

Large Scale Automatic Computations for Feynman Diagrams with up to Five Loops

E. de Doncker[1(✉)], F. Yuasa[2], O. Olagbemi[1], and T. Ishikawa[2]

[1] Western Michigan University, Kalamazoo, MI 49008, USA
{elise.dedoncker,omofolakunmiel.olagbemi}@wmich.edu
[2] High Energy Accelerator Research Organization (KEK), 1-1 OHO, Tsukuba, Ibaraki 305-0801, Japan
{fukuko.yuasa,tadashi.ishikawa}@kek.jp
http://www.cs.wmich.edu/elise

Abstract. We give results by automatic integration methods for finite and UV-divergent 4-loop diagrams and a finite 5-loop case with massless internal lines. Non-adaptive methods include DE (Double Exponential), and Quasi-Monte Carlo (QMC) techniques. The latter are based on optimal lattice rules, implemented in Cuda-C for GPUs; or, for execution on PEZY/ Exascaler, the host program is written in C++ and the kernel is generated using the Goose compiler interface. DE is executed on similar hardware as QMC, with or without parallel libraries for MPI. Transformations are incorporated to alleviate or smoothen singularities on the boundaries of the domain. For adaptive integration we use the ParInt package layered over MPI on a cluster, as well as a new adaptive scheme that performs GPU evaluations of the cubature rules. For the UV-divergent diagram we apply a nonlinear extrapolation on a sequence of integral approximations generated for dimensional regularization. Some results are verified using computationally intensive symbolic/numerical evaluations with pySecDec.

Keywords: Feynman loop diagrams · Adaptive/non-adaptive integration · High performance computing · Dimensional regularization

1 Introduction

1.1 Feynman Loop Integrals

After the discovery of the Higgs particle in the Large Hadron Collider (LHC) experiment at CERN, the Standard Model of particle physics has been investigated precisely to find a tiny deviation of the experimental observations from theoretical predictions.

With marked improvements in the technology of high energy physics experiments, accurate theoretical predictions are required more than ever, with an increased importance of higher order corrections in the perturbation method.

© Springer Nature Switzerland AG 2020
O. Gervasi et al. (Eds.): ICCSA 2020, LNCS 12253, pp. 145–162, 2020.
https://doi.org/10.1007/978-3-030-58814-4_11

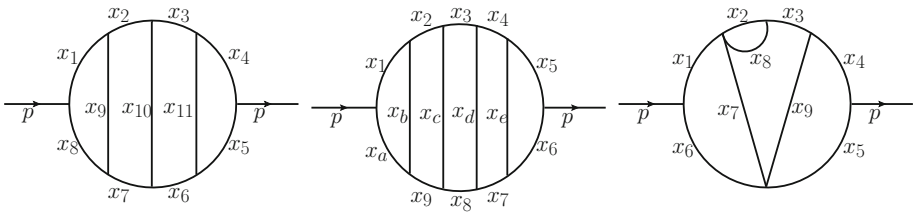

Fig. 1. (a) 4-loop $N = 11$ [4], (b) 5-loop $N = 14$ [4], (c) 4-loop $N = 9$ [4]

In particular, these will play a crucial role in the precise investigation of the Higgs sectors and electroweak interactions in both upcoming HL-LHC (High-Luminosity LHC) and future Linear Collider experiments.

In calculating higher order corrections to observables, the Feynman diagrammatic approach is commonly used today since a prescription for the approach has been well established. Feynman loop integrals emerge in the course of the calculations. The one-loop level has been studied completely in an analytic manner by 't Hooft and Veltman [18] and there are already several computer programs to evaluate one-loop integrals (such as LoopTools [17]). For corrections including multi-loop integrals with various mass and momentum values, a general and analytic treatment is not possible so that a numerical approach is mandatory. Various approaches have been proposed and their implementations have been published, for example, pySecDec [5] utilizes symbolic/algebraic manipulations and numeric calculations. It is further reported that Forcer [34], a FORM program, has been used for the computation of challenging loop problems such as the five-loop beta function for Yang-Mills theory with fermions [32,33].

As the number of loops increases, the number of dimensions of integration increases in general as well as the degree of difficulty. Thus numerical multi-dimensional integration is one of the key ingredients in the computation of higher order corrections. An excellent example of the successful application of large-scale numerical methods is the computation of the full two-loop electroweak corrections to the muon anomalous magnetic moment [19]. The methods presented in this paper are fully numerical and based on adaptive or non-adaptive multi-dimensional integration techniques, with the goal of enabling an accurate high-performance evaluation of Feynman Integrals at the multi-loop level with any masses and momenta.

A scalar Feynman integral in Feynman parameter (Euclidean) space can be represented by $\mathcal{F} = (4\pi)^{-\nu L/2}\mathcal{I}$ where

$$\mathcal{I} = \Gamma(N - \frac{\nu L}{2})\,(-1)^N \int_{\mathcal{C}_N} \prod_{r=1}^{N} dx_r\, \delta(1 - \sum x_r) \frac{C^{N-\nu(L+1)/2}}{(D - i\varrho C)^{N-\nu L/2}} \tag{1}$$

$$= \Gamma(N - \frac{\nu L}{2})\,(-1)^N \int_{\mathcal{S}_{N-1}} \prod_{r=1}^{N-1} dx_r\, \frac{C^{N-\nu(L+1)/2}}{(D - i\varrho C)^{N-\nu L/2}} \tag{2}$$

where L and N denote the number of loops and internal lines, respectively; C and D are polynomials determined by the topology of the corresponding diagram and physical parameters, and we let $\varrho = 0$ as D does not vanish in the interior of the domain for the current application. The space-time dimension is $\nu = 4 - 2\varepsilon$ where ε is a parameter that tends to zero for regularization. Whereas the integral representation (1) extends over the N-dimensional unit cube $\mathcal{C}_N$, the integral (2) over the $d = (N-1)$-dimensional unit simplex $\mathcal{S}_d = \{\mathbf{x} \in \mathcal{C}_d \mid \sum_{j=1}^{d} x_j \leq 1\}$ is obtained by eliminating the δ-function.

We will treat the diagrams of Fig. 1, including: (a) a finite 4-loop diagram ($N = 11$), (b) a finite 5-loop diagram ($N = 14$), and (c) a UV-divergent 4-loop diagram ($N = 9$).

1.2 UV Singularities

The polynomial C in the denominator may vanish at the boundaries of the domain, giving rise to an ultra-violet (UV) singularity in case the exponent $N - \nu(L+1)/2 = N - (2-\varepsilon)(L+1)$ is negative. We incur this situation for the diagram of Fig. 1(c) with $N = 9$ massless internal lines and $L = 4$ loops, where the integral satisfies an expansion of the form

$$\mathcal{I} = \mathcal{I}(\varepsilon) = \gamma(\varepsilon) \sum_{k \geq \kappa} C_k \varepsilon^k \tag{3}$$

with $\kappa = -1$ as given by [4]. For regularization we will generate an approximate sequence to $\{\varepsilon_\ell \mathcal{I}(\varepsilon_\ell)\}$ as $\varepsilon_\ell \to 0$ (i.e., space-time dimension $\nu \to 4$).

With the goal of obtaining the limit numerically, we use the ϵ-algorithm [41] for a nonlinear extrapolation, and an implementation based on the code of *qext()* from the QuadPack package [30]. The computation produces a triangular table, with a sequence based on integral approximations as the entry sequence in the first column. The *qext()* procedure computes a new lower diagonal when a new entry is available, together with a measure of the distance of each element from preceding neighboring elements. Then *qext()* returns the new lower diagonal element with the smallest value of the distance estimate as the extrapolation result.

1.3 Automatic Integration

Classic automatic integration software includes HALF [7], QuadPack [30], Adapt [14], Dcuhre [3] and Cuba [16]. An automatic integration method is supplied with a specification of the integral including the dimension, domain and integrand function, as well as an error tolerance and a bound on the allowed amount of work, such as the maximum number of integrand evaluations to be performed in the course of the integration. While the procedure acts as a black-box, the computation aims to deliver an approximation to the integral and an error estimate.

Defining the integrand on the domain $\mathcal{D}$ as a vector function with s components, $f : \mathcal{D} \subset \mathbb{R}^d \to \mathbb{R}^s$, the integral can be represented by $\mathcal{I} = \mathcal{I}f =$

$\int_{\mathcal{D}} f(\mathbf{x})\, d\mathbf{x}$. Using $\mathcal{Q}$ and E_a for the final integral approximation and absolute error estimate, respectively, a possible target accuracy requirement is

$$||\mathcal{Q} - \mathcal{I}|| \leq E_a \leq t = \max\{t_a, t_r||\mathcal{I}||\} \tag{4}$$

(in infinity norm), where t_a and t_r are the user-specified absolute and relative error tolerances. Unless otherwise noted it is assumed that $s = 1$.

1.4 Transformations

Since we rely on the loop integral representation (2) over a simplex domain, a transformation is needed from the simplex to the unit cube. Unless otherwise stated we transform the integral over $\mathcal{S}_d$ of (2) to $\mathcal{C}_d$ by the coordinate transformation

$$x_j = (1 - \sum_{k=1}^{j-1} x_k)\ y_j, \quad 1 \leq j \leq d \tag{5}$$

so that $\mathbf{y} \in \mathcal{C}_d$ and the Jacobian is $\prod_{j=1}^{d-1}(1 - \sum_{k=1}^{j-1} x_k)$.

For Fig. 1(c) we use the following simplex-to-cube transformation that also aims to reduce singular behavior on the boundaries,

$$\begin{aligned}
x_1 &= y_1\, y_2\, y_5\, y_6 & x_6 &= y_1\, y_2\, y_5\, y_{6m} \\
x_2 &= y_{1m}\, y_3\, y_4 & x_7 &= y_1\, y_2\, y_{5m} \\
x_3 &= y_{1m} y_{3m} & x_8 &= y_{1m}\, y_3\, y_{4m} \\
x_4 &= y_1 y_{2m} y_7 y_8 & x_9 &= y_1\, y_{2m}\, y_{7m} \\
x_5 &= y_1 y_{2m} y_7 y_{8m}
\end{aligned} \tag{6}$$

with $y_{jm} = 1 - y_j$, $\sum_{k=1}^{d+1} x_k = 1$ and Jacobian $y_1^5\, y_{1m}^2\, y_2^2\, y_{2m}^2\, y_3\, y_5\, y_7$.

Furthermore, while lattice rules are applied with a periodizing transformation of the integrand, we select a transformation that also suppresses singularities on the boundaries of the domain. In some cases these transformations also improve the performance of adaptive methods for dealing with boundary singularities. We make use of Sidi's $\sin^{\mathrm{m}}$-transformations [36] for $m = 4$ and 6,

$$\Psi_4(t) = t + (-8\sin(2\pi t) + \sin(4\pi t))/(12\pi), \quad \Psi_4'(t) = \frac{8}{3}\sin^4(\pi t) \tag{7}$$

$$\Psi_6(t) = t - (45\sin(2\pi t) - 9\sin(4\pi t) + \sin(6\pi t))/(60\pi), \quad \Psi_6'(t) = \frac{16}{5}\sin^6(\pi t) \tag{8}$$

Subsequently in this paper, the adaptive methods ParInt and ParAdapt will be discussed in Sects. 2 and 3, respectively. A double exponential (DE) transformation is used in a non-adaptive technique as described in Sect. 4.1. Quasi-Monte Carlo (QMC) approaches based on lattice rules are covered briefly in Sect. 4.2, also including embedded lattice rules and stochastic families of rules. Finally, results for the diagrams of Fig. 1 are given in Sect. 5.

2 Adaptive Multivariate Integration — ParInt

The ParInt package provides a parallel automatic integration method implemented in C and layered over MPI (Message Passing Interface [23]). ParInt allows for integration over hyper-rectangular and simplex regions. The adaptive strategy continually refines the integration domain by region subdivisions in a distributed task partitioning algorithm, cf. [10,13]. In view of the global termination criterion (4), the strategy adheres to a class of the meta-algorithm termed global adaptive [31]. With the notations of Sect. 1.3, the accuracy requirement to be attained for the integral $\mathcal{I}$ is given by (4), and the termination criterion tested in the course of the computations is of the form $E_a \leq t = \max\{t_a, t_r||\mathcal{Q}||\}$, where $\mathcal{Q}$ is the current overall integral approximation.

For estimating the integral and error over each subregion, the integration rules are linear combinations of function values, $\sum_{j=1}^{n} w_j f(\mathbf{x}_j)$. An integration rule is of polynomial degree of accuracy k if it integrates all polynomials of degrees through k exactly and does not integrate all polynomials of degree $k+1$ exactly. For hyper-rectangular regions, ParInt provides basic rules of polynomial degrees 7 and 9 [2,14].

The given domain is initially distributed over a number of processes (workers), each of which successively selects its largest error subregion, subdivides (bisects it) and integrates over the parts. The selected region is thereby deleted from the worker's local heap, and the children regions are inserted. The contributions of the selected region to the integral and to the error estimate are replaced by those of the children regions. The resulting differences are accumulated into updates for the overall integral and error estimate, and eventually sent to the controller process. Executed on a separate process, the controller may also perform as a worker.

The update messages from the workers further contain information on their busy or idle status, which is used to initiate load balancing, where work is sent from busy to idle or near-idle workers. As an additional feature, specifying a maximum heap size leads to maintaining a double-ended heap (deap) data structure where low-priority elements are deleted when adding regions, in order to maintain constant size once the maximum heap size is reached.

3 ParAdapt

ParAdapt, based on Adapt [14], targets global adaptive integration over hyper-rectangu- lar regions in moderate dimensions, say $d \geq 10$, by performing its (degree 7) rule evaluations on GPU [27,28]. The number of function evaluations per region, $2^d + 2d(d+2) + 1$, ranges between 1,265 and 33,279 for dimensions $d = 10$ to 15. Apart from the integration rule, fourth divided differences are computed through the region center in each coordinate direction, in order to gauge in which directions the integrand varies most. In ParAdapt the selected region can be subdivided into 2, 4 or 8 subregions of equal volume by halving the sides corresponding to the coordinate directions of the largest, second-largest

and third-largest divided difference. Subdivision into 16 is done by subdividing into 8 as above and choosing one of the remaining directions in a round-robin manner.

The rule evaluations over the children regions are performed on the GPU simultaneously. The rule points for the unit d-cube as the standard region are stored on the GPU and properly scaled for each subregion, i.e., the point $\tilde{\mathbf{p}}$ in $\mathcal{C}_d$ is mapped to the hyper-rectangle with edges from a_j to b_j in the jth coordinate direction as $p_j = a_j + (b_j - a_j)\tilde{p}_j$. Thus each rule point $\tilde{\mathbf{p}}$ has an image $\mathbf{p}$ in each of the children regions.

As outlined for CUDA programming in [35], the threads are laid out over a rectangular array where each row represents a block, and the width of a block corresponds to the number of threads per block. Figure 2 depicts a one-dimensional (x) grid, where we denote the number of blocks (array height) $g = gridDim.x$, and the number of threads per block (array width) $b = blockDim.x$. The blocks are indexed by $blockIdx.x = 0, 1, \ldots, g-1$ (row numbers). The thread ID within a block is $threadIdx.x$ (column number). Globally the threads are numbered consecutively in a row-major fashion as indicated by the *tid*s (global thread IDs) listed in the grid cells.

blockIdx.x	*tid*s			
0	0	1	$\cdots$	$b-1$
1	b	$b+1$	$\cdots$	$2b-1$
2			$\cdots$	
.				
.				
.				
$g-2$	$(g-2)b$	$(g-2)b+1$	$\cdots$	$(g-1)b-1$
$g-1$	$(g-1)b$	$(g-1)b+1$	$\cdots$	$gb-1$

Fig. 2. Threads layout in blocks; block IDs and global thread IDs (*tid*) are shown.

Then grid cell (i, j) corresponds to $tid = ib + j$ (i.e., grid cell $(blockIdx.x, threadIdx.x)$ corresponds to $tid = blockIdx.x * blockDim.x + threadIdx.x$). Following $tid = gb - 1$ the *tid*s wrap around, so that grid cell $(0, 0)$ corresponds to $tid = 0, bg, 2bg, \ldots$, and grid cell (i, j) to $tid = ib + j, ib + j + bg, ib + j + 2bg, \ldots$

We assign the function evaluations for a point $\mathbf{p}$ in successive subregions to successive *tid*s. Figure 3 shows two blocks, (*0* and *1*), with the global *tid*s, and *ptIndex* and *rgnIndex* assigned to each *tid*, for $r = numRgns$ regions ($0 \leq rgnIndex < r$) and assuming that r divides the number of threads per block b (this can be adjusted in case r does not divide b). The global *tid*s indexing the block elements are written in italics outside the blocks, as well as the block numbers. An example for $r = 4$ regions and $b = 256$ is also given. In this case, two blocks handle 128 points of the integration rule applied in four regions.

	0	*1*	⋯	*r−1*	*r*	*r+1*	⋯	*2r−1*	⋯			⋯	*b−1*	*tid*
0	0	1	⋯	$r-1$	0	1	⋯	$r-1$	⋯	0	1	⋯	$r-1$	*rgnIndex*
	0	0	⋯	0	1	1	⋯	1	⋯	$\frac{b}{r}-1$	$\frac{b}{r}-1$	⋯	$\frac{b}{r}-1$	*ptIndex*
1	0	1	⋯	$r-1$	0	1	⋯	$r-1$	⋯	0	1	⋯	$r-1$	*rgnIndex*
	$\frac{b}{r}$	$\frac{b}{r}$	⋯	$\frac{b}{r}$			⋯		⋯	$\frac{2b}{r}-1$	$\frac{2b}{r}-1$	⋯	$\frac{2b}{r}-1$	*ptIndex*
	b	*b+1*	⋯				⋯		⋯			⋯	*2b−1*	*tid*

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	⋯	*252*	*253*	*254*	*255*	*tid*
block 0	0	1	2	3	0	1	2	3	⋯	0	1	2	3	*rgnIndex*
	0	0	0	0	1	1	1	1	⋯	63	63	63	63	*ptIndex*
block 1	0	1	2	3	0	1	2	3	⋯	0	1	2	3	*rgnIndex*
	64	64	64	64	65	65	65	65	⋯	127	127	127	127	*ptIndex*
	256	*257*	*258*	*259*	*260*	*261*	*262*	*263*	⋯	*508*	*509*	*510*	*511*	*tid*

Fig. 3. Relationship between *tid*, *rgnIndex* and *ptIndex* for $r = numRgns$ regions. It is assumed that r divides the number of threads per block b. The second array shows an example for $r = 4$ regions and $b = 256$.

```
while(tid < N) {
    ptIndex = tid/numRgns;
    rgnIndex = tid − ptIndex ∗ numRgns;
    // get original point p̃ from point list at ptIndex
    // and region information at rgnIndex
    for (j = 0; j < d; j++)  p[j] = a[j] + ( b[j] − a[j] ) ∗ p̃[j];   // scaling
    funvls = f(p);  // evaluate function
    sumWeightsFunvls  +=  weights[ptIndex] ∗ funvls;   // rule sum accumulation
        ...  // accumulation for error estimate
    tid += blockDim.x ∗ gridDim.x;
}
```

Fig. 4. Kernel with function evaluation and accumulation of partial rule sum

For a given *tid* the algorithm can thus determine *ptIndex* to reference the integration points and weights, and *rgnIndex* for properly scaling the point and applying its contribution to the weighted sum of function values for the required region. Denoting the total number of points over the r regions resulting from a subdivision by $N = rn$, where n is the number of points in the rule sum, the rule evaluation loop of the CUDA kernel roughly conforms to the description of Fig. 4. Only the accumulation of the partial rule sum pertaining to the integral approximation (into $temp_sumWeightsFunvls[ptIndex]$) is shown. Further accumulations are needed for the error estimate.

The *sumWeightsFunvls* values across a block are stored in a cache array that is shared by all threads in the block. A regular reduction operation as outlined in [35] then leaves the results for the r regions in the first r threads in the block. The $(gridDim.x) \times r$ values over all blocks are returned from the kernel, and summed further into the r region results.

4 Non-adaptive Integration

4.1 Double-Exponential Method

The double-exponential method (DE) [38,39] transforms the one-dimensional integral

$$\int_0^1 f(x)\,dx = \int_{-\infty}^{\infty} f\left(\phi(t)\right)\phi'(t)\,dt, \qquad \text{using}$$
$$x = \phi\left(t\right) = \frac{1}{2}(\tanh\left(\frac{\pi}{2}\sinh\left(t\right)\right)+1), \qquad \phi'(t) = \frac{\pi\cosh\left(t\right)}{4\cosh^2(\frac{\pi}{2}\sinh\left(t\right))} \tag{9}$$

in order to obtain a very rapid decrease of the Jacobian in the transformed integrand as $|t|$ increases. This renders the integral suitable for a trapezoidal rule approximation, leading to the integral approximation

$$I_h^{N_{eval}} = \sum_{k=-N_-}^{k=N_+} f(\phi(kh))\,\phi'(kh) \tag{10}$$

with mesh size h. The number of function evaluations is $N_{eval} = N_- + N_+ + 1$. The formula (9) is used in a product manner for multivariate integration.

4.2 Lattice Rules

Rank-1 lattice rules provide an integral approximation over the unit cube of the form

$$\mathcal{Q}f = \mathcal{Q}(\mathbf{z},n)f = \frac{1}{n}\sum_{j=0}^{n-1} f(\{\frac{j}{n}\mathbf{z}\}) \tag{11}$$

where $\mathbf{z}$ is a generator vector with components $z \in \mathcal{Z}_n = \{1 \le z < n \mid gcd(z,n) = 1\}$, and $\{\mathbf{x}\}$ denotes a vector obtained by taking the fractional part of each component of $\mathbf{x}$ (see [37]). Classically n was taken prime [20,21] (so that $\mathcal{Z}_n = \{1,2,\ldots,n-1\}$); extensions to non-prime n are given in [24,26]. We precomputed the generators $\mathbf{z}$ for various n in [8,9,11,12], using the component by component (CBC) algorithm [25,26] either directly or via Lattice Builder [22].

We further use a sequence embedding of lattice rules defined in [37], based on a rank-1 rule Q_0 and where each rule of the sequence applies a scaled copy of Q_0 to the subregions obtained by subdividing $\mathcal{C}_d$ into m equal parts in each of r coordinate directions. Denoting

$$\mathcal{Q}_r f = \mathcal{Q}_r^{(m)}(\mathbf{z},n)f = \frac{1}{m^r n}\sum_{k_r=0}^{m-1}\cdots\sum_{k_1=0}^{m-1}\sum_{j=0}^{n-1} f\left(\{\frac{j}{n}\mathbf{z} + \frac{1}{m}(k_1,\ldots,k_r,0,\ldots,0)\}\right) \tag{12}$$

for n and m relatively prime, the points of Q_r are embedded in Q_{r+1} for $0 \le r < d$. Q_r has $m^r n$ points and is of rank r for $1 \le r \le d$, and Q_d is composed of m^d scaled copies of Q_0. An error estimate for Q_d is calculated using a sequence of rules $Q^{(i)}$, $1 \le i \le d$, of order $m^{d-1}n$ and embedded in Q_d [37].

A stochastic family of rules can be considered as proposed by Cranley and Patterson [6], by adding a random vector $\mathbf{\Delta}$ to the integration points,

$$\mathcal{Q}(\mathbf{z}, n, \mathbf{\Delta})f = \frac{1}{n}\sum_{j=0}^{n-1} f(\{\frac{j}{n}\mathbf{z} + \mathbf{\Delta}\}) \tag{13}$$

Averaging over a number of randomized (shifted) rule sums of the form (13) where $\mathbf{\Delta}$ has a multivariate uniform random distribution on $\mathcal{C}_d$ yields an integral approximation that has the integral $\mathcal{I}f$ as its expected value [37]. An error estimate is given in terms of the standard deviation.

5 Numerical Results

5.1 4-Loop Finite Diagram

The 4-loop diagram $G[4b]$ from [4] shown in Fig. 1(a), with $N = 11$ massless internal lines and integrand $C^{1+5\varepsilon}/D^{3+4\varepsilon}$ in (2), corresponds to an expansion

$$\mathcal{I}(\varepsilon) = G[4b] \sim -(-s)^{-3-4\varepsilon}\Gamma(3+4\varepsilon)\,(C_0 + C_1\varepsilon + \ldots) \tag{14}$$

For the finite diagram, the constant C_0 in (14) can be computed by setting $\varepsilon = 0$. The analytic value is $C_0 = 35\,\zeta(7) \approx 35.2922$. The value given for C_0 in [4] is 35.10. In [11] we reported 35.16 computed by a lattice rule with about 400M (million) points after a Sidi Ψ_6 transformation. Table 1 gives results by various lattice rules and embedded lattice rules (for $m = 1$ or 2 in (12)) on GPU and PEZY accelerators.

The GPU program is implemented in CUDA C and executed on the *thor* cluster at WMU, where the host process runs on a node with Intel Xeon E5-2670, 2.6 GHz dual CPU, and the lattice rule is evaluated on a Kepler-20m GPU, using 64 blocks and 512 or 1024 threads per block. The K20m has 2496 CUDA cores and 4800 MBytes of global memory. The lattice generator vector $\mathbf{z}$ is precomputed and communicated as a one-dimensional array of length d from the main program to the CUDA kernel. GPU results for lattice rules ($m = 1$) are given for $n = 400$M and 5^{13} points, and embedded versions with $m = 2$ for $n = 10$M and 100M. Note that the multiple M represents the next higher prime rounded down to the indicated multiple of a million function evaluations. Apart from the integral approximation, an absolute error estimate is also listed for $m = 2$. Furthermore, the implementation of the singular integrand C/D^3 requires testing that the denominator does not vanish before performing the division. We test $|D|^3 > Thr$, where (threshold) Thr is given in Table 1. A small positive value needs to be assigned particularly for $m > 1$, in view of the proximity of rule points to the boundaries of the domain.

Further results in Table 1 are obtained on the Suiren2 system of the High Energy Accelerator Research Organization (KEK) in Tsukuba, Japan, which is based on the PEZY-SC2 accelerator [29,40] and uses direct liquid immersion cooling. The configuration has 48 Xeon D-16C 1.3 GHz nodes and 8 PEZY-SC2

Table 1. Results of lattice and embedded lattice rules for $G[4b]$

n	m	*Result*	*Time[s]*	*Accel.*	*Thr*
400M	1	35.162005	1.83	K20m	0
5^{13}	1	35.163998	5.56	K20m	0
10M	2	35.265933	86.85	K20m	1.0 e-48
100M	2	35.279573	558.75	K20m	1.0 e-48
100M	2	35.279573	8.85	SC2	1.0 e-48
400M	2	35.289696	28.77	SC2	1.0 e-60
5^{13}	2	35.272921	83.82	SC2	1.0 e-48

boards per node. The node interconnect is Infiniband EDR. Each PEZY-SC2 board has 64 GB of memory, and the total amount of memory is 26,880 GB. The PEZY-SC2 programming model utilizes a kernel in an OpenCL-like language (PZCL) and a host program in C++. However, we generated the kernels via the Goose compiler interface [15], which takes a program instrumented with Goose compiler directives to parallelize the code. The SC2 results in Table 1 are obtained using 8 nodes and 8 tasks/node.

Table 2 provides results from ParAdapt on K20m GPU for $r = 4$, 8 and 16 regions per subdivision ($\#R = r$), various total numbers of integrand evaluations ($\#Ev$ in B = billion), numbers of blocks ($\#Bks = g$) and threads per block ($\#TpB = b$) (where the notations r, g and b were introduced in Sect. 3). Entries with multiple values of $\#Bks$ and $\#TpB$ (separated by commas) are included with execution times (separated by '/') or a range of times.

All Table 2 runs are for threshold $Thr = 0$. Results for ParAdapt with up to 10B evaluations and $Thr = 1.0\,\text{e-}48$ were included in [27]. This typically yields slightly lower integral approximations than $Thr = 0$, e.g., with 10B evaluations and $Thr = 1.0\,\text{e-}48$ the result is 35.278714, and with $Thr = 1.0\,\text{e-}60$ the result is 35.298195. Integral approximations for high numbers of evaluations are reported for comparison and do not change significantly, but the error estimate (bound) decreases. Some ParInt (long double) results obtained on the *thor* cluster with 64 processes on 4 nodes are also given.

For confirmation we further ran pySecDec 1.4.3 [5], which gave: 35.2379 ± 0.0175 using Vegas, and 35.2906 ± 0.0013 with QMC Korobov rank-3. With pySecDec 1.4.1 we previously calculated 35.2417 ± 0.0176 with Vegas. The number of sectors generated was 1968.

5.2 5-Loop Finite Diagram

The integrand for the 5-loop diagram $G[5]$ of Fig. 1(b) with $N = 14$ massless internal lines is $C^{2+6\varepsilon}/D^{4+5\varepsilon}$ in (2), and the expansion given in [4] is

$$\mathcal{I}(\varepsilon) = G[5] \sim (-s)^{-4-5\varepsilon}\Gamma(4+5\varepsilon)\,(C_0 + \ldots) \tag{15}$$

Table 2. Results of adaptive methods for $G[4b]$

ParAdapt						
$\#R$	$\#Ev[B]$	*Result*	E_a	*Time[s]*	$\#Bks$	$\#TpB$
4	15	35.329827	1.430 e-01	598−784	10,20,32,64	128,256,512
4	100	35.302547	4.828 e-02	8795	10	512
4	500	35.294968	1.931 e-02	32950	10	512
8	1	35.152057	9.861 e-01	28.0/27.1	64/20	128/512
8	10	35.307276	2.416 e-01	267.8/257.3	64/20	128/512
8	100	35.300016	5.906 e-01	2618/2517	64/20	128/512
8	500	35.294643	2.270 e-02	12897/20148	64/20	128/512
8	800	35.293851	1.722 e-02	20944	64	128
16	10	35.303108	3.956 e-01	166−193	64,40	128,512,256
16	100	35.303019	1.015 e-01	1695	64	256
16	200	35.299403	6.732 e-02	3263	64	256
16	500	35.296194	3.916 e-02	8481	64	256
16	800	35.294992	2.966 e-02	13656	64	256
ParInt						
	100	35.326279	1.50 e-01	3913		
	500	35.301808	5.98 e-02	19585		
	1000	35.297771	3.97 e-02	39195		
	2000	35.295374	2.61 e-02	78354		

with $C_0 = 40.53$. For the finite diagram we let $\varepsilon = 0$. Our computational experiments indicate a somewhat higher answer than 40.53, i.e., ≈ 42 rounded.[1] The analytic value is $42\,\zeta(9) \approx 42.0844$.

Table 3 displays results of large ParAdapt runs on Intel Xeon E5-2670, 2.6 GHz dual CPU with K20m (on *thor*), for 4 or 8 regions evaluated simultaneously on the GPU. The integral approximations, estimated absolute and relative errors, and times in seconds are shown. Sidi's Ψ_4 transformation was used. Some long double ParInt runs on 4 nodes/64 processes of the *thor* cluster are also listed.

With pySecDec 1.4.1 [5] we calculated an integral value of 41.9105 ± 0.0460 using Vegas; 27320 sectors were produced. The elapsed times were roughly around 40 h to generate the sectors, 40 h to compile, and 20.5 h for the integration. Using QMC Korobov rank-3 delivered the following results:

(i) 42.0941 ± 0.0221 obtained with pySecDec as an average over 32 randomized shifts of the form (13) in the range of 41.7559 to 42.3096, on AMD EPYC 7501 using 128 threads, in a computation time of about 14 days;

[1] For those interested in English literature, "42" is the answer to the Ultimate Question of Life, the Universe and Everything, cf., Douglas Adams, "The Hitchhiker's Guide to the Galaxy" [1].

Table 3. Results of adaptive methods for $G[5]$

ParAdapt					
r	*#Ev[B]*	*Result*	E_a	E_r	*Time[s]*
4	400	42.585786	1.183 e+00	2.778 e-02	5858
4	800	42.441678	8.972 e-01	2.114 e-02	10643
4	2000	42.323422	6.061 e-01	1.432 e-02	41105
4	4000	42.257989	4.401 e-01	1.041 e-02	67445
8	800	42.451987	9.765 e-01	2.300 e-02	11728
8	1200	42.387342	8.246 e-01	1.945 e-02	19593
8	2000	42.324564	6.621 e-01	1.564 e-02	19593
8	2500	42.300792	5.999 e-01	1.418 e-02	24671
8	3000	42.283284	5.527 e-01	1.307 e-02	29956
8	4000	42.257705	4.843 e-01	1.146 e-02	53734
ParInt					
	200	43.318466	1.59 e+00	3.670 e-02	8060
	1000	42.606834	8.97 e-01	2.105 e-02	40574
	2000	42.438615	7.06 e-01	1.666 e-02	81612

(ii) 42.1227 ± 0.0215, averaged over 32 shifts ranging from 41.891 to 42.3492, and run on Intel(R) Xeon(R) CPU E5-2687W v4 @ 3.00 GHz using 48 threads. The outputs for both (i) and (ii) include intermediate status reports; (ii) has one "status = exceeded max number of iterations" while (i) has six.

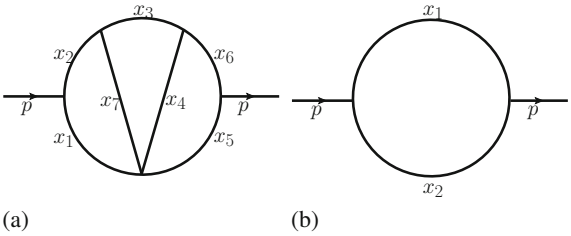

Fig. 5. Diagrams constituting Fig. 1(c)

5.3 4-Loop UV-Divergent Diagram

The diagram corresponding to $G[4a]$ in [4] and shown in Fig. 1(c) has $N = 9$ massless internal lines, leading to the integrand $C^{-1+5\varepsilon}/D^{1+4\varepsilon}$ in (2) and the

expansion

$$\mathcal{I}(\varepsilon) = G[4a] \sim -\frac{1}{\varepsilon}(-s)^{-1-4\varepsilon}\Gamma(1+4\varepsilon)\,B(1-\varepsilon,1-\varepsilon)(C_{-1}+C_0\varepsilon+C_1\varepsilon^2+\ldots) \tag{16}$$

where $C_{-1} = 20\,\zeta(5) \approx 20.73855510$ and $C_0 = 50\,\zeta(6) - 80\,\zeta(5) - 4\,\zeta(3)^2 \approx -37.86683051$. With reference to Fig. 1(c), the diagram can be considered as consisting of two parts, given by Fig. 5(a) and (b). Figure 5(a) corresponds to the diagram 6(a) in [13] and gives rise to $C_{-1}+\ldots$, while the second part corresponds to $\frac{1}{\varepsilon}B(1-\varepsilon,1-\varepsilon)$ in Eq. (16). In [13] we were able to compute the integral for diagram Fig. 5(a) as $20.73855508 \approx 20.73855510$ with $E_a = 9\times10^{-8}$ by ParInt using 50B evaluations in 90 s (double) and 181 s (long double precision).

Our goal is to approximate the factor C_{-1} of $1/\varepsilon$ for the combined diagram of Fig. 1(c) directly. For an examination of the integrand, the polynomials C an D in the Feynman integral representation (2) are given by

$$\begin{aligned} C &= x_{28}x_{459}(x_{16}x_{37}+x_3x_7) + x_{167}(x_2x_{459}x_8 + x_{28}x_{45}x_9)\\ D &= x_2x_8(x_{45}x_6x_7 + x_4x_5x_{167} + x_{16}x_5x_9 + x_{56}x_7x_9 + x_1x_6x_{459})\\ &\quad + x_{28}((x_1x_5+x_4x_6)x_7x_9 + x_3(x_{16}x_4x_5 + x_1x_{45}x_6 + (x_4x_5+x_{45}x_6)x_7\\ &\quad + (x_{16}x_5+x_1x_6)x_9 + x_{56}x_7x_9) + x_4x_5(x_7x_9+x_{16}x_{79}) + x_1x_6(x_7x_9+x_{45}x_{79})) \end{aligned} \tag{17}$$

with $x_9 = 1 - \sum_{k=1}^{8} x_k$ and $x_{i_1\ldots i_j} = \sum_{k=i_1}^{i_j} x_k$ (where the indices i_1 to i_j are in increasing order). It can be seen that C and D vanish at boundaries of the 8-dimensional unit simplex, including: $C = D = 0$ if $x_2 = 0$ and $x_8 = 0$, or if $x_1 = 0$, $x_6 = 0$ and $x_7 = 0$, or if $x_4 = 0$, $x_5 = 0$ and $x_9 = 0$. Upon further inspection, the transformation (6) has a smoothening effect. After substitution, the C and D polynomials are factored as

$$\begin{aligned} C &= y_1^2\, y_{1m}\, y_2\, y_{2m}\, y_3\, A_1\\ D &= y_1^3\, y_{1m}\, y_2\, y_{2m}\, y_3\, A_2 \end{aligned} \tag{18}$$

where A_1 and A_2 are polynomials of the y_i variables. This produces a factor

$$\tilde{f} = \frac{(y_1^2\, y_{1m}\, y_2\, y_{2m}\, y_3)^{-1+5\varepsilon}}{(y_1^3\, y_{1m}\, y_2\, y_{2m}\, y_3)^{1+4\varepsilon}} = y_1^{-5-2\varepsilon}(y_{1m}\, y_2\, y_{2m}\, y_3)^{-2+\varepsilon} \tag{19}$$

that is partially cancelled out by the Jacobian $J = y_1^5\, y_{1m}^2\, y_2^2\, y_{2m}^2\, y_3\, y_5\, y_7$ in the transformed integrand, since

$$\tilde{f}J = y_1^{-2\varepsilon}(y_{1m}\, y_2\, y_{2m})^{\varepsilon} y_3^{-1+\varepsilon}\, y_5\, y_7. \tag{20}$$

As $\varepsilon \to 0$, the factor $y_3^{-1+\varepsilon}$ is especially problematic, and it determines the integrand behavior near $y_3 = 0$. This can be addressed by a simple transformation of the form

$$x = \varphi(t) = t^{\frac{1}{\varepsilon}} \tag{21}$$

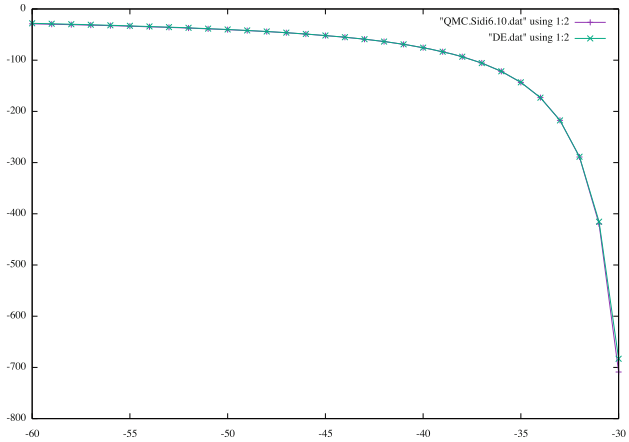

Fig. 6. Integration results $\varepsilon\mathcal{Q}(\varepsilon)$ as a function of $-\ell$ for $\varepsilon = 1.05^{-\ell}$

which alleviates the singularity (assuming g is smooth) in the integral

$$\int_0^1 x^{-1+\varepsilon} g(x)\, dx = \int_0^1 t^{\frac{-1+\varepsilon}{\varepsilon}} \varphi'(t)\, g(\varphi(t))\, dt$$
$$= \frac{1}{\varepsilon} \int_0^1 t^{-\frac{1}{\varepsilon}+1}\, t^{\frac{1}{\varepsilon}-1} g(\varphi(t))\, dt = \frac{1}{\varepsilon} \int_0^1 g(\varphi(t))\, dt \qquad (22)$$

For example, the transformation $x = t^2$ eliminates the singularity in $\int_0^1 1/\sqrt{x}\, dx$.

One numerical approach for regularization replaces ε by a small value; Fig. 6 displays integration values $\varepsilon\mathcal{Q}(\varepsilon)$ as ε decreases (from right to left). The horizontal axis is labeled with the exponents $(-\ell)$ of $\varepsilon = 1.05^{-\ell}$ and the vertical axis shows the approximations converging to $-C_{-1}$. Data plotted as '+' are obtained with QMC for the original integrand with C and D given by (17), and a lattice rule with 5^{13} points [12] after a Ψ_6 transformation. The '×' values are computed with DE, cf. (10) with an even number of evaluations $N_{evals} = 128$, mesh size $h = 0.052565$ for the third dimension, and $N_{evals} = 32$, mesh size $4h$ for the other dimensions, and the integrand is coded using transformation (6) (without (21)). Figure 6 shows good agreement of the two methods for the range of $\varepsilon = 1.05^{-30} \approx 0.231378$ to $1.05^{-60} \approx 0.0535357$.

Table 4 gives a nonlinear extrapolation by the ϵ-algorithm [41] to $\varepsilon\mathcal{I}(\varepsilon)$ as $\varepsilon \to 0$ for the $G[4a]$ expansion (16). The entry sequence $\varepsilon_\ell \mathcal{Q}(\varepsilon_\ell)$ is computed with ParInt in long double precision, for the geometric sequence of $\varepsilon_\ell = 1.2^{-\ell}, \ell = 12, 13, \ldots$. ParInt is executed using 64 processes on 4 cluster nodes over MPI with 16 procs/node, and the maximum number of integrand evaluations set at 100B. The cube-to-simplex transformation (6) is applied, together with the substitution (21) in the y_3-direction for improving the accuracy. In Table 4, the entry sequence is given in the first column, followed by the corresponding absolute and relative error estimates in columns 2 and 3, respectively, the extrapolation results in columns 4–5, and the time in seconds in column 6.

Table 4. Extrapolation to $\varepsilon\mathcal{I}(\varepsilon)$ as $\varepsilon \to 0$ for $G[4a]$ diagram

$-\ell$	$R_\ell = -\varepsilon_\ell \mathcal{Q}(\varepsilon_\ell)$	E_a	E_r	*Extr. #*	*Extrapol. result*	*Time (s)*
−12	53.45672718118757	0.0461	0.000862			1004.5
−13	43.22634363076326	0.0120	0.000278			1013.6
−14	38.32164736111470	0.0153	0.000400	1	33.804663	998.2
−15	33.26266821115664	0.00917	0.000276	2	199.14802	1000.7
−16	30.41639542582190	0.00631	0.000207	3	26.755147	999.5
−17	28.33223869037645	0.00632	0.000223	4	22.632701	1001.0
−18	26.76557914867488	0.00555	0.000208	5	21.977077	1000.3
−19	25.56825964014690	0.00610	0.000238	6	21.354362	997.8
−20	24.63846007737136	0.00556	0.000226	7	20.423179	995.1
−21	23.90564850811475	0.00492	0.000206	8	20.470130	994.2
−22	23.32114992156617	0.00356	0.000153	9	20.145763	1001.1
−23	22.85256745147067	0.00304	0.000133	10	20.232313	1018.1
−24	22.47414242303327	0.00267	0.000119	11	20.149453	1025.0
−25	22.16670775499503	0.00228	0.000103	12	20.733395	1023.4
−26	21.91616295171026	0.00182	8.29 e-05	13	20.751014	1025.0

Note that the extrapolated results are selected from the extrapolation table by the ϵ-algorithm procedure. Furthermore, the differences between consecutive entry elements R_ℓ increase at the $4th$ entry element ($R_4 - R_3 \approx 5.06$) and decrease from there on, contributing to the spurious behavior at the $2nd$ extrapolation. However, the ϵ-algorithm achieves the extrapolation to $\varepsilon\mathcal{I}(\varepsilon)$, notwithstanding the steep behavior of the integrand over the small range of ε.

On a programming note, obtaining results for this type of integration problem with vanishing denominators requires that checks are implemented to protect against arithmetic exceptions in the function subprogram (such as division by 0). The integrand is set to zero in the vicinity of the singularity. However, the computation is sensitive to the value of the threshold as the function is evaluated for values of the denominator closer to zero, so that tests are needed that take the machine constants into account, such as the underflow and overflow numbers and machine precision.

6 Conclusions

We improved on numerical results in the literature for the loop integrals corresponding to two finite diagrams, 4-loop with $N = 11$ and 5-loop with $N = 14$, and obtained results for a 4-loop UV-divergent diagram with $N = 9$ internal lines. We found good agreement between different tools, using adaptive (ParInt and ParAdapt) and non-adaptive (DE and QMC) integration. ParAdapt, which performs its rule evaluations on GPU, appears to be a viable tool for dimensions

≥ 10. For lower dimensions, the number of points in the degree 7 rules is fairly low to justify the execution on GPU, although in future versions this may be addressed, e.g., by new adaptive schemes that enable generating multiple regions more effectively for the GPU application. Good agreement was also found with recent runs of the symbolic/numeric pySecDec system [5], which took multiple days with MC, or two weeks with QMC for the 5-loop diagram.

Acknowledgments. We acknowledge the support from the National Science Foundation under Award Number 1126438 that funded the cluster used for the computations with ParInt and ParAdapt in this paper. Furthermore we rely on the Grant-in-Aid for Scientific Research (17K05428) of JSPS, and on partial support by the Large Scale Computational Sciences with Heterogeneous Many-Core Computers, Grant-in-Aid for High Performance Computing with General Purpose Computers from MEXT (Ministry of Education, Culture, Sports, Science and Technology-Japan). We also sincerely thank the reviewers of this paper for their valuable comments.

References

1. Adams, D.: The Hitchhiker's Guide to the Galaxy. Pocket Books, Simon & Shuster, Inc. (1981). ISBN 0-671-52721-5
2. Berntsen, J., Espelid, T.O., Genz, A.: An adaptive algorithm for the approximate calculation of multiple integrals. ACM Trans. Math. Softw. **17**, 437–451 (1991)
3. Berntsen, J., Espelid, T.O., Genz, A.: Algorithm 698: DCUHRE-an adaptive multidimensional integration routine for a vector of integrals. ACM Trans. Math. Softw. **17**, 452–456 (1991)
4. Binoth, T., Heinrich, G.: Numerical evaluation of multi-loop integrals by sector decomposition. Nucl. Phys. B **680**, 375 (2004). hep-ph/0305234v1
5. Borowka, S., Heinrich, G., Jahn, S., Jones, S.P., Kerner, M., Schlenk, J.: A GPU compatible quasi-Monte Carlo integrator interfaced to pySecDec. Comput. Phys. Commun. **240**, 120–137 (2019). Preprint: arXiv:1811.11720v1 [hep-ph]. https://arxiv.org/abs/1811.11720. https://doi.org/10.1016/j.cpc.2019.02.015
6. Cranley, R., Patterson, T.N.L.: Randomization of number theoretic methods for multiple integration. SIAM J. Numer. Anal. **13**, 904–914 (1976)
7. De Ridder, L., Van Dooren, P.: An adaptive algorithm for numerical integration over an n-dimensional cube. J. Comput. Appl. Math. **2**(3), 207–210 (1976)
8. de Doncker, E., Almulihi, A., Yuasa, F.: High speed evaluation of loop integrals using lattice rules. J. Phys.: Conf. Series (JPCS) IOP Series **1085**, 052005 (2018). http://iopscience.iop.org/article/10.1088/1742-6596/1085/5/052005
9. de Doncker, E., Almulihi, A., Yuasa, F.: Transformed lattice rules for Feynman loop integrals. J. Phys.: Conf. Series (JPCS) IOP Series **1136**, 012002 (2018). https://doi.org/10.1088/1742-6596/1136/1/012002
10. de Doncker, E., Genz, A., Gupta, A., Zanny, R.: Tools for distributed adaptive multivariate integration on NOW's: PARINT1.0 release. In: Supercomputing 1998 (1998)
11. de Doncker, E., Yuasa, F., Almulihi, A.: Efficient GPU integration for multi-loop Feynman diagrams with massless internal lines. In: Okada, H., Atluri, S.N. (eds.) ICCES 2019. MMS, vol. 75, pp. 737–747. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-27053-7_62

12. de Doncker, E., Yuasa, F., Almulihi, A., Nakasato, N., Daisaka, H., Ishikawa, T.: Numerical multi-loop integration on heterogeneous many-core processors. In: The Journal of Physics: Conference Series (JPCS), IOP Series (ACAT 2019) (2019, to appear)
13. de Doncker, E., Yuasa, F., Kato, K., Ishikawa, T., Kapenga, J., Olagbemi, O.: Regularization with numerical extrapolation for finite and UV-divergent multi-loop integrals. Comput. Phys. Commun. **224**, 164–185 (2018). https://doi.org/10.1016/j.cpc.2017.11.001
14. Genz, A., Malik, A.: An adaptive algorithm for numerical integration over an n-dimensional rectangular region. J. Comput. Appl. Math. **6**, 295–302 (1980)
15. Goose - GRAPE9-MPx - for goose version 1.5.0: K & F Computing Research Co. (2014). (in Japanese)
16. Hahn, T.: Cuba - a library for multidimensional numerical integration. Comput. Phys. Commun. **176**, 712–713 (2007). https://doi.org/10.1016/j.cpc.2007.03.006
17. Hahn, T., Pérez-Victoria, M.: Automated one-loop calculations in four and D dimensions. Comput. Phys. Commun. **118**(2–3), 153–165 (1999). hep-ph/9807565
18. 't Hooft, G., Veltman, M.: Scalar one-loop integrals. Nucl. Phys. B **153**, 365–401 (1979)
19. Ishikawa, T., Nakazawa, N., Yasui, Y.: Numerical calculation of the full two-loop electroweak corrections to muon (g-2). Phys. Rev. D **99**, 073004 (2019)
20. Korobov, N.M.: The approximate computation of multiple integrals. Dokl. Akad. Nauk SSSR **124**, 1207–1210 (1959). (Russian)
21. Korobov, N.M.: Properties and calculation of optimal coefficients. Doklady Akademii Nauk SSSR **132**, 1009–1012 (1960). Russ. Eng. trans. Soviet Math. Doklady **1**, 696–700
22. L'Equyer, P., Munger, D.: Algorithm 958: lattice builder: a general software tool for constructing rank-1 lattice rules. ACM Trans. Math. Softw. **42**(2), 15:1–15:30 (2016)
23. MPI. http://www-unix.mcs.anl.gov/mpi/index.html
24. Niederreiter, H.: Existence of good lattice points in the sense of hlawka. Monatshefte für Mathematik **86**, 203–219 (1978)
25. Nuyens, D., Cools, R.: Fast algorithms for component-by-component construction of rank-1 lattice rules in shift-invariant reproducing kernel Hilbert spaces. Math. Comput. **75**, 903–920 (2006)
26. Nuyens, D., Cools, R.: Fast component-by-component construction of rank-1 lattice rules with a non-prime number of points. J. Complex. **22**, 4–28 (2006)
27. Olagbemi, O.E.: Scalable algorithms and hybrid parallelization strategies for multivariate integration with ParAdapt and CUDA. Ph.D. thesis, Western Michigan University (2019)
28. Olagbemi, O.E., de Doncker, E.: Scalable algorithms for multivariate integration with ParAdapt and CUDA. In: Proceedings of 2019 International Conference on Computational Science and Computational Intelligence. IEEE Computer Society (2019). https://american-cse.org/csci2019/pdfs/CSCI2019-14dQVW1stBtXVEInMQPd3t/558400a481/558400a481.pdf
29. PEZY Computing/Exascaler Inc. http://www.exascaler.co.jp/
30. Piessens, R., de Doncker, E., Überhuber, C.W., Kahaner, D.K.: QUADPACK. A Subroutine Package for Automatic Integration. Springer Series in Computational Mathematics, vol. 1. Springer, Heidelberg (1983). https://doi.org/10.1007/978-3-642-61786-7
31. Rice, J.R.: A metalgorithm for adaptive quadrature. J. Assoc. Comput. Mach. **22**, 61–82 (1975)

32. Ruijl, B., Herzog, F., Ueda, T., Vermaseren, J.A.M., Vogt, A.: The R^*-operation and five loops calculations. In: 13th International Symposium on Radiative Corrections (2017)
33. Ruijl, B., Herzog, F., Ueda, T., Vermaseren, J.A.M., Vogt, A.: The R^*-operation and combinatorial challenges at five loops. J. Phys.: Conf. Series (JPCS) IOP Series **1085**, 052006 (2018). https://doi.org/10.1088/1742-6596/1085/5/052006
34. Ruijl, B., Ueda, T., Vermaseren, J.A.M.: Forcer, a FORM program for the parametric reduction of four-loop massless propagator diagrams. Technical report, 2017-019, Nikhef (2017). arXiv:1704.06650 [hep-ph]
35. Sanders, J., Kandrot, E.: CUDA by Example: An Introduction to General-Purpose GPU Programming. Pearson, London (2010)
36. Sidi, A.: A new variable transformation for numerical integration. Int. Ser. Numer. Math. **112**, 359–373 (1993)
37. Sloan, I., Joe, S.: Lattice Methods for Multiple Integration. Oxford University Press, Oxford (1994)
38. Sugihara, M.: Optimality of the double exponential formula - functional analysis approach. Numer. Math. **75**(3), 379–395 (1997)
39. Takahasi, H., Mori, M.: Double exponential formulas for numerical integration. Publ. Res. Inst. Math. Sci. **9**(3), 721–741 (1974)
40. Torii, S., Ishikawa, H.: ZettaScaler: liquid immersion cooling manycore based supercomputer. Technical report, ExaScaler Inc., PEZY Computing K. K. (2017)
41. Wynn, P.: On a device for computing the $e_m(s_n)$ transformation. Math. Tables Aids Comput. **10**, 91–96 (1956)