A Parallel Implementation of the Chemically Reacting CFD Code, SPARK

John C. Otto

NASA Langley Research Center
Hampton, VA 23665-5225

Abstract

This paper describes a parallel version of the two-dimensional, chemically reacting CFD code, SPARK. The sequential code has been ported to run on the Intel iPSC/860-based parallel computers. Routines have been added to the code which partition the problem based on the global mesh, and then assign the resulting subdomains across the processors. Two subdomain mappings have been considered. The routines which compute spatial derivatives and the routine which adds artificial viscosity to the discretization were modified to handle the subdomain boundaries interior to the global domain, and an effort has been made to overlap the required communication/computation. Measurements of the performance of the code have been made for two test problems exercising all of the available options of the parallel code thus far. While the parallel efficiency of the code is quite good, the single-node performance has been much lower than expected for this architecture.

1: Introduction

This project has been undertaken to assess the suitability of current state-of-the-art parallel computers for running computationally intensive CFD codes. This has been accomplished through porting the 2-D version of the SPARK code [1, 2] to the Intel iPSC/860-based parallel computers. The potential to run very large, memory-intensive problems on distributed parallel computers has also been a driving factor.

SPARK is a 2-D Navier-Stokes and chemistry code used to resolve high-speed reacting flows. Numerically, it is an explicit, time-accurate code which uses MacCormack's technique to integrate the equations in time and space. Four different methods can be employed to compute the spatial derivatives. The chemistry routines are written so that any chemical reaction can be modeled by making minor changes to the parameters in one of the property routines [3]. The source terms on the chemistry equations can be treated explicitly or in a point-implicit sense. For this project, a 9-species, 18-path H2–Air reaction was used as the chemistry model. Two of the four differencing schemes and both the explicit and implicit chemistry have been examined in this version of the parallel code.

The majority of the work for this project has been done on the 32–node Intel iPSC/860 hypercube operated by ICASE at NASA Langley Research Center. Runs requiring more than 32 nodes have been made on the 128–node hypercube at NASA Ames Research Center, and one problem has been solved on the 528–node Intel Delta machine at Caltech. These machines each have distributed memories requiring that logic and calls to the Intel communication primitives be added to the code to handle calculations which require data stored off-processor. The need for communication arises in SPARK when derivatives are computed along subdomain boundaries which do not correspond to the natural boundaries of the global domain. Since the derivatives are evaluated using relatively lower-order, explicit schemes, only the nearest off-processor data point is ever needed to compute a derivative at a boundary.

The SPARK code has been modified through the addition of several routines and additional lines of code to account for calculations involving non-local data. A decomposition routine has been written which divides the global domain into equally sized subdomains and assigns each subdomain to a processor. The routines which contain loops with an index shift (derivative calculations and the artificial viscosity routine) have been modified so that they can handle the calculations along subdomain boundaries where off-processor data must be used.

2: Analysis

2.1: Governing Equations

Since the details of the SPARK code are well documented elsewhere [1–3], they will not be rigorously discussed here. The vector form of the time-dependent governing equations can be expressed in terms of the Cartesian coordinate system (x,y) as

$$\frac{\partial(Q)}{\partial t} + \frac{\partial(F)}{\partial x} + \frac{\partial(G)}{\partial y} = H$$

(1)
where

\[
Q = \begin{bmatrix}
\rho \\
\rho u \\
\rho v \\
\rho E \\
\rho f_i
\end{bmatrix}
\]

\[
F = \begin{bmatrix}
\rho u \\
\rho u^2 - \sigma_x \\
\rho uv - \sigma_y \\
(\rho E - \sigma_x)u - \tau_{xy}v + q_x \\
\rho(u + \bar{u})f_i
\end{bmatrix}
\]

\[
G = \begin{bmatrix}
\rho v \\
\rho uv - \sigma_y \\
\rho v^2 - \sigma_y \\
(\rho E - \sigma_y)v - \tau_{xy}u + q_y \\
\rho(v + \bar{v})f_i
\end{bmatrix}
\]

\[
H = \begin{bmatrix}
\rho \sum_{i=1}^{n} f_i b_{ix} \\
\rho \sum_{i=1}^{n} f_i b_{iy} \\
\rho \sum_{i=1}^{n} f_i \hat{b} \cdot (\hat{\mathbf{V}} + \mathbf{v}_i)
\end{bmatrix}
\]

The first four entries in the \(Q\), \(F\), \(G\), and \(H\) vectors (Equations 2–5) form the continuity, conservation of momentum in the x- and y-directions, and the conservation of energy equations, respectively. The last row of terms in these vectors (one for each species 'i') form the conservation of species equations. Details of the modelling of the terms in brackets [2] will not be repeated here.

2.2: Numerical Scheme

The numerical method used to advance the governing equations (Equations 1–5) impacts the parallel implementation of SPARK. The equations expressed in physical space \((x,y)\) are first transformed to a uniform computational space \((\xi,\eta)\) and then discretized. The SPARK code incorporates the spatially unsplit, predictor-corrector scheme of MacCormack. The formulation is fully explicit and second-order accurate in both time and space. Neglecting the source terms and geometric conservation law terms for simplicity, equation 1 is expressed with this formulation by

Predictor:

\[
\tilde{Q}^{n+1}_{i,j} = \tilde{Q}^n_{i,j} - \Delta t \left( \tilde{R}^n_{i,j} \right)
\]

Corrector:

\[
Q^{n+1}_{i,j} = \frac{1}{2} \left[ \tilde{Q}^n_{i,j} + Q^{n+1}_{i,j} - \Delta t \left( \tilde{R}^{n+1}_{i,j} \right) \right]
\]

where the forward and backward differenced residuals are calculated by

\[
\tilde{R}^n_{i,j} = \frac{1}{\Delta x} \left( \hat{F}^n_{i+1,j} - \hat{F}^n_{i,j} \right) + \frac{1}{\Delta y} \left( \hat{G}^n_{i,j+1} - \hat{G}^n_{i,j} \right)
\]

\[
\tilde{R}^{n+1}_{i,j} = \frac{1}{\Delta x} \left( \hat{F}^{n+1}_{i+1,j} - \hat{F}^{n+1}_{i,j} \right) + \frac{1}{\Delta y} \left( \hat{G}^{n+1}_{i,j+1} - \hat{G}^{n+1}_{i,j} \right)
\]

and \(\tilde{Q}, \hat{F}, \hat{G}\), and \(\tilde{R}\) are now expressed in the computational space \((\xi,\eta)\). The stencil for this scheme extends one gridpoint in each direction from the point in question, requiring a relatively low amount of communication. The global domain boundaries are handled in the same manner in the parallel code as they are in the sequential version of SPARK.

A second numerical scheme, the compact-cross MacCormack scheme based on the MacCormack formulation just described, is also included in the parallel SPARK code. This scheme uses an alternate method to compute the residuals given in Equations 8 and 9. The forward and backward differenced residuals for this scheme are expressed as

\[
\tilde{R}^n_{i,j} = \frac{1}{\Delta x} \left( 1 + \frac{\delta^2}{3!} \right) F^n_{i+1,j} - F^n_{i,j}
\]

\[
+ \frac{1}{\Delta y} \left( 1 + \frac{\delta^2}{3!} \right) G^n_{i,j+1} - G^n_{i,j}
\]

\[
\tilde{R}^{n+1}_{i,j} = \frac{1}{\Delta x} \left( F^{n+1}_{i+1,j} - F^{n+1}_{i,j} \right) + \frac{1}{\Delta y} \left( G^{n+1}_{i,j+1} - G^{n+1}_{i,j} \right)
\]

where \(\delta^2 \phi = \phi_{i+1} - 2\phi_i + \phi_{i-1}\). This scheme has a numerical stencil similar to the second-order MacCormack method, but now the nearest diagonal points are also involved in the derivative calculation, resulting in a nine-point stencil. The communication requirement is approximately three times as great as that of the second-order
scheme due to the message passing required to calculate the central differences $\delta^2_x$ and $\delta^2_y$.

For flows where the chemical reaction timescale is much smaller than the fluid flow timescale, the source terms of the chemistry equations limit the time-step size. The option to implicitly update the source terms of the chemistry equations exists to alleviate the stiffness of the equations. Since the formulation is implicit in only a pointwise sense, the parallel implementation is not impacted by the implicit chemistry.

2.3: Decomposition Routine

The SPARK code solves the equations utilizing a fully explicit MacCormack technique over a structured mesh. As a result, the data arrays correspond to the mesh points consistently during a run, allowing the problem to be distributed among the processors based on the physical space. A routine has been written which partitions the global domain into equally sized subdomains. This routine contains two options for how the global domain is to be divided. One of two mappings is then utilized to assign the decomposed global domain to the processors. The decomposition routine then sets the value of various parameters which define the relationship between the subdomains and the global domain.

The first method of partitioning the global arrays, referred to here as a (Block, Block) decomposition, is shown on the left in Figure 1. Here the global domain is partitioned in both indicial directions. The partitioned mesh is then assigned across the processors based either on a simple sequential numbering of the subdomains or on a binary Gray code mapping, as indicated in parentheses in Figure 1. The former assures that all communication will occur between directly connected processors on the 2-D mesh architecture of the Delta machine, while the latter assures nearest neighbor communication on the hypercube architecture.

The (Block, Block) decomposition produces subdomains which are nearly square based on their dimensions. It has the advantage that the ratio of boundary lengths to the subdomain area is near the minimum, reducing the communication/computation ratio, which impacts the efficiency of the parallel implementation. The (Block, Block) decomposition has two main drawbacks. The first arises from the fact that processors assigned to the interior subdomains must potentially initiate four communications (one with each adjacent processor) for each derivative calculation. Second, message passing in the first indicial direction is slightly more complicated than in the last direction due to FORTRAN array storage.

A second decomposition method has been employed in the parallel SPARK code with the intention of simplifying the communication requirements over the (Block, Block) decomposition. This method, the (*, Block) decomposition (right side of Figure 1), partitions the global array in only the second indicial direction. The number of communication initializations is reduced to, at most, two per processor, but the subdomain boundary length/subdomain area ratio is larger. This method eliminates the need to pass messages in the first indicial direction, thus simplifying the communication. As with the (Block, Block) decomposition, both the Gray code mapping and sequential ordering are employed with the (*, Block) decomposition.

2.4: Communication Strategy

Routines in SPARK which compute spatial differences potentially require off-processor data when computing values along local domain boundaries. SPARK is highly modular, and since the majority of the routines perform point calculations, they require very minimal modification to run in the parallel environment. The routines which compute spatial derivatives and the routine which adds artificial viscosity for stability were modified to handle calculations which require off-processor data.

The communication requirement can best be described through a simple example. Consider the following central difference in the x-direction

$$\delta^2_x \phi_{i,j} = \phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}$$

where $\phi$ is a two-dimensional array similar to the arrays in SPARK, and $i$ and $j$ refer to the first and second index, respectively. To perform this calculation on a global mesh distributed across four processors (Figure 2), communication is required to obtain values along subdomain boundaries. More specifically, to perform the calculation on the right hand boundary of processor zero, the nearest column of data is required from processor 1. Likewise, to perform the calculation on the left hand boundary of processor one, the nearest column of data is needed from processor zero. A similar relationship exists between processor two and processor three. Calculations over the unshaded regions in Figure 2 do not require off-processor
data and, therefore, are performed in the same manner as in the sequential code.

To accomplish calculations similar to Equation 12, code has been added to SPARK to identify if a communication is required, to perform the needed communication, and then to perform the calculation. The asynchronous message-passing pairs of ISEND/IRECV have been used for all the communications. A typical derivative calculation takes the following steps:

1. The required communication is initiated through the ISEND/IRECV calls
2. While the communication is taking place, calculations are performed over the interior of the subdomain
3. Execution is halted by a MSGWAITO call until communication is complete
4. Subdomain boundaries which require off-processor data are updated
5. Global boundaries are updated as they are in the sequential code

If step (ii) above requires as much or more time than is required to perform the communication, the communication penalty will, in effect, be hidden. Only a slight overhead is incurred to initiate the commands and step (iii), the MSGWAITO command, will not stop execution.

Figure 2. Global mesh distributed across 4 processors.

2.5: Test Problems

Several problems were used as test cases during this project. Only two will be discussed here, as they have received the most detailed attention and best illustrate the capabilities and shortcomings of the parallel code. The first problem was used throughout the port to exercise the chemistry routines and to validate the parallel implementation. It models the flow through a nozzle with a 9-species, 18-reaction path H₂-Air reaction (Figure 3). This problem was run primarily on the Intel iPSC/860 hypercubes at NASA's Langley and Ames Research Centers. It was used for all the performance studies documented in this paper.

The second problem, a reacting shear layer [4], was implemented on the 528-node Touchstone Delta machine at Caltech. A schematic of this problem is given in Figure 4. It uses the same 9-species, 18-reaction path H₂-Air reaction chemistry model. A hyperbolic tangent velocity profile is enforced at the inflow. This problem has been run only once, on a 160x144 mesh, and has not been extensively explored with the parallel code.

Figure 3. 40x40 global mesh of 2-D nozzle.

Figure 4. Schematic of reacting shear layer problem.

3: Results

Before examining the performance of the parallel version of SPARK, it was first necessary to verify that the parallel and sequential versions of the code would produce the same results for identical problems. To ensure that the size of the floating point numbers would not have an effect, double precision real numbers (64-bit) were used throughout the parallel code. This was done to account for the 64-bit floating point numbers used by the Cray computers as a default.

Comparisons of the residual plots and of solutions to a given problem for various numbers of processors and decomposition methods have been used to check that the parallel code functions correctly. The residual for both the
parallel code and sequential code were written for 15,000 time-steps of the nozzle problem and compared. They agreed very well and followed the same trends. The parallel code produced solutions which were independent of the number of processors, decomposition, and processor mapping. This was confirmed through comparisons of binary dumps of the various flow quantities and by comparing the extent that the solution advanced in time after a given number of time-steps. The comparisons indicated that the global numerical scheme used in the sequential code was preserved in the distributed implementation.

The following section describes the results from various performance measurements that have been made of the parallel SPARK code. The efficiency of the code has been measured when the number of processors working on a fixed problem size is increased, and also when the problem is scaled with the number of processors. The nozzle problem is the subject of each of these tests. Lastly, the performance of the reacting shear layer problem is described for one case which was run on the Delta machine.

3.1: Parallel Efficiency

The efficiency of the parallel code indicates how well the implementation exploits the parallel architecture. For this study, the size of the global mesh was held constant while increasing the number of processors working on the problem. The single processor memory size limited this study to a $36 \times 40$ global mesh. For each case, the problem was advanced 100 time-steps and the execution time was measured. The efficiency was calculated by $\epsilon = \frac{T_1}{T_N}$, where $T_1$ is the time to run on one processor and $T_N$ is the time required to run on $N$ processors. The timings do not include any input/output operations.

Figure 5 shows a plot of the efficiency verses the order of the hypercube. The “hypercube order” is the base-two logarithm of the number of processors. The four cases shown in the figure correspond to the different combinations of the second-order and compact MacCormack schemes and the explicit and point-implicit chemistry routines. A (Block, Block) decomposition was used with a Gray code mapping.

The efficiency decreases for each case as the number of processors increases and falls off markedly for the 32-processor case. The decrease in the efficiency can be attributed to two sources. The first and most obvious is the increased amount of communication which takes place relative to the amount of calculations performed on each processor. The second factor which affects the efficiency is the size of the subdomain assigned to each processor. The small subdomains, which are the result of increasing the processors, reduce the vector length in the do-loops.

Figure 5. Parallel efficiency vs. hypercube order holding global mesh size constant.

Measurements indicate that the i860 performance is sensitive to do-loop length and, in most cases, longer vectors give higher Mflop rates. The reduced performance on short vector lengths has been observed for the SPARK code as well.

It is also evident from Figure 5 that the second-order MacCormack scheme has a higher efficiency than the compact scheme, and the implicit chemistry cases perform better than the explicit chemistry cases. The compact scheme is significantly more complicated than the second-order scheme and, as such, requires a greater amount of communication to calculate the spatial derivatives. The increased communication requirement for this scheme results in a lower efficiency. A far greater number of calculations, which do not require off-processor data, take place when updating the source terms on the chemistry equations implicitly. The result is an increased parallel efficiency when the implicit chemistry option is exercised.

The Mflop rates corresponding to the cases of Figure 5 are given in Table 1. They were calculated by a direct count of the number of operations performed per run. Intrinsic function calls, such as square roots and exponentials, were counted as a single operation. These values give a sense of the actual realized performance of the parallel code for the various options. The Mflop rates are significantly higher for the two cases which utilize the point-implicit chemistry routines. This can mostly be attributed to the fact that these routines operate very efficiently while increasing the operation count by about 150% over the explicit chemistry. The nonlinear increase in the performance, with respect to the number of processors, is a result of the inefficiency of the code for larger numbers of processors.

The communication time, for the two cases which use the second-order MacCormack scheme, has been measured for this problem. Since the asynchronous communication
Table 1. Mflop rates for 36x40 global mesh held constant.

<table>
<thead>
<tr>
<th>Processors</th>
<th>2nd order Implicit</th>
<th>2nd order Explicit</th>
<th>Compact Implicit</th>
<th>Compact Explicit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.11</td>
<td>1.29</td>
<td>2.15</td>
<td>1.37</td>
</tr>
<tr>
<td>2</td>
<td>4.06</td>
<td>2.52</td>
<td>4.08</td>
<td>2.41</td>
</tr>
<tr>
<td>4</td>
<td>7.61</td>
<td>4.66</td>
<td>7.53</td>
<td>4.35</td>
</tr>
<tr>
<td>8</td>
<td>14.41</td>
<td>8.62</td>
<td>13.80</td>
<td>7.69</td>
</tr>
<tr>
<td>16</td>
<td>28.03</td>
<td>16.28</td>
<td>26.87</td>
<td>14.59</td>
</tr>
<tr>
<td>32</td>
<td>49.44</td>
<td>28.26</td>
<td>45.63</td>
<td>24.03</td>
</tr>
</tbody>
</table>

Table 1. Mflop rates for 36x40 global mesh held constant (second-order MacCormack).

Table 2. Communication penalty for 36x40 global mesh held constant (second-order MacCormack).

<table>
<thead>
<tr>
<th>Processors</th>
<th>Communication penalty and code efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Implicit</td>
</tr>
<tr>
<td>1</td>
<td>.0003</td>
</tr>
<tr>
<td>2</td>
<td>.0344</td>
</tr>
<tr>
<td>4</td>
<td>.0918</td>
</tr>
<tr>
<td>8</td>
<td>.0918</td>
</tr>
</tbody>
</table>

With the effect of the increased communication to computation ratio removed from the run times, the efficiency of the parallel code is increased significantly. This is shown in Figure 6, where the efficiency plotted in Figure 5, which includes the communication penalty, is compared with the adjusted efficiencies given in Table 2. The explicit chemistry case experiences the greatest increase in efficiency. This is expected since the explicit chemistry requires much fewer calculations per step than the implicit, with an equal communication requirement. Therefore, it pays a greater penalty, relative to the total run time, to perform the communications. When the communication is eliminated from the explicit case, it in turn benefits more than the implicit case. The adjusted efficiencies are expected to remain near 1.00, independent of the number of processors, but this is not the case. The reason the adjusted efficiencies decrease with the number of processors is due mainly to the decreased computational performance on the shorter vector lengths. The vectors for the 16-processor case are dimensioned to 9x10, and the vectors for the 32-processor case are 9x5. The performance with the short vectors is worse than the single-processor case operating on 36x40 vectors.

Figure 6. Adjusted parallel efficiency vs. hypercube order holding global mesh size constant.

One of the reasons for porting SPARK to a distributed memory machine was to exploit the higher total memory potential that exists with this type of computer. The 9-species, 18-reaction path chemistry problem is limited, by the 8-Mbyte memory per processor, to approximately a 38x38 domain size per processor. In order to study the efficiency of the maximum sized problem, the local mesh size was held constant at 38x38 points and scaled with the number of processors. The (Block, Block) decomposition was used with a Gray code mapping. The problem was advanced 100 steps, and the time was measured using the Intel DCLOCK() routine. The efficiency was calculated from the equation \( \epsilon = \frac{T_1}{T_N} \), where \( T_1 \) is the time to run on one processor and \( T_N \) is the time required to run on \( N \) processors.

The results from this study are shown in Figure 7. The efficiency is plotted against the order of the hypercube (base-two logarithm of the number of processors). All the cases shown in this figure follow the same trends as they did in Figure 5, but the magnitude of the drop off in efficiency is much less. The efficiency for these cases never drops below 92%. For the 16, 32, and 64-processor cases (4 through 6 on the plot), the run time for 100 steps changes very little. This results from the fact that the
communication requirement for a processor assigned to an interior subdomain reaches a maximum at 16 processors and the communication/computation ratio then remains constant.

![Graph showing parallel efficiency vs. hypercube order holding local mesh size constant.](image)

**Figure 7.** Parallel efficiency vs. hypercube order holding local mesh size constant.

Table 3 shows the Mflop rates for the cases plotted in Figure 7. In these cases the vector sizes remain constant and the communication/computation ratio reaches its maximum for the 16-processor case. This gives a near linear increase in performance when the problem size scales with the processors. The higher Mflop rate, seen previously for the point-implicit chemistry cases, is also evident.

<table>
<thead>
<tr>
<th>Processors</th>
<th>Mflop rates of the parallel code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2nd order Implicit</td>
</tr>
<tr>
<td>1</td>
<td>2.11</td>
</tr>
<tr>
<td>2</td>
<td>4.18</td>
</tr>
<tr>
<td>4</td>
<td>8.28</td>
</tr>
<tr>
<td>8</td>
<td>16.35</td>
</tr>
<tr>
<td>16</td>
<td>32.52</td>
</tr>
<tr>
<td>32</td>
<td>65.09</td>
</tr>
<tr>
<td>64</td>
<td>130.65</td>
</tr>
</tbody>
</table>

* measured on Ames Research Center hypercube

**Table 3.** Mflop rates for 38x38 local mesh held constant.

This study also provides an indication of the maximum problem size which can be run on the various computers used in the project. The 38x38 subdomain size translates to a maximum global mesh of 152x304 for 32 processors and a global mesh of 304x608 points if all 128 processors of the Ames machine were used. On the Delta machine, which has a 16-megabyte memory per processor, a problem size of 1216×1216 mesh points could be solved with 512 processors.

### 3.2: Reacting Shear Layer

The Intel Touchstone Delta machine at Caltech has been used to solve one problem with the parallel SPARK code. The intention of this exercise was to demonstrate the resource that this computer provides for memory-intensive problems. The reacting shear layer problem has been chosen since it is well documented [4] and is a 2-D problem whose solution, thus far, has been limited by the mesh size. The 160×144 mesh used for this problem is far less than the maximum that potentially could be solved on the Delta machine. A larger size was prohibited by the time it would take to generate a solution. The problem was run on 256 processors, and a (Block, Block) decomposition was incorporated. This results in a 10×9 local problem size which utilizes only about 5% of each individual processor memory. The subdomains were mapped using a sequential numbering scheme instead of the Gray code mapping since the Delta is configured in a 2-dimensional mesh architecture.

The timing results from this run are summarized in Table 4. The problem was solved in a time-accurate mode with the second-order MacCormack scheme and the point-implicit chemistry routines. The problem was advanced in time to a statistical steady state, and the flow field was then sampled every 50 steps, for a total for 11 samples. The total run time measured for this problem was about one-fourth of what is expected on the Cray Y-MP. The problem did not utilize the processors very efficiently, however, and this is reflected in the 1.66 single-processor Mflop rate. The poor performance can be attributed to the very high communication/computation ratio for such a small local problem size. If the maximum sustained Mflop rate measured for larger local problem sizes had been obtained here, the speed would have exceeded that of the Cray by more than five times.

<table>
<thead>
<tr>
<th>Mflop rates</th>
<th>426 Mflops</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run time:</td>
<td>1798 sec.</td>
</tr>
<tr>
<td>4050 steps</td>
<td>.44 sec./step</td>
</tr>
</tbody>
</table>

**Table 4.** Results from reacting shear layer problem.
4: Concluding Remarks

The chemically reacting CFD code, SPARK, has been ported to the Intel parallel computers. Several problems have been used during the port to validate the parallel code. The project was undertaken to assess current state-of-the-art parallel computers for use on problems of this type, and this has been accomplished. The port has provided a good indication of the potential that these computers provide and has identified areas of the code in need of improvement.

The parallel computer provides a one-of-a-kind resource for solving very large, memory-intensive CFD problems similar to what have been explored in this project. The complex chemical reactions modelled by the SPARK code carry with them a sizable memory requirement. The 128-processor hypercube at Ames Research Center has 1.024 gigabytes of memory, and the 528-processor Delta machine at Caltech has a total memory capacity of 8.448 gigabytes. Both machines surpass resources currently used. The ability to solve a problem equivalent to the largest solved to date with the SPARK code has been demonstrated. Using half the Delta machine, it requires about 25% of the time that is needed on the Cray Y-MP. Larger problems would run more efficiently on the parallel machines and require still smaller percentages of the Cray time to compute a solution.

The parallel version of SPARK has been able to surpass the Cray Y-MP performance for cases solved on 64 or more processors. While this fact does hold some promise, it also indicates that perhaps greater speeds could be achieved with the parallel code. The maximum sustained computing rate per node has been measured at approximately 2.1 Mflops. Rates two and possibly three times this value should be achievable. When Mflop rates such as these are attained in the parallel code, Cray performance will be achieved with somewhere between 16 and 32 processors on the Intel machines.

Several areas of the code have been identified as severely restricting the overall code performance. One area in particular involves a call to the square root routine. When counted as a single operation, this routine has been measured at 0.3 Mflops for vector lengths typically employed in the SPARK code. One single call to this routine in a chemistry routine of the SPARK code accounts for less than 1% of the total operations but over 25% of the total run time. If this call was not included in the performance results of the reacting shear layer (Table 4), the Mflop rates would increase from 1.66 Mflops/node (426 Mflops total) to 2.21 Mflops/node (556 total). This increase in performance reduces the time required per step from .44 seconds to .33 seconds. Clearly, for the performance of the SPARK code to reach expected levels, these issues must be resolved.

References


