A parallel multiblock method for 3D incompressible flows in complex geometries
D. Drikakis, R. Zahner

Lehrstuhl für Strömungsmechanik, Universität Erlangen-Nürnberg, Cauerstr. 4, D-91058 Erlangen, Germany

Abstract

The development of a parallel three-dimensional code for incompressible flows is presented. The solver is based on a third order upwind method. The artificial compressibility formulation is used for coupling the continuity with the momentum equations. The time integration is obtained by an explicit Runge-Kutta scheme. Parallelization on block-structured grids is obtained by using shared-memory, as well as, message-passing model.

1 Introduction

The last few years the requirements for solving increasingly more complex problems have provided a driving force for the extensive development of parallel computing machines. The research on parallel computational methods and codes is continuously growing with the development of new solvers and the investigation of parallelization strategies [1,2,3]. The performance of parallel computations depends on hardware and software factors [2,3]. Many industrial problems related to fluid mechanics e.g simulation of the flow over a car or into an engine, are also expected to be studied with higher accuracy by using high performance parallel computers. In addition for the investigation of flows in complex geometries multiblock structured or unstructured grids have to be used. For the parallelization of these techniques three-dimensional grid partitioning algorithms have to be developed providing flexibility and generality in complex geometries. In order to achieve the
best performance for these computations, different parallelization strategies have initially to be investigated.

In this paper the development of a parallel three-dimensional incompressible code based on multiblock structured grid is presented. For the parallelization of the present code the shared memory, as well as, the message passing model were employed. Comparison between these two parallelization strategies is shown on different grids and number of processors.

## 2 Numerical method

### 2.1 Artificial compressibility formulation

The governing equations are the 3D incompressible Navier-Stokes equations. The system of equations in matrix form and curvilinear coordinates can be written as:

\[
(JU)_t + (E_I)_\xi + (F_I)_\eta + (G_I)_\zeta = (E_V)_\xi + (F_V)_\eta + (G_V)_\zeta
\]

The unknown solution vector is:

\[
U = (p, \beta, u, v, w)^T
\]

where \( p \) is the pressure and \( u, v, w \) are the velocity components. The parameter \( \beta \) is the artificial compressibility parameter, and it has to be chosen to ensure the fastest convergence to steady state. The terms \( E_I, F_I, G_I \), as well as, \( E_V, F_V, G_V \), are the inviscid and viscous fluxes, respectively. \( J \) is the Jacobian of the transformation from Cartesian \( (x, y, z) \) to generalized \( (\xi, \eta, \zeta) \) coordinates.

### 2.2 A high resolution scheme for 3D flows

The inviscid terms of the Navier-Stokes equations are discretized by a characteristic based method [4]. The method has similar concept with the Riemann solvers developed in the past for the solution of the compressible equations. The characteristic based method constructs Riemann solutions on each flow direction and the primitive variables \( (p, u, v, w) \) are defined as functions of their values \( (p_l, u_l, v_l, w_l) \) on the characteristics \( l = 0, 1, 2 \):

\[
u = R\ddot{x} + u_0(\ddot{y}^2 + \ddot{z}^2) - v_0\ddot{x}\ddot{y} - w_0\ddot{x}\ddot{z}
\]

\[
v = R\ddot{y} + v_0(\ddot{x}^2 + \ddot{z}^2) - w_0\ddot{z}\ddot{y} - u_0\ddot{x}\ddot{y}
\]

\[
w = R\ddot{z} + w_0(\ddot{y}^2 + \ddot{x}^2) - v_0\ddot{z}\ddot{y} - u_0\ddot{x}\ddot{z}
\]
where 
\[ R = \frac{1}{2s} \left( p_1 - p_2 + \dot{x}(\lambda_1 u_1 - \lambda_2 u_2) + \dot{y}(\lambda_1 v_1 - \lambda_2 v_2) + \dot{z}(\lambda_1 w_1 - \lambda_2 w_2) \right) \]
and 
\[ s = \sqrt{\lambda_0^2 + \beta}, \quad \text{and} \quad \xi_k = \frac{\xi_k}{\sqrt{\xi_x^2 + \xi_y^2 + \xi_z^2}}, \quad k = x, y, z \]

Consequently the pressure is computed by one of the following equations:

\[ p = p_1 - \lambda_1 \left( \dot{x}(u - u_1) + \dot{y}(v - v_1) + \dot{z}(w - w_1) \right) \quad \text{(5)} \]
or

\[ p = p_2 - \lambda_2 \left( \dot{x}(u - u_2) + \dot{y}(v - v_2) + \dot{z}(w - w_2) \right) \quad \text{(6)} \]

In the above formulae \( \lambda_0, \lambda_1, \lambda_2 \) are the eigenvalues. The inviscid terms are discretized on the center of the volume by using the above formulae for the primitive variables on the cell faces. The viscous terms are defined by central differences, and the variables \( u, v, w, p \) are defined by third order upwind differences [4]. The time integration of the Navier-Stokes equations is obtained by an explicit Runge-Kutta scheme. In addition the local time stepping technique is used for steady state computations.

3 Parallelization on multiblock structured grids

3.1 Multiblock environment and data structure

The multiblock technique is used for simulating flows in complex geometries. In these cases the grid generation on a single-block either it is not possible or it is very complicated. The geometry is subdivided into a number of blocks and the grid is generated on each of these blocks. All the matrices of the computational code follow an one-dimensional storage. The boundary conditions are stored into a layer of phantom cells, containing two cells on each direction around the computational domain. This layer is separately defined on each block. The boundary conditions are defined on each block face and their type (e.g. inflow, outflow, no-slip, symmetry, farfield etc.) is declared by an integer matrix. Each face can simultaneously contain different type of boundary conditions. Using the above macroblock definition of boundary conditions the definition of several small blocks that contain homogeneous conditions on the faces is avoided. The latest should be followed for achieving better performance on parallel computers, due to the load balancing effect. For the definition of the overlapping boundary volumes between neighbour blocks, additional index matrices are defined. The values of the overlapping volumes are exchanged between the blocks on each Runge-Kutta iteration.
3.2 Grid-partitioning

A grid-partitioning algorithm for the decomposition of the whole domain into several subdomains was developed. Each block can be decomposed in one (1D-partitioning), two (2D-partitioning) or three directions (3D-partitioning), respectively. An example of a decomposed multiblock domain is shown in Fig. (1), where the shading represents fictitious cells used for the storage of the subdomains’ boundary values. In the present study the partitioning strategy is the following: One or more geometrical blocks can be assigned per processor, and each geometrical block can additionally be subdivided in several subdomains. Furthermore, each processor can contain more than one subdomains.

3.3 Parallelization using message-passing model

In the message passing model the computational grid is subdivided into non-overlapping domains, and each domain is assigned to one processor. Each processor stores its own data. The boundary values are exchanged between the processors during the numerical solution. For the parallelization of the code subroutines for distributing the grid data on each processor, for exchanging data between the processors (i.e. local and global communication), and for gathering data from all processors on the host processor were constructed. The main part of these routines is hardware independent and only a part concerning the “read” and “writing” of the data between the processors depends on the hardware.

3.4 Parallelization using shared-memory model

In the shared memory model the data are stored in a common memory. Each time a processor needs data from a neighbouring processor can take these data from the shared memory. Synchronization is obtained by synchronization primitive routines. The shared-memory model is significantly simpler than the message-passing one. For the parallelization of a code using shared-memory the development of complicated routines for the distribution of data and communication between the processors is not needed. The local and global communication is obtained by receiving information from the common memory. For the present method local communication between the processors is needed after each Runge-Kutta iteration for updating the values along the subdomain boundaries. Global communication is needed for checking the convergence at each time step.
Table 1: Computational volumes on each grid

<table>
<thead>
<tr>
<th></th>
<th>block 1</th>
<th>block 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>grid-1</td>
<td>5x6x6</td>
<td>8x6x12</td>
</tr>
<tr>
<td>grid-2</td>
<td>10x12x12</td>
<td>16x12x24</td>
</tr>
<tr>
<td>grid-3</td>
<td>20x24x24</td>
<td>32x24x48</td>
</tr>
</tbody>
</table>

4 Results and conclusions

A three-dimensional backward-facing step geometry was used for investigating the performance characteristics of the shared memory and message passing parallelization models. A schematic of the two-block geometry is shown in Fig. (2). Experimental results for this geometry and for large aspect ratio (8:1) of the channel are given by Durst et al. [5]. Comparison of the predictions with the experimental results for the velocity profiles at different locations in the channel are shown in Fig. (2).

The parallel code has been divided into two parts. The first one is the algorithmic part including the necessary definitions and programming extensions for the parallelization, and the second one includes the routines used for send-receive data between the processors. The latter is hardware dependent but the structure of the code is such that its extension on another parallel platform is not an immense effort. In the present study the performance investigations have been done on a KSR-1 system with eight processors. Each processor has 32 Mbytes memory and the measured setup time required to enable message passing is 110 \( \mu \text{sec} \). Computations on three different grids have been performed by using the shared memory, as well as, the message passing model. The computational volumes on each grid are given in Table (1).

In Tables (2) and (3) results for the total efficiency \( E_n \% \) (\( E_n \% = \frac{T_1}{nT_n} \), \( T_1 \) = computing time on one processor, \( T_n \) = computing time on \( n \) processors) using the shared memory and the message passing model are shown, respectively. The efficiencies are given for computations on three different grids. In the first two columns of these tables the grid-patitioning on each block is given (e.g a 1x2x3 partitioning is made on six processors, two in \( y \)- and three in the \( z \)-direction). The comparison of the total efficiencies shows that better performance is achieved by using the shared-memory model. For fine grids and large number of processors the differences between shared memory and message-passing are large e.g on eight processors the efficiencies for the fine grid are 75\%, and 37\% using shared memory, and message passing, respectively.

The low efficiency of the message-passing model is due to the high set-up
Table 2: Grid partitioning and total efficiencies on three-different grids using shared memory.

<table>
<thead>
<tr>
<th></th>
<th>block 1</th>
<th>block 2</th>
<th>(E_n)%, grid-1</th>
<th>(E_n)%, grid-2</th>
<th>(E_n)%, grid-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 proc.</td>
<td>-</td>
<td>-</td>
<td>100.</td>
<td>100.</td>
<td>100.0</td>
</tr>
<tr>
<td>2 procs.</td>
<td>1\times1\times1</td>
<td>1\times1\times1</td>
<td>70.9</td>
<td>65.9</td>
<td>69.1</td>
</tr>
<tr>
<td>3 procs.</td>
<td>1\times1\times1</td>
<td>1\times1\times2</td>
<td>60.8</td>
<td>70.0</td>
<td>89.2</td>
</tr>
<tr>
<td>4 procs.</td>
<td>1\times1\times1</td>
<td>1\times1\times3</td>
<td>48.5</td>
<td>74.4</td>
<td>94.6</td>
</tr>
<tr>
<td>5 procs.</td>
<td>1\times1\times1</td>
<td>2\times1\times2</td>
<td>40.1</td>
<td>61.0</td>
<td>83.6</td>
</tr>
<tr>
<td>6 procs.</td>
<td>1\times1\times2</td>
<td>2\times1\times2</td>
<td>33.1</td>
<td>57.2</td>
<td>81.3</td>
</tr>
<tr>
<td>8 procs.</td>
<td>1\times1\times2</td>
<td>1\times2\times3</td>
<td>23.0</td>
<td>43.3</td>
<td>75.0</td>
</tr>
</tbody>
</table>

Time needed to enable message passing. As in Ref. [3] has been shown, the most important factor influencing the performance of computations using message passing is the ratio of the above set-up time to the time needed for one floating point operation. The best results are always achieved for parallel systems with the smallest above ratio [3]. Due to the load balancing effects the efficiency on two processors is smaller than the efficiency on three processors. The effect of the load imbalance is also shown in Table (3) using different grid partitioning of the fine grid on eight processors. Better efficiencies, as it is expected, are achieved when the processors contain almost the same number of grid points. The effect of the load imbalance is stronger on the efficiency when the shared memory model is used.

In this paper development and results from the parallelization of a multiblock three-dimensional incompressible code were presented by using shared memory, as well as, message passing models. The implementation of the code on other parallel platforms will be a subject of our future research work.

Table 3: Grid partitioning and total efficiencies on three-different grids using message passing.

<table>
<thead>
<tr>
<th></th>
<th>block 1</th>
<th>block 2</th>
<th>(E_n)%, grid-1</th>
<th>(E_n)%, grid-2</th>
<th>(E_n)%, grid-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 proc.</td>
<td>-</td>
<td>-</td>
<td>100.</td>
<td>100.</td>
<td>100.0</td>
</tr>
<tr>
<td>2 procs.</td>
<td>1\times1\times1</td>
<td>1\times1\times1</td>
<td>71.7</td>
<td>63.1</td>
<td>64.8</td>
</tr>
<tr>
<td>3 procs.</td>
<td>1\times1\times1</td>
<td>1\times1\times2</td>
<td>60.1</td>
<td>68.4</td>
<td>73.8</td>
</tr>
<tr>
<td>4 procs.</td>
<td>1\times1\times1</td>
<td>1\times1\times3</td>
<td>49.2</td>
<td>63.3</td>
<td>72.8</td>
</tr>
<tr>
<td>5 procs.</td>
<td>1\times1\times1</td>
<td>2\times1\times2</td>
<td>39.5</td>
<td>51.0</td>
<td>57.6</td>
</tr>
<tr>
<td>6 procs.</td>
<td>1\times1\times2</td>
<td>2\times1\times2</td>
<td>33.3</td>
<td>41.6</td>
<td>54.2</td>
</tr>
<tr>
<td>8 procs.</td>
<td>1\times1\times2</td>
<td>1\times2\times3</td>
<td>23.7</td>
<td>29.8</td>
<td>37.0</td>
</tr>
</tbody>
</table>
Table 4: Load balancing effects for grid partitioning on 8 processors using shared memory and message passing models.

<table>
<thead>
<tr>
<th>block 1</th>
<th>block 2</th>
<th>$E_n%$</th>
<th>Time (sec.)/iter.</th>
<th>$E_n%$</th>
<th>Time (sec.)/iter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1x1x2</td>
<td>1x2x3</td>
<td>75.0</td>
<td>43.9</td>
<td>37.0</td>
<td>89.0</td>
</tr>
<tr>
<td>4x1x1</td>
<td>4x1x1</td>
<td>44.3</td>
<td>74.4</td>
<td>25.9</td>
<td>127.2</td>
</tr>
<tr>
<td>1x1x4</td>
<td>1x1x4</td>
<td>55.8</td>
<td>59.0</td>
<td>31.1</td>
<td>105.8</td>
</tr>
<tr>
<td>1x6x1</td>
<td>1x1x2</td>
<td>32.0</td>
<td>103.0</td>
<td>21.4</td>
<td>154.0</td>
</tr>
<tr>
<td>2x1x1</td>
<td>1x1x6</td>
<td>73.2</td>
<td>45.0</td>
<td>37.5</td>
<td>87.8</td>
</tr>
</tbody>
</table>

Acknowledgements

The financial support from the Bavarian Science Foundation and the Bavarian Ministry of Education, through the Bavarian Consortium for High Performance Scientific Computing (FORTWIHR), as well as, the financial support from the EU project Copernicus ”Highly Efficient 3D CFD Codes for Industrial Applications” COP-1294 are greatly appreciated.

References


Figure 1: Schematic representation of grid partitioning.

Figure 2: Schematic representation of the backward-facing step geometry and comparison of the numerical predictions with the experimental results.