Performance and Irregular Behavior of Adaptive Task Partitioning *

Elise de Doncker, Rodger Zanny, Karlis Kaugars, and Laurentiu Cucos[1]

Department of Computer Science
Western Michigan University
Kalamazoo, MI 49008, USA
{elise,rrzanny,kkaugars,lcucos}@cs.wmich.edu

Abstract. We study the effect of irregular function behavior and dynamic task partitioning on the parallel performance of the adaptive multivariate integration algorithm currently incorporated in ParInt. In view of the implicit hot spots in the computations, load balancing is essential to maintain parallel efficiency. A convergence model is given for a class of singular functions. Results are included for the computation of the cross section of a particle interaction. The adaptive meshes produced by ParInt for these problems are represented using the ParVis visualization tool.

1 Introduction

We focus on the irregularity of adaptive multivariate integration, resulting from irregular function behavior as well as the dynamic nature of the adaptive subdivision process. The underlying problem is to obtain an approximation Q to the multivariate integral $I = \int_{\mathcal{D}} f(\mathbf{x})d\mathbf{x}$ and an absolute error bound E_a such that $E = |I - Q| \leq E_a \leq \varepsilon = \max\{\varepsilon_a, \varepsilon_r|I|\}$, for given absolute and relative error tolerances ε_a and ε_r, respectively. The integration domain $\mathcal{D}$ is a $d-$dimensioanl hyper-rectangular region.

For cases where the number of dimensions is relatively low, our software package ParInt (available at [2]) provides a parallel adaptive subdivision algorithm. Particularly for functions with singularities, necessarily leading to *hot-spots* in the computations, load balancing is crucial in keeping high error subregions distributed over the processors, and henceforth avoiding work anomalies caused by unnecessary subdivisions or idle times. We examine the effects of irregular behavior on algorithm convergence and scalability (using an isoefficiency model) for several types of centralized and decentralized load balancing. Furthermore we use a tool, ParVis [3], for visualizing these effects by displaying properties of the domain decomposition performed during the computations.

In Section 2 we describe the adaptive algorithm; load balancing strategies are outlined in Section 3. In Section 4 we examine scalability and algorithm convergence. Results (in Section 5) include execution times and the visualization of region subdivision patterns for an application in particle physics.

* Supported in part by the National Science Foundation under grant ACR-0000442

2 Adaptive Task Partitioning

Adaptive task partitioning is a general technique for dynamically subdividing a problem domain into smaller pieces, automatically focusing on the more difficult parts of the domain. Adaptive task partitioning is used here in the context of region partitioning for multivariate integration. The ideas also apply to problems in areas such as adaptive optimization, branch and bound strategies, progressive (hierarchical) radiosity for image rendering, adaptive mesh refinement and finite element strategies.

The adaptive algorithm for numerical integration tends to concentrate the integration points in areas of the domain $\mathcal{D}$ where the integrand is the least well-behaved. The bulk of the computational work is in the calculation of the function f at the integration points. The granularity of the problem primarily depends on the time needed to evaluate f and on the dimension; an expensive f generally results in a problem of large granularity, and in higher dimensions we evaluate more points per integration rule (i.e., per subregion or task).

Initially the integration rules are applied over the entire region, and the region is placed on a priority queue ordered by the estimated error of the regions. The regular loop iteration consists of: removing the region with the highest estimated error from the priority queue; splitting this region in half; evaluating the two new subregions; updating the overall result and error estimate; and inserting the new subregions into the queue. This process continues until the estimated error drops below the user's desired threshold, or, a user-specified limit on the number of function evaluations is reached.

In the ParInt distributed, asynchronous implementation of the adaptive partitioning algorithm, all processes act as *integration worker* processes; one process additionally assumes the role of *integration controller*. The initial region is divided up among the workers. Each executes the adaptive integration algorithm on their own portion of the initial region, largely independent of the other workers, while maintaining a local priority queue of regions (stored as a heap). All workers periodically send updates of their results to the controller; in turn, the controller provides the workers with updated values of the estimated tolerated error τ, calculated as $\tau = \max\{\varepsilon_a, \varepsilon_r|Q|\}$. A worker becomes *idle* if the ratio R_E of its total local error to the total tolerated error τ falls below its fraction R_V of the total volume (of the original domain $\mathcal{D}$). We define the *error ratio* as the ratio R_E/R_V; thus a worker becomes idle when its error ratio reaches 1. To maintain efficiency, a dynamic load balancing technique is employed to move work to the idle workers, and to generally keep the load distributed over all the workers.

A useful model for analyzing the behavior of an adaptive partitioning algorithm is the *region subdivision tree*. Each node in the tree corresponds to a region that was evaluated during the execution of the algorithm. The root represents the initial region $\mathcal{D}$; each other node has a parent node corresponding to the region from which it was formed and either zero or two children. The leaf nodes correspond to the regions on the priority queue(s) at the end of execution, and the number of nodes in the tree is equivalent to the number of region evaluations during execution [1]. Note that, as any adaptive partitioning algorithm will have methods for prioritizing, selecting, and partitioning work, the notion of a domain subdivision tree exists in any problem domain for which some sort of dynamic, progressive partitioning can be utilized.

3 Load Balancing

3.1 Scheduler Based (SB)

Consider a *receiver initiated, scheduler based* technique where the controller acts as the scheduler and keeps an IDLE-STATUS list of the workers.

The controller is kept aware of the idle status of the workers via the workers' update messages. When a worker i informs the controller via a regular update message of its non-idle status, the controller selects (in a round-robin fashion) an idle worker j and sends i a message containing the id of j. Worker i will then send a work message to j containing either new work or an indication that it has no work available. Worker j receives this message and either resumes working or informs the controller that it is still idle. This strategy was implemented in ParInt1.0.

Apart from the disadvantage of a possible bottleneck at the controller, the amount of load balancing performed by the implementation appears limited for certain test cases. In order to tailor the strategy to increase the amount of load balancing, we consider sending larger amounts of work in individual load balancing steps. The controller could furthermore select multiple idle workers to be matched up with busy workers within a single load balancing step. As described subsequently, we will also investigate decentralizations of the strategy.

3.2 Controller Info, Decentralized (CID)

In addition to the updated tolerated error τ, in this technique the controller provides the workers with its current IDLE-STATUS information. Once workers have this information, they can perform load balancing independently of the controller.

An idle worker j issues a work request to a busy worker i (selected round-robin from the list). Upon receiving the request, i either sends j work, while setting the status of j locally to non-idle, or informs j that it has no work available. In the latter case, j will change its status of i to idle. If as a result of the load balancing step, j transfers from idle to non-idle status, it informs the controller (in a regular update message), and the latter updates its IDLE-STATUS list. Note this is a receiver-initiated strategy.

3.3 Random Polling (RP) / Allocation (RA)

RP and RA are potentially simpler decentralized strategies. In RA, a busy processor periodically sends off work to a randomly selected processor. This results in a random distributing of the load and has the advantage that each load balancing step requires only one communication. As a disadvantage, no idle-status information of the target processor is used. The latter could be remedied by providing the workers with an IDLE-STATUS list (resulting in a sender-initiated version of CID), however, this may result in a large number of collisions of work allocations to potentially few idle targets.

In RP, an idle processor requests work from a randomly selected processor. Compared to RA, two communications are needed per load balancing step. However, the first is a small request message. Compared to RA, work gets transferred only if warranted according to the target processor's (non-)idle status.

4 Simple Scalability Analysis

We will use the isoefficiency model of [4] to address scalability issues, particularly with respect to load balancing. In this model, all of the load W is initially in one processor; the analysis focuses on the work needed to distribute the load over all the processors. Note that, for our application, this may be thought of as the type of situation in the case of a point singularity, where one processor contains the singularity.

For our application, the work W can be characterized as an initial amount of error of the problem (or total estimated error in the adaptive algorithm). In a load balancing step, a fraction of a worker's error w will be transferred to another worker. The model assumes that when work w is partitioned in two parts, ψw and $(1-\psi)w$, there is a constant $\alpha > 0$ (which may be arbitrarily small) such that $1-\alpha \geq \psi \geq \alpha$; $\alpha \leq 0.5$, so that the load balancing step leaves both workers with a portion bounded by $(1-\alpha)w$.

With respect to receiver-initiated load balancing, let $V(p)$ denote the number of requests needed for each worker to receive at least one request, as in [4]. If we suppose that the total error would remain constant, then as a result of load balancing, the amount of error at any processor does not exceed $(1-\alpha)W$ after $V(p)$ load balancing steps; this is used in [4] to estimate the number of steps needed to attain a certain threshold, since under the given assumptions the work at any processor does not exceed δ after $(\log_{\frac{1}{1-\alpha}} \frac{W}{\delta})V(p)$ steps. Note that this only models spreading out the work, while the total work remains constant.

In order to allow taking numerical convergence into account, let us assume that during a load balancing cycle (of $V(p)$ steps), the total error has decreased by an amount of βW as a result of improved approximations. For the sake of simplicity consider that we can treat the effect of load balancing separately from that of the corresponding integration computations, by first replacing W by $W - \beta W = (1-\beta)W$, then as a result of load balancing the amount of error at any processor does not exceed $(1-\alpha)(1-\beta)W$ at the end of the cycle. This characterizes a load balancing phase (until the load has settled) in the course of which the total load decreases. Realistically speaking, it is fair to assume that the global error will decrease monotonically (although not always accurate, for example when a narrow peak is "discovered").

Our current implementations do not send a fraction of the error in a load balancing step, but rather one or more regions (which account for a certain amount of error). In the case of a local singularity, the affected processor's net decrease in error greatly depends on whether it sends off its singular region. Furthermore our scheme is driven more by the attempt to supply idle or nearly idle processors with new work, rather than achieving a balanced load.

In the next subsection we study the numerical rate of convergence for the case of a singular function class.

4.1 Algorithm Convergence Ratio for Radial Singularities

Consider an integrand function f_ρ which is *homogeneous* of degree ρ around the origin [5], characterized by the property $f_\rho(\lambda \mathbf{x}) = \lambda^\rho f_\rho(\mathbf{x})$ (for $\lambda > 0$ and all $\mathbf{x}$). For example, $f(\mathbf{x}) = r^\rho$ where r represents the radius $\sqrt{\sum_{j=1}^d x_j^2}$ is of this form, as well as $f(\mathbf{x}) = (\sum_{j=1}^d x_j)^\rho$. Note that an integrand function of the form $f(\mathbf{x}) = r^\rho g(\mathbf{x})$, where $g(\mathbf{x})$ is a smooth function, can be handled via a Taylor expansion of $g(\mathbf{x})$ around the origin.

Let us focus on the representative function $f(\mathbf{x}) = r^\rho$, integrated over the $d-$dimensional unit hypercube. Our objective is to estimate the factor $(1-\beta)$ by which the error decreases due to algorithm convergence. We assume that the error is dominated by that of the region containing the singularity at the origin. Since our adaptive algorithm uses region bisection, it takes at least d bisections to decrease the size of the subregion containing the origin by a factor of two in each coordinate direction. We will refer to this sequence of subdivisions as a *stage*.

The error associated with this type of region is given by

$$E(k, \rho) = \int_0^{\frac{1}{2^k}} \cdots \int_0^{\frac{1}{2^k}} r^\rho d\mathbf{x} - \sum_{i=1}^{q} w_i r_i^\rho,$$

where q is the number of points in the cubature formula and r_i is the value of r at the i-th cubature point. Then $E(k+1, \rho) = 2^{-(d+\rho)} E(k, \rho)$.

Recall that a load balancing step corresponds to an update, which is done every n_s subdivisions (n_s is a parameter in our implementation). On the average $\mathcal{O}(p)$ load balancing steps are needed for each processor to do an update, i.e., get a chance to either be put on the IDLE-STATUS list as idle, or act as a donor. Note that $\sim p$ updates corresponds to $\sim n_s p$ bisections. At the processor with the singularity this corresponds to $\sim \frac{n_s p}{d}$ stages (assuming that it continually subdivides toward the singularity). Since each stage changes the error by a factor of $\frac{1}{2^{d+\rho}}$ and there are $\sim \frac{n_s p}{d}$ stages, we estimate $1 - \beta \sim (\frac{1}{2^{d+\rho}})^{\frac{n_s p}{d}}$.

Note that $d + \rho > 0$ for the integral to exist. For a *very* singular problem, $d + \rho$ may only be slightly larger than zero, thus $\frac{1}{2^{d+\rho}}$ only slightly smaller than 1 (i.e., there will be slow convergence).

4.2 Isoefficiency of SB

In our SB strategy, the work requests are driven by the worker updates (at the controller). An update from a busy processor will result in an idle processor requesting work from the busy processor (if there are idle processors). So, ignoring communication delays, we consider an update by a processor, done when there are idle processors, as a request for work to that processor (via the controller). This assumes that the computation time in between a worker's successive updates are large compared to the communication times. If we assume that each worker sends an update after a fixed number (n_s) of subdivisions (and the system is considered homogeneous), the update from a specific worker will be received within $\mathcal{O}(p)$ updates on the average. Therefore, $V(p) = \mathcal{O}(p)$.

Under the above conditions, this scheme behaves like the DONOR based SB and GRR in [4]. Consequently, the isoefficiencies for communication and contention on a network of workstations (NOW) are $\mathcal{O}(p \log p)$ and $\mathcal{O}(p^2 \log p)$, respectively (dominated by $\mathcal{O}(p^2 \log p)$).

4.3 Isoefficiency of CID

Without the IDLE-STATUS list at each worker, this scheme would be as Asynchronous Round Robin (ARR) in [4]). However, with (current) versions of the IDLE-STATUS list available at the workers, it inherits properties of SB, but avoiding algorithm contention, and attempting to avoid network contention. Further analysis is required to determine the isoefficiency of this load balancing technique.

4.4 Isoefficiency of RP

The isoefficiencies for communication and network contention on a NOW are known to be $\mathcal{O}(p \log^2 p)$ and $\mathcal{O}(p^2 \log^2 p)$, respectively (dominated by $\mathcal{O}(p^2 \log^2 p)$).

5 ParVis and Results

ParVis [3] is an interactive graphical tool for analyzing region subdivision trees for adaptive partitioning algorithms. Point singularities give rise to typical subdivision structures which can be recognized using ParVis, by displays of the subdivision tree or projections of the subdivisions on coordinate planes. The data within the nodes of the tree further indicate the location of the singularity. Non-localized singularities generally span large portions of the tree, but can often be detected in the region view projections on the coordinate planes. Error ratios between a node and its descendants provide information on asymptotic error behavior and possible conformity to established error models [1].

Figure 1 shows the subdivision tree and the projection of the subdivisions on the x_0, x_1−plane for the function

$$f(\mathbf{x}) = \frac{1}{(x_0^2 + x_1^2 + x_2^2 + x_3^2)^{\alpha}(x_0^2 + x_1^2 + (1 - x_2)^2 + (1 - x_3)^2)^{\beta}},$$

with $\alpha = .9$, $\beta = .7$, over the $4-$dimensional unit cube, to a relative error tolerance of 10^{-6}. The two pruned subtrees and the clustered subdivisions correspond to the integrand singularities at the origin and at $(0, 0, 1, 1)$.

ParVis can color the tree nodes by the range of the error estimate. With regard to performance characteristics, nodes can also be colored according to the time of generation or by the processor that owns the region. The latter gives information on the efficacy of load balancing.

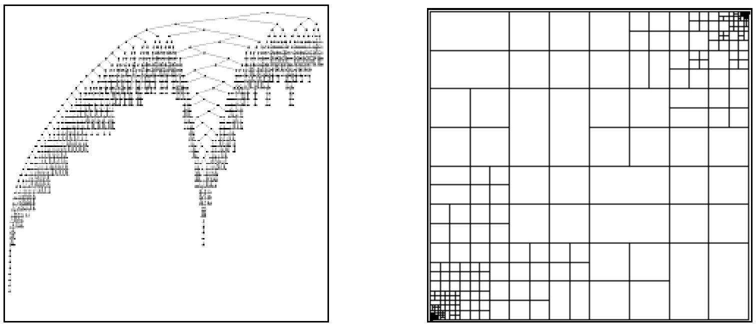

Fig. 1. Left: Subdivision tree for $f(\mathbf{x})$; Right: Region projection on x_0, x_1-plane

When the adaptive partitioning technique encounters a narrow peak in a function, the estimated error can momentarily increase as the peak is discovered, before being reduced again as the peak is adequately subdivided. We tested this behavior by creating a parameterized integrand function, where the parameters can be varied randomly to define a "family" of similar integrand functions. Each function is over a certain dimension, and contains a number of random peaks, each of some varying height and width. The function definition is

$$f(\mathbf{x}) = \sum_{i=0}^{n-1}(((\gamma_i(\sum_{j=0}^{d-1}(x_j - p_{i,j})^2))^{\rho/2}) + \frac{1}{\mu_i})$$

where n is the number of peaks and d is the dimension. The values γ_i, ρ_i, and, μ_i determine the height and "width" of peak i, and the j^{th} coordinate of the i^{th} peak is given by $p_{i,j}$. Figure 2 shows the log of the error ratio as a function of the current number of iterations. The fluctuations illustrate the temporary error increases incurred as the algorithm subdivides around the peak.

Figure 3 displays the subregions produced when one of these functions is integrated; the function parameters used specify 100 peaks for $d = 2$. The clusters of smaller subdivisions indicate the locations of the peaks. Figure 4 gives the obtained speedup vs. the number of processors, from the adaptive ParInt algorithm using our SB load balancing scheme. These runs were done on *Athena*, our Beowulf cluster, which consists of 32 (800MHz) Athlon processors connected via a fast ethernet switch.

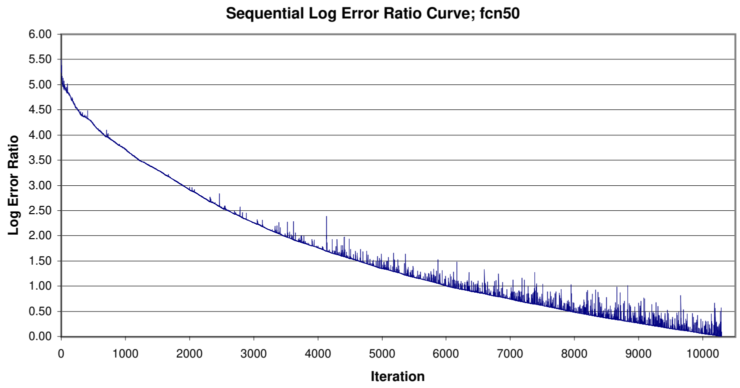

Fig. 2. Estimated error ratio log curve

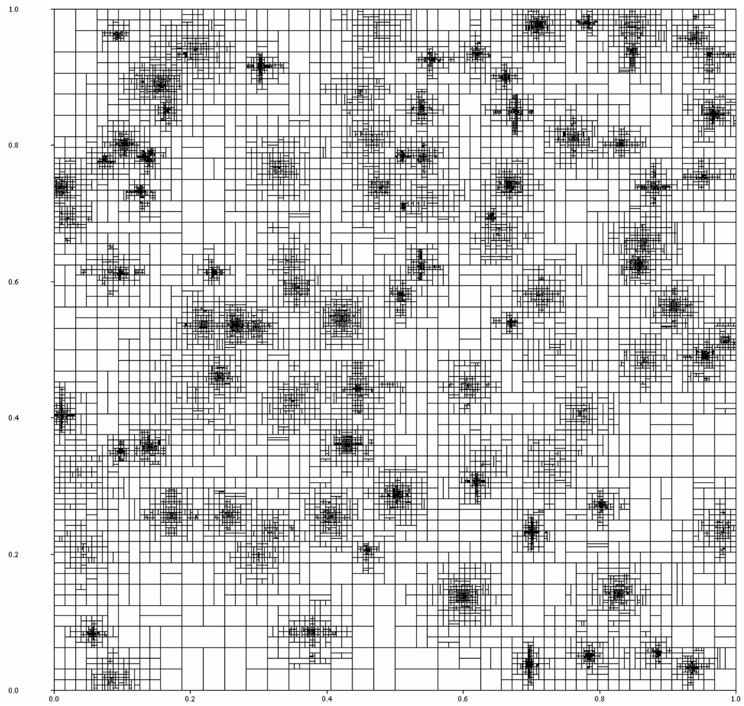

Fig. 3. Plot of subregions produced during integration of parameterized peak function

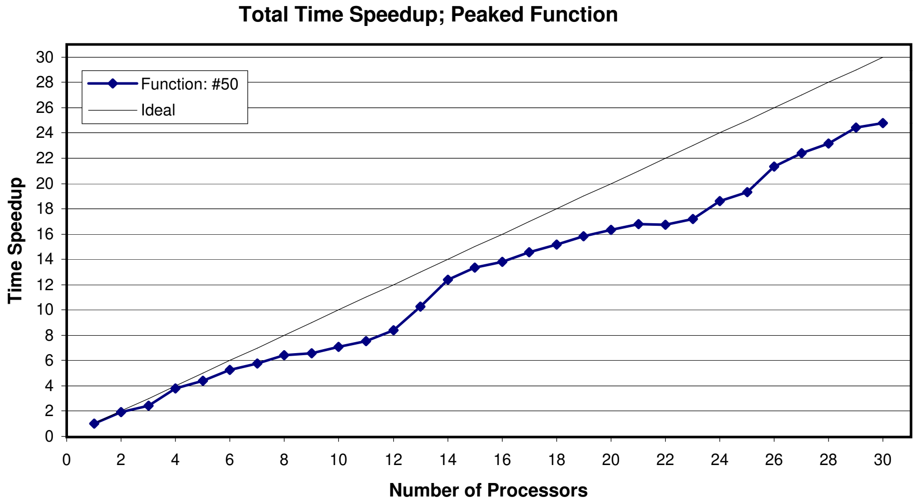

Fig. 4. Speedup graph for 2D parameterized peak function

We now consider a high energy physics problem [6]) concerned with the $e^+\ e^- \rightarrow \mu^+\ \mu^-\ \gamma$ interaction (1 electron and 1 positron collide to produce 2 muons and a photon), which leads to a 4-dimensional integration problem to calculate the *cross section* of the interaction. Adding radiative correction leads to a 6-dimensional problem. The problem dimension is increased further by adding more particles.

The function has boundary singularities, as is apparent from concentrations of region subdivisions along the boundaries of the integration domain. For the 4-dimensional problem, Figure 5 (left) depicts a projection of the subregions onto the x_1, x_2-plane, which also reveals a ridged integrand behavior occurring along the diagonals of that plane as well as along the region boundaries. Figure 5 (right) shows a time graph vs. the number of processors, from the adaptive ParInt algorithm using our SB load balancing scheme, for a requested relative accuracy of 0.05 (which results in fairly small runs).

From this figure it appears that the runtime does not further improve above 10 processors. More detailed investigation has revealed that work redundancy is a major cause of this behavior.

6 Conclusions

We studied the effect of irregular function behavior and dynamic task partitioning on the parallel performance of an adaptive multivariate integration algorithm. In view of the singular/peaked behavior of the integrands, cross sections for collisions such as $e^+\ e^- \rightarrow \mu^+\ \mu^-\ \gamma$ in high energy physics are very computational intensive and require supercomputing to obtain reasonable accuracy.

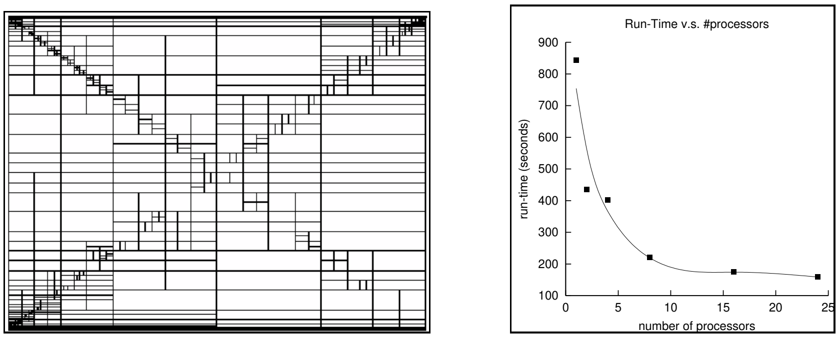

Fig. 5. Left: Region projection on x_1, x_2-plane; Right: Time graph on *Athena*

Adaptive subdivision methods hold promise on distributed computing systems, providing that the load generated by the singularities can be balanced effectively. We use the adaptive subdivision technique in ParInt to manage the singularities, in combination with the ParVis visualization tool for a graphical integrand analysis.

Further work is needed for a detailed analysis of the interactions of particular aspects of the parallel adaptive strategy (e.g., of the balancing method) with the singular problem behavior.

Acknowledgement. The authors thank Denis Perret-Gallix (CERN) (CNRS Bureau Director, Japan) for his cooperation.

References

1. E. de Doncker and A. Gupta. Multivariate integration on hypercubic and mesh networks. *Parallel Computing*, 24:1223–1244, 1998.
2. Elise de Doncker, Ajay Gupta, Alan Genz, and Rodger Zanny. http://www.cs.wmich.edu/˜parint, ParInt Web Site.
3. K. Kaugars, E. de Doncker, and R. Zanny. ParVis: Visualizing distributed dynamic partitioning algorithms. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'00)*, pages 1215–1221, 2000.
4. V. Kumar, A. Y. Grama, and N. R. Vempaty. Scalable load balancing techniques for parallel computers. *Journal of Parallel and Distributed Computing*, 22(1):60–79, 1994.
5. J. N. Lyness. Applications of extrapolation techniques to multidimensional quadrature of some integrand functions with a singularity. *Journal of Computational Physics*, 20:346–364, 1976.
6. K. Tobimatsu and S. Kawabata. Multi-dimensional integration routine DICE. Technical Report 85, Kogakuin University, 1998.