Lawrence Berkeley National Laboratory
Recent Work

Title

Achieving Full Chemistry and Combustion Models Using POET (parallel Object-Oriented Environment and Toolkit)

Permalink

https://escholarship.org/uc/item/3106k4h6

Author

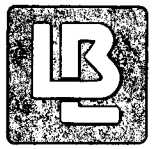
MacFarlane, J.F.

Publication Date

1993-02-01

eScholarship.org Powered by the California Digital Library
University of California

LBL-33779
UC-400



Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

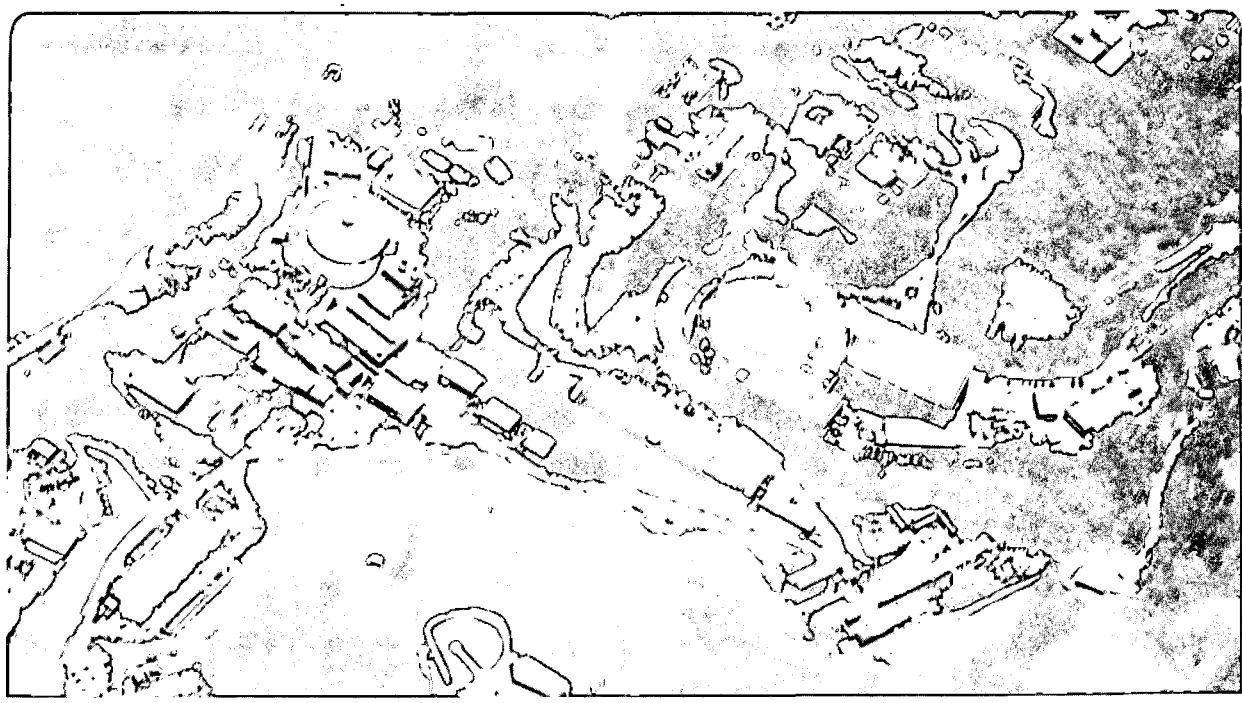
Information and Computing Sciences Division

Presented at the 1993 SCS Simulation Multiconference, Arlington, VA, March 27–April 1, 1993, and to be published in the Proceedings

Achieving Full Chemistry in Combustion Models Using POET (Parallel Object-Oriented Environment and Toolkit)

J.F. Macfarlane

February 1993



| REFERENCE COPY |
| Does Not |
| Circulate |
| Bldg. 50 Library. |
LBL-33779
Copy 1

Prepared for the U.S. Department of Energy under Contract Number DE-AC03-76SF00098

DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. Neither the United States Government nor any agency thereof, nor The Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or The Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or The Regents of the University of California and shall not be used for advertising or product endorsement purposes.

Lawrence Berkeley Laboratory is an equal opportunity employer.

This publication has been reproduced from the best available copy

DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

LBL-33779

Achieving Full Chemistry in Combustion Models Using POET (Parallel Object-Oriented Environment and Toolkit)

J.F. Macfarlane
Lawrence Berkeley Laboratory
University of California
Berkeley, CA 94720

February 1993

This work was supported by the Director's Exploratory Research and Development Funds of Lawrence Berkeley Laboratory under the U.S. Department of Energy, Contract No. DE-AC03-76SF00098

ACHIEVING FULL CHEMISTRY IN COMBUSTION MODELS USING POET (Parallel Object-Oriented Environment and Toolkit)

J. F. Macfarlane
Lawrence Berkeley Laboratory
Berkeley, CA 94720

R. C. Armstrong, R.E. Cline, Jr., M. L. Koszykowski
Sandia National Laboratories
Livermore, CA 94551

ABSTRACT

The development of new combustion devices that combine both high efficiency and low emission relies heavily on our understanding of the interactions between turbulence and chemical reactions during the combustion process. One approach, used for studying turbulent non premixed jet flames, is the Monte Carlo solution of the joint probability density function (PDF) in turbulent flows. The integration of stiff chemical kinetics and their interaction with the turbulence model greatly increases the computational requirements and accounts for over 98% of the total CPU time. As a result, the models quickly become compute bound and generally have been limited to reduced mechanisms and simple systems. Parallel computing offers a solution to these limitations by allowing the independent chemical kinetic calculations for each statistical sample in the Monte Carlo simulation to be distributed across many processors.

A hybrid moment-PDF model has been adapted to run under the newly developed Parallel Object-oriented Environment and Toolkit (POET). POET is the result of a collaboration between computational and physical scientists to develop MIMD applications for distributed and massively parallel systems. POET provides a mechanism that permits the computational and physical scientist to contribute their expertise to an integrated scalable application through the use of a well defined object-oriented interface.

INTRODUCTION

In the last decade scientists in all disciplines have seen an enormous increase in available computing technology. However, along with these resources came complex and sophisticated computing techniques necessary to optimally access them. As such, the computationally naive scientist has been faced with either learning an entirely new discipline or continued use of technologically outdated models and techniques. Complicating the matter further is the inherent complexity of computational models in the physical sciences. Managing the model development and reliability of the subsequent computational codes is also a difficult task. In order to effectively develop and utilize these models and also take advantage of high performance computing systems, a well planned approach to the computational science becomes a crucial part of the development.

POET is part of a larger effort called the Advanced Combustion Modeling Environment (ACME) [Kozsykowski *et al* 1992]. The goal of ACME is to provide a computational modeling environment for the combustion scientist that (a) *enables* and *manages* the development of complex combustion models (b) accesses high performance next-generation computing systems and (c) allows simultaneous development of both the computational and modeling sciences. The purpose of POET is to provide a transparent link to the power of parallel distributed computing.

Our approach is similar in design methodology to that employed by the developers of the X toolkit. We have designed a high level object-oriented framework that isolates the physical model description from the code that implements the parallel algorithm and data flow. As such, the combustion scientist need only be concerned with application specific code.

COMBUSTION MODELS

In the coming decades, the U. S. is challenged to achieve energy efficiency while minimizing environmental damage from combustion devices such as appliances, furnaces, boilers, gas turbines, and internal combustion engines. Computational combustion modeling can provide industry with design tools to address this challenge. To date, much progress has been made. Further progress is impeded by several obstacles imposed by limitations in computing capabilities.

A goal of ACME is to develop a suite of models with significant improvements that will benefit industry in the design of modern combustion devices. Improved combustion modeling is necessary for U. S. industry to design combustors with high combustion efficiency and

reduced pollutant emissions. Currently, the combustion process is modeled by coupling turbulent flow models with simple chemical kinetics models. Industry must have significantly more advanced models in order to predict combustion properties in turbulent flows with sufficient chemistry to evaluate environmental concerns.

Presently, one of the most important problems in turbulent combustion modeling is correctly approximating the coupling between reactive and diffusive processes on the smallest scales. Future progress in combustor design is very promising if modeling capabilities are further extended so that detail at a finer level of structure can be predicted. To be sufficiently accurate to be used as design tools, these models and their corresponding computational codes must include both chemistry, fluid mechanics, and their interactions over a broad range of time and length scales.

While fluid-mechanical turbulence models and detailed-chemistry flame models in simple flows are solvable on standard vector supercomputers, the combination of turbulent flow and detailed chemistry in the same model requires the next generation supercomputer: the massively parallel machine. As an initial demonstration of POET, we have investigated a probability density function (PDF) code for a jet flame diffusion problem. The PDF algorithm involves mostly Monte Carlo calculations and is highly amenable to an efficient parallel implementation. POET is used to partition the algorithmic portions of the code (e.g. equation solver, Monte Carlo simulation) from the application specific code.

POET ARCHITECTURE

Existing tools for parallel software development generally fall into two categories: 1) high-level tools and compilers that hide the parallelization details, making them easy to use but also hiding the pitfalls that lead to bottlenecks; or, 2) low-level tools for message passing that create scalable code, but require such detailed knowledge of algorithms and software that they are difficult for the non-systems programmer to use. The purpose of POET is to allow a physical scientist to create an integrated and scalable application code that transparently accesses parallel computing resources and avoids the traditional pitfalls associated with parallel computing. POET is a high-level object-oriented framework for parallel computing that is designed for direct integration of existing application codes. It is written in C++ in an extensible manner that guarantees scalable code.

Because the paradigm for object-oriented (OO) programming and the physical structure of MIMD parallel computing are so similar, the parallel processing community is becoming increasingly aware of the benefits of OO computing to MIMD processing. Most common MIMD computers have the CPU and memory together on a single unit and computations proceed by message-passing between collections of these units. The OO design methodology is based on collecting processing methods and the data to be processed into single objects, with computations proceeding by passing messages between objects. As such, the OO paradigm is a natural fit for MIMD architectures.

We refer to POET as a toolkit because it is more than a collection of objects that simply provide computational processing capabilities. Figures 1 through 3 indicate three distinct approaches[1] to parallel software development. The first approach, Figure 1, is the traditional approach often embodied when porting existing code from vector machines to parallel machines. Most often, the user is responsible for all aspects of the code: the application specific code, the parallel algorithm, and the data flow. This approach requires a specialist in parallel code development in order to achieve an effective speed up.

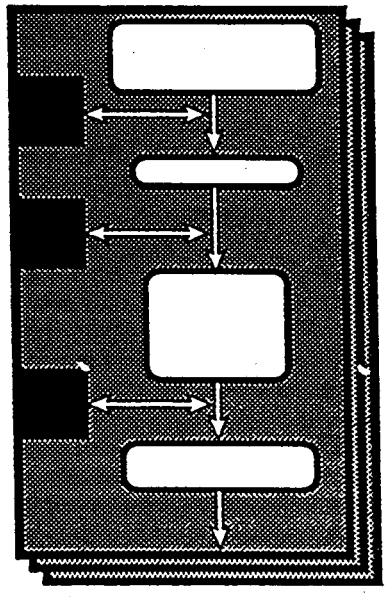


Figure 1. Traditional Approach to Parallel Computing

The second approach, Figure 2, consists of providing an object-oriented class library that has already been designed for a parallel implementation. However, typical C++ class libraries require the user to write the main routine that drives the objects instantiated from the library. The user is made responsible for the highest level portion of the application and consequently he/she must be aware of the environment in which the application is to run. As such, the user has control and *must* control the highest level of the application. With unsophisticated users this

[1]This list is not meant to be complete. These examples are provided in order to highlight the difference between a toolkit approach and other approaches.

can lead to highly complicated structures that will cancel all the benefits gained from the OO class library. As an example, consider the X windows programming environment. X windows requires constant arbitration between the Server and the application's window event loop. It would be disastrous to hand over the top level of an X program to a naive user and trust that the main event loop would be consulted at the right time and often enough to avoid poor behavior or outright deadlock. In Xt the top level is handled by an object called top level shell. The user's main program exists only to configure objects, called *widgets*, by adding callback routines or modifying attributes under the top level shell. Then the widgets are realized by calling the main-loop function for the top level, from which control will not return until the application ends.

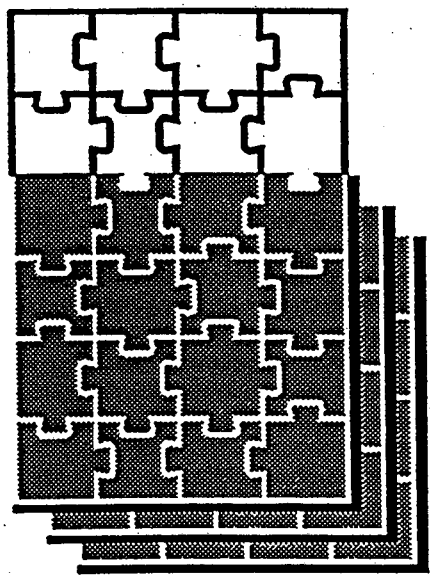


Figure 2. Object-oriented Class Library Approach

As we define it, a toolkit is a library of objects that also includes a top-level control object. The POET approach, Figure 3, seeks to provide a similar mechanism for MIMD parallel computations. POET separates the physics specific to a problem from the parallel algorithm needed for solution. Our goal is to embed as much of the computational control as possible into the toolkit. The physics specific to the users particular application problem is handled as user add-ons. For our first implementation, these add-ons take the form of callbacks that specialize a general algorithm embodied in an object selected from the toolkit. Taking into account the object the user has requested, the toolkit decomposes the problem across multiple processors. The provided callbacks, however, need not be aware of this decomposition nor the architecture of the parallel environment, because the physics that specialized the problem are independent from the object chosen for the solution. A detailed FORTRAN example for a turbulent reacting PDF code is given in our example.

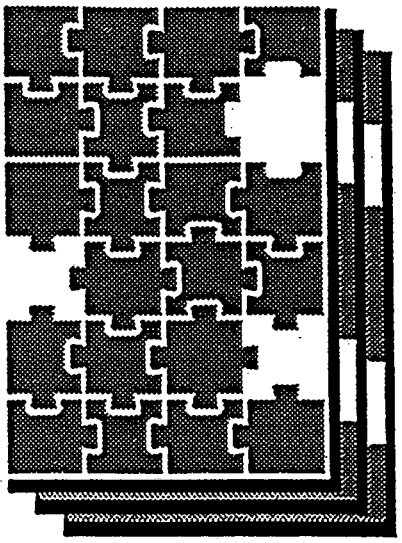


Figure 3. Toolkit Approach to Parallel Computing

For this first implementation, we have provided a FORTRAN interface because users in the computational physics community that are most likely to benefit from this toolkit are either solely or mostly familiar with FORTRAN. Special limitations imposed by integrating FORTRAN required the use of callbacks rather than inheritance, which would be more appropriate to the native language of POET: C++. The top level of the toolkit is initialized by the FORTRAN subroutine: PTINITIALIZE. POET objects, called *pidgets*[2], are sent messages through a FORTRAN subroutine: PTSNDMSG. It is only through these two functions that the toolkit is accessed from FORTRAN. Our current FORTRAN interface is somewhat cumbersome and inefficient. However, as was previously mentioned, our most likely user has developed FORTRAN models as applications.

Currently the user is required to configure and message the pidgets necessary to implement a solution. A high-level intelligent user interface is under development that will completely isolate the user from the details of pidget creation. This will be developed as a frame based interface that provides abstracted representations for standard combustion models. The user will be provided with a graphical user interface in which to customize their particular problem. Automatic code generation will provide the link between the pidgets and the users problem representation. In addition, we expect to eventually

[2]We have named our objects *pidget* as analogous to the widget in the X toolkit.

incorporate automatic code generation for chemical mechanisms.

The frame based user interface will provide the mechanism for managing the complexity of the models. As users customize each application to include more detailed chemistry and turbulence models, the models will become increasingly difficult to manage and verify. In addition to providing an environment that is natural for the user to use, high-level algorithms that provide constraints on model development will force the user to create code that is computationally consistent and is consistent with the modeling algorithms.

CHEMICALLY REACTING FLOW PROBLEM

We present an example application that models a turbulent reacting jet flame [Chen *et al.* 1990]. A cylindrical jet injects a premixed fuel and is ignited. The model is composed of three parts: the turbulent motion model, the chemical reaction model and the coupling between chemical reactions and turbulence. The downstream exhaust is computed using an advancing grid beginning from the inlet and advancing downstream until chemical activity is complete, Figure 4. Symmetry allows this problem to be modeled using a one dimensional grid that effectively represents a radial slice of the cylindrical nozzle. The model is based on the probability density function (PDF) approximation [Pope 1976].

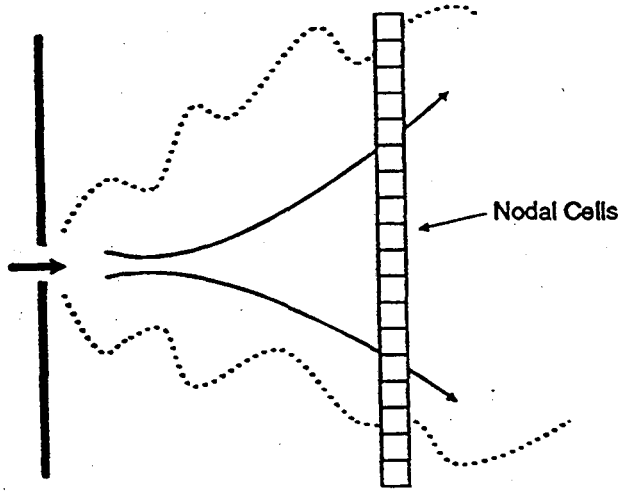


Figure 4 (a) Schematic of the turbulent reacting jet. It is computed by advancing a one-dimensional grid along the flow direction from the inlet outward. The computationally intensive chemistry and turbulent diffusion calculation is done in parallel using FORTRAN callbacks in the manner of Figure 4 (b).

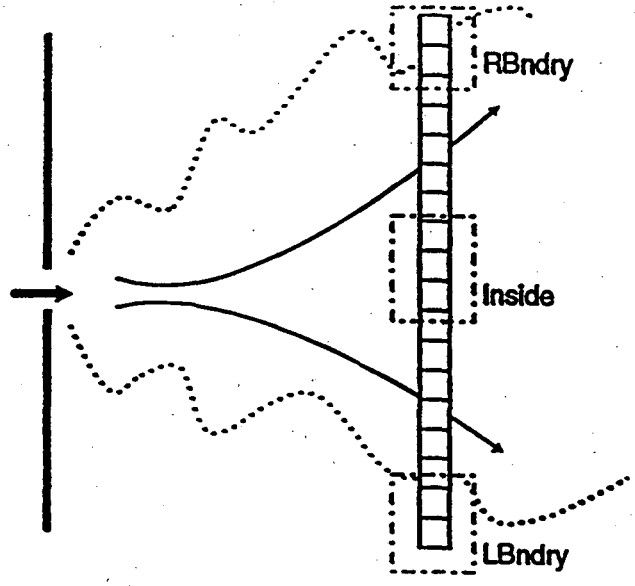


Figure 4 (b) The advancing one-dimensional grid is spatially domain decomposed on the distributed processors. Three callbacks are provided by the user that encapsulate the physics of the reacting jet.

The algorithm is a piece wise Monte Carlo method in which the Monte-Carlo aspect of the problem comes from the transport (or mixing) of particles to and from nearest-neighbor cells. This mixing is a result of fluid-mechanical turbulence and Fickian diffusion. The chemistry that accounts for most of the computation, is local to each cell in which particles from one node cell are required to communicate with only the right and left neighbor cells.

The POET implementation builds upon a piece wise Monte Carlo (PMC) object. The first step is to spatially-domain decompose the linear grid. The PMC object then exploits the nearest-neighbor dependence of the physics of this problem and defines three types of cells: left boundary, right boundary, and interior. The data necessary to compute an interior cell is the cell itself and its right and left neighbors. We define a callback around this called Inside. It receives three objects called *Data Racks* that hold all of the data relevant to the cell and its neighbors. The callback will recover the data necessary to do its computation by sending messages to the Data Racks and, once the computation is accomplished, the callback will send a message updating the relevant data in the Data Racks. The two leftmost cell's Data Racks will be sent to another callback (LBndry) to handle the left boundary condition. The rightmost two cell's Data Racks will be sent to a third callback (RBndry) to handle the right boundary condition. These three routines are sufficient to accomplish the PDF calculation.

The PMC object takes care of processor-to-processor communication such as updating the ghost nodes (nodes that are overlap processors) and returning the results to a host processor. Independent callback routines fill out stubs on the PMC object to provide the physics particular to the PDF problem.

Actual excerpts from the FORTRAN source can give a better idea for how the toolkit is accessed. First, the top level pidget is instantiated and initialized.

```
c ...
    call ptinitialize(itop)
c ...
```

Here itop is an integer that is a label for the topLevel pidget. Other pidgets can be instantiated by sending itop messages, such as:

```
c ...
call ptsndmsg(itop, PtTopInstantiate,
PtMesh1D,ncells,mesh)
c ...
```

Here a mesh pidget is instantiated. Identifiers that begin with "Pt" are defined in an include file and replaced by integers by CPP at compile time. The mesh pidget identifies the problem specific portion of the code. In this case the problem is one dimensional and has ncells cells. The call returns another integer identifier, mesh, identifying the instantiated mesh pidget. Now mesh can be sent messages of its own. Now that the problem space has been identified, an indication of the machine environment is needed.

```
c ...
call ptsndmsg(itop, PtTopInstantiate,
PtMachine, 3, PVM, mach, 'h2jet\0')
c ...
```

This instantiates a machine pidget and returns mach as its identifier. The arguments indicate that there are to be three processors, the communications environment is to use PVM [Beguelin *et al.* 1991] for message passing and the component name to be used by PVM is h2jet. The configuration is the problem environment is complete and we may now create a PMC pidget.

```
c ...
call ptsndmsg7(itop,PtTopInstantiate,
PtPMC,mesh,mach,lbexec,rbexec,mcexec,
ipmc)
c ...
```

To instantiate ipmc we need a mesh and mach pidget and the three callbacks lbexec, rbexec, and mcexec. The first two handle the left and right boundaries respectively and the last handles the interior cells. mcexec for example is declared as:

```
c ..
  subroutine mcexec(idrl,idrc, idrr)
  integer idrl, idrc, idrr
c ...
```

idrl, idrc, and idrr are the left, current and right cell's data embodied as Data Rack objects. The Data Racks are pidgets and the data necessary to do the computation is recovered by messaging the Data Racks. Only the idrc, the current Data Rack, is expected to be modified with updated data. Data associated the cell may be easily accessed by the FORTRAN routine through a simple message. For example, the number of particles in the current cell can be obtained by:

```
c ...
    call ptsndmsg2(idrc,
         PtDRGetDataByString,np,'np\0')
c ...
```

Here np is the FORTRAN variable that will take on the value of the number of particles in the cell and 'np\0' is a null-terminated string identifying that quantity to the idrc pidget. A similar message allows the results of the calculation done by mcexec to be communicated back to the Data Rack.

RESULTS

The entire one-dimensional grid of cells plus boundary conditions for the two cells on left and right ends of the grid compose the Monte Carlo portion of the problem and accounts for 99.3% of the computer time, even with minimal chemistry. At the appropriate time, statistics computed from the particles distributed over the cells contribute to an update of the flow field, either as an iteration converging on a steady state or as a time step in a non steady problem. The addition of realistic chemistry increases the computation required by 10 to 100 fold.

Figures 5 - 7 show the results of this calculation. Figure 5 indicates the temperature profile of the jet flame, where the color is graded from blue that identifies low temperature and red that identifies high temperature. Figures 6 and 7 indicate the NO and OH concentrations, with the color graded as blue for low concentration and red for high concentration. The ability to predict pollutant concentrations, such as NO, as well as intermediate species, such as OH is critical if we are to make an impact on the combustion community. Species such as OH and NO can be measured experimentally to verify models. The model was implemented on a distributed network of 25 IBM RS6000 workstations. The figures were generated using the Advanced Visualization System. The combustion scientist responsible for the physics modeling associated with the PDF model was J-Y Chen of UC Berkeley Mechanical Engineering Department. Dr. Chen was intimately involved in the development of the model

and modified his existing FORTRAN code to integrate with the POET implementation.

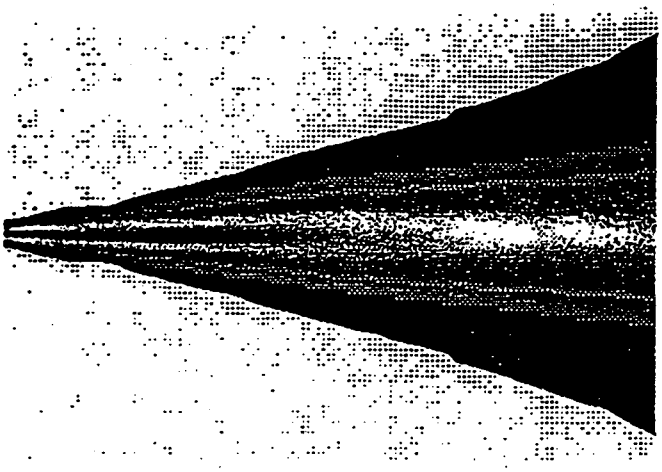


Figure 5. Temperature Profile for Diffusion Simulation

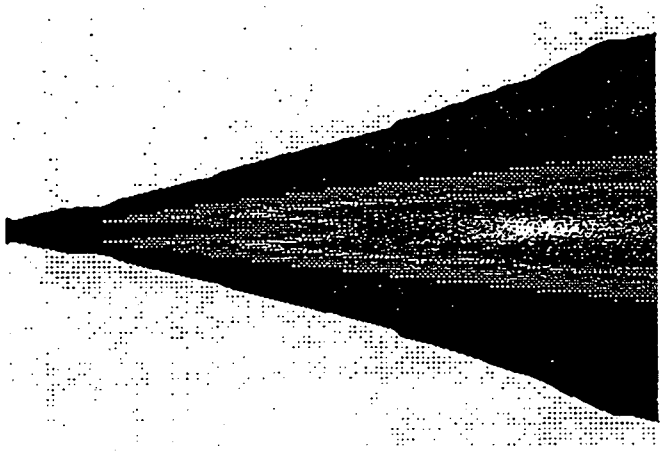


Figure 6. The NO concentration for the same jet flame allows the combustion scientist to predict the formation of pollutants

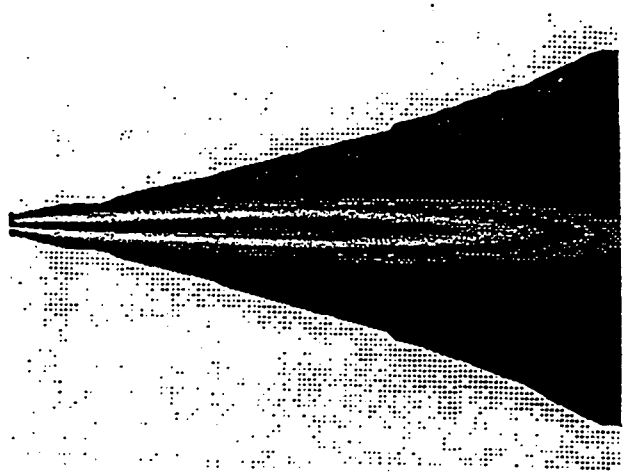


Figure 7. Intermediate species concentrations, such as OH, can be compared with experiment to verify models.

The parallel version of the PDF code utilizing POET is capable of doing an H_2 jet calculation with tens of chemical species and reactions in less than a day on our collection of IBM workstations. We plan to implement the code on a massively parallel computer, an Intel Parragon, and expect to produce the results in minutes.

Through POET, the power necessary to compute PDF chemically reacting flow problems with realistic chemistry on massively parallel and distributed systems has been provided. Moreover, by encapsulating the PMC algorithm in an autonomous object and thus insulating the specific physics of a PDF problem from the algorithm, we produce reusable code that can be used on other chemically reacting flow problems. Ultimately, the update of the flow field will require an equation solver; however, in this early implementation we anticipate avoiding this issue by solving the entire flow field on every processor. Since this step requires little computation and the communication of very few averaged numbers to each processor, it is expected that the lack of scalability in this step will have little impact on the speed-up necessary for this application.

SUMMARY

The results of interfacing the PDF jet flame diffusion model with POET have generated the first investigation of a turbulent non premixed flame with a full chemical mechanism. This benchmark calculation provides an opportunity for examining the limitations of various reduced chemical mechanisms. The described design methodology in concert with the use of an object-oriented language has allowed the development of code that can be continually evolved by both computational scientists and physical scientists. POET has been designed in a modular form that allows a physical scientist to easily modify

their current model for both the fluid mechanics and the chemistry without affecting the code that actually distributes the problem to the various processing elements.

Future extensions of POET to support higher dimensionality and applications requiring efficient equation solvers are planned. Advanced computing techniques, such as automatic code generation for chemical mechanisms, will impose the partitioning necessary to handle the models as their complexity increases. This will create a well defined path for combustion modelers and other computational scientists to formulate and access high performance models customized to their particular application.

This approach to software development is vital to next-generation systems. It is particularly important in computational physics where you have state of the art techniques being applied in both the physical science and the computer science. The partitioning that we are introducing allows development to occur in both scientific areas and allows easy access to distributed resources otherwise untapped by the naive user.

Acknowledgment: The authors would like to gratefully acknowledge the large amount of patience and effort expended by J-Y Chen of the UC Berkeley Mechanical Engineering Department.

Beguelin, A., J. Dongarra, A. Geist, R. Mancheck, and V. Sunderam, "A Users' Guide to PVM Parallel Virtual Machine", *ORNL/TM-11826*, July 1991.

Chen, J-Y, R.W. Dibble, and R.W. Bilger, "PDF Modeling of Turbulent Nonpremixed CO/H2/N2 Jet Flames with Reduce Mechanisms", Twenty-Third International Symposium on Combustion / The Combustion Institute, 1990, pp. 775-780.

Koszykowski, M.L., RC. Armstrong, R.E. Cline Jr., J.F. Macfarlane, J-Y Chen, and N.J. Brown, "The Advanced Combustion Modeling Environment", *American Chemical Society*, Washington DC, August 24-28 1992.

Pope, S.B. ,"The Probability Approach to the Modeling of Turbulent Reacting Flows", *Combustion and Flame*, 27, (1976), p. 299.

LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
TECHNICAL INFORMATION DEPARTMENT
BERKELEY, CALIFORNIA 94720