Turbulence and Molecular Processes in Combustion
T. Takeno (Editor)

PDF/Monte Carlo Methods for Turbulent Combustion and Their Implementation on Parallel Computers

S.B. Pope

Cornell University, Ithaca, NY 14853, U.S.A.

Abstract

Turbulent combustion models are used in engineering design and development to calculate the flow and thermochemical fields within combustion devices. There is considerable research activity aimed at improving the accuracy of the models, at increasing the level of description, and at developing better computational implementations. A brief review is given of PDF methods used in turbulent combustion modelling, and also of the particle/Monte Carlo methods that are used as numerical solution algorithms. Recent work on the implementation of these Monte Carlo methods on parallel computers is then described. The three strategies that have been implemented in combination are: multiple independent simulations; particle partitioning; and, mesh work partitioning. These strategies can be implemented with minimal changes to a serial code by using a small number of communication subroutines.

1. INTRODUCTION

Turbulent combustion models are used in engineering design and development to calculate the flow and thermochemical fields within combustion devices. To make such calculations, the following ingredients are required: a turbulence model; a simplified thermochemical model; a numerical solution algorithm; a computer code implementing the solution algorithm; and, computer hardware.

In the past decade, several different pdf models have been proposed in which a modelled transport equation is solved for the joint pdf of certain fluid properties in a turbulent flow. This approach is particularly attractive for combustion problems, since the direct effects of reaction can be incorporated without the usual closure problem (Pope 1990). The preferred approach to solving the modelled pdf equations is to use particle-based Monte Carlo methods. Here too, several different methods have been proposed.

In this paper, after a brief survey of PDF/Monte Carlo methods, recent work is reported on the implementation of these methods on parallel computers. In the literature on PDF-methods development, most attention is focused on modelling issues, with some work reported on numerical methods. With the prospect of vastly increased computer power being available through parallel and distributed computing, it is also worthwhile to address the issues of implementing PDF/Monte Carlo methods in such a computing environment.

2. PDF/MONTE CARLO METHODS

The first applications of PDF methods to combustion were based on the composition joint pdf, i.e. the joint pdf of scalar variables describing the thermochemical state of the fluid (Dopazo & O'Brien 1974, Pope 1976, O'Brien 1980). A Monte Carlo method to solve the composition joint pdf eqation is described by Pope (1981). In this method there is a grid in physical space, and at each grid node the joint pdf is represented by an ensemble of particles. Jet-flame calculations using this method are reported by (among others) Nguyen & Pope (1984), Chen, Dibble & Bilger (1990) and Hsu (1991). The same method has been applied to recirculating flows by Roekaerts (1991).

Haworth & Pope (1986, 1987) developed a modelled equation for the joint pdf of velocity and composition. The advantage of including the velocity in the formulation is that the direct effects of convection appear in closed form, and hence the gradient-diffusion assumption (invoked in the composition joint pdf model) is avoided. The Monte Carlo method developed for this level of closure (Pope 1985) is a particle-mesh method. Statistics, such as means and variances, are evaluated at mesh points, whereas the particles are continuously distributed in physical space. (In the simplest case of constant-density flow in Cartesian coordinates, the expected particle-number density is uniform in physical space.) Jet-flame calculations using this type of method have been reported by Pope & Correa (1988) and Masri & Pope (1990). The method has also been used to make calculations of one-dimensional premixed flames (Pope & Anand 1985, Anand & Pope 1987).

Extensions of this method to recirculating flows have been reported by Anand, Pope & Mongia (1989a) and by Haworth & El Tahry (1991).

A hybrid finite-volume/Monte Carlo method has proved to be a useful transition step in incorporating PDF methods into industrial combustor codes. Consider, first, a constant-density reacting flow. A standard finite-volume code incorporating a turbulence model (e.g. $k-\varepsilon$) is used to calculate the mean velocity fields $\langle \mathbf{U}(\mathbf{x},t)\rangle$ and the fields of $k(\mathbf{x},t)$ and $\varepsilon(\mathbf{x},t)$. A Monte Carlo code then runs in tandem to calculate the joint pdf of velocity and composition. Suppose that in the standard Monte Carlo method the velocity $\mathbf{U}^*(t)$ of a particle located at $\mathbf{x}^*(t)$ is modelled to evolve by

$$\frac{d}{dt}\mathbf{U}^* = \mathbf{A}(\mathbf{U}^*, \mathbf{x}^*). \tag{1}$$

Then, in the hybrid method, the evolution equation is modified to

$$\frac{d}{dt}\mathbf{U}^* = A(\mathbf{U}^*,\ \mathbf{x}^*) + \mathbf{B}(\mathbf{x}^*) + \mathbf{U}^* C(\mathbf{x}^*), \tag{2}$$

where the functions $\mathbf{B}(\mathbf{x})$ and $C(\mathbf{x})$ are determined by the conditions that the mean velocity and the kinetic energy obtained from the joint pdf calculation match those obtained from the finite-volume turbulence model calculation.

From a modelling viewpoint, this hybrid method is inferior to a stand-alone joint-pdf model, because of the additional terms in Eq. (2). On the other hand, it is distinctly superior to the composition joint pdf method because the gradient-diffusion assumption is avoided. (This has numerical advantages also.)

For variable-density flows, the mean density is determined from the joint pdf calculation, and is fed back to the finite-volume code.

Calculations based on this method for recirculating flows are reported by Correa & Pope (1992) and by Fox (1992).

The modelled equation for the joint pdf of velocity and composition does not provide a complete closure. This is because a separate model is needed to determine the turbulence frequency $\langle\omega\rangle \equiv \varepsilon/k$ that is used in the pdf model. For example, Anand, Pope & Mongia (1989b) use a modelled equation for $\langle\omega\rangle$ similar to the standard ε equation.

In part to remedy this shortcoming, Pope & Chen (1990) and Pope (1991) developed a model based on the joint pdf of velocity, frequency and composition where the frequency, ω, is defined as the instantaneous dissipation divided by k. An additional advantage is the ability of the method to account for turbulence of different scales, including turbulent/non-turbulent intermittency. The method has been applied to jet flames by Norris & Pope (1991). A detailed numerical investigation is provided by Pope (1992).

3 PARALLEL PROCESSING IN PDF/MONTE CARLO METHODS

We describe here several strategies for exploiting parallel computers in PDF/Monte Carlo methods. These strategies are based on the single-program-multiple-data model, and are designed (primarily) for message-passing distributed memory systems. They have been implemented on: a 32 node Intel i-860; a cluster of 7 IBM RS6000-560 workstations; and on a 64 processor Kendall Square Research KSR1.

The basic question addressed is the following: given a serial code to perform PDF/Monte Carlo calculations, how can this code be modified to take advantage of P distributed processors? The advantage sought is either:

1. to decrease the run time T, ideally by a factor of P; or
2. for the same run time T, to increase the useful computation (ideally by a factor of P) so as to reduce numerical error; or
3. some combination of 1 and 2.

3.1 PDF/Monte Carlo Method

We consider the type of Monte Carlo/PDF method used by Correa & Pope (1992). This is a multidimensional simulation in which there is a mesh that divides physical space into C cells. There are N particles, which are continuously distributed in physical space. The number of particles per cell

$$N' \equiv N/C, \tag{3}$$

is typically about 100.

Mean fields—e.g. mean velocity, mean density, turbulent kinetic energy etc.—are defined on the mesh. These mean fields are estimated from the corresponding particle

properties as weighted averages. For the purposes of the present discussion, it is simplest to think of means at cell centers as being estimated as the arithmetic means of the properties of all the particles within the cell.

The properties of each particle evolve in time according to a coupled set of stochastic differential equations, the coefficients of which involve the mean fields. It is important to note, however, that there are no direct particle-particle interactions. Conceptually, then, the algorithm involves two types of operation: summing particle properties to form means; and, advancing particle properties in time (independently).

3.2 Multiple Independent Simulations

The simplest use of P processors is to perform $S = P$ independent simulations. To understand the benefits and limitations of this strategy, it is necessary to appreciate the nature of the numerical errors.

Let Q denote an output from the simulation, e.g. the turbulent kinetic energy at a particular point and time. Because of the Monte Carlo nature of the simulation, Q is a pseudo-random variable: different seeds to the pseudo-random number generator yield different values of Q.

The convergence of the method requires that as N tends to infinity, Q tends to a (non-random) limit—Q_∞, say. A finite value of N results in a numerical error that can be decomposed into a *bias* B_Q and a *statistical error* Σ_Q. The bias is defined by

$$B_Q \equiv \langle Q \rangle - Q_\infty, \tag{4}$$

and can be expected to scale as N^{-1} (Pope 1992), i.e.

$$B_Q \approx b_Q / N, \tag{5}$$

where b_Q is a constant.

The statistical error can be written

$$\Sigma_Q = \xi \sigma_Q N^{-1/2}, \tag{6}$$

where ξ is a standardized random variable ($\langle \xi \rangle = 0$, $\langle \xi^2 \rangle = 1$), and the standard error σ_Q is

$$\sigma_Q = \sqrt{\text{var}\,(Q)}. \tag{7}$$

The run time T for such a simulation can be well approximated by

$$T = \alpha (N + N_0), \tag{8}$$

where α and N_0 are constants—N_0 representing the non-particle work expressed as an equivalent number of particles.

Consider now that we have a distributed-memory parallel computer with P processors. The simplest possible way to use these processors is for each to perform a statistically identical but independent simulation. That is, the same (essentially serial)

code is run on each processor, but with different pseudo-random number seeds, leading to statistically independent simulations. Then we obtain $S = P$ independent samples of Q—that from the i-th simulation being denoted by $Q^{(i)}$. From these the sample mean $\bar{Q}$ is obtained

$$\bar{Q} \equiv \frac{1}{S} \sum_{i=1}^{S} Q^{(i)}, \tag{9}$$

and similarly the sample variance can be obtained.

There are two benefits of this approach. First $\bar{Q}$ provides a more accurate estimate of Q_∞. Specifically the statistical error is less by a factor of $S^{1/2} = P^{1/2}$, *but* the bias is unchanged. Second, from the sample mean and variance, a confidence interval for $\bar{Q}$ (as an estimate of $\langle Q \rangle$) can be obtained.

We make the following observations on this method of *multiple independent simulations.*

1. The amount of communication required between the processors is extremely small. Hence the communication overhead is negligible.

2. Each processor has essentially the same run time given by Eq. (8) (i.e. there is excellent load balancing).

3. The statistical error can be reduced at will by increasing the number of processors, P. But once the statistical error is small compared to the bias there is little benefit in increasing P further.

4. Instead of using the P processors to increase accuracy, they can be used to decrease run time for the same statistical error. This can be achieved by decreasing N by a factor of P. This approach is less attractive because the bias increases, by a factor of P (Eq. 5). Further, with increasing P, it is evident from Eq. (8) that the saving in run time becomes slight, once N becomes comparable to N_0.

3.3 Particle Partitioning

Particle partitioning provides a simple way to use P processors to perform a single simulation (i.e. $S = 1$). It has been used previously in a particle method for plasma simulations (Azari & Lee 1991).

With P processors, the ensemble of N particles is divided into P statistically-identical sub-ensembles of approximately $n = N/P$ particles each. Then each processor is assigned one of these sub-ensembles, whose evolution it calculates, based on the common mean fields. That is, each processor has a copy of the mean fields which are estimated from the full ensemble.

The mean fields are formed (as weighted sums of particle properties from the whole ensemble) in two stages. First, each processor forms the weighted sums over its sub-ensemble. Second, the results of the first stage are summed over all the processors. As a result, each processor has a copy of the same mean fields.

It is important to observe that in this technique the amount of communication scales with the mesh size C, while the dominant contribution to the CPU time scales with N—which is typically two orders of magnitude larger than C. Consequently, although the communication is non-trivial, for normal values of C, N and P, the communication overhead is modest.

As described, the advantage of using *particle partitioning* is to reduce the run time from $\alpha(N + N_0)$ to $\alpha([N/P] + N_0)$, while *both* the bias and the statistical error remain the same. As with *multiple independent simulations*, this advantage peters out once $n = N/P$ becomes small compared to N_0.

Alternatively the technique can be used to decrease the error by a factor of $P^{1/2}$ for approximately the same run time. This is achieved by keeping the number of particles per processor (n) fixed, and hence the total number of particles $N = nP$ increases with P.

With this latter approach (fixed n), as P increases, at some point the communication overhead becomes significant, and eventually dominant. The run time can be modelled as

$$T = \alpha(n + N_0) + \gamma C \Gamma(P), \tag{10}$$

where γ is a constant, and the function $\Gamma(P)$ accounts for the increase in communication time with the number of processors. The form of $\Gamma(P)$ depends both on the interconnect topology and on the communications software. The best that can be expected is:

$$\begin{aligned} \Gamma(P) &= \log_2(P), \text{ for a hypercube} \\ \Gamma(P) &= P^{1/2}, \text{ for a 2D mesh} \\ \Gamma(P) &= P, \text{ for a ring} \end{aligned} \tag{11}$$

A hybrid scheme—*multiple independent simulations with particle partitioning*—has two advantages over particle partitioning alone. In such a scheme, the P processors are divided into S sub-sets of $p = P/S$ processors each. Then a statistically identical but independent simulation is performed on each sub-set of processors, using particle partitioning over the p processors in the subset. The first advantage of this hybrid scheme is that (for $S \geq 4$, say) confidence intervals can be obtained.

The second advantage is that (compared to particle partitioning) the amount of communication can be reduced, without significantly increasing the error. The two relevant errors are the statistical error (which scales as $N^{-1/2} = (nP)^{-1/2}$) and the bias (which scales as $(np)^{-1} = (nP/S)^{-1}$). It is evident from these scalings that with particle partitioning alone (i.e. $S = 1$), the bias decreases more rapidly with P than does the statistical error. With the hybrid scheme, it is reasonable to allow S to scale as $P^{1/2}$ (for fixed n). For then the bias and statistical error scale in the same way. The number of processors per simulation then scales as $p \sim P^{1/2}$, and hence the communication overhead scales as $\Gamma(P^{1/2})$ (rather than as $\Gamma(P)$ for particle partitioning).

3.4 Mesh Work Partitioning

For a serial code implementing the type of Monte Carlo/PDF method described above, the run time can be modelled as $T = \alpha(N_0+N)$ (i.e. Eq. 8). Typically the particle-dependent contribution αN dominates the particle-independent contribution αN_0; and hence, in coding for efficiency, the focus has been on the former rather than on the latter.

With the strategies described above, the run time can be modelled as

$$\begin{aligned} T &= \alpha(N/p + N_0) + \gamma C\Gamma(p) \\ &= C\{\alpha(n' + n_0) + \gamma\Gamma(p)\}, \end{aligned} \tag{12}$$

where $n' \equiv N/(pC)$ is the number of particles per cell per processor, and $n_0 \equiv N_0/C$. Since the dominant contribution to the non-particle work (αN_0) scales with the number of cells, it is best expressed as $\alpha C n_0$, where α and n_0 are (to a first approximation) constants.

If the number of particles per processor n is fixed as P increases, then these strategies are extremely effective in reducing the numerical error (by a factor of approximately $P^{1/2}$), while there is just a slow rise in the run time due to communication. For normal values of the parameter, the dominant contribution to the run time is the particle work $\alpha n = \alpha C n'$.

But if these strategies are used to reduce the run time for fixed N, then n' decreases with P and so the other contributions to T become important. In particular, for $n' < n_0$ (i.e. $p > N/(n_0 C)$) the non-particle work exceeds the particle work. The strategy of *mesh work partitioning*, now described, alleviates this limitation.

The non-particle work $\alpha C n_0$ arises from several sources. A significant source is computations performed on mesh quantities, i.e. mean fields. This can either be in the process of estimating mean fields, or in evaluating coefficients in the stochastic differential equations from mean fields. With particle partitioning, each of the p processors performing a simulation has identical copies of the mean fields, and hence performs identical operations on them. Instead, each processor can perform the operations on a disjoint set of C/p cells or mesh points, and then the results can be shared with the $(p-1)$ other processors performing the simulation.

Of the non-particle work ($\alpha C n_0$), let $\alpha C \tilde{n}_0$ be the contribution from mesh work that can effectively be partitioned between processors, and let $\alpha C \bar{n}_0$ be the remainder. Then the total run time (Eq. 12) is modified to

$$T = C\{\alpha(n' + \bar{n}_0 + \tilde{n}_0/p) + \tilde{\gamma}\Gamma(p)\}. \tag{13}$$

The constant $\tilde{\gamma}$ is somewhat larger than γ due to the additional message passing required to share the results of the mesh computations. Experiments on an Intel hypercube show that for the code used by Correa & Pope (1992), $\bar{n}_0$ is 4.5.

3.5 Summary of Strategies and Performance

The strategies described above allow P processors to be used to perform S simulations with N particles each. The statistical error and bias scale as $(NS)^{-1/2}$ and N^{-1}, respectively. With $S \geq 4$, say, confidence intervals can be determined.

If non-particle work and message-passing were negligible, the run time would be

$$\hat{T} \equiv \alpha N/p, \tag{14}$$

(where $p \equiv P/S$ is the number of processors per simulation). Accounting for these overheads, the run time T can be approximated as

$$T/\hat{T} = 1 + \frac{\bar{n}_0}{n'} + \frac{\tilde{n}_0}{n'p} + \frac{\tilde{\gamma}}{\alpha n'}\Gamma(p). \tag{15}$$

The inverse of this quantity, $\hat{T}/T$, can be interpreted as an *efficiency*, and hence the last three terms of the right-hand side represent sources of inefficiency. The main conclusions to be drawn from this expression are:

1. n'—the number of particles per cell per processor—is a dominant parameter in determining efficiency.

2. The inefficiency associated with mesh work that can be partitioned—$\tilde{n}_0/(n'p)$—*decreases* with increasing p (for fixed n').

3. The inefficiency associated with the non-particle work that cannot be partitioned—$\bar{n}_0/n'$—sets a lower limit on the size of simulation that can be performed efficiently. For $N < \bar{n}_0 pC$, greater than 50% efficiency cannot be achieved.

4. The communications inefficiency varies as $(n')^{-1}$, and increases weakly with P.

3.6 Implementation

The strategies described above have been implemented on several machines (Intel i-860, KSR1, 7xRS6000) using a variety of communication protocols (e.g. PICL and PVM). This is achieved by:

(a) writing a small number of communication subroutines

(b) modifying a serial Fortran Monte Carlo code by incorporating a few calls to the communication subroutines.

This approach has the virtue that minimal alterations to the serial code are required; and the same code can be used with different machines and protocols. The communication subroutines have to be modified for different protocols—but in total they amount to only 200 lines of code.

The functions of the principal communication subroutines are now described.

The following Fortran segment illustrates how mesh work appears in a serial code.

```
c
c  loop over cells to evaluate variable g in each cell
c
      do 100 ic = 1, nc
      .
      .
      .
100   g(ic) = . . . .
c
```

Mesh work partitioning is achieved by replacing the above code by:

```
c  determine the cells that this processor should treat
c
      call divwrk( nc, ifirst, ilast)
c
c  loop over those cells
c
      do 100 ic = ifirst, ilast
      .
      .
      .
100   g(ic) = . . . .
c
c  share results between processors
c
      call shares( g, nc)
c
```

Thus, the implementation of mesh work partition simply requires modifying the do-loop indices and adding two calls to the communication routines, `divwrk` and `shares`.

Particle partitioning is implemented by adding a single subroutine call for each particle sum that is formed. In a serial code, the generic summing loop is:

```
c
c  loop over cells to form sum
c
      do 200 ic = 1, nc
      sum (ic) = 0.
c
c  loop over particles in cell to form sum
c
         do 200 ip = nf(ic), nl(ic)
         .
         .
```

```
200       sum(ic) = sum(ic) + ...
```

With particle partitioning, this sum has to be summed over all p processors performing the simulation. This is simply achieved through the call:

```
call sumsub( sum, nc)
```

(This routine then calls other communication routines to determine which other processors are in the same sub-set.)

The output from a serial code is a set of statistics, with Fortran names `q(i)`, `i=1,2,...,ns`, say. With multiple independent simulations, the subroutine call

```
call stats( q, ns, qmean, qlow, qup)
```

yields (with obvious notation) the sample mean and the lower and upper bounds of the confidence interval.

CONCLUSIONS

Several different PDF methods implemented via several different particle/Monte Carlo methods have previously been applied to problems in turbulent combustion. In this paper we have described three simple strategies for exploiting parallel and distributed computing in such Monte Carlo methods. In drawing conclusions, we first consider using parallel processing to reduce the numerical error (compared to a serial code) for approximately the same run time.

The use of *multiple independent simulations* requires little communication and is effective in reducing statistical error. However, it does not reduce the bias.

Particle partitioning can be used effectively to reduce both the statistical error and the bias. The communication is, in this case, non-trivial, but nevertheless the communication overhead is small (in normal circumstances).

An alternative objective is to reduce the run time (compared to a serial code) for approximately the same numerical error. In addition to the two strategies already mentioned, *mesh work partitioning* can also be used to meet this objective.

With the aid of a small number of communication subroutines, these strategies can readily be implemented in a serial Monte Carlo code.

ACKNOWLEDGEMENTS

I gratefully acknowledge numerous discussions on the parallel implementation of particle methods with Professors D.A. Caughey, T.F. Coleman, S.-Y. Lee, N.F. Otani, S.A. Vavasis and Dr. S. Chinchalkar. This work was supported in part by Grant No. CTS-9113236 from the National Science Foundation. Computations supporting the research were performed using the resources of the Cornell Theory Center, which receives major

funding from the National Science Foundation and IBM Corporation, with additional support from New York State Science and Technology Foundation and Members of the Corporate Research Institute.

REFERENCES

1. Anand, M.S., Pope, S.B. and Mongia,H.C. (1989a) in *Turbulent Reactive Flows* Lecture Notes in Engineering (Springer-Verlag, Berlin) Vol. 40, p. 672.

2. Anand, M.S., Pope, S.B. and Mongia, H.C. (1989b) "*Calculations of axisymmetric turbulent jets by the PDF method*," Seventh Symp. on Turbulent Shear Flows, Stanford, CA.

3. Anand, M.S. and Pope, S.B. (1987) *Combust. Flame* **67**, 127.

4. Azari, N.G. and Lee, S.-Y. (1991) "*A hybrid task partitioning for particle-in-cell simulation on shared memory systems.*" The 1991 Int'l. Conf. on Distributed Computing Systems, Dallas, TX, p. 526.

5. Chen, J.-Y., Dibble, R.W. and Bilger, R.W. (1990) *Twenty-third Symp. (Int'l.) on Combust.*, The Combustion Institute, p. 775.

6. Correa, S.M. and Pope, S.B. (1992) *Twenty-fourth Symp. (Int'l.) on Combust.*, The Combustion Institute (in press).

7. Dopazo, C. and O'Brien, E.E. (1974) *Acta Astronaut*, **1**, 1239.

8. Fox, R.O. (1992) *Chem. Engrg. Sci.* **47**, 2853.

9. Haworth, D.C. and El Tahry, S.H. (1991) *AIAA J.* **29**, 208.

10. Haworth, D.C. and Pope, S.B. (1987) *Phys. Fluids* **30**, 1026.

11. Hawarth, D.C. and Pope, S.B. (1986) *Phys. Fluids* **29**, 387.

12. Hsu, A.T. (1991) "*A study of hydrogen diffusion flames using PDF turbulence model*" AIAA paper no. 91-1780.

13. Masri, A.R. and Pope, S.B. (1990) *Combust. Flame* **81**, 13.

14. Nguyen, T.V. and Pope, S.B. (1984) *Combust. Sci. Technol.* **42**, 13.

15. Norris, A.T. and Pope, S.B. (1991) "*Application of PDF methods to piloted diffusion flames: sensitivity to model parameters*" Eighth Symp. on Turbulent Shear Flows, Munich.

16. O'Brien, E.E. (1980) in *Turbulent Reactive Flows.* Springer-Verlag, Berlin.

17. Pope, S.B. (1992) "Particle Method for Turbulent Flows: Integration of Stochastic Differential Equations" Cornell Report FDA-92-03 (submitted to J. Comp. Phys.).

18. Pope, S.B. (1991) *Phys. Fluids* **A 3**, 1947.

19. Pope, S.B. (1990) *Twenty-third Symp. (Int'l) on Combust.*, The Combustion Institute, p. 591.

20. Pope, S.B. and Chen, Y.L. (1990) *Phys. Fluids* **A 2**, 1437.

21. Pope, S.B. and Correa, S.M. (1988) *Twenty-First Symp. (Int'l.) on Combust.*, The Combustion Institute, p. 1341.

22. Pope, S.B. (1985) *Prog. Energy Combust. Sci.* **11**, 119.

23. Pope, S.B. and Anand, M.S. (1985) *Twenty-First Symp. (Int'l) on Combust.*, The Combustion Institute, p. 403.

24. Pope, S.B. (1981) *Combust. Sci. Technol.* **25**, 159.

25. Pope, S.B. (1976) *Combust. Flame* **27**, 299.

26. Roekaerts, P. (1991) *Applied Scientific Research*, **48**, 271.