execution times of about 1 second to 587 seconds, with 64 MPI (Message Passing Interface) processes running on four cluster nodes (16 per node) — compare to the GPU times up to the order of a second. Iterated adaptive integration, also explored in [23] and implemented as a multi-threaded program on one node, took considerably longer than the adaptive method. Even though

### Table 3. Results for 3-loop self-energy diagrams, LR with tanh transformation

| Diagram | Exact | $E_{c}$ | $E_{d}$ | $T(s)$ | $T_{seq}$ | $T(s)$ | $T_{seq}$ | $T(s)$ | $T_{seq}$ | $T(s)$ |
|---------|-------|--------|--------|--------|-----------|--------|-----------|--------|-----------|--------|
| 10M     | 5.0e-04 | 6.5e-05 | 0.03  | 3.2e-04 | 2.1e-04 | 0.03  | 2.6e-04 | 9.8e-04 | 0.04  | 8.6e-05 | 7.5e-06 | 0.04  | 5.6e-05 | 4.6e-06 | 0.04  |
| 100M    | 9.8e-06 | 1.1e-05 | 0.31  | 8.9e-06 | 7.9e-06 | 0.31  | 1.7e-05 | 2.1e-05 | 0.37  | 4.0e-06 | 4.5e-06 | 0.37  | 5.6e-07 | 7.7e-07 | 0.37  |
| 200M    | 1.5e-05 | 2.5e-06 | 0.63  | 3.1e-04 | 2.0e-04 | 0.62  | 8.9e-07 | 5.2e-07 | 0.73  | 1.4e-07 | 1.8e-06 | 0.75  | 7.5e-07 | 3.1e-07 | 0.75  |
| 350M    | 1.1e-05 | 5.7e-07 | 1.11  | 1.8e-05 | 3.0e-07 | 1.09  | 5.5e-07 | 2.7e-07 | 1.28  | 4.6e-07 | 5.5e-07 | 1.31  | 2.3e-07 | 1.8e-07 | 1.31  |
| 400M    | 3.2e-06 | 5.7e-07 | 1.26  | 3.5e-06 | 3.8e-07 | 1.24  | 3.0e-07 | 2.4e-07 | 1.46  | 2.4e-08 | 5.1e-08 | 1.49  | 1.9e-10 | 9.1e-08 | 1.49  |

### Table 4. Results for 3-loop self-energy diagrams, LR with $\Psi_6$ transformation

| Diagram | Exact | $E_{c}$ | $E_{d}$ | $T(s)$ | $T_{seq}$ | $T(s)$ | $T_{seq}$ | $T(s)$ | $T_{seq}$ | $T(s)$ |
|---------|-------|--------|--------|--------|-----------|--------|-----------|--------|-----------|--------|
| 10M     | 3.2e-06 | 5.9e-06 | 0.03  | 1.0e-05 | 5.9e-07 | 0.03  | 8.7e-05 | 4.8e-06 | 0.04  | 3.6e-06 | 2.2e-05 | 0.04  | 2.1e-05 | 1.8e-06 | 0.04  |
| 100M    | 2.5e-07 | 6.6e-07 | 0.31  | 2.7e-06 | 1.4e-06 | 0.31  | 5.3e-07 | 1.6e-06 | 0.37  | 2.3e-06 | 7.5e-07 | 0.37  | 6.9e-08 | 8.9e-08 | 0.36  |
| 200M    | 3.6e-07 | 7.6e-07 | 0.63  | 2.3e-05 | 6.1e-07 | 0.63  | 7.5e-08 | 5.0e-09 | 0.73  | 1.3e-07 | 2.3e-08 | 0.73  | 2.7e-08 | 2.7e-08 | 0.72  |
| 350M    | 1.0e-07 | 4.4e-08 | 1.10  | 2.7e-07 | 1.9e-07 | 1.10  | 1.4e-07 | 2.8e-08 | 1.31  | 4.3e-08 | 4.6e-09 | 1.28  | 4.8e-09 | 3.0e-08 | 1.26  |
| 400M    | 1.1e-07 | 6.9e-07 | 1.25  | 9.3e-06 | 3.5e-08 | 1.25  | 5.9e-08 | 7.8e-09 | 1.49  | 3.1e-08 | 5.6e-07 | 1.46  | 4.2e-08 | 2.2e-09 | 1.44  |
Table 5. Speedup results of LR with \( \tanh \) or \( \Psi_6 \) transformation

| \( n \)  | \( T_{seq}(ms) \) | \( T(m) \) | \( S \) | \( T_{seq}(ms) \) | \( T(m) \) | \( S \) |
|---|---|---|---|---|---|---|
| 10M | 12652.1 | 32.39 | 391 | 15840.4 | 31.55 | 502 |
| 100M | 125939 | 317.3 | 397 | 156570 | 311.7 | 502 |
| 200M | 251200 | 636.1 | 395 | 313278 | 623.6 | 502 |
| 400M | 503561 | 1271 | 396 | 630059 | 1247 | 505 |

the adaptive strategies are more versatile and may further improve the accuracy (up to a certain point) by increasing the number of region subdivisions, the lattice rule program for this application is far simpler and currently runs on one node plus GPU.

5. Conclusions
We presented an approach for the evaluation of loop integrals on GPUs for classes of 2-loop box and 3-loop self-energy Feynman diagrams. Lattice rules are particularly suited for GPU computations, and can be applied with transformations to deal with integrand singularities at the boundaries of the integration region. For the target application in this paper, Sidi’s \( \sin^m \)-transformation (with \( m = 6 \)) overall attained better accuracy than the \( \tanh \)-transformation, and the combined lattice rule and transformation techniques achieved excellent accuracy and parallel efficiency.

The goal of this work is to establish high speed numerical methods to derive precise results for important classes of Feynman loop integrals. As such it contributes to the development of computer simulations in particle physics, and has potential for extensions in this and other physics areas, such as in condensed matter physics where Feynman integrals play an important role.

Acknowledgments
We acknowledge the support from the National Science Foundation under Award Number 1126438, which funded the equipment used for the computations in this paper, as well as the funding by Grant-in-Aid for Scientific Research (17K05428) of JSPS, and by the Large Scale Simulation Program No.16/17-21 of KEK. We further express our appreciation for the interesting comments and suggestions by the reviewers of this paper.

Appendix A. Lattice generator vectors
Table A1 lists the generator vectors \( z \) used for the computations in this paper.

Appendix B. CUDA kernel for rule sum

```c
__global__ void lattice_kernel( int *z, double *q ) {
    // Input: generator z
    // Output: array q contains partial lattice rule sums over the blocks
    __shared__ double cache[threadsPerBlock]; // cache array is shared over block

    int tid = threadIdx.x + blockIdx.x * blockDim.x; // thread ID
    int cacheIndex = threadIdx.x;
    double temp = 0;
    double x[dim], temp1, temp2, intpart;
    int i;

    while (tid < n) { // add function evaluations for thread tid into temp
        for( i = 0; i < dim; i++) { // compute evaluation point x
            temp1 = (double) tid * z[i];
```
if (cacheIndex == 0) // store sum over block in q[blockIdx.x]
    q[blockIdx.x] = cache[0];

rp /= 2;
__syncthreads(); // synchronize threads after this stage of reduction
if (cacheIndex < rp)
    tid += blockDim.x * gridDim.x;

temp2 = templ/(double)n;
x[i] = modf(temp2, &intpart); // add function evaluation to temp

temp += fl(x); // synchronize threads before reduction

if (cacheIndex == 0) // store sum over block in q[blockIdx.x]
    q[blockIdx.x] = cache[0];

Table A1. Lattice rule generators

| d  | n   | z components |
|----|-----|--------------|
| 4  | 10M | (1, 2928962, 1859617, 3250721) |
| 4  | 100M| (1, 3827807, 3388112, 4595883) |
| 4  | 200M| (1, 76384079, 90140983, 20512317) |
| 4  | 350M| (1, 146940205, 127904721, 59229843) |
| 4  | 400M| (1, 154642716, 171950363, 128667067) |
| 5  | 10M | (1, 3675449, 4456704, 3844450, 4934806) |
| 5  | 100M| (1, 3827807, 3388112, 4595883) |
| 5  | 200M| (1, 76384079, 90140983, 20512317) |
| 5  | 350M| (1, 138694234, 154078417, 125929561) |
| 5  | 400M| (1, 154642716, 171950363, 128667067, 118799181) |
| 10 | 10M | (1, 2928962, 1859617, 4304839, 2689131) |
| 10 | 100M| (1, 3827807, 3388112, 4595883) |
| 10 | 200M| (1, 76384079, 90140983, 20512317) |
| 10 | 350M| (1, 146940205, 127904721, 59229843) |
| 10 | 400M| (1, 154642716, 171950363, 128667067) |

Lattice rule generators

<doi:10.1088/1742-6596/1136/1/012002>
Appendix C. CUDA device functions for sample integrands

Appendix C.1. Device function for \[3ls\](e) integrand with tanh transformation

```c
#include <math.h>

const int dim = 7;
const double C = 3.0;
const double half = 0.5;
const double emach = 5.0e-16; // machine accuracy

__device__ double fl(double x[]) {
    // Integrand for \[3ls\](e) with tanh transformation
    // Integration result will be multiplied with pow(C*half,7.0) in main()
    double x0,x1,x2,x3,x4,x5,x6,x7,x1sq,x4sq,x5sq,x6sq;
    double x12sq,x13sq,x12,x13,x1235,x0124,x4567,x02457;
    double arg[7],phip,dphi,deni,u2i,ai,t,tmp,dd,f0;
    int i;
    f0 = 0;
    dphi = 1.0;
    phip = 1.0;
    for(i = 0; i < dim; i++) {
        t = 2.0*x[i]-1.0;
        u2i = t*t;
        deni = 1.0-u2i;
        if(fabs(deni) <= emach) return(0);
        ai = C*t/deni;
        arg[i] = dphi*half*(tanh(ai)+1.0);
        tmp = deni*cosh(ai);
        phip = phip*dphi*(1.0+u2i)/tmp/tmp;
        dphi = dphi-arg[i];
    }
    if(phip == 0) return(0);
    x0 = arg[0];
    x1 = arg[1];
    x2 = arg[2];
    x3 = arg[3];
    x4 = arg[4];
    x5 = arg[5];
    x6 = arg[6];
    x7 = 1.0-x0-x1-x2-x3-x4-x5-x6;
    x1sq = x1*x1;
    x4sq = x4*x4;
    x5sq = x5*x5;
    x6sq = x6*x6;
    x12 = x1+x2;
    x13 = x1+x3;
    x1235 = x1+x2+x3+x5;
    x0124 = x0+x1+x2+x4;
    x4567 = x4+x5+x6+x7;
    x02457 = x0+x2+x4+x5+x7;
    x12sq = x12*x12;
    x13sq = x13*x13;
    dd = x02457*x1235*x4sq + x13aq*x4sq + x02457*x12sq*x4567
        - x0124*x02457*x1235*x4567 - x1sq*x1235*x4567 + 2.0*x1*x12*x13*x4567
        - x0124*x13sq*x4567 + 2.0*x02457*x12*x6*x5 + 2.0*x1*x13*x4*x5
        + x124*x02457*x5sq + x1sq*x5sq + 2.0*x1*x1235*x4*x6 - 2.0*x12*x13*x4*x6
        + 2.0*x1*x12*x5*x6 - 2.0*x0124*x13*x5*x6 + x12sq*x6sq - x0124*x1235*x6sq;
    if(fabs(dd) > 0.0) f0 = 128.0*phip/dd/dd;
    return f0;
}
```
Appendix C.2. Device function for $[2lb](b)$ integrand with $\Psi_6$ transformation

```c
#include <math.h>

#define pi 3.14159265358979323846  
#define pi60 188.49555921538759430760  
#define con 335.54432

const int dim = 5;

__device__ double f1( double x[] ) {
    //Integrand for $[2lb](b)$ with $\Psi_6$ transformation
    double x1,x2,x3,x4,x5,x6,dd,f0;
    double arg[5],phi,pi;
    int i;

    f0 = 0;
    phi = 1.0;
    for(i = 0; i < dim; i++) {
        pi = pi*x[i];
        arg[i] = x[i]+(-45*sin(2*pi)+9*sin(4*pi)-sin(6*pi))/pi60;
        phi = phi*phi*phi*(sin(pi),6);
    }
    if(phi == 0) return(0);
    x1 = arg[0]; // Simplex-to-cube transformation
    x2 = (1.0-x1)*arg[1];
    x3 = (1.0-x1-x2)*arg[2];
    x4 = (1.0-x1-x2-x3)*arg[3];
    x5 = (1.0-x1-x2-x3-x4)*arg[4];
    x6 = 1.0-x1-x2-x3-x4-x5;
    phi = phi*(1.0-x1)*(1.0-x1-x2)*(1.0-x1-x2-x3)*(1.0-x1-x2-x3-x4);
    if(phi == 0) return(0);
    dd = (x1*x1+x5*x5+x1*x5)*(x2+x3+x4+x6)
        +(x2*x2+x3+x6+x6+x2+x3+x2*x6+x3*x6)+(x1+x4+x5)
        +x3*x4*(x1+x2+x3+x5+x6)+3.0*x4*x5*x6+
        2.0*x1*x4*(x2+x3+x6)+2.0*x4*x5*(x2+x3);
    if(fabs(dd) > 0.0) f0 = con*phi/dd/dd;
    return f0;
}
```

References

[1] Aad G et al. 2012 Phys. Letts. B 716 1–29
[2] Chatrchyan S et al. 2012 Phys. Letts. B 716 30–61
[3] Butterworth J et al. 2013 Les Houches 2013: Physics at TeV colliders Standard Model Working Group report arXiv:1405.0167 [hep-ph], https://arxiv.org/abs/1405.0167
[4] Fuji K et al. 2017 Physics case for the 250 GeV stage of the International Linear Collider arXiv:1710.07621 [hep-ph], https://arxiv.org/abs/1710.07621
[5] de Doncker E, Shimizu Y, Fujimoto J and Yuasa F 2004 Computer Physics Communications 159 145–156
[6] de Doncker E, Shimizu Y, Fujimoto J, Yuasa F, Cucos L and Van Voorst J 2004 Nuclear Instruments and Methods in Physics Research A 539 269–273 hep-ph/0405098
[7] Yuasa F, de Doncker E, Fujimoto J, Hamaguchi N, Ishikawa T and Shimizu Y 2007 Precise numerical results of IR-vertex and box integration with extrapolation PoS (ACAT07) 687 arXiv:0709.0777v2 [hep-ph]
[8] Yuasa F, Ishikawa T, Fujimoto J, Hamaguchi N, de Doncker E and Shimizu Y 2008 Numerical evaluation of Feynman integrals by a direct computation method PoS (ACAT08) 122 arXiv:0904.2823 [hep-ph]
[9] de Doncker E, Fujimoto J, Kurihara Y, Hamaguchi N, Ishikawa T, Shimizu Y and Yuasa F 2010 Recursive box and vertex integrations for the one-loop hexagon reduction in the physical region PoS (ACAT10) 073
[10] de Doncker E, Fujimoto J, Hamaguchi N, Ishikawa T, Kurihara Y, Shimizu Y and Yuasa F 2010 Springer Lecture Notes in Computer Science (LNCS) 6017 139–154
[11] de Doncker E, Fujimoto J, Hamaguchi N, Ishikawa T, Kurihara Y, Ljubic M, Shimizu Y and Yuasa F 2010 Extrapolation algorithms for infrared divergent integrals arXiv:hep-ph/1110.3587, PoS (CPP2010)011
