On the Scalable Computation of Large Sets of Integrals

Elise de Doncker, Ajay Gupta and Laurentiu Cucos*
Department of Computer Science
Western Michigan University
Kalamazoo, MI 49008-5371

Abstract

We consider efficient strategies for the parallel and distributed computation of large sets of multivariate integrals. These arise in many applications such as computational chemistry, high energy physics and finite element problems. We use a hierarchical architecture which is highly scalable and allows parallelization not only at the domain level but also at the integral level. We present experimental results on a family of parametrized integrals and show that our approach leads to an efficient scalable load balanced algorithm. The methods are easily applicable to other areas where a large number of problems need to be solved efficiently on a parallel and distributed system.

Keywords. High performance computing, cluster computing, load balancing, integration software library, task allocation.

1 Introduction

Problems leading to the computation of large numbers of integrals occur in various disciplines such as chemistry, physics, engineering, fluid dynamics, etc. In ab initio computations in molecular orbital theory, the derivation of molecular properties requires large numbers (of the order of, e.g., $10^4 - 10^6$) of integrals [8]. Perturbation calculations yielding corrections to the interaction cross section in particle physics rely on the computation of many (Feynman diagram) loop integrals which, for a particular problem, differ in a polynomial factor in the integrand [6].

The ParInt research project [1] has led to the development of an integration software system, currently available as ParInt1.2, that allows application developers to solve their computationally intensive integration problems on a distributed computing system such as a cluster of workstations. ParInt uses MPI [2] and is available for free download from [1]. The current ParInt version provides scalable methods to compute (individual) multivariate integrals. In this paper, we describe extensions to ParInt that allow the scalable computation of large sets of multivariate integrals, in the light of current application demands. More specifically, we examine parallel integration methods and combine them into a scalable and load balanced hierarchical strategy.

In ParInt we employ adaptive, Monte Carlo and quasi-Monte Carlo methods to deal with the general integration problems. If the integrand is smooth or if it has typical difficulties (such as peaks, singularities) that can be absorbed into a fixed weight function, the application of fixed cubature rules may suffice. However, in order to be able to continually refine the integral approximations and to deal with more general integrand functions, we use adaptive strategies. These typically start with a fixed rule integral and error approximation over the given integration region $\mathcal{D}$. Subsequently $\mathcal{D}$ is subdivided, the resulting subregions are integrated over, the total integral and error approximations are updated and an "important" subregion is selected for the next subdivision. In global adaptive methods, this procedure is iterated until a global acceptance condition is satisfied, generally determined by the user's tolerated error and the allowed number of subdivisions. Important regions are determined using a variety of criteria based on the local error estimates, such as absolute error, absolute error per unit area, etc. [10].

Adaptive strategies are generally suitable when the problem dimension is relatively low, say, not exceeding 15. ParInt implements a parallel non-adaptive method based on Korobov lattice rules [4] for moderate to high dimensions and fairly smooth functions, with various applications in statistics and computational finance. We provide parallel Monte Carlo integration for high dimensions and/or erratic integrands and regions.

This paper is further organized as follows. In Section 2, we present the hierarchical approach. On the lowest level, the integration is parallelized on the subregion level. These

*Research supported in part by Western Michigan University and by the National Science Foundation under grants ACI-0000442 and ACI-0203776. Authors' e-mail addresses {elise,gupta,lcucos}@cs.wmich.edu, phone (616) 387-5645 and fax (616) 387-5997.

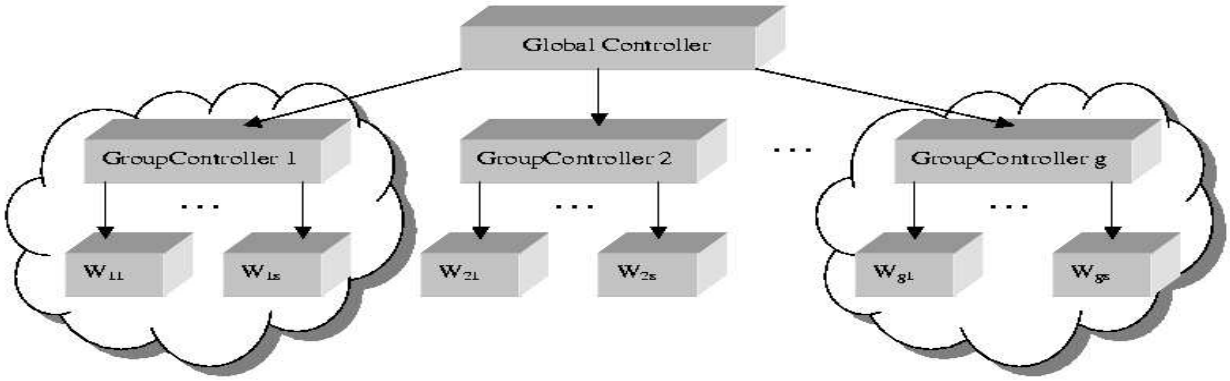

Figure 1: Hierarchical architecture

processes are then grouped for a parallelization on the integral level. Algorithm outlines are given in Section 3. Section 4 gives test results obtained on a linux cluster for six integral families, using varying group sizes and load sharing among the group processes engaged on the subregion level.

2 Hierarchical Approach for Sets of Multivariate Integrals

Let $\{I_1, I_2, \cdots, I_P\}$ be the set of P integrals. Our problem is to efficiently calculate a numerical approximation Q_k to the integral I_k, together with an error estimate E_k, for $1 \leq k \leq P$. Integral I_k is of the form $I_k = \int_{\mathcal{D}_k} F_k(\vec{x}) d\vec{x}$ where $\mathcal{D}_k$ is a hyper-rectangular region in $\Re^N$ and $F_k(\vec{x})$ is an N-dimensional integrand function over $\mathcal{D}$. The approximation Q_k and its error estimate E_k need to satisfy the condition $|I_k - Q_k| \leq E_k \leq Max\{\varepsilon_{ak}, \varepsilon_{rk}|I_k|\}$ where ε_{ak} and ε_{rk} are given absolute and relative tolerances, respectively.

Let p be the number of processors in a given parallel and distributed computing system. One approach would be to insert the integrals I_k into a queue and compute each integral using all p processors until the queue is empty. In cases where some of the integrals are easy to compute this may lead to a wastage of resources because of system overheads. Load balancing not only within an individual integral computation but also among the integrals I_k thus becomes an important issue. Furthermore, in situations where communication overheads in the given p-processor system are considerable, it may be more efficient to use, say, $s < p$ processors to compute an individual integral I_k. This would allow the computation of $g = [p/s]$ integrals in parallel and would also lead to a scalable solution. This suggests a three level hierarchical approach as depicted in Figure 1. Note that in practice the number of workers does not need to be equal in the different groups.

The global controller manages g group controllers and is responsible for assigning integrals (tasks) to groups initially and whenever a group becomes idle. For example, the global controller may give one task to each group initially; whenever a group becomes idle, it is supplied with a new integration problem. If the problems are all sufficiently similar and P is a multiple of g, this is equivalent to assigning $\frac{P}{g}$ tasks to each group initially and letting each group evaluate its sequence of integrals. If the problems are not similar, then this results in assigning the integrals depending on their computational time requirements in a load balanced manner.

Parallelization with very different characteristics can be obtained by varying the number of groups g. When $g = 1$, all p processors participate in the integration for each integral, i.e., the parallelization is on the subregion level, which may be appropriate when p is relatively small and the integrals are difficult. When p is large and/or communication costs are high, this approach may lead to too much communication overhead. When $g = p$, the integrations are performed sequentially by the individual processors, i.e., the parallelization is on the integral level. In situations where integrals of varying difficulty are involved, this may lead to load imbalances and breaking loss. For example, one processor may be working on a difficult problem while all the other ones have finished and are idle. Other values of g, $1 < g < p$, result in a true hierarchical structure. Load balancing within a group may be needed depending on the

```
global_controller() {
   initialize g groups with controllers GC_1, ..., GC_g;
   initialize integral_queue with {I_1, I_2, ..., I_P};
   send integral I_k to GC_k, for 1 ≤ k ≤ g;
   while(integral_queue is not empty) {
      wait until results from some GC_x are available;
      receive the integration results;
      delete an integral I_i from integral_queue;
      send I_i to GC_x;
   }
}
```

Figure 2: Global controller's algorithm outline

integration problem.

The hierarchical algorithm can be analyzed given the details of the communication and computation costs of the p-processor system, specific implementation of load balancing and priority queues within a group, and details of the integral approximation locally in a worker. For a detailed theoretical study we refer the reader to [5].

3 Algorithms

With reference to Figure 1, the group controllers are responsible for solving the assigned integration problems with their workers and for reporting the results to the global controller. We give an outline of the algorithm for the global controller in Figure 2. It is assumed that $P > g$. For efficiency purposes, the global controller may be implemented with a dual role - that of a group controller in addition to problem assignment.

Each group controller acts as the controller of an adaptive integration strategy (described later in the worker's algorithm outline) for its group of integration workers. We assume that each group controller also acts as a worker within its group, i.e., the group controller participates in the integration (region partitioning and rule evaluation) in addition to managing its workers. Figure 3 shows an outline of the algorithm of the group controller. After receiving an integral I_j from the global controller, the group controller and its workers continue to refine the integral approximation until the acceptance condition is satisfied. The accuracy condition used is $\sum_{R \in \mathcal{D}_j} E(R) \leq Max\{\varepsilon_{aj}, \varepsilon_{rj}|Q_j|\}$, where the sum is over all subregions in the partition and $E(R)$ represents the absolute error estimate over R.

The set of subregions at a worker are maintained in a local heap (or deap if the size of the heap is restricted by the user), in order to facilitate the fast selection of important subregions. We present an outline of the worker's algorithm in Figure 4.

A worker first receives an initial subregion R and initializes its local max-heap which will be keyed on the subregion error estimates. It then selects the subregion with the highest error estimate and deletes it from its heap in the **get_region()** function. Within **process_regions()**, this region is subdivided into smaller regions (currently we use bisection) and the resulting subregions are integrated over. The latter are added to the heap within **put_regions()** (to be considered for further subdivision). This subdivision sequence is performed ni times. The local result and error estimate are updated within **update_locals()**. If the obtained differences with the previous local result and error estimate are significant, they are sent to the group controller, in order to be applied to the global integral approximation and error estimate. If the local error tolerance is met, the worker offers its services to other busy workers for load balancing purposes.

We essentially use a receiver-initiated form of load balancing [11] in which the group controller acts as a mediator. The negotiation aspect of sharing work is handled by the pairs of workers themselves: when a worker detects that its local estimate of the integral is good enough (indicated by the locally done indicator in line 13 of Figure 4), it sends its update to the group controller with an indicator that it needs new work, and waits for a message from some other worker possibly sending work or a done message from the controller (lines 21-22).

The group controller keeps a list of all idle workers and, whenever it receives an update from a non-idle worker, sends that worker a message identifying the next available idle worker, and removes the latter from the list of idle workers. All workers regularly check for a message from the group controller offering the services of some other worker (line 15 of Figure 4). If the recipient of the offer of help is still working and not too close to finishing (line 16), it accepts the offer, sending one of its subregions (lines 17 and 18) to the helper; otherwise, it rejects the offer of help. When the idle worker receives a rejection, it informs the controller that it is still idle.

This rather simple load balancing mechanism has the advantage that the increase in the communication cost is small. Indeed, a worker only informs the group controller with a regular updating message that it is available for accepting additional work; the group controller sends one message per idle worker offering the help of that worker to a presumed non-idle worker; and a worker offered help sends one message in response to that offer. Moreover, since the worker who is offered help responds directly to the helper, the message bottleneck at the group controller is only minimally increased. In addition, the asynchronous aspect of the basic algorithm remains unchanged.

```
group_controller(){
    initialize;
    while(there is work from the global controller) {
        receive an integral I from the global controller;
        partition I's region D among its workers including itself;
        while(acceptance criterion for I is not satisfied) {
            receive partial results updates from its workers;
            update the global results;
            perform mediator actions for subregion-level load balancing;
            perform region partitioning actions as a worker;
        }
    }
}
```

Figure 3: Group controller's adaptive integration algorithm outline

```
1.  worker_integration_algorithm() {
2.      initialize;
3.      receive an initial region R from the group controller;
4.      initialize local heap of subregions with one region R;
5.      while(done message from group controller not received) {
6.          process_msg_queue();
7.          for(i=0;i<ni;i++) {
8.              get_region();
9.              process_regions();
10.             put_regions();
11.             update_locals(); }

12.         if (global criteria not satisfied) {
13.             if (local tolerance satisfied) set locally done indicator;
14.             send result updates to the group controller;
15.             if (offer of help received)
16.                 if (total local error exceeds fixed multiple of local tolerance)
17.                     get_region();
18.                     send region to idle worker;
19.                 else inform idle worker that no help is needed;
20.             while (locally done and not globally done) {
21.                 check for done message from the group controller;
22.                 check for response from a worker to offer of help;
23.             }
24.         }
25.     }
26. }
```

Figure 4: Worker's adaptive integration algorithm outline

Integrand Family		$\|\|a\|\|_1$ **Attribute**
$f_1(\vec{x}) = \cos(2\pi u_1 + \sum_{i=1}^{N} a_i x_i)$		$110\beta/\sqrt{N^3}$ Oscillatory
$f_2(\vec{x}) = \prod_{i=1}^{N} 1/(a_i^{-2} + (x_i - u_i)^2)$		$600\beta/N^2$ Product Peak
$f_3(\vec{x}) = 1/(1 + \sum_{i=1}^{N} a_i x_i)^{(s+1)}$		$600\beta/N^2$ Corner Peak
$f_4(\vec{x}) = \exp(-\sum_{i=1}^{N} a_i^2 (x_i - u_i)^2)$		$100\beta/N$ Gaussian
$f_5(\vec{x}) = \exp(-\sum_{i=1}^{N} a_i \|x_i - u_i\|)$		$150\beta/N^2$ C_0 Function
$f_6(\vec{x}) = \{$	0 if $x_1 > u_1 \vee x_2 > u_2$	$100\beta/N^2$ Discontinuous
	$\exp(\sum_{i=1}^{N} a_i x_i)$ otherwise	

Table 1: Genz's test integrand families

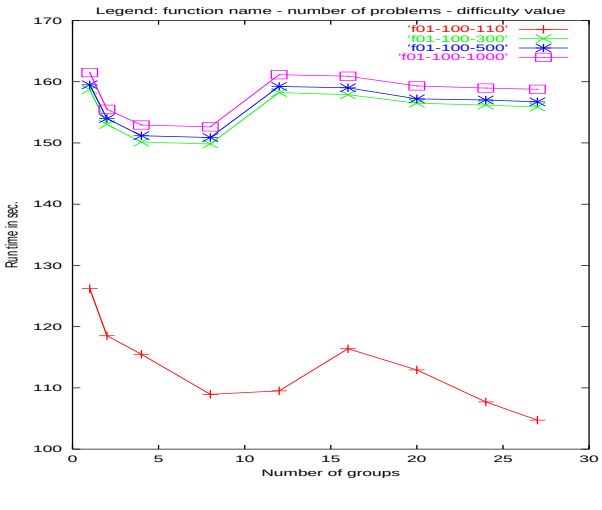

Figure 5: Timing results for $f_1(\vec{x}), \varepsilon_r = 10^{-9}$

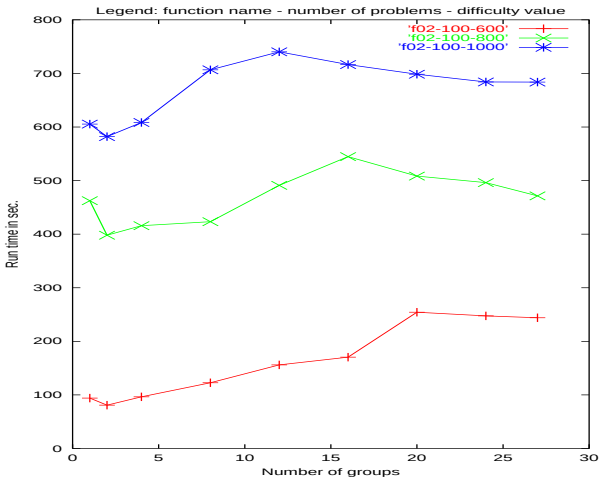

Figure 6: Timing results for $f_2(\vec{x}), \varepsilon_r = 10^{-6}$

4 Experimental Results and Analysis

We have implemented the hierarchical algorithm as an extension to ParInt 1.2 and expect that it will be available in our next release of the software package. In order to test the viability of the hierarchical approach, we selected six function families proposed by Genz [7], which characterize different peculiarities in the integrand behavior. Schürer also considers these integral families in [9] and compares quasi-Monte Carlo techniques with adaptive methods based on cubature rules. The function families are given in Table 1. In this table, N is the dimension of the integral, the u_i's and a_i's, $1 \leq i \leq N$, are the unaffective and affective parameters, respectively, and the $||a||_1$ attribute determines the difficulty level.

Our test platform was a 64-node linux cluster (*Athena* [3]), of which we used its available AMD Thunderbird 1.2GHz processors each with 512 MB RAM, and its Myrinet interconnect. The head node of the cluster is a dual-processor 1.5GHz AMD Athlon processor with 1GB RAM.

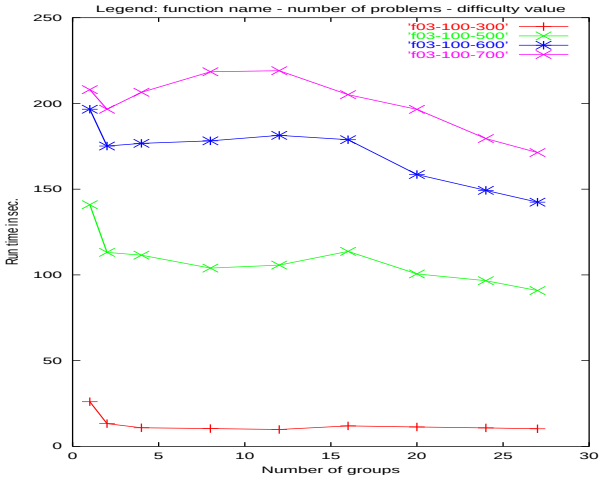

Figure 7: Timing results for $f_3(\vec{x}), \varepsilon_r = 10^{-4}$

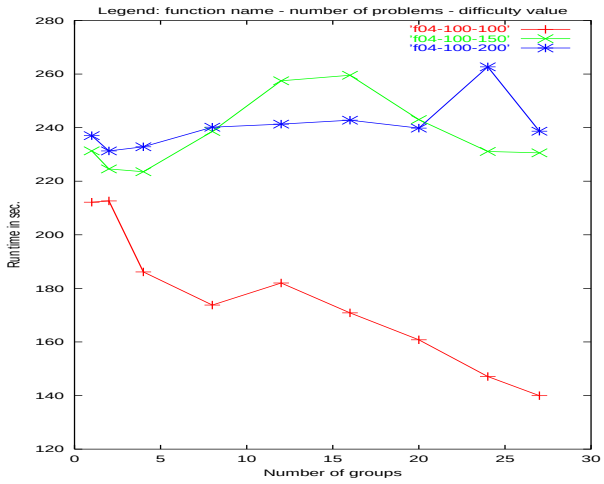

Figure 8: Timing results for $f_4(\vec{x}), \varepsilon_r = 10^{-5}$

We report typical test results for $N = 10$ and various levels of difficulty. The u_i's and a_i's are randomly chosen from $[\frac{1}{20}, 1 - \frac{1}{20}]$ and the a_i's scaled to obtain the desired difficulty level. Figures 5 to 10 graph the run times as a function of the group size for 100 instances of each parametrized family and using 28 processors. The function evaluation limit for each problem was set to 10^8. The difficulty values are given in the legends.

For the runs of Figure 5 most problems were hard, indicating a uniform work spread over the processors. For the higher difficulty levels the problems parallelize optimally using a small number of fairly large groups. The other runs typically involved a mix of easy to hard problems, leading to breaking loss when many small groups were used. We further observed, especially for the higher difficulty level of function 6 (Figure 10) and to some extent for function 3 (Figure 10) that the number of function evaluations performed was significantly higher using large groups of processors, leading to a decreased efficiency. This work anomaly effect may result due to a very local nature of the difficulties in the integrand behavior, such as the corner peak of function 3 and the discontinuity in function 6. As a result, some workers perform less important tasks than those available at the processors which have difficult subregions [12].

5 Conclusions

In this paper we investigated a system for the hierarchical computation of large sets of integrals, consisting of groups of workers with a local controller, and a global controller which deals with the allocation of tasks to the

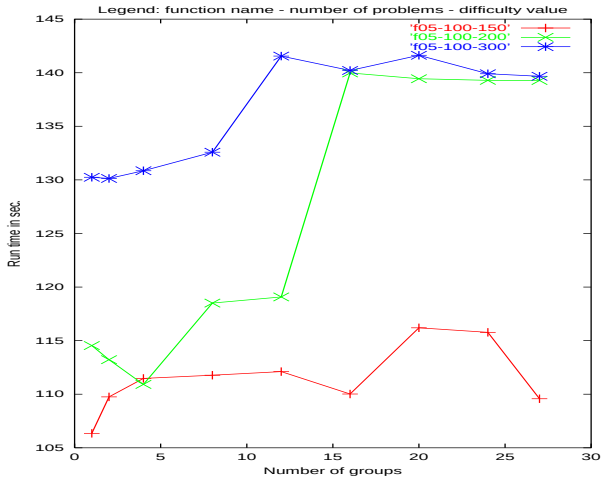

Figure 9: Timing results for $f_5(\vec{x}), \varepsilon_r = 10^{-3}$

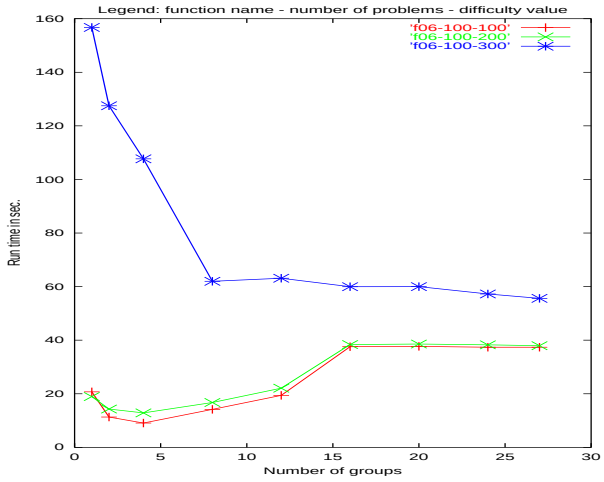

Figure 10: Timing results for $f_6(\vec{x}), \varepsilon_r = 10^{-9}$

groups. The system thus achieves a combination of parallelism at the individual problem level with that at the problem set level. It is important to recognize conditions which allow to determine suitable group sizes. Results indicate that situations which easily lead to breaking loss at the end of the problem set computation generally benefit from using larger groups.

Another reason to use groups of workers is to allow solving problems which require more memory than that available in a single processor. Furthermore, if turn-around time for subsets of problems is important and individual problems parallelize well, it will be favorable to solve individual problems on larger processor groups.

Some prior knowledge of the work structure of the problem set or the size of the individual tasks is thus useful to determine the number and size of the groups. As a future project it would be feasible to examine an alternative strategy where problems are assigned directly to workers dynamically. Initially each processor gets a problem to solve. Once a problem is solved, a new problem is assigned to the worker. If there are no more problems to be assigned, a worker can become a helper for another worker, which would help eliminate the idle time at the end of computation.

Acknowledgments

The authors thank Shalini Kollipara, Suchitha Koneru and Rodger Zanny who worked on a part of the initial code development of the hierarchical method in ParInt.

References

[1] http://www.cs.wmich.edu/parint, PARINT web site.

[2] http://www-unix.mcs.anl.gov/mpi/index.html, MPI web site.

[3] https://aegis.cs.wmich.edu, ATHENA web site.

[4] 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.

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

[6] FUJIMOTO, J., SHIMIZU, Y., KATO, K., AND OYANAGI, Y. Numerical approach to two-loop integrals. In *Proc. of the VIIth Workshop on High Energy Physics and Quantum Field Theory* (1992).

[7] GENZ, A. Testing multidimensional integration routines. In *Tools, Methods and Languages for Scientific and Engineering Computation* (1984), pp. 81–94.

[8] HEHRE, W. J., RADOM, L., SCHLEYER, P., AND POPLE, J. A. *Ab Initio Molecular Orbital Theory*. John Wiley & Sons, 1986.

[9] SCHUERER, R. Parallel high-dimensional integration: Quasi-Monte Carlo versus adaptive cubature rules. In *Lecture Notes in Computer Science* (2001), vol. 2073, Springer-Verlag, pp. 1262–1271.

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

[11] SMITH, S., AND SCHNABEL, R. Centralized and distributive dynamic scheduling for adaptive parallel algorithms. Tech. rep., Department of Computer Science, University of Colorado at Boulder, Boulder, CO, 1991. CU-CS-516-91.

[12] ZANNY, R., AND DE DONCKER, E. Work anomaly in distributed adaptive partitioning algorithms. In *Proceedings of the High Performance Computing Symposium (HPC'00)* (2000), pp. 178–183.