AIAA JOURNAL
Vol. 35, No. 3, March 1997

Load Balancing and Performance Issues for Data Parallel Simulation of Stiff Chemical Nonequilibrium Flows

James W. Weber Jr.*
U.S. Air Force Wright Laboratory, Wright–Patterson Air Force Base, Ohio 45433
Elaine S. Oran[†]
U.S. Naval Research Laboratory, Washington, D.C. 20375
Gopal Patnaik[‡]
Berkeley Research Associates, Springfield, Virginia 22150
and
John D. Anderson Jr.[§]
University of Maryland, College Park, Maryland 20742

A data parallel program is presented that solves the reactive Euler equations for stiff chemical nonequilibrium flows on Connection Machines CM-2/200 and CM-5/5E. The program is written in CM Fortran and uses direction and time-step splitting to couple representations of the chemical and fluid dynamic processes on a structured Cartesian grid. An explicit high-order monotone algorithm with nonlinear damping is used to integrate the convection terms, and a hybrid asymptotic/modified-Euler approach is used to solve the system of ordinary differential equations from the chemical source terms. Integration of the fluid dynamics was conservatively determined to be 9.4 and 12.0 Gflops on a 512-node CM-5 and CM-5E, respectively. The fluid dynamics solver scaled well for large problems; however, both the performance and the scaling are significantly affected by the nearest-neighbor communications, which accounted for at least 24% of the execution time on the CM-5. For the integration of the chemistry source terms, poor load balancing significantly affected performance of the program. Therefore, a new load-balancing algorithm was developed that reduces the chemistry integration time by a factor of six for the test problem, a detonation propagating in a hydrogen–oxygen–argon mixture. Moreover, the chemistry integration time, with the load balancing, is slightly less than the time required to integrate the fluid dynamics. As a result, an efficient data parallel program for solving stiff chemical nonequilibrium flows is available for problems that were too expensive to solve in the past.

Nomenclature

a_b, b_b, d_b	= backward rate constant coefficients
a_f, b_f, d_f	= forward rate constant coefficients
D_1, D_2, D_3	= generalized source terms
e	= specific internal energy
e_t	= specific total energy
$\boldsymbol{F}$, $\boldsymbol{G}$	= flux vectors
k_f, k_b	= forward and backward rate constants
$\mathcal{L}$	= numerical solution operator
N_r	= number of elementary reactions
N_s	= number of chemical species
n_i	= number density of species i
p	= pressure
$\boldsymbol{Q}$	= solution vector
R	= specific gas constant
r	= generalized spatial coordinate
$\boldsymbol{S}$	= source vector
T	= temperature
t	= time
u, v	= Cartesian velocity components
$\dot{w}_i$	= production rate of species i
x, y	= Cartesian coordinates
Δt_{cfl}	= fluid dynamic time step
Δt_{chem}	= chemistry time step
ν_i', ν_i''	= reactant and product mole number of species i
ϕ	= equivalence ratio
ρ	= density

Subscript

i	= chemical species index

Superscripts

k	= subcycle index
n	= time-step index

Introduction

A WIDE range of important aerospace problems use computational fluid dynamics (CFD) approaches that require high-performance computer hardware and software. Such applications range from descriptions of subsonic, low-Reynolds-number flight problems to hypersonic vehicle re-entry, and to the design of engines for all types of aircraft and spacecraft. Many of these problems require the solution of nonequilibrium chemically reacting flows, which may involve integrating many coupled nonlinear ordinary differential equations (ODEs), which themselves are coupled to the solution for the fluid dynamics. As the number of reacting species increases, finding a solution economically becomes more difficult.

Large improvements in the performance of computer hardware now come from advances in scalable parallel computers that simultaneously execute different parts of a program on multiple processors. A parallel computer can be classified on the basis of how memory is accessed. One type of machine uses shared-memory parallelism, in which each processor sees all of the memory directly. The other type is distributed-memory parallelism, in which each processor has a certain amount of memory, and the processors can

Presented as Paper 95-0570 at the Aerospace Sciences Meeting, Reno, NV, Jan. 9–12, 1995; received April 21, 1995; revision received Nov. 22, 1996; accepted for publication Nov. 25, 1996; also published in *AIAA Journal on Disc*, Volume 2, Number 2. This paper is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

*Aerospace Engineer, Propulsion Systems Branch. Member AIAA.
[†]Senior Scientist. Fellow AIAA.
[‡]Staff Scientist, P.O. Box 852. Member AIAA.
[§]Professor, Department of Aerospace Engineering. Fellow AIAA.

interchange data through special information-passing routines. As an example, the Cray C-90 is a shared-memory machine, as is one tower of an SGI Power Challenger. Distributed-memory machines usually are designed to work in one of two modes: the data parallel approach and the task parallel (or control parallel) approach. In the data parallel paradigm, the same mathematical operations are performed in each set of computer processors, although the operations may be performed on different values. In the task parallel paradigm, different operations may be performed on the same set of data. The Connection Machines (CM) generally are used as data parallel machines. Although the CM-5 has, in principle, the ability to work in task parallel mode, that has not proved to be the most efficient use of this machine. In fact, most distributed-memory computers can be run in data parallel mode. From one point of view, data parallel can be considered a subset of task parallel.

To date, a number of CFD programs have been developed for solving chemical nonequilibrium flows on parallel computers. Many of these programs are task parallel programs that have been developed for parallel computers by adding domain decomposition routines and message-passing constructs to preexisting code. For example, Otto[1,2] performed some of the first reacting flow calculations on a parallel computer by porting the family of SPARK codes to the Intel iPSC/860. Recently, Patnaik et al.[3] developed task parallel Fortran codes for simulating the detailed structure of laminar flames and tested these on parallel computers. There also have been efforts to develop chemically reacting data parallel codes that use a high-level parallel programming language such as CM Fortran, Fortran 90, or High-Performance Fortran (HPF). Candler et al.[4] presented a data parallel program in Fortran 90 for solving inviscid, chemical, and vibrational nonequilibrium flows on the Thinking Machines, CM-5, and the MasPar MP-1/2. These authors achieved code performances that are near the theoretical peak performance for large problems.

In this paper, we describe a data parallel program, written in CM Fortran, that solves the equations for inviscid, nonequilibrium chemically reacting flows on the CM computers.[5] The basic methods, Flux-Corrected Transport (FCT) for the coupled continuity equations[6,7] and Selected Asymptotic Integration Method (SAIM) or CHEMEQ for the ODEs,[8,9] are coupled by direction and time-step splitting. The relevance of this work goes significantly beyond the optimization of a particular code on a particular computer. Rather, it is an example of how the use of the data parallel paradigm with the proper load-balancing algorithms can produce a highly optimized code. In fact, many of the specific improvements in algorithms and performance reported here are generalizable to the task parallel paradigm. Finally, we believe that, in the near future, HPF will be able to use this type of code directly as is, or with minor modifications. HPF is currently available to some extent on all distributed-memory machines and some shared-memory machines.

The increase in performance achieved through load balancing (described below) has allowed us to perform various types of multidimensional reacting flow computations. At the end of this paper, we describe selected features of a computation of a propagating detonation. These computations use a detailed chemical kinetics model for hydrogen combustion. Other applications, reported elsewhere, extended this code to include the viscous terms of the Navier–Stokes equations[10] and applied the result to computations of the interactions of shocks with boundary layers,[11] viscous, nonreacting[12] and reacting shear flows,[13] interactions of vortices with boundary layers,[14] and counterflow diffusion flames.[15]

Governing Equations

The governing equations considered in this study are the two-dimensional Euler equations for a chemically reacting gas mixture, which are a coupled system of nonlinear hyperbolic conservation equations with source terms. The equations are expressed in a Cartesian coordinate system and conservation form to improve capturing of shock waves and discontinuities. The resulting system of equations, in the vector form, is

$$\frac{\partial \boldsymbol{Q}}{\partial t} + \frac{\partial \boldsymbol{F}}{\partial x} + \frac{\partial \boldsymbol{G}}{\partial y} = \boldsymbol{S} \tag{1}$$

where

$$\boldsymbol{Q} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho e_t \\ n_i \end{pmatrix}, \qquad \boldsymbol{F} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ \rho u e_t + p u \\ n_i u \end{pmatrix}, \qquad \boldsymbol{G} = \begin{pmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ \rho v e_t + p v \\ n_i v \end{pmatrix} \tag{2}$$

$$\boldsymbol{S} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \dot{w}_i \end{pmatrix}, \qquad \text{for} \qquad i = 1, \ldots, N_s$$

To close this system of equations, the total internal energy is defined as $e_t = e + (u^2 + v^2)/2$, and the auxiliary equations between the dependent variables are given by the thermal equation of state for a perfect gas, $p = \rho RT$, and a caloric equation of state for a reacting mixture, $e = e(n_i, T)$. In the present study an 8-species, 48 reaction H_2–O_2 mechanism is used.[16] The reacting species in this kinetic mechanism include H, O, H_2, OH, H_2O, O_2, HO_2, and H_2O_2. For the test case, a significant amount of argon diluent was added to the gas. The production rate of each chemical species $\dot{w}_i$ is given by combining the elementary chemical reactions in the kinetic model so that

$$\dot{w}_i = \sum_{r}^{N_r} (\nu''_i - \nu'_i)_r \left\{ k_f \prod_{i}^{N_s} n_i^{\nu'_i} - k_b \prod_{i}^{N_s} n_i^{\nu''_i} \right\}_r \tag{3}$$

where

$$k_f = a_f T^{b_f} \exp(d_f / T) \tag{4}$$

and

$$k_b = a_b T^{b_b} \exp(d_b / T) \tag{5}$$

are the forward and backward rate constants.

Numerical Method

Equation 1 is solved by a time- and direction-splitting approach in which $\boldsymbol{Q}^{n+1}$ is determined from $\boldsymbol{Q}^n$ according to the operator function

$$\boldsymbol{Q}^{n+1} = \mathcal{L}_s \mathcal{L}_y \mathcal{L}_x \boldsymbol{Q}^n \tag{6}$$

where $\mathcal{L}_x \boldsymbol{Q}^n$ is the numerical solution operator for the equation

$$\frac{\partial \boldsymbol{Q}}{\partial t} + \frac{\partial \boldsymbol{F}}{\partial x} = 0 \tag{7}$$

and $h = \Delta t$ is the time step for stable integration of the fluid dynamics. The time-step–splitting algorithm solves Eq. (1) in three steps: steps one and two alternate between integration of Eq. (7) and the y-direction convection terms

$$\frac{\partial \boldsymbol{Q}}{\partial t} + \frac{\partial \boldsymbol{G}}{\partial y} = 0 \tag{8}$$

and the third step integrates the system of ODEs that results from splitting the source vector

$$\frac{\mathrm{d} n_i}{\mathrm{d} t} = \dot{w}_i \qquad \text{for} \qquad i = 1, \ldots, N_s \tag{9}$$

The splitting method allows separate solution of the homogeneous terms and the chemical source terms. As a result, the fluid dynamic terms can be integrated at the Courant–Friedrichs–Lewy (CFL) time step Δt_{cfl} for explicit, time-accurate solutions, rather than the characteristic time step of the chemical reactions. A splitting method therefore provides a more efficient solution technique for solving unsteady chemically reacting flows with fast chemical reactions. In addition, there are several advantages to the splitting approach. Oran and Boris[17] advocate this method for developing modular code because it facilitates changing the algorithm that is applied to the fluid dynamics or chemical reactions. Yee[18] indicates that the prospect of developing a stable method for integrating hyperbolic conservation equations with source terms is better for splitting methods than fully coupled (global implicit) methods that integrate the homogeneous and source terms simultaneously. LeVeque and Yee[19] have found that splitting methods perform somewhat better than fully coupled methods for solving scalar hyperbolic conservation equations with stiff source terms.

The integration of the one-dimensional Euler equations (7) and (8) is performed using the LCPFCT algorithm,[20] which is a recent version of the one-dimensional FCT algorithm developed by Boris and Book.[6] LCPFCT is an explicit finite volume algorithm that solves one-dimensional continuity equations of the form

$$\frac{\partial \rho}{\partial t} = -\frac{1}{r^{\alpha-1}}\frac{\partial (r^{\alpha-1}\rho u)}{\partial r} - \frac{1}{r^{\alpha-1}}\frac{\partial (r^{\alpha-1} D_1)}{\partial r} + C_2\frac{\partial D_2}{\partial r} + D_3 \qquad (10)$$

where the value of α defines the reference coordinate system that is used: Cartesian ($\alpha = 1$), cylindrical ($\alpha = 2$), or spherical ($\alpha = 3$). LCPFCT employs nonlinear damping and fourth-order phase accuracy to capture sharp gradients in the flow without generating nonphysical oscillations and excessive numerical dissipation. For unsteady problems, second-order accuracy in time is obtained by integrating the system of equations with a two-stage Runge–Kutta method, in which the time step for stable integration is governed by the CFL criterion. For FCT algorithms, a CFL number less than 0.5 typically is required for stable integration. Methods exist that are stable and allow double the time step,[21] but for highly exothermic reacting flows and flows with steep shocks, it is usually necessary to decrease the global time step anyway.

The solution of Eq. (9), for the chemical species number densities, is obtained using the algorithm SAIM.[9] SAIM is a second-order accurate algorithm for solving stiff systems of ODEs associated with finite-rate chemical reactions. The SAIM algorithm employs an explicit single-step approach that solves the stiff equations asymptotically and the normal (nonstiff) equations with a modified Euler step. To obtain stable integration of Eq. (9), SAIM uses an internal time step Δt_{chem} that is based on the convergence properties of the ODE solution. If the system of equations is stiff, then the resulting value of Δt_{chem} will be smaller than the integration time interval of Δt_{cfl}. Consequently, the SAIM algorithm subcycles until the accumulated integration time equals the fluid dynamic time step.

Young and Boris[9] maintain that second-order integration of Eq. (9) is sufficient for chemical nonequilibrium flows because the fluid dynamics are only calculated to 2 or 3% accuracy, and the uncertainty of the chemical rate constants generally is accepted to be 10% or higher. Moreover, second-order methods are preferred because higher-order methods use explicit multistep or implicit time-marching algorithms, which have large data storage and matrix inversion requirements. Young[8] has noted that results from the SAIM algorithm agree to two or three significant figures with results obtained from higher-order methods.

Issues in Data Parallel Code Development

In general, the data parallel paradigm is well suited for reacting flow computations because many of the same mathematical operations in CFD are repeated at each computational cell. However, there are two major factors that can significantly influence the performance of such a parallel program. One is the cost of interprocessor communication. On this issue, the code developer is often at the mercy of the computer architecture. Another factor is load balancing, which is achieved when all of the processors are kept doing useful computations for as much of the time as possible. Load balancing problems are endemic to any parallel paradigm, although the exact form they take and the solution found can differ somewhat depending on the particular type of computer and problem.

First, consider interprocessor communications. On a distributed-memory computer, the data for each cell are stored on the processor allocated to that cell. Operations that require data from neighboring cells, which are allocated to other processors, require message-passing calls. In CFD applications, difference equations often are used to construct fluxes at cell interfaces or to approximate spatial derivatives, and so, interprocessor communications can become a bottleneck to good performance. Therefore, it is important when developing a data parallel CFD code to minimize the number of operations that result in interprocessor communications. How this is achieved in the data parallel paradigm is discussed in the next section.

The second factor is load balancing. To obtain the maximum performance from an application on a parallel computer, the workload must be distributed so that each processor always is employed in a useful calculation. If a parallel application is using only half of its allocated processors, the performance of the program is reduced significantly. The worst-case scenario occurs when one processor is performing all of the useful work, and the remaining processors are either idle or performing useless computations. In this situation, the performance of the application on a parallel computer is worse than the performance on a single-processor workstation. One example of a process that requires some effort with respect to load balancing is the implementation of boundary conditions, which require different operations than those required in the computational domain. Another is the integration of the coupled, nonlinear ODEs describing the chemical reactions at each computational cell. In some cells, these equations may be very stiff and require many interactions compared to other cells, where they may be integrated in one or two steps.

The data parallel program described below consists of CM Fortran subroutines developed from a series of Fortran 77 subroutines. The algorithms and programming in the Fortran 77 subroutines represent many years of development. They have been documented, tested, and applied to a wide variety of problems. The computational work in the code is done primarily in two subroutines: CONVECT, which is the CM Fortran version of LCPFCT[20]; and CHEMEQ, which uses SAIM[9] to integrate the ODEs describing the chemical reactions. The new CM Fortran program was tested by comparing the solutions to a series of standard benchmark problems. Detailed comparisons of the results of these test problems, and their comparison to the Fortran 77 code results, were presented previously.[5,22] CONVECT reproduced the computations described in the LCPFCT report to within machine accuracy in a series of standard one- and two-dimensional problems.[22] CHEMEQ alone was tested against results obtained previously for integrating the hydrogen combustion mechanism,[16] and again, the results were the same to within machine accuracy. The complete simulation was compared to previous Cray computations of earlier versions of the code. At all times, the code was kept extremely modular and general, even at the expense of optimizations for specific problems.

Data Parallel Performance Results

Performance profiles of the data parallel program show that the majority of execution time is spent by subroutines CONVECT and CHEMEQ performing the numerical integration of Eqs. (7–9). Efforts to reduce the total run time of the program therefore were aimed at optimizing these two subroutines. Because the algorithms of CONVECT and CHEMEQ, which are based on the LCPFCT and SAIM algorithms, respectively, are very different in form, they required different optimization strategies.

CONVECT had two types of problems. The first was the need for load balancing to efficiently evaluate the boundary conditions. This problem, which has been dealt with previously and is well documented, is summarized below for completeness. CONVECT performs the same number of floating-point operations at each cell. However, the evaluation of cell interface quantities and flux differences generates numerous interprocessor communication calls. In CM Fortran, nearest-neighbor communications are accomplished with the CSHIFT function. Hence, the strategy for reducing the execution time of CONVECT was to minimize the use of the CSHIFT statement.

In contrast, CHEMEQ generates no interprocessor communication calls; however, the use of local time steps by the SAIM algorithm at each cell causes the number of floating-point operations to vary across the grid. Consequently, the amount of useful work performed on each processor is different, so that poor load balancing reduces the performance of the program. The optimization of CHEMEQ addressed the workload distribution for the chemistry integration.

Performance of Subroutine Convect

The calculation of boundary conditions can have a severe impact on the performance of a data parallel program. A serial approach to calculating the boundary conditions can increase the total computation time by more than 50% (Refs. 22 and 23). Several approaches to load balancing of boundary conditions have been tested for the CMs. In one approach, the computation is stopped, boundary-condition computations are scattered to the entire computer, the computations are done, and then they are gathered and distributed appropriately. In another approach, the boundary computations are done on the entire domain and only retained at the boundary locations.

The idea underpinning the boundary-condition algorithm used here is to build the calculation of the boundary conditions into the computations of the flow. Then, at each time step, both boundary and internal conditions may be evaluated for any point in the computational domain from computed fluxes and appropriate multiplying coefficients. This approach also allows simultaneous calculation of different conditions at different locations both in and on the boundaries of the domain. Weber et al.[22] evaluated both this integrated approach, called the Uniform Boundary Algorithm (UBA), and another approach in which the boundary conditions are computed separately, and then compared the timing and the results of a number of computations. Using the UBA, the total increase in computational time was 10% or less.

The floating-point performance of CONVECT was measured on the CM-2, CM-5, and CM-5E to determine the program's performance relative to the theoretical peak speed of the CMs. Because performance software was not available on the CM, the performance of CONVECT was determined by counting the number of floating-point operations and dividing by the CM busy time. The CM busy time is the total time that the code spends on the CM processing nodes, and is less than the CM elapsed time, which includes the CM busy time, the time spent performing scalar operations, and the time that the program is swapped out by the time-sharing daemon. Since time-sharing can have a significant impact on the CM elapsed time (which was unrepeatable), the performance values were based on the more consistent CM busy times. For the present evaluation, a conservative estimate of the floating-point performance was made: All arithmetic operations (add, subtract, multiply, and divide) were counted as one floating-point operation; no weight was given to communications calls (for example, CSHIFT); and no transcendental functions were evaluated in CONVECT.

Figure 1 shows the floating-point performance of CONVECT on the CM as a function of the computation size N_{cells}. As the computation size increases, the performance of CONVECT also improves. This is due to the fact that the efficiency of CSHIFT, the nearest-neighbor communications function, improves as the ratio of N_{cells} to the number of processor nodes N_{nodes} increases. Consequently, the maximum performance of CONVECT on each CM model or partition is obtained for the largest problem that can fit on the configuration.

The performance of CONVECT on a 16K CM-2 is shown in Fig. 1 to range from 252 Mflops, on a 16K grid (K = 1024), to 421 Mflops on a 256K grid, for 32-bit calculations. For 64-bit calculations on the CM-2, these results translate to a maximum performance of 210 Mflops, since 64-bit calculations generally require twice as much time as 32-bit calculations on the CM-2. This performance level is typical of the performance of CFD programs achieved in practice on conventional, vector-processor supercomputers. In addition, the peak performance of 421 Mflops compares quite well with a maximum performance of 650 Mflops that was obtained for a number of benchmark programs that were tested on the CM-2 by the system managers. Yet 421 Mflops is only 12% of the CM-2's theoretical peak performance.

Figure 1 also shows the performance of CONVECT on a 64-, 256-, and 512-node CM-5. In nearly all of the cases considered, the performance of CONVECT exceeds 1.0 Gflop, and, as expected, improves with the size of the partition. The timings on the CM-5 are reported for a range of computation sizes, from 128K cells to 8M cells (M = 1024^2), and the highest performance is given as 9.4 Gflops, which was obtained on a 512-node CM-5 with an 8M grid. The maximum performance corresponds to 14.3% of the CM-5's theoretical peak speed of 65.5 Gflops and is typical of the relative performances that have been reported for other parallel CFD programs.

A comparison of the performance of CONVECT on a 256-node CM-5 and CM-5E also is shown in Fig. 1. The results indicate that the maximum performance achieved by CONVECT on a 256-node CM-5 is 4.7 Gflops, whereas the corresponding performance on the CM-5E, for the same value of N_{cells}, is 6.0 Gflops. For all of the grid sizes that were tested on the CM-5 and CM-5E, the performance of the CM-5E is greater, as expected. However, Fig. 1 shows that the difference in performance between the two computers decreases as the computation size increases. A comparison of the performance of CONVECT on a 512-node CM-5E also can be made with a 512-node CM-5, since the performance of CONVECT is found to scale ideally with the number of nodes. When the maximum performance of CONVECT on a 256-node CM-5E is scaled to 512 nodes, a performance of 12.0 Gflops is obtained, which is a 28% improvement over the 9.4 Gflops obtained on the CM-5. On the CM-5E, a higher percentage of the theoretical peak performance is achieved by CONVECT, where 12.0 Gflops corresponds to 15% of the theoretical peak speed.

To determine the importance of nearest-neighbor communications (known as NEWS communications) for the performance of CONVECT on the CM-5, Fig. 2 shows the NEWS communication ratio as a function of the ratio $N_{\text{cells}}/N_{\text{nodes}}$. As seen in Fig. 2, the NEWS communication ratio, which is the ratio of nearest-neighbor communications time to total time spent by the node on communications and CPU, is only a function of $N_{\text{cells}}/N_{\text{nodes}}$. For a 64-, 256-, and 512-node partition, the NEWS communication ratio of CONVECT is the same for a given value of $N_{\text{cells}}/N_{\text{nodes}}$. The results in Fig. 2 show that the NEWS communication ratio is more significant for smaller problems, accounting for 46% of the total node time, yet is still substantial at 24% for the largest problem considered. These results support the previous statement that interprocessor communications can serve as a bottleneck to the performance of a CFD code.

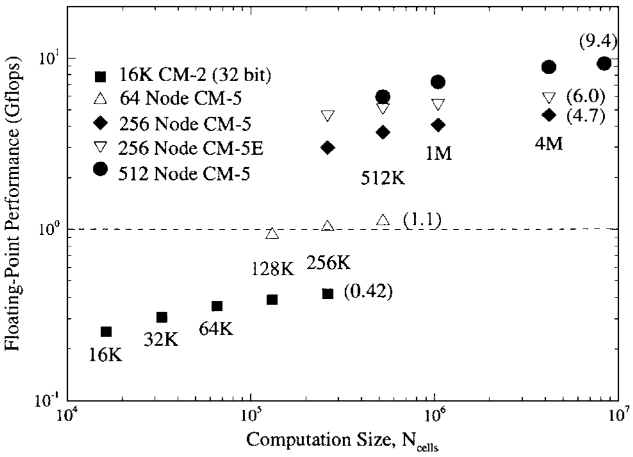

Fig. 1 Floating-point performance of subroutine CONVECT on the CM-2, CM-5, and CM-5E.

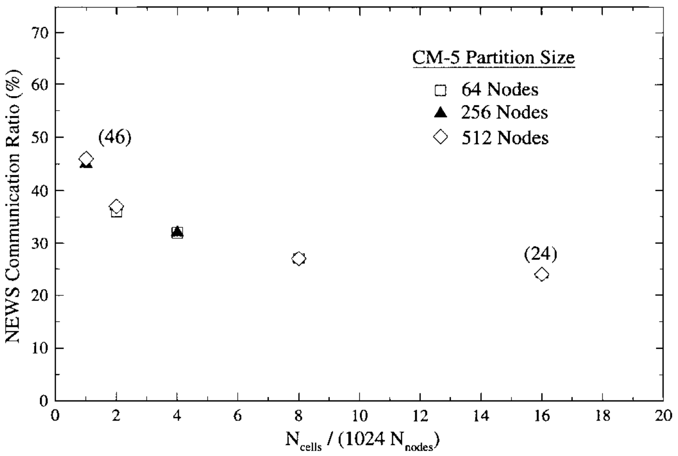

Fig. 2 Ratio of nearest-neighbor communication time to total node time for subroutine CONVECT on the CM-5.

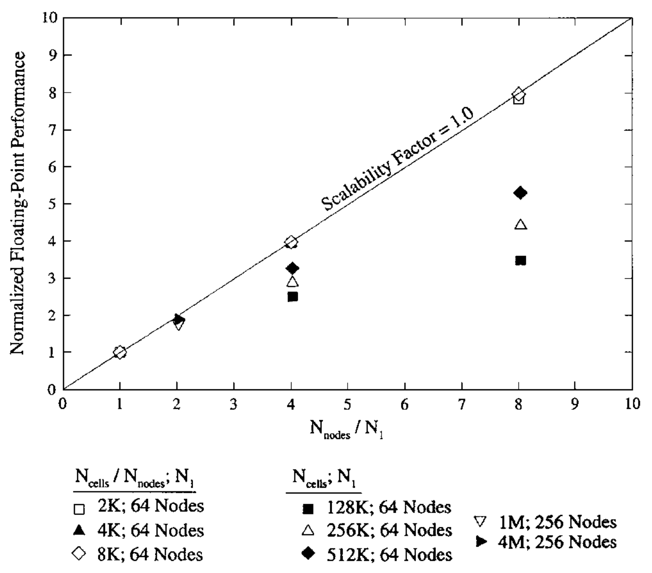

Fig. 3 Scalability of subroutine CONVECT on the CM-5 for a scaled and fixed computational size.

However, for the present program, the maximum performance obtained after the communications time is subtracted from the CM busy time is 12.4 Gflops (up from 9.4 Gflops on the 512-node CM-5), which is only 19% of the theoretical peak speed. Hence, significant improvements in the performance of CONVECT could be accomplished by further optimization and improvements in the CM-5 system software and CM Fortran compiler.

The scalability of CONVECT on a CM-5 for a scaled and fixed computation size is shown in Fig. 3. The scalability for a scaled computation size is demonstrated in the graph of the floating-point performance (normalized by the performance obtained on a partition size of N_1 nodes) as a function of the normalized partition size N_{nodes}/N_1. This shows that as the number of nodes and computational cells is proportionally increased, the floating-point performance of CONVECT increases linearly with a slope of one. Hence, subroutine CONVECT scales ideally when the computation size is scaled proportionally. The scalability for a fixed computation size also is shown. For small computation sizes (for example, 128K and 256K cells), the performance of CONVECT scales poorly with the number of processors. For instance, on a 256K grid the floating-point performance increases by a factor of 4.5 when the number of nodes is increased by a factor of 8. For larger computation sizes, a limited amount of data is available to judge the scalability of CONVECT. On a 512K grid, the performance of CONVECT increases by a factor of 5.4 when the number of nodes is increased by a factor of 8, and on a 4M grid the performance increases by a factor of 1.9 when the number of processor nodes is doubled. Hence, it appears that CONVECT scales well only for large problems if the computation size is fixed.

Although the floating-point performance of CONVECT is a small percentage of the CM's theoretical peak performance and the subroutine scales well only for large problems, Gflop performance levels can be achieved readily on the CM-5 without significant efforts to optimize the code. Further optimization of the code to take advantage of the CM-5 vector processor architecture, and improvements to the system software and CM Fortran compilers, should provide additional performance gains for CONVECT.

Some Comments on Relative Timings of CONVECT

Although it sounds straightforward to compare the relative timings of this code on different computers, such comparisons can be quite confusing. There are three factors that limit the speed of a computation: 1) the basic speed of the processor, 2) the rate at which information can be communicated among the processors, and 3) the ratio of the processor bandwidth to memory. The lag in communications between processors leads to what is called latency in processor communication. The bandwidth-to-memory ratio measures how long it takes to get the information from a processor's memory into the processor where it can be manipulated. The delay results in memory latency. All of these factors play a part in the final timings that can be obtained for a parallel computer.

With these in mind, we can give some comparisons of the CM speed to that of other computers. Figure 1 shows that we are achieving about 4.5 Gflops for 256 processors of a CM-5 for CONVECT when there are 2×10^6 computational cells in a two-dimensional simulation. Since this work was done, other work has shown that specifically optimized versions of this code run at 100 Mflops on 16 processors of an Intel iPSC/860; 400 Mflops on one processor of a C-90; about 100 Mflops/processor on an SGI Power Challenge; and about 400 Mflops on 16 processors of an IBM SP2. A complete description of the limiting factors in each of these machines is beyond the scope of this paper. However, an SGI workstation (equivalent to one Power Challenge processor) runs at about 100 Mflops, a tower may have as many as 16–18 processors, and there may be as many as 16 processors on a Cray C-90. In all of these machines, when many processors are used, there are latencies that do not exist when there is only one processors to consider. However, by combining processors, the results are impressive.

Load Balancing for Subroutine CHEMEQ

The test problem selected for studying the impact of load balancing on the data parallel calculation of chemical nonequilibrium flows is the propagation of a multidimensional detonation wave. Detonation waves are unsteady, multidimensional shock structures that are driven by chemical energy released when the shock system passes through a premixed, unreacted combustible material. Accurate simulation of a detonation wave requires spatial resolution of a large range of length scales (representing chemical induction zones and the system length scale) and temporal resolution of the fluid dynamic and chemical reaction time scales. As a result, detonation wave solutions require a large number of discretized points in both space and time.

The fast production of chemical species behind a detonation wave is limited to a small region behind the detonation (shock) structure, so that the chemical reactions associated with a detonation wave are spatially stiff. This characteristic of propagating detonations makes them ideally suited for the study of load balancing in data parallel calculations. Figure 4 illustrates the complexity of the flowfield behind a detonation wave traveling in a tube filled with a mixture of H_2:O_2:Ar/2:1:7 at $p = 6.7$ kPa and $T = 298$ K. These results, which are described in detail in other references,[5,24] were obtained with the present data parallel code after 24,000 time steps.

The distribution of the number of internal time steps by CHEMEQ was used to characterize the load balancing between the processors.

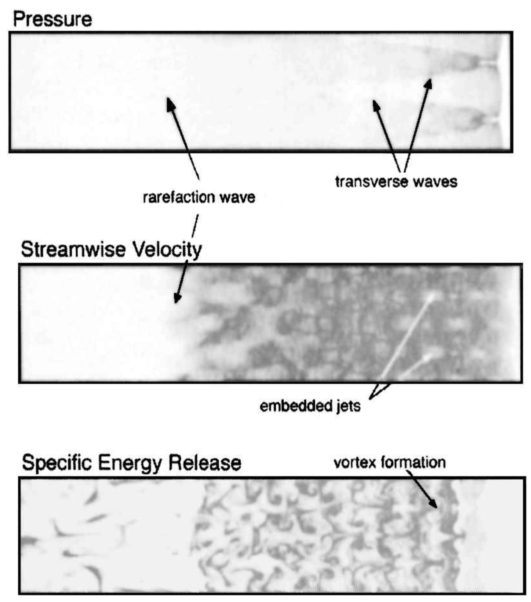

Fig. 4 Instantaneous contours of pressure, streamwise velocity, and specific energy release at step 24,000, time 415.9 μs, for a two-dimensional detonation propagating from left to right in a 6-cm channel in a H_2:O_2:Ar/2:1:7 mixture, at $p = 6.7$ kPa, 298 K. The computational grid is 2048×256 (see Table 1).

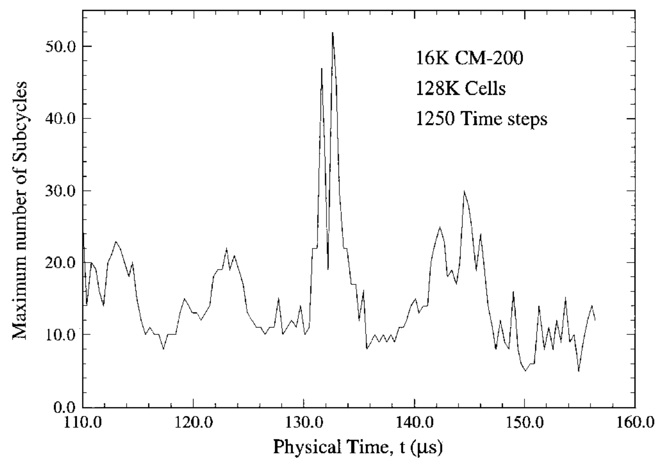

Fig. 5 Maximum number of subcycles performed by subroutine CHEMEQ for a detonation simulation.

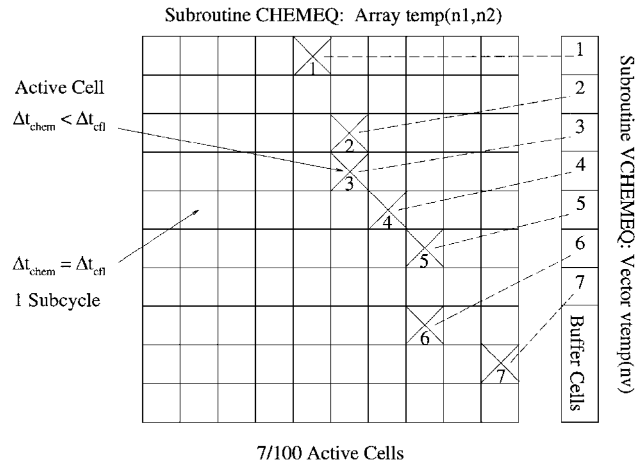
Subroutine CHEMEQ: Array temp(n1,n2)
Active Cell
$\Delta t_{chem} < \Delta t_{cfl}$
$\Delta t_{chem} = \Delta t_{cfl}$
1 Subcycle
Subroutine VCHEMEQ: Vector vtemp(nv)
Buffer Cells
7/100 Active Cells

Fig. 6 Schematic of the load-balancing strategy for reducing the chemistry integration time.

A time history of the percentage of cells (not shown) on the grid that required one subcycle ($\Delta t_{\text{chem}} = \Delta t_{\text{cfl}}$) and two or more subcycles ($\Delta t_{\text{chem}} < \Delta t_{\text{cfl}}$) to advance the solution of Eq. (9) during a detonation wave simulation shows that 99.5% of the cells required one subcycle to complete the chemistry integration task. In many instances the number of subcycles performed for a few cells was significant. Figure 5 shows the maximum number of subcycles integrated for the entire grid as a function of the propagation time of the detonation wave. The maximum number of subcycles is usually greater than 10, sometimes reaching 20 or much more. This result was typical for the unsteady detonation wave simulations.

Figure 5 shows that CHEMEQ uses between 10 and 50 internal time steps at each computational cell to perform the integration of Eq. (9) over the time-step interval of Δt_{cfl}. Moreover, the results from these internal time steps are discarded for more than 99% of those cells after the first internal time step. This result indicates that during the execution of CHEMEQ, the majority of processors are effectively idle, so that the overall performance of the subroutine is significantly reduced. Indeed, the use of the parallel architecture is quite poor since the workload is not distributed across the processors. In this case, the data parallel structure of the program and the use of an internal time step by the splitting algorithm are the factors contributing to the poor performance of CHEMEQ on the CM. If a control parallel program were used, or the chemistry was integrated using a global time step, the parallel performance of the chemistry integration would be much improved. However, the use of array indexing or a global chemistry time step might introduce other inefficiencies.

To improve the performance of CHEMEQ, and the use of the parallel computer architecture, a new load-balancing algorithm was developed to address the inefficiencies associated with the current approach. Given the distribution and number of subcycles in Fig. 5, an apparent solution to improve the load balancing is to return from CHEMEQ after one subcycle and complete the integration of the active cells (i.e., those cells that have not completed the integration) on a smaller data structure that is distributed across the processing nodes.

Figure 6 illustrates for a sample 10×10 grid the new load-balancing strategy that was developed to increase the performance of CHEMEQ. Figure 6 shows that 93% of the cells are assumed to have a chemistry time step equal to Δt_{cfl}, and the remaining 7% a chemistry time step that is less than Δt_{cfl}. In the load-balancing algorithm, CHEMEQ returns to an executive subroutine (called CHEMOD) after one subcycle, and another subroutine is called that enumerates the active cells. The data from the active cells are then gathered into one-dimensional arrays that serve as the data structures for a vector version of CHEMEQ called V_CHEMEQ. V_CHEMEQ completes the chemistry integration after the data from the active cells have been loaded. The size of the data structures in V_CHEMEQ is typically a few percent of those in CHEMEQ, hence the integration for the active cells is completed in less time than if the complete integration were performed in CHEMEQ. In addition, because the data structures in V_CHEMEQ are distributed across the processors, the load is more effectively balanced. Once the integration is completed in V_CHEMEQ, the results are scattered back from the one-dimensional arrays to their original location in the computational grid.

The load balancing is performed with CM Fortran subroutines, which use functions from the CM Fortran Utility Library to create and store array send addresses and to perform gather/scatter operations. In addition, a C subroutine is used to accomplish the enumeration task since no comparable CM Fortran functions are available. The specific procedures and functions employed by the load-balancing subroutines are given as follows:

1) Integrate through one subcycle of subroutine CHEMEQ on the parent array and exit.

2) Mark the array elements that have not completed the chemistry integration in CHEMEQ. Define the marked elements as active.

3) Enumerate the active array elements using a C language subroutine.

4) Create send addresses for the parent array using the library function CMF_my_send_address.

5) Create send addresses for an arbitrary vector of specified length and store them at the active array elements using the library function CMF_deposit_grid_coordinate.

6) Gather the parent array send address, temperature, and species number densities from the active array elements into a vector using the library function CMF_send_overwrite.

7) Complete the chemistry integration for active cells in subroutine V_CHEMEQ on the vector.

8) Scatter the temperature and number densities back from the vector to the parent array using the library function CMF_send_overwrite.

A simulation of an unsteady detonation wave was performed with and without the new algorithm to assess the performance improvements that could be obtained through load balancing. The calculations were performed on a 16K CM-200 at the U.S. Naval Research Laboratory (NRL) using a 1024×128 grid, for 1250 time steps. Figure 7 shows the CM busy time for CHEMOD as a function of the propagation time of the detonation wave. CHEMOD is the executive program that calls CHEMEQ, V_CHEMEQ, and the load-balancing subroutines, so that the CM busy time reflects the time spent performing the chemistry integration and the gather/scatter operations. The timings show that without load balancing the CM busy time for integrating Eq. (9) ranges from 1.2 to 7.8 s per time step. However, the use of the load-balancing subroutines significantly reduces the CM busy time for CHEMOD by an approximate factor of six, to 0.6 s per time step, and the range of the CM busy time becomes 0.2 s. These results indicate that load balancing can effectively reduce the solution time of spatially stiff chemical nonequilibrium flows in a data parallel environment.

The impact of the load-balancing algorithm on the performance of CHEMOD also is illustrated by the ratio of the CM busy times for CHEMOD and CONVECT. Figure 7 shows the CM busy time

Table 1 Computational grid, computed energy release, and mode number

Case	Grid size	Δx, cm	Δy, cm	Total energy release, ergs	Maximum energy density, ergs/cm^3	Mode
0	8192×1024	0.00375	0.005875	7.50×10^7	4.99×10^6	4
1	4096×512	0.0075	0.01175	7.49×10^7	4.91×10^6	4
2	2048×256	0.015	0.0235	7.45×10^7	4.74×10^6	4
3	1536×320	0.0225	0.03525	7.36×10^7	4.20×10^6	4
4	1024×128	0.030	0.0470	7.25×10^7	3.48×10^6	6

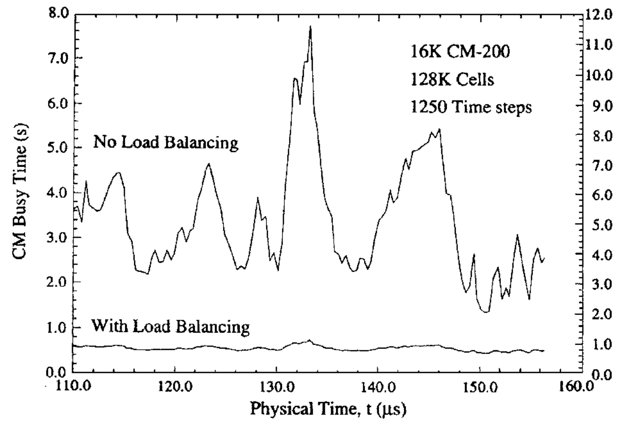

Fig. 7 Left axis: Effect of load balancing on the CM busy time of CHEMOD for a detonation simulation. Right axis: Effect of load balancing on the ratio of CM busy times for CHEMOD and CONVECT.

for CHEMOD, normalized by the CM busy time for CONVECT, as a function of the propagation time of the detonation wave. The figure shows that, without load balancing, the CM busy time for CHEMOD exceeds the CM busy time for CONVECT by a factor ranging from 2 to 11.6. This result is typical of reacting flow calculations in which the cost of integrating finite-rate chemistry is known to be significantly more expensive than integrating the fluid dynamics. Indeed, the cost of including chemical nonequilibrium effects often makes their consideration prohibitive in a large calculation. When the load-balancing algorithm is used, however, the results in Fig. 7 show that the normalized CM busy time for CHEMOD is approximately 0.9. The chemistry integration is actually less than the convection integration. (Keep in mind that, for this test case, subroutine CONVECT solves 13 conservation equations.) For all of the detonation wave simulations performed on the CM-200 and the CM-5, the normalized CM busy time for CHEMOD was less than one with the load-balancing algorithm.

Such large reductions in the CM busy time for CHEMOD were possible because the communications time for the gather and scatter operations was small compared to CHEMOD's CM busy time. In fact, the communications time for the gather/scatter operations is included in the CM busy times shown in Fig. 7. For a detonation simulation performed on a 256-node CM-5 on a 2048×512 grid, the communications time for the gather/scatter operations accounted for only 5% of the CM busy time during the chemistry integration.

Data Parallel Code Tests: Computations of Multidimensional Detonation Structure

As described above, Fig. 4 is taken from a computation of a detonation propagating in a dilute, low-pressure mixture of H_2, O_2, and Ar. Detonations in this limit are studied mainly because of the behavior of the leading shocks of the detonation, and the regularity of the patterns and structures they form. This type of mixture has been studied previously both experimentally and computationally, and thus provides a good system for testing the data parallel code and deepening our insight into the system and the physics of multidimensional detonations. The physical interpretation of this computation, the limits of such computations with respect to simulating experiments, and additional features of the structure that emerge as the resolution is improved have been discussed by Weber et al.[5,24]

Figure 4 shows that the leading shock is not planar; in fact, it consists of Mach stems and incident shocks, connected by transverse shock waves, all of which intersect at triple points at the leading shock front. These structures are dynamically changing in time. The triple points trace out a regular pattern called a detonation cell. The mode of the detonation is the number of triple points that characterize the leading shock.

The first tests of the coupled code were performed for a one-dimensional detonation and compared to results previously reported by Lefebvre et al.[25] The earlier computations were performed on a standard vector Cray computer. Note that the algorithms, general code setup, and input conditions and data were the same. The answers were also the same to extremely high accuracy. For example, at the same time step, computed profiles of density, pressure, temperature, and species concentrations were the same to within at least several decimal places. Overlaid curves look identical. This served to test the overall connections in the data parallel code, since the individual portions were tested separately, as summarized above. Although it was not possible to make a similar comparison for the two-dimensional computation with the full chemical model, we were able to test the data parallel program for the problem of a simplified induction-parameter chemistry model.[25] This required some significant simplification in the CM code, which was then in a form somewhat like the older CM C∗code reported previously,[23,26] which was used to study ignition in Mach stems[27] and imploding detonations.[28]

In general, Fig. 4 is a result of a groundbreaking computation for which there is no direct comparison to other computations. There are important physical reasons that explain why comparisons to existing experiments should be qualitative and not quantitative.[24,29] In this situation, there are standard techniques for testing the solution. Besides the types of tests described above of the various pieces and limits of the code, the major tests of the entire model involve varying the computational cell and time step in the code, and noting how and if the important global and local properties converge.

Table 1 shows the results of five computations in which the grid was successively refined. Because of the size of the computations, the five calculations were not all initialized the same way. In fact, cases 0 and 1 were initialed by interpolating the results of the less-resolved computation onto the finer grids. The result shows convergence of the parameters. The detonation velocity $D_{CJ} = 1625$ m/s is the time-averaged value of the leading shock in the middle of the system. It is the least sensitive parameter and is the same to within several decimal places for all of the calculations. (This is about 0.4% different from the detonation velocity of 1618 m/s, computed from the equilibrium computation.) The mode number quickly converges so that the number of detonation cells in the system at any time converges quickly. The specific energy density is defined as

$$\epsilon_{i,j} = \sum_{t}\left(\Delta e_{\text{chem}}^{i,j}\right) \tag{11}$$

where Δe_{chem} is the chemical energy change during a time step. Then the total energy release is calculated by summing the specific energy density, Eq. (11), over an area extending back from the leading shock by approximately 16 cm, and the maximum energy density is the maximum of $\epsilon_{i,j}$. The energy parameters are more sensitive to the change in resolution but converge quickly. As shown,[24] the structure of the detonation does not change significantly for the finest resolutions.

Conclusions

The need for greater computer performance in the next decade has driven the development of parallel computers that are capable of providing higher performance levels than conventional vector supercomputers. As a result, new CFD programs have been developed to solve problems in fluid mechanics that were too expensive to solve in the past. This paper describes a data parallel program that was developed for the CM to solve inviscid compressible flows with nonequilibrium chemical reactions. This work is the first to examine and address the issue of load balancing for solutions of stiff chemical nonequilibrium flows.

The results have shown that the CM is a suitable platform for performing reacting flow simulations. The integration of the Euler equations on the CM was especially fast, where timings showed that the floating-point performance of CONVECT ranged from 451 Mflops on the 16K CM-2 (for 32-bit calculations), to 9.4 Gflops on a 512-node CM-5 and 12.0 Gflops on a CM-5E (for 64-bit calculations). Similar timings are now reported for Navier–Stokes versions of this code on the CM. Thus, the peak performance of CONVECT on the CM-5E corresponds to 15% of the CM-5E's theoretical peak speed, which is in good agreement with the results that have been obtained for other CFD programs on parallel computers.

Because the integration of spatially stiff reacting flows can be inefficient on the CM due to poor load balancing, a new load-balancing algorithm was developed. Significant reductions in the computer time were obtained. The load-balancing algorithm reduced the chemistry integration time by a factor of six, and made the chemistry simulation less expensive than the integration of the convection equations for a multispecies gas.

One major gain from parallel computing is scalability: More processors make larger computations possible and even more economical as overhead decreases. This is illustrated explicitly in the description of the fluid dynamics algorithm given above, and it is implicit in the discussion of the integrator for the chemical reaction mechanism. However, the biggest win for the reactive-flow code is the decrease in the time that it takes for a chemical integration on a very large grid. Figure 7 summarizes a breakthrough in what can be computed. To our knowledge, such results can be achieved only by considering load balancing on parallel computers.

Finally, the data parallel program that is presented in this paper has already become a tool for conducting research on chemically reacting flows. Currently, this code and its derivative codes are being used to study fluid-chemical problems such as unsteady detonation waves, interactions of shocks and boundary layers, interactions of vortices and boundary layers, the effects of ribs on channel flow, and counterflow diffusion flames. In addition, we are porting the code to other parallel computers. Moreover, the results presented here have motivated new efforts by other researchers to develop load-balancing algorithms for chemically reacting flows that are less stiff, and to incorporate this technology into control parallel programs that are run on other types of parallel computers.

Acknowledgments

Funding for this project was provided by the Office of Naval Research, the Defense Advanced Research Projects Applied and Computational Mathematics Program, and the PALACE KNIGHT program of the U.S. Air Force. The authors would like to thank the University of Maryland Institute for Advanced Computer Studies, the Army High Performance Computing Research Center, the Naval Research Laboratory (NRL), and the National Center for Supercomputing Applications for providing access to and support of their CMs. The authors would like to acknowledge the help of Robert Whaley from Thinking Machines Corp. for his help and crucial advice. The authors thank Theodore Young and David Fyfe of NRL for providing general information on parallel computing and parallel computers. Finally, the authors would like to thank Eliza Stefaniw for her extensive help and comments on the manuscript.

References

[1]Otto, J. C., "A Parallel Implementation of the Chemically Reacting CFD Code, SPARK," *Proceedings of the Scalable High Performance Computing Conference*, IEEE Computer Society, 1992, pp. 342–349.

[2]Otto, J. C., "Parallel Execution of a Three-Dimensional, Chemically Reacting, Navier–Stokes Code on Distributed Memory Machines," AIAA Paper 93-3307, 1993.

[3]Patnaik, G., Kailasanath, K., and Sinkovits, R. S., "A New Time-Dependent, Three-Dimensional, Flame Model for Laminar Flames," *Proceedings of the 26th International Symposium on Combustion*, Combustion Inst. Pittsburgh, PA (to be published).

[4]Candler, G. V., Wright, M. J., and McDonald, J. D., "A Data-Parallel LU-SGS Method for Reacting Flows," AIAA Paper 94-0410, 1994.

[5]Weber, J. W., Jr., "Physical and Numerical Aspects of Two-Dimensional Detonation Simulations Including Detailed Chemical Kinetics on a Massively Parallel Connection Machine," Ph.D. Thesis, Dept. of Aerospace Engineering, Univ. of Maryland, College Park, MD, 1994.

[6]Boris, J. P., and Book, D. L., "Flux-Corrected Transport I: SHASTA—A Fluid Transport Algorithm That Works," *Journal of Computational Physics*, Vol. 11, 1973, pp. 38–69.

[7]Boris, J. P., and Book, D. L., "Solution of the Continuity Equation by the Method of Flux-Corrected Transport," *Methods in Computational Physics*, Vol. 16, 1976, pp. 85–129.

[8]Young, T. R., "CHEMEQ—A Subroutine for Solving Stiff Ordinary Differential Equations," U.S. Naval Research Lab., NRL Memorandum Rept. 4091, Washington, DC, 1980.

[9]Young, T. R., and Boris, J. P., "A Numerical Technique for Solving Stiff Ordinary Differential Equations Associated with the Chemical Kinetics of Reactive-Flow Problems," *Journal of Physical Chemistry*, Vol. 81, 1977, pp. 2424–2427.

[10]Weber, Y. S., "The Numerical Simulation of the Reflected Shock-Boundary-Layer Interaction in High Performance Shock Tubes," Ph.D. Thesis, Dept. of Aerospace Engineering, Univ. of Maryland, College Park, MD, 1994.

[11]Weber, Y. S., Oran, E. S., Boris, J. P., and Anderson, J. D., Jr., "The Numerical Simulation of Shock Bifurcation Near the End Wall in a Shock Tube," *Physics of Fluids*, Vol. 7, 1995, pp. 2475–2488.

[12]Vuillermoz, P., and Oran, E. S., "Mixing Regimes in a Spatially Confined Two-Dimensional Compressible Mixing Layer," *Proceedings of the Royal Society of London, Series A: Mathematical and Physical Sciences*, Vol. 449, 1995, pp. 351–380.

[13]Piana, J., "Study of the Application of Large Eddy Simulations to Turbulent Combustion," Ph.D. Dissertation, École Centrale Paris, France, 1996.

[14]Nguyen, T. X., Saint-Martin-Tillet, X. N., and Oran, E. S., "Interactions of Vortical Structures in a Ribbed Channel," AIAA Paper 97-0434, Jan. 1997.

[15]Ott, J., Ph.D. Dissertation, Dept. of Aerospace Engineering, Univ. of Maryland, College Park, MD (in preparation).

[16]Burks, T. L., and Oran, E. S., "A Computational Study of the Chemical Kinetics of Hydrogen Combustion," U.S. Naval Research Lab., NRL Memorandum Rept. 4446, Washington, DC, 1981.

[17]Oran, E. S., and Boris, J. P., *Numerical Simulation of Reactive Flow*, Elsevier, New York (currently distributed by Prentice–Hall, Englewood Cliffs, NJ), 1987.

[18]Yee, H. C., "A Class of High-Resolution Explicit and Implicit Shock-Capturing Methods," NASA TM 101008, 1989.

[19]LeVeque, R. J., and Yee, H. C., "A Study of Numerical Methods for Hyperbolic Conservation Laws with Stiff Source Terms," *Journal of Computational Physics*, Vol. 86, [No.], 1990, pp. 187–210.

[20]Boris, J. P., Landsberg, A. M., Oran, E. S., and Gardner, J. H., "LCPFCT—A Flux-Corrected Transport Algorithm for Solving Generalized Continuity Equations," U.S. Naval Research Lab., NRL-6410-93-7192, Washington, DC, 1993.

[21]Odstrčil, D., "A New Optimized FCT Algorithm for Shock Wave Problems," *Journal of Computational Physics*, Vol. 91, 1990, pp. 71–93.

[22]Weber, Y. S., Weber, J. W., Anderson, J. D., Jr., Oran, E. S., and Li, C., "The Uniform Boundary Algorithm for Supersonic and Hypersonic Flows," *Parallel Computational Fluid Dynamics '92: Proceedings of the Conference on Parallel CFD '92* (New Brunswick, NJ), 1993, pp. 395–406.

[23]Oran, E. S., Boris, J. P., and DeVore, C. R., "Reactive-Flow Computations on a Massively Parallel Computer," *Fluid Dynamics Research*, Vol. 10, 1992, pp. 251–271.

[24]Weber, J. W., Jr., Oran, E. S., and Anderson, J. A., Jr., "A Numerical Study of a Two-Dimensional H_2–O_2–Ar Detonation with a Detailed Chemical Reaction Mode," *Combustion and Flame* (submitted for publication).

[25]Lefebvre, M. H., Oran, E. S., and Kailasanath, K., "Computations of Detonation Structure: The Influence of Model Input Parameters," U.S. Naval Research Lab., Memorandum Rept. NRL/MR/4404-92-6961, Washington, DC, 1992.

[26]Oran, E. S., Boris, J. P., Whaley, R. O., and Brown, E. F., "Exploring Fluid Dynamics on a Connection Machine," *Supercomputing Review*, May 1990, pp. 52–60.

[27]Oran, E. S., Boris, J. P., Jones, D. A., and Sichel, M., "Ignition in a Complex Mach Structure," *Dynamics of Detonations*, edited by A. L. Kuhl, J.-C. Leyer, A. A. Borisov, and W. A. Sirignano, Vol. 153, Progress in Astronautics and Aeronautics, AIAA, Washington, DC, 1993, pp. 241–252.

[28]Oran, E. S., and DeVore, C. R., "The Stability of Imploding Detonations: Results of Numerical Simulations," *Physics of Fluids*, Vol. 6, 1994, pp. 1–12.

[29]Lefebvre, M. H., Oran, E. S., Kailasanath, K., and Van Tiggelen, P. J., "The Influence of the Heat Capacity and Diluent on Detonation Structure," *Combustion and Flame*, Vol. 95, 1993, pp. 206–218.

W. Oberkampf
Associate Editor