The Pennsylvania State University

The Graduate School

Department of Computer Science and Engineering

PARALLEL ALGORITHMS AND SOFTWARE FOR MULTI-SCALE MODELING OF CHEMICALLY REACTING FLOWS AND RADIATIVE HEAT TRANSFER

A Thesis in

Computer Science and Engineering

by

Ivana Veljkovic

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

May 2006

The thesis of Ivana Veljkovic has been reviewed and approved* by the following:

Paul Plassmann
Associate Professor of Computer Science and Engineering
Thesis Adviser
Chair of Committee

Jesse Barlow
Professor of Computer Science and Engineering

Padma Raghavan
Professor of Computer Science and Engineering

Daniel Haworth
Professor of Mechanical and Nuclear Engineering

Michael Modest
Professor of Mechanical and Nuclear Engineering

Raj Acharya
Professor of Computer Science and Engineering
Head of the Department of Computer Science and Engineering

*Signatures are on file in the Graduate School.

Abstract

Multi-scale modeling, the simulation of coupled physical processes that occur on different temporal or spatial scales, is becoming an increasingly important area of research in computational science. Solving such problems can be computationally intensive; however, because of the increasing availability of large computational resources, their solution can be feasible. An important multi-scale application is the numerical simulation of combustion. In combustion, three different physical processes govern the dynamics of the problem: fluid flow (which can be turbulent or laminar), chemical reactions, and heat transfer, with radiative heat transfer being a dominant mode. The objective of this thesis is to create efficient sequential and parallel algorithms and software that improve accuracy and performance of combustion applications. Another purpose of this work is to create clean, easy-to-use software interfaces that can be readily used from both C/C++ and FORTRAN applications without significant changes to the original code.

In this thesis, we introduce two new software systems that enable modeling of multi-scale phenomena in combustion applications on single processor and distributed memory multiprocessor systems and improve their accuracy and performance. The first system is called the Database On-Line for Efficient Function Approximation (DOLFA) for speeding up chemistry calculations in combustion applications. A second system, called Photon Monte Carlo (PMC) [1], is used for solving the Radiative Transfer Equation (RTE) by calculating the radiative heat fluxes for the volume elements of a computational mesh. The PMC software system is capable of handling computational domains with complex enclosures and various radiation configurations.

Table of Contents

List of Tables

List of Figures

Nomenclature

α	absorptance of the surfaces
α	plane in the composite space
β, β_η	extinction coefficient that describes total attenuation (total, spectral)
$\boldsymbol{\chi}$	Vector difference, $\boldsymbol{\chi} = \boldsymbol{Q} \cdot (\boldsymbol{\theta} - \boldsymbol{\xi})$
Δ	difference vector, $\Delta = \boldsymbol{\theta} - \boldsymbol{\xi}$
ϵ	User specified tolerance (for ILDM method)
η	wave number, $[\mathrm{cm}_{-1}]$
λ	wavelength, [μm]
λ_i	i-th eigenvalue of the Jacobian matrix in the ILDM method
Λ	scaling matrix (length of ellipsoid axis)
ν	distance from the given point to the EOA
Ω	solid angle, [sr]
Φ	dissipation function, [J/kg m^2
$\Phi(\hat{\mathbf{s}}_i, \hat{\mathbf{s}})$, $\Phi_\eta(\hat{\mathbf{s}}_i, \hat{\mathbf{s}})$	phase function that describes the probability that a ray coming from direction $\hat{\mathbf{s}}_i$ will scatter into the direction $\hat{\mathbf{s}}$ (total, spectral)
$\boldsymbol{\psi}_0$	arbitrary point in the flame
$\boldsymbol{\psi}_m$	point corresponding to $\boldsymbol{\psi}_0$ in the manifold

$\boldsymbol{\theta}, \boldsymbol{\xi}, \boldsymbol{\varphi}$	points in the composite space
$\{\boldsymbol{\xi}_1, ..., \boldsymbol{\xi}_n\}$	list of points associated with a region T
τ	User specified tolerance
τ_p	Global minimal physical time scale (in the ILDM method)
σ	Stefan—Boltzmann constant = $5.67x10^{-8}\mathrm{W/m^2K^4}$
σ_s, $\sigma_{s\eta}$	scattering coefficient (total, spectral)
Σ	Eigenvalues of matrix $A = J^T J$, possibly with some zero entries
a	normal vector of the plane
a_i	normal vector of the plane in the i-th stage of the transformation of the composite space C
`avgLocal`	prediction of the processor p_i of the number of calculations it will have in the lists $\mathcal{B}^{(i)}$ and $\mathcal{N}^{(i)}$
`avgDirect`	the average number of direct retrievals (over all processors) in the previous iteration
A	$A = J^T J$
BSP tree	Binary Space Partition tree
$\mathcal{B}^{(i)}$	portion of the list $\mathcal{L}^{(i)}$ that belongs to the subtree of p_i

c	modifying factor for the EOA (when growing or shrinking)
	free coefficient of the plane
c_v	specific heat, [kJ/kg K]
c_i	free coefficient of the plane in the i-th stage of the transformation of the composite space C
`CleanPoint`, `CleanDelete`	user-specified parameters for the cleaning of the database
$\mathcal{C}^{(i)}$	the list of points for which processor p_i does the direct calculations
C_i	number of direct calculations on the processor p_i during one iteration
C	composite space(N-dimensional space where $N = N_s + 2$. It consists of N_s chemical species, temperature and pressure)
E(N,P)	scaled efficiency as a function of problem size N and number of processors P
$\mathbf{C}_{\boldsymbol{\xi}}$	transformed composite space in which the point $\boldsymbol{\xi}$ of the original composite is the origin and EOA centered in that point is a unit sphere
$dt, \Delta t$	time step
d	critical distance in ray/particle interaction

d	dimension of the space in which the enclosure for PMC lies
E	total emissive energy of the enclosure, [W]
E	energy of a ray, [W]
E	edge in a mesh
EOA	Ellipsoid Of Accuracy
E_i	original EOA in the i-th stage of the transformation of the composite space C
E_i	emissive energy of the enclosure element i, [W]
$\mathcal{E}^{(i)}$	the list of points that p_i received from other processors since they belong to its subtree T_i
f_start, *f_end*	the faces a ray is intersecting on its way through the element
$\mathbf{F}(...)$	general mapping of any N-dimensional space
$\boldsymbol{F}_{\boldsymbol{\xi}}(\boldsymbol{\theta})$	the approximate function value in point $\boldsymbol{\theta}$, computed with the function previously calculated for point $\boldsymbol{\xi}$
g	cumulative k-distribution
I, I_η	radiative intensity (total, spectral)
J	Jacobian matrix
I,J,K	cells in the discretized domain
k	thermal conductivity, [W/m K]

k_η	spectral absorption coefficient, $[\mathrm{cm}_{-1}]$
k_{P}	Planck-mean absorption coefficient, $[\mathrm{cm}_{-1}]$
K	average number of cells that each ray traverses
K	number of subdomains resulting from the partitioning of the global domain, used in hybrid load decomposition
$\mathcal{L}^{(i)}$	the list of points submitted to processor p_i
$\mathcal{L}^{(i)}_j$	the portion of the list $\mathcal{L}^{(i)}$ that belongs to subtree T_j of processor p_j
$\mathcal{L}_{CPU}$	computational load in parallel PMC
$\mathcal{L}_{Comm}$	communication load in parallel PMC
$\mathbf{L}$	list of photon bundles to trace
L	characteristic length of the subdomain
L	number of processor groups for hybrid load decomposition
$\boldsymbol{M}_n$	mean value of the set of n points in the N-dimensional space
$\mathbf{M}$	pointer to the mesh
M	number of cells in the mesh
n	number of points in a given region T of the BSP tree
N	dimension of the composite space

N	total number of rays emitted from the enclosure
N_i	number of rays emitted from the enclosure element i
N_s	number of chemical species
N_s	dimension of the manifold in the ILDM method
$\mathcal{N}^{(i)}$	portion of the list $\mathcal{L}^{(i)}$ that belongs to the subtrees of other processors
$\mathcal{N}_C$	number of communicated rays during the parallel PMC simulation
p	pressure,[bar]
p_i	i-th processor in the parallel computing environment
P	number of processors
$\mathbf{P}$	convex region (terminal leaf, polytope) in the BSP tree
q_r	radiative flux, [W/cm^2]
$\dot{Q}'''$	heat production per unit volume, [W/cm^3]
$Q_{em,j}$	radiative emission from volume cell j,[W]
$Q_{abs,j}$	radiative absorption from volume cell j,[W]
Q	rotation matrix (position of ellipsoid axis)
$\mathbf{R}(...)$	reaction mapping of the composite space
R_{part}	radius of a particle

R_{max}	maximum radius of a cone
$\hat{\mathbf{s}}$	direction of the photon bundle propagation
S	sphere representing the transformed EOA
S	number of independent subsamples
$Reg(\boldsymbol{\xi})$	region of accuracy around point $\boldsymbol{\xi}$
t	instance of time
t_0	initial time
$\mathbf{T}$	convex region (terminal leaf, polytope) in the BSP tree
T_{l}, T_{r}	left and right subregion of the region T
T_i	subtree of the global BSP tree that belongs to processor p_i
T	temperature, [K]
T(N,P)	Total CPU time for a problem of size N on P processors
$\mathbf{v}$	velocity vector, [cm/s]
V	principal vectors of the Jacobian matrix in the ILDM method
V	volume, [cm^3]
V_s	eigenvectors of the Jacobian matrix in the ILDM method corresponding to the slow time scales

V_f	eigenvectors of the Jacobian matrix in the ILDM method corresponding to the fast time scales

Acknowledgments

I would like to express my gratitude to my thesis advisor, Dr. Paul Plassmann for his support and guidance. He always encouraged, motivated, and challenged me throughout my research and collaboration with him. With his highly positive philosophy he encouraged me to become more independent and creative researcher. I would also like to thank the professors from the research group, Dr. Michael Modest and Dr. Daniel Haworth, for their patience, understanding and guidance. Next, I offer my gratitude to my colleagues from the research group, especially Anquan Wang and Yuhui Wu, for their help with solving various problems I encountered during my research. Last but not least, I would like to acknowledge the support and assistance given by Dr. Padma Raghavan and Dr. Jesse Barlow. This research was supported by NSF grants CTS–0121573, ACI–0305743, and ACI–9908057. The use of parallel computers was supported in part by NSF grants EIA–0202007 and DGE–9987589.

Chapter 1

Introduction

1.1 Background

Multi-scale modeling, the simulation of coupled physical processes that occur on different temporal or spatial scales, is becoming an increasingly important area of research in computational science. Problems where multi-scale interactions are significant are common, and can be found in a wide variety of disciplines including combustion, material science, chemistry, nanotechnology, and biology. Solving such problems can be computationally intensive; however, because of the increasing availability of large computational resources, their solution can be feasible. The challenge of these multi-scale models is their efficient implementation—new algorithms and software must be developed to dramatically improve the performance of these models.

An important multi-scale application is the numerical simulation of combustion. In combustion, three different physical processes govern the dynamics of the problem: fluid flow (which can be turbulent or laminar), chemical reactions, and heat transfer, with radiative heat transfer being a dominant mode. These physical processes occur on vastly different time scales. For example, to capture most of the properties of a turbulent, non-steady fluid flow in a typical application, time steps of order 10^{-3} to 10^{-6} seconds are required. Chemical reactions cover a much wider range of time scales—from 1 to 10^{-10} seconds. Finally, radiative heat transfer occurs at the speed of light, resulting in time scales of order of 10^{-9}. two processes. To enable accurate

modeling of many combustion applications, the inclusion of the interaction between these three processes is essential.

A number of investigations has been conducted to examine the influence of the chemistry model on the accuracy of the combustion application (see, for instance [6]). Reduced chemistry mechanisms can result in substantial erroneous predictions of mass fractions of important chemical species (such as CO). Similarly, omitting radiative heat transfer from the physical model in combustion applications can lead to vast misspredictions of the temperature profile. This exclusion can also affect the stability and accuracy of the calculation of other variables (such as mass fractions of chemical species). As shown in Fig. 1.1, taken from DiLorenzo at al. [2], the temperature profile obtained from an application that simulates adiabatic conditions (no radiative heat exchange) is compared with the temperature profile from the simulation when radiative losses are included and with experimental results from [2]. The temperature distribution is presented along 1.0 centimeter of an atmospheric $CH_4 - O_2$ flame with equivalence ratio 2.54. We observe a significant temperature decrease caused by radiative losses. Thus, development of feasible computational models that will accurately describe these processes represents a significant challenge.

1.2 Thesis Contributions

With a new generation of supercomputers, such as the Earth Simulator and Blue Gene/L [7], and the availability of commodity parallel computers, such as Beowulf clusters [8], the computational power available to scientists today is immense. As a result, extremely complex and sophisticated physical models can be simulated on these computers. Nonetheless,

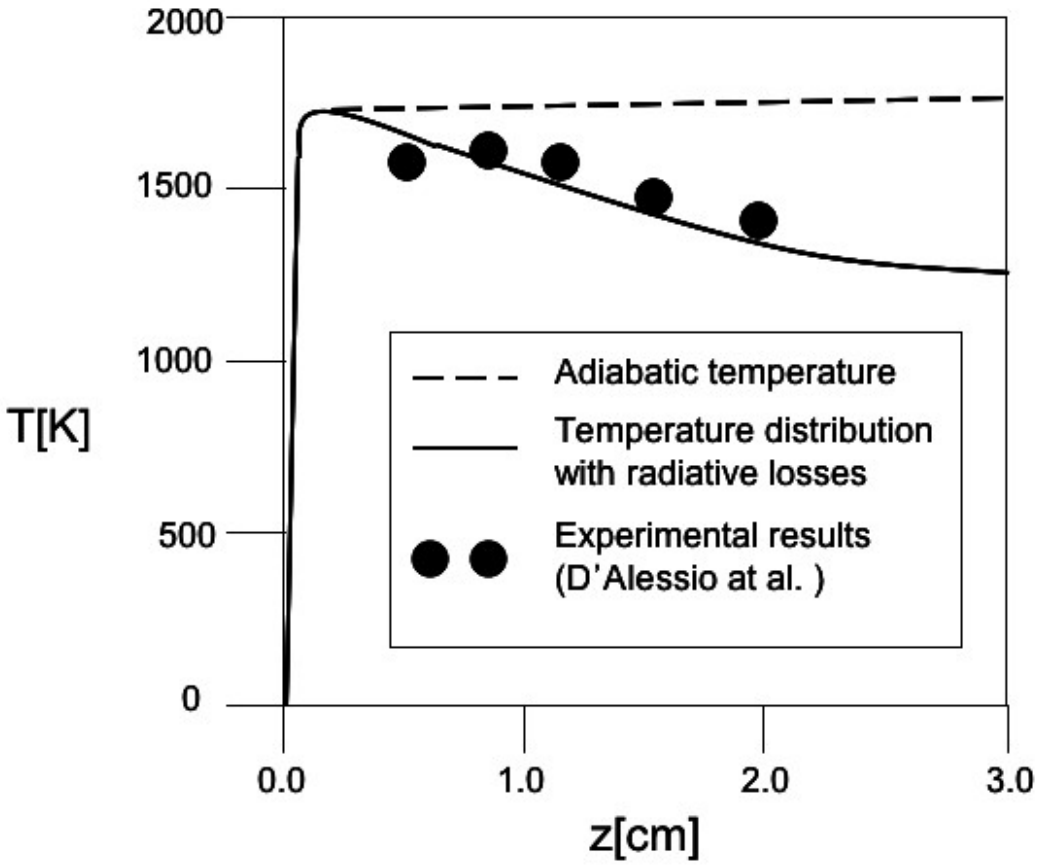


Fig. 1.1. Temperature decrease caused by radiative losses, from [2]

the complete multi-scale simulation of phenomena such as combustion can still be computationally prohibitive. Therefore, the objective of this thesis is to create efficient sequential and parallel algorithms and software that improve accuracy and performance of combustion applications. Another purpose of this work is to create clean, easy-to-use software interfaces that can be readily used from both C/C++ and FORTRAN applications without significant changes to the original code.

In this thesis, we introduce two new software systems that enable modeling of multi-scale phenomena in combustion applications on single processor and distributed memory multiprocessor systems and improve their accuracy and performance. The first system is called the Database On-Line for Efficient Function Approximation (DOLFA) [9, 10]. DOLFA is used to speed up chemistry calculations in combustion applications. A second system, called Photon Monte Carlo (PMC) [1], is used for solving the Radiative Transfer Equation (RTE) by calculating the radiative heat fluxes for the volume elements of a computational mesh. The PMC software system is capable of handling computational domains with complex enclosures and various radiation configurations. Specific contributions include the following.

For the sequential implementation of DOLFA, we have proposed a new algorithm of building the search tree, which enables a guaranteed retrieval—if a suitable point exists in the database, it will be found. Also, we have designed a cleaning strategy that can be useful in non-steady-state simulations to keep the database size manageable. Finally, we have implemented constant and hybrid approximation schemes within DOLFA and have shown that they are more suitable for most applications than more computationally expensive linear approximation. For the parallel implementation of DOLFA, we have developed new algorithms for distributing the search tree. The distributed search tree is built off-line, using the list of points that are submitted

to DOLFA with the first call to the database. In the case of transient simulations, the search space can change dramatically which can cause a significant load imbalance. We have designed a redistribution algorithm that redistributes the parts of the search tree based on the computational load of the previous call to the database. Also, we have introduced three effective heuristics for managing the computational load and analyzed their performance.

For the PMC implementation, we have designed an extensible and robust framework that allows the implementation of various models of radiation in a wide range of applications. We have developed new algorithms for tracing photon bundles in the participating medium where scalar and vector properties of the medium are not given on mesh elements but on particles that are distributed over the simulation domain. Also, we have created a scalable parallel framework for Photon Monte Carlo with domain and ray partitioning and introduced various metrics for load balancing and analyzed their effectiveness.

This thesis is organized as follows. In Chapter 2, scientific databases for computational chemistry are introduced. Existing and newly developed algorithms for the sequential tabulative techniques used in the calculation of reacting flows are described. Computational results from representative combustion applications that emphasize approximation accuracy as well as the speedups achieved when using DOLFA software are presented. In Chapter 3 algorithms for parallel implementation of DOLFA are discussed and experimental results from representative combustion applications as well as from simulations that are designed for testing of the scalability of parallel DOLFA are presented. Chapter 4 gives an overview of basic radiative properties, radiative transfer equations and the methodology of Monte Carlo ray tracing. Chapter 5 describes the extensible Monte Carlo software framework that enables independent implementation of various radiation models. It also gives overview of newly developed algorithms that enable the ray

tracing through particulate media. In Chapter 6 basic methods of parallelization of Monte Carlo ray tracing are presented. Load balancing is discussed with possible solutions and remedies to this problem and performance results are presented. Finally, Chapter 7 gives an overview of possible future directions in further development of these two software frameworks.

Chapter 2

Scientific Databases for Computational Chemistry

A detailed description of combustion chemistry usually involves tens of chemical species and hundreds chemical reactions modeled with equations with strong nonlinear properties[11]. The numerical integration of these systems of differential equations is often extremely stiff with time scales that can vary from 1 to 10^{-10} seconds. These different time scales are evident in Fig. 2.1, where we observe the time dependent profile of mole fractions of a hydrogen-air ignition problem at atmospheric pressure, with an initial temperature of 1000K [3]. The sharp gradients in the mole fraction profiles of these species show that any computational model that represents the evolution of the chemical kinetics must be stiff to be able to capture the rapid change in mole fraction profiles.

The modeling of the chemical reactions in these combustion simulations can be summarized as follows. Chemical reactions are simulated for discrete objects (volume elements or fluid particles) obtained from a representation of the computational domain. Specifically, if we are given mass fractions of chemical species at one point in our computational domain in time t, the mass fractions of chemical species after time dt are computed as the solution of a system of Partial Differential Equations (PDEs) or Ordinary Differential Equations (ODEs). Here dt is usually determined from time-scales associated with the turbulent properties of the fluid simulation. Most of the chemical reactions happen on much faster time scales, which must be resolved.

Moreover, chemical reactions present in the combustion processes occur on vastly different time-scales [12]. Hence, the resulting systems of equations are extremely stiff and the computational resources required in their solution dominates those required in the fluid simulation.

Solving the conservation equations for the chemically reactive flow, even for simple two-dimensional laminar flows, with this kind of chemistry is computationally expensive [13]. For more complex reacting flows, especially those involving turbulence effects, calculations that include detailed chemistry models are typically so complex and demanding that they often exceed the computational resources currently available. Database techniques offer an alternative approach—instead of solving the chemistry equations for every point in space, one can solve it for some significantly smaller number of points, and use those solutions to interpolate the solution for other nearby points. In his paper [11], Pope proposed to tabulate previously computed function values; when a new function value is required, the set of previously computed values is searched and, if possible, the function is approximated based on these values.

Function approximations retrieved from the database should satisfy user-specified error tolerances. Significant speedups, of the order 100 to 1000, have been achieved relative to direct integration of the chemistry equations through the use of such tabulation schemes [11]. Our focus in this chapter is on combustion problems that involve detailed computational chemistry. However, this database approach can be used with any kind of computation that involves frequent function evaluations that are computationally expensive.

In the following section several issues related to finding the reaction mapping (one of the main problems in computational chemistry) are briefly reviewed. The rest of this chapter is organized as follows. Section 2.3 gives a description of existing algorithms used for designing the Database On-Line for Function Approximation (DOLFA). In Section 2.4 newly developed

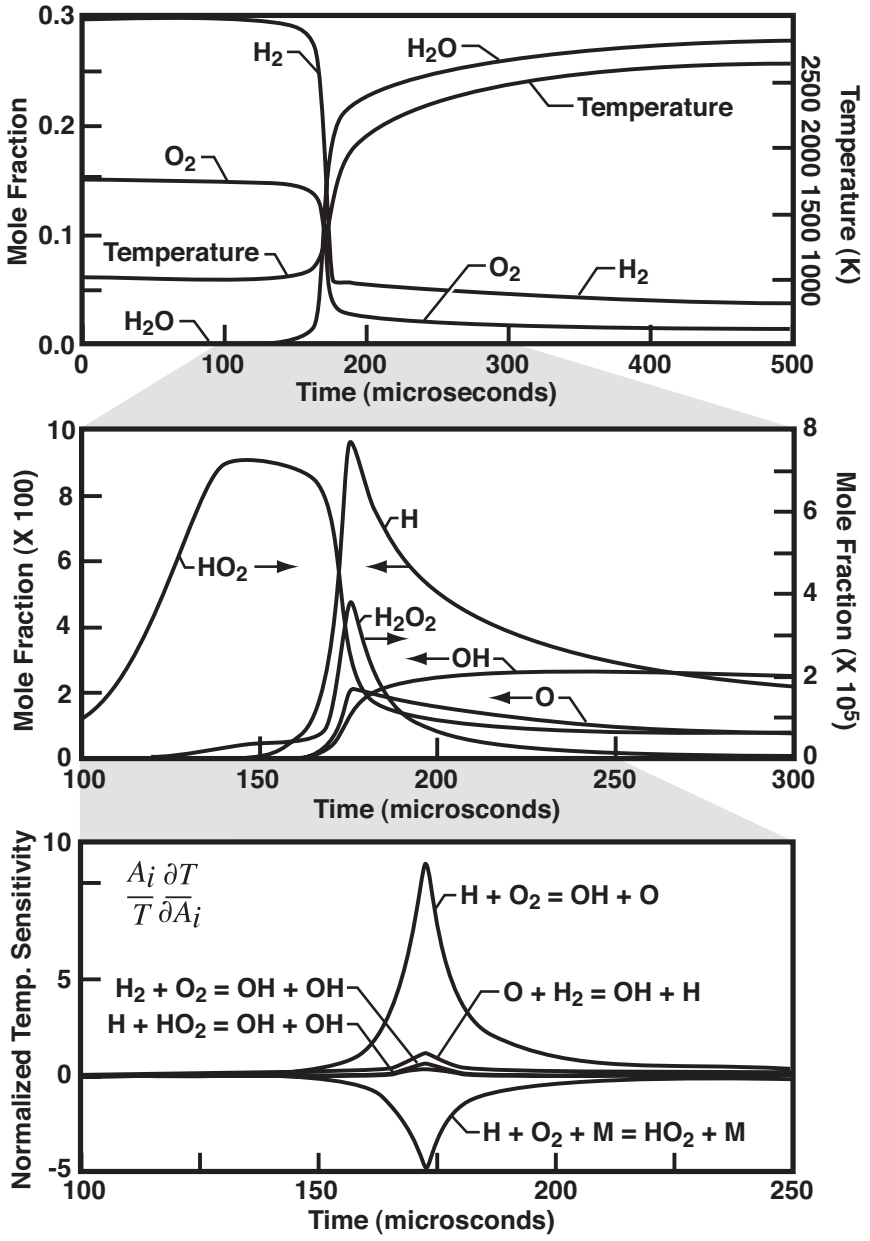


Fig. 2.1. Mole fractions of a hydrogen-air ignition problem at atmospheric pressure, initial T = 1000K, initial mole fractions are $\mathrm{H}_2 = 0.286$, $\mathrm{O}_2 = 0.143$, and $\mathrm{N}_2 = 0.571$. From [3].

algorithms for the DOLFA implementation that address some of the problems that were present in existing schemes are described. Finally, in Section 2.5 experimental results for the sequential case are presented.

2.1 The Reaction Mapping Function for Computational Chemistry

For a given underlying chemical reaction mechanism a finite number, N_s, of chemical compounds (species) are involved. At any point and time in the reactive flow, the thermochemical state of the mixture is characterized by the chemical species mass fractions Y_i $(i = 1, 2, ..., N_s)$, temperature T (or enthalpy h) and the pressure p. Hence, the thermochemical state of this system can be described by an $N = N_s + 2$ dimensional space—often called the *composite space* **C**. Furthermore, only a subset of **C** is actually accessed during calculations; this space is called the *accessed space*. It is also possible to reduce the dimension of the composite space by investigating the functional dependencies between mass fractions of the species [14–16].

Knowing the thermochemical state of a point at a time t_0, $\boldsymbol{\xi}_0 \equiv \boldsymbol{\xi}(t_0)$, we would like to find the thermochemical state of a point at a time $t_0 + \Delta t$. If Δt is fixed, then $\boldsymbol{\xi}(t_0 + \Delta t)$ is a unique function of $\boldsymbol{\xi}_0$ denoted by $\boldsymbol{R}(\boldsymbol{\xi}_0)$, called the *reaction mapping*. The calculation of the reaction mapping usually involves computing the solution to a set of stiff ODEs or PDEs. The problem of calculating reaction rates for a given gas-phase reaction mechanism and mass fractions of chemical species can be solved by using one of a number of the available software packages. Perhaps the most popular such package is CHEMKIN [17–19].

This process is often computationally extremely expensive. Instead of directly calculating the reaction mapping for every element of the mesh (cell or particle), the functional database tries to approximate the reaction mapping at a new query point $\boldsymbol{\theta}$ with the reaction mapping at

some nearby point ξ (if it exists), that has been previously calculated and stored in the database. Using the fact that this mapping is continuous and differentiable, the reaction mapping at a new query point can be approximated by the constant or piecewise linear approximation based on a Taylor series expansion. Linear approximation requires additional computational time and storage because the Jacobian at point ξ must be computed. Constant approximation is less accurate, but it is often preferable, particularly in higher dimensions because there is no need for the Jacobian to be calculated and stored. This simplification can significantly reduce memory space and computational time required for the tabulation software.

2.2 Existing Numerical Methodologies for Approximation of Chemical Kinetics

In this section, we describe two existing alternative methods for reducing the CPU time used for resolving chemical reactions that employ different methodology from scientific databases. The first approach is called Piecewise Reusable Implementation of Solution Mapping (PRISM), and it develops a polynomial model to locally represent the evolution of chemical kinetics. The second approach attempts to determine Intrinsic Low-Dimensional Manifolds (ILDM) that are present in chemically reacting flows. This method locally reduces the number of chemical species involved in the reaction, thus reducing the size of the ODE or PDE systems. There are no public domain or proprietary software packages, which implement these methods, that have been tested and whose performance has been analyzed.

2.2.1 Piecewise Reusable Implementation of Solution Mapping

PRISM [20] was first introduced in 1999 and it uses a piecewise polynomial approximation of the reaction mapping to develop a surrogate polynomial model for the evolution of

chemical reactions. This model approximates **R** in two phases—first, a surrogate model is constructed and second, the surrogate model is used to calculate the reaction mappings. In the first phase, solutions to the ODEs (or PDEs) are calculated for selected points, and local polynomial representations to these solutions are constructed. In the second phase, we do not calculate the reaction mappings through the original differential systems of equations. Instead, we perform the calculations through the constructed surrogate model.

The accuracy of these models is controlled during the construction phase. Since the equations describing chemical reactions are highly nonlinear, polynomial representations can be valid only in some localized neighborhood. Two main questions in constructing the surrogates are: (1) how many local surrogates are required to represent the chemical compositions and temperatures needed for a given problem; and (2) is there sufficient reuse of the surrogates to offset the cost of their construction. The answers to these questions are related to the geometry of the solution in composite space, or more precisely to the geometry of the accessed space (refer to Section 2.1). As noted by Pope [11] from an analytical perspective, *a priori* bounds on the dimension of the active part of composite space **C** that must be represented by surrogates can be developed.

In PRISM the surrogate is defined on regions that are "hypercubes" in the composite space **C** [14]. The sizes of these hypercubes are determined by accuracy requirements, defining a granularity scale for covered portions of **C**. The main drawback of PRISM is that we do not know how many points will be accessed from each hypercube that we use to create our local surrogate model. Therefore, it is possible that a significant number of hypercubes that are created in the first phase do not get accessed in the second phase. Hence, the surrogate model on those hypercubes does not contribute to speeding up the overall chemistry calculation. The

other drawback is the high dimensionality of the composite space. The piecewise polynomial representation of the arbitrary, highly nonlinear function in a high-dimensional space is usually limited to a very small region because of the accuracy requirements. Therefore, the number of hypercubes that have to be used for computation of the surrogate model can be large and the cost of their construction can inhibit the overall performance of this scheme. For more details about PRISM, see [20, 21].

2.2.2 Intrinsic Low-Dimensional Manifolds

The method of Intrinsic Low-Dimensional Manifolds (ILDM) [22], first introduced in 1992, uses the fact that in many chemically reacting flows chemical processes occur with time scales differing by many orders of magnitude (on the order of 1 to 10^{-10} seconds in combustion processes), while the time scales of the flow, molecular transport, and turbulence usually cover a much smaller range of time scales. Based on local time scale analysis, intrinsic low-dimensional manifolds in the composition space are identified. The chemistry relaxes rapidly to these manifolds, and after a short time the evolution of chemical kinetics can be characterized within these subspaces. In practical applications the intrinsic low-dimensional manifolds are calculated before the simulation starts and are then stored for the use in reacting flow calculations. To introduce this method, we start from the governing equation of the chemical kinetics:

$$\frac{\partial \boldsymbol{\varphi}}{\partial t} = \mathbf{F}(\boldsymbol{\varphi}), \tag{2.1}$$

where $\boldsymbol{\varphi}$ belongs to the N-dimensional composite space **C** (normalized space of species mass fractions), $\boldsymbol{\varphi} \in \mathbf{C}$, and $\mathbf{F}(\boldsymbol{\varphi})$ denotes the rates of change in the mass fractions of chemical species due to the chemical reaction.

The eigenvalues and the eigenvectors of the Jacobian $\mathbf{F}_{\boldsymbol{\varphi}}$ can be obtained by the eigenvalue decomposition of $\mathbf{F}_{\boldsymbol{\varphi}}$:

$$\mathbf{F}_{\boldsymbol{\varphi}} = V \Lambda \tilde{V}, \tag{2.2}$$

where $\tilde{V} = V^{-1}$. If we are sufficiently close to the chemical equilibrium point, all eigenvalues are real and negative. The local time scales in the phase space are given by the inverse of the magnitudes of the eigenvalues, $1/|\lambda_1|, ..., 1/|\lambda_n|$, and are ordered from the slowest to fastest. Thus, we can represent matrix V as:

$$V = \begin{pmatrix} V_s \\ V_f \end{pmatrix}, \tag{2.3}$$

where V_s are eigenvectors corresponding to the slow time scales and V_f are eigenvectors corresponding to the fast time scales. The algebraic equation of the manifold then becomes [22]:

$$\tilde{V}_f \mathbf{F}(\boldsymbol{\varphi}) = 0. \tag{2.4}$$

The equation that describes the chemical kinetics is then simplified as:

$$\tilde{V}_s \frac{\partial \boldsymbol{\varphi}}{\partial t} = \tilde{V}_s \mathbf{F}(\boldsymbol{\varphi}). \tag{2.5}$$

Since we eliminated fast time scales, this ODE system has reduced stiffness when compared to equation (2.1). To further reduce the computational time, the system of equations (2.4) is solved

a priori in a predetermined domain of the N-dimensional phase space. The ILDM obtained by the solution of the system of equations (2.4) is stored in a table and parameterized by chosen N_s state variables, where N_s is the dimension of the manifold [23]. There are three broad choices to determine the local dimension N_s of the manifold described below.

1. Fix the dimension based on the global time scales of the reacting flow. This is not a suitable choice, because the required minimal dimension of the manifold can change locally (very low dimension near the equilibrium and higher dimension in the transient regions).

2. The local dimension of the manifold N_s is based on the local time scales of the reaction. If we suppose that the global minimal physical time scale is given by τ_p, we calculate N_s as the number of the local time scales that are slower than τ_p, or, more precisely, the number of the eigenvalues λ_i for which $\lambda_i > -\frac{1}{\tau_p}$.

3. The local dimension of the manifold N_s is based on the local error of the manifold. The minimal N_c for which $\|\boldsymbol{\psi}_m - \boldsymbol{\psi}_0\| < \epsilon$ is found, where $\| \bullet \|$ denotes a weighted norm of the difference between the values $\boldsymbol{\psi}_0$ in the flame and the corresponding values $\boldsymbol{\psi}_m$ on the manifold. In this way no physical time scale has to be identified, but the analysis is based on the local error. For example, if diffusion increases, $\boldsymbol{\psi}_m$ and $\boldsymbol{\psi}_0$ start to differ, and the dimension of the manifold is increased as soon as the error tolerance is reached.

One serious disadvantage of the ILDM method lies in the huge memory requirement for storing information about ILDMs. A second disadvantage lies in the fact that, with the strong diffusion effects, the dimension of the manifold increases and their reusability does not justify the cost of their creation and storage.

Both PRISM and ILDM-based methods have a problem of trying to *a priori* estimate the accessed space. Performance benefits are lost due to many calculations, either of the ILDMs or surrogate models, in parts of the composite space that is not frequently accessed. On the other hand, database techniques described in the next section are applied on-line during the application and make no assumptions about the accessed space. For more details about ILDM, see references [13, 15, 22, 24].

2.3 Existing Algorithms for Scientific Databases for Function Approximation

In-Situ Adaptive Tabulation (ISAT) [11, 25] is a tabulative technique that allows the approximation of computationally expensive functions by archiving previously computed exact values. DOLFA is an extension of some of the algorithms implemented in ISAT, which addresses issues such as dynamic cleaning the database and guaranteed retrieval. In this section we give an overview of existing algorithms developed for the ISAT implementation since most of these algorithms are the basis of algorithms used in DOLFA, and discuss the accuracy requirements for the function approximation.

2.3.1 Computational Accuracy

The function approximation returned by the database software should be within a user-specified range of accuracy. Let τ denote such a user-specified tolerance. If we denote by $\boldsymbol{F}_{\boldsymbol{\xi}}(\boldsymbol{\theta})$ the approximate function value at point $\boldsymbol{\theta}$, computed with the function previously calculated for point $\boldsymbol{\xi}$, and let $\boldsymbol{F}(\boldsymbol{\theta})$ be the true function value, our goal is to ensure the following:

$$\|\boldsymbol{F}(\boldsymbol{\theta}) - \boldsymbol{F}_{\boldsymbol{\xi}}(\boldsymbol{\theta})\| \leq \tau. \tag{2.6}$$

Therefore, for every point $\boldsymbol{\xi}$ that is already in the database, we would like to compute the region of accuracy $Reg(\boldsymbol{\xi})$, such that for every point $\boldsymbol{\theta}$ that belongs to this region, $\boldsymbol{\theta} \in Reg(\boldsymbol{\xi})$, equation (2.6) holds. Because we do not know the function value $\boldsymbol{F}(\boldsymbol{\theta})$, we have to estimate this region of accuracy with available information in a manner that is affordable to compute. If we denote the difference vector $\boldsymbol{\theta} - \boldsymbol{\xi}$ by $\boldsymbol{\Delta}$ and a Jacobian matrix as $\boldsymbol{J}$, in the case of the linear approximation where $\mathbf{F}_{\boldsymbol{\xi}}(\boldsymbol{\theta}) = \mathbf{F}(\boldsymbol{\xi}) + \boldsymbol{J}(\boldsymbol{\xi})\boldsymbol{\Delta}$, the accuracy condition leads to the following requirement:

$$\|\mathbf{F}(\boldsymbol{\xi}) - \mathbf{F}_{\boldsymbol{\xi}}(\boldsymbol{\theta})\| \leq \tau, \tag{2.7}$$

which is equivalent to:

$$\|\boldsymbol{J}(\boldsymbol{\xi})\boldsymbol{\Delta}\| \leq \tau, \tag{2.8}$$

$$\Leftrightarrow$$

$$\boldsymbol{\Delta}^T \boldsymbol{J}(\boldsymbol{\xi})^T \boldsymbol{J}(\boldsymbol{\xi})\boldsymbol{\Delta} \leq \tau^2, \tag{2.9}$$

$$\Leftrightarrow$$

$$\boldsymbol{\Delta}^T \boldsymbol{A}\boldsymbol{\Delta} \leq 1, \tag{2.10}$$

where $\boldsymbol{A}$ is equal to $\boldsymbol{J}^T\boldsymbol{J}/\tau^2$. The matrix $\boldsymbol{A}$ is symmetric and positive semi-definite, and we denote its eigenvalue decomposition with $\boldsymbol{Q}^T\boldsymbol{\Sigma}\boldsymbol{Q}$. Because $\boldsymbol{A}$ may be singular, we modify $\boldsymbol{\Sigma}$ by setting the zero eigenvalues to some predetermined value. Since a zero eigenvalue corresponds to a direction where the function does not change locally, if the predetermined value is small enough, this modification should not affect the accuracy of the computation. If we denote a modified eigenvalue decomposition of $\boldsymbol{A}$ by $\boldsymbol{Q}^T\boldsymbol{\Lambda}\boldsymbol{Q}$, we define a region of accuracy with the

following relation [11]:

$$\boldsymbol{\Delta}^T \boldsymbol{Q}^T \boldsymbol{\Lambda} \boldsymbol{Q} \boldsymbol{\Delta} \leq 1. \tag{2.11}$$

Geometrically, this region of accuracy is a hyperellipsoid that we call the *ellipsoid of accuracy* (EOA) centered at point $\boldsymbol{\xi}$.

In the case of the constant approximation (when the Jacobian is not available), we artificially introduce the EOA and represent it by the sphere centered at $\boldsymbol{\xi}$, with radius $r = \tau$. Because we have no knowledge of the Jacobian matrix, we do not know how fast the function is changing in any direction and we set strict restrictions our tolerance. The EOA is only an approximation to the region for which $\|\mathbf{F}(\boldsymbol{\theta}) - \mathbf{F}_{\boldsymbol{\xi}}(\boldsymbol{\theta})\| \leq \tau$. Therefore, the EOAs are not static objects—we allow them to change dynamically during the computation to approximate the true region of accuracy more precisely as additional information is obtained on the accuracy of the local approximation.

In the next section we describe algorithms that are implemented in software systems ISAT and DOLFA. The term *on-line* emphasizes the fact that each query issued to the database must be processed in order, without detailed knowledge of future queries. To describe this aspect of the problem, Pope uses the term *in-situ* [11].

2.3.2 Scientific Database Algorithms

For each point $\boldsymbol{\xi}$ that is stored in the database, we also store the computed function value at $\boldsymbol{\xi}$, $\mathbf{F}(\boldsymbol{\xi})$, and a diagonal scaling matrix $\boldsymbol{\Lambda}(\boldsymbol{\xi})$ that is used to determine the length of the axis of EOA centered at $\boldsymbol{\xi}$. In addition, in the case of a linear approximation approach, the Jacobian matrix $\boldsymbol{J}(\boldsymbol{\xi})$ and an orthogonal rotation matrix $\boldsymbol{Q}$ that determines the axes of the EOA are also

stored, as shown in equation (2.11). When $\mathbf{F}(\boldsymbol{\theta})$ is to be calculated for some new point $\boldsymbol{\theta}$, the database is queried to check whether this value can be approximated with some previously calculated value.

Because the database can be large, all the points in the database cannot be traversed; therefore, more effective schemes must be employed. To solve this problem, Pope [11] introduced a spatial decomposition into convex regions (polytopes) that can be efficiently searched. To accomplish a feasible space decomposition, a Binary Space Partition (BSP) tree is used. In Pope's implementation (ISAT software system), each polytope is associated with one database point $\boldsymbol{\xi}$ [25]. When the region is located in which the query point $\boldsymbol{\theta}$ lies, we check to see if it belongs to $EOA(\boldsymbol{\xi})$. If it does, we calculate an approximation to $\mathbf{F}(\boldsymbol{\theta})$, which we denote as $\mathbf{F}_{\boldsymbol{\xi}}(\boldsymbol{\theta})$.

The problem with this approach is that the EOAs do not conform to the polytope defined by the BSP tree. Fig. 2.2 illustrates this fact. In the convex region shown on this figure we can see that, although $\mathbf{F}(\boldsymbol{\theta})$ cannot be approximated with $\boldsymbol{F}(\psi)$, it can be approximated with $\boldsymbol{F}(\boldsymbol{\xi})$. However, since $\boldsymbol{\theta}$ and $\boldsymbol{\xi}$ lie in different regions, the point $\boldsymbol{\xi}$ will never be inspected by the original ISAT algorithm. This problem becomes more prevalent when using the linear approximation in a higher-dimensional space and with functions with a high degree of nonlinearity because their associated EOAs become very eccentric.

To partially solve this problem, Pope produces multiple spatial decompositions using several BSP trees. However, this approach still does not guarantee that point $\boldsymbol{\xi}$ will be found. It would be convenient if the list of all possible points that can be used for approximation of $\boldsymbol{F}(\boldsymbol{\theta})$ could be made. We can see from Fig. 2.2 that the list of possibly suitable points $\mathbf{L} = \{\boldsymbol{\xi}\}$ that we may want to use to approximate $\boldsymbol{F}(\boldsymbol{\theta})$ is the list of points whose EOAs partially or totally

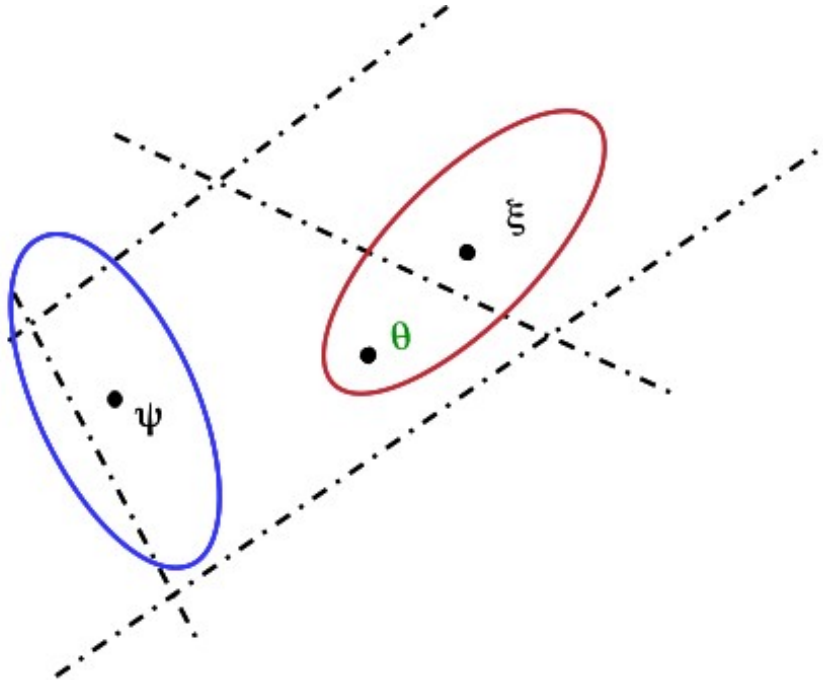


Fig. 2.2. The EOAs do not conform to the convex regions defined by the spatial decomposition of the BSP tree

intersect the region in which $\boldsymbol{\theta}$ lies. Using this fact, we modify the BSP decomposition of the accessed space for DOLFA. This way we are able to guarantee that, if $\boldsymbol{F}(\boldsymbol{\theta})$ can be approximated with some point $\boldsymbol{\xi}$, the point $\boldsymbol{\xi}$ will be found.

Both in ISAT and DOLFA, the database is organized as a BSP tree; internal nodes represent planes in the N-dimensional space and terminal nodes (leaves) represent polytope determined by the planes on the path from the given terminal node to the root of the BSP tree. This structure is illustrated in Fig. 2.3. As we mentioned before, while in ISAT each region is associated with only one point, in DOLFA for each region we create a list of database points whose EOAs intersect the region [9]. To maintain the list of EOAs that intersect a specific region, it is crucial to develop efficient algorithms that can determine which regions a given EOA intersects. These algorithms are used either when we design a new $\mathrm{EOA}(\boldsymbol{\theta})$—when we add point $\boldsymbol{\theta}$ to the database, or when we change an existing EOA by performing shrink or grow operations. In addition, to avoid storing points that are unlikely to be used for approximation again, we employ new techniques based on usage statistics that allow us to remove such points from the database.

Hence, there are two major directions for performance improvement:

- Increase the retrieval rate by keeping a list of EOAs that intersect a given region instead of only one point that belongs to the region;
- Make the database search more efficient by decreasing the database size by periodically deleting unused data and rebalancing the BSP tree.

Algorithm 2.1 presents an outline of the implementation of our scientific database (neglecting the periodical database cleaning, which will be addressed later).

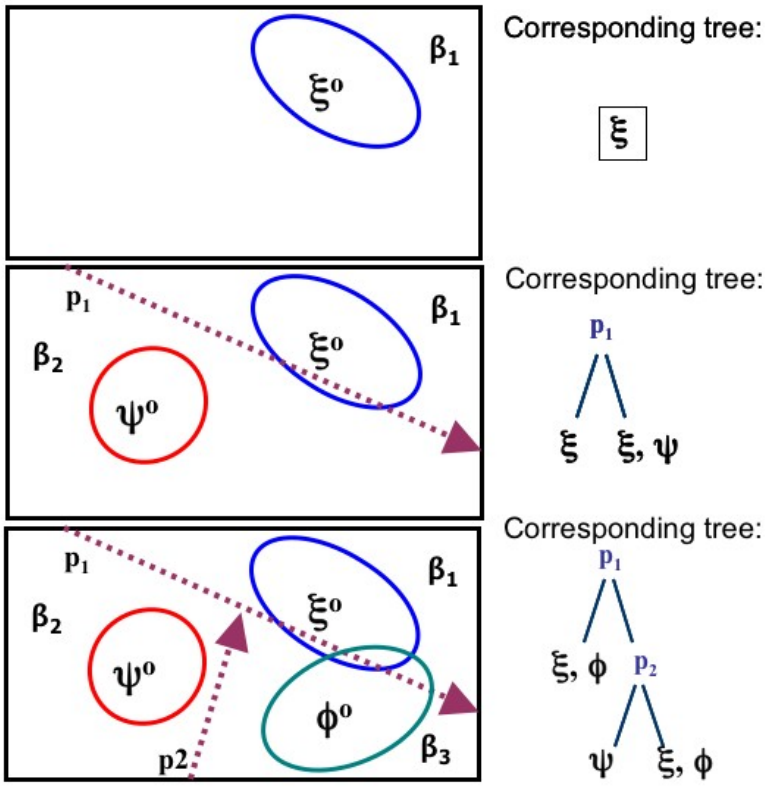


Fig. 2.3. A two-dimensional schematic of a spatial decomposition by a BSP tree. The arrows determine the orientation of the planes.

ALGORITHM 2.1 A scientific database algorithm

BEGIN

Let $\boldsymbol{\theta}$ be the new query point

if (database empty)

Compute and add $\boldsymbol{F}(\boldsymbol{\theta})$, $EOA(\boldsymbol{\theta})$, and $\boldsymbol{\theta}$ to the database

else

Find a region T such that $\boldsymbol{\theta} \in T$

Let **L** be the list of points $\{\boldsymbol{\xi}\}$ such that $EOA(\boldsymbol{\xi}) \cap T \neq \emptyset$

foreach ($\boldsymbol{\xi} \in$ L)

if($\boldsymbol{\theta} \in EOA(\boldsymbol{\xi})$)

Approximate $\boldsymbol{F}(\boldsymbol{\theta})$ with $\mathbf{F}_{\boldsymbol{\xi}}(\boldsymbol{\theta})$

if (TimeToCheck)

Check whether to shrink EOA

endif

return

endif

endfor //*we could not approximate function*

Calculate $\boldsymbol{F}(\boldsymbol{\theta})$

Let $\boldsymbol{\psi}$ be the EOA for which the distance between $\boldsymbol{\theta}$ and EOA($\boldsymbol{\psi}$) is minimal

Check whether we can grow EOA($\boldsymbol{\psi}$)

Add the point $\boldsymbol{\theta}$ to the database

endif

END

2.3.3 Database Update

In this section we describe algorithms for updating the database. Also, we introduce a way to monitor the error in the case when some EOAs accept points that do not yield a satisfactory approximation error. This monitoring was not present in the original approach proposed by Pope [11].

Retrieval and Shrink. If we find a point $\boldsymbol{\xi}$ such that $\boldsymbol{\theta} \in EOA(\boldsymbol{\xi})$, we calculate $\mathbf{F}_{\boldsymbol{\xi}}(\boldsymbol{\theta})$ using a linear or constant approximation. With the linear approximation, the retrieval cost is $O(N^2)$, while with the constant approximation the cost is $O(N)$. Occasionally, we force the direct calculation of $\boldsymbol{F}(\boldsymbol{\theta})$ and use this value to compute the true error. If the true error is greater than the user-defined tolerance, $\|\mathbf{F}_{\boldsymbol{\xi}}(\boldsymbol{\theta}) - \boldsymbol{F}(\boldsymbol{\theta})\| \geq \tau$, we have underestimated the error in our model. If we denote $\boldsymbol{\chi}$ with $\boldsymbol{Q} \cdot (\boldsymbol{\theta} - \boldsymbol{\xi})$ (in the constant case $\boldsymbol{\chi} = \boldsymbol{\theta} - \boldsymbol{\xi}$), we want to compute new eigenvalues $\boldsymbol{\Lambda_{new}}$ such that $\boldsymbol{\xi}^T \Lambda_{new} \boldsymbol{\xi} = 1 + \delta$, where δ is a predetermined parameter. If the distance from the point to the EOA was ν, namely $\boldsymbol{\xi}^T \Lambda \boldsymbol{\xi} = \nu$, the simplest formula to compute new eigenvalues is $\boldsymbol{\Lambda_{new}} = c \cdot \boldsymbol{\Lambda}$ where $c = \frac{1+\delta}{\nu}$. To avoid a large number of accuracy checks, we use a random number generator to determine when to perform an additional direct calculation. Not more than 1% of all retrieves should be checked for accuracy—every time we retrieve a point, we generate a random number and only if the number is less than 0.01 we perform the accuracy check.

Growing the EOA. If we do not find a suitable point for the approximation, we calculate $\boldsymbol{F}(\boldsymbol{\theta})$ directly. For every point $\boldsymbol{\xi}$ in the list associated with region T, with $\boldsymbol{\theta} \in T$, we check whether $\|\boldsymbol{F}(\boldsymbol{\theta}) - \mathbf{F}_{\boldsymbol{\xi}}(\boldsymbol{\theta})\| < \tau$. If this is true, we have overestimated the error in our model and we should grow the $EOA(\boldsymbol{\xi})$ to accept the point $\boldsymbol{\theta}$. If we denote $\boldsymbol{\chi} = \boldsymbol{Q} \cdot (\boldsymbol{\theta} - \boldsymbol{\xi})$

(in the constant case $\boldsymbol{\chi} = \boldsymbol{\theta} - \boldsymbol{\xi}$), we want to compute new eigenvalues $\boldsymbol{\Lambda_{new}}$ such that $\boldsymbol{\xi}^T \Lambda_{new} \boldsymbol{\xi} = 1 - \delta$, where δ is a predetermined parameter. If the distance from the point to the EOA was ν, namely $\boldsymbol{\xi}^T \Lambda \boldsymbol{\xi} = \nu$, the simplest formula to compute new eigenvalues is $\boldsymbol{\Lambda_{new}} = c \cdot \boldsymbol{\Lambda}$ where $c = \frac{1-\delta}{\nu}$. However, since the EOAs are only the approximation of the real region of accuracy, we should be careful when growing the EOA and c should not be too large. This is especially the case for the constant approximation because we have no information about the Jacobian and the local directional rate of change of the function we are calculating.

Adding New Points to the Database. After calculating $\boldsymbol{F}(\boldsymbol{\theta})$ and $EOA(\boldsymbol{\theta})$ we add the point $\boldsymbol{\theta}$ (together with other items needed) to the database. In the original implementation of ISAT, every region was associated with only one point. Therefore, introducing a new point into the database would cause the splitting of the region to which it belongs into two regions, each containing one point with one EOA. However, this heuristic can lead to extremely unbalanced BSP trees, thus making the search for the appropriate point/region inefficient. In our implementation, we first insert $\boldsymbol{\theta}$ into the list of points associated with the region T that $\boldsymbol{\theta}$ belongs to. If the list of points is too long (longer than some predetermined parameter), inserting $\boldsymbol{\theta}$ to the list will cause the splitting of T into two subregions, T_{l} and T_{r}. The region T is split by determining a cutting plane as described next in this section. In the BSP tree, we replace T with an internal node P representing the cutting plane. The left child of P is T_{l} and the right child of P is T_{r}.

Determining a Cutting Hyperplane. For a given region T, if a vector $\boldsymbol{\theta}$ is in the list of points that belong to the region T, we know that EOA($\boldsymbol{\theta}$) intersects this region. However, point

$\boldsymbol{\theta}$ may or may not be in the region. When we are determining the cutting plane that will split region T into two parts, we want to consider only the points that are in this region. Moreover, a cutting plane that we design should have the property that the minimal number of EOAs that belong to a given region are intersected with the cutting plane. However, since this operation is performed frequently, it should be simple and efficient. For a given region, a cutting plane is a certain coordinate plane that passes through a certain point in the region that: (a) is easy and fast to compute, and (b) does not intersect a large number of EOAs that belong to the region. Limiting cutting planes only to coordinate planes reduces the search space. Also, with coordinate planes the operation of determining on which side of the cutting plane the given query point lies is of constant cost. The cost of the same operation with cutting planes that are generally positioned is O(n), where n is the size of the accessed space. Moreover, calculating a cutting plane that is not limited to coordinate planes but also yield a reasonably small number of EOAs it intersects can be an extremely costly operation, since the calculation of the intersection of EOA and given plane is not a trivial algorithm.

Therefore, given a set of n points that are in the given region, we compute a mean of this set, $\langle \boldsymbol{M}_n \rangle = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{\xi}_i$. Then, we determine the *maximal variance direction* j for which we have the following conditions satisfied:

- maximal variance $\sum_{i=1}^{n} (\boldsymbol{\xi}_j^i - \langle \boldsymbol{M}_n \rangle_j)^2$, and
- minimal difference between the number of points that have $\boldsymbol{\xi}_j < \langle \boldsymbol{M}_n \rangle_j$ and the number of points that have $\boldsymbol{\xi}_j \geq \langle \boldsymbol{M}_n \rangle_j$.

The cutting plane we use goes through the mean point $\langle M_p \rangle$ and is orthogonal to the maximal variance direction to guarantee a good separation of the points.

If we calculate a new EOA for some point ξ or change the shape of the existing EOA by growing it, we have to traverse the BSP tree to identify new regions that the EOA intersects and add ξ to their lists. When shrinking the EOA, we must identify the regions that the EOA no longer intersects, and delete the EOA from the lists corresponding to these regions.

2.4 New Algorithms Implemented in DOLFA

In this section, we describe new algorithms introduced with DOLFA [9] as a part of the original work performed for this thesis. The algorithms are divided into three general categories: new approximation schemes, new geometric algorithms, and a new algorithm for cleaning the database. In Section 2.4.1 we describe a hybrid type of the approximation, where we use a linear approximation but the EOA is axis-aligned. In Section 2.4.2 algorithms designed to determine the relationship between a given ellipsoid and a polytope are described. These algorithms are important to determine which EOAs intersect a given polytope since, in our binary space decomposition, each polytope is associated with a list of ellipsoids that intersect it. Section 2.4.3 contains details about the algorithm for cleaning of the database. Based on performance measurements, we determine when to clean the database and which points to delete from the database. After cleaning the database, we have to build the BSP tree off-line from the given set of remaining points and ensure that the tree is well-balanced.

2.4.1 The Hybrid Approach

We expect a linear approximation to be more accurate than a constant approximation. However, it also results in an EOA that is defined not only by the length of its axis but also its principal directions (i.e., the rotation matrix **Q** is not a unit matrix). This not only results in larger memory requirements, but also results in a much higher computational time for many algorithms implemented in DOLFA because of the additional matrix-vector multiplications that have to be performed. In the case of the hybrid approximation we calculate the Jacobian **J** and use a linear approximation whenever we have to approximate the function value. However, the EOA is calculated in a different way—instead of performing a costly eigenvalue decomposition, the axes of the EOA are assumed to be axis aligned (the Q matrix is a unit matrix) and Λ are diagonal elements of the $\boldsymbol{J}^T\boldsymbol{J}$ matrix divided by the squared tolerance. A comparison of the performance of the linear, constant and hybrid approximation for various types of simulations is provided in Section 2.5.

2.4.2 Computing Ellipsoid/Polytope Intersections

In this section, we use a transformation from the original space **C** to the space $\mathbf{C}_{\boldsymbol{\xi}}$, where the given $EOA(\boldsymbol{\xi}) = (\boldsymbol{\theta} - \boldsymbol{\xi})^T Q^T \Lambda Q(\boldsymbol{\theta} - \boldsymbol{\xi})$ is represented as a unit sphere S centered at the coordinate origin.

Transformations of the Composite Space. Transformations that are needed to accomplish this task are as follows:

1. translation of EOA($\boldsymbol{\xi}$) to E_1, to get ellipsoid E_1 centered at the coordinate origin,
2. rotation of E_1 to E_2, to get ellipsoid E_2 that is axis aligned, and

3. scaling E_2 to E_3, to get E_3 as a unit sphere.

If we have a plane α_s: $a^T x + b = 0$ in space **C**, the following transformations have to be performed.

1. With translation the normal vector does not change, i.e., $a_1 = a$, the free coefficient becomes $b_1 = -a_1^T x_1 = -a^T(x + \boldsymbol{\xi}) = b + a^T \boldsymbol{\xi}$.
2. With rotation the normal vector is transformed, $a_2 = Qa_1$, but the free coefficient does not change,i.e., $b_2 = -a_2^T x_2 = -(Qa_1)^T (Qx_1) = -a_1^T x_1 = b_1$.
3. With scaling, the normal vector is transformed, $a_3 = \Lambda^{-\frac{1}{2}} a_2$ and $b_3 = -a^T x_3 = -(\Lambda^{-\frac{1}{2}} a_2)^T (\Lambda^{\frac{1}{2}} x_2) = -a_2^T x_2 = b_2$.

We note that the cost of this operation is $O(N)$ for the constant and the hybrid approximation case and $O(N^2)$ for the linear approximation, where N is the dimension of the composite space.

Determining Whether the $EOA(\boldsymbol{\xi})$ Intersects a Plane α. When we transform from **C** to $\mathbf{C}_{\boldsymbol{\xi}}$, the question "Does the EOA intersect plane α?" becomes "Is the free coefficient of transformed plane $\alpha_{\boldsymbol{\xi}} \leq 1$?". Given the plane α, this algorithm also computes halfspaces (left, right, or both), to which the EOA belongs. This algorithm is used in the two cases described below.

- When we split the region T into two subregions with a cutting plane, we must also split the list of points whose EOAs intersect the region. To determine which EOA belongs to what subregion, for each point $\boldsymbol{\xi}$ in the list associated with T we check whether $\mathrm{EOA}(\boldsymbol{\xi})$ intersects the cutting plane.

- When we shrink the $\mathrm{EOA}(\boldsymbol{\xi})$. For every EOA we maintain a list $\{\mathrm{T}\}$ of regions it intersects. If, for a given region T, $\boldsymbol{\xi} \in T$ then $\mathrm{EOA}(\boldsymbol{\xi})$ belongs to this region. But if $\boldsymbol{\xi} \notin T$, we check for every plane α_i that determines a border of *T*, whether the EOA lies (partially or entirely) in the halfspace in which region *T* lies.

Determining Whether the EOA Intersects a Region T. If we transform space **C** into $\mathbf{C}_{\boldsymbol{\xi}}$, this problem is reduced to determining whether the unit sphere S centered at the origin belongs to the transformed polytope P, $P \equiv T_{\boldsymbol{\xi}}$. To perform this check, we construct the polytope P^* by translating planes which determine the polytope P. We translate the planes by a unit normal vector in directions outward from the polytope P. We observe the following: **if** the sphere S intersects the polytope P, **then** the origin belongs to the polytope P^*. These statements are not equivalent, as we observe from Fig. 2.4. Here we can see that sphere S is erroneously identified to intersect the region. Nevertheless, in practice our approximation does not produce a large overhead of false positive answers.

Starting from the root of the tree, we perform a depth-first search. In the case when the tree is too deep, we start from arbitrary depth and traverse only a subtree. This reduces time needed to determine which polytope a given ellipsoid intersects. However, in this partial search, some polytopes may be omitted. We translate the plane (represented by the inner node of the BSP tree) by unit normal vectors in the left direction. If the origin lies on the right side of this translated plane, we search the right subtree. Then we repeat the procedure, translate planes in the right direction, and traverse the left subtree if required.

To prevent the lists of points for a given region becoming too large, we employ a cleaning strategy of these lists. When we search the list $\{\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \boldsymbol{\xi}_3, ...\}$ in region T for a suitable

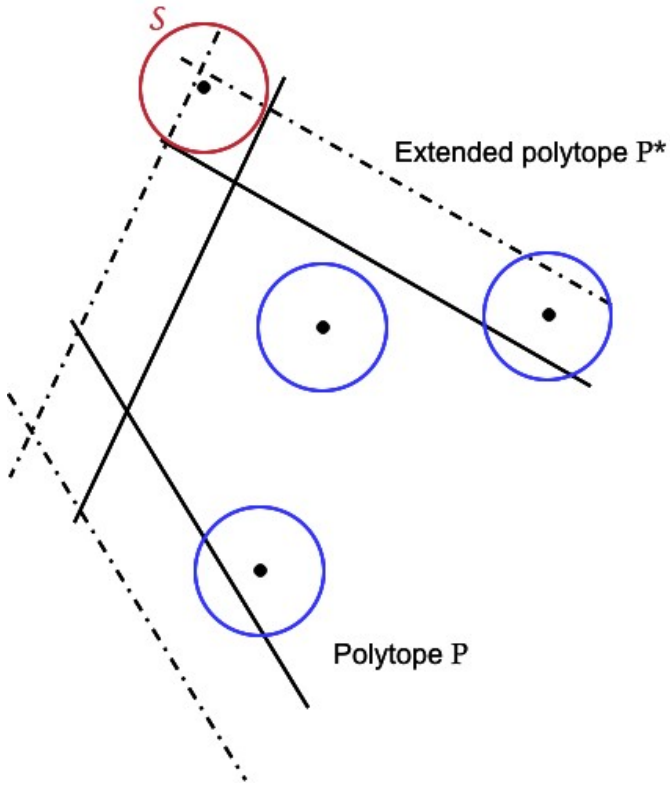


Fig. 2.4. Polytopes P and P^* and different sphere positions for which the algorithm computes an intersection with the polytope P

point to approximate $\boldsymbol{F}(\boldsymbol{\theta})$, we check how many times the point $\boldsymbol{\xi}$ was used for approximations in T, and how many times it was used overall. If it has not been used inside this region, but it has been used several times overall (through other regions the EOA intersects), we delete $\boldsymbol{\xi}$ from T's list.

2.4.3 Dynamical Database Algorithms

In this section we describe algorithms for maintaining a minimal database size by periodically deleting points from the database. The user can either choose to delete points which are not likely to be used for approximations in future queries or to delete the entire database.

Cleaning of the Database. This algorithm is performed periodically during the simulation. For each point in the database, we keep track of when this point was last used to satisfy a database query. During the cleaning procedure, if it is determined that the point has not been used recently, we delete it from the database. This approach is important in transient computations because the queries for non-steady state problems move around the composite space. This algorithm is referred to as a "periodical clean-up." There are two user-specified parameters associated with periodical clean-up—`CleanPoint` and `CleanDelete`. We perform this database cleaning after every `CleanPoint` queries. If the underlying problem involves a grid structure imposed on some region in space, `CleanPoint` should be proportional to the number of grid points. In addition, we delete all the points not accessed during the last `CleanDelete` queries. The cost of this operation is linear in the number of points stored in the database. Then we have to rebuild the BSP tree, since many regions will contain no points or a very small number of points,

which may lead to highly unbalanced BSP trees. We start this process with one large region that encompasses all the points in that are left in the database. Then we recursively divide the region into smaller subregions by calculating cutting planes as described in section 2.3.3.

Deleting the Database. The clean-up operation can be expensive, especially if many points are deleted from the database. In that case, building a BSP tree off-line out of the remaining points can be a very costly operation. In the case when more than 30% of points are left in the database, the entire database is deleted. This way the database may loose a significant number of points that can be used for approximation. However, the procedure can save time by avoiding the building of a BSP tree out of a large number of points that remained in the database. Alternatively, the user has a choice to enforce deleting of the entire database whenever the clean-up is needed.

2.5 Computational Results

There is a number of factors that can influence the choice of various parameters used for coupling DOLFA with a combustion simulation. The first factor is homogeneity of the medium and the second factor is the time dependence of the application. If the medium is nearly homogeneous, the retrieval rates can be very high since the query points do not vary much over the medium. Similarly, if the medium is strongly nonhomogeneus, we expect retrieval rates to be significantly lower, since the values of the query points are more distributed over the composite space. If the application is steady-state, the advantage of using DOLFA can be tremendous, since the queries repeat in the course of the simulation. Since the composition of the mesh elements

do not change with time, after some number of iterations when convergence is almost achieved, the retrieval from DOLFA can be close to 100%. However, the problem with this type of simulation is that cleaning the database can significantly slow down the performance of the software. Because of the steady-state nature of this type of simulationxs, the points that are not reused for a long time can still be reused later during the application. Therefore, cleaning of the database should not be employed with this type of simulation. If the problem we are modeling is transient, the accuracy of the approximation must be closely monitored. Time-dependent simulations are usually much more susceptible to numerical instabilities and therefore, the demanded error tolerance is usually stricter. However, cleaning the database can be useful in transient cases, since the queries do not repeat and are usually useful only for a few time-steps.

Due to the lack of a precise theoretical estimate of the approximation error and the instability that can be caused by the wrong estimate of the approximation tolerance, it is recommended that the user first runs a particular simulation not using DOLFA. This simulation can be run on a coarser mesh and with a simplified chemical reaction mechanism. The user should also determine what is a suitable tolerance error and if a constant, linear or a hybrid type of approximation should be used. This is usually based on the properties of the problem and numerical stability of the calculations. Using these parameters, the user can then run other simulations of the given problem type solely using DOLFA. Therefore, DOLFA is most suitable for users that run parametric analysis of their simulations, such as testing the influence of different initial and boundary conditions or test different chemistry, turbulence or radiation models with a particular type of problem.

2.5.1 Performance of the DOLFA Library Compared to the ISAT Library

In this section we compare the performance of the DOLFA library to that of the version of the ISAT library available to us, released in August 2000. We tested the sequential version of the DOLFA software with a code that implements a detailed chemistry mechanism in an HCCI piston engine [26, 27]. This is a transient case with a homogeneous medium—it is not as numerically instable as are problems with nonhomogeneous media. The test code uses an n-heptane mechanism of 40 species and 165 reactions with the composite space reduced to 7 independent variables and computed on a 100-cell mesh. This problem runs through 60 crank-angle degrees, from 30 degrees before piston top-dead-center to 30 degrees after. The n-heptane mechanism is implemented using the CHEMKIN library [17]. As an ODE solver, the robust stiff ODE solver VODE was used [28–30].

Tests were performed on a computer with a 2GHz Intel Xeon CPU and 3GB of main memory. The constant approximation is employed in the database with $\epsilon_{abs} = 10^{-2}$ and $\epsilon_{rel} = 10^{-2}$. With these parameters the tolerance becomes a function of a database point as $\tau(\boldsymbol{\xi}) = \epsilon_{abs} + \epsilon_{rel}\|\boldsymbol{F}(\boldsymbol{\xi})\|$. Using this test-code we also compared the performance of the DOLFA software to ISAT (released in August 2000) [25].

Using direct integration, (i.e., without the use of the database), the program takes approximately 15 minutes to complete. 90% of this time is spent performing function evaluations. Employing the DOLFA database with constant approximation, it takes about 90 seconds of wall-clock time to run. Out of this time, approximately 8.6 seconds is spent performing database operations and function evaluations. Employing the DOLFA database with linear approximation,

it takes about 160 seconds of wallclock time to run. Of this time, approximately 79 seconds is spent doing database operations and function evaluations.

A standard way to verify the accuracy of these results is to compare the computed temperature and pressure as a function of the Crank Angle Degree (CAD). These results, presented in Fig. 2.5, demonstrate that the function values computed by the database have an insignificant effect on the accuracy of the solution. However, the standard error analysis is not helpful in this case. When the combustion starts, the approximated temperature profile is shifted one time step compared to the temperature profile from the direct calculations. As can be observed in Fig. 2.6, the relative error of the temperature profile is very high (up to 18%), even though the temperature profiles are almost undistinguishable (Fig. 2.5). However, instead of calculating the relative error, we calculate the shifted relative error when the shift in the temperature field occurs. More precisely, instead of calculating relative error as $\epsilon = abs((\tilde{T}_{approx} - T)/T)$, we calculate the shifted relative error by partially shifting the temperature profile, $\epsilon_s = abs((\tilde{T} - T_s)/T_s)$. We then observe that the maximum relative error of the approximation is around 4%.

In Fig. 2.7 we plotted cumulative time for database operations for ISAT and DOLFA, with constant or linear approximation. The difference in running times results from the smaller number of direct calculations preformed in DOLFA than in ISAT for same problem parameters. In the case of constant approximation, DOLFA has 7% less direct calculations and in the linear case, DOLFA achieves 5% less direct calculations. This is due to the improved BSP searching strategy.

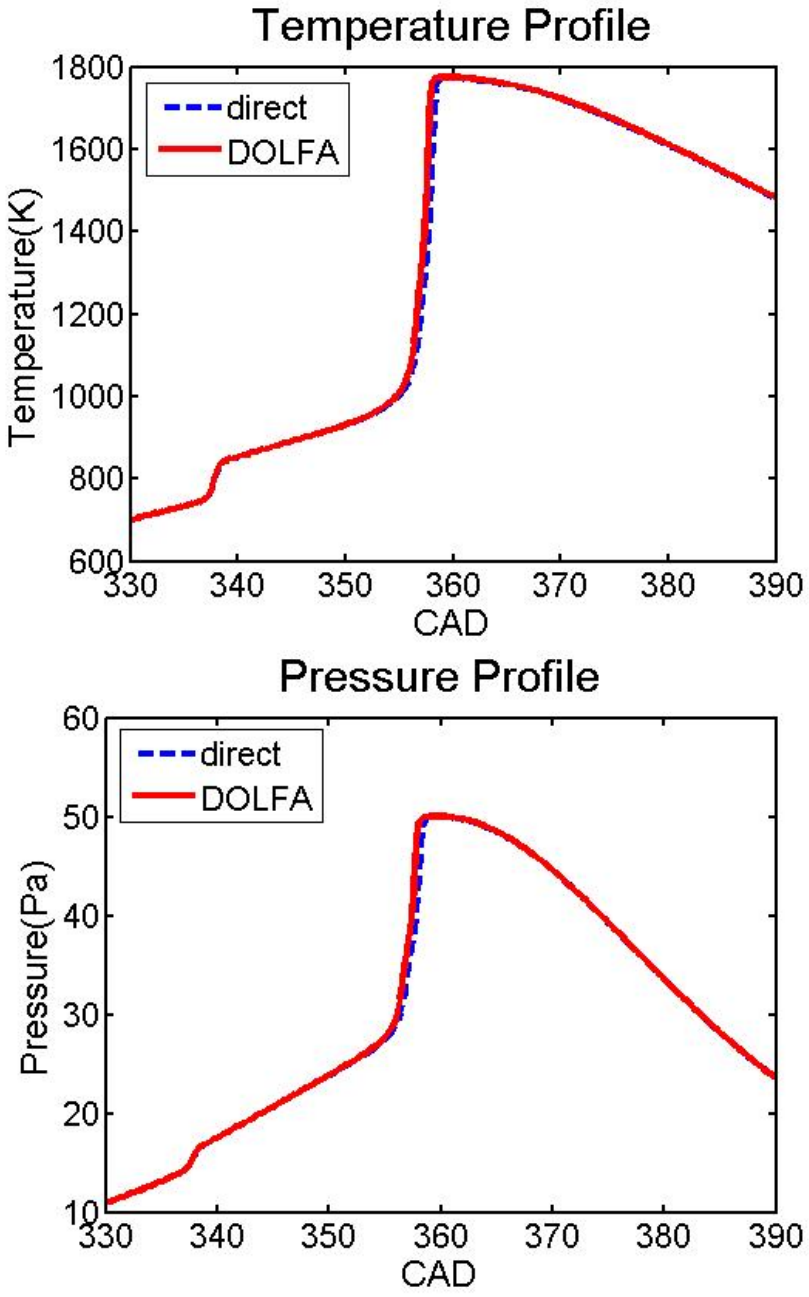


Fig. 2.5. Experimental results comparing the pressure (top) and the temperature (bottom) as a function of CAD for direct calculation (dashed line) and the DOLFA software (solid line)

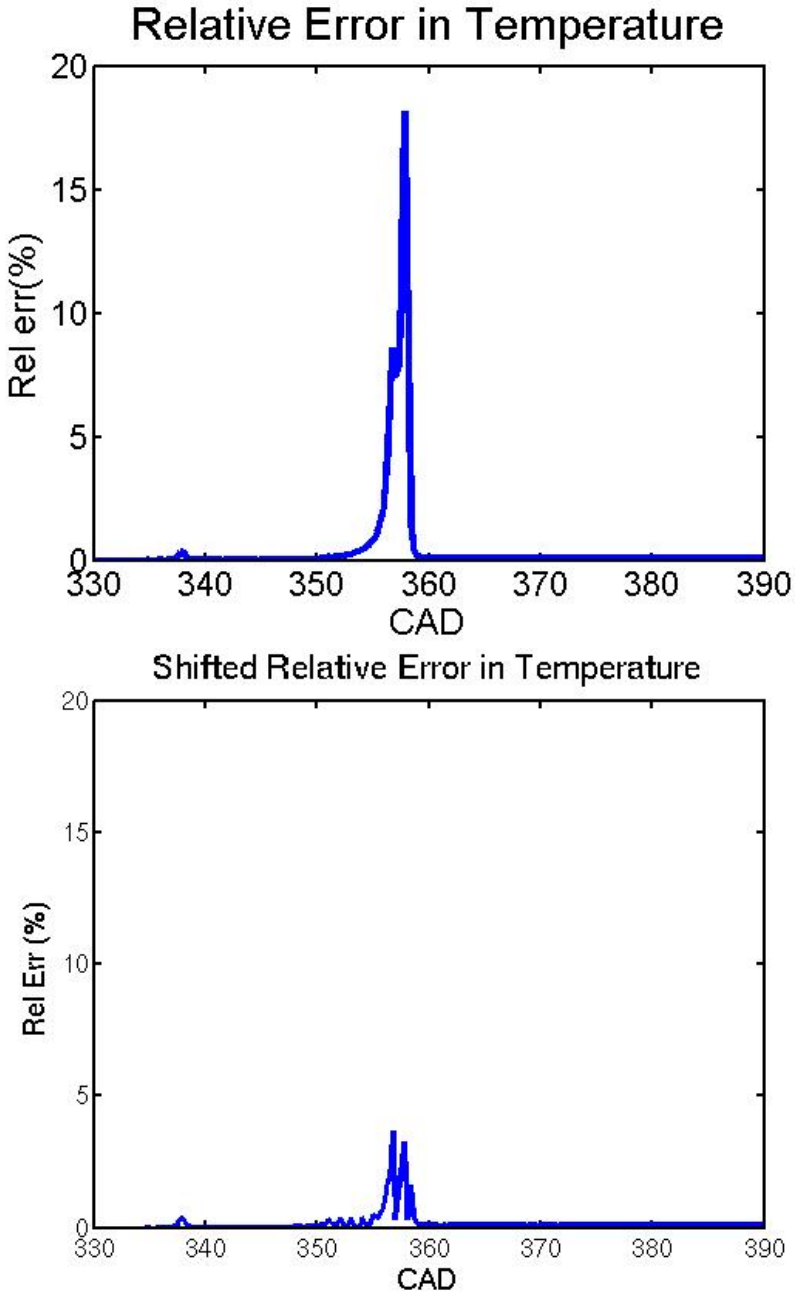


Fig. 2.6. Relative error (top) and shifted relative error (bottom) in the temperature profile showing the inadequacy of applying the classical relative error to asses the accuracy of DOLFA.

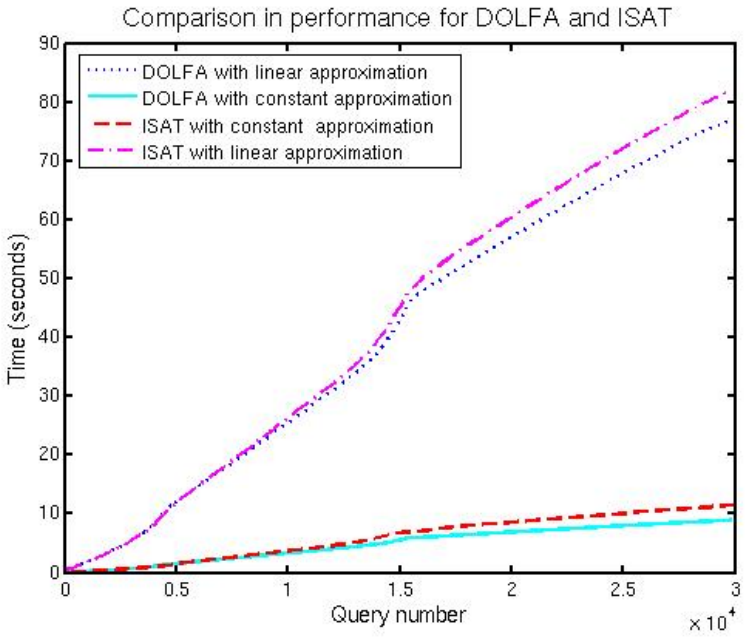


Fig. 2.7. Comparisons of the CPU time spent doing database operations for ISAT using one BSP tree (dashed line for constant approximation and dash-dot line for linear approximation) and DOLFA (solid line for constant approximation and dotted line for linear approximation)

2.5.2 Comparison of the Approximation Methods on Various Types of Applications

In this section we compare the performance of the constant, linear and hybrid approximation scheme for three representative problems. Depending on the type of the approximation, for a given tolerance we can have different quality of the approximation. For most of the problems, to achieve a certain level of accuracy, the error tolerance for the constant approximation has to be tighter than for the linear and hybrid approximation. The tolerance level that enables DOLFA to achieve the 4% of shifted relative error in temperature profile is chosen for each type of application and for each type of approximation method.

The first problem is a simulation of an HCCI engine mentioned in the previous section. This problem is transient and homogeneous, and is very stable with respect to the numerical perturbations that can be introduced by the DOLFA simulation. Hence, the absolute and relative tolerances can be liberal. Also, the total number of queries is relatively small, so there is no advantage to applying a dynamical cleaning of the database. A finite–difference method is used to calculate the Jacobian matrix. Performance results for this case are given in Table 2.1. We observe that, in the case of the HCCI engine, the constant approximation yields superior performance, both in the number of approximated function values and the CPU time.

Type of approximation	Error tolerance	Number of retrieves	Retrieval rate	Cumulative time
Direct integration	0	0	0%	867.5 s
Constant approximation	1.0e-2	30012	99.05 %	9.03 s
Linear approximation	1.0e-2	30000	99.01 %	79.2 s
Hybrid approximation	1.0e-2	30000	99.01 %	79.0 s

Table 2.1. Performance results for simulating the HCCI engine with different approximation schemes

The second problem is a simulation of an DI (Direct Injection) compression-ignition reciprocating-piston engine with complex chemistry (40-species, n-heptane/air mechanism). Similar to the HCCI problem, this simulation is transient, but since it is nonhomogeneous, it is much more sensitive to the numerical perturbances that can be introduced with DOLFA. Therefore, the tolerance used in this case is stricter. A finite–difference method is used for calculation of the Jacobian matrix. Performance results for this case are given in Table 2.2. In the case of direct injection, we observe that there is very little advantage to using DOLFA database, due to the extreme nonhomogeneity of the medium. However, the constant approximation did not perform slower than the case where DOLFA was not used, while this simulation using DOLFA with the linear or hybrid approximations ran 10 times longer than the direct integration case. In Table 2.2, we analyze the impact of the cleaning of the database on the performance of DOLFA with the constant approximation. We can see that the cleaning algorithm or deleting of the entire database does not introduce benefits for the performance of the simulation. The number of regions is significantly smaller than in the case when the cleaning strategy was not used, but the retrieval rates are also smaller, because some points that could be used for retrieval were deleted. However, if we compare the cleaning strategy for a different transient case that is no so highly inhomogeneous, we can see that, in this case the cleaning the database was helpful. Also, we observe that the deletion of the database is detrimental as it significantly reduces the retrieval rate.

The third problem is a simulation of a statistically stationary piloted nonpremixed turbulent jet flame (Sandia D). Since the flame is stationary, we expect large retrieval rates. We did not use any cleaning strategy since it would remove points that are useful in the function approximation. An alternative method was used for the estimation of the Jacobian matrix. This method

Type of approximation	Error tolerance	Number of retrieves	Retrieval rate	Cumulative time
Direct integration	0	0	0%	4.0e+4 s
Constant approximation	1.0e-3	$7x10^5$	54%	3.4e+4 s
Linear approximation	5.0e-3	$8.6x10^5$	66%	2.4e+5 s
Hybrid approximation	5.0e-3	$8.6x10^5$	66%	2.3e+5 s

(a) Comparison of various approximation methods

Type of simulation	Retrieval rate	Cumulative time
Constant approximation, no cleaning	54%	3.4e+4 s
Constant approximation, cleaning	46%	3.6e+4 s
Constant approximation, deleting	37%	3.6e+4 s

(b) Comparison of cleaning strategies

Table 2.2. Performance results for simulating the DI engine with different approximation schemes and cleaning strategies

Type of simulation	Retrieval rate	Cumulative time
Constant approximation, no cleaning	90%	1.50e+03 s
Constant approximation, cleaning	90%	1.49e+03 s
Constant approximation, deleting	20%	1.67e+04 s

Table 2.3. Performance results for simulating the spray engine for different cleaning strategies

Type of approximation	Error tolerance	Number of retrieves	Retrieval rate	Cumulative time
Direct integration	0	0	0%	5.5e+4s
Constant approximation	1.0e-5	1.17×10^7	91.4%	2.2e+4s
Linear approximation	5.0e-3	1.16×10^7	91.5%	1.9e+4s
Hybrid approximation	5.0e-3	1.21×10^7	95.6%	1.5e+4s

Table 2.4. Performance results for simulating the jet flame with different approximation schemes

approximates the Jacobian from the sensitivity coefficients computed by the ODE solver. Performance results for this case are given in Table 2.4. In the modeling of the jet flame, we observe for the first time that there are advantages to using the hybrid approximation. Even when the Jacobian is calculated by finite–differences, the overall running time is higher, but comparable to the times when we used a constant approximation.

2.6 Conclusions

The use of scientific databases for speeding up chemistry calculations in chemically reacting flows can significantly reduce the CPU time of the simulations. The new algorithms we introduced show great potential, but are not suitable for all classes of problems that are present in combustion applications. The hybrid and linear approximations can be successfully used only if there is an efficient way to calculate the Jacobian matrix. In most of the cases, the constant approximation can be just as efficient, even though it demands stricter accuracy requirements. Cleaning of the database is only useful in highly transient cases with a large number of queries. In other cases, either many points that can be used for the function approximation are deleted,

or many points are left in the database thus making the rebuilding of the database very expensive. In the next chapter, we describe the algorithms needed for a parallel implementation of the DOLFA software.

Chapter 3

Parallel Algorithms for Scientific Databases

In developing a parallel version of the database approach discussed in the preceding chapter, the most straightforward approach would be to use the sequential version of DOLFA on every processor with no interprocessor communication. This approach would work; however, it suffers from two significant drawbacks. First, this approach cannot take advantage of direct calculations done on other processors—hence, we would expect a significant amount of redundant function evaluations. Secondly, the memory required to store the database can be very high—without partitioning the database among processors, we cannot take full advantage of the total memory available on a parallel computer. In this chapter, we present parallel algorithms that address each of these issues [10, 31, 32]. The development of these algorithms is a part of the original work performed for this thesis. The importance of addressing the first issue of redundant evaluations becomes obvious if we compare typical times for function evaluation and interprocessor communication. Namely, for the HCCI combustion simulation, which we considered in Section 2.5, one function evaluation takes 0.3 seconds (an ODE system with 40 chemical species). On the other hand, on the Beowulf cluster where our experiments were conducted the time to communicate the computed results (40 double precision floating point numbers) was approximately 2.7 microseconds. This bandwidth is typical for the distributed memory systems with a high speed interconnect. The relative difference in times is significant, but even if all the synchronization issues are taken into consideration, it is still important to design a parallel

approach that will coordinate direct calculations between processors to minimize the number of redundant direct calculations. Thus, our goal is to design a parallel implementation of DOLFA that minimizes the total number of direct calculations performed on all processors. However, this minimization must be obtained while maintaining good load balancing and minimizing the total interprocessor communication.

It is important to note that for the parallel algorithm the search space is distributed—it is created by points from the entire parallel environment and distributed among available processors. Therefore, the parallel algorithm must maintain a global BSP tree to be able to conduct an efficient search of previously computed function values. To ensure that the distributed BSP tree uses the total memory available in the parallel environment for storing its data, we partition the BSP tree over all processors. This partitioning must be constructed in a way that allows every processor i to know what part of the composite space is stored in the BSP tree belonging to another processor j.

The nontrivial part of the algorithm is to determine how to partition the function queries among processors in a manner that is both efficient and obtains good load balancing. The parallel algorithm we present in this chapter consists of three main parts. The first part describes the approach used for building the distributed (global) BSP tree; the second part defines the heuristics developed for managing interprocessor communication and the assignment of direct function evaluations; the third part details the algorithm for BSP tree redistribution for maintaining the load balance during the simulation of transient applications. In the next two sections, algorithms implemented in the parallel DOLFA framework are introduced.

3.1 Building the Global Binary Space Partition Tree

Before each call to the database from the main driver application, all processors must be synchronized to be able to participate in global database operations. This synchronization may be costly if conducted often. For example, if the parallel database is called with a single point at a time, in the same manner as sequential version of DOLFA, the overhead of inserting barriers before each call can be overwhelming. Modifying the DOLFA interface by forcing calls that use a list of query points instead of one point at a time minimizes this synchronization overhead and enables more efficient interprocessor communication. Fortunately, in combustion simulations, updates to the chemical species mass fractions happen independently of one another on the mesh subdomain assigned to each processor. More precisely, the chemical reaction at point $\boldsymbol{\xi}$ at time t does not affect the chemical reaction at point $\boldsymbol{\theta}$ at the same time t. Therefore, the points for which we want to know function values are packed into a list and simultaneously submitted to the database. Every processor receives a different list of points to work on. The list is usually determined by the domain decomposition of the fluid solver. The output is a list of function values at points given in the input list.

As with the case of sequential processing, we usually have no *a priori* knowledge of the distribution of queries within the search space. However, since the database is called with the list of points for which we want to know the function values, we have an acceptable description of initial composite space we are accessing before we start evaluating the queries. Thus, the global BSP tree can be built before we start the database operations, based on the list of points that are queried. The BSP tree can then be partitioned into unique subtrees which are assigned to processors as illustrated in Fig. 3.1.

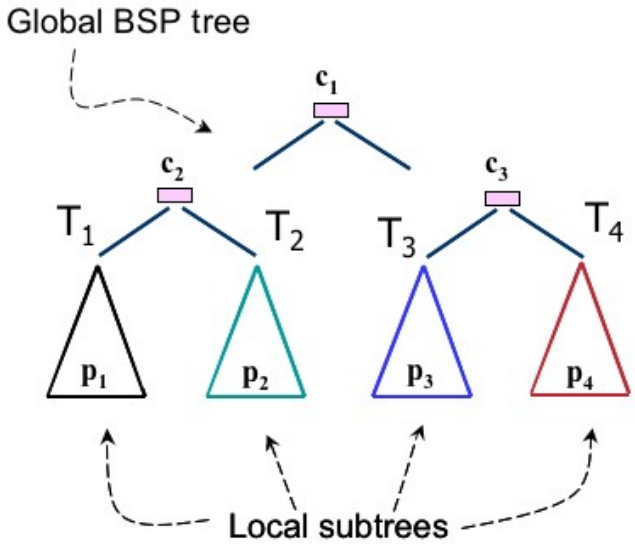


Fig. 3.1. An illustration of how the global BSP tree is partitioned among processors—each processor owns a unique subtree of the global tree

These subtrees are computed by recursively decomposing the global search space through the introduction of cutting planes that roughly divide the set of points equally among each subregion. The resulting subtrees (and their corresponding subregions) are then assigned to individual processors. The choice of cutting planes is limited to coordinate planes chosen by computing the dimension with maximum variance, as described in Section 2.4.3. The calls to `MPI_Allreduce()` dominate this algorithm and the depth of recursion is $\log_2(P)$, where $)$ is the number of processors. Under the assumption that the network latency is much bigger than both the time needed for one floating point operation and network bandwidth, the overall cost of this operation is proportional to $(\log(P))^2$.

As mentioned before, each processor owns its subtree but also has the information about the portion of the search space stored on every other processor in the environment. Therefore, the search for a suitable point $\boldsymbol{\xi}$ to use for approximation for point $\boldsymbol{\theta}$ is conducted in a coordinated manner between processors. More precisely, if the query point $\boldsymbol{\theta}$ is submitted to processor p_i but belongs to the subtree of processor p_j, processor p_i will be able to determine this fact locally, i.e., without interprocessor communication.

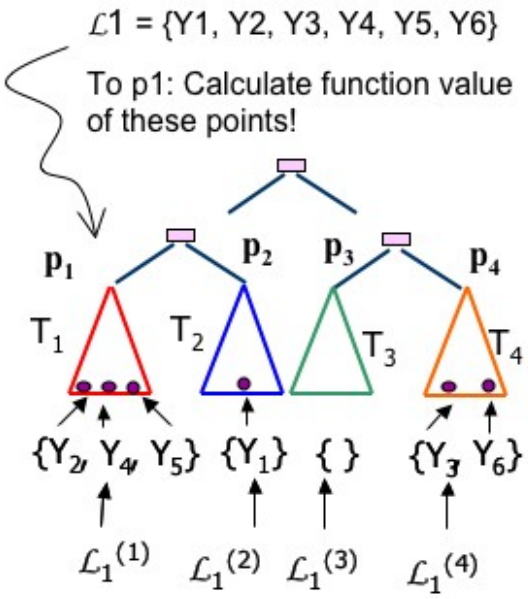
3.2 Managing Interprocessor Communication

Based on the assumption that function evaluations are relatively expensive, the goal of this parallel implementation is to design a search scheme that will minimize the number of direct evaluations that must be performed. Without lack of generality, consider processors p_1 and p_2, to illustrate the management of the distribution of the workload. Whenever there is a possibility that another processor may be able to approximate the function value at a given point $\boldsymbol{\theta}$, the query point $\boldsymbol{\theta}$ should be communicated to that processor for processing. This means that, for a given

query point $\boldsymbol{\theta}$ submitted to processor p_1, there are two possibilities. First, when point $\boldsymbol{\theta}$ belongs to a subtree T_1 of p_1 (see Fig. 3.2), we simply proceed with the sequential algorithm. The more challenging case occurs when the point $\boldsymbol{\theta}$ belongs to a subtree of some other processor, say p_2, that is, $\boldsymbol{\theta} \in T_2$. As p_1 stores some number of points that belong to processor p_2, p_1 may still be able to approximate this query. If it cannot, the processor sends the point to processor p_2. If p_2 can approximate it, it sends back the corresponding result. But if p_2 can also not approximate the query and a direct calculation has to be performed, the question is which processor should be assigned with the task of calculating $\mathbf{F}(\boldsymbol{\theta})$.

Two issues affect the choice of a strategy for answering this question. First, as we stated in the preceding section, the number of redundant calculations should be minimized. However, the load imbalance between the processors should also be minimized. If the computational load is based on the number of direct calculations performed on each processor, this number should be roughly equal across all the processors in the environment. Therefore, the most important issue to consider in answering this question is to determine the effect on load balancing between processors. It may happen that, for a particular time step, all the points for which we want to calculate function values belong to a small subregion of the accessed space. This situation is common in many combustion simulations. If the load imbalance is large, we may lose the performance advantages gained by minimizing the number of redundant direct calculations. Thus, the answer to the question: "Should p_1 or p_2 conduct the direct calculation?" proves to be a crucial part of the parallel implementation of DOLFA.

To explore this problem, we introduce the following notation. We denote by $\mathcal{L}^{(i)}$ the list of points submitted to processor p_i. This list is decomposed into sublists $\mathcal{L}^{(i)} = \bigcup_{j=1}^{n} \mathcal{L}_j^{(i)}$



Working lists for processor p_1:

- $\mathcal{B}_1 = \mathcal{L}_1^{(1)}$
- $\mathcal{N}_1 = \bigcup_{j \neq 1} \mathcal{L}_1^{(j)}$
- $\mathcal{E}_1 = \bigcup_{j \neq 1} \mathcal{L}_j^{(1)}$

Fig. 3.2. Three different types of points that are submitted to a particular processor (p_1 in this case) for evaluation

where $\mathcal{L}_j^{(i)}$ is the portion of the list that belongs to subtree T_j of processor p_j, $\mathcal{L}_j^{(i)} = \mathcal{L}^{(i)} \cap T_j$. As illustrated in Fig. 3.2, every processor p_i will have three different types of lists to deal with, described below.

- $\mathcal{B}^{(i)} = \mathcal{L}_i^{(i)}$ – the list of points that belong to the subtree of p_i. This is the list of the points that processor p_i has to calculate.
- $\mathcal{N}^{(i)} = \bigcup_{j \neq i} \mathcal{L}_j^{(i)}$ – the list of points that belong to the subtrees of other processors. This is the list of the points that processor p_i will send off to other processors.
- $\mathcal{E}^{(i)} = \bigcup_{j \neq i} \mathcal{L}_i^{(j)}$ – the list of points that p_i received from other processors since they belong to its subtree T_i.

With $\mathcal{C}^{(i)}$ we denote the list of points for which processor p_i does the direct calculations. The open question remains: "If $\boldsymbol{\theta}$ belongs to $\mathcal{L}_2^{(1)}$ – it was submitted to p_1 for calculation but it belongs to a subtree T_2 of processor p_2 – which processor will calculate $\mathbf{F}(\boldsymbol{\theta})$?". We present three approaches for solving this problem.

Approach 1. The owner of the subtree (p_2 in the above example) performs the direct integration.

In this approach, every processor p_i does the direct calculations for all the points in the lists $\mathcal{B}^{(i)}$ and $\mathcal{E}^{(i)}$, $\mathcal{C}^{(i)} = \mathcal{B}^{(i)} \cup \mathcal{E}^{(i)}$. This approach achieves maximum retrieval rates but, since problems in turbulent chemistry usually include turbulence and non-homogeneous media, this approach can also result in significant load imbalance. However, as this approach minimizes the number of direct calculations, it can be used as a benchmark to determine a lower bound for the overall number of direct calculations. This lower bound can be used for analyzing the performance of other approaches to estimate the number of redundant function evaluations.

Approach 2. The processor to which the query was submitted (p_1 in the above example) does the direct calculations. In this approach, every processor p_i performs the direct calculations for all the points in the list $\mathcal{B}^{(i)}$ and for some of the points in the list $\mathcal{N}^{(i)}$, where $\mathcal{C}^{(i)} = \mathcal{B}^{(i)} \cup \bar{\mathcal{N}}^{(i)}$ (bar denotes a subset of the list). This approach may mitigate the load imbalance, but it will introduce a communication overhead as we have additional two-way communication. This communication consists of processor p_2 letting processor p_1 know that it cannot approximate $\mathbf{F}(\boldsymbol{\theta})$ and processor p_1 returning to p_2 the calculated $\mathbf{F}(\boldsymbol{\theta})$ together with some other items needed to calculate the ellipsoid of accuracy discussed in Section 2.3.1. This communication pattern can be described as a "ping-pong" pattern. Another problem with this approach is the significant reduction in retrieval rates. For example, similar points to $\boldsymbol{\theta}$ may also be present in lists on processors p_3 and p_4 that can be approximated with $\mathbf{F}(\boldsymbol{\theta})$. In Approach 1, we would have only one direct calculation for $\mathbf{F}(\boldsymbol{\theta})$. In this approach, since p_2 will send off point $\boldsymbol{\theta}$ back to original processors to calculate it (in this case, p_1, p_3 and p_4), we would have these three processors redundantly calculate $\mathbf{F}(\boldsymbol{\theta})$.

The Hybrid Approach. We propose a hybrid version that combines the best of the two previous approaches. In this approach, processor p_i does the direct calculations for all the points in the list $\mathcal{B}^{(i)}$ and for some of the points in the lists $\mathcal{N}^{(i)}$ and $\mathcal{E}^{(i)}$, where $\mathcal{C}^{(i)} = \mathcal{B}^{(i)} \cup \bar{\mathcal{N}}^{(i)} \cup \bar{\mathcal{E}}^{(i)}$. Processor p_i decides whether to calculate $\mathbf{F}(\boldsymbol{\theta})$ for some $\boldsymbol{\theta} \in \mathcal{E}^{(i)}$ based on the average number of direct retrievals in the previous iteration and additional metric parameters. In particular, processor p_i predicts the number of calculations it will have in the lists $\mathcal{B}^{(i)}$ and $\mathcal{N}^{(i)}$ —denote this number with `avgLocal`. The number of direct

calculations can vary substantially in the course of computation; therefore, our prediction has to be local and we calculate it based on the two previous iterations. Processor p_i also knows `avgDirect`, the average number of direct retrievals (over all processors) in the previous iteration. Then, processor p_i will do no more than `avgExtra` = `avgDirect` — `avgLocal` direct calculations on the $\mathcal{E}^{(i)}$ list. For the rest of the points in the $\mathcal{E}^{(i)}$ list, p_i will send them back to the original processors to calculate them, using the "ping-pong" communication model. Based on this decision, $\mathbf{F}(\boldsymbol{\theta})$ is either computed on this processor or sent to the original processor for evaluation.

In the experimental section 3.3, we demonstrate that the hybrid strategy jointly minimizes the number of redundant calculations and the load imbalance but that in most of the cases, Approach 1 has the best overall CPU performance. However, if the BSP tree is extremely unbalanced with respect to points that are submitted to the processors, both Approach 1 and the Hybrid Approach will fail to ensure reasonable performance. As these cases are not rare (in particular for non-steady state combustion calculations) a strategy that deals with this case has to be designed. In the following section we propose such an approach.

3.2.1 Redistributing the BSP Tree

A combustion simulation is usually an iterative process that incorporates computational fluid dynamics (CFD), chemical reaction mechanisms, and potentially various radiation and turbulence models. Therefore, it can be assumed that for each iteration step, it is possible to call the DOLFA database only once, with a comprehensive list of the points that we want to calculate. We have observed that, even if the BSP tree is well balanced regarding the initial points queried to the database, it can occur that in some future calls, most of the points will belong

to subtrees assigned to a small subset of processors. We have also noticed that this load shifts gradually, and the difference between the number of direct integrations on a particular processor is not large, between two iterations (i.e., two database calls) but this difference accumulates over time. Therefore, we propose to redistribute the BSP tree after each iteration to prevent the accumulation of a load imbalance.

The internal BSP tree structure, without the leaves that represent convex regions, takes only approximately 0.1% of the memory for the entire tree for typical combustion simulations. Therefore, we first extend our global BSP tree distribution so that every processor keeps in its memory a local copy of most of the internal structure of the entire global BSP tree. The internal structure of the subtrees that belong to other processors may become outdated and it should be updated after every iteration.

Every processor knows the number of direct calculations it performed during the previous iteration. It also maintains the number of direct calculations performed for each region in its part of the BSP tree. In the first stage of the redistribution algorithm, processors exchange the new information about the internal structure of their BSP subtrees. Then, we sort the list of processors $\{p_i\}$ according to their computational load C_i during the previous database call. As a measure for the computational load, we use the number of direct calculations performed on the processor during the last iteration. If there are P processors in the environment, we introduce $P/2$ pairs of processors, $[p_i, p_{P-i}]$. Processor p_i will assign a part of its subtree to processor p_{P-i} such that the computational load for the leafs (i.e., the number of direct calculations in the regions during the previous iteration) in the subtree is roughly $(C_i - CP - i)/2$. Processor p_i will send a message containing the database points that belong to the convex regions (leaves) it has decided to assign to its pair processor. Processor p_i will mark those regions that it sent as if they belong

to processor p_{P-i} and will not perform any direct calculations for the points belonging to these regions. However, if possible, it will retrieve the points from those regions.

We observe that after a number of iterations, our subtree does not look like the one in the Fig. 3.1. The reason is, processors do not own the entire subregions that were initially assigned to them. An illustration of the new BSP tree distribution is shown in Fig. 3.3. We believe that this approach shows great promise for improving the load balancing of the database algorithms through this redistribution of the BSP tree among processors.

3.3 Performance Results

First, we analyze the impact of the three approaches for distribution of direct calculations on the scalability of the parallel DOLFA library. We tested our software on a Beowulf cluster with 94 dual processor AMD Opteron 250 nodes, each with 4GB RAM with a fully connected 4-way Infiniband Topspin switch interconnection network. For this analysis, we use the simulation which mimics a two-dimensional laminar reacting flow. The model parameters are as follows: the number of time steps is 20, the subdomain size on every processor is 100×100 and the vector size (size of query point) is 40. The user-defined tolerance is 0.01 and it takes approximately 0.1s to perform one direct evaluation. In the first section, we describe the performance of the three strategies used for the managing of the direct calculation of the points. In the second section, we highlight the performance of the new redistribution algorithm.

For both sections, we will utilize more suitable metrics suitable for analyzing the parallel algorithms introduced in this chapter than standard metrics, such as scaled speedup and efficiency. For example, if we use the scaled efficiency metrics to analyze the performance of our parallel software, we can get misleading results that can lead to choosing an inappropriate

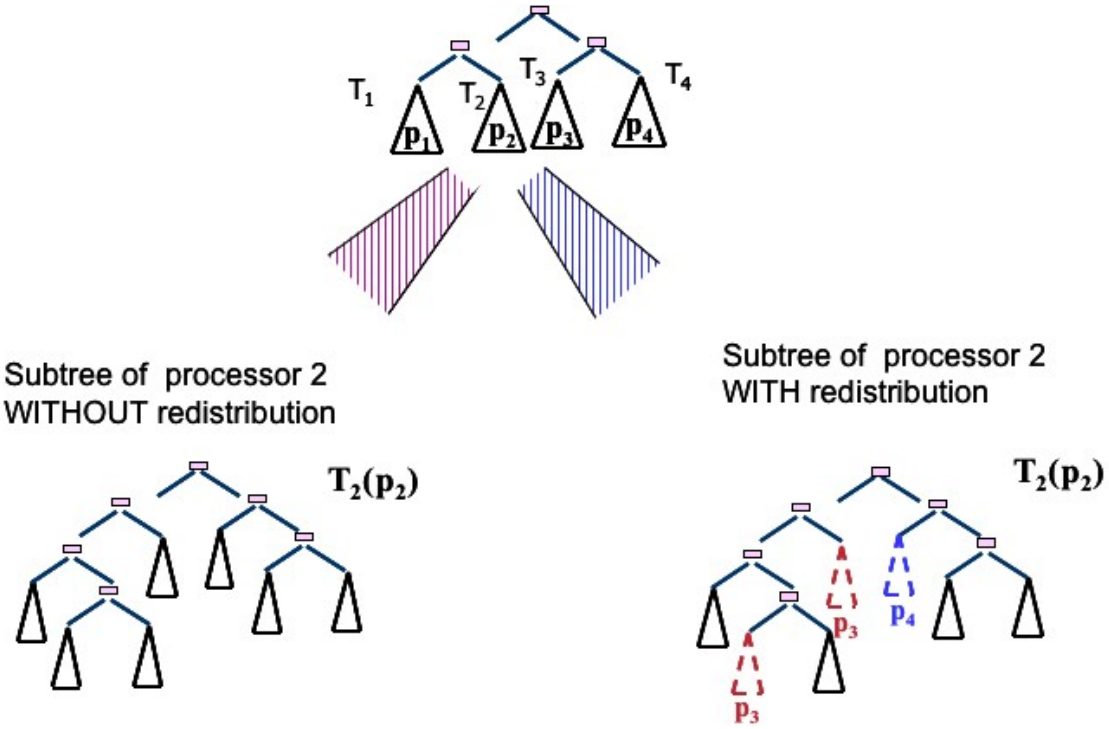


Fig. 3.3. An illustration of how the global BSP tree is distributed among the processors when redistribution is applied. We observe that, in the case when the redistribution is applied, the parts of the subtree of processor p_2 now belongs to other processors (p_3 and p_4 in this case).

communication strategy for the distribution of the computational load. The scaled efficiency is a function of the problem size and the number of processors, $E(N,P) = T(N,1)/T(PN,P)$, where N is the problem size, P is number of processors, $T(N,1)$ is the time required to solve a problem of size N on a single processor, and $T(PN,P)$ is time required to solve the same problem of size NP on P processors.

We can analyze the performance of the straightforward approach to to implementing parallel DOLFA where we run the separate version of DOLFA database on each processor using the scaled efficiency. If the number of approximations does not vary much within subdomains, we can deduce that this approach has a scaled efficiency close to 1, since there is no communication involved. However, we have concluded that this approach suffers from two significant drawbacks: (a) significant number of the overall direct calculations; and (b) non-scalable usage of the total memory available on a parallel computer.

To avoid those problems, we will use a performance metrics that is more suitable for analyzing the performance of the parallel DOLFA database, called relative load balancing. The relative load balancing factor is defined as the difference between maximum and minimum number of direct calculations across all processors divided by the average number of direct calculations. The relative load balancing of zero implies perfrect load distribution. With this metrics, we will the weak scaling performance of the parallel DOLFA, where the domain size per processor stays the same as we increase the number of processors.

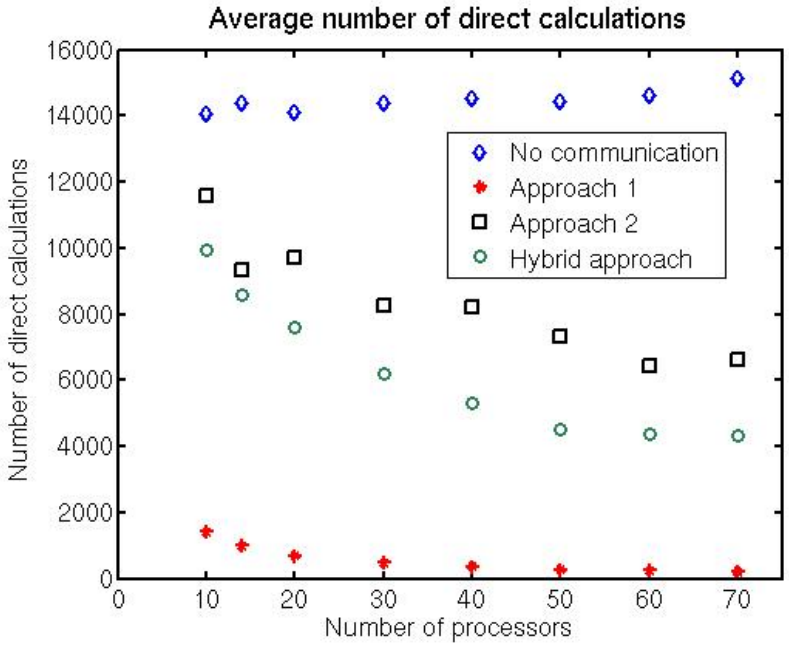
3.3.1 Comparison of the Three Communication Heuristics

To investigate the performance of the three strategies for managing the computational load, we ran two limiting case experiments, which we refer to as Experiments 1 and 2. These

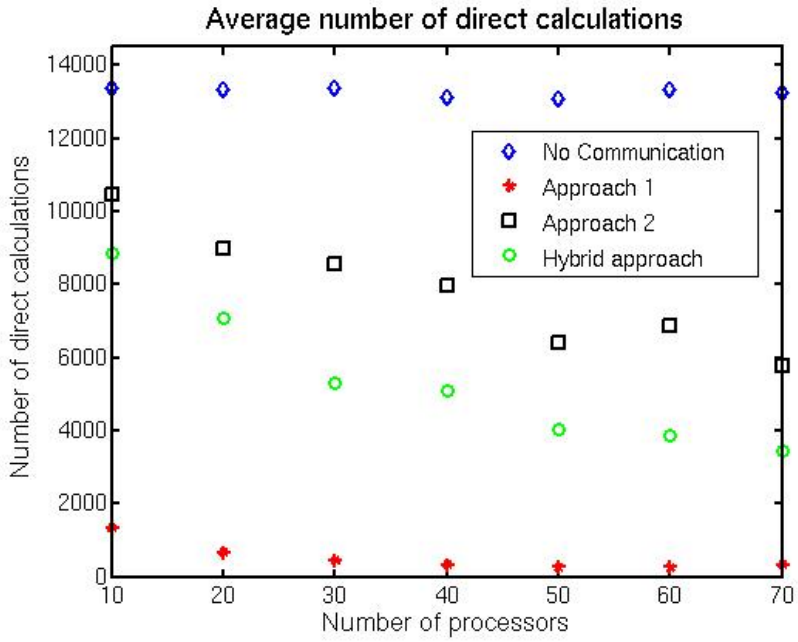
experiments mimic the properties of a typical steady-state combustion simulation. For Experiment 1, we assign each processor a different domain. Thus, for this case, the number of direct calculations should be roughly a linear function of the number of processors. For Experiment 2, each processor got the same domain. For this case the number of direct calculations should be roughly constant function of the number of processors. The objective is to observe how the load balancing and communication patterns behave in these limiting cases.

First, we compared the average number of direct calculations for each of the communication heuristics described in the preceding section. We can see in Fig. 3.4 that, both for Experiment 1 and Experiment 2, Approach 1 minimizes the number of direct calculations. Another metric that reveals the properties of these three approaches is the relative load balancing. The perfect load balancing implies that this factor is zero. If we compare the relative load balancing factor for the three approaches for the two limiting cases in Fig. 3.5, we can see that for Approach 1 this factor can reach 2.5. The best relative load balancing factor is obtained by Approach 2. However, if we compare the average CPU times for the three communication approaches together with the no-communication scheme in Fig. 3.6 we can see that for both experiments Approach 1 still offers superior execution times, even though its computational load is not well balanced.

From these figures we note that the two goals, minimizing the number of redundant calculations and minimizing load imbalance, cannot be met simultaneously. Approach 1 obtains the minimal number of direct calculations, but it has an inherent load balancing problem whose solution may prove to be essential for running transient simulations. However, regardless of the load imbalance, Approach 1 still proves to be scalable in the sense that the overhead remains constant when we double the load and the number of processors. On the other hand, Approach

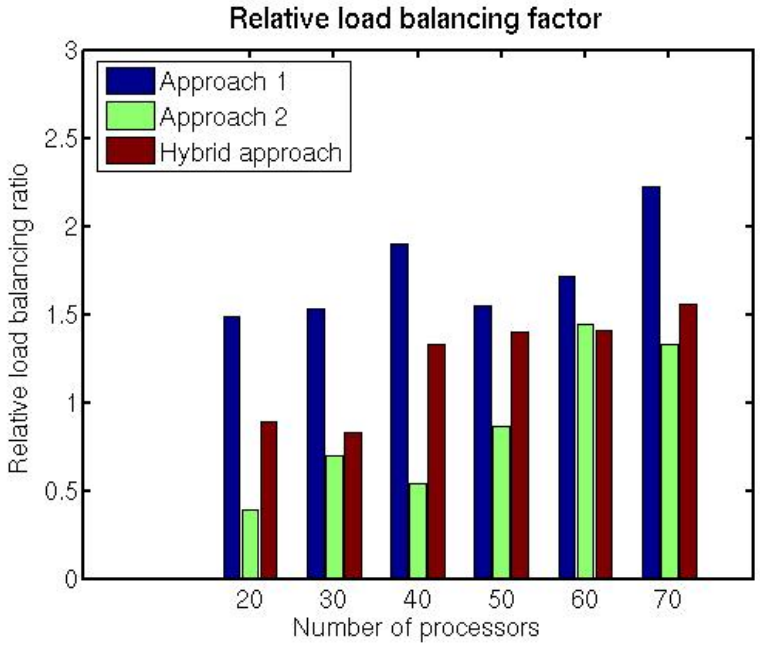


(a) Experiment 1

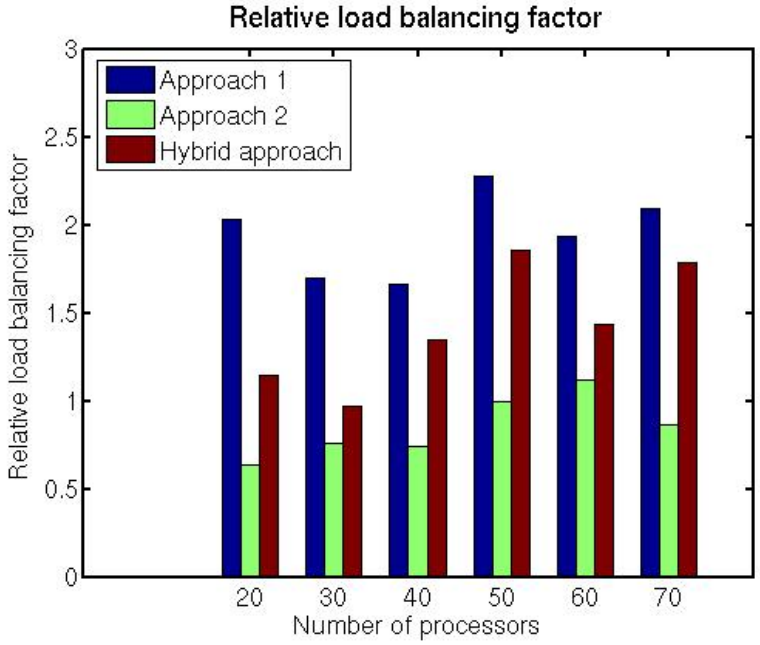


(b) Experiment 2

Fig. 3.4. The average number of direct calculations for the three communication approaches simulated for Experiment 1 (top) and Experiment 2 (bottom)

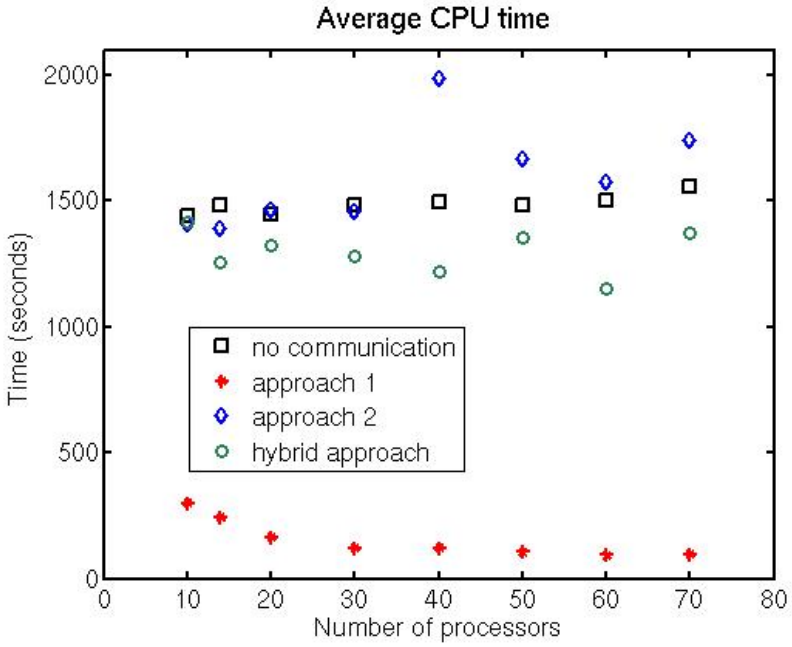


(a) Experiment 1

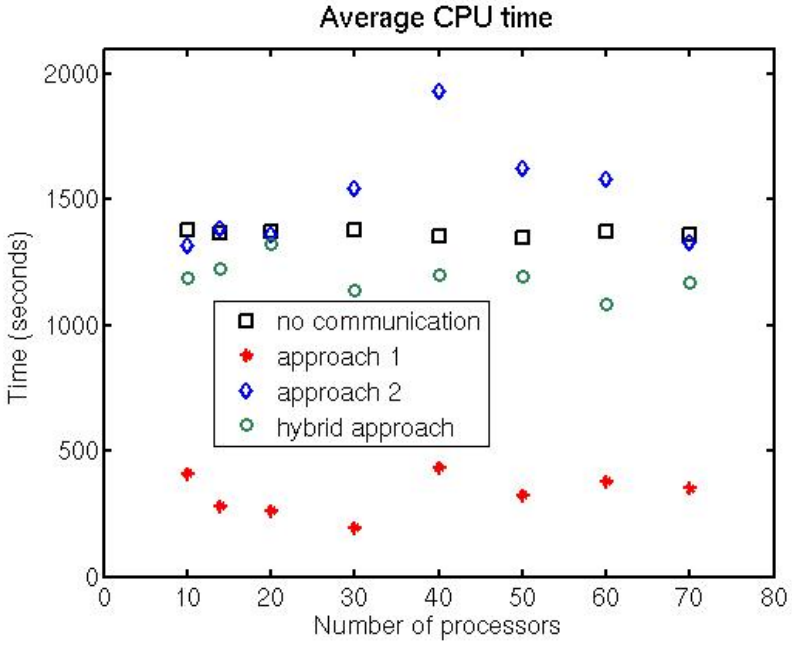


(b) Experiment 2

Fig. 3.5. The relative load balancing factor for the three communication approaches simulated for Experiment 1 (top) and Experiment 2 (bottom)



(a) Experiment 1



(b) Experiment 2

Fig. 3.6. The average CPU times for the three communication approaches simulated for Experiment 1 (top) and Experiment 2 (bottom)

2 results in a large number of redundant direct calculations. The Hybrid Approach is scalable and has the smallest overhead compared to the other two methods, but it still yields a significant number of redundant calculations. However, as we will see in the next section, for the case of transient simulations it has a comparable CPU time to Approach 1.

3.3.2 Performance of the Redistribution Algorithm

To investigate the properties of the redistribution algorithm, we ran a simulation that mimics the properties of a typical transient combustion simulation with Experiment 1 (each processor gets a different domain on which to calculate). We compared the relative load balancing and the overall CPU times for Approach 1 and the Hybrid Approach. In Fig. 3.7, we compared the relative load balancing for the case when Approach 1 is used with and without redistribution. We observe that the redistribution significantly reduces the load imbalance, on average by a factor of 2. In Fig. 3.7 we also compare the relative load balancing for the case when the Hybrid Approach is used, with and without redistribution. Again, the load imbalance is significantly smaller when redistribution is used, and it is on average 4 times smaller than for the same case with Approach 1. In Fig. 3.8 we compare the ratio between total CPU time of the simulation and the cumulative CPU time needed for redistribution. We can see that the time for redistribution takes approximately 1% of total CPU time for both Hybrid Approach and Approach 1 which is an acceptable overhead considering the positive effect the redistribution has on load balancing. Finally, in Fig. 3.9 we compare the CPU times for the two approaches to manage the computational load with redistribution. While for the small number of processors Approach 1 is superior to the Hybrid Approach, we can see that when we increase the number of processors, the parallel overhead becomes large for Approach 1.

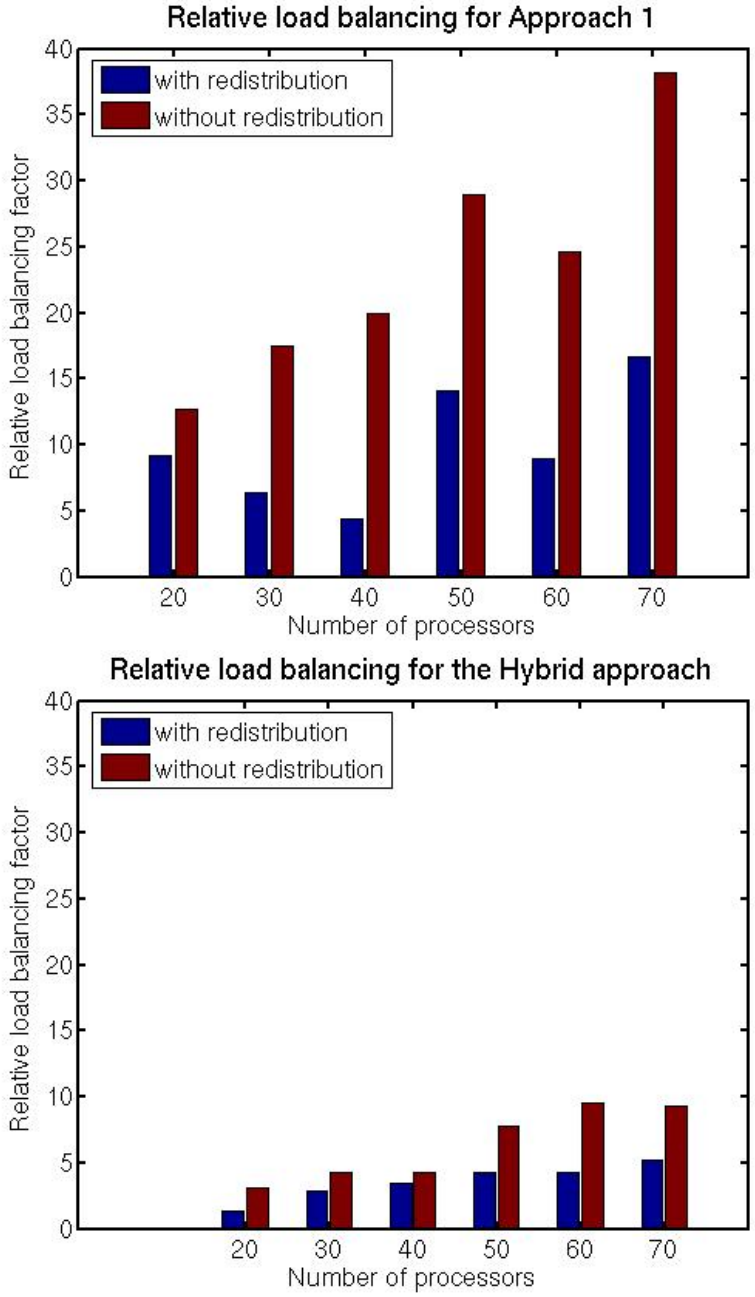


Fig. 3.7. Top: a comparison of the relative load balancing for a typical transient application using DOLFA with and without redistribution with Approach 1. Bottom: a comparison of the relative load balancing for the Hybrid Approach, with and without redistribution.

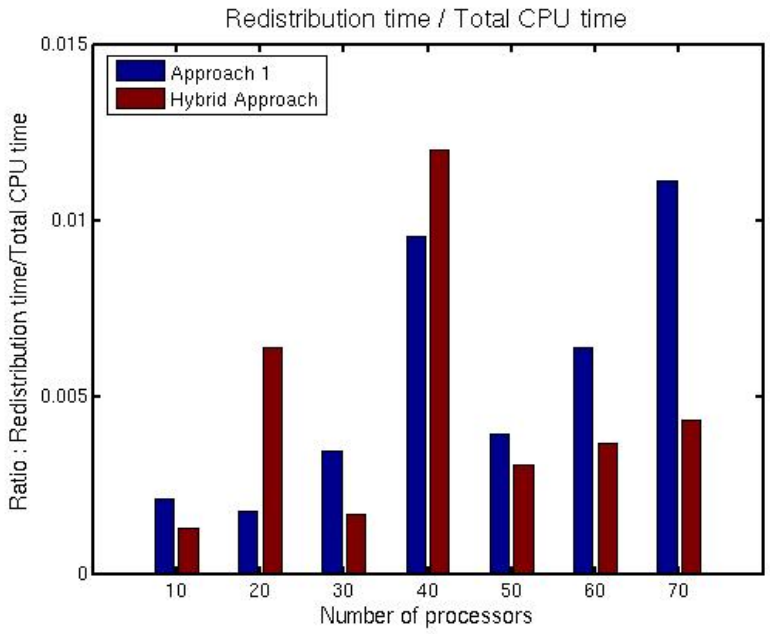


Fig. 3.8. Ratio of CPU time needed for redistribution and total CPU time for Approach 1 and the Hybrid Approach

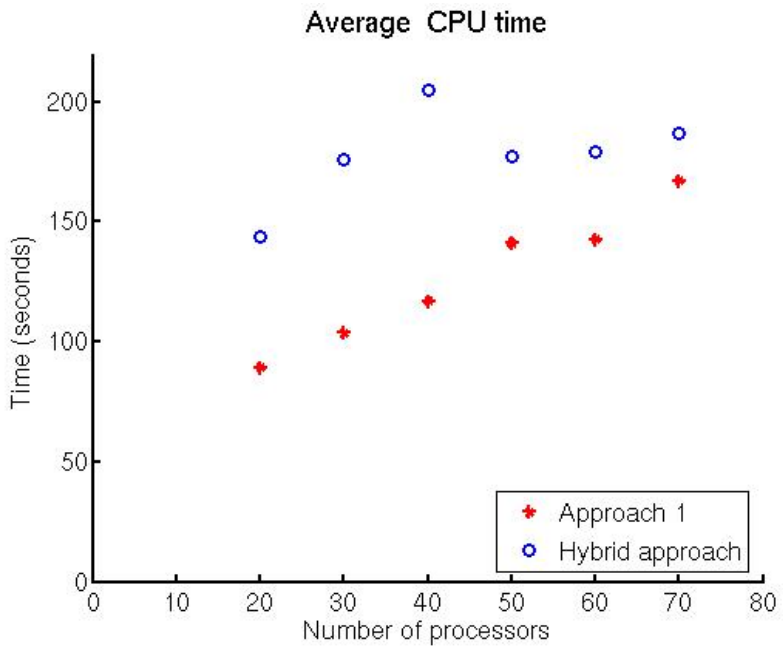


Fig. 3.9. A comparison of the overall CPU time for Approach 1 and the Hybrid Approach with redistribution

Our overall recommendation would be that Approach 1 should be used in a steady-state-type simulations where the computational domain does not change significantly during the simulation. Approach 1 can also be used with transient simulations, but only for a small number of processors. If the user has a large number of processors available (> 50), then the Hybrid Approach should be used.

3.4 Conclusions

In this chapter we described the algorithms and heuristics implemented in the parallel version of DOLFA. Algorithms for building a global BSP tree and the redistribution of the BSP tree if the computational load is not balanced have been presented. Also, three heuristics for managing the computational load have been introduced. We presented computational results that demonstrate that our implementation is scalable and shows significant speedup when compared to the case when there is no communication between processors (by using the sequential version of DOLFA on each processor).

Chapter 4

Introduction to the Photon Monte Carlo Method for Radiative Heat Transfer

Thermal radiation plays an important role in many engineering applications [4]. Since Radiative Heat Transfer (RHT) rates are generally proportional to the fourth power of the temperature difference, applications that simulate combustion processes, nuclear reactions, solar energy collection and many other physical phenomena are highly influenced by the accuracy of models that deal with radiative effects. Radiative properties of materials are often difficult to measure. Directional dependencies can be strong and these properties can vary depending on the state of the material. For surfaces, it becomes important how the material is prepared and maintained. Moreover, the radiative properties of gases may have a strong wavelength dependence and vary strongly over the spectrum [4].

Radiative Heat Transfer differs from other physical phenomena because of its strong nonlocal effects. This nonlocality arises from the fact that photons that carry radiation have large mean free paths (the average path from the point where the photon is released to the point where it is absorbed), as illustrated in Fig. 4.1. Because of these nonlocal effects, conservation laws cannot be applied over an infinitesimal volume (as is common for many fluid mechanics problems), but must be applied over the entire computational domain. The resulting formulation leads to complex models based on integro-differential equations. In this chapter, we give a brief introduction to the basic properties of RHT. This summary is not meant to be comprehensive,

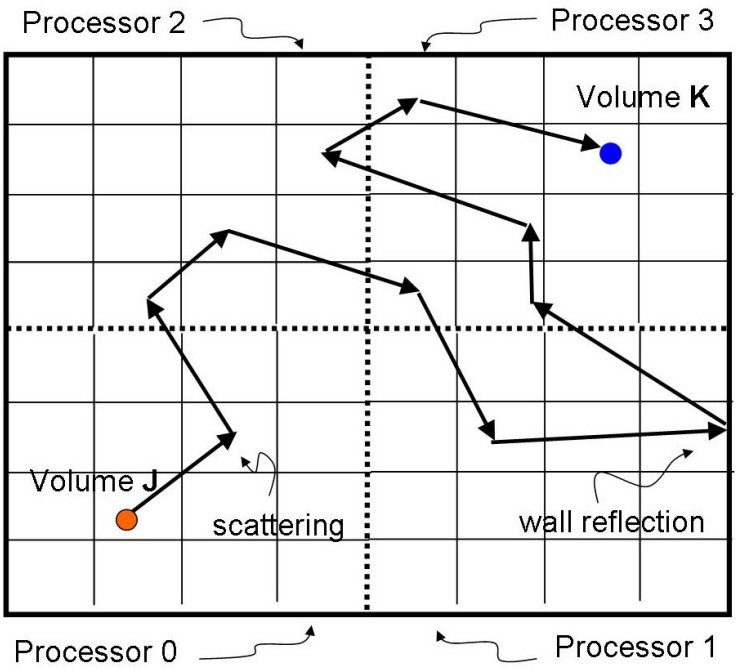


Fig. 4.1. An example illustrating the nonlocal influence of radiation. Here we show the path of a photon in a participating medium with scattering and reflective walls. We observe that energy emitted from volume J can affect the absorbed energy of volume K, even though they are quite distant and do not belong to the same processor sub-domain.

but is sufficient to emphasize the complexity of RHT and motivate the introduction of the Photon Monte Carlo (PMC) algorithms. A detailed description of specific aspects of the radiative transfer calculations are included in Appendix A. For a comprehensive study of radiative heat transfer see Modest,[4].

4.1 Properties of Radiative Heat Transfer

Thermal radiative energy can be viewed as consisting of electromagnetic waves or as consisting of photons. The choice of the model usually depends on the properties of the computational method. For the ray tracing algorithms a "quantum mechanical" view of of radiation as represented by a collection of massless photons is consistent with the modeling principles. In Fig. 4.2, we presented a schematic of various types of interactions of the photon with the participating medium. If rays on their path through a medium are absorbed or scattered from the incoming direction, the energy is removed from the direction of propagation. This process is called *attenuation.* In the case of scattering, they add their energy to some other direction. Rays can also be emitted and in that case they also add energy to the direction of propagation. Adding the energy to the direction of propagation by scattering or emitting is called *augmentation.*

The change in intensity I_η from point s to point $s + ds$ is calculated by summing the contributions from emission, absorption, scattering away from direction $\hat{\mathbf{s}}$ and scattering into the same direction $\hat{\mathbf{s}}$ at wavenumber η, [4]. We express this infinitesimal change in intensity as

$$\begin{aligned} \frac{dI_\eta(\hat{\mathbf{s}})}{ds} &= \hat{\mathbf{s}} \cdot \nabla I_\eta(\hat{\mathbf{s}}) \\ &= k_\eta I_{b\eta} - \beta_\eta I_\eta(\hat{\mathbf{s}}) + \frac{\sigma_{s\eta}}{4\pi} \int_{4\pi} I_\eta(\hat{\mathbf{s}}_i) \Phi_\eta(\hat{\mathbf{s}}_i, \hat{\mathbf{s}}) d\Omega_i. \end{aligned} \tag{4.1}$$

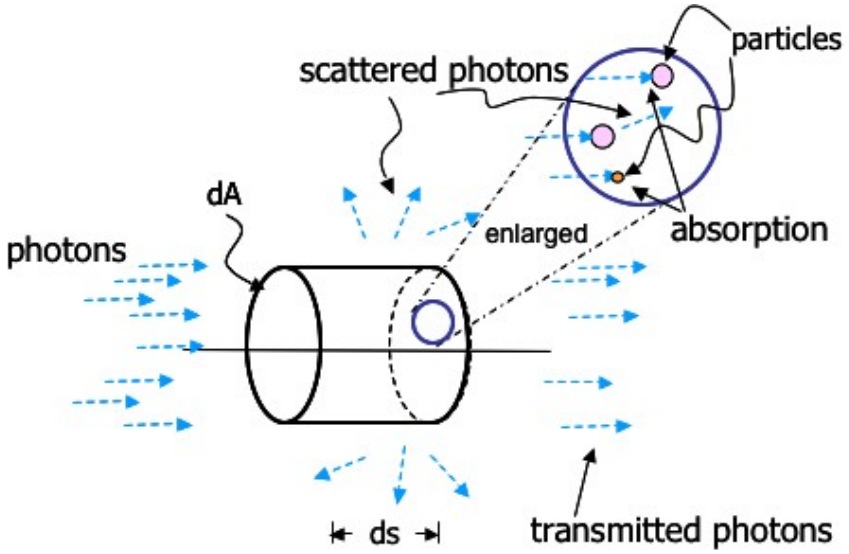


Fig. 4.2. An illustration of the possible interactions of the photon with the participating medium, from [4]

where k_η is the absorption coefficient, $\sigma_{s\eta}$ is the scattering coefficient (dependent on the wave number), β_η is the extinction coefficient that describes total attenuation, and $\Phi_\eta(\hat{\mathbf{s}}_i, \hat{\mathbf{s}})$ is the phase function that describes the probability that a ray coming from direction $\hat{\mathbf{s}}_i$ will scatter into the direction $\hat{\mathbf{s}}$. Integration is performed over the entire solid angle Ω_i.

Equation (4.1) is called the Radiative Transfer Equation (RTE). It is an integro-differential equation in five-dimensional space (three spatial variables and the two angles that determine the direction of $\hat{\mathbf{s}}$). An exact analytical solution to this equation is typically impossible to obtain. Exact solutions are possible only for a few very simple situations, usually omitting scattering, assuming constant properties and gray, diffuse radiation. Therefore, research on radiative heat transfer in participating media has generally proceeded in two directions, finding exact solutions of highly idealized situations, or developing approximate solution methods for more complex scenarios.

The issues that make radiative heat transfer calculations difficult include the complex geometry of the enclosure, nonuniform temperature field, scattering and nonuniform, nonlinear and nongray radiative properties. Classical methods such as spherical harmonics, first developed by Jeans [33] or discrete ordinates, developed by Chandrasekhar [34], have been successfully implemented in numerous applications. However, they have failed to address various important issues that can arise in the applications where more complex radiative properties have to be modeled. For example, with the increasing complexity of the enclosure geometry, the complexity of the previously described classical methods grows rapidly since their intricate mathematical formulations become more involved.

4.2 Alternative Methods to the Photon Monte Carlo for Radiative Heat Transfer Calculations

As discussed in the preceding section, the problem of determining the radiative intensity in a gray medium or on a spectral basis as the function of five independent variables represents a difficult task. For optically thick cases approximate methods such as diffusion approximation can be used. This method is fairly simple, but it is not valid near a boundary and can be useful only in optically extremely thick situations (for example, heat transfer through hot glass) [4]. In this section we present two methods that are designed to be used for a wide range of different problems. These methods transform the equation of transfer, equation (4.1), into a set of simultaneous partial differential equations that can be solved by more traditional methods.

4.2.1 The Method of Spherical Harmonics

The method of *spherical harmonics* was first proposed by Jeans [33] in 1917. It can provide an approximate solution to the radiative intensity of arbitrarily high order. In this method, the radiative intensity field $I(\mathbf{r}, \hat{\mathbf{s}})$ can be represented as a function given on a unit sphere with a center at a fixed location $\mathbf{r}$. Any such function can be expressed in terms of a two-dimensional, generalized Fourier series, where the coefficients are functions of the point $\mathbf{r}$ given as:

$$I(\mathbf{r}, \hat{\mathbf{s}}) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \underset{l}{I}^{m}(\mathbf{r}) \underset{l}{Y}^{m}(\hat{\mathbf{s}}), \tag{4.2}$$

where $\underset{l}{I}^{m}(\mathbf{r})$ are position dependent coefficients and $\underset{l}{Y}^{m}(\hat{\mathbf{s}})$ are the spherical harmonics, given with analytical formulas based on the Legendre polynomials [4].

We can substitute equation (4.2) into the general equation of the radiative transfer, equation (4.1), and integrate over the unit sphere. If we truncate the infinite series in equation (4.2) to some finite number of terms N, and exploit the orthogonality of the spherical harmonics, we get $(N+1)^2$ PDEs with the coefficients $I_l^m(\mathbf{r})$ as unknowns.

The advantage of this method is the conversion of the integro-differential equations to relatively simple PDEs. The disadvantage of the method is that low-order approximations are usually accurate only in media with almost isotropic radiative intensity. In addition, the accuracy improves slowly for higher order approximations while the mathematical complexity of the method grows rapidly.

4.2.2 The Method of Discrete Ordinates

While the method of spherical harmonics uses Fourier series expansions to simplify equation (4.1), the method of discrete ordinates uses a finite-differencing approach. With this method, the total solid angle range of 4π is discretized into a set of n different directions $\hat{\mathbf{s}}_i$, $i = 1, ..., n$. The RTE, equation (4.1), is solved for for these directions where the integrals over direction are replaced with numerical quadrature as follows:

$$\int_{4\pi} f(\hat{\mathbf{s}}) d\Omega \simeq \sum_{i=1}^{n} \omega_i f(\hat{\mathbf{s}}_i), \tag{4.3}$$

where the ω_i are quadrature weights associated with the directions $\hat{\mathbf{s}}_i$. The choice of quadrature is arbitrary, but there can be restrictions on the directions $\hat{\mathbf{s}}_i$ and weights ω_i based on accuracy and performance issues. This method was first proposed by Chandrasekhar [34]. With this

method equation (4.1) can be approximated by n linear PDEs as:

$$\begin{aligned}\hat{\mathbf{s}}_i \cdot \nabla I_\eta(\mathbf{r}, \hat{\mathbf{s}}_i) &= k_\eta(\mathbf{r}) I_{b\eta}(\mathbf{r}) - \beta_\eta(\mathbf{r}) I_\eta(\mathbf{r}, \hat{\mathbf{s}}_i) \\ &\quad + \frac{\sigma_{s\eta}(\mathbf{r})}{4\pi} \sum_{j=1}^{n} \omega_j I_\eta(\mathbf{r}, \hat{\mathbf{s}}_j) \Phi_\eta(\mathbf{r}, \hat{\mathbf{s}}_i, \hat{\mathbf{s}}_j). \end{aligned} \tag{4.4}$$

One of the serious shortcomings of the discrete ordinates method is *false scattering*. This effect is a consequence of spatial discretization error and it is similar to numerical diffusion in Computational Fluid Dynamics (CFD). More precisely, if a ray is traced through the enclosure by the discrete ordinates method, the ray will slowly widen as it moves farther away from its point of origin. This nonphysical diffusion of the radiative intensity can be mitigated by using a finer mesh of control volumes, at greater computational cost.

Another drawback is the so-called *ray effect*, which is a consequence of the angular discretization. The problem of ray effect can be illustrated by considering an enclosure with a very small element with extremely strong emission. Energy from this element will be carried away from it into the directions of the discrete ordinates. Far away from the emission zone these rays will become so far apart that some control volumes and/or surface elements may not receive any energy from this zone. The ray effect can be reduced by increasing the size of control volumes and surface zones. The consequence of these two drawbacks is that that with the finer mesh, a finer angular discretization must be used. These requirements can lead to a high performance penalty, as the cost of this method grows as O(h^5), where h is a characteristic length of the mesh.

4.3 Photon Monte Carlo Ray Tracing Methods

Another approach, the Photon Monte Carlo (PMC) method, has a fundamentally simple formulation because it is in its core only a sampling method—the complexity of its implementation grows very slowly as the problems becomes more difficult to solve. As illustrated in Fig. 4.3, we note that the complexity of the formulation for the PMC methods grows linearly with the complexity of the problem and the same relationship holds for CPU time. For other conventional methods, the complexity grows almost exponentially. Therefore, we conclude that the PMC ray tracing method has a relative advantage over these methods with respect to its generality. We can also note that, to take the advantage of the PMC method, we have to employ it in an application that explores complex radiative properties and/or complex geometries.

The PMC methods have been widely used in computer graphics for radiosity calculations, computing effects such as soft shadows and motion blur [35]. Various algorithms are implemented to increase the efficiency of the PMC calculations. However, for implementing the PMC method for combustion applications we have to use fundamentally different approach since we are conducting ray tracing through the participating medium, which is discretized. Moreover, the greatest complexity in the PMC methods for radiosity calculations arises from the intricate shape and distribution of the surfaces. However, in combustion applications, surfaces are of simple shape and regular distribution (for example, engine walls, furnace valves) as compared to a typical scene rendered by computer graphics. Therefore, we cannot apply the same methodology and algorithms to design a PMC method for solving the RHT problem in combustion applications.

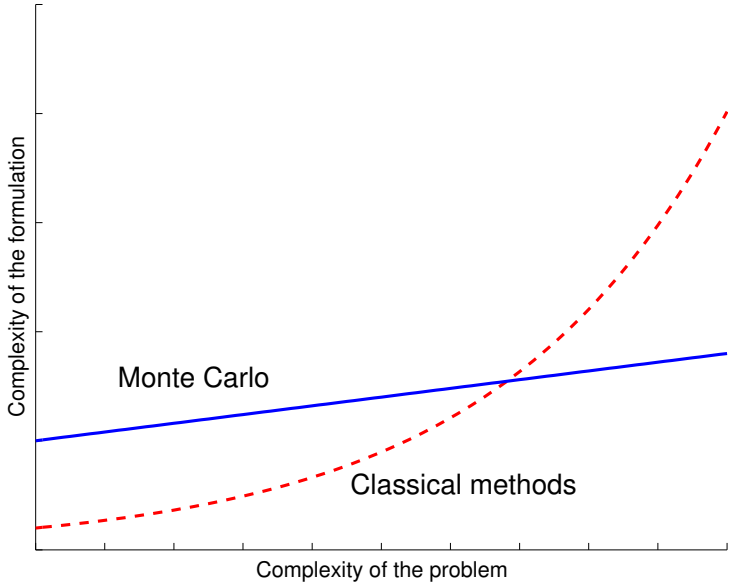


Fig. 4.3. A graph comparing the complexity of the PMC method and other conventional solution techniques, from [4]

The PMC method traces a statistically significant sample of photon bundles from their point of emission to their point of absorption. When the photon bundle is absorbed, its energy is added to the energy of the element. With this approach, the PMC method is able to calculate energy gains for every element (or fluid particle) in the enclosure. The total emissive energy E of the enclosure is calculated as the sum of the emissive energies of surface and volumes in the enclosure. Then if the total number of rays emitted is N, for each element i in the enclosure with emissive energy E_i the number of rays we emit from it is $N_i = NE_i/E$.

The main issue for the PMC algorithm is how to coordinate the randomness of the sample with the physical properties of the enclosure. With the PMC methods, problems such as determining the number of photon rays, points of emission within a surface or a volume, wavelength, points of absorption and many others must be solved based on the enclosure properties. These basic relations needed for implementing the PMC ray tracing algorithm are described in Appendix B.

If in a given problem conduction and/or convection are important, the RHT equation must be solved simultaneously with the overall conservation of energy equation:

$$\rho c_v \frac{\mathrm{D}T}{\mathrm{D}t} = -\nabla \cdot (k\nabla T) - p\nabla \cdot \mathbf{v} + \mu\Phi + \dot{Q}''' - \nabla \cdot \boldsymbol{q_r}, \tag{4.5}$$

where c_v is the specific heat, T is the temperature, p is the pressure, k is the thermal conductivity, $\mathbf{v}$ is the velocity vector, Φ is the dissipation function, $\dot{Q}'''$ is the heat generated within the medium (for example, the energy release due to chemical reactions), and $\nabla \cdot \boldsymbol{q_r}$ is the net radiative source.

The energy equation is usually solved by conventional numerical methods, such as Finite Volume (FV) or Finite Element (FE) methods, and the radiative term $\nabla \cdot \boldsymbol{q_r}$ is included as a

source term. In that case, an iteration of the temperature field is necessary using the following steps.

- The temperature field is used by the PMC method to solve the radiation problem, leading to volume emission and absorption rates, $Q_{em,j}$ and $Q_{abs,j}$ for each sub-volume. The net radiative source is then calculated from equation (4.7) which is introduced in the next section.

- Radiative sources are fed back into equation (4.5) which is then solved to update the temperature field.

4.3.1 Estimating the Statistical Error from the Standard Deviation

During a large-scale PMC simulations, billions of random numbers are required. Therefore, it is essential that the random number generator has a long period, has good statistical properties and is efficient. One such random number generator is Mersenne Twister [36]. It has a very long period, $2^{19937} - 1$ (around 10^{997}), and 623-dimensional equidistribution property (for more details about statistical properties of the random number generators, see [37]). It is also very efficient, using a working area of only 624 words. However, every random process introduces a standard deviation error and we have to be able to predict this error and adjust the total number of rays we emit accordingly.

If we emit a total of N rays from the enclosure, we divide N into S independent subsamples, $N_s = N/S$. For each subsample i, we then emit N_s samples from the enclosure (surface and volume elements) and calculate absorbed energy Q^i_{abs} for every enclosure element. The mean value of the absorbed energy is given by $\langle Q_{abs} \rangle = \left(\sum_{i=1}^{S} Q^i_{abs} \right) / S$ and the variance is

given by:

$$\sigma_m^2 = \frac{1}{S(S-1)} \sum_{i=1}^{S} [Q_{abs}^i - \langle Q_{abs} \rangle]^2. \tag{4.6}$$

The *central limit theorem* states that the mean of the subsamples $\langle Q_{abs} \rangle$ follows a Gaussian distribution, independent of the distribution of the subsamples. This fact implies that we can say with great confidence that the correct answer Q_{abs} lies within the interval $\langle Q_{abs} \rangle \pm 2\sigma_m$.

For a given problem, we do not know *a priori* how many rays we have to emit from the enclosure to achieve a certain confidence interval. To solve this problem, we start with some small number of rays, N_{small} and calculate a standard deviation. Based on this number, we estimate the number of rays needed to achieve a standard deviation of some predetermined value. For example, Fig. 4.4 illustrates how the standard deviation of absorbed energy that PMC calculates decreases when the number of rays is increased. This figure represents the standard deviation of absorbed energy for an infinite parallel plate problem with an emitting/absorbing participating medium and a case of gray, diffuse radiation for four different mesh discretizations. Temperatures of the plates are 800K and 1000K respectively, and they are perfect absorbers. The optical thickness of the medium is 0.4. Shown is the standard deviation as function of the number of rays for four different discretizations of the participating medium. Note that the standard deviation increases as the discretization (number of elements between the plates) increases.

4.4 The PMC Photon Monte Carlo Algorithm

The PMC method traces a statistically significant sample of photon bundles from their point of emission within surface/volume j to their point of absorption within surface/volume k. When the photon bundle is absorbed, its energy is added to the energy of the absorbing element.

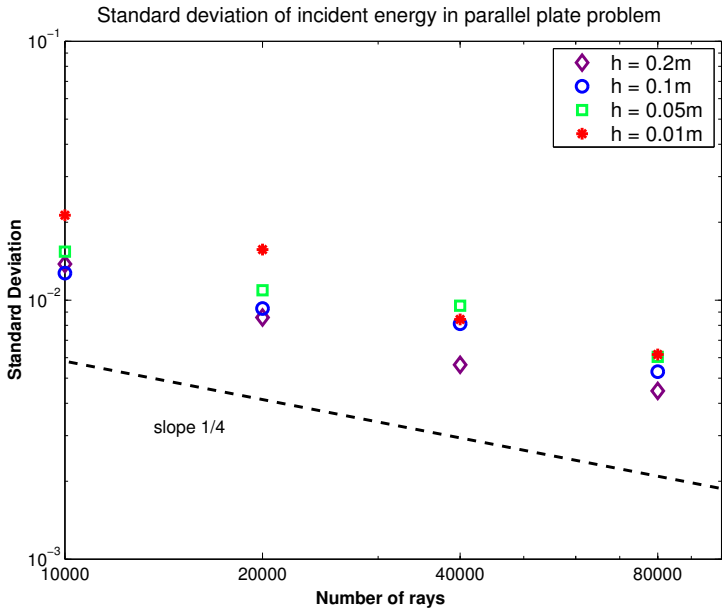


Fig. 4.4. The standard deviation of absorbed energy for the parallel plate problem

Using this approach, the PMC method is able to calculate the energy gains/losses for every element (or fluid particle) in the enclosure. The emitted energy from the element is expressed as $\int_V 4\sigma T(\mathbf{r})^4 k_{\mathrm{P}}(\mathbf{r}) dV$, where σ is the Stephan-Boltzmann constant and k_{P} is the Planck-mean absorption coefficient. Various quadrature formulas can be employed for the calculation of this integral. Then, we calculate the divergence of the heat flux for every sub-volume in the domain (and the radiative heat flux for the surface elements) as:

$$(\nabla \cdot \boldsymbol{q}_{\boldsymbol{r}})_i = \frac{1}{V_i}(Q_{em,i} - Q_{abs,i}). \tag{4.7}$$

This flux is needed as the radiation component along with convection and conduction in the energy conservation equation.

There are two possibilities of how to conduct the tracing— trace one ray at a time, or trace a ray through one mesh element and then proceed to the next ray. We decided to implement the second approach by organizing rays in the linked list—after a ray has been traced through one element, the next element it will go through is calculated and ray is pushed back into the list. One advantage of this approach is a better data locality, since rays that are close in the list have more chance of traversing the same element. The second advantage becomes obvious for the parallel implementation, which we describe in the next chapter. For the ray tracing, it is necessary to know the coordinates of the vertices in the surface/volume elements, the face normals in the volume elements and surface normals in surface elements and face neighbors. For tracing photon bundles through a particulate medium, the edge neighbors of volume elements are also required, as shown in Fig. 4.5.

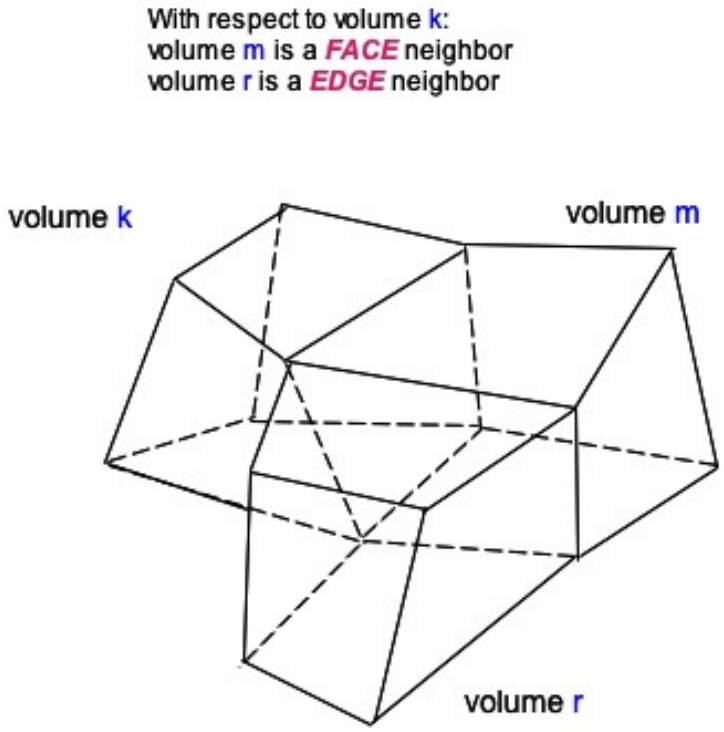


Fig. 4.5. Two main types of neighbors for a volume element

Algorithm 4.1 presents an outline of our implementation of the PMC ray tracing algorithm. Algorithm 4.2 presents an outline of how element-photon interactions can be determined. An illustration of this function is given in Fig. 4.6.

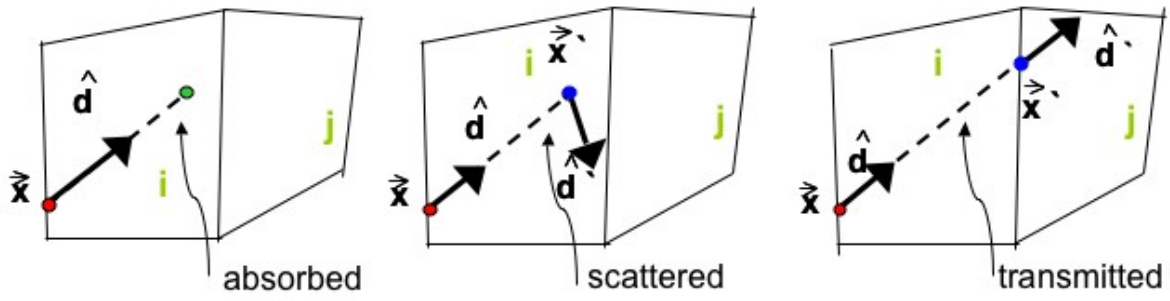


Fig. 4.6. Possible photon-volume interactions in a two-dimensional domain

4.4.1 Energy Partitioning

In the general PMC method, a ray of fixed energy is traced until absorbed. In the absence of a participating medium (where rays bounce from one surface to another), it will take $1/\alpha$ tracings before the ray is absorbed and contributes to the statistical sample (where α is the absorptance of the surfaces). To improve the efficiency of the tracings, the *energy partitioning* scheme was introduced by Sparrow at al. [38, 39] in 1973, and extended by Modest et al. [40–42] in 1980. In this method, every ray deposits a portion of its energy in the volumes and surfaces with which it interacts. When its energy becomes smaller than some predetermined threshold,

ALGORITHM 4.1 The PMC ray tracing algorithm

BEGIN

Let **N** be the number of rays to emit

Let **M** be the pointer to the mesh

Let **L** be the list of photon bundles to trace (initially empty)

Each photon data structure consists of a current position and direction ($\vec{x}$, $\hat{d}$)

for every element **e** in the mesh **M**

$\mathbf{E_e}$ = `compute_emissive_energy` (**e**)

$\mathbf{L_e}$ = `generate_list_of_photons`(**e**, **N**)

L.`append`($\mathbf{L_e}$)

endfor

while (**L** $\neq \emptyset$)

// pop next photon bundle from the photon list

photon = **L**.`pop`( )

// element number the ray is currently traversing

e = **photon**.element( )

// determine next starting point and direction of the ray

// also determine possible deposit of energy of the ray

if(**e**.elementType( ) = = SURFACE)

($\vec{x}'$, $\hat{d}'$) = `determine_surface_interactions` (**e**, **M**, **photon**)

else //if volume element

($\vec{x}'$, $\hat{d}'$) = `determine_volume_interactions` (**e**, **M**, **photon**)

endif

// if the ray is not absorbed, push it back to the list for further tracing

if($\hat{d}'$ $\neq$ NULL)

photon.PositionDirection( ) = ($\vec{x}'$, $\hat{d}'$)

L.`push`(**photon**)

endif

endwhile

END

ALGORITHM 4.2 Determining volume interactions

Determine_volume_interactions(**e**, **M**, **photon**)

e is a mesh element

M is a pointer to the mesh

BEGIN

$\vec{x}'$ = `determine_intersection` (**e**, **M**, **photon**)

n is the next element the photon will go through

// extract coordinates, faces and other data

Data = `get_element_data`(**e**, **M**)

// Q is the energy the photon will deliver to element e

//$\vec{d}'$ *is a new direction of the photon, different from the old one if the scattering is present*

(**Q**, $\vec{d}'$) = `radiative_ properties`(**e**, Data)

`update_mesh_energy`(**Q**, **e**)

// if direction is NULL, we do not track photon any more

if($\vec{d}' \neq$ NULL)

photon.element( ) = **n**

endif

return ($\vec{x}'$, $\hat{d}'$)

END

the remainder is dropped in the last element on the ray's path (to satisfy the conservation of energy).

If a ray, carrying energy E, hits the surface element at point X and the surface absorptance at that point is $\alpha(X)$, the ray deposits $\alpha(X)E$ energy onto the surface and leaves it with $(1 - \alpha(X))E$ energy. On the other hand, if the ray travels through a volume element from point X to point Y, we first calculate the absorptivity of this volume element, $\alpha_{\lambda} = 1 - exp\left(-\int_{Y-X} k_{\lambda} ds\right)$. Again, the ray deposits $\alpha_{\lambda} E$ energy and leaves the volume with $(1 - \alpha_{\lambda})E$ energy. Both PMC schemes are implemented in our software framework.

In the next chapter we describe a newly developed software framework that allows the implementation of a wide range of radiation models. In addition, we introduce new algorithms that enable efficient tracking of photon bundles through a particulate medium.

Chapter 5

A Software Framework for the Photon Monte Carlo Method for Radiative Heat Transfer

5.1 Implementation of Photon Monte Carlo Algorithm

As we observed in the previous section, the basic idea behind the PMC algorithm is simple and geometrical ray tracing is relatively straightforward to implement. There have been many software libraries that implement this algorithm, and we describe below two well known implementations.

- **MCML** Monte Carlo for Multi-Layered media [43]. MCML is used in medical applications for the simulation of light propagation in tissue. It uses a cylindrical mesh and the assumption of gray radiation. It is a powerful code in its specific field, but not for general use.

- **MNCP** A Monte Carlo N-Particle Transport Code was developed over many years at the Los Alamos National Laboratory. Over 250 people were involved in the development of this software. It is primarily used for applications in radiation protection and dosimetry, radiography, medical physics and reactor design. It includes various modules for handling different radiative properties and models, arbitrary meshes and variance reduction techniques. However, it is as yet not parallelized for distributed memory machines, and it is not an open source code. Thus, new modules cannot be added and the documentation and software availability is not in the public domain.

The problem with existing sequential software libraries that implement the PMC method is either in their lack of generality or in the fact that new models cannot be incorporated by the user. These two disadvantages can significantly reduce the performance and successful implementation of existing libraries into combustion applications. In this chapter we will describe the methodology designed for building a general, extensible and robust software framework, as well as new algorithms designed for tracing photon bundles through different types of domains. The sequential PMC library is capable of modeling RHT in problems with:

- arbitrary geometry with Cartesian mesh , structured or unstructured, tetrahedral, hexahedral or hybrid,
- reflective or periodic boundary condition,
- gray, non-gray radiation (non-gray functionality is not fully tested)
- isotropic scattering, and
- classical medium discretization (data given on mesh elements) or with particles.

The development of these algorithms is a part of the original work performed for this thesis, with the exception of the schemes for ray-particle interaction, developed by A. Wang at al, [44].

In the simulation of combustive processes, the PMC library is only a part of a large-scale software system that can include a number of different components, such as mesh generators and CFD solvers, that are influenced by the turbulence models and numerous methods for modeling chemistry. Different mesh generators employ various data structures and store different data about the mesh. To enable portability and to ensure that the PMC model is independent on the type of the mesh generator used, our PMC software system does not have its own mesh data.

Our PMC software system depends on the mesh generator used in other parts of the application, such as fluid flow calculations, to provide data about the mesh that is necessary for ray tracing. Therefore, there has to exist an interface between the PMC system and the mesh generator. This interface will query the mesh data structure and provide on demand data about the mesh. Table 5.1 illustrates the kind of data the PMC system needs about the mesh.

Data fields	**Data description**
VolGeometryData	Coordinates of the vertices, face normals, edge definitions
VolRadiativeData	Radiative properties of the element (for example, absorption coefficient,scattering coefficient, mole fraction of chemical species, temperature,pressure)
SurfGeometryData	Coordinates of vertices and surface normal
SurfRadiativeData	Radiative properties of the element (for example, emissivity, temperature)
ElementType	surface or volume
VolNeighbors	For each volume face, its neighbor and neighbor type (volume, surface, periodic or reflective boundary)
SurfNeighbors	surface neighbor

Table 5.1. A list of items PMC system needs from the mesh. The list is not comprehensive.

Since the Photon Monte Carlo software should be capable of handling any geometry with any type of radiation, it is clear that for each of the random number relations from Appendix B, there are many choices of how to perform calculations, depending on the nature of the problem. For example, if the properties are constant or cell-centered, the point of emission is very easily calculated by scaling generated random numbers. If the data in the cell are given on vertices, we

employ bilinear interpolation and calculate simple integrals to determine the point of emission. But for some methods, higher-order accuracy for the interpolation of temperature and absorption coefficient on a given cell is needed. For such methods, data points from neighboring elements may be required.

Similarly, a typical combustion application can treat turbulence and chemistry separately or employ some model that describes their interaction. Depending on the choice, the scalar and vector characteristics of the simulation domain may be given on the vertices of the mesh, in the centers of the mesh elements, or the entire domain may be discretized with particles that carry these properties, as is the case with Probability Density Function (PDF) methods. Therefore, a robust software implementation has to be able to handle any kind of mesh, structured or unstructured, and the various methods for describing the properties of the domain (e.g., discretization through mesh elements or field particles).

All these issues lead us to the fact that our software has to have ‘plug and play’ capabilities. Namely, the functions that determine random number relations are only given as placeholders. The user of the library can then ‘plug’ the functions that already exist in the library or she/he can write new functions that comply with certain established interface, depending on the nature of the problem. For example, in Algorithm 4.2 the function `radiative_properties()` can be written by the user. The overall structure of the PMC software is illustrated in the Fig. 5.1. In our library, the PMC Radiation part of the software framework implements the modules listed below.

- `pmcradEmissDir` modules that calculate emission point for cell-centered or constant properties and for properties given on cell vertices. These calculations can be performed on an arbitrary triangle, convex rectangle, tetrahedral or convex prism.

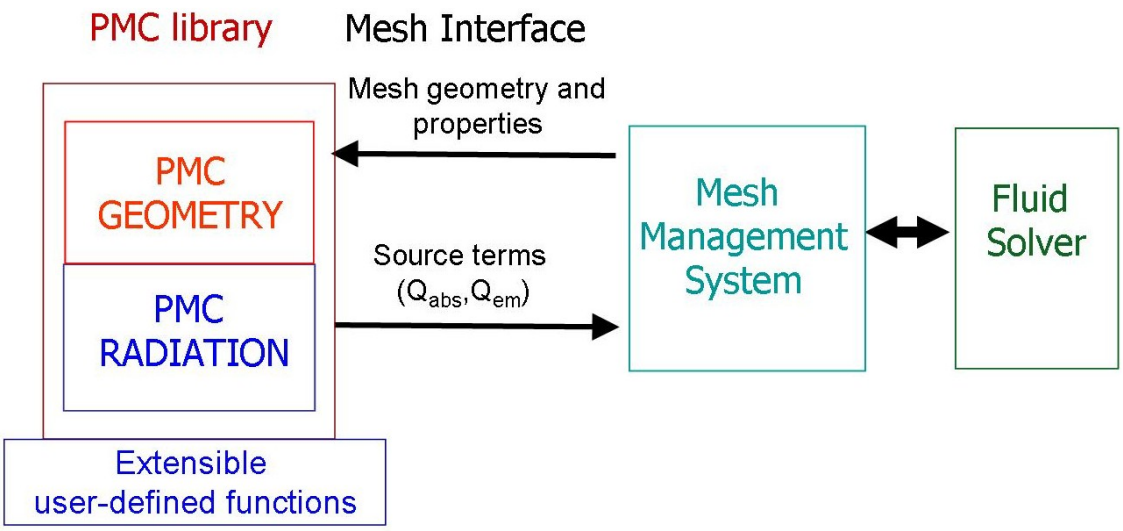


Fig. 5.1. A block diagram of the interactions within the PMC software framework

- `pmcradOpticalThicknessG` modules that calculate the optical thickness for gray radiation, based on cell-centered or vertex given absorption coefficient.

- `pmcradScatterDir` modules that calculate the scattering direction for linear anisotropic and isotropic scattering.

- `pmcradReflectGD` modules that calculate the reflection for gray, diffuse elements.

- `pmcradEmissDirPart` and `pmcradOpticalThicknessPartG` modules that calculate emission and optical thickness for particulate enclosures for gray radiation, as described in [44].

In the next section, we describe some of the features that are implemented in the PMC Geometry part of our software library.

5.2 Features of the PMC Software

5.2.1 Tracing a Ray Through a Medium with a Mesh-Based Discretization

In a medium with mesh-based discretization, the scalar and vector properties of the domain are discretized with respect to the mesh of the enclosure. These properties can be given on the vertices, cell centers or face centers of the mesh elements. This kind of domain discretization is used in Finite Element, Finite Volume or Finite Difference methods. The other type of the medium is particulate, where the entire domain is discretized with particles, independently of the mesh generation, and these particles are characterized with scalar and vector values. This type of discretization is used in PDF methods. In our software, we implemented ray tracing schemes

for both types of discretization. Here we describe methodology of ray tracing through a medium with a mesh-based discretization.

In PMC, a ray is uniquely determined by its starting position and its unit direction. Its starting position and possibly direction will change as we traverse from one element to another. When a ray traverses a volume element J, its starting point must be within the element or on the element boundary. There are three steps in tracing the ray through the volume J as described below.

1. We first determine the face F_j this ray may intersect on its way out of the element J, and the intersection point T on this face. We also determine the next volume element that the ray may traverse—this is the volume that shares the face F_j with the volume J.

2. If scattering is present, we first determine whether the ray will scatter or not. If the ray scatters, we calculate the point of scattering within the volume J. This point will be the new starting point of the ray. Next we calculate the new direction of the ray and, in the case of energy partitioning, how much energy it deposited to the volume before it scattered. After determining the new starting point and the direction, we restart tracing the ray through volume J (from the step 1).

3. If the medium has no scattering, or the ray has not scattered, we calculate how much energy the ray deposits in the volume J by first calculating the optical thickness along the path through the volume the ray traverses. If the energy that the ray carries drops below certain cutt-off threshold, the ray is terminated and its remaining energy, (if any), is deposited in the element. Otherwise, the intersection point T on the exit face is set to

be the new starting point for T and it traverses the neighboring volume K that shares the face F_j with volume J.

5.2.2 General Positioning of the Point of Emission

When determining the emission point of a ray or the intersection of the ray with a mesh element, it can happen that the point lies on the edge of the mesh element. In that case, as can be observed from Fig. 5.2, the next element the ray will go through cannot be identified based on the information we are extracting from the mesh. This problem is solved using the general positioning algorithm, which consists of calculating the center of the mass of the element and modifying the original starting point of a ray towards the center of mass. An illustration is given in Fig. 5.3.

5.2.3 Tracing a Ray Through a Particulate Medium

There are three possible ways of tracing rays through a medium filled with fluid particles, as illustrated in Fig. 5.4. They are developed by Anquan Wang at al., as discussed in [44] and are briefly described below.

- LS (Line-Sphere) treats a ray as a line (nondimensional photon beam) and a particle as a sphere of either constant or variable density. The ray interacts with the particle if it goes through the particle sphere.

- CP (Cone-Point) treats a ray as a cone and a particle as a point. The ray interacts with the particle if the ray-cone contains the point-particle.

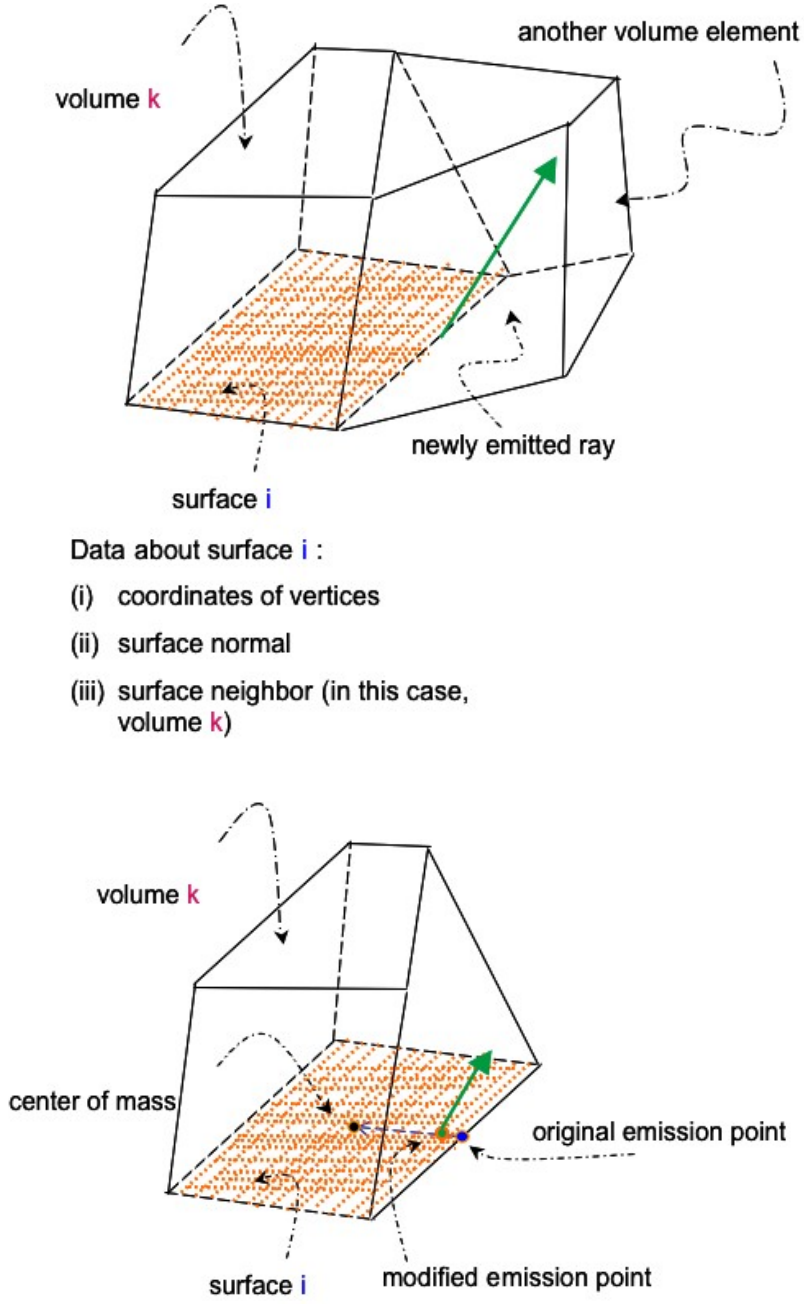


Fig. 5.2. A schematic of a ray that is emitted from a surface but it does not traverse the surface neighbor

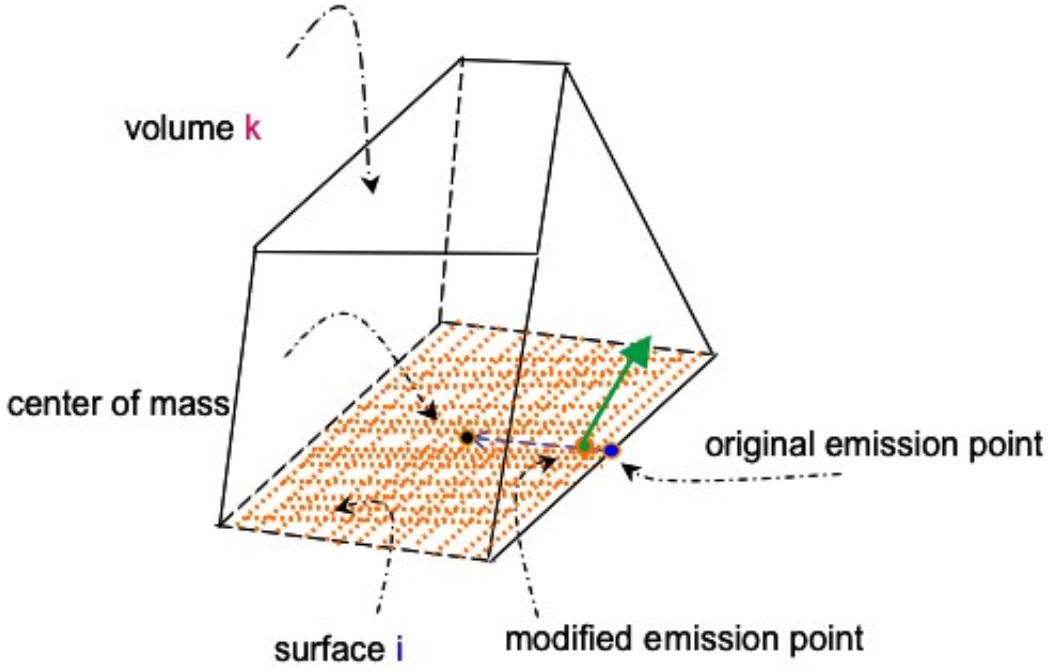


Fig. 5.3. A schematic showing the modification of the emission point so that now the ray traverses the neighboring volume

- CS (Cone-Sphere) treats a ray as a cone and a particle as a sphere of either constant or variable density. The ray interacts with the particle if the cone intersects the particle sphere and if the particle is ‘downstream’ from the rays starting position in a given cell.

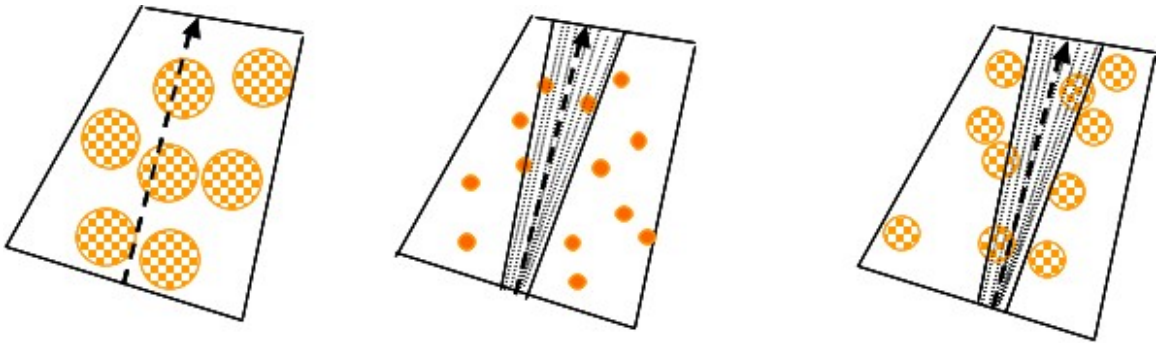


Fig. 5.4. A two dimensional representation of the three different models for tracking photon bundles through a particulate medium: Line-Point (left), Cone-Point (center), and Cone-Sphere (right)

For tracing rays through a particulate medium, the straightforward approach is to trace the ray element by element and calculate the interactions between the ray and fluid particles in a given element. However, this way many particles that interact with the ray may be omitted from the calculation, as illustrated in Fig. 5.5 for the LS scheme, thus leading to erroneous results.

To solve this problem, we propose maintaining a list of the *ghost particles* for each volume element k. Ghost particles are the particles that belong to the neighboring volumes of volume k, but they may potentially interact with the ray that traverses volume element k. If R_{part} is the radius of a given particle P and R_{max} is maximal radius of the cone (if the ray is

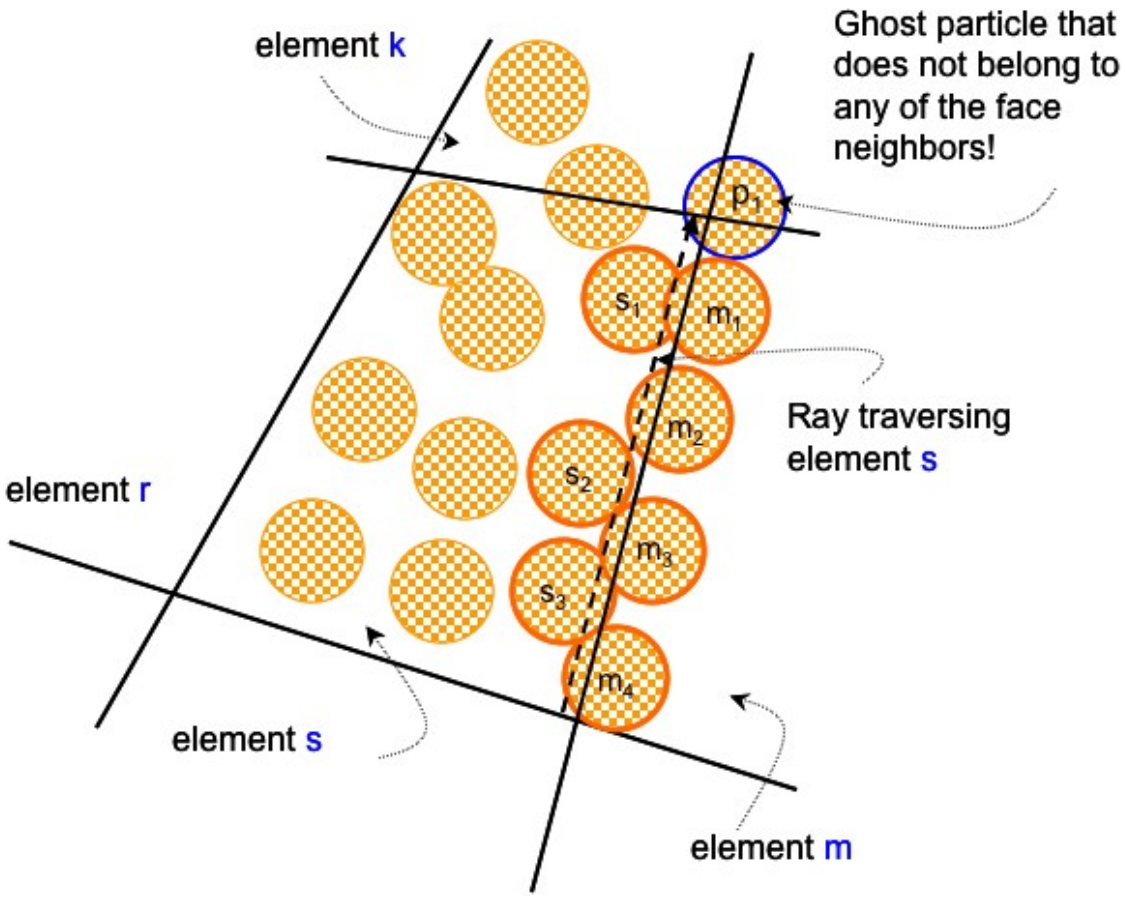


Fig. 5.5. A two-dimensional schematic of the interactions of the ray with particles while it is traversing the volume element **s**. If only particles in the cell that the ray is currently traversing are considered for interaction, particles s_1, s_2 and s_3 will be correctly identfied, but particles p_1,m_1, m_2,m_3, and m_4 will be missed.

treated as a cone) we introduce a new variable, called *critical distance d*: $d = R_{part}$ in the LS case, $d = R_{max}$ in the CP case, and $d = R_{part} + R_{max}$ in the CS case. Then, we observe that the particle P will be included in a list of the ghost particles for a volume k if the distance between the particle P and some face of volume k is smaller than the critical distance d. This procedure is performed each time the PMC function is called, since the positions of the particles may change. While for tracing the rays we only needed information about face neighbors of a given volume element, to accurately determine ghost particles, we have to consider edge neighbors as well (as can be seen in Fig. 5.5 where the ghost particle p_1 belongs to the edge neighbor).

With the introduction of ghost particles, we modify our algorithm for calculating the ray/particle interaction. Instead of considering only particles that belong to the cell that the ray is currently traversing, we also consider ghost particles for that cell. However, there are two issues with this approach as well. First, if rays are treated as cones and the maximal cone radius is big, the list of ghost particles for a given cell may also be large although some of them will never actually interact with rays in a given cell. If this is the case, we perform many unneeded calculations that check whether there is an interaction between a ray and a particle and this can significantly reduce the performance of the ray tracing. The second problem is that with this approach, some particles may be considered twice, thus unnecessarily increasing their energy level, as can be observed in Fig. 5.6.

To solve the first problem, we determine from which neighbors we should consider ghost particles for ray/particle interaction. In Fig. 5.5, it is obvious that ghost particles from element r cannot interact with the ray. Therefore, we calculate the distance between ray and faces of the volume element. If the the distance from the particular face is smaller than the critical distance

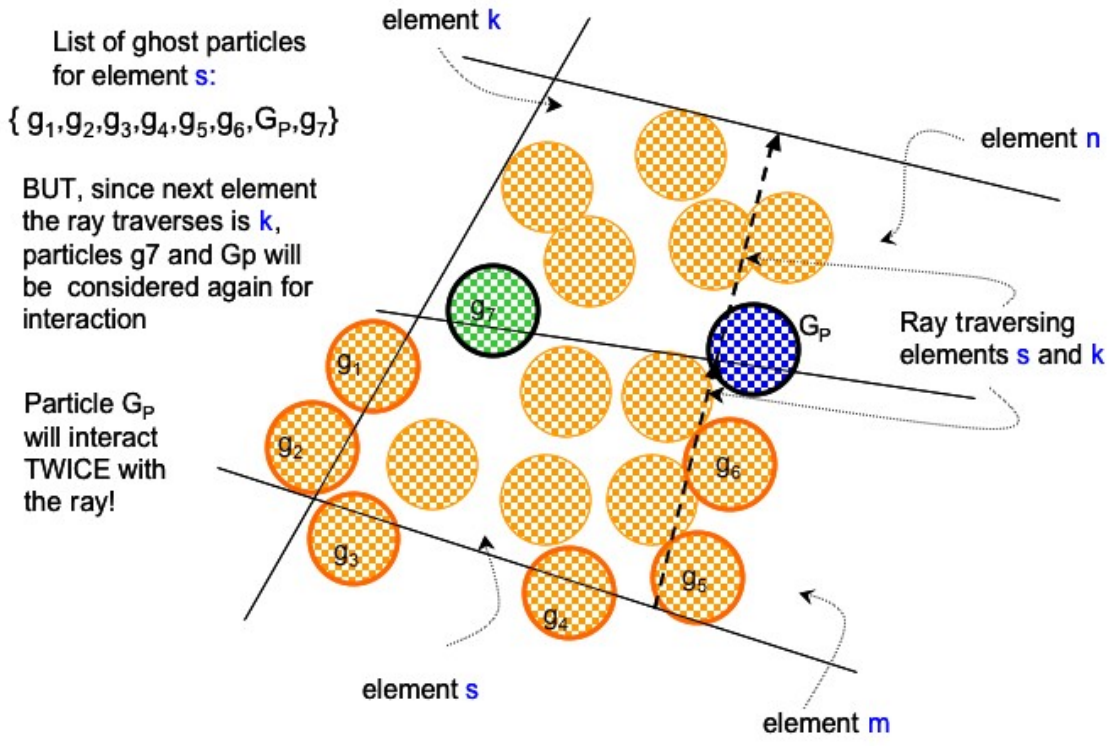


Fig. 5.6. A two-dimensional schematic of an interaction of the ray with particles while it is traversing a volume **s**. If all ghost particles are considered for the interaction with the ray, then some particles may erroneously interact more than once.

d, we should consider ghost particles from that face neighbor. Moreover, if there are two faces that share an edge E and the distance between ray and these faces is smaller than d, we should also consider ghost particles from edge neighbors corresponding to edge E.

To solve the problem of redundant particle interactions, we must further limit the list of neighbors from which we should consider ghost particles for interaction. If *f_start* and *f_end* are the faces the ray is intersecting on its way through the element, then we should not consider the ghost particles from the face neighbor corresponding to the faces *f_start* and *f_end*, since those particles are either going to be considered next when the ray traverses the neighboring volume or have been considered in the previous volume ray was traversing. From the same reasons, we should also not consider ghost particles from the edge neighbors that correspond to edges of the faces *f_start* and *f_end*. An illustration of these two limitations can be found in Fig. 5.7.

5.3 Validation

We validate the accuracy of our algorithms on a problem of modeling the radiative heat transfer between two infinite parallel plates, as illustrated in Fig. 5.8. For this problem there is a well established numerical solution.

For the case without scattering, we compare the relative error between the numerical solution for the heat fluxes in the participating medium and the solution calculated with the PMC in Fig. 5.9. For the problem with isotropic scattering we compare the numerical solution of the heat fluxes of the participating medium with the solution calculated with PMC, in Fig. 5.10. The relative standard deviation of these calculations was in the order of the relative error between the numerical solution and the PMC solution. From these comparisons,we conclude

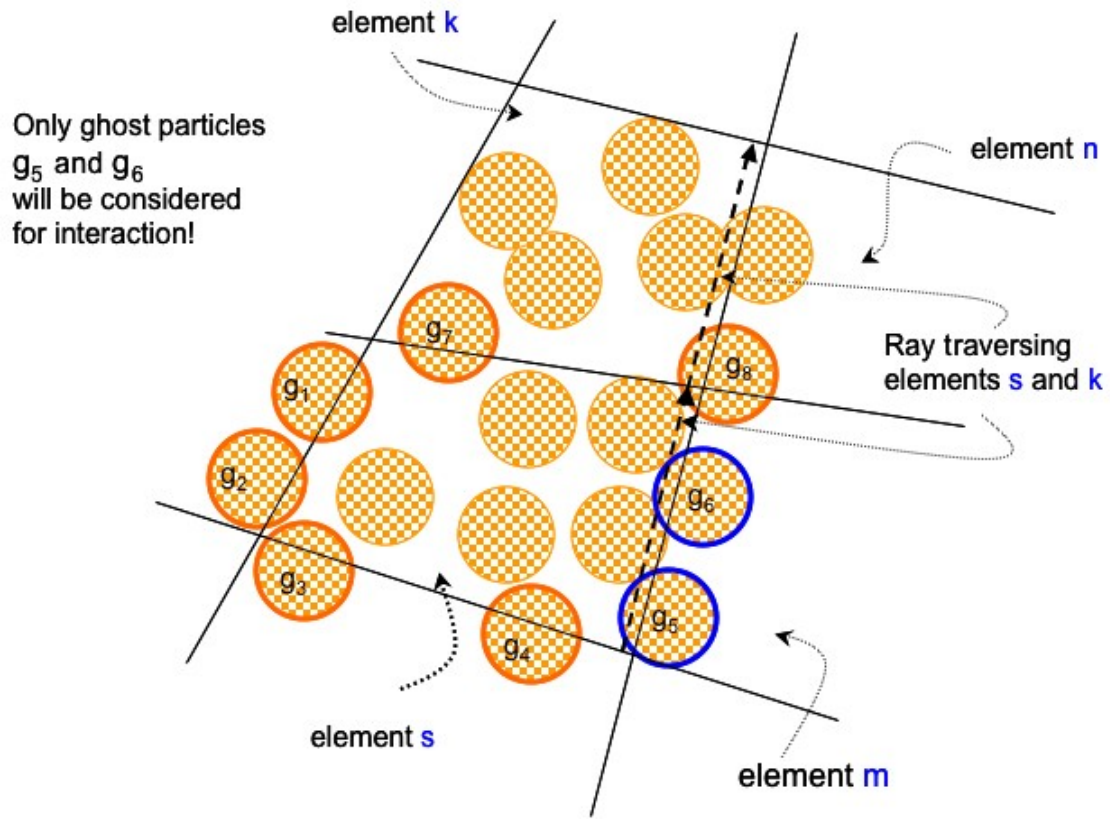


Fig. 5.7. A two-dimensional schematic of the limitations on which ghost particles should be considered for ray/particle interaction

that our PMC implementation can predict the heat fluxes for this problem with a reasonable accuracy.

This concludes the chapter about the sequential implementation of the PMC software. In the next chapter, we describe the implementation of the parallel PMC software framework and analyze its performance.

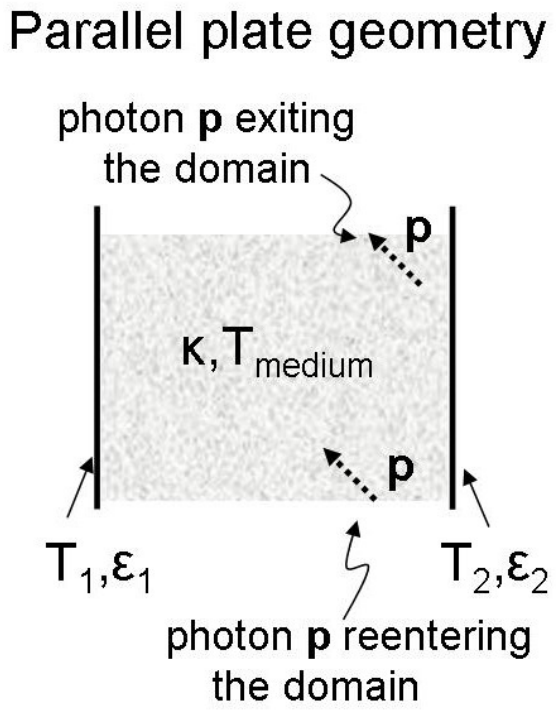


Fig. 5.8. The geometry of the parallel plate problem

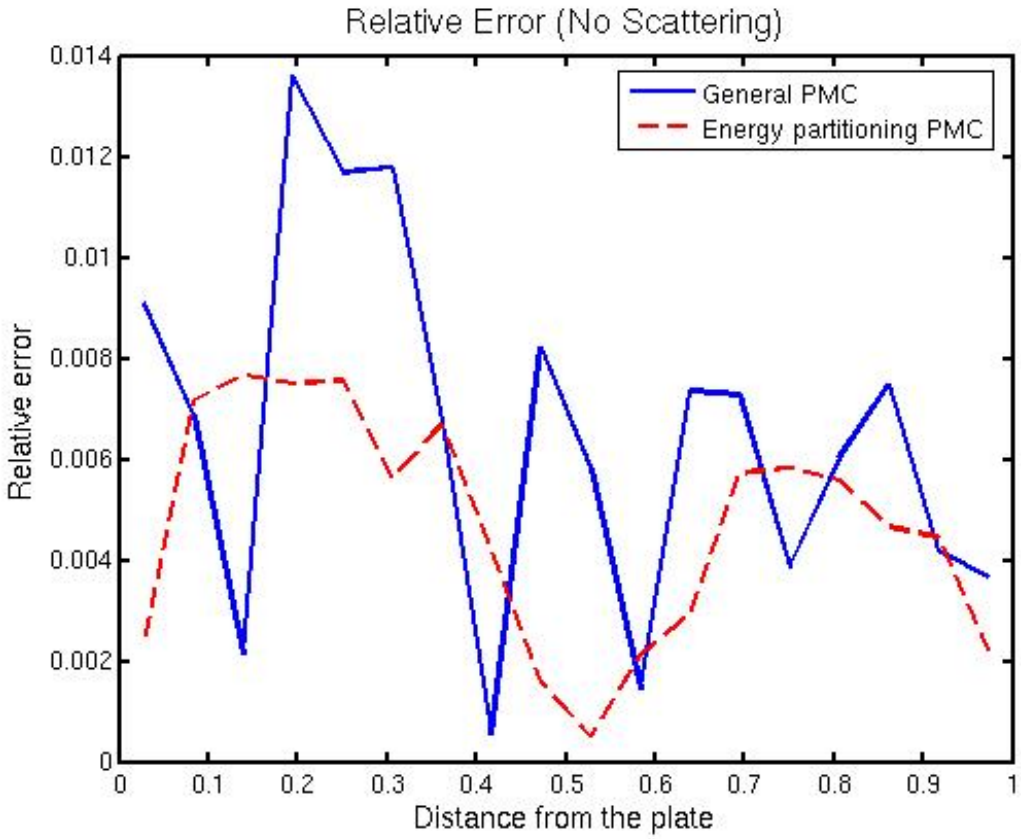


Fig. 5.9. The parallel plate problem without scattering: relative error between the well established numerical solution and PMC solution for general PMC (solid line) and energy partitioning (dashed line).

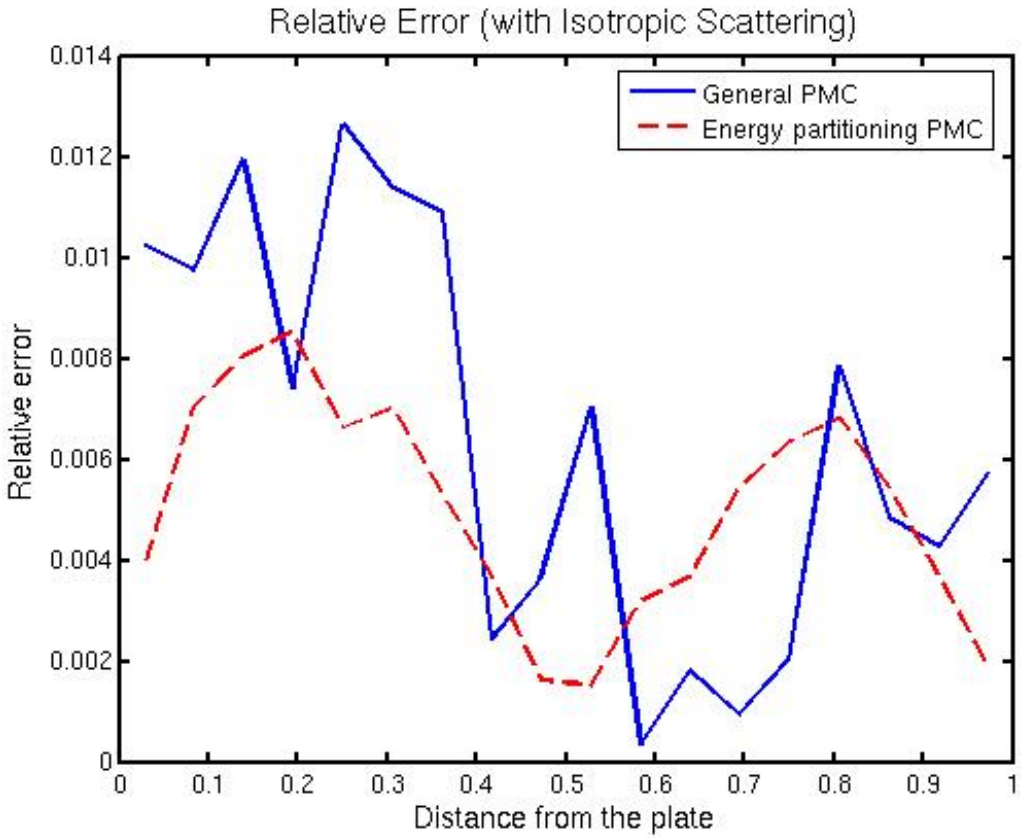


Fig. 5.10. The parallel plate problem with scattering: relative error between the well established numerical solution and PMC solution for general PMC (solid line) and energy partitioning (dashed line).

Chapter 6

Parallelization of the Photon Monte Carlo Algorithm

For conventional Radiative Heat Transfer (RHT) methods, such as spherical harmonics or discrete ordinates, there are readily available parallel software implementations that usually apply angular, spatial or spectral parallel decomposition to divide the computational load among the processors. For more details, see [45, 46]. The efficiency of these software implementations depends on the discretization method as well as on radiative and geometric properties of the domain. However, when considering parallel software for PMC ray tracing methods, existing software packages are usually either designed to solve a particular type of radiation problem [47, 48], or they are tailored for specific parallel architectures that are not readily available to a wide scientific community. One example of such software is that designed by Burns at al.[49] for Cray X-MP/Y-MP architecture.

In this chapter we describe how we incorporated extensibility of the sequential code described in the previous section into the parallel software for distributed memory systems. The development of these algorithms is a part of the original work performed for this thesis. When developing parallel PMC algorithms we have two broad choices for the parallelization of data: ray partitioning and domain partitioning. In the ray partitioning method, the total number of rays is divided among the processors while the simulation domain may or may not be partitioned, depending on the other segments of the simulation (such as the fluid dynamics solver). In the

domain partitioning method, the simulation domain is partitioned and every processor emits rays from its portion of the mesh [1]. We consider these two approaches in the following sections.

6.1 Parallelization Through Ray Partitioning

In the ray partitioning method, the total number of rays is divided among the processors. A processor tracks each of the assigned rays through its entire path until it is absorbed (in standard PMC) or its energy is depleted (in energy partitioning PMC). An advantage of this method is that a good load balance between processors can be achieved with a reasonable assignment of statistically independent rays to the processors.

The problem with this method lies in the fact that tracking of rays relies on the underlying discretization of the enclosure. With ray partitioning every processor either has to have the entire mesh structure in its memory, or it has to have intensive communication with the processors that own the mesh elements that are traversed by the rays. In the case when mesh data structure can fit on the memory of a single processor this method is feasible and preferable, since it introduces minimal communication overhead. If the mesh data structure has to be distributed and the entire computational domain is optically thick, this method can also be feasible since every processor has to keep a constant, small number of ghost cells that the rays may go through. However, with a distributed mesh and a participating medium that is fully or partially optically thin, this method cannot be used because of the large communication overhead.

6.2 Parallelization Through Domain Partitioning

For large-scale simulations the domain discretization has to be very fine to accurately encompass all of the physical phenomena occurring during the simulation. This implies that the

mesh data structure can be very large and may possibly not fit on the memory of one processor. Moreover, the situation in which the entire simulation domain is optically thick rarely occurs. Therefore, ray partitioning is not a feasible method to use for parallelization of PMC when applied to such cases.

In the domain partitioning method, the underlying mesh is divided among processors and every processor emits rays from its portion of the mesh. When the ray crosses the mesh boundary, it is communicated to the neighboring processor. To minimize the communication cost, rays that are to be communicated to the neighboring processors are packed into a contiguous message. This message is sent after it reaches a certain predetermined size or until all rays in the domain are processed. Simultaneously, packets of rays are received from other processors. Therefore, we have two alternating phases of computation, a calculation phase and a communication phase. Both phases are executed in turn until all the rays in the global domain are absorbed. Algorithm 6.1 presents an outline of our implementation of the parallel Photon Monte Carlo ray tracing algorithm. Algorithm 6.2 presents an outline of the communication procedure between processors.

As mentioned above, this method should be used when the underlying mesh data structure is large and has to be distributed among all processors. The domain partitioning is inherited from the mesh generator and, therefore, no special software is needed. Using the mesh partitioning with appropriate weights, we can resolve the issues of communication overhead and memory scalability, The computational load has to be distributed according to the physical properties of the domain. If one part of the enclosure is hot, it emits more rays than regions with lower temperature, since the emitted energy depends on the fourth power of the temperature. In this case, the processor that owns the hot, emitting region is overwhelmed with work while the processor

which owns the cold region has much fewer number of rays to track. However, the computational load depends not only on the number of emitted rays from a given subdomain, but also on the number of mesh elements that are traversed by the rays. If the subdomain is optically thick, rays travel short distances and the average number of elements a ray traverses is small. The opposite holds for optically thin media. The optical thickness of the medium depends on the absorption coefficient, and in most of the cases the absorption coefficient decreases as the temperature increases.

The issue of the distribution of the computational load among the processors is called load balancing. In every parallel application with extensive communication, it is very important that the computational load is well distributed. Otherwise, the imbalance in the computational load may result in substantial time that processors spend idle waiting for work. The problem of load balancing can partially be solved with adaptive domain decomposition, which ensures an even distribution of computational load among the processors.

Unfortunately, the domain decomposition has to be balanced with the CFD program. Furthermore, this balancing can be a complicated procedure. Considering that the number of photon bundles emitted from a subdomain is proportional to the $\kappa_P T^4 V$, (where T is the average volume temperature, κ_P is the Planck mean absorption coefficient, and V is the subdomain volume), this issue can be mitigated by the use a partitioning algorithm that includes estimates for the number of emitted rays for a given subdomain in the partitioning strategy. However, because the properties of the subdomains may change during the execution of the application and the mesh repartitioning may be costly, the parallel PMC software should be able to provide a reasonable efficiency even if the partitioning is not sufficiently well balanced with respect to the

PMC computation. In Section 6.5 we present some examples of various partitioning strategies and their impact on parallel performance.

The parallel overhead is another metric we have to consider. It is defined as the ratio between the overall execution time and the communication time. Reducing parallel overhead is the most important issue in the implementation of any parallel algorithm that has extensive communication requirements. In the following subsection, we are going to analyze the parallel overhead issues for simplified cases, where the entire domain is either optically thick or optically thin and for a domain that is a combination of optically thick and thin regions.

6.2.1 Parallel Overhead for Domain Partitioning

In this simplified analysis, we estimate the communication overhead (the ratio of the computational load and the communication load) for optically thick and optically thin medium. The communication load is measured by the number of rays that are communicated to neighboring processors and the computational load is measured by the number of elements each ray traverses before it is absorbed. We suppose that a mesh partitioner used for domain decomposition produces subdomains with bounded aspect ratios. Namely, we assume that the global mesh in an arbitrary d–dimensional space contains M cells and that there are P processors in our parallel environment. Then, each processor receives a subdomain containing M/P cells. Since the subdomains have bounded aspect ratio, the characteristic length of the subdomains is $L = O((M/P)^{1/d})$ and the surface of the subdomain is $O(L^{(d-1)/d})$. In addition, we assume that we emit exactly N rays from each cell and that the medium is diffuse. Moreover, boundaries of the subdomains are isotropic, namelly all subdomains are surrounded by other

ALGORITHM 6.1 A parallel Photon Monte-Carlo ray tracing algorithm

BEGIN

Let **myrank** be the rank of the processor

Let **N** be the number of rays to emit

Let **M** be the pointer to the mesh

Let **L** be the list of photon bundles to trace (initially empty)

Let $\mathbf{S_p}$ be the list of photon bundles to send to neighboring processor **p**

for every element **e** in the mesh **M**

$\mathbf{E_e}$ = `compute_emissive_energy` (**e**)

$\mathbf{L_e}$ = `generate_list_of_photons` (**e**,**N**)

L.`append`($\mathbf{L_e}$)

endfor

while (**L** $\neq \emptyset$)

photon = **L**.`pop`( )

e = **photon**.element( )

($\vec{x}'$, $\hat{d}'$) = `determine_object_interactions` (**e**,**photon**)

if($\hat{d}'$ $\neq$ NULL)

photon.PositionDirection( ) = ($\vec{x}'$, $\hat{d}'$)

if(**photon** at the border with processor p)

$\mathbf{S_p}$.`push`(**photon**)

else

L.`push`(**photon**)

endif

endif

`exchange_photonbundles`( )

endwhile

END

ALGORITHM 6.2 Communication between processors

```
exchange_photonbundles( )
BEGIN
    foreach(neighboring processor p)
        if(S_p has sufficient photon bundles)
            send_package(B_p , p)
        endif
        if(there is a package from p)
            R_p = receive_package (p)
            L.append(R_p)
        endif
    foreach
END
```

subdomains (there are no boundary walls—every ray that reaches the border of the subdomain continues to travel through the subdomain of the neighboring processor).

Each ray on average travels K cells—in the optically thick case K is a small constant number and in the optically thin case K is a factor of the characteristic length of the subdomain. With the parameters stated above, the computational load $\mathcal{L}_{CPU}$ is of order of: $\mathcal{L}_{CPU} = O(N \cdot K \cdot M/P) = O(N \cdot K \cdot L^d)$. To estimate the communication load, we observe that only rays that are at most K cells away from the boundary of the domain can be communicated. Since the surface of the boundary is $O(L^{(d-1)/d})$, we can say that at most $O(N \cdot K \cdot L^{d-1})$ will cross the boundary. Hence, the total number of communicated rays $\mathcal{N}_C$ is of order of: $\mathcal{N}_C = O(N \cdot K \cdot H^{d-1})$. Therefore, for both optically thin and optically thick cases, the communication overhead is $O(\mathcal{H})$, of order of the dimension of the subdomain. It does not depend on K, the thickness of the medium or on N, the number of the rays. In the case of an optically thick medium, the number of rays that are communicated is small, but the computational load involved in the tracking of rays is also small since the rays do not travel far. On the other hand, for an optically thin medium, both communication load and computational load are large.

However, the parallel overhead is different from the communication overhead since it also incorporates the time the processor spends idle, waiting for messages containing photons on which to work. In the case of the optically thin medium, rays travel through many subdomains. After a certain period of time, if the medium is not homogeneous or if the boundaries of the domain are not isotropic, it can happen that there is a small number of rays that are still active. The small number of rays implies the small computational load and a large number of small messages that have to be passed. As these remaining rays are traversing the global domain, they are preventing processors from declaring that the tracking is completed. This is called the *end*

game, in which processors wait idle for small chunks of work. After short period of computation, the processors then pass onto their neighbors until all the rays are absorbed. Therefore, in the case of the optically thin medium, the overall parallel overhead can be significant. In Fig. 6.1 we compared the parallel overhead for the parallel plate problem described in the previous section for the optically thick and optically thin media. The description of the problem is given in the next section. The participating medium in the problem which we analyzed has constant properties and all the rays travel approximately the same distance. Therefore, the parallel overhead should not be significant. However, we still observe that the parallel overhead is larger by a factor of two when compared to the optically thick case.

However, in Fig. 6.2 we observe that parallel overhead can be even more significant for the problems where the participating medium has variable properties. Here we artificially introduced two layers in the medium between the two plates—one layer is optically thin and the second one is extremely optically thick. The problem of the parallel overhead cannot be solved with the perfect mesh partitioner, since the computational load can be balanced and parallel overhead can still be large. In our analysis of communication overhead, we implicitly assumed that the time it takes to track one ray through a volume is equal to the time to communicate one ray. The fact that this may not be correct does not affect the communication overhead since it just contributes a constant factor to the overhead. However, it can significantly contribute to the analysis of the parallel overhead. To demonstrate this fact, we conducted two series of simulations for the parallel plate problem with a mixed medium. For one simulation sequence, we introduced a scattering coefficient, thus making the time a ray spends traversing a cell much larger due to the higher probability of scattering within a cell. In Fig. 6.3 we show results comparing the correlation between the relative load balancing factor and the parallel overhead factor

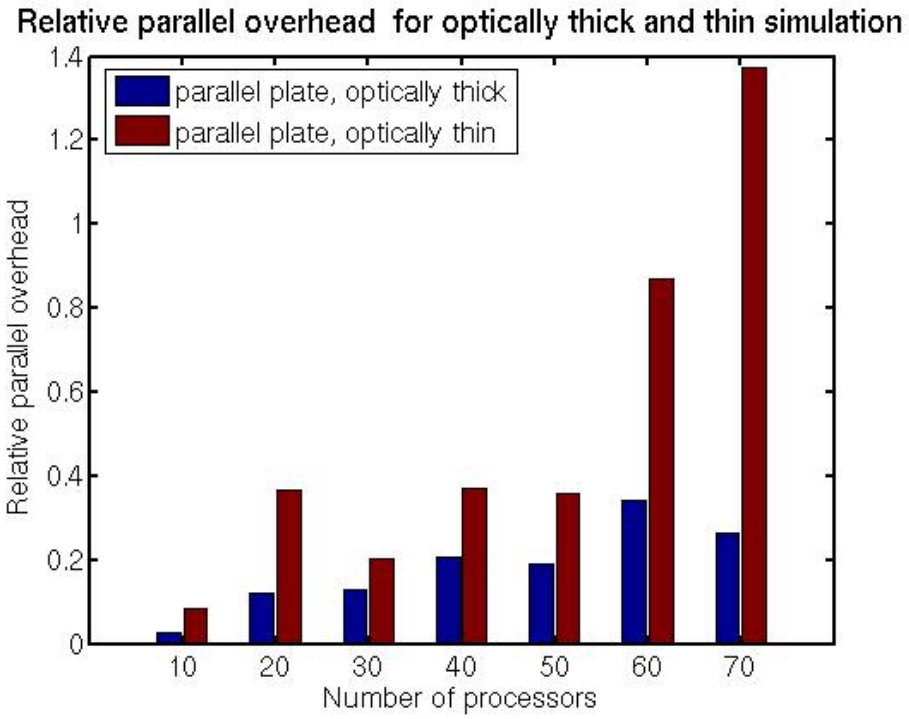


Fig. 6.1. A comparison of the relative parallel overhead between simulations of the parallel plate problem with the optically thin and the optically thick medium

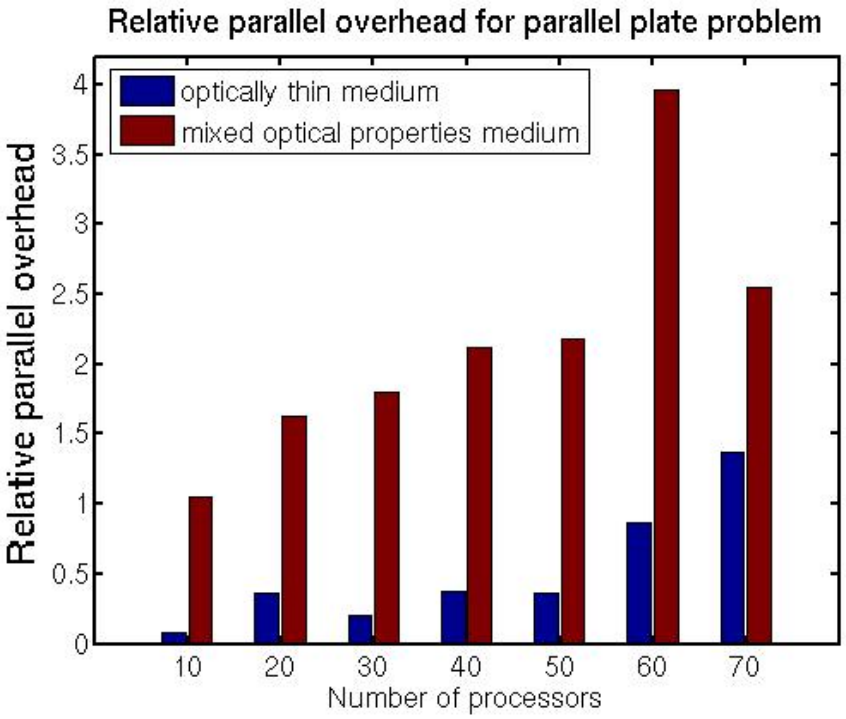
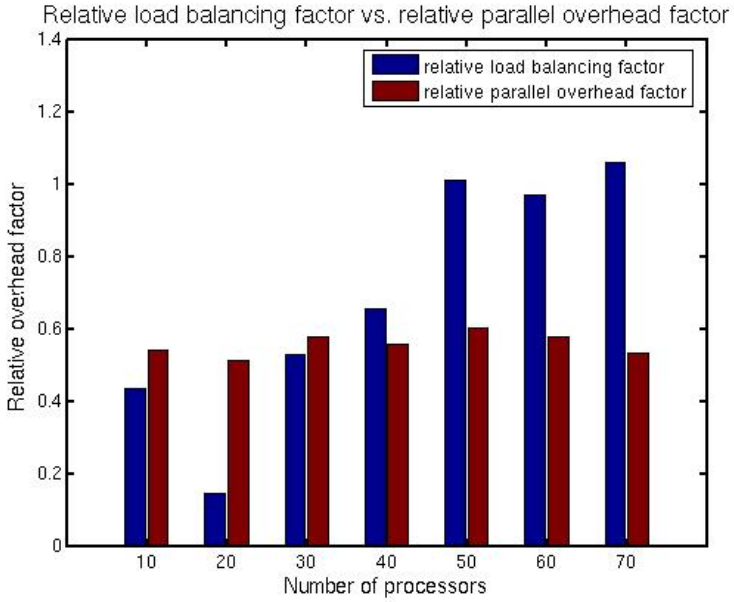


Fig. 6.2. A comparison of the relative parallel overhead between simulations of the parallel plate problem with an optically thin and the medium with variable optical thickness

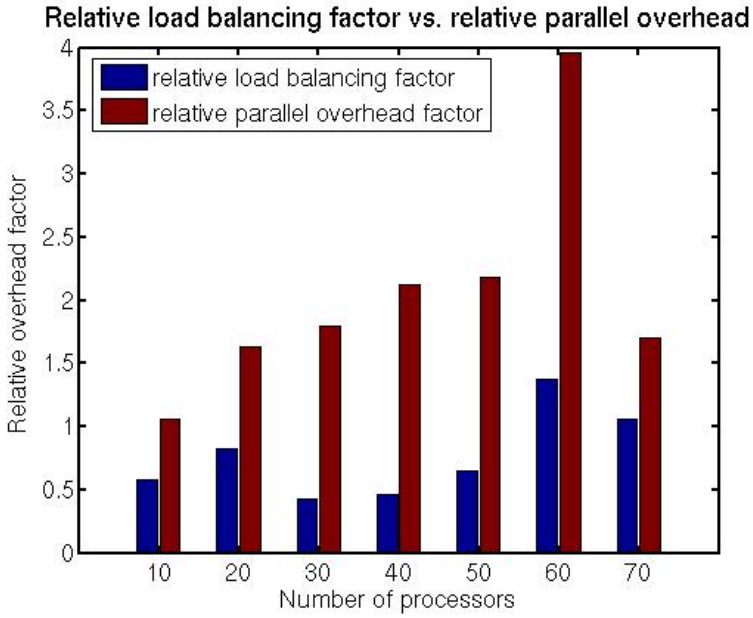
for the case where rays spend a relatively short period of time traversing the cell (no scattering) and where this time is much larger (case with scattering). We can see that parallel overhead in the case when scattering is present is directly correlated to the relative load balancing factor since and the idle time is almost eliminated. However, this is not the case for the simulation when scattering is not present. If the computation is expensive relative to communication, processors spend much more time in the computation phase. This computation to communication ratio compensates for the idle time that causes the parallel overhead. In this case, the parallel overhead is not totally eliminated but it is greatly reduced. Tracking a ray through a cell in the medium without scattering involves a small number of floating-point-operations. For a medium where the strong scattering is present, tracking a ray through a cell can be expensive, since the ray will scatter many times within the cell before it continues to traverse the neighboring cell. Therefore, if PMC is used for simulations of RHT in media with strong scattering properties, the parallel overhead is not going to be significant.

6.3 Hybrid Partitioning

Alternatively, if there are more processors available than needed to store the mesh data structure, we combine the two previously described approaches. If we have a total of P processors, we break them up into L groups, each containing K processors. The mesh is partitioned into K subdomains and every of the L groups receives the same partition. Therefore, within one group, we are performing the domain partitioning. The *i-th* processor in every of the L groups has the same subdomain—we divide the computational load of ray tracing through the subdomains across the L groups, thus introducing ray partitioning, as illustrated in Fig. 6.4. This method is not scalable since all other calculations (for example, fluid and chemistry calculations)



(a) The parallel plate problem with scattering, expensive optical thickness calculation



(b) The parallel plate problem without scattering, inexpensive optical thickness calculation

Fig. 6.3. A comparison of the relative load balancing factor and relative parallel overhead for the parallel plate problem with mixed optical thickness for the cases when optical thickness calculation is expensive (top) and inexpensive (bottom)

will be performed on only one group of processors while processors in other groups will remain idle.

6.4 The Choice of Random Number Generators

For the successful implementation of parallel PMC, it is essential to use random number generators that offer uncorrelated, uniformly distributed sequences of random numbers with large periods not only for individual processor but also across the parallel environment. Moreover, due to efficiency issues, the random numbers should be generated independently on each processor. In our parallel implementation we used a method called a *splitting technique* [50] to generate random numbers on parallel systems . The splitting technique partitions the original sequence into P very long consecutive blocks where every block is defined by a unique seed. Each of our P processors is then assigned a different block. Since the Mersenne Twister has a very large period [36], we can choose the initial values (the seeds) randomly and obtain as many subsequences as we need. It is highly improbable that any two processors will use the same seeds or that the two subsequences will overlap [51].

6.5 Computational Results

To demonstrate the capabilities of the parallel PMC software framework, we constructed two simple experiments, whose geometries are illustrated in Figures 5.8(in the previous chapter) and 6.5 (see below). The first experiment models the radiative heat transfer between two infinite parallel plates described in the previous chapter, and the second example computes one step of PMC during a simulation of a combustion process inside a jet flame. For the parallel plate experiment, the properties inside the medium and on plates are constant and the notion of the

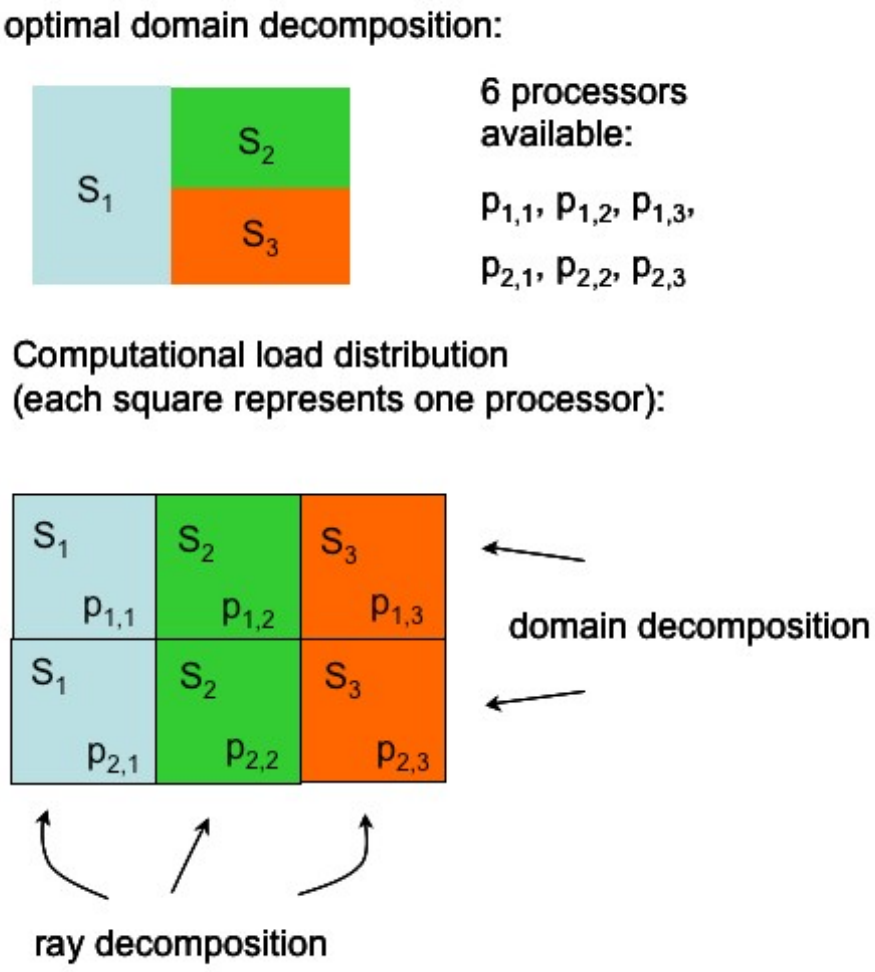


Fig. 6.4. An illustration of how the hybrid partitioning works. The number of processors P is 6, the number of groups L is 2, and the mesh is partioned into 3 subdomains (K is 3).

'infinite' plates is modeled by *periodic* boundary conditions (the photon bundle that exits the domain on one side, reenters it on the opposite side, as shown in Fig. 5.8) in the previous chapter.

For the jet flame experiment, the dimensions of the modeled jet flame are : jet radius = 0.005m, jet length = 1.8m, and the radius of the outer cylinder = 0.225m, as illustrated in Fig. 6.5 with the temperature profile given in Fig. 6.6. The walls of the engine are set to be black at the local temperature. Since the combustion of the jet flame is axisymmetric, we only model one quarter of the entire space. Hence, on the axis planes we introduce *reflective* boundary conditions—photon bundles that reach the axis planes reflect themselves back into the simulation domain. The temperature profile and concentrations of mass species were taken from [5]. The Planck mean absorption coefficient profile for the gray case is calculated from the formula presented in the Sandia workshop on measurement and computation of turbulent nonpremixed flames, [52]. The absorption coefficient for the nongray case is calculated from the database of narrow-band k-distributions for water vapor and carbon dioxide, as described in [53].

For both experiments, a structured hexahedral mesh is used to discretize the participating medium. Also, the mesh-based discretization of the domain properties is used where all the scalar quantities (such as temperature, and concentrations) are given at cell centers. For the parallel plate problem we only considered the gray radiation case. For the jet flame problem we consider both cases of gray and nongray radiation.

Since the amount of work grows linearly with the overall number of photon bundles that are tracked through the domain, as we increase the number of processors, we proportionally increase the number of photon bundles and keep the number of cells constant. This approach enables us to examine the scalability of the PMC implementation. We used the Parametric Binary Dissection (PBD) [54] algorithm which minimize the difference for a chosen metric between the

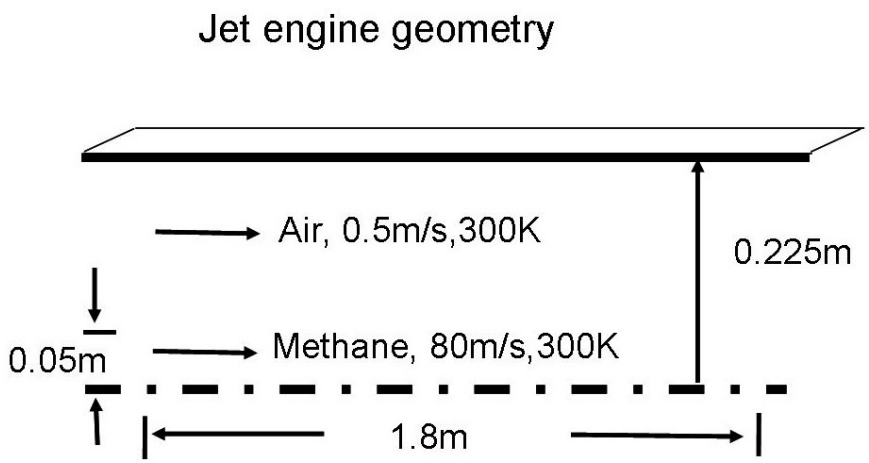


Fig. 6.5. The geometry of the jet flame, from [5]

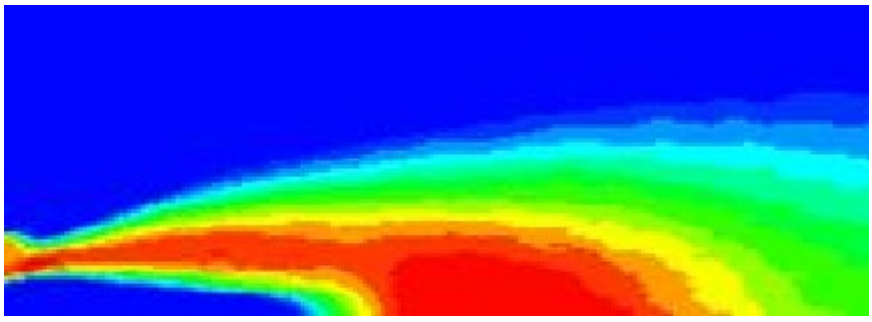


Fig. 6.6. The temperature contours at one time-step in the simulation of the jet flame [5]

two subregions it generates at each step of the algorithm. An illustration of how PBD works is given in Fig. 6.7. There are four types of metrics used in the numerical experiments that are presented in this section and they are described below.

The aspect ratio of the subdomain. This metric is used as a model for communication load for the subdomain.

The volume of the subdomain. This metric is used as a model for the computational load of the subdomain. It represents the computational load of fluid calculations fairly accurately but it can fail completely when predicting the load for radiation calculations. The volume and aspect ratio metrics are combined together into the *geometric* metric.

The emissive energy of a subdomain. If we assume that the subdomain contains M volumes and N surfaces, the total emission can be calculated in the following way:

$$E = \sum_{i=0}^{M} 4\kappa_P \sigma T^4 V_i + \sum_{j=0}^{N} \epsilon \sigma T^4 A_j, \tag{6.1}$$

where V_i is the volume of the volume element i and A_j is the area of the surface element j.

The CPU metric. This metric represents the CPU time spent tracing rays through the subdomains. This metric cannot be, precisely determined, but it can be approximated in the following way. We first run the sequential PMC on a coarse mesh and measure the time spent for calculations in each volume and surface element. Then, we extrapolate the CPU times for each element of the coarse mesh onto the fine mesh that we want use in our

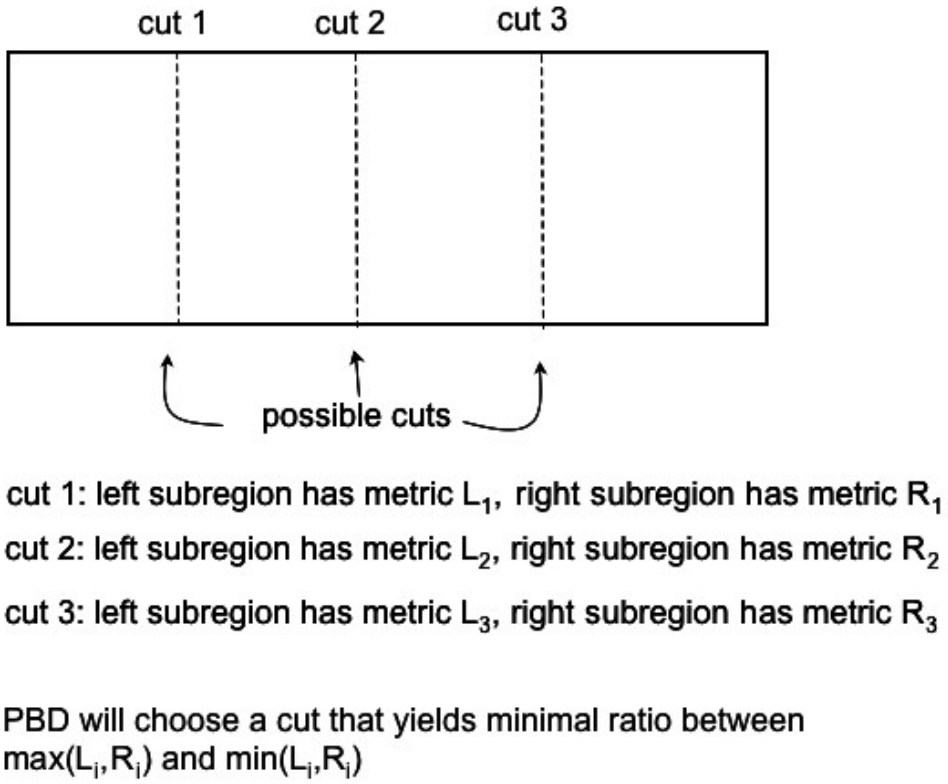


Fig. 6.7. A two-dimensional illustration of how the Parametric Binary Dissection algorithm works

parallel simulations. We use trilinear interpolation with the respect to the element's center of mass to derive CPU times for the fine mesh.

We can also combine these metrics into a composite metric (such as combining the aspect ratio and volume metric into a geometric metric). We analyzed five possible metrics for PBD: geometric, CPU metric, emissive energy, combination of CPU and geometric metric, and combination of emissive energy and geometric metric.

In both cases, the combustion simulation supplies the PMC functions with the temperature and absorption coefficient field, and the PMC library outputs the absorbed energy for each element in the domain. In the case of the jet flame simulation, the flame is optically thin, so we expect that extensive communication between processors will take place. Also, if we observe the temperature profile in Fig. 6.6, we can see that we have a region with relatively high temperature and a region that is relatively cold.

We tested our software on a Beowulf cluster with 94 dual processor AMD Opteron 250 nodes, each with 4GB RAM with a fully connected 4-way Infiniband Topspin switch interconnection network. For conducting the experiments, we adopted the following methodology: first, we compared the performance of our five PBD metrics for the the parallel plate problem with the optically thin medium; second, we then compared the three most successful metrics for the jet flame problem. We kept the ratio between the number of processors and overall computational load constant—as we doubled the number of processors, we doubled the computational load. Since the computational load depends linearly on the total number of rays emitted from the simulation domain, we used the number of rays as a measure of the computational load. The mesh size is constant for all of the experiments.

We analyzed two performance metrics, relative load balancing factor and relative parallel overhead factor. Load balancing factor is defined as the difference between maximum and minimum time spent in calculation across all processors. Relative load balancing factor is the ratio between load balancing factor and the average time spent in calculation. The parallel overhead is defined as the difference between average overall CPU time and average time spent in calculation. The relative parallel overhead is the ratio between parallel overhead and the the average time spent in calculation.

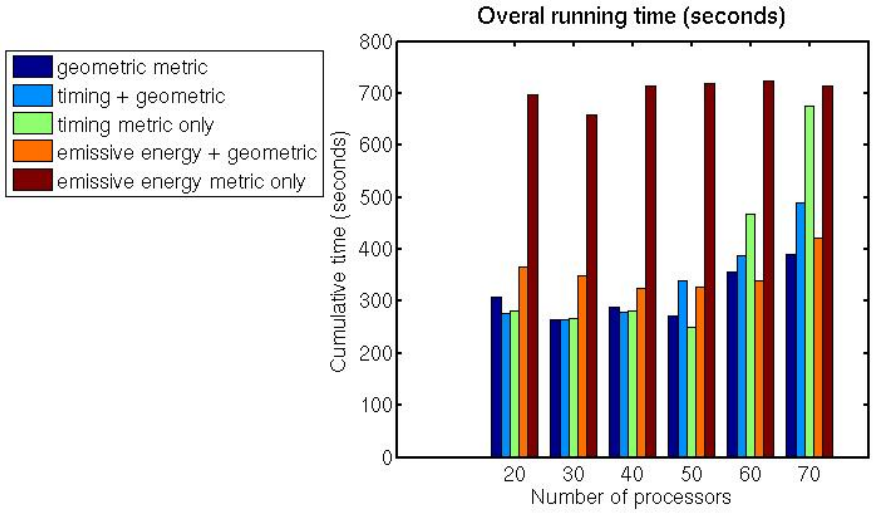
The overall time spent in the simulations does not change significantly for any metric when we vary the number of processors. This demonstrates that our software is scalable and can be used with various partitioning strategies. Moreover, the simulations that use only the geometric metric for domain partitioning have shown to yield decent performance statistics. This is very important since in the majority of applications, users will use well-known metrics, such as subdomain volume and aspect ratios. Also, the radiative properties of the medium can change significantly during the simulation but repartitioning the global domain may not be possible.

Also, we observe that the metric that uses the total emissive energy of the subdomains is not a good choice. The total emissive energy is proportional to the number of rays that are emitted from a particular subdomain. However, the computational load has two major constituents—the total number of emitted rays and the total number of volumes traversed by the rays. From Fig. 6.8 and 6.9 we see that this metric fails to accurately predict the total number of volumes that are traversed by the rays. If we analyze the performance of the CPU metric, we also see that, when used alone, it has the best relative load balancing factor in all the cases. However, we can see that it also yields a high communication overhead since this metric only represents the computational load of the radiative computations. In this metric, the parallel load is not a result

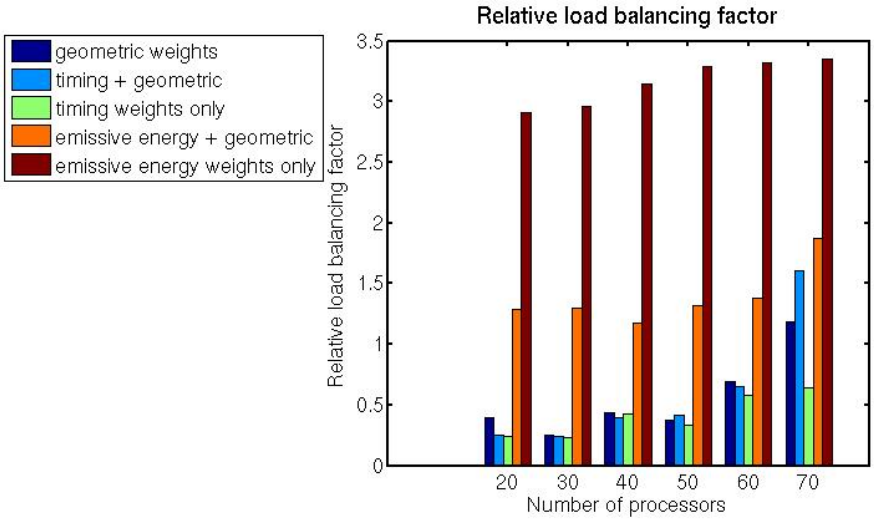
of load imbalance but rather the 'end game' described in the preceding section. On the other hand, we note that for the two best metrics, the geometric metric and the combined CPU and geometric metric, the parallel overhead is the direct result of the load imbalance.

Next we analyze the performance of the metrics for the jet flame simulation. First we analyze the gray jet flame simulation and the impact of the chosen metrics on the load balancing. The jet flame is optically thin, but it has a temperature profile that is highly nonhomogeneous. In Fig. 6.10 and 6.11 we show the overall CPU time for computation, the relative load balancing factor and the relative parallel overhead. We observe that, even though the participating medium does not have constant properties, the parallel overhead is dominated by the relative load balancing factor because the jet flame is optically thin. The nonhomogeneous temperature profile does not have a significant effect on the parallel performance of the PMC library. Note that the metric based on the emissive energy of the subdomain to predict the computational load fails here as well. Thus, we can safely conclude that, for the cases with gray radiation where initial emission of the rays is not expensive to calculate, the computational load cannot be accurately predicted from the elements' emissive energy.

Finally we analyze the nongray jet flame case. To calculate the spectral absorption coefficient we use the full spectrum correlated-k (FSCK) distribution method (see [4]) with a high-accuracy database of narrow-band k-distributions for water vapor, [53]. We assume the constant ratio of 2:1 for mass fractions of water vapor and CO_2. To determine the wavelength, or, for this particular case, the cumulative k-distribution g, of the each ray that is emitted from the domain, we must solve a nonlinear equation in the form of $F(g) = R_g$ where R_g is a random number associated with the g value and F is a nonlinear function. As suggested in [4], this relationship is tabulated before the first energy bundle is emitted, by evaluating function F at some number



(a) The overall CPU time for the parallel plate problem, optically thin case



(b) The relative load balancing factor for parallel plate problem, optically thin case

Fig. 6.8. The overall CPU time and relative load balancing factor for the parallel plate problem with an optically thin medium

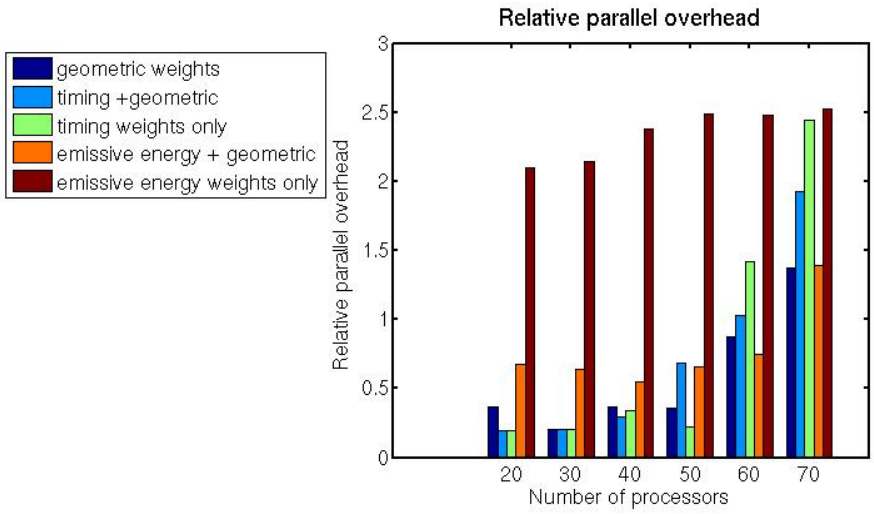
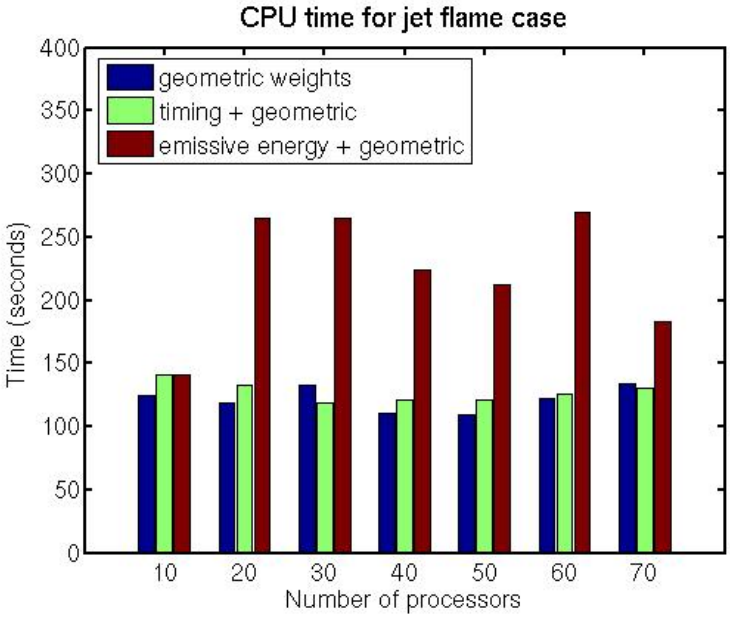
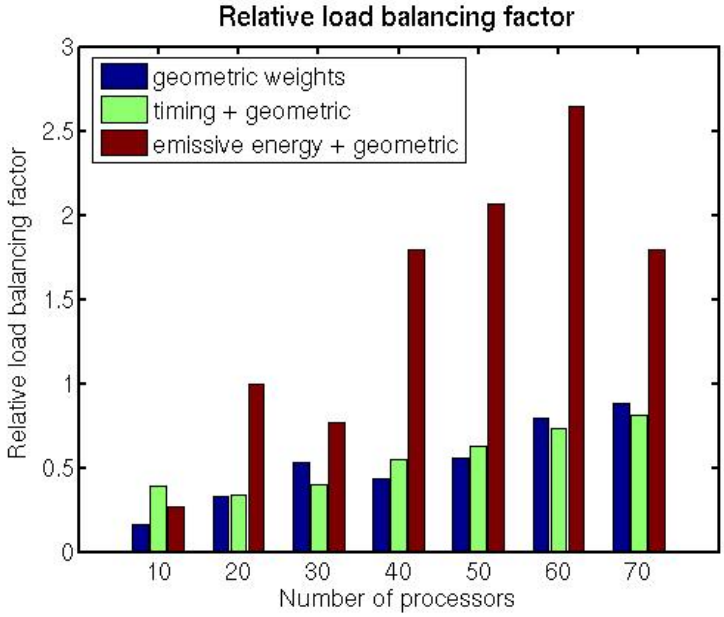


Fig. 6.9. The relative parallel overhead for the parallel plate problem with an optically thin medium



(a) The overall CPU time for the jet flame simulation



(b) The relative load balancing factor for the jet flame simulation

Fig. 6.10. The overall CPU time and relative load balancing factor for the jet flame simulation

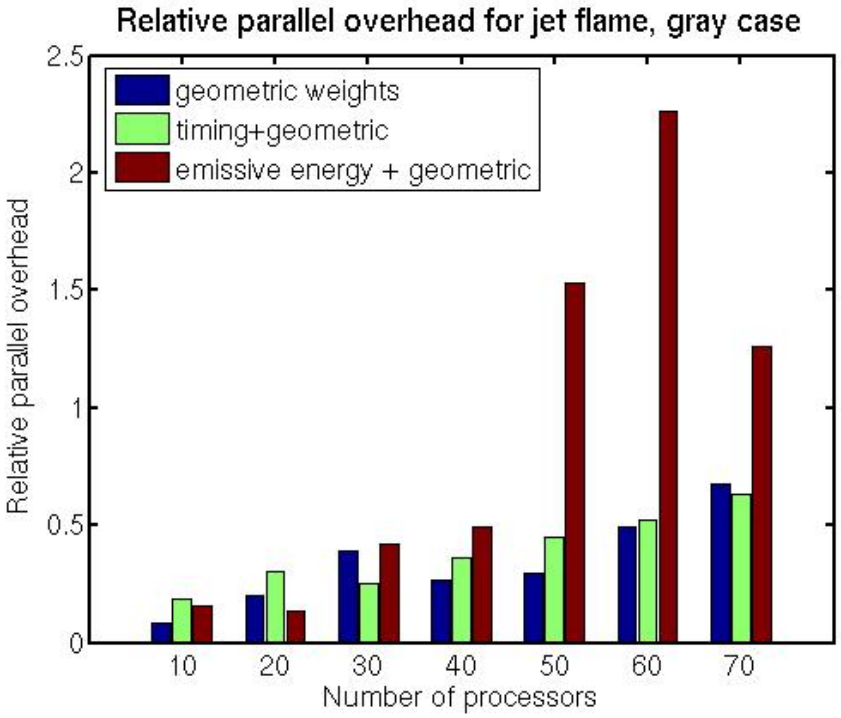


Fig. 6.11. The relative parallel overhead for the jet flame simulation

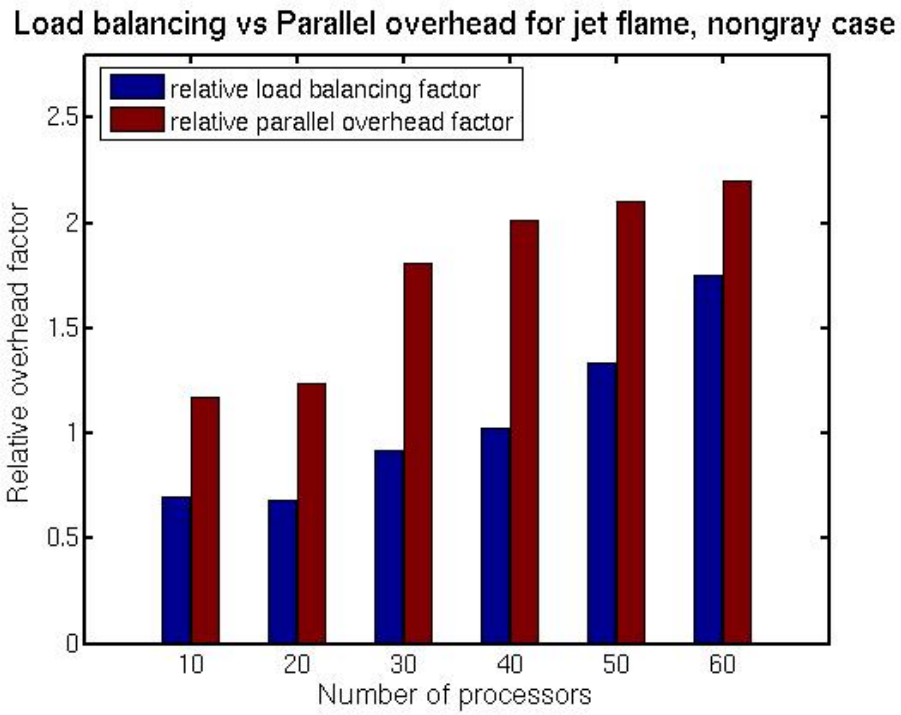


Fig. 6.12. A comparison between the relative load balancing and relative parallel overhead for the jet flame, nongray case

J of g points and calculating $R_{g,j} = F(g_j)$ for $j = 1, ...J$. Then, for arbitrary R_g, a linear interpolation is used to extract the proper g value. Furthermore, for optical thickness calculation we use the narrow-band k-distribution database ([53]).

The computational cost of tabulating the nonlinear relationship between the random number R_g and the g value can be balanced if the number of cells in the subdomains does not vary much between processors. This can be easily achieved by using the PBD algorithm with the geometric metric for the domain decomposition. Since in the previous cases we concluded that all other metrics we analyzed on average have not significantly better performance, for this case we only used the geometric metric. Moreover, unlike in the problems with strong scattering properties, the cost of tracing ray bundles through the medium is not significantly higher in the nongray case than in the gray (without scattering). However, the nongray problems have potentially different behavior with respect to the parallel overhead than the gray problems. The optical thickness of the cell element is now dependent on the wavelength of the ray. Therefore, the rays emitted from the same part of the medium can travel short or long distances, depending on their wavelength. This is reflected in Fig. 6.12 where we can see the correlation between the parallel overhead and load balancing is similar to the case with the mixed properties medium. Namely, the parallel overhead is again considerably larger than the load balancing factor, indicating the notable presence of idle time that processors spent waiting for more work. The case remains optically thin, but the absorption coefficient varies significantly over the spectrum. Hence, some rays travel much further than the others, depending on their wavelength. However, we can again observe the scalability of our PMC software, since the parallel overhead does not grow rapidly when we increase the number of processors.

6.6 Conclusions

In this chapter we described the parallel implementation of the PMC software framework. The main difficulty in implementing the parallel framework is how to conduct the partitioning of the work. Two broad choices, ray partitioning and domain partitioning are presented. Since ray partitioning is not scalable and it demands either the copy of the entire mesh on each processor or that the participating medium is extremely thick, domain partitioning is more suitable for the large-scale combustion simulations. The distribution of the computational load to the processors depends heavily on how the global mesh is partitioned among the processors. Therefore, we studied various metrics that are used in domain partitioning and presented an empirical study of their performance. We concluded that the geometric metric works best for various types of problems. Moreover, metrics that do not include communication load estimates can introduce significant parallel overhead.

Chapter 7

Conclusions and Recommendations for Future Research

7.1 Conclusions

This chapter summarizes the main results and conclusions from the research presented in this thesis. The objective of this thesis was to create efficient sequential and parallel algorithms and software that improve accuracy and performance of combustion applications. The accomplishments of this research are listed below.

- We developed a new sequential and parallel software system called Database On-Line for Efficient Function Approximation (DOLFA) that is used to speed up chemistry calculations in combustion applications.

- We designed a new sequential and parallel software system called Photon Monte Carlo (PMC), used for solving Radiative Transfer Equations (RTEs) by calculating radiative heat fluxes for each element of a complex enclosure and various radiation configurations.

- We created clean, easy-to-use software interfaces that can be readily used from both C/C++ and FORTRAN applications without significant adaptations to the original code.

For the sequential implementation of the DOLFA software, we have implemented the existing methods previously developed by Pope [11] and have developed new algorithms that enhance the search through the database and keep database size manageable. More specifically, we have extended the existing algorithms by allowing the terminal nodes (convex regions in the

composite space) of the Binary Space Partition (BSP) tree to contain more than one point, thus making the BSP tree more balanced and easier to search. We also have modified the existing methods by allowing points that do not belong to the given region to be included in that region if their Ellipsoid of Accuracy (EOA) intersects the region. Finally, we have demonstrated the advantages of using DOLFA for various types of combustion simulations.

For the parallel implementation of the DOLFA software, we have designed algorithms that build the global BSP tree and control the BSP tree by performing periodic redistribution. The global BSP tree is built off-line, using the list of points that are submitted to DOLFA in the first iteration. In the case of transient simulations the search space can change dramatically, which can cause considerable load imbalance. The redistribution algorithm uses the computational load from the previous iteration to redistribute the parts of the BSP tree to the processors that had the least amount of work in the previous iteration. Also, we have introduced three heuristics for managing the computational load and compared their performance. We have concluded that the heuristic that yields the minimal number of redundant calculations also introduces a large load imbalance. This heuristic outperforms all other heuristics in the case of steady state simulations and when used on a small number of processors in transient simulations. However, we have observed that for large-scale transient simulations the hybrid approach should be used instead.

For the sequential implementation of the Photon Monte Carlo (PMC) software, we have introduced an extensible and robust software framework that allows for implementation of various models of radiation in a wide range of applications. So far, the framework contains models for gray radiation calculations. However, it has a ‘plug and play’ capability, which enables it to encompass a wide range of models for RHT and incorporate methods to calculate different physical properties. Also, we have introduced new algorithms for tracking photon bundles through a

medium, where physical properties are given through particles distributed over the mesh, such as in the Probability Density Function (PDF) methods.

For the parallel implementation of the PMC software, we have considered two broad choices of partitioning the computational load. The choice most commonly used in various PMC software implementations, ray partitioning, is not suitable for large-scale combustion simulations, since it is not scalable. Domain partitioning is more appropriate for such simulations, but it introduces parallel overhead that depends on the type of simulation and balancing of the computational load among the processors. We have used a Parametric Binary Dissection algorithm for the partitioning of the mesh and we have investigated various metrics and analyzed their performance in different types of problems. The parallel overhead in cases with optically thick participating media is insignificant. Moreover, in the cases that are optically thin, the parallel overhead is mostly a result of load imbalance. However, if the medium has mixed properties, the parallel overhead can be significant, especially if the computation of the radiative properties is not expensive (as in cases of gray radiation).

7.2 Recommendations for Future Work

This research was focused on developing scalable and robust parallel software that will either speed up the calculations of existing models or enable the implementation of new models into the combustion simulations. A few areas identified for further research are given in the following paragraphs.

Further investigation of accuracy requirements for DOLFA. The accuracy requirement for a particular problem can so far only be found by trial and error. This problem becomes

particularly acute in the case of the constant approximation where we do not have any information about the behavior of the function in the neighborhood of the database point. On the other hand, using the linear and hybrid approximations can be costly and not a reasonable alternative for some problems.

Further investigation of the performance of parallel DOLFA software. The redistribution algorithm can be improved to more successfully manage the computational load. Also, the cleaning procedure is currently performed on each processor independently. The current cleaning procedure reduces the database size, but it can also contribute to highly unbalanced BSP trees. The parallel cleaning could be implemented and its performance analyzed.

Further implementation and validation of nongray radiation models into PMC. So far the functions that model gray radiation have been implemented and fully validated in the PMC software framework. Functions that model nongray radiation are implemented but they have not been extensively validated.

Further investigation of partitioning strategies for parallel PMC. Additional partitioning algorithms and metrics can be considered to improve the parallel efficiency of the PMC.

Appendix A

The Basic Properties of Radiative Heat Transfer

Radiative Properties of Surfaces and Gases

The basic properties of opaque surfaces include the following:

Reflectance $\rho = \frac{\text{reflected part of incoming radiation}}{\text{total incoming radiation}}$,

Absorptance $\alpha = \frac{\text{absorbed part of incoming radiation}}{\text{total incoming radiation}}$,

Transmittance $\tau = \frac{\text{transmitted part of incoming radiation}}{\text{total incoming radiation}}$, and

Emittance $\epsilon = \frac{\text{energy emitted from the surface}}{\text{energy emitted by a black surface at the same temperature}}$.

All four properties may be functions of temperature, wavelength and incoming, and, in the case of reflectance and transimittance, outgoing directions [4].

A participating medium (typically a gas) is also characterized with properties such as transmissivity, absorptivity and scattering. As radiative energy penetrates through a participating medium, it gradually becomes attenuated by absorption and scattering. Experience and theoretical development has shown that this attenuation leads to an exponential decay of incident radiation. Therefore, the *optical thickness* (*transmissivity*) of a homogeneous isothermal medium may be written as $t = e^{-(k_\eta + \sigma_{s\eta})s}$, where s is the thickness of the medium, k_η is the

absorption coefficient and $\sigma_{s\eta}$ is the scattering coefficient. The absorptivity of the gas is given by $\alpha_\eta = 1 - \tau_\eta$. To completely characterize scattering, we define a *scattering phase function*, $\Phi_\eta(\hat{\mathbf{s}}_i, \hat{\mathbf{s}})$, which describes the probability that a ray from one direction $\hat{\mathbf{s}}_i$ will be scattered into direction $\hat{\mathbf{s}}$, where $\hat{\mathbf{s}}$ and $\hat{\mathbf{s}}_i$ are unit vectors.

Attenuation by Absorption and Scattering

- **Absorption**

  With absorption, part of the energy is removed from the direction of propagation by converting it into internal energy.

$$\text{absorbed incident energy: } (dI_\eta)_{abs} = -k_\eta I_\eta ds, \tag{A.1}$$

  where k_η is *the linear absorption coefficient.*

- **Scattering**

  With scattering, part of the energy is removed from the direction of propagation $\hat{\mathbf{s}}$ by redirecting it into another direction where it appears as augmentation.

$$\text{out-scattered incident energy: } (dI_\eta)_{sca} = -\sigma_{s\eta} I_\eta ds, \tag{A.2}$$

  where $\sigma_{s\eta}$ is *the linear scattering coefficient.*

To express total attenuation, it is useful to define the *extinction coefficient* as $\beta_\eta = k_\eta + \sigma_{s\eta}$.

Augmentation by Emission and Scattering

- **Emission**

$$\text{emitted incident energy: } (dI_\eta)_{em} = k_\eta I_{b\eta} ds, \tag{A.3}$$

where k_η is *the linear absorption coefficient* and $I_{b\eta}$ is the blackbody intensity. In absorbing-emitting (without scattering) medium, the equation of transfer is:

$$\frac{dI_\eta}{ds} = k_\eta (I_{b\eta} - I_\eta). \tag{A.4}$$

The solution for an isothermal gas layer of thickness s is:

$$I_\eta(s) = I_\eta(0) e^{-\tau_\eta} + I_{b\eta}(1 - e^{-\tau_\eta}). \tag{A.5}$$

- **Scattering**

Augmentation due to scattering is difficult to asses since it has contributions from all directions and, therefore, must be calculated by integrating over all solid angles (see Fig. A.1). The equation for energy flux scattered into direction $\hat{\mathbf{s}}$ from *all* incoming directions $\hat{\mathbf{s}}_i$ is

$$\text{in-scattered incident energy: } (dI_\eta)_{sca}(\hat{\mathbf{s}}) = ds \frac{\sigma_{s\eta}}{4\pi} \int_{4\pi} I_\eta(\hat{\mathbf{s}}_i) \Phi_\eta(\hat{\mathbf{s}}_i, \hat{\mathbf{s}}) d\Omega_i, \tag{A.6}$$

where $\sigma_{s\eta}$ is *the linear scattering coefficient* and Φ_η is the *scattering phase function*, which describes the probability that a ray from one direction $\hat{\mathbf{s}}_i$ will be scattered into a

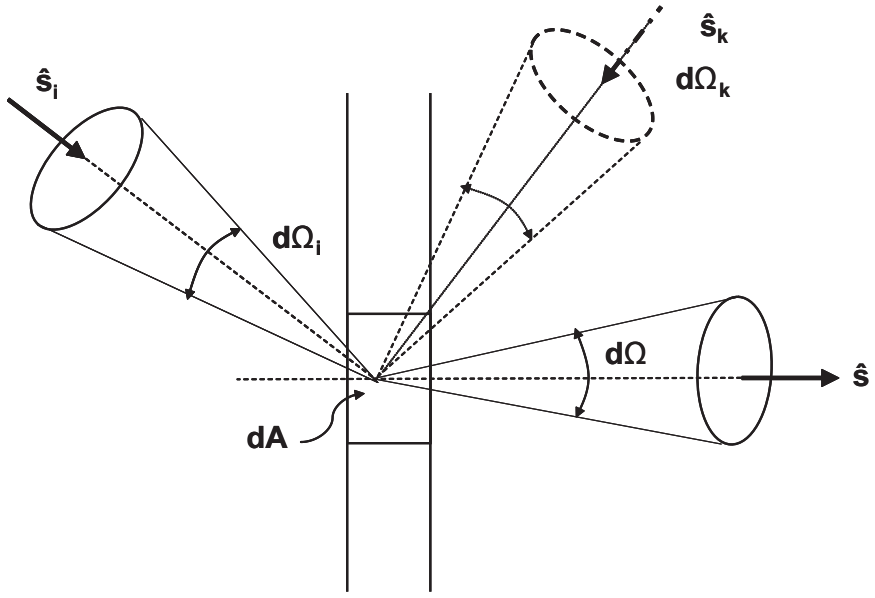


Fig. A.1. Radiative intensity scattered from a surface. From [4].

direction $\hat{\mathbf{s}}$. Since Φ_η is a probability-density function, we conclude that

$$\frac{1}{4\pi}\int_{4\pi}\Phi_\eta(\hat{\mathbf{s}}_i,\hat{\mathbf{s}})d\Omega = 1. \tag{A.7}$$

If $\Phi_\eta = 1$ (constant), equal amounts of energy are scattered into all directions (*isotropic scattering*).

The Equation of Radiative Transfer in Enclosure with Participating Media

In this section we briefly overview general relationships that govern the behavior of radiative heat transfer in the presence of an absorbing, emitting and/or scattering medium. Rays on their path through a medium can be absorbed or scattered from the incoming direction. In this case, the energy is removed from the direction of propagation. This process is called *attenuation.* In the case of scattering, they add their energy to the outgoing direction. Rays can also be emitted and in that case they also add energy to the direction of the propagation. The second two cases are called *augmentation.*

The change in intensity from point s to point $s+ds$ is found by summing the contributions from emission, absorption, scattering away from direction $\hat{\mathbf{s}}$ and scattering into the same direction $\hat{\mathbf{s}}$ (equation (A.8)) at wavenumber η, [4]. We express this infitesimal change in intensity as:

$$\begin{aligned}\frac{dI_\eta(\hat{\mathbf{s}})}{ds} &= \hat{\mathbf{s}}\cdot\nabla I_\eta(\hat{\mathbf{s}}) \\ &= k_\eta I_{b\eta} - \beta_\eta I_\eta(\hat{\mathbf{s}}) + \frac{\sigma_{s\eta}}{4\pi}\int_{4\pi} I_\eta(\hat{\mathbf{s}}_i)\Phi_\eta(\hat{\mathbf{s}}_i,\hat{\mathbf{s}})d\Omega_i.\end{aligned} \tag{A.8}$$

To include radiative heat transfer with conduction and convection in the energy conservation we need: (1) the *radiative heat flux* for the surface boundaries, $q_\eta \cdot \hat{\mathbf{n}} = \int_{4\pi} I_\eta(\hat{\mathbf{s}})\hat{\mathbf{n}} \cdot \hat{\mathbf{s}} d\Omega$, and (2) the *divergence of heat flux* for the volume elements, $\nabla \cdot q_\eta = 4\pi k_\eta I_{b\eta} - k_\eta \int_{4\pi} I_\eta(\hat{\mathbf{s}}) d\Omega + \sigma_{s\eta} \int_{4\pi} I_\eta(\hat{\mathbf{s}}_i) d\Omega_i$.

Equation (A.8) is an integro-differential equation in five-dimensional space (three spatial variables and two angles that determine the direction of $\hat{\mathbf{s}}$). An exact analytical solution to this equation is typically impossible to obtain.

Appendix B

Random Number Relations for Surfaces and Participating Medium

Point of Emission

Surface: Total emission from surface A_j can be expressed as:

$$E_j = \int_{A_j} \epsilon\sigma T^4 dA. \tag{B.1}$$

Since integration over area is a double integral, we can rewrite this equation as:

$$\begin{aligned} E_j &= \int_0^X \int_0^Y \epsilon\sigma T^4 dy dx \\ &= \int_0^X E_j^y dx, \end{aligned} \tag{B.2}$$

where

$$E_j^y(x) = \int_0^Y \epsilon\sigma T^4 dy. \tag{B.3}$$

To find the x coordinate of the point of emission, we have to solve the equation:

$$R_x = \frac{1}{E_j} \int_0^x E_j^y dx, \tag{B.4}$$

where R_x is a random number drawn from a uniform distribution on [0,1]. This equation is solved for x, $x = x(R_x)$, where R_x is a random number in [0,1]. To find the y coordinate

of the point of ray emission, we have to solve the equation:

$$R_y = \frac{1}{E_j^y} \int_0^y \epsilon \sigma T^4 dy. \tag{B.5}$$

This equation is solved for y, $y = y(R_y, x)$, where R_y is a random number in [0,1].

Volume: Similarly as with emission point from the surface, total emission from surface A_j can be expressed as:

$$\begin{aligned} E_k &= \int_{V_k} 4\kappa_P \sigma T^4 dV, && \text{(B.6)} \\ E_k &= \int_0^X \int_0^Y \int_0^Z 4\kappa_P \sigma T^4 dz dy dx \\ &= \int_0^X E_k^{yz} dx, && \text{(B.7)} \end{aligned}$$

where

$$\begin{aligned} E_k^{yz}(x) &= \int_0^Y \int_0^Z 4\kappa_P \sigma T^4 dz dy \\ &= \int_0^Y E_k^z dy, && \text{(B.8)} \\ E_k^z(x) &= \int_0^Z 4\kappa_P \sigma T^4 dz. && \text{(B.9)} \end{aligned}$$

Following development from the previous subsections, we can express the random numbers as:

$$R_x = \frac{1}{E_k} \int_0^x E_k^{yz} dx, \tag{B.10}$$

$$R_y = \frac{1}{E_k^{yz}} \int_0^y E_k^{z} dx, \tag{B.11}$$

$$R_z = \frac{1}{E_k^{z}} \int_0^z 4\kappa_P \sigma T^4 dx, \tag{B.12}$$

or,

$$x = x(R_x),\ y = y(R_y, x),\ z = z(R_z, x, y). \tag{B.13}$$

where R_x, R_y, R_z are random numbers drawn from uniform distribution on [0,1].

Wavelength of Emission

In the case of gray radiation, radiative properties do not have spectral dependency and calculation of wavelength of emission is not needed. In the case of radiative properties that exhibit spectral dependence, we must calculate λ:

Surface: $R_\lambda = \frac{1}{\epsilon\sigma T^4} \int_0^\lambda \epsilon_\lambda E_{b\lambda} d\lambda$, and

Volume: $R_\lambda = \frac{\pi}{\kappa_P \sigma T^4} \int_0^\lambda k_\lambda I_{b\lambda} d\lambda$.

After the inversion, $\lambda = \lambda(R_\lambda, x, y)$ for surfaces and $\lambda = \lambda(R_\lambda, x, y, z)$ for volumes. Often these integrals cannot be successfully calculated by applying standard quadrature formulas. The reason for this lies in the fact that the absorption coefficient k_λ can be a strongly gyrating function of wavelength. Thus, many digits of accuracy would be needed for evaluation of the above

integrals, but this is also not feasible since accurate knowledge of spectral variations of k_λ is rarely known. Usually, narrow band models or full-spectrum k-g distribution are used to find wavelengths of emitted rays, [4].

Directions of Emission

Surface: Spectral emissive power is given of $E_\lambda = \frac{1}{\pi} E_{b\lambda} \int_0^{2\pi} \int_0^{\pi/2} \epsilon_\lambda^s \cos\theta \sin\theta d\theta d\psi$, where ϵ_λ^s is emittance for wavelength λ in direction s. Then we have:

$$R_\psi = \frac{E_{b\lambda}}{\pi E_\lambda} \int_0^{\psi} \int_0^{\pi/2} \epsilon_\lambda^s \cos\theta \sin\theta d\theta d\psi, \tag{B.14}$$

$$R_\theta = \int_0^{\theta} \epsilon_\lambda^s \cos\theta \sin\theta d\theta \int_0^{\pi/2} \epsilon_\lambda^s \cos\theta \sin\theta d\theta. \tag{B.15}$$

or,

$$\psi = \psi(R_\psi, x, y, \lambda), \theta = \theta(R_\theta, x, y, \lambda, \psi). \tag{B.16}$$

Volume: Under the thermodynamic equilibrium conditions, emission within a participating medium is isotropic. In that case, $\epsilon_\lambda = 2\int_0^{\pi/2} \sin\theta \cos\theta d\theta$ so integrals from equations (B.15) and (B.15) are readily solved, $R_\psi = \frac{\psi}{2\pi}$, $R_\theta = \sin^2\theta$.

Absorption

Surface: When the ray hits a surface, fraction α_λ^s is absorbed and the rest is reflected. The absorptance α_λ^s depends on the wavelength of the ray λ and the incoming direction s. This can be simulated by picking a random number, R_α and comparing it to α_λ^s: if it is smaller, the ray is absorbed, otherwise, it is reflected.

Volume: When the ray travels through a participating medium, its energy is attenuated by absorption and scattering. Absorptivity for a photon path of length l is:

$$\alpha_{\lambda} = 1 - exp\left(-\int_0^l k_{\lambda} ds\right), \tag{B.17}$$

$$R_k = exp\left(-\int_0^l k_{\lambda} ds\right). \tag{B.18}$$

If we divide path l into subpaths where k_{λ} is approximately constant, we have:

$$\int_0^l k_{\lambda} ds \simeq \sum_j k_{\lambda j} l_j. \tag{B.19}$$

Then, as long as:

$$\int_0^l k_{\lambda} ds < \ln \frac{1}{R_k}, \tag{B.20}$$

the bundle is not absorbed and is allowed to travel on.

Reflection and Scattering

Reflection from Surfaces: The direction of reflection depends on the bidirectional reflection function of the material. In combustion simulations, most of the surfaces can be approximated as diffuse reflectors, and in that case the equations are given as follows:

$$R_{\psi_r} = \frac{\psi_r}{2\pi},\ R_{\theta_r} = \sin^2 \theta_r. \tag{B.21}$$

For a purely specular reflector, the reflection direction is determined by the law of optics (no random numbers are needed):

$$\psi_r = \psi_i + \pi,\ \theta_r = \theta_i. \tag{B.22}$$

Scattering Inside a Volume: If we introduce the *scattering phase function*, $\Phi(\hat{\mathbf{s}} \cdot \hat{\mathbf{s}}')$, and assume linear anisotropic scattering, we can represent the *scattering phase function* in the following way:

$$\begin{aligned} \Phi(\hat{\mathbf{s}} \cdot \hat{\mathbf{s}}') &= 1 + A_1 \hat{\mathbf{s}} \cdot \hat{\mathbf{s}}' \\ &= 1 + A_1 \cos\theta, \end{aligned} \tag{B.23}$$

where θ is the angle between the incoming direction $\hat{\mathbf{s}}$ and a new direction $\hat{\mathbf{s}}'$. Then, polar and azimuthal angles for scattering can be established in the following way:

$$R_{\psi'} = \frac{\psi'}{2\pi}, \tag{B.24}$$

$$R_{\theta'} = \frac{1}{2}\left(1 - \cos\theta' + \frac{A_1}{2}\sin^2\theta'\right). \tag{B.25}$$

If scattering is isotropic (outgoing direction does not depend on incoming direction), the equations above can be readily used. On the other hand, if scattering is not isotropic, the new direction vector $\hat{\mathbf{s}}'$ must be found by introducing a local coordinate system at the point of scattering with $\hat{\mathbf{s}}$ pointing into its z-direction, as shown in Fig. B.1.

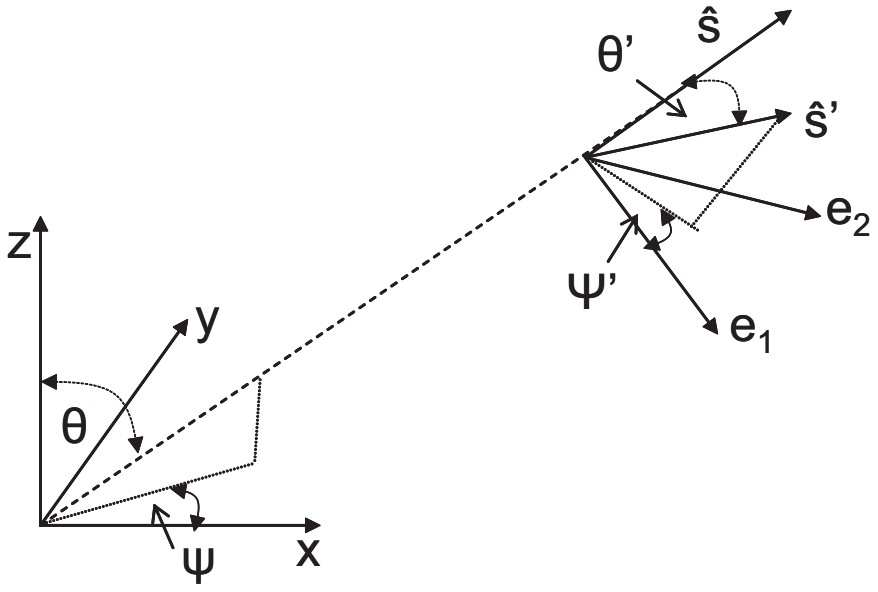


Fig. B.1. Local coordinate system for scattering direction, from [4]

References

[1] Veljkovic, I., Plassmann, P.E.: Scalable photon monte carlo algorithms and software for the solution of radiative heat transfer problems. In Yang, L.T., Rena, O.F., Martino, B.D., Dongarra, J., eds.: High Performance Computing and Communications. Volume 3726 of Lecture Notes in Computer Science (LNCS). Springer (2005) 928–937

[2] DAlesion, A., Di Lorenzo, A., Beretta, F., , Venitozzi, C.: Optical and chemical investigations on fuel-rich methane-oxygen premixed flames at atmospheric pressure. Fourteenth Symposium (International) on Combustion, The Combustion Institute (1973) 941–953

[3] Kee, R., Coltrin, M., Glarborg, P.: Chemically Reacting Flow, theory and practice. Wiley-Interscience (2003)

[4] Modest, M.F.: Radiative Heat Transfer, Second edition. Academic Press (2003)

[5] Li, G.: Investigation of Turbulence-Radiation Interactions by a Hybrid FV/PDF Monte Carlo Method. PhD thesis, The Pennsylvania State University (2002)

[6] Lindstedt, R., Louloudi, S., Vaos, E.: Joint scalar probability density function modeling of polutant formation in piloted turbulent jet diffusion flames with comprehensive chemistry. Proceedings of the Combustion Institute, Vol. 28 (2000) 149–156

[7] Blumrich, M., Chen, D., Coteus, P., Gara, A., Giampapa, M., Heidelberger, P., Singh, S., Steinmacher-Burow, B., Takken, T., Vranas, P.: Design and analysis of the BlueGene/L Torus interconnection network. RC23025 (2003)

[8] The Beowulf Cluster Site: http://www.beowulf.org/overview/index.html . (2004)

[9] Veljkovic, I., Plassmann, P.E., Haworth, D.C.: A scientific on-line database for efficient function approximation. Computational Science and Its Applications—ICCSA 2003, The Springer Verlag Lecture Notes in Computer Science (LNCS 2667) series, part I (2003) 643–653

[10] Veljkovic, I., Plassmann, P., Haworth, D.: A parallel implementation of scientific on-line database for efficient function approximation. Post-Conference Proceedings, The 2004 International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, USA (2004) 24–29

[11] Pope, S.: Computationally efficient implementation of combustion chemistry using in-situ adaptive tabulation. Combustion Theory Modelling **1** (1997) 41–63

[12] Chung, T.: Numerical Modeling in Combustion. Taylor and Francis (1993)

[13] Maas, U., Pope, S.B.: Laminar flame calculations using simplified chemical kinetics based on intrinsic low-dimensional manifolds. Twenty-Fifth Symposium (International) Combustion/The Combustion Institute (1994) 1349–1356

[14] Maas, U., Schmidt, D.: Analysis of the intrinsic low-dimensional manifolds of strained and unstrained flames. In: 3rd Workshop on Modelling of Chemical Reaction Systems, Heidelberg. (1996)

[15] Bender, R., Blasenbrey, T., Maas, U.: Coupling of detailed and ILDM-reduced chemistry with turbulent mixing. Proceedings of the Combustion Institute **28** (2000) 101–106

[16] Rabitz, H., Alis, O.: General foundations of high dimensional model representations. Journal of Math. Chemistry **25** (1999) 197–233

[17] Kee, R., Rupley, F., Miller, J.: CHEMKIN-II: A Fortran chemical kinetics package for the analysis of gas phase chemical kinetics. Sandia National Laboratories Report SAND89-8009B. (1989)

[18] Chemkin Suite: http://www.reactiondesign.com (2002)

[19] Chemkin User Group home page: http://www.groups.yahoo.com/group/Chemkin (2002)

[20] Tonse, S.R., Moriarty, N.W., Brown, N.J., Frenklach, M.: PRISM: Piecewise reusable implementation of solution mapping: an economical strategy for chemical kinetics. Israel Journal of Chemistry **39** (1999) 97–106

[21] Tonse, S.R., Moriarty, N.W., Brown, N.J., Frenklach, M.: Computational economy improvements in PRISM. International Journal of Chemical Kinetics, Volume 35 (2003) 438–452

[22] Maas, U., Pope, S.: Simplifying chemical kinetics: Intrinsic low-dimensional manifolds in composition space. Combustion and Flame **88** (1992) 239–264

[23] Singh, S., Powers, J., Paolucci, S.: On slow manifolds of chemically reactive systems. Journal of Chemical Physics **117(4)** (2002) 1482–1496

[24] Maas, U., Pope, S.: Implementation of simplified chemical kinetics based on intrinsic low-dimensional manifolds. Twenty-Fourth Symposium (International) Combustion/The Combustion I nstitute (1992) 103–112

[25] Pope, S.: ISAT-CK User Manual (Version 3.0). (2000)

[26] Embouazza, M., Haworth, D., Darabiha, N.: Implementation of detailed chemical mechanisms into multidimensional CFD using in situ adaptive tabulation: Application to HCCI engines. Society of Automotive Engineers, Inc (2002)

[27] Haworth, D., Wang, L., Kung, E., Veljkovic, I., Plassmann, P., Embouazza, M.: Detailed chemical kinetics in multidimensional CFD using storage/retrieval algorithms. In: 13th International Multidimensional Engine Modeling User's Group Meeting, Detroit. (2003)

[28] Brown, P.N., Byrne, G.D., Hindmarsh, A.C.: VODE: A variable coefficient ODE solver. SIAM J. Sci. Stat. Comput. Vol 10(5) (1989) 1038–1051

[29] Byrne, G.D., Hindmarsh, A.C.: Polyalgorithm for the numerical solution of ordinary differential equations. ACM Trans. Math. Software (1975) 71–96

[30] Byrne, G.D., Hindmarsh, A.C.: PVODE, an ODE solver for parallel computers. Int. J. High Perf. Comput. Appl., Vol. 13, No. 4 (1999) 354–365

[31] Veljkovic, I., Plassmann, P.E.: Parallel heuristics for an on-line scientific database for efficient function approximation. In Wasniewski, J., Madsen, K., Dongarra, J., eds.: Workshop on State-of-the-Art in Scientific Computing (PARA'04). Volume 3732 of Lecture Notes in Computer Science (LNCS). Springer (2006) 644–653

[32] Veljkovic, I., Plassmann, P.E.: A scalable scientific database for chemistry calculations in reacting flow simulations. In Yang, L.T., Rena, O.F., Martino, B.D., Dongarra, J., eds.: High Performance Computing and Communications. Volume 3726 of Lecture Notes in Computer Science (LNCS). Springer (2005) 948–957

[33] Jeans, J.: The equations of radiative transfer of energy. Monthly Notices Royal Astronomical Society (1917) 28–36

[34] Chandrasekhar, S.: Radiative Transfer. Dover Publications (1960)

[35] Shirley, P.: Hybrid radiosity/Monte Carlo methods. In: Siggraph 94 Advanced Radiosity Course. (1994)

[36] Matsumoto, M., Nishimura, T.: Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation **8** (1998) 3–30

[37] L'Ecuyer, P.: Uniform random number generators : A review. Proceedings of the 1997 Winter Simulation Conference (1997) 127–134

[38] Heinisch, R., Sparrow, E., Shamsundar, N.: Radiant emission from baffled conical cavities. Journal of the Optical Society of America, Vol 63, no 2 (1973) 152–158

[39] Shamsundar, N., Sparrow, E., Heinisch, R.: Monte Carlo solutions - effect of energy partitioning and number of rays. International Journal of Heat and Mass Transfer, Vol 16 (1973) 690–694

[40] Modest, M., Poon, S.: Determination of three-dimensional radiative exhange factors for the space shuttle by Monte Carlo. ASME paper 77-HT-49 (1977)

[41] Modest, M.: Determination of radiative exchange factors for three dimensional geometries with nonideal surface properties. Numerical Heat Transfer, Vol 1 (1978) 403–416

[42] Modest, M.: Radiative heat transfer fluxes through the exit of the GE combustor transition piece. Technical Report (private communication to General Electric Co.) (1980)

[43] Wang, L., Jacques, S., Zheng, L.: MCML - Monte Carlo modeling of photon transport in multi-layered tissues. Computer Methods and Programs in Biomedicine **47** (1995) 131–146

[44] Wang, A., Modest, M.: Monte Carlo schemes for radiative transfer in media represented by particle fields. Proceedings of IMECE: International Mechanical Engineering Congrtess and Exposition (November,2005)

[45] Tal, J., Ben-Zvi, R., Kribus, A.: A high efficiency parallel solution of the radiative transfer equation. Numerical Heat Transfer,Part B, Vol 44 (2003) 295–308

[46] Lu, X., Hsu, P.F.: Parallel computing of two numerical quadratures for an integral formulation of transient radiation transport. Proceedings of 2003 ASME Summer Heat Transfer Conference, Las Vegas (2003)

[47] Sawetprawichkul, A., Hsu, P.F., Mitra, K.: Parallel computing of three-dimensional Monte Carlo simulation of transient radiative transfer in participating media. Proceedings of the 8th AIAA/ASME Joint Thermophysics and Heat Transfer Conf., St Louis, Missouri (2002)

[48] Dewaraja, Y., Ljungberg, M., A. Majumdar, A.B., Koral, K.: A parallel Monte Carlo code for planar and SPECT imaging: Implementation, verification and applications in I-131 SPECT. Computer Methods and Programs in Biomedicine (2002)

[49] Burns, P., Christon, M., Schweitzer, R., Lubeck, O., Wasserman, H., Simmons, M., Pryor, D.: Vectorization of Monte Carlo particle transport: An architectural study using the LANL benchmark GAMTEB. Proceedings, Supercomputing 1989 (1988) 10–20

[50] Wikramaratna, R.: Pseudo-random number generation for Parallel Monte Carlo:a splitting approach. SIAM News **33** (2000)

[51] Hellekalek, P.: Pseudo-random number generation for Parallel Monte Carlo:a splitting approach. Proceedings of the twelfth workshop on Parallel and distributed simulatio (1998) 82–89

[52] International Works on Measurement and Computation of Turbulent Nonpremixed Flames: www.ca.sandia.gov/TNF/radiation.html (1998)

[53] Wang, A., Modest, M.F.: High-accuracy, compact database of narrow-band k-distributions for water vapor and carbon dioxide. In: Proceedings of the ICHMT 4th International Symposium on Radiative Transfer, Turkey. (2004)

[54] Bokhari, S., Crockett, T., Nicol, D.: Binary dissection: Variants and applications. ICASE Report No. 93-39 (1993)

Vita

EDUCATION

- April 2006: Ph.D. in Computer Science and Engineering, Computer Science and Engineering Department at Pennsylvania State University, University Park, Pennsylvania.
- June 2000: Bachelor of Science from Faculty of Mathematics, Computer Science and Information Group, University of Belgrade, Yugoslavia.

PROFESSIONAL EXPERIENCE

- Spring 2002–December 2005: Research assistant working with Dr. Paul E. Plassmann. Part of the NSF ITR project: Scalable Portable Algorithms for Thermal Radiation/Turbulence/Chemistry Interactions.
- Summer 2004: Internship in the MCS Division, Argonne National Laboratory, working with Dr. Boyana Norris.
- Summer 2003: Internship in the MCS Division, Argonne National Laboratory, working with Dr. Paul Hovland.
- Fall 2001: Teaching assistant in the Department of Computer Science and Engineering at Pennsylvania State University.

HONORS AND AWARDS

- The Wallace Givens Research Associate appointment in the MCS division at Argonne National Laboratory, May-Aug 2004.
- Student Paper Prize Competition Winner, Eleventh SIAM Conference on Parallel Processing for Scientific Computing (PP04), 2004
- SIAM Student Travel Award to attend the Eleventh SIAM Conference on Parallel Processing for Scientific Computing (PP04), 2004
- Dean’s Fellowship for Graduate Students, Penn State University, 2001.

RESEARCH SOFTWARE AUTHORED

- DOLFA, Database On-Line for Function Approximation, by I. Veljkovic, P. Plassmann
- PMC, Photon Monte Carlo, by I. Veljkovic, P. Plassmann