# A Dataparallel Implementation of an Explicit Method for the Three-Dimensional Compressible Navier-Stokes Equations

Pelle Olsson and S. Lennart Johnsson

YALEU/DCS/TR-747
October 1989

# A Dataparallel Implementation of an Explicit Method for the Three-Dimensional Compressible Navier-Stokes Equations.

Pelle Olsson[1]
Department of Scientific Computing
Uppsala University
Uppsala, Sweden
and
Department of Computer Science
Yale University
New Haven, CT 06520

S. Lennart Johnsson
Thinking Machines Corp.
245 First Street
Cambridge, MA 02142
and
Departments of Computer Science
and Electrical Engineering
Yale University
New Haven, CT 06520

October 2, 1989

[1]This project was partially sponsored by The Swedish Board of Technical Development (STU), reg-no 61008767, by Jacob Letterstedts fond, The Royal Swedish Academy of Sciences, by the Office of Naval Research under Contract No. N00014-86-K-0310, and by the United States Air Force under contract AFOSR-89-0382.

## Abstract

The fluid flow in a three-dimensional twisted channel is modeled by both the compressible Navier-Stokes equations, and the Euler equations. A three stage Runge-Kutta method is used for integrating the system of equations in time. A second order accurate, centered difference scheme is used for spatial derivatives of the flux variables. For both the Euler and the Navier-Stokes equations artificial viscosity introduced through fourth order centered differences are used to stabilize the numeric scheme. By using lower order difference approximations on or close to the boundary than in the interior, the difference stencils can be evaluated at *all* grid points concurrently. A few different difference molecules for the boundaries, and different factorizations of the fourth order difference operators were evaluated. With the appropriate factorization of the difference stencils, six variables per lattice point suffice for the evaluation of the difference stencils occurring in the code. The three fourth order stencils we investigated, including three different factorizations of one of these stencils, account for three out these six variables. The convergence rate for all stencils and their factorizations is approximately the same for the first 1000 – 1500 steps at which point the residual has reached a value of $10^{-2}$ – $10^{-3}$. From this point on the convergence rate for one of the factorizations of the fourth order stencil is approximately twice that of one of the unfactored stencils.

A performance of 1.05 Gflops/s was demonstrated on a 65,536 processor Connection Machine system with 512 Mbytes of primary storage. The performance scales in proportion to the number of processors. The performance on 8k processor configurations was 135 Mflops/s, on 16k processors 265 Mflops/s, and 525 Mflops/s on 32k processors. The efficiency is independent of the machine size. The evaluation of the boundary conditions accounted for less than 5% of the total time. A performance improvement by a factor of about three is expected with optimized implementations of functional kernels such as convolution, and matrix-vector multiplication.

# 1 Introduction

Fluid flow problems are computationally very demanding. The full Navier-Stokes equations completely describe the fluid flow at the macroscopic level. The equations describe the balance of mass, linear momentum and energy, and model the turbulent phenomena that occur in viscous flow. In order to resolve all the small scale effects, very fine grids are needed. It has been shown [11], that for incompressible flow the minimum scale needed to resolve turbulent phenomena is no smaller than

$$\lambda_{min} = \frac{1}{\sqrt{Re||D\mathbf{u}||_\infty}},$$

where $Re$ is the Reynolds number. The quantity $||D\mathbf{u}||_\infty$ is defined as

$$||D\mathbf{u}||_\infty = \sup_{t,x,y,z} \left( |\frac{\partial}{\partial x}\mathbf{u}(t,x,y,z)|_2, |\frac{\partial}{\partial y}\mathbf{u}(t,x,y,z)|_2, |\frac{\partial}{\partial z}\mathbf{u}(t,x,y,z)|_2 \right).$$

The vector $\mathbf{u}$ represents the velocity field, i.e.

$$\mathbf{u} = \begin{pmatrix} u \\ v \\ w \end{pmatrix},$$

where $u, v$ and $w$ are the Cartesian velocity components; $|\cdot|_2$ denotes the Euclidean length of the vector.

In three space dimensions the bound for the velocity gradient $||D\mathbf{u}||_\infty$ is not known. A possible bound is

$$||D\mathbf{u}||_\infty \sim \sqrt{Re}.$$

This choice corresponds to the so called Kolmogoroff scale of

$$\lambda_{min} = \frac{1}{\sqrt[4]{Re^3}}.$$

A realistic treatment of the inflow means a Reynolds number of $\sim 50,000$. This implies that the Kolmogoroff scale is

$$\lambda_{min} = 0.0003.$$

For such a length scale the grid point spacing must not exceed $\Delta x = 0.00015$. This condition holds for each space dimension. Hence, without a turbulence model, a grid consisting of approximately 300 billion points is needed. At each grid point we use roughly 150 variables, which are stored as floating point numbers. Consequently, 240,000 Gbytes of memory are needed to represent the state of the Navier-Stokes equations for a resolution capturing all scales. Assuming in the order of 2,500 floating point operations for each variable per time step approximately 1,000 trillion operations are required per time step. Clearly, both storage and computational demands are beyond the capacity of today's

computers. Turbulence models are the only feasible alternative for the study of turbulent phenomena. The idea in the various turbulence models is to separate small scale effects from large scale effects by introducing extra differential equations. The implementation described below does not include a turbulence model. Even though it is not feasible to compute all scales by the Navier-Stokes equations they describe the solution more accurately near solid walls than do the Euler equations, since viscous effects are very prominent near solid boundaries.

The next generation supercomputers with a performance of a trillion operations per second are expected to have thousands to tens of thousands of processing units interconnected by a hierarchy of buses or some other network. It is important to develop algorithms that can make effective use of such architectures. Explicit methods are obvious candidates for highly concurrent architectures, and is the subject of this investigation. We focus on the data structures for the data parallel implementation, the programming primitives in such an implementation, and to some extent the algorithms. Load balance and maximum communication efficiency are issues of particular concern in a data parallel implementation, but the ability to express the computations in terms of standard functions, as for instance the Basic Linear Algebra Subroutines (BLAS) is always important. The latter simplify the programming and enhance performance (by virtue of being efficiently implemented). We present the techniques used in our implementation for concurrent evaluation of the difference stencils at all grid points.

The model problem is the simulation of air flow in a twisted channel. This problem is motivated by a study to improve the combustion in a diesel engine by redesigning the combustion chamber and cylinder inlet. Of particular interest is the air flow near the cylinder inlet. The study reported here only includes the inlet channel. The fluid flow simulations are reported in [26, 25] which also addresses numerical issues. The work reported here does not include any turbulence model.

The outline of this paper is as follows. We present the Euler and Navier-Stokes equations in section 2. The numerical method used in the simulation is presented in section 3. In section 4 we give a brief description of the data parallel programming model, the Connection Machine system model CM–2, and issues related to load balance and communication efficiency. Implementation characteristics of the finite difference, explicit in time algorithms are presented in section 5. Summary and conclusions are given in section 6.

# 2 Mathematical Model

## 2.1 The Domain

The physical and computational domains are shown in Figure 1. The cross-sectional physical side lengths are $L_x$ and $L_y$, and the physical channel length is $L_z$. The relationship between the variables $\tau$,$\xi$, $\eta$ and $\zeta$ in the computational domain, and $t$, $x$, $y$, and $z$ in the

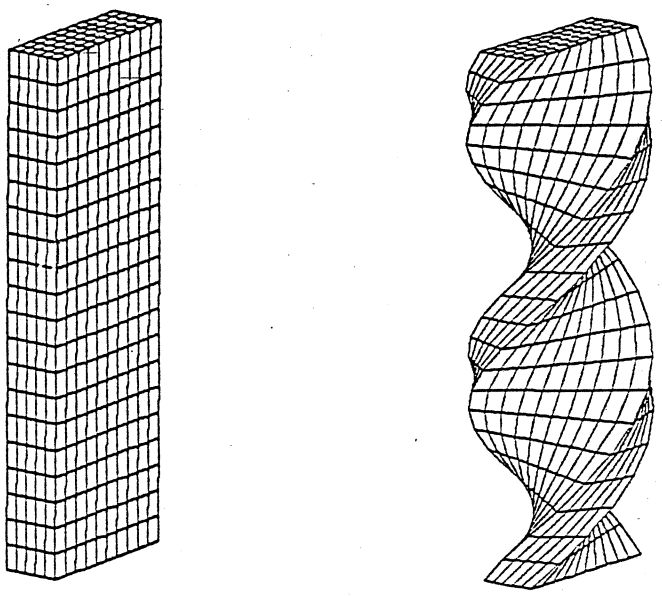

Figure 1: Computational and Physical Domain

physical domain is formally given by

$$\begin{cases} \tau &= t \\ \xi &= \xi(t,x,y,z) \\ \eta &= \eta(t,x,y,z) \\ \zeta &= \zeta(t,x,y,z) \end{cases}$$

assuming differentiability and non-singularity as needed. The Jacobian matrix $\mathbf{J}$ is defined by

$$\mathbf{J} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \xi_t & \xi_x & \xi_y & \xi_z \\ \eta_t & \eta_x & \eta_y & \eta_z \\ \zeta_t & \zeta_x & \zeta_y & \zeta_z \end{pmatrix}. \tag{1}$$

The subscripts denote partial differentiation. The functional determinant of the Jacobian $|\mathbf{J}|$ corresponds to the reciprocal cell volume:

$$|\mathbf{J}| = \det \mathbf{J} = \det \begin{pmatrix} 1 & 0 & 0 & 0 \\ \xi_t & \xi_x & \xi_y & \xi_z \\ \eta_t & \eta_x & \eta_y & \eta_z \\ \zeta_t & \zeta_x & \zeta_y & \zeta_z \end{pmatrix} = \det \begin{pmatrix} (\nabla\xi)^T \\ (\nabla\eta)^T \\ (\nabla\zeta)^T \end{pmatrix}, \tag{2}$$

where

$$\nabla^T = \begin{pmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \end{pmatrix}. \tag{3}$$

For the discretization of the domain we use two grids: for viscous flows a stretched grid so as to produce fine grids near the solid walls, and for inviscid flows a grid with

equidistant grid spacing. The computational grid $\xi \times \eta \times \zeta$, $0 \leq \xi_j \leq N_\xi$, $0 \leq \eta_k \leq N_\eta$ and $0 \leq \zeta_l \leq N_\zeta$ is mapped to the physical domain through the transformation

$$\left\{ \begin{array}{lcl} x &=& \xi' \cos(\omega(\zeta)/N_\zeta) \;-\; \eta' \sin(\omega(\zeta)/N_\zeta) \\ y &=& \xi' \sin(\omega(\zeta)/N_\zeta) \;+\; \eta' \cos(\omega(\zeta)/N_\zeta) \\ z &=& \zeta L_z / N_\zeta \end{array} \right. , \tag{4}$$

where

$$\left\{ \begin{array}{lcl} \xi' &=& \frac{L_x}{2} \frac{\tanh\left(\Xi\left(2\xi/N_\xi - 1\right)\right)}{\tanh(\Xi)} \\ \eta' &=& \frac{L_y}{2} \frac{\tanh(\Xi(2\eta/N_\eta - 1))}{\tanh(\Xi)} \end{array} \right. \tag{5}$$

and

$$\omega(\zeta) = \left\{ \begin{array}{ll} 0 & \zeta \in [0, \zeta_0) \\ \bar{\omega}(3\zeta_1 - \zeta_0 - 2\zeta)(\zeta - \zeta_0)^2(\zeta_1 - \zeta_0)^{-3} & \zeta \in [\zeta_0, \zeta_1) \\ \bar{\omega} & \zeta \in [\zeta_1, N_\zeta] \end{array} \right. . \tag{6}$$

Grid stretching is introduced through the transformation $\frac{\tanh(\psi)}{\tanh(\Xi)}, -\Xi \leq \psi \leq \Xi$. A very small value of $\Xi$ results in a practically unstretched grid. For the Euler equations we use $\Xi = 0.0001$. A value of $\Xi = 1.2$ is used for the Navier-Stokes equations. The relationship between the physical and computational grids is such that the origin of the physical coordinates coincides with the center of the channel cross-section at the inflow boundary. $\omega(\zeta)$ is a $C^1$-spline on the interval $[0, N_\zeta]$. Introducing this spline function enables a $C^1$-grid transformation, which is such that the homogeneous inflow condition $u = 0,\ v = 0,\ w = w_0$, yields continuous velocity gradients even at the inflow boundary. The function $\omega(\zeta)$ describes how the twisting factor (angular frequency) increases along the $\zeta$-axis, which coincides with the physical $z$-axis. The value of $\bar{\omega}$ is

$$\bar{\omega} = 2\pi / N_\zeta .$$

In [25] we show that this grid is well conditioned [7]. The metric coefficients are derived from relations (4), (5) and (6) and

$$\mathbf{J}'' = \begin{pmatrix} \xi_x & \xi_y & \xi_z \\ \eta_x & \eta_y & \eta_z \\ \zeta_x & \zeta_y & \zeta_z \end{pmatrix} = \begin{pmatrix} x_\xi & x_\eta & x_\zeta \\ y_\xi & y_\eta & y_\zeta \\ z_\xi & z_\eta & z_\zeta \end{pmatrix}^{-1} . \tag{7}$$

Note that the grid is not stretched in the $z$-direction.

## 2.2 Fluid Flow Models

We have implemented two models: A compressible Navier-Stokes model, and an Euler model. We first describe the Euler flow model in conservative form, then define the complete Navier-Stokes model by appending terms for the viscosity.

### 2.2.1 The Interior

The conservative form of the Euler equations is

$$\frac{\partial \mathbf{q}}{\partial \tau} = \frac{\partial \mathbf{F}}{\partial \xi} + \frac{\partial \mathbf{G}}{\partial \eta} + \frac{\partial \mathbf{H}}{\partial \zeta}, \tag{8}$$

where the variable vector $\mathbf{q}$ is given by

$$\mathbf{q} = |\mathbf{J}|^{-1} \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ e \end{pmatrix}. \tag{9}$$

The components of the variable vector $\mathbf{q}$ have the following meaning:

| | |
|---|---|
| $\rho$ | density |
| $\rho u$ | x-component of linear momentum |
| $\rho v$ | y-component of linear momentum |
| $\rho w$ | z-component of linear momentum |
| $e$ | total energy. |

The flux vectors $\mathbf{F}$, $\mathbf{G}$ and $\mathbf{H}$ have the following components (inviscid case):

$$\mathbf{F} = -|\mathbf{J}|^{-1} \begin{pmatrix} \rho U \\ \rho u U + p\xi_x \\ \rho v U + p\xi_y \\ \rho w U + p\xi_z \\ (e+p)U - p\xi_t \end{pmatrix} = -U\mathbf{q} - p|\mathbf{J}|^{-1} \begin{pmatrix} 0 \\ \nabla\xi \\ U - \xi_t \end{pmatrix} \tag{10}$$

$$\mathbf{G} = -|\mathbf{J}|^{-1} \begin{pmatrix} \rho V \\ \rho u V + p\eta_x \\ \rho v V + p\eta_y \\ \rho w V + p\eta_z \\ (e+p)V - p\eta_t \end{pmatrix} = -V\mathbf{q} - p|\mathbf{J}|^{-1} \begin{pmatrix} 0 \\ \nabla\eta \\ V - \eta_t \end{pmatrix} \tag{11}$$

$$\mathbf{H} = -|\mathbf{J}|^{-1} \begin{pmatrix} \rho W \\ \rho u W + p\zeta_x \\ \rho v W + p\zeta_y \\ \rho w W + p\zeta_z \\ (e+p)W - p\zeta_t \end{pmatrix} = -W\mathbf{q} - p|\mathbf{J}|^{-1} \begin{pmatrix} 0 \\ \nabla\zeta \\ W - \zeta_t \end{pmatrix}, \tag{12}$$

and $U$, $V$ and $W$ are the contravariant velocity components:

$$\begin{pmatrix} U \\ V \\ W \end{pmatrix} = \mathbf{J}' \begin{pmatrix} 1 \\ u \\ v \\ w \end{pmatrix}. \tag{13}$$

$\mathbf{J}'$ denotes the $3 \times 4$ matrix obtained by deleting the first row of $\mathbf{J}$. The pressure $p$ is related to the total energy according to

$$p = (\gamma - 1)(e - \frac{\rho}{2}(u^2 + v^2 + w^2)), \tag{14}$$

where $\gamma$ is the ratio of the specific heats $c_p/c_v$. We assume that $\gamma$ is a constant having the value 1.4.

The advantage of the conservative formulation of the Euler equations is the ability to capture shocks. Using this Euler formulation, the Navier-Stokes equations are obtained by adding the viscous flux vectors $\mathbf{F}_\nu$, $\mathbf{G}_\nu$ and $\mathbf{H}_\nu$ to $\mathbf{F}$, $\mathbf{G}$ and $\mathbf{H}$, respectively,

$$\mathbf{F}_\nu = |\mathbf{J}|^{-1}Re^{-1}\begin{pmatrix} 0 \\ \xi_x\tau_{xx} + \xi_y\tau_{xy} + \xi_z\tau_{xz} \\ \xi_x\tau_{xy} + \xi_y\tau_{yy} + \xi_z\tau_{yz} \\ \xi_x\tau_{xz} + \xi_y\tau_{yz} + \xi_z\tau_{zz} \\ \xi_x f_5 + \xi_y g_5 + \xi_z h_5 \end{pmatrix} = |\mathbf{J}|^{-1}Re^{-1}\begin{pmatrix} 0 \\ \begin{pmatrix} \boldsymbol{\tau} \\ \boldsymbol{\kappa}^T \end{pmatrix} \nabla\xi \end{pmatrix} \tag{15}$$

$$\mathbf{G}_\nu = |\mathbf{J}|^{-1}Re^{-1}\begin{pmatrix} 0 \\ \eta_x\tau_{xx} + \eta_y\tau_{xy} + \eta_z\tau_{xz} \\ \eta_x\tau_{xy} + \eta_y\tau_{yy} + \eta_z\tau_{yz} \\ \eta_x\tau_{xz} + \eta_y\tau_{yz} + \eta_z\tau_{zz} \\ \eta_x f_5 + \eta_y g_5 + \eta_z h_5 \end{pmatrix} = |\mathbf{J}|^{-1}Re^{-1}\begin{pmatrix} 0 \\ \begin{pmatrix} \boldsymbol{\tau} \\ \boldsymbol{\kappa}^T \end{pmatrix} \nabla\eta \end{pmatrix} \tag{16}$$

$$\mathbf{H}_\nu = |\mathbf{J}|^{-1}Re^{-1}\begin{pmatrix} 0 \\ \zeta_x\tau_{xx} + \zeta_y\tau_{xy} + \zeta_z\tau_{xz} \\ \zeta_x\tau_{xy} + \zeta_y\tau_{yy} + \zeta_z\tau_{yz} \\ \zeta_x\tau_{xz} + \zeta_y\tau_{yz} + \zeta_z\tau_{zz} \\ \zeta_x f_5 + \zeta_y g_5 + \zeta_z h_5 \end{pmatrix} = |\mathbf{J}|^{-1}Re^{-1}\begin{pmatrix} 0 \\ \begin{pmatrix} \boldsymbol{\tau} \\ \boldsymbol{\kappa}^T \end{pmatrix} \nabla\zeta \end{pmatrix}. \tag{17}$$

The entries of the viscous stress tensor

$$\boldsymbol{\tau} = \begin{pmatrix} \tau_{xx}\tau_{xy}\tau_{xz} \\ \tau_{xy}\tau_{yy}\tau_{yz} \\ \tau_{xz}\tau_{yz}\tau_{zz} \end{pmatrix} \tag{18}$$

are

$$\begin{cases} \tau_{xx} = 2\mu u_x + \lambda(u_x + v_y + w_z) = \lambda(v_y + w_z - 2u_x) \\ \tau_{yy} = 2\mu v_y + \lambda(u_x + v_y + w_z) = \lambda(u_x + w_z - 2v_y) \\ \tau_{zz} = 2\mu w_z + \lambda(u_x + v_y + w_z) = \lambda(u_x + v_y - 2w_z) \\ \tau_{xy} = \mu(u_y + v_x) \\ \tau_{xz} = \mu(u_z + w_x) \\ \tau_{yz} = \mu(w_y + v_z). \end{cases} \tag{19}$$

The subscripts of the entries of the viscous stress tensor signify the index of each element in $\boldsymbol{\tau}$. They do *not* denote partial differentiation, as opposed to the subscripts of the

velocity components, where the indices denote partial differentiation. By the chain rule

$$\left\{ \begin{array}{rcl} u_x &=& \xi_x u_\xi + \eta_x u_\eta + \zeta_x u_\zeta \\ u_y &=& \xi_y u_\xi + \eta_y u_\eta + \zeta_y u_\zeta \\ u_z &=& \xi_z u_\xi + \eta_z u_\eta + \zeta_z u_\zeta \end{array} \right\} \equiv \mathbf{J}''^T \begin{pmatrix} u_\xi \\ u_\eta \\ u_\zeta \end{pmatrix}. \tag{20}$$

Similar expressions are obtained for $\nabla v$, $\nabla w$ and $\nabla a^2$. The effects of heat conduction and the work done by the viscous forces are captured in

$$\boldsymbol{\kappa} = \begin{pmatrix} f_5 \\ g_5 \\ h_5 \end{pmatrix} = \boldsymbol{\tau} \begin{pmatrix} u \\ v \\ w \end{pmatrix} + \frac{k}{Pr \cdot (\gamma - 1)} \nabla a^2 = \boldsymbol{\tau} \begin{pmatrix} u \\ v \\ w \end{pmatrix} + \nabla \hat{a}^2, \tag{21}$$

where

$$\hat{a}^2 = \frac{k}{Pr \cdot (\gamma - 1)} a^2, \tag{22}$$

since $k$, the conductivity coefficient, $Pr$, the Prandtl number, and $\gamma$ are assumed constant in time and space. We have assumed the Stokes condition $3\lambda + 2\mu = 0$ to be valid, where $\mu$ and $\lambda$ are the viscosity coefficients. This condition means that the viscous forces do not exert any normal compressive stress. The speed of sound $a$ is related to pressure and density according to

$$a^2 = \gamma \frac{p}{\rho}, \tag{23}$$

and

$$\hat{a}^2 = \frac{k}{Pr(1 - 1/\gamma)} \frac{p}{\rho}. \tag{24}$$

### 2.2.2 The Boundary

In [25] the in- and outflow boundary conditions are considered in detail. At each point on the inflow boundary, $\zeta = 0$, and the outflow boundary, $\zeta = N_\zeta$, the values of the density, the axial velocity, and the pressure at the new time level, $\rho, w, p$, are determined by solving a 3 by 3 system of equations

$$\begin{pmatrix} a_0^2 & 0 & -1 \\ 0 & \rho_0 a_0 & 1 \\ 0 & -\rho_0 a_0 & 1 \end{pmatrix} \begin{pmatrix} \rho \\ w \\ p \end{pmatrix} = \begin{pmatrix} \phi_3 \\ \phi_4 \\ \phi_5 \end{pmatrix}. \tag{25}$$

The variables $\phi_3, \phi_4$ and $\phi_5$ are the characteristic variables. These variables are determined from the density, the axial velocity, and the pressure at the reference level, $\rho_0$, $w_0$, $p_0$, and the output state vector from the Runge-Kutta routine (57), $\tilde{\rho}, \tilde{w}, \tilde{p}$. Here $\tilde{\rho}, \tilde{w}$ and $\tilde{p}$ represent the values of the density, the axial velocity and the pressure *before* the boundary corrections. Determining the characteristic variables by the equations below leads to a well posed problem [25].

- Subsonic inflow: Compute $\phi_3, \phi_4$ and $\phi_5$ according to:

$$\begin{cases} \phi_3 &= \rho_0 a_0^2 - p_0 \\ \phi_4 &= \rho_0 a_0 w_0 + p_0 \\ \phi_5 &= -\rho_0 a_0 \tilde{w} + \tilde{p} \end{cases}$$

- Subsonic outflow: Compute $\phi_3, \phi_4$ and $\phi_5$ according to:

$$\begin{cases} \phi_3 &= \tilde{\rho} a_0^2 - \tilde{p} \\ \phi_4 &= \rho_0 a_0 \tilde{w} + \tilde{p} \\ \phi_5 &= -\rho_0 a_0 w_0 + p_0 \end{cases}$$

- Supersonic inflow: Compute $\phi_3, \phi_4$ and $\phi_5$ according to:

$$\begin{cases} \phi_3 &= \rho_0 a_0^2 - p_0 \\ \phi_4 &= \rho_0 a_0 w_0 + p_0 \\ \phi_5 &= -\rho_0 a_0 w_0 + p_0 \end{cases}$$

- Supersonic outflow: Compute $\phi_3, \phi_4$ and $\phi_5$ according to:

$$\begin{cases} \phi_3 &= \tilde{\rho} a_0^2 - \tilde{p} \\ \phi_4 &= \rho_0 a_0 \tilde{w} + \tilde{p} \\ \phi_5 &= -\rho_0 a_0 \tilde{w} + \tilde{p} \end{cases}$$

Having computed the characteristic variables according to the above procedure, equation (25) is solved for $\rho, w$ and $p$. For supersonic inflow $\rho = \rho_0, w = w_0$ and $p = p_0$, and for supersonic outflow $\rho = \tilde{\rho}, w = \tilde{w}$ and $p = \tilde{p}$. At the inflow $u = u_0 = v = v_0 = 0$ for both subsonic and supersonic flow.

At the solid walls $u$, $v$ and $w$ are obtained by solving $3 \times 3$ systems of equations. Define

$$\begin{pmatrix} \tilde{U}' \\ \tilde{V}' \\ \tilde{W}' \end{pmatrix} = \begin{pmatrix} x_\xi & y_\xi & z_\xi \\ x_\eta & y_\eta & z_\eta \\ x_\zeta & y_\zeta & z_\zeta \end{pmatrix} \begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{pmatrix}. \tag{26}$$

At $\xi = 0$ and $\xi = N_\xi$ $u$, $v$, and $w$ are obtained by solving the 3 by 3 system

$$\begin{pmatrix} \xi_x & \xi_y & \xi_z \\ x_\eta & y_\eta & z_\eta \\ x_\zeta & y_\zeta & z_\zeta \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} 0 \\ \tilde{V}' \\ \tilde{W}' \end{pmatrix}. \tag{27}$$

At $\eta = 0$ and $\eta = N_\eta$ $u$, $v$, and $w$ are obtained by solving the 3 by 3 system

$$\begin{pmatrix} x_\xi & y_\xi & z_\xi \\ \eta_x & \eta_y & \eta_z \\ x_\zeta & y_\zeta & z_\zeta \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} \tilde{U}' \\ 0 \\ \tilde{W}' \end{pmatrix}. \tag{28}$$

Finally, at the channel edges the 3 by 3 system

$$\begin{pmatrix} \xi_x & \xi_y & \xi_z \\ \eta_x & \eta_y & \eta_z \\ x_\zeta & y_\zeta & z_\zeta \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \tilde{W}' \end{pmatrix} \tag{29}$$

is solved. The above procedure corresponds to setting the normal component of the flow field to zero at the solid walls. For the Euler equations these boundary conditions suffice [25]. For the Navier-Stokes equations the conditions at the solid walls are *nonslip* conditions, i.e., $u = v = w = 0$. For further discussion of boundary conditions for the Navier-Stokes equations see [24].

# 3 Numerical Methods

With the physical domain being a channel, a regular discretization is efficient and a finite difference method a plausible choice of numeric method. A finite difference method with explicit time stepping parallelizes easily and efficiently. Centered spatial difference operators are used in the interior and one-sided operators at the boundaries [9]. Nonlinear phenomena, such as shocks and aliasing, cause numerical instabilities. To remedy this situation numerical dissipation is introduced. It is created through a fourth order difference term (in $\mathbf{q}$), which is turned off near shocks so as not to cause any spurious effects. At shocks we use a second order difference to filter the solution. In this section we define a few dissipation operators that have been used in the simulations. The derivation of the operators, and an analysis of their properties can be found in [25]. Implementation issues are discussed in section 4. For notational convenience the artificial viscosity is discussed for the Euler equations. Artificial viscosity is also needed for the Navier-Stokes equations [25]. We use the same dissipation operators for both sets of equations.

## 3.1 Artificial Viscosity, Interior Points

Artificial viscosity is introduced by adding a term to the analytic equations in the previous section. For the semidiscrete Euler equations and a time independent geometry the equations become

$$\frac{\partial \check{\mathbf{q}}_{jkl}}{\partial \tau} = |\mathbf{J}_{jkl}|[D_0^\xi \mathbf{F}_{jkl} + D_0^\eta \mathbf{G}_{jkl} + D_0^\zeta \mathbf{H}_{jkl} - D_{AV}\check{\mathbf{q}}_{jkl}], \tag{30}$$

where

$$\begin{cases} \mathbf{F}_{jkl} = \mathbf{F}(\mathbf{q}_{jkl}) \\ \mathbf{G}_{jkl} = \mathbf{G}(\mathbf{q}_{jkl}) \\ \mathbf{H}_{jkl} = \mathbf{H}(\mathbf{q}_{jkl}) \end{cases}.$$

The discrete difference operators $D_0^\xi, \Delta_+^\xi$ and $\Delta_-^\xi$ are defined by

$$\left\{ \begin{array}{lcl} D_0^\xi \phi_{jkl} & = & (\phi_{j+1,kl} - \phi_{j-1,kl})/(2\Delta\xi) \\ \Delta_+^\xi \phi_{jkl} & = & \phi_{j+1,kl} - \phi_{jkl} \\ \Delta_-^\xi \phi_{jkl} & = & \phi_{jkl} - \phi_{j-1,kl} \end{array} \right. . \tag{31}$$

The remaining operators are defined analogously. The computational coordinates are chosen such that $\Delta\xi = \Delta\eta = \Delta\zeta = 1$.

For $D_{AV}\check{\mathbf{q}}_{jkl}$ the following three dissipation operators [15, 8, 27] are considered. The operators (33) and (34) are conservative, and (32) is non-conservative

$$D_{AV}\check{\mathbf{q}}_{jkl} = |\mathbf{J}_{jkl}|^{-1}\sigma_{jkl}\{\varepsilon^\xi[\Delta_+^\xi\Delta_-^\xi]^2 + \varepsilon^\eta[\Delta_+^\eta\Delta_-^\eta]^2 + \varepsilon^\zeta[\Delta_+^\zeta\Delta_-^\zeta]^2\}\check{\mathbf{q}}_{jkl} \tag{32}$$

$$\begin{array}{rcl} D_{AV}\check{\mathbf{q}}_{jkl} & = & \Delta_-^\xi[|\mathbf{J}_{jkl}|^{-1}\sigma_{jkl}\varepsilon^\xi\Delta_+^\xi\Delta_-^\xi\Delta_+^\xi\check{\mathbf{q}}_{jkl}] \\ & + & \Delta_-^\eta[|\mathbf{J}_{jkl}|^{-1}\sigma_{jkl}\varepsilon^\eta\Delta_+^\eta\Delta_-^\eta\Delta_+^\eta\check{\mathbf{q}}_{jkl}] \\ & + & \Delta_-^\zeta[|\mathbf{J}_{jkl}|^{-1}\sigma_{jkl}\varepsilon^\zeta\Delta_+^\zeta\Delta_-^\zeta\Delta_+^\zeta\check{\mathbf{q}}_{jkl}] \end{array} \tag{33}$$

$$\begin{array}{rcl} D_{AV}\check{\mathbf{q}}_{jkl} & = & \Delta_-^\xi\Delta_+^\xi[|\mathbf{J}_{jkl}|^{-1}\sigma_{jkl}\varepsilon^\xi\Delta_-^\xi\Delta_+^\xi\check{\mathbf{q}}_{jkl}] \\ & + & \Delta_-^\eta\Delta_+^\eta[|\mathbf{J}_{jkl}|^{-1}\sigma_{jkl}\varepsilon^\eta\Delta_-^\eta\Delta_+^\eta\check{\mathbf{q}}_{jkl}] \\ & + & \Delta_-^\zeta\Delta_+^\zeta[|\mathbf{J}_{jkl}|^{-1}\sigma_{jkl}\varepsilon^\zeta\Delta_-^\zeta\Delta_+^\zeta\check{\mathbf{q}}_{jkl}], \end{array} \tag{34}$$

where

$$\check{\mathbf{q}}_{jkl} = |\mathbf{J}_{jkl}|\mathbf{q}_{jkl} \tag{35}$$

$$\sigma_{jkl} = |U_{jkl}| + |V_{jkl}| + |W_{jkl}| + a_{jkl}(|\nabla\xi_{jkl}|_2 + |\nabla\eta_{jkl}|_2 + |\nabla\zeta_{jkl}|_2) \tag{36}$$

$$\varepsilon^\xi = \varepsilon^\eta = \varepsilon^\zeta = \vartheta_4 \tag{37}$$

with no smoothing at shocks. The constant $\vartheta_4$ has a value of approximately 0.01. For a better treatment of shocks we add one of the following quantities to $D_{AV}$ (the first choice corresponds to a conservative formulation, the second to a non-conservative formulation):

$$-|\mathbf{J}_{jkl}|^{-1}\sigma_{jkl}\{\epsilon_{jkl}^\xi\Delta_+^\xi\Delta_-^\xi + \epsilon_{jkl}^\eta\Delta_+^\eta\Delta_-^\eta + \epsilon_{jkl}^\zeta\Delta_+^\zeta\Delta_-^\zeta\}\check{\mathbf{q}}_{jkl} \tag{38}$$

$$-\Delta_-^\xi[|\mathbf{J}_{jkl}|^{-1}\sigma_{jkl}\epsilon_{jkl}^\xi\Delta_+^\xi\check{\mathbf{q}}_{jkl}] - \Delta_-^\eta[|\mathbf{J}_{jkl}|^{-1}\sigma_{jkl}\epsilon_{jkl}^\eta\Delta_+^\eta\check{\mathbf{q}}_{jkl}] - \Delta_-^\zeta[|\mathbf{J}_{jkl}|^{-1}\sigma_{jkl}\epsilon_{jkl}^\zeta\Delta_+^\zeta\check{\mathbf{q}}_{jkl}]. \tag{39}$$

The coefficient in front of the difference operator in the $\xi$-direction is given by

$$\epsilon_{jkl}^\xi = \vartheta_2 \max(\Upsilon_{j+l,kl}^\xi, \Upsilon_{jkl}^\xi, \Upsilon_{j-1,kl}^\xi) \tag{40}$$

with

$$\Upsilon_{jkl}^\xi = \frac{|p_{j+1,kl} - 2p_{jkl} + p_{j-1,kl}|}{|p_{j+1,kl} + 2p_{jkl} + p_{j-1,kl}|}. \tag{41}$$

A typical value of the constant $\vartheta_2$ is 1/4. The other coefficients are defined analogously. The max-function is used to increase the second order dissipation coefficients near the shock. The fourth order coefficients $\varepsilon^\xi, \varepsilon^\eta, \varepsilon^\zeta$ can no longer be treated as constants. Instead we set

$$\begin{aligned} \varepsilon^\xi &= \varepsilon^\xi_{jkl} = \max(0, \vartheta_4 - \epsilon^\xi_{jkl}) \\ \varepsilon^\eta &= \varepsilon^\eta_{jkl} = \max(0, \vartheta_4 - \epsilon^\eta_{jkl}) \\ \varepsilon^\zeta &= \varepsilon^\zeta_{jkl} = \max(0, \vartheta_4 - \epsilon^\zeta_{jkl}). \end{aligned} \tag{42}$$

Near shocks the pressure gradients are very strong causing the max-function to switch off the fourth order dissipation.

## 3.2 Artificial Viscosity, Boundary Points

On the boundaries the fourth order dissipation operators cannot be used as presented in section 3.1. In the interior applying the fourth order operator $D_4 = [\Delta_+\Delta_-]^2$ to a variable $\phi$ yields

$$D_4\phi_j = [\Delta_+\Delta_-]^2\phi_j = \phi_{j-2} - 4\phi_{j-1} + 6\phi_j - 4\phi_{j+1} + \phi_{j+2}. \tag{43}$$

In matrix notation, the most general form of the fourth order dissipation operator, subject to the constraint that the bandwidth be constant is

$$\mathbf{D}_4 = \begin{pmatrix} \alpha_0 & \alpha_1 & \alpha_2 & & & & \\ \beta_0 & \beta_1 & \beta_2 & \beta_3 & & & \\ 1 & -4 & 6 & -4 & 1 & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & 1 & -4 & 6 & -4 & 1 \\ & & & \gamma_{N-3} & \gamma_{N-2} & \gamma_{N-1} & \gamma_N \\ & & & & \delta_{N-2} & \delta_{N-1} & \delta_N \end{pmatrix}, \tag{44}$$

The restriction of a constant bandwidth is made for computational efficiency, which will become clear in section 4.6.1. In [25] we show that the dissipation operator [8]

$$\mathbf{D}_4 = \begin{pmatrix} 1 & -2 & 1 & & & & \\ -2 & 5 & -4 & 1 & & & \\ 1 & -4 & 6 & -4 & 1 & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & 1 & -4 & 6 & -4 & 1 \\ & & & 1 & -4 & 5 & -2 \\ & & & & 1 & -2 & 1 \end{pmatrix} \tag{45}$$

preserves the zeroth and first order moments, the best possible if we want the operator to be positive semidefinite. Since the choice of artificial viscosity affects the rate of

convergence [27], it is of interest to study the impact of a positive definite operator

$$\tilde{\mathbf{D}}_4 = \begin{pmatrix} 5 & -4 & 1 & & & & \\ -4 & 6 & -4 & 1 & & & \\ 1 & -4 & 6 & -4 & 1 & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & 1 & -4 & 6 & -4 & 1 \\ & & & 1 & -4 & 6 & -4 \\ & & & & 1 & -4 & 5 \end{pmatrix}. \tag{46}$$

The operator (46) is obtained by deleting the first and last rows and columns of (45). Deleting these rows and columns is equivalent to prescribing Dirichlet boundary conditions.

We also include a non-symmetric fourth order dissipation operator

$$\bar{\mathbf{D}}_4 = \begin{pmatrix} 0 & 0 & 0 & & & & \\ -1 & 3 & -3 & 1 & & & \\ 1 & -4 & 6 & -4 & 1 & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & 1 & -4 & 6 & -4 & 1 \\ & & & 1 & -3 & 3 & -1 \\ & & & & 0 & 0 & 0 \end{pmatrix}. \tag{47}$$

This operator is indefinite and preserves only zeroth order moments.

Different implementations of the operator $\mathbf{D}_4$ have different characteristics with respect to computational efficiency, and numeric stability. We evaluate these properties by considering the unfactored operator $D_4$ and the factorizations

$$\begin{aligned} \mathbf{D}_4 &= \mathbf{D}_1^- \mathbf{D}_3^+ \\ \mathbf{D}_4 &= \mathbf{D}_1^+ \mathbf{D}_3^- \\ \mathbf{D}_4 &= \mathbf{D}_1^- \mathbf{D}_1^+ \mathbf{D}_2 \\ \mathbf{D}_4 &= \mathbf{D}_1^+ \mathbf{D}_1^- \mathbf{D}_2, \end{aligned}$$

where

$$\mathbf{D}_1^- = \begin{pmatrix} 1 & & & & & & \\ -1 & 1 & & & & & \\ & -1 & 1 & & & & \\ & & \ddots & \ddots & & & \\ & & & -1 & 1 & & \\ & & & & -1 & 1 & \\ & & & & & -1 & 1 \end{pmatrix} \tag{48}$$

$$\mathbf{D}_1^+ = \begin{pmatrix} 1 & -1 & & & & & \\ & 1 & -1 & & & & \\ & & 1 & -1 & & & \\ & & & \ddots & \ddots & & \\ & & & & 1 & -1 & \\ & & & & & 1 & -1 \\ & & & & & & 1 \end{pmatrix} \tag{49}$$

$$\mathbf{D}_2 = \begin{pmatrix} 0 & 0 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & 0 & 0 \end{pmatrix} \tag{50}$$

$$\mathbf{D}_3^- = \begin{pmatrix} 0 & 0 & & & & & \\ -1 & 2 & -1 & & & & \\ 1 & -3 & 3 & -1 & & & \\ & \ddots & \ddots & \ddots & \ddots & & \\ & & 1 & -3 & 3 & -1 & \\ & & & 1 & -3 & 3 & -1 \\ & & & & 1 & -2 & 1 \end{pmatrix} \tag{51}$$

$$\mathbf{D}_3^+ = \begin{pmatrix} 1 & -2 & 1 & & & & \\ -1 & 3 & -3 & 1 & & & \\ & -1 & 3 & -3 & 1 & & \\ & & \ddots & \ddots & \ddots & \ddots & \\ & & & -1 & 3 & -3 & 1 \\ & & & & -1 & 2 & -1 \\ & & & & & 0 & 0 \end{pmatrix}. \tag{52}$$

(53)

Note that $\mathbf{D}_1^{+T} = \mathbf{D}_1^-$. Using these factorizations the total artificial viscosity operator in one space dimension is factored as

$$\mathbf{D}_1^- \boldsymbol{\Sigma} \mathbf{D}_3^+ \tag{54}$$

$$\mathbf{D}_1^+ \boldsymbol{\Sigma} \mathbf{D}_3^- \tag{55}$$

$$\mathbf{D}_1^- \mathbf{D}_1^+ \boldsymbol{\Sigma} \mathbf{D}_2, \tag{56}$$

respectively, where

$$\boldsymbol{\Sigma} = \mathrm{diag}(\varepsilon_0 \sigma_0 |\mathbf{J}_0|^{-1}, \ldots, \varepsilon_N \sigma_N |\mathbf{J}_N|^{-1}).$$

Here $\sigma_j, \varepsilon_j$ and $|\mathbf{J}_j|^{-1}$ are the one-dimensional equivalents of (36), (42) and (2).

The first factorization corresponds to splitting the fourth order operator $[\Delta_-\Delta_+]^2$ into $\Delta_-$ and $\Delta_+\Delta_-\Delta_+$. Since $\Delta_+$ and $\Delta_-$ commute $\Delta_+$ and $\Delta_-\Delta_+\Delta_-$ is an alternate splitting, which is used in the second case. The different operators are summarized in Table 1.

| Dissipation Operator | Definiteness | Preservation of Moments | Dimension of Null Space |
|---|---|---|---|
| $\mathbf{D}_4$ | positive semidefinite | $\leq$ first order | 2 |
| $\tilde{\mathbf{D}}_4$ | positive definite | none | 0 |
| $\bar{\mathbf{D}}_4$ | indefinite | zeroth order | 3 |
| $\mathbf{D}_1^- \boldsymbol{\Sigma} \mathbf{D}_3^+$ | | | 2 |
| $\mathbf{D}_1^+ \boldsymbol{\Sigma} \mathbf{D}_3^-$ | | | 2 |
| $\mathbf{D}_1^- \mathbf{D}_1^+ \boldsymbol{\Sigma} \mathbf{D}_2$ | positive semidefinite | $\leq$ first order | 2 |

Table 1: Properties of the Dissipation Operators

## 3.3 Time Discretization

We use a three stage Runge-Kutta method for the time integration of the semidiscrete equations. The integration method belongs to a class of algorithms that can be written

$$\begin{cases} \check{\mathbf{q}}_{jkl}^0 &= \check{\mathbf{q}}_{jkl}^n \\ \check{\mathbf{q}}_{jkl}^1 &= \check{\mathbf{q}}_{jkl}^0 + \alpha_1 \Delta\tau \mathbf{R}(\check{\mathbf{q}}_{jkl}^0) \\ &\vdots \\ \check{\mathbf{q}}_{jkl}^m &= \check{\mathbf{q}}_{jkl}^0 + \alpha_m \Delta\tau \mathbf{R}(\check{\mathbf{q}}_{jkl}^{m-1}) \\ \check{\mathbf{q}}_{jkl}^{n+1} &= \check{\mathbf{q}}_{jkl}^m \end{cases} . \tag{57}$$

The vector $\mathbf{R}$ is the right member of an equation of the same type as equation (30). For a three stage Runge-Kutta method $m = 3$. We have used $\alpha_1 = \alpha_2 = \alpha_3 = 1$. In [25] we derive the stability condition

$$\Delta\tau \leq \frac{CFL}{|U| + |V| + |W| + a(|\nabla\xi|_2 + |\nabla\eta|_2 + |\nabla\zeta|_2)} \equiv \frac{CFL}{\sigma}. \tag{58}$$

for Euler flows without the dissipation operator. *CFL* is the Courant-Friedrichs-Lewy number. In general, $\sigma$ varies in space, and the stability criterion above can be used to determine the local time step, for time independent problems. The fourth order artificial viscosity slightly shifts the spectrum of the discrete space operator into the left half of the complex plane. From Figure 2 it is evident that the shifted spectrum will remain in the stability region of the three stage Runge-Kutta method, if the viscosity coefficient is sufficiently small.

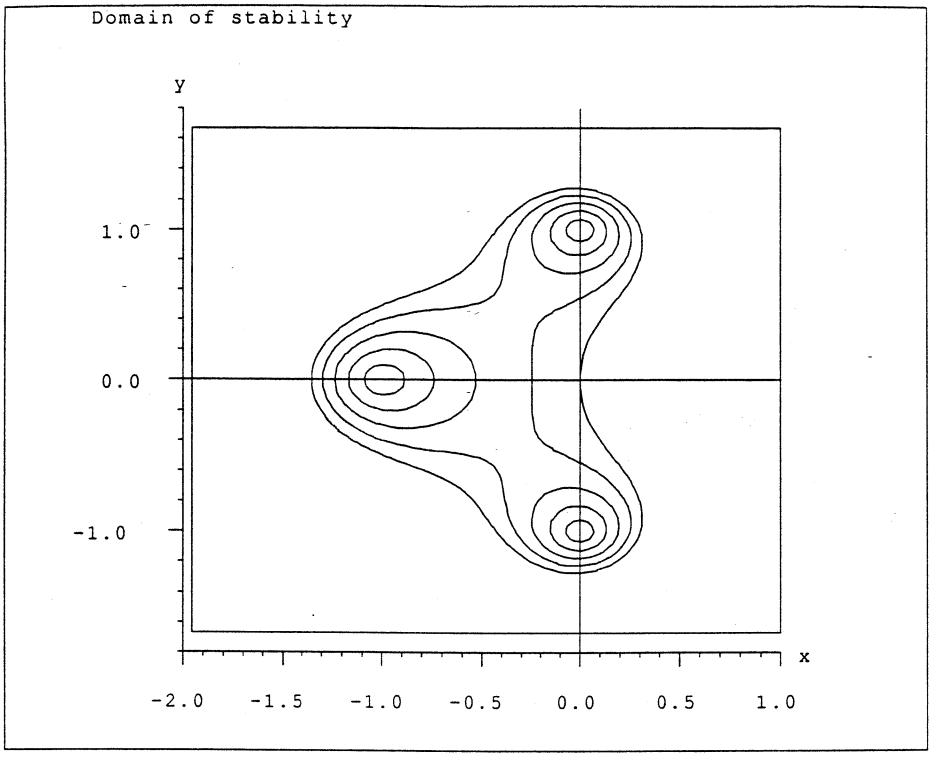


Figure 2: Stability Region of the Three Stage Runge Kutta Method

## 3.4 Numerical Methods, Summary

In summary, the Euler flow equations are based on the twisted grid as defined by equations (4), (5) and (6) with $\Xi = 0.0001$, and the semidiscrete equations

$$\begin{aligned}\frac{\partial \check{\mathbf{q}}_{jkl}}{\partial \tau} &= |\mathbf{J}_{jkl}|[D_0^\xi \mathbf{F}_{jkl} + D_0^\eta \mathbf{G}_{jkl} + D_0^\zeta \mathbf{H}_{jkl}] \\ &- \sigma_{jkl}\{\varepsilon^\xi_{jkl}[\Delta^\xi_+\Delta^\xi_-]^2 + \varepsilon^\eta_{jkl}[\Delta^\eta_+\Delta^\eta_-]^2 + \varepsilon^\zeta_{jkl}[\Delta^\zeta_+\Delta^\zeta_-]^2\}\check{\mathbf{q}}_{jkl} \\ &+ \sigma_{jkl}\{\epsilon^\xi_{jkl}\Delta^\xi_+\Delta^\xi_- + \epsilon^\eta_{jkl}\Delta^\eta_+\Delta^\eta_- + \epsilon^\zeta_{jkl}\Delta^\zeta_+\Delta^\zeta_-\}\check{\mathbf{q}}_{jkl}\end{aligned} \tag{59}$$

in non-conservative form and

$$\begin{aligned}\frac{\partial \check{\mathbf{q}}_{jkl}}{\partial \tau} &= |\mathbf{J}_{jkl}|\{D_0^\xi \mathbf{F}_{jkl} + D_0^\eta \mathbf{G}_{jkl} + D_0^\zeta \mathbf{H}_{jkl} \\ &- \Delta^\xi_-[|\mathbf{J}_{jkl}|^{-1}\sigma_{jkl}\varepsilon^\xi_{jkl}\Delta^\xi_+\Delta^\xi_-\Delta^\xi_+\check{\mathbf{q}}_{jkl}] \\ &- \Delta^\eta_-[|\mathbf{J}_{jkl}|^{-1}\sigma_{jkl}\varepsilon^\eta_{jkl}\Delta^\eta_+\Delta^\eta_-\Delta^\eta_+\check{\mathbf{q}}_{jkl}] \\ &- \Delta^\zeta_-[|\mathbf{J}_{jkl}|^{-1}\sigma_{jkl}\varepsilon^\zeta_{jkl}\Delta^\zeta_+\Delta^\zeta_-\Delta^\zeta_+\check{\mathbf{q}}_{jkl}] \\ &+ \Delta^\xi_-[|\mathbf{J}_{jkl}|^{-1}\sigma_{jkl}\epsilon^\xi_{jkl}\Delta^\xi_+\check{\mathbf{q}}_{jkl}] \\ &+ \Delta^\eta_-[|\mathbf{J}_{jkl}|^{-1}\sigma_{jkl}\epsilon^\eta_{jkl}\Delta^\eta_+\check{\mathbf{q}}_{jkl}] \\ &+ \Delta^\zeta_-[|\mathbf{J}_{jkl}|^{-1}\sigma_{jkl}\epsilon^\zeta_{jkl}\Delta^\zeta_+\check{\mathbf{q}}_{jkl}]\}\end{aligned} \tag{60}$$

or

$$\frac{\partial \check{\mathbf{q}}_{jkl}}{\partial \tau} = |\mathbf{J}_{jkl}|\{D_0^\xi \mathbf{F}_{jkl} + D_0^\eta \mathbf{G}_{jkl} + D_0^\zeta \mathbf{H}_{jkl}$$

$$
\begin{aligned}
&- \Delta_-^\xi \Delta_+^\xi [|\mathbf{J}_{jkl}|^{-1} \sigma_{jkl} \varepsilon_{jkl}^\xi \Delta_-^\xi \Delta_+^\xi \check{\mathbf{q}}_{jkl}] \\
&- \Delta_-^\eta \Delta_+^\eta [|\mathbf{J}_{jkl}|^{-1} \sigma_{jkl} \varepsilon_{jkl}^\eta \Delta_-^\eta \Delta_+^\eta \check{\mathbf{q}}_{jkl}] \\
&- \Delta_-^\zeta \Delta_+^\zeta [|\mathbf{J}_{jkl}|^{-1} \sigma_{jkl} \varepsilon_{jkl}^\zeta \Delta_-^\zeta \Delta_+^\zeta \check{\mathbf{q}}_{jkl}] \\
&+ \Delta_-^\xi [|\mathbf{J}_{jkl}|^{-1} \sigma_{jkl} \epsilon_{jkl}^\xi \Delta_+^\xi \check{\mathbf{q}}_{jkl}] \\
&+ \Delta_-^\eta [|\mathbf{J}_{jkl}|^{-1} \sigma_{jkl} \epsilon_{jkl}^\eta \Delta_+^\eta \check{\mathbf{q}}_{jkl}] \\
&+ \Delta_-^\zeta [|\mathbf{J}_{jkl}|^{-1} \sigma_{jkl} \epsilon_{jkl}^\zeta \Delta_+^\zeta \check{\mathbf{q}}_{jkl}]\}
\end{aligned} \tag{61}
$$

in conservative form. The reciprocal cell volume at point $jkl$ is computed from equation (1), $\mathbf{F}_{jkl}$, $\mathbf{G}_{jkl}$ and $\mathbf{H}_{jkl}$ are given by equations (10), (11) and (12). The vector $\check{\mathbf{q}}_{jkl}$ is given by equation (35), $\sigma_{jkl}$ by equation (36), $\varepsilon_{jkl}$ by equation (42), $\epsilon_{jkl}$ by equation (40), and the difference operators for interior and boundary points are given by equation (31). For boundary points higher order difference operators are modified as defined by one of equations (45), (46) or (47) and by (50). For the conservative form the fourth order operator equations (48) and (52) are used. The in- and outflow boundary conditions are defined by equations (25). At the solid boundaries the normal component of the velocity field is zero according to equations (27), (28) and (29).

For the Navier-Stokes equations $\mathbf{F}_{jkl}$, $\mathbf{G}_{jkl}$, and $\mathbf{H}_{jkl}$ are replaced by $\mathbf{F}_{jkl}+\mathbf{F}_{\nu_{jkl}}$, $\mathbf{G}_{jkl}+\mathbf{G}_{\nu_{jkl}}$, and $\mathbf{H}_{jkl}+\mathbf{H}_{\nu_{jkl}}$, respectively, where $\mathbf{F}_{\nu_{jkl}}$, $\mathbf{G}_{\nu_{jkl}}$, and $\mathbf{H}_{\nu_{jkl}}$ are given by equations (15), (16), and (17). The stretching parameter $\Xi$ equals 1.2. Artificial viscosity is required even when solving the Navier-Stokes equations. The second derivatives are approximated by twice applying the operators $D_0^\xi$, $D_0^\eta$, and $D_0^\zeta$, rather than by the standard operators $D_-^\xi D_+^\xi$ and so forth. From equations (15), (16), (17), (19), (20) and (31) it follows that the viscous flux vectors contain centered second order accurate approximations of the velocity gradients. The inviscid and viscous flux vectors are added and the accumulated flux vectors differentiated (equation (8)). Implicitly, $u_{\xi\xi}, u_{\eta\eta}$, etc. are computed using the centered operator $D_0$. This procedure minimizes the number of numerical differentiations, and thereby reduces the number of arithmetic operations. At the solid walls $u = v = w = 0$ and the pressure and density are extrapolated. At the in/outflow we use the same boundary conditions as for the Euler equations, since viscous effects are assumed to be negligible except in boundary layers near solid walls.

# 4 Data Parallel Programming

## 4.1 Arrays and virtual processors

Architectures in which tens of thousands of operations can be performed concurrently are often referred to as *data parallel* to distinguish them from *control parallel* architectures, which offer a considerably lower degree of concurrency. Algorithms are designed based on the structure and representation of the problem domain. Objects in data parallel

languages are represented by higher level data types such as the array extensions of Fortran 8X [23]. In a language with an array syntax, a number of nested loops (often equal to the number of axes in the array) disappear from the code, compared to a language without the array syntax. We illustrate this property by the computation of a 7-point stencil in three dimensions.

```
      SUBROUTINE PSOLVE(PHI, OMEGA, N, ITER)
      REAL PHI(N, N, N), OMEGA(N, N, N), FACTOR
      FACTOR = 1.0/6.0
      DO 100 I=1,ITER,1
        PHI = FACTOR * (
     $     CSHIFT(PHI,DIM=1,SHIFT=-1) +
     $     CSHIFT(PHI,DIM=2,SHIFT=-1) +
     $     CSHIFT(PHI,DIM=3,SHIFT=-1) +
     $     CSHIFT(PHI,DIM=1,SHIFT= 1) +
     $     CSHIFT(PHI,DIM=2,SHIFT= 1) +
     $     CSHIFT(PHI,DIM=3,SHIFT= 1)) +
     $     OMEGA
  100 CONTINUE
      RETURN
      END
```

The operation CSHIFT defines a circular shift. The first argument is the variable to which the shift is applied, the second defines the axis along which the shift takes place, and the third argument defines the length and direction of the shift. Since there is no conditional statement in the code, the difference stencil is applied at every interior and boundary point. The code implements periodic boundary conditions. Note that there are no explicit loops enumerating grid points along the three array axes.

In the data parallel programming model it is often convenient to define elementary objects subject to the same set of transformations. These objects are referred to as *virtual processors* [12]. For the Navier-Stokes, and Euler, equations the three-dimensional Cartesian topology of the computational lattice is one of the basic characteristics of the data structure representing the fluid, and the state of the flow. A grid point, its state, and the part of the problem description associated with it are chosen as a *virtual processor*, or a *virtual processor set* in the Connection Machine programming terminology.

The data parallel programming model is particularly effective when the same operation is applied to a large number of virtual processors. A single instruction suffices to apply a given operation to all virtual processors that are subject to the same computations. Virtual processors are assigned to *physical processors*. In the current implementation on the Connection Machine system CM–2, the number of virtual processors and the number of physical processors are both powers of two. The ratio between the number of virtual and physical processors is known as the *virtual processor ratio*. Every physical processor is assigned this many virtual processors. The total number of virtual processors is limited by the size of the primary storage, and the storage required by each virtual processor. The virtual processor concept makes the size of the physical machine transparent to the programmer. Only the size of the physical storage matters. The scheduling of virtual processors on physical processors is also transparent to the programmer.

| Computation | Registers only | 4 Mbit chips | 256 4 Mbit chips (board) | 256 Boards |
|---|---|---|---|---|
| Mtx mpy | 0.5 | 104 | 1600 | 26000 |
| 3-d Relaxation | 0.17 | 4.27 | 26.7 | 170.7 |
| FFT | 1 | 18.8 | 28.8 | 38.8 |

Table 2: Number of operations per remote reference of a single variable.

| Computation | 4 Mbit 1 proc. 1 chip | 256 Procs. = Board | 256 boards = Machine |
|---|---|---|---|
| Mtx mpy | 1 | 10 | 160 |
| 3-d relaxation | 32 | 480 | 24600 |
| FFT | 3 | 1140 | 160000 |
| no locality | 300 | 76800 | 19660800 |

Table 3: Number of bits across the chip/board/system boundary per cycle.

Data parallel architectures achieve supercomputer performance through a large number of processing units, and a large number of memory modules. Supercomputers with a performance of 1 Tflops/s or more, requires a memory bandwidth of several Tbytes/s, which can be realized by thousands of processors and thousands of memory units. It is feasible to build such computers in state-of-the-art technology [16, 5]. A network interconnecting the processors and the memory modules is required to achieve a sufficient data motion capability. The most critical property of the technology with respect to sustained performance is packaging. The technology for packing integrated circuits, and connecting printed circuit boards, is such that the communication bandwidth on a chip is about two orders of magnitude higher than the communication bandwidth at the chip boundary, and the bandwidth on a board about two orders of magnitude higher than the bandwidth at the board boundary. A sustained performance close to the peak performance is only possible if locality of reference exists in the problem, and can be exploited in the implementation. This fact is illustrated by Tables 2 and 3 [18]. Table 3 gives the number of bits that on the average must cross the chip, board, and system boundaries during a single cycle, assuming the optimum locality, or no locality of reference. It is assumed that each chip has one processing unit, that a board has 256 processing units, and that all variables are in single precision.

Next we address the issues of load balance, and the efficient use of the communication system in the particular case of the Connection Machine system model CM–2 for the model problem. The techniques are generally applicable to data parallel architectures.

## 4.2 The Connection Machine Architecture

The Connection Machine [12] is a data parallel architecture. It has a total primary storage of 512 Mbytes using 256 kbit memory chips, and 2 Gbytes with 1 Mbit memory chips. The data transfer rate to storage is approximately 45 Gbytes/s at a clock rate of 7 MHz. The primary storage has 64k ports and a simple 1-bit processor for each port. The storage per processor is 8 kbytes for a total storage of 512 Mbytes and 32 kbytes with 1 Mbit memory chips. The Connection Machine model CM–2 can be equipped with hardware for floating-point arithmetic. With the floating-point option, 32 Connection Machine processors share a floating-point unit, which is an industry standard, single chip floating-point multiplier and adder with a few registers. The peak performance available from the standard instruction set and the higher level languages is in the range 1.5 Gflops/s - 2.2 Gflops/s. The higher level languages do not at the present time make efficient use of the registers in the floating-point unit for operations that vectorize. With optimum use of the registers, a performance that is one order of magnitude higher is possible. For instance, for large local matrices, a peak performance in excess of 25 Gflops/s has been measured.

The Connection Machine needs a host computer. Currently, three families of host architectures are supported: the VAX family with the BI-bus, SUN 4, and the Symbolics 3600 series. The Connection Machine memory is mapped into the address space of the host. The program code resides in the storage of the host. It fetches the instructions, does the complete decoding of scalar instructions, and executes them. Instructions to be applied to variables in the Connection Machine are sent to a microcontroller, which decodes and executes instructions for the Connection Machine. Variables defined by array constructs are allocated to the Connection Machine, unless allocation on the front-end is requested. The architecture is depicted in Figure 3. The Connection Machine can also be equipped with a secondary storage system known as the data vault. There exist 8 I/O channels, each with a block transfer rate of up to approximately 30 Mbytes/s. The size of the secondary storage system is in the range 5 Gbytes to 640 Gbytes. The Connection Machine can also be equipped with a frame buffer for fast high resolution graphics. An update rate of about 15 frames per second can be achieved.

The Connection Machine processors are organized with 16 processors to a chip, and the chips interconnected as a 12-dimensional Boolean cube. The communication is bit-serial and pipelined. Concurrent communication on all ports is possible. Through the bit-serial pipelined operation of the communication system, remote processor references require no more time than nearest neighbor references provided there is no contention for communication channels. For communication in arbitrary patterns, the Connection Machine is equipped with a router which selects one of the shortest paths between source and destination, unless all of these paths are occupied. The router has several options for resolving contention for communication channels.

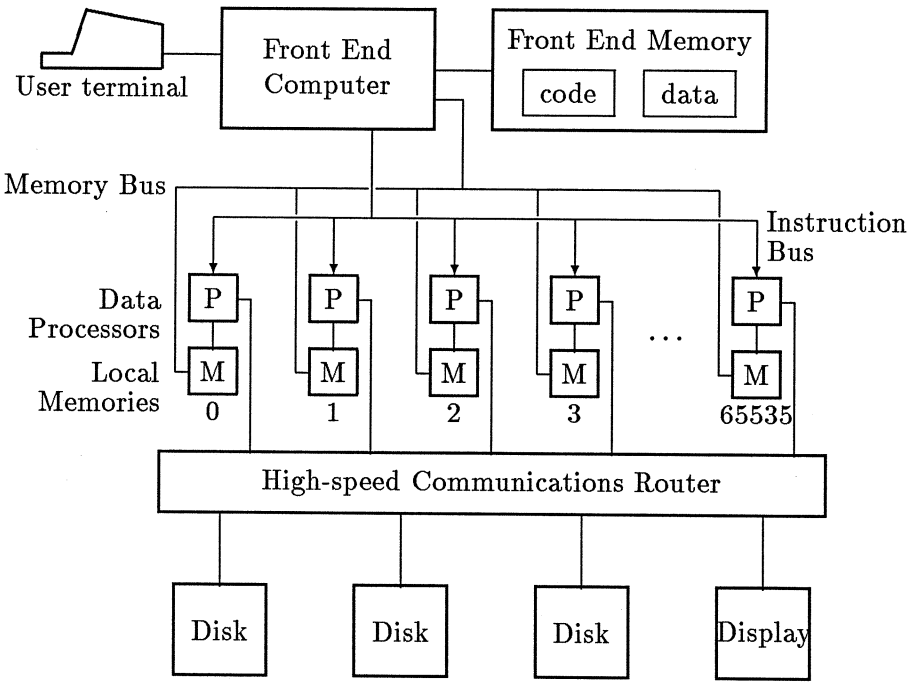


Figure 3: The Connection Machine System

## 4.3 Configuring the address space

The address field of the Connection Machine is divided into three parts: (off-chip|on-chip|memory). The off-chip field consists of 12 bits that encode the Connection Machine processor chips, the on-chip field encodes the 16 processors on each Connection Machine processor chip, and the lower order bits encode the memory addresses local to a processor. The lowest order off-chip bit encodes pairs of processor chips sharing a floating-point unit. On-chip communication is considerably faster than inter-chip communication. On-chip communication is a local memory reference. Off-chip communication is slower due to the limited bandwidth at the chip boundary. The non-uniformity in access time impacts the optimum data allocation [17, 18].

The default data allocation scheme on the Connection Machine first determines how many data elements there are to each processor, then stores that many successive elements in each processor, *consecutive storage* [17]. With the $n$ highest order bits encoding the processors and the lower order bits encoding memory addresses in each processor, the consecutive assignment can be illustrated as follows:

*Consecutive* assignment: $(\underbrace{x_{m-1}x_{m-2}\ldots x_{m-n}}_{rp}\underbrace{x_{m-n-1}x_{m-n-2}\ldots x_0}_{vp})$.

The field denoted $rp$ encodes *real processor* addresses as opposed to *local memory* addresses $vp$. For a data set of $M = 2^m$ real numbers, $m$ address bits are required, $n$ of which are processor address bits. Hence, there are $m - n$ local memory address bits. In *cyclic* assignment the lowest order address bits determine the *real processor* address.

$$\textit{Cyclic} \text{ assignment:} \quad (\underbrace{x_{m-1}x_{m-2}\ldots x_n}_{vp}\underbrace{x_{n-1}x_{n-2}\ldots x_0}_{rp}).$$

In the cyclic assignment, all data elements in a processor have the same $n$ low order bits. In the consecutive assignment, the elements in a processor have the same $n$ high order bits. The cyclic allocation scheme currently is not supported on the Connection Machine system. However, for some computations, it offers certain performance advantages [17, 20].

Current implementations of the Connection Machine languages encode each axis of a multi-dimensional array separately. Each axis is extended to a length that is equal to some power of two. For an axis length $N$, $\lceil \log_2 N \rceil$ address bits are assigned to the encoding of the elements along that axis. The consecutive allocation scheme is used for each axis. The encoding of the axes in the total address space attempts to configure each part of the address space (off-chip, on-chip, and memory) to conform with the array. To the extent possible, all axes have a segment of each address field, and the ratio of the lengths of segments for different axes is the same as that of the length of the axes. The address space of an $N_1 \times N_2 \times \ldots \times N_d$ array is configured as

$$(\overbrace{\underbrace{x_{|\gamma|+|\beta|+|\alpha|-1}\cdots x_{|\beta|+|\alpha|}}_{rp}}^{\text{off-chip}}\overbrace{\underbrace{x_{|\beta|+|\alpha|-1}\cdots x_{|\alpha|}}_{rp}}^{\text{on-chip}}\overbrace{\underbrace{x_{|\alpha|-1}\cdots x_0}_{vp}}^{\text{mem}}),$$

where $\alpha_i + \beta_i + \gamma_i = \lceil \log_2 N_i \rceil$ with

$$\frac{\alpha_i}{\alpha_j} = \frac{\beta_i}{\beta_j} = \frac{\gamma_i}{\gamma_j} = \frac{N_i}{N_j}$$

whenever possible. Using standard multi-index notation, $|\alpha|$ is defined as $|\alpha| = \alpha_1 + \ldots + \alpha_d$. Analagous expressions hold for $|\beta|$ and $|\gamma|$.

For a computation in which the interaction between virtual processors is equally frequent in each direction, the total amount of communication is minimized if the virtual processors assigned to a physical processor, or actually a processor chip, forms a single subdomain with an aspect ratio as close to one as possible [18]. The different Connection Machine languages provide different means for user controlled data allocation. In CM-Fortran, compiler directives allow a user to specify an axis as **SERIAL**, which implies that the axis is allocated to a single processor. In PARIS (PARallel Instruction Set), the Connection Machine native language, a user has full control over what dimensions of the

address space an axis occupies. But, only consecutive allocation of data to processors is supported.

If an array has fewer elements than the number of *real* processors in the configuration, the array is extended such that there is one element per real processor. In CM-Fortran an axis is added to the array with a length equal to the number of instances of the specified array that matches the number of real processors.

## 4.4 Encoding of array axes

In the common binary encoding, successive integers may differ in an arbitrary number of bits. For instance, 63 and 64 differs in 6 bits, and hence are at a *Hamming* distance of 6 in the Boolean cube. A *Gray* code by definition has the property that successive integers differ in precisely one bit. The most frequently used Gray code for the embedding of arrays in Boolean cubes is a *binary-reflected* Gray code [17, 21, 28]. This Gray code is periodic. The code preserves adjacency for any loop (periodic one-dimensional lattice) of even length, and for loops of odd length one edge in the loop is mapped into a path of length two [17]. For the embedding of multi-dimensional arrays, each axis may be encoded by the *binary-reflected* Gray code. The embedding of an $N_1 \times N_2 \times \ldots \times N_d$ array requires $\sum_{i=1}^{d} \lceil \log_2 N_i \rceil$ bits. The *expansion*, i.e., the ratio between the consumed address space and the actual array size, is $2^{\sum_{i=1}^{d} \lceil \log_2 N_i \rceil} / \Pi_{i=1}^{d} N_i$, which may be as high as $\sim 2^d$ [10, 13]. The expansion can be reduced by allowing some successive array indices to be encoded at a Hamming distance of two. The *dilation* is the maximum Hamming distance between any pair of adjacent array indices. Every two-dimensional array can be embedded with minimum expansion and dilation 2 [3]. Minimum expansion dilation 2 embeddings for a large class of two-dimensional arrays are given in [13], which also provides a technique for reducing the expansion of higher dimensional arrays. Minimal expansion dilation 7 embeddings are possible for all three-dimensional arrays [4]. Embeddings with dilation 2 for many three-dimensional arrays are given in [14].

The lattice emulation by a binary-reflected Gray code embedding is part of the standard programming environment on the Connection Machine system. In CM-Fortran, array axes are by default encoded in a binary-reflected Gray code for the off-chip segment of the address field. In the other Connection Machine languages, the Gray code encoding is invoked by configuring the Connection Machine as a lattice of the appropriate number of dimensions. The benefit of the lattice emulation feature is twofold: the virtual processors are assigned to physical processors such that the communication requirements are minimized, and lattice organized computations are often easier to express by virtue of programming constructs corresponding directly to the operations in the problem domain.

## 4.5 Programming Languages

The data parallel languages on the Connection Machine currently are CM-Fortran, *Lisp, and C*. These languages are extensions of the well known languages for sequential machines. In all three languages the extensions effectively consist of higher level data types, and the operations associated with them. In the case of CM-Fortran the additional data types are the array extensions of Fortran 8X [23]. C* [2] is derived from C++, and the additional data types named **poly** serve the same purpose as the array extensions in CM-Fortran. Objects of the same type are members of the same **domain**. Referencing a domain implies a subselection of virtual processors. Every instance of a domain is associated with a virtual processor. In *Lisp [1] the corresponding data types are **pvars** for parallel variables. Parallel variables with the same layout form a virtual processor set. The higher level data types define collections of variables assigned to the Connection Machine system. Scalar data are assigned to the memory of the host, together with the program code.

The data parallel languages on the Connection Machine system include instructions for reduction, copy/broadcast, and scan operations. These operations can be performed both on all virtual processors, or concurrently on different sets of virtual processors forming *segments*. A segment consists of a set of virtual processors with contiguous addresses. Different segments have disjoint sets of addresses. Virtual processors can be subselected within segments, such that a given operation only applies to the selected set. Below, we give a few examples of data parallel programming.

### 4.5.1 CM-Fortran

The default encoding of array axes in CM-Fortran is by a binary-reflected Gray code on the part of the address defining Connection Machine processor chips. As an illustration of the use of array operations and layout directives we show a code fragment from a Navier-Stokes solver with periodic boundary conditions. The implementation of non-periodic boundary conditions will be discussed in section (4.6.1). The piece of code shown below illustrates how the inviscid flux vectors, equations (10), (11), (12), and the viscous flux vectors, equations (15), (16), (17), are accumulated and differentiated. At each grid point the inviscid flux vectors make up a 5 by 3 array, which is stored in IFLUX. Similarly, VFLUX holds the viscous flux vectors. The result is stored in RQ, which represents the right member of equation (30) before adding the artificial viscosity. The loop over the three lattice axes of the array are unrolled. The expression within the parenthesis defines a convolution operator for which all communication can be executed concurrently.

```
CMF$LAYOUT IFLUX(:SERIAL, :SERIAL, , , ), VFLUX(:SERIAL, :SERIAL, , , ) RQ(:SERIAL, , , )
REAL IFLUX(5,3,32,32,32), VFLUX(5,3,32,32,32), R(5,32,32,32)
R = 0.0
DO I=1,5
  R(I,:,:,:) = R(I,:,:,:) + 0.5*(CSHIFT(IFLUX(I,1,:,:,:) + VFLUX(I,1,:,:,:),1, 1) -
                                 CSHIFT(IFLUX(I,1,:,:,:) + VFLUX(I,1,:,:,:),1,-1) +
                                 CSHIFT(IFLUX(I,2,:,:,:) + VFLUX(I,2,:,:,:),2, 1) -
                                 CSHIFT(IFLUX(I,2,:,:,:) + VFLUX(I,2,:,:,:),2,-1) +
                                 CSHIFT(IFLUX(I,3,:,:,:) + VFLUX(I,3,:,:,:),3, 1) -
                                 CSHIFT(IFLUX(I,3,:,:,:) + VFLUX(I,J,:,:,:),3,-1))
END DO I
```

In the above code segment, it should be noted that the grid spacing equals 0.5 in each dimension.

CM-Fortran provides a second grid communication scheme:

EOSHIFT(ARRAY, DIM, SHIFT, BOUNDARY).

Function **EOSHIFT** is used to shift off all rank-one sections of **ARRAY** in the dimension specified by **DIM** at one end. At the other end copies of a boundary matrix **BOUNDARY** are shifted in. If the rank of **ARRAY** is $n$, then the rank of **BOUNDARY** must be $n-1$ or 0, i.e., a scalar. The usage of **EOSHIFT** provides for very efficient implementation of boundary conditions of Dirichlet type, whereas **CSHIFT** implements periodic boundary conditions.

It is possible to order the virtual processors by their send addresses. This is accomplished through the layout directive **:SEND**. The default option is **:NEWS**, which corresponds to the Cartesian grid topology. Specifying the **:SEND** option implies that the router will be used for communication. Finally, the layout directive **:SERIAL** tells the compiler that all element along that axis should be allocated within the same virtual processor. The previously defined variable **IFLUX** is thus defined to emulate a 5 by 3 matrix at each grid point.

Assignment statements formally look like those in regular Fortran. Consider the following assignment:

```
A = 3.5 + X*Y.
```

If all variables are scalar, then this is nothing but an ordinary Fortran statement. However, if **A** and either **X** or **Y** are CM-arrays, then the above assignment will take place in every virtual processor. For example, an array is considered a CM-array if there is one axis whose layout directive is **:NEWS**.

In general, whether or not an operation shall be carried out is a function of the state. Examples of conditionals in CM-Fortran are **IF**, **CASE** and **WHERE**. The syntax of the **IF** and **CASE** statements is the same as that of Fortran 8X. To illustrate the usage of the **WHERE** statement we give the example

```
WHERE (X .GE. 0.0) X = SQRT(X).
```

**WHERE** subselects the processors for which **(X .GE. 0.0)** is true.

CM-Fortran also has some powerful global operators. Of particular interest for numeric applications are **MIN**, **MAX**, **MINLOC**, **MAXLOC** and **SUM**.

```
MAX(A1,A2,...)
MIN(A1,A2,...)
MAXLOC(ARRAY,MASK)
MINLOC(ARRAY,MASK)
SUM(ARRAY,DIM,MASK)
```

**MAX** and **MIN** returns an array whose elements are the largest of the corresponding elements of the arguments. **MAXLOC**, **MINLOC** and **SUM** compute the location of maximum and minimum, and the sum of the values in **ARRAY** for those elements (virtual processors) where **MASK** is true. **MASK** and **DIM** are optional.

## 4.6 Load balance

For the model problem the only concern with respect to load balance is the treatment of the boundary points. The boundary stencils we use are structurally subgraphs of the interior stencils. The application of the difference stencils to virtual processors representing grid points on or close to the boundary can be performed concurrently with the interior points, both with respect to computation and communication. However, the evaluation of the boundary conditions are performed *after* the interior points have been properly updated. Though the separate evaluation of the boundary conditions causes a loss in performance, this loss is small for the model problem implemented on the Connection Machine. We also address the issue of efficient memory usage for the implementation of the difference stencils.

### 4.6.1 Difference Stencils

The difference stencils can be evaluated in the interior as well as at the boundary concurrently, if the number of neighbors in a given dimension at the boundaries does not exceed the number of neighbors in the same dimension in the interior. For the flux vectors we use a second order accurate stencil in the interior (7-points in 3D). With only one neighbor in the direction of the normal at the boundary, the approximation of the boundary operators is only first order accurate. However, for linear problems the overall approximation is still second order accurate [9]. Hence, for each grid dimension one difference operator at the left boundary, and one at the right boundary is needed in addition to the difference operator in the interior. For a one-dimensional problem the difference stencil $\mathbf{D}_1^0$ in matrix form is

| Processor | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Sup diag. | 1.0 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0 |
| Main diag. | -1.0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.0 |
| Sub diag. | 0 | -0.5 | -0.5 | -0.5 | -0.5 | -0.5 | -0.5 | -1.0 |
| Variable | $\phi_0$ | $\phi_1$ | $\phi_2$ | $\phi_3$ | $\phi_4$ | $\phi_5$ | $\phi_6$ | $\phi_7$ |

Figure 4: Storage layout for the difference operator $\mathbf{D}_1^0$.

$$\mathbf{D}_1^0 = \begin{pmatrix} -1 & 1 & & & & & & \\ -0.5 & 0 & 0.5 & & & & & \\ & -0.5 & 0 & 0.5 & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & -0.5 & 0 & 0.5 & & \\ & & & & -0.5 & 0 & 0.5 & \\ & & & & & -1 & 1 & \end{pmatrix}. \tag{62}$$

In three dimensions the number of stencils for a difference molecule with $2p+1$ points in each of the three dimensions is $8p^3+12p^2+6p+1$, even if the stencil contains elements off the three axes. The total number of multiplications per stencil evaluation is $2dp+1$, if all elements of the stencil fall along the axes in $d$ dimensions. The difference stencils for all grid points may be stored across the virtual processors as illustrated in Figure 4.

The one-dimensional difference stencil defined by (62) can be implemented as:

Multiplication of the center point:

$$\begin{array}{cccccccc} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \phi_0 & \phi_1 & \phi_2 & \phi_3 & \phi_4 & \phi_5 & \phi_6 & \phi_7 \end{array}$$

The right neighbor. Multiplication, left cyclic array shift, addition:

$$\begin{array}{cccccccc} 1 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0 \\ \phi_1 & \phi_2 & \phi_3 & \phi_4 & \phi_5 & \phi_6 & \phi_7 & \phi_0 \end{array}$$

The left neighbor. Multiplication, right cyclic array shift, addition:

$$\begin{array}{cccccccc} 0 & -0.5 & -0.5 & -0.5 & -0.5 & -0.5 & -0.5 & -1 \\ \phi_7 & \phi_0 & \phi_1 & \phi_2 & \phi_3 & \phi_4 & \phi_5 & \phi_6 \end{array}.$$

The implementation in CM-Fortran is

```
CMF$LAYOUT F(:SERIAL, )
REAL F(3,8), D(8), S(8)
D = F(1, :)*S
D = D + F(2, :)*CSHIFT(S, 1, 1)
D = D + F(3, :)*CSHIFT(S, 1,-1),
```

where the main diagonal is stored in the first row of F. The super- and subdiagonals are stored on the second and third rows, respectively. In general, $p-1$ shifts and $2p-1$ floating-point operations are needed to evaluate a stencil containing $p$ points. No unnecessary operations are performed, since, in general, the coefficients occurring in the difference stencils will be different from 1 and 0. If the differential equation has constant coefficients and the difference approximation is applied at all grid points including the boundary points, (periodic boundary conditions) then the storage of a single instance of the difference stencils suffices. With the stencil stored in the host computers memory the numerical differentiation is performed by broadcasting each coefficient of the stencil, one at a time, to every processor. The time needed to distribute a scalar value from the host to each processor is negligible compared to the time for shifting an array. When many different stencils are needed, as for instance is the case for higher order difference methods for boundary value problems, the host rapidly becomes a bottleneck. Moreover, subselection is required for the broadcasting of each stencil, and potentially also for the application of the stencil. The degree of concurrency in evaluating the stencils may be reduced significantly. The more complex the stencils, the more urgent it is to store them on the data parallel computer.

Our implementation of the Navier-Stokes equations requires the difference operators $\mathbf{D}_1^0, \mathbf{D}_2$ and $\mathbf{D}_4$ in the non-conservative case, and the storage of $\mathbf{D}_1^0, \mathbf{D}_1^+, \mathbf{D}_1^-$ and $\mathbf{D}_3^+$ in the conservative case, calling for a total of 27 and 25 variables per grid point, respectively. Table 4 shows the number of variables per grid point required for each difference operator.

| Difference Operator | Storage Requirement Array elem. in d Dim |
|---|---|
| $\mathbf{D}_1^0$ | 2d + 1 |
| $\mathbf{D}_1^+$ | d + 1 |
| $\mathbf{D}_1^-$ | d + 1 |
| $\mathbf{D}_2$ | 2d + 1 |
| $\mathbf{D}_3^+$ | 3d + 1 |
| $\mathbf{D}_3^-$ | 3d + 1 |
| $\mathbf{D}_4$ | 4d + 1 |
| $\tilde{\mathbf{D}}_4$ | 4d + 1 |
| $\bar{\mathbf{D}}_4$ | 4d + 1 |

Table 4: Memory Requirements for the Difference Operators

If each virtual processor stores all stencils required for the grid point it represents, then the memory requirements for the difference stencils may dominate the storage requirements. For our model problem the coefficients are constants, but storing the stencils

on the front-end would reduce the performance by a factor of three. However, since the coefficients are constants, and the same constants shared between some of the difference operators we use, the storage per virtual processor can be reduced without loss of performance. The main diagonals in $\mathbf{D}_1^+$ and $\mathbf{D}_1^-$ are equal to 1, and need not be stored. The evaluation of the subdiagonal in $\mathbf{D}_1^-$ yields

$$\begin{array}{cccccccc} 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ \phi_7 & \phi_0 & \phi_1 & \phi_2 & \phi_3 & \phi_4 & \phi_5 & \phi_6 \end{array}.$$

which can be accomplished by an *end-off* shift (boundary value zero) and a coefficient array of -1:

$$\begin{array}{cccccccc} -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ 0 & \phi_0 & \phi_1 & \phi_2 & \phi_3 & \phi_4 & \phi_5 & \phi_6 \end{array}.$$

Since all entries of the coefficient array are the same, using an end-off shift operation allow the stencils for $\mathbf{D}_1^-$ and $\mathbf{D}_1^+$ to be stored as scalars, without a performance degradation. To evaluate $\mathbf{D}_1^0$ it suffices to store its main diagonal, since the subdiagonal is equal to $-0.5(\mathbf{I} + diag(\mathbf{D}_1^0))$, and the superdiagonal is $0.5(\mathbf{I} - diag(\mathbf{D}_1^0))$. For the evaluation of $\mathbf{D}_2$ one more coefficient array is needed:

$$(1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1).$$

Since $\mathbf{D}_4 = \mathbf{D}_1^-\mathbf{D}_1^+\mathbf{D}_2$ and $\mathbf{D}_3^+ = \mathbf{D}_1^+\mathbf{D}_2$, it is immediately clear that no more storage is required to evaluate $\mathbf{D}_4$ and $\mathbf{D}_3^+$. Even $\tilde{\mathbf{D}}_4$ could be evaluated without any extra storage, since $diag(\tilde{\mathbf{D}}_4) = 5\mathbf{I} + 0.5 diag(\mathbf{D}_2)$, the remaining sub- and super-diagonals being constants. Hence, a total of $2d$ variables per virtual processor suffice. Six variables per virtual processor covers both the conservative *and* the non-conservative 3D-Navier-Stokes equations.

## 4.7 Virtual Processor Operations.

Many of the computations in each virtual processor is of the type defined in levels one and two of the Basic Linear Algebra Subroutine packages [22, 6]. The advantage of using these functions is that architectural features such as registers and caches can be used effectively for performance enhancements. On the Connection Machine, the performance for scalar operations in each virtual processor is in the range 1.5 - 2.2 Gflops/s for 32-bit floating-point addition, subtraction, and multiplication. Kernels of the matrix-vector type achieve a peak performance in excess of 20 Gflops/s [19] when effective use of the registers on the floating-point units is made. The performance quoted in this paper was obtained without any such features. Using vector, matrix-vector and matrix-matrix operations for the computations in each grid point the arithmetic performance can be improved significantly.

The computations of $\boldsymbol{\tau}$ and $\hat{a}^2$. involve scalar arithmetic only. The evaluation of the velocity gradients $\nabla u, \nabla v, \nabla w$ and $\nabla \hat{a}^2$ comprises a matrix-matrix multiplication, the matrices being $3 \times 3$ and $3 \times 4$. The computation of $\boldsymbol{\kappa}$, equation (21) includes a matrix-vector operation where the matrix is a $3 \times 3$ real, symmetric, matrix. The computation of the inviscid flux vectors is of the type $vector \leftarrow scalar_1 \times vector_1 + scalar_2 \times vector_2$ for each of the equations (10), (11), and (12), where $vector_1$ and $scalar_2$ are common for all three equations. The vectors are of length 5. Each vector operation can be viewed as a $5 \times 2$ matrix-vector multiplication. The computation of the viscous flux vectors are of type matrix-vector multiplication, where the matrix is common to all three equations. The computations of all three viscous flux vectors, equations (15), (16), and (17), can be performed as a matrix-matrix multiplication, where the first matrix is a $4 \times 3$ matrix and the second a $3 \times 3$ matrix. The computation of the contravariant velocities involves a $3 \times 4$ matrix-vector multiplication. The computation of $\sigma_{jkl}$, $p_{jkl}$, $\Upsilon_{jkl}$, $\epsilon_{jkl}$, $\varepsilon_{jkl}$ consists of scalar operations. Finally, the implementation of the difference stencils is similar to multi-dimensional convolution in that a weighted sum of a set of neighborhood values is computed for every point. The difference operation is vector valued, with vectors of length 5. The width of the kernel depends on the difference operator, and ranges from two to five in each of the three space dimensions.

In summary, the computations in each virtual processor contains the following set of BLAS primitives

- Scaling of vectors of length 3.
- 2-norms ($||\cdot||_2$) on vectors of length 3.
- Inner-products for vector length 3.
- Difference Stencils on vector arguments. Vectors of length 5.
- Matrix Vector multiplication in each virtual processor
  - $5 \times 2$ matrix vector multiplication
  - $3 \times 4$ matrix-vector multiplication
  - $3 \times 3$ matrix-vector multiplication
- Matrix-matrix multiplication
  - $4 \times 3$ by $3 \times 3$ matrix-matrix multiplication
  - $3 \times 3$ by $3 \times 4$ matrix-matrix multiplication

# 5 Simulations

The required computations are divided into a preprocessing step in which time independent variables are computed, and the main simulation step for time dependent variables. In the preprocessing step the grid is generated and the Jacobian computed as well as some other variables as follows

1. Compute $\nabla\xi$, $\nabla\eta$, and $\nabla\zeta$ by equations (4), (5), (6) and (7). Note that we assume a time independent geometry.
2. Compute $|\mathbf{J}_{jkl}|^{-1}$ from equation (1).
3. Compute $|\mathbf{J}_{jkl}|^{-1}Re^{-1}$.
4. Compute $(|\nabla\xi_{jkl}|_2 + |\nabla\eta_{jkl}|_2 + |\nabla\zeta_{jkl}|_2)$.
5. Compute $\dfrac{k}{Pr(1-1/\gamma)}$.

The simulation of the Navier-Stokes equations consists of

1. Compute $u, v, w$ from $\mathbf{q}$.
2. Compute the contravariant velocities $U$, $V$ and $W$ by equation (13).
3. Compute $p$ by equation (14).
4. Compute $\hat{a}^2$ by equation (24).
5. Compute $u_x, \ldots, w_z$ by equations (20) and (31) for interior and boundary points, respectively.
6. Compute $\nabla a^2$ by the analogue of (20) and equations (31).
7. Compute $\boldsymbol{\tau}$ by equation (19).
8. Compute $\boldsymbol{\kappa}$ by equation (21).
9. Compute $\sigma_{jkl}$ by equation (36).
10. Compute $\mathbf{F}_{jkl}, \mathbf{G}_{jkl}, \mathbf{H}_{jkl}$ by equations (10), (11), and (12).
11. Compute $\mathbf{F}_{\nu_{jkl}}, \mathbf{G}_{\nu_{jkl}}, \mathbf{H}_{\nu_{jkl}}$ by equations (15), (16), and (17).
12. Compute $\Upsilon_{jkl}$ by equation (41).
13. Compute $\epsilon$ by equation (40).

14. Compute $\varepsilon$ by equation (42).

15. Compute the fourth order difference by one of equations (45), (46) or (47) or by equations (48) and (52).

16. Compute the second order difference by equation (50).

17. Compute $\frac{\partial \check{\mathbf{q}}_{jkl}}{\partial \tau}$ by equation (30).

18. Compute one Runge-Kutta step by equation (57).

For the implementation, the Connection Machine was configured as a regular 3D-grid. No low level coding, or optimization with respect to the architecture, has been undertaken. Fast library routines of the BLAS type were not available at the time of the experiments. The implementation of the combination of the $D_0, D_1$ and $D_3$ stencils that was numerically best behaved was not optimized with respect to storage, and required approximately 170 variables per virtual processor. In the simulations additional stencils were tried. The code had 218 variables per virtual processor, and the maximum virtual processor ratio with $8k$ bytes of storage per physical processor was 8. The subgrid for each processor was a $2 \times 2 \times 2$ grid. Each floating-point unit services a $4 \times 8 \times 8$ subgrid. The measured bandwidth for nearest neighbor communication in this grid was on the average about 2.5 Gbytes/s for a fully configured Connection Machine.

The execution times as a function of the virtual processor ratio, and the physical machine size are given in Table 5. The physical domain was $3.5 \times 1.75 \times 14$ cm$^3$, the flow velocity 80 m/s, and the Reynolds number 140. The aspect ratio of any pair of grid dimensions was either one or two, and the size ranging from $16 \times 16 \times 32$ to $64 \times 64 \times 64$. The number of floating-point operations per virtual processor was 2,550 per time step, and the number of lattice communications of single variables 492. The execution time per time step is independent of the machine size, as expected, Table 5 and Figure 5. The processor utilization increases by a factor of 2.75 as the virtual processor ratio increases from 1 to 8. The work increases by a factor of 8, but the execution time only by a factor of 2.9.

| Virtual processor ratio | Machine Size | | | | | |
|---|---|---|---|---|---|---|
| | 8k | | 16k | | 32k | |
| | CM-time | Total time | CM-time | Total time | CM-time | Total time |
| 1 | 0.261 | 0.443 | 0.261 | 0.442 | 0.261 | 0.442 |
| 2 | 0.412 | 0.474 | 0.412 | 0.474 | 0.412 | 0.474 |
| 4 | 0.707 | 0.708 | 0.704 | 0.705 | 0.701 | 0.702 |
| 8 | 1.270 | 1.270 | 1.283 | 1.283 | 1.283 | 1.283 |

Table 5: Execution Time for Different Virtual Processor Ratios.

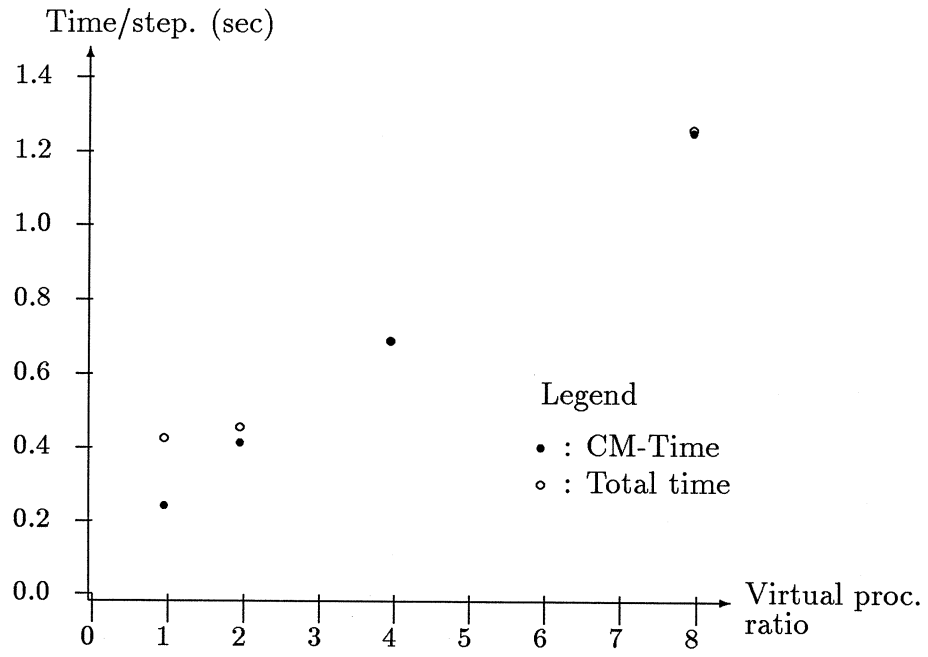


Figure 5: The Execution Time for a Single Time Step.

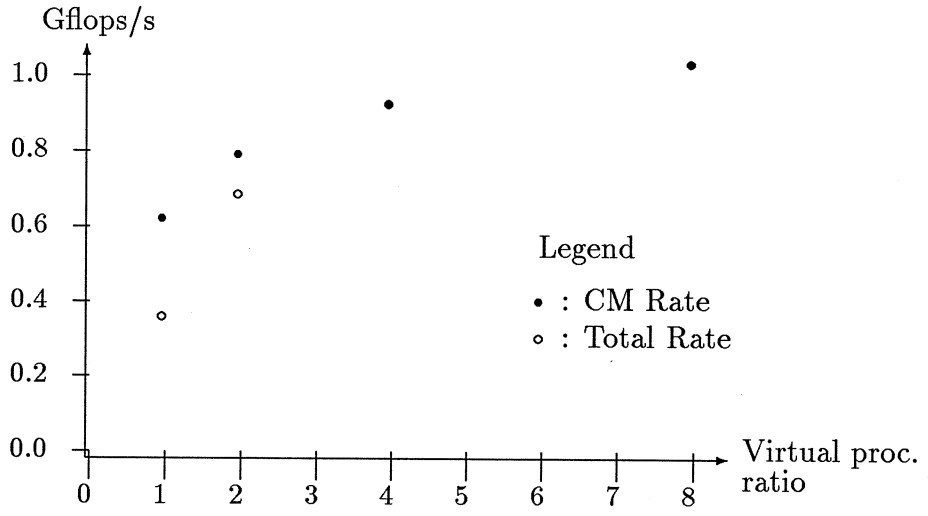


Figure 6: The Execution Speed for the Compressible Navier-Stokes Solver.

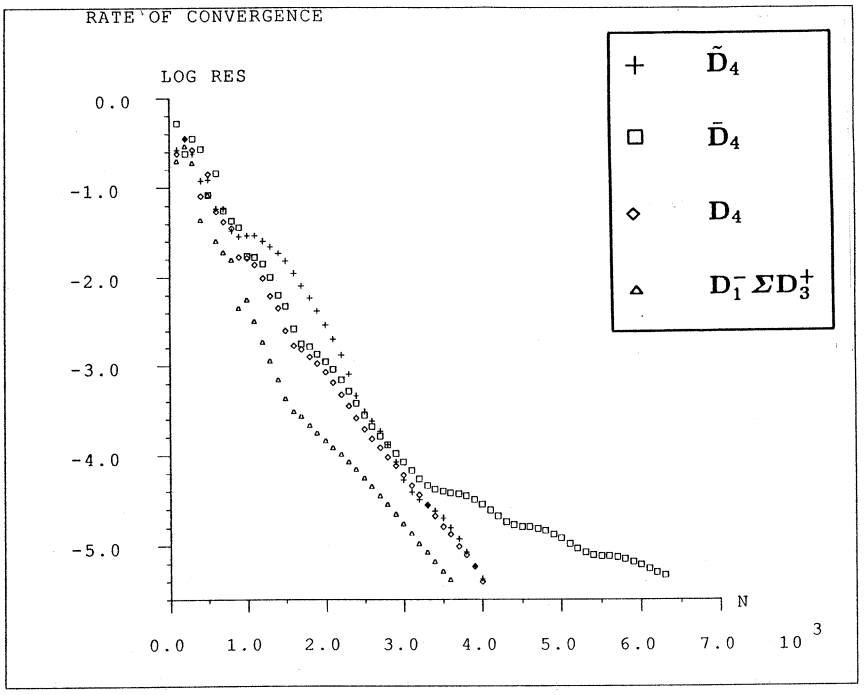


Figure 7: Rate of Convergence for the Different Dissipation Operators, Irrotational Inflow

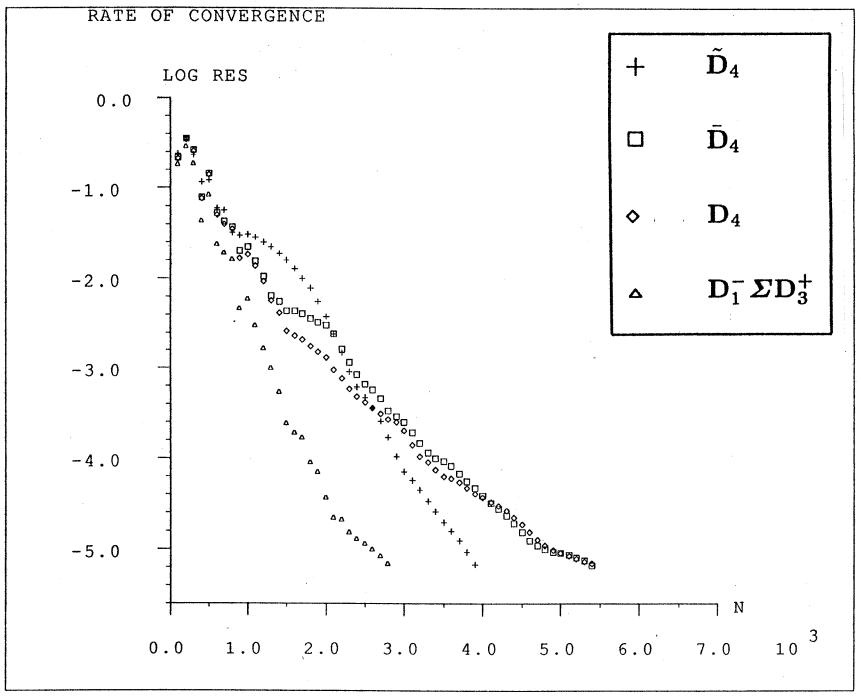


Figure 8: Rate of Convergence for the Different Dissipation Operators, Weakly Rotational Inflow

Figure 6 shows the achieved floating-point rate for *Lisp. Computations expressed in *Lisp have a peak performance of about 2.3 Gflops/s. The difference between the peak and the measured performance is mostly due to the communication required for the difference stencils. The evaluation of the boundary conditions accounts for less than five percent of the total execution time. The utilization of the Connection Machine exceeds 99 %. With a virtual processor ratio of 8, grids with up to 524,288 points were simulated.

The rate of convergence for the Euler equations is illustrated in Figures 7 and 8. The residual at time step $n$ is computed as

$$\frac{\|\rho^n - \rho^{n-1}\|_2}{\|\rho^n\|_2},$$

On a 4 k CM–2 it takes approximately 40 minutes to reach the steady state for a grid of 32,768 points. This time is the same as the execution time for the original Fortran code on a CRAY-1S (Linköping, Sweden). The execution time on a single processor Alliant FX/8 is approximately ten times longer.

# 6 Summary and Conclusions

By making the stencils on or close to the boundary be subgraphs of the stencil in the interior of the computational domain, the stencils can be evaluated for *all* grid points concurrently. Three different fourth order stencils were tried, as well as three different factorizations of one of these stencils. The difference between the three fourth order stencils were due to different treatment of the boundary stencils. The different treatment of the boundary for the artificial viscosity resulted in significantly different flow fields at the outflow. The convergence properties were also significantly different. Factoring of the higher order stencils as products of lower order stencils affected the convergence behavior. The operator $\mathbf{D}_1^- \boldsymbol{\Sigma} \mathbf{D}_3^+$ had the best convergence behavior. For residuals of less than about $10^{-2} - 10^{-3}$ this operator had a convergence rate of approximately twice that of the operator $\mathbf{D_4}$.

The factorization of the higher order stencils also provides a mechanism for conserving storage. For the three dimensional Navier-Stokes equations six variables per lattice point suffice for the generation of all difference stencils needed for the flux vectors and the three different fourth order operators tried for the artifical viscosity, including the different factorizations of one of these operators.

Computationally, the most frequent functions in the Euler version of the code are evaluation of three-dimensional difference stencils on vectors of length five, matrix multiplication at each lattice point for the computation of the contravariant velocity components, and inner product evaluation for the computation of the pressure at a lattice point. The vector length for the inner product is three, and the matrices required for the contravariant velocity components are of size $3 \times 4$. The Navier-Stokes code requires in addition

three matrix-vector multiplications with the same $4 \times 3$ matrix, and all three resulting vectors scaled with the same variable ($|\mathbf{J}|^{-1}Re^{-1}$), one $3 \times 3$ matrix-vector multiplication, and a SAXPY operation for computing $\boldsymbol{\kappa}$. The Euler code requires approximately $1,900$ floating-point operations per time step, and the Navier-Stokes code approximately $2,550$ operations per time step. The number of variables communicated per time step is 396 in the Euler case and 492 in the Navier-Stokes case.

The explicit method parallelizes easily, and a floating-point rate of 1.05 Gflops/s was achieved for a 64k processor configuration and a virtual processor ratio of 8. The evaluation of the boundary conditions accounted for less than 5% of the total time. The processor utilization increases by a factor of 2.75 in increasing the virtual processor ratio from 1 to 8. We estimate that with optimization of the storage of the coefficients for the difference stencils, and the use of temporary variables a virtual processor ratio of 32 is feasible for 8k bytes of storage per processor (and 128 for 32 kbytes per processor). Such an increase in the virtual processor ratio is expected to directly increase the performance by about 5%. A much more significant increase in performance is likely to be achieved by the use of optimized BLAS functions, and convolution kernels. For the matrix and vector sizes in the code the performance of optimized kernels is expected to be three to five times higher than the scalar performance in the current code. The communication times can also be reduced by performing concurrent communication on all three axes. The communication primitives used in our implementation only performed communication in one direction along one axis at the time. A total performance improvement by a factor of about three is expected by optimizing arithmetic and communication functions.

## Acknowledgement

Many thanks are due Bertil Gustafsson for valuable suggestions and critique throughout the project, and for many comments on the manuscript. The first implementation of the Euler and Navier-Stokes solvers was made in Release 4 of the Connection Machine software. Porting of the program to Release 5 was made by Louis Howell and David Serafini of Thinking Machines Corporation.

# References


[1] *Lisp release notes. Thinking Machines Corp., 1987.

[2] Programming in C*. Thinking Machines Corp., 1987.

[3] M.Y. Chan. Dilation-2 embeddings of grids into hypercubes. Technical Report UT-DCS 1-88, Computer Science Dept., University of Texas at Dallas, 1988.

[4] M.Y. Chan. Embeddings of 3-dimensional grids into optimal hypercubes. Technical report, Computer Science Dept., University of Texas at Dallas, 1988. To appear in the Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications, March, 1989.

[5] Monty M. Denneau, Peter H. Hochschild, and Gideon Shichman. The switching network of the TF-1 parallel supercomputer. *Supercomputing Magazine*, 2(4):7–10, 1988.

[6] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. An extended set of fortran basic linear algebra subprograms. Technical Report Technical Memorandum 41, Argonne National Laboratories, Mathematics and Computer Science Division, November 1986.

[7] Rickard Enander and Johan Sowa. Numerical simulation of fluid flow in a twisted channel. Technical Report 88-01, Department of Scientific Computing, Uppsala University, 1988.

[8] L. E. Eriksson. Boundary conditions for artificial dissipation operators. Technical Report FFA TN 1984-53, The Aeronautical Research Institute of Sweden, Aerodynamics Department, Stockholm, Sweden, 1984.

[9] Bertil Gustafsson. The convergence rate for differential approximations to general mixed initial boundary value problems. *SIAM J Numer. Anal.*, 18(2), 1981.

[10] I. Havel and J. Móravek. B-valuations of graphs. *Czech. Math. J.*, 22:338–351, 1972.

[11] W.D. Henshaw, H.-O. Kreiss, and L.G. Reyna. On the smallest scale for the incompressible Navier-Stokes equations, 1987. Personal Correspondence.

[12] W. Daniel Hillis. *The Connection Machine.* MIT Press, Cambridge, MA, 1985.

[13] Ching-Tien Ho and S. Lennart Johnsson. On the embedding of arbitrary meshes in Boolean cubes with expansion two dilation two. In *1987 International Conf. on Parallel Processing*, pages 188–191. IEEE Computer Society, 1987.

[14] Ching-Tien Ho and S. Lennart Johnsson. Embedding meshes in boolean cubes by graph decomposition. *Journal of Parallel and Distributed Computing*, 7(3), December 1989. Technical Report YALEU/DCS/RR-689, Department of Computer Science Yale University, March 1989.

[15] A. Jameson, W. Schmidt, and E. Turkel. Numerical solutions of the Euler equations by finite volume methods using Runge-Kutta time-stepping schemes. *AIAA Paper*, 81-1259, 1981.

[16] S. Lennart Johnsson. Future high performance computation: The megaflop per dollar alternative. Technical Report YALEU/DCS/RR-360, Dept. of Computer Science, Yale University, January 1985.

[17] S. Lennart Johnsson. Communication efficient basic linear algebra computations on hypercube architectures. *J. Parallel Distributed Comput.*, 4(2):133–172, April 1987.

[18] S. Lennart Johnsson. *Optimal Communication in Distributed and Shared Memory Models of Computation on Network Architectures.* Morgan Kaufman, 1989.

[19] S. Lennart Johnsson, Tim Harris, and Kapil K. Mathur. Multiplication of arbitrary matrices on a data parallel computer. Technical report, Thinking Machines Corp., 1989. in Preparation.

[20] S. Lennart Johnsson, Robert L. Krawitz, Douglas MacDonald, and Roger Frye. Cooley-tukey fft on the connection machine. Technical report, Thinking Machines Corp., 1989. in Preparation.

[21] S. Lennart Johnsson and Peggy Li. Solutionset for ama/cs 146. Technical Report 5085:DF:83, California Institute of Technology, May 1983.

[22] C. L. Lawson, R.J. Hanson, D.R. Kincaid, and F.T. Krogh. Basic linear algebra subprograms for fortran usage. *ACM TOMS*, 5(3):308–323, September 1979.

[23] Michael Metcalf and John Reid. *Fortran 8X Explained.* Oxford Scientific Publications, 1987.

[24] Jan Nordström. Open boundary conditions for the navier-stokes equation. Technical Report FFA TN 1988-21, The Aeronautical Research Institute of Sweden, Stockholm, Sweden, 1988.

[25] Pelle Olsson and S. Lennart Johnsson. A study of dissipation operators for the eulers equations and three-dimensional channel flow. *Journal of Scientific Computing.* Technical Report CS89-3, Thinking Machines Corp., February 1989.

[26] Pelle Olsson and S. Lennart Johnsson. Solving three-dimensional Navier-Stokes equations by explicit methods on a data parallel computer. Technical report, Thinking Machines Corp., 1989. in preparation.

[27] Thomas E. Pulliam. Artificial dissipation models for the Euler equations. *AIAA*, 24(12), December 1986.

[28] E M. Reingold, J Nievergelt, and N Deo. *Combinatorial Algorithms.* Prentice-Hall, Englewood Cliffs. NJ, 1977.