

PFLAMELET

An Unsteady Flamelet Solver
for Parallel Computers

Implementation and Testing

CS 267 - Spring 2004 - K. Yelick

Fabrizio Bisetti

Department of Mechanical Engineering

UC Berkeley

14th May 2004

I. Introduction

As reported in Warnatz *et al.* [1], combustion is an old technology, which at present provides about 90% of our worldwide energy support (automotive, electrical power generation, heating). Hence, combustion is not likely to become obsolete in the near future. Combustion also challenges our fundamental understanding of fluid mechanics and chemistry, as they come together in describing one of the most complex processes of academic and industrial relevance. In the present paper we are concerned with the solution of a simplified one-dimensional model of a reactive mixture. Such model is considered a valuable building block to be used in describing more complex type of reactive flows [2] (e.g. turbulent flames). The model is known as “unsteady flamelet”. Its main feature is the capability of decoupling chemistry and fluid mechanics, so that a complex process can be reduced to the sum of two other processes, which can be modeled almost independently. This simplification has the direct result of reducing the computational time required to simulate a real system. The unsteady flamelet model has only recently received more attention due to the limited computational resources available in the past.

In this work we describe the steps taken to parallelize an existing unsteady flamelet code using the collection of libraries SUNDIALS [3]. At first we introduce the mathematical problem at hand. Secondly we describe the approach taken. Then we give a description of the targeted platforms (Linux clusters). We also present some results regarding the performance of the program on the targeted architectures for a selected case. We then discuss and analyze the obtained results in order to gain insight on the actual mechanisms internal to the program, thus allowing to rationally plan future work. Finally we draw some conclusions on the work conducted.

II. Problem definition

The solution of an unsteady flamelet model proceeds through the solution of a system of non-linear PDEs. The size of the PDE system grows with the number of chemical species included in the reaction mechanism. Mechanisms can include as little as 10 species for hydrogen ignition [4],

or as much as 500 species for the detailed chemistry of gasoline.

Here below in Eq. 1 we give the structure of the equations forming the system of PDEs, which in simple terms contains only diffusion and source terms (due to chemistry). Note that w_j (where j refers to a given chemical species) is the function relating the composition of the mixture to the production rate of the species (at each point in space). Also, if M is the total number of chemical species considered, the actual number of PDEs is $M + 1$ due to the extra equation needed to solve for the mixture temperature. Note that the evaluation of the source terms, which we have described simply as w_j , is computationally intensive, and the time requirements grow with the number of species considered. Also, the calculation of w_j involves the use of the CHEMKIN collection of libraries [5].

$$\frac{\partial Y_j}{\partial t} = D \frac{\partial^2 Y_j}{\partial z^2} + w_j \quad (1)$$

The PDE system is then discretized over Z points in z -space, yielding an IVB (Initial Value Boundary) problem in $Z(M + 1)$ variables. Use subscript i for the $i - th$ z -point.

$$\frac{\partial Y_{i,j}}{\partial t} = D \frac{\partial^2 Y_{i,j}}{\partial z^2} + w_j \quad (2)$$

Then a three-point stencil is used to discretize the second derivative in z -space, and an equation is written for every z -point, and every chemical species plus temperature. Also an implicit formulation is used for time. Use superscript n for the $n - th$ time step.

$$\frac{Y_{i,j}^{n+1} - Y_{i,j}^n}{\Delta t} = D \frac{Y_{i,j+1}^{n+1} - 2Y_{i,j}^{n+1} + Y_{i,j-1}^{n+1}}{\Delta z^2} + w_{i,j}^{n+1} \quad (3)$$

As an example, gasoline modeling would require $M = 500$. Also we would need at the most 200 points in z -space ($Z = 200$). We then obtain a system of size $N = 100200$. This is arguably the biggest system of interest today. Note that for the development of the present code a smaller, more tractable problem was selected.

A Jacobian arises from the linearization of the non-linear system of ODEs (non-linearity intro-

duced by chemical source terms). In the present configuration, the variables describing the mixture composition at a given z_j point are adjacent one to the other. The resulting Jacobian is a banded matrix with equal upper and lower bandwidths $b_U = b_L = M + 2$. Given the specifications of the problem at hand, an approximation to the upper bound of the sparsity of the matrix can be given as follows

$$sparsity = \frac{2(M + 1) + 2(M + 1)(Z - 3) + (M + 1)^2(Z - 2)}{(M + 1)^2 Z^2} \approx \frac{1}{Z} \quad (4)$$

For example, in the case of $Z = 100$ the sparsity is in the order of 1%. Hence this problem is a sparse one.

III. Approach taken

After describing the mathematical problem at hand, it is clear that a numerical solver has to be used. The sequential version of the code previously developed within our research group uses VODE, a solver for systems of ODEs developed in the past by Brown *et al.* [6], which relies on variable coefficient multistep integration methods for both stiff and non-stiff systems. VODE uses direct linear algebraic techniques to solve the underlying banded or dense linear systems of equations in conjunction with a modified Newton method (stiff ODE case) [6]. Direct methods are very good tools to solve linear systems, but they might require an unacceptable amount of time and space as the system becomes larger [7]. Also, their parallelization seems to pose more difficulties than the parallelization of iterative methods.

A new parallel solver within the SUNDIALS package [3] was selected as a the candidate solver to be implemented. Note that the ODEs solvers contained in SUNDIALS are the underlying engines for the solution of ODE systems in PETSc [8]. Here below we briefly describe the structure of PVODE, SUNDIALS solver for IVPs. The interested reader can learn more about PVODE reading the manual [9].

PVODE is the name of the front-end subroutine to be called to solve an IVB problem. The

problem is currently solved adopting a Backward Differentiation Formula (BDF), recommended for stiff systems. For this kind of systems, Newton iteration is used, and for each iteration an underlying linear system, obtained linearizing the problem, must be solved. PVODE solves the linear system with the iterative method SPGMR (scaled, preconditioned GMRES), a Krylov subspace method [9]. Note that PVODE allows the user to supply routines to calculate the preconditioner. In the present paper, no external routine was supplied due to the limited amount of development time available. Hence, we relied on the module GMRES, which is included in the SUNDIALS package, to handle the preconditioning. Within PVODE and all of the other SUNDIALS modules parallelism is achieved through the so-called NVECTOR module. The idea is that the system of ODEs is distributed over several processors. For any necessary operation, the processor applies the operation on the contiguous elements it owns, while resorting to communication (usually global reductions) when needed [9]. Communication is done in MPI (or SHMEM on Cray's T3E machine). Note that the communication between processors within PVODE will prove to be a problem (see later sections). Finally, a few words on the preconditioner used. The routines responsible for providing a preconditioner matrix belong to the module CVBBDPRE. The preconditioner is block-diagonal with banded blocks. The blocking corresponds to the distribution of the dependent variable vector Y among the processors. Each preconditioner block is generated from the Jacobian of the local part (on the current processor). The blocks are generated by a difference quotient scheme on each processor independently, utilizing an assumed banded structure with given half-bandwidths. A separate pair of half-bandwidths defines the band matrix retained [9].

In order to use PVODE the user should at least provide a subroutine (CVFUN) to compute the RHS of the system, i.e. to compute $\partial Y/\partial t$ given the solution vector Y . Note that the exact calculation of the RHS involves near-neighbour communication due to the diffusion terms. As far as the source terms are concerned, instead, the problem is embarrassingly parallel. Also, PVODE allows the user to provide a second subroutine for the RHS (CVLOCFUN), which is used to compute an approximation to $\partial Y/\partial t$. This local approximation is used when the local part of the preconditioner needs to be calculated. As we will see, this was implemented in a second version of the code

called *pflamelet v2.1*.

After implementing a first working version of the parallel unsteady flamelet code, we also explored the possibility of instrumenting the code using the profiling tool TAU [10] (available on Seaborg). TAU proved to be extremely difficult to use, and we abandoned the idea of complete profiling and tracing in favour of more simple timing calls inserted within the code itself. Nevertheless we highly regret not having been able to successfully profile the code. This would have most certainly enhanced our understanding of code behavior.

IV. Targeted platforms

The targeted platform is, in general terms, a cluster of PCs running Linux. The reasons for choosing this platform lie both in the availability of a newly installed cluster at the Mechanical Engineering Department (which the author will be using for production runs of the developed code), and in the fact that a Linux cluster is probably a more suitable platform for initial code development, as compared to a large machine like Seaborg. In particular, code development, testing and performance analysis have been carried out both on the DECF cluster (kepler.me.berkeley.edu), and on Alvarez (alvarez.nersc.gov). Here below we give a small description of the two clusters, emphasizing the nature of the interconnect between nodes.

A. Davis Etcheverry Computing Facility (DECF)

It's a Linux cluster consisting of 18 PCs. Each node has a single CPU (Pentium IV, 3.02 GHz), with 528 MB RAM memory. The interconnect is Gigabit Ethernet. The cluster is not regulated by any batch system. Nevertheless, the tests were run at night, far away from peak load ours. Also, the results showed very good repeatability.

B. Alvarez cluster within NERSC

Alvarez is a Linux cluster of 80 compute nodes connected by Myrinet. Each node has two 866 MHz Pentium III, and 1 GB of SDRAM. Alvarez runs the batch system PBS, through which jobs need to be submitted.

V. Code development

As we started this project, we set as our goal to implement a working version of the code, and to perform tests on it. These goals have been met. Firstly the available FORTRAN sequential code has been thoroughly cleaned up and reorganized. Secondly the first version of the code has been implemented (*pflamelet v1.1*). This version implements CVFUN as the only function to compute the RHS. After performing some tests, we implemented also a second version, *pflamelet v2.1*, in which a local approximation to CVFUN is used most of the time while calculating the block banded preconditioner. This second version exploits non-blocking communication. Also an intermediate version using local approximation, but blocking communication has been used (*pflamelet v1.2*).

VI. Test case description and benchmarking

When testing the performance of PVODE for the specific application in which we are interested, great care has to be put in the selection of a test case. The following considerations apply. The computation time taken at each time step depends strongly on the initial conditions, given the iterative nature of the linear system solver (Krylov subspace method), and the type of physics characteristic of igniting mixtures. Hence, the simulation has to proceed far enough in time that the system starts to exhibit its real behavior, and forces the iterative solver to a more substantial number of iterations. This means that microbenchmarking, i.e. running a very small number of time steps, can be misleading and rather difficult to perform. Secondly, controlling the problem size is subtle. In fact, as M increases, not only the size of the ODE system increases, but also the time required

for the chemistry subroutines to return the source terms increases more than proportionally. This means that test cases which have the same overall system size N but different number of species shouldn't be compared. Problems with more chemical species inherently require a lot more computation time. On the other hand, a simple way of enlargening the problem size would be this of increasing the resolution in z-space. This would increase the size of the system, without increasing the time spent in computing the source terms at each z-point. This is not very satisfying either, as $Z = 100$ is a good enough resolution for real applications, and any further addition of points would be artificial and instrumental to force a better scalability only. To conclude, the ideal test case should be small enough not to require too long to compute, but still be large enough to display some scalability. In particular we select a well known chemistry mechanism for methane combustion, GRI3.0 [11]. This mechanism has 53 species and 325 reactions. We select $Z = 100$. Hence the ODE system has size $N = 5400$. We also take a sufficient number of steps in time as to reach ignition of the mixture. As we will see in the next section, judging on the basis of the data on scalability, this proved to be a somewhat small problem. Nevertheless, it allowed studying the code behavior, and postulating an explanation for the degradation of performances as the number of PEs increases.

VII. Results of the tests

Tests were conducted both on the DECF cluster and on Alvarez. For the DECF cluster, the different versions of the code have been run on only up to 4 PEs, due to the fact that the cluster was always extremely busy. For Alvarez, instead, the code has been tested on up to 20 PEs (10 nodes consisting of 2 cpus each). As far as *pflamelet v2.1* (local approximation to RHS) is concerned, quantitative comparison between runs on a different number of PEs has to be done carefully, since somewhat different convergence patterns were detected, depending on the number of PEs used. This is due to the different approximations calculated locally, which change depending on the distribution of the Jacobian among processors. We devised a way of normalizing the data so that meaningful

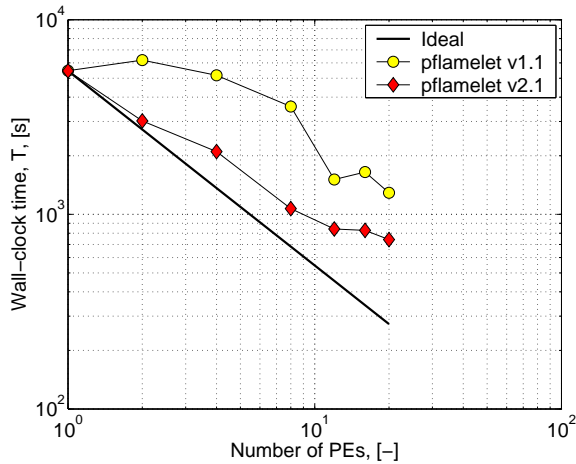


Figure 1: Wallclock time plots for runs of *pflamelet v1.1* and *pflamelet v2.1* on Alvarez cluster

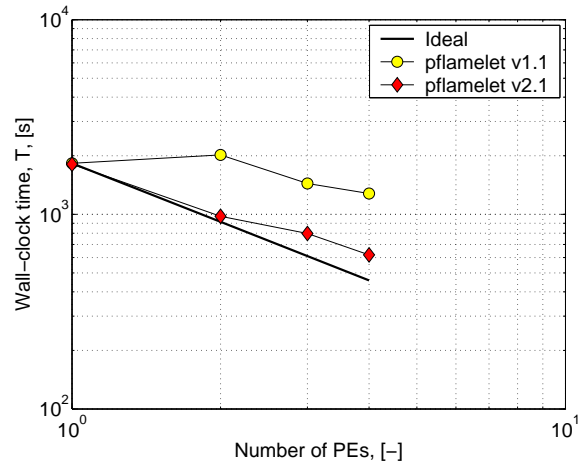


Figure 2: Wallclock time plots for runs of *pflamelet v1.1* and *pflamelet v2.1* on DECF cluster

comparisons can be made. This is explained in the subsection D. We present wall-clock time plots, speed-up plots, and efficiency plots based on the total wall-clock time. These are probably the most important pieces of information in real applications. Also we present time break-down plots which try to explain the performance behavior of the code. Also we briefly report on the convergence patterns of the Krylov method.

A. Wall-clock time

As clearly shown in Fig. 1 and Fig. 2 the decrease in time doesn't follow the ideal behavior. This is true for both platforms. Test runs on Alavarez generally took longer due to the slower processors. Just as a side note, for a single cpu (PE = 1), the same test case ran approximately 3.4 times faster on DECF (Pentium IV, 3.02 GHz) than on Alvarez (Pentium III, 880 MHz), which is the ratio of the clock-frequencies. In particular, *pflamelet v1.1* performed extremely poorly on both platforms. Reasons are probably related both to the convergence pattern of the method, and to scalability issues of a certain part of the code, namely this responsible of managing the preconditioner.

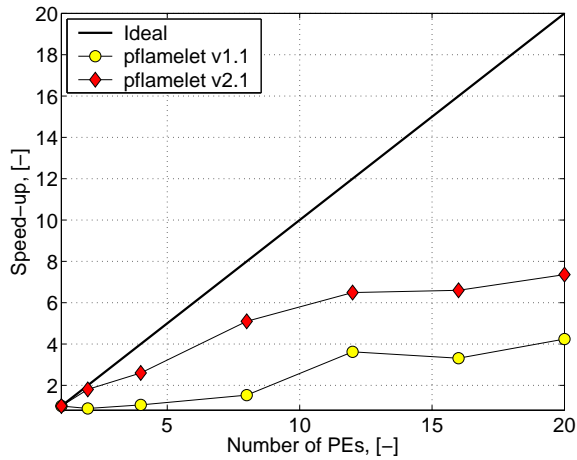


Figure 3: Speed-up plots for runs of *pflamelet v1.1* and *pflamelet v2.1* on Alvarez cluster

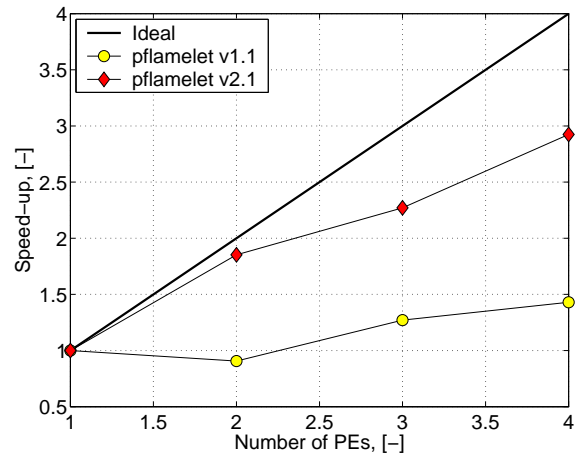


Figure 4: Speed-up plots for runs of *pflamelet v1.1* and *pflamelet v2.1* on DECF cluster

B. Speed-up

The achieved speed-up is very bad for *pflamelet v1.1* and barely acceptable for *pflamelet v2.1* (see Fig. 3 and Fig. 4). For this second version, at the max number of PEs tested, the parallel efficiency was down to 40% on Alvarez (20 PEs), and to 70% on the DECF cluster (4 PEs only). For the case of *pflamelet v1.1*, the behavior of the parallel code is so bad that scalability drops below one, indicating that the time taken running the code on two processors takes even longer than in the serial case. Hence the use of CVLOCFUN to approximate locally the RHS of the ODE system is absolutely necessary. Generally speaking the behavior of *pflamelet v2.1* is almost the same on the two platforms. Unfortunately, due to the heavy use of the DECF cluster, more extensive profiling (more PEs) could not be performed.

The behavior of the two versions of the code is rather similar on both clusters. The scalability on Alvarez seems to be achieving a slightly less efficiency. This is probably somewhat counter-intuitive, at least for two reasons. Firstly the interconnect, and secondly the fact that PEs in Alvarez are grouped two by two in a single node.

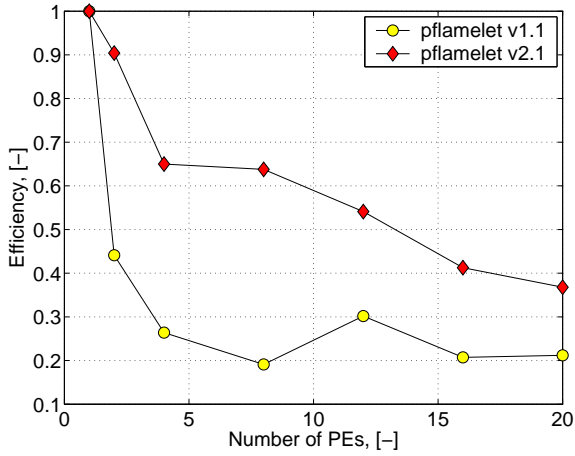


Figure 5: Efficiency plots for runs of *pflamelet v1.1* and *pflamelet v2.1* on Alvarez cluster

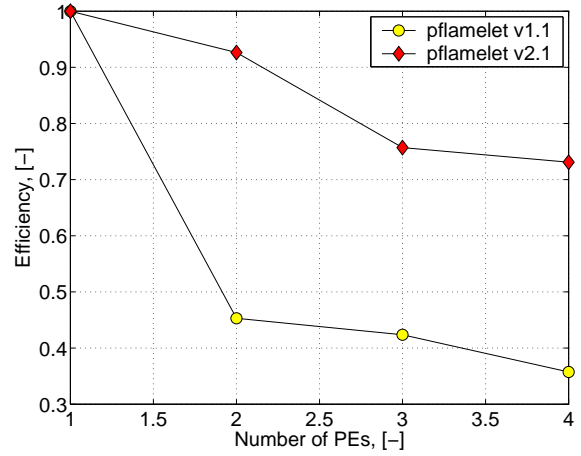


Figure 6: Efficiency plots for runs of *pflamelet v1.1* and *pflamelet v2.1* on DECF cluster

C. Efficiency

We also show the efficiency plots for the data reported in the previous subsections. Figure 5 and Fig. 6 show the achieved efficiencies. Note that, for the same number of PEs, the code seems to be scaling better on the DECF cluster.

D. Time break-down

Inspection of the time taken by each of subroutines during code execution is necessary to describe the progressive loss of scalability of the code. But first let us present a schematic diagram of the convergence pattern of *pflamelet v1.1* and of *pflamelet v2.1* run on both Alvarez and DECF. Figure 7 shows on the x-axis the number of exact RHS evaluations (CVFUN), and on the y-axis the number of local RHS evaluations (CVLOCFUN). Each point corresponds to a run. The legend on the side allows to identify the platform and the version of the code. The same symbol is used for runs with different number of processors. Note that *pflamelet v1.1* always required many more evaluations, while the convergence pattern associated with *pflamelet v2.1* required way less evaluations. Also, in the case of *pflamelet v2.1* the data are less scattered, indicating that, regardless of the number of PEs, the number of CVFUN/CVLOCFUN evaluations, and the number of preconditioner

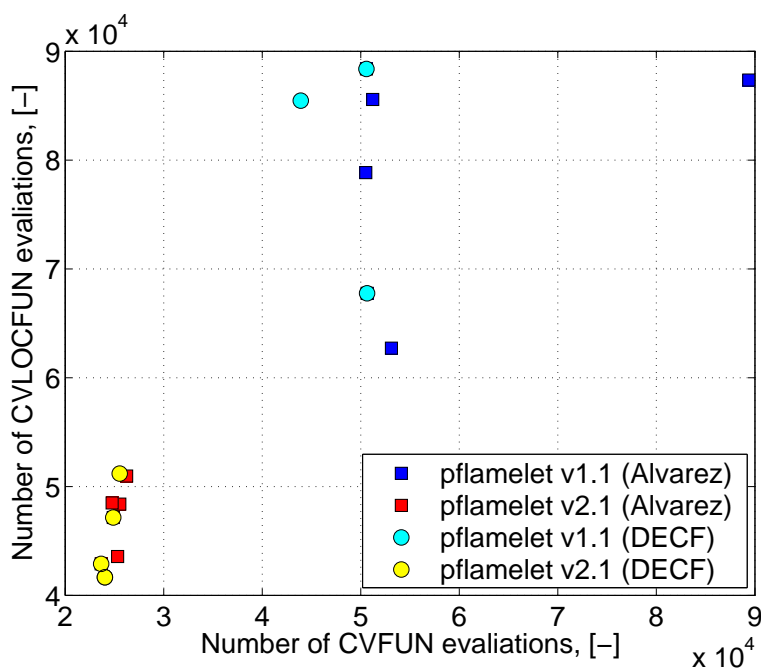


Figure 7: Plot showing the convergence pattern for *pflamelet v1.1* and *pflamelet v2.1* run on Alvarez and on DECF. Note the higher number of evaluations taken by *pflamelet v1.1*

setup, evaluation, and solutions changes moderately as the number of PEs changes. Hence, time break-down will be presented for *pflamelet v2.1* only, as this can be considered almost constant among tests run on different number of PEs.

In the time break-down plots note that the cumulative time across processors is reported (average multiplied by the number of PEs). Figure 8 shows the time break-down for *pflamelet v2.1* on Alvarez. In this plot note how the time spent in the chemical subroutines scales perfectly. Also the MPI time within CVFUN, which is necessary to calculate the diffusion terms, doesn't burden the final running time. As a result, the evaluation of the RHS scales perfectly. The real problem lies in the time labeled "OTHER", which was obtained subtracting the time spent in CVFUN or CVLOCFUN from the total time in PVODE. Arguably this share of computational time is related to various operations connected with algorithm implemented in PVODE, like the preconditioner setup, evaluation, and solution. These operations are likely to be burdened by excessive communication overhead, compared to the parallel work in each PE. Unfortunately we were unable to time the details of the mechanisms internal to PVODE. Nevertheless, it is clear that the evaluation of

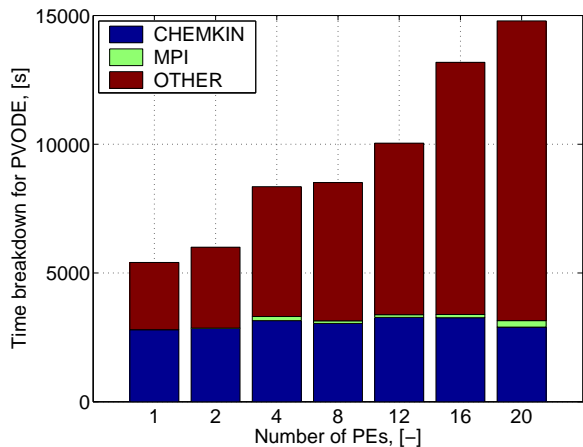


Figure 8: Cumulative time break-down for runs of *pflamelet v2.1* on Alvarez cluster

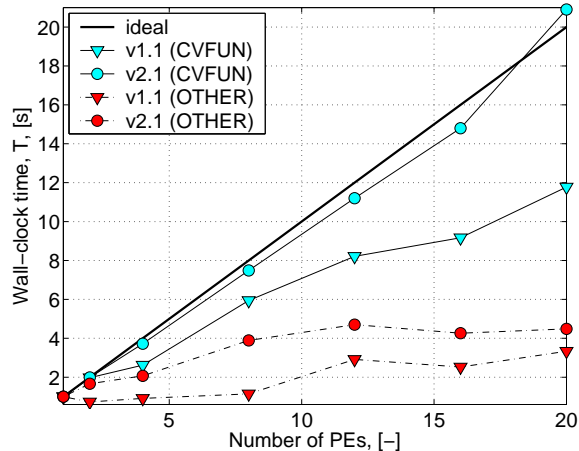


Figure 9: Scalability plots for single components of PVODE for *pflamelet v2.1* run on Alvarez cluster

the RHS scales almost perfectly (see Fig. 9), while the remaining part of the algorithm scales very poorly. Note the unbounded increase of the share of time labeled OTHER in Fig. 8. The same considerations apply to the tests run on the DECF cluster (see Fig. 10 and Fig. 11). The reasons for such bad scalability are most certainly due to the small size of the problem, i.e. $N = 5400$. Future work will involve testing the code for larger mechanisms. This couldn't be done due to time constraints.

In order to produce Fig. 9 and Fig. 11 we normalized the total time spent in CVFUN/CVLOCFUN by the number of calls to these subroutines, in order to be able to compare timings across different numbers of PEs.

VIII. Comments

In this section we'd like to report some general comments mainly related to the use of performance tools. Through the development of the present project it has been felt the need to profile the code with something more adequate and detailed than simple timers manually inserted at key-points inside the code. A rather extensive effort has been undertaken to implement TAU [10], but with no success. The author wishes to express his hope that in the future performance tools will become

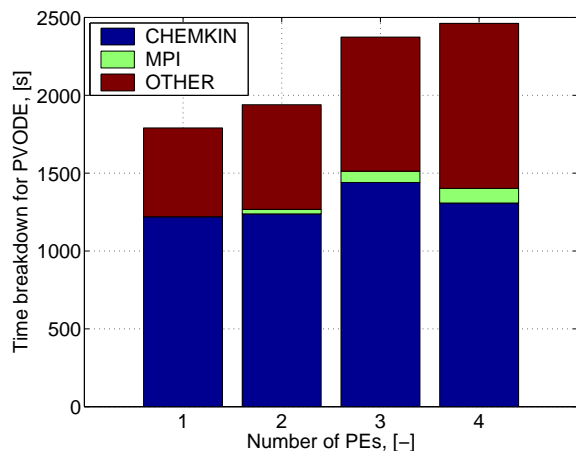


Figure 10: Cumulative time break-down for runs of *pflamelet v2.1* on DECF cluster

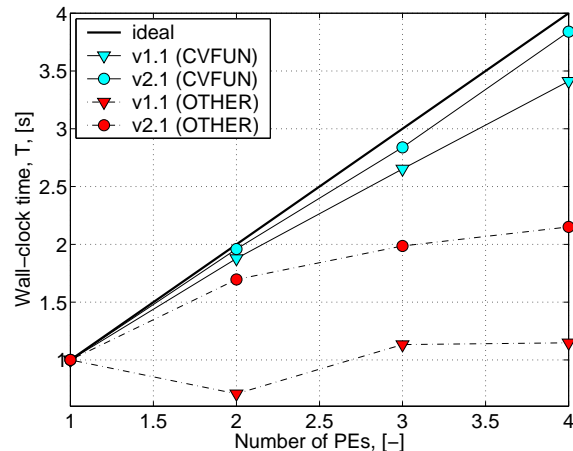


Figure 11: Scalability plots for single components of PVODE for *pflamelet v2.1* run on DECF cluster

more user-friendly, especially considering the fact that profiling tools are to be used also by a large community of people whose main interest is to analyse the results of numerical simulations, rather than to spend time on the implementation of the code itself. Furthermore, developers of mathematical libraries should also look into the possibility of providing pre-instrumented versions of the libraries, as to facilitate the users in the assesment of the performances of their programs.

IX. Conclusions

A parallel code for unsteady flamelet simulation have been implemetned and tested on two different Linux clusters. Two different versions of the code has been produced and profiled. The version including a local approximation to the Jacobian (*pflamelet v2.1*) showed better performance. In any case, the test runs indicate that the code doesn't scale too well. The user defined function evaluation (CVFUN, or CVLOCFUN) scales nearly in an ideal fashion (almost embarassingly parallel), while the steps internal to the PVODE solver don't scale well, probably due to the small size of the problem (ODE system of size 5400). Unfortunately, time constraints prohibited testing the code on a larger problem (ODE sytem size 100000). This will be most certainly done during production runs involving more detailed gasoline chemistry.

Acknowledgments

The author wishes to thank Jason Riedy (UC Berkeley), Tony Drummond (LBNL), and Osni Marques (LBNL) for the help and words of advice kindly granted in the process of developing the code, implementing the SUNDIALS library, and attempting to use TAU.

References

- [1] Warnatz J., Maas U., Dibble R. W., “Combustion”, Berlin, Springer, 2003.
- [2] Peters N., “Turbulent combustion”, New York, Cambridge University Press, 2000.
- [3] <http://www.llnl.gov/CASC/sundials>
- [4] Mueller M. A., Kim T. J., Yetter R. A., and Dryer F., *Int. J. Chem. Kin.*, **31**:113 (1999)
- [5] Kee R. J., Rupley F., Miller J., Coltrin M., Gear J., Meeks E., Moffat H., Lutz A., Dixon-Lewis G., Smooke M., Warnatz J., Evans G., Larson R. Mitchell R., Petzold L., Reynolds L., Caracotsios M., Stewart W., and Glarborg P., *User Manual, The CHEMKIN Collection III* (1999)
- [6] Brown P. N., Byrne G. D., Hindmarsh A. C., *VODE, a variable-coefficient ODE solver*, *SIAM J. Sci. Stat. Comput.*, **10** (1989), 1038-1051.
- [7] Demmel J. W., *Applied numerical linear algebra*, SIAM, Philadelphia, 1997.
- [8] <http://www-unix.mcs.anl.gov/petsc/petsc-2/index.html>
- [9] Byrne G. D., Hindmarsh A. C., *User documentation for PVODE, an ODE solver for parallel computers*, Center for Applied Scientific Computing, UCRL-ID-130884 (1998)
- [10] <http://www.cs.uoregon.edu/research/paracomp/tau/tautools/>
- [11] http://www.me.berkeley.edu/gri_mech/