Numerical simulation of reactive flow on the IBM ES/3090 Vector Multiprocessor

by F. K. Hebeker
R. R. Maly
S. U. Schoeffel

Prohibiting knock damage in internal combustion engines presents severe restrictions for engineers. Laboratory experiments are expensive or even impossible; nevertheless, numerical attempts that employ supercomputers have been rarely undertaken. The numerical approach described in this paper combines a recent shock-capturing finite-volume scheme for the compressible Navier Stokes equations, with semi-implicit treatment of the chemical source terms.

An algorithm is described and validated by experiment that is optimally adapted to vector and parallel computers. The algorithm has been implemented on the IBM Enterprise System/3090™ (ES/3090™) Vector Multiprocessor. Performance measurements are discussed. The potential of the code is illustrated by an example: formation of pseudo shock waves due to interaction of a shock wave with turbulent boundary layer flow.

This paper reports on some results of a joint research project undertaken to gain insights into physical reasons of knock damage in internal combustion engines. The work combines engineering and mathematical modeling with algorithmic and code development for vector and parallel processing on IBM supercomputers.

Due to its significance to many problems of engineering application, the development of numerical algorithms to simulate reactive flow has attracted considerable interest over recent years.[1] At the same time, research on the problem of knocking phenomena in combustion engines seems to be rarely reported in the literature.[2] Aside from the corporate sensitivity, one reason for minimal reporting of the knocking phenomenon might be its numerical complexity, essentially due to complex interaction of diffusive effects with shock waves and reaction fronts (propagating and interacting with each other and with walls) and due to the presence of nonequilibrium combustion phenomena. This paper reports on the combination of some recent numerical methods of computational fluid dynamics (CFD), namely novel shock-capturing techniques introduced by VanLeer[3] and Roe et al.,[4] with (semi-)implicit algorithms to treat stiff chemical source terms.[5]

A FORTRAN program called PICUS (PIston Crevice nUmerical Simulation)[6] was developed to study such flows. The algorithm has been validated by experimental data and is currently under application to model flow under knock conditions in an engine knock simulator. This paper concentrates on some performance considerations concerning

vector and parallel processing on the IBM Enterprise System/3090* (ES/3090*). PICUS has excellent features to take advantage of vector and parallel computer architectures.

The mathematical model uses the compressible Navier Stokes equations (in thin-layer approximation) in an L-shaped domain, with appropriate (rather complex) boundary conditions. The chemical model deals with a two-step model of exothermic reaction (triggered after the induction-delay period), which describes the globally complex detailed reaction kinetics mechanism.

The numerical approach is based on the splitting idea.[7] The flow domain is treated by means of a domain-splitting approach. This proves particularly useful in view of vector and parallel processing.[6] The L-shaped domain is split into rectangles and, by alternating between both subregions, dimensional splitting is used to reduce the problem further to successive one-dimensional ones. Finally, by splitting the operator, the model allows the resulting system of conservation laws (including source terms) to be split up with respect to diffusive, convective, and reactive terms, according to the governing transport phenomena. In particular, the convective part is treated by means of a second-order VanLeer finite-volume scheme,[3] including Roe's approximate Riemann solver.[4] For the reactive part, a semi-implicit method of discretization (trapezoidal rule) is employed.[5]

The main part of the paper is devoted to the question of optimizing the code and how to vectorize and parallelize it to run efficiently on the IBM ES/3090. The paper briefly recalls the tools available to optimize FORTRAN codes for IBM supercomputers and presents performance results. A speedup of about 2.5 due to vectorization and an additional speedup of about 1.75 due to parallelization using two processors has been achieved. The latter corresponds to an activated degree of parallelism of more than 85 percent.

The paper deals with an extensive physical example, the interaction of a reflected shock wave interfering with the shock-tube boundary layer and forming pseudo shock waves (wave train).

Governing differential equations

The underlying mathematical model takes advantage of the nonstationary thin-layer Navier Stokes equations under exothermic chemical reactions, which read in divergence form

$$\frac{\partial}{\partial t} U + \operatorname{div} F(U) = D(U) + S(U) \tag{1}$$

where

$$U = (\rho, \rho u, \rho v, e, Y)^T,$$

$$F = \begin{pmatrix} \rho u & \rho v \\ \rho u^2 + p & \rho uv \\ \rho uv & \rho v^2 + p \\ (e+p)u & (e+p)v \\ uY & vY \end{pmatrix},$$

$$D = \begin{pmatrix} 0 \\ \mu u_{yy} \\ \mu v_{yy} \\ \lambda \Theta_{yy} + \mu(u_y^2 + v_y^2) \\ 0 \end{pmatrix},$$

$$S = \begin{pmatrix} 0 \\ 0 \\ 0 \\ q\sigma \\ -\sigma \end{pmatrix}$$

and

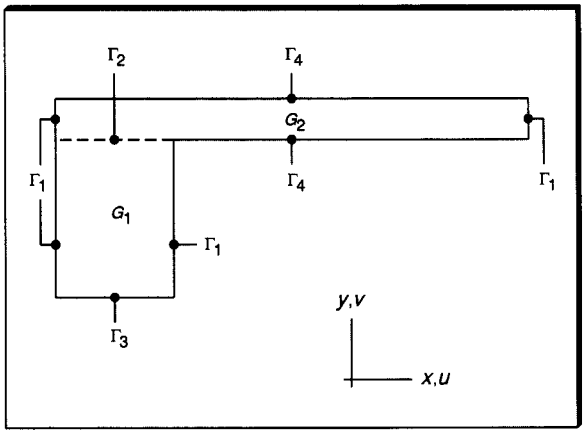
$$p = (\kappa - 1)\left(e - \frac{1}{2}\rho(u^2 + v^2)\right), \quad \Theta = \frac{p}{\rho R}$$

$$\sigma = Y \exp\left(\beta\left(1 - \frac{1}{\Theta}\right)\right) \tag{2}$$

Here $\rho, u, v, e, Y, p, \Theta$ denote density, velocity in x-, y-direction, total energy, reaction progress parameter (according to a two-substances model, here: volume concentration of fuel), thermodynamic pressure, and temperature in the flow field; $\kappa, \lambda, \mu, q, \beta$ denote the isentropic exponent, thermal conductivity, dynamic viscosity, heat release, and global activation energy of the explosive gas (all quantities assumed as constants).

The underlying L-shaped computational domain G for the flow problem studied here is shown in Figure 1. This domain is split into two subdomains G_1 and G_2. We assume Euler flow in G_1:

Figure 1 Domain, its decomposition and boundaries



$$\lambda = \mu = 0 \; in \; G_1 \tag{3}$$

so that we assume for all flow quantities as boundary conditions:

$$\frac{\partial}{\partial n} = 0 \; on \; \Gamma_1 \tag{4}$$

with the exception of vanishing normal velocity:

$$v = 0 \; on \; \Gamma_1. \tag{5}$$

On the coupling boundary Γ_2 all corresponding variables are continuously aligned when alternating between both subregions. The open boundary condition on Γ_3 is chosen in a way to allow for smooth outflow of the reflected waves (absorbing boundary condition).

In G_2 we assume the thin-layer equations to hold with

$$\lambda, \; \mu > 0 \; in \; G_2.$$

Consequently, we choose the same boundary condition on Γ_1 as before, but require no-slip conditions on Γ_4:

$$u = v = 0 \; on \; \Gamma_4. \tag{6}$$

For the temperature we set

$$\Theta = \Theta_w \; on \; \Gamma_4 \tag{7}$$

where Θ_w is obtained from the condition of continuous heat flux (Fourier's law). Following Hirschel and Groh,[8] we obtain an additional condition for ρ from the continuity equation:

$$\rho_t + \rho v_y = 0 \; on \; \Gamma_4. \tag{8}$$

Finally, the thermal equation of state then yields the wall pressure on Γ_4.

The numerical algorithm

The algorithm takes full advantage of the splitting idea, combining domain splitting with dimension and operator splitting on each time step, with time increment Δt (in fact, the method uses Strang's symmetric second-order splitting).

First, the L-shaped domain is split into two rectangular subdomains G_1 and G_2 (see Figure 1). Each of the rectangles is decomposed into a total of $(NXj - 3) \times (NYj - 3)$, $j = 1, 2$ equally sized interior cells, and a lot of further cells serve for smooth data flow between the subdomains as well as for proper modeling of the boundary conditions. Both subregions are treated alternatingly, so that the following numerical procedure needs to be described for one subdomain only. The essence of this procedure is twofold:

1. Complex geometries are reduced to a set of rectangular subdomains, allowing for dimensional splitting.
2. The subdomains may be treated independently, such that coarse-grained parallelism is introduced.

Let us treat, as an example, the subdomain G_2. We use dimensional splitting to update the flow quantities alternatingly in x- and in y-direction. This means that, for any time step, for instance in x-direction, a system in one space dimension

$$\frac{\partial}{\partial t} U + \frac{\partial}{\partial x} F^1(U) = D(U) + S(U) \tag{9}$$

is solved (F^1 denoting the first column of F), subjected to the given boundary conditions. The essence of this procedure is just to reduce a two-dimensional problem to a one-dimensional one.

Due to their essentially different qualitative behavior, we split the system (Equation 9) further into a convective-diffusive part

$$\frac{\partial}{\partial t} U + \frac{\partial}{\partial x} F^1(U) = D(U) \tag{10}$$

and a chemokinetic part

$$\frac{\partial}{\partial t} U = S(U) \tag{11}$$

both to be solved numerically alternatingly on short time intervals.

First consider Equation 11. Assuming the density and the flow velocity as constant in time during the reaction step, we obtain a system of ordinary differential equations (ODE) to be solved for $(\Theta, Y)^T$ for each cell. Since the system is stiff in general, we used a semi-implicit trapezoidal rule (which corresponds to solve a 2×2 linear algebraic system for each time step) in order to update the unknowns.

Finally, as the core of the algorithm, consider the convection-diffusion problem (Equation 10). Here we adopt novel shock-capturing schemes for the convective part with explicit treatment of the diffusive terms. This strategy, proposed by MacCormack and applicable to high Reynolds number flow, is called "Rapid Solver" algorithm.[9] Consequently, the algorithm is composed as follows. Assuming U^n to be given on the time level t_n, we compute U^{n+1} from this strategy (where V summarizes the primitive variables ρ, u, v, p, Y corresponding to U, and Δx represents the mesh width).

The algorithm is stated as follows:

1. Compute

$$(V^n_{i\pm 1/2})^{\mp} = V^n_i \pm \frac{1}{2}\,\delta V^n_i \tag{12}$$

where

$$\delta V_i = ave(V_{i+1} - V_i,\, V_i - V_{i-1}) \tag{13}$$

with VanAlbada's slope limiter

$$ave(a,b) = \frac{a+b}{2}\left(1 - \frac{(a-b)^2}{a^2 + b^2 + \sigma}\right) \tag{14}$$

($\sigma > 0$ a small bias of order $O(\Delta x^2)$).

2. For an intermediate time step $t_{n+1/2}$ compute

$$U^{n+1/2}_i = U^n_i - \frac{\Delta t}{2\Delta x}\{F(V^n_{i+1/2})^- - F(V^n_{i-1/2})^+\} + \frac{\Delta t}{2} D(U^n_i)$$

then

$$(V^{n+1/2}_{i\pm 1/2})^{\mp} = V^{n+1/2}_i \pm \frac{1}{2}\,\delta V^n_i \tag{15}$$

and finally

$$F_{i+1/2} = F_{Roe}((V^{n+1/2}_{i+1/2})^-,\, (V^{n+1/2}_{i+1-1/2})^+) \tag{16}$$

(see[4] for the averaged Roe fluxes F_{Roe}).

3. Update

$$U^{n+1}_i = U^n_i - \frac{\Delta t}{\Delta x}(F_{i+1/2} - F_{i-1/2}) + \Delta t\, D(U^{n+1/2}_i). \tag{17}$$

Concerning the time step size, we assume the latter to be governed by the convective-diffusive part, since the chemical part is treated semi-implicitly (with unlimited stability). Further, since the diffusive coefficients are only small, the well-known restrictions for the time step are of the same order of magnitude for both diffusive and convective parts of the transport equations. This is the reason that, for simplicity, the diffusive terms are treated here by means of an explicit predictor-corrector scheme. In fact, for maximum time step size Δt_{n+1} we allow

$$\Delta t_{n+1} = 0.9 \min(\Delta t^{\times},\, 2\Delta t^{\times} - \Delta t_n) \tag{18}$$

where

$$\Delta t^{\times} = \left(\frac{1}{\Delta t_D} + \frac{1}{\Delta t_C}\right)^{-1} \tag{19}$$

(Δt_D, Δt_C denote the maximum time steps due to explicit treatment of the diffusive or convective parts of the differential equations, respectively.)

The PICUS program

Based on the present algorithm, a new FORTRAN program called PICUS has been developed as a flexible tool to study a wide range of phenomena in the realm of gas dynamics of combustion (reactive compressible flow). We discuss the implementation of the (two-dimensional) version labeled PICUS-2 (or its basic version PICUS-200) on the IBM ES/3090 with the Vector Facility.

Both subdomains (see Figure 1) are served by a total of six large arrays, *STATEj*, *PLINj*, and *FLUXj* (j = 1, 2), each of which has the dimension

$$5 \times (NXj + 1) \times (NYj + 1).$$

In the following we restrict ourselves to consider only the domain G_2, since for G_1 similar terms hold. In practical applications we commonly used

$$NX2 \approx 1000, \quad NY2 \approx 100$$

(underscoring the need of supercomputers).

The solution is stored in *STATE2*, alternatingly in primitive or conservative variables. The array *PLIN2* contains the (limited) slopes as entries of the approximate Riemann solver. And the array *FLUX2* serves as a work space for both the primitive variables and the fluxes. Consequently, the specified large arrays are exploited as far as possible.

For achieving high performance rates with the Vector Facility it is crucial that the first argument of these arrays is equal to the largest size (*NX2*), which leads to long vectors.

The program is structured as follows. The main program contains preprocessing (initialization, domain decomposition, ...) and the time stepping loop. For any time step the twin subroutines *ADVANCEX* and *ADVANCEY*, serving to update the solution by dimension splitting, are called for each of the subregions G_1 and G_2.

The principal subroutines then are *PIECELIN#* and *SOLVEROE#* (where # stands for x or y), called once by *ADVANCE#*. Here the routine *PIECELIN#* serves to compute the flow quantities for an intermediate time step by means of a (slope-limited) finite-volume method. *PIECELIN#* contains the solver for the ODE system, too. After that the second routine *SOLVEROE#* is called, where Roe's scheme to compute the fluxes (on the intermediate time level) followed by conservatively updating the solution is used. Both routines *PIECELIN#* call a function *AVE* where the slope limiter is evaluated.

The merits of the new code PICUS-2 show up when compared with PLM2DTL, an academic code (of the RWTH Aachen) with related objectives in mind.[10] Here the parameters were $NX2 = 128$, $NY2 = 16$, with 10 time steps run (the domain G_1 treated by PICUS only is modeled by a coarse grid so that its contribution to the total CPU time is low). It turns out that the basic version PICUS-200 reduces the elapsed time by factor 10 and its tuned version PICUS-230 by factor 20 (all runs carried out in vector mode).

The subsequent section shows how to achieve this improvement.

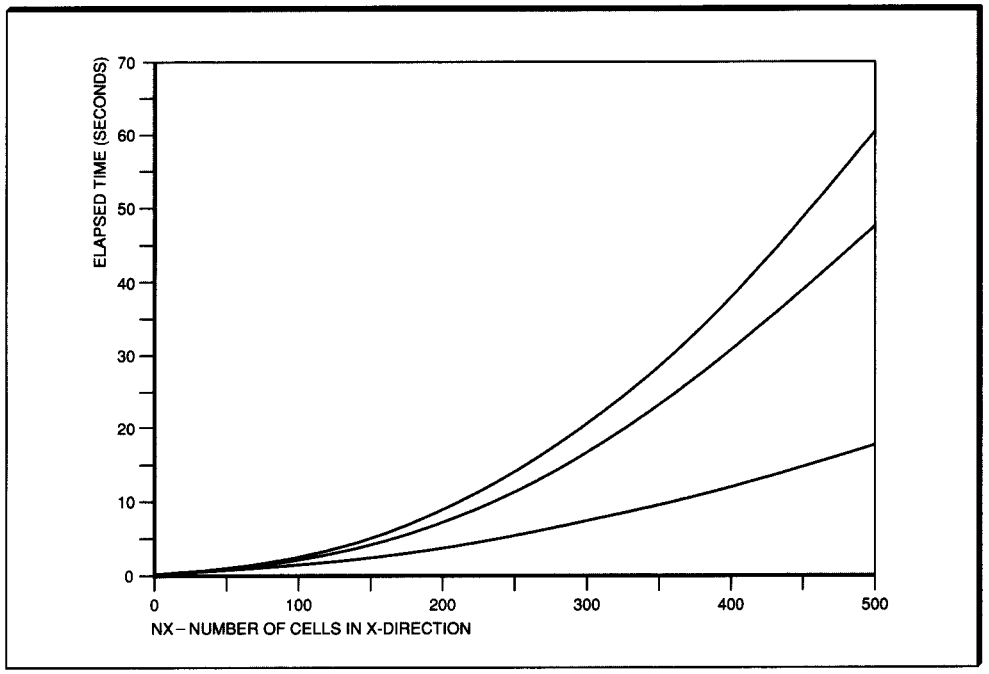
Vectorizing and optimizing

Available since the midseventies, vectorization is now well established as a practical tool to essentially reduce the CPU time for application codes. Taking full advantage of the IBM ES/3090 Vector Facility (VF) requires some knowledge of its special structure. But the present section shows that only little special skill of the VF is required to attain an essential saving of CPU time. The key is to structure the data and their flow favorably (arrays *STATEj*, *PLINj*, *FLUXj*, and their handling!).

The result is shown in Figure 2 where elapsed times (in seconds) are given as a function of $NX2$ (with $NY2 = 0.1 \times NX2$) for: the basic version PICUS-200 running in scalar mode (upper graph), the basic version PICUS-200 running in vector mode (middle graph), and the optimized version PICUS-230 running in vector mode (lower graph).

Consequently, even the basic version enjoys a slight vector-to-scalar speedup of about 1.2 for the IBM ES/3090, which has an excellent scalar processor. In the following we explain how to tune this code in order to attain a vector-to-scalar

Figure 2 PICUS-200 basic scalar (upper graph) vs PICUS-200 basic vector (middle graph) vs PICUS-230 optimized vector (lower graph): elapsed time (seconds) vs number of cells in x-direction



speedup of about 2.5 (see the smoothed graph of Figure 3).

The following strategy leads to efficient reduction of CPU time:[11]

1. A "hot-spot analysis" (a tool provided by the IBM VS FORTRAN Interactive Debug) yields run-time statistics of the utilization of the total CPU time required by the subroutines. This points out which subroutines are tunable with best efficiency.
2. The compiler vector report points out those DO-loops run in vector or scalar mode, indicating where vectorizing of the single DO-loops should be started.
3. In some cases it is hardly possible to create vectors in a DO-loop (due to some recurrence $a_i = a_{i-1}$, for instance). In this case an exchange of the whole subroutine or the pertaining part of the algorithm is required.

Limiting the problem to the essential features of the code, we summarize the tools and means to optimize. The main tools are as follows:

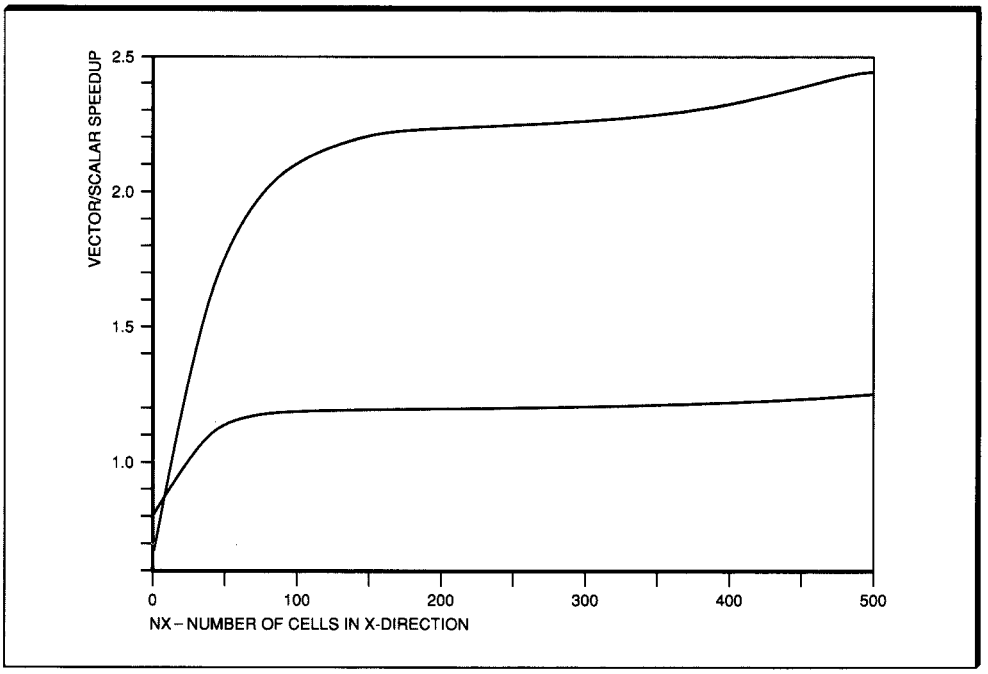
- Removal of external subroutine calls in DO-loops by generating in-line code (eventually by use of the precompiler VAST-2)
- Removal of recurrences by splitting DO-loops and eventually modifying the source code
- Resort to the IBM Engineering and Scientific Subroutine Library[12] (ESSL), which consists of currently 288 optimized numerical subroutines
- Avoidance of multiple computation of quantities sent to memory, and further improvement of data flow

Let us describe this with a relatively simple example:

```
      W = 0.0
      DO 50 J = NY,1, - 1
      DO 50 I = 1,NX
        ...
        W = DMAX1 (DABS(U-A), DABS(U+A), W)
        ...
50 CONTINUE
```

As the vector compiler reports, these loops are not in vector form, due to recurrence. This draw-

Figure 3 PICUS-200 basic version (lower graph) vs PICUS-230 optimized version (upper graph): vector-to-scalar speedup vs number of cells in x-direction



back is removed as follows: we split the inner loop and resort to the subroutine *IDAMAX* of the ESSL library,[12] which serves to find the position of the first occurrence of a vector element that has maximum modulus (the IBM ESSL subroutines are assembler-written and optimized for the IBM ES/3090 VF). For this, two auxiliary arrays have been employed in a preprocessing step. Consequently, the tuned version looks as follows, where the innermost loop is now in vector form.

```
      W = 0.0
      DO 50 J = NY,1, - 1
      DO 90 I = 1,NX
         ...
       UMA(I) = U-A
       UPA(I) = U+A
         ...
90    CONTINUE
      IMA = IDAMAX (NX, UMA, 1)
      IPA = IDAMAX (NX, UPA, 1)
      W = DMAX1 (DABS(UMA(IMA)),
                              DABS(UPA(IPA)), W)
50    CONTINUE
```

The simple modifications lead to the final vector form, called PICUS-230, with a CPU time reduced by factor 2.65 as compared with the original version PICUS-200 (both run in vector mode, with NX2 = 500, see Figure 2). By efficient use of the Vector Facility, a saving of more than 60 percent of the overall CPU time has been achieved as compared with the basic version (in vector mode). As compared with the basic version run in scalar mode, even a saving of about 70 percent results. See Figure 2.

The final version PICUS-230 enjoys a vector-to-scalar speedup of about 2.5 (see Figure 3). This shows good cost efficiency of the VF, particularly if the moderate effort is taken into account.

On parallelization

Whereas vectorization is now considered as a well-established tool to essentially reduce the overall CPU time on supercomputers, the situation is less clear today with multitasking (parallelization) on shared-memory systems. Here the

elapsed time only is reduced (the overall time a job is running on the machine), whereas the total CPU time generally is even enlarged, due to software overhead, communication and synchronization costs (a feature not welcome to computing centers). Consequently, when it comes to optimizing an application code, it is quite often advantageous to perform vectorization first, and afterwards to search for means of parallelization.

For the IBM ES/3090 series (or its successor, the IBM ES/9000* series), Parallel FORTRAN is implemented as VS FORTRAN Version 2 Release 5, enhancing the previous Multitasking Facility with a set of parallel functions ranging from automatic parallelization of DO-loops (implicit parallelism) to explicit parallel language constructs.[13] Concerning automatically generated parallel code for DO-loops, the compiler is responsible for a loop running smoothly in parallel, computationally equivalent to the serial code. On the other hand, concerning use of explicit parallel language constructs as PARALLEL DO, PARALLEL SECTIONS, PARALLEL CALL, or parallel subroutine scheduling, the user is responsible for proper processing.

As stated above, the code PICUS is well fitted both for vectorization and parallelization, due to its data structure and flow. Generally speaking, the following strategy has been employed to optimize the code with little effort but considerable speedup:

- Treat the subdomains in parallel (coarse-grained parallelism)
- Perform vectorization in x-direction (stride 1), but perform parallelization in y-direction

The numerical results have been obtained by use of two dedicated processors of the IBM ES/3090-30E at the IBM Heidelberg Scientific Center. We investigated the parallel performance of two versions:

- Version PICUS-230, optimized for vectorization (see above), using automatically-generated parallel DO-loops
- Version PICUS-240, using PREFER PARALLEL (or PARALLEL DO) in several cases where the compiler is unable to detect inherent parallelism, and enhanced by parallel processing of the subdomains using coarse-grained PARALLEL SECTIONS

Let us shortly consider how the PARALLEL SECTIONS have been introduced to PICUS. The time stepping loop of the main program has this structure:

```
    DO 80 ITIME = 1,IE
      ... (updating parameters for G1)
     CALL ADVANCEX (STATE1, ...)
     CALL ADVANCEY (STATE1, ...)
      ... (updating parameters for G2)
     CALL ADVANCEX (STATE2, ...)
     CALL ADVANCEY (STATE2, ...)
80 CONTINUE
```

ADVANCEX and ADVANCEY are those computationally expensive subroutines that serve to update the flow quantities. After checking computational and data independence of those sections dealing with G_1 and G_2, respectively, we introduced explicit parallelism by use of PARALLEL CALL or PARALLEL SECTIONS. The first way is more complicated, requiring an auxiliary subroutine, so for our final version PICUS-240 we preferred to use PARALLEL SECTIONS:

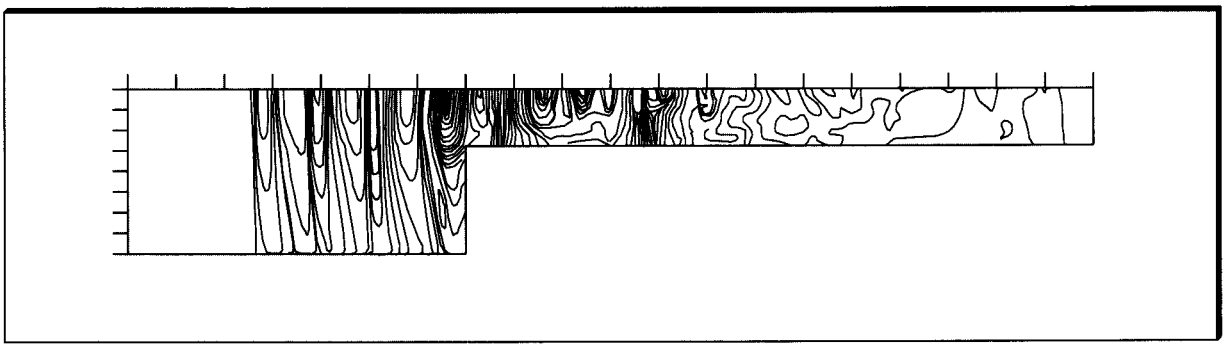
```
    DO 80 ITIME = 1,IE
     PARALLEL SECTIONS
     LOCAL ...
     SECTION 1
       ... (updating parameters for G1)
      CALL ADVANCEX (STATE1, ...)
      CALL ADVANCEY (STATE1, ...)
     SECTION 2
       ... (updating parameters for G2)
      CALL ADVANCEX (STATE2, ...)
      CALL ADVANCEY (STATE2, ...)
     END SECTIONS
80 CONTINUE
```

Here a lot of steering parameters for the data flow in G_1 or G_2 have to be assigned to each parallel thread as private variables by means of the LOCAL specification.

The following numerical results have been obtained in case of the parameters $NX1 = 511$, $NY2 = 27$, $NX1 = 1023$, $NY2 = NY1$. Here 10 time steps have been carried out, and all runs used vector mode. We observed a parallel speedup S_2 (employing two dedicated processors) as follows: for PICUS-230 we have $S_2 = 1.24$, but for the explicitly parallelized version PICUS-240 we obtained

$$S_2 = 1.75$$

Figure 4 Pseudo shock waves leaving the gap



As a result, by Amdahl's law, we obtain an activated degree of parallelism of

$p = 0.86$

for the latter version.

Physical example

The physical character of the analysis is shown by the following example.

Let two shock waves successively enter the channel, in a way that the second wave enters in just that moment where the first one is reflected at the end wall of the channel. Both waves are then interacting, and they produce (in addition to a shock wave turning again toward the end of the channel) a wave train, or so-called *pseudo shock waves* (which are due to massive viscous-inviscid interaction between the reflected shock wave and its turbulent boundary layer). Rigorous quantitative investigation of this kind of phenomenon has been commenced only recently.[2] Figure 4 shows the isopycnics of multiple pseudo shock waves shedding from the gap into the inlet combustion chamber, a phenomenon particularly observed for small gap width of the order of 20 to 30 μm. Concerning influence of exothermic chemical reactions, our numerical results show a reduction of shock-wave-boundary-layer interaction due to local separation and vortex formation that substantiates a damping effect of heat release in nonequilibrium flow.

Typical values of dynamic viscosity μ for hydrocarbon air mixtures are chosen. Thermal conductivity is calculated from dynamic viscosity employing the Eucken relation and Curtiss-Hirschfelder gas kinetics relations. The minimum propagation Mach number Ma of the shock wave entering G_2 coming from G_1 is taken as about 1.3 in correspondence with typical measurements of pressure increase due to knock events in engines.

This computation consumed about 20 hours of CPU time on the IBM ES/3090.

Conclusions

We described the algorithmic properties of a new FORTRAN code PICUS to simulate shock waves in reacting Navier Stokes flows. The algorithm is based on VanLeer's second-order shock-capturing scheme (using Roe's approximate Riemann solver), combined with a semi-implicit ODE solver for the nonequilibrium chemistry part. On the IBM ES/3090 Vector Multiprocessor, the new code (validated now by experiment) presents a vector-to-scalar speedup of about 2.5 as well as a supplementary parallel speedup of about 1.75 by use of two parallel processors. The long-time behavior of PICUS has been checked for an important physical detail, namely the formation of multiple pseudo shock waves in an internal combustion engine.

Acknowledgment

The authors would like to express sincere thanks to R. Janssen for helpful discussions and valuable support of the present work.

Cited references

1. E. S. Oran and J. P. Boris, *Numerical Simulation of Reactive Flow*, Elsevier, New York (1987).
2. H. Sugiyama, H. Takeda, Z. Zhang, and F. Abe, "Multiple Shock Wave and Turbulent Boundary Layer Interaction in a Rectangular Duct," *Shock Tubes and Waves*, H. Groenig, Editor, *Proceedings of the 16th International Symposium at Aachen 1987*, VCH, Weinheim, Germany (1988), pp. 185–191.
3. B. VanLeer, "Towards the Ultimate Conservative Difference Scheme (Part 5)," *Journal of Computational Physics* **32**, 101–136 (1979).
4. P. L. Roe, "Characteristic-based Schemes for the Euler Equations," *Annual Review of Fluid Mechanics* **18**, 337–365 (1986).
5. M. Fey, H. Jarausch, R. Jeltsch, and P. Karmann, "On the Interaction of Euler and ODE Solver When Computing Reactive Flow," *Adaptive Methods for PDEs*, J. E. Flaherty, P. J. Paslow, M. S. Shephard, and J. D. Vasilakis, Editors, SIAM (Society for Industrial and Applied Mathematics), Philadelphia (1989), pp. 29–42.
6. R. J. Gathmann, F. K. Hebeker, and S. Schoeffel, "On the Numerical Simulation of Shock Waves in an Annular Crevice and Its Implementation on the IBM ES/3090 with Vector Facility," *Application of Supercomputers in Engineering II*, C. A. Brebbia, D. Howard, and A. Peters, Editors, Elsevier Applied Science, London (1991), pp. 319–330.
7. H. C. Yee, *A Class of High-Resolution Explicit and Implicit Shock-Capturing Methods*, Von Kármán Institute, Brussels, Lecture Series 1989-04 (1989).
8. E. H. Hirschel and A. Groh, "Wall-Compatibility Conditions for the Solution of the Navier Stokes Equations," *Journal of Computational Physics* **53**, 346–350 (1984).
9. R. W. MacCormack, "A Rapid Solver for Hyperbolic Systems of Equations," *Lecture Notes in Physics* **59**, 307–317 (1978).
10. J. J. Kloeker, "Shock Induced Self-Ignition of a Reactive Gas Mixture in a L-Shaped Duct," *Numerical Combustion, Proceedings of the 3rd International Conference at Antibes*, France (1989).
11. W. Gentzsch and S. Glückert, "The Processor IBM 3090 with Vector Facility" (in German), *Praxis der Informationsverarbeitung* **10**, 24–30 (1987).
12. *IBM Engineering and Scientific Subroutine Library: Guide and Reference,* SC23-0184, IBM Corporation (1990); available through IBM branch offices.
13. *IBM VS FORTRAN Version 2: Programming Guide for CMS and MVS,* SC26-4222, IBM Corporation (1991); available through IBM branch offices.

Accepted for publication August 10, 1992.

Friedrich K. Hebeker *IBM Germany, Heidelberg Scientific Center, P.O. Box 10 30 68, D-6900 Heidelberg, Germany.* Dr. Hebeker is a research staff member of the Heidelberg Scientific Center. He received a Dr. and a DSc. degree in applied mathematics from Paderborn University in 1980 and 1985, respectively. In 1988 he joined IBM Germany and has worked and published in the areas of parallel algorithms, computational fluid dynamics, and boundary element methods. He is involved in joint research projects with engineering departments of academia and industry. In addition, Dr. Hebeker is a Privat-Dozent of Mathematics at the Technical University of Darmstadt.

Rudolf R. Maly *Daimler-Benz Research Laboratories, P.O. Box 80 02 30, D-7000 Stuttgart 80, Germany.* Dr. Maly received his Dr.-Ing degree in physical electronics in 1974 from the University of Stuttgart. He spent 16 years at the Institute for Physical Electronics at the University of Stuttgart as a research scientist, assistant professor, and head of the combustion group. For the past nine years Dr. Maly has worked at the Daimler-Benz Research Laboratories. His position is head of the competence center for thermo- and aerodynamics. Dr. Maly's research interests include spark and compression ignition, physics and chemistry of combustion, engine knock, and combustion diagnostics.

Stefan U. Schoeffel *Daimler-Benz Research Laboratories, P.O. Box 80 02 30, D-7000 Stuttgart 80, Germany.* Dr. Schoeffel is a research staff member of the Mercedes-Benz Research Institute of Daimler-Benz. He received a Dr.-Ing. degree in mechanical engineering from Kaiserslautern University in 1987. Gas dynamics, reactive flow, light-water reactor safety, and environmental technology have been his topics of interest. In 1989 he joined Daimler-Benz and has been working on advanced turbulence modeling for internal combustion engines since then. From 1989 to 1992 Dr. Schoeffel was visiting scientist at the Heidelberg Scientific Center of IBM Germany.

Reprint Order No. G321-5498.