Parallel computation of Feynman loop integrals

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

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

E-mail: elise.dedoncker@@wmich.edu

E-mail: fukuko.yuasa@@kek.jp

Abstract. The need for large numbers of compute-intensive integrals, arising in quantum field theory perturbation calculations, justifies the parallelization of loop integrals. In earlier work, we devised effective multivariate methods by iterated (repeated) adaptive numerical integration and extrapolation, applicable for some problem classes where standard multivariate integration techniques fail through integrand singularities. In repeated integration, the function evaluations for the outer integral are independent integral computations for the next lower level and can be distributed to threads in a multi-core parallelization. The distribution level determines the number of dimensions in the inner integral below it and thus the granularity of the computation. We discuss a multi-threaded implementation over the OpenMP Application Program Interface.

1. Introduction

We present a multi-threaded approach for the computation of iterated integrals. An important application of (numerical) iterated integration has been the computation of Feynman loop integrals, which are needed in higher order perturbation calculations of the cross section for particle interactions. Thousands of integrals may be needed for one interaction.

Furthermore, Feynman loop integrals are generally affected by non-integrable singularities, through vanishing denominators in the interior and/or at the boundaries of the integration domain. In order to handle singularities inside the domain, a value for the integral is calculated by introducing a parameter $i\delta$ in the integrand denominator, effectively moving the singularity into the complex plane, and by taking the limit of the integral value as $\delta \to 0$. We make use of numerical iterated integration for computing an integral sequence as δ decreases. In view of the computational expense we propose a technique utilizing multi-threading for a parallel computation of the individual integrals.

The goal of an *automatic* integration procedure is to obtain an approximation $Q(f)$ to an integral

$$If = \int_{\mathcal{D}} f(\vec{x})\, d\vec{x} \tag{1}$$

and an error estimate $\mathcal{E}f$, in order to satisfy a specified accuracy requirement for the error $Ef = |Qf - If|$, so that

$$|\,Qf - If\,| \;\le\; \mathcal{E}f \;\le\; \max\{\, t_a\,,\, t_r\,|\,If\,|\,\} \tag{2}$$

for a given integrand function f, region $\mathcal{D}$ and (absolute/relative) error tolerances t_a and t_r, respectively. This can be interpreted as a *black box* method for the computation of the integral (1), where the inputs specify the integral dimension, the region $\mathcal{D}$, the function $f(\vec{x})$ and the requested accuracy. The output contains the result Qf and error estimate $\mathcal{E}f$. No specification of the integrand behavior is needed by the general automatic integration procedure.

Section 2 below gives an outline of global adaptive and iterated integration techniques in the framework of automatic integration. Our parallelization for a multi-threaded environment is explained in Section 3, and some background on Feynman loop integrals is provided in Section 4. Timing results using the shared memory programming OpenMP [1] Application Program Interface (API) are given in Section 5.

2. Automatic numerical integration

2.1. Global adaptive procedure

A versatile type of algorithm to implement the black box approach is by adaptive partitioning of the integration region. At each step, a region is subdivided, integral and error estimates are computed over the subregions, and the overall result and error estimate are updated. Various strategies are possible for the selection of the region to be subdivided at each step. A *global adaptive* strategy maintains a priority queue on the subregion collection (e.g., a linked list or a heap), keyed with the local error estimates of the subregions. Examples of global adaptive integrators are the 1D adaptive programs of Quadpack [2], and the multivariate Dcuhre [3] for integration over an (N-dimensional) cube, which select the region with the highest (absolute) error estimate for subdivision at each step. As a result, sample points are concentrated in the vicinity of irregular integrand behavior such as singularities, discontinuities, peaks or ridges and troughs. The Cubpack [4] package targets adaptive integration over a collection of cubes and simplices.

The local integral (over a subregion or an interval) is approximated by a (cubature or quadrature) rule which is a linear combination of function values of the form $\sum_{k=1}^{K} w_k f(\vec{x_k})$. By evaluating more than one rule over the subregion, a local error estimate can be obtained as a function of the difference between local integral approximations. For example, the (1D) program Dqag of the Quadpack package uses a pair of approximations, given by an r-point Gauss rule and the interlacing $(2r+1)$-point Kronrod rule. The user has the option of choosing one of the pairs, with $r = 7, 10, 15, 20, 25$ or 30. The Kronrod rule supplies the local integral approximation, and the Gauss rule (together with the Kronrod rule) serves to obtain the local error estimate. The r-point Gauss rule is of polynomial degree of accuracy $2r-1$ (i.e., it integrates all polynomials of degrees $0, \cdots, 2r-1$ exactly, and there are polynomials of degree $2r$ which are not integrated exactly). The Kronrod rule is of polynomial degree $3r+1$ if r is even, and of degree $3r+2$ when r is odd (because of symmetry). The multivariate program Dcuhre applies an embedded sequence of cubature rules [5] in an adaptive setting.

2.2. Iterated integration

For integration over a product region $\mathcal{D} = \mathcal{D}_1 \times \ldots \times \mathcal{D}_\ell$, we consider the *iterated (repeated) integral*

$$If = \int_{\mathcal{D}_1} d\vec{x}^{(1)} \ldots \int_{\mathcal{D}_\ell} d\vec{x}^{(\ell)}\; f(\vec{x}^{(1)}, \ldots, \vec{x}^{(\ell)}).$$

The integration over the regions $\mathcal{D}_j$ can be performed with a multivariate or a 1D integration code, and different methods can be applied on different levels $j = 1, \ldots, \ell$. For example, assume that the computation of a 2D integral over the unit square requires the outer integration over a subinterval $[a, b] \subseteq [0, 1]$. The integral over $[a, b]$ is evaluated in the form

$$\int_a^b dx \int_0^1 dy\ f(x, y) \ \approx\ \sum_{k=1}^{K} w_k F(x_k), \tag{3}$$

where the w_k and x_k are the weights and abscissae of the local rule scaled to the interval $[a, b]$, applied in the x-direction. The function evaluation

$$F(x_k) = \int_0^1 f(x_k, y)\ dy, \qquad 1 \le k \le K, \tag{4}$$

is itself an integral in the y-direction, and is computed by the method for the inner integration. The latter may be adaptive, and subject to an error control condition of the form (2). Note that the error incurred in the inner integration will contribute to the overall integration error. Work on the integration error interface is reported in [6, 7, 8].

Subsequently, in Section 5, we give results obtained with 1D repeated integration by the program DQAG from QUADPACK, using the (10, 21)-points Gauss-Kronrod pair. Thus in that case, the weights and abscissae on the right of the approximation (3) represent those of the Kronrod rule scaled to $[a, b]$.

Figure 1 from [9] shows adaptive subdivision patterns which may be obtained when the integrand has a ridge or singular behavior along the (anti-)diagonal of the (square) integration domain. Figure 1(a) is that of a standard 2D subdivision where each selected subregion is partitioned into four subsquares of equal size. Figure 1(b) depicts an iterated integration. The integrand is evaluated at three points in the x-direction, and an adaptive subdivision is shown vertically at each of the abscissae. As noted in [9] and in low dimensions, the adaptive iterated integration is found to be more effective than the general multivariate adaptive partitioning, for handling singularities that do not line up with subdivision directions.

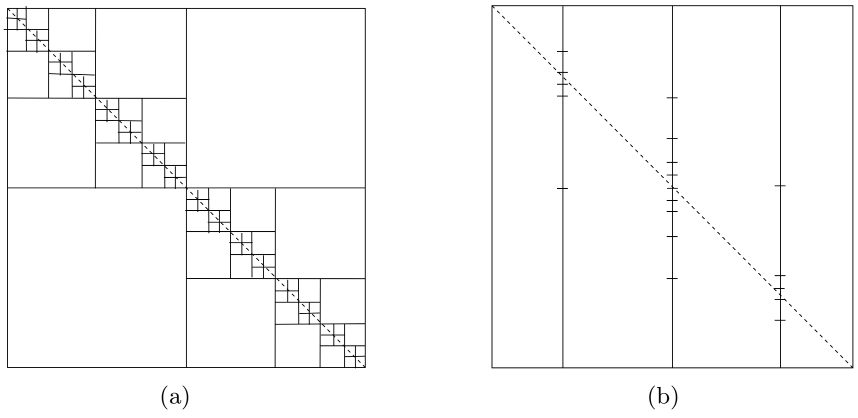


Figure 1. (a) Standard subdivision; (b) Iterated adaptive strategy for irregular behavior on diagonal [9]

3. Parallel automatic integration approach

Non-adaptive methods such as Monte Carlo, Quasi Monte Carlo and lattice rules can be parallelized in a straightforward way, by assigning function evaluations to the available processes or threads. In the approximation of the integral (1) by

$$If = \int_{\mathcal{D}} f \approx \sum_k w_k\ f(\vec{x_k}), \tag{5}$$

the function evaluations $f(\vec{x_k})$ in the weighted sum can be performed in an *embarrassingly parallel* way. While these methods are available for high-dimensional integration, they have a slow rate of convergence.

Adaptive methods are suited for integrals of low to moderate dimensions (say, ≤ 10). These are more complex than the non-adaptive techniques, and usually parallelized on the subregion level. Typically a parallel task pool scheme is implemented in a master-slave approach, for maintaining the priority queue data structure over the subregion collection.

In shared memory applications the task pool is shared, so that some type of mutual exclusion is required for task pool updates and maintenance. This causes various parallel losses (analyzed, e.g., in our "archives" [10, 11, 12]). On distributed memory systems, the task pool is distributed and updated using message passing, as done in programs layered over the Message Passing Interface (MPI [13] or Open MPI [14]). Load balancing is then needed to control the size of the local priority queues for problems where the integration has hot spots due to intensive region partitioning, in the vicinity of singularities and other irregular behavior (see, e.g., the ParInt strategies [15, 16, 17] and [18, 19]). Apart from the communication times, breaking loss (incurred at the end of the computations when slaves are working during termination detection) is an important characteristic of these methods. This also leads to the fact that parallel executions may be very different from the sequential computation, in an irreproducible manner.

For iterated integration, it would be possible to use a parallelization of the outer integration on the subregion level. However, in this paper we propose a parallelization on the function evaluation level(s) of the outer integration. For example, with regard to (3) and (4), this means that the values of the inner integrals $F(x_k)$ are computed in parallel. More generally, if $\mathcal{D} = \mathcal{D}_1 \times \mathcal{D}_2$ and S is a selected subregion $\subseteq \mathcal{D}_1$, we approximate

$$\int_{S} d\vec{x}^{(1)}\; F(\vec{x}^{(1)}) \;\approx\; \sum_{k=1}^{K} w_k\, F(\vec{x_k}^{(1)}), \tag{6}$$

$$\text{with} \quad F(\vec{x_k}^{(1)}) \;=\; \int_{\mathcal{D}_2} d\vec{x}^{(2)}\; f(\vec{x_k}^{(1)}, \vec{x_k}^{(2)})$$

and the integrals $F(\vec{x_k}^{(1)})$ are computed in parallel.

Important properties of this method include that

(i) the granularity of the parallel integration is large, especially when the inner integrals $F(\vec{x_k}^{(1)})$ are of dimension two or greater;

(ii) the points $\vec{x_k}^{(1)}$ where the function F is evaluated in parallel are the same as those of the sequential evaluation; i.e., apart from possibly the order of the summation in (6), the parallel calculation is the same as the sequential one.

4. Feynman loop integrals

In addition to the lowest order or tree level, higher order corrections are required for an accurate theoretical prediction of an interaction cross section, and for checking its agreement with the data observed at colliders. Feynman loop diagrams need to be taken into account, necessitating the calculation of *loop integrals.*

A diagram with L loops and N propagators is given in Feynman parameter space as

$$\mathcal{I} \;=\; \frac{\Gamma\left(N - \frac{nL}{2}\right)}{(4\pi)^{nL/2}}(-1)^N \int_0^1 \prod_{j=1}^{N} dx_j\, \delta(1 - \sum x_j) \frac{C^{N-n(L+1)/2}}{(D - i\,\delta\, C)^{N-nL/2}}, \tag{7}$$

where C and D are polynomials determined by the topology of the corresponding diagram and the physical parameters at hand. Loop integrals are generally divergent when denominators vanish in the integration region. We assume $\mathcal{I}$ does not suffer from other divergences, such as infrared (IR) divergence, and exists in the limit as $\delta \to 0$. To calculate the limit numerically [20], we generate a sequence of $\mathcal{I} = \mathcal{I}(\delta)$ for (geometrically) decreasing values of δ, and apply convergence acceleration or extrapolation to the limit with the ε-algorithm [21, 22].

Automatic packages, based on symbolic computations, are available for one-loop integrals [23, 24, 25, 26, 27, 28, 29, 30, 31]. However, many diagrams are required for an interaction, and the symbolic reduction leads to large sets of integrals. Furthermore, analytic integration is not possible in general, for higher orders and for general mass configurations. Therefore we apply the numerical *Direct Computation Method (DCM)* of [20, 32, 33, 34] as a building block for the computation, using (automatic) iterated integration and convergence acceleration to the limit as $\delta \to 0$. The motivational intensive nature of the computations motivates a parallelization of the individual integrations, in particular for 3D and higher-dimensional integrals.

The *DCM* building block can also be used for integrals resulting from a reduction that is not necessarily carried through to the primitive level. For example, in [35] we implemented a reduction by sector decompositions (and dimensional regularization) proposed in [36] for one-loop integrals through the hexagon, and applied *DCM* to a resulting set of 2D (vertex) and 3D (box) integrals. Especially the box integrals may still be computational intensive.

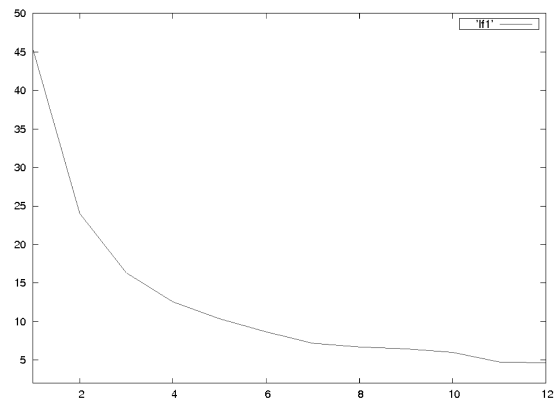
5. Results

We implemented the method of Section 2, as a multi-threaded application layered over OpenMP [1], and obtained timing results on a multi-core Intel Xeon X5680@3.33GHz CPU with 6 cores/12 logical cores, for the integrals of $If_1(\delta)$, $If_2(\delta)$ and If_3 given below in (8-10), and for a non-scalar (box) loop integral, $M_4(f, g; \delta)$ from [16]. The function f_1 is given in [37] as a test function (Dice3), for the Monte Carlo integration program Dice, and f_2 is a 3D version of the function Dice1. We treated the non-scalar loop integral M_4 in [16], using a calculation that required the integral for values of $\delta = 1.2^{30-\ell}$, $\ell = 0, \cdots, 16$, which are large compared to the value $\delta = 0.01$ used below. We also gathered preliminary results for If_3, on a Power7@3.83GHz CPU of the HITACHI SR16000 system at KEK, Japan, using up to 16 threads (maximum) assigned to our OpenMP application on this system.

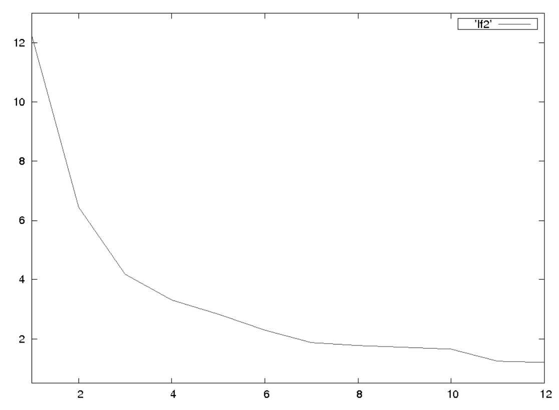
$$\begin{aligned} If_1(\delta) &= \int_{-1}^{1} dx \int_{-1}^{1} dy \int_{-1}^{1} dz \frac{\delta\sqrt{x^2+y^2+z^2}\ \theta(1-x^2-y^2-z^2)}{(x^2+y^2+z^2-a^2)^2+\delta^2}, \qquad (8)\\ & \delta = 0.01, \quad a = 0.8 \\ \theta(u) &= \ 1 \text{ if } u > 0 \\ & \frac{1}{2}, \text{ if } u = 0 \\ & 0, \text{ if } u < 0 \\ If_2(\delta) &= \int_{0}^{1} dx \int_{0}^{1} dy \int_{0}^{1} dz \frac{2\,\delta\, y}{(x+y+z-1)^2+\delta^2}, \qquad (9)\\ & \delta = 10^{-6} \\ I(f_3) &= \int_{0}^{1} dx \int_{0}^{1} dy \int_{0}^{1} dz \frac{1}{(x+y+z)^2} \qquad (10) \end{aligned}$$

and

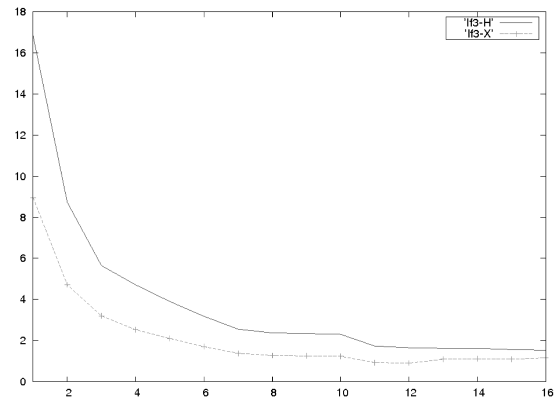
$$\begin{aligned} M_4(f, g; \delta) &= \int_0^1 dx \int_0^{1-x} dy \int_0^{1-x-y} dz \left[\frac{f(x,y,z)}{(D_4 - i\delta)^2} - 2\frac{g(x,y,z)}{D_4 - i\delta}\right], \qquad (11)\\ & \delta = 0.1 \end{aligned}$$



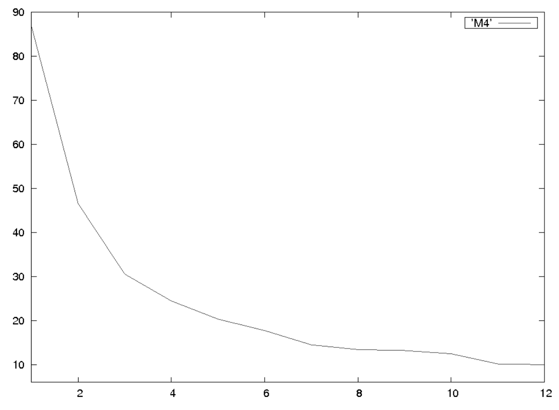
(a)



(b)



(c)



(d)

Figure 2. Times (in s): (a) Time for If_1, $\delta = 0.01$, $a = 0.8$, $t_r = 10^{-8}$; (b) Time for If_2, $\delta = 10^{-6}$, $t_r = 10^{-14}$; (c) Time for If_3, $t_r = 10^{-12}$ on Xeon X5680 (If3-X) and on SR16000 (Hitachi) node (If3-H); (d) Time for $M_4(f, g, \delta)$, $\delta = 0.1$ $t_r = 10^{-8}$

where $D_4 = \mathbf{x}^T A\mathbf{x} + 2\mathbf{v}\cdot\mathbf{x} + C$, and $A_{j,k} = q_j \cdot q_k$, $q_1 = -p_{e^-}$, $q_2 = p_{e^+}$, $q_3 = p_{e^+} - p_{W^+}$, $C = M_0^2 = M_Z^2$, $v_k = \frac{1}{2}(-q_k^2 + M_k^2 - M_0^2)$ with $M_1 = m_e, M_2 = M_W, M_3 = m_e$.

The integrals $If_1(\delta)$ and $If_2(\delta)$ exhibit a behavior not unlike that of loop integrals, with a ridge that becomes steeper and narrower as $\delta \to 0$. The integrand f_1 furthermore has a discontinuity on the spherical surface $x^2 + y^2 + z^2 = 1$, The function f_2 has a peaked behavior at $z = 1 - x - y$, and f_3 has a singularity at the origin. The integrand of M_4 in (11) has a singularity on a hyperbolic surface where $D_4 = 0$ within the (unit simplex) integration region. Note that we treated the non-scalar loop integral $M_4(f, g, \delta)$ in [16], using a calculation that required the integral for values of $\delta = 1.2^{30-\ell}$, $\ell = 0, \cdots, 16$, which are large compared to the value $\delta = 0.1$ used below.

On the Intel machine, the Intel *ifort* Fortran XE compiler was used with flag *-openmp.* The f90 compiler with flag *-omp* was used on the SR16000 node. In all cases the main loop in the rule routine DQK21 of QUADPACK was parallelized (for the outer integration), by means of a *omp parallel do* pragma for the Gauss and Kronrod rule evaluations. On comparing loop schedules, we found that the dynamic loop schedule outperforms the default static schedule for the selected test problems. In the static schedule, the participating threads are assigned their loop iterations at the beginning of the loop, in a round-robin fashion; thus the assigned iterations are determined only by the thread ID and the total number of iterations. The dynamic schedule adjusts the assignments at runtime. The threads can request more work at the completion of their previously assigned iterations, which is beneficial in cases where the integrand behavior over the subregion is non-uniform.

The timings show a good speedup pattern for a number of threads within the available number of logical cores. The runs were performed while several other users were on the system. We were able to get preliminary results on a node of the Hitachi SR16000 supercomputer at KEK (just before the system was taken down – and will be down for a move during the coming month). The SR16000 is suited for multi-core computations on each node, and for message passing applications between nodes. We plan on performing hybrid runs for loop integral calculations in the future. It appears that our multi-threaded OpenMP application runs faster on the Xeon system for the same problem parameters.

6. Concluding remarks

For compute-intensive integral calculations we have developed a multi-core parallelization on the (outer) function evaluation level in the iterated integration procedure. In this paper we give timing results of the method implemented in OpenMP, which show good speedups for tests on a Xeon X5680 based system and on a (Hitachi) SR16000 system node.

So far, only the rule application was parallelized in the outer integration, with a Gauss-Kronrod scheme for the local integrations. Future extensions of the method include hybrid (distributed shared memory) parallelizations, where individual problems are distributed to multi-core nodes, and computed using multi-threading on the nodes. Furthermore, when more cores are available per node, the rules can easily be evaluated in parallel over subregions resulting from the local subdivision of a region. Nesting of parallel constructs is also possible, so that more than one level can be parallelized. GPUs can be used on the lower evaluation level(s), particularly for cubature rules (of higher dimensions) with many evaluation points. Our future work will naturally involve implementations and testing for more computational intensive loop integrals.

Acknowledgments

This work is partially supported by Grant-in-Aid for Scientific Research (20340063) of JSPS and by the CPIS program of Sokendai (Japan). We also acknowledge the support from Western Michigan University and the College of Engineering and Applied Sciences (CEAS).

References

[1] http://www.openmp.org, OpenMP web site
[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 (Springer-Verlag)
[3] Berntsen J, Espelid T O and Genz A 1991 *ACM Trans. Math. Softw.* **17** 452–456
[4] Cools R and Haegemans A 2003 *ACM Transactions on Mathematical Software* **29** 287–296
[5] Berntsen J, Espelid T O and Genz A 1991 *ACM Trans. Math. Softw.* **17** 437–451
[6] Fritsch F N, Kahaner D K and Lyness J N 1981 *ACM TOMS* **7** 46–75
[7] Kahaner D, Moler C and Nash S 1988 *Numerical Methods and Software* (Prentice Hall)
[8] de Doncker E and Kaugars K 2010 *Procedia Computer Science* **1**
[9] Li S December 2005 *Online Support for Multivariate Integration* PhD dissertation Western Michigan University
[10] de Doncker E and Kapenga J 1987 *Numerical Integration; Recent Developments, Software and Applications* NATO ASI Series ed Keast P and Fairweather G (Reidel) pp 207–218
[11] de Doncker E and Kapenga J A 1987 *Proceedings of the Third SIAM Conference on Parallel Processing for Scientific Computing, Los Angeles* pp 109–113
[12] Kapenga J and de Doncker E 1988 *Parallel Computing* **7** 211–225
[13] http://www-unix.mcs.anl.gov/mpi/index.html, MPI web site
[14] http://www.open-mpi.org, Open MPI web site
[15] de Doncker E, Genz A, Gupta A and Zanny R 1998 *Supercomputing '98*

[16] de Doncker E, Kaugars K, Cucos L and Zanny R 2001 *Proc. of Computational Particle Physics Symposium (CPP 2001)* pp 110–119
[17] de Doncker E, Zanny R, Kaugars K and Cucos L 2001 *Lecture Notes in Computer Science* vol 2074 (Springer-Verlag) pp 118–127
[18] de Doncker E and Gupta A 1998 *Parallel Computing* **24** 1223–1244
[19] Achalla C, de Doncker E, Kaugars K and Van$\tilde{V}$oorst J 2004 *Parallel and Distributed Computing and Systems*
[20] de Doncker E, Shimizu Y, Fujimoto J and Yuasa F 2004 *Computer Physics Communications* **159** 145–156
[21] Shanks D 1955 *J. Math. and Phys.* **34** 1–42
[22] Wynn P 1956 *Mathematical Tables and Aids to Computing* **10** 91–96
[23] van Oldenborgh G J and Vermaseren J A M 2000 *Nucl. Phys. Proc. Suppl.* **89** 231
[24] Bauer C 2002 *KEK Proceedings 2002-11* vol [MZ-TH/02-04] pp 179–185 do Hoang Son, Ph.D. thesis at the Physics Department, Johannes Gutenberg Univ., 2003
[25] Bélanger G, Boudjema F, Fujimoto J, Ishikawa T, Kaneko T, Kato K and Shimizu Y 2006 *Physics Reports* **430** 117–209
[26] Giele W T and Zanderighi G 2008 *JHEP* **0806** 038
[27] van Hameren A, Papadopoulos C G and Pittau R 2009 *JHEP* **0909** 106
[28] Hirschi V, Frederix R, Frixione S, Garzelli M V, Maltoni F and Pittau R Automation of one-loop qcd corrections arXiv:1103.0621 [hep-ph]
[29] Berger C F, Bern Z, Dixon L J, Febres Cordero F, Forde D, Gleisberg T, Ita H, Kosower D A and Maitre D 2009 *Phys. Rev. D* **80** 074036 arXiv:0907.1984 [hep-ph]
[30] Binoth T, Guillet J P, Heinrich G, Pilon E and Reiter T 2009 *Comput. Phys. Commun.* **180** 2317
[31] Heinrich G, Ossola G, Reiter T and Tramontano F 2010 *JHEP* **1010** 105 arXiv:1008.2441 [hep-ph]
[32] Yuasa F, Ishikawa T, Fujimoto J, Hamaguchi N, de Doncker E and Shimizu Y 2008 *XII Adv. Comp. and Anal. Tech. in Phys. Res.* PoS (ACAT08) 122; arXiv:0904.2823
[33] 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
[34] de Doncker E, Fujimoto J, Hamaguchi N, Ishikawa T, Kurihara Y, Shimizu Y and Yuasa F 2011 *Journal of Computational Science (JoCS)* doi:10.1016/j.jocs.2011.06.003
[35] 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
[36] Binoth T, Heinrich G and Kauer N 2003 *Nuclear Physics B* **654** 277 hep-ph/0210023
[37] Tobimatsu K and Kawabata S 1998 Multi-dimensional integration routine DICE Tech. Rep. 85 Kogakuin University