Online Support for Multivariate Integration

Shujun Li
Western Michigan University

Follow this and additional works at: https://scholarworks.wmich.edu/dissertations

Part of the Computer Sciences Commons

Recommended Citation
Li, Shujun, "Online Support for Multivariate Integration" (2005). Dissertations. 964.
https://scholarworks.wmich.edu/dissertations/964

This Dissertation-Open Access is brought to you for free and open access by the Graduate College at ScholarWorks at WMU. It has been accepted for inclusion in Dissertations by an authorized administrator of ScholarWorks at WMU. For more information, please contact wmu-scholarworks@wmich.edu.
ONLINE SUPPORT FOR MULTIVARIATE INTEGRATION

by

Shujun Li

A Dissertation
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Doctor of Philosophy
Department of Computer Science

Western Michigan University
Kalamazoo, Michigan
December 2005
In this dissertation, we present an online problem solving environment for multivariate numerical integration. To build an environment for users with various backgrounds, we address issues of Web interface support for end-users, remote application programming interfaces for distributed system programmers, visualization of multivariate integrand functions for problem analysis, and highly efficient integration methods. Compared to previous implementations, our new load balancing schemes increase the scalability of parallel integration and hence are suitable for large scale computations. A parallel integration service, called PI service, was implemented based on Web services technologies to support both remote integration and remote visualization. The multivariate function visualization makes use of adaptive numerical integration to sample points more efficiently. On the other hand, it helps solve numerical integration problems. Theoretical and experimental results show that the iterated integration method outperforms traditional adaptive strategies significantly for a broad class of problems, including some with non-smooth integrands. Iterated methods were previously considered unsuitable for these. We implemented this approach in our integration package and recommend it as one of the basic numerical integration methods.
This work was completed with the generous contributions of so many faculty members, colleagues, friends, and family members. I want to express sincere appreciation for their encouragement, support, inspiration, patience, and guidance.

I would like to gratefully acknowledge my dissertation committee chair and advisor, Dr. Elise de Doncker for her critical eye and enlightened mentoring. Without her this dissertation would not have been possible.

Special gratitude goes to committee member Dr. Karlis Kaugars for his inspired guidance that helped me understand computer science in a different way. I deeply appreciate committee member Dr. Ian Robinson for his constructive feedback.

Many thanks are also due to my colleagues Chandrasekhar Achalla, Stephanie Brenton, Paul Castone, Laurentiu Cucos, and Jeffrey R. Van Voorst for their help and support.

I owe tremendous gratitude to my parents for their whole-hearted love. I would like to dedicate my dissertation to my wife, my daughters and my sisters. Their support and patience led to the completion of my graduate study.
# TABLE OF CONTENTS

ACKNOWLEDGMENTS ............................................................................. ii  
LIST OF TABLES .................................................................................. vi  
LIST OF FIGURES ................................................................................. vii  

1 **Introduction** ................................................................. 1  
   1.1 Distributed Numerical Integration ............................... 1  
   1.2 Highly Efficient Integration Methods ......................... 3  
   1.3 Challenges of Distributed Numerical Integration .......... 4  

2 **Background on Distributed Computation and PARINT** ............ 7  
   2.1 Numerical Integration Overview ................................ 7  
   2.2 Web-based Computation ............................................. 8  
   2.3 Grid Computing and Web Services Technologies ........... 9  
   2.4 Multidimensional Function Visualization .................... 10  
   2.5 Iterated Methods ...................................................... 12  

3 **Parallel Integration Online** ................................................. 14  
   3.1 Introduction to Problem Solving Environment ............... 14  
   3.2 Online Integration .................................................... 16  
   3.3 Performance Evaluation ............................................. 24  
   3.4 Conclusions ........................................................... 26  

4 **Numerical Integration Service** .............................................. 27  
   4.1 Platform Independent Integration ............................ 27
### Table of Contents - Continued

4.2 Format of Integration Parameters ................................................. 29  
4.3 Integration Service ................................................................. 31  
4.4 Client-side Adaptive Algorithms ................................................ 33  
4.5 Performance Evaluation ............................................................. 34  
4.6 Summary .................................................................................. 36  

5 Adaptive Multidimensional Function Visualization ......................... 37  
5.1 Overview ................................................................................. 38  
5.2 Adaptive Visualization ............................................................... 40  
5.3 Two-dimensional Function Visualization ....................................... 41  
5.4 Three-dimensional Function Visualization ..................................... 46  
5.5 Dimension Reduction ................................................................. 53  
5.6 Summary .................................................................................. 56  

6 Iterated Integration Method ............................................................... 58  
6.1 Definition of the Iterated Method .................................................. 58  
6.2 Implementation .......................................................................... 59  
6.3 Complexity Analysis .................................................................... 65  
6.4 Numerical Results of Iterated and Adaptive Methods ..................... 73  
6.5 Summary .................................................................................. 80  

7 New Load Balancing Method Implementation in PARINT .................... 81  
7.1 Introduction ............................................................................... 81  
7.2 Load Balancing Methods in PARINT3 ......................................... 81  
7.3 Speedup and Efficiency ............................................................... 85
# Table of Contents

7.4 Summary ................................................................. 87

8 Conclusions ................................................................. 88

Appendices ........................................................................ 90

A Visualization of Adaptive Partitioning ............................ 90

B A Sample XML File for Integration ................................. 92

Bibliography ................................................................. 98
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Numerical results for (4.1)</td>
<td>35</td>
</tr>
<tr>
<td>6.1</td>
<td>Performance of the adaptive method for ridges with various orientations.</td>
<td>71</td>
</tr>
<tr>
<td>6.2</td>
<td>Numerical results for a collection of functions (I)</td>
<td>75</td>
</tr>
<tr>
<td>6.3</td>
<td>Numerical results for a collection of functions (II)</td>
<td>76</td>
</tr>
<tr>
<td>6.4</td>
<td>Functions with a corner peak</td>
<td>77</td>
</tr>
<tr>
<td>6.5</td>
<td>Functions $fcn8$ and $fcn9$</td>
<td>78</td>
</tr>
<tr>
<td>6.6</td>
<td>Oscillatory functions</td>
<td>79</td>
</tr>
<tr>
<td>6.7</td>
<td>Miscellaneous functions</td>
<td>79</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

3.1 Pion architecture. ................................................................. 17
3.2 Adding a new integrand function. ........................................... 18
3.3 Speedup trials. ................................................................. 25

5.1 Uniform sampling wireframes with Maple. ............................... 42
5.2 Color scale for all AdaptView images. ...................................... 43
5.3 Visualization of cell result of (5.1), where $\varepsilon = 0.001$. .......... 44
5.4 Visualization of cell max of (5.1), where $\varepsilon = 0.001$. .......... 44
5.5 Visualization of cell max of (5.1), where $\varepsilon = 0.0001$. ......... 45
5.6 Visualization of cell result of (5.1), where $\varepsilon = 0.1$. .......... 45
5.7 Snapshot of cell error animation of formula 5.1, where $\varepsilon = 0.001$. 46
5.8 Snapshot of computation for function (5.2) (I). ......................... 48
5.9 Snapshot of computation for function (5.2) (II). ......................... 48
5.10 Snapshot of computation for function (5.2) (III). ..................... 49
5.11 Dice 3D visualization for $\varepsilon = 0.001$. ............................. 49
5.12 Four point loop integral in mode max/np (I). ......................... 51
5.13 Four point loop integral in mode max/np (II). ......................... 51
5.14 Four point loop integral in mode max/np (III). ....................... 52
5.15 Four point loop integral in mode result/error. ....................... 52
5.16 Dimension reduction. ..................................................... 55
List of Figures - Continued

5.17 Dimension reduction of a five dimensional function. .......................... 56
5.18 Visualization of a four-dimensional physics function. ........................ 57
6.1 Corner peak with adaptive method. .................................................. 67
6.2 Corner peak with iterated method. .................................................. 67
6.3 Parallel to axis with adaptive method. .......................................... 68
6.4 Parallel to axis with iterated method. .......................................... 69
6.5 Diagonal ridge with adaptive method. ........................................... 69
6.6 Diagonal ridge with iterated method. ........................................... 69
6.7 Diagonal surface of 3D function. ................................................... 71
6.8 Function fcn8. ................................................................. 78
7.1 Speedup and efficiency. ................................................................. 86
CHAPTER 1

Introduction

1.1 Distributed Numerical Integration

Numerical integration plays an important role in particle physics, computational chemistry, finance, and many other fields. PARINT [76] is a software package that numerically solves integration problems in parallel. It integrates a function over hyper-rectangular or simplex regions, using adaptive cubature (quadrature), Quasi-Monte Carlo (QMC) rules, or Monte-Carlo (MC) rules.

Although integration can be done locally on a standalone machine, providing parallel/distributed support for numerical integration enables high performance computation. Computationally intensive problems can be solved with less effort. In order to build a problem solving environment (PSE) [87, 71, 64, 28, 42] for distributed integration, we need a user-friendly interface, a remote application programming interface for advanced users, a way to analyze the problem for selecting a better method, and a powerful integration engine (program) running on the back-end machines to support computational requests.

Our solution to the user interface issue is Pion (Parallel Integration ONline) described in Chapter 3. Users with or without parallel programming experience are able to access parallel computational resources via Pion from a Web browser. The user submits an integrand function with integration parameters to the server, which does the computation in parallel and gives the results back.

Unlike Pion, PI (Parallel Integration) service described in Chapter 4 aims at both end-
users and programmers. It is based on Web services [89] technologies to provide a globally accessible service over protocols such as HTTP. Our PI client can be used to access multiple service end points. Programmers have the option to write their own client that handles load balancing in a special way. PI service serves as a basis for massive scale distributed numerical integration. More work still needs to be done in this Grid [35] computing area.

1.1.1 Multidimensional Function Visualization

Function visualization gives an view of the integrand function. It assists us in selecting the proper integration method. The efficiency of a method depends greatly on the behavior of the integrand function. Unfortunately, the shape of the function is often difficult, if not impossible, to imagine. Visualization can also help us monitor the computation. Visualization can be done locally or remotely in a distributed environment.

Numerical function integration and visualization are closely related. Both involve function evaluation. The quality of function sampling determines the quality of the integration or the visualization. The areas containing information of interest for visualization are the non-smooth areas, which will be evaluated more often with an adaptive integration method.

We implemented a multidimensional function visualization tool named AdaptView. Adaptive integration is utilized to reduce the number of function evaluations, and generate time series data. The integration domain is partitioned into a uniform grid. The size of a cell is specified by the user and independent of the sub-regions used in the adaptive method. A grid cell can be sampled many times, or not sampled at all, depending on the function properties and the integration rule used. A cell contains function evaluation data
fields such as the estimated integration error and the maximum function value. These fields are filled by PARINT and visualized by AdaptView.

The Grid-based visualization service [52] for AdaptView helps monitor the progress of the computation, and gives a vivid view of the shape of a multidimensional function. We introduced a dedicated data processing method to support client image rendering. The whole system still has room for optimization.

AdaptView is a tool for general function visualization. It uses integration, and is used for integration. Compared to the traditional function visualization methods [41, 3, 37, 24, 85, 38], it has many advantages, including: tracking non-smooth regions of a function efficiently, visualizing two-dimensional functions effectively, exploring higher-dimensional ($n \geq 3$) functions by aggregation or projection, separating evaluation and visualization, and helping solve integration problems in many other ways.

1.2 Highly Efficient Integration Methods

Powerful integration engines are designed to support online integration. Many integration rules, methods and computer programs are available for numerical integration [14, 83, 61, 11, 66, 68, 29] PARINT [76] is one such tool. Two important requirements of the integration engine are memory usage and scalability. Adaptive cubature methods typically consume large amounts of memory for a difficult multi-dimensional problem. New load balancing methods are preferred for PARINT if we want to extend the scale of parallelism.

It is clear that integration can be iterated as stated by Fubini’s theorem in mathematical
analysis. However, the potential of the iterated method cannot be easily noticed from the perspectives of theory and implementation. We analyze the time complexities of the numerical iterated method for functions with different properties. The space complexity is much less than that of adaptive cubature methods. Therefore, in practice, many processes can run on one machine concurrently. Simulations have been performed on a large class of functions. The results show that the iterated method is extremely efficient for many problems. We believe that iterated methods should be included in the common numerical packages as the basic integration methods.

New load balancing methods are designed and implemented in the new version of PARINT to increase the scalability of distributed computation. Regions are pre-split adaptively into a number of sub-regions, which are then distributed to the worker nodes. This method can generally increase efficiency especially for time-consuming problems. An adaptive Quasi-Monte Carlo method that makes use of domain decomposition was added to PARINT. It can be used for high performance integration of some classes of high-dimensional problems.

1.3 Challenges of Distributed Numerical Integration

There are many challenges of numerical integration. The first is how to select or design an integration method for a given problem. The question has been asked repeatedly and has also been answered over and over again. A general belief is that, when the dimension is not too high (less than 10, for example), adaptive methods based on high polynomial precision cubature rules usually perform well, while MC and QMC methods can be considered for
higher-dimensional problems. A cubature rule is of polynomial degree of precision \( d \) if it
gives the exact result for all polynomials of degree 0 through \( d \), and not for all polynomi-
als of degree \( d + 1 \). Schürer reports in [69] that adaptive methods outperform MC/QMC
techniques for a set of problems of dimensions up to 100. QMC methods can also be used
effectively in low dimensions as demonstrated with the two-dimensional program r2d2lri in
[66]. A convincing comparison in terms of the type of integrand function, dimension, error
tolerance, and integration method is not available yet. Integration service will make this
study more practical when all important methods are available by calling remote functions.

The second challenge is the scalability of integration. A scalable problem can be solved
efficiently in parallel. There is no doubt that there exist near linear scaling problems. Our
main task is to evaluate the scalability of a practical problem and select or design a method
to solve it. Our work reveals that many computationally intensive problems scale well, and
thus are suitable to be handled with integration service. We will show that the workload
can be distributed rather evenly with the parallel iterated method.

The third challenge is that of software interfaces. The problem definition is similar for
many integration packages. Therefore, service end points could have standardized inter-
faces. We proposed an XML-based data format for parallel numerical integration [23]. It is
extensible. Appendix B gives an example input file in XML format. This work will involve
the cooperation of multiple parties. Its progress depends on many factors.

Other issues such as authentication and event notification are also critical. They are not
directly related to numerical integration but are the underlying components for distributed
systems.
This dissertation is partially based on the work of the articles [49, 51, 23, 50, 52]. The following chapters are organized as follows. Chapter 2 provides some background of PARINT and distributed numerical integration. Chapter 3 introduces a Web-based system (Pion) for end-users to solve integration online. Chapter 4 describes the construction of integration (PI) service using Web services technologies for computationally intensive problems. Chapter 5 proposes a multidimensional function visualization method based on adaptive integration. A tool named AdaptView can be utilized to monitor distributed integration remotely. Chapter 6 explores iterated methods. Chapter 7 deals with the implementation of new load balancing methods. Conclusions and future work will be outlined in Chapter 8.
CHAPTER 2

Background on Distributed Computation and PARINT

2.1 Numerical Integration Overview

The aim of numerical integration is to calculate an approximation $Q$ of the integral,

$$I = \int_D f(x) \, dx,$$  \hspace{1cm} (2.1)

over a given $n$-dimensional hyper-rectangular or simplex region $D$. Many rules and packages have been designed to handle this important problem [14, 83, 10, 33, 61, 76, 66, 65].

PARINT [76, 90] is a software package that numerically solves integration problems in parallel. Multivariate integrand functions are integrated over hyper-rectangular or simplex regions, using adaptive cubature, Quasi-Monte Carlo (QMC), or Monte-Carlo (MC) rules. For a function $f(x)$, PARINT attempts to find an approximation $Q$ of integral (2.1) and an error estimate $E_a$, such that

$$|I - Q| \leq E_a \leq \max\{\varepsilon_a, \varepsilon_r |Q|\},$$  \hspace{1cm} (2.2)

where $\varepsilon_a$ and $\varepsilon_r$ are user specified absolute and relative error tolerances, respectively. The integrand may be a vector function $f(x)$, as long as the component functions are sufficiently similar (to allow their integral approximation using the same strategy). For convenience, cubature stands for interpolatory cubature in this thesis, even if the dimension is one in some situations.
PARINT will evaluate the integrand function a number of times to compute the result. The number of function evaluations is used as a measure of the amount of effort or time spent in the computation. The user can set an upper limit on it to ensure that the computation will stop. It is quite possible that an integration is not able to be achieved within the given error tolerances due to the nature of the integrand function, the accuracy requirement, the computation time, the effect of possible round-off error in the computation, and the limits on machine precision. If the function evaluation limit is reached, the required accuracy has probably not been achieved. Many other algorithm parameters can be specified by the user for optimization and speedup of the computation.

2.2 Web-based Computation

A Web browser is often the default graphical user interface for distributed computing. The so-called Problem Solving Environment (PSE) [87, 71, 64, 28, 42] provides a complete environment for solving large computationally intensive problems. Many of them provide Web-based interfaces for end-users to solve their specific problems. We implemented a client/server system for PARINT with a Java Applet as the client. Unfortunately, we found that the Java Applet had its drawbacks for the implementation of a complex user interface.

A Web-based computational system is usually a three-tiered or n-tiered system including the browser, middleware, and other components such as databases and computational engines.

The middleware layer is typically coded in PHP (a widely-used Open Source general-purpose scripting language that is especially suited for Web development) or JSP (JavaServer
Pages) with database support for handling the interactions between the browser and other components. The commercial ASP of Microsoft rarely appears in research work.

2.3 Grid Computing and Web Services Technologies

Grid computing is a world-wide effort to develop a distributed environment for users to access computers, data, and experimental facilities across administrative domains to solve massive computational problems. Service-based Grid computations allow users to utilize a grid computing service as a component of a larger application architecture. The developer of the large application may utilize services to accomplish sub-tasks in much the same way the developer of a traditional application executes function calls.

Grid computing applications are based on distributed computing technologies. RPC (Remote Procedure Call), CORBA (Common Object Request Broker Architecture) [80, 34], Java RMI (Java Remote Method Invocation) [75] are some important distributed computing technologies. RPC is not object-oriented. The main noticeable difference between CORBA and RMI is that CORBA is language-independent whereas RMI is only for Java. From a technical point of view, RMI is the best mechanism for distributed computing at present. In practice, we have to consider interoperability with respect to systems and languages. Thus Web services are becoming increasingly popular for both academic and business applications.

The Web Services Architecture Working Group [89] defines a Web service as, a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically, WSDL).
Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. Web services are based on industry-standard protocols [89] (WSDL, SOAP, UDDI, HTTP) and are defined in eXtensible Markup Language (XML) to ensure their independence of platforms and programming languages. With the support of major IT companies, Web services technologies are becoming the backbone for exchanging business data.

The Grid computing community has extended Web services and proposed Grid services. The Open Grid Services Architecture (OGSA) [35] defines calling conventions and WSDL interfaces for Grid services, which can contain both the traditional stateless Web services and stateful services.

The Globus Alliance [79] distributed the Globus Toolkit for implementing Grid applications. The toolkit uses Axis [78] and gSoap [84] as its Java and C++ SOAP implementations, respectively.

2.4 Multidimensional Function Visualization

Function visualization typically consists of the visualization of relations and the visualization of mathematical formulas. Star Coordinates [41], Grand Tour [3], and Parallel Coordinates [37, 36] are some of the methods designed for the former. We mainly focus on mathematical functions, although some relational discrete data can be interpolated for sampling at intended positions.

The concept of Worlds within Worlds was proposed by Feiner and Beshers [24]. Up
to three variables at each level consist of a world. Worlds are nested together to generate a hierarchical view of a multi-dimensional function. An outer world consists only of axes for locating the origin of its inner world. The weakness of the method is the lack of an overview of the function.

Mihalisin et al. generalized the two-dimensional heat map to four or more dimensions. The scheme resembles hierarchical axes [56], where the inner axes correspond to the fast running variables. The method uses the space of the screen effectively, as shown by a four-dimensional example with 194,481 points.

Instead of displaying all slices, van Wijk and van Liere proposed HyperSlice [85], which is similar to the scatterplot matrix [9]. For a focus \( c = (c_1, c_2, \ldots, c_n) \), a two-dimensional slice \( S^j_i \), with \( i \neq j \), visualizes \( f(x) \), where \( x_i \) and \( x_j \) vary and the other components equal the corresponding components of \( c \). This method cannot provide a function overview. Navigation ability is supported to change the focal point \( c \). Santos and Brodlie extended the idea of HyperSlice to three-dimensional visualization, and called the method HyperCell [22]. Another similar technique is HyperBox [2], which uses \( n(n-1)/2 \) faces to depict an \( n \)-dimensional function.

PolarEyez [38], by Jayaraman and North, maps a pyramid, whose base is a face of a cube and whose pinnacle is a specified focal point, to a triangular pie slice of a two-dimensional polygon. This visualization provides an integrated overview of the function, while giving the points around the focal point more screen space than the distant points. Compared to the slicing methods, it is not easy to reconstruct the function from a PolarEyez visualization.
**PARVIS** [44, 21] is a visualization tool for distributed adaptive partitioning algorithms. It supports the analysis of load balancing techniques, error patterns, and parallel work anomaly. The leaves of region trees contain region data, which are placed to fit the available display area. **PARVIS** is also used to analyze numerical integration, but its visualization method is significantly different from that of AdaptView described in Chapter 5.

### 2.5 Iterated Methods

According to Fubini’s theorem, if

$$\int_{A \times B} |f(x, y)| \, d(x, y) < \infty,$$

(2.3)

where the integral is taken with respect to a product measure on the space over $A \times B$, then

$$\int_A (\int_B f(x, y) \, dy) \, dx = \int_B (\int_A f(x, y) \, dx) \, dy = \int_{A \times B} f(x, y) \, d(x, y),$$

(2.4)

where the first two integrals are called iterated integrals, and the third is an integral with respect to the product measure.

Analytically, the inner integral is performed first and the outer integration variable is treated as a constant. Numerically, when $f(x, y)$ is evaluated, the values of $x$ and $y$ must be known. Unlike the former, numerical iterated integration starts logically from the outer integral, although evaluation is also done first in the inner integral. We will call an integration method *iterated*, if lower-dimensional methods are used for the integration in different coordinate directions [50, 83, 40].

NAG Fortran routine D01FBF computes an estimate of a $n$-dimensional integral ($1 \leq n \leq 20$), given suitable Gaussian weights and abscissae [57]. It uses $n$-dimensional product
rules [13]. D01FBF is not an iterated routine. The iterated method discussed in Chapter 6 will degenerate to a product rule when the number of evaluations on each direction are limited to the number of points of the rule used for that direction.

D01DAF [57] moves a step further and attempts to evaluate a double integral by repeated applications of the method described by Patterson [45]. The method uses a family of interlacing common point formulae. Rules using 7, 15, 31, 63, 127 and finally 255 points are derived from the 3-point Gauss rule at the beginning of the sequence.

D01DAF, as well as D01FBF, is not well-suited to integrands with some type of non-smooth behavior in, on the boundary of, or near, the integration region.

Iterated integration methods were implemented in FORTRAN for the computations of Feynman loop integrals in [17, 18]. Two-dimensional iterated numerical integration is also explained in [40] and [61].

We use quadrature rules from QuadPack [61] to compute one-dimensional integrals. The $n$-dimensional iterated method was implemented in C, and can be used with a single function call. It can be applied to a broader set of problems, including classes with non-smooth integrands.
CHAPTER 3

Parallel Integration Online

This chapter describes the design and implementation of a problem solving environment based on PARINT and Web technologies. Online computational systems are not uncommon. Our system differs from the others in its automatic parallelization of the user specified problem for high performance computation (HPC). Furthermore, it provides the user with an interface as simple as online shopping. We call it Pion (Parallel Integration ONline). No knowledge of parallel programming or operating system details is needed for the users.

3.1 Introduction to Problem Solving Environment

The solution to bridge the gap between the user and the HPC system is the Problem Solving Environment (PSE) [87, 71, 64, 28]. The PSE hides the complexity of hardware and software, so that the user can focus on the problem. It provides a complete environment for solving large computationally intensive problems for both end-users and experts.

The characteristics of a PSE [49] are:

**Problem domain.** PSEs offer an integrated environment for solving specialized problems, including means for composing, compiling, and running applications, while handling security issues.

**Parallel resources.** PSEs provide easy access to distributed computing resources, and state of the art problem solving power to the enduser.
Black box configuration. PSEs attempt to remove hardware and software complexity, in order to eliminate the need for the end-user to be an expert in the solution method used.

Grid distribution. PSEs can be implemented on top of grid services.

Visualization. PSEs need to address the visualization of computations for steering in order to allow a user to explore a problem interactively, as well as the analysis and interpretation of resulting data.

3.1.1 Previous Work

Since the 1990’s, with the advance of computer hardware and software, many computational problem-oriented systems have been developed. NETCARE [42], powered by PUNCH [43, 42] and Condor [77], is a Web-accessible distributed infrastructure of tools and computing resources for the computer architecture community. Unlike Pion, NETCARE is not intended for computationally intensive problems. Cactus [1] provides a unified modular and parallel computational framework for engineers and physicists. Article [46] presents a model for a World Wide Web computational server, which uses Maple as its underlying engine. The more specific PSEs, WBCSim [63], VizCraft [63], PDELab [64] and Ecce [67] are for wood science, aircraft design, solving partial differential equations, and computational chemistry, respectively.

3.1.2 Options for Online Integration

Our environment should be Web-based to meet the requirement of being user-friendly. To develop, deploy, and maintain a conventional HPC program with a graphical user interface would be more time consuming. Furthermore, we cannot reuse current PSEs such as
NETCARE, because some parameters of PARINT are interdependent, and PARINT needs to compile C functions provided by users. In addition, the computation may take a long time. These require Pion to be a dedicated online system. We do not want to use Java Applet or Java Web Start [74], because these technologies do not meet our HPC requirement and PARINT is coded in C. The P2P scheme seems to be promising. However, we have to admit that numerical integration is not something that will attract a large number of common Web surfers. Web services or Grid services technologies are supported by many parties. Parallel integration (PI) service will be explored in the following chapter.

3.2 Online Integration

The system (Figure 3.1) is a typical three-tiered architecture with database support. PHP is used for the Web application scripting language. The middle layer provides user management, integrand function management, job management, and result management. The user interface is a Web browser that supports JavaScript and DOM (Document Object Model) [86]. A MySQL database keeps all user data. PARINT Plug-in Compiler (PPC) [90] and GCC compile the user function and integration parameters to an object file, which is then loaded dynamically to execute. The integration engine PARINT runs on a cluster of machines to perform the numerical computation.

PARINT can be run by a user locally, which is the traditional way of using PARINT. PI client and PI server are used for integration service as discussed in the following chapter.
3.2.1 User Management

For Web applications, we usually need to keep user data for various reasons. The simplest way is to make a directory for each user. It has the advantages of less access time and easy deployment. This method is suitable in small applications. MetaΨ [4] uses LDAP (Lightweight Directory Access Protocol) and a Web server as the middleware to locate computing resources and to keep user profiles. For Pion, we found that using a database is a better choice. In addition to the normal fields such as the user name and the email address, the user table has two additional fields, $maxRunTime$, and $maxJobs$. The user’s computing resource access levels are defined by the two values. Depending on the workload trends of the system, more fields may be added to classify the users. A Web-based computational
system must have a way to control the computing resources. This issue will be discussed in the job control section below.

3.2.2 Function Management

There are two stages in setting up the numerical integration problem, namely defining the problem and specifying the algorithm parameters. On the “New Function” page as shown in Figure 3.2, the user needs to enter the dimension of the integrand function, the
absolute and relative error tolerances, the type of integration region, the region itself and the integrand function. There is an option for uploading the function definition from the local disk. When the function is submitted to the server, all the data will be saved to a function table in the database. The function is then compiled to generate a .ppl file, which will later be linked dynamically with the PARINT executable. If there are errors, another page will show them with highlighted line numbers. The user can then correct the errors and re-submit the data. For a multivariate integral of dimension $n$, $2n$ and $n(n + 1)$ text boxes are required for specifying hyper-rectangular and simplex integration regions, respectively. Sometimes, the user may want to change the dimension or switch from one region type to another. We use JavaScript to manipulate the DOM [86] object of the Web page to solve this problem. When the change of dimension or region type is confirmed by another event, the JavaScript code will compute the new numbers of rows and columns and refresh the text box table.

DOM [86] is a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents. It can be accessed via any language available in the browser, including Java, JavaScript, and VBScript (for Internet Explorer only). DOM is supported completely starting with Internet Explorer 5 and Netscape 6. The dynamic characteristics of DOM are useful to handle parameters depending on each other.

All functions owned by a user are listed in a table. The user can select a function to edit, clone or delete. Depending on the region type, the algorithms available to solve the integration problem are shown. For problems with simplex regions, we only provide
an adaptive cubature method. For those with hyper-rectangular integration regions, we provide adaptive, Quasi-Monte Carlo, and Monte Carlo methods.

For low-dimensional problems, the adaptive algorithm is preferred, unless the integrand behavior is erratic. For erratic integrands and/or high dimensions, Monte Carlo integration is advised. Quasi-Monte Carlo integration is justified for some high-dimensional problems, under certain smoothness conditions on the integrands. The user needs to specify a number of processors, a function evaluation limit, a timeout limit, and an integration rule. Optional algorithm parameters are intended for users who have some knowledge of the algorithm implementation. Advanced parameters can be specified by clicking the “Advanced Options” button. These are intended for experts only.

### 3.2.3 PPC Compiler

The prototype of an integrand function in the C language is

```c
int function_name(int* ndims, pi_base_t* x, int* nfcns,
                    pi_base_t* funvls),
```

where `pi_base_t` can be `double` or `long double`, `ndims` is the integrand dimension, `x` contains the coordinates of the evaluation point, `nfcns` is the number of component functions in a vector function, and `funvls` contains the function value(s). Normally, the function returns 0.

Before the function is compiled, it is combined with the integration parameters to generate an intermediate file. The intermediate file is compiled by a pre-processor called PPC (PARINT Plug-in Compiler) to generate a temporary C source file [90]. This temporary file is then passed to the C compiler to be compiled into a .ppl file. The C compiler reports
errors in terms of the intermediate file, not the function that the user entered in the function body box. The middleware of Pion adjusts the error line numbers to reflect problems with respect to the original file, and highlights them for the end-user. The user will be informed of a successful compilation with a short message.

Allowing the user to write a function and run it on the system is potentially a security hole. Therefore some system calls are not allowed in the .ppl file. They are: accept(), bind(), fopen(), msgmsg(), msgget(), open(), pause(), poll(), putmsg(), select(), semop(), wait(), alarm(), brk(), chdir(), dclose(), dlerror(), dlopen(), dlclose(), exec(), fork(), popen(), pthread_create(), sbrk(), hostent(), setgid(), setuid(), signal(), system(), thr_create(), and umask(). The restriction with respect to certain function calls is important in an environment where the user’s real identity may be unknown.

PPC is also used in PI service described in Chapter 4, where the integrand and parameters are passed to the server via a remote function call.

3.2.4 Job Control

Load balancing is always an issue in distributed computing. When a user submits a job, the middleware will call the MPI [55] mpirun command, which typically uses the first n (a user specified number) processors defined in a machine file to execute the job in parallel. In a multi-user environment, it likely that the first several processors are overloaded, while the last ones are idle. Pion has a module to check the workloads of all machines and select suitable processors for the mpirun command. The “System Info” pages show the status of the system in detail for both users and administrators.
A globally accessible computational system should have a mechanism to optimize system utilization. Unlike an online shopping application where the transaction time range is well known, the time for a computational job is unpredictable in the situation at hand. It can be either a millisecond or a year, depending on the size of the problem. A user might submit a job that runs forever and never returns any result. As mentioned before, there is a field in the user table to specify the maximum execution time \((\text{maxRunTime})\) for the user job. The value can be increased by the system administrator upon request. Another value is the maximum number of jobs \((\text{maxJobs})\) that a user can start concurrently in the system. The value is 1 by default, but can also be increased upon request. Our policy is to allocate just enough resources for a user. The user interface is designed to encourage the users to manage their jobs interactively. The same logic is applied to the input box for the number of processors. Its default value is always 1, while most other boxes display the last entered values.

Killing a job is not an easy task in a Web-based computing system. The designer must answer questions like: who can kill the job (the user, the system, or both); how the job is killed; when the job should be killed; what action should be taken if the user forgets to kill it; how the user is informed if the system has killed a job.

A user can refer to a status table on the Web page, which shows the numbers of running jobs, done jobs, and stopped jobs (killed by the user or the system). Let us first cover how a user executes a job, then discuss the implementation behind the scene. After entering the integrand function and its problem parameters, the user: 1) specifies execution parameters including the number of processors and a timeout limit (should be less than \(\text{maxRunTime}\));
2) gets the result back if the execution time is less than about one second; or 3) waits for
the result while the screen is being refreshed; 4) increases the timeout limit (but not greater
than \textit{maxRunTime}) if necessary; 5) aborts the job execution; or 6) closes the browser and
comes back later to check the job status; 7) saves or discards the result.

The database has a transaction table with fields including: \textit{startTime}, \textit{timeout}, \textit{endTime},
and \textit{isChecked}. When the user submits a job, the timeout value and a timestamp
are stored to the \textit{timeout} field and \textit{startTime} field, respectively. The system checks the
transaction table periodically and kills all jobs that are timed out. After a job has stopped
running, a timestamp is written to the \textit{endTime} field. After the user has viewed the result,
the \textit{isChecked} field is changed to true, indicating that this transactional record will not be
manipulated again unless for statistical purposes.

The implementation of job management depends greatly on the scripting language of
the middleware. Special techniques may be needed to keep track of the PIDs and the output.

\section*{3.2.5 Result Data Management}

Result management is simple but critical to problem solving environments on high
performance computer systems. Simple Web applications rarely save the results for the
user, because it is easy to repeat the computations. Pion, however, keeps the results for the
user even if the user closes the browser and ends the session. After the output has been
generated, it is saved to a file.

Unlike other PSEs, Pion does not allow the users to access the file systems of the server
directly, nor does it provide a virtual hierarchical file system. From the user’s point of view,
file manipulation is unnecessary. The results of an integration problem are associated with the problem and can be accessed via the links in a table.

### 3.3 Performance Evaluation

Pion was designed and implemented to be a globally accessible Web-based problem solving environment for high performance computation. Much attention has been paid to the security and load balancing of the system. It can be accessed at the following URL, http://aegis.cs.wmich.edu/pion/. Our Beowulf type Linux cluster consists of 64 nodes, including 32 AMD Thunderbird nodes at 1.2 GHz and 32 AMD Athlon nodes at 800MHz. They are connected by both Fast Ethernet and Myrinet. For those users who want to run PARINT on their own hardware resources, the PARINT package is available for download. It can be compiled to parallel executables for a cluster, or sequential executables for a standalone Unix/Linux machine.

The administrative overhead of Pion is insignificant. The compilation time of an integrand function and the functions it calls depends on the size of the source code. The overhead of the Pion middleware for a job execution is about 0.1 second plus the extra waiting time described below. When a job is submitted, the middleware will wait 0.3 seconds for the result. If the result is produced within 0.3 seconds, it is returned to the user. If the execution time is longer, a window confirms that the job is being executed. The page is reflashed to pull the result in an interval of two seconds at first. The interval is incremented by 2 seconds for subsequent reflashes until it reaches 10 seconds.

Myrinet is the default device. The system administrator can switch it to Ethernet if
Figure 3.3  (a) Speedup trial on a smaller scale problem. (b) Speedup graph when using Ethernet only. (c) A large number of processors can be used efficiently for the larger problem.
needed. If there are not enough working Myrinet nodes available, Ethernet is selected automatically for a job that requires a larger number of processors.

Before submitting a large job, the user can use the speedup trial option to estimate a reasonable number of processors by running a smaller scale problem, for example, by specifying a less strict error tolerance.

As a sample application, we computed a four-dimensional cross-section integral [59]. For a trial run, we specified a relative error tolerance of 0.25 and a function evaluation limit of 100,000,000 and obtained the speedup graph shown in Figure 3.3 (a). For a number of processors not exceeding 25, Myrinet was used, otherwise Ethernet was used. When we disabled Myrinet and changed the relative error tolerance to 0.01, we generated a speedup graph Figure 3.3 (b). Note that the CPU speed of the upper nodes is faster than that of the lower ones. It took a lower node 1,063 seconds to perform 58,499,155 function evaluations to get a result of 0.0286 for the relative error tolerance of 0.25. When we reduced the relative error tolerance to 0.01, which caused the function evaluation limit to be reached, we obtained the graph shown in Figure 3.3 (c). From the trial run we expected that a relatively large number of processors (at least 20) can be used effectively for the larger scale problem.

The images of the graphs were generated dynamically with the PHP package JpGraph [60].

### 3.4 Conclusions

Pion facilitates, for both common end-users and experts, solving computationally intensive numerical integration problems in parallel. It accepts user code, takes care of security, parallelizes the problem automatically, and keeps the user data, including final results.
CHAPTER 4

Numerical Integration Service

The previous chapter introduced the user interface of online integration. This chapter describes a distributed numerical integration paradigm that provides remote users with an application programming interface (API). Traditionally, numerical integration is carried out sequentially or in parallel on a limited number of machines. XML, Web services, Grid services and other new technologies enable the design and implementation of massive scale integration architectures, which utilize world wide resources independent of platforms and programming languages. We propose an XML data format for the exchange of integration related data among different computing units. This format will be extended in Chapter 7 to support new parameters of new methods. We can design new algorithms on the client side to invoke the Monte Carlo (MC), Quasi-Monte Carlo (QMC) and/or adaptive cubature programs supported by the services. For example, the client pre-splits the regions adaptively and sends them to the servers, where they may be processed with an MC method. Experimental results showed that this distributed approach is an effective way for computationally intensive numerical integration problems.

4.1 Platform Independent Integration

Numerical integration is a basic technique employed to solve various problems. Methods, rules or algorithms are applied for general or specific problems. On the other hand, parallel processing architectures are constructed to utilize more computing power for an
individual problem. However, some problems, such as computing a physics constant, are so critical that no practical upper limit of computing power can satisfy their requirements. Some other problems require the computation of a large number of integrals. The current technologies make it possible to use all available computing resources for a single problem, although our target is not limited to such hard problems.

Many integration packages have been implemented based on different algorithms and computer architectures. Some are sequential [61, 11, 29, 66] programs for standalone computers; the others are parallel [76, 81, 68] programs that are executed by multiple processing elements such as those of a cluster. The next major step is to increase the parallelism to its possible limit.

Web services technologies are becoming popular in both commercial and academic environments. They fit the needs of massive scale numerical integration. A service can be implemented neutrally on multiple operating systems and in multiple programming languages. Assuming that we know a thousand end points of a service for integration, we can divide an integration region into a thousand sub-regions and send them to the end points to compute the integral.

In addition, we are developing our computational paradigm for integration algorithms in a neutral manner. Many integration packages implementing MC, QMC, and adaptive cubature methods share a somewhat common interface. The user is required to specify the integration problem, including an integrand function, a region, an error tolerance and a function evaluation limit (or time) to calculate the integral within the error tolerance. When these packages are adapted to provide a Web service, the developer and the user are free
to implement their programs that send requests to the service. In other words, they can implement new algorithms on top of the service algorithms. As described later, one can write a client program that handles the integration region adaptively based on the results from a Web service powered by a Monte Carlo method program. In its simplest scenario, the end-user enters the data of a problem and submits the job via a client program to a broker program that utilizes the worldwide resources to find an answer.

4.2 Format of Integration Parameters

XML [89] is suitable for the exchange of structured data. Multivariate integration programs such as PARINT [15] accept a long list of parameters especially for high-dimensional integrands. A plain text input file may solve this problem. However, when we consider new algorithms or new computational models as described in this thesis, we realize that an XML style format is necessary. If we agree on a standard data format, programs will be able to handle the algorithm parameters that they support, and deliver results that can be understood by others. The following is a variant of the XML parameter file used in our experimental implementation. The tag names, or even the whole structure need to be standardized. A more complete XML parameter file can be found in Appendix B.

```xml
<integration>
  <problem>
    <name>myFunction</name>
    <dimension>2</dimension>
    <absoluteError>1E-06</absoluteError>
    <relativeError>1E-06</relativeError>
    <region default="rectangular">
      <rectangular>
```
The problem section defines the integration problem requirements. We recommend that all programs support the hyper-rectangular region type, although they do not have to support multiple hyper-rectangular regions as specified in the input file. The integrand function needs to be standardized in terms of languages, arguments and return types. The algorithm parameters may differ from program to program. The extended parameters appear at the end of the file. Most elements are optional. A program can just pick up what it supports.
Note that the result will also be represented in an XML format. Because the output styles of different programs vary significantly, more work is needed to standardize it.

### 4.3 Integration Service

The architecture consists of provider agents (servers or providers for simplicity), requester agents (requesters, users or clients) and brokers. The brokers are both requesters to the providers and providers to the requesters. Depending on the requirements, the requester may send requests to any number of servers and brokers. In Figure 3.1, PI Client is the requester. PI Server provides integration service. It can also be a broker by sending requests to other PI servers.

An integration service takes the integrand function and the parameters from the requester and returns the results to it. To support a Web service for integration, typically, we need at least a Web server, a package implementing SOAP (Simple Object Access Protocol) [89], a middleware application and an integration program. SOAP is a simple XML based protocol to let applications exchange structured and typed data in a distributed environment usually over HTTP. Databases and other components may be required for building an integration service.

Most of the existing numerical integration packages are implemented in Fortran or C/C++. Unless the integrand function is a pre-defined function in the package, its code must be compiled and linked statically or dynamically with other modules. Considering flexibility, modularity, scheduling, accounting and parallel computing, we suggest that the code which performs integration be implemented separately, instead of as a part of the mid-
dleware application. The middleware application is responsible for calling a compiler to compile the integrand function, starting the integration program, terminating it if necessary, etc.

We found that passing parameters as command-line arguments to the integration executable is tedious. Some programs write regions together with their intermediate results to a file and read them back to process again. In many cases, XML offers a convenient way of data representation. We wrote a module to process parameters from an XML file for PARINT. Nevertheless, there is no unanimous agreement on whether this is necessary for all integration programs.

The flexibility of the requester enables a wide spectrum of its implementations. We implemented a multi-threaded requester program in Java. When we want to connect to more than one end point simultaneously, the requester starts multiple threads and handles the results from all of the connections. We should not expect all end-users to write multi-threaded programs or even to know the details of Web services. With the client package, they only need to fill out an XML template file with their integrand and parameters, and start the program to get the result.

The broker takes requests from the requester and prepares it to be processed further. Centralized management has many advantages. First, it simplifies the requester implementation (user client). Algorithms can be implemented and updated without involving any effort of the end-users. Secondly, it simplifies the provider implementation by taking care of accounting, load balancing, etc. The computational logic in the broker is similar to that of the requester, as be discussed in the following section.
4.4 Client-side Adaptive Algorithms

The adaptive code of PARINT divides a region into two sub-regions adaptively and uses cubature rules to compute the integral and the estimated error. For some integrand functions with singularities, this method fails to hit the singularity. A pre-splitting approach, in which the region is divided into a number of sub-regions, can improve the accuracy. But this may involve a lot of unnecessary function evaluations. We applied a similar method to the requester implementation. A task, which is represented by a node of a heap, keeps the data of a region, the error tolerances, and the result data of the region. The nodes of the heap are extracted in descending order of the estimated error of the result. The asynchronous algorithm is described in brief as follows:

1. Let numberOfTasks equal 0; divide the integration region into \( N_0 \) sub-regions; divide the absolute error tolerance accordingly; insert the corresponding task data of the sub-regions into a queue \( Q_0 \).
2. Take a task from \( Q_0 \) if it is not empty; start a thread to send the task to a free service provider; increment numberOfTasks.
3. Notify the main thread after a task is done. If the result fails to meet the error requirement, divide the task into \( N_i \) sub-tasks, where \( N_i \) does not have to equal \( N_0 \). Insert them into the heap \( Q_1 \).
4. Extract the task with the largest estimated error from \( Q_1 \) and send it to a free service provider; increment numberOfTasks.
5. If numberOfTasks equals MAXTASKS, stop sending out new tasks. If both \( Q_0 \) and \( Q_1 \) are empty and no thread is waiting for its result, print the final results and stop. Otherwise,
6. go to 2.

We have not mentioned the algorithm used by the integration program. If it is an MC, a QMC or an adaptive algorithm, the overall algorithm will be an MC/adaptive, a
QMC/adaptive or an adaptive/adaptive algorithm, respectively. We know that adaptive cubature methods are good for low dimensions, while MC and QMC methods are suitable for higher dimensions. We hope that the new combined algorithms will be useful where adaptive, MC and QMC methods are weak alone.

4.5 Performance Evaluation

We used PARINT [90] with adaptive cubature, MC and QMC methods as the integration package. The Web server and Servlet container is Apache Tomcat. The SOAP implementation is Apache Axis [78]. A Sun Sunfire V880 and two clusters, Athena and Tango, supported the simulation.

Overheads were estimated using a 2.2 GHz Pentium client and a 1.5 GHz AMD server. It took 0.35 seconds of turnaround time to invoke the method Call.invoke() without starting the integration program. It took 1.2 seconds to return an empty string. Including the times for compiling the integrand and starting the executable with one processor, the overhead amounted to 1.4 seconds. It increased to 2.4 seconds when we used eight processors.

Considering the wide area network latency, we can say that an overhead of 1 to 3 seconds is added to a transaction. There are ways to optimize these times. It is reported that another SOAP implementation gSOAP is faster than Axis [8]. If a numerical integration must be done in less than a second, the Web services approach fails to satisfy it.

We used the following integral to evaluate the adaptive/adaptive algorithm described above.

\[
I = \int_{-1}^{1} \int_{-1}^{1} \int_{-1}^{1} dx_1 dx_2 dx_3 \frac{\varepsilon \sqrt{x_1^2 + x_2^2 + x_3^2}}{(x_1 + x_2 + x_3 - a^2)^2 + \varepsilon^2},
\]  
(4.1)
where \( \theta(t) = 1 \) for \( t \geq 0 \), and 0 otherwise. For \( a = 0.64 \), we tried \( \varepsilon = 0.1, 0.01, 0.001, \) and 0.0001. The integrand has a peaked behavior at the spherical surface with radius \( a \). As the value of \( \varepsilon \) decreases, the ridge will become very sharp. The integral, which we refer to as DICE3, is a good example for testing new algorithms [81].

Monte Carlo methods have difficulties to achieve high accuracy in view of their slow convergence. Adaptive cubature methods do not work well in high dimensions. Furthermore, adaptive cubature methods may fail to hit the singularity of a function adequately. Typically, in an adaptive method for the hyper-rectangle, the number of sub-regions may be 2 or \( 2^n \), where \( n \) is the dimension of the integrand function. In our algorithm described above, users can customize the way that region is divided. When \( N \) is small, the algorithm behaves like other adaptive algorithms. When \( N \) is large enough, the scheme has the ability to handle singularities reasonably.

<table>
<thead>
<tr>
<th>( \varepsilon )</th>
<th>Abs. Error</th>
<th>Numerical</th>
<th>Analytic</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 10^{-1} )</td>
<td>( 10^{-3} )</td>
<td>10.57452±5.9E-4</td>
<td>10.57453405</td>
</tr>
<tr>
<td>( 10^{-2} )</td>
<td>( 10^{-4} )</td>
<td>12.42196±4.2E-4</td>
<td>12.42245990</td>
</tr>
<tr>
<td>( 10^{-3} )</td>
<td>( 10^{-5} )</td>
<td>12.61202±4.9E-4</td>
<td>12.61202527</td>
</tr>
<tr>
<td>( 10^{-4} )</td>
<td>( 10^{-5} )</td>
<td>12.630986±3.1E-5</td>
<td>12.63098679</td>
</tr>
</tbody>
</table>

Table 4.1 Numerical results for (4.1). The function evaluation limits of each sub-region are \( 10^9 \) for \( \varepsilon = 10^{-1} \) and \( 10^8 \) for the other \( \varepsilon \).

Table 4.5 lists the analytic and numerical results for (4.1). For all integrals, the integration region was initially divided into 4,096 sub-regions. If the integral of a region failed to meet the absolute error tolerance requirement, it would be divided into 16 sub-regions. For \( \varepsilon = 0.1, 0.01, \) and 0.001, all 4,096 tasks were computed successfully, and no further sub-
division is needed. Maximum parallelism can be achieved if 4,096 processing elements are available. The integral can be computed in about one second. For $\varepsilon = 0.0001$, we can see the adaptive effect; 1,122 regions are further subdivided and processed.

Note that the adaptive method is not the best method for this problem. It will be discussed again in the following chapters.

4.6 Summary

We described a new paradigm for massive scale numerical integration and provided an initial implementation. The architecture is most suitable for problems that require intensive computation. Latency depends mainly on the network bandwidth and the middleware implementation. This architecture may not be suitable for problems that need to be solved in less than a second. Because of its flexibility, new integration algorithms can be implemented easily in both the client and the server sides. The data format and the details of the service remote functions should be standardized.
CHAPTER 5

Adaptive Multidimensional Function Visualization

In this chapter, we introduce a new function visualization method and demonstrate that numerical integration and visualization of multi-dimensional functions are closely related. Adaptive numerical integration is utilized to reduce the number of function evaluations, and generate time series data. A grid cell can be sampled many times, or is not sampled at all, depending on the function properties and the integration rule. Function properties are extracted during the process of function evaluation. An aging technique helps visualize functions by retaining the most recently sampled areas and making the older ones transparent. This also results in giving the non-smooth areas more attention than the smooth areas. We efficiently and accurately visualize two-dimensional functions, directly visualize three-dimensional functions, and effectively reduce the dimensionality of higher-dimensional functions in visualization. The new function visualization method gives a view of the whole function while elaborating on important areas such as ridges and troughs, which are critical in many fields, including numerical integration.

As discussed in the following chapter, the knowledge of the shape or behavior of a function can help select the methods for integration and load balancing before a time-consuming simulation is carried out.
5.1 Overview

Function visualization is important in many fields of science and engineering. Considerable progress has been made recently [88, 36, 54, 22, 38, 44]. A background of function visualization has been introduced in Chapter 2.

It is physically difficult to visualize a non-discrete function of dimension three or higher. Two-dimensional functions are usually displayed with heat (or color) maps, contour plots, or three-dimensional mountain plots. However, a class of important functions in physics and other areas needs special treatment. These functions have narrow peaks or singularities. Traditional uniform sampling may fail to depict those areas accurately. For an $n$-dimensional function, if we sample $m$ points in every direction, there are $m^n$ points in total, which easily outnumber the pixels on the screen for moderate values of $m$ and $n$.

A common practice is slicing which shows only part of the function domain, dimension stacking which emphasizes a subset of the axes, or mapping which projects the function onto a two-dimensional plane.

Both numerical integration and function visualization require function evaluation. For adaptive numerical integration, some areas of the domain $\mathcal{D}$ are more important in error reduction and therefore need more sampling. Coincidentally, the areas are often deemed to contain more information of interest of the function. Our tool or method, which utilizes numerical integration techniques in function visualization, is named AdaptView. It has the following functionalities:

- Tracking non-smooth regions of a function efficiently.
• Visualizing three-dimensional functions effectively.

• Exploring high-dimensional \( (n \geq 3) \) functions by aggregation or projection.

• Separating of evaluation and visualization. The data volume can be reduced significantly, which is critical for distributed data visualization.

• Helping with the analysis of integration problems.

There are many reasons why we need to visualize a function for numerical integration. First, AdaptView can locate troughs and ridges that have not been evaluated appropriately by an integration method. When an integration method is employed to solve a problem to certain accuracy, it may fail to completely evaluate a narrow sharp region, and give a result that is unreliable. In that case, we can use AdaptView to locate the region to help analyze the result. We demonstrated this type of situation in previous work [50]. Furthermore, the intermediate and final images provide useful information for choosing an integration method. In addition, the near real-time computation visualization ability of AdaptView provides the user with the current results and errors of each grid cell, and shows the hot spots of the computation.

In this work, we focus on the application of AdaptView to numerical integration. However, the visualization technique can be used for general function visualization. We use \textsc{ParInt} [76, 90] as the underlying numerical integration package, although it can be replaced by other similar packages with a little modification.
5.2 Adaptive Visualization

Visualizing a function adaptively often reduces the number of function evaluations significantly, while the details of the function are kept intact. The technique is based on earlier work by K. Kaugars.

Because both function evaluation and visualization can be computationally intensive, they are separated and done in parallel. An \( n \)-dimensional space is partitioned into an \( m_1 \times m_2 \times \ldots \times m_n \) uniform grid, which is used for data reduction and communications. In this chapter, we use the term \textit{region} for an adaptively subdivided space, \textit{area} for a specific part of the space, and \textit{cell} for a visualization grid cell. Note that an integration region may be contained in a cell, coincide with a cell, or contain one or more cells. The integration engine may perform many function evaluations in a cell. This is necessary in a non-smooth area to achieve an accurate local integration over a cell, or to locate the maximum (or minimum) function value inside a cell. Conversely, the integration engine may not perform any evaluation in a cell at all if the neighborhood of the cell is smooth. In this case, the cell values can be inferred from those of the neighboring cells for the purpose of visualization.

The user can choose the set of data to be displayed. For our application, a cell contains either \((\text{result, error})\) or \((\text{np, min, max, sum})\), where \text{result} is the integral approximation, \text{error} is the estimated absolute error, \text{np} is the number of function evaluations, \text{min} is the minimum, \text{max} is the maximum, and \text{sum} is the sum of all evaluations in a cell. If the cells contain \((\text{np, min, max, sum})\), we say that AdaptView runs in mode \text{max/np}. In this mode, when a point is evaluated, the corresponding cell will be updated. If the cells contain
(result, error) instead, we say that AdaptView runs in mode result/error. One or more cells are updated when a region of the adaptive integration has been evaluated.

5.3 Two-dimensional Function Visualization

Our two-dimensional visualization outperforms the uniform sampling method for non-smooth functions. When used for monitoring integration, it tracks computational hot spots in near real-time animations of a computation.

5.3.1 Non-smooth Function View

A uniform sampling wireframe [9] can generally display a smooth two-dimensional functions as a surface in a three-dimensional visualization. It is often difficult, if not impossible, to draw a function with singularities or sharp peaks using wireframes. We refer to the integrand of

\[
I = \int_{-1}^{1} \int_{-1}^{1} dx_1 dx_2 \frac{\varepsilon x_2^2 \theta(1 - x_1^2 - x_2^2)}{(x_1^2 + x_2^2 - a^2)^2 + \varepsilon^2},
\]

(5.1)
as the DICE2 function, where \( \theta(x) = 0 \), if \( x < 0 \), and \( \theta(x) = 1 \), if \( x \geq 0 \). It has two ridges around the circle of radius \( a \), and a discontinuity at the circle of radius 1 [82]. The height of the ridges can be adjusted by changing the small positive number \( \varepsilon \).

Figures 5.1a-c were drawn using Maple [53] for \( \varepsilon = 0.01, 0.001, \) and \( 0.0001 \), respectively. Figures 5.1a and b were drawn on 128x128 grids. Figure 5.1c was drawn on a 512x512 grid.

In this work, the color scale shown in Figure 5.2 is used for all images of AdaptView. This scale ranges from green for the smallest value, to red for the largest value. Unless oth-
Figure 5.1  Uniform sampling wireframes with Maple. (a) $\varepsilon = 0.01$, on 128x128 grid. (b) $\varepsilon = 0.001$, on 128x128 grid. (c) $\varepsilon = 0.0001$, on 512x512 grid.
Figure 5.2  Color scale for all AdaptView images.

otherwise specified, for all mountain plots of AdaptView, max or result of a cell is represented by height, np or error is represented by color, and the grid size is 128x128.

Figure 5.3 was drawn with AdaptView in mode result/error on a 128x128 grid, for \( \varepsilon = 0.001 \). The number of function evaluations is 262,144. Figure 5.4 was drawn using AdaptView in mode max/np. In this instance, mode max/np gives a smoother surface than result/error. However, when the number of function evaluations is relatively small, the latter is smoother. Compared to Figures 5.1a and b, Figures 5.3 and 5.4 were drawn with more function evaluations.

We can also use the iterated method \([25, 83, 17]\) to compute the integral approximation and draw a graph of the integrand function. Compared to Figure 5.1c, Figure 5.5 depicts the same function much more accurately for the same value of \( \varepsilon \) and using the same number of function evaluations. Rectangular bars, instead of triangular strips, were used to render the surface.

5.3.2  Computational Hot Spots

AdaptView can be used to locate the hot spots of a computation. Knowing how the regions are subdivided is very important in the design and use of numerical integration algorithms. Usually, the regions with large results also have large error. On the contrary, when \( \varepsilon \) of (5.1) equals 0.1, significant errors occur around the outer circle, where the function values are relatively small, as shown in Figure 5.6.
Figure 5.3  Visualization of cell result of (5.1), where $\varepsilon = 0.001$.

Figure 5.4  Visualization of cell max of (5.1), where $\varepsilon = 0.001$. 

44
Figure 5.5  Visualization of cell \textit{max} of (5.1), where $\varepsilon = 0.0001$. Iterated method and rectangular bars were used.

Figure 5.6  Visualization of cell \textit{result} of (5.1), where $\varepsilon = 0.1$. The red cells have large errors, indicating that most of the computation was performed around the circle.
5.3.3 Near Real-time Animations

In an adaptive integration method, regions are typically subdivided until a termination condition is met. AdaptView can be used to visualize this process. One direct application of the near real-time animation with AdaptView is to monitor estimated errors, either through coloration by error value or direct mapping of error value to height as shown in Figure 5.7. See Appendix A for snapshots showing the process of adaptive region partitioning.

5.4 Three-dimensional Function Visualization

To display three-dimensional functions, AdaptView uses a grid of \( m_1 \times m_2 \times m_3 \) cells. The function value (or another metric) of a cell is represented by a color using the color scale shown in Figure 5.2. Initially, an age of zero is assigned to each cell. The age is incremented by one if AdaptView makes a function call to receive data but the data of the cell have not been updated. Otherwise, if the cell has been updated, its age will be reset to zero. The transparency of a cell is a function of its age, ranging from opaque for an age of
zero, to transparent for an age greater than a predefined limit.

The data update rate and the aging scheme control the percentage of grid cells displayed on the screen, because the transparent cells will not be drawn. Adaptive integration programs perform more computation in non-smooth regions, and hence if a function has regions with singularities, where the number of function evaluations is large, these regions can often be tracked.

Three functions are used as examples to show the effects. The function,

\[ f(x, y, z) = x^{-0.2}y^{-0.2}z^{-0.2}(x + y + z)^{-0.2}, \tag{5.2} \]

approaches infinity when one or more of its variables approaches zero. Figures 5.8, 5.9, and 5.10 illustrate the computation of this function in the domain of the unit cube. AdaptView ran in mode result/error.

The colors of Figures 5.8, 5.9, and 5.10 (as well as those of 5.11, 5.15, and 5.18) were rescaled by sorting the \( m \) (equal to \( m_1 \times m_2 \times m_3 \)) values and mapping them to the \( m \) uniformly-spaced points on the color scale. From the animations, we can infer that there is a peak at the origin; that points near \( x, y, \) and \( z \) axes have large values; and that the values of points near \( xy, yz, \) and \( zx \) planes are larger than those of points more distant from the planes.

The integrand of (4.1) is named the DICE3 [82]. Figure 5.11 shows two spheres containing computational hot areas for \( \varepsilon = 0.001 \). The inner sphere has a three dimensional peaked behavior. The outer sphere corresponds with the discontinuity of the \( \theta \) function.
Figure 5.8  Snapshot of computation for function (5.2) (I).

Figure 5.9  Snapshot of computation for function (5.2) (II).
Figure 5.10  Snapshot of computation for function (5.2) (III).

Figure 5.11  Dice 3D visualization for $\varepsilon = 0.001$. AdaptView was in mode result/error.
A four-point loop integrand function arising in a particle physics problem [17, 26], is

\[ f(x, y, z) = s_1 s_2 \frac{d^2 - \varepsilon^2}{(d^2 + \varepsilon^2)^2}, \]  

(5.3)

where,

\[ d = (x^2 + \chi^2) + t \zeta^2 - (t - t_m^2) \zeta \]
\[ - (e_m^2 - z_m^2)(x + \chi) + (2e_m^2 - spar)x\chi \]
\[ + (t + e_m^2 - t_m^2)(x + \chi)\zeta \]
\[ - (\zeta - 1 + \chi + x)e_m^2, \]
\[ \zeta = s_2 z, \]
\[ s_2 = 1 - (x + \chi), \]
\[ \chi = s_1 y, \]
\[ s_1 = 1 - x, \]
\[ spar = 0.25 \times 10^6, \]
\[ e_m^2 = 0.261 \times 10^{-6}, \]
\[ t_m^2 = 0.225 \times 10^5, \]
\[ z_m^2 = 0.828 \times 10^4, \]
\[ t = -0.102 \times 10^6, \]
\[ \varepsilon = 16. \]

The integration domain is the unit cube. We ran AdaptView in mode max/np to generate Figures 5.12, 5.13, and 5.14. The function values are mapped linearly to the color scale. One can easily identify a badly behaved surface.

Figure 5.15 displays a result/error visualization of function (5.3). The areas containing large boxes are relatively smooth. The areas with long bars are smooth along the z direction. Most importantly, the irregular behavior is concentrated in at least two surfaces, one with red colors for large values, and one with green colors for small values (negative here). Careful examination of this function with two variables set to constant revealed that there
Figure 5.12  Four point loop integral in mode $max/np$ (I).

Figure 5.13  Four point loop integral in mode $max/np$ (II).
Figure 5.14  Four point loop integral in mode $\max/np$ (III).

Figure 5.15  Four point loop integral in mode $\text{result/error}$. 
exist two peaks and a trough. Results for the integration are reported in [18] using an iterated integration method combined with an extrapolation technique. Up to 6- to 7-figure accuracy is achieved. For this type of problem, double or even 128-bit long double data types are preferred to obtain a good numerical integral approximation.

5.5 Dimension Reduction

Consider an \( n \)-dimensional integral represented as,

\[
I = \int_{D_1} \int_{D_2} d\mathbf{x}_1 d\mathbf{x}_2 f(\mathbf{x}_1, \mathbf{x}_2)
\]

(5.4)

where \( D_1 \) is the inner domain of dimension \( n_1 \) (can be 1, 2, or 3), and the outer domain \( D_2 \) is of dimension \( n_2 = n - n_1 \). Let,

\[
f(\mathbf{x}_1) = \int_{D_2} d\mathbf{x}_2 f(\mathbf{x}_1, \mathbf{x}_2).
\]

(5.5)

The integral (5.4) can then be written as,

\[
I = \int_{D_1} d\mathbf{x}_1 f(\mathbf{x}_1).
\]

(5.6)

When a point of the integrand of (5.6) is evaluated, the right hand side of formula (5.5) needs to be computed, which is equivalent to performing another numerical integration. The \( n_2 \) dimensions are projected onto the \( n_1 \) dimensional outer space, where the integration can be visualized using AdaptView. If we just consider the results and the maxima, there are three options for the projection, namely,

- summing data over the \( n_2 \) inner dimensions for the \( n_1 \)-dimensional visualization in mode result/error,
• summing data over the $n_2$ dimensions for the $n_1$-dimensional visualization in mode $max/np$, and

• selecting the integrand maxima over the $n_2$ dimensions for the $n_1$-dimensional visualization in mode $max/np$.

Projecting maxima over the $n_2$ dimensions for the $n_1$-dimensional visualization in mode $result/error$ does not appear to be meaningful.

When the number of cells per axis over all the $n_2$ axes is set to 1, the inner dimensions are projected or aggregated automatically. The resulting graphs for the two modes depict different function characteristics. Figures 5.16a and b show the integration results for function (5.3) by aggregating (summing) the results along the z direction, and the maxima by projecting along the z direction, respectively. The number of function evaluations (used for coloring in Figure 5.16b) were aggregated in both cases. From Figure 5.16a, it is apparent that there is a narrow positive surface and a narrow negative surface in the three-dimensional space.

Figure 5.17 shows an example of projecting maxima over the $n_2$ dimensions for the $n_1$-dimensional visualization in mode $max/np$. The function is the integrand of a five-dimensional loop integral [18]. Three of the five dimensions were projected onto the xy plane.

A four-dimensional integration problem arises from the cross-section computation for the collision of an electron and a positron (producing two muons and a photon) [59]. Its integrand is represented by about a thousand lines of C code. A three-dimensional
Figure 5.16  Dimension reduction. (a) Maxima projection. (b) Result aggregation.
Figure 5.17  Dimension reduction of a five dimensional function. Visualization enables us to view more details. AdaptView ran in mode result/error and aggregated the fourth dimension of the function onto the xyz space as shown in Figure 5.18.

5.6 Summary

Several functions were visualized by AdaptView. According to the complexity analysis in Chapter 6, the adaptive cubature method is efficient for the integration of (5.2); the iterated method is efficient for DICE2, DICE3, and the four-point loop integral (5.3). Iterated integration and extrapolation were used in [18] for the problem of Figure 5.17. The adaptive cubature method is inefficient for the problem of Figure 5.18. Preliminary analysis indicates that it can potentially be solved by the iterated method.
Figure 5.18  Visualization of a four-dimensional physics function.
We introduce iterated integration methods in this chapter by giving a definition, a complexity analysis, an implementation, and experimental results.

6.1 Definition of the Iterated Method

As mentioned in Chapter 2, we will call an integration method iterated, if lower-dimensional methods are used for the integration in different coordinate directions [50, 83, 40]. In practice a numerical iterated method is different from an analytical iterated method. From a programming point of view, an iterated method is better coded by calling functions recursively, instead of in an iterated manner.

The work of [17, 18] shows that the iterated method is much more efficient in integrating certain Feynman loop integrals of particle physics.

We will show the procedure by using one-dimensional integration methods in each coordinate direction to compute an $n$-dimensional ($n \geq 2$) integral numerically. The integral of an $n$-dimensional function $f(x_1, x_2, ..., x_n)$ over a hyper-rectangular region $D$ in $\mathbb{R}^n$ is

$$I = \int_D f(x_1, x_2, ..., x_n) \, dx_1 \, dx_2 \ldots dx_n,$$

which can be rewritten as

$$I = \int_{x_1^a}^{x_1^b} \int_{D'} f(x_1, x_2, ..., x_n) \, dx_2 \ldots dx_n,$$
if condition (2.3) is met. Let

\[ F(x_1) = \int_{D'} f(x_1, x_2, \ldots, x_n) \, dx_2 \ldots dx_n, \]

then integral (6.2) becomes

\[ I = \int_{x_1^b}^{x_1^a} F(x_1) \, dx_1, \]

which is a one-dimensional integral. We then start from (6.3) and repeat the process for the remaining coordinate directions. One does not need to use only one-dimensional methods to solve an \( n \)-dimensional problem. For example, a one-dimensional routine can be called in the \( x \) direction, while for every evaluation point \( x \), a three-dimensional routine can be called. Therefore, there are totally \( 2^{n-1} \) possible methods for an \( n \)-dimensional integration, if the non-iterated method is also counted. Unless otherwise specified, we will use the term *iterated method* to refer to the method implemented with one-dimensional adaptive routines.

### 6.2 Implementation

An implementation of the iterated method in the C language is provided in this section. Parallelization is discussed. Issues such as setting input parameters are covered.

#### 6.2.1 An Implementation in C

An iterated method has been implemented for PARINT1.2 [76]. We use quadrature rules adapted from QuadPack [61] to compute one-dimensional integrals. Because the implementation is a little tricky, we list part of it below.
The Iterated Method:

extern void
iterint(pi_ifcn_t ifcn_ptr, int nfcns, int ndims, 
        pi_base_t a[], pi_base_t b[], pi_base_t eps_a, 
        pi_base_t eps_r, int irule, pi_total_t fcn_eval_limit, 
        pi_base_t result[], pi_base_t estabs[], 
        int *fcn_limit_flag, pi_total_t *fcncount) {
    integrand = ifcn_ptr;
    local_ndims = ndims;
    local_a = a;
    local_b = b;
    local_eps_a = eps_a;
    local_eps_r = eps_r;
    local_irule = irule;
    local_fcn_eval_limit = fcn_eval_limit;
    local_fcncount = 0;
    local_fcn_limit_flag = 0;
    int flag = 0;
    currentAxis = 0;
    xx = (pi_base_t*)malloc(ndims * sizeof(pi_base_t));
    seq_integrate(foo, nfcns, 1, a, b, eps_a, eps_r, irule, 
                   fcn_eval_limit, result, estabs, &flag, fcncount);
    *fcncount = local_fcncount;
    *fcn_limit_flag = local_fcn_limit_flag || flag;
    free(xx);
}

int foo(int *ndim, pi_base_t *x, int *nfcns, 
        pi_base_t *funvls) {
    pi_base_t estabs_dummy;
    pi_total_t fcncount_dummy;
    int flag = 0;
    pi_base_t* my_a;
    pi_base_t* my_b;
    xx[currentAxis] = x[0];
    if (currentAxis == local_ndims - 1) {
        integrand(ndim, xx, nfcns, funvls);
        local_fcncount++;
    } else {
        currentAxis++;
        my_a = local_a + currentAxis;
        my_b = local_b + currentAxis;
        }
seq_integrate(foo, *nfcns, 1, my_a, my_b, local_eps_a,
    local_eps_r, local_irule, local_fcn_eval_limit,
    funvls, &estabs_dummy, &flag, &fcncount_dummy);
currentAxis--; 
    local_fcn_limit_flag = local_fcn_limit_flag || flag; 
} 
return 0; 

Real data type pi_base_t can be double or long double. Function pointer ifcn_ptr points to
a C function for the user's integrand. The prototype of the C function is listed in Chapter 3
and is the same as that of foo. To perform a one-dimensional integration in the above code
seq_integrate is invoked. For convenience, \(x[0]\) and \(x[n - 1]\) represent the first and the last
variables, respectively.

The user defines the integrand (see [90] for details) and other input parameters, passes
them into iterint, and gets the result and error back via result and estabs, respectively. In
iterint, integration is done along the \(x[0]\)-axis with the integrand foo, instead of integrand.
In foo, the current axis is now \(x[1]\) \(currentAxis = 1\). The lower and the upper limits
are adjusted to those of the \(x[1]\) direction. The process is repeated recursively until the last
direction is reached, where the real integrand function, integrand, is invoked to carry out
the function evaluation.

6.2.2 Input Error Tolerances

The total error tolerance is the same as the outer integration error tolerance. For a
given total error tolerance, selecting the error tolerances of the inner integrals is non-trivial.
Currently we use the same relative error tolerance for all levels, and use an absolute error
tolerance for only the outer integration.
D. Kahaner and co-workers outline the contribution of the inner and the outer integration errors and a heuristic estimation of the total error for a two-dimensional iterated integration [40]. They suggest that the inner integral be computed about a factor of ten more accurately than the outer. If the total absolute error tolerance is $\varepsilon_a$, then

$$\varepsilon_a^O = 0.9\varepsilon_a, \quad (6.5)$$

and

$$\varepsilon_{ai}^I = \frac{\varepsilon_a}{10(x_b^i - x_a^i)}. \quad (6.6)$$

The total estimated error is given by,

$$\text{error}_O + (x_b^i - x_a^i) \max_{x_i} \text{error}_I(x_i), \quad (6.7)$$

where $\text{error}_O$ is the estimated error of the outer integral, and $\text{error}_I(x_i)$ is that of the inner integral for a given value $x_i$ of $x$. For a two-dimensional iterated method, computing the inner integration ten times more accurately than the outer does not incur much extra work. For higher-dimensional problems, this practice is discouraged.

Fritsch and co-workers study the error tolerance assignment for a two-dimensional iterated method in [25]. The total absolute error tolerance is,

$$\varepsilon_a^T = \varepsilon_a^O + \varepsilon_{ai}^I, \quad (6.8)$$

where $\varepsilon_a^O$ and $\varepsilon_{ai}^I$ are the absolute error tolerances for the outer and the inner integrations, respectively. For $m$-panel ($m+1$-point) closed Newton-Cotes rules, an optimal ratio is given by

$$\frac{\varepsilon_{ai}^I}{\varepsilon_a^O} = \frac{m^O + 2}{m^I + 2}. \quad (6.8)$$

If the function has no peaks or untoward behavior, a constant error tolerance $\varepsilon_{ai}^I$ is used for all points $x_i$ in the x direction. If the function is well behaved over most of its domain but
has peaks or oscillations in one or more reasonably local sections, the assignment is to have
\[ \varepsilon_{\text{rel}}^I |W_i| = \text{constant}, \]
where \( W_i \) is the weight assigned at the first appearance of \( F(x_i) \) in (6.3). Note that the adaptive method used in [25] is basically different from those used in our scheme.

Further study is necessary to find a simple and efficient error assignment scheme.

### 6.2.3 Some other Issues

The quadrature routine and its parameters for a direction can be specified independently of those of the other directions. If the integrand oscillates along a direction, a routine suited for oscillatory behavior may be better for that direction. Assigning relative error tolerances for each direction is not an easy issue in practice. A unique value, \( \text{eps} \_r \), applies to all directions in the above code. Note that the absolute error tolerance, \( \text{eps} \_a \), is used only for the first direction; it is the overall absolute error tolerance.

By default, integration starts in the \( x[0] \) direction, resulting in evaluating direction \( x[n-1] \) first because of the nature of the recursion. The end-users can permute the variables in their integrand. The lower and upper limits should be permuted accordingly.

Our implementation is only for one case of all \( 2^{n-1} \) possible implementations for \( n \)-dimensional integration. An implementation supporting all cases will be considered later.

If the region is a simplex, a transformation can be done in the function \( \text{foo} \). The method can further be applied for arbitrary product regions.
6.2.4 Parallelization

Domain decomposition always works in parallelizing a Riemann integrable problem. In principle, a region is divided into a number of sub-regions, which are distributed to multiple processors to be processed by iterated methods. The drawback of iterated methods is that when the dimension is not small, only a limited number of points can be sampled in each direction. This method may not parallelize the problem as needed. Below we will outline schemes for parallelizing iterated methods. No implementation has been done so far.

We introduce a method called rule parallelization. The integration rule will be parallelized. The iterated mechanism makes the computation coarse-grained. On the other hand, rule parallelization makes it fine-grained. Parallelism is expected to be feasible.

If the number of processors is not large (\( \leq 61 \), for example), we can just parallelize the first layer, namely the \( x_1 \) direction. Assuming that a rule evaluates the integrand at \( m \) points, \( x_{1i}, i = 1, \ldots, m \), we have \( m \) sub-problems of dimension \((n - 1)\),

\[
I_i = \int_{D_i} f(x_{1i}, x_2, \ldots, x_n) \, dx_2 \ldots dx_n, \text{ where } i = 1, \ldots, m. \tag{6.9}
\]

These sub-problems are sent to other processors using a scheduling method. Since the regions along \( x_1 \) can be divided adaptively to obtain the specified accuracy, more tasks are available as more sub-regions are added to the priority queue.

As mentioned above, the user can designate any variable to be \( x_1 \). If the integrand is difficult along some coordinate direction, this direction is a good candidate for the \( x_1 \)-axis.

If the computation time of (6.9) varies greatly, and/or the evaluation limit on the \( x_1 \)-axis is small, a second layer parallelization may be carried out. In the \((x_1, x_2)\) plane, the
The integrand is evaluated at points \((x_{1i}, x_{2j})\) in the course of adaptive computation, leading to sub-problems of the form,

\[
I_{ij} = \int_{D'} f(x_{1i}, x_{2j}, \ldots, x_n) \, dx_3 \ldots dx_n, \quad \text{where } i, j = 1, \ldots, m. \tag{6.10}
\]

The ranges of \(i\) and \(j\) are not limited to \([1, m]\), when the integration is done along \(x_1\) and \(x_2\) adaptively.

The second layer parallelization scheme should be able to achieve good results for many large applications. If the computation scale is huge, or the problem is of a particular type (with one very narrow peak, for example), a third layer parallelization can be carried out.

### 6.3 Complexity Analysis

It was mistakenly believed that the iterated method was not suitable for non-smooth functions. Our analysis reveals cases where the iterated method is better than the adaptive method, and vice versa.

#### 6.3.1 Assumptions

In this section, we will analyze the performance of the iterated and the globally adaptive methods. Assume that the one-dimensional routines used by the iterated method are also adaptive. The time complexity (in terms of the number of function evaluations) of several special cases will be analyzed. We consider integrals with a corner peak, a ridge (or singularity) parallel to an axis, or a ridge parallel to a diagonal. The peak or the ridge are steep and well-behaved. Without loss of generality, assume that the integration region is a hyper-cube.
Furthermore, we assume that if an region does not contain any peak or ridge, it will not be processed again. Let us consider \( I = \int_0^1 \frac{1}{x^{0.9}} \, dx \) as an example. Only the leftmost region will be taken from the region queue for further processing at each step. In practice, regions such as \([0.5, 1]\) may be partitioned into a number of sub-regions. If this number is bounded by a constant, the conclusions of the time complexity analysis remain valid.

Finally, we assume that the execution will stop when the shortest side of all regions reaches \( \delta \) in size. \( N = RL/\delta \) is defined as the problem size regardless of the integrand dimension, where \( L \) is the side length of the hyper-cube, and \( R \) is the number of function evaluations per rule. \( N \) is the total number of function evaluations in a coordinate direction if the regions is subdivided uniformly into \( L/\delta \) intervals in that direction. If different rules are applied in different coordinate directions, the number in direction \( i \) is \( N_i = R_i L/\delta \).

According to the assumptions, the complexity of a one-dimensional trapezoidal rule is \( O(N) \).

### 6.3.2 Two-dimensional Integration

For two-dimensional integration, functions with a corner peak, a ridge parallel to axis, a ridge parallel to diagonal, and ridge with other orientations are analyzed below.

**Corner Peak**

Corner peak functions are often used as test cases for adaptive methods. As shown in Figure 6.1, approximately \( 4 \log N \) region evaluations are needed to meet the termination condition. Assuming that the number of function evaluations per rule, \( R \), is a constant, the complexity is \( O(\log N) \). Adaptive methods are particularly effective in solving these type
Figure 6.1  Corner peak with adaptive method.

Figure 6.2  Corner peak with iterated method.
Figure 6.3 Parallel to axis with adaptive method.

Figure 6.2 shows only part of the points on the x-axis. Red lines depict the region partitioning on the x-axis. The y direction integration is performed along the vertical black lines, on which region partitioning points are marked. The area is smooth on the right, so less region partitioning is needed there. As x approaches 0, a vertical line is partitioned in a way similar to that of the x-axis. If we assume that all vertical lines are subdivided like the x-axis, which is an overestimate, the complexity of the iterated method is $O(\log^2 N)$.

**Parallel to Axis**

To simplify the situation where a ridge is parallel to an axis, we assume that the integrand is constant along the other axis. The complexity of the adaptive method (see Figure 6.3) is still $O(\log N)$. That of the iterated method is $O(R_1 \log N)$, where $R_1$ is the number of evaluations per rule along the first direction of the iterated method. It is a constant (3 in Figure 6.4). Therefore, the complexity of the iterated method is also $O(\log N)$. 

68
Figure 6.4  Parallel to axis with iterated method.

Figure 6.5  Diagonal ridge with adaptive method.

Figure 6.6  Diagonal ridge with iterated method.
**Diagonal Ridge**

With the ridge along or parallel to a diagonal, the adaptive method is very inefficient compared to the iterated method. The partitioning process for the former is similar to that for a corner peak. The length of the ridge is $O(N)$, so the complexity of the adaptive method is $O(N \log N)$. That of the iterated method is the same as the parallel ridge case, namely $O(\log N)$. The two methods are illustrated by Figure 6.5 and Figure 6.6.

**Other Orientations**

When the angle between the ridge and an axis is between $0^\circ$ and $45^\circ$, the performance of the iterated method remains almost the same, but that of the adaptive method ranges somewhat between that of the best ($0^\circ$) and the worst ($45^\circ$).

The integrand of the following integral (6.11) is like that of Dice 1 (6.12). It has a ridge along a diagonal, but its crest at height $2/\varepsilon$ is parallel to the xy-plane.

\[
I = \int_0^1 \int_0^1 dx_1 dx_2 \frac{2\varepsilon}{(x + y - 1)^2 + \varepsilon^2}, \quad (6.11)
\]

We rotated it with respect to the point $(0.5, 0.5)$ and performed numerical integration for a variety of angles between the ridge and the $x$ axis, for $\varepsilon = 0.01$ and a relative error tolerance of $10^{-10}$. Table 6.1 lists the relationship between the ridge angle and the number of function evaluations performed. When the ridge is parallel to the $x$ axis, the number of function evaluations is 5,355, which is close to that of the iterated method (6,975). When the ridge is along the diagonal, 6,621,069 evaluations are needed, corresponding to the degenerated efficiency of the adaptive method.
Table 6.1  Performance of the adaptive method for ridges with various orientations.

<table>
<thead>
<tr>
<th>Angle</th>
<th>0</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
<th>35</th>
<th>45</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evals</td>
<td>5.4E3</td>
<td>8.7E5</td>
<td>1.7E6</td>
<td>2.5E6</td>
<td>3.4E6</td>
<td>4.2E6</td>
<td>4.8E6</td>
<td>5.4E6</td>
<td>6.6E6</td>
</tr>
</tbody>
</table>

Figure 6.7  Diagonal surface of 3D function.

6.3.3  Three-dimensional Integration

For the corner peak case, the complexities of the adaptive method and of the iterated method are $O(\log N)$ and $O(\log^3 N)$, respectively. The ridged behavior now located at a two-dimensional surface, as shown in Figure 6.7. If this surface is parallel to any boundary surface of the hyper-cube, the complexity of the adaptive method is still $O(\log N)$. However if it is along a diagonal surface as in Figure 6.7, that will be $O(N^2 \log N)$. The method adaptive method will be unacceptably slow. The complexity of the iterated method is $O(\log N)$ regardless of the orientation of the surface.

6.3.4  Multi-dimensional Integration

For higher-dimensional problems, no general conclusion can be made easily. The omitted constants must be counted. They can lead to exponential growth of the evaluation count...
as a function of the dimension for the iterated method as well as for the adaptive method.

The exponential growth is an inherent nature of the iterated method, so it cannot be used for high-dimensional problems. The evaluation count of the adaptive method may not grow exponentially as a function of the dimension if the integrand is smooth and a suitable rule is applied.

### 6.3.5 Dimension Limits

The minimum number of function evaluations of the iterated method grows exponentially as a function of the integrand dimension $n$. What is the limit of $n$ for applying this method? If two points are computed in each direction, $n$ can be as high as around 35, assuming the region is not subdivide. If five points are computed on each direction, the limit of $n$ is about 15. The minimum number of function evaluations of the one-dimensional Gauss-Kronrod rule in PARINT is 15. Therefore, they can be utilized to try problems of dimensions up to 9 or 10. We can add Gauss-Kronrod pairs with less points.

### 6.3.6 A Sample Application

We refer to the following integral as DICE1 [81],

$$I = \int_{-1}^{1} \int_{-1}^{1} dx_1 dx_2 \frac{2\varepsilon y}{(x + y - 1)^2 + \varepsilon^2},$$

where $\theta(t) = 1$ for $t \geq 0$, and 0 otherwise. The DICE1 integrand has a ridge of height $2y/\varepsilon$ along the diagonal $y = 1 - x$. The DICE2 function of (5.1) has a ridge along the circle of radius $a$ centered at the origin, and a discontinuity at the unit circle.

Two-dimensional adaptive cubature methods are not very effective for computing these
integrals, which mimic the behavior of certain integration problems arising in high-energy physics computations. Results are given in [16] for $a = 0.8$, $\varepsilon = 10^{-1}, 10^{-2}, \ldots, 10^{-6}$, a relative error tolerance of $10^{-5}$, and a function evaluation limit of 250 million, which is much larger than that of the iterated method.

6.4 Numerical Results of Iterated and Adaptive Methods

In order to confirm the results of the proceeding section, we performed experiments with both the iterated and the adaptive methods for a family of functions.

6.4.1 Test Functions

Iterated methods deserve a thorough investigation. We used functions from the test function library in the PARINT package to compare the performance of the traditional adaptive method implemented in PARINT with that of the iterated method.

There are two reasons for this numerical experiment. One is to confirm the complexity analysis in the former section. Another is to support the claim that iterated methods should be among the default methods for lower-dimensional problems in the major numerical software packages.

6.4.2 Results

Table 6.2 lists the function names, analytical integral results, function evaluation limits along each axis for the iterated method, and the actual number function evaluations performed by the iterated method. The latter was used as the function evaluation limit for the adaptive method. The absolute error tolerance and the relative error tolerance were set
to 0 and to $10^{-6}$, respectively for all simulations unless specified otherwise. The function definitions, integration lower limits and upper limits are listed in Tables 6.4 to 6.7. Also see to the stdlib.c file in PARINT 12 [76] for further details.

In PARINT the first four rules, PI_IRULE_DIM2_DEG13 (a two-dimensional rule of polynomial degree 13), PI_IRULE_DIM3_DEG11 (a three-dimensional rule of polynomial degree 11), PI_IRULE_DEG9_OSCIL (a general multivariate rule of polynomial degree 9, preferred for oscillatory functions), and PI_IRULE_DEG7 (a general multivariate rule of polynomial degree 7) are from Genz and Malik [31, 32], with refined error estimation techniques from [5]. The univariate rules are from the Quadpack package [61].

Results and estimated relative errors are listed in Table 6.3. The integrals for the adaptive and the iterated methods were computed for varying Limit values.

For the rows fcn5 (limit = 100), fcn8 (limit = 100), fcn9 (limit = 10,000), and fcn15 (limit = 100), the evaluation limits of the adaptive method evaluations were not reached. For fcn9 (limit = 1,000), the relative error tolerance, $\varepsilon_r$, is $10^{-13}$. For fcn22 (limit = 10,000) and for fcn32 (limit = 1,000), $\varepsilon_r$ is $10^{-12}$.

6.4.3 Discussions

The functions in Table 6.4 all have a corner peak at the origin. Functions fcn18, fcn19, fcn21, and fcn22 differ from the other ones in that they are divergent when any of their coordinate components is zero. These problems have two advantages for the adaptive method, namely a peak and a singular parallel surface. Compared to the adaptive method, the iterated method performs poorly on these problems.
<table>
<thead>
<tr>
<th>Name</th>
<th>Analytical</th>
<th>Evaluations</th>
<th>Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>fcn1</td>
<td>1.43476188839726</td>
<td>11390625</td>
<td>100</td>
</tr>
<tr>
<td>fcn2</td>
<td>0.57536414490356</td>
<td>50625</td>
<td>100</td>
</tr>
<tr>
<td>fcn3</td>
<td>2.15214283259589</td>
<td>3375</td>
<td>100</td>
</tr>
<tr>
<td>fcn4</td>
<td>16.00000000000000</td>
<td>759375</td>
<td>100</td>
</tr>
<tr>
<td>fcn5</td>
<td>0.18390715290765</td>
<td>151875</td>
<td>100</td>
</tr>
<tr>
<td>fcn6</td>
<td>-4.00000000000000</td>
<td>2025</td>
<td>100</td>
</tr>
<tr>
<td>fcn7</td>
<td>0.86304621735534</td>
<td>211005</td>
<td>100</td>
</tr>
<tr>
<td>fcn7</td>
<td>0.86304621735534</td>
<td>40632075</td>
<td>1000</td>
</tr>
<tr>
<td>fcn8</td>
<td>1.04759111314287</td>
<td>4725</td>
<td>100</td>
</tr>
<tr>
<td>fcn8</td>
<td>1.04759111314287</td>
<td>34875</td>
<td>1000</td>
</tr>
<tr>
<td>fcn9</td>
<td>499.124944224122</td>
<td>4725</td>
<td>100</td>
</tr>
<tr>
<td>fcn9</td>
<td>499.124944224122</td>
<td>8775</td>
<td>1000</td>
</tr>
<tr>
<td>fcn9</td>
<td>499.124944224122</td>
<td>39375</td>
<td>10000</td>
</tr>
<tr>
<td>fcn10</td>
<td>1.43656365691809</td>
<td>10125</td>
<td>100</td>
</tr>
<tr>
<td>fcn14</td>
<td>4.00000000000000</td>
<td>1575</td>
<td>100</td>
</tr>
<tr>
<td>fcn14</td>
<td>4.00000000000000</td>
<td>34875</td>
<td>1000</td>
</tr>
<tr>
<td>fcn15</td>
<td>12.83333333333333</td>
<td>225</td>
<td>100</td>
</tr>
<tr>
<td>fcn16</td>
<td>2.95249244199546</td>
<td>225</td>
<td>100</td>
</tr>
<tr>
<td>fcn18</td>
<td>1.667629948</td>
<td>11025</td>
<td>100</td>
</tr>
<tr>
<td>fcn18</td>
<td>1.667629948</td>
<td>534645</td>
<td>1000</td>
</tr>
<tr>
<td>fcn19</td>
<td>2.211</td>
<td>4100625</td>
<td>45</td>
</tr>
<tr>
<td>fcn21</td>
<td>1.887873</td>
<td>1157625</td>
<td>100</td>
</tr>
<tr>
<td>fcn22</td>
<td>4.00000000000000</td>
<td>11025</td>
<td>100</td>
</tr>
<tr>
<td>fcn22</td>
<td>4.00000000000000</td>
<td>1010025</td>
<td>1000</td>
</tr>
<tr>
<td>fcn22</td>
<td>4.00000000000000</td>
<td>5546025</td>
<td>10000</td>
</tr>
<tr>
<td>fcn32</td>
<td>1.3862943611</td>
<td>5385</td>
<td>100</td>
</tr>
<tr>
<td>fcn32</td>
<td>1.3862943611</td>
<td>547335</td>
<td>1000</td>
</tr>
<tr>
<td>fcn34</td>
<td>0.33979807084</td>
<td>7487025</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 6.2  Numerical results for a collection of functions (I).
<table>
<thead>
<tr>
<th>Name</th>
<th>Iterated</th>
<th>Error</th>
<th>Adaptive</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>fcn1</td>
<td>1.43476188839726</td>
<td>1.6E-14</td>
<td>1.43476188832569</td>
<td>6.4E-09</td>
</tr>
<tr>
<td>fcn2</td>
<td>0.57536414490356</td>
<td>1.8E-12</td>
<td>0.57536414141783</td>
<td>1.6E-07</td>
</tr>
<tr>
<td>fcn3</td>
<td>2.15214283259589</td>
<td>2.3E-11</td>
<td>2.15214283202701</td>
<td>2.2E-08</td>
</tr>
<tr>
<td>fcn4</td>
<td>16.0000000000000</td>
<td>2.5E-13</td>
<td>16.0000000006955</td>
<td>1.0E-08</td>
</tr>
<tr>
<td>fcn5</td>
<td>0.18390715290765</td>
<td>2.0E-15</td>
<td>0.18390715290765</td>
<td>1.7E-15</td>
</tr>
<tr>
<td>fcn6</td>
<td>-4.0000000000000</td>
<td>8.3E-11</td>
<td>-4.00000000673113</td>
<td>1.7E-06</td>
</tr>
<tr>
<td>fcn7</td>
<td>0.86276463816444</td>
<td>8.7E-02</td>
<td>0.86304621095491</td>
<td>2.5E-08</td>
</tr>
<tr>
<td>fcn8</td>
<td>0.86304621574112</td>
<td>7.0E-07</td>
<td>0.86304621735450</td>
<td>6.5E-13</td>
</tr>
<tr>
<td>fcn9</td>
<td>499.114531836706</td>
<td>2.7E+02</td>
<td>499.124944226826</td>
<td>1.6E-07</td>
</tr>
<tr>
<td>fcn10</td>
<td>1.43655777711722</td>
<td>2.8E-05</td>
<td>1.4365649036740</td>
<td>2.5E-05</td>
</tr>
<tr>
<td>fcn14</td>
<td>3.93540097246011</td>
<td>1.3E+00</td>
<td>3.99971912244947</td>
<td>6.8E-03</td>
</tr>
<tr>
<td>fcn15</td>
<td>12.8333333333333</td>
<td>1.4E-13</td>
<td>12.8333333333333</td>
<td>2.4E-16</td>
</tr>
<tr>
<td>fcn16</td>
<td>2.95249244201256</td>
<td>3.3E-14</td>
<td>2.95249244207235</td>
<td>2.9E-07</td>
</tr>
<tr>
<td>fcn18</td>
<td>1.66659604917836</td>
<td>7.0E-02</td>
<td>1.6667507747749</td>
<td>3.5E-03</td>
</tr>
<tr>
<td>fcn19</td>
<td>1.66672992842095</td>
<td>9.8E-07</td>
<td>1.66672994758581</td>
<td>1.7E-14</td>
</tr>
<tr>
<td>fcn21</td>
<td>1.88632258109814</td>
<td>6.7E-02</td>
<td>1.88787319219291</td>
<td>7.5E-06</td>
</tr>
<tr>
<td>fcn22</td>
<td>3.93566178710756</td>
<td>6.6E-01</td>
<td>4.001972727759546</td>
<td>4.2E-01</td>
</tr>
<tr>
<td>fcn22</td>
<td>3.99999980285437</td>
<td>2.0E-05</td>
<td>4.00000000000014</td>
<td>6.4E-13</td>
</tr>
<tr>
<td>fcn22</td>
<td>3.99999999999967</td>
<td>3.7E-12</td>
<td>4.00000000000014</td>
<td>4.0E-14</td>
</tr>
<tr>
<td>fcn32</td>
<td>1.38603663561484</td>
<td>8.8E-02</td>
<td>1.38629437515789</td>
<td>2.9E-06</td>
</tr>
<tr>
<td>fcn32</td>
<td>1.38629436111965</td>
<td>8.5E-11</td>
<td>1.38629436111990</td>
<td>2.1E-12</td>
</tr>
<tr>
<td>fcn34</td>
<td>0.33965014959162</td>
<td>4.3E-02</td>
<td>0.33979807315611</td>
<td>8.4E-09</td>
</tr>
</tbody>
</table>

**Table 6.3**

Numerical results for a collection of functions (II).
The results for the two-dimensional problems \textit{fcn22} and \textit{fcn32} indicate that the iterated method is slower than the adaptive method, but still achieves very accurate results with a moderate additional number of function evaluations. From the data of \textit{fcn32}, \textit{fcn7} and \textit{fcn34}, we can see that the adaptive method becomes less efficient with increasing dimensions, but it is much more accurate for these problem.

We also tested \textit{fcn18} and \textit{fcn22} with r2d2lri [66], which is a non-adaptive algorithm using a sixth-order Sidi transformation [72] applied to a sequence of embedded lattice rules. For estimated relative errors of $10^{-7}$ and $2 \times 10^{-7}$, it took 353 and 709 function evaluations to integrate \textit{fcn18} and \textit{fcn22}, respectively. For functions with boundary singularities as in \textit{fcn18} and \textit{fcn22}, applying suitable transformation before employing the iterated method is expected to increase its efficiency significantly.

<table>
<thead>
<tr>
<th>Name</th>
<th>Expression</th>
<th>Lower Limits</th>
<th>Upper Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>fcn7</td>
<td>$1/(x_0 + x_1 + x_2)^2$</td>
<td>0, 0, 0</td>
<td>1, 1, 1</td>
</tr>
<tr>
<td>fcn18</td>
<td>$x_0^{-0.2} x_1^{-0.2} (x_0 + x_1)^{-0.2}$</td>
<td>0, 0</td>
<td>1, 1</td>
</tr>
<tr>
<td>fcn19</td>
<td>fcn18 in 4 dims</td>
<td>0, 0, 0, 0</td>
<td>1, 1, 1, 1</td>
</tr>
<tr>
<td>fcn21</td>
<td>fcn18 in 3 dims</td>
<td>0, 0, 0</td>
<td>1, 1, 1</td>
</tr>
<tr>
<td>fcn22</td>
<td>$1/(x_0 x_1)^{0.5}$</td>
<td>0, 0</td>
<td>1, 1</td>
</tr>
<tr>
<td>fcn32</td>
<td>$1/(x_0 + x_1)$</td>
<td>0, 0</td>
<td>1, 1</td>
</tr>
<tr>
<td>fcn34</td>
<td>$1/(x_0 + x_1 + x_2 + x_3)^3$</td>
<td>0, 0, 0</td>
<td>1, 1, 1, 1</td>
</tr>
</tbody>
</table>

Table 6.4 Functions with a corner peak.

Functions \textit{fcn8} (see Figure 6.8) and \textit{fcn9} shown in Table 6.5 do not have singularities. The two methods are efficient in integrating them.

Of the three oscillatory functions in Table 6.6 the \textit{fcn5} problem is trivial. Even though, for \textit{fcn4} and \textit{fcn6}, degree nine oscillatory rules were used by the adaptive method, the
<table>
<thead>
<tr>
<th>Name</th>
<th>Expression</th>
<th>Lower Limits</th>
<th>Upper Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>fcn8</td>
<td>$605x_1/(1 + 120(1 - x_1))/((1 + 120(1 - x_1))^2 + 25x_0^2x_1^2)$</td>
<td>0, 0</td>
<td>1, 1</td>
</tr>
<tr>
<td>fcn9</td>
<td>$1/((x_0^2 + 0.00001)((x_1 + 0.25)^2 + 0.0001))$</td>
<td>0, 0</td>
<td>1, 1</td>
</tr>
</tbody>
</table>

Table 6.5 Functions $fcn8$ and $fcn9$.  

Figure 6.8 Function $fcn8$.  

78
iterated method outperforms it.

<table>
<thead>
<tr>
<th>Name</th>
<th>Expression</th>
<th>Lower Limits</th>
<th>Upper Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>fcn4</td>
<td>$\cos(x_0 + x_1 + x_2 + x_3 + x_4)$</td>
<td>0, 0, 0, 0</td>
<td>$\pi, \pi, \pi, \pi/2$</td>
</tr>
<tr>
<td>fcn5</td>
<td>$\sin(10x_1)$</td>
<td>0, 0, 0, 0</td>
<td>1, 1, 1, 1</td>
</tr>
<tr>
<td>fcn6</td>
<td>$\cos(x_0 + x_1)$</td>
<td>0, 0</td>
<td>$3\pi, 3\pi$</td>
</tr>
</tbody>
</table>

Table 6.6 Oscillatory functions.

The remaining functions of Tables 6.2 and 6.3 are listed in Table 6.7. Both $fcn14$ and $fcn15$ are functions of only variable, and are integrated over a two-dimensional region. For $fcn14$, both methods achieve very high accuracies with their minimum evaluations ($15 \times 15 = 225$ for the iterated, 21 for the adaptive method). For $fcn10$, which has a discontinuous first degree derivative, both methods work reasonably well. The the iterated method does better for the rest of the functions. The rule PI_RULE_DQK61 (Gauss 30 points - Kronrod 61 points) gives better result than PI_RULE_DQK15 (Gauss 7 points - Kronrod 15 points) for the one-dimensional routine used by the iterated method to integrate $fcn10$.

<table>
<thead>
<tr>
<th>Name</th>
<th>Expression</th>
<th>Lower Limits</th>
<th>Upper Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>fcn1</td>
<td>$x_0x_1^2\sin(x_2)/(4 + x_3 + x_4 + x_5)$</td>
<td>0, 0, 0, 0, -1, -1, -1</td>
<td>2, 1, $\pi$, 1, 1, 1</td>
</tr>
<tr>
<td>fcn2</td>
<td>$x_2^2x_3\exp(x_2x_3)/(x_0 + x_1 + 1)^2$</td>
<td>0, 0, 0, 0</td>
<td>1, 1, 1, 2</td>
</tr>
<tr>
<td>fcn3</td>
<td>$8/(1 + 2(x_0 + x_1 + x_2))$</td>
<td>0, 0, 0</td>
<td>1, 1, 1</td>
</tr>
<tr>
<td>fcn10</td>
<td>$\exp(</td>
<td>x_0 + x_1 - 1</td>
<td>)$</td>
</tr>
<tr>
<td>fcn14</td>
<td>$1/\sqrt{x}$ if $x_0 \leq 1, 1/\sqrt{x} - 2$ others</td>
<td>0, 0</td>
<td>2, 1</td>
</tr>
<tr>
<td>fcn15</td>
<td>$x_0^2 + 5x_0 + 10$</td>
<td>0, 0</td>
<td>1, 1</td>
</tr>
<tr>
<td>fcn16</td>
<td>$\exp(x_0 + x_1)$</td>
<td>0, 0</td>
<td>1, 1</td>
</tr>
</tbody>
</table>

Table 6.7 Miscellaneous functions.
6.5 Summary

We studied the iterated numerical integration method and showed that it is extremely efficient in solving certain classes of problems. One class of these problems are loop integrals of fairly low dimensions (less than 6) [73, 18, 17, 6, 27, 7]. Note that some problems involving multi-dimensional integrals can be reduced to the computation of many two- and three-dimensional integrals as indicated in [7]. A multidimensional integral can be approximated by using a combination of lower-dimensional or one-dimensional adaptive methods iteratively. We implemented an iterated method using one-dimensional adaptive routines.

If the integrand function has only a limited number of singular points (i.e., localized singular behavior), or if a singular ridge is parallel to one of the axes, adaptive methods is usually effective.

We tried integrands with extremely sharp ridges, and found that limitations of the machine precision, instead of the algorithm, prevented us from further improving the results for some problems. In this cases, setting the function evaluation limit to a large value may result in unnecessary function evaluations and possibly a less accurate result. We need to make use of reasonable summation methods [39] in future PARINT releases.

Iterated integration yields a good candidate for parallel/distributed integration methods because of the coarse granularity of the inner integral evaluations.
CHAPTER 7

New Load Balancing Method Implementation in PARINT

PARINT1.2 [21, 90, 16] and former releases are mainly implemented in the C language. PARINT3, based on PARINT1.2, is implemented in C++. To support a larger scale of distributed computation, new load balancing methods are developed for PARINT3.

7.1 Introduction

One of the main targets of upgrading PARINT is to increase the algorithm scalability. In a domain with a larger number of processing units or on an environment across domains, scalability becomes more critical for integration to perform well. The network bandwidth and latency call for coarse-grained load balancing methods. In PARINT1.2 and former releases, each worker gets one region from the controller to start with. Consequently, the original workload is usually distributed unevenly, because the integrand often behaves differently over the sub-regions of the original region. Sending a single region causes too much communication overhead. A set of regions should be sent instead.

7.2 Load Balancing Methods in PARINT3

Three new methods are described in this section, namely an adaptive cubature method, an adaptive Quasi-Monte Carlo method, and a hybrid method. For the latter two, load balancing is only applied during the region distribution stage, therefore it is static.
7.2.1 Adaptive Cubature Method

Of the \( p \) nodes or processes, one acts as a controller, and the others are workers. The controller is responsible for balancing the workload and for other administrative tasks. In the following algorithm, the controller also participates in the region evaluation.

**Algorithm for the Controller:**

1. Receive and process the initial regions from an input file or another source.
2. Process the regions adaptively, in order to generate a certain number of regions for each node.
3. Group the regions in a round robin fashion and distribute the region sets. For example, regions \( 1, p + 1, 2p + 1, \ldots \), are sent to the first worker.
4. Send the computed absolute error tolerance to all workers.
5. Probe to see if there is a message from any worker. This may be a blocking probe or a non-blocking probe, depending on the controller’s state.
6. If the message is an update message, update the global result, error estimate, absolute error tolerance, number of integrand evaluations, and the worker’s state; then send the updated absolute error tolerance to all workers, if necessary.
7. If regions are sent by a worker, receive and process the regions (in this case, the controller participates in the region evaluation).
8. Process a number of regions, and inform a busy worker to send regions to an idle worker, if necessary. Send the updated absolute error tolerance to all workers, if needed.
9. If the termination condition has not been met yet, go to step 4; otherwise inform all workers to send their last update and stop working.

An worker update message includes the change of the worker’s estimated absolute error since its previous update. This information is used to determine which busy worker should send regions to an idle worker in the next load balancing step. The regions to be offered to an idle worker are some of those with the largest estimated absolute errors.
A worker or the controller can be in one of several states, e.g. busy, idle, etc. The state data are used for load balancing and termination.

The worker’s algorithm is simpler. It needs to decide when to send an update to the controller. The worker’s current state is kept in a field of the update message.

**Algorithm for a Worker:**

1. Receive and process the initial regions from the controller.
2. Receive the global absolute error tolerance.
3. Send an update to the controller.
4. Probe for a message. This may be a blocking probe or a non-blocking probe, depending on the worker’s state.
5. If regions are offered by a worker (or the controller), receive and process the regions.
6. If an ID is received and if the worker is busy, send regions to the node represented by the ID; otherwise inform the controller that it is not busy.
7. If the incoming data is the global absolute error tolerance, update the corresponding variable.
8. If it is a stop command, send the last update and stop working.
9. Process a number of regions.
10. Send an update if necessary.

An alternative algorithm is similar to the one used in the PI service client. The controller pre-splits the original region into a set of sub-regions, which are sorted by estimated absolute error. The sub-regions with larger error are sent to the workers, but not all regions are delivered to the workers. Some are kept for load balancing. The worker sends an update to the controller when necessary. The controller sends some of the remaining regions to the worker, if the worker’s local estimated relative error is less than or near the relative error
tolerance. Regions are not communicated between workers. Network traffic is therefore reduced.

7.2.2 Adaptive Quasi-Monte Carlo Method

Quasi-Monte Carlo (QMC) methods [20, 58, 19, 12, 69] can be used for high-dimensional problems (under certain smoothness conditions on the integrand). They have also been used effectively for low-dimensional problems (combined with a transformation to handle singularities) [66]. Adaptive MC/QMC methods [47, 48, 70, 62] can be applied in some cases where the variation of the integrand is concentrated in small areas of the integration domain.

The QMC method in PARINT1.2 [20, 12] uses the lattice (Korobov) rules from [30]. The practical maximum number of function evaluations is limited by the rules. The new method is based on the former implementation for a single unit of work, and employs domain decomposition. There are three major stages.

The first stage is a largest-region-first pre-splitting stage. During this stage, the regions are not sorted by error. The purpose of this step is to detect regions where the integrand is smooth. A region with the largest volume is taken from the region set, subdivided, and integrated. A region, with an estimated relative error less than a specified value, will not be processed with the QMC method in stage three. When this stage is over, there may be a number of regions with estimated absolute errors significantly larger than those of the rest. To achieve a good load balance for multiple processing units, those regions are subdivided in stage two.
Stage two is a largest-error-first pre-splitting stage. This is a typical adaptive procedure applied to the regions remaining from stage one.

In stage three, regions will not be further partitioned. A QMC rule is assigned to a region based on its estimated absolute error or its result, and the remaining function evaluation limit. The regions are grouped in round robin fashion (as in the adaptive cubature method) and distributed to the workers. The workers compute their regions and send the results to the controller.

7.2.3 Hybrid Method

The hybrid method is similar to the adaptive QMC method described above except for the fact that during the pre-splitting stages a cubature rule of high polynomial degree is applied to evaluate a region. The estimated error will be small if the integrand is smooth over the region. The region can be discarded while its region and error are kept. This method is effective when the function is smooth over most of the integration domain. Since the number of points of the cubature rule increases significantly as a function of the integrand dimension, this method cannot be applied to high-dimensional problems.

7.3 Speedup and Efficiency

Only the upper nodes of the Athena cluster participated in the experimental performance tests for the adaptive cubature methods. The test problem is DICE3 with $\epsilon = 10^{-3}$, a relative error tolerance of 0.03, a function evaluation limit of 800 million, and integration rule PI_IRULE_DIM3_DEG11 (dimension 3 rule of polynomial degree 11). For PARINT1.2, all other parameters are set to their default value. For PARINT3, the minimum
Figure 7.1  Speedup and efficiency. (a) Speedup. (b) Efficiency.
region chunk sent in a load balancing step is 100; the initial number of regions per worker is 400; and an update is sent to the controller only if at least 4000 region evaluations have been done since the previous update.

The speedup and efficiency are depicted in Figure 7.1. The evaluation limit was never reached, thus no estimated relative errors exceed 3%. Except for the case with two workers, the speedup of the new method is significantly better than that of the former method under the given test conditions.

The largest run time for these test runs was less than 3 minutes. We expect the new method to scale well, if we increase both the amount of computation and the number of processors.

7.4 Summary

The new methods in PARINT3 are designed to be scalable. More tests are to be performed before a stable version can be released. Initial experimental runs showed the expected scalability. Different rules, point sequences or even methods (adaptive, iterated, MC, QMC) can be combined in it in order to deal with various type of problems.
CHAPTER 8

Conclusions

We addressed the construction of an online distributed environment for solving integration problems. We built and tested experimental system components. Special attention was paid to improving the integration engine. A useful tool for problem analysis was implemented.

Pion provides a user-friendly interface for end-users to access high performance computational resources. The integration engine parallelizes the computation automatically. The user submits only an integrand function and parameters.

We implemented new load balancing methods in PARINT to improve the scalability of distributed integrations. Scalability is critical in large scale grid computation. The idea behind building PI service is that large scale integration can be built on multiple service end points. Each end point is a building block that can be invoked by other programs.

Standardizing the interfaces of integration packages, either sequential or distributed, is still an issue. We proposed an XML format for exchanging data between applications. An agreement on a standard needs to be reached by different groups.

AdaptView, as a function visualization tool, gives an overall view and also shows the important details of a function. It employs integration methods for visualization, and is useful for analyzing integration problems. For example, for some problem, it can be used to decide whether an adaptive cubature method or an iterated method should be used.

We showed that the iterated method is very efficient in many situations for low-dimensional
problems. We gave a time complexity analysis for several classes of integrands. The results can be used to predict the complexity for problems with an integrand behavior somewhat in between the extreme cases. Use of an iterated method can reduce the execution time tremendously, or make the solution of some problems feasible, where regular adaptive cubature methods fail. Iterated methods have their disadvantages. First, iterated methods are not suitable for high-dimensional problems. Furthermore, if the dimension is not very low, the maximum number of function evaluations cannot be precisely controlled. In addition, setting the error tolerance for the integration in different directions is not trivial.

Numerical integration is still an active field of research. In view of the diversity of the integrand function, no single rule or method can meet all requirements. However, in our opinion, iterated methods deserve more attention in the future.
Appendices

A Visualization of Adaptive Partitioning

Figure A.1 Visualization of adaptive partitioning (I). Colors represent estimated error.
Figure A.2  Visualization of adaptive partitioning (II). Colors represent estimated errors.
B  A Sample XML File for Integration

<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- parameter file for integration programs --> <parint>

<!-- parameters that describe the problem -->
<problem>
  <!-- plugin name, the binary file that contains the function -->
  <L>stdfuncs.ppl</L>
  <!-- The function name, which must be the same as the function name in body section below, if present -->
  <f>fcn7</f>
  <!-- A description of the integrand function, which is usually optional -->
  <description>
    The function is f(x) = 1 / (x0 + x1 + x2)^2
  </description>
  <!-- Dimension -->
  <dimension>3</dimension>
  <!-- Absolute error tolerance -->
  <ea>1E-6</ea>
  <!-- Relative error tolerance -->
  <er>1E-6</er>
  <!-- Relative error tolerance as # of digits -->
  <ed></ed>
  <!-- Number of integrands, usually 1 -->
  <nfcns>1</nfcns>
  <!-- The region(s) used may also be a default, or determined by the rule number below -->
  <rgns>
    <!-- Define a multiple region set -->
    <rect>
      <!-- First region -->
      <rgn>
        <pt> 0.0 0.0 0.0 </pt>
        <pt> 1.0 1.0 1.0 </pt>
      </rgn>
      <!-- Second region -->
    </rect>
  </rgns>
</problem>
<!-- Third region -->

<rgn>
  <pt> 0.0 0.0 0.0 </pt>
  <pt> 1.0 0.0 0.0 </pt>
  <pt> 0.0 1.0 0.0 </pt>
  <pt> 0.0 0.0 -1.0 </pt>
</rgn>
  <!-- Fourth region -->

<rgn>
  <pt> 0.0 0.0 0.0 </pt>
  <pt> -1.0 0.0 0.0 </pt>
  <pt> 0.0 -1.0 0.0 </pt>
  <pt> 0.0 0.0 -1.0 </pt>
</rgn>
</splx>
</rgns>

<!-- Function body, contains an integrand function (see also the ParInt manual) -->

<body>
  <![CDATA[
    int fcn7(int *ndims, pi_base_t *x, int *nfcns, pi_base_t *funvls){
      pi_base_t z = x[0] + x[1] + x[2];
      z *= z;
      funvls[0] = (z != 0.0 ? 1.0 / z : 0.0);
      return 0;
    }
  ]]>}
</body>
</algorithm>

<!-- Function evaluation limit, can be very large -->

<lf>400000</lf>
Number of runs, usually 1 -->
<onr>1</onr>

Load balancing? 1, yes; 0, no -->
<lb>1</lb>

Controller also works? 1, yes; 0, no -->
<caw>1</caw>

Number of processors -->
<np>1</np>

Adaptive algorithm parameters -->
<adaptive>

Integration rules
PI_IRULE_DIM2_DEG13 = 1;
PI_IRULE_DIM3_DEG11 = 2;
PI_IRULE_DEG9_OSCIL = 3;
PI_IRULE_DEG7 = 4;
PI_IRULE_DQK15 = 5;
PI_IRULE_DQK21 = 6;
PI_IRULE_DQK31 = 7;
PI_IRULE_DQK41 = 8;
PI_IRULE_DQK51 = 9;
PI_IRULE_DQK61 = 10;
PI_IRULE_QMC = 11;
PI_IRULE_SIMPLEX_DEG3 = 12;
PI_IRULE_SIMPLEX_DEG5 = 13;
PI_IRULE_SIMPLEX_DEG7 = 14;
PI_IRULE_SIMPLEX_DEG9 = 15;

Integration rule, must be a number -->
<r>2</r>

Error ratio threshold for load balancing, >= 1.0 -->
<ohr>1.2</ohr>

Worker granularity -->
<ons>5</ons>

Maximum heap size. 0, no limit -->
<ohs>0</ohs>

Max # of region evaluations -->
<lr> </lr>

Percentage threshold controlling eps messages -->
<oet>0.01</oet>

Percentage threshold controlling
update message -->
<out>0.1</out>

<!-- parameters for the new load balancing method -->
<minRegionChunk> 100 </minRegionChunk>
<maxRegionChunk> 100000 </maxRegionChunk>
<initRegionsPerWorker> 1 </initRegionsPerWorker>
<initProducedRegions> 1000 </initProducedRegions>
<evaluationsPerRound> 100 </evaluationsPerRound>
<evaluationsPerUpdate> 100 </evaluationsPerUpdate>
<sendUpdateThreshold> 0.2 </sendUpdateThreshold>
<sendRegionThreshold> 1.2 </sendRegionThreshold>
<sendNewEpsAThreshold> 0.1 </sendNewEpsAThreshold>
<absIdleThreshold> 0.1 </absIdleThreshold>
<relIdleThreshold> 0.1 </relIdleThreshold>
<freeHeap> false </freeHeap>
</adaptive>

<!--- QMC algorithm parameters -->
<qmc>
  <!-- qmc_table_length -->
  <oqcl>10</oqcl>
  <!-- qmc_table_width -->
  <oqcm>6</oqcm>
  <!-- Start row -->
  <b></b>
  <!-- Number of columns -->
  <c></c>
  <!-- split each cell into <count> pieces -->
  <k></k>
</qmc>

<!-- MC algorithm parameters -->
<mc>
  <!-- Work unit time (in seconds) -->
  <t>0.1</t>
</mc>

<!-- parameters for the adaptive QMC and hybrid methods -->
<adaptiveqmc>
  <splitRegions> 1 </splitRegions>
  <splitPerWorker> 100000 </splitPerWorker>
  <columns> 2 </columns>
</adaptiveqmc>
<rounds> 10000 </rounds>
<fraction> 0.1 </fraction>
<maxNumUpdates> 1 </maxNumUpdates>
<gmcSplit> 1 </gmcSplit>
<findSplitAxis> 1 </findSplitAxis>
<adaptiveQMCRule> 8 </adaptiveQMCRule>
<noLastStep> 1 </noLastStep>
</adaptiveqmc>
</algorithm>

<hierarchical>
<!-- Turn on hierarchical method and set the number of groups -->
<hiergroup></hiergroup>
<!-- Set function parameter file for hierarchical run -->
<hierfile></hierfile>
</hierarchical>

<!-- IO settings --> <io>
<!-- Less verbose? -->
<lessverb>0</lessverb>
<!-- Display temporary results every n evals -->
<tr></tr>
<!-- Output file name (records tagged values) -->
<o>result.txt</o>
<!-- Output file for logging -->
<plfile>temp.txt</plfile>
<!-- Read the regions set from this file -->
<InRgnfile>file.in</InRgnfile>
<!-- Write regions to this file -->
<OutRgnfile>file.out</OutRgnfile>
<!-- Input file system type used for inRgnfile, where:
  0 - read files from NFS mounted partition;
  1 - read files from a partition local to each worker;
  2 - the files are read only by the controller -->
<InFS></InFS>
<!-- Output file system type used for outRgnfile, where:
  1 - each worker writes a separate file;
  2 - the controller (only) writes the files. -->
<OutFS>1</OutFS>
<!-- Output file type: b - binary, t - text -->
<webService>
  <!-- Service to be requested:
       0 - computation;
       1 - max points;
       2 - result and error -->
  <operationType>2</operationType>
  <!-- Endpoints for Web service -->
  <endPoints>
    <endPoint>
      http://aegis.cs.wmich.edu:8080/axis/services/ParIntService
    </endPoint>
    <endPoint>
      http://lupata.cs.wmich.edu:8080/axis/services/ParIntService
    </endPoint>
    <endPoint>
      http://localhost:8080/axis/services/ParIntService
    </endPoint>
    <endPoint>
      http://demeter.cs.wmich.edu:8080/axis/services/ParIntService
    </endPoint>
  </endPoints>
  <!-- Remote computation visualization -->
  <visualization>
    <speed>100</speed>
    <grid>
      <cells>128</cells>
      <cells>128</cells>
      <cells>10</cells>
    </grid>
  </visualization>
  <!-- number of partitions, for pre-splitting -->
  <numPartitions>8</numPartitions>
</webService>


68. SCHUERER, R. Hintlib – high-dimensional integration library. [http://www.cosy.sbg.ac.at/~rschuer/hintlib/](http://www.cosy.sbg.ac.at/~rschuer/hintlib/).


89. WORLD WIDE WEB CONSORTIUM. http://www.w3.org/.