A Parallel Domain Decomposition Algorithm for 3D Turbulence Modeling

G. Bärwolff
TU Berlin, Hermann-Föttinger-Institut
Berlin, Germany

H. Schwandt
TU Berlin, Fachbereich Mathematik
Berlin, Germany

Abstract

In this paper we present a parallel procedure for the solution of large laminar and turbulent flow problems. The Navier-Stokes equations for an incompressible fluid are treated by finite volume method. The momentum equations are discretized in time explicitly. By a domain decomposition strategy we obtain a highly parallel method which scales well on massively parallel systems. The resulting linear subsystems are solved by a modified iterative method based on a red-black ordering. We describe the parallelization strategy and its implementation in several parallel environments. Numerical results from several vector and parallel computers are included.

Keywords: turbulence, Navier-Stokes equations, domain decomposition, parallel methods

1 Introduction

The numerical simulation of three-dimensional turbulent flow is one of the great challenges in Computational Fluid Dynamics. In addition to the obvious mathematical problems the treatment of realistic problems requires the solution of extremely large systems of equations resulting from the discretization of the underlying time-dependent nonlinear PDE in three space dimensions. Both the arithmetic complexity and the memory requirement of numerical solution procedures approach or even exceed the limits of modern supercomputing. In this paper we present a finite volume method for the numerical solution of the nonstationary 3D Navier-Stokes equations. The starting point for our work have been an approach and its numerical realization in a code MLET both developed at the Universität der Bundeswehr in Munich [1]. In an interdisciplinary research project this approach will be developed further under several aspects. The ultimate goal consists in the modeling of turbulent shear flow by a large eddy simulation (LES, see [4], e.g.). Direct numerical simulation (DNS) will be used where applicable for comparisons. In the present context we focus on the parallelization of the method which is indispensable in view of the enormous numerical complexity. To perform calculations for realistical Reynolds numbers, three dimensional grids with many millions of cells are necessary. Running problems of this size leads to very large turnaround times (weeks or months) in existing production environments.

In this paper we describe as a first step a simple but efficient domain decomposition method which leads to highly parallel numerical algorithms. The parallelization strategy is governed by the need that any code developed from the solution method should be reasonably portable. On the one hand, one aim of the current work consists in the development of parallel methods which are intended to be tested in different parallel computing environments. On the other hand, the fluid physical problems of the project are treated at several sites. Therefore, the adaption to different computing environments like vector computers and parallel systems ranging from workstation clusters to parallel vector systems and massively parallel computers using (virtual) shared memory or message passing strategies should be feasible with a tolerable amount of work.

The paper is organized as follows. In section 2 we briefly describe the mathematical model and the principles of the (sequential) solution method. In section 3 we describe the physical background for a reference problem. The numerical solution of this problem gives a good insight into the complexity of a realistical application. In section 4 we describe the domain decomposition method and the general principles of parallelization applied in this context. In section 5 we report the performance results of several numerical experiments on Cray vector computers and the Cray MPP system T3D using message passing. In the conclusion we give an outlook on future work.

2 The mathematical model and the solution method

The basic equations for the description of laminar as well as turbulent incompressible flow problems are the Navier-Stokes equations

$$\frac{\partial u_i}{\partial t} + \frac{\partial}{\partial x_j}(u_i u_j) = -\frac{\partial p}{\partial x_i} + \frac{\partial S_{ij}}{\partial x_j} \quad , i,j = 1,2,3, \tag{1}$$

and the continuity equation

$$\frac{\partial u_j}{\partial x_j} = 0 \tag{2}$$

on a domain $\Omega \times [0,T], \Omega \subset \mathrm{I\!R}^3, T > 0$, with appropriate initial and boundary conditions. Equations (1) and (2) may be discretized in space dimensions by a finite volume method on staggered grids for the velocity components $\vec{u} = (u,v,w) = (u_1,u_2,u_3)$, the pressure p and the strain rate tensor $S_{ij} = \frac{\nu}{2}(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i})$. ν is the effective or molecular viscosity. In the sequel we write the equations (1) and (2) in the short operator form

$$\frac{\partial \vec{u}}{\partial t} + K(\vec{u}) = -\nabla p + D(\vec{u}) \quad , \tag{3}$$

and

$$\nabla \cdot \vec{u}_h = 0\,. \tag{4}$$

The space discretization is done by a finite volume method on staggered grids. The details of this discretization method are described in [1] and [6]. Four staggered finite volume grids are used to discretize the three components of the momentum equation and the continuity equation. For convenience, we describe as an example the discretization of the continuity equation. Let Ω_h be the grid for the continuity equation and c_{ijk} a grid-cell. We integrate the equation (2) over c_{ijk} and, applying the Gauss theorem (5), we get

$$\int_{c_{ijk}} \frac{\partial u_j}{\partial x_j} \quad d\,c_{ijk} \quad = \quad \int_{c_{ijk}} \nabla \cdot \vec{u} \quad d\,c_{ijk} \quad = \int_{\gamma(c_{ijk})} \vec{u} \cdot \vec{n} \quad d\gamma(c_{ijk}) \quad = \quad 0\,, \tag{5}$$

where $\vec{n}$ denotes the outer normal vector and $\gamma(c_{ijk})$ the boundary of c_{ijk}. In the case of a regular quadrilateral cell c_{ijk}, we approximate the values of $\vec{u}\cdot\vec{n}$ on the cell boundaries by values at the centers representing the mean values of the areas of the boundary faces, and from (5) we get the finite volume discretization

$$\frac{u_{i+\frac{1}{2},j,k} - u_{i-\frac{1}{2},j,k}}{\Delta x_i} + \frac{v_{i,j+\frac{1}{2},k} - v_{i,j-\frac{1}{2},k}}{\Delta y_j} + \frac{w_{i,j,k+\frac{1}{2}} - w_{i,j,k-\frac{1}{2}}}{\Delta z_k} \quad = \quad 0\,,$$

shortly described as

$$\nabla_h \cdot \vec{u}_h \quad = \quad 0\,. \tag{6}$$

Figure 1 illustrates the position of the velocity components on the vertices of a the finite volume cell c_{ijk}.

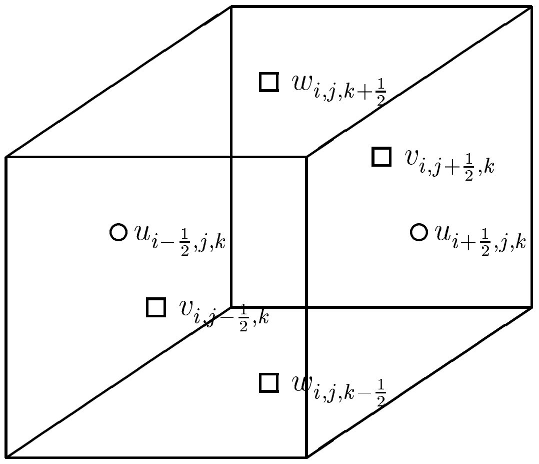

Figure 1: *finite volume discretization of a cell* c_{ijk}

Summarizing the results of the finite volume discretization we get the following system of ode's for the velocity components at every grid point:

$$\frac{\partial \vec{u}_h}{\partial t} + K_h(\vec{u}_h) \quad = \quad -\nabla_h p_h + D_h(\vec{u}_h) \quad , \tag{7}$$

with the restriction $\nabla_h \cdot \vec{u}_h = 0$. K_h, $\nabla_h p_h$ and D_h are the discrete operators resulting from the finite volume discretization of the convective terms, the pressure gradient and the viscous terms of the momentum equation.

The discretization is of second order ($O(h^2)$). It is conservative in the sense that the mass, the momentum and the kinetic energy are conserved by the finite volume form of the model equations. The time integration is carried out either by a leapfrog method or an Adams-Bashforth method. Thus, we have to solve in every time step the system of equations

$$\frac{\vec{u}_h^{n+1} - \tilde{\vec{u}}_h}{\tau} = -\beta \nabla_h \delta p_h^{n+1} \quad , \quad \nabla_h \cdot \vec{u}_h^{n+1} = 0\,, \tag{8}$$

where $\delta p_h^{n+1} = p_h^{n+1} - p_h^n$ and where $\tilde{\vec{u}}_h$ is a result from an explicit predictor step. τ is the time step and β is a constant depending on the time integration method. In the case of a leapfrog method we get $\tilde{\vec{u}}_h$ from the equation

$$\frac{\tilde{\vec{u}}_h - \vec{u}_h^{n-1}}{2\tau} + K_h(\vec{u}^n) = \nabla_h p_h^n + D_h(\vec{u}^{n-1}) \tag{9}$$

with $\beta = 2$ (see also [1]). For the Adams-Bashforth method we note

$$\frac{\tilde{\vec{u}}_h - \vec{u}_h^n}{\tau} + 2K_h(\vec{u}^n) - K_h(\vec{u}^{n-1}) = \nabla_h p_h^n + 2D_h(\vec{u}^n) - D_h(\vec{u}^{n-1}) \tag{10}$$

with $\beta = 1$. Thus, both time integration methods are explicit and of second order.

The differentiation of the equations (8) yields

$$-\Delta_h \delta p_h^{n+1} = -\frac{1}{\tau\beta} \nabla_h \cdot \tilde{\vec{u}}_h \quad . \tag{11}$$

There are two alternatives for the determination of the approximations $(\vec{u}_h^{n+1}, p_h^{n+1})$ in every time step. They can be found either by the solution of a Poisson equation (11) for p_h^{n+1} followed by an explicit fill-in step to get $\vec{u}_h^{n+1}$ from (8) or iteratively. Both possibilities have been considered in [1],[2]. In the present context, we restrict the discussion to the iterative solution of a linear system of equations of the form

$$\begin{pmatrix} E_x & \bar{0} & \bar{0} & -C_x \\ \bar{0} & E_y & \bar{0} & -C_y \\ \bar{0} & \bar{0} & E_z & -C_z \\ D_x & D_y & D_z & \bar{0} \end{pmatrix} \begin{pmatrix} u_h^{n+1} \\ v_h^{n+1} \\ w_h^{n+1} \\ \delta p_h^{n+1} \end{pmatrix} = \begin{pmatrix} r_u \\ r_v \\ r_w \\ r_p \end{pmatrix} . \tag{12}$$

The system of equations (12) constitutes the matrix form of (8). E_x, E_y and E_z are diagonal matrices, C_x, C_y and C_z are sparse matrices resulting from the discretization of the ∇_h operator. Analogously, D_x, D_y and D_z are sparse matrices resulting from the discretization of the $\nabla_h \cdot$. The components r_u, r_v, r_w and r_p on the righthand side depend on $\tilde{\vec{u}}_h$ and on the boundary conditions.

3 The fluid physical background and a reference problem

The fluid physical background can be briefly described as follows. The principal goal of the present work consists in the development of a conceptual and algorithmic framework for numerical simulations of turbulent flow with high resolutions either not using any turbulence model (Direct Numerical Simulation) or using subgrid scale models (Large Eddy Simulation). The practical aim of the current simulation of a backward facing step problem consists in the investigation of a possible reduction of the the drag or of the energy loss (reduction of the production of turbulence energy, decreasing of recirculation zones) by special manipulations of the flow (acoustic manipulations/loudspeaker, other forms of blowing or suction of boundary or shear layers) and, in particular, the reduction of the recirculation zone behind the step. A parameter study is carried out in order to investigate the behavior of the flow depending on the amplitude of the manipulation disturbances with a fixed frequency.

As a reference model, we consider a rectangular backward-facing step channel. The acoustic manipulation is realized with disturbance (loudspeaker in front of the seperation edge with 50 Hz). The Reynolds number calculated using the step height is approximately equal to 3000. In the spanwise direction periodic boundary conditions are used. The inflow profile is a block profile with $u_1 = u_x = 1$ and $u_2 = u_3 = 0$. At the bottom of the channel the no-slip boundary condition is assumed. At the top of the region boundary conditions like $\frac{\partial \vec{u}}{\partial \vec{n}} = 0$ with the outer normal vector $\vec{n}$ are used. The outflow boundary condition is set to $\frac{\partial \vec{u}}{\partial x} = 0$.

The structured grid consists of 516 cells in the X-direction, 132 cells in the Y-direction (spanwise) and 164 cells in the Z-direction. This fine grid allows us to perform a DNS and thus a subgrid-scale model is not necessary. This reference example is, therefore, suitable for future comparisons of LES and DNS.

The figures 2 and 3 show results for the backward facing step problem as isoline plots of an instantaneous velocity field and the mean velocity field (statistic of first order) in an X-Z cut (symmetry plane, near bottom region).

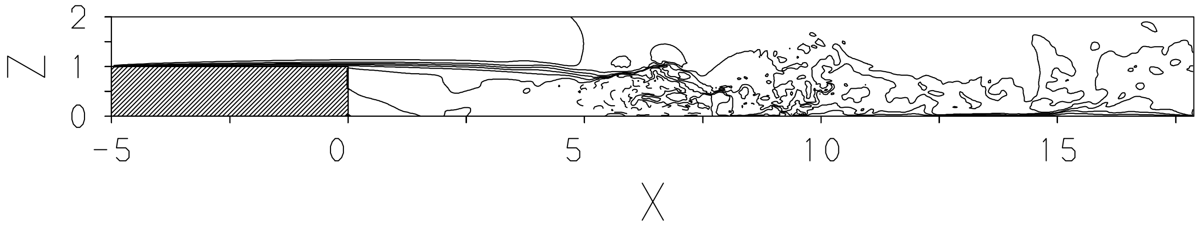

Figure 2: *velocity isolines of* u_1, $t = 180$

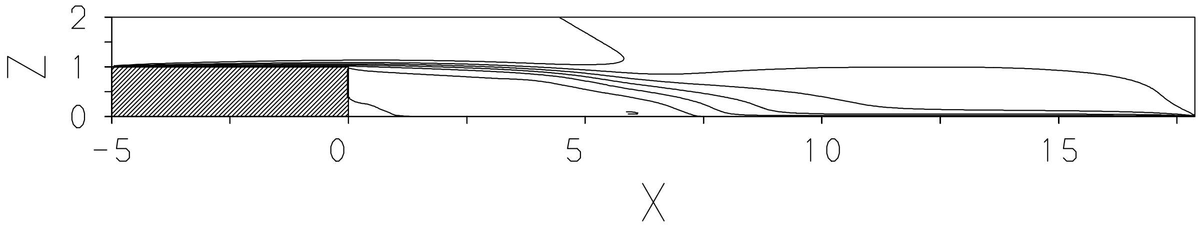

Figure 3: *velocity isolines of* $\overline{u_1}$, $t = 180$

In order to postprocess the results for a comparison with the results of practical experiments smooth first and second order statistic ($\overline{\vec{u}}$, $\overline{u_i{}' \, u_j{}'}$, and other) are necessary. For the latter we need a few hundreds of thousands flow realizations or instantaneous flow fields or time steps respectively. We, therefore, have to choose the dimensionless constant T for the time interval approximately four times larger than the dimensionless length of the channel, i.e. in the present example $T = 60$. Using 64 processors on a CRAY T3D a chain of production jobs for the above problem takes about 600 CPU hours, which was achieved in two months. Due to the large amount of required main memory, the reference job could not run on the available Cray Y-MP4/464. Simulations of smaller problems with approximately 2 million grid points took about 10 months on a Cray Y-MP using the original MLET code. At present, we run a problem with 10^8 grid points. As this problem requires about 15 GB of main memory, 256 processors are needed on a Cray T3D, i.e. almost the whole memory of the available environment.

4 The parallelization

The original (sequential) version of MLET [1] uses a simple scheme for the pressure velocity iteration for the solution of the system of equations (8). The velocities are updated sequentially cell by cell: for the update of the velocity component u, v, w, resp., of the cell $c_{i,j,k}$ that of the cells $c_{i-1,j,k}$, $c_{i,j-1,k}$, $c_{i,j,k-1}$, resp., must be available. In the case of cells like $c_{0,j,k}$ or $c_{i,0,k}$ boundary conditions are used.

This process is in some sense similar to a single step or Gauss-Seidel like procedure. This method has been chosen for the current implementation because, on the one hand, of the necessity of a rapidly available production code. On the other hand, this method can be competitive as experience has shown that in each time step the iteration can be stopped after a small number of steps in the current applications.

The updating of cell $c_{i,j,k}$ can be roughly described by the following code fragment.

```
update_cell(i,j,k) :
  dp         = delta_p(i,j,k)
  p(i,j,k)   = p(i,j,k)   + dp
  u(i,j,k)   = u(i,j,k)   + du(dp)
  u(i-1,j,k) = u(i-1,j,k) - du(dp)
  v(i,j,k)   = v(i,j,k)   + dv(dp)
  v(i,j-1,k) = v(i,j-1,k) - dv(dp)
  w(i,j,k)   = w(i,j,k)   + dw(dp)
  w(i,j,k-1) = w(i,j,k-1) - dw(dp)
```

The variable dp is defined by

$$dp = -[\frac{\partial \nabla_h \cdot \vec{u}_h}{\partial p_h}]^{-1} \nabla_h \cdot \vec{u}_h, \tag{13}$$

where $\frac{\partial \nabla_h \cdot \vec{u}_h}{\partial p_h}$ depends only on grid parameters (see also [5], [6]). du, dv and dw are proportional to dp. We note

$$du(dp) = \tau dp/\Delta x_i, \quad dv(dp) = \tau dp/\Delta y_j \quad \text{and} \quad dw(dp) = \tau dp/\Delta z_k.$$

The pressure correction $dp = \delta p_h$ is proportional to the divergence of the velocity field and will be updated using the actual velocity components from (13).

One step of the iteration in any of the time steps can now be expressed by the following loop. For simplicity we do not discuss the updating of the non-Dirichlet boundary conditions.

```
  do i=1, N_x
  do j=1, N_y
  do k=1, N_z
      update_cell(i,j,k)
```

In each time step, the iteration is stopped if the condition

$$\max_{1\leq i\leq N_x, 1\leq j\leq N_y, 1\leq k\leq N_z} |(\delta p_h)_{i,j,k}| \leq \epsilon \tag{14}$$

is fulfilled. In the vector version of MLET [1], the k-loop is splitted into two loops with stride 2 due to the recurrrence in the update of $w_{i,j,k}$, $w_{i,j,k-1}$.

The parallelization is based on a domain decomposition. Although we have to treat a three dimensional problem we define a two dimensional domain decomposition in the x- and y-coordinates. On the one hand this simplifies the communication, on the other hand we keep the subproblem size fixed in one (z) dimension. This can be useful in view of an optimization for a single CPU of a parallel system: we maintain a rather large length of inner loops, i.e. a large vector length (which is important in the case of vector or RISC processors) independently of the number of processors. For simplicity we assume a regular grid and Ω to be a cube of dimension $N_x \times N_y \times N_z$ cells. We further assume a two dimensional processor topologie with $P_x \times P_y$ processors. We partion Ω into $P_x \times P_y$ cubes. For simplicity, we assume that P_x and P_y divide N_x and N_y, resp. Then each cube is of size $N_x/P_x \times N_y/P_y \times N_z$. Figure 4 illustrates the above partition.

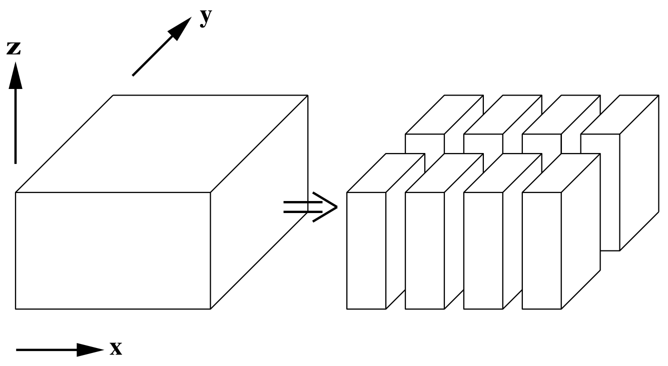

Figure 4: *domain decomposition*

The partitioning introduces artificial boundaries in the x and y directions. Each artificial boundary is a rectangle of size $N_y/P_y \times N_z$ or $N_y/P_y \times N_z$ in the $x-z$ or the $y-z$ plane, resp. The partition is overlapping as the last (first) plane of a subdomain is an artificial boundary plane of that subdomain and at the same time the first (last) inner plane of the neighbouring subdomain. In this paper we only consider this minimal overlap. It is, however, conceptually easy to define larger overlaps. Figure 5 illustrates the overlap principle for a two dimensional cut in the $x-y$ plane for the case of four subdomains.

Each processor updates all cells he has been assigned. At the artificial boundaries he needs data from neighbouring processors. The above mentioned sequential pressure velocity iteration cannot be parallelized. To eliminate the recurrencies, the computation is carried out in two passes, the first for the "black" cells and the second for the "white" cells. The following program fragment shows the this parallel code modification of a pressure velocity iteration step:

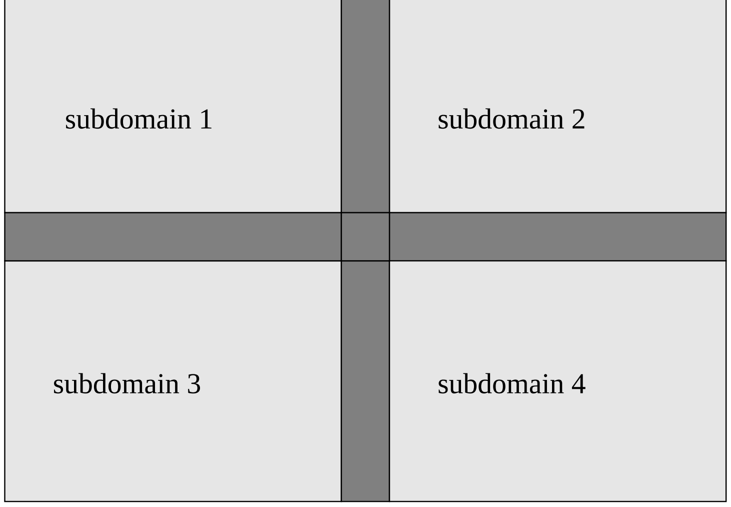

Figure 5: *overlapping domain decomposition in the $x - y$ plane*

```
do i_x = 1,P_x
do j_y = 1, P_y parallel on P(ix,iy)

  do i = (i_x-1)*N_x/P_x+1,i_x*N_x/P_x,2
    do j = (j_y-1)*N_y/P_y+1,j_y*N_y/P_y,2
      do k = 1,N_z
        update_cell(i,j,k)
    exchange_boundaries_y(v)
    do j = (j_y-1)*N_y/P_y+2,j_y*N_y/P_y,2
      do k = 1,N_z
        update_cell(i,j,k)

  exchange_boundaries_x(u)

  do i = (i_x-1)*N_x/P_x+2,i_x*N_x/P_x,2
    do j = (j_y-1)*N_y/P_y+1,j_y*N_y/P_y,2
      do k = 1,N_z
        update_cell(i,j,k)
    exchange_boundaries_y(v)
    do j = (j_y-1)*N_y/P_y+2,j_y*N_y/P_y,2
      do k = 1,N_z
        update_cell(i,j,k)

  exchange_boundaries_x(u)
```

The subroutines *exchange_boundaries_x* and *exchange_boundaries_y* manage the transfer of the boundary information between two processors. Communication is essentially restricted to these two routines (except for the test of the global convergence criterion). Each call to *exchange_boundaries_x* and *exchange_boundaries_y*, resp., induces a transfer of a hyperplane with $N_x/P_x * N_z$ values of u and $N_y/P_y * N_z$ values of v, resp. Although we only need roughly half of that number of points (either "black" or "white" values), we prefer to send the respective whole hyperplane for reasons of efficiency. In message passing implementations (see below) the sending of only the really needed values leads to a stride of 2 in the messages and to rather complicated case distinctions.

The above considerations describe almost completely the relatively simple parallization which we use as a basic version for future developments. Under the above condition on the geometry and an ideal domain decomposition into equally sized subdomains we obtain a perfect load balancing.

5 Implementation details and numerical results

The original MLET code [1] is a vector code. It can, therefore, be easily modified for multiprocessor systems with shared memory. In view of the intended very large production runs on MPP systems, we prefer explicit message passing to (virtual) shared memory models. The parallel code is organized according to an SPMD model, i.e. we run the same program on each processor. As a consequence we can run the code on any number of processors, i.e. also on one processor, without any modification. Every processor also handles restart and plot files related to the data of his domain. Simple pre- and postprocessing steps have been implemented which split the global files into domains and vice versa before and after running a parallel program.

We have implemented two parallel versions for explicit message passing on the Cray T3D. The sequential code has been transformed for both in 4 weeks. Using conditional compilation we can maintain several program versions (vector, PVM, `shmem` etc.) in one general code.

In the `shmem` version message passing is carried out by the Cray specific `shmem` routines. Instead of `send` and `receive` operations `put` and `get` routines are implemented which read from and write directly into the memory of the respective other processor without any synchronization. The latter is done where necessary by calls to a `barrier` routine. In order to avoid cache inconsistencies, further routines flushing the cache have to be called.

The PVM version uses Cray PVM, a PVM version for the T3D. The user interface widely corresponds principally to that of the common public domain version. Cray PVM requires the SPMD model and does not allow for more than one process per processor. In contrast to public domain PVM a process runs on each processor automatically after the start of the program, i.e. processes cannot and must not be spawned. Cray PVM is highly optimized for the hardware of the T3D. In particular, the implementation of the `send` and `receive` operations is based on calls to the above mentioned `shmem_get` and `shmem_put` routines. Therefore, we cannot expect the PVM version to be more efficient than the `shmem` version which is optimal for the T3D. Nevertheless, the PVM version is of great interest for the reason of the intended portability.

The performance of the parallel code is illustrated for a reference job derived from the reference model described in section 3 (three dimensional backward facing step flow problem with periodic boundary conditions). We consider a FV-grid with a size of $516 \times 132 \times 164$ or 11083776 cells, i.e. a system of equations with approximately 45 millions of unknowns has to be solved for every time step. We apply a domain decomposition with minimal overlap. In the reference job the outer iteration is always stopped after exactly 100 time steps. For the inner iteration which is controlled by the convergence criterion (14) with $\epsilon = 10^{-4}$ we observed an average of 25 iterative steps with a very small variation. The test job requires approximately 1.3 gigabytes of main memory. Therefore, on a Cray T3D 32 processors are the minimal possible configuration.

Table 1 illustrates the performance of the parallel algorithm on several Cray systems: T3D (Konrad-Zuse-Zentrum für Informationstechnik (ZIB), Berlin, 256 processors, each with 64 MB of main memory), J90 (ZIB, 16 processors, 4 GB of main memory), C90 (Cray Research, Eagan/Minn., 16 processors, 4 GB of main memory). For the T3D we report results for the version using Cray specific `shmem` routines, for the J90 and the C90 runs we used the Cray shared memory autotasking concept for parallelization. For the T3D we also note the 2D processor topology used in the respective test. The results show that in this application 9 RISC processors of the T3D are roughly equivalent to one J90 vector CPU and 25 T3D processors are equivalent to one C90 vector CPU. Obviously the C90 has the most powerful processor. The results show, however, that, only on an MPP system like the Cray T3D the large speedups can be expected which are necessary to make feasible the simulation of realistic, i.e. very large CFD problems. A significantly increasing number of processors seems to be more promising than the comparatively slow development of the performance of vector processors. This assumption is only valuable for codes like the one we discuss here which show reasonable speedup factors for an increasing number of processors while the problem size is kept constant. Obviously the efficiency will degrade in this example for more than 16 processors on the J90 and for more than 256 processors on the T3D. On the other hand, larger MPP systems like a Cray T3D with 512 or 1024 processors would enable us to increase the problem size accordingly, which would be the principally interesting case. All computing times are wall clock times. The tests on the C90 and J90 have been carried out in dedicated mode.

Table 1: *performance on different parallel systems*

system	#procs		time t [sec]	mflops	t_{C90}/t	$S_p = t_{J90(1)}/t$
J90	1		16428	92	0.27	1.00
J90	4		4173	357	1.03	3.94
J90	8		2244	673	1.95	7.32
J90	16		1390	1087	3.15	11.81
C90	1		4380	338	1.00	
system	#procs	proc$_X$ $*$ proc$_Y$	time t [sec]	mflops	t_{C90}/t	$S_p = t_{T3D(32)}/t$
T3D	32	8 * 4	3329	443	1.31	1.00
T3D	64	8 * 8	1671	886	2.62	1.99
T3D	128	16 * 8	871	1700	5.03	3.82
T3D	256	16 *16	437	3387	10.02	7.61

The results from the T3D reveal a noticeable degradation of the efficiency from 64 to 128 processors. This is most probably a cache coherence problem. In view of the intended portability we did not perform any specific single CPU optimization.

Figure 6 illustrates again the development of the speedup on a T3D for an increasing number of processors. As a reference the results for one and four CPU's on a C90 are indicated.

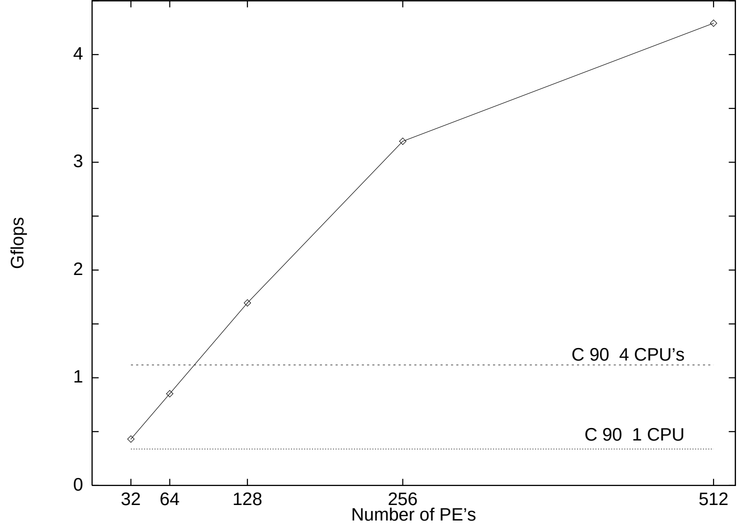

Figure 6: *speedup on the Cray T3D*

The following Table 2 shows a comparison of the PVM and the `shmem` versions on the T3D for several configurations.

Table 2: *performance of the* `shmem` *versus the PVM version on the T3D*

#proc	$proc_X * proc_Y$	shmem time t [sec]	$S_p = t_{32}/t_{proc}$	PVM time t [sec]	$S_p = t_{32}/t_{proc}$	t_{PVM}/t_{shmem}
32	8 * 4	3329	1.00	3670	1.00	1.10
64	8 * 8	1671	1.99	1923	1.91	1.15
64	16 * 4	1675	1.99	1850	1.98	1.10
128	16 * 8	871	3.82	955	3.84	1.10
256	16 * 16	437	7.62	510	7.20	1.17
256	32 * 8	460	7.24	491	7.47	1.07

Due to the favourable relation of the computational complexity to the communication, the PVM overhead leads to a performance degradation of only $10\% - 15\%$. The dependence on the applied processor topology of $3\% - 5\%$ seems not to be significant. The PVM version seems to be a little bit more sensible to this aspect. The above PVM results are based on the use of the "DataInPlace" option reading from and wrighting directly in the memory of the respective other processor. Due to the mostly relatively large messages this option has been slightly faster.

6 Conclusion

The parallelization of the MLET algorithm by domain decomposition presented in this paper permits the treatment on MPP systems of large flow problems with high Reynolds numbers and very fine grids resolving all important flow structures. Due to memory and CPU restrictions, it was impossible to handle problems of this complexity on existing vector computers. The implementation of this algorithm can be adapted to various parallel environments with a fair amount of work and has proved to be well scalable.

The code is frequently used for production runs on a Cray T3D for the modeling of turbulent flow on grids with more than 11,000,000 cells and a very fine time resolution to realize periodic stimulations of approximately 50 Hz of the flow over the boundary conditions.

The final version of the paper will include numerical results for a large sized problem with 10^8 grid points of the backward facing step problem running on a Cray T3D with 256 processors. We will also report results of the reference test discussed in section 5 for code versions adapted for IBM workstation clusters and an IBM SP-2 under PVM as well as for MPI implementations for both the IBM SP-2 and the Cray T3D.

The present status of our work yields a basic framework for future developments which will include

- the treatment of other flow problems like crystal melt flows, i.e. Stefan problems
- the improvement of the iterative solver by possibly more adequate solvers, in particular CG-methods or overlapping Schwarz type methods (multigrid versions are under development at the Universität der Bundeswehr at Munich)
- the implementation of the alternative solution method mentioned in section 2 incorporating the solution of Poisson's equation.

Acknowledgment.

The authors are indebted to Cray Research, Inc., especially K. Ketelsen for the very serious help to learn using the explicit message passing concept and the Konrad-Zuse-Zentrum für Informationstechnik (ZIB), Berlin, for the intensive support of the numerical experiments.

References

[1] Werner, H.: Grobstruktursimulation der turbulenten Strömung über eine querliegende Rippe in einem Plattenkanal bei hoher Reynoldszahl, PhD thesis, TU München, 1991,

[2] Bärwolff, G. and Seifert, G.: Efficient 2D and 3D Navier-Stokes solver, Proceedings of the 5. ISCFD Sendai/Japan, 1993 (Ed. H. Daiguji).

[3] Bärwolff, G., Ketelsen, K. and Thiele, F.: Parallelization of a Finite-Volume Navier-Stokes solver on a T3D massively parallel system, Proceedings of the 6. ISCFD Lake Tahoe/USA, 1995 (Ed. M. Hafez).

[4] Akselvoll, K. and Moin, P.: Large eddy simulation of a backward facing step flow, Engineering Turbulence Modelling and Experiments 2, Elsevier Science Publishers, Amsterdam, 1993 (Ed. W. Rodi and F. Martinelli).

[5] Chorin, A.: Numerical Solution of the Navier–Stokes Equation, Math. Comp., 22 (1968).

[6] Bärwolff, G.: Numerische Berechnung von Transportprozessen fluider Medien, ZWG/AdW–Report 2/88, Berlin,1988.