PARALLEL MULTIVARIATE INTEGRATION: PARADIGMS AND APPLICATIONS

Elise de Doncker, Laurentiu Cucos, Rodger Zanny, and Karlis Kaugars
Western Michigan University, Kalamazoo, MI 49008 USA
{elise,lcucos,rrzanny,kkaugars}@cs.wmich.edu

Key Words: parallel multivariate integration, adaptive strategies, work distribution.

Abstract:

We examine current paradigms in parallel strategies for multivariate integration algorithms. These include process structure (for example, with centralized control), and work distribution strategies (static or dynamic) in synchronous or asynchronous implementations. The target algorithm classes are Monte Carlo, quasi-Monte Carlo and adaptive. Strengths and weaknesses of the resulting parallel algorithms will be discussed, for various problem categories, in order to delineate parallel algorithm areas of effective applicability.

1. Introduction

It is our goal to give an outline of parallel computing strategies and paradigms which are of use for multivariate integration. Discussions as to why they are useful will be aimed at their parallel efficiency $E_p = S_p/p$ where $S_p = T_s/T_p$ is the speedup for p processes. Here T_p is the time of the algorithm incurred with p processes; T_s is sequential time and is interpreted either as the time of the algorithm using one process ($p = 1$), or the time of the "best" sequential algorithm.

In our work we have made ample use of the *master-slave* paradigm, as it is ultimately necessary to gather the portions of the result computed in parallel. Our more recent experiences were acquired solely on message passing systems. Layered over the MPI (Message Passing Interface) standard, our code will port on top of MPI to other message passing and some shared memory systems.

As far as we know, our distributed algorithm reported on in [13] was the first of its kind in the area of multivariate integration. It was asynchronous, meaning that the participating processes may be at entirely different stages of their execution, without the need to wait for each other. It also implemented a wait-receive loop at the beginning of its iteration, where the processes receive messages and branch to perform the appropriate actions accordingly.

PARINT [8] is a culmination of our efforts in this area. It contains general purpose algorithms designed for multivariate integration over hyper-rectangular and simplical regions and fairly low dimensions, say, not exceeding 10. These are based on adaptive methods, which improve their result by continuing to partition the integration domain. We use a global adaptive strategy, which maintains a global result and error estimate (over the entire domain), and terminates when the error estimate falls below the tolerated error level. Alternative adaptive schemes include, e.g., local adaptive and region-size adaptive [28, 7].

Our software package includes stochastic integration techniques. Quasi-Monte Carlo (QMC) methods are effective for higher dimensions and fairly smooth functions. At the basis, we use a sequence of Korobov lattice rules as in [20]. Of each rule, a fixed number of randomized samples are computed by adding a random vector to the integrand evaluation points. The randomization serves the purpose of obtaining an estimated error based on the sample variance. Laying out the successive rules column-wise and the randomized samples of each rule row-wise, the objective is to compute a table of randomized rule evaluations, where successively higher order rules are computed as needed to satisfy the required accuracy (and each row is used to estimate the error of the corresponding rule).

Our current parallel implementation distributes the computation of each row (sliced horizontally) over the processes. The first row included depends on the number of workers p, i.e., the rules of the table which involve a number of function evaluations lower than p are omitted. Indeed, a large number of processes will only be effective for large problems (requiring considerable computation and higher order rules).

We use l'Equyer's random number generator [27]. Our parallel implementation is organized so that in parallel the same sequence is computed as sequentially. For the generation of a row in QMC, the workers need to share the seed for the random number generation, since the work units are slices of the same row. The controller pre-computes the seed and sends it to the workers.

Furthermore, we provide parallel code for the special purpose applications multivariate normal and t distribution integrals, based on our parallel QMC algorithm. For high dimensions and possibly erratic conditions we include a parallel Monte Carlo (MC) method.

Section 2. describes classifications of parallel strategies based on the creation of the work. This is done with respect to the form of the overall work in Section 2.1; and with respect to where the tasks originate and task assignment (centralized, local, or hierarchical) in Section 2.2. Section 2.3 discusses is-

Supported in part by NSF grants CISE ACR-0000442, EIA-0130857, ACI-0203776

```
Evaluate 4th order differences over rgn selected for subdivision
Initialize priority queue with difference values
while (meshing limit not reached and diff spread is large)
    Retrieve largest difference from priority queue
    Bisect associated region
    Evaluate differences of new subregions
    Insert new differences into priority queue
```

Figure 1: Adaptive meshing procedure

sues related to determining task size. In Section 3. we address pre-splitting and load balancing. Load balancing issues such as scalability and work anomaly are briefly covered in Section 3.3.

2. Creation of Tasks/Work Units

2.1 Classification with Respect to the Form of Work

A first classification of strategies for automatic integration is with respect to the form of the overall work, fixed or variable.

The *fixed* form corresponds to non-adaptive methods, such as the evaluation of a pre-determined sequence of rules, until the requested accuracy is presumed to be met.

In the *variable* form, as in adaptive integration, there is no a priori outline of the work. This type of strategy adapts itself to the problem at hand and concentrates the evaluation points in areas of concern, consisting of subregions of high estimated error for the local cubature rules. With the rules based on polynomial approximations, the incentive to subdivide results from their inability to model the integrand accordingly.

2.2 Classification with Respect to Process Origin or Assignment

2.2.1 Centralized Work Generation

The work generation may be *centralized*, as in QMC and MC. Here the controller creates parametrized work units and assigns them to the workers. In QMC these are proportioned slices across a number of randomized rule samples.

2.2.2 Local Work Generation

The work may be generated *locally* in each process, as in adaptive task partitioning, where the partitioning of a task yields subtasks consisting of the evaluation of child subregions. This requires a measure of task importance, relative to that of other processes, in order to avoid the subdivision of unimportant tasks.

Uniform subdivision of multivariate regions with respect to coordinate directions generally results in exponentially increasing work (with respect to the number of dimensions) which becomes intolerable in higher dimensions. Therefore, subdivision of the domain one bisection at a time has become a classic, from [17] to [1] to now. The direction of each subdivision is based on 4th order differences of the integrand. These are computed on line segments through the centroid, in the coordinate directions for hyper-rectangular regions, or parallel to simplex edges. The adaptive PARINT algorithms employ this method.

More recently there have been efforts to allow subdivisions into more pieces, cf. the hybrid strategy of [5], and subdivision of simplex regions into two to four parts in [21], based on 4th order differences. We are examining strategies for subdivision into arbitrary numbers of subregions, for use as a component in global adaptive algorithms. This leads to the creation of an irregular mesh over a region selected for subdivision, according to the integrand behavior.

An adaptive meshing strategy can be used starting from the selected region where, at each step, the cell with the largest difference is bisected (see Figure 1). The resulting cells are integrated over and the next subregion is selected by the global adaptive integration algorithm (based on its error estimate). We are currently working on the conditions of the while loop of Figure 1, in view of the trade-off between the gain in efficiency on the one hand (by not having to rely on integration error estimates for creating the partition), and generating too many cells on the other hand. Furthermore, measures of integrand behavior other than the classic 4th order differences can be considered.

As an illustration, Figure 2 displays 10 bisections according to the regular global adaptive strategy for the integration of x^9y^2 over the unit triangle Δ. We find that the meshing procedure of Figure 1 yields a very similar partition. Other types of functions, e.g., $1/(x+y)$ (with a singularity at the origin) have given similar results.

2.2.3 Combinations of Centralized and Local Work Generation

On a large cluster, it is common practice to schedule problems on a number of subclusters simultaneously. Furthermore, with the current support for grid computing, it has become feasible to distribute the computation of large sets of problems over multiple sites. Figure 3 illustrates a process structure where a global controller GC schedules or assigns the problems to computational units. On the second level, each local controller LC is in charge of its group, and can even use an algorithm different from the other groups. We analyzed this type of hierarchical structure in [10].

2.3 Task Size

2.3.1 Factors in Performance

There are a number of issues that need to be addressed in determining the size of the work units assigned in any distributed application. First, in regard to the work carried as a whole, the following factors influence the performance:

- **Total work size** Do we know (in advance) the total amount of work that needs to be done?
- **Work structure** Does the work increment in known chunk sizes or do we have no estimate of a suitable work increase? For example, in QMC, we know that in order to

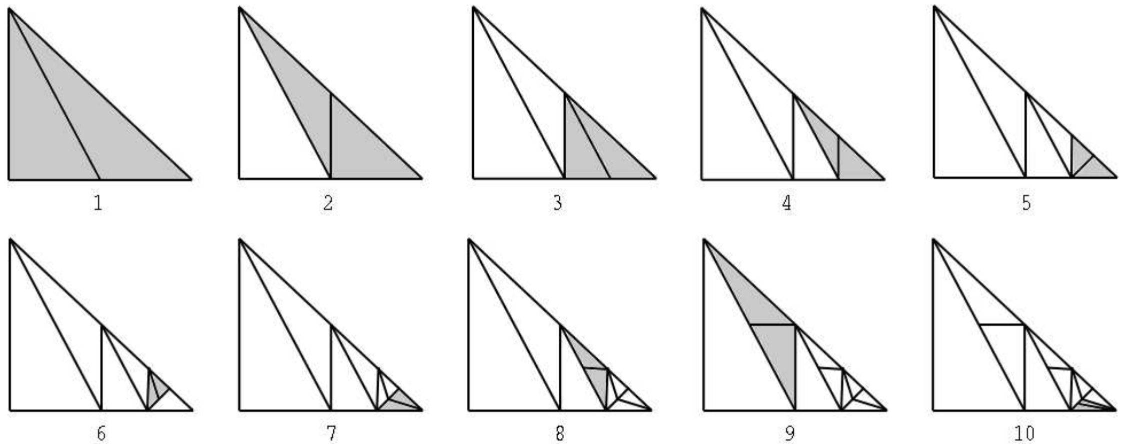

Figure 2: Triangle subdivision sequence for $\int_{\Delta} x^9 y^2 \, dx \, dy$

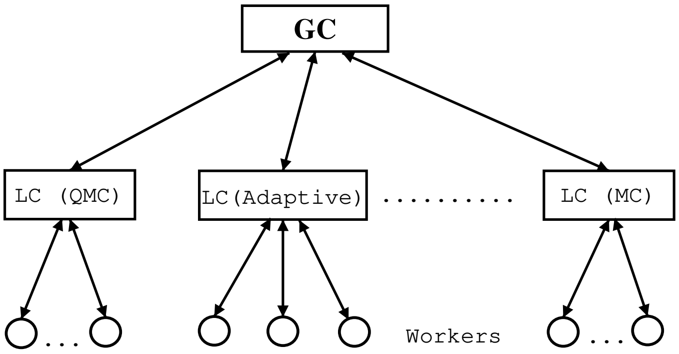

Figure 3: Hierarchical work structure

increase the accuracy we have to perform at least x more function evaluations. In adaptive integration, however, it is hard to estimate the amount of work needed to properly evaluate a subset of the region collection.

We measure the size of a task assigned to a worker in terms of the number of integrand evaluations involved, or the time the worker will spend to complete the work. Factors involved in deciding the task size for individual workers include:

- **Environmental factors:**
  - ◦ **Communication bandwidth** Low bandwidth or high latency between processors requires a larger task size.
  - ◦ **Worker computing power** Less powerful workers must receive proportionally smaller work units.

  These factors are either static (defined by the physical characteristics of the system), or dynamic (defined by the system response time when additional processes use the system concurrently).

- **Distributed computing factors:**
  - ◦ **Breaking loss** This effect occurs at the end of the computation when some processes are waiting for other ones to finish. This may happen with task sizes that are too large.
  - ◦ **Controller bottleneck** This occurs when too much information flows through the controller, as with relatively small tasks and a relatively large number of workers communicating with the controller. The overall efficiency decreases as the controller accumulates a significant number of messages in its queue and receives messages more rapidly than it can handle.

2.3.2 Classification of Task Size

For methods where we can control the task size, such as in MC and QMC (in a homogeneous or heterogeneous environment), we make the following classification:

- **Worker-independent**. The task size is independent of the worker's computing power:
  - ◦ **Time-constant**. Task size remains constant throughout the computation.
  - ◦ **Time-variable**. The task size, though predetermined, changes as the computation advances, e.g., in the assignment of QMC randomized rule evaluations as work units [16]. Their increasing size may introduce significant breaking loss [6] or communication clutter. Uniform assignment can be applied effectively for homogeneous systems. In heterogeneous environments, slower workers will process less work than faster ones.
- **Worker-dependent**. The size of the work unit depends on the worker's computing power:
  - ◦ **Static assignment**. Each worker receives an amount of work proportional with its processing power. However, it is difficult to estimate the power of the workers accurately.
  - ◦ **Dynamic assignment**. The work unit does not have a specific size but a specific processing time. Each worker processes as much as it can in the specified time in-

```
Time one function evaluation
Est. # of function evals (NFPT) that can be performed
in the requested time interval
do {Read the time after NFPT/10 function evaluations
    Use new time to adjust NFPT and next time-read interval
} until (Time expired)
```

Figure 4: Timing algorithm

terval. This handles most of our problems introduced by heterogeneous systems nicely.

We are currently experimenting with worker-dependent/dynamic task size assignment for distributed MC. In this integration method there are no dependencies between work units, and each unit can be made as small or as big as desired. The idea is to let the workers decide how much work to perform in a given interval of time. The time interval must be set up to obtain a fair trade-off between communication overhead and possible breaking loss.

Our parallel implementation of the Monte Carlo method uses a sender initiated work assignment strategy. Each worker independently evaluates the integrand function at random points, computes an approximation of the integral, and reports the results to the controller at fixed time intervals.

Issues that need to be resolved include how to gather and use timing information precisely and efficiently. Since we want to report results after a set number of function evaluations, we can use time per function evaluation as a measurement unit. Figure 4 outlines a simple algorithm.

As an alternative method, a new process/thread can be created that deals with communications and is able to keep track of elapsed time.

3. Work Distribution

In order to use parallel processes effectively, the workers should be provided with meaningful work at the start, and should be kept busy throughout the parallel computation. *Load balancing* methods attempt to move work from busy to less busy (or even idle) processes.

3.1 Static

Static work distribution involves a fixed work partitioning, for example, in an MC method where each of the p workers is assigned N/p function evaluations and N is the total number of function evaluations. If an adaptive algorithm uses a uniform initial subdivision, that is a static work distribution.

3.2 Dynamic

Dynamic work distribution is required when conditions change during the computation. For example, the amount of work at a worker in adaptive integration grows dynamically. Or, the external load on a processor may change over time, affecting the apparent processing power of the worker.

3.2.1 Initial Work Distribution

Allocating the initial tasks dynamically will usually lead to a non-uniform pre-splitting of the original task. For example, in [9] an evolutionary strategy was used for pre-splitting of the integration domain. As another strategy, the adaptive meshing procedure of Section 2.2.2 could be used.

3.2.2 Task Migration

Load balancing methods can be Sender or Receiver Initiated. Other classifications are with respect to the method used to determine the load balancing partners. We consider the following:

- **Centralized load balancing**. PARINT currently uses centralized load balancing. The controller acts as a mediator, maintaining a list of idle workers on the basis of information relayed in update messages from the workers to the controller. When a busy worker sends an update, it will receive the ID of an idle worker as a possible helper. Further handshaking and the actual transfer of work happens solely between the involved workers.
- **Neighbor or neighborhood load balancing**. Processes perform load balancing with their assigned neighbors in a neighborhood; or load balancing is performed between neighboring processors in the underlying physical or imposed architecture. Neighbor load balancing is used in the multivariate integration routine by Lapegna [26]. In [3], a ring structure is imposed and the processes communicate with their neighbors in the ring.
- **Random Polling, Random Allocation**. These do not involve a mediator and therefore scales well to large numbers of workers. Random polling is receiver-initiated, where an idle or near idle worker polls another worker for tasks, and will receive work if the request can be met. In Random Allocation, the Sender distributes work to a Receiver chosen at random. In some of our tests, Random Polling has outperformed our current centralized load balancing scheme.

3.3 Work Distribution and Scalability

Scalability refers to the ability of the parallel method to maintain efficiency as the number of processors increases. Scalability may degrade as a result of work anomaly factors. *Work anomaly* arises due to an accumulation of useless tasks. In numerical integration, this leads to performing more function evaluations as the number of processes increases. Causes may be inherent to the problem, as with local singularities. For example, a point singularity - and thus the focus of partitioning - will generally be contained within a small number of processors. Measures for inherent work anomaly are given in [31].

A model to measure scalability is given in terms of the *iso-effiency function* in [25]. This models the total work W as a

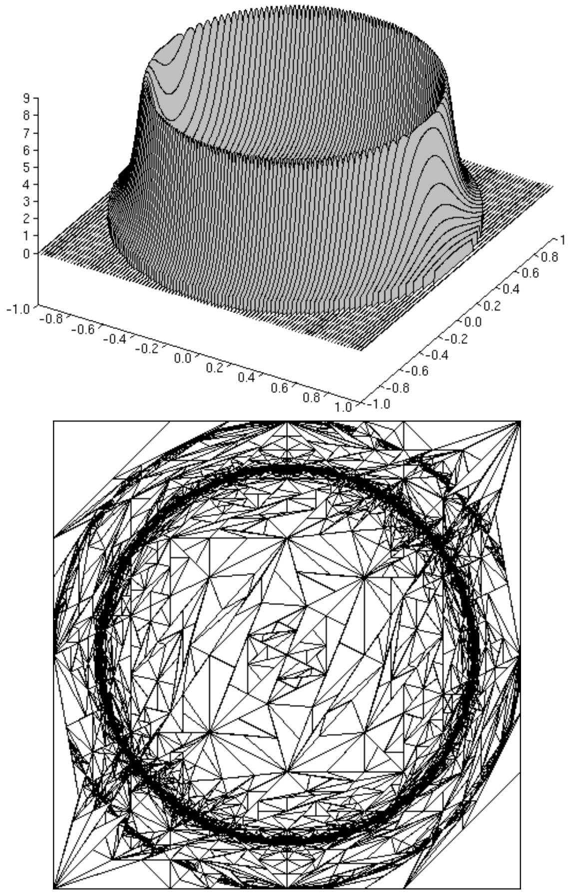

Figure 5: Surface plot (ε=0.1), Simplex regions ($\varepsilon = 10^{-6}$)

function of the number of processors (p), such that the parallel efficiency is kept constant as p increases. For example, $W = \mathcal{O}(p^2)$ would mean that the total work needs to increase as $\mathcal{O}(p^2)$ in order to maintain efficiency. The isoefficiency of various load balancing methods is studied in [25]. In [7] we used the isoefficiency model to assess the performance of parallel local and region-size adaptive integration methods.

4. Applications

In this section we focus on a problem with applications for the calculation of cross sections in high energy physics [29]. A sample integral is $\int_{-1}^{1}\int_{-1}^{1} dx_1 dx_2 \frac{\varepsilon\sqrt{x_1^2+x_2^2}\theta(1-x_1^2-x_2^2)}{(x_1^2+x_2^2-a^2)^2+\varepsilon^2}$, shown in Figure 5 (top).

Results given in [15] indicated that regular adaptive methods failed to integrate this function to a relative accuracy of 10^{-5} for small values of ε, e.g., $\varepsilon = 10^{-5}$. This was ascribed to a problem with the error estimates over the subregions generated, which caused the adaptive algorithm to focus on certain sections of the singularity, but leave large portions untouched.

We are now able to integrate this function for the sequence of $\varepsilon = 10^{-1}, 10^{-2}, \ldots, 10^{-6}$ using a method of pre-partitioning, where the subregion set resulting from the previous problem ($\varepsilon = 10^{-i}$) is used to initialize the next problem ($\varepsilon = 10^{-(i+1)}$), for $i = 1, \ldots, 5$. For the initial problem ($\varepsilon = 10^{-1}$), the unit square integration domain is subdivided into two triangles. Figure 5 (bottom) shows the subdivision obtained for the final $\varepsilon = 10^{-6}$ using the simplex integration code SMPINT [19].

5. Conclusions

In this paper we covered parallel paradigms which, in our experience, have been important for the design of parallel multivariate integration algorithms. Our treatment was targeted to heterogeneous, asynchronous, message passing systems. Our early work on numerical integration shared memory applications includes [11, 14, 24, 12]. Vectorization was utilized by Gladwell [22]. Later work on shared memory and virtual shared memory applications includes [18]. Some other work in the last decade is by Gladwell and Napierala [23], Bull and Freeman [2], and Ciegis and Sablinskas [4]. Note also that information on literature in the area of parallel integration is available at [30].

References

[1] BERNTSEN, J., ESPELID, T. O., AND GENZ, A. Algorithm 698: DCUHRE-an adaptive multidimensional integration routine for a vector of integrals. *ACM Trans. Math. Softw. 17* (1991), 452–456.

[2] BULL, J. M., AND FREEMAN, T. L. Parallel algorithms for multi-dimensional integration. *Parallel and Distributed Computing Practices 1*, 1 (1998), 89–102.

[3] CARIÑO, R., ROBINSON, I., AND DE DONCKER, E. Adaptive integration of singular functions over a triangularized region on a distributed system. In *Proc. of the 7th SIAM Conf. on Parallel Processing for Scientific Computing* (1994), R. Schreiber, Ed.

[4] CIEGAS, R., AND SABLINSKAS, R. Hyper-rectangle selection and distribution algorithm for parallel adaptive numerical integration. *Informatica 10*, 2 (1999), 161–170.

[5] COOLS, R., AND MAERTEN, B. A hybrid subdivision srategy for adaptive integration. *Journal of Universal Computer Science 4*, 5 (1998), 485–499.

[6] CUCOS, L., AND DE DONCKER, E. Distributed qmc algorithms: New strategies and performance evaluation. In *Proceedings of the High Performance Computing Symposium (HPC'02)* (2002), pp. 155–159.

[7] DE DONCKER, E., AND GUPTA, A. Multivariate integration on hypercubic and mesh networks. *Parallel Computing 24* (1998), 1223–1244.

[8] DE DONCKER, E., GUPTA, A., GENZ, A., AND ZANNY, R. http://www.cs.wmich.edu/parint, PARINT Web Site.

[9] DE DONCKER, E., GUPTA, A., AND GREENWOOD, G. Adaptive integration using evolutionary strategies. In *Proceedings of the International Conference on High Performance Computing (HiPC)* (1996), pp. 94–99.

[10] DE DONCKER, E., GUPTA, A., AND ZANNY, R. Large scale parallel numerical integration. *Journal of Computational and Applied Mathematics 112* (1999), 29–44.

[11] DE DONCKER, E., AND KAPENGA, J. Parallelization of adaptive integration methods. In *Numerical Integration; Recent Developments, Software and Applications* (1987), P. Keast and G. Fairweather, Eds., NATO ASI Series, Reidel, pp. 207–218.

[12] DE DONCKER, E., AND KAPENGA, J. Parallel systems and adaptive integration. In *Statistical Multiple Integration* (1991), N. Flournoy and R. K. Tsutakawa, Eds., vol. 115 of *Contemporary Mathematics*, pp. 33–51.

[13] DE DONCKER, E., AND KAPENGA, J. Parallel cubature on loosely coupled systems. In *NATO ASI Series C: Mathematical and Physical Sciences* (1992), T. O. Espelid and A. C. Genz, Eds., pp. 317–327.

[14] DE DONCKER, E., AND KAPENGA, J. A. A portable parallel algorithm for multivariate numerical integration and its performance analysis. In *Proceedings of the Third SIAM Conference on Parallel Processing for Scientific Computing, Los Angeles* (1987), pp. 109–113.

[15] DE DONCKER, E., KAUGARS, K., CUCOS, L., AND ZANNY, R. Current status of the ParInt package for parallel multivariate integration. In *Computational Particle Physics Conference (CPP 2001)* (2001), pp. 110–119.

[16] DE DONCKER, E., ZANNY, R., CIOBANU, M., AND GUAN, Y. Distributed Quasi Monte-Carlo methods in a heterogeneous environment. In *Proceedings of the IPDPS Heterogeneous Computing Workshop* (2000), pp. 200–206.

[17] DE RIDDER, L., AND VAN DOOREN, P. An adaptive algorithm for numerical integration over an n-dimensional cube. *Journal of Computational and Applied Mathematics 2*, 3 (1976), 207–210.

[18] FREEMAN, T. L., AND BULL, J. M. Shared memory and message passing implementations of parallel algorithms for numerical integration. In *Parallel Scientific Computing* (1994), J. Dongarra and J. Wasniewski, Eds., vol. 879 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 219–228.

[19] GENZ, A. SMPINT. Available from web page at http://www.sci.wsu.edu/math/faculty/genz/homepage.

[20] GENZ, A. MVTDST/MVTPACK: Numerical computation of multivariate t integrals, 2000. Available from web page at http://www.sci.wsu.edu/math/faculty/genz/homepage.

[21] GENZ, A., AND COOLS, R. An adaptive numerical integration algorithm for a collection of simplices, 1997. To be submitted to *Numerical Algorithms*.

[22] GLADWELL, I. Vectorisation of one dimensional quadrature codes. In *Numerical Integration; Recent Developments, Software and Applications* (1987), P. Keast and G. Fairweather, Eds., NATO ASI Series, Reidel, pp. 231–238.

[23] GLADWELL, I., AND NAPIERALA, M. A. Testing parallel multidimensional integration algorithms. *Parallel and Distributed Computing in Practice 1* (1998), 103–118.

[24] KAPENGA, J., AND DE DONCKER, E. A parallelization of adaptive task partitioning algorithms. *Parallel Computing 7* (1988), 211–225.

[25] KUMAR, V., GRAMA, A. Y., AND VEMPATY, N. R. Scalable load balancing techniques for parallel computers. *Journal of Parallel and Distributed Computing 22*, 1 (1994), 60–79.

[26] LAPEGNA, M., AND D' ALESSIO, A. A scalable parallel algorithm for the adaptive multidimensional quadrature. In *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing* (1993), pp. 933–936.

[27] L'EQUYER, P. Combined multiple recursive random number generators. *Operations Research 44* (1996), 816–822.

[28] SHAPIRO, H. D. Increasing robustness in global adaptive quadrature through interval selection heuristics. *ACM Transactions on Mathematical Software 10*, 2 (1984), 117–139.

[29] TOBIMATSU, K., AND KAWABATA, S. Multi-dimensional integration routine DICE. Tech. Rep. 85, Kogakuin University, 1998.

[30] ZANNY, R. http://www.cs.wmich.edu/parint/intg-info.html, PARINT Integration Resources.

[31] ZANNY, R., KAUGARS, K., AND DE DONCKER, E. Scalability of branch-and-bound and adaptive integration. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'01)* (2001), pp. 674–680.