Parallel High-Dimensional Integration: Quasi-Monte Carlo versus Adaptive Cubature Rules

Rudolf Schürer

Department of Scientific Computing, University of Salzburg, AUSTRIA

Abstract. Parallel algorithms for the approximation of a multi-dimensional integral over an hyper-rectangular region are discussed. Algorithms based on quasi-Monte Carlo techniques are compared with adaptive algorithms, and scalable parallel versions of both algorithms are presented. Special care has been taken to point out the role of the cubature formulas the adaptive algorithms are based on, and different cubature formulas and their impact on the performance of the algorithm are evaluated. Tests are performed for the sequential and parallel algorithms using Genz's test function package.

1 Introduction

We consider the problem of estimating an approximation $\mathrm{Q}f$ for the multi-variate integral

$$\mathrm{I}f := \int_{C_s} f(\boldsymbol{x})\, d\boldsymbol{x}$$

for a given function $f : C_s \to \mathbb{R}$, where C_s denotes an s-dimensional hyper-rectangular region $[r_1, t_1] \times \cdots \times [r_s, t_s] \subset \mathbb{R}^s$. Common methods to tackle this problem on (parallel) computer systems are presented in [1,2].

Numerical integration in high dimensions is usually considered a domain of Monte Carlo and quasi-Monte Carlo techniques. This paper will show that adaptive algorithms can be preferable for dimensions as high as $s = 40$.

2 Algorithms

2.1 Quasi-Monte Carlo Integration

Quasi-Monte Carlo methods are the standard technique for high-dimensional numerical integration and have been successfully applied to integration problems in dimensions beyond $s = 300$.

In this implementation a quasi-Monte Carlo algorithm based on Sobol's (t, s)-sequence [3,4] is used. The sequence used in particular is an $(s-1)$-dimensional sequence using Gray code order to speed up generation as described in [5]. The first point of the sequence (the corner $(0, \ldots, 0)$ of the unit cube) is skipped and a (t, m, s)-net of dimension s is constructed by adding an additional equidistributed coordinate as described in [3].

This algorithm can be parallelized easily: The net is split into equal sized blocks, with each processing node taking care of one of them. This can be implemented efficiently, because Sobol's sequence allows fast jumping to arbitrary positions. The only communication required is the final gathering of the estimates calculated by each node.

2.2 Adaptive Algorithm

The key concept of adaptive integration is to apply a basic cubature rule successively to smaller subregions of the original integration domain. The selection of these subregions adapts to "difficult" areas in the integration domain by refining subregions with large estimated errors.

The basic sequential algorithm can be outlined as follows:

1. The basic rule is applied to the whole integration domain to estimate the integral and the error of this approximation.
2. The region is stored in the region collection.
3. The region with the largest estimated error is taken from the region collection.
4. This region is split into two subregions.
5. Estimations for result and error for the new subregions are calculated.
6. Both regions are stored in the region collection.
7. If not done, goto step 3

The loop can be terminated either when a certain absolute or relative error is reached or when the number of integrand evaluations exceeds an upper bound.

Parallelization. The most straight-forward parallelization of this algorithm uses a dedicated manager node maintaining the region collection and serving all other nodes with regions for refinement (see e. g. [6,7]). However, as was shown in [8], this approach scales badly even for moderate numbers of processing nodes.

To improve scalability, all global communication has to be removed. This implies that there can not be a dedicated manager node. To balance workload, some communication between processing nodes is required. However, communication has to be restricted to a small number of nodes in the neighborhood of each processing node.

The basic idea of this algorithm is that each node executes the sequential algorithm on a subset of subregions of the initial integration domain. It uses its own (local) region collection to split regions into subregions and to adapt to difficulties found there. The union of all subregions maintained by all processing nodes is always equal to the whole integration domain. So the final result can be obtained easily by summing up the results from all processing nodes.

If this algorithm is used without further load balancing, eventually most of the processing nodes will work on irrelevant refinements on regions with low (global) estimated errors, while only a few nodes tackle "bad" regions. To avoid this problem, regions with large estimated errors have to be redistributed evenly among processing nodes.

To accomplish this, the nodes are arranged in a G-dimensional periodical mesh. If the number k of processing nodes is a power of 2, a hypercube with dimension $G = \log_2 k$ provides the optimal topology.

After a certain number of refinement steps is performed, each node contacts its direct neighbor in the first of G possible directions and exchanges information about the total estimated error in its local region collection. The node with the larger error sends its worst regions to the other node to balance the total error in both region collections. When redistribution takes place again, it is performed for the next direction. After G redistribution steps, each node has exchanged regions with all its direct neighbors, and a bad region that was sent during the first redistribution step may have propagated to any other node in the mesh by this time. This ensures that bad regions are distributed evenly among all processing nodes.

The basic idea of this algorithm was first published in [9] and was further developed in [10,11,12].

3 Cubature Rules

Adaptive algorithms are based on cubature rules with error estimation, which are defined by two terms

$$\mathrm{Q}^{(1)} f := \sum_{i=1}^{n^{(1)}} w_i^{(1)} f(\boldsymbol{x}_i^{(1)}) \quad \text{and} \quad \mathrm{Q}^{(2)} f := \sum_{i=1}^{n^{(2)}} w_i^{(2)} f(\boldsymbol{x}_i^{(2)}) \ ,$$

with $\mathrm{Q}f := \mathrm{Q}^{(1)} f$ being an approximation for $\mathrm{I}f$, and $\mathrm{E}f := \left|\mathrm{Q}^{(1)} f - \mathrm{Q}^{(2)} f\right|$ being an estimated error bound for the integration error $|\mathrm{I}f - \mathrm{Q}f|$.

Usually interpolatory rules are used, which means that $\mathrm{Q}p$ is exact for all (multivariate) polynomials p up to a certain degree d, i. e. $\mathrm{I}p = \mathrm{Q}p$ for all $p \in \mathbb{P}_d^s$. $\mathrm{Q}^{(2)}$ is usually a cubature rule with a degree less than the degree of $\mathrm{Q}^{(1)}$ and requiring significantly less integrand evaluations, i. e. $n^{(2)} \ll n^{(1)}$. In some cases, the abscissas of $\mathrm{Q}^{(2)}$ are even a subset of the abscissas of $\mathrm{Q}^{(1)}$. For these *embedded rules* no extra integrand evaluations are required to estimate the integration error.

Most empirical evaluations of adaptive integration routines focus on the comparison of different adaptive algorithms, but little is known about how the underlying cubature rules affect the performance of the algorithm. After an initial evaluation of 9 basic cubature rules, the adaptive algorithm described in the previous section was evaluated based on four different cubature rules with error estimation, leading to significantly different results.

3.1 Evaluated Rules

Table 1 lists the basic cubature rules used, together with their degree, their number of abscissas, their sum of the absolute weights $\sum_{i=1}^{n} |w_i|$, which serves as a quality parameter and should be as low as possible, and a reference to the literature.

Table 1. Basic cubature rules

Name	Degree	n	$\sum_{i=1}^{n} \|w_i\|$	Reference
Octahedron	3	$O(s)$	1	[13]
Hammer & Stroud	5	$O(s^2)$	$0.62s^2 + O(s)$	[14]
Stroud	5	$O(s^2)$	$1.4s^2 + O(s)$	[15]
Phillips	7	$O(s^3)$	$0.23s^3 + O(s^2)$	[16]
Stenger	9	$O(s^4)$	$1.2^4 + O(s^3)$	[17,18]
Genz & Malik	7	$O(2^s)$	$0.041s^2 + O(s)$	[19]

Table 2 lists the four cubature formula pairs that were actually used by the integration algorithm. Formula 7-5-5 is a special case, because it uses 2 additional basic rules $\mathrm{Q}^{(2)}$ and $\overline{\mathrm{Q}^{(2)}}$ for error estimation: $\mathrm{E}f$ is calculated by the formula

$$\mathrm{E}f := \max\left\{\left|\mathrm{Q}^{(1)}f - \mathrm{Q}^{(2)}f\right|, \left|\mathrm{Q}^{(1)}f - \overline{\mathrm{Q}^{(2)}}f\right|\right\} \quad .$$

As we will see, this construction leads to superior results for discontinuous functions, which comes at little additional cost in high dimensions, because there the total number of abscissas is dominated by the nodes of $\mathrm{Q}^{(1)}$.

Table 2. Cubature rules with error estimation

Name	$\mathrm{Q}^{(1)}$	$\mathrm{Q}^{(2)}$
5-3	Hammer & Stroud	Octahedron
7-5-5	Phillips	Hammer & Stroud, Stroud
9-7	Stenger	Phillips
7-5	Genz & Malik	

Genz & Malik does already contain an embedded fifth degree rule for error estimation, so no additional basic rule is required for 7-5. This formula has often been used throughout the literature, primarily due to its small number of abscissas for $s \leq 10$ and its exceptionally low value for $\sum_{i=1}^{n} |w_i|$. For higher dimensions, however, the number of nodes increases exponentially, resulting in a strong performance degradation.

4 Testing

Numerical tests have been performed using the test function package proposed by Genz [20]. This package defines six function families, each of them characterized by some peculiarity. Table 3 gives an overview of these functions.

Table 3. Genz's test integrand families

Integrand Family	$\|\boldsymbol{a}\|_1$	**Attribute**
$f_1(\boldsymbol{x}) := \cos\left(2\pi u_1 + \sum_{i=1}^{s} a_i x_i\right)$	$\frac{110}{\sqrt{s^3}}$	Oscillatory
$f_2(\boldsymbol{x}) := \prod_{i=1}^{s} \frac{1}{a_i^{-2} + (x_i - u_i)^2}$	$\frac{600}{s^2}$	Product Peak
$f_3(\boldsymbol{x}) := \left(1 + \sum_{i=1}^{s} a_i x_i\right)^{-(s+1)}$	$\frac{600}{s^2}$	Corner Peak
$f_4(\boldsymbol{x}) := \exp\left(-\sum_{i=1}^{s} a_i^2 (x_i - u_i)^2\right)$	$\frac{100}{s}$	Gaussian
$f_5(\boldsymbol{x}) := \exp\left(-\sum_{i=1}^{s} a_i \lvert x_i - u_i \rvert\right)$	$\frac{150}{s^2}$	$\mathcal{C}_0$ Function
$f_6(\boldsymbol{x}) := \begin{cases} 0 & x_1 > u_1 \vee x_2 > u_2 \\ \exp\left(\sum_{i=1}^{s} a_i x_i\right) & \text{otherwise} \end{cases}$	$\frac{100}{s^2}$	Discontinuous

For each family, $n = 20$ instances are created by choosing unaffective and affective parameters u_i and a_i pseudo-randomly from $[\frac{1}{20}, 1 - \frac{1}{20}]$. Afterwards the vector of affective parameters $\boldsymbol{a} = (a_1, \ldots, a_s)$ is scaled so that $\|\boldsymbol{a}\|_1$ meets the requested difficulty as specified in Table 3.

For each instance k ($1 \le k \le n = 20$) of an integrand family, the error e_k relative to the average magnitude of the integral of the current function family is calculated by the formula

$$e_k = \frac{\lvert \mathrm{I} f_k - \mathrm{Q} f_k \rvert}{\frac{1}{n} \sum_{i=1}^{n} \lvert \mathrm{I} f_i \rvert} \qquad \text{for } k = 1, \ldots, n \ .$$

For easier interpretation of e_k, the number of correct digits d_k in the result is obtained by the formula

$$d_k = -\log_{10} e_k \qquad \text{for } k = 1, \ldots, n \ .$$

Based on these values derived for each test integrand instance, statistical methods are used to evaluate the integrand family. The following charts show mean values with error bars based on standard deviation.

5 Results

All algorithms are implemented in a C++ program using MPI for inter-process communication. Standard double precision floating point numbers are used for all numerical calculations, forcing an upper bound of about 15 decimal digits on the accuracy of all algorithms. Tests are performed on an SGI Power Challenge GR located at the RIST++ (University of Salzburg), based on 20 R10000 MIPS processors.

We have chosen 20 test functions from each family in dimension 5, 7, 10, 15, 20, 25, 30, and 40. This set of functions is used to test all parallel algorithms, running on 2, 4, 8, and 16 processors. All calculations are also performed by the sequential algorithms to measure speed-up and evaluate the accuracy of the parallel algorithms. The number of allowed integrand evaluations is raised by a factor of 2 up to a maximal number of 2^{25} evaluations.

5.1 Which Algorithm Performs Best?

If quasi-Monte Carlo or an adaptive algorithm performs best, depends highly on the integrand and its dimension. While quasi-Monte Carlo degrades little for high dimensions and non-smooth integrands, the domain of adaptive algorithms is clearly that of smooth integrands in low dimensions. Which cubature rule leads to the best results is also dependent on the integrand and the dimension: While 9-7 works great for smooth integrands, 7-5-5 is better for discontinuous functions, while 7-5 is especially powerful in low dimensions.

Figure 1 shows which algorithm performs best for a given integrand family in a given dimension. If two algorithms are reported as "best" for a given problem, the one in the major part of the field achieved the best performance, but the one in the lower right corner is expected to beat the first one eventually, if the number of allowed integrand evaluations is increased beyond 2^{25}.

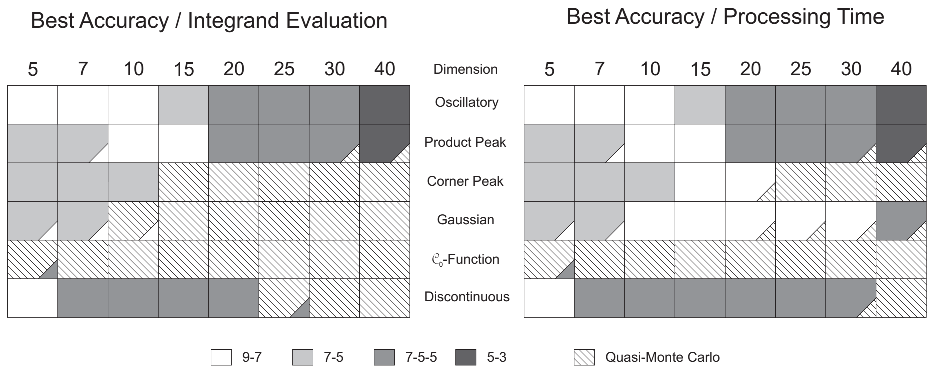

Fig. 1. Best algorithm depending on integrand family and dimension

The left chart shows which algorithm achieves the highest accuracy per integrand evaluation, while the right chart measures accuracy depending on execution time. So the right chart is most appropriate for integrands that are fast to evaluate (like the test integrands used here), while the left chart should be used for integrands that require expensive calculations. In this case the time required for integrand evaluations will dominate the total calculation time, making it most important to achieve optimal accuracy with a minimum number of integrand evaluations. The difference between these two charts is due to the different speed of the abscissa set generation for quasi-Monte Carlo and cubature rules: The time for generating a single point increases linearly with the dimension for the quasi Monte-Carlo algorithm, while the adaptive algorithm actually speeds up for increasing dimensions due to smaller region collections.

Both charts show the results for parallel algorithms running on 16 processing nodes. However, for a different number of processors, or even for the sequential algorithm, the charts are almost identical. This proves that both parallel algorithms scale equally well to at least 16 processing nodes. It follows that choosing the best algorithm does not depend on the number of processing nodes available, but only on the dimensionality of the problem and the properties of the integrand function.

5.2 Details and Discussion

This section will show in more detail how the results in Figure 1 have been obtained. For each function family and dimension, two charts have been created: The first one showing the number of correct digits depending on the number of integrand evaluation, the other one depending on execution time. Due to the huge amount of data, only a few examples can be discussed here.

The left chart in Figure 2 shows the results for Genz function f_1 (Oscillatory) for $s = 40$. This integrand is very smooth, so the adaptive algorithm performs better than quasi-Monte Carlo even for a dimension as high as $s = 40$. Only two adaptive algorithms can be seen in this chart, because the cubature rules 9-7 and 7-5 require too many points to be evaluated even a single time. 5-3 and 7-5-5, however, proof to be optimal even for high dimensions if the integrand is smooth enough.

The cubature rule showing the best performance depends on the dimension: Due to the smoothness of f_1, 9-7 performs better than 7-5 for s up to 10. For $s \geq 20$, 7-5-5 performs best and is overtaken by 5-3 for $s = 40$, because of the significantly smaller number of integrand evaluation this rule requires in this dimension.

For Genz function f_4 (Gaussian), the adaptive algorithm with the cubature rule 7-5-5 shows the best performance for dimensions up to $s = 10$. For dimensions $s = 15$ and up, however, all cubature rules are inferior to the quasi-Monte Carlo algorithm. The right chart in Figure 2 contains the result for $s = 15$, showing quasi-Monte Carlo integration superior to all four cubature rule based algorithms.

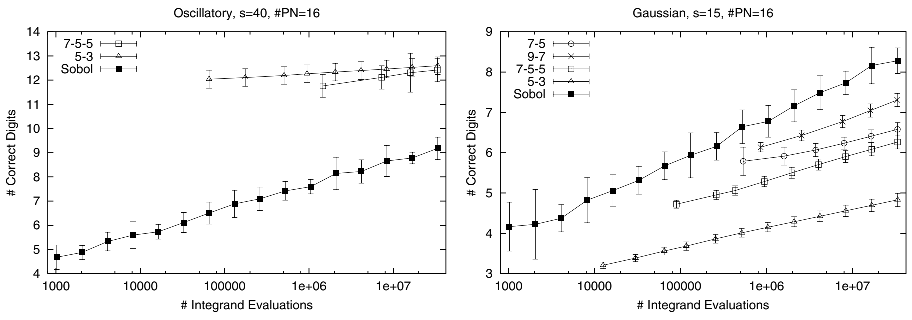

Fig. 2. Results for f_1 Oscillatory (left) and for f_4 Gaussian (right)

The left chart in Figure 3 shows Genz function f_5 ($\mathcal{C}_0$-Function) for $s = 5$. For this type of integrand, with f not differentiable on s hyperplanes, the quasi-Monte Carlo algorithm is optimal for all dimensions. For $s = 5$, it seems possible that 7-5-5 may converge faster if the number of abscissas increases beyond 2^{25}. For higher dimensions, however, the adaptive algorithms are completely outperformed.

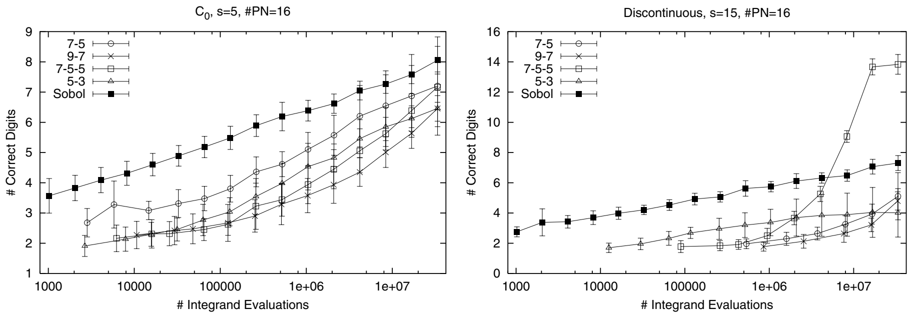

Fig. 3. Results for f_5 $\mathcal{C}_0$-Function (left) and for f_6 Discontinuous (right)

The right chart in Figure 3 shows the results for Genz function f_6 (Discontinuous) for $s = 15$. It would be reasonable to assume that the adaptive algorithms perform even worse here than for f_5. However, this is not the case. The solution to this paradox is that f_5 is discontinuous only on two hyperplanes ($x_1 = u_1 \vee x_2 = u_2$), while f_5 in not differentiable on s hyperplanes. Especially the adaptive algorithm 7-5-5 with its additional basic rule for error detection can cope with this situation, is able to refine on the regions with discontinuity and can beat the quasi-Monte Carlo algorithm even in dimensions as high as $s = 25$.

6 Conclusion

Algorithms based on quasi-Monte Carlo techniques as well as adaptive algorithms are suitable approaches for numerical integration up to at least $s = 40$ dimensions. Both algorithms can be implemented efficiently on parallel systems and provide good speedups up to at least 16 processing nodes.

If quasi-Monte Carlo or adaptive algorithms perform better depends on the dimension s of the integration problem, but also on the smoothness of the integrand. For smooth integrands, adaptive algorithms may outperform quasi-Monte Carlo techniques in dimensions as high as $s = 40$. For discontinuous or $\mathcal{C}_0$-functions, on the other hand, quasi-Monte Carlo may be superior for dimensions as low as $s = 5$.

The performance of adaptive algorithms depends highly on the cubature formula the algorithm is based on. Depending on the dimension and the type of integrand, different cubature rules should be used.

7 Acknowledgments

This work was partially supported by Österreichische Nationalbank, Jubiläumsfonds project no. 6788.

References

1. A. Krommer and C. Überhuber. *Numerical Integration on Advanced Computer Systems.* Number 848 in Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, 1994.
2. A. Krommer and C. Überhuber. *Computationl integration.* SIAM Society for Industrial and Applied Mathematics, Philadelphia, USA, 1998.
3. I. Sobol. On the distribution of points in a cube and the approximate evaluation of integrals. *U. S. S. R. Computational Mathematics and Mathematical Physics*, 7(4):86–112, 1967.
4. I. Sobol. Uniformly distributed sequences with an additional uniform property. *U. S. S. R. Computational Mathematics and Mathematical Physics*, 16:236–242, 1976.
5. I. Antonov and V. Saleev. An economic method of computing LP_τ-sequences. *U. S. S. R. Computational Mathematics and Mathematical Physics*, 19(1):252–256, 1979.
6. V. Miller and G. Davis. Adaptive quadrature on a message-passing multiprocessor. *Journal of Parallel and Distributed Computing*, 14:417–425, 1992.
7. R. Čiegis, R. Šablinskas, and J. Waśniewski. Numerical integration on distributed-memory parallel systems. *Informatica*, 9(2):123–140, 1998.
8. R. Schürer. Adaptive numerical integration on message-passing systems. In G. Okša, R. Trobec, A. Uhl, M. Vajteršic, R. Wyrzykowski, and P. Zinterhof, editors, *Proceedings of the International Workshop Parallel Numerics ParNum 2000*, pages 93–101. Department of Scientific Computing, Salzburg University and Department of Informatics, Slovak Academy of Science, 2000.

9. A. Genz. The numerical evaluation of multiple integrals on parallel computers. In P. Keast and G. Fairweather, editors, *Numerical Integration. Recent developments, software and applications*, number C 203 in ASI Ser., pages 219–229. NATO Adv. Res. Workshop, Halifax/Canada, 1987.
10. J. Bull and T. Freeman. Parallel algorithms for multi-dimensional integration. *Parallel and Distributed Computing Practices*, 1(1):89–102, 1998.
11. M. D'Apuzzo, M. Lapegna, and A. Murli. Scalability and load balancing in adaptive algorithms for multidimensional integration. *Parallel Computing*, 23:1199–1210, 1997.
12. I. Gladwell and M. Napierala. Comparing parallel multidimensional integraion algorithms. *Parallel and Distributed Computing Practices*, 1(1):103–122, 1998.
13. A. Stroud. Remarks on the disposition of points in numerical integration formulas. *Mathematical Tables and other Aids to Computation*, 11:257–261, 1957.
14. P. Hammer and A. Stroud. Numerical evaluation of multiple integrals II. *Mathematical Tables and other Aids to Computation*, 12(64):272–280, 1958.
15. A. Stroud. Extensions of symmetric integration formulas. *Mathematics of Computation*, 22:271–274, 1968.
16. G. Phillips. Numerical integration over an n-dimensional rectangular region. *The Computer Journal*, 10:297–299, 1967.
17. F. Stenger. Numerical integration in n dimensions, 1963.
18. A. Stroud. *Approximate Calculation of Multiple Integrals.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1971.
19. A. Genz and A. Malik. Remarks on algorithm 006: An adaptive algorithm for numerical integration over an n-dimensional rectangular region. *Journal of Computational and Applied Mathematics*, 6(4):295–302, 1980.
20. A. Genz. Testing multidimensional integration routines. *Tools, Methods and Languages for Scientific and Engineering Computation*, pages 81–94, 1984.