A Parallel Solver for the 3D Incompressible Navier-Stokes Equations on the Austrian Grid

Ulrich Langer* and Huidong Yang†

Abstract. *We present a parallel, MPI (Message Passing Interface) based solver for the 3d incompressible Navier-Stokes equations. The technique of artificial compressibility, the explicit time integration method and the finite difference discretization in space are employed to obtain stationary solutions in which we are primarily interested in this paper. Using a mapping from the unit cube to the computational domain, we generate a structured mesh and the corresponding finite difference discretization. Domain decomposition in connection with a simple load-balancing technique is used to distribute the data over the several machines in the Austrian Grid. We analyses the performance of our Navier-Stokes code at different computing nodes of the Austrian Grid. We also give some outlook for our next implicit time integration scheme equipped with fast parallel multigrid solvers at each time step.*
Key words. *Incompressible Navier-Stokes equations, finite differences, domain decomposition, MPI, grid computing, Austrian grid*

1. Introduction

Usually, the grid infrastructure is composed of 3 layers: infrastructure layer which contains the grid resources, middleware layer and the application layer containing the simulation code. Large scale CFD (Computational Fluid Dynamics) applications, in particular, the so-called Direct Numerical Simulation (DNS), need huge computer resources. Grid computing facilities certainly open new perspectives for accessing huge computer resources. Fast parallel solvers for the incompressible Navier-Stokes equations play an important role in both scientific computing and industrial applications. The car industry, the aircraft industry and meteorology are only a few areas of industrial CFD applications. Much work aiming at developing efficient parallel solvers of these problems has been done by many people (see, e.g., [19] and the references therein).

Our explicit parallel solver for the 3d Navier-Stokes equations can be presented in the following steps:

1. generate the mesh from a given geometry at the local machine using transfinite interpolation (algebraic grid generation),
2. transfer data into nodes using gsiftp,
3. decompose the domain into sub-domains with overlapping parts, and each processor reads data into its local memory,
4. solve equations on each processor, and do communication in each time step,

*Institute of Computational Mathematics, Johannes Kepler University Linz, Austria, email: ulanger@numa.uni-linz.ac.at, Special Research Program SFB F013 “Numerical and Symbolic Scientific Computing”.

†Institute of Computational Mathematics, Johannes Kepler University Linz, Austria, email: huidong@numa.uni-linz.ac.at, supported by the Austrian Grid Project.

5. get final stationary solutions, write data to files, and visualize the solution.

This code allows us to solve problems with more than half a billion of unknowns. However, very small time steps are required for the explicit time integration scheme due to the stability condition. Therefore, we also discuss the use of implicit time integration schemes with coupled AMG (Algebraic MultiGrid) solvers for the arising Oseen problem [22, 23, 24].

2. 3D Incompressible Navier-Stokes Equations

For a bounded subdomain $\Omega \subset \mathbb{R}^3$ with a sufficiently smooth boundary, the Navier-Stokes equations governing the motion of an incompressible viscous fluid inside Ω read as follows:

$$u_t - \nu \Delta u + (u \cdot \nabla) u + \nabla p = f \qquad \text{in } \Omega \times (0,T), \tag{1}$$
$$\nabla \cdot u = 0 \qquad \text{in } \Omega \times (0,T), \tag{2}$$
$$\mathcal{B} u = g \qquad \text{on } \Gamma \times (0,T), \tag{3}$$
$$u = u_0 \qquad \text{in } \Omega \ \text{ at } t = 0, \tag{4}$$

where $\mathcal{B}$ denotes some boundary operators representing the boundary conditions imposed on Γ, u is the unknown velocity field, p is the unknown pressure field, f is a body force, T is the final time, and ν denotes the given viscosity that is inversely proportional to the Reynolds number Re. The equations (1) and (2) represent conservation of momentum and mass, respectively. They contain the nonlinear advection terms $(u \cdot \nabla) u$ and the dissipation terms $\nu \Delta u$. The turbulence arises for small viscosity term ν. To determine p uniquely, in case only Dirichlet boundary condition is imposed on $\Gamma = \Gamma_D$, we have to require the additional condition $\int_\Omega p dx = 0$.

3. MPI Library and Globus Toolkit

In the distributed memory system, the parallel computers are sets of processors with their own local memory. The processors can send data to each other through a network. In programs, the communication can be implemented by calls to functions from a special communication library MPI (message passing interface) [5], which is a library specification for message-passing, proposed as a standard by a broadly based committee of vendors, implementors, and users. The new grid-supporting version MPICH-G2 which is a grid-enabled implementation of the MPIv1.1 Standard, will be released soon. Using services from the Globus Toolkit [4], we can couple multiple machines with potentially different architectures and implement the communication for intermachine and intramachine messaging.

Globus toolkit [4], provides a way the job can be submitted from our local machine to the grid nodes and runs there. In order to run our MPI parallel program on the grid nodes, we need to set job type to MPI. This can be done by sort of RSL file, a type of the globus resource specification language. It provides a common interchange language to describe resources, and a interface for the users to manage the resources at the grid nodes from theirs local machines.

Before submitting jobs to run, we need to make a file in our local directory that characterizes our job type using the globus resource specification language RSL, see example in Figure 1. In that naive example, we specify resource allocation type, executable directory, job type, number of processors and hosts, and path for standard output. Relying on such RSL strings, the globus resource allocation manager (GRAM) performs its management and coordinates other globus software.

```
+
(&     (resourceManagerContact="altix1.jku.austriangrid.at/jobmanager-pbs")

(executable = /home/local/agrid/ag10035/Pro_05/globus_NS/3d_Nav/NS_Sol)
(directory = /home/local/agrid/ag10035/Pro_05/globus_NS/3d_Nav)
(maxtime = 5000)
(count =  64)
(label = "navier-stokes equations 0")
(hostcount = 4)
(jobtype=mpi)

(stdout = https://agrid-01.numa.uni-linz.ac.at:45000/dev/stdout)
(stderr = https://agrid-01.numa.uni-linz.ac.at:45000/dev/stderr)
)
```

Figure 1. An example using the Globus Resource Specification Language RSL.

4. Numerical Scheme

4.1. Explicit Scheme

If we solve equation (1) by means of the simplest time integral scheme, namely by the explicit Euler method [11], then we will obtain

$$\frac{u^{n+1}-u^n}{\tau} - \nu\Delta u^n + (u^n\cdot\nabla)u^n + \nabla p^n = f^n, \tag{5}$$

where τ is the time step and superscript n denotes the time level. Equation (5) can be trivially solved by spatial discretization. However, a fundamental problem arises using this method. The new velocity u^{n+1} does not, in general, fulfill the divergence free equation (2). Moreover, there is no direct computation of p^{n+1}.
A possible remedy is to introduce a pressure at p^{n+1} in equation (5), which leaves two unknowns, u^{n+1} and p^{n+1} to solve in equations

$$u^{n+1} + \tau\nabla p^{n+1} = \tau f^n + u^n - \tau(u^n\cdot\nabla)u^n + \tau\nu\Delta u^n, \tag{6}$$

$$\nabla\cdot u^{n+1} = 0. \tag{7}$$

Eliminating u^{n+1} by taking the divergence of (6) to obtain a Poisson equation for the pressure,

$$\Delta p^{n+1} = \frac{1}{\tau}\nabla\cdot(\tau f^n + u^n - \tau(u^n\cdot\nabla)u^n + \tau\nu\Delta u^n). \tag{8}$$

However, we do not know the boundary conditions for p^{n+1} naturally. It is too expensive to solve it this way either. Instead of solving them directly, we use a kind of artificial compressibility p_t [6], and add a penalty term $\varepsilon\Delta p$ [17], where $\varepsilon \ll 1$, by relaxing the incompressibility constraint in the appropriate way:

$$p_t + \varepsilon\Delta p + \nabla\cdot u = 0. \tag{9}$$

The new system can be approximated by explicit time integral method, for instance, by the first order explicit Euler scheme

$$\frac{u^{n+1}-u^n}{\tau} - \nu\Delta u^n + (u^n\cdot\nabla)u^n + \nabla p^n = f, \tag{10}$$

$$\frac{p^{n+1}-p^n}{\tau} + \varepsilon\Delta p^n + \nabla\cdot u^n = 0, \tag{11}$$

that we can solve equations like a IVP (initial value problem):

$$\frac{dU}{dt} = r(U), \tag{12}$$

with given initial conditions,

$$p_t = 0, \quad u_t = 0,$$

where U = $\begin{pmatrix} u \\ p \end{pmatrix}$ and $\varepsilon \ll \nu$. Equation (9) will not describe the flow correctly in time, but as the solution converges to steady state, the time dependent terms p_t vanishes and the continuity equation (2) is satisfied with additional assumption that we take sufficiently small correcting terms $\varepsilon\Delta p$ in equation (9). Error estimates for this method are given in [15].

4.2. Mesh Generation

In this simple case, a mesh generation class in two dimensions is given by the algebraic grid generation formula. A mapping is defined from the unit square to the physical domain : $(r,s) \rightarrow (x(r,s), y(r,s))$, with

$$\begin{aligned} x(r,s) &= (1-r)x(0,s) + rx(1,s) + (1-s)x(r,0) + sx(r,1) \\ &\quad - (1-r)(1-s)x(0,0) - r(1-s)x(1,0) - (1-r)sx(0,1) - rsx(1,1), \\ y(r,s) &= (1-r)y(0,s) + ry(1,s) + (1-s)y(r,0) + sy(r,1) \\ &\quad - (1-r)(1-s)y(0,0) - r(1-s)y(1,0) - (1-r)sy(0,1) - rsy(1,1), \end{aligned}$$

see more detail in [1], where they implement it by so called arc length parameterization for the given curve boundaries. Then the interior points are implemented by interpolation first between sides $r = 0$ and $r = 1$, second $s = 0$ and $s = 1$, and last subtraction by four corner terms. We parameterize x and y as function of $p \in [0,1]$. The arc length is calculated as $L = \int_a^b \sqrt{(x'(p)^2 + y'(p)^2)}dp$. In order to determine $(x(p), y(p))$, we have to know the parameter p by solving the arc length equation: $s = \frac{1}{L}\int_a^t \sqrt{(x'(p)^2 + y'(p)^2)}dp$, where $\frac{1}{s}$ should be equal to the number of equivalent distance points in the boundary. An efficient algorithm to solve this equation is Newton- Raphson method: $p_{k+1} = p_k - \frac{f(p_k)}{f'(p_k)}$, where $f(p_k) = \int_a^{p_k} \sqrt{(x'(p)^2 + y'(p)^2)}dp - sL$. Then by creating such two dimensional objects, we develop three dimensional mesh grids on the physical domain by extending mesh object in two dimensions along the third direction, see Figure 2. In fact, we do not solve our PDE on such domain directly, instead, on the unit cube as in Figure 3, and then transform the solutions back.

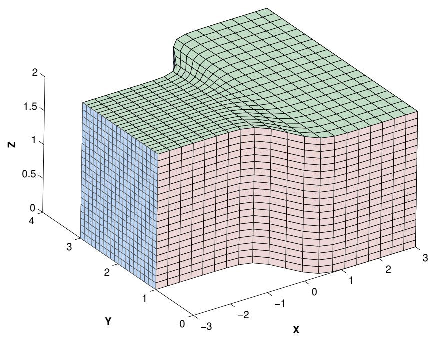

Figure 2. Mesh on physical domain.

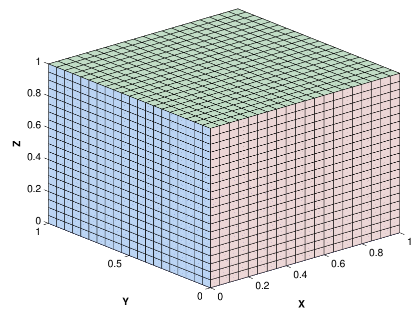

Figure 3. Mesh grids on unit cubic.

4.3. Mapping from Physical Domain to Unit Cube

An alternative method to solve partial differential equations is to transform the physical domain into the unit cube by grid mapping, and approximate equations there. Assume given the grid mapping $x(q,r,s)$, $y(q,rs)$ and $z(q,r,s)$ from the physical domain to the unit cube $(q,r,s) \in [0,1]^3$, by chain rule, the explicit form of the metric coefficients matrix can be calculated by hand

$$\begin{pmatrix} q_x & r_x & s_x \\ q_y & r_y & s_y \\ q_z & r_z & s_z \end{pmatrix} = \frac{1}{J} \begin{pmatrix} y_r z_s - z_r y_s & y_s z_q - z_s y_q & y_q z_r - z_q y_r \\ z_r x_s - x_r z_s & z_s x_q - x_s z_q & z_q x_r - x_q z_r \\ x_r y_s - y_r x_s & x_s y_q - y_s x_q & x_q y_r - y_q x_r \end{pmatrix}, \tag{13}$$

where

$$J = \begin{vmatrix} x_q & y_q & z_q \\ x_r & y_r & z_r \\ x_s & y_s & z_s \end{vmatrix}.$$

Thus, a partial differential differential equation,

$$u_t + F(u)_x + G(u)_y + H(u)_z = 0, \tag{14}$$

equipped with the metric coefficient matrix, can be approximated on a unit cube instead by

$$\begin{aligned} u_t = -\frac{1}{J}&((J(q_x F(u) + q_y G(u) + q_z H(u)))_q \\ &+ (J(r_x F(u) + r_y G(u) + r_z H(u)))_r \\ &+ (J(s_x F(u) + s_y G(u) + s_z H(u)))_s). \end{aligned} \tag{15}$$

A single first derivative u_x can be expressed in the unit cube, i.e., by

$$u_x = \frac{1}{J}((y_r z_s - z_r y_s)u_q + (y_s z_q - z_s y_q)u_r + (y_q z_r - z_q y_r)u_s). \tag{16}$$

The second derivative u_{xx} is attained by applying the mapping once more, yielding

$$u_{xx} = \frac{1}{J}((y_r z_s - z_r y_s)(u_x)_q + (y_s z_q - z_s y_q)(u_x)_r + (y_q z_r - z_q y_r)(u_x)_s). \tag{17}$$

4.4. Finite Difference Stencil

Non-staggered grid stencil is employed, which means velocity u and pressure p are located at the same grid point. The previous steps have prepared for the numerical approximation on the unit cube. Index (i,j,k) corresponds to direction (q,r,s). The second order of accuracy in space is obtained by using values at middle points $(i \pm \frac{1}{2}, j, k)$, $(i, j \pm \frac{1}{2}, k)$ and $(i, j, k \pm \frac{1}{2})$. See stencil Figure 4 below, i.e., $(u_q)_{i,j,k} = \frac{u_{i+1,j,k} - u_{i-1,j,k}}{2\Delta q}$. More technical details can be found in [3].
In order to get the second derivative for velocity u with respective to x at point (i,j,k), the first derivatives $(u_q)_{i\pm\frac{1}{2},j,k}$, $(u_r)_{i,j\pm\frac{1}{2},k}$, $(u_s)_{i,j,k\pm\frac{1}{2}}$ at its neighboring middle points in three directions (q,r,s) respectively are necessary to be known. While for such values at these middle points, for example,$(u_q)_{i-\frac{1}{2},j,k}$, we would then use values at the standard points $(i-1,j,k)$, $(i-1,j\pm 1,k)$ and $(i-1,j,k\pm 1)$, etc. We may call it ten-point stencil as in Figure 5 below. The central finite difference in space is employed, i.e., $(u_q)_{i-\frac{1}{2},j,k} = \frac{u_{i,j,k} - u_{i-1,j,k}}{\Delta q}$, $(u_r)_{i-\frac{1}{2},j,k} = \frac{(u_r)_{i-1,j,k} + (u_r)_{i,j,k}}{2}$, and

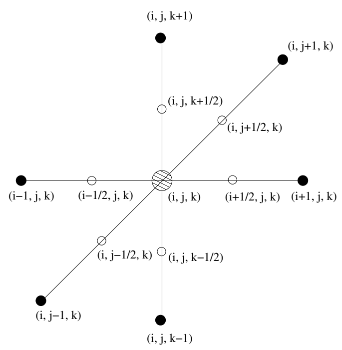

Figure 4. Standard point (i, j, k).

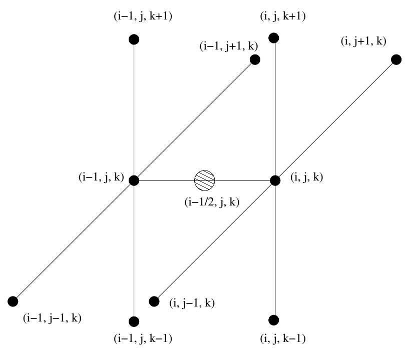

Figure 5. Middle point $(i-1/2, j, k)$.

$(u_s)_{i-\frac{1}{2},j,k} = \frac{(u_s)_{i-1,j,k}+(u_s)_{i,j,k}}{2}$.
The time step involves one or more evaluations of the difference scheme. We write the semi-discrete problem as $\frac{dU_{i,j,k}}{dt} = r(\ldots,U_{i,j,k},\ldots)$, where $r(U)$, the residual contains all spatial derivatives of the PDE. For steady state problems we want to solve $r(U) = 0$, numerically, $||r(U)|| < Tolerance$. The residual is computed by successive function call. A forward Euler method is applied here: $U^{n+1}_{i,j,k} = U^n_{i,j,k} + \tau r^n_{i,j,k}$.

4.5. Boundary and Initial Conditions

We set boundary and initial conditions for the test domain as in Figure 6. Face 0 is inflow boundary, $P = 1, u_x = 0, v = 0, w = 0$; face 1, the outflow boundary, $P = 0, u_x = 0, v = 0, w = 0$. All the other faces are walls, Dirichlet boundary conditions are applied for velocity fields, $u = v = w = 0$, and Neumann boundary condition for pressure, $\frac{\partial P}{\partial n} = 0$.
The initial value for velocity is $u_0 = v_0 = w_0 = 0$, and the pressure $p_0 = 1-(x+3)/6, x \in [-3,3]$, see figure 7. The pressure only linearly depends on the variable x, which fulfills the boundary conditions, 1 at the inflow, 0 at the outflow, and $\frac{\partial P}{\partial n} = 0$ on the other boundary faces. Because such particular pressure distribution only dependents on the $x-$direction, the velocity filed along the $z-$direction would be homogeneous.

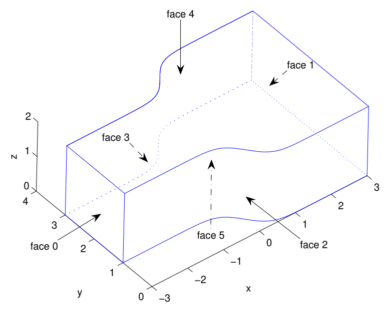

Figure 6. Boundaries on the domain.

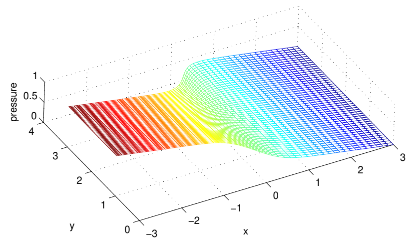

Figure 7. Initial pressure.

4.6. Implicit Scheme

If we linearize the Navier-Stokes equations (1)-(2), we get the so-called *Oseen equations* (Carl Wihelm Oseen, 1879-1944)

$$u_t - \nu\Delta u + (w\cdot\nabla)u + \nabla p = f, \tag{18}$$
$$\nabla\cdot u = 0, \tag{19}$$

where w is the previous velocity approximation u^n of the velocity u. Applying the implicit Euler scheme to the linearized system (18)-(19), we arrive at

$$\frac{u^{n+1}-u^n}{\tau} - \nu\Delta u^{n+1} + (u^n\cdot\nabla)u^{n+1} + \nabla p^{n+1} = f, \tag{20}$$
$$\nabla\cdot u^{n+1} = 0. \tag{21}$$

At each time step, the following linear system must be solved:

$$\begin{pmatrix} I-\tau\nu\Delta+\tau(u^n\cdot\nabla) & \tau\nabla \\ \tau\nabla\cdot & 0 \end{pmatrix}\begin{pmatrix} u^{n+1} \\ p^{n+1} \end{pmatrix} = \begin{pmatrix} \tau f + u^n \\ 0 \end{pmatrix}. \tag{22}$$

Using the spatial discretization FDM (Finite Difference Method) or FEM (Finite Element Method), we can get the resulting linear saddle point system $\mathbf{K}\mathbf{y} = \mathbf{b}$ as the following general form

$$\underbrace{\begin{pmatrix} A(u^n) & B^T \\ B & 0 \end{pmatrix}}_{\mathbf{K}}\underbrace{\begin{pmatrix} u^{n+1} \\ p^{n+1} \end{pmatrix}}_{\mathbf{y}} = \underbrace{\begin{pmatrix} \tau f + u^n \\ 0 \end{pmatrix}}_{\mathbf{b}}. \tag{23}$$

Some non-multigrid iterative solvers are applicable to solve this saddle point problem such as the *the general minimal residual method* (GMRES) introduced in [14]. Another popular method is *Semi-Implicit Method for Pressure-Linked Equations* (SIMPLE) developed by Patankar and Spalding [13]. Inexact Uzawa Method can also be applied to solve this system. Details to it can be found e.g. in [12, 27]. For each time level, we employ a parallel multigrid method (Algorithm 1), see for more details in [7]. We develop our implicit solver (Algorithm 2) which includes the multigrid solver based

Algorithm 1 PMGM(A_q, K_q, y_q, b_q, q)

if $q == 1 :=$ CoarsestLevel **then**
  Coarsest grid solver : $K_q \cdot y_q = b_q$
else
  Presmoothing : $y_q = S_{pre}(y_q)$
  Defect calculation (new r.h.s) : $d_q = b_q - K_q \cdot y_q$
  Restriction of defect : $d_{q-1} = I_q^{q-1}\cdot d_q$
  Coarse grid initial guess : $m_{q-1}^0 = 0$
  Solve defect system : $m_{q-1} = \mathbf{PMGM}(A_{q-1}, K_{q-1}, m_{q-1}^0, d_{q-1}, q-1)$
  Interpolation of correction : $m_q = I_{q-1}^q \cdot m_{q-1}$
  Addition of correction : $y_q = y_q + m_q$
  Postsmoothing : $y_q = S_{post}(y_q)$
end if

on Warbro's previous work [23].

Algorithm 2 IMP_NS_Sol(y^0, q)

Give the initial value at time 0 : $y^n = y^0$
while $\| y^n \| > \alpha \| y^0 \|$ **do**
  Define the initial guess for level q : $y_q = y_{q,0}$
  while $\| b_q - K_q y_q \| > \varepsilon \| b_q - K_q y_{q,0} \|$ **do**
    $y_q =$ **PMGM(**A_q, K_q, y_q, b_q, q**)**
  end while
  Update the new solution at this time step : $y^0 = y^n, y^n = y_q$
end while

4.7. Explicit VS Implicit Scheme

Comparing these two methods with each other, we can find some advantages and disadvantages for both of them. For the explicit methods, no storage for large matrix is needed and they are easier to implement. However, we can only get correct solution for stationary state, and also, for stability reasons, we have strict limitation for time step size. For the implicit methods, we can get reasonable solutions at each time step and it has no strict restriction to time step size. However, it has memory requirement for large scale matrix storage and much work to do for matrix decomposition.

5. Parallel Implementation

Here, a simple domain decomposition method is given by load balancing. In this way, each processor will hold nearly the same number of grid points, which means they have the same amount of work to do, including both communication and computation (see more details from [2]). Here we develop the algorithm for the three dimensional case.

5.1. A Simple Domain Decomposition

We decompose the domain Ω into overlapping sub-domains Ω_i such that $\Omega = \bigcup_{i=1}^{np} \Omega_i$, where np denotes the number of processors. The decomposition $\Pi(GS, PS)$ is dependent on the number of global grid size $GS = (mg, ng, bg)$ in three directions, and the processor number $PS = (p_1, p_2, p_3)$ as well[1]. The number of processors in each direction can be calculated by minimizing the communication cost function:

$$C(p_1, p_2, p_3) = (p_1 - 1)ngbg + (p_2 - 1)mgbg + (p_3 - 1)mgng \tag{24}$$

subject to $p_1 p_2 p_3 = np$. Then we would like to have $mg/p_1 = ng/p_2 = bg/p_3$. This is of course not always possible to fulfill with integers, and we then try numerically instead to determine p_1, p_2, and p_3 such that modified cost expression

$$\hat{C}(p_1, p_2, p_3) = |mg/p_1 - ng/p_2|^2 + |ng/p_2 - bg/p_3|^2 \tag{25}$$

is minimized under the restriction $p_1 p_2 p_3 = np$ again. Taking the number of processor np as a number with many factors, would greatly improve decomposition efficience. Using fixed number of grid size 321^3, we did the experiments with varying number of processors. By minimizing the communication cost function numerically, we have less communication work to do for a specified case. Work is

[1] We assume here that a homogeneous parallel computing system is being used in which each processor has the same performance characteristic

measured in number of grid points, see Table 1. In this special case, $mg = ng = bg = 321$, the cost function is then reduced to

$$C(p_1, p_2, p_3) = mg^2(p_1 + p_2 + p_3 - 3) = mg^2(\sum_{i=1}^{3} p_i - 3), \tag{26}$$

which linearly dependents on the function of $\sum_{i=1}^{3} p_i$. We can check this fact easily from Table 1.

np	$p_1 \times p_2 \times p_3$	$C(p_1, p_2, p_3)$	$\frac{C(p_1,p_2,p_3)}{mgngbg}$	$\sum_{i=1}^{3} p_i$
1	$1 \times 1 \times 1$	0	0	3
2	$1 \times 1 \times 2$	103041	0.0031	4
4	$1 \times 2 \times 2$	206082	0.0062	5
8	$2 \times 2 \times 2$	309123	0.0093	6
12	$2 \times 2 \times 3$	412164	0.0125	7
16	$2 \times 2 \times 4$	515205	0.0156	8
32	$2 \times 4 \times 4$	721287	0.0218	10
48	$3 \times 4 \times 4$	824328	0.0249	11
64	$4 \times 4 \times 4$	927369	0.0280	12

Table 1. Cpu distribution computed by minimizing communication costs. Mesh size is $321^3 = 33076161$.

Thus, for this simple case, the grid points are distributed over the processors as evenly as possible in one dimension and then it can be applied to three dimension case easily. Assume the processors are enumerated from 1 to p and s_k points in processor k. Let the local index i_k in processor k vary from 1 to s_k. The width of overlapping region in our case is $a = 2$. The global index i, varies between l and n. We choose

$$s = [\frac{N + (p-1)a}{p}], \tag{27}$$

where $[x]$ denotes the integer part of x, $N = n - l + 1$. Define the remainder,

$$r = (N + (p-1)a) \bmod p,$$

and we distributed these remaining points evenly into the first processors:

$$s_k = \begin{cases} s+1 & \text{if } k \leq r \\ s & \text{if } k > r. \end{cases} \tag{28}$$

The index transformation t_k, an integer such that $i_k + t_k = i$, is

$$t_k = l - 1 + (k-1)(s-a) + \min(k-1, r). \tag{29}$$

The processor ID (c_1, c_2, c_3) in the processor cube is required for the communication of the overlapping part. The processor coordinates can be determined by inverting the mapping:

$$pr = c_1 - 1 + p_1(c_2 - 1) + p_1 p_2(c_3 - 1). \tag{30}$$

Here $1 \leq c_i \leq p_i$, $0 \leq pr \leq np - 1$, and pr is the identity in the one dimensional processor enumeration given by the function MPI_Comm_rank from the MPI communication library.

5.2. Communication Model

The blocking point-to-point MPI routines MPI_Recv and MPI_Send are employed in program. We also developed so called *Red_Black* algorithm for 3D case as in figure 8. Suppose each processor has a color (red or black) in each direction such that no neighbors have the same color. Neighbors have their own color coordinate. The color can be implemented by the *R_B* function:

$$R\_B = \begin{cases} red & \text{if } c_i \bmod 2 == 0, \\ black & \text{if } c_i \bmod 2 == 1, \end{cases} \qquad i = 1,2,3. \tag{31}$$

Processors need to communicate the information for every overlapping part. Thus, in our example, each processor has to call MPI communication subroutines MPI_Send and MPI_Recv six times in order to send/receive necessary data information at the interface to/from its neighbors suppose each cpu has six neighbors.

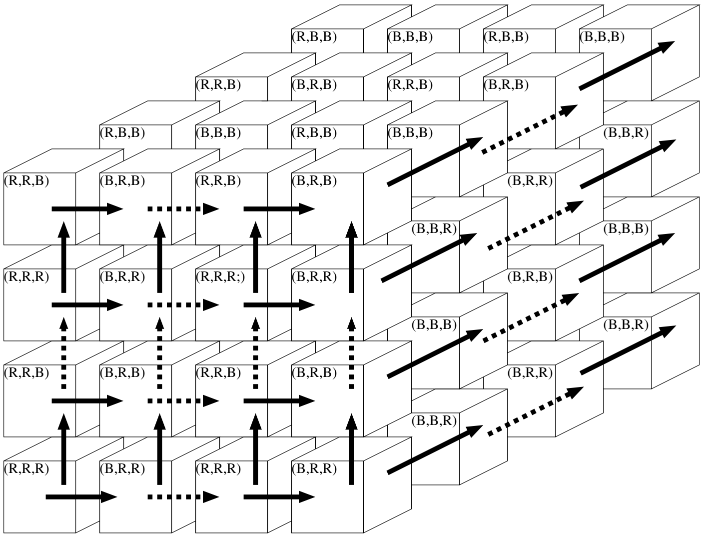

Figure 8. An illustration of *Red_Black* algorithm in three sweeps, West→ East, Down→Top, and Forth→ Back.

Applying the *Red_Black* algorithm, the communication time can be reduced since some of them can send/receive data to/from their neighbors simultaneously. However, because blocking communication functions are applied here, which means that the program execution will be suspended until the message buffer is safe to use. The MPI standards specify that a blocking SEND or RECV does not return until the send buffer is safe to reuse (for MPI_SEND), or the receive buffer is ready to use (for MPI_RECV). Besides the three sweeps shown in Figure 8, every processor still exits the other three sweeps, $East \to West$, $Top \to Down$, and $Back \to Forth$. In order to avoid the deadlock, neighbors cannot call MPI_SEND/MPI_RECV at the same turn. Every send in a processor should respond to a receive in its neighbor and on the other hand it cannot receive data safely only until its neighbor sends data. We give an communication example in $West \leftrightarrow East$ sweep. See algorithm 3.

5.3. Performance Model

The time to send a message with n bytes between two processors is often modeled as $T_c(n) = \alpha + n\beta$, where α is the start-up time to initialize the communication and β is the time to send one byte of

Algorithm 3 Red_Black Algorithm in West $\leftrightarrow$ East Sweep

West$\rightarrow$East :
(R,*,*) send to east (B,*,*)   (B,*,*) receive from west (R,*,*)
(B,*,*) send to east (R,*,*)   (R,*,*) receive from west (B,*,*)
East$\rightarrow$West :
(R,*,*) send to west (B,*,*)   (B,*,*) receive from east (R,*,*)
(B,*,*) send to west (R,*,*)   (R,*,*) receive from east (B,*,*)

data. In practice, $\alpha \gg \beta$, so that the starting time is the dominating term. It is a very simplified model of reality. However, we would like to give another performance model for the total execution time, communication time plus computing, $T(N,np)$. Here, N is the problem size and np is the number of processors as before. The behavior for large N is often used to test the performance. Now the speed-up is defined as the ration of time to run the problem on one processor, to that of on np processors:

$$S(N,np) = \frac{T(N,1)}{T(N,np)}. \tag{32}$$

Perfect speed-up is obtained when $S(N,np) = np$.

5.4. Numerical Examples for Explicit Scheme

At this moment, we show some results from explicit scheme. Using middle-ware globus, we submit and run our jobs on different Austrian grid nodes and try to do the simulation there.

5.4.1. Tests on Nodes Altix1.jku.austriangrid.at

There are Altix1$\sim$4 at Linz University. It is an SGI Altix 3500 system, consisting of 4 nodes. Each of them contains 16 microprocessors (Intel Itanium 2, 900MHz/1.5/MB L3 Cache each), 64Gbyte RAM and 64-bit PCI-X buses. First we give the test results in Table 2 of two small size problems which are tested only on one node consisting of 16 cpus at JKU. Approximately, for really large prob-

#unknowns	$21^3 * 4 = 37,044$	$51^3 * 4 = 530,604$
Residual	$4.48e-14$	$1.43268e-14$
#steps	$100,000$	$100,000$
Time	$80s$	$610s$
#CPU	16	16
Step size	0.0012	0.0005
Test node	altix1 at jku	altix1 at jku

Table 2. Two test results.

lem, for instance, $321^3 * 4 = 132,304,644$, $time \approx 285,720s \approx 80h$. We really need more powerful resources (more grid nodes) in order to reduce the cost. The next example is to try to run our problem on different nodes Altix1$\sim$4 at JKU. We can use number of cpus from 1 up to 64. We run our program ten time steps and compared the const with different number of grid points from 21^3 up to 501^3. Obviously, as we see from Table 3, for large problem, for instance, with number of unknowns $501^3 * 4 = 503,006,004$, we can benefit a lot from using grid resources. However, because of extra communication time needed among nodes, the speedup does not purely linearly depend on the number of cpus, see Figure 9.

np	time measured with different grid size					
	21^3	41^3	81^3	161^3	321^3	501^3
2	0.0327	0.2702	3.0309	29.6238	362.9065	1757.745
4	0.0191	0.1169	1.5258	13.4702	142.2860	662.1513
8	0.0129	0.0672	0.7601	6.4801	65.4366	288.4072
12	0.0116	0.0522	0.4933	4.2527	42.9452	186.5697
16	0.0110	0.0434	0.3576	3.2602	32.4913	131.8930
32	9.2776	0.0445	0.17296	1.8554	25.7761	109.9527
48	10.1021	0.0605	0.1765	1.3465	19.3027	84.9540
64	7.6453	0.0804	0.1728	1.3435	18.0354	75.792

Table 3. Time consumption for different size of problems with varying number of processors.

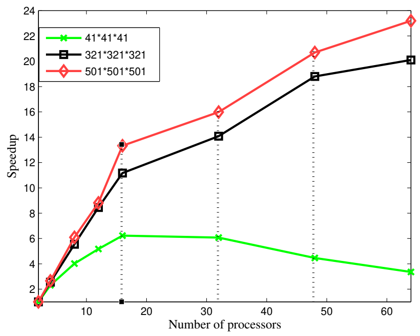

Figure 9. Speedup at local nodes.

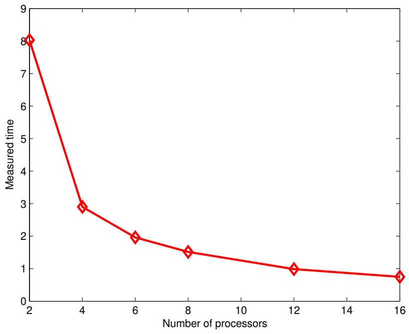

Figure 10. Measured time at remote node.

5.4.2. Test on Remote Nodes Altix1.uibk.ac.at

Here is another example which is tested on a remote node Altix1.uibk.ac.at. The number of unknowns is $101^3 * 4 = 4,121,204$, and run it with 11 time steps. Then we get the following time measured in second as shown in Figure 10.

5.4.3. Stationary Fields of Velocity and Pressure

Velocity fields are shown in Figure 11 and Figure 12. The pressure distribution at the steady state would be comparable to the initial value except numerical non-accuracy at the corners coming from numerical boundary conditions for pressure. The residual terms, which tells us if the solutions have forwarded to the final stationary state is recorded in Figure 14. We pick up the maximal residual value $r = U(\ldots, U_{i,j,k}, \ldots)$ for the equations from all grid points.

6. Summary and Future Work

This type of numerical methods for the modified Navier-Stokes equations with artificial compressibility can work by using such a naive numerical approximation method on parallel machines at Austrian grid nodes. However, such explicit time stepping method we developed here is not suitable very well for grid computing via connecting different nodes distributed in several universities. Lots of communications are needed through networking at each time step. It would be interesting to improve

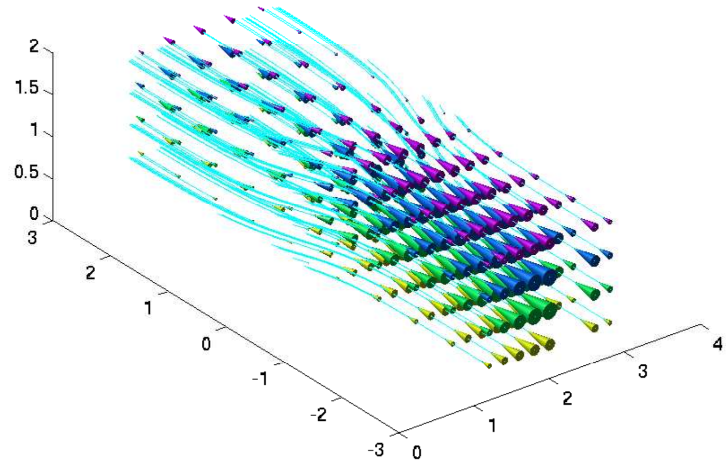

Figure 11. Visualization of the velocity fields using cone plotting and stream lines.

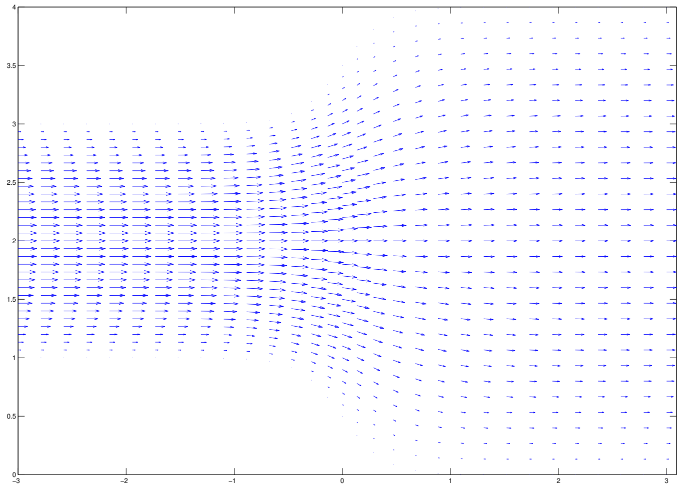

Figure 12. Velocity field in a $X - Y$ cutting plane.

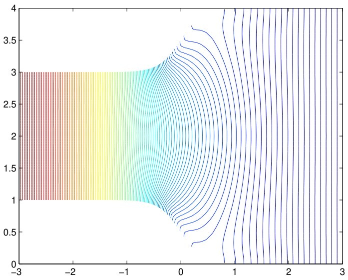

Figure 13. Pressure distribution.

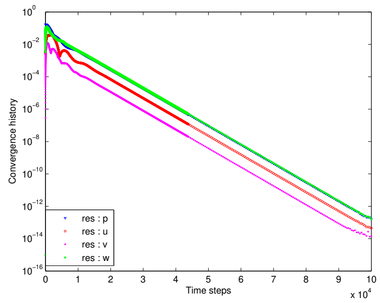

Figure 14. Convergence history.

this solver by using more sophisticated numerical technologies in order to get the convergence more quickly, for example, via implicit time stepping methods with fast solvers at each time step [22]. Unfortunately, at this moment, we do not have grid-supporting MPI library installed on our machine which can run a single job at the same time on machines from different geographical nodes. We are not quite sure if this would improve the performance quite well since extra time would be spent on the communication among nodes with long distance.

Acknowledgements

The authors would like to thank Dr. Joachim Schöberl and Prof. Walter Zulehner for their useful discussion about Navier-Stokes equations, and Peter Praxmarer, Markus Baumgartner, and Rene Kober for their technical support about Austrian grid. Last but not least, the financial support of the Austrian Grid Project and the FWF (Austrian Science Fund) Special Research Program SFB F013 are gratefully acknowledged.

References

[1] Internet. Algebraic grid generation. http://www.nada.kth.se/kurser/kth/2D1263/l3.pdf, 17-19.

[2] Internet. Distributions of an array on a parallel computer. http://www.nada.kth.se/kurser/kth/2D1263/lecture_notes5.pdf/, 93-96.

[3] Internet. Finite difference approximation. http://www.nada.kth.se/kurser/kth/2D1263/l6.pdf, 44-55.

[4] Internet. Globus Toolkit package. http://www.globus.org/toolkit/.

[5] Internet. MPI package. http://www-unix.mcs.anl.gov/mpi/.

[6] A.J.Chorin, Numerical solution of the Navier-Stokes equations, *Math. Comp.,* **22** (1968), 745-762.

[7] C.C.Douglas, G.Hasse, U.Langer, A Tutorial on Elliptic PDE Solvers and Their Parallelization, *SIAM*, 2003, Philadelphia.

[8] W.Hackbusch, Multi-Grid Methods and Applications, Springer-Verlag, Berlin-Heidelberg, 1985.

[9] D.S.Henningson, M.Berggren, Computational Fluid Dynamics, lecture notes in Computational Fluid Dynamics, March, 2005, Royal Institute of Technology, Sweden.

[10] W.Kress, High Order Finite Difference Methods in Space and Time, 2003, Acta Universitatis Upsaliensis, Sweden.

[11] H.P.Langtangen, K.-A.Mardal, R.Winther, Numerical Methods for Incompressible Viscous Flow, *Advances in Water Resources,* **Vol 25(8-12)** (2002), 1125-1146.

[12] U.Langer, W.Queck, On the convergence factor of Uzawa's algorithm, *J.Comput.Appl.Math.,* **15** (1986), 191-202.

[13] S.Patankar, D.Spalding, A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows, *Int.J.Heat Mass Transfer,* **15** (1972), 1787-1806.

[14] Y.Sadd, M.H.Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear system. *SIAM J.Sci.Stat.Comput.,* **7** (1986), 856-869.

[15] J.Shen, Pseudo-compressibility methods for the unsteady incompressible Navier-stokes equations, 11th AIAA Computational Fluid Dynamic Conference, 6-9 July, 1993, AIAA paper 93-3361, Orlando, FL, USA.

[16] John C.Strikwerda, Finite Difference Schemes and Partial Differential Equations, Wadsworth, Brooks, 1989.

[17] R.Teaman, On error estimates of the penality method for the unsteady Navier-Stokes equations, *SIAM J. Numer.Anal.,* **32-2**(1995).

[18] S.Tsuge, The Kolmogorov turbulence theory in the light of six-dimensional Navier-Stokes' equation, *eprint arXiv:nlin/0303013,* **Vol 2** (2003).

[19] S.Turek, Efficient Solvers for Incompressible Flow Problems, Lecture Notes in Computational Science and Engineering, Springer-Verlag, Berlin-Heidelberg, 1999.

[20] S.Turek, Multigrid techniques for a divergence-free finite element discretization, *East-West J. Numer. Math.,* **Vol.2, No.3** (1994), 229-255.

[21] S.Turek, Multigrid techniques for a simple discretely divergence-free finite element space, in Hemker / Wesseling, Multigrid Methods IV, 1994, 321-332.

[22] M.Wabro, AMGe-Coarsening Strategies and Application to the Oseen-Equations, *SIAM J.Sci.Comput.,* accepted for publication.

[23] M.Warbro, Algebraic Multigrid Methods for the Numerical Solution of the Incompressible Navier-Stokes Equations, PhD thesis, Johannes Kepler University Linz, 2003.

[24] M.Wabro, Coupled Algebraic Multigrid Methods for the Oseen Problem, *Computing and Visualization in Science,* **7, No 3-4** (2004), 141-151.

[25] J.Xu, Iterative methods by space decomposition and subspace correction, *SIAM Review*, **34** (1992), 581-613.

[26] W.Zulehner, A class of smoothers for saddle point problems. *Computing*, **65(3)** (2000), 227-246.

[27] W.Zulehner, Analysis of iterative methods for saddle point problems: a unified approach, *Mathematics of computation,* **71** (2002), 479-505.