Internal report Nr 98/12

CHALMERS UNIVERSITY OF TECHNOLOGY
Department of Thermo and Fluid Dynamics

CALC-PVM: A Parallel SIMPLEC Multiblock Solver for Turbulent Flow in Complex Domains

by

Håkan Nilsson and Lars Davidson

Göteborg, August 1998

1 abstract

A finite volume code CALC-BFC (Boundary Fitted Coordinates) is extended with PVM (Parallel Virtual Machine) for parallel computations of turbulent flow in complex multiblock domains. The main features are the use of general curvilinear coordinates, pressure correction scheme (SIMPLEC), Cartesian velocity components as principal unknowns, and colocated grid arrangement together with Rhie and Chow interpolation. Taking advantage of the communication facilities of PVM, the problem can be run in parallel on several processors on everything from NOW (Network Of Workstations) to distributed and shared memory supercomputers. Subdividing a 130x82x82 node turbulent backward facing step flow calculation into 2x1x2 blocks and distributing it on four processors gives a speedup of 4.3 and subdividing it into 4x1x2 blocks and distributing it on eight processors gives a speedup of 8.3 on a 64 processor SUN Enterprise 10000 shared memory machine.

Contents

2 Introduction

The governing equations of fluid flow are discretized using a finite volume method with a colocated grid arrangement. In order to deal with complex geometries, a non-orthogonal boundary fitted coordinate multiblock method is used. For grid generation, an interface to the commercial grid generator ICEM has been implemented. The SIMPLEC method provides the pressure-velocity coupling, needed to solve the Navier-Stokes and continuity equations. In order to solve the equations, the computational domains (blocks) need to exchange interblock boundary conditions. Taking advantage of the communication facilities of PVM (Parallel Virtual Machine, see [7] and appendix B), the interblock communication is fairly easy to implement and the problem may be run in parallel on everything from NOW (Network Of Workstations) to distributed and shared memory supercomputers. The code may optionally subdivide single block domains into equally sized smaller blocks for parallel computations. The gains of this operation are several: the computational speed may be increased, larger problems can be solved since the memory requirements is divided between the processors, more exact solutions can be obtained because of the extra memory available and parallel supercomputers may be utilized. Subdividing a 130x82x82 node turbulent backward facing step flow calculation into 2x1x2 blocks and distributing it on four processors gives a speedup of 4.3 and subdividing it into 4x1x2 blocks and distributing it on eight processors gives a speedup of 8.3 on a 64 processor SUN Enterprise 10000 shared memory machine [2] (speedup is defined as the elapsed time to reach convergence). This is actually more than linear speedup, probably because of less memory demands on each processor.
Some technical descriptions of the code is included in the appendix.

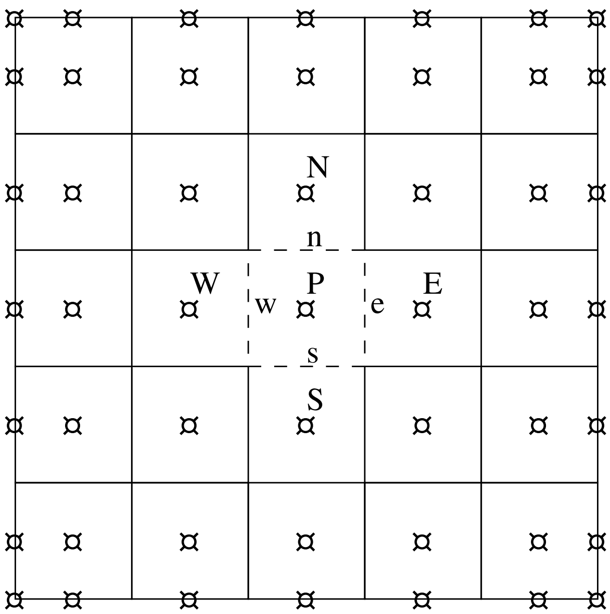

Figure 3.1: The division of the domain into a finite number of control volumes. Two-dimensional example. The nodes are placed in the center of the control volumes except at the boundaries, where they are placed at the boundary. At the center control volume (dashed line), the nomenclatures for control volume nodes (P, E, W, N, S) and faces (e, w, n, s) are introduced.

3 Solution algorithm

The Navier Stokes equations for incompressible flow is discretized using a finite volume method with a colocated grid arrangement. These equations will depend on the pressure distribution, which must be solved together with the velocities. Since there is no equation for the pressure for incompressible flow, some kind of pressure-velocity coupling is needed. The pressure-velocity coupling used in this code is called SIMPLEC and is described in the following section.

3.1 Pressure-velocity coupling: the SIMPLEC method

In order to solve the Navier Stokes- and continuity-equations the SIMPLEC[5] method (Semi-Implicit Method for Pressure-Linked Equations, Consistent) supplying the pressure-velocity coupling, is used. The method has its origin in staggered grid methodology and is adapted to colocated grid methodology through the use of Rhie & Chow interpolation, described in the next section. The nomenclature used to derive the expressions in this section is capital index letters; E, W, N, S, H, L for non-staggered (scalar) control volumes and small index letters; e, w, n, s, h, l for staggered (velocity) control volumes ie. on the scalar control volume faces (see fig. 3.1). The derivations are made only on the e staggered control volume, but the

other directions are treated in a similar way.
Defining pressure and velocity corrections p' and u'_i as the difference between the pressure or velocity field p and u_i at the current iteration (*new*) and the pressure or velocity field p^* and u^*_i from the previous iteration (*old*), we have

$$\begin{aligned} p &= p^* + p' \\ u_i &= u^*_i + u'_i \end{aligned} \tag{3.1}$$

The discretized momentum equations for the *old* velocities, in a staggered control volume

$$a_e u^*_{i\,e} = \sum a_{nb} u^*_{i\,nb} + \left(p^*_P - p^*_E\right) A_{ie} + b_{u_{i\,e}}$$

are substracted from the discretized momentum equations for their *new* values, ie.

$$a_e u_{ie} = \sum a_{nb} u_{inb} + \left(p_P - p_E\right) A_{ie} + b_{u_{i\,e}}$$

yielding

$$a_e u'_{ie} = \sum a_{nb} u'_{inb} + \left(p'_P - p'_E\right) A_{ie} \tag{3.2}$$

The omission of the term $\sum a_{nb} u'_{nb}$ is the main approximation of the SIMPLE [8] algorithm, giving

$$u'_{ie} = \left(p'_P - p'_E\right) \frac{A_{ie}}{a_e}$$

This omission will have no impact on the final solution, since $u'_{nb} = 0$ for a converged solution. However, this omission is rather inconsistent since the term on the left hand side of equation 3.2 is of the same order as those omitted. A more consistent approach is obtained by substracting the term $\sum a_{nb} u'_{ie}$ from both sides of eq. 3.2. This yields

$$\left(a_e - \sum a_{nb}\right) u'_{ie} = \sum a_{nb} \left(u'_{inb} - u'_{ie}\right) + \left(p'_P - p'_E\right) A_{ie}$$

The omission of the term $\sum a_{nb} \left(u'_{inb} - u'_{ie}\right)$ is the consistent approximation of the SIMPLEC algorithm, giving

$$u'_{ie} = \frac{A_{ie}}{a_e - \sum a_{nb}} \left(p'_P - p'_E\right)$$

where $a_e = a'_e/\alpha$ and α is the velocity underrelaxation. Finally, we get an expression for the *new* face velocities from eq. 3.1 as

$$u_{ie} = u^*_{i\,e} + d_e \left(p'_P - p'_E\right) \tag{3.3}$$

where

$$d_e = \frac{A_{ie}}{a_e - \sum a_{nb}}$$

and, by analogy

$$\begin{aligned}
u_{iw} &= u^*_{i\,w} - d_w\left(p'_P - p'_W\right) \\
u_{in} &= u^*_{i\,n} + d_n\left(p'_P - p'_N\right) \\
u_{is} &= u^*_{i\,s} - d_s\left(p'_P - p'_S\right) \\
u_{ih} &= u^*_{i\,h} + d_h\left(p'_P - p'_H\right) \\
u_{il} &= u^*_{i\,l} - d_l\left(p'_P - p'_L\right)
\end{aligned} \tag{3.4}$$

Inserting these corrected velocities into the discretized continuity equation of a non-staggered control volume

$$\sum_{C.S.}(\rho u_{inb} A_{inb}) = 0$$

and identifying coefficients gives a discretized equation for the pressure correction

$$a_P p'_P = a_E p'_E + a_W p'_W + a_N p'_N + a_S p'_S + a_H p'_H + a_L p'_L + b_{p'\,P}$$

where

$$\begin{aligned}
a_P &= a_E + a_W + a_N + a_S + a_H + a_L \\
a_E &= \rho_e \frac{A_e^2}{a_e - \sum a_{nb}} \\
&\vdots \\
b_{p'\,P} &= -\sum_{C.S.}\left(\rho u^*_{i\,nb} A_{inb}\right)
\end{aligned}$$

This Poisson-equation for the pressure correction is solved using a_{nb}- and $u^*_{i\,nb}$ values (on the scalar control volume faces) from the momentum equations. Since this code is utilizing a colocated grid arrangement, the a_{nb} values is obtained from linear interpolation of the a_P values and the u_{inb} values is obtained from Rhie & Chow interpolation (described in section 3.2) of the $u_{i\,P}$ values. The *new* pressure field may then be obtained from eq. 3.1 and the *new* convections (through the scalar control volume faces) and the node velocities are corrected according to eqs. 3.3 and 3.4.

3.2 Rhie & Chow interpolation

Since this code is utilizing a colocated grid arrangement the convections through the faces, needed for the pressure correction equation, are obtained from Rhie & Chow interpolation, described below.
The face velocity is usually obtained by linear interpolation, ie.

$$u_{ie} = \frac{1}{2}\left(u_{iE} + u_{iP}\right)$$

In a colocated grid arrangement this may however lead to pressure oscillations. To avoid this, the face velocities is calculated by substracting and adding the pressure gradient, ie.

$$u_{ie} \;=\; \frac{1}{2}\left(u_{iE}+u_{iP}\right)-\left(-\frac{\partial p}{\partial x_i}\frac{\delta V}{a_P}\right)_e+\left(-\frac{\partial p}{\partial x_i}\frac{\delta V}{a_P}\right)_e$$

The pressure gradient terms in this expression are calculated in different ways. The first one is calculated as the mean value of the pressure gradient in the P and E nodes, ie.

$$\begin{aligned}\left(-\frac{\partial p}{\partial x_i}\frac{\delta V}{a_P}\right)_e &= -\frac{1}{2}\left(\left(\frac{\partial p}{\partial x_i}\right)_E+\left(\frac{\partial p}{\partial x_i}\right)_P\right)\left(\frac{\delta V}{a_P}\right)_e\\ &= -\frac{1}{2}\left(\frac{p_{EE}-p_P}{\delta x_{iPEE}}+\frac{p_E-p_W}{\delta x_{iWE}}\right)\left(\frac{\delta V}{a_P}\right)_e\end{aligned}$$

The second one is calculated on the face, ie.

$$\left(-\frac{\partial p}{\partial x_i}\frac{\delta V}{a_P}\right)_e \;=\; -\left(\frac{p_E-p_P}{\delta x_{iPE}}\right)\left(\frac{\delta V}{a_P}\right)_e$$

Equidistant grid yields $\delta x_{iPEE}=\delta x_{iWE}=2\delta x_{iPE}$ and

$$u_{ie} \;=\; \frac{1}{2}\left(u_{iE}+u_{iP}\right)+\frac{1}{4\delta x_{iPE}}\left(\frac{\delta V}{a_P}\right)_e\left[p_{EE}-3p_E+3p_P-p_W\right]$$

which is used to calculate the convections ($conv_{face}$) through the control volume faces. In order to deal with non-equidistant grids, the first term is calculated as a weighted average. The second term is however kept as it is, as it represents a fourth-order derivative term to dampen oscillations [6].

3.3 Numerical procedure

The numerical procedure can be summarized as follows.
The velocity and pressure fields together with any other scalar field is calculated by guessing initial values of the fields and iterating through pts. I - X until convergence.

I The discretized momentum equations

$$a_P u^*_{i\,P}=\sum a_{NB}u^*_{i\,NB}+\left(P^*_W-P^*_E\right)A_{iP}+b_{u_iP}$$

are solved.

II The inter-block boundary conditions for a_P from the discretized momentum equations are exchanged, since they are needed for the Rhie & Chow interpolation.

III The convections are calculated using Rhie & Chow interpolation

$$conv_e = (\rho u_i A_i)_e + \left(\frac{A}{4a_P}\right)_e [p_{EE} - 3p_E + 3p_P - p_W]$$

where the $(a_P)_e$ values are obtained from linear interpolation.

IV The continuity error, needed for the source term in the pressure correction equation, is calculated from these convections.

V The discretized pressure correction equation

$$a_P p'_P = a_E p'_E + a_W p'_W + a_N p'_N + a_S p'_S + a_H p'_H + a_L p'_L + b_{p'\,P}$$

where $b_{p'\,P}$ is the continuity error, is solved.

VI The inter-block boundary conditions for the pressure correction is exchanged.

VII The pressure, convections and velocities are corrected as

$$\begin{aligned} p_P &= p^*_P + \left(p'_P - p'_{Pref}\right) \\ conv_e &= conv_e + \rho_e A_{ie} d_e \left(p'_P - p'_E\right) \\ u_{iP} &= u^*_{i\,P} + d_P \left(p'_W - p'_E\right) \end{aligned}$$

where p'_{Pref} is a reference value for p' from one point of the global computational domain. The velocity correction is actually not necessary, but it has proven to increase the convergence rate.

VIII Inter-block boundary conditions for all variables are exchanged.

IX Other discretized transport equations are solved.

X The residual is calculated (see appendix A) and compared with the convergence criteria.

4 Boundary conditions

For the pressure, implicit inhomogenous Neumann boundary conditions are used at all boundaries, ie.

$$\frac{\partial^2 p}{\partial n^2} = 0$$

where n is the coordinate direction normal to the boundary.
The pressure correction p' has an implicit homogenous Neumann boundary condition on all boundaries [8], ie.

$$\frac{\partial p'}{\partial n} = 0$$

where n is the coordinate direction normal to the boundary.
Boundary conditions for the velocities and other variables are described in the following sections.

4.1 Walls

Either wall functions or no-slip boundary conditions are used at walls.

4.2 Inlet/outlet conditions

At an inlet, all flow properties are prescribed to an approximate velocity profile. They can be interpolated from experimental data or from a fully developed profile, for instance: a parabolic profile for laminar flow or a 1/7-profile for a turbulent flow.

At a large outlet, sufficiently far downstream and without area change, the flow may be assumed as fully developed, which implies negligible streamwise gradients of all variables, i.e.

$$\frac{\partial \Phi}{\partial n} = 0$$

where n is the coordinate direction normal to the outlet.

In order to get a mathematically well posed SIMPLEC algorithm, mass flux must be globally conserved [1]. It is a constraint necessary for the pressure correction equation to be consistent. It also considerably increases convergence rate and has positive effects on open boundaries where inflow is occurring. A velocity increment

$$u_{incr} = \frac{\dot{m}_{in} - \dot{m}_{out}^{comp}}{(\rho A)_{out}},$$

where $\dot{m}_{in}$ is the convection into the domain at the inlet, $\dot{m}_{out}^{comp}$ is the computed convection out of the domain at the outlet and A is the outlet area, is added to the computed velocity at the outlet, i.e.

$$u_{out} = u_{out}^{comp} + u_{incr}$$

This ensures that global continuity is fulfilled during the iterations.

4.3 Symmetric boundaries

At symmetry planes, there is no flux of any kind normal to the boundary, either convective or diffusive. Thus, the normal velocity component, as well as the normal gradients of the remaining dependent variables, are set to zero.

4.4 Periodic boundaries

Periodic boundaries can be of two types: translational and rotational. These types of boundary conditions must come in pairs, one boundary connected to another. A three dimensional rotational transformation of vector quantities,

$$\vec{V} = \begin{bmatrix} \cos(\Delta\Theta)\cos(\Delta\varphi) & -\cos(\Delta\Theta)\sin(\Delta\varphi) & \sin(\Delta\Theta) \\ \sin(\Delta\varphi) & \cos(\Delta\varphi) & 0 \\ -\sin(\Delta\Theta)\cos(\Delta\varphi) & \sin(\Delta\Theta)\sin(\Delta\varphi) & \cos(\Delta\Theta) \end{bmatrix} \vec{V}'$$

where $\Delta\Theta$ and $\Delta\varphi$ are the differences in characteristic orientation of the planes, is applied. Then the periodic boundaries may be treated as if they were connected to each other, since rotational periodicity has no impact on scalar quantities and translational periodicity has no impact on neither vector nor scalar quantities.

4.5 Degenerate boundaries

A cylindrical degenerate boundary is the boundary obtained when merging two of the boundary edges, forming a line boundary condition. In order to derive the boundary condition for a cylindrical degenerate boundary the velocity vector in an arbitrary point ($\vec{U} = u\hat{x} + v\hat{y} + w\hat{z}$) is examined. Transformation of the velocity vector to cylindrical coordinates with the z-axis as main axis yields

$$\vec{U} = (u\cos\alpha + v\sin\alpha)\hat{\rho} + (v\cos\alpha - u\sin\alpha)\hat{\alpha} + w\hat{z}$$

or

$$\vec{U} = U_\rho\hat{\rho} + U_\alpha\hat{\alpha} + U_z\hat{z}$$

Looking at a degenerate boundary, going through origo, the line is specified independent of α. Requireing that $U_\rho = 0$ on the degenerate boundary gives $u = v = 0$, since cos and sin are orthogonal functions.
The cylindrical degenerate boundary condition for a boundary going through origo with the z-axis as main axis becomes

$$\begin{aligned} u &= 0 \\ v &= 0 \\ \frac{\partial w}{\partial \rho} &= 0 \end{aligned}$$

5 Turbulence modelling

In addition to the pressure and velocity fields, turbulent quantities and their impact on the pressure and velocity fields may be solved simultaneously. At this moment the $k - \epsilon$ model with wall functions [3] and Large Eddy Simulation (LES) with the Smagorinsky model [2] is implemented, but other models will soon be included.

6 Validation

The domain decomposition of the code has been validated in two-dimensional turbulent backward facing step flow. The domain is described as a 134x42x4 node rectangular grid in the z-plane (with a few nodes in the z-direction for computational reasons) with height $H = 1$ and length $L = 16$. The inlet flow is set as a fully developed turbulent $1/7$-profile into the upper half of the domain. The Reynolds number is $Re = u_{max}H/\nu = 24000$, based on $u_{max} = 30$ denoting

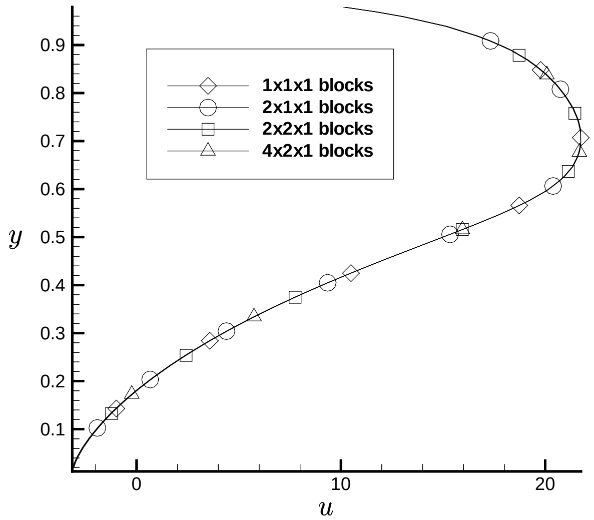

Figure 6.1: Two-dimensional backward-facing step. Steady state u-velocity at $x = 2$. Multitask vs. singletask calculations.

maximum inlet velocity. Splitting the domain into 2x1x1, 2x2x1 and 4x2x1 subdomains introduces no error to the steady state solution as can be seen in figure 6.1. The single block computations has been validated in reference [7].

7 Speedup

Two different speedup tests have been made. First, the computational times for the validation decompositions described in section 6.1 was compared with the computational time for the non-decomposed grid. Since this was a rather small problem, the delay times for interblock connection communication is apparent. In larger problems this will not be the case. To show this, a three dimensional backward facing step calculation has been made [2]. This time with $H/h = 6$, $D = 3H$ and $L = 11.5H$, where L is the length, H is the height and D is the depth of the domain. h is the inlet height. The grid was consisting of 130x82x82 nodes which was split into 2x1x2 and 4x1x2 domains. Since the Reynolds number was $Re = 5000$, the flow was turbulent and the $k - \epsilon$ model was used. The speedup is displayed in the following table.

Case	Number of processors	Domain decomposition	Speedup	Efficiency
2D	1	1x1x1	1	100%
2D	2	2x1x1	2.0	100%
2D	4	2x2x1	3.1	78%
2D	8	4x2x1	5.7	71%
3D	1	1x1x1	1	100%
3D	4	2x1x2	4.3	108%
3D	8	4x1x2	8.3	104%

The figures in this table are calculated from the elapsed (wall) time to convergence. An important thing to note is that since the computational times are compared with the computational time of a non-decomposed grid, both domain decomposition and message passing effects are included.

8 Future modifications

Some modifications that may be made in the future are listed below.

- Domain decomposition methods will be tested in order to further increase the computational speed.
- It will be possible to split already decomposed gridfiles. Specifying maximum sizes of the domains and decomposing the domains that do not fit will load balance the computations.
- An interface to the commercial post-processor Ensight will be implemented.
- More general boundary conditions will be implemented.
- More turbulence models will be implemented.
- A domain may only have one (1) connection to another boundary. This may be fixed using PVM features.
- The memory allocation is the same for all domains, regardless of the local memory need. This should, if possible, be fixed. Messages are however as small as possible.
- It should be fairly easy to make the domain decomposition of the code sequential if needed.
- More error handling facilities (see appendix I) should be included.
- A coarse grid correction scheme should be tested in order to increase convergence rate.

9 Discussion

A finite volume code CALC-BFC (Boundary Fitted Coordinates) [4] has been extended with PVM (Parallel Virtual Machine) [7] for parallel computations of turbulent flow in complex multiblock domains. For grid generation, an interface to the commercial grid generator ICEM has been implemented. Taking advantage of the communication facilities of PVM, the interblock communication was fairly easy to implement and the code may be run in parallel on everything from NOW (Network Of Workstations) to distributed and shared memory supercomputers. The validation of the extension shows that the solution is unaffected by the modifications. When running in parallel, large problems has a linear speedup at least up to eight processors. The code will thus be very efficient in predicting complex three dimensional fluid flows.

References

[1] BLOSCH, E., SHYY, W., AND SMITH, R. The role of mass conservation in pressure-based algorithms. *Numer. Heat Transfer. Part B 24* (1993), 415–429.

[2] DAHLSTRÖM, S., NILSSON, H., AND DAVIDSON, L. Lesfoil: 7-months progress report by chalmers. Tech. rep., Dept. of Thermo and Fluid Dynamics, Chalmers University of Technology, Gothenburg, 1998.

[3] DAVIDSON, L. An introduction to turbulence models. Int.rep. 97/2, Thermo and Fluid Dynamics, Chalmers University of Technology, Gothenburg, 1997.

[4] DAVIDSON, L., AND FARHANIEH, B. CALC-BFC: A finite-volume code employing collocated variable arrangement and cartesian velocity components for computation of fluid flow and heat transfer in complex three-dimensional geometries. Rept. 92/4, Thermo and Fluid Dynamics, Chalmers University of Technology, Gothenburg, 1992.

[5] DOORMAAL, J., AND G.D.RAITHBY. Enhancements of the SIMPLE method for predicting incompressible fluid flows. *Num. Heat Transfer 7* (1984), 147–163.

[6] JOHANSSON, P., AND DAVIDSON, L. Modified collocated SIMPLEC algorithm applied to buoyancy-affected turbulent flow using a multigrid solution procedure. *Num. Heat Transfer, Part B 28* (1995), 39–57.

[7] NILSSON, H. A parallel multiblock extension to the CALC-BFC code using pvm. Int.rep. 97/11, Thermo and Fluid Dynamics, Chalmers University of Technology, Gothenburg, 1997.

[8] PATANKAR, S. *Numerical Heat Transfer and Fluid Flow*. McGraw-Hill, New York, 1980.

A Convergence criteria

The iterations terminates when the convergence criteria described below is fulfilled.

For the velocities, the residuals ($\mathrm{resor}(u_i)$) are calculated as

$$\mathrm{resor}(u) = \sum_{\text{all tasks}} \sum_{\text{all C.V.}} \left| a_P {u_i^*}_P - \left(\sum_{NB} a_{NB} {u_i^*}_{NB} + \left(P_W^* - P_E^*\right) A_{iP} + b_{u_i P} \right) \right|$$

For the continuity, the residual ($\mathrm{resor}(cont)$) is calculated as

$$\mathrm{resor}(cont) = \sum_{\text{all tasks}} \sum_{\text{all C.V.}} |\delta \dot{m}|$$

where

$$\begin{aligned} \delta \dot{m} &= (u_e - u_w)\, \Delta_y \Delta_z \\ &+ (v_n - v_s)\, \Delta_x \Delta_z \\ &+ (w_h - w_l)\, \Delta_x \Delta_y \end{aligned}$$

(i.e. the continuity error).
Velocity reference residuals ($\mathrm{reref}(vel)$) are chosen as the largest residual amongst all the velocities, i.e.

$$\mathrm{reref}(vel) = \max_{all\,iter.} \left(\mathrm{reref}(vel), \mathrm{resor}(u), \mathrm{resor}(v), \mathrm{resor}(w)\right)$$

A continuity reference residual ($\mathrm{reref}(cont)$) is chosen as the largest residual, for the continuity, i.e.

$$\mathrm{reref}(cont) = \max_{all\,iter.} \left(\mathrm{reref}(cont), \mathrm{resor}(cont)\right)$$

The largest relative residual at each iteration is then calculated as

$$\mathrm{resmax} = \max \left(\frac{\mathrm{resor}(u)}{\mathrm{reref}(vel)}, \frac{\mathrm{resor}(v)}{\mathrm{reref}(vel)}, \frac{\mathrm{resor}(w)}{\mathrm{reref}(vel)}, \frac{\mathrm{resor}(cont)}{\mathrm{reref}(cont)} \right)$$

where 'relative' indicates that

$$0 \leq \mathrm{resmax} \leq 1$$

The convergence criteria is fulfilled when

$$\mathrm{resmax} \leq sormax$$

i.e. when the largest residual has been reduced by a factor $sormax$, the discretized equations are considered solved. This convergence criterion has also been used in references [5, 7].

B Introduction to PVM

To parallelize the program, PVM (Parallel Virtual Machine), available on **netlib**[1], is used. All the information needed is available on **Internet**, but some information relevant to this work is summarized here.
PVM is a message passing system that enables a network of UNIX (serial, parallel and vector) computers to be used as a single distributed memory parallel computer. This network is referred to as the virtual machine and a member of the virtual machine is referred to as a host. PVM is a very flexible message passing system. It supports everything from NOW (Network Of Workstations), with inhomogeneous architecture, to MPP (massively parallel systems). By sending and receiving messages, multiple tasks of an application can cooperate to solve a problem in parallel. PVM provides routines for packing and sending messages between tasks. The model assumes that any task can send a message to any other PVM task, and that there is no limit to the size or number of such messages. Message buffers are allocated dynamically. So the maximum message size that can be sent or received is limited only by the amount of available memory on a given host.

B.1 Pvm troubleshooting

This section contains a list of some of the problems and solutions encountered during the implementation of the code. If any other problem occurs, read the manual, consult the information on **Internet** or send a question to the news group (`comp.parallel.pvm`).

- When a couple of `pvmfreduce` calls are placed directly after each other, there is a risk of losing information. Temporarily, this has been solved by calling `pvmfbarrier` between the calls to `pvmfreduce`.
- When starting PVM, a PVM-daemon file is temporarily placed in `/tmp/pvmd.<uid>`. If PVM has been abnormally stopped, this file continues to exist and prevents PVM from being restarted on that particular host. Remove the file and PVM will be able to start again.
- If the program has been abnormally stopped, there might be a number of hosts still running. When trying to run the program again, with the same `group`-name, the tasks will get inappropriate task id numbers. The program can still be run, but the execution will halt at the first communication since PVM can not find the appropriate hosts. By typing `reset` on a PVM command line, the remaining active hosts will disappear and enable the program to be run again.
- When spawning a task, it is always started in the HOME-directory. To make it possible for child tasks to find necessary files, some UNIX environment variables has to be exported. By writing `setenv PVM_EXPORT`

[1] `http://www.netlib.org/`. See also `http://www.epm.ornl.gov/pvm/pvm_home.html`

PVM_ROOT:PVM_DPATH[2] on a UNIX command line (or in .cshrc), the variables PVM_ROOT and PVM_DPATH will be exported when spawning. PVM_ROOT is the directory of the executables and PVM_DPATH is the path of the daemon.

- When reading/writing from/to files, the whole path has to be declared since the child tasks are started in the HOME-directory.

- To be able to run several PVM programs at the same time, on the same hosts, the group-name must be program-specific. The reasons for this is that: **1)** the groupname is included in the program-name, **2)** the task id numbers becomes wrong otherwise and **3)** the output is written to group-specific files (to avoid interference with other programs).

- For a PVM-program to work, the executable must have the same name as the *task*-string used when spawning the child tasks with pvmfspawn.

- When sending large messages, use 'setenv PVMBUFSIZE 0x1000000'.

- If there is not enough memory available, some of the domains may write out EOF and halt without any error message. This is a PVM feature.

C CALC-PVM working directories

CALC-PVM uses some direct paths to essential files. The root directory of CALC-PVM is ~/pvm3. The essential directories are displayed below.

- ~/pvm3/grids should contain the grid files grpgrid.topo and grpgrid.geo, where grp is the first three letters in the group name specified in setup.f.

- ~/pvm3/bin/SUNMP should contain the executable when compiling and running on a SUN Enterprise 10000 machine. The executable should be named calc_grp, where grp is the first three letters in the group name specified in setup.f.

- ~/pvm3/bin/SGIMP64 should contain the executable when compiling and running on an SGI Origin 2000 machine.The executable should be named calc_grp, where grp is the first three letters in the group name specified in setup.f.

- ~/pvm3/output will contain the output files after running the program.

[2] PVM_ROOT = ~/pvm3, PVM_DPATH = $PVM_ROOT/lib/pvmd

- `~/pvm3/output/restart` will contain the restart files after running the program.
- `~/pvm3/save.file` and `~/pvm3/stop.file` saves output or stops the execution when they contain the integer '1'. If they contain '0' they will do nothing.

D Preprocessing

There are two ways of creating the computational domain. It can be created as a single block or multiblock domain. To create a single block domain, the grid points can be calculated and saved and the boundary conditions can be set and saved (in the format described in appendices E and F) using any type of programming language. The domain may then be split into a number of equally sized smaller domains for parallel computations. This is described in appendix D.1. To create a multiblock domain, one usually has to take advantage of some commercial grid generation package. CALC-PVM utilizes the grid generator ICEM CFD/CAE to generate and save the grid and the boundary conditions in the correct format. This is described in section D.2. It is possible to use any grid generator as long as the output has the correct format.

D.1 Splitgrid

The computational grid can be generated using any programming language as long as the information is saved as defined in appendices E and F. If the computational domain is described using a single block grid CALC-PVM has an automatic blocking feature. When CALC-PVM encounters a single block grid it checks whether to split the domain into smaller approximately equally sized fractions as in figure D.1. The user decides if and how to split the single block domain using the variables `lnodes`, `mnodes` and `nnodes` in the ICEM_interface file. `lnodes` is the number of domains to create in the i-direction, `mnodes` is the number of domains to create in the j-direction and `nnodes` is the number of domains to create in the k-direction. `(lnodes, mnodes, nnodes)=(1, 1, 1)` will thus not split the single block domain and `(lnodes, mnodes, nnodes)=(2, 3, 4)` will split the single block domain into 2x3x4 = 24 equally sized smaller domains. The sub-grids produced are extended to overlap four control volumes (one boundary node and one dummy-node from each task) at inner boundaries. This ensures that every node is calculated and that second order accurate calculations can be performed at inner boundaries. These domains will then be calculated in parallel, using PVM.

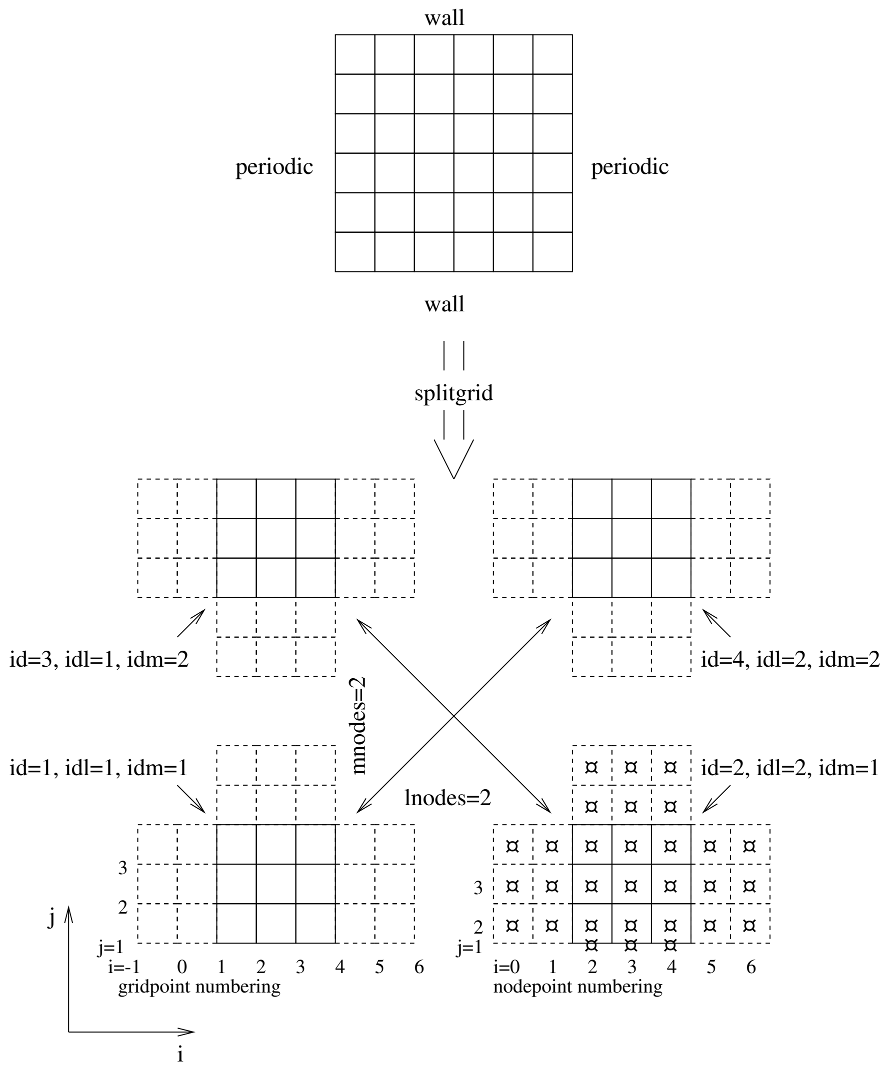

Figure D.1: Splitting the grid, in subroutine `ICEM_interface`

D.1.1 Setting boundary conditions

The boundary conditions are set in the topo-file, described in appendix E. It is very important to get the correct format to this file otherwise CALC-PVM will be unable to read it. When using splitgrid, the boundary conditions should be set as for a single block domain. There must be exactly one boundary condition set for each boundary surface family. Boundary conditions for lines or points are not recognized. The boundary conditions recognized by CALC-PVM are the following four-character strings: WALL (section 4.1), INLE (section 4.2), SYMM (section 4.3), OUTL (section 4.2), DEGE (section 4.5) and LID_ (moving boundary). These are still under implementation and more boundary conditions may easily be included. See the modify.f and ICEM_interface.f files for details.

D.2 ICEM interface

The commercial grid generator ICEM CFD/CAE can be used to generate the geometry and the multiblock grid and to set the boundary conditions. ICEM CFD/CAE is a very large package that includes everything from CAD geometry creation to computational grid generation and postprocessing. In the following sections some of its features are described. For further information please see the on-line manual delivered with the package.
Starting ICEM the 'ICEM Manager', from where all ICEM packages can be reached, will be displayed. Use the following packages to create a computational grid and to set the boundary conditions from scratch:
DDN - To create the geometry.
DDN Mesher Interface - To organize the geometry before meshing.
HEXA - To create blocking, grid and to set the boundary conditions.

D.2.1 Create geometry

Starting a new CFD project a geometry has to be created. It can be created in any CAD system and imported to ICEM in IGES format or it can be created inside ICEM directly using the built-in CAD package DDN.
If the geometry is described as point, line or surface coordinates they can be imported to ICEM using the Input menu in the Manager.
When the geometry has been created make sure that all entities are converted to B-spline curves and surfaces before saving the geometry and quitting DDN.

D.2.2 Create families

Using the package DDN Mesher Interface the geometry is organized in families. The entities to organize must be B-spline curves and surfaces. Later, grid specifications and boundary conditions will be assigned to these families. Remember to add a family for the fluid. It shall not contain any geometrical entities but it will be used in the grid generator to define the interior of the computational domain. Note that the family setup must be saved in the GPL window before quitting the DDN Mesher Interface.

D.2.3 Create blocks

Inside Hexa, the blocking of the domain is specified. It is saved with 'file/save blocking' before quitting Hexa or continuing with the mesh generation.

D.2.4 Setup grid parameters

Inside Hexa, grid parameters are assigned to the families defined in DDN Mesher Interface. The blocking should be projected to the surfaces when computing the grid. Save the blocking and the grid parameters using 'file/save blocking' before quitting Hexa or continuing with setting the boundary conditions.

D.2.5 Set boundary conditions

Inside Hexa, boundary conditions are assigned to the families defined in DDN Mesher Interface. This is done under 'file/set boco'. At this point the complete boundary conditions are set by typing a four character string in the STRING1 box. Do not touch anything else. There must be exactly one boundary condition set for each boundary surface family. Boundary conditions for lines or points are not recognized. The boundary conditions recognized by CALC-PVM are the following four-character strings: WALL (section 4.1), INLE (section 4.2), SYMM (section 4.3), OUTL (section 4.2), DEGE (section 4.5) and LID_ (moving boundary). These are still under implementation and more boundary conditions may easily be included. See the modify.f and ICEM_interface.f files for details. Periodic boundary conditions should be set both in the DDN Mesher Interface, under modals/periodicity, and in Hexa, under Blocking/periodic nodes. They are saved and treated as interblock connections.
Save the blocking, the grid parameters and the boundary conditions using 'file/save blocking' and save the grid with multiblock and boundary condition information in ICEM format using 'file/multiblock' before quitting Hexa.

D.2.6 Saving input to CALC-PVM

To write input files to CALC-PVM do the following in the ICEM Manager:
Chose translator; Multiblock Info.
Write Input. Answer yes to the default save options.
Open the transfer shell. Two files has been created; info.topo and info.geo. Move these to '∼/pvm3/grids' and rename them to grpgrid.topo and grpgrid.geo, where grp stands for the first three characters in the group name defined in setup.f.
Now, the computational domain is ready to be used.

E The topology file format

The multiblock-info file format of ICEM CFD/CAE is used for the input files. This file should be divided into three sections. The first section lists the name and range of all domains composing the mesh. The second section describes the connectivity of each block with the rest of the grid. Finally, the boundary conditions associated with each domain are listed in the last section. The format is as follows.

```
# block name imin jmin kmin imax jmax kmax

domain.1 imin jmin kmin nim1 njm1 nkm1
domain.2 imin jmin kmin nim1 njm1 nkm1
dom ...
⋮

# Connectivity for domain.1

conn_no block_name orientation type beg1 beg2 beg3 end1 end2 end3
conn_no adjac_name orientation type beg1 beg2 beg3 end1 end2 end3
conn_no block_name ori ...
⋮

# Connectivity for domain.2

conn_no block_name orient ...
⋮

# Connectiv ...
⋮

# Boundary conditions and/or properties for domain.1

flag1 flag2 type imin jmin kmin imax jmax kmax
⋮

# Boundary conditions and/or properties for domain.2

flag1 fl ...
⋮

# Boundary condi ...
⋮
```

where

- The domain names (`domain.x`) should count from x=1 to the number of domains. They do not have to be in the correct order, but they will have to be in the same order in the geometry file and the topology file
- The lines starting with # and the empty line following are ignored (CALC_PVM skips two lines for comments of this, or any other, form)
- `imin`, `jmin` and `kmin` are minimum indices in the i, j and k directions
- `imax`, `jmax` and `kmax` are maximum indices in the i, j and k directions
- `conn_no` is the connectivity number of each connection
- `adjac_name` is the name of the domain sharing the entity
- `orientation` is the index type (+ or - i, j or k) of directions 1, 2 and 3. For example, if the first direction corresponds to growing k indices, the second direction to decreasing i indices and the third direction corresponds to j indices which are constant with its lowest value on the plane, the orientation should be " k-i-j"
- `type` is the topological type; b=block, f=face, e=edge and v=vertex. Only faces are relevant here, so lines containing other than type f is ignored
- `beg1, beg2, beg3` are the starting indices in direction 1, 2 and 3
- `end1, end2, end3` are the ending indices in direction 1, 2 and 3
- `flag1, flag2` are character strings attached to the topological entity. If `flag1` starts with an asterisk, the line is treated as a comment.

Regarding the exact Fortran format, have a look in ICEM_interface.f.

F The geometry file format

The multiblock-info file format of ICEM CFD/CAE is used for the input files. The geometry file should contain the range and list of grid coordinates for each domain of the grid according to:

```
domain.1 nim1 njm1 nkm1
xc yc zc
```

⋮

```
domain.2 nim1 njm1 nkm1
xc yc zc
⋮
dom...
⋮
```

where the block names (domain.x) should count from $x = 1$ to the number of domains. They do not have to be in the correct order, but they will have to be in the same order in the geometry file and the topology file. nim1, njm1 and nkm1 are the number of grid points in the i, j and k directions for each domain. xc, yc and zc are the grid cartesian coordinates. Note that i is the fastest running index, then j, then k.
Regarding the exact Fortran format, have a look in ICEM_interface.f.

G The connection matrix innerbound

Syntax: innerbound(ibound,info)
In order to keep track of the connectivity between the domains there is an integer matrix containing all the information necessary for the inter block communication. This matrix is called innerbound. For each line in innerbound the connectivity for a connection to another domain is described. This connection may be an entire side of the domain as well as a smaller part of a side of the domain. The columns of the matrix contains the information for each connection according to the list specified below, where the numbers refer to the integer info.

info	
1	The domain number of the neighbour.
2-3	Change in characteristic theta- and phi-angles between planes[3].
4	Send-message tag, when two blocks have more than one connection[3].
5	Receive-message tag, when two blocks have more than one connection[3].
6-7	Not used.
8-10	The low i/j/k grid index, when sending one plane.
11-13	The high i/j/k grid index, when sending one plane.
14-16	The beg i/j/k grid index, when receiving one plane.
17-19	The end i/j/k grid index, when receiving one plane.
20-22	The low i/j/k grid index, when sending two planes.
23-25	The high i/j/k grid index, when sending two planes.
26-28	The beg i/j/k grid index, when receiving two planes.
29-31	The end i/j/k grid index, when receiving two planes.
32-34	Not used.
35-37	1 on tangential indexes, 0 on normal index.
38	Low / high boundary, -1 / 1.

39	The plane of the boundary, i=1, j=2, k=3.
40-42	Receiving index corresponding to sending i/j/k
43-45	Connectivity corresponding to receiving i/j/k
46-69	As 8-31, but for node indices.

H The boundary condition matrix `outerbound`

Syntax: `outerbound(ibound,info)`
The boundary conditions of each domain are kept in a matrix called `outerbound`. For each line in `outerbound` a boundary condition is described. This boundary may be an entire side of the domain as well as a smaller part of a side of the domain. The columns of the matrix contains the information for each boundary according to the list specified below, where the numbers refer to the integer `info`.

`info`	
1	Type of boundary condition. 1 = 'WALL' 2 = 'SYMM' 3 = 'LID_' 4 = 'INLE' 5 = 'OUTL' 6 = 'INL1' 7 = 'DEGE'
2	Indexplane.
3	low / high (-1 / 1) boundary.
4- 6	Boundary min-values of i/j/k.
7- 9	Boundary max-values of i/j/k.

I Error handling facilities

Since the code utilizes PVM, an error occuring in one of the blocks will not be sensed by the other blocks. Typing out an error message and simply stop the execution would deadlock the other blocks, waiting for messages from the exiting domain. Using a flag (`ierror`) that is zero when everything is alright and negative when an error has occurred, the error message may be typed and all the domains will leave the computations simultaneously. In this way there will be no executables left running on the machine.
Some of the errors that are controlled are listed below.

- No periodicity is yet allowed when splitting grid.

[3] This is working, but the ICEM CFD/CAE topo file format for this is yet unclear.

- Boundary conditions or connections to another domain may only be set to an entire domain face or subface. Boundary conditions or connections to another domain for lines or points are not allowed.

- The number of grid points in each direction in each domain must be larger than four.

- The parameters `it/jt/kt` used to set the dimensions of all variables has to be at least the largest value of `nim1+2/njm1+2/nkm1+2` amongst the domains.

- The parameters `nphit/nphito` used when setting the dimension of the `phi/phio` variables has to be as large as the number of variables that are solved for.

- Every boundary node of the domain must have exactly one connection to another domain or a boundary condition.

- The send buffers in `dummynodes.f`, `exch_vel.f` and `exchange.f` has to be large enough.

- Some boundary condition errors are handled in `modify.f`.

- The correct syntax when calling some subroutines are checked.

More of these error handling facilities can be added by writing out an error message and setting `ierror`=-1, which will stop the execution of the program.

J Postprocessing

To view your results in Tecplot, move to ~`/pvm3/output`. Here you will find several output files on the form:
`grpid001.dat`
`grpid002.dat`
`grpid003.dat`
`grpid004.dat`
where `grp` is the first three letters in the group name specified in `setup.f`. Type '`cat grpid???.dat > grp.dat`' to concatenate them and '`preplot grp.dat`' to prepare for tecplot following with '`tecplot.7.0.2 grp.plt`' to view the results.