Distributed and multi-core computation of 2-loop integrals

E de Doncker[1] **and F Yuasa**[2]

[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@wmich.edu, fukuko.yuasa@kek.jp

Abstract. For an automatic computation of Feynman loop integrals in the physical region we rely on an extrapolation technique where the integrals of the sequence are obtained with iterated/repeated adaptive methods from the QUADPACK 1D quadrature package. The integration rule evaluations in the outer level, corresponding to independent inner integral approximations, are assigned to threads dynamically via the OpenMP runtime in the parallel implementation. Furthermore, multi-level (nested) parallelism enables an efficient utilization of hyperthreading or larger numbers of cores. For a class of loop integrals in the unphysical region, which do not suffer from singularities in the interior of the integration domain, we find that the distributed adaptive integration methods in the multivariate PARINT package are highly efficient and accurate. We apply these techniques without resorting to integral transformations and report on the capabilities of the algorithms and the parallel performance for a test set including various types of two-loop integrals.

1. Introduction

The integral for an n-dimensional scalar Feynman diagram with L loops and N internal lines can be represented in Feynman parameter space as

$$\mathcal{I} = \frac{\Gamma\left(N - \frac{nL}{2}\right)}{(4\pi)^{nL/2}}(-1)^N \int_0^1 \prod_{j=1}^{N} dx_j \, \delta(1 - \sum x_i) \frac{C^{N-n(L+1)/2}}{(D - i\varepsilon C)^{N-nL/2}}. \tag{1}$$

Here the functions $\mathcal{C}$ and $\mathcal{D}$ are polynomials determined by the topology of the corresponding Feynman diagram [1] and we consider $n = 4$.

Section 2 of this paper outlines a multi-core algorithm for iterated/repeated integration and extrapolation, which allows for parallelization at multiple levels of the integration scheme. A parameter ε is introduced in the integrand denominator and decreased according to a geometric sequence. The corresponding integrals are evaluated by iterated integration, using a one-dimensional adaptive program from the QUADPACK package [2]. The ϵ-algorithm [3, 4] is applied for convergence acceleration of the integral sequence as ε tends to zero.

Section 3 gives a brief description of the distributed adaptive algorithm in the PARINT multivariate integration (cubature) package [5]. Written in C and layered over MPI [6], PARINT consists of parallel adaptive, Monte Carlo and quasi-Monte Carlo methods. Not unlike the QUADPACK routines, these are implemented as tools for *automatic* integration, where the user defines the integrand function and the domain, and specifies a relative and absolute error tolerance for the computation (t_r and t_a, respectively). Denoting the integral by $\mathcal{I}f$ over the domain $\mathcal{D}$,

$$\mathcal{I}f = \int_{\mathcal{D}} f(\vec{x}) d\vec{x}, \tag{2}$$

it is then the objective to return an approximation Qf and absolute error estimate E_af such that $| Qf - \mathcal{I}f | \leq \max\{ t_a, t_r| \mathcal{I}f | \} \leq E_af$, or indicate with an error flag that the requested accuracy cannot be achieved. When a relative accuracy (only) needs to be satisfied we set $t_a = 0$ (and vice-versa).

For iterated integration over a d-dimensional product region we will express (2) as

$$\mathcal{I}f = \int_{\alpha_1}^{\beta_1} dx_1 \int_{\alpha_2}^{\beta_2} dx_2 \ldots \int_{\alpha_d}^{\beta_d} dx_d \, f(x_1, x_2, \ldots, x_d), \tag{3}$$

where the limits of integration are given by functions $\alpha_j = \alpha_j(x_1, x_2, \ldots, x_{j-1})$ and $\beta_j = \beta_j(x_1, x_2, \ldots, x_{j-1})$. In particular, the boundaries of the d-dimensional unit simplex $\mathcal{S}_d$ are $\alpha_j = 0$ and $\beta_j = 1 - \sum_{k=1}^{j-1} x_k$, i.e.,

$$\mathcal{S}_d = \{ (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d \mid \sum_{j=1}^{d} x_j \leq 1 \text{ and } x_j \geq 0 \}. \tag{4}$$

Note that, after removing the δ-function in (1) and eliminating one of the variables in view of $\sum_{j=1}^{N} x_j = 1$, the integration is over a d-dimensional simplex of the form (4) with $d = N - 1$.

The multivariate adaptive approach yields excellent results for a set of two-loop box integrals specified by Laporta [7]. On the other hand, the iterated scheme with extrapolation has been shown to succeed for various classes of integrals in the physical region [8, 9, 10, 11, 12], where multivariate adaptive integration fails.

Results for the Laporta diagrams are reported in Sections 4 and 5 of this paper. The treatment of the sample integrals does not rely on integral transformations as proposed in [12] to improve the integrand behavior. We obtained most of the numerical approximations and timings on the HPCS (High Performance Computational Science) cluster at WMU. For this purpose we used 16-core cluster nodes with Intel Xeon E5-2670, 2.6GHz processors and 128GB of memory, and the cluster's Infiniband interconnect for message passing via MPI.

Some sample sequential results were also collected from runs on a PC node at minamivt005.kek.jp with 6-core Intel Xeon X5680@3.33GHz CPU, and on a 2.6GHz Intel Core i7 Mac-Pro with 4 cores. For the inclusion of OpenMP [13] multi-threading compiler directives in the iterated integration code (based on the Fortran version of QUADPACK) we used the (GNU) *gfortran* compiler and the Intel Fortran compiler, with the flags *-fopenmp* and *-openmp*, respectively. PARINT and its integrand functions were compiled with gcc. Code sections of the sample integrands are supplied in Appendix A.

2. Parallel iterated integration and extrapolation

The numerical integration over the interval $[\alpha_j, \beta_j], 1 \leq j \leq d$ in (3) can be performed with the 1D adaptive integration code DQAGE from the QUADPACK package [2] in each coordinate direction. Via an input parameter to DQAGE we select the 15-point Gauss-Kronrod rule pair for the integral (and error) approximation on each subinterval. If an interval $[a, b]$ arises in the partitioning of $[\alpha_j, \beta_j]$, then the local integral approximation over $[a, b]$ is of the form

$$\int_a^b dx_j \; F(c_1, \ldots, c_{j-1}, x_j) \;\approx\; \sum_{k=1}^{K} w_k F(c_1, \ldots, c_{j-1}, x^{(k)}), \tag{5}$$

where the w_k and $x^{(k)}, 1 \leq k \leq K(= 15)$, are the weights and abscissae of the local rule scaled to the interval $[a, b]$ and applied in the x_j-direction. For $j = 1$ this is the outer integration direction. The function evaluation

$$F(c_1, \ldots, c_{j-1}, x^{(k)}) = \int_{\alpha_{j+1}}^{\beta_{j+1}} dx_{j+1} \ldots \int_{\alpha_d}^{\beta_d} dx_d \, f(c_1, \ldots, c_{j-1}, x^{(k)}, x_{j+1}, \ldots, x_d), \quad 1 \leq k \leq K, \tag{6}$$

is itself an integral in the $x_{j+1}, \ldots, x_d$-directions for $1 \leq j < d$, and is computed by the method(s) for the inner integrations. For $j = d$, (6) is the evaluation of the integrand function

$$F(c_1, \ldots, c_{d-1}, x^{(k)}) = f(c_1, \ldots, c_{d-1}, x^{(k)}).$$

Note that successive coordinate directions may be combined into layers in the iterated integration scheme. Furthermore, the error incurred in any inner integration will contribute to the integration error in all of its subsequent outer integrations [14, 15, 16].

Since the $F()$ evaluations on the right of (5) are independent of one another they can be evaluated in parallel. Important benefits of this approach include that

(i) the granularity of the parallel integration is large, especially when the inner integrals $F()$ are of dimension two or greater;

(ii) the points where the function F is evaluated in parallel are the same as those of the sequential evaluation; i.e., apart from the order of the summation in (5), the parallel calculation is essentially the same as the sequential one. This important property facilitates the debugging of parallel code. As another characteristic the parallelization does not increase the total amount of computational work.

In addition, the memory required for the procedure is determined by (the sum of) the amounts of memory needed for the data pertaining to the subintervals incurred in each coordinate direction (corresponding to the length of the recursion stack for a recursive implementation). Consequently the total memory increases linearly as a function of the dimension d.

For a numerical extrapolation with the ϵ-algorithm [3, 4] we generate a sequence of integral values $\mathcal{I}(\varepsilon_\ell)$, using a geometric progression of ε_ℓ which tends to zero with increasing ℓ (see also [8, 10]). The extrapolation results given here are obtained with a version of the ϵ-algorithm code (DQEXT) from QUADPACK. Note that the accuracy of the extrapolated result is generally limited by the accuracy of the input sequence, which in turn is determined in PARINT by the user-prescribed error tolerance for the integration.

The input parameters of DQAGE include the maximum number (*limit*) of subdivisions allowed in the partitioning of the given interval by the adaptive algorithm. A related output parameter is the error code *iflag*, which returns 1 to indicate that the number of subdivisions reached its maximum, *limit*, upon termination (and is 0 otherwise). In an iterated integration it is sometimes beneficial to keep track of the number of times *limit* is reached throughout the integration on each particular level, as an indication of the difficulty of the integrand in different coordinate directions.

3. PARINT parallel adaptive integration

Layered over MPI [6] for distributed computing, the PARINT adaptive integration executable launches a user-specified number of processes which assume the role of controller and workers. The integration domain is divided initially among the workers. Each on its own part of the domain, the workers engage in an adaptive partitioning strategy similar to that of DQAGE, DCUHRE [17] and HALF [18] by successive bisections, and thus each generate a local priority queue of subregions as a task pool. The priority queue is implemented as a max-heap keyed with the estimated integration errors over the subregions, so that the subregion with the largest estimated error is stored in the root of the heap. However, if the user specifies a maximum size for the heap structure on the worker, the task pool is stored as a *deap* or *double-ended heap* which allows deleting of the maximum as well as the minimum element efficiently (in order to maintain the size of the data structure once it reaches its maximum).

Each task consists of the subdivision (bisection) of the associated subregion (generating two children regions), integration over the children, deletion of the parent region (root of the heap) and insertion of the children back into the heap. The bisection of a region is performed perpendicularly to the coordinate direction in which the integrand is found to vary the most, according to fourth-order differences computed

Table 1. Times for $N = 6$ and $N = 7$ parallel iterated integration (HPCS cluster)

Diagram:		Fig 1(b)		Fig 1(c)	Fig 1(d)	Fig 1(e)
N =	6	6	6	7	7	7
limit =	10	15	15	10	14	14
t_r (1D) =	10^{-8}		10^{-11}	10^{-6}	10^{-8}	10^{-11}
ε_0 =	1.2×10^{-10}		1.2×10^{-20}	1.2×10^{-15}	1.2×10^{-14}	1.2×10^{-13}
$\lvert Error \rvert$ =	5.0×10^{-6}	5.0×10^{-6}	1.8×10^{-10}	1.2×10^{-5}	2.6×10^{-9}	4.6×10^{-9}
# THREADS	TIME (S)					
1	1155.6	1167.5	37381.0	4151.0	15316.8	
2	716.2	627.2	36808.4	3620.0	14706.1	
4	360.5	360.4	18248.5	1994.0	7443.2	
6	255.4	254.8	12068.2	1011.3	4793.6	
8	226.5	221.4	9045.1	800.7	3847.1	
10	206.8	213.7	7409.4	649.2	3482.7	
12	201.4	207.2	6978.2	587.2	3287.3	
15	193.9	198.8	5634.6	549.9	3191.1	
$(15, 15)$	109.0	115.4	3237.7	282.5	1216.8	5837.9

in each direction [17, 18]. The integration rule on a subregion is determined by the type of region and the user-selected polynomial degree. The available cubature rules in PARINT include a set of rules for the d-dimensional cube [19, 20, 21], the Quadpack rules for $d = 1$, and a set of rules for the d-dimensional simplex [22, 23]. The latter are the simplex rules by Grundmann and Möller [23] (which were also found by de Doncker [24] via extrapolation).

An important mechanism of the distributed integration algorithm is the load balancing strategy, which is receiver-initiated and geared to keeping the loads on the worker task pools balanced, particularly in cases of irregular integrand behavior. The message passing is performed in a non-blocking and asynchronous manner, and permits overlapping of computation and communication. The user has the option of turning load balancing on or off, as well as allowing or dis-allowing the controller to also act as a worker. Optionally the installation can be configured to use long doubles instead of doubles.

The multivariate partitioning (sequentially or in parallel) exhibits a dimensional effect (exponential as a function of d) with respect to the total number of regions generated and thus the required total memory. To some extent this is helped by the distribution of the subregions over the participating processors' local memories in the parallel system.

As a result of the asynchronous processing and message passing on MPI, PARINT executes on a hybrid platform (multi-core and distributed) by assigning multiple processes to each node. However, it is generally not possible to repeat an execution exactly. The parallelization also leads to *breaking loss* or extra work performed at the end of the computation, due to the time elapsed while workers continue to generate subregions after the termination message has been issued but before receiving it. This may increase the total amount of work (compared to the sequential work), particularly when the number of processes is large.

4. Results of multi-threaded iterated integration

We consider five two-loop box diagrams from Laporta [7], given in Fig 1. Table 1 gives an overview of the problem specifications, the execution times for a varying number of threads, and the absolute accuracy achieved (|Error|) with respect to the values C_0 pp. 46-47 in [7] for the diagrams in Fig 1(b-d). For Fig 1(e) we compare with the result from PARINT executed in long double precision (cf., Section 5 below). Detailed results for the $N = 5$ diagram of Fig 1(a) are listed in Table 2.

The multi-threading is invoked in the outer (x_1) integration in (3), except for the entry denoted $(15, 15)$ which represents nested parallelization applied in the outer two (x_1 and x_2) integrations. It emerges that the latter outperforms the single level parallelization, showing a more efficient use of the available cores. The integrations are performed over the simplex $\mathcal{S}_d$ for $d = N - 1$ (see (4)). Appendix A.1-5 provides code segments for the integrands corresponding to Fig 1(a-e). The integrand evaluation in

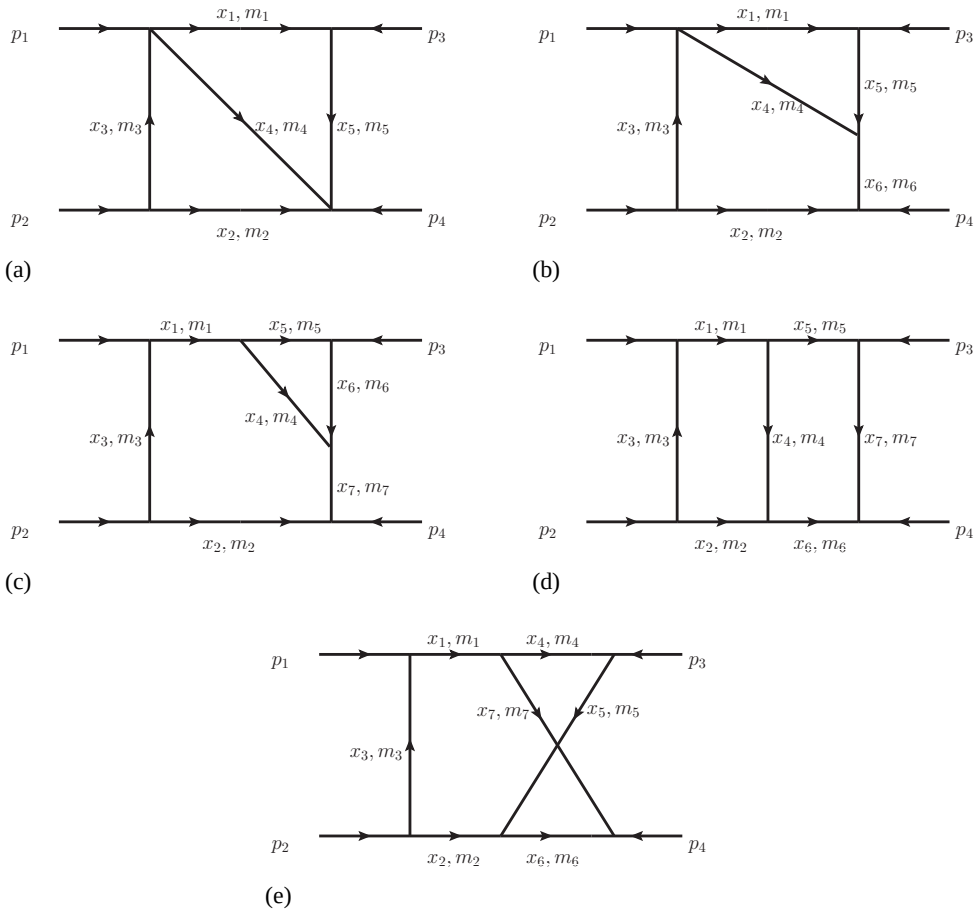

Figure 1. Diagrams (a) $N = 5$, (b) $N = 6$, (c) $N = 7$, (d) $N = 7$ ladder, (e) $N = 7$ crossed

the code corresponds to an expression of the real part integrand where the factor εC in the denominator of (1) is replaced by ε for the extrapolation as $\varepsilon \to 0$. Note also that the factor $1/(4\pi)^{nL/2}$ is omitted.

The problem specifications in Table 1 include: the number of internal lines N, the relative error tolerance t_r requested for the 1D integrations of the iterated scheme, the maximum value *limit* for the number of subdivisions of the adaptive integration procedure in each coordinate direction, and the starting value of $\varepsilon = \varepsilon_0$ for the extrapolation sequence. For the sequence we use $\varepsilon_\ell = \varepsilon_0 \times 1.2^{-\ell}$, $0 \le \ell < \ell_{\max}$, where the length of the sequence ($\ell_{\max}$) is around 15. In view of the trade-off between the amount of work done and the accuracy achieved, we choose smaller ε_0 and larger *limit* for higher accuracy work.

4.1. Diagram with five internal legs

For a sequential run on minamivt005, with target relative accuracy of 10^{-6} for the 1D integrations, and the maximum number of subdivisions *limit* = 10, 10, 20, 20 in the x_1, x_2, x_3 and x_4 coordinate directions, respectively, extrapolation with the ϵ-algorithm on a sequence of 15 integrals for $\varepsilon_\ell = 1.2^{-15-\ell}$, $0 \le \ell < 15$ converges with relative error 4×10^{-6} in a total elapsed time of 30.3 seconds.

A more accurate computation leads to the convergence results in Table 2, where in successive columns the following are given: ℓ, ε_ℓ, the integration time, the integral approximation corresponding to ε_ℓ and the extrapolated value for C_0. The integration time listed is that of a sequential run on a node of the HPCS cluster. The target 1D relative accuracy for the integrations was 10^{-10}, $\varepsilon_0 = 1.2^{-20}$ and the maximum number of subdivisions *limit* = 30, 30, 50, 50 in the successive coordinate directions.

Table 2. Extrapolation, $N = 5,\ \varepsilon_0 = 1.2^{-20}$, rel. tol. 10^{-10}, seq. times (HPCS cluster)

ℓ	ε_ℓ	TIME (s)	$\mathcal{I}(\varepsilon_\ell)$ Approx.	C_0 Approx.
0	0.260841e-01	15.0	0.879998554543172662	
1	0.217367e-01	15.2	0.892831432362168109	
2	0.181139e-01	16.7	0.903287762406520423	0.9492935019767
3	0.150949e-01	23.0	0.911812341919760572	0.9494302717234
4	0.125791e-01	24.1	0.918769150341187113	0.9490308675735
⋮	⋮	⋮	⋮	⋮
12	0.292550e-02	65.1	0.943930996955826496	0.9509233190442
13	0.243792e-02	82.8	0.945122576092676248	0.9509234745063
14	0.203160e-02	99.5	0.946108275775448759	0.9509235737400
15	0.169300e-02	105.4	0.946924415927191898	0.9509235597296
16	0.141083e-02	108.9	0.947600717832237871	0.9509235626613
	TOTAL TIME:	843.4	LAPORTA VALUE:	0.9509235623171

Table 3. Extrap. results $N = 6,\ \varepsilon_0 = 1.2^{-20}$, rel. tol. 10^{-11}, *limit* = 15, # threads (reg. nodes) = 15

ℓ	ε_ℓ	TIME (s)	$\mathcal{I}(\varepsilon_\ell)$ Approx.	C_0 Approx.	$\mid (C_0(\ell) - C_0(\ell-1)) \mid$
0	0.260841e-01	65.4	0.221532578551221299		
1	0.217367e-01	99.9	0.230888677472657844		
2	0.181139e-01	109.1	0.238665585251643331	0.276963858955	
3	0.150949e-01	161.6	0.245113995206973190	0.276414002035	5.50e-04
4	0.125791e-01	185.5	0.250453078514603744	0.275897645038	5.16e-04
5	0.104826e-01	209.5	0.254870342845598397	0.275953613377	5.60e-05
⋮	⋮	⋮	⋮	⋮	⋮
10	0.421272e-02	687.1	0.267830329163125991	0.276209336599	1.08e-06
11	0.351060e-02	841.0	0.269250457421247535	0.276209925289	5.89e-07
12	0.292550e-02	721.5	0.270427723407943632	0.276209228098	6.97e-07
13	0.243792e-02	904.6	0.271404161999583338	0.276209225178	2.92e-09
TOTAL TIME (15t):		5634.6	LAPORTA VALUE:	0.276209225359	

The extrapolation converges within eight to nine digits which agree with Laporta's result. In view of the small computation times of the individual integrals, parallelization is not highly beneficial. Some speedup is observed for up to four threads with a parallel time of 445s, compared to 536s for two threads, and 843s for one thread (i.e., the sequential time of Table 2. Parallel speedup is defined as the ratio of sequential to parallel time.

4.2. Diagram with six internal legs

In test runs for a relative tolerance of 10^{-8}, and allowed maximum number of subdivisions *limit* = 10 in each coordinate direction, extrapolation on the sequence with $\varepsilon_\ell = 1.2^{-10-\ell},\ 0 \le \ell < 15$ converges to Laporta's result within 4 to 5 digits, in about 1,156 seconds on a node of the HPCS cluster for the sequential computation (764 seconds on Mac Pro). Timings for *limit* = 10 and 15, and for a varying number of threads are given in columns 2 and 3 of Table 1.

We achieved a higher accuracy (differing 1.8×10^{-10} from Laporta's result, i.e., with a relative accuracy of 6.5×10^{-10}), as specified in column 4 of Table 1. Note how the time decreases overall from 37,381 seconds for the sequential run to 3,238 seconds with the parallel version, yielding a speedup of about 12.6. The convergence results in Table 3 and execution times per iteration, using 15 threads, were obtained on the HPCS cluster. Corresponding to $C_0 = C_0(\ell)$, the last column represents $|C_0(\ell) - C_0(\ell-1)|$ which serves for the purpose of gauging the convergence. Since it is based on only two consecutive elements it may not be a good estimate of the error. The ϵ-algorithm routine DQEXT in QUADPACK produces a more sophisticated (and more conservative) error estimate.

Table 4. Extrap. results $N = 7$ crossed, $\varepsilon_0 = 1.2^{-13}$, rel. tol. 10^{-11}, *limit* = 14, # threads $(15, 15)$

ℓ	ε_ℓ	TIME (s)	CUMULATIVE TIME (s)	$\mathcal{I}(\varepsilon_\ell)$ Approx.	C_0 Approx.	$\mid (C_0(\ell) - C_0(\ell-1)) \mid$
0	0.934639e-01	56.4	56.4	0.16248135119206037 8e-01		
1	0.778866e-01	100.7	157.1	0.28473350663804193 6e-01		
2	0.649055e-01	109.2	266.3	0.39826039894024202 5e-01	0.1875391249e+00	
3	0.540879e-01	118.6	384.9	0.49725743476387022 9e-01	0.1171759149e+00	7.04e-02
4	0.450732e-01	147.0	531.9	0.57967796350048589 3e-01	0.9894834504e-01	1.82e-02
5	0.375610e-01	159.1	691.0	0.64598048134520469 3e-01	0.8382861180e-01	1.51e-02
6	0.313009e-01	268.8	959.8	0.69796585401931621 2e-01	0.8555394058e-01	1.73e-03
7	0.260841e-01	195.1	1154.8	0.73794597386928118 0e-01	0.8541352574e-01	1.40e-04
8	0.217367e-01	333.7	1488.5	0.76824543245676293 9e-01	0.8535482088e-01	5.87e-05
⋮	⋮	⋮	⋮	⋮	⋮	
14	0.727958e-02	831.2	5032.2	0.84091249669057344 1e-01	0.8535140207e-01	3.10e-08
15	0.606632e-02	805.6	5837.9	0.84443377896948043 6e-01	0.8535139544e-01	6.63e-09

4.3. Diagrams with seven internal legs

The integrands for the ladder diagram of Fig 1(d) and the crossed diagram of Fig 1(e) correspond to those for the planar and the non-planar diagram, respectively, before the transformations applied in [12].

4.3.1. Diagram of Fig 1(c). For the diagram Fig 1(c), a sequential run on minamivt005 generated a result with relative error of 1.2×10^{-5} in 14,719 seconds, using relative integration error tolerance 10^{-6}, maximum number of subdivisions *limit* = 10 in all coordinate directions, and extrapolation on an input sequence of 15 integrals starting at $\varepsilon_0 = 1.2^{-15}$.

By examining the *iflag* parameters, the output from the execution reveals that the maximum number of subdivisions is reached particularly toward the inner integrations. For this problem, reversing the order of the integration variables as shown in the function definition in Appendix A.3, improves the performance dramatically. Keeping all other inputs the same, the sequential execution time decreases to 4,151 seconds while the accuracy of the result is maintained. Parallel times obtained when varying the number of threads (on HPCS cluster nodes) are given in Table 1 (column of Fig 1(c)). A speedup of about 14.7 is achieved with nested parallelism.

4.3.2. Two-loop ladder box diagram Fig 1(d). For the diagram of Fig 1(d) a preliminary minamivt005 run with *limit* = 5 in all directions, relative integration tolerance 10^{-6} and 15 integrations starting at $\varepsilon = 1.2^{-10}$, gave the result 0.1036438 which, compared to Laporta's listed value of 0.10364072, has an absolute accuracy of about 3×10^{-6} (relative accuracy 3×10^{-5}), after an execution time of 7,848 seconds. The *iflag* parameters indicated difficult behavior in the inner integrations for this problem. By changing the order of the integration variables as listed in the code segment of Appendix A.4 for the integrand function, while keeping all other inputs unchanged, the program runs in 1,072 seconds on Mac Pro and delivers the extrapolation result 0.1036451 (with relative error 4.2×10^{-5}).

We produced a more accurate result and timed the executions with *limit* = 14 in all directions, relative 1D integration tolerance 10^{-8}, for 17 integrations starting at $\varepsilon_0 = 1.2^{-14}$. For the timing results see Table 1 (column of Fig 1(d)). The extrapolated integral, 0.1036407236, has a relative error of 2.5×10^{-8}. Note the overall gain in performance by the parallelization, from 15,317 seconds (sequential) to 1,217 seconds (parallel), or a speedup of about 12.6.

4.3.3. Two-loop crossed box diagram Fig 1(e). A sequential run on minamivt005 with *limit* = 5 in all directions, relative integration tolerance 10^{-6}, $\varepsilon_0 = 1.2^{-10}$ generated the result 0.85354533e-01 at $\ell = 15$, in 2,901 seconds. The integration variables were set as: $x1 = x(1)$, $x2 = 1 - x(1) - x(2) - x(3) - x(4) - x(5) - x(6)$, $x3 = x(2)$, $x4 = x(3)$, $x5 = x(4)$, $x6 = x(5)$, $x7 = x(6)$. With the order

of the integration variables changed to that of the code segment in Appendix A.5 for the integrand, the program returned the result 0.85355139e-01 at $\ell = 14$, and 0.85349174e-01 at $\ell = 15$, in a total time of 801 seconds on Mac Pro.

Higher accuracy runs were performed with relative integration tolerance 10^{-11}, *limit* = 14 and $\varepsilon_0 = 1.2^{-13}$, resulting in 0.85351402e-01 at $\ell = 14$ in 5,032 seconds, and 0.85351395e-01 at $\ell = 15$ in 5,838 seconds (total). Table 4 gives an overview of the convergence and the times for the individual iterations using nested multi-threading $(15, 15)$. As another comparison, since the problem is actually unphysical for the specified kinematics, it is possible to set $\varepsilon = 0$ in the integrand. With 1D relative error tolerance 10^{-8} and *limit* = 15, this gives 0.853513981536e-01 (with absolute error estimate of the outer integration 2.2e-09) in 4,383 seconds, using $(15, 15)$ nested multi-threading. Laporta's method did not solve this problem at the time of the writing of his article.

5. PARINT results

PARINT contains a parallel adaptive method with load balancing for d-dimensional rectangular regions roughly based on the subdivision strategy of [19, 20, 21] and for simplex regions [22, 25], with results provided by d-dimensional integration rules of degrees 7 and 9 for the cube, and 3, 5, 7 and 9 for the simplex. For the problems at hand, some testing reveals that it is more efficient for PARINT to transform the integral from the (unit) simplex to the (unit) cube, followed by an integration over the cube. We use the transformation $(x(1), \ldots, x(d)) \rightarrow (x_1, (1-x_1)x(2), \ldots, (1-x_1-\ldots-x_{d-1})x(d))$ with Jacobian $(1-x_1)(1-x_1-x_2)\ldots(1-x_1-\ldots-x_{d-1})$.

Since the integrals with kinematics specified by Laporta [7] are unphysical we set $\varepsilon = 0$, or *eps* = 0 in the integrand codes of Appendix A.1-5. However, care needs to be taken to handle boundary singularities in case the integration rule employs function evaluations on the boundary. For example, if $x_1 = x_4 = x_5 = 0$ (and *eps* = 0) in the definition of *fc* in A.1, then *cc* = *dd* = 0. We set the integrand to 0 to avoid a divide-by-0 floating point exception.

Fig 2 presents timing results obtained with PARINT on the HPCS cluster, corresponding to the five diagrams in Fig 1. Up to 64 MPI processes were spawned on up to 4 cluster nodes, using 16 procs (slots) per node. When referring to numbers of integrand evaluations, *million* and *billion* are abbreviated by "M" and "B", respectively. Since the times vary when the same run is repeated, the plots show timings usually averaged over three or four runs for the same p. The times T_p are expressed in seconds, as a function of the number of MPI processes $p, 1 \leq p \leq 64$.

Table 5 gives a brief overview of pertinent test specifications, T_1, T_p and the speedup $S_p = T_1/T_p$ for $p = 64$. For example, the times of the $N = 5$ diagram decrease from 32.6s at $p = 1$ to 0.74s at $p = 64$ for $t_r = 10^{-10}$ (reaching a speedup of 44); whereas the $N = 7$ crossed diagram times range between 27.6s and 0.49s for $t_r = 10^{-7}$ (with speedup exceeding 56).

For the two-loop crossed box problem we also ran PARINT in *long double* precision. The results for an increasing allowed maximum number of evaluations and increasingly strict (relative) error tolerance t_r (using 64 processes) are given in Table 6, as well as the corresponding double precision results. Listed are: the integral approximation, actual relative error estimate E_r, number of function evaluations reached

Table 5. Test specifications and range of times in Fig 2(a-d)

DIAGRAM	N	REL TOL t_r	MAX EVALS	$T_1[s]$	$T_{64}[s]$	SPEEDUP S_{64}
Fig 1(a) / 2(a)	5	10^{-10}	400M	32.6	0.74	44.1
Fig 1(e) / 2(a) crossed	7	10^{-7}	300M	27.6	0.49	56.3
Fig 1(b) / 2(b)	6	10^{-9}	3B	213.6	5.06	42.2
Fig 1(d) / 2(b) ladder	7	10^{-8}	2B	189.9	4.33	43.9
Fig 1(c) / 2(c)	7	10^{-8}	5B	507.9	8.83	57.5
Fig 1(e) / 2(d) crossed	7	10^{-9}	20B	1892.5	34.6	54.7

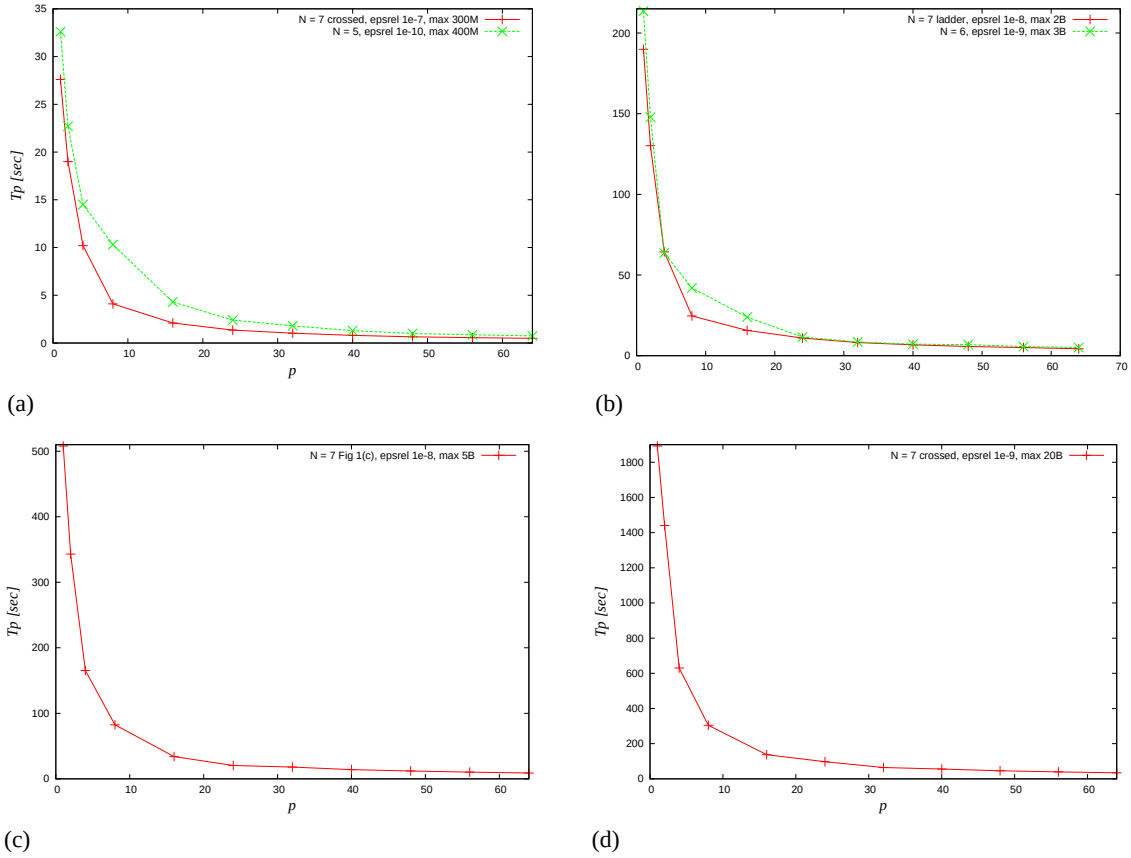

Figure 2. (a) $N = 5, t_r = 10^{-10}$ and $N = 7$ crossed, $t_r = 10^{-7}$; (b) $N = 6, t_r = 10^{-8}$ and $N = 7$ ladder, $t_r = 10^{-8}$; (c) $N = 7$ [Fig 1(c)], $t_r = 10^{-8}$; (d) $N = 7$ crossed, $t_r = 10^{-9}$

Table 6. PARINT double and long double results for crossed diagram Fig 1(e)

t_r	Max	*long double* precision					*double* precision			
	Evals	INTEGRAL APPROX	E_r	*iflag*	#EVALS	TIME	E_r	*iflag*	#EVALS	TIME
10^{-08}	600M	0.085351397048123	3.5e-07	1	600000793	1.7	3.4E-07	1	600001115	0.95
	1B	0.085351397753978	1.7e-07	1	1000000141	2.9	1.6e-07	1	1000000141	1.6
	2B	0.085351398064779	2.9e-08	1	2000000443	6.0	2.9e-08	1	2000000765	3.3
	6B	0.085351398130559	5.6e-09	0	4164032999	14.3	8.0e-09	0	4424455329	8.6
10^{-09}	10B	0.085351398143465	5.5e-09	1	10000001571	29.7	5.5e-09	1	10000000927	16.4
	50B	0.085351398152623	9.9e-10	0	35701321579	124.4	4.5e-10	0	9799638359	16.0
10^{-10}	80B	0.085351398153315	5.5e-10	1	80000001137	240.2	5.5e-10	1	80000000171	133.5
	100B	0.085351398153507	4.1e-10	1	100000000093	302.0	4.1e-10	1	100000000093	168.3
	300B	0.085351398153798	9.1e-11	0	238854968513	642.3	1.3e-10	1	300000000279	587.2

and time taken in *long double* precision, followed by the relative error estimate, number of function evaluations reached and running time in *double* precision. Allowing for 16 processes per node and up to 64 processes total, the MPI host file has four lines of the form *nx slots=16* where *nx* represents a selected node name. The running time is reported (in seconds) from the PARINT executable, and comprises all computation not inclusive the process spawning and PARINT initialization. For a comparable number of function evaluations, the time using long doubles is slightly less than twice the time taken using doubles. The *iflag* parameter returns 0 when the requested accuracy is assumed to be achieved, and 1 otherwise. Reaching the maximum number of evaluations results in abnormal termination with *iflag* = 1.

The integral approximation for the *double* computation is not listed in Table 6; it is consistent with the *long double* result within the estimated error (which appears to be over-estimated). Using doubles the program terminates abnormally for the requested relative accuracy of $t_r = 10^{-10}$. Normal termination is achieved in this case around 239B evaluations with long doubles.

6. Concluding remarks

In this paper we explored *parallel* integration methods for the *automatic* computation of a class of two-loop box integrals [7]: (i) multi-threaded iterated/repeated integration with extrapolation, and (ii) distributed adaptive multivariate integration. With regard to (i) we demonstrated the capabilities of the extrapolation for convergence acceleration, and of the parallelization on multiple levels of the iterated integration scheme to speed up the execution.

For physical kinematics we rely on calculating a sequence of integrals for extrapolation as the parameter ε decreases [8, 9, 10, 11, 12], and it has generally not been possible to calculate the integrals for small enough ε using adaptive partitioning of the multivariate domain. However, the test cases specified in [7] allow setting $\varepsilon = 0$ in the integrand, and it emerges that the distributed adaptive approach of PARINT solves these problems very efficiently (although iterated integration has an advantage with respect to low memory usage). PARINT executes on multiple nodes, and multiple cores per node. Future work in this area will address, for example, the distributed (non-adaptive) quasi-Monte Carlo and Monte Carlo methods in the current PARINT package, parallel random number generators and utilizing GPUs and other accelerators for the computations.

Future work on the iterated strategy will include improved error handling across the levels of the iterated scheme. Furthermore, the parallelization of the iterated strategy can be extended to handle infrared divergence as presented in [27]. Higher-dimensional problems can be attempted with iterated integration by combining some successive 1D into multivariate layers as was done in [8]. Evaluating the points in the 2D (or 3D) outer layer on GPUs may yield an alternative in some cases, if the structure of the computation is sufficiently similar for all evaluation points [26]. In view of the computationally intensive nature of the underlying problems, roundoff error guards and suitable summation techniques need to be considered. Finally, the multi-threaded iterated integration can be incorporated within the distributed setting of PARINT on MPI [6].

Acknowledgments

We acknowledge the support from the National Science Foundation under Award Number 1126438. This work is further supported by Grant-in-Aid for Scientific Research (24540292) of JSPS, and the Large Scale Simulation Program No.12-07 of KEK. We also thank Dr. L. Cucos and O. Olagbemi for help with PARINT.

Appendix A. Code of two-loop box integrands

Code segments for the integrands corresponding to the two-loop box diagrams of Fig 1(a-e) are given below. Note that '&' indicates continuation, the *sqeps* variable represents ε^2 and *sqrt3* represents $\sqrt{3}$.

Appendix A.1. Diagram Fig 1(a) (N = 5)

```
      x1 = x(1)
      x2 = 1-x1-x(2)-x(3)-x(4)
      x3 = x(2)
      x4 = x(3)
      x5 = x(4)
      cc = (x1+x5)*(x2+x3+x4)+x4*(x2+x3)
      dd =
     & -x1**2*x2-x1**2*x3-x1**2*x4-x1*x2**2-x1*x2*x3-2*x1*x2*x4-x1*x2*x5
     & -x1*x3**2-2*x1*x3*x4-x1*x3*x5-x1*x4**2-x1*x4*x5-x2**2*x4-x2**2*x5
```

```
& -x2*x3*x4-x2*x3*x5-x2*x4**2-2*x2*x4*x5-x2*x5**2-x3**2*x4-x3**2*x5
& -x3*x4**2-2*x3*x4*x5-x3*x5**2-x4**2*x5-x4*x5**2

 fc = -dd/cc/(dd**2+sqeps)
```

Appendix A.2. Diagram Fig 1(b) (N = 6)

```
 x6 = x(1)
 x5 = x(2)
 x4 = x(3)
 x3 = x(4)
 x2 = x(5)
 x1 = 1-x6-x2-x3-x4-x5

 dd = (x1**2+x5**2+x1*x5)*(x2+x3+x4+x6)
&     +(x2**2+x3**2+x6**2+x2*x3+x2*x6+x3*x6)*(x1+x4+x5)
&     +x4**2*(x1+x2+x3+x5+x6)+3*x4*x5*x6+
&     2*x1*x4*(x2+x3+x6)+2*x4*x5*(x2+x3)

 fd = (dd-eps)*(dd+eps)/(dd**2+sqeps)**2
```

Appendix A.3. Diagram of Fig 2(a) (N = 7)

```
 x7 = x(1)
 x6 = x(2)
 x5 = x(3)
 x4 = x(4)
 x3 = x(5)
 x2 = x(6)
 x1 = 1-x7-x2-x3-x4-x5-x6
 dd =
& -(x1**2+x2**2+x3**2+x7**2+x1*x2+x1*x3+x1*x7+x2*x3+x2*x7+x3*x7)
& *(x4+x5+x6)-x4**2*(1-x4)-(x5**2+x6**2+x5*x6)*(1-x5-x6)
& -3*x4*(x1*x5+x6*x7)-2*((x1+x2+x3)*x4*x6+(x2+x3+x7)*x4*x5)

 cc = x1*x4+x1*x5+x1*x6+x2*x4+x2*x5+x2*x6+x3*x4+x3*x5+x3*x6+x4*x5
&   +x4*x6+x4*x7+x5*x7+x6*x7

 fg = -2*cc*dd*(dd-sqrt3*eps)*(dd+sqrt3*eps)/(dd**2+eps**2)**3
```

Appendix A.4. Ladder box diagram Fig 1(d) (N = 7)

```
 x7 = x(1)
 x6 = x(2)
 x5 = x(3)
 x4 = x(4)
 x3 = x(5)
 x2 = x(6)
 x1 = 1-x2-x3-x4-x5-x6-x7

 x1234 = x1+x2+x3+x4
 x45 = x4+x5
 x67 = x6+x7
 x4567 = x45 + x67

 cc = x1234*x4567 - x4**2
 dd = cc*(x1+x2+x3+x4+x5+x6+x7) - (x1*x2*x4567
&    + x5*x6*x1234 + x1*x4*x6 + x2*x4*x5 + x3*x4*x7
&    + x3*(x1*x4 + x1*x5 + x1*x6 + x1*x7 + x4*x5)
&    + x3*(x2*x4 + x2*x5 + x2*x6 + x2*x7 + x4*x6)
&    + x7*(x1*x4 + x1*x5 + x2*x5 + x3*x5 + x4*x5)
&    + x7*(x1*x6 + x2*x4 + x2*x6 + x3*x6 + x4*x6))

 fh = 2*cc*dd*(dd-sqrt3*eps)*(dd+sqrt3*eps)/(dd**2+eps**2)**3
```

Appendix A.5. Crossed box diagram Fig 1(e) (N = 7)

```
      x7 = x(1)
      x6 = x(2)
      x5 = x(3)
      x4 = x(4)
      x3 = x(5)
      x2 = x(6)
      x1 = 1-x2-x3-x4-x5-x6-x7

      cc = (x1+x2+x3+x4+x5)*(x1+x2+x3+x6+x7) - (x1+x2+x3)**2

      dd = cc - (x1*x2*x4 + x1*x2*x5 + x1*x2*x6 + x1*x2*x7
     &    + x1*x5*x6 + x2*x4*x7 - x3*x4*x6
     &    + x3*(-x4*x6 + x5*x7)
     &    + x3*(x1*x4 + x1*x5 + x1*x6 + x1*x7 + x4*x6 + x4*x7)
     &    + x3*(x2*x4 + x2*x5 + x2*x6 + x2*x7 + x4*x6 + x5*x6)
     &    + x1*x4*x5 + x1*x5*x7 + x2*x4*x5 + x2*x4*x6
     &    + x3*x4*x5 + x3*x4*x6 + x4*x5*x6 + x4*x5*x7
     &    + x1*x4*x6 + x1*x6*x7 + x2*x5*x7 + x2*x6*x7
     &    + x3*x4*x6 + x3*x6*x7 + x4*x6*x7 + x5*x6*x7)

      fi = 2*cc*dd*(dd-sqrt3*eps)*(dd+sqrt3*eps)/(dd**2+eps**2)**3
```

References

[1] Nakanishi N 1971 *Graph Theory and Feynman Integrals* (Gordon and Breach, New York)
[2] Piessens R, de Doncker E, Überhuber C W and Kahaner D K 1983 *QUADPACK, A Subroutine Package for Automatic Integration* (*Springer Series in Computational Mathematics* vol 1) (Springer-Verlag)
[3] Shanks D 1955 *J. Math. and Phys.* **34** 1–42
[4] Wynn P 1956 *Mathematical Tables and Aids to Computing* **10** 91–96
[5] de Doncker E, Kaugars K, Cucos L and Zanny R 2001 *Proc. of Computational Particle Physics Symposium (CPP 2001)* pp 110–119
[6] Open-MPI http://www.open-mpi.org
[7] Laporta S 2000 *Int. J. Mod. Phys. A* **15** 5087–5159 arXiv:hep-ph/0102033v1
[8] de Doncker E, Shimizu Y, Fujimoto J and Yuasa F 2004 *Computer Physics Communications* **159** 145–156
[9] de Doncker E, Fujimoto J, Kurihara Y, Hamaguchi N, Ishikawa T, Shimizu Y and Yuasa F 2010 *XIV Adv. Comp. and Anal. Tech. in Phys. Res.* PoS (ACAT10) 073
[10] de Doncker E, Fujimoto J, Hamaguchi N, Ishikawa T, Kurihara Y, Shimizu Y and Yuasa F 2011 *Journal of Computational Science (JoCS)* **3** 102–112
[11] de Doncker E, Yuasa F and Assaf R 2013 *Journal of Physics: Conf. Ser.* **454**
[12] Yuasa F, de Doncker E, Hamaguchi N, Ishikawa T, Kato K, Kurihara Y and Shimizu Y 2012 *Journal Computer Physics Communications* **183** 2136–2144
[13] OpenMP website, http://www.openmp.org
[14] Fritsch F N, Kahaner D K and Lyness J N 1981 *ACM TOMS* **7** 46–75
[15] Kahaner D, Moler C and Nash S 1988 *Numerical Methods and Software* (Prentice Hall)
[16] de Doncker E and Kaugars K 2010 *Procedia Computer Science* **1** 117–124
[17] Berntsen J, Espelid T O and Genz A 1991 *ACM Trans. Math. Softw.* **17** 452–456
[18] De Ridder L and Van Dooren P 1976 *Journal of Computational and Applied Mathematics* **2** 207–210
[19] Genz A and Malik A 1980 *Journal of Computational and Applied Mathematics* **6** 295–302
[20] Genz A and Malik A 1983 *SIAM J. Numer. Anal.* **20** 580–588
[21] Berntsen J, Espelid T O and Genz A 1991 *ACM Trans. Math. Softw.* **17** 437–451
[22] Genz A 1990 *Lecture Notes in Computer Science* vol 507 ed Sherwani N A, de Doncker E and Kapenga J A pp 279–285
[23] Grundmann A and Möller H 1978 *SIAM J. Numer. Anal.* **15** 282–290
[24] de Doncker E 1979 *Math. Comp.* **33** 1003–1018
[25] Genz A and Cools R 2003 *ACM Trans. Math. Soft.* **29** 297–308
[26] Yuasa F, Ishikawa T, Hamaguchi N, Koike T and Nakasato N 2013 *Journal of Physics: Conf. Ser.* **454**
[27] de Doncker E, Yuasa F and Kurihara Y 2012 *Journal of Physics: Conf. Ser.* **368**